**CONTENTS:**

1 Overview 1
   1.1 Introduction ................................................................. 1
   1.2 Illustrations of the model in action .................................... 4
      1.2.1 Global atmosphere: ‘Held-Suarez’ benchmark .................. 4
      1.2.2 Ocean gyres .............................................................. 5
      1.2.3 Global ocean circulation ............................................ 5
      1.2.4 Convection and mixing over topography ......................... 9
      1.2.5 Boundary forced internal waves .................................. 9
      1.2.6 Parameter sensitivity using the adjoint of MITgcm ........... 11
      1.2.7 Global state estimation of the ocean ........................... 11
      1.2.8 Ocean biogeochemical cycles .................................... 14
      1.2.9 Simulations of laboratory experiments ....................... 15
   1.3 Continuous equations in ‘r’ coordinates ............................. 15
      1.3.1 Kinematic Boundary conditions .................................... 18
      1.3.2 Atmosphere .............................................................. 18
      1.3.3 Ocean ................................................................. 19
      1.3.4 Hydrostatic, Quasi-hydrostatic, Quasi-nonhydrostatic and Non-hydrostatic forms .................. 20
      1.3.5 Solution strategy .................................................... 22
      1.3.6 Finding the pressure field ....................................... 23
      1.3.7 Forcing/dissipation ................................................ 25
      1.3.8 Vector invariant form .............................................. 26
      1.3.9 Adjoint ............................................................... 26
   1.4 Appendix ATMOSPHERE .................................................... 26
      1.4.1 Hydrostatic Primitive Equations for the Atmosphere in Pressure Coordinates .................. 26
   1.5 Appendix OCEAN ............................................................ 28
      1.5.1 Equations of Motion for the Ocean ............................... 28
   1.6 Appendix OPERATORS ...................................................... 31
      1.6.1 Coordinate systems ................................................. 31

2 Discretization and Algorithm 35
   2.1 Notation ................................................................. 35
   2.2 Time-stepping ............................................................ 36
   2.3 Pressure method with rigid-lid ....................................... 36
   2.4 Pressure method with implicit linear free-surface ................ 38
   2.5 Explicit time-stepping: Adams-Bashforth ........................... 39
      2.5.1 Adams-Bashforth II ................................................ 39
      2.5.2 Adams-Bashforth III ............................................. 40
   2.6 Implicit time-stepping: backward method .......................... 40
   2.7 Synchronous time-stepping: variables co-located in time ...... 44
   2.8 Staggered baroclinic time-stepping ................................ 47
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.9 Non-hydrostatic formulation</td>
<td>49</td>
</tr>
<tr>
<td>2.10 Variants on the Free Surface</td>
<td>50</td>
</tr>
<tr>
<td>2.10.1 Crank-Nicolson barotropic time stepping</td>
<td>51</td>
</tr>
<tr>
<td>2.10.2 Non-linear free-surface</td>
<td>52</td>
</tr>
<tr>
<td>2.11 Spatial discretization of the dynamical equations</td>
<td>57</td>
</tr>
<tr>
<td>2.11.1 The finite volume method: finite volumes versus finite difference</td>
<td>57</td>
</tr>
<tr>
<td>2.11.2 C grid staggering of variables</td>
<td>58</td>
</tr>
<tr>
<td>2.11.3 Grid initialization and data</td>
<td>58</td>
</tr>
<tr>
<td>2.11.4 Horizontal grid</td>
<td>58</td>
</tr>
<tr>
<td>2.11.5 Vertical grid</td>
<td>61</td>
</tr>
<tr>
<td>2.11.6 Topography: partially filled cells</td>
<td>62</td>
</tr>
<tr>
<td>2.12 Continuity and horizontal pressure gradient term</td>
<td>63</td>
</tr>
<tr>
<td>2.13 Hydrostatic balance</td>
<td>63</td>
</tr>
<tr>
<td>2.14 Flux-form momentum equations</td>
<td>64</td>
</tr>
<tr>
<td>2.14.1 Advection of momentum</td>
<td>64</td>
</tr>
<tr>
<td>2.14.2 Coriolis terms</td>
<td>65</td>
</tr>
<tr>
<td>2.14.3 Curvature metric terms</td>
<td>66</td>
</tr>
<tr>
<td>2.14.4 Non-hydrostatic metric terms</td>
<td>66</td>
</tr>
<tr>
<td>2.14.5 Lateral dissipation</td>
<td>67</td>
</tr>
<tr>
<td>2.14.6 Vertical dissipation</td>
<td>68</td>
</tr>
<tr>
<td>2.14.7 Derivation of discrete energy conservation</td>
<td>69</td>
</tr>
<tr>
<td>2.14.8 Mom Diagnostics</td>
<td>69</td>
</tr>
<tr>
<td>2.15 Vector invariant momentum equations</td>
<td>71</td>
</tr>
<tr>
<td>2.15.1 Relative vorticity</td>
<td>72</td>
</tr>
<tr>
<td>2.15.2 Kinetic energy</td>
<td>72</td>
</tr>
<tr>
<td>2.15.3 Coriolis terms</td>
<td>72</td>
</tr>
<tr>
<td>2.15.4 Shear terms</td>
<td>73</td>
</tr>
<tr>
<td>2.15.5 Gradient of Bernoulli function</td>
<td>73</td>
</tr>
<tr>
<td>2.15.6 Horizontal divergence</td>
<td>74</td>
</tr>
<tr>
<td>2.15.7 Horizontal dissipation</td>
<td>74</td>
</tr>
<tr>
<td>2.15.8 Vertical dissipation</td>
<td>74</td>
</tr>
<tr>
<td>2.16 Tracer equations</td>
<td>75</td>
</tr>
<tr>
<td>2.16.1 Time-stepping of tracers: ABII</td>
<td>75</td>
</tr>
<tr>
<td>2.17 Advection schemes</td>
<td>76</td>
</tr>
<tr>
<td>2.17.1 Linear advection schemes</td>
<td>76</td>
</tr>
<tr>
<td>2.17.2 Non-linear advection schemes</td>
<td>79</td>
</tr>
<tr>
<td>2.17.3 Comparison of advection schemes</td>
<td>82</td>
</tr>
<tr>
<td>2.18 Shapiro Filter</td>
<td>87</td>
</tr>
<tr>
<td>2.18.1 SHAP Diagnostics</td>
<td>91</td>
</tr>
<tr>
<td>2.19 Nonlinear Viscosities for Large Eddy Simulation</td>
<td>91</td>
</tr>
<tr>
<td>2.19.1 Eddy Viscosity</td>
<td>92</td>
</tr>
<tr>
<td>2.19.2 Mercator, Nondimensional Equations</td>
<td>97</td>
</tr>
</tbody>
</table>

3 Getting Started with MITgcm

<table>
<thead>
<tr>
<th>Subsection</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1 Where to find information</td>
<td>99</td>
</tr>
<tr>
<td>3.2 Obtaining the code</td>
<td>99</td>
</tr>
<tr>
<td>3.2.1 Method 1</td>
<td>100</td>
</tr>
<tr>
<td>3.2.2 Method 2</td>
<td>100</td>
</tr>
<tr>
<td>3.3 Updating the code</td>
<td>100</td>
</tr>
<tr>
<td>3.4 Model and directory structure</td>
<td>101</td>
</tr>
<tr>
<td>3.5 Building the model</td>
<td>102</td>
</tr>
<tr>
<td>3.5.1 Quickstart Guide</td>
<td>102</td>
</tr>
<tr>
<td>3.5.2 Generating a Makefile using genmake2 .</td>
<td>103</td>
</tr>
<tr>
<td>3.5.3 make commands</td>
<td>109</td>
</tr>
<tr>
<td>Section</td>
<td>Title</td>
</tr>
<tr>
<td>---------</td>
<td>----------------------------------------------------------------------</td>
</tr>
<tr>
<td>3.5.4</td>
<td>Building with MPI</td>
</tr>
<tr>
<td>3.5.5</td>
<td>Building with OpenMP</td>
</tr>
<tr>
<td>3.6</td>
<td>Running the model</td>
</tr>
<tr>
<td>3.6.1</td>
<td>Running with MPI</td>
</tr>
<tr>
<td>3.6.2</td>
<td>Running with OpenMP</td>
</tr>
<tr>
<td>3.6.3</td>
<td>Output files</td>
</tr>
<tr>
<td>3.6.4</td>
<td>Looking at the output</td>
</tr>
<tr>
<td>3.7</td>
<td>Customizing the Model Configuration - Code Parameters and Compilation Options</td>
</tr>
<tr>
<td>3.7.1</td>
<td>Model Array Dimensions</td>
</tr>
<tr>
<td>3.7.2</td>
<td>C Preprocessor Options</td>
</tr>
<tr>
<td>3.7.3</td>
<td>Preprocessor Execution Environment Options</td>
</tr>
<tr>
<td>3.8</td>
<td>Customizing the Model Configuration - Runtime Parameters</td>
</tr>
<tr>
<td>3.8.1</td>
<td>Parameters: Configuration, Computational Domain, Geometry, and Time-Discretization</td>
</tr>
<tr>
<td>3.8.2</td>
<td>Parameters: Main Algorithmic Parameters</td>
</tr>
<tr>
<td>3.8.3</td>
<td>Parameters: Equation of State</td>
</tr>
<tr>
<td>3.8.4</td>
<td>Parameters: Momentum Equations</td>
</tr>
<tr>
<td>3.8.5</td>
<td>Parameters: Tracer Equations</td>
</tr>
<tr>
<td>3.8.6</td>
<td>Parameters: Model Forcing</td>
</tr>
<tr>
<td>3.8.7</td>
<td>Parameters: Simulation Controls</td>
</tr>
<tr>
<td>3.8.8</td>
<td>Parameters Used In Optional Packages</td>
</tr>
<tr>
<td>3.8.9</td>
<td>Execution Environment Parameters</td>
</tr>
<tr>
<td>3.9</td>
<td>MITgcm Input Data File Format</td>
</tr>
<tr>
<td>4</td>
<td>MITgcm Tutorial Example Experiments</td>
</tr>
<tr>
<td>4.1</td>
<td>Barotropic Ocean Gyre</td>
</tr>
<tr>
<td>4.1.1</td>
<td>Equations Solved</td>
</tr>
<tr>
<td>4.1.2</td>
<td>Discrete Numerical Configuration</td>
</tr>
<tr>
<td>4.1.3</td>
<td>Configuration</td>
</tr>
<tr>
<td>4.1.4</td>
<td>Building and running the model</td>
</tr>
<tr>
<td>4.1.5</td>
<td>Model Solution</td>
</tr>
<tr>
<td>4.2</td>
<td>Baroclinic Ocean Gyre</td>
</tr>
<tr>
<td>4.2.1</td>
<td>Equations solved</td>
</tr>
<tr>
<td>4.2.2</td>
<td>Discrete Numerical Configuration</td>
</tr>
<tr>
<td>4.2.3</td>
<td>Configuration</td>
</tr>
<tr>
<td>4.2.4</td>
<td>Building and running the model</td>
</tr>
<tr>
<td>4.2.5</td>
<td>Running with MPI</td>
</tr>
<tr>
<td>4.2.6</td>
<td>Running with OpenMP</td>
</tr>
<tr>
<td>4.2.7</td>
<td>Model solution</td>
</tr>
<tr>
<td>4.3</td>
<td>Southern Ocean Reentrant Channel Example</td>
</tr>
<tr>
<td>4.3.1</td>
<td>Equations Solved</td>
</tr>
<tr>
<td>4.3.2</td>
<td>Discrete Numerical Configuration</td>
</tr>
<tr>
<td>4.3.3</td>
<td>Configuration</td>
</tr>
<tr>
<td>4.3.4</td>
<td>Building and running the model</td>
</tr>
<tr>
<td>4.3.5</td>
<td>Model Solution</td>
</tr>
<tr>
<td>4.4</td>
<td>Ocean Gyre Advection Schemes</td>
</tr>
<tr>
<td>4.4.1</td>
<td>Advection and tracer transport</td>
</tr>
<tr>
<td>4.4.2</td>
<td>Introducing a tracer into the flow</td>
</tr>
<tr>
<td>4.4.3</td>
<td>Selecting an advection scheme</td>
</tr>
<tr>
<td>4.4.4</td>
<td>Comparison of different advection schemes</td>
</tr>
<tr>
<td>4.5</td>
<td>Global Ocean Simulation</td>
</tr>
<tr>
<td>4.5.1</td>
<td>Overview</td>
</tr>
<tr>
<td>4.5.2</td>
<td>Discrete Numerical Configuration</td>
</tr>
<tr>
<td>4.5.3</td>
<td>Experiment Configuration</td>
</tr>
<tr>
<td>4.6</td>
<td>Global Ocean Simulation in Pressure Coordinates</td>
</tr>
</tbody>
</table>
5 Contributing to the MITgcm

5.1 Bugs and feature requests

5.2 Using Git and Github

5.2.1 Quickstart Guide

5.2.2 Detailed guide for those less familiar with Git and GitHub

5.3 Coding style guide

5.4 Creating MITgcm packages
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.4.1</td>
<td>Package structure</td>
</tr>
<tr>
<td>5.4.2</td>
<td>Package boot sequence</td>
</tr>
<tr>
<td>5.4.3</td>
<td>Package S/R calls</td>
</tr>
<tr>
<td>5.4.4</td>
<td>Package “mypackage”</td>
</tr>
<tr>
<td>5.5.1</td>
<td>MITgcm code testing protocols</td>
</tr>
<tr>
<td>5.5.2</td>
<td>Test-experiment directory content</td>
</tr>
<tr>
<td>5.5.3</td>
<td>The testreport utility</td>
</tr>
<tr>
<td>5.5.4</td>
<td>The do_tst_2+2 utility</td>
</tr>
<tr>
<td>5.5.5</td>
<td>Daily Testing of MITgcm</td>
</tr>
<tr>
<td>5.6.1</td>
<td>Section headings</td>
</tr>
<tr>
<td>5.6.2</td>
<td>Internal document references</td>
</tr>
<tr>
<td>5.6.3</td>
<td>Citations</td>
</tr>
<tr>
<td>5.6.4</td>
<td>Other embedded links</td>
</tr>
<tr>
<td>5.6.5</td>
<td>Symbolic Notation</td>
</tr>
<tr>
<td>5.6.6</td>
<td>Figures</td>
</tr>
<tr>
<td>5.6.7</td>
<td>Tables</td>
</tr>
<tr>
<td>5.6.8</td>
<td>Other text blocks</td>
</tr>
<tr>
<td>5.6.9</td>
<td>Other style conventions</td>
</tr>
<tr>
<td>5.6.10</td>
<td>Building the manual</td>
</tr>
<tr>
<td>5.7</td>
<td>Reviewing pull requests</td>
</tr>
<tr>
<td>6.1</td>
<td>Overall architectural goals</td>
</tr>
<tr>
<td>6.2</td>
<td>WRAPPER</td>
</tr>
<tr>
<td>6.2.1</td>
<td>Target hardware</td>
</tr>
<tr>
<td>6.2.2</td>
<td>Supporting hardware neutrality</td>
</tr>
<tr>
<td>6.2.3</td>
<td>WRAPPER machine model</td>
</tr>
<tr>
<td>6.2.4</td>
<td>Machine model parallelism</td>
</tr>
<tr>
<td>6.2.5</td>
<td>Communication mechanisms</td>
</tr>
<tr>
<td>6.2.6</td>
<td>Communication primitives</td>
</tr>
<tr>
<td>6.2.7</td>
<td>Memory architecture</td>
</tr>
<tr>
<td>6.2.8</td>
<td>Summary</td>
</tr>
<tr>
<td>6.3</td>
<td>Using the WRAPPER</td>
</tr>
<tr>
<td>6.3.1</td>
<td>Specifying a domain decomposition</td>
</tr>
<tr>
<td>6.3.2</td>
<td>Starting the code</td>
</tr>
<tr>
<td>6.3.3</td>
<td>Controlling communication</td>
</tr>
<tr>
<td>6.4</td>
<td>MITgcm execution under WRAPPER</td>
</tr>
<tr>
<td>6.4.1</td>
<td>Annotated call tree for MITgcm and WRAPPER</td>
</tr>
<tr>
<td>6.4.2</td>
<td>Measuring and Characterizing Performance</td>
</tr>
<tr>
<td>6.4.3</td>
<td>Estimating Resource Requirements</td>
</tr>
<tr>
<td>7.1</td>
<td>Some basic algebra</td>
</tr>
<tr>
<td>7.1.1</td>
<td>Forward or direct sensitivity</td>
</tr>
<tr>
<td>7.1.2</td>
<td>Reverse or adjoint sensitivity</td>
</tr>
<tr>
<td>7.1.3</td>
<td>Storing vs. recomputation in reverse mode</td>
</tr>
<tr>
<td>7.2</td>
<td>TLM and ADM generation in general</td>
</tr>
<tr>
<td>7.2.1</td>
<td>General setup</td>
</tr>
<tr>
<td>7.2.2</td>
<td>Building the AD code using TAF</td>
</tr>
<tr>
<td>7.2.3</td>
<td>The AD build process in detail</td>
</tr>
<tr>
<td>7.2.4</td>
<td>The cost function (dependent variable)</td>
</tr>
<tr>
<td>7.2.5</td>
<td>The control variables (independent variables)</td>
</tr>
</tbody>
</table>
7.3 The gradient check package ........................................ 394
  7.3.1 Code description .................................................. 395
  7.3.2 Code configuration .................................................. 395
7.4 Adjoint dump & restart – divided adjoint (DIVA) ................ 396
  7.4.1 Introduction .......................................................... 396
  7.4.2 Recipe 1: single processor ........................................ 397
  7.4.3 Recipe 2: multi processor (MPI) ................................. 398
7.5 Adjoint code generation using OpenAD ............................ 398
  7.5.1 Introduction .......................................................... 399
  7.5.2 Downloading and installing OpenAD ............................ 399
  7.5.3 Building MITgcm adjoint with OpenAD ......................... 399
  7.5.4 Building the MITgcm adjoint using an OpenAD Singularity container ......................... 399

8 Packages I - Physical Parameterizations .......................... 401
  8.1 Overview .............................................................. 401
  8.1.1 Using MITgcm Packages ........................................... 401
  8.2 Packages Related to Hydrodynamical Kernel .................... 406
    8.2.1 Generic Advection/Diffusion .................................... 406
    8.2.2 Momentum Packages .............................................. 408
    8.2.3 Shapiro Filter ................................................... 408
    8.2.4 FFT Filtering Code .............................................. 409
    8.2.5 exch2: Extended Cubed Sphere Topology ...................... 409
    8.2.6 Gridalt - Alternate Grid Package .............................. 416
  8.3 General purpose numerical infrastructure packages .......... 420
    8.3.1 OBCS: Open boundary conditions for regional modeling .. 420
    8.3.2 RBCS Package ..................................................... 429
    8.3.3 PTRACERS Package ............................................... 432
  8.4 Ocean Packages ..................................................... 435
    8.4.1 GMREDI: Gent-McWilliams/Redi SGS Eddy Parameterization . 435
    8.4.2 KPP: Nonlocal K-Profile Parameterization for Vertical Mixing .... 442
    8.4.3 GGL90: a TKE vertical mixing scheme .......................... 448
    8.4.4 OPPS: Ocean Penetrative Plume Scheme ....................... 448
    8.4.5 KL10: Vertical Mixing Due to Breaking Internal Waves ....... 449
    8.4.6 BULK_FORCE: Bulk Formula Package .......................... 452
    8.4.7 EXF: The external forcing package .................................. 455
    8.4.8 CAL: The calendar package ....................................... 464
  8.5 Atmosphere Packages ................................................ 468
    8.5.1 Atmospheric Intermediate Physics: AIM ....................... 468
    8.5.2 Land package ..................................................... 470
    8.5.3 Fizhi: High-end Atmospheric Physics .......................... 471
  8.6 Ice and Sea Ice Packages ......................................... 510
    8.6.1 THSICE: The Thermodynamic Sea Ice Package ................. 510
    8.6.2 SEAICE Package .................................................. 516
    8.6.3 SHELFICE Package ............................................... 540
    8.6.4 SHELFICE Remeshing ............................................ 547
    8.6.5 STREAMICE Package .............................................. 551
  8.7 Biogeochemistry Packages ...................................... 562
    8.7.1 GCHEM Package ................................................... 562
    8.7.2 DIC Package ..................................................... 564

9 Packages II - Diagnostics and I/O ................................. 567
  9.1 pkg/diagnostics – A Flexible Infrastructure .................... 567
  9.1.1 Introduction .......................................................... 567
  9.1.2 Equations ........................................................... 567
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>11.1</td>
<td>MITgcmutils</td>
<td>627</td>
</tr>
<tr>
<td>11.1.1</td>
<td>mds</td>
<td>627</td>
</tr>
<tr>
<td>11.1.2</td>
<td>mnc</td>
<td>629</td>
</tr>
<tr>
<td>11.1.3</td>
<td>diagnostics</td>
<td>631</td>
</tr>
<tr>
<td>11.1.4</td>
<td>ptracers</td>
<td>631</td>
</tr>
<tr>
<td>11.1.5</td>
<td>jmd95</td>
<td>632</td>
</tr>
<tr>
<td>11.1.6</td>
<td>mdjwf</td>
<td>633</td>
</tr>
<tr>
<td>11.1.7</td>
<td>cs</td>
<td>634</td>
</tr>
<tr>
<td>11.1.8</td>
<td>llc</td>
<td>635</td>
</tr>
<tr>
<td>11.1.9</td>
<td>glueinementbig</td>
<td>637</td>
</tr>
<tr>
<td>12</td>
<td>Related Projects and Highlighted Papers</td>
<td>639</td>
</tr>
<tr>
<td>12.1</td>
<td>Projects Related to MITgcm</td>
<td>639</td>
</tr>
<tr>
<td>12.1.1</td>
<td>Estimating the Circulation and Climate of the Ocean (ECCO)</td>
<td>639</td>
</tr>
<tr>
<td>12.1.2</td>
<td>Gcmfaces: Gridded Earth Variables In Matlab And Octave</td>
<td>639</td>
</tr>
<tr>
<td>12.1.3</td>
<td>MITprof: In-Situ Ocean Data In Matlab And Octave</td>
<td>639</td>
</tr>
<tr>
<td>12.1.4</td>
<td>OceanParcels - Lagrangian Particle Tracker</td>
<td>639</td>
</tr>
<tr>
<td>12.1.5</td>
<td>Southern Ocean State Estimation (SOSE)</td>
<td>640</td>
</tr>
<tr>
<td>12.1.6</td>
<td>Xgcm: General Circulation Model Postprocessing with xarray</td>
<td>640</td>
</tr>
<tr>
<td>12.1.7</td>
<td>Xmitgcm</td>
<td>640</td>
</tr>
<tr>
<td>12.2</td>
<td>Highlighted Papers</td>
<td>640</td>
</tr>
</tbody>
</table>

**Bibliography** 641

**Python Module Index** 653

**Index** 655
This document provides the reader with the information necessary to carry out numerical experiments using MITgcm. It gives a comprehensive description of the continuous equations on which the model is based, the numerical algorithms the model employs and a description of the associated program code. Along with the hydrodynamical kernel, physical and biogeochemical parameterizations of key atmospheric and oceanic processes are available. A number of examples illustrating the use of the model in both process and general circulation studies of the atmosphere and ocean are also presented.

1.1 Introduction

MITgcm has a number of novel aspects:

- it can be used to study both atmospheric and oceanic phenomena; one hydrodynamical kernel is used to drive forward both atmospheric and oceanic models - see Figure 1.1

![Figure 1.1: MITgcm has a single dynamical kernel that can drive forward either oceanic or atmospheric simulations.](image)

- it has a non-hydrostatic capability and so can be used to study both small-scale and large scale processes - see Figure 1.2

![Figure 1.2:](image)
MITgcm has non-hydrostatic capabilities, allowing the model to address a wide range of phenomena - from convection on the left, all the way through to global circulation patterns on the right.

- finite volume techniques are employed yielding an intuitive discretization and support for the treatment of irregular geometries using orthogonal curvilinear grids and shaved cells - see Figure 1.3

- tangent linear and adjoint counterparts are automatically maintained along with the forward model, permitting sensitivity and optimization studies.

- the model is developed to perform efficiently on a wide variety of computational platforms.


Finite Volume: Shaved Cells

Figure 1.3: Finite volume techniques (bottom panel) are used, permitting a treatment of topography that rivals $\sigma$ (terrain following) coordinates.
general circulation model and application to Atlantic heat transport variability, J. Geophysical Res., 104(C12), 29,529-29,547 [MGZ+99]


We begin by briefly showing some of the results of the model in action to give a feel for the wide range of problems that can be addressed using it.

## 1.2 Illustrations of the model in action

MITgcm has been designed and used to model a wide range of phenomena, from convection on the scale of meters in the ocean to the global pattern of atmospheric winds - see Figure 1.2. To give a flavor of the kinds of problems the model has been used to study, we briefly describe some of them here. A more detailed description of the underlying formulation, numerical algorithm and implementation that lie behind these calculations is given later. Indeed many of the illustrative examples shown below can be easily reproduced: simply download the model (the minimum you need is a PC running Linux, together with a FORTRAN77 compiler) and follow the examples described in detail in the documentation.

### 1.2.1 Global atmosphere: ‘Held-Suarez’ benchmark

A novel feature of MITgcm is its ability to simulate, using one basic algorithm, both atmospheric and oceanographic flows at both small and large scales.

Figure 1.4 shows an instantaneous plot of the 500 mb temperature field obtained using the atmospheric isomorph of MITgcm run at 2.8° resolution on the cubed sphere. We see cold air over the pole (blue) and warm air along an equatorial band (red). Fully developed baroclinic eddies spawned in the northern hemisphere storm track are evident. There are no mountains or land-sea contrast in this calculation, but you can easily put them in. The model is driven by relaxation to a radiative-convective equilibrium profile, following the description set out in Held and Suarez (1994) [HS94] designed to test atmospheric hydrodynamical cores - there are no mountains or land-sea contrast.

As described in Adcroft et al. (2004) [ACHM04], a ‘cubed sphere’ is used to discretize the globe permitting a uniform gridding and obviated the need to Fourier filter. The ‘vector-invariant’ form of MITgcm supports any orthogonal curvilinear grid, of which the cubed sphere is just one of many choices.

Figure 1.5 shows the 5-year mean, zonally averaged zonal wind from a 20-level configuration of the model. It compares favorable with more conventional spatial discretization approaches. The two plots show the field calculated using the cube-sphere grid and the flow calculated using a regular, spherical polar latitude-longitude grid. Both grids are supported within the model.
1.2.2 Ocean gyres

Baroclinic instability is a ubiquitous process in the ocean, as well as the atmosphere. Ocean eddies play an important role in modifying the hydrographic structure and current systems of the oceans. Coarse resolution models of the oceans cannot resolve the eddy field and yield rather broad, diffusive patterns of ocean currents. But if the resolution of our models is increased until the baroclinic instability process is resolved, numerical solutions of a different and much more realistic kind, can be obtained.

Figure 1.6 shows the surface temperature and velocity field obtained from MITgcm run at $\frac{1}{5}$ horizontal resolution on a lat-lon grid in which the pole has been rotated by 90° on to the equator (to avoid the converging of meridian in northern latitudes). 21 vertical levels are used in the vertical with a ‘lopped cell’ representation of topography. The development and propagation of anomalously warm and cold eddies can be clearly seen in the Gulf Stream region. The transport of warm water northward by the mean flow of the Gulf Stream is also clearly visible.

1.2.3 Global ocean circulation

Figure 1.7 shows the pattern of ocean currents at the surface of a 4° global ocean model run with 15 vertical levels. Lopped cells are used to represent topography on a regular lat-lon grid extending from 70°N to 70°S. The model is driven using monthly-mean winds with mixed boundary conditions on temperature and salinity at the surface. The transfer properties of ocean eddies, convection and mixing is parameterized in this model.

Figure 1.8 shows the meridional overturning circulation of the global ocean in Sverdrups.

Figure 1.4: Instantaneous plot of the temperature field at 500 mb obtained using the atmospheric isomorph of MITgcm
Figure 1.5: Five year mean, zonally averaged zonal flow for cube-sphere simulation (top) and latitude-longitude simulation (bottom) and using Held-Suarez forcing. Note the difference in the solutions over the pole — the cubed sphere is superior.
Figure 1.6: Instantaneous temperature map from a $\frac{1}{6}^\circ$ simulation of the North Atlantic. The figure shows the temperature in the second layer (37.5 m deep).
Figure 1.7: Pattern of surface ocean currents from a global integration of the model at 4° horizontal resolution and with 15 vertical levels.

Figure 1.8: Meridional overturning stream function (in Sverdrups) from a global integration of the model at 4° horizontal resolution and with 15 vertical levels.
1.2.4 Convection and mixing over topography

Dense plumes generated by localized cooling on the continental shelf of the ocean may be influenced by rotation when the deformation radius is smaller than the width of the cooling region. Rather than gravity plumes, the mechanism for moving dense fluid down the shelf is then through geostrophic eddies. The simulation shown in Figure 1.9 (blue is cold dense fluid, red is warmer, lighter fluid) employs the non-hydrostatic capability of MITgcm to trigger convection by surface cooling. The cold, dense water falls down the slope but is deflected along the slope by rotation. It is found that entrainment in the vertical plane is reduced when rotational control is strong, and replaced by lateral entrainment due to the baroclinic instability of the along-slope current.

Figure 1.9: MITgcm run in a non-hydrostatic configuration to study convection over a slope.

1.2.5 Boundary forced internal waves

The unique ability of MITgcm to treat non-hydrostatic dynamics in the presence of complex geometry makes it an ideal tool to study internal wave dynamics and mixing in oceanic canyons and ridges driven by large amplitude barotropic tidal currents imposed through open boundary conditions.

Figure 1.10 shows the influence of cross-slope topographic variations on internal wave breaking - the cross-slope velocity is in color, the density contoured. The internal waves are excited by application of open boundary conditions on the left. They propagate to the sloping boundary (represented using MITgcm’s finite volume spatial discretization) where they break under non-hydrostatic dynamics.
Figure 1.10: Simulation of internal waves forced at an open boundary (on the left) impacting a sloping shelf. The along slope velocity is shown colored, contour lines show density surfaces. The slope is represented with high-fidelity using lopped cells.
1.2.6 Parameter sensitivity using the adjoint of MITgcm

Forward and tangent linear counterparts of MITgcm are supported using an ‘automatic adjoint compiler’. These can be used in parameter sensitivity and data assimilation studies.

As one example of application of the MITgcm adjoint, Figure 1.11 maps the gradient $\frac{\partial J}{\partial \mathcal{H}}$ where $J$ is the magnitude of the overturning stream-function shown in Figure 1.8 at 60°N and $\mathcal{H}(\lambda, \varphi)$ is the mean, local air-sea heat flux over a 100 year period. We see that $J$ is sensitive to heat fluxes over the Labrador Sea, one of the important sources of deep water for the thermohaline circulations. This calculation also yields sensitivities to all other model parameters.

![Sensitivity of the Meridional Overturning Ocean](image)

Figure 1.11: Sensitivity of meridional overturning strength to surface heat flux changes. Contours show the magnitude of the response (in $\text{Sv} \times 10^{-4}$) that a persistent +1 Wm$^{-2}$ heat flux anomaly at a given grid point would produce.

1.2.7 Global state estimation of the ocean

An important application of MITgcm is in state estimation of the global ocean circulation. An appropriately defined ‘cost function’, which measures the departure of the model from observations (both remotely sensed and in-situ) over an interval of time, is minimized by adjusting ‘control parameters’ such as air-sea fluxes, the wind field, the initial conditions etc. Figure 1.12 and Figure 1.13 show the large scale planetary circulation and a Hopf-Muller plot of Equatorial sea-surface height. Both are obtained from assimilation bringing the model in to consistency with altimetric and in-situ observations over the period 1992-1997.
Figure 1.12: Circulation patterns from a multi-year, global circulation simulation constrained by Topex altimeter data and WOCE cruise observations. This output is from a higher resolution, shorter duration experiment with equatorially enhanced grid spacing.
Figure 1.13: Equatorial sea-surface height in unconstrained (left), constrained (middle) simulations and in observations (right).
1.2.8 Ocean biogeochemical cycles

MITgcm is being used to study global biogeochemical cycles in the ocean. For example one can study the effects of interannual changes in meteorological forcing and upper ocean circulation on the fluxes of carbon dioxide and oxygen between the ocean and atmosphere. Figure 1.14 shows the annual air-sea flux of oxygen and its relation to density outcrops in the southern oceans from a single year of a global, interannually varying simulation. The simulation is run at $1^\circ \times 1^\circ$ resolution telescoping to $\frac{1}{3}^\circ \times \frac{1}{3}^\circ$ in the tropics (not shown).

Figure 1.14: Annual air-sea flux of oxygen (shaded) plotted along with potential density outcrops of the surface of the southern ocean from a global $1^\circ \times 1^\circ$ integration with a telescoping grid (to $\frac{1}{3}^\circ$) at the equator.
1.2.9 Simulations of laboratory experiments

Figure 1.16 shows MITgcm being used to simulate a laboratory experiment (Figure 1.15) inquiring into the dynamics of the Antarctic Circumpolar Current (ACC). An initially homogeneous tank of water (1 m in diameter) is driven from its free surface by a rotating heated disk. The combined action of mechanical and thermal forcing creates a lens of fluid which becomes baroclinically unstable. The stratification and depth of penetration of the lens is arrested by its instability in a process analogous to that which sets the stratification of the ACC.

![Figure 1.15: A 1 m diameter laboratory experiment simulating the dynamics of the Antarctic Circumpolar Current.](image)

1.3 Continuous equations in ‘r’ coordinates

To render atmosphere and ocean models from one dynamical core we exploit ‘isomorphisms’ between equation sets that govern the evolution of the respective fluids - see Figure 1.17. One system of hydrodynamical equations is written down and encoded. The model variables have different interpretations depending on whether the atmosphere or ocean is being studied. Thus, for example, the vertical coordinate ‘r’ is interpreted as pressure, \( p \), if we are modeling the atmosphere (right hand side of Figure 1.17) and height, \( z \), if we are modeling the ocean (left hand side of Figure 1.17).

The state of the fluid at any time is characterized by the distribution of velocity \( \vec{v} \), active tracers \( \theta \) and \( S \), a ‘geopotential’ \( \phi \) and density \( \rho = \rho(\theta, S, p) \) which may depend on \( \theta, S, \) and \( p \). The equations that govern the evolution of these fields, obtained by applying the laws of classical mechanics and thermodynamics to a Boussinesq, Navier-Stokes fluid are, written in terms of a generic vertical coordinate, \( r \), so that the appropriate kinematic boundary conditions can be applied isomorphically see Figure 1.18.

\[
\frac{D\vec{v}_h}{Dt} + \left(2\vec{\Omega} \times \vec{v}\right)_h + \nabla_h \phi = \mathcal{F}_h \text{ horizontal momentum} \tag{1.1}
\]

\[
\frac{D\vec{r}}{Dt} + \hat{k} \cdot \left(2\vec{\Omega} \times \vec{v}\right) + \frac{\partial\phi}{\partial r} + b = \mathcal{F}_r \text{ vertical momentum} \tag{1.2}
\]
Figure 1.16: A numerical simulation of the laboratory experiment using MITgcm.

### z-p Isomorphism

**Ocean (z coordinates)**

\[
\begin{align*}
\frac{d\mathbf{v}}{dt} + f \times \mathbf{v} + \nabla_z P &= F \\
\nabla_z \phi + \partial_z P &= 0 \\
\nabla_z \cdot \mathbf{v} + \partial_z w &= 0 \\
\n\frac{d\theta}{dt} &= Q \\
\n\frac{dS}{dt} &= S \\
\partial_z \eta + \nabla_z (\eta + H) &= \Pi_E
\end{align*}
\]

**Atmosphere (p coordinates)**

\[
\begin{align*}
\frac{d\mathbf{v}}{dt} + f \times \mathbf{v} + \nabla_p \Phi &= F \\
\nabla_p \phi + \partial_p \Phi &= 0 \\
\n\nabla_p \cdot \mathbf{v} + \partial_p \omega &= 0 \\
\n\frac{d\theta}{dt} &= Q \\
\n\frac{dq}{dt} &= S \\
\partial_p \eta + \nabla_p (\eta + H) &= \Pi_E
\end{align*}
\]

Figure 1.17: Isomorphic equation sets used for atmosphere (right) and ocean (left).
**z-p Isomorphism**

Figure 1.18: Vertical coordinates and kinematic boundary conditions for atmosphere (top) and ocean (bottom).

\[
\nabla h \cdot \vec{v}_h + \frac{\partial \dot{r}}{\partial r} = 0 \quad \text{continuity (1.3)}
\]

\[
b = b(\theta, S, r) \quad \text{equation of state (1.4)}
\]

\[
\frac{D\theta}{Dt} = Q_\theta \quad \text{potential temperature (1.5)}
\]

\[
\frac{DS}{Dt} = Q_S \quad \text{humidity/salinity (1.6)}
\]

Here:

- \( r \) is the vertical coordinate

\[
\frac{D}{Dt} = \frac{\partial}{\partial t} + \vec{v} \cdot \nabla \quad \text{is the total derivative}
\]

\[
\nabla = \nabla_h + \hat{k} \frac{\partial}{\partial r} \quad \text{is the ‘grad’ operator}
\]

with \( \nabla_h \) operating in the horizontal and \( \hat{k} \frac{\partial}{\partial r} \) operating in the vertical, where \( \hat{k} \) is a unit vector in the vertical

- \( t \) is time

\[
\vec{v} = (u, v, \dot{r}) = (\vec{v}_h, \dot{r}) \quad \text{is the velocity}
\]

- \( \phi \) is the ‘pressure’/‘geopotential’

\[
\vec{\Omega} \quad \text{is the Earth’s rotation}
\]

- \( b \) is the ‘buoyancy’

- \( \theta \) is potential temperature

- \( S \) is specific humidity in the atmosphere; salinity in the ocean

1.3. Continuous equations in ‘r’ coordinates
The $\mathcal{F}$'s and $\mathcal{Q}$'s are provided by ‘physics’ and forcing packages for atmosphere and ocean. These are described in later chapters.

### 1.3.1 Kinematic Boundary conditions

#### 1.3.1.1 Vertical

at fixed and moving $r$ surfaces we set (see Figure 1.18):

$$\dot{r} = 0 \text{ at } r = R_{\text{fixed}}(x, y) \text{ (ocean bottom, top of the atmosphere)} \quad (1.7)$$

$$\dot{r} = \frac{Dr}{Dt} \text{ at } r = R_{\text{moving}}(x, y) \text{ (ocean surface, bottom of the atmosphere)} \quad (1.8)$$

Here

$$R_{\text{moving}} = R_o + \eta$$

where $R_o(x, y)$ is the ‘$r$—value’ (height or pressure, depending on whether we are in the atmosphere or ocean) of the ‘moving surface’ in the resting fluid and $\eta$ is the departure from $R_o(x, y)$ in the presence of motion.

#### 1.3.1.2 Horizontal

$$\vec{v} \cdot \vec{n} = 0 \quad (1.9)$$

where $\vec{n}$ is the normal to a solid boundary.

### 1.3.2 Atmosphere

In the atmosphere, (see Figure 1.18), we interpret:

$$r = p \text{ is the pressure} \quad (1.10)$$

$$\dot{r} = \frac{Dp}{Dt} = \omega \text{ is the vertical velocity in p coordinates} \quad (1.11)$$

$$\phi = g z \text{ is the geopotential height} \quad (1.12)$$

$$b = \frac{\partial \Pi}{\partial p} \theta \text{ is the buoyancy} \quad (1.13)$$

$$\theta = T\left(\frac{p_c}{p}\right)^{\kappa} \text{ is potential temperature} \quad (1.14)$$

$$S = q \text{ is the specific humidity} \quad (1.15)$$

where

- $T$ is absolute temperature
- $p$ is the pressure
In the above the ideal gas law, \( p = \rho RT \), has been expressed in terms of the Exner function \( \Pi(p) \) given by (1.16) (see also Section 1.4.1)

\[
\Pi(p) = c_p \left( \frac{p}{p_c} \right)^\kappa
\]

where \( p_c \) is a reference pressure and \( \kappa = R/c_p \) with \( R \) the gas constant and \( c_p \) the specific heat of air at constant pressure.

At the top of the atmosphere (which is ‘fixed’ in our \( r \) coordinate):

\[
R_{\text{fixed}} = p_{\text{top}} = 0
\]

In a resting atmosphere the elevation of the mountains at the bottom is given by

\[
R_{\text{moving}} = R_o(x,y) = p_o(x,y)
\]

i.e. the (hydrostatic) pressure at the top of the mountains in a resting atmosphere.

The boundary conditions at top and bottom are given by:

\[
\omega = 0 \text{ at } r = R_{\text{fixed}} \text{ (top of the atmosphere)} \tag{1.17}
\]

\[
\omega = \frac{D\eta}{Dt} \text{ at } r = R_{\text{moving}} \text{ (bottom of the atmosphere)} \tag{1.18}
\]

Then the (hydrostatic form of) equations (1.1)-(1.6) yields a consistent set of atmospheric equations which, for convenience, are written out in \( p \)-coordinates in Section 1.4.1 - see eqs. (1.59)-(1.63).

### 1.3.3 Ocean

In the ocean we interpret:

\[
\dot{r} = \frac{Dz}{Dt} = w \text{ is the vertical velocity} \tag{1.20}
\]

\[
\phi = \frac{p}{\rho_c} \text{ is the pressure} \tag{1.21}
\]

\[
b(\theta, S, r) = \frac{g}{\rho_c} (\rho(\theta, S, r) - \rho_c) \text{ is the buoyancy} \tag{1.22}
\]

where \( \rho_c \) is a fixed reference density of water and \( g \) is the acceleration due to gravity.

In the above:

At the bottom of the ocean: \( R_{\text{fixed}}(x,y) = -H(x,y) \).

The surface of the ocean is given by: \( R_{\text{moving}} = \eta \)

The position of the resting free surface of the ocean is given by \( R_o = Z_o = 0 \).

Boundary conditions are:

\[
w = 0 \text{ at } r = R_{\text{fixed}} \text{ (ocean bottom)} \tag{1.23}
\]

\[
w = \frac{D\eta}{Dt} \text{ at } r = R_{\text{moving}} = \eta \text{ (ocean surface)} \tag{1.24}
\]

where \( \eta \) is the elevation of the free surface.

Then equations (1.1)-(1.6) yield a consistent set of oceanic equations which, for convenience, are written out in \( z \)-coordinates in Section 1.5.1 - see eqs. (1.98) to (1.103).
1.3.4 Hydrostatic, Quasi-hydrostatic, Quasi-nonhydrostatic and Non-hydrostatic forms

Let us separate \( \phi \) into surface, hydrostatic and non-hydrostatic terms:

\[
\phi(x, y, r) = \phi_s(x, y) + \phi_{hyd}(x, y, r) + \phi_{nh}(x, y, r)
\]

and write (1.1) in the form:

\[
\frac{\partial \vec{v}}{\partial t} + \nabla \cdot \vec{h} \phi_s + \nabla \cdot \vec{h} \phi_{hyd} + \epsilon_{nh} \nabla \phi_{nh} = \vec{G}_{\vec{v}}
\]

\[
\frac{\partial \phi_{hyd}}{\partial r} = -b
\]

\[
\epsilon_{nh} \frac{\partial \hat{r}}{\partial t} + \frac{\partial \phi_{nh}}{\partial r} = G_{\hat{r}}
\]

Here \( \epsilon_{nh} \) is a non-hydrostatic parameter.

The \( \left( \vec{G}_{\vec{v}}, G_{\hat{r}} \right) \) in (1.26) and (1.28) represent advective, metric and Coriolis terms in the momentum equations. In spherical coordinates they take the form\(^1\) - see Marshall et al. (1997a) [MHPA97] for a full discussion:

\[
G_u = - \vec{v} \cdot \nabla u \quad \text{advection}
\]

\[
- \left( \frac{u \dot{r}}{r} - \frac{uv \tan \varphi}{r} \right) \quad \text{metric}
\]

\[
- \left( -2\Omega v \sin \varphi + 2\Omega \dot{r} \cos \varphi \right) \quad \text{Coriolis}
\]

\[
+ F_u \quad \text{forcing/dissipation}
\]

\[
G_v = - \vec{v} \cdot \nabla v \quad \text{advection}
\]

\[
- \left( \frac{v \dot{r}}{r} - \frac{u^2 \tan \varphi}{r} \right) \quad \text{metric}
\]

\[
- \left( 2\Omega u \sin \varphi \right) \quad \text{Coriolis}
\]

\[
+ F_v \quad \text{forcing/dissipation}
\]

\[
G_{\hat{r}} = - \vec{v} \cdot \nabla \hat{r} \quad \text{advection}
\]

\[
- \left( \frac{u^2 + v^2}{r} \right) \quad \text{metric}
\]

\[
+ 2\Omega u \cos \varphi \quad \text{Coriolis}
\]

\[
+ F_{\hat{r}} \quad \text{forcing/dissipation}
\]

In the above ‘\( r \)’ is the distance from the center of the earth and ‘\( \varphi \)’ is latitude (see Figure 1.20).

Grad and div operators in spherical coordinates are defined in Coordinate systems.

---

\(^1\) In the hydrostatic primitive equations (HPE) all underlined terms in (1.29), (1.30) and (1.31) are omitted; the singly-underlined terms are included in the quasi-hydrostatic model (QH). The fully non-hydrostatic model (NH) includes all terms.
1.3.4.1 Shallow atmosphere approximation

Most models are based on the ‘hydrostatic primitive equations’ (HPE’s) in which the vertical momentum equation is reduced to a statement of hydrostatic balance and the ‘traditional approximation’ is made in which the Coriolis force is treated approximately and the shallow atmosphere approximation is made. MITgcm need not make the ‘traditional approximation’. To be able to support consistent non-hydrostatic forms the shallow atmosphere approximation can be relaxed - when dividing through by \( r \) in, for example, (1.29), we do not replace \( r \) by \( a \), the radius of the earth.

1.3.4.2 Hydrostatic and quasi-hydrostatic forms

These are discussed at length in Marshall et al. (1997a) [MHPA97].

In the ‘hydrostatic primitive equations’ (HPE) all the underlined terms in Eqs. (1.29) → (1.31) are neglected and ‘\( r \)’ is replaced by ‘\( a \)’, the mean radius of the earth. Once the pressure is found at one level - e.g. by inverting a 2-d Elliptic equation for \( \phi_s \) at \( r = R_{moving} \) - the pressure can be computed at all other levels by integration of the hydrostatic relation, eq (1.27).

In the ‘quasi-hydrostatic’ equations (QH) strict balance between gravity and vertical pressure gradients is not imposed. The \( 2\Omega u \cos \varphi \) Coriolis term are not neglected and are balanced by a non-hydrostatic contribution to the pressure field: only the terms underlined twice in Eqs. (1.29) → (1.31) are set to zero and, simultaneously, the shallow atmosphere approximation is relaxed. In QH all the metric terms are retained and the full variation of the radial position of a particle monitored. The QH vertical momentum equation (1.28) becomes:

\[
\frac{\partial \phi_{nh}}{\partial r} = 2\Omega u \cos \varphi
\]

making a small correction to the hydrostatic pressure.

QH has good energetic credentials - they are the same as for HPE. Importantly, however, it has the same angular momentum principle as the full non-hydrostatic model (NH) - see Marshall et al. (1997a) [MHPA97]. As in HPE only a 2-d elliptic problem need be solved.

1.3.4.3 Non-hydrostatic and quasi-nonhydrostatic forms

MITgcm presently supports a full non-hydrostatic ocean isomorph, but only a quasi-non-hydrostatic atmospheric isomorph.

Non-hydrostatic Ocean

In the non-hydrostatic ocean model all terms in equations Eqs. (1.29) → (1.31) are retained. A three dimensional elliptic equation must be solved subject to Neumann boundary conditions (see below). It is important to note that use of the full NH does not admit any new ‘fast’ waves in to the system - the incompressible condition (1.3) has already filtered out acoustic modes. It does, however, ensure that the gravity waves are treated accurately with an exact dispersion relation. The NH set has a complete angular momentum principle and consistent energetics - see White and Bromley (1995) [WB95]; Marshall et al. (1997a) [MHPA97].
Quasi-nonhydrostatic Atmosphere

In the non-hydrostatic version of our atmospheric model we approximate $\dot{r}$ in the vertical momentum eqs. (1.28) and (1.30) (but only here) by:

$$\dot{r} = \frac{Dp}{Dt} = \frac{1}{g} \frac{D\phi}{Dt}$$  \hspace{1cm} (1.32)

where $p_{hy}$ is the hydrostatic pressure.

1.3.4.4 Summary of equation sets supported by model

Atmosphere

Hydrostatic, and quasi-hydrostatic and quasi non-hydrostatic forms of the compressible non-Boussinesq equations in $p$–coordinates are supported.

Hydrostatic and quasi-hydrostatic

The hydrostatic set is written out in $p$–coordinates in *Hydrostatic Primitive Equations for the Atmosphere in Pressure Coordinates* - see eqs. (1.59) to (1.63).

Quasi-nonhydrostatic

A quasi-nonhydrostatic form is also supported.

Ocean

Hydrostatic and quasi-hydrostatic

Hydrostatic, and quasi-hydrostatic forms of the incompressible Boussinesq equations in $z$–coordinates are supported.

Non-hydrostatic

Non-hydrostatic forms of the incompressible Boussinesq equations in $z$–coordinates are supported - see eqs. (1.98) to (1.103).

1.3.5 Solution strategy

The method of solution employed in the HPE, QH and NH models is summarized in Figure 1.19. Under all dynamics, a 2-d elliptic equation is first solved to find the surface pressure and the hydrostatic pressure at any level computed from the weight of fluid above. Under HPE and QH dynamics, the horizontal momentum equations are then stepped forward and $\dot{r}$ found from continuity. Under NH dynamics a 3-d elliptic equation must be solved for the non-hydrostatic pressure before stepping forward the horizontal momentum equations; $\dot{r}$ is found by stepping forward the vertical momentum equation.

There is no penalty in implementing QH over HPE except, of course, some complication that goes with the inclusion of $\cos \varphi$ Coriolis terms and the relaxation of the shallow atmosphere approximation. But this leads to negligible increase in computation. In NH, in contrast, one additional elliptic equation - a three-dimensional one - must be
inverted for $p_{nh}$. However the ‘overhead’ of the NH model is essentially negligible in the hydrostatic limit (see detailed discussion in Marshall et al. (1997) [MHPA97] resulting in a non-hydrostatic algorithm that, in the hydrostatic limit, is as computationally economic as the HPEs.

Figure 1.19: Basic solution strategy in MITgcm. HPE and QH forms diagnose the vertical velocity, in NH a prognostic equation for the vertical velocity is integrated.

### 1.3.6 Finding the pressure field

Unlike the prognostic variables $u$, $v$, $w$, $\theta$ and $S$, the pressure field must be obtained diagnostically. We proceed, as before, by dividing the total (pressure/geo) potential in to three parts, a surface part, $\phi_s(x, y)$, a hydrostatic part $\phi_{hyd}(x, y, r)$ and a non-hydrostatic part $\phi_{nh}(x, y, r)$, as in (1.25), and writing the momentum equation as in (1.26).

#### 1.3.6.1 Hydrostatic pressure

Hydrostatic pressure is obtained by integrating (1.27) vertically from $r = R_o$ where $\phi_{hyd}(r = R_o) = 0$, to yield:

$$\int_r^{R_o} \frac{\partial \phi_{hyd}}{\partial r} dr = [\phi_{hyd}]_{r=R_o} = \int_r^{R_o} -bdr$$

and so

$$\phi_{hyd}(x, y, r) = \int_r^{R_o} bdr$$

(1.33)

The model can be easily modified to accommodate a loading term (e.g. atmospheric pressure pushing down on the ocean’s surface) by setting:

$$\phi_{hyd}(r = R_o) = \text{loading}$$

(1.34)
1.3.6.2 Surface pressure

The surface pressure equation can be obtained by integrating continuity, (1.3), vertically from \( r = R_{\text{fixed}} \) to \( r = R_{\text{moving}} \):

\[
\int_{R_{\text{fixed}}}^{R_{\text{moving}}} (\nabla_h \cdot \nabla_h + \partial_r r) dr = 0
\]

Thus:

\[
\frac{\partial \eta}{\partial t} + \nabla \cdot \vec{v} + \int_{R_{\text{fixed}}}^{R_{\text{moving}}} \nabla_h \cdot \nabla_h dr = 0
\]

where \( \eta = R_{\text{moving}} - R_o \) is the free-surface \( r \)-anomaly in units of \( r \). The above can be rearranged to yield, using Leibnitz’s theorem:

\[
\frac{\partial \eta}{\partial t} + \nabla_h \cdot \int_{R_{\text{fixed}}}^{R_{\text{moving}}} \nabla_h dr = \text{source}
\]

where we have incorporated a source term.

Whether \( \phi \) is pressure (ocean model, \( p/\rho_c \)) or geopotential (atmospheric model), in (1.26), the horizontal gradient term can be written

\[
\nabla_h \phi_s = \nabla_h (b_s \eta)
\]

where \( b_s \) is the buoyancy at the surface.

In the hydrostatic limit (\( \epsilon_{nh} = 0 \)), equations (1.26), (1.35) and (1.36) can be solved by inverting a 2-d elliptic equation for \( \phi_s \) as described in Chapter 2. Both ‘free surface’ and ‘rigid lid’ approaches are available.

1.3.6.3 Non-hydrostatic pressure

Taking the horizontal divergence of (1.26) and adding \( \frac{\partial}{\partial r} \) of (1.28), invoking the continuity equation (1.3), we deduce that:

\[
\nabla_{\theta}^2 \phi_{nh} = \nabla \cdot \vec{G} - \left( \nabla_{\theta}^2 \phi_s + \nabla^2 \phi_{hyd} \right) = \nabla \cdot \vec{F}
\]

For a given rhs this 3-d elliptic equation must be inverted for \( \phi_{nh} \) subject to appropriate choice of boundary conditions. This method is usually called The Pressure Method [Harlow and Welch (1965) [HW65]; Williams (1969) [Wil69]; Potter (1973) [Pot73]. In the hydrostatic primitive equations case (HPE), the 3-d problem does not need to be solved.

Boundary Conditions

We apply the condition of no normal flow through all solid boundaries - the coasts (in the ocean) and the bottom:

\[
\vec{v} \cdot \hat{n} = 0
\]

where \( \hat{n} \) is a vector of unit length normal to the boundary. The kinematic condition (1.38) is also applied to the vertical velocity at \( r = R_{\text{moving}} \). No-slip (\( v_T = 0 \)) or slip (\( \partial v_T / \partial n = 0 \)) conditions are employed on the tangential component of velocity, \( v_T \), at all solid boundaries, depending on the form chosen for the dissipative terms in the momentum equations - see below.

Eq. (1.38) implies, making use of (1.26), that:

\[
\hat{n} \cdot \nabla \phi_{nh} = \hat{n} \cdot \vec{F}
\]
where

\[ \vec{F} = \vec{G} - (\nabla h \phi_s + \nabla \phi_{hyd}) \]

presenting inhomogeneous Neumann boundary conditions to the Elliptic problem (1.37). As shown, for example, by Williams (1969) [Wil69], one can exploit classical 3D potential theory and, by introducing an appropriately chosen \( \delta \)-function sheet of ‘source-charge’, replace the inhomogeneous boundary condition on pressure by a homogeneous one. The source term \( rhs \) in (1.37) is the divergence of the vector \( \vec{F} \). By simultaneously setting \( \vec{n} \cdot \vec{F} = 0 \) and \( \vec{n} \cdot \nabla \phi_{nh} = 0 \) on the boundary the following self-consistent but simpler homogenized Elliptic problem is obtained:

\[ \nabla^2 \phi_{nh} = \nabla \cdot \vec{F} \]

where \( \vec{F} \) is a modified \( \vec{F} \) such that \( \vec{F} \cdot \vec{n} = 0 \). As is implied by (1.39) the modified boundary condition becomes:

\[ \vec{n} \cdot \nabla \phi_{nh} = 0 \]  

(1.40)

If the flow is ‘close’ to hydrostatic balance then the 3-d inversion converges rapidly because \( \phi_{nh} \) is then only a small correction to the hydrostatic pressure field (see the discussion in Marshall et al. (1997a,b) [MHPA97] [MAH+97]. The solution \( \phi_{nh} \) to (1.37) and (1.39) does not vanish at \( r = R_{moving} \), and so refines the pressure there.

1.3.7 Forcing/dissipation

1.3.7.1 Forcing

The forcing terms \( \mathcal{F} \) on the rhs of the equations are provided by ‘physics packages’ and forcing packages. These are described later on.

1.3.7.2 Dissipation

Momentum

Many forms of momentum dissipation are available in the model. Laplacian and biharmonic frictions are commonly used:

\[ D_V = A_h \nabla^2 v + A_v \frac{\partial^2 v}{\partial z^2} + A_4 \nabla^4 v \]  

(1.41)

where \( A_h \) and \( A_v \) are (constant) horizontal and vertical viscosity coefficients and \( A_4 \) is the horizontal coefficient for biharmonic friction. These coefficients are the same for all velocity components.

Tracers

The mixing terms for the temperature and salinity equations have a similar form to that of momentum except that the diffusion tensor can be non-diagonal and have varying coefficients.

\[ D_{T,S} = \nabla \cdot [K \nabla (T, S)] + K_4 \nabla^4 (T, S) \]  

(1.42)

where \( K \) is the diffusion tensor and the \( K_4 \) horizontal coefficient for biharmonic diffusion. In the simplest case where the subgrid-scale fluxes of heat and salt are parameterized with constant horizontal and vertical diffusion coefficients, \( K \), reduces to a diagonal matrix with constant coefficients:

\[ K = \begin{pmatrix} K_h & 0 & 0 \\ 0 & K_h & 0 \\ 0 & 0 & K_v \end{pmatrix} \]  

(1.43)
where \( K_h \) and \( K_v \) are the horizontal and vertical diffusion coefficients. These coefficients are the same for all tracers (temperature, salinity ...).

### 1.3.8 Vector invariant form

For some purposes it is advantageous to write momentum advection in eq (1.1) and (1.2) in the (so-called) ‘vector invariant’ form:

\[
\frac{D\vec{v}}{Dt} = \frac{\partial \vec{v}}{\partial t} + (\nabla \times \vec{v}) \times \vec{v} + \nabla \left[ \frac{1}{2} (\vec{v} \cdot \vec{v}) \right]
\] (1.44)

This permits alternative numerical treatments of the non-linear terms based on their representation as a vorticity flux. Because gradients of coordinate vectors no longer appear on the rhs of (1.44), explicit representation of the metric terms in (1.29), (1.30) and (1.31), can be avoided: information about the geometry is contained in the areas and lengths of the volumes used to discretize the model.

### 1.3.9 Adjoint

Tangent linear and adjoint counterparts of the forward model are described in Section 7.

### 1.4 Appendix ATMOSPHERE

#### 1.4.1 Hydrostatic Primitive Equations for the Atmosphere in Pressure Coordinates

The hydrostatic primitive equations (HPE’s) in \( p \)–coordinates are:

\[
\frac{D\vec{v}}{Dt} + f\hat{k} \times \vec{v} + \nabla_p \phi = \vec{F}
\]

\[
\frac{\partial \phi}{\partial p} + \alpha = 0
\] (1.46)

\[
\nabla_p \cdot \vec{v}_h + \frac{\partial \omega}{\partial p} = 0
\] (1.47)

\[
p\alpha = RT
\] (1.48)

\[
c_v \frac{DT}{Dt} + p \frac{D\alpha}{Dt} = Q
\] (1.49)

where \( \vec{v}_h = (u, v, 0) \) is the ‘horizontal’ (on pressure surfaces) component of velocity, \( \frac{D}{Dt} = \frac{\partial}{\partial t} + \vec{v}_h \cdot \nabla + \omega \frac{\partial}{\partial p} \) is the total derivative, \( f = 2\Omega \sin \phi \) is the Coriolis parameter, \( \phi = gz \) is the geopotential, \( \alpha = 1/\rho \) is the specific volume, \( \omega = \frac{\partial p}{\partial t} \) is the vertical velocity in the \( p \)–coordinate. Equation (1.49) is the first law of thermodynamics where internal energy \( e = c_v T \), \( T \) is temperature, \( Q \) is the rate of heating per unit mass and \( p \frac{D\alpha}{Dt} \) is the work done by the fluid in compressing.

It is convenient to cast the heat equation in terms of potential temperature \( \theta \) so that it looks more like a generic conservation law. Differentiating (1.48) we get:

\[
p \frac{D\alpha}{Dt} + \alpha \frac{Dp}{Dt} = R \frac{DT}{Dt}
\]

which, when added to the heat equation (1.49) and using \( c_p = c_v + R \), gives:

\[
c_p \frac{DT}{Dt} - \alpha \frac{Dp}{Dt} = Q
\] (1.50)
Potential temperature is defined:

$$\theta = T\left(\frac{p_c}{p}\right)^\kappa$$  \hspace{1cm} (1.51)

where $p_c$ is a reference pressure and $\kappa = R/c_p$. For convenience we will make use of the Exner function $\Pi(p)$ which is defined by:

$$\Pi(p) = c_p\left(\frac{p}{p_c}\right)^\kappa$$ \hspace{1cm} (1.52)

The following relations will be useful and are easily expressed in terms of the Exner function:

$$c_p T = \Pi \theta ; \quad \frac{\partial \Pi}{\partial p} = \frac{\kappa \Pi}{p} ; \quad \alpha = \frac{\kappa \Pi \theta}{p} = \frac{\partial \Pi}{\partial p} \theta ; \quad \frac{D \Pi}{Dt} = \frac{\partial \Pi}{\partial p} \frac{Dp}{Dt}$$

where $b = \frac{\partial \Pi}{\partial p} \theta$ is the buoyancy.

The heat equation is obtained by noting that

$$c_p T = \Pi \theta ; \quad \frac{D \theta}{Dt} = \Pi \frac{D \Pi}{Dt} + \alpha \frac{Dp}{Dt}$$

and on substituting into (1.50) gives:

$$\Pi \frac{D \theta}{Dt} = Q$$ \hspace{1cm} (1.53)

which is in conservative form.

For convenience in the model we prefer to step forward (1.53) rather than (1.49).

### 1.4.1.1 Boundary conditions

The upper and lower boundary conditions are:

- **at the top**: $p = 0$, $\omega = \frac{Dp}{Dt} = 0$ \hspace{1cm} (1.54)
- **at the surface**: $p = p_s$, $\phi = \phi_{topo} = g Z_{topo}$ \hspace{1cm} (1.55)

In $p$–coordinates, the upper boundary acts like a solid boundary ($\omega = 0$); in $z$–coordinates the lower boundary is analogous to a free surface ($\phi$ is imposed and $\omega \neq 0$).

### 1.4.1.2 Splitting the geopotential

For the purposes of initialization and reducing round-off errors, the model deals with perturbations from reference (or ‘standard’) profiles. For example, the hydrostatic geopotential associated with the resting atmosphere is not dynamically relevant and can therefore be subtracted from the equations. The equations written in terms of perturbations are obtained by substituting the following definitions into the previous model equations:

$$\theta = \theta_o + \theta'$$ \hspace{1cm} (1.56)

$$\alpha = \alpha_o + \alpha'$$ \hspace{1cm} (1.57)

$$\phi = \phi_o + \phi'$$ \hspace{1cm} (1.58)

The reference state (indicated by subscript ‘$o$’) corresponds to horizontally homogeneous atmosphere at rest ($\theta_o, \alpha_o, \phi_o$) with surface pressure $p_o(x, y)$ that satisfies $\phi_o(p_o) = g Z_{topo}$, defined:

$$\theta_o(p) = f^n(p)$$
\[ \alpha_o(p) = \Pi_p \theta_o \]
\[ \phi_o(p) = \phi_{topo} - \int_{p_0}^p \alpha_o dp \]

The final form of the HPE’s in \( p \)-coordinates is then:

\[
\frac{D\vec{v}_h}{Dt} + f\hat{k} \times \vec{v}_h + \nabla_p \phi' = \vec{F} \tag{1.59}
\]
\[
\frac{\partial \phi'}{\partial p} + \alpha' = 0 \tag{1.60}
\]
\[
\nabla_p \cdot \vec{v}_h + \frac{\partial \omega}{\partial p} = 0 \tag{1.61}
\]
\[
\frac{\partial \Pi}{\partial p} \theta' = \alpha' \tag{1.62}
\]
\[
\frac{D\theta}{Dt} = \frac{Q}{\Pi} \tag{1.63}
\]

1.5 Appendix OCEAN

1.5.1 Equations of Motion for the Ocean

We review here the method by which the standard (Boussinesq, incompressible) HPE’s for the ocean written in \( z \)-coordinates are obtained. The non-Boussinesq equations for oceanic motion are:

\[
\frac{D\vec{v}_h}{Dt} + f\hat{k} \times \vec{v}_h + \frac{1}{\rho} \nabla_z p = \vec{F} \tag{1.64}
\]
\[
\epsilon_{nh} \frac{Dw}{Dt} + g + \frac{1}{\rho} \frac{\partial p}{\partial z} = \epsilon_{nh} F_w \tag{1.65}
\]
\[
\frac{1}{\rho} \frac{D\rho}{Dt} + \nabla_z \cdot \vec{v}_h + \frac{\partial w}{\partial z} = 0 \tag{1.66}
\]
\[
\rho = \rho(\theta, S, p) \tag{1.67}
\]
\[
\frac{D\theta}{Dt} = Q_\theta \tag{1.68}
\]
\[
\frac{DS}{Dt} = Q_s \tag{1.69}
\]

These equations permit acoustics modes, inertia-gravity waves, non-hydrostatic motions, a geostrophic (Rossby) mode and a thermohaline mode. As written, they cannot be integrated forward consistently - if we step \( \rho \) forward in (1.66), the answer will not be consistent with that obtained by stepping (1.68) and (1.69) and then using (1.67) to yield \( \rho \). It is therefore necessary to manipulate the system as follows. Differentiating the EOS (equation of state) gives:

\[
\frac{D\rho}{Dt} = \frac{\partial \rho}{\partial \theta} \left|_{S,P} \right. \frac{D\theta}{Dt} + \frac{\partial \rho}{\partial S} \left|_{\theta,P} \right. \frac{DS}{Dt} + \frac{\partial \rho}{\partial p} \left|_{\theta,S} \right. \frac{Dp}{Dt} \tag{1.70}
\]

Note that \( \frac{\partial \rho}{\partial p} = \frac{1}{c_s^2} \) is the reciprocal of the sound speed (\( c_s \)) squared. Substituting into (1.66) gives:

\[
\frac{1}{\rho c_s^2} \frac{Dp}{Dt} + \nabla_z \cdot \vec{v} + \partial_z w \approx 0 \tag{1.71}
\]
where we have used an approximation sign to indicate that we have assumed adiabatic motion, dropping the \( \frac{D\theta}{Dt} \) and \( \frac{DS}{Dt} \). Replacing (1.66) with (1.71) yields a system that can be explicitly integrated forward:

\[
\frac{D\vec{v}_h}{Dt} + f\hat{k} \times \vec{v}_h + \frac{1}{\rho} \nabla_z p = \vec{f} \quad (1.72)
\]

\[
\epsilon_{nh} \frac{Dw}{Dt} + g + \frac{1}{\rho} \frac{\partial p}{\partial z} = \epsilon_{nh} F_w \quad (1.73)
\]

\[
\frac{1}{\rho c_s^2} \frac{Dp}{Dt} + \nabla_z \cdot \vec{v}_h + \frac{\partial w}{\partial z} = 0 \quad (1.74)
\]

\[
\rho = \rho(\theta, S, p) \quad (1.75)
\]

\[
\frac{D\theta}{Dt} = Q_\theta \quad (1.76)
\]

\[
\frac{DS}{Dt} = Q_s \quad (1.77)
\]

### 1.5.1.1 Compressible z-coordinate equations

Here we linearize the acoustic modes by replacing \( \rho \) with \( \rho_o(z) \) wherever it appears in a product (ie. non-linear term) - this is the ‘Boussinesq assumption’. The only term that then retains the full variation in \( \rho \) is the gravitational acceleration:

\[
\frac{D\vec{v}_h}{Dt} + f\hat{k} \times \vec{v}_h + \frac{1}{\rho_o} \nabla_z p = \vec{f} \quad (1.78)
\]

\[
\epsilon_{nh} \frac{Dw}{Dt} + \frac{g \rho_o}{\rho_o} + \frac{1}{\rho_o} \frac{\partial p}{\partial z} = \epsilon_{nh} F_w \quad (1.79)
\]

\[
\frac{1}{\rho_o c_s^2} \frac{Dp}{Dt} + \nabla_z \cdot \vec{v}_h + \frac{\partial w}{\partial z} = 0 \quad (1.80)
\]

\[
\rho = \rho(\theta, S, p_o) \quad (1.81)
\]

\[
\frac{D\theta}{Dt} = Q_\theta \quad (1.82)
\]

\[
\frac{DS}{Dt} = Q_s \quad (1.83)
\]

These equations still retain acoustic modes. But, because the “compressible” terms are linearized, the pressure equation (1.80) can be integrated implicitly with ease (the time-dependent term appears as a Helmholtz term in the non-hydrostatic pressure equation). These are the truly compressible Boussinesq equations. Note that the EOS must have the same pressure dependency as the linearized pressure term, ie. \( \left. \frac{\partial \rho}{\partial p} \right|_{\theta, S} = \frac{1}{c_s^2} \), for consistency.

### 1.5.1.2 ‘Anelastic’ z-coordinate equations

The anelastic approximation filters the acoustic mode by removing the time-dependency in the continuity (now pressure-) equation (1.80). This could be done simply by noting that \( \frac{Dp}{Dt} \approx -g \rho_o \frac{Dz}{Dt} = -g \rho_o w \), but this leads to an inconsistency between continuity and EOS. A better solution is to change the dependency on pressure in the EOS by splitting the pressure into a reference function of height and a perturbation:

\[
\rho = \rho(\theta, S, p_o(z) + \epsilon_s p')
\]
Remembering that the term $\frac{Dp}{Dt}$ in continuity comes from differentiating the EOS, the continuity equation then becomes:

$$\frac{1}{\rho_o c_s^2} \left( \frac{Dp_o}{Dt} + \epsilon_s \frac{Dp'}{Dt} \right) + \nabla_z \cdot \vec{v}_h + \frac{\partial w}{\partial z} = 0$$

If the time- and space-scales of the motions of interest are longer than those of acoustic modes, then $\frac{Dp'}{Dt} << (\frac{Dp_o}{Dt}, \nabla \cdot \vec{v}_h)$ in the continuity equations and $\frac{Dp'}{Dt} \theta, S << \frac{Dp}{Dt} \theta, S$ in the EOS (1.70). Thus we set $\epsilon_s = 0$, removing the dependency on $p'$ in the continuity equation and EOS. Expanding $\frac{Dp_o(z)}{Dt} = -g \rho_o \omega$ then leads to the anelastic continuity equation:

$$\nabla_z \cdot \vec{v}_h + \frac{\partial w}{\partial z} - \frac{g}{c_s^2} \omega = 0 \quad (1.84)$$

A slightly different route leads to the quasi-Boussinesq continuity equation where we use the scaling $\frac{Dp'}{Dt} + \nabla \cdot \rho' \vec{v} << \nabla_3 \cdot \rho_o \vec{v}$ yielding:

$$\nabla_z \cdot \vec{v}_h + \frac{1}{\rho_o} \frac{\partial (\rho_o \omega)}{\partial z} = 0 \quad (1.85)$$

Equations (1.84) and (1.85) are in fact the same equation if:

$$\frac{1}{\rho} \frac{\partial \rho}{\partial z} = -\frac{g}{c_s^2}$$

Again, note that if $\rho_o$ is evaluated from prescribed $\theta_o$ and $S_o$ profiles, then the EOS dependency on $p_o$ and the term $\frac{D}{c_s^2}$ in continuity should be referred to those same profiles. The full set of ‘quasi-Boussinesq’ or ‘anelastic’ equations for the ocean are then:

$$\frac{D\vec{v}_h}{Dt} + f \vec{k} \times \vec{v}_h + \frac{1}{\rho_o} \nabla_z p = \vec{f} \quad (1.86)$$

$$\epsilon_{nh} \frac{D\omega}{Dt} + \frac{g \rho}{\rho_o} + \frac{1}{\rho_o} \frac{\partial p}{\partial z} = \epsilon_{nh} F_w \quad (1.87)$$

$$\nabla_z \cdot \vec{v}_h + \frac{1}{\rho_o} \frac{\partial (\rho_o \omega)}{\partial z} = 0 \quad (1.88)$$

$$\rho = \rho(\theta, S, p_o(z)) \quad (1.89)$$

$$\frac{D\theta}{Dt} = Q_\theta \quad (1.90)$$

$$\frac{DS}{Dt} = Q_s \quad (1.91)$$

### 1.5.1.3 Incompressible z-coordinate equations

Here, the objective is to drop the depth dependence of $\rho_o$ and so, technically, to also remove the dependence of $\rho$ on $p_o$. This would yield the “truly” incompressible Boussinesq equations:

$$\frac{D\vec{v}_h}{Dt} + f \vec{k} \times \vec{v}_h + \frac{1}{\rho_c} \nabla_z p = \vec{f} \quad (1.92)$$

$$\epsilon_{nh} \frac{D\omega}{Dt} + \frac{g \rho}{\rho_c} + \frac{1}{\rho_c} \frac{\partial p}{\partial z} = \epsilon_{nh} F_w \quad (1.93)$$

$$\nabla_z \cdot \vec{v}_h + \frac{\partial w}{\partial z} = 0 \quad (1.94)$$
\[ \rho = \rho(\theta, S) \]  
(1.95)

\[ \frac{D\theta}{Dt} = Q_\theta \]  
(1.96)

\[ \frac{DS}{Dt} = Q_s \]  
(1.97)

where \( \rho_c \) is a constant reference density of water.

### 1.5.1.4 Compressible non-divergent equations

The above “incompressible” equations are incompressible in both the flow and the density. In many oceanic applications, however, it is important to retain compressibility effects in the density. To do this we must split the density thus:

\[ \rho = \rho_o + \rho' \]

We then assert that variations with depth of \( \rho_o \) are unimportant while the compressible effects in \( \rho' \) are:

\[ \rho_o = \rho_c \]
\[ \rho' = \rho(\theta, S, p_o(z)) - \rho_c \]

This then yields what we can call the semi-compressible Boussinesq equations:

\[ \frac{D\hat{v}_h}{Dt} + f \mathbf{k} \times \hat{v}_h + \frac{1}{\rho_c} \nabla_z p' = \mathbf{F} \]  
(1.98)

\[ \epsilon_{nh} \frac{Dw}{Dt} + \frac{g \rho'}{\rho_c} + \frac{1}{\rho_c} \frac{\partial \rho'}{\partial z} = \epsilon_{nh} F_w \]  
(1.99)

\[ \nabla_z \cdot \hat{v}_h + \frac{\partial w}{\partial z} = 0 \]  
(1.100)

\[ \rho' = \rho(\theta, S, p_o(z)) - \rho_c \]  
(1.101)

\[ \frac{D\theta}{Dt} = Q_\theta \]  
(1.102)

\[ \frac{DS}{Dt} = Q_s \]  
(1.103)

Note that the hydrostatic pressure of the resting fluid, including that associated with \( \rho_c \), is subtracted out since it has no effect on the dynamics.

Though necessary, the assumptions that go into these equations are messy since we essentially assume a different EOS for the reference density and the perturbation density. Nevertheless, it is the hydrostatic (\( \epsilon_{nh} = 0 \)) form of these equations that are used throughout the ocean modeling community and referred to as the primitive equations (HPE’s).

### 1.6 Appendix OPERATORS

#### 1.6.1 Coordinate systems

##### 1.6.1.1 Spherical coordinates

In spherical coordinates, the velocity components in the zonal, meridional and vertical direction respectively, are given by:

\[ u = r \cos \phi \frac{D\lambda}{Dt} \]
\[ v = r \frac{D\varphi}{Dt} \]
\[ \dot{r} = \frac{Dr}{Dt} \]

(see Figure 1.20) Here \( \varphi \) is the latitude, \( \lambda \) the longitude, \( r \) the radial distance of the particle from the center of the earth, \( \Omega \) is the angular speed of rotation of the Earth and \( D/Dt \) is the total derivative.

The 'grad' (\( \nabla \)) and 'div' (\( \nabla \cdot \)) operators are defined by, in spherical coordinates:

\[
\nabla \equiv \left( \frac{1}{r \cos \varphi} \frac{\partial}{\partial \lambda}, \frac{1}{r \cos \varphi} \frac{\partial}{\partial \varphi}, \frac{\partial}{\partial r} \right)
\]

\[
\nabla \cdot v \equiv \frac{1}{r \cos \varphi} \left\{ \frac{\partial}{\partial \lambda} u + \frac{\partial}{\partial \varphi} (v \cos \varphi) \right\} + \frac{1}{r^2} \frac{\partial}{\partial r} \left( r^2 \dot{r} \right)
\]
Figure 1.20: Spherical polar coordinates: longitude $\lambda$, latitude $\varphi$ and $r$ the distance from the center.
CHAPTER
TWO

DISCRETIZATION AND ALGORITHM

This chapter lays out the numerical schemes that are employed in the core MITgcm algorithm. Whenever possible links are made to actual program code in the MITgcm implementation. The chapter begins with a discussion of the temporal discretization used in MITgcm. This discussion is followed by sections that describe the spatial discretization. The schemes employed for momentum terms are described first, afterwards the schemes that apply to passive and dynamically active tracers are described.

2.1 Notation

Because of the particularity of the vertical direction in stratified fluid context, in this chapter, the vector notations are mostly used for the horizontal component: the horizontal part of a vector is simply written $\vec{v}$ (instead of $v_h$ or $\vec{v}_h$ in chapter 1) and a 3D vector is simply written $\vec{v}$ (instead of $\hat{v}$ in chapter 1).

The notations we use to describe the discrete formulation of the model are summarized as follows.

General notation:

$\Delta x, \Delta y, \Delta r$ grid spacing in X, Y, R directions

$A_c, A_w, A_s, A_\zeta$: horizontal area of a grid cell surrounding $\theta, u, v, \zeta$ point

$\mathcal{V}_u, \mathcal{V}_v, \mathcal{V}_w, \mathcal{V}_\theta$: Volume of the grid box surrounding $u, v, w, \theta$ point

$i, j, k$: current index relative to X, Y, R directions

Basic operators:

$\delta_i: \delta_i \Phi = \Phi_{i+1/2} - \Phi_{i-1/2}$

$-i: \Phi^i = (\Phi_{i+1/2} + \Phi_{i-1/2})/2$

$\delta_x: \delta_x \Phi = \frac{1}{\Delta x} \delta_i \Phi$

$\nabla =$ horizontal gradient operator: $\nabla \Phi = \{\delta_x \Phi, \delta_y \Phi\}$
\( \nabla \cdot = \) horizontal divergence operator: \( \nabla \cdot \vec{f} = \frac{1}{\mathcal{A}} \{ \delta_i \Delta y f_x + \delta_j \Delta x f_y \} \)

\( \nabla^2 = \) horizontal Laplacian operator: \( \nabla^2 \Phi = \nabla \cdot \nabla \Phi \)

### 2.2 Time-stepping

The equations of motion integrated by the model involve four prognostic equations for flow, \( u \) and \( v \), temperature, \( \theta \), and salt/moisture, \( S \), and three diagnostic equations for vertical flow, \( w \), density/buoyancy, \( \rho/\beta \), and pressure/geo-potential, \( \phi_{\text{hyd}} \). In addition, the surface pressure or height may be described by either a prognostic or diagnostic equation and if non-hydrostatics terms are included then a diagnostic equation for non-hydrostatic pressure is also solved. The combination of prognostic and diagnostic equations requires a model algorithm that can march forward prognostic variables while satisfying constraints imposed by diagnostic equations.

Since the model comes in several flavors and formulation, it would be confusing to present the model algorithm exactly as written into code along with all the switches and optional terms. Instead, we present the algorithm for each of the basic formulations which are:

1. the semi-implicit pressure method for hydrostatic equations with a rigid-lid, variables co-located in time and with Adams-Bashforth time-stepping;
2. as 1 but with an implicit linear free-surface;
3. as 1 or 2 but with variables staggered in time;
4. as 1 or 2 but with non-hydrostatic terms included;
5. as 2 or 3 but with non-linear free-surface.

In all the above configurations it is also possible to substitute the Adams-Bashforth with an alternative time-stepping scheme for terms evaluated explicitly in time. Since the over-arching algorithm is independent of the particular time-stepping scheme chosen we will describe first the over-arching algorithm, known as the pressure method, with a rigid-lid model in Section 2.3. This algorithm is essentially unchanged, apart for some coefficients, when the rigid lid assumption is replaced with a linearized implicit free-surface, described in Section 2.4. These two flavors of the pressure-method encompass all formulations of the model as it exists today. The integration of explicit in time terms is out-lined in Section 2.5 and put into the context of the overall algorithm in Section 2.7 and Section 2.8. Inclusion of non-hydrostatic terms requires applying the pressure method in three dimensions instead of two and this algorithm modification is described in Section 2.9. Finally, the free-surface equation may be treated more exactly, including non-linear terms, and this is described in Section 2.10.2.

#### 2.3 Pressure method with rigid-lid

The horizontal momentum and continuity equations for the ocean ((1.98) and (1.100)), or for the atmosphere ((1.45) and (1.47)), can be summarized by:

\[
\begin{align*}
\partial_t u + g \partial_x \eta &= G_u \\
\partial_t v + g \partial_y \eta &= G_v \\
\partial_x u + \partial_y v + \partial_z w &= 0
\end{align*}
\]

where we are adopting the oceanic notation for brevity. All terms in the momentum equations, except for surface pressure gradient, are encapsulated in the \( G \) vector. The continuity equation, when integrated over the fluid depth, \( H \), and with the rigid-lid/no normal flow boundary conditions applied, becomes:

\[
\partial_x H \hat{u} + \partial_y H \hat{v} = 0
\]

(2.1)
Here, $\hat{u} = \int u \, dz$ is the depth integral of $u$, similarly for $\hat{v}$. The rigid-lid approximation sets $w = 0$ at the lid so that it does not move but allows a pressure to be exerted on the fluid by the lid. The horizontal momentum equations and vertically integrated continuity equation are be discretized in time and space as follows:

\begin{align*}
    u^{n+1} + \Delta t g \partial_x \eta^{n+1} & = u^n + \Delta t G_u^{(n+1)/2} \\
    v^{n+1} + \Delta t g \partial_y \eta^{n+1} & = v^n + \Delta t G_v^{(n+1)/2} \\
    \partial_x H \hat{u}^{n+1} + \partial_y H \hat{v}^{n+1} & = 0
\end{align*}

(2.2) \quad (2.3) \quad (2.4)

As written here, terms on the LHS all involve time level $n+1$ and are referred to as implicit; the implicit backward time stepping scheme is being used. All other terms in the RHS are explicit in time. The thermodynamic quantities are integrated forward in time in parallel with the flow and will be discussed later. For the purposes of describing the pressure method it suffices to say that the hydrostatic pressure gradient is explicit and so can be included in the vector $G$.

Substituting the two momentum equations into the depth integrated continuity equation eliminates $u^{n+1}$ and $v^{n+1}$ yielding an elliptic equation for $\eta^{n+1}$. Equations (2.2), (2.3) and (2.4) can then be re-arranged as follows:

\begin{align*}
    u^* & = u^n + \Delta t G_u^{(n+1)/2} \\
    v^* & = v^n + \Delta t G_v^{(n+1)/2} \\
    \partial_x \Delta t g H \partial_x \eta^{n+1} + \partial_y \Delta t g H \partial_y \eta^{n+1} & = \partial_x \hat{u}^* + \partial_y \hat{v}^* \\
    u^{n+1} & = u^* - \Delta t g \partial_x \eta^{n+1} \\
    v^{n+1} & = v^* - \Delta t g \partial_y \eta^{n+1}
\end{align*}

(2.5) \quad (2.6) \quad (2.7) \quad (2.8) \quad (2.9)

Equations (2.5) to (2.9), solved sequentially, represent the pressure method algorithm used in the model. The essence of the pressure method lies in the fact that any explicit prediction for the flow would lead to a divergence flow field so a pressure field must be found that keeps the flow non-divergent over each step of the integration. The particular location in time of the pressure field is somewhat ambiguous; in Figure 2.1 we depicted as co-located with the future flow field (time level $n+1$) but it could equally have been drawn as staggered in time with the flow.

The correspondence to the code is as follows:

- the prognostic phase, equations (2.5) and (2.6), stepping forward $u^n$ and $v^n$ to $u^*$ and $v^*$ is coded in timestep.F
- the vertical integration, $H \hat{u}^*$ and $H \hat{v}^*$, divergence and inversion of the elliptic operator in equation (2.7) is coded in solve_for_pressure.F
- finally, the new flow field at time level $n+1$ given by equations (2.8) and (2.9) is calculated in correction_step.F

The calling tree for these routines is as follows:

**Pressure method calling tree**

```plaintext
FORWARD_STEP
   DYNAMICS
      TIMESTEP   \quad u^*, v^* (2.5), (2.6)
   SOLVE_FOR_PRESSURE
      CALC_DIV_GHAT \quad H \hat{u}^*, H \hat{v}^* (2.7)
      CG2D
      MOMENTUM_CORRECTION_STEP
         CALC_GRAD_PHI_SURF \quad \nabla \eta^{n+1}
```

2.3. Pressure method with rigid-lid
Figure 2.1: A schematic of the evolution in time of the pressure method algorithm. A prediction for the flow variables at time level \( n + 1 \) is made based only on the explicit terms, \( G^{(n+1/2)} \), and denoted \( u^*, v^* \). Next, a pressure field is found such that \( u^{n+1}, v^{n+1} \) will be non-divergent. Conceptually, the \( * \) quantities exist at time level \( n + 1 \) but they are intermediate and only temporary.

In general, the horizontal momentum time-stepping can contain some terms that are treated implicitly in time, such as the vertical viscosity when using the backward time-stepping scheme (\texttt{implicitViscosity = .TRUE.}). The method used to solve those implicit terms is provided in Section 2.6, and modifies equations (2.2) and (2.3) to give:

\[
\begin{align*}
    u^{n+1} - \Delta t \partial_x A \partial_x u^{n+1} + \Delta t g \partial_x \eta^{n+1} &= u^n + \Delta t G_u^{(n+1/2)} \\
    v^{n+1} - \Delta t \partial_y A \partial_y v^{n+1} + \Delta t g \partial_y \eta^{n+1} &= v^n + \Delta t G_v^{(n+1/2)}
\end{align*}
\]

### 2.4 Pressure method with implicit linear free-surface

The rigid-lid approximation filters out external gravity waves subsequently modifying the dispersion relation of barotropic Rossby waves. The discrete form of the elliptic equation has some zero eigenvalues which makes it a potentially tricky or inefficient problem to solve.

The rigid-lid approximation can be easily replaced by a linearization of the free-surface equation which can be written:

\[
\partial_t \eta + \partial_x H \hat{u} + \partial_y H \hat{v} = \mathcal{P} - \mathcal{E} + \mathcal{R}
\]

which differs from the depth integrated continuity equation with rigid-lid ((2.1)) by the time-dependent term and freshwater source term.

Equation (2.4) in the rigid-lid pressure method is then replaced by the time discretization of (2.10) which is:

\[
\eta^{n+1} + \Delta t \partial_x H \hat{u}^{n+1} + \Delta t \partial_y H \hat{v}^{n+1} = \eta^n + \Delta t(\mathcal{P} - \mathcal{E})
\]

where the use of flow at time level \( n + 1 \) makes the method implicit and backward in time. This is the preferred scheme since it still filters the fast unresolved wave motions by damping them. A centered scheme, such as Crank-Nicholson (see Section 2.10.1), would alias the energy of the fast modes onto slower modes of motion.
As for the rigid-lid pressure method, equations (2.2), (2.3) and (2.11) can be re-arranged as follows:

\[ u^* = u^n + \Delta t G_u^{(n+1/2)} \]  
\[ v^* = v^n + \Delta t G_v^{(n+1/2)} \]  
\[ \eta^* = \epsilon_f \eta^n + \Delta t (P - E) - \Delta t (\partial_x H u^* + \partial_y H v^*) \]  
\[ \partial_x g H \partial_x \eta^{n+1} + \partial_y g H \partial_y \eta^{n+1} = -\frac{\eta^n}{\Delta t^2} \]  
\[ u^{n+1} = u^* - \Delta t g \partial_x \eta^{n+1} \]  
\[ v^{n+1} = v^* - \Delta t g \partial_y \eta^{n+1} \]

Equations (2.12) to (2.17), solved sequentially, represent the pressure method algorithm with a backward implicit, linearized free surface. The method is still formally a pressure method because in the limit of large \( \Delta t \) the rigid-lid method is recovered. However, the implicit treatment of the free-surface allows the flow to be divergent and for the surface pressure/elevation to respond on a finite time-scale (as opposed to instantly). To recover the rigid-lid formulation, we use a switch-like variable, \( \epsilon_f \) (freesurfFac), which selects between the free-surface and rigid-lid; \( \epsilon_f = 1 \) allows the free-surface to evolve; \( \epsilon_f = 0 \) imposes the rigid-lid. The evolution in time and location of variables is exactly as it was for the rigid-lid model so that Figure 2.1 is still applicable. Similarly, the calling sequence, given here, is as for the pressure-method.

### 2.5 Explicit time-stepping: Adams-Bashforth

In describing the pressure method above we deferred describing the time discretization of the explicit terms. We have historically used the quasi-second order Adams-Bashforth method (AB-II) for all explicit terms in both the momentum and tracer equations. This is still the default mode of operation but it is now possible to use alternate schemes for tracers (see Section 2.16), or a 3rd order Adams-Bashforth method (AB-III). In the previous sections, we summarized an explicit scheme as:

\[ \tau^* = \tau^n + \Delta t G^{(n+1/2)} \]  

where \( \tau \) could be any prognostic variable (\( u, v, \theta \) or \( S \)) and \( \tau^* \) is an explicit estimate of \( \tau^{n+1} \) and would be exact if not for implicit-in-time terms. The parenthesis about \( n + 1/2 \) indicates that the term is explicit and extrapolated forward in time. Below we describe in more detail the AB-II and AB-III schemes.

#### 2.5.1 Adams-Bashforth II

The quasi-second order Adams-Bashforth scheme is formulated as follows:

\[ G^{(n+1/2)} = (3/2 + \epsilon_{AB}) G^n - (1/2 + \epsilon_{AB}) G^{n-1} \]  

This is a linear extrapolation, forward in time, to \( t = (n + 1/2 + \epsilon_{AB}) \Delta t \). An extrapolation to the mid-point in time, \( t = (n + 1/2) \Delta t \), corresponding to \( \epsilon_{AB} = 0 \), would be second order accurate but is weakly unstable for oscillatory terms. A small but finite value for \( \epsilon_{AB} \) stabilizes the method. Strictly speaking, damping terms such as diffusion and dissipation, and fixed terms (forcing), do not need to be inside the Adams-Bashforth extrapolation. However, in the current code, it is simpler to include these terms and this can be justified if the flow and forcing evolves smoothly. Problems can, and do, arise when forcing or motions are high frequency and this corresponds to a reduced stability compared to a simple forward time-stepping of such terms. The model offers the possibility to leave terms outside the Adams-Bashforth extrapolation, by turning off the logical flag forcing_In_AB (parameter file data, namelist PARM01, default value = TRUE) and then setting tracForcingOutAB (default=0), momForcingOutAB (default=0), and
momDissip_In_AB (parameter file data, namelist PARM01, default value = TRUE), respectively for the tracer terms, momentum forcing terms, and the dissipation terms.

A stability analysis for an oscillation equation should be given at this point.

A stability analysis for a relaxation equation should be given at this point.

2.5.2 Adams-Bashforth III

The 3rd order Adams-Bashforth time stepping (AB-III) provides several advantages (see, e.g., Durran 1991 [Dur91]) compared to the default quasi-second order Adams-Bashforth method:

- higher accuracy;
- stable with a longer time-step;
- no additional computation (just requires the storage of one additional time level).

The 3rd order Adams-Bashforth can be used to extrapolate forward in time the tendency (replacing (2.19)) as:

\[ G^{(n+1/2)}_r = (1 + \alpha_{AB} + \beta_{AB})G^n_r - (\alpha_{AB} + 2\beta_{AB})G^{n-1}_r + \beta_{AB}G^{n-2}_r \]  (2.20)

3rd order accuracy is obtained with \((\alpha_{AB}, \beta_{AB}) = (1/2, 5/12)\). Note that selecting \((\alpha_{AB}, \beta_{AB}) = (1/2 + \epsilon_{AB}, 0)\) one recovers AB-II. The AB-III time stepping improves the stability limit for an oscillatory problem like advection or Coriolis. As seen from Figure 2.3, it remains stable up to a CFL of 0.72, compared to only 0.50 with AB-II and \(\epsilon_{AB} = 0.1\). It is interesting to note that the stability limit can be further extended up to a CFL of 0.786 for an oscillatory problem (see Figure 2.3) using \((\alpha_{AB}, \beta_{AB}) = (0.5, 0.281)\) but then the scheme is only second order accurate.

However, the behavior of the AB-III for a damping problem (like diffusion) is less favorable, since the stability limit is reduced to 0.54 only (and 0.64 with \(\beta_{AB} = 0.281\)) compared to 1.0 (and 0.9 with \(\epsilon_{AB} = 0.1\)) with the AB-II (see Figure 2.4).

A way to enable the use of a longer time step is to keep the dissipation terms outside the AB extrapolation (setting momDissip_In_AB to .FALSE. in main parameter file data, namelist PARM03, thus returning to a simple forward time-stepping for dissipation, and to use AB-III only for advection and Coriolis terms.

The AB-III time stepping is activated by defining the option \#define ALLOW_ADAMSBASHFORTH_3 in CPP_OPTIONS.h. The parameters \(\alpha_{AB}, \beta_{AB}\) can be set from the main parameter file data (namelist PARM03) and their default values correspond to the 3rd order Adams-Bashforth. A simple example is provided in verification/advect_xy/input.ab3_c4.

AB-III is not yet available for the vertical momentum equation (non-hydrostatic) nor for passive tracers.

2.6 Implicit time-stepping: backward method

Vertical diffusion and viscosity can be treated implicitly in time using the backward method which is an intrinsic scheme. Recently, the option to treat the vertical advection implicitly has been added, but not yet tested; therefore, the description hereafter is limited to diffusion and viscosity. For tracers, the time discretized equation is:

\[ \tau^{n+1} - \Delta t \kappa \partial_r \tau^{n+1} = \tau^n + \Delta t G^{(n+1/2)}_r \]  (2.21)

where \(G^{(n+1/2)}_r\) is the remaining explicit terms extrapolated using the Adams-Bashforth method as described above. Equation (2.21) can be split split into:

\[ \tau^* = \tau^n + \Delta t G^{(n+1/2)}_r \]  (2.22)
Figure 2.2: Oscillatory and damping response of quasi-second order Adams-Bashforth scheme for different values of the $\epsilon_{AB}$ parameter (0.0, 0.1, 0.25, from top to bottom) The analytical solution (in black), the physical mode (in blue) and the numerical mode (in red) are represented with a CFL step of 0.1. The left column represents the oscillatory response on the complex plane for CFL ranging from 0.1 up to 0.9. The right column represents the damping response amplitude (y-axis) function of the CFL (x-axis).
Figure 2.3: Oscillatory response of third order Adams-Bashforth scheme for different values of the $(\alpha_{AB}, \beta_{AB})$ parameters. The analytical solution (in black), the physical mode (in blue) and the numerical mode (in red) are represented with a CFL step of 0.1.
Damping response of AB−3 : \( \lambda = 1.0 ; \lambda^*dt = 0.0 \rightarrow 1.1 \) every 0.1

\[ \alpha, \beta = 0.50, 0.000 ; \mu_c = 1.0000 \]

\[ \alpha, \beta = 0.60, 0.000 ; \mu_c = 0.9091 \]

\[ \alpha, \beta = 0.50, 0.417 ; \mu_c = 0.5455 \]

\[ \alpha, \beta = 0.50, 0.281 ; \mu_c = 0.6401 \]

Figure 2.4: Damping response of third order Adams-Bashforth scheme for different values of the \((\alpha_{AB}, \beta_{AB})\) parameters. The analytical solution (in black), the physical mode (in blue) and the numerical mode (in red) are represented with a CFL step of 0.1.

2.6. Implicit time-stepping: backward method
\[
\tau^{n+1} = L^{-1}_\tau(\tau^n)
\]  

(2.23)

where \(L^{-1}_\tau\) is the inverse of the operator

\[
L_\tau = [1 + \Delta t \partial_r \kappa_u \partial_r]
\]

Equation (2.22) looks exactly as (2.18) while (2.23) involves an operator or matrix inversion. By re-arranging (2.21) in this way we have cast the method as an explicit prediction step and an implicit step allowing the latter to be inserted into the overall algorithm with minimal interference.

The calling sequence for stepping forward a tracer variable such as temperature with implicit diffusion is as follows:

**Adams-Bashforth calling tree**

- **FORWARD_STEP**
  - **THERMODYNAMICS**
  - **TEMP_INTEGRATE**
  - **GAD_CALC_RHS**
    - \(G_\theta^n = G_\theta(u, \theta^n)\)
    - either
      - **EXTERNAL_FORCING**
        - \(G_\theta^n = G_\theta^n + Q\)
      - **ADAMS_BASHFORTH2**
        - \(G_\theta^{(n+1)/2} (2.19)\)
    - or
      - **EXTERNAL_FORCING**
        - \(G_\theta^{(n+1)/2} = G_\theta^{(n+1)/2} + Q\)
      - **TIMESTEP_TRACER**
        - \(\tau^* (2.18)\)
      - **IMPLDIFF**
        - \(\tau^{(n+1)} (2.23)\)

In order to fit within the pressure method, the implicit viscosity must not alter the barotropic flow. In other words, it can only redistribute momentum in the vertical. The upshot of this is that although vertical viscosity may be backward implicit and unconditionally stable, no-slip boundary conditions may not be made implicit and are thus cast as a an explicit drag term.

### 2.7 Synchronous time-stepping: variables co-located in time

The Adams-Bashforth extrapolation of explicit tendencies fits neatly into the pressure method algorithm when all state variables are co-located in time. The algorithm can be represented by the sequential solution of the following equations:

\[
G_{\theta,S}^n = G_{\theta,S}(u^n, \theta^n, S^n)
\]  

(2.24)

\[
G_{\theta,S}^{(n+1)/2} = (3/2 + \epsilon_{AB})G_{\theta,S}^n - (1/2 + \epsilon_{AB})G_{\theta,S}^{n-1}
\]  

(2.25)

\[
(\theta^*, S^*) = (\theta^n, S^n) + \Delta t G_{\theta,S}^{(n+1)/2}
\]  

(2.26)

\[
(\theta^{n+1}, S^{n+1}) = L^{-1}_{\theta,S}(\theta^*, S^*)
\]  

(2.27)

\[
\phi_{\text{hyd}}^n = \int b(\theta^n, S^n)dr
\]  

(2.28)

\[
\vec{G}_{\theta}^n = \vec{G}_{\theta}(\vec{v}^n, \phi_{\text{hyd}}^n)
\]  

(2.29)

\[
\vec{G}_{\theta}^{(n+1)/2} = (3/2 + \epsilon_{AB})\vec{G}_{\theta}^n - (1/2 + \epsilon_{AB})\vec{G}_{\theta}^{n-1}
\]  

(2.30)

\[
\vec{v}^* = \vec{v}^n + \Delta t \vec{G}_{\theta}^{(n+1)/2}
\]  

(2.31)
Figure 2.5: A schematic of the explicit Adams-Bashforth and implicit time-stepping phases of the algorithm. All prognostic variables are co-located in time. Explicit tendencies are evaluated at time level $n$ as a function of the state at that time level (dotted arrow). The explicit tendency from the previous time level, $n-1$, is used to extrapolate tendencies to $n + 1/2$ (dashed arrow). This extrapolated tendency allows variables to be stably integrated forward-in-time to render an estimate ($*$-variables) at the $n + 1$ time level (solid arc-arrow). The operator $\mathcal{L}$ formed from implicit-in-time terms is solved to yield the state variables at time level $n + 1$. 

2.7. Synchronous time-stepping: variables co-located in time
\[ \vec{v}^{**} = L_q^{-1}(\vec{v}^*) \]  
(2.32)

\[ \eta^* = \epsilon_f s (\eta^n + \Delta t (P - E)) - \Delta t \nabla \cdot H \vec{v}^{**} \]  
(2.33)

\[ \nabla \cdot g H \nabla \eta^{n+1} - \frac{\epsilon_f s \eta^{n+1}}{\Delta t^2} = -\frac{\eta^*}{\Delta t^2} \]  
(2.34)

\[ \vec{v}^{n+1} = \vec{v}^{**} - \Delta t g \nabla \eta^{n+1} \]  
(2.35)

Figure 2.5 illustrates the location of variables in time and evolution of the algorithm with time. The Adams-Bashforth extrapolation of the tracer tendencies is illustrated by the dashed arrow, the prediction at \( n + 1 \) is indicated by the solid arc. Inversion of the implicit terms, \( L_{q,S}^{-1} \), then yields the new tracer fields at \( n + 1 \). All these operations are carried out in subroutine THERMODYNAMICS and subsidiaries, which correspond to equations (2.24) to (2.27). Similarly illustrated is the Adams-Bashforth extrapolation of accelerations, stepping forward and solving of implicit viscosity and surface pressure gradient terms, corresponding to equations (2.29) to (2.35). These operations are carried out in subroutines DYNAMICS, SOLVE_FOR_PRESSURE and MOMENTUM_CORRECTION_STEP. This, then, represents an entire algorithm for stepping forward the model one time-step. The corresponding calling tree for the overall synchronous algorithm using Adams-Bashforth time-stepping is given below. The place where the model geometry (hFac factors) is updated is added here but is only relevant for the non-linear free-surface algorithm. For completeness, the external forcing, ocean and atmospheric physics have been added, although they are mainly optional.

---

**Synchronous Adams-Bashforth calling tree**

1. **FORWARD_STEP**
   - EXTERNAL_FIELDS_LOAD
   - DO_ATMOSPHERIC_PHYS
   - DO_OCEANIC_PHYS
2. **THERMODYNAMICS**
   - CALC_GT
     - GAD_CALC_RHS \( G^n_\theta = G_\theta(u, \theta^n) \) (2.24)
     - EXTERNAL_FORCING \( G^n_\theta = G_\theta + \mathcal{Q} \)
     - ADAMS_BASHFORTH2 \( G^{(n+1)/2}_\theta \)
   - TIMESTEP_TRACER \( \theta^* \) (2.26)
   - IMPLDIFF \( \theta^{(n+1)} \) (2.27)
3. **DYNAMICS**
   - CALC_PHI_HYD \( \phi^n_{hyd} \) (2.28)
   - MOM_FLUXFORM or MOM_VECINV \( \vec{v}^{*} \) (2.30), (2.31)
   - IMPLDIFF \( \vec{v}^{**} \) (2.32)
4. **UPDATE_R_STAR or UPDATE_SURF_DR** (NonLin-FS only)
5. **SOLVE_FOR_PRESSURE**
   - CALC_DIV_GHAT \( \eta^* \) (2.33)
   - CG2D \( \eta^{n+1} \) (2.34)
6. **MOMENTUM_CORRECTION_STEP**
   - CALC_GRAD_PHI_SURF \( \nabla \eta^{n+1} \)
   - CORRECTION_STEP \( u^{n+1}, v^{n+1} \) (2.35)
7. **TRACERS_CORRECTION_STEP**
   - CYCLE_TRACER \( \theta^{n+1} \)
8. **SHAP_FILT_APPLY_TS or ZONAL_FILT_APPLY_TS**
9. **CONVective_ADJUSTMENT**
2.8 Staggered baroclinic time-stepping

For well-stratified problems, internal gravity waves may be the limiting process for determining a stable time-step. In the circumstance, it is more efficient to stagger in time of thermodynamic variables with the flow. Explicit momentum tendencies are evaluated at time level $n - 1/2$ as a function of the flow field at that time level $n - 1/2$. The explicit tendency from the previous time level, $n - 3/2$, is used to extrapolate tendencies to $n$ (dashed arrow). The hydrostatic pressure/geo-potential $\phi_{hyd}$ is evaluated directly at time level $n$ (vertical arrows) and used with the extrapolated tendencies to step forward the flow variables from $n - 1/2$ to $n + 1/2$ (solid arc-arrow). The implicit-in-time operator $L_{u,v}$ (vertical arrows) is then applied to the previous estimation of the flow field ($*$-variables) and yields to the two velocity components $u, v$ at time level $n + 1/2$. These are then used to calculate the advection term (dashed arc-arrow) of the thermo-dynamics tendencies at time step $n$. The extrapolated thermodynamics tendency, from time level $n - 1$ and $n$ to $n + 1/2$, allows thermodynamic variables to be stably integrated forward-in-time (solid arc-arrow) up to time level $n + 1$.

For well-stratified problems, internal gravity waves may be the limiting process for determining a stable time-step. In the circumstance, it is more efficient to stagger in time of thermodynamic variables with the flow variables. Figure 2.6 illustrates the staggering and algorithm. The key difference between this and Figure 2.5 is that the thermodynamic variables are solved after the dynamics, using the recently updated flow field. This essentially allows the gravity wave terms to leap-frog in time giving second order accuracy and more stability.

The essential change in the staggered algorithm is that the thermodynamics solver is delayed from half a time step, allowing the use of the most recent velocities to compute the advection terms. Once the thermodynamics fields are updated, the hydrostatic pressure is computed to step forward the dynamics. Note that the pressure gradient must also be taken out of the Adams-Bashforth extrapolation. Also, retaining the integer time-levels, $n$ and $n + 1$, does not give a user the sense of where variables are located in time. Instead, we re-write the entire algorithm, (2.24) to (2.35),

---

**Figure 2.6:** A schematic of the explicit Adams-Bashforth and implicit time-stepping phases of the algorithm but with staggering in time of thermodynamic variables with the flow. Explicit momentum tendencies are evaluated at time level $n - 1/2$ as a function of the flow field at that time level $n - 1/2$. The explicit tendency from the previous time level, $n - 3/2$, is used to extrapolate tendencies to $n$ (dashed arrow). The hydrostatic pressure/geo-potential $\phi_{hyd}$ is evaluated directly at time level $n$ (vertical arrows) and used with the extrapolated tendencies to step forward the flow variables from $n - 1/2$ to $n + 1/2$ (solid arc-arrow). The implicit-in-time operator $L_{u,v}$ (vertical arrows) is then applied to the previous estimation of the flow field ($*$-variables) and yields to the two velocity components $u, v$ at time level $n + 1/2$. These are then used to calculate the advection term (dashed arc-arrow) of the thermo-dynamics tendencies at time step $n$. The extrapolated thermodynamics tendency, from time level $n - 1$ and $n$ to $n + 1/2$, allows thermodynamic variables to be stably integrated forward-in-time (solid arc-arrow) up to time level $n + 1$. 

---

For well-stratified problems, internal gravity waves may be the limiting process for determining a stable time-step. In the circumstance, it is more efficient to stagger in time of thermodynamic variables with the flow variables. Figure 2.6 illustrates the staggering and algorithm. The key difference between this and Figure 2.5 is that the thermodynamic variables are solved after the dynamics, using the recently updated flow field. This essentially allows the gravity wave terms to leap-frog in time giving second order accuracy and more stability.

The essential change in the staggered algorithm is that the thermodynamics solver is delayed from half a time step, allowing the use of the most recent velocities to compute the advection terms. Once the thermodynamics fields are updated, the hydrostatic pressure is computed to step forward the dynamics. Note that the pressure gradient must also be taken out of the Adams-Bashforth extrapolation. Also, retaining the integer time-levels, $n$ and $n + 1$, does not give a user the sense of where variables are located in time. Instead, we re-write the entire algorithm, (2.24) to (2.35),
annotating the position in time of variables appropriately:

\[
\phi^n_{\text{hyd}} = \int b(\theta^n, S^n) dr
\]  

(2.36)

\[
\mathbf{G}^{n-1/2} = \mathbf{G}_v(\bar{\nu}^{n-1/2})
\]  

(2.37)

\[
\mathbf{G}^{(n)}_v = (3/2 + \epsilon_{AB}) \mathbf{G}^{n-1/2}_v - (1/2 + \epsilon_{AB}) \mathbf{G}^{n-3/2}_v
\]  

(2.38)

\[
\bar{\nu}^{*} = \bar{\nu}^{n-1/2} + \Delta t \left( \mathbf{G}^{(n)}_v - \nabla \phi^n_{\text{hyd}} \right)
\]  

(2.39)

\[
\bar{\nu}^{**} = \mathbf{L}^{-1}_v(\bar{\nu}^{*})
\]  

(2.40)

\[
\eta^* = \epsilon_{fs} \left( \eta^{n-1/2} + \Delta t(\mathbf{P} - \mathbf{E})^n \right) - \Delta t \nabla \cdot H \bar{\nu}^{**}
\]  

(2.41)

\[
\nabla \cdot gH \nabla \eta^{n+1/2} - \frac{\epsilon_{fs} \eta^{n+1/2}}{\Delta t^2} = - \frac{\eta^*}{\Delta t^2}
\]  

(2.42)

\[
\bar{\nu}^{n+1/2} = \bar{\nu}^{**} - \Delta t g \nabla \eta^{n+1/2}
\]  

(2.43)

\[
G^n_{\theta,S} = G^n_{\theta,S}(u^{n+1/2}, \theta^n, S^n)
\]  

(2.44)

\[
G^{(n+1/2)}_{\theta,S} = (3/2 + \epsilon_{AB}) G^n_{\theta,S} - (1/2 + \epsilon_{AB}) G^{n-1}_{\theta,S}
\]  

(2.45)

\[
(\theta^*, S^*) = (\theta^n, S^n) + \Delta t G^{(n+1/2)}_{\theta,S}
\]  

(2.46)

\[
(\theta^{n+1}, S^{n+1}) = \mathbf{L}^{-1}_{\theta,S}(\theta^*, S^*)
\]  

(2.47)

The corresponding calling tree is given below. The staggered algorithm is activated with the run-time flag stagger-TimeStep = .TRUE. in parameter file data, namelist PARM01.

**Staggered Adams-Bashforth calling tree**

```
FORWARD_STEP
  EXTERNAL_FIELDS_LOAD
  DO_ATMOSPHERIC_PHYS
  DO_OCEANIC_PHYS
DYNAMICS
  CALC_PHI_HYD \[\phi^n_{\text{hyd}} (2.36)\]
  MOM_FLUXFORM or MOM_VECINV \[\mathbf{G}^{n-1/2}_v (2.37)\]
  TIMESTEP \[\bar{\nu}^* (2.38), (2.39)\]
  IMPLDIFF \[\bar{\nu}^{**} (2.40)\]
UPDATE_R_STAR or UPDATE_SURF_DR (NonLin-FS only)
SOLVE_FOR_PRESSURE
  CALC_DIV_GHAT \[\eta^* (2.41)\]
  CG2D
MOMENTUM_CORRECTION_STEP
  CALC_GRAD_PHI_SURF \[\nabla \eta^{n+1/2} (2.43)\]
  CORRECTION_STEP \[\eta^{n+1/2}, \nu^{n+1/2} (2.43)\]
THERMODYNAMICS
  CALC_GT
    GAD_CALC_RHS \[G^n_{\theta} = G_{\theta}(u, \theta^n) (2.44)\]
    EXTERNAL_FORCING \[G^n_{\theta} = G^n_{\theta} + Q\]
    ADAMS_BASHFORTH2 \[G^{(n+1/2)}_{\theta} (2.45)\]
```
The only difficulty with this approach is apparent in equation (2.44) and illustrated by the dotted arrow connecting $u, v^{n+1/2}$ with $G^\circ_\theta$. The flow used to advect tracers around is not naturally located in time. This could be avoided by applying the Adams-Bashforth extrapolation to the tracer field itself and advecting that around but this approach is not yet available. We’re not aware of any detrimental effect of this feature. The difficulty lies mainly in interpretation of what time-level variables and terms correspond to.

### 2.9 Non-hydrostatic formulation

The non-hydrostatic formulation re-introduces the full vertical momentum equation and requires the solution of a 3-D elliptic equations for non-hydrostatic pressure perturbation. We still integrate vertically for the hydrostatic pressure and solve a 2-D elliptic equation for the surface pressure/elevation for this reduces the amount of work needed to solve for the non-hydrostatic pressure.

The momentum equations are discretized in time as follows:

\[
\frac{1}{\Delta t} u^{n+1} + g \frac{\partial_x \eta^{n+1}}{\Delta t} + \partial_x \phi_{nh}^{n+1} = \frac{1}{\Delta t} u^n + G_u^{(n+1/2)}
\]

\[
\frac{1}{\Delta t} v^{n+1} + g \frac{\partial_y \eta^{n+1}}{\Delta t} + \partial_y \phi_{nh}^{n+1} = \frac{1}{\Delta t} v^n + G_v^{(n+1/2)}
\]

\[
\frac{1}{\Delta t} w^{n+1} + \partial_r \phi_{nh}^{n+1} = \frac{1}{\Delta t} w^n + G_w^{(n+1/2)}
\]

which must satisfy the discrete-in-time depth integrated continuity, equation (2.11) and the local continuity equation

\[
\partial_x u^{n+1} + \partial_y v^{n+1} + \partial_r w^{n+1} = 0
\]

As before, the explicit predictions for momentum are consolidated as:

\[
\begin{align*}
    u^* &= u^n + \Delta t G_u^{(n+1/2)} \\
    v^* &= v^n + \Delta t G_v^{(n+1/2)} \\
    w^* &= w^n + \Delta t G_w^{(n+1/2)}
\end{align*}
\]

but this time we introduce an intermediate step by splitting the tendency of the flow as follows:

\[
\begin{align*}
    u^{n+1} &= u^{**} - \Delta t \partial_x \phi_{nh}^{n+1} \\
    v^{n+1} &= v^{**} - \Delta t \partial_y \phi_{nh}^{n+1} \\
    w^{n+1} &= w^{**} - \Delta t \partial_r \phi_{nh}^{n+1}
\end{align*}
\]

Substituting into the depth integrated continuity (equation (2.11)) gives

\[
\partial_x H \frac{\partial_x (g \eta^{n+1} + \phi_{nh}^{n+1})}{\Delta t} + \partial_y H \frac{\partial_y (g \eta^{n+1} + \phi_{nh}^{n+1})}{\Delta t} - \frac{\epsilon f_x \eta^{n+1}}{\Delta t^2} = - \frac{\eta^*}{\Delta t^2}
\]

which is approximated by equation (2.15) on the basis that i) $\phi_{nh}^{n+1}$ is not yet known and ii) $\nabla \phi_{nh} << \nabla \eta$. If (2.15) is solved accurately then the implication is that $\phi_{nh} \approx 0$ so that the non-hydrostatic pressure field does not drive barotropic motion.

---

2.9. Non-hydrostatic formulation 49
The flow must satisfy non-divergence (equation (2.51)) locally, as well as depth integrated, and this constraint is used to form a 3-D elliptic equations for $\phi_{nh}^{n+1}$:

$$
\partial_{xx} \phi_{nh}^{n+1} + \partial_{yy} \phi_{nh}^{n+1} + \partial_{rr} \phi_{nh}^{n+1} = \partial_x u^* + \partial_y v^* + \partial_r w^*(2.53)
$$

The entire algorithm can be summarized as the sequential solution of the following equations:

$$
u^* = u^n + \Delta t G_w^{(n+1/2)} (2.54)
$$

$$v^* = v^n + \Delta t G_v^{(n+1/2)} (2.55)
$$

$$w^* = w^n + \Delta t G_w^{(n+1/2)} (2.56)
$$

$$\eta^* = \epsilon f_s (\eta^n + \Delta t(P - E)) - \Delta t \left( \partial_r H u^* + \partial_y H v^* \right) (2.57)
$$

$$\partial_r g H \partial_r \eta^{n+1} + \partial_y g H \partial_y \eta^{n+1} - \frac{\epsilon f_s \eta^{n+1}}{\Delta t^2} = - \frac{\eta^*}{\Delta t^2} (2.58)
$$

$$u^{**} = u^* - \Delta t g \partial_x \eta^{n+1} (2.59)
$$

$$v^{**} = v^* - \Delta t g \partial_y \eta^{n+1} (2.60)
$$

$$\partial_{xx} \phi_{nh}^{n+1} + \partial_{yy} \phi_{nh}^{n+1} + \partial_{rr} \phi_{nh}^{n+1} = \partial_x u^{**} + \partial_y v^{**} + \partial_r w^* (2.61)
$$

$$u^{n+1} = u^{**} - \Delta t \partial_x \phi_{nh}^{n+1} (2.62)
$$

$$v^{n+1} = v^{**} - \Delta t \partial_y \phi_{nh}^{n+1} (2.63)
$$

$$\partial_r w^{n+1} = - \partial_x u^{n+1} - \partial_y v^{n+1} (2.64)
$$

where the last equation is solved by vertically integrating for $w^{n+1}$.

### 2.10 Variants on the Free Surface

We now describe the various formulations of the free-surface that include non-linear forms, implicit in time using Crank-Nicholson, explicit and [oneday] split-explicit. First, we’ll reiterate the underlying algorithm but this time using the notation consistent with the more general vertical coordinate $r$. The elliptic equation for free-surface coordinate (units of $r$), corresponding to (2.11), and assuming no non-hydrostatic effects ($\epsilon_{nh} = 0$) is:

$$\epsilon f_s \eta^{n+1} - \nabla_h \cdot \Delta t^2 (R_o - R_{fixed}) \nabla_h b_s \eta^{n+1} = \eta^* (2.65)
$$

where

$$\eta^* = \epsilon f_s \eta^n - \Delta t \nabla_h \cdot \int_{R_{fixed}}^{R_o} \vec{v}^n \, dr + \epsilon f_w \Delta t(P - E)^n (2.66)
$$

### S/R SOLVE_FOR_PRESSURE

$$u^* : gU (\text{DYNVARS.h})$$

$$v^* : gV (\text{DYNVARS.h})$$

$$\eta^* : cg2d_b (\text{SOLVE_FOR_PRESSURE.h})$$

$$\eta^{n+1} : etaN (\text{DYNVARS.h})$$
Once \( \eta^{n+1} \) has been found, substituting into (2.2), (2.3) yields \( \vec{v}^{n+1} \) if the model is hydrostatic (\( \epsilon_{nh} = 0 \)):

\[
\vec{v}^{n+1} = \vec{v}^* - \Delta t \nabla h \cdot \vec{b}_s \eta^{n+1}
\]

This is known as the correction step. However, when the model is non-hydrostatic (\( \epsilon_{nh} = 1 \)) we need an additional step and an additional equation for \( \phi'_{nh} \). This is obtained by substituting (2.48), (2.49) and (2.50) into continuity:

\[
[\nabla^2_h + \partial_{rr}] \phi'_{nh}^{n+1} = \frac{1}{\Delta t} \nabla h \cdot \vec{v}^{**} + \partial_r r^*
\]

(2.67)

where

\[
\vec{v}^{**} = \vec{v}^* - \Delta t \nabla h \cdot \vec{b}_s \eta^{n+1}
\]

Note that \( \eta^{n+1} \) is also used to update the second RHS term \( \partial_r r^* \) since the vertical velocity at the surface (\( \dot{r}_{surf} \)) is evaluated as \( (\eta^{n+1} - \eta^n) / \Delta t \).

Finally, the horizontal velocities at the new time level are found by:

\[
\vec{v}^{n+1} = \vec{v}^{**} - \epsilon_{nh} \Delta t \nabla h \phi'_{nh}^{n+1}
\]

(2.68)

and the vertical velocity is found by integrating the continuity equation vertically. Note that, for the convenience of the restart procedure, the vertical integration of the continuity equation has been moved to the beginning of the time step (instead of at the end), without any consequence on the solution.

---

**S/R CORRECTION_STEP**

- \( \eta^{n+1} \): etaN (DYNVARS.h)
- \( \phi'_{nh}^{n+1} \): phi_nh (NH_VARS.h)
- \( u^* \): gU (DYNVARS.h)
- \( v^* \): gV (DYNVARS.h)
- \( u^{n+1} \): uVel (DYNVARS.h)
- \( v^{n+1} \): vVel (DYNVARS.h)

Regarding the implementation of the surface pressure solver, all computation are done within the routine SOLVE_FOR_PRESSURE and its dependent calls. The standard method to solve the 2D elliptic problem (2.65) uses the conjugate gradient method (routine CG2D); the solver matrix and conjugate gradient operator are only function of the discretized domain and are therefore evaluated separately, before the time iteration loop, within INI_CG2D. The computation of the RHS \( \eta^* \) is partly done in CALC_DIV_GHAT and in SOLVE_FOR_PRESSURE.

The same method is applied for the non-hydrostatic part, using a conjugate gradient 3D solver (CG3D) that is initialized in INI.CG3D. The RHS terms of 2D and 3D problems are computed together at the same point in the code.

### 2.10.1 Crank-Nicolson barotropic time stepping

The full implicit time stepping described previously is unconditionally stable but damps the fast gravity waves, resulting in a loss of potential energy. The modification presented now allows one to combine an implicit part \( (\gamma, \beta) \) and an explicit part \( (1 - \gamma, 1 - \beta) \) for the surface pressure gradient \( \gamma \) and for the barotropic flow divergence \( \beta \). For instance, \( \gamma = \beta = 1 \) is the previous fully implicit scheme; \( \gamma = \beta = 1/2 \) is the non damping (energy conserving), unconditionally stable, Crank-Nicolson scheme; \((\gamma, \beta) = (1, 0) \) or \((0, 1) \) corresponds to the forward - backward scheme that conserves energy but is only stable for small time steps. In the code, \( \gamma, \beta \) are defined as parameters, respectively implicSurfPress, implicDiv2DFlow. They are read from the main parameter file data (namelist PARM01) and are set by default to 1.1.

---

### 2.10. Variants on the Free Surface

---
Equations (2.12) – (2.17) are modified as follows:

\[
\frac{\nabla n+1}{\Delta t} + \nabla_n b_n [\eta n+1 + (1 - \gamma)\eta^n] + \epsilon_{nh} \nabla_n \phi_{nh}^{n+1} = \frac{\nabla^n}{\Delta t} + G_{\phi (n+1/2)} + \nabla_n \phi_{hyd}^{(n+1/2)}
\]

\[
\epsilon f_n \eta n+1 - \eta^n + \nabla_n \cdot \int_{R_{fixed}} [\beta \nabla n+1 + (1 - \beta)\nabla^n]dr = \epsilon_{fw}(P - E)
\]

We set

\[
\nabla^* = \nabla^n + \Delta t G_{\phi (n+1/2)} + (\gamma - 1)\Delta t \nabla_n b_n \eta^n + \Delta t \nabla_n \phi_{hyd}^{(n+1/2)}
\]

\[
\eta^* = \epsilon_{fw} \eta^n + \epsilon_{fw} \Delta t (P - E) - \Delta t \nabla_n \cdot \int_{R_{fixed}} [\beta \nabla^* + (1 - \beta)\nabla^n]dr
\]

In the hydrostatic case \(\epsilon_{nh} = 0\), allowing us to find \(\eta^{n+1}\), thus:

\[
\epsilon_{fw} \eta^{n+1} - \nabla_n \cdot \gamma \beta \Delta t b_n (R_a - R_{fixed}) \nabla_n \eta^{n+1} = \eta^*
\]

and then to compute (CORRECTION_STEP):

\[
\nabla^{n+1} = \nabla^* - \gamma \Delta t \nabla_n b_n \eta^{n+1}
\]

Notes:

1. The RHS term of equation (2.69) corresponds to the contribution of fresh water flux, \((\mathcal{P-E})\) to the free-surface variations (\(\epsilon_{fw} = 1\), \(\text{useRealFreshWaterFlux} = \text{TRUE}\). In parameter file data). In order to remain consistent with the tracer equation, specially in the non-linear free-surface formulation, this term is also affected by the Crank-Nicolson time stepping. The RHS reads: \(\epsilon_{fw}(\beta(P - E)^{n+1/2} + (1 - \beta)(P - E)^{n-1/2})\)

2. The stability criteria with Crank-Nicolson time stepping for the pure linear gravity wave problem in cartesian coordinates is:
   - \(\gamma + \beta < 1\): unstable
   - \(\gamma \geq 1/2\) and \(\beta \geq 1/2\): stable
   - \(\gamma + \beta \geq 1\): stable if \(c_{max}^2(\gamma - 1/2)(\beta - 1/2) + 1 \geq 0\) with \(c_{max} = 2\Delta t \sqrt{gH} \sqrt{1/\Delta x^2 + 1/\Delta y^2}\)

3. A similar mixed forward/backward time-stepping is also available for the non-hydrostatic algorithm, with a fraction \(\gamma_{nh}\) (0 < \(\gamma_{nh} \leq 1\)) of the non-hydrostatic pressure gradient being evaluated at time step \(n + 1\) (backward in time) and the remaining part (1 - \(\gamma_{nh}\)) being evaluated at time step \(n\) (forward in time). The run-time parameter \(\text{implicitNHPress}\) corresponding to the implicit fraction \(\gamma_{nh}\) of the non-hydrostatic pressure is set by default to the implicit fraction \(\gamma\) of surface pressure (\(\text{implicSurfPress}\)), but can also be specified independently (in main parameter file data, namelist \(\text{PARAM}\)).

### 2.10.2 Non-linear free-surface

Options have been added to the model that concern the free surface formulation.

#### 2.10.2.1 Pressure/geo-potential and free surface

For the atmosphere, since \(\phi = \phi_{topo} - \int_{p_o}^p \alpha_o dp\), subtracting the reference state defined in section Section 1.4.1.2:

\[
\phi_o = \phi_{topo} - \int_{p_o}^p \alpha_o dp \quad \text{with} \quad \phi_o(p_o) = \phi_{topo}
\]
we get:

$$\phi' = \phi - \phi_o = \int_p^{P_s} \alpha dp - \int_P^{P_o} \alpha_o dp$$

For the ocean, the reference state is simpler since \(\rho_o\) does not depend on \(z\) (\(b_o = g\)) and the surface reference position is uniformly \(z = 0\) (\(\theta_o = 0\)), and the same subtraction leads to a similar relation. For both fluids, using the isomorphic notations, we can write:

$$\phi' = \int_r^{r_{surf}} b \, dr - \int_r^{R_o} b_o \, dr$$

and re-write as:

$$\phi' = \int_r^{r_{surf}} b \, dr + \int_r^{R_o} (b - b_o) \, dr$$

(2.70)

or:

$$\phi' = \int_r^{r_{surf}} b_o \, dr + \int_r^{r_{surf}} (b - b_o) \, dr$$

(2.71)

In section Section 1.3.6, following eq. (2.70), the pressure/geo-potential \(\phi'\) has been separated into surface (\(\phi_s\)) and hydrostatic anomaly (\(\phi'_{hyd}\)). In this section, the split between \(\phi_s\) and \(\phi'_{hyd}\) is made according to equation (2.71). This slightly different definition reflects the actual implementation in the code and is valid for both linear and non-linear free-surface formulation, in both r-coordinate and r*-coordinate.

Because the linear free-surface approximation ignores the tracer content of the fluid parcel between \(R_o\) and \(r_{surf} = R_o + \eta\), for consistency reasons, this part is also neglected in \(\phi_{hyd}'\):

$$\phi_{hyd}' = \int_r^{r_{surf}} (b - b_o) \, dr \simeq \int_r^{R_o} (b - b_o) \, dr$$

Note that in this case, the two definitions of \(\phi_s\) and \(\phi_{hyd}'\) from equations (2.70) and (2.71) converge toward the same (approximated) expressions: \(\phi_s = \int_r^{r_{surf}} b_o \, dr\) and \(\phi_{hyd}' = \int_r^{R_o} b' \, dr\). On the contrary, the unapproximated formulation (see Section 2.10.2.2) retains the full expression: \(\phi_{hyd}' = \int_r^{r_{surf}} (b - b_o) \, dr\). This is obtained by selecting nonlinFreeSurf =4 in parameter file data. Regarding the surface potential:

$$\phi_s = \int_{R_o}^{R_o + \eta} b_o \, dr = b_s \eta \quad \text{with} \quad b_s = \frac{1}{\eta} \int_{R_o}^{R_o + \eta} b_o \, dr$$

\(b_s \simeq b_o(R_o)\) is an excellent approximation (better than the usual numerical truncation, since generally \(|\eta|\) is smaller than the vertical grid increment).

For the ocean, \(\phi_s = g \eta\) and \(b_s = g\) is uniform. For the atmosphere, however, because of topographic effects, the reference surface pressure \(R_o = p_o\) has large spatial variations that are responsible for significant \(b_s\) variations (from 0.8 to 1.2 \(m^3/kg\)). For this reason, when uniformLin_PhiSurf = FALSE (parameter file data, namelist PARAM01) a non-uniform linear coefficient \(b_s\) is used and computed (INI_LINEAR_PHISURF) according to the reference surface pressure \(p_o\): \(b_s = b_o(R_o) = c_p \eta (P_{SL}^{\rho}(\alpha - 1) \theta_{ref}(p_o))\), with \(P_{SL}\) the mean sea-level pressure.

### 2.10.2.2 Free surface effect on column total thickness (Non-linear free-surface)

The total thickness of the fluid column is \(r_{surf} - R_{fixed} = \eta + R_o - R_{fixed}\). In most applications, the free surface displacements are small compared to the total thickness \(\eta \ll H_o = R_o - R_{fixed}\). In the previous sections and in older version of the model, the linearized free-surface approximation was made, assuming \(r_{surf} - R_{fixed} \simeq H_o\) when computing horizontal transports, either in the continuity equation or in tracer and momentum advection terms. This approximation is dropped when using the non-linear free-surface formulation and the total thickness, including the time
varying part $\eta$, is considered when computing horizontal transports. Implications for the barotropic part are presented hereafter. In section Section 2.10.2.3 consequences for tracer conservation is briefly discussed (more details can be found in Campin et al. (2004) [CAHM04]) ; the general time-stepping is presented in section Section 2.10.2.4 with some limitations regarding the vertical resolution in section Section 2.10.2.5.

In the non-linear formulation, the continuous form of the model equations remains unchanged, except for the 2D continuity equation (2.11) which is now integrated from $R_{\text{fixed}}(x, y)$ up to $r_{\text{surf}} = R_0 + \eta$:

$$\epsilon_f s \frac{\partial \eta}{\partial t} = \frac{\partial}{\partial r} \left( \epsilon_f w (P - E) \right) = -\nabla h \cdot \int_{R_{\text{fixed}}}^{R_0 + \eta} \frac{\nabla}{\partial r} \left( \frac{\partial}{\partial e} \frac{\partial}{\partial h} b_s \right) dr + \epsilon_f w (P - E)$$

Since $\eta$ has a direct effect on the horizontal velocity (through $\nabla h \Phi_{\text{surf}}$), this adds a non-linear term to the free surface equation. Several options for the time discretization of this non-linear part can be considered, as detailed below.

If the column thickness is evaluated at time step $n$, and with implicit treatment of the surface potential gradient, equations (2.65) and (2.66) become:

$$\epsilon_f s \eta^{n+1} - \nabla h \cdot \Delta t^2 (\eta^n + R_0 - R_{\text{fixed}}) \nabla h b_s \eta^{n+1} = \eta^*$$

where

$$\eta^* = \epsilon_f s \eta^n - \Delta t \nabla h \cdot \int_{R_{\text{fixed}}}^{R_0 + \eta^n} \frac{\partial}{\partial e} \frac{\partial}{\partial h} b_s \left( \frac{\partial}{\partial e} \frac{\partial}{\partial h} \right) dr + \epsilon_f w \Delta t (P - E)^n$$

This method requires us to update the solver matrix at each time step.

Alternatively, the non-linear contribution can be evaluated fully explicitly:

$$\epsilon_f s \eta^{n+1} - \nabla h \cdot \Delta t^2 (R_0 - R_{\text{fixed}}) \nabla h b_s \eta^{n+1} = \eta^* + \nabla h \cdot \Delta t^2 (\eta^n) \nabla h b_s \eta^n$$

This formulation allows one to keep the initial solver matrix unchanged though throughout the integration, since the non-linear free surface only affects the RHS.

Finally, another option is a “linearized” formulation where the total column thickness appears only in the integral term of the RHS (2.66) but not directly in the equation (2.65).

Those different options (see Table 2.1) have been tested and show little differences. However, we recommend the use of the most precise method (nonlinFreeSurf =4) since the computation cost involved in the solver matrix update is negligible.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nonlinFreeSurf</td>
<td>-1</td>
<td>linear free-surface, restart from a pickup file produced with #undef EXACT_CONSERV code</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>linear free-surface (= default)</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>full non-linear free-surface</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>same as 4 but neglecting $\int_{R_0}^{R_0 + \eta} \nabla h b_s dr$ in $\Phi_{\text{surf}}$</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>same as 3 but do not update cg2d solver matrix</td>
</tr>
<tr>
<td>select_rStar</td>
<td>1</td>
<td>same as 2 but treat momentum as in linear free-surface</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>do not use $r^*$ vertical coordinate (= default)</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>use $r^*$ vertical coordinate</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>same as 2 but without the contribution of the slope of the coordinate in $\nabla \Phi$</td>
</tr>
</tbody>
</table>

Table 2.1: Non-linear free-surface flags
2.10.2.3 Tracer conservation with non-linear free-surface

To ensure global tracer conservation (i.e., the total amount) as well as local conservation, the change in the surface level thickness must be consistent with the way the continuity equation is integrated, both in the barotropic part (to find $\eta$) and baroclinic part (to find $w = \dot{r}$).

To illustrate this, consider the shallow water model, with a source of fresh water ($P$):

$$\partial_t h + \nabla \cdot (h \vec{v}) = P$$

where $h$ is the total thickness of the water column. To conserve the tracer $\theta$ we have to discretize:

$$\partial_t (h\theta) + \nabla \cdot (h\theta \vec{v}) = P\theta_{\text{rain}}$$

Using the implicit (non-linear) free surface described above (Section 2.4) we have:

$$h^{n+1} = h^n - \Delta t \nabla \cdot (h^n \vec{v}^{n+1}) + \Delta t P$$

The discretized form of the tracer equation must adopt the same “form” in the computation of tracer fluxes, that is, the same value of $h$, as used in the continuity equation:

$$h^{n+1} \theta^{n+1} = h^n \theta^n - \Delta t \nabla \cdot (h^n \theta^n \vec{v}^{n+1}) + \Delta t P\theta_{\text{rain}}$$

The use of a 3 time-levels time-stepping scheme such as the Adams-Bashforth make the conservation slightly tricky. The current implementation with the Adams-Bashforth time-stepping provides an exact local conservation and prevents any drift in the global tracer content (Campin et al. (2004) [CAHM04]). Compared to the linear free-surface method, an additional step is required: the variation of the water column thickness (from $h^n$ to $h^{n+1}$) is not incorporated directly into the tracer equation. Instead, the model uses the $G_\theta$ terms (first step) as in the linear free surface formulation (with the “surface correction” turned “on”, see tracer section):

$$G^n_\theta = \left(-\nabla \cdot (h^n \theta^n \vec{v}^{n+1}) - \vec{v}^{n+1}_{\text{surf}} \theta^n\right)/h^n$$

Then, in a second step, the thickness variation (expansion/reduction) is taken into account:

$$\theta^{n+1} = \theta^n + \Delta t \frac{h^n}{h^{n+1}} \left(G^{(n+1)/2}_\theta + P(\theta_{\text{rain}} - \theta^n)/h^n\right)$$

Note that with a simple forward time step (no Adams-Bashforth), these two formulations are equivalent, since $(h^{n+1} - h^n)/\Delta t = P - \nabla \cdot (h^n \vec{v}^{n+1}) = P + \vec{v}^{n+1}_{\text{surf}}$

2.10.2.4 Time stepping implementation of the non-linear free-surface

The grid cell thickness was hold constant with the linear free-surface; with the non-linear free-surface, it is now varying in time, at least at the surface level. This implies some modifications of the general algorithm described earlier in sections Section 2.7 and Section 2.8.

A simplifed version of the staggered in time, non-linear free-surface algorithm is detailed hereafter, and can be compared to the equivalent linear free-surface case (eq. (2.37) to (2.47)) and can also be easily transposed to the synchronous time-stepping case. Among the simplifications, salinity equation, implicit operator and detailed elliptic equation are omitted. Surface forcing is explicitly written as fluxes of temperature, fresh water and momentum, $Q^{n+1/2}, P^{n+1/2}, \mathcal{F}_n$ respectively. $h^n$ and $dh^n$ are the column and grid box thickness in r-coordinate.

$$\phi_{\text{hyd}} = \int b(\theta^n, S^n, r)dr$$ (2.72)
the vertically integrated continuity equation (2.77) has been added (exactConserv = .TRUE., in parameter file
the Two steps have been added to linear free-surface algorithm (eq. (2.37) to (2.47)): Firstly, the model "geometry" (here

\[ H \]

in the following cases:

Regarding the restart procedure, two 2D fields \( h^{n+1} \) and \( (h^n - h^{n-1})/\Delta t \) in addition to the standard state variables and tendencies \( (\eta^{n-1/2}, v^{n-1/2}, \theta^n, S^n, G^n_{v^{n-3/2}}, G^n_{\theta,S^{n-1}}) \) are stored in a "pickup" file. The model restarts reading this pickup file, then updates the model geometry according to \( h^{n-1} \), and compute \( h^n \) and the vertical velocity before starting the main calling sequence (eq. (2.72) to (2.79), FORWARD_STEP).

\[ S/R \text{ INTEGR}_\text{CONTINUITY} \]

\[ h^{n+1} - H_o : \text{etaH (DYNVARS.h)} \]
\[ h^n - H_o : \text{etaHnm1 (SURFACE.h)} \]
\[ (h^{n+1} - h^n)/\Delta t : \text{dEtaHdt (SURFACE.h)} \]
2.10.2.5 Non-linear free-surface and vertical resolution

When the amplitude of the free-surface variations becomes as large as the vertical resolution near the surface, the surface layer thickness can decrease to nearly zero or can even vanish completely. This latter possibility has not been implemented, and a minimum relative thickness is imposed (hFacInf, parameter file data, namelist PARM01) to prevent numerical instabilities caused by very thin surface level.

A better alternative to the vanishing level problem relies on a different vertical coordinate $r^*$: The time variation of the total column thickness becomes part of the $r^*$ coordinate motion, as in a $\sigma_z, \sigma_p$ model, but the fixed part related to topography is treated as in a height or pressure coordinate model. A complete description is given in Adcroft and Campin (2004) [AC04].

The time-stepping implementation of the $r^*$ coordinate is identical to the non-linear free-surface in $r$ coordinate, and differences appear only in the spatial discretization.

2.11 Spatial discretization of the dynamical equations

Spatial discretization is carried out using the finite volume method. This amounts to a grid-point method (namely second-order centered finite difference) in the fluid interior but allows boundaries to intersect a regular grid allowing a more accurate representation of the position of the boundary. We treat the horizontal and vertical directions as separable and differently.

2.11.1 The finite volume method: finite volumes versus finite difference

The finite volume method is used to discretize the equations in space. The expression “finite volume” actually has two meanings; one is the method of embedded or intersecting boundaries (shaved or lopped cells in our terminology) and the other is non-linear interpolation methods that can deal with non-smooth solutions such as shocks (i.e. flux limiters for advection). Both make use of the integral form of the conservation laws to which the weak solution is a solution on each finite volume of (sub-domain). The weak solution can be constructed out of piece-wise constant elements or be differentiable. The differentiable equations can not be satisfied by piece-wise constant functions.

As an example, the 1-D constant coefficient advection-diffusion equation:

$$\partial_t \theta + \partial_x (u \theta - \kappa \partial_x \theta) = 0$$

can be discretized by integrating over finite sub-domains, i.e. the lengths $\Delta x_i$:

$$\Delta x_i \partial_t \theta + \delta_i (F) = 0$$

is exact if $\theta(x)$ is piece-wise constant over the interval $\Delta x_i$ or more generally if $\theta_i$ is defined as the average over the interval $\Delta x_i$.

The flux, $F_{i-1/2}$, must be approximated:

$$F = u \tilde{\theta} - \frac{\kappa}{\Delta x_i} \partial_i \theta$$

and this is where truncation errors can enter the solution. The method for obtaining $\tilde{\theta}$ is unspecified and a wide range of possibilities exist including centered and upwind interpolation, polynomial fits based on the the volume average definitions of quantities and non-linear interpolation such as flux-limiters.

Choosing simple centered second-order interpolation and differencing recovers the same ODE’s resulting from finite differencing for the interior of a fluid. Differences arise at boundaries where a boundary is not positioned on a regular or smoothly varying grid. This method is used to represent the topography using lopped cell, see Adcroft et al. (1997) [AHM97]. Subtle difference also appear in more than one dimension away from boundaries. This happens because each direction is discretized independently in the finite difference method while the integrating over finite volume implicitly treats all directions simultaneously.
2.11.2 C grid staggering of variables

The basic algorithm employed for stepping forward the momentum equations is based on retaining non-divergence of the flow at all times. This is most naturally done if the components of flow are staggered in space in the form of an Arakawa C grid (Arakawa and Lamb, 1977 [AL77]).

Figure 2.7 shows the components of flow \((u,v,w)\) staggered in space such that the zonal component falls on the interface between continuity cells in the zonal direction. Similarly for the meridional and vertical directions. The continuity cell is synonymous with tracer cells (they are one and the same).

![Figure 2.7: Three dimensional staggering of velocity components. This facilitates the natural discretization of the continuity and tracer equations.](image)

2.11.3 Grid initialization and data

Initialization of grid data is controlled by subroutine INI_GRID which in calls INI_VERTICAL_GRID to initialize the vertical grid, and then either of INI_CARTESIAN_GRID, INI_SPHERICAL_POLAR_GRID or INI_CURVILINEAR_GRID to initialize the horizontal grid for cartesian, spherical-polar or curvilinear coordinates respectively.

The reciprocals of all grid quantities are pre-calculated and this is done in subroutine INI_MASKS_ETC which is called later by subroutine INITIALISE_FIXED.

All grid descriptors are global arrays and stored in common blocks in GRID.h and a generally declared as _RS.

2.11.4 Horizontal grid

The model domain is decomposed into tiles and within each tile a quasi-regular grid is used. A tile is the basic unit of domain decomposition for parallelization but may be used whether parallelized or not; see section [sec:domain_decomposition] for more details. Although the tiles may be patched together in an unstructured manner (i.e. irregular or non-tessilating pattern), the interior of tiles is a structured grid of quadrilateral cells. The horizontal coordinate system is orthogonal curvilinear meaning we can not necessarily treat the two horizontal directions as separable. Instead, each cell in the horizontal grid is described by the length of it’s sides and it’s area.

The grid information is quite general and describes any of the available coordinates systems, cartesian, spherical-polar or curvilinear. All that is necessary to distinguish between the coordinate systems is to initialize the grid data (descriptors) appropriately.
In the following, we refer to the orientation of quantities on the computational grid using geographic terminology such as points of the compass. This is purely for convenience but should not be confused with the actual geographic orientation of model quantities.

Figure 2.8: Staggering of horizontal grid descriptors (lengths and areas). The grid lines indicate the tracer cell boundaries and are the reference grid for all panels. a) The area of a tracer cell, $A_c$, is bordered by the lengths $\Delta x_g$ and $\Delta y_g$. b) The area of a vorticity cell, $A_c$, is bordered by the lengths $\Delta x_c$ and $\Delta y_c$. c) The area of a u cell, $A_u$, is bordered by the lengths $\Delta x_u$ and $\Delta y_u$. d) The area of a v cell, $A_v$, is bordered by the lengths $\Delta x_f$ and $\Delta y_u$.

Figure 2.8 (a) shows the tracer cell (synonymous with the continuity cell). The length of the southern edge, $\Delta x_g$, western edge, $\Delta y_g$ and surface area, $A_c$, presented in the vertical are stored in arrays $dxG$, $dyG$ and $rA$. The “g” suffix indicates that the lengths are along the defining grid boundaries. The “c” suffix associates the quantity with the cell centers. The quantities are staggered in space and the indexing is such that $dxG(i,j)$ is positioned to the south of $rA(i,j)$ and $dyG(i,j)$ positioned to the west.

Figure 2.8 (b) shows the vorticity cell. The length of the northern edge, $\Delta x_c$, eastern edge, $\Delta y_c$ and surface area, $A_c$, presented in the vertical are stored in arrays $dxC$, $dyC$ and $rAz$. The “c” suffix indicates that the lengths are measured between the cell centers and the “$\zeta$” suffix associates points with the vorticity points. The quantities are staggered in space and the indexing is such that $dxC(i,j)$ is positioned to the north of $rAz(i,j)$ and $dyC(i,j)$ positioned to the east.

Figure 2.8 (c) shows the “u” or western (w) cell. The length of the southern edge, $\Delta x_u$, eastern edge, $\Delta y_f$ and surface area, $A_u$, presented in the vertical are stored in arrays $dxV$, $dyF$ and $rAw$. The “v” suffix indicates that the length is measured between the v-points, the “f” suffix indicates that the length is measured between the (tracer) cell faces and the “w” suffix associates points with the u-points (w stands for west). The quantities are staggered in space and the indexing is such that $dxV(i,j)$ is positioned to the south of $rAw(i,j)$ and $dyF(i,j)$ positioned to the east.

Figure 2.8 (d) shows the “v” or southern (s) cell. The length of the northern edge, $\Delta x_f$, western edge, $\Delta y_u$ and surface area, $A_s$, presented in the vertical are stored in arrays $dxF$, $dyU$ and $rAs$. The “u” suffix indicates that the length is measured between the u-points, the “f” suffix indicates that the length is measured between the (tracer) cell faces and
the “s” suffix associates points with the v-points (s stands for south). The quantities are staggered in space and the indexing is such that \(dx_F(i,j)\) is positioned to the north of \(rA_s(i,j)\) and \(dy_U(i,j)\) positioned to the west.

S/R INI_CARTESIAN_GRID, INI_SPHERICAL_POLAR_GRID, INI_CURVILINEAR_GRID

\[ A_c, A_\zeta, A_w, A_s : rA, rAz, rAw, rAs \quad (GRID.h) \]
\[ \Delta x_g, \Delta y_g : dxG, dyG \quad (GRID.h) \]
\[ \Delta x_c, \Delta y_c : dxC, dyC \quad (GRID.h) \]
\[ \Delta x_f, \Delta y_f : dxF, dyF \quad (GRID.h) \]
\[ \Delta x_v, \Delta y_u : dxV, dyU \quad (GRID.h) \]

### 2.11.4.1 Reciprocals of horizontal grid descriptors

Lengths and areas appear in the denominator of expressions as much as in the numerator. For efficiency and portability, we pre-calculate the reciprocal of the horizontal grid quantities so that in-line divisions can be avoided.

For each grid descriptor (array) there is a reciprocal named using the prefix `recip_`. This doubles the amount of storage in `GRID.h` but they are all only 2-D descriptors.

S/R INI_MASKS_ETC

\[ A_c^{-1}, A_\zeta^{-1}, A_w^{-1}, A_s^{-1} : recip_rA, recip_rAz, recip_rAw, recip_rAs \quad (GRID.h) \]
\[ \Delta x_g^{-1}, \Delta y_g^{-1} : recip_dxG, recip_dyG \quad (GRID.h) \]
\[ \Delta x_c^{-1}, \Delta y_c^{-1} : recip_dxC, recip_dyC \quad (GRID.h) \]
\[ \Delta x_f^{-1}, \Delta y_f^{-1} : recip_dxF, recip_dyF \quad (GRID.h) \]
\[ \Delta x_v^{-1}, \Delta y_u^{-1} : recip_dxV, recip_dyU \quad (GRID.h) \]

### 2.11.4.2 Cartesian coordinates

Cartesian coordinates are selected when the logical flag `usingCartesianGrid` in namelist `PARM04` is set to true. The grid spacing can be set to uniform via scalars `dxSpacing` and `dySpacing` in namelist `PARM04` or to variable resolution by the vectors `DELX` and `DELY`. Units are normally meters. Non-dimensional coordinates can be used by interpreting the gravitational constant as the Rayleigh number.

### 2.11.4.3 Spherical-polar coordinates

Spherical coordinates are selected when the logical flag `usingSphericalPolarGrid` in namelist `PARM04` is set to true. The grid spacing can be set to uniform via scalars `dxSpacing` and `dySpacing` in namelist `PARM04` or to variable resolution by the vectors `DELX` and `DELY`. Units of these namelist variables are always degrees. The horizontal grid descriptors are calculated from these namelist variables have units of meters.
2.11.4.4 Curvilinear coordinates

Curvilinear coordinates are selected when the logical flag usingCurvilinearGrid in namelist PARM04 is set to true. The grid spacing cannot be set via the namelist. Instead, the grid descriptors are read from data files, one for each descriptor. As for other grids, the horizontal grid descriptors have units of meters.

2.11.5 Vertical grid

As for the horizontal grid, we use the suffixes “c” and “f” to indicate faces and centers. Figure 2.9 (a) shows the default vertical grid used by the model. $\Delta r_f$ is the difference in $r$ (vertical coordinate) between the faces (i.e. $\Delta r_f \equiv -\delta_k r$ where the minus sign appears due to the convention that the surface layer has index $k = 1$).

The vertical grid is calculated in subroutine INI_VERTICAL_GRID and specified via the vector delR in namelist PARM04. The units of “r” are either meters or Pascals depending on the isomorphism being used which in turn is dependent only on the choice of equation of state.

There are alternative namelist vectors delZ and delP which dictate whether z- or p- coordinates are to be used but we intend to phase this out since they are redundant.

The reciprocals $\Delta r_f^{-1}$ and $\Delta r_c^{-1}$ are pre-calculated (also in subroutine INI_VERTICAL_GRID). All vertical grid descriptors are stored in common blocks in GRID.h.

---

Figure 2.9: Two versions of the vertical grid. a) The cell centered approach where the interface depths are specified and the tracer points centered in between the interfaces. b) The interface centered approach where tracer levels are specified and the w-interfaces are centered in between.
The above grid Figure 2.9 (a) is known as the cell centered approach because the tracer points are at cell centers; the cell centers are mid-way between the cell interfaces. This discretization is selected when the thickness of the levels are provided (delR, parameter file data, namelist PARM04) An alternative, the vertex or interface centered approach, is shown in Figure 2.9 (b). Here, the interior interfaces are positioned mid-way between the tracer nodes (no longer cell centers). This approach is formally more accurate for evaluation of hydrostatic pressure and vertical advection but historically the cell centered approach has been used. An alternative form of subroutine INI_VERTICAL_GRID is used to select the interface centered approach This form requires to specify \( N_r + 1 \) vertical distances delRe (parameter file data, namelist PARM04, e.g. ideal_2D_oce/input/data) corresponding to surface to center, \( N_r - 1 \) center to center, and center to bottom distances.

```
S/R INI_VERTICAL_GRID
   \( \Delta r_f, \Delta r_c : \text{drF, drC (GRID.h)} \)
   \( \Delta r_f^{-1}, \Delta r_c^{-1} : \text{recip_drF, recip_drC (GRID.h)} \)
```

### 2.11.6 Topography: partially filled cells

Adcroft et al. (1997) [AHM97] presented two alternatives to the step-wise finite difference representation of topography. The method is known to the engineering community as intersecting boundary method. It involves allowing the boundary to intersect a grid of cells thereby modifying the shape of those cells intersected. We suggested allowing the topography to take on a piece-wise linear representation (shaved cells) or a simpler piecewise constant representation (partial step). Both show dramatic improvements in solution compared to the traditional full step representation, the piece-wise linear being the best. However, the storage requirements are excessive so the simpler piece-wise constant or partial-step method is all that is currently supported.

![Figure 2.10: A schematic of the x-r plane showing the location of the non-dimensional fractions \( h_c \) and \( h_w \). The physical thickness of a tracer cell is given by \( h_c(i, j, k) \Delta r_f(k) \) and the physical thickness of the open side is given by \( h_w(i, j, k) \Delta r_f(k) \).](image)

Figure 2.10 shows a schematic of the x-r plane indicating how the thickness of a level is determined at tracer and u points. The physical thickness of a tracer cell is given by \( h_c(i, j, k) \Delta r_f(k) \) and the physical thickness of the open side is given by \( h_w(i, j, k) \Delta r_f(k) \). Three 3-D descriptors \( h_c, h_w \) and \( h_s \) are used to describe the geometry: hFacC,
hFacW and hFacS respectively. These are calculated in subroutine INI_MASKS_ETC along with their reciprocals recip_hFacC, recip_hFacW and recip_hFacS.

The non-dimensional fractions (or h-facs as we call them) are calculated from the model depth array and then processed to avoid tiny volumes. The rule is that if a fraction is less than hFacMin then it is rounded to the nearer of 0 or hFacMin or if the physical thickness is less than hFacMinDr then it is similarly rounded. The larger of the two methods is used when there is a conflict. By setting hFacMinDr equal to or larger than the thinnest nominal layers, \( \min(\Delta z_f, h_{FacMinDr}) \), but setting hFacMin to some small fraction then the model will only lop thick layers but retain stability based on the thinnest unlopped thickness; \( \min(\Delta z_f, h_{FacMinDr}) \).

S/R :filelink:INI_MASKS_ETC

\( h_c, h_w, h_s : h_{FacC}, h_{FacW}, h_{FacS} \ (GRID.h) \)
\( h_c^{-1}, h_w^{-1}, h_s^{-1} : \text{recip}_h_{FacC}, \text{recip}_h_{FacW}, \text{recip}_h_{FacS} \ (GRID.h) \)

### 2.12 Continuity and horizontal pressure gradient term

The core algorithm is based on the “C grid” discretization of the continuity equation which can be summarized as:

\[
\frac{\partial_t u}{\Delta x_c} + \frac{1}{\Delta x_c} \delta_i \frac{\partial \Phi}{\partial r} \bigg|_{s} \eta + \frac{\epsilon_{nh}}{\Delta z} \delta_i \Phi_{n, h} = G_u - \frac{1}{\Delta x_c} \delta_i \Phi_{h} \quad (2.79)
\]

\[
\frac{\partial_t v}{\Delta y_c} + \frac{1}{\Delta y_c} \delta_j \frac{\partial \Phi}{\partial r} \bigg|_{s} \eta + \frac{\epsilon_{nh}}{\Delta y_c} \delta_j \Phi_{n, h} = G_v - \frac{1}{\Delta y_c} \delta_j \Phi_{h} \quad (2.80)
\]

\[
\epsilon_{nh} \left( \frac{\partial_t w}{\Delta r_f} + \frac{1}{\Delta r_f} \delta_k \Phi_{n, h} \right) = \epsilon_{nh} G_w + b^k - \frac{1}{\Delta r_c} \delta_k \Phi_{h} \quad (2.81)
\]

where the continuity equation has been most naturally discretized by staggering the three components of velocity as shown in Figure 2.7. The grid lengths \( \Delta x_c \) and \( \Delta y_c \) are the lengths between tracer points (cell centers). The grid lengths \( \Delta y_g \) and \( \Delta y_c \) are the grid lengths between cell corners. \( \Delta r_f \) and \( \Delta r_c \) are the distance (in units of \( r \)) between level interfaces (w-level) and level centers (tracer level). The surface area presented in the vertical is denoted \( A_c \). The factors \( h_w \) and \( h_s \) are non-dimensional fractions (between 0 and 1) that represent the fraction cell depth that is “open” for fluid flow.

The last equation, the discrete continuity equation, can be summed in the vertical to yield the free-surface equation:

\[
A_c \partial_t \eta + \delta_i \sum_k \Delta y_g \Delta r_f \Delta x_f h_w u + \delta_j \sum_k \Delta x_g \Delta r_f \Delta x_f h_s v + \delta_k A_c w = A_c (P - E)_{r=0} \quad (2.83)
\]

The source term \( P - E \) on the rhs of continuity accounts for the local addition of volume due to excess precipitation and run-off over evaporation and only enters the top-level of the ocean model.

### 2.13 Hydrostatic balance

The vertical momentum equation has the hydrostatic or quasi-hydrostatic balance on the right hand side. This discretization guarantees that the conversion of potential to kinetic energy as derived from the buoyancy equation exactly matches the form derived from the pressure gradient terms when forming the kinetic energy equation.

In the ocean, using \( z \)-coordinates, the hydrostatic balance terms are discretized:

\[
\epsilon_{nh} \partial_t w + g \rho^k + \frac{1}{\Delta z} \delta_k \Phi_{h} = \ldots \quad (2.84)
\]
In the atmosphere, using p-coordinates, hydrostatic balance is discretized:

\[ \bar{p}^k + \frac{1}{\Delta \Pi} \delta_k \Phi_h = 0 \]  

(2.85)

where \( \Delta \Pi \) is the difference in Exner function between the pressure points. The non-hydrostatic equations are not available in the atmosphere.

The difference in approach between ocean and atmosphere occurs because of the direct use of the ideal gas equation in forming the potential energy conversion term \( \alpha \omega \). Because of the different representation of hydrostatic balance between ocean and atmosphere there is no elegant way to represent both systems using an arbitrary coordinate.

The integration for hydrostatic pressure is made in the positive \( r \) direction (increasing \( k \)-index). For the ocean, this is from the free-surface down and for the atmosphere this is from the ground up.

The calculations are made in the subroutine \texttt{CALC\_PHI\_HYD}. Inside this routine, one of other of the atmospheric/oceanic form is selected based on the string variable \texttt{buoyancyRelation}.

2.14 Flux-form momentum equations

The original finite volume model was based on the Eulerian flux form momentum equations. This is the default though the vector invariant form is optionally available (and recommended in some cases).

The “G’s” (our colloquial name for all terms on rhs!) are broken into the various advective, Coriolis, horizontal dissipation, vertical dissipation and metric forces:

\[
G_u = G_u^{\text{advo}} + G_u^{\text{cor}} + G_u^{\text{h-diss}} + G_u^{\text{v-diss}} + G_u^{\text{metric}} + G_u^{\text{nh-metric}} \\
G_v = G_v^{\text{advo}} + G_v^{\text{cor}} + G_v^{\text{h-diss}} + G_v^{\text{v-diss}} + G_v^{\text{metric}} + G_v^{\text{nh-metric}} \\
G_w = G_w^{\text{advo}} + G_w^{\text{cor}} + G_w^{\text{h-diss}} + G_w^{\text{v-diss}} + G_w^{\text{metric}} + G_w^{\text{nh-metric}}
\]  

(2.86)  

(2.87)  

(2.88)

In the hydrostatic limit, \( G_w = 0 \) and \( \epsilon_{nh} = 0 \), reducing the vertical momentum to hydrostatic balance.

These terms are calculated in routines called from subroutine \texttt{MOM\_FLUXFORM} and collected into the global arrays \texttt{gU}, \texttt{gV}, and \texttt{gW}.

S/R MOM\_FLUXFORM

\[
G_u : gU ( \text{DYNVARS.h} ) \\
G_v : gV ( \text{DYNVARS.h} ) \\
G_w : gW ( \text{NH\_VARS.h} )
\]

2.14.1 Advection of momentum

The advective operator is second order accurate in space:

\[
A_u \Delta r_f h_w G_u^{\text{advo}} = \delta_i \bar{U}^i \bar{v}^i + \delta_j \bar{V}^j \bar{v}^i + \delta_k \bar{W}^i \bar{v}^k \\
A_v \Delta r_f h_s G_v^{\text{advo}} = \delta_i \bar{U}^i \bar{w}^i + \delta_j \bar{V}^j \bar{w}^i + \delta_k \bar{W}^i \bar{w}^k \\
A_w \Delta r_c G_w^{\text{advo}} = \delta_i \bar{U}^i \bar{w}^i + \delta_j \bar{V}^j \bar{w}^i + \delta_k \bar{W}^i \bar{w}^k
\]  

(2.89)  

(2.90)  

(2.91)
and because of the flux form does not contribute to the global budget of linear momentum. The quantities $U$, $V$ and $W$ are volume fluxes defined:

$$U = \Delta y_j \Delta r_f h_w u$$ (2.92)

$$V = \Delta x_j \Delta r_f h_x v$$ (2.93)

$$W = A_c w$$ (2.94)

The advection of momentum takes the same form as the advection of tracers but by a translated advective flow. Consequently, the conservation of second moments, derived for tracers later, applies to $u^2$ and $v^2$ and $w^2$ so that advection of momentum correctly conserves kinetic energy.

**S/R MOM_U_ADV_UU, MOM_U_ADV_VU, MOM_U_ADV_WU**

$uu, vu, wu : fZon, fMer, fVerUkp$ (local to MOM_FLUXFORM.F)

**S/R MOM_V_ADV_UV, MOM_V_ADV_VV, MOM_V_ADV_WV**

$uv, vv, vw : fZon, fMer, fVerVkp$ (local to MOM_FLUXFORM.F)

### 2.14.2 Coriolis terms

The “pure C grid” Coriolis terms (i.e. in absence of C-D scheme) are discretized:

$$A_w \Delta r_f h_w G_u^{Cor} = f A_c \Delta r_f h_c \bar{w}^i - \epsilon_n h f' A_c \Delta r_f h_c \bar{w}^k$$ (2.95)

$$A_s \Delta r_f h_s G_v^{Cor} = -f A_c \Delta r_f h_c \bar{w}^j$$ (2.96)

$$A_c \Delta r_c G_w^{Cor} = \epsilon_n h f' \epsilon_n h A_c \Delta r_f h_c \bar{w}^k$$ (2.97)

where the Coriolis parameters $f$ and $f'$ are defined:

$$f = 2\Omega \sin \varphi$$

$$f' = 2\Omega \cos \varphi$$

where $\varphi$ is geographic latitude when using spherical geometry, otherwise the $\beta$-plane definition is used:

$$f = f_o + \beta y$$

$$f' = 0$$

This discretization globally conserves kinetic energy. It should be noted that despite the use of this discretization in former publications, all calculations to date have used the following different discretization:

$$G_u^{Cor} = f_u \bar{w}^i - \epsilon_n h f'_u \bar{w}^k$$ (2.98)

$$G_v^{Cor} = -f_v \bar{w}^j$$ (2.99)

$$G_w^{Cor} = \epsilon_n h f'_w \bar{w}^k$$ (2.100)

where the subscripts on $f$ and $f'$ indicate evaluation of the Coriolis parameters at the appropriate points in space. The above discretization does not conserve anything, especially energy, but for historical reasons is the default for the code. A flag controls this discretization: set run-time integer `selectCoriScheme` to two (=2) (which otherwise defaults to zero) to select the energy-conserving conserving form (2.95), (2.96), and (2.97) above.

**S/R CD_CODE_SCHEME, MOM_U_CORIOLIS, MOM_V_CORIOLIS**
\[ G_{\text{Cor}}^u, G_{\text{Cor}}^v : cF \text{ (local to MOM_FLUXFORM.F)} \]

### 2.14.3 Curvature metric terms

The most commonly used coordinate system on the sphere is the geographic system \((\lambda, \varphi)\). The curvilinear nature of these coordinates on the sphere lead to some "metric" terms in the component momentum equations. Under the thin-atmosphere and hydrostatic approximations these terms are discretized:

\[
A_w \Delta r_f h_w G_{\text{metric}}^u = \frac{\pi^i}{a} \tan \varphi A_c \Delta r_f h_c \pi^j
\]

\[
A_s \Delta r_f h_s G_{\text{metric}}^v = -\frac{\pi^j}{a} \tan \varphi A_c \Delta r_f h_c \pi^i
\]

\[ G_{\text{metric}}^w = 0 \]

where \(a\) is the radius of the planet (sphericity is assumed) or the radial distance of the particle (i.e. a function of height). It is easy to see that this discretization satisfies all the properties of the discrete Coriolis terms since the metric factor \(\frac{\pi^i}{a} \tan \varphi\) can be viewed as a modification of the vertical Coriolis parameter: \(f \rightarrow f + \frac{\pi^i}{a} \tan \varphi\).

However, as for the Coriolis terms, a non-energy conserving form has exclusively been used to date:

\[ G_{\text{metric}}^u = \frac{u \pi^j}{a} \tan \varphi \]

\[ G_{\text{metric}}^v = \frac{v \pi^j}{a} \tan \varphi \]

where \(\tan \varphi\) is evaluated at the \(u\) and \(v\) points respectively.

S/R MOM_U_METRIC_SPHERE, MOM_V_METRIC_SPHERE

\[ G_{\text{metric}}^u, G_{\text{metric}}^v : mT \text{ (local to MOM_FLUXFORM.F)} \]

### 2.14.4 Non-hydrostatic metric terms

For the non-hydrostatic equations, dropping the thin-atmosphere approximation re-introduces metric terms involving \(w\) which are required to conserve angular momentum:

\[
A_w \Delta r_f h_w G_{\text{metric}}^w = -\frac{\pi^i \pi^k}{a} A_c \Delta r_f h_c
\]

\[
A_s \Delta r_f h_s G_{\text{metric}}^w = -\frac{\pi^j \pi^k}{a} A_c \Delta r_f h_c
\]

\[ G_{\text{metric}}^w = \frac{u^2 + v^2}{a} A_c \Delta r_f h_c \]

Because we are always consistent, even if consistently wrong, we have, in the past, used a different discretization in the model which is:

\[
G_{\text{metric}}^u = -\frac{u \pi^k}{a}
\]

\[
G_{\text{metric}}^v = -\frac{v \pi^k}{a}
\]

\[
G_{\text{metric}}^w = \frac{1}{a} \left( \frac{\pi^k}{a} \right)^2 + \frac{\pi^k}{a} \right)
\]

66 Chapter 2. Discretization and Algorithm
2.14.5 Lateral dissipation

Historically, we have represented the SGS Reynolds stresses as simply down gradient momentum fluxes, ignoring constraints on the stress tensor such as symmetry.

\[ A_u \Delta x f h_u G_u^{h-\text{diss}} = \delta_i \Delta y_f \Delta x_v \tau_{11} + \delta_j \Delta x_v \Delta y_f \tau_{12} \]

\[ A_v \Delta x f h_v G_v^{h-\text{diss}} = \delta_i \Delta y_u \Delta x_v \tau_{21} + \delta_j \Delta x_v \Delta y_u \tau_{22} \]

The lateral viscous stresses are discretized:

\[ \tau_{11} = A_h c_{11} \Delta (\varphi) \frac{1}{\Delta x_f} \delta_i u - A_4 c_{11} \Delta z (\varphi) \frac{1}{\Delta x_f} \delta_i \nabla^2 u \]

\[ \tau_{12} = A_h c_{12} \Delta (\varphi) \frac{1}{\Delta y_u} \delta_j u - A_4 c_{12} \Delta z (\varphi) \frac{1}{\Delta y_u} \delta_j \nabla^2 u \]

\[ \tau_{21} = A_h c_{21} \Delta (\varphi) \frac{1}{\Delta x_v} \delta_i v - A_4 c_{21} \Delta z (\varphi) \frac{1}{\Delta x_v} \delta_i \nabla^2 v \]

\[ \tau_{22} = A_h c_{22} \Delta (\varphi) \frac{1}{\Delta y_f} \delta_j v - A_4 c_{22} \Delta z (\varphi) \frac{1}{\Delta y_f} \delta_j \nabla^2 v \]

where the non-dimensional factors \( c_{lm} \Delta (\varphi), \{l, m, n\} \in \{1, 2\} \) define the “cosine” scaling with latitude which can be applied in various ad-hoc ways. For instance, \( c_{11} \Delta = c_{21} \Delta = (\cos \varphi)^{3/2} \), \( c_{12} \Delta = c_{22} \Delta = 1 \) would represent the anisotropic cosine scaling typically used on the “lat-lon” grid for Laplacian viscosity.

It should be noted that despite the ad-hoc nature of the scaling, some scaling must be done since on a lat-lon grid the converging meridians make it very unlikely that a stable viscosity parameter exists across the entire model domain.

The Laplacian viscosity coefficient, \( A_h \) (viscAh), has units of \( m^2 s^{-1} \). The bi-harmonic viscosity coefficient, \( A_4 \) (viscA4), has units of \( m^4 s^{-1} \).

Two types of lateral boundary condition exist for the lateral viscous terms, no-slip and free-slip.

The free-slip condition is most convenient to code since it is equivalent to zero-stress on boundaries. Simple masking of the stress components sets them to zero. The fractional open stress is properly handled using the lopped cells.

The no-slip condition defines the normal gradient of a tangential flow such that the flow is zero on the boundary. Rather than modify the stresses by using complicated functions of the masks and “ghost” points (see Adcroft and Marshall (1998) [AM98]) we add the boundary stresses as an additional source term in cells next to solid boundaries. This has
the advantage of being able to cope with “thin walls” and also makes the interior stress calculation (code) independent of the boundary conditions. The “body” force takes the form:

\[
G_{\text{side-drag}}^u = \frac{4}{\Delta z_f} (1 - h_\zeta) \frac{\Delta x_y}{\Delta y_u} \left( A_h c_{12\Delta}(\phi) u - A_4 c_{12\Delta z}(\phi) \nabla^2 u \right)
\]

(2.113)

\[
G_{\text{side-drag}}^v = \frac{4}{\Delta z_f} (1 - h_\zeta) \frac{\Delta y_u}{\Delta x_v} \left( A_h c_{21\Delta}(\phi) v - A_4 c_{21\Delta z}(\phi) \nabla^2 v \right)
\]

(2.114)

In fact, the above discretization is not quite complete because it assumes that the bathymetry at velocity points is deeper than at neighboring vorticity points, e.g. \(1 - h_w < 1 - h_\zeta\)

\[
\text{S/R} \text{ MOM_U_SIDEDRAG, MOM_V_SIDEDRAG}
\]

\[
G_{\text{side-drag}}^u, G_{\text{side-drag}}^v : vF \text{ ( local to MOM_FLUXFORM.F )}
\]

### 2.14.6 Vertical dissipation

Vertical viscosity terms are discretized with only partial adherence to the variable grid lengths introduced by the finite volume formulation. This reduces the formal accuracy of these terms to just first order but only next to boundaries; exactly where other terms appear such as linear and quadratic bottom drag.

\[
G_{\text{v-diss}}^u = \frac{1}{\Delta r_f h_w} \delta_k \tau_{13}
\]

(2.115)

\[
G_{\text{v-diss}}^v = \frac{1}{\Delta r_f h_s} \delta_k \tau_{23}
\]

(2.116)

\[
G_{\text{v-diss}}^w = \epsilon_{nh} \frac{1}{\Delta r_f h_d} \delta_k \tau_{33}
\]

(2.117)

represents the general discrete form of the vertical dissipation terms.

In the interior the vertical stresses are discretized:

\[
\tau_{13} = A_v \frac{1}{\Delta r_c} \delta_k u
\]

\[
\tau_{23} = A_v \frac{1}{\Delta r_c} \delta_k v
\]

\[
\tau_{33} = A_v \frac{1}{\Delta r_f} \delta_k w
\]

It should be noted that in the non-hydrostatic form, the stress tensor is even less consistent than for the hydrostatic (see Wajsowicz (1993) [Waj93]). It is well known how to do this properly (see Griffies and Hallberg (2000) [GH00]) and is on the list of to-do’s.

\[
\text{S/R} \text{ MOM_U_RVISCFLUX, MOM_V_RVISCFLUX}
\]

\[
\tau_{13} : fVrUp, fVrDw \text{ ( local to MOM_FLUXFORM.F )}
\]

\[
\tau_{23} : fVrUp, fVrDw \text{ ( local to MOM_FLUXFORM.F )}
\]

As for the lateral viscous terms, the free-slip condition is equivalent to simply setting the stress to zero on boundaries. The no-slip condition is implemented as an additional term acting on top of the interior and free-slip stresses. Bottom
drag represents additional friction, in addition to that imposed by the no-slip condition at the bottom. The drag is cast as a stress expressed as a linear or quadratic function of the mean flow in the layer above the topography:

\[ \tau_{bottom-drage} = \left( 2A_e \frac{1}{\Delta r_c} + r_b + C_d \sqrt{2KE^i} \right) u \]  

(2.118)

\[ \tau_{bottom-drage}^2 = \left( 2A_e \frac{1}{\Delta r_c} + r_b + C_d \sqrt{2KE^j} \right) v \]  

(2.119)

where these terms are only evaluated immediately above topography. \( r_b \) (bottomDragLinear) has units of \( m s^{-1} \) and a typical value of the order 0.0002 \( m s^{-1} \). \( C_d \) (bottomDragQuadratic) is dimensionless with typical values in the range 0.001–0.003.

S/R MOM_U_BOTTOMDRAG, MOM_V_BOTTOMDRAG

\[ \tau_{bottom-drage} / \Delta r_f, \tau_{bottom-drage}^2 / \Delta r_f : vF \ (\text{local to MOM_FLUXFORM.F}) \]

### 2.14.7 Derivation of discrete energy conservation

These discrete equations conserve kinetic plus potential energy using the following definitions:

\[ KE = \frac{1}{2} \left( u_i^2 + v_j^2 + \epsilon_{nh} w_k^2 \right) \]  

(2.120)

### 2.14.8 Mom Diagnostics

<table>
<thead>
<tr>
<th>&lt;-Name-&gt;</th>
<th>Levs</th>
<th>&lt;-parsing code-&gt;</th>
<th>&lt;- Units -&gt;</th>
<th>&lt;- Tile (max=80c)</th>
</tr>
</thead>
<tbody>
<tr>
<td>VISCAHZ</td>
<td>15</td>
<td>SZ MR</td>
<td>m^2/s</td>
<td>Harmonic Visc Coefficient (m2/s) (ZetaPt)</td>
</tr>
<tr>
<td>VISCA4Z</td>
<td>15</td>
<td>SZ MR</td>
<td>m^4/s</td>
<td>Biharmonic Visc Coefficient (m4/s) (ZetaPt)</td>
</tr>
<tr>
<td>VISCAHD</td>
<td>15</td>
<td>SM MR</td>
<td>m^2/s</td>
<td>Harmonic Viscosity Coefficient (m2/s)</td>
</tr>
<tr>
<td>VISCA4D</td>
<td>15</td>
<td>SM MR</td>
<td>m^4/s</td>
<td>Biharmonic Viscosity Coefficient (m4/s)</td>
</tr>
<tr>
<td>VAHZMAX</td>
<td>15</td>
<td>SZ MR</td>
<td>m^2/s</td>
<td>CFL-MAX Harm Visc Coefficient (m2/s)</td>
</tr>
<tr>
<td>VA4ZMAX</td>
<td>15</td>
<td>SZ MR</td>
<td>m^4/s</td>
<td>CFL-MAX Biharm Visc Coefficient (m4/s)</td>
</tr>
<tr>
<td>VAHDMAX</td>
<td>15</td>
<td>SM MR</td>
<td>m^2/s</td>
<td>CFL-MAX Harm Visc Coefficient (m2/s)</td>
</tr>
<tr>
<td>VA4DMAX</td>
<td>15</td>
<td>SM MR</td>
<td>m^4/s</td>
<td>CFL-MAX Biharm Visc Coefficient (m4/s)</td>
</tr>
<tr>
<td>VAHZMIN</td>
<td>15</td>
<td>SZ MR</td>
<td>m^2/s</td>
<td>RE-MIN Harm Visc Coefficient (m2/s)</td>
</tr>
<tr>
<td>VA4ZMIN</td>
<td>15</td>
<td>SZ MR</td>
<td>m^4/s</td>
<td>RE-MIN Biharm Visc Coefficient (m4/s)</td>
</tr>
<tr>
<td>VAHDMIN</td>
<td>15</td>
<td>SM MR</td>
<td>m^2/s</td>
<td>RE-MIN Harm Visc Coefficient (m2/s)</td>
</tr>
</tbody>
</table>

(continues on next page)
### Table of Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Category</th>
<th>Scale</th>
<th>Units</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VA4DMIN</td>
<td>(Div Pt)</td>
<td>SM MR</td>
<td>m^4/s</td>
<td>RE-MIN Bihamr Visc Coefficient (m4/s)</td>
</tr>
<tr>
<td>VA4ZLTH</td>
<td>(Zeta Pt)</td>
<td>SZ MR</td>
<td>m^2/s</td>
<td>Leith Harm Visc Coefficient (m2/s)</td>
</tr>
<tr>
<td>VA4DLTH</td>
<td>(Div Pt)</td>
<td>SM MR</td>
<td>m^4/s</td>
<td>Leith Biharm Visc Coefficient (m4/s)</td>
</tr>
<tr>
<td>VAHZLTHD</td>
<td>(Zeta Pt)</td>
<td>SZ MR</td>
<td>m^2/s</td>
<td>LeithD Harm Visc Coefficient (m2/s)</td>
</tr>
<tr>
<td>VA4ZLTHD</td>
<td>(Zeta Pt)</td>
<td>SM MR</td>
<td>m^4/s</td>
<td>LeithD Biharm Visc Coefficient (m4/s)</td>
</tr>
<tr>
<td>VAHZSMAG</td>
<td>(Zeta Pt)</td>
<td>SM MR</td>
<td>m^2/s</td>
<td>Smagorinsky Harm Visc Coefficient (m2/s)</td>
</tr>
<tr>
<td>VA4ZSMAG</td>
<td>(Zeta Pt)</td>
<td>SM MR</td>
<td>m^4/s</td>
<td>Smagorinsky Biharm Visc Coefficient (m4/s)</td>
</tr>
<tr>
<td>momKE</td>
<td></td>
<td>SM MR</td>
<td>m^2/s^2</td>
<td>Kinetic Energy (in momentum Eq.)</td>
</tr>
<tr>
<td>momHDiv</td>
<td></td>
<td>SM MR</td>
<td>s^-1</td>
<td>Horizontal Divergence (in momentum Eq.)</td>
</tr>
<tr>
<td>momVort3</td>
<td></td>
<td>SZ MR</td>
<td>s^-1</td>
<td>3rd component (vertical) of Vorticity</td>
</tr>
<tr>
<td>Strain</td>
<td></td>
<td>SZ MR</td>
<td>s^-1</td>
<td>Horizontal Strain of Horizontal Velocities</td>
</tr>
<tr>
<td>Tension</td>
<td></td>
<td>SM MR</td>
<td>s^-1</td>
<td>Horizontal Tension of Horizontal Velocities</td>
</tr>
<tr>
<td>UBotDrag</td>
<td></td>
<td>UU 129MR</td>
<td>m/s^2</td>
<td>U momentum tendency from Bottom Drag</td>
</tr>
<tr>
<td>VBotDrag</td>
<td></td>
<td>VV 128MR</td>
<td>m/s^2</td>
<td>V momentum tendency from Bottom Drag</td>
</tr>
<tr>
<td>USidDrag</td>
<td></td>
<td>UU 131MR</td>
<td>m/s^2</td>
<td>U momentum tendency from Side Drag</td>
</tr>
<tr>
<td>VSidDrag</td>
<td></td>
<td>VV 130MR</td>
<td>m/s^2</td>
<td>V momentum tendency from Side Drag</td>
</tr>
<tr>
<td>Um_Diss</td>
<td></td>
<td>UU 133MR</td>
<td>m/s^2</td>
<td>U momentum tendency from Dissipation</td>
</tr>
<tr>
<td>Vm_Diss</td>
<td></td>
<td>VV 132MR</td>
<td>m/s^2</td>
<td>V momentum tendency from Dissipation</td>
</tr>
<tr>
<td>Um_Advec</td>
<td></td>
<td>UU 135MR</td>
<td>m/s^2</td>
<td>U momentum tendency from Advection terms</td>
</tr>
<tr>
<td>Vm_Advec</td>
<td></td>
<td>VV 134MR</td>
<td>m/s^2</td>
<td>V momentum tendency from Advection terms</td>
</tr>
<tr>
<td>Um_Cori</td>
<td></td>
<td>UU 137MR</td>
<td>m/s^2</td>
<td>U momentum tendency from Coriolis term</td>
</tr>
<tr>
<td>Vm_Cori</td>
<td></td>
<td>VV 136MR</td>
<td>m/s^2</td>
<td>V momentum tendency from Coriolis term</td>
</tr>
<tr>
<td>Um_Ext</td>
<td></td>
<td>UU 137MR</td>
<td>m/s^2</td>
<td>U momentum tendency from external forcing</td>
</tr>
<tr>
<td>Vm_Ext</td>
<td></td>
<td>VV 138MR</td>
<td>m/s^2</td>
<td>V momentum tendency from external forcing</td>
</tr>
<tr>
<td>Um_AdvZ3</td>
<td></td>
<td>UU 141MR</td>
<td>m/s^2</td>
<td>U momentum tendency from Vorticity-Advection</td>
</tr>
<tr>
<td>Vm_AdvZ3</td>
<td></td>
<td>VV 140MR</td>
<td>m/s^2</td>
<td>V momentum tendency from Vorticity-Advection</td>
</tr>
<tr>
<td>Um_AdvRes</td>
<td></td>
<td>UU 143MR</td>
<td>m/s^2</td>
<td>U momentum tendency from vertical-Advection (Explicit part)</td>
</tr>
<tr>
<td>Vm_AdvRes</td>
<td></td>
<td>VV 142MR</td>
<td>m/s^2</td>
<td>V momentum tendency from vertical-Advection (Explicit part)</td>
</tr>
</tbody>
</table>
### 2.15 Vector invariant momentum equations

The finite volume method lends itself to describing the continuity and tracer equations in curvilinear coordinate systems. However, in curvilinear coordinates many new metric terms appear in the momentum equations (written in Lagrangian or flux-form) making generalization far from elegant. Fortunately, an alternative form of the equations, the vector invariant equations are exactly that; invariant under coordinate transformations so that they can be applied uniformly in any orthogonal curvilinear coordinate system such as spherical coordinates, boundary following or the conformal spherical cube system.

The non-hydrostatic vector invariant equations read:

\[
\partial_t \mathbf{v} + (2\mathbf{\Omega} + \mathbf{\zeta}) \wedge \mathbf{v} - b \mathbf{r} + \nabla B = \nabla \cdot \tau \quad (2.121)
\]

which describe motions in any orthogonal curvilinear coordinate system. Here, \( B \) is the Bernoulli function and \( \mathbf{\zeta} = \nabla \wedge \mathbf{v} \) is the vorticity vector. We can take advantage of the elegance of these equations when discretizing them and use the discrete definitions of the grad, curl and divergence operators to satisfy constraints. We can also consider the analogy to forming derived equations, such as the vorticity equation, and examine how the discretization can be adjusted to give suitable vorticity advection among other things.

The underlying algorithm is the same as for the flux form equations. All that has changed is the contents of the “G’s”. For the time-being, only the hydrostatic terms have been coded but we will indicate the points where non-hydrostatic contributions will enter:

\[
G_u = G^f_u + G^C_{uv} + G^C_{uw} + G^{\partial_y B} + G^{\partial_z \tau^y} + G^{h - \text{dissip}} + G^{w - \text{dissip}} \\
G_v = G^f_v + G^C_{uv} + G^C_{vw} + G^{\partial_x B} + G^{\partial_z \tau^x} + G^{h - \text{dissip}} + G^{w - \text{dissip}} \\
G_w = G^f_w + G^C_{vw} + G^C_{wu} + G^{\partial_x B} + G^{h - \text{dissip}} + G^{w - \text{dissip}}
\]

---

**S/R MOM_VECINV**

2.15. **Vector invariant momentum equations**
2.15.1 Relative vorticity

The vertical component of relative vorticity is explicitly calculated and used in the discretization. The particular form is crucial for numerical stability; alternative definitions break the conservation properties of the discrete equations. Relative vorticity is defined:

\[ \zeta_3 = \frac{\Gamma}{A_\zeta} = \frac{1}{A_\zeta} (\delta_i \Delta y_c v - \delta_j \Delta x_c u) \]  

(2.125)

where \( A_\zeta \) is the area of the vorticity cell presented in the vertical and \( \Gamma \) is the circulation about that cell.

S/R MOM_CALC_RELVORT3

\( \zeta_3 : \text{vort3 (local to MOM_VECINV.F)} \)

2.15.2 Kinetic energy

The kinetic energy, denoted \( KE \), is defined:

\[ KE = \frac{1}{2} (u^2_i + v^2_j + \epsilon_{nh} w^2_k) \]  

(2.126)

S/R MOM_CALC_KE

\( KE : \text{KE (local to MOM_VECINV.F)} \)

2.15.3 Coriolis terms

The potential enstrophy conserving form of the linear Coriolis terms are written:

\[ G_{fu}^f = \frac{1}{\Delta x_c} \frac{f^i}{h_\zeta} \frac{\Delta x_c h_w u_j}{\Delta y_c v^i} \]  

(2.127)

\[ G_{fu}^f = -\frac{1}{\Delta y_c} \frac{f^j}{h_\zeta} \frac{\Delta y_c h_w u_j}{\Delta y_c v^i} \]  

(2.128)

Here, the Coriolis parameter \( f \) is defined at vorticity (corner) points.

The potential enstrophy conserving form of the non-linear Coriolis terms are written:

\[ G_{fu}^{\zeta_u} = \frac{1}{\Delta x_c} \frac{\zeta_3^i}{h_\zeta} \frac{\Delta x_c h_w u_j}{\Delta y_c v^i} \]  

(2.129)

\[ G_{fu}^{\zeta_u} = -\frac{1}{\Delta y_c} \frac{\zeta_3^j}{h_\zeta} \frac{\Delta y_c h_w u_j}{\Delta y_c v^i} \]  

(2.130)
The Coriolis terms can also be evaluated together and expressed in terms of absolute vorticity \( f + \zeta_3 \). The potential enstrophy conserving form using the absolute vorticity is written:

\[
G_f^u v + G_{\zeta 3}^u v = \frac{1}{\Delta x_c} \frac{f + \zeta_3}{h_\zeta} \overline{x_g h_x v^j}^j
\]  
(2.131)

\[
G_f^v u + G_{\zeta 3}^v u = -\frac{1}{\Delta y_c} \frac{f + \zeta_3}{h_\zeta} \overline{y_g h_y u^j}^j
\]  
(2.132)

The distinction between using absolute vorticity or relative vorticity is useful when constructing higher order advection schemes; monotone advection of relative vorticity behaves differently to monotone advection of absolute vorticity. Currently the choice of relative/absolute vorticity, centered/upwind/high order advection is available only through commented subroutine calls.

S/R MOM_VI_CORIOLIS, MOM_VI_U_CORIOLIS, MOM_VI_V_CORIOLIS

\( G_f^u, G_{\zeta 3}^u : uCf \) (local to MOM_VECINV.F)

\( G_f^v, G_{\zeta 3}^v : vCf \) (local to MOM_VECINV.F)

### 2.15.4 Shear terms

The shear terms \((\zeta_2 w \text{ and } \zeta_1 w)\) are are discretized to guarantee that no spurious generation of kinetic energy is possible; the horizontal gradient of Bernoulli function has to be consistent with the vertical advection of shear:

\[
G_{\zeta 2}^w u = \frac{1}{\mathcal{A}_w \Delta r_f h_w} \mathcal{A}_w^j (\delta_k u - \epsilon_{n1} \delta_i w)^k
\]  
(2.133)

\[
G_{\zeta 1}^w v = \frac{1}{\mathcal{A}_s \Delta r_f h_s} \mathcal{A}_s^j (\delta_k v - \epsilon_{n1} \delta_j w)^k
\]  
(2.134)

S/R MOM_VI_U_VERTSHEAR, MOM_VI_V_VERTSHEAR

\( G_{\zeta 2}^w : uCf \) (local to MOM_VECINV.F)

\( G_{\zeta 1}^w : vCf \) (local to MOM_VECINV.F)

### 2.15.5 Gradient of Bernoulli function

\[
G_{\phi'}^{\partial_x B} = \frac{1}{\Delta x_c} \delta_i (\phi' + KE)
\]  
(2.135)

\[
G_{\phi'}^{\partial_y B} = \frac{1}{\Delta x_y} \delta_j (\phi' + KE)
\]  
(2.136)

S/R MOM_VI_U_GRAD_KE, MOM_VI_V_GRAD_KE

\( G_{\phi'}^{\partial_x KE} : uCf \) (local to MOM_VECINV.F)

\( G_{\phi'}^{\partial_y KE} : vCf \) (local to MOM_VECINV.F)
2.15.6 Horizontal divergence

The horizontal divergence, a complimentary quantity to relative vorticity, is used in parameterizing the Reynolds
stresses and is discretized:

\[
D = \frac{1}{A_c h_c} \left( \delta_i \Delta y_g h_w u + \delta_j \Delta x_g h_s v \right)
\]

(2.137)

S/R MOM_CALC_KE

\[
D : h\text{Div} \ (\text{local to MOM_VECINV.F})
\]

2.15.7 Horizontal dissipation

The following discretization of horizontal dissipation conserves potential vorticity (thickness weighted relative vortic-
ity) and divergence and dissipates energy, enstrophy and divergence squared:

\[
G^h_{\text{dissip}} = \frac{1}{\Delta x_c} \delta_i \left( A_D D - A_{D4} D^* \right) - \frac{1}{\Delta y_c} \delta_j h_\zeta \left( A_\zeta \zeta - A_{\zeta 4} \zeta^* \right)
\]

\[
G^h_{\text{dissip}} = \frac{1}{\Delta x_c h_s} \delta_i h_\zeta \left( A_\zeta \zeta - A_{\zeta 4} \zeta^* \right) + \frac{1}{\Delta y_c} \delta_j \left( A_D D - A_{D4} D^* \right)
\]

(2.138)

where

\[
D^* = \frac{1}{A_c h_c} \left( \delta_i \Delta y_g h_w \nabla^2 u + \delta_j \Delta x_g h_s \nabla^2 v \right)
\]

\[
\zeta^* = \frac{1}{A_\zeta} \left( \delta_i \Delta y_c \nabla^2 v - \delta_j \Delta x_c \nabla^2 u \right)
\]

S/R MOM_VI_HDISSIP

\[
G^h_{\text{dissip}} : u\text{Dissip} \ (\text{local to MOM_VI_HDISSIP.F})
\]

\[
G^h_{\text{dissip}} : v\text{Dissip} \ (\text{local to MOM_VI_HDISSIP.F})
\]

2.15.8 Vertical dissipation

Currently, this is exactly the same code as the flux form equations.

\[
G^v_{\text{diss}} = \frac{1}{\Delta r_f h_w} \delta_k \tau_{13}
\]

(2.140)

\[
G^v_{\text{diss}} = \frac{1}{\Delta r_f h_s} \delta_k \tau_{23}
\]

(2.141)

represents the general discrete form of the vertical dissipation terms.

In the interior the vertical stresses are discretized:

\[
\tau_{13} = A_v \frac{1}{\Delta r_c} \delta_k u
\]

\[
\tau_{23} = A_v \frac{1}{\Delta r_c} \delta_k v
\]

S/R MOM_U_RVISCFLUX, MOM_V_RVISCFLUX

\[
\tau_{13}, \tau_{23} : vrf \ (\text{local to MOM_VECINV.F})
\]
2.16 Tracer equations

The basic discretization used for the tracer equations is the second order piece-wise constant finite volume form of the forced advection-diffusion equations. There are many alternatives to second order method for advection and alternative parameterizations for the sub-grid scale processes. The Gent-McWilliams eddy parameterization, KPP mixing scheme and PV flux parameterization are all dealt with in separate sections. The basic discretization of the advection-diffusion part of the tracer equations and the various advection schemes will be described here.

2.16.1 Time-stepping of tracers: ABII

The default advection scheme is the centered second order method which requires a second order or quasi-second order time-stepping scheme to be stable. Historically this has been the quasi-second order Adams-Bashforth method (ABII) and applied to all terms. For an arbitrary tracer, $\tau$, the forced advection-diffusion equation reads:

$$ \partial_t \tau + G_{adv}^\tau = G_{diff}^\tau + G_{forc}^\tau $$

(2.142)

where $G_{adv}^\tau$, $G_{diff}^\tau$ and $G_{forc}^\tau$ are the tendencies due to advection, diffusion and forcing, respectively, namely:

$$ G_{adv}^\tau = \partial_x u \tau + \partial_y v \tau + \partial_r w \tau - \tau \nabla \cdot \mathbf{v} $$

(2.143)

$$ G_{diff}^\tau = \nabla \cdot \mathbf{K} \nabla \tau $$

(2.144)

and the forcing can be some arbitrary function of state, time and space.

The term, $\tau \nabla \cdot \mathbf{v}$, is required to retain local conservation in conjunction with the linear implicit free-surface. It only affects the surface layer since the flow is non-divergent everywhere else. This term is therefore referred to as the surface correction term. Global conservation is not possible using the flux-form (as here) and a linearized free-surface (Griffies and Hallberg (2000) [GH00], Campin et al. (2004) [CAHM04]).

The continuity equation can be recovered by setting $G_{diff} = G_{forc} = 0$ and $\tau = 1$.

The driver routine that calls the routines to calculate tendencies are CALC_GT and CALC_GS for temperature and salt (moisture), respectively. These in turn call a generic advection diffusion routine GAD_CALC_RHS that is called with the flow field and relevant tracer as arguments and returns the collective tendency due to advection and diffusion. Forcing is add subsequently in CALC_GT or CALC_GS to the same tendency array.

S/R GAD_CALC_RHS

$\tau : \text{tau (argument)}$

$G^{(n)} : \text{gTracer (argument)}$

$F_r : \text{fVerT (argument)}$

The space and time discretization are treated separately (method of lines). Tendencies are calculated at time levels $n$ and $n - 1$ and extrapolated to $n + 1/2$ using the Adams-Bashforth method:

$$ G^{(n+1/2)} = \left( \frac{3}{2} + \epsilon \right) G^{(n)} - \left( \frac{1}{2} + \epsilon \right) G^{(n-1)} $$

(2.145)

where $G^{(n)} = G_{adv}^\tau + G_{diff}^\tau + G_{src}^\tau$ at time step $n$. The tendency at $n - 1$ is not re-calculated but rather the tendency at $n$ is stored in a global array for later re-use.

S/R ADAMS_BASHFORTH2

$G^{(n+1/2)} : \text{gTracer (argument on exit)}$
\( G^{(n)} : \text{gTracer ( argument on entry) } \)
\( G^{(n-1)} : \text{gTrNm1 ( argument) } \)
\( \epsilon : \text{ABeps ( PARAMS.h) } \)

The tracers are stepped forward in time using the extrapolated tendency:

\[
\tau^{(n+1)} = \tau^{(n)} + \Delta t G^{(n+1/2)} \tag{2.146}
\]

**S/R TIMESTEP_TRACER**

\( \tau^{(n+1)} : \text{gTracer ( argument on exit) } \)
\( \tau^{(n)} : \text{tracer ( argument on entry) } \)
\( G^{(n+1/2)} : \text{gTracer ( argument) } \)
\( \Delta t : \text{deltaTtracer ( PARAMS.h) } \)

Strictly speaking the ABII scheme should be applied only to the advection terms. However, this scheme is only used in conjunction with the standard second, third and fourth order advection schemes. Selection of any other advection scheme disables Adams-Bashforth for tracers so that explicit diffusion and forcing use the forward method.

### 2.17 Advection schemes

#### 2.17.1 Linear advection schemes

The advection schemes known as centered second order, centered fourth order, first order upwind and upwind biased third order are known as linear advection schemes because the coefficient for interpolation of the advected tracer are linear and a function only of the flow, not the tracer field itself. We discuss these first since they are most commonly used in the field and most familiar.

#### 2.17.1.1 Centered second order advection-diffusion

The basic discretization, centered second order, is the default. It is designed to be consistent with the continuity equation to facilitate conservation properties analogous to the continuum. However, centered second order advection is notoriously noisy and must be used in conjunction with some finite amount of diffusion to produce a sensible solution. The advection operator is discretized:

\[
A_c \Delta r f h_c G_{adv}^r = \delta_i F_x + \delta_j F_y + \delta_k F_r \tag{2.147}
\]

where the area integrated fluxes are given by:

\[
F_x = U \tau^i \\
F_y = V \tau^j \\
F_r = W \tau^k
\]

The quantities \( U, V \) and \( W \) are volume fluxes defined as:

\[
U = \Delta y_g \Delta r f h_w u \\
V = \Delta x_g \Delta r f h_x v \\
W = A_c w
\]
For non-divergent flow, this discretization can be shown to conserve the tracer both locally and globally and to globally conserve tracer variance, $\tau^2$. The proof is given in Adcroft (1995) [Adc95] and Adcroft et al. (1997) [AHM97].

\[ S/R \text{ GAD}_2\text{ ADV}_X \]
\[
F_x : uT \text{ ( argument )} \\
U : uTrans \text{ ( argument )} \\
\tau : \text{ tracer ( argument )}
\]

\[ S/R \text{ GAD}_2\text{ ADV}_Y \]
\[
F_y : vT \text{ ( argument )} \\
V : vTrans \text{ ( argument )} \\
\tau : \text{ tracer ( argument )}
\]

\[ S/R \text{ GAD}_2\text{ ADV}_R \]
\[
F_r : wT \text{ ( argument )} \\
W : rTrans \text{ ( argument )} \\
\tau : \text{ tracer ( argument )}
\]

### 2.17.1.2 Third order upwind bias advection

Upwind biased third order advection offers a relatively good compromise between accuracy and smoothness. It is not a “positive” scheme meaning false extrema are permitted but the amplitude of such are significantly reduced over the centered second order method.

The third order upwind fluxes are discretized:

\[
F_x = U\tau - \frac{1}{6}\delta_{ii}\tau + \frac{1}{2}|U|\delta_{i}\frac{1}{6}\delta_{ii}\tau \\
F_y = V\tau - \frac{1}{6}\delta_{jj}\tau + \frac{1}{2}|V|\delta_{j}\frac{1}{6}\delta_{jj}\tau \\
F_r = W\tau - \frac{1}{6}\delta_{kk}\tau + \frac{1}{2}|W|\delta_{k}\frac{1}{6}\delta_{kk}\tau
\]

At boundaries, $\delta_{nn}$ is set to zero allowing $\delta_{nn}$ to be evaluated. We are currently examine the accuracy of this boundary condition and the effect on the solution.

\[ S/R \text{ GAD}_3\text{ ADV}_X \]
\[
F_x : uT \text{ ( argument )} \\
U : uTrans \text{ ( argument )} \\
\tau : \text{ tracer ( argument )}
\]

\[ S/R \text{ GAD}_3\text{ ADV}_Y \]
\[
F_y : vT \text{ ( argument )}
\]
2.17.1.3 Centered fourth order advection

Centered fourth order advection is formally the most accurate scheme we have implemented and can be used to great effect in high resolution simulations where dynamical scales are well resolved. However, the scheme is noisy, like the centered second order method, and so must be used with some finite amount of diffusion. Bi-harmonic is recommended since it is more scale selective and less likely to diffuse away the well resolved gradient the fourth order scheme worked so hard to create.

The centered fourth order fluxes are discretized:

\[
F_x = U \tau - \frac{1}{6} \delta_{ij} \tau
\]
\[
F_y = V \tau - \frac{1}{6} \delta_{ij} \tau
\]
\[
F_r = W \tau - \frac{1}{6} \delta_{ij} \tau
\]

As for the third order scheme, the best discretization near boundaries is under investigation but currently \( \delta_i \tau = 0 \) on a boundary.
2.17.1.4 First order upwind advection

Although the upwind scheme is the underlying scheme for the robust or non-linear methods given in Section 2.17.2, we haven’t actually implemented this method for general use. It would be very diffusive and it is unlikely that it could ever produce more useful results than the positive higher order schemes.

Upwind bias is introduced into many schemes using the abs function and it allows the first order upwind flux to be written:

\begin{align*}
F_x &= U \tau_i - \frac{1}{2} |U| \delta_i \tau \\
F_y &= V \tau_j - \frac{1}{2} |V| \delta_j \tau \\
F_z &= W \tau_k - \frac{1}{2} |W| \delta_k \tau
\end{align*}

If for some reason the above method is desired, the second order flux limiter scheme described in Section 2.17.2.1 reduces to the above scheme if the limiter is set to zero.

2.17.2 Non-linear advection schemes

Non-linear advection schemes invoke non-linear interpolation and are widely used in computational fluid dynamics (non-linear does not refer to the non-linearity of the advection operator). The flux limited advection schemes belong to the class of finite volume methods which neatly ties into the spatial discretization of the model.

When employing the flux limited schemes, first order upwind or direct-space-time method, the time-stepping is switched to forward in time.

2.17.2.1 Second order flux limiters

The second order flux limiter method can be cast in several ways but is generally expressed in terms of other flux approximations. For example, in terms of a first order upwind flux and second order Lax-Wendroff flux, the limited flux is given as:

\[ F = F_1 + \psi(r) F_{LW} \quad (2.148) \]

where \( \psi(r) \) is the limiter function,

\[ F_1 = u \tau_i - \frac{1}{2} |u| \delta_i \tau \]

is the upwind flux,

\[ F_{LW} = F_1 + \frac{|u|}{2} (1 - c) \delta_i \tau \]

is the Lax-Wendroff flux and \( c = \frac{u \Delta t}{\Delta x} \) is the Courant (CFL) number.

The limiter function, \( \psi(r) \), takes the slope ratio

\[ r = \frac{\tau_{i+1} - \tau_i}{\tau_i - \tau_{i-1}} \quad u > 0 \]

\[ r = \frac{\tau_{i-1} - \tau_{i-2}}{\tau_{i-1} - \tau_{i-2}} \quad u < 0 \]

as its argument. There are many choices of limiter function but we only provide the Superbee limiter (Roe 1995 [Roe85]):

\[ \psi(r) = \max[0, \min[1, 2r], \min[2, r]] \]
2.17.2.2 Third order direct space-time

The direct space-time method deals with space and time discretization together (other methods that treat space and time separately are known collectively as the “Method of Lines”). The Lax-Wendroff scheme falls into this category; it adds sufficient diffusion to a second order flux that the forward-in-time method is stable. The upwind biased third order DST scheme is:

\[
F = \begin{cases} 
    u \left( \tau_i - d_0 (\tau_{i-1} - \tau_{i-1}) + d_1 (\tau_{i-1} - \tau_{i-2}) \right) & \forall \ u > 0 \\
    u \left( \tau_i - d_0 (\tau_{i-1} - \tau_{i-1}) - d_1 (\tau_{i+1} - \tau_{i}) \right) & \forall \ u < 0 
\end{cases}
\] (2.149)

where

\[
d_0 = \frac{1}{6} (2 - |c|)(1 - |c|) \\
d_1 = \frac{1}{6} (1 - |c|)(1 + |c|)
\]

The coefficients \(d_0\) and \(d_1\) approach 1/3 and 1/6 respectively as the Courant number, \(c\), vanishes. In this limit, the conventional third order upwind method is recovered. For finite Courant number, the deviations from the linear method are analogous to the diffusion added to centered second order advection in the Lax-Wendroff scheme.

The DST3 method described above must be used in a forward-in-time manner and is stable for \(0 \leq |c| \leq 1\). Although the scheme appears to be forward-in-time, it is in fact third order in time and the accuracy increases with the Courant number! For low Courant number, DST3 produces very similar results (indistinguishable in Figure 2.12) to the linear third order method but for large Courant number, where the linear upwind third order method is unstable, the scheme is extremely accurate (Figure 2.13) with only minor overshoots.
2.17.2.3 Third order direct space-time with flux limiting

The overshoots in the DST3 method can be controlled with a flux limiter. The limited flux is written:

\[
F = \frac{1}{2}(u + |u|)\left(\tau_{i-1} + \psi(r^+)(\tau_i - \tau_{i-1})\right) + \frac{1}{2}(u - |u|)\left(\tau_{i-1} + \psi(r^-)(\tau_i - \tau_{i-1})\right)
\]

(2.150)

where

\[
r^+ = \frac{\tau_{i-1} - \tau_{i-2}}{\tau_i - \tau_{i-1}}
\]

\[
r^- = \frac{\tau_{i+1} - \tau_i}{\tau_i - \tau_{i-1}}
\]

and the limiter is the Sweby limiter:

\[
\psi(r) = \max[0, \min[\min(1, d_0 + d_1 r), \frac{1 - c}{c} r]]
\]
2.17.2.4 Multi-dimensional advection

In many of the aforementioned advection schemes the behavior in multiple dimensions is not necessarily as good as the one dimensional behavior. For instance, a shape preserving monotonic scheme in one dimension can have severe shape distortion in two dimensions if the two components of horizontal fluxes are treated independently. There is a large body of literature on the subject dealing with this problem and among the fixes are operator and flux splitting methods, corner flux methods, and more. We have adopted a variant on the standard splitting methods that allows the flux calculations to be implemented as if in one dimension:

\[
\begin{align*}
\tau_{n+1/3} &= \tau^n - \Delta t \left( \frac{1}{\Delta x} \delta_x F_x(\tau^n) - \tau^n \frac{1}{\Delta x} \delta_x u \right) \\
\tau_{n+2/3} &= \tau^{n+1/3} - \Delta t \left( \frac{1}{\Delta y} \delta_y F_y(\tau^{n+1/3}) - \tau^n \frac{1}{\Delta y} \delta_y v \right) \\
\tau_{n+3/3} &= \tau^{n+2/3} - \Delta t \left( \frac{1}{\Delta r} \delta_r F_r(\tau^{n+2/3}) - \tau^n \frac{1}{\Delta r} \delta_r w \right)
\end{align*}
\]  

(2.151)

In order to incorporate this method into the general model algorithm, we compute the effective tendency rather than update the tracer so that other terms such as diffusion are using the \( n \) time-level and not the updated \( n+3/3 \) quantities:

\[
G_{adv}^{n+1/2} = \frac{1}{\Delta t}(\tau^{n+3/3} - \tau^n)
\]

So that the over all time-stepping looks like:

\[
\tau^{n+1} = \tau^n + \Delta t \left( G_{adv}^{n+1/2} + G_{diff}(\tau^n) + G_{forcing}^n \right)
\]

S/R GAD_ADVECTION

- \( \tau \): tracer (argument)
- \( G_{adv}^{n+1/2} \): gTracer (argument)
- \( F_x, F_y, F_r \): aF (local)
- \( U \): uTrans (local)
- \( V \): vTrans (local)
- \( W \): rTrans (local)

A schematic of multi-dimension time stepping for the cube sphere configuration is show in Figure 2.11.

2.17.3 Comparison of advection schemes

Table 2.2 shows a summary of the different advection schemes available in MITgcm. “AB” stands for Adams-Bashforth and “DST” for direct space-time. The code corresponds to the number used to select the corresponding advection scheme in the parameter file (e.g., \texttt{tempAdvScheme=3} in file data selects the 3rd order upwind advection scheme for temperature advection).
Figure 2.11: Multi-dimensional advection time-stepping with cubed-sphere topology.
Table 2.2: MITgcm Advection Schemes

<table>
<thead>
<tr>
<th>Advection Scheme</th>
<th>Code</th>
<th>Use AB?</th>
<th>Use multi-dim?</th>
<th>Stencil (1-D)</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>1st order upwind</td>
<td>1</td>
<td>no</td>
<td>yes*</td>
<td>3</td>
<td>linear $\tau$, non-linear $\vec{v}$</td>
</tr>
<tr>
<td>centered 2nd order</td>
<td>2</td>
<td>yes</td>
<td>no</td>
<td>3</td>
<td>linear</td>
</tr>
<tr>
<td>3rd order upwind</td>
<td>3</td>
<td>yes</td>
<td>no</td>
<td>5</td>
<td>linear $\tau$</td>
</tr>
<tr>
<td>centered 4th order</td>
<td>4</td>
<td>yes</td>
<td>no</td>
<td>5</td>
<td>linear</td>
</tr>
<tr>
<td>2nd order DST (Lax-Wendroff)</td>
<td>20</td>
<td>no</td>
<td>yes*</td>
<td>3</td>
<td>linear $\tau$, non-linear $\vec{v}$</td>
</tr>
<tr>
<td>3rd order DST</td>
<td>30</td>
<td>no</td>
<td>yes*</td>
<td>5</td>
<td>linear $\tau$, non-linear $\vec{v}$</td>
</tr>
<tr>
<td>2nd order flux limiters</td>
<td>77</td>
<td>no</td>
<td>yes*</td>
<td>5</td>
<td>non-linear</td>
</tr>
<tr>
<td>3rd order DST flux limiter</td>
<td>33</td>
<td>no</td>
<td>yes*</td>
<td>5</td>
<td>non-linear</td>
</tr>
<tr>
<td>piecewise parabolic w/&quot;null&quot; limiter</td>
<td>40</td>
<td>no</td>
<td>yes</td>
<td>7</td>
<td>non-linear</td>
</tr>
<tr>
<td>piecewise parabolic w/&quot;mono&quot; limiter</td>
<td>41</td>
<td>no</td>
<td>yes</td>
<td>7</td>
<td>non-linear</td>
</tr>
<tr>
<td>piecewise parabolic w/&quot;weno&quot; limiter</td>
<td>42</td>
<td>no</td>
<td>yes</td>
<td>7</td>
<td>non-linear</td>
</tr>
<tr>
<td>piecewise quartic w/&quot;null&quot; limiter</td>
<td>50</td>
<td>no</td>
<td>yes</td>
<td>9</td>
<td>non-linear</td>
</tr>
<tr>
<td>piecewise quartic w/&quot;mono&quot; limiter</td>
<td>51</td>
<td>no</td>
<td>yes</td>
<td>9</td>
<td>non-linear</td>
</tr>
<tr>
<td>piecewise quartic w/&quot;weno&quot; limiter</td>
<td>52</td>
<td>no</td>
<td>yes</td>
<td>9</td>
<td>non-linear</td>
</tr>
<tr>
<td>7th order one-step method w/monotonicity preserving limiter</td>
<td>7</td>
<td>no</td>
<td>yes</td>
<td>9</td>
<td>non-linear</td>
</tr>
<tr>
<td>second order-moment Prather</td>
<td>80</td>
<td>no</td>
<td>yes</td>
<td>3</td>
<td>non-linear</td>
</tr>
<tr>
<td>second order-moment Prather w/limiter</td>
<td>81</td>
<td>no</td>
<td>yes</td>
<td>3</td>
<td>non-linear</td>
</tr>
</tbody>
</table>

*indicates that either the multi-dim advection algorithm or standard approach can be utilized, controlled by a namelist parameter multiDimAdvection (in these cases, given that these schemes were designed to use multi-dim advection, using the standard approach is not recommended). The minimum size of the required tile overlap region (OLx, OLx) is (stencil size -1)/2. The minimum overlap required by the model in general is 2, so for some of the above choices the advection scheme will not cost anything in terms of an additional overlap requirement, but especially given a small tile size, using scheme 7 for example would require costly additional overlap points (note a cube sphere grid with a “wet-corner” requires doubling this overlap!) In the ‘Comments’ column, $\tau$ refers to tracer advection, $\vec{v}$ momentum advection.

Shown in Figure 2.12 and Figure 2.13 is a 1-D comparison of advection schemes. Here we advect both a smooth hill and a hill with a more abrupt shock. Figure 2.12 shown the result for a weak flow (low Courant number) whereas Figure 2.13 shows the result for a stronger flow (high Courant number).

Figure 2.14, Figure 2.15 and Figure 2.16 show solutions to a simple diagonal advection problem using a selection of schemes for low, moderate and high Courant numbers, respectively. The top row shows the linear schemes, integrated with the Adams-Bashforth method. Theses schemes are clearly unstable for the high Courant number and weakly unstable for the moderate Courant number. The presence of false extrema is very apparent for all Courant numbers. The middle row shows solutions obtained with the unlimited but multi-dimensional schemes. These solutions also exhibit false extrema though the pattern now shows symmetry due to the multi-dimensional scheme. Also, the schemes are stable at high Courant number where the linear schemes weren’t. The bottom row (left and middle) shows the limited schemes and most obvious is the absence of false extrema. The accuracy and stability of the unlimited non-linear schemes is retained at high Courant number but at low Courant number the tendency is to lose amplitude in sharp peaks due to diffusion. The one dimensional tests shown in Figure 2.12 and Figure 2.13 show this phenomenon.

Finally, the bottom left and right panels use the same advection scheme but the right does not use the multi-dimensional method. At low Courant number this appears to not matter but for moderate Courant number severe distortion of the feature is apparent. Moreover, the stability of the multi-dimensional scheme is determined by the maximum Courant...
Figure 2.12: Comparison of 1-D advection schemes: Courant number is 0.05 with 60 points and solutions are shown for T=1 (one complete period). a) Shows the upwind biased schemes; first order upwind, DST3, third order upwind and second order upwind. b) Shows the centered schemes; Lax-Wendroff, DST4, centered second order, centered fourth order and finite volume fourth order. c) Shows the second order flux limiters: minmod, Superbee, MC limiter and the van Leer limiter. d) Shows the DST3 method with flux limiters due to Sweby with $\mu = 1$, $\mu = c / (1 - c)$ and a fourth order DST method with Sweby limiter, $\mu = c / (1 - c)$. 

2.17. Advection schemes
Figure 2.13: Comparison of 1-D advection schemes: Courant number is 0.89 with 60 points and solutions are shown for $T=1$ (one complete period). a) Shows the upwind biased schemes: first order upwind and DST3. Third order upwind and second order upwind are unstable at this Courant number. b) Shows the centered schemes; Lax–Wendroff, DST4. Centered second order, centered fourth order and finite volume fourth order are unstable at this Courant number. c) Shows the second order flux limiters: minmod, Superbee, MC limiter and the van Leer limiter. d) Shows the DST3 method with flux limiters due to Sweby with $\mu = 1, \mu = c/(1-c)$ and a fourth order DST method with Sweby limiter, $\mu = c/(1-c)$. 

86 Chapter 2. Discretization and Algorithm
number applied of each dimension while the stability of the method of lines is determined by the sum. Hence, in the high Courant number plot, the scheme is unstable.

With many advection schemes implemented in the code two questions arise: “Which scheme is best?” and “Why don’t you just offer the best advection scheme?”. Unfortunately, no one advection scheme is “the best” for all particular applications and for new applications it is often a matter of trial to determine which is most suitable. Here are some guidelines but these are not the rule:

- If you have a coarsely resolved model, using a positive or upwind biased scheme will introduce significant diffusion to the solution and using a centered higher order scheme will introduce more noise. In this case, simplest may be best.
- If you have a high resolution model, using a higher order scheme will give a more accurate solution but scale-selective diffusion might need to be employed. The flux limited methods offer similar accuracy in this regime.
- If your solution has shocks or propagating fronts then a flux limited scheme is almost essential.
- If your time-step is limited by advection, the multi-dimensional non-linear schemes have the most stability (up to Courant number 1).
- If you need to know how much diffusion/dissipation has occurred you will have a lot of trouble figuring it out with a non-linear method.
- The presence of false extrema is non-physical and this alone is the strongest argument for using a positive scheme.

### 2.18 Shapiro Filter

The Shapiro filter (Shapiro 1970) [Sha70] is a high order horizontal filter that efficiently remove small scale grid noise without affecting the physical structures of a field. It is applied at the end of the time step on both velocity and tracer fields.

Three different space operators are considered here (S1, S2 and S4). They differ essentially by the sequence of derivative in both X and Y directions. Consequently they show different damping response function specially in the diagonal directions X+Y and X-Y.

Space derivatives can be computed in the real space, taking into account the grid spacing. Alternatively, a pure computational filter can be defined, using pure numerical differences and ignoring grid spacing. This later form is stable whatever the grid is, and therefore specially useful for highly anisotropic grid such as spherical coordinate grid. A damping time-scale parameter $\tau_{shap}$ defines the strength of the filter damping.

The three computational filter operators are:

- $S1c : \left[ 1 - \frac{\Delta t}{\tau_{shap}} \left\{ \frac{1}{4} \delta_{ii}^n + \frac{1}{4} \delta_{jj}^n \right\} \right]$
- $S2c : \left[ 1 - \frac{\Delta t}{\tau_{shap}} \left\{ \frac{1}{8} (\delta_{ii} + \delta_{jj})^n \right\} \right]$
- $S4c : \left[ 1 - \frac{\Delta t}{\tau_{shap}} \left( \frac{1}{4} \delta_{ii}^n \right) \right] \left[ 1 - \frac{\Delta t}{\tau_{shap}} \left( \frac{1}{4} \delta_{jj}^n \right) \right]$

In addition, the S2 operator can easily be extended to a physical space filter:

- $S2g : \left[ 1 - \frac{\Delta t}{\tau_{shap}} \left( \frac{L_{shap}^2}{8} \nabla^2 \right)^n \right]$

with the Laplacian operator $\nabla^2$ and a length scale parameter $L_{shap}$. The stability of this S2g filter requires $L_{shap} < \text{Min}^{(Global)}(\Delta x, \Delta y)$.
Figure 2.14: Comparison of advection schemes in two dimensions; diagonal advection of a resolved Gaussian feature. Courant number is 0.01 with $30 \times 30$ points and solutions are shown for $T=1/2$. White lines indicate zero crossing (i.e., the presence of false minima). The left column shows the second order schemes; top) centered second order with Adams-Bashforth, middle) Lax-Wendroff and bottom) Superbee flux limited. The middle column shows the third order schemes; top) upwind biased third order with Adams-Bashforth, middle) third order direct space-time method and bottom) the same with flux limiting. The top right panel shows the centered fourth order scheme with Adams-Bashforth and right middle panel shows a fourth order variant on the DST method. Bottom right panel shows the Superbee flux limiter (second order) applied independently in each direction (method of lines).
Figure 2.15: Comparison of advection schemes in two dimensions; diagonal advection of a resolved Gaussian feature. Courant number is 0.27 with $30 \times 30$ points and solutions are shown for $T=1/2$. White lines indicate zero crossing (i.e., the presence of false minima). The left column shows the second order schemes; top) centered second order with Adams-Bashforth, middle) Lax-Wendroff and bottom) Superbee flux limited. The middle column shows the third order schemes; top) upwind biased third order with Adams-Bashforth, middle) third order direct space-time method and bottom) the same with flux limiting. The top right panel shows the centered fourth order scheme with Adams-Bashforth and right middle panel shows a fourth order variant on the DST method. Bottom right panel shows the Superbee flux limiter (second order) applied independently in each direction (method of lines).
Figure 2.16: Comparison of advection schemes in two dimensions; diagonal advection of a resolved Gaussian feature. Courant number is 0.47 with 30 × 30 points and solutions are shown for T=1/2. White lines indicate zero crossings and initial maximum values (i.e., the presence of false extrema). The left column shows the second order schemes; top) centered second order with Adams-Bashforth, middle) Lax-Wendroff and bottom) Superbee flux limited. The middle column shows the third order schemes; top) upwind biased third order with Adams-Bashforth, middle) third order direct space-time method and bottom) the same with flux limiting. The top right panel shows the centered fourth order scheme with Adams-Bashforth and right middle panel shows a fourth order variant on the DST method. Bottom right panel shows the Superbee flux limiter (second order) applied independently in each direction (method of lines).
2.18.1 SHAP Diagnostics

<table>
<thead>
<tr>
<th>&lt;-Name-&gt;</th>
<th>Levs</th>
<th>parsing code</th>
<th>&lt;-Units-&gt;</th>
<th>&lt;- Tile (max=80c)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SHAP_dT</td>
<td>5</td>
<td>SM MR</td>
<td>K/s</td>
<td>Temperature Tendency due to Shapiro Filter</td>
</tr>
<tr>
<td>SHAP_dS</td>
<td>5</td>
<td>SM MR</td>
<td>g/kg/s</td>
<td>Specific Humidity Tendency due to Shapiro Filter</td>
</tr>
<tr>
<td>SHAP_dU</td>
<td>5</td>
<td>UU 148MR</td>
<td>m/s²</td>
<td>Zonal Wind Tendency due to Shapiro Filter</td>
</tr>
<tr>
<td>SHAP_dV</td>
<td>5</td>
<td>VV 147MR</td>
<td>m/s²</td>
<td>Meridional Wind Tendency due to Shapiro Filter</td>
</tr>
</tbody>
</table>

2.19 Nonlinear Viscosities for Large Eddy Simulation

In Large Eddy Simulations (LES), a turbulent closure needs to be provided that accounts for the effects of subgridscale motions on the large scale. With sufficiently powerful computers, we could resolve the entire flow down to the molecular viscosity scales ($L_\nu \approx 1cm$). Current computation allows perhaps four decades to be resolved, so the largest problem computationally feasible would be about 10m. Most oceanographic problems are much larger in scale, so some form of LES is required, where only the largest scales of motion are resolved, and the subgridscale effects on the large-scale are parameterized.

To formalize this process, we can introduce a filter over the subgridscale $L$: $u_\alpha \rightarrow \tilde{u}_\alpha$ and $b \rightarrow \tilde{b}$. This filter has some intrinsic length and time scales, and we assume that the flow at that scale can be characterized with a single velocity scale ($\bar{V}$) and vertical buoyancy gradient ($N^2$). The filtered equations of motion in a local Mercator projection about the gridpoint in question (see Appendix for notation and details of approximation) are:

\[
\frac{D\tilde{\pi}}{Dt} - \frac{\tilde{v} \sin \theta}{\cos \theta_0} \frac{M_{Ro}}{Ro} \frac{\partial \pi}{\partial x} = - \left( \frac{D\tilde{u}}{Dt} - \frac{D\tilde{v}}{Dt} \right) + \frac{\nabla^2\tilde{\pi}}{Re} \tag{2.152}
\]

\[
\frac{D\tilde{\pi}}{Dt} - \frac{\tilde{u} \sin \theta}{\cos \theta_0} \frac{M_{Ro}}{Ro} \frac{\partial \pi}{\partial y} = - \left( \frac{D\tilde{u}}{Dt} - \frac{D\tilde{v}}{Dt} \right) + \frac{\nabla^2\tilde{\pi}}{Re} \tag{2.153}
\]

\[
\frac{D\tilde{\pi}}{Dt} + \frac{\partial \tilde{\pi}}{\partial z} - \frac{\tilde{b}}{Fr^2 \lambda^2} = - \left( \frac{D\tilde{w}}{Dt} - \frac{D\tilde{b}}{Dt} \right) + \frac{\nabla^2\tilde{\pi}}{Re}
\]

\[
\frac{D\tilde{b}}{Dt} + \bar{w} = - \left( \frac{D\tilde{w}}{Dt} - \frac{D\tilde{b}}{Dt} \right) + \frac{\nabla^2\tilde{b}}{Pr Re}
\]

\[
\mu^2 \left( \frac{\partial \tilde{\pi}}{\partial x} + \frac{\partial \tilde{\pi}}{\partial y} + \frac{\partial \tilde{\pi}}{\partial z} = 0 \right) \tag{2.154}
\]

Tildes denote multiplication by $\cos \theta / \cos \theta_0$ to account for converging meridians.

The ocean is usually turbulent, and an operational definition of turbulence is that the terms in parentheses (the ’eddy’ terms) on the right of (2.152) - (2.154)) are of comparable magnitude to the terms on the left-hand side. The terms proportional to the inverse of , instead, are many orders of magnitude smaller than all of the other terms in virtually every oceanic application.
2.19.1 Eddy Viscosity

A turbulent closure provides an approximation to the ‘eddy’ terms on the right of the preceding equations. The simplest form of LES is just to increase the viscosity and diffusivity until the viscous and diffusive scales are resolved. That is, we approximate (2.152) - (2.154):

\[
\begin{align*}
\left( \frac{D\bar{u}}{Dt} - \frac{D\bar{u}}{Dt} \right) & \approx \nabla^2 \bar{u} \frac{\dot{\theta}}{Re} + \frac{\partial^2 \bar{u}}{\partial z^2} \frac{\dot{\theta}}{Re} , \\
\left( \frac{D\bar{v}}{Dt} - \frac{D\bar{v}}{Dt} \right) & \approx \nabla^2 \bar{v} \frac{\dot{\theta}}{Re} + \frac{\partial^2 \bar{v}}{\partial z^2} \frac{\dot{\theta}}{Re} , \\
\left( \frac{D\bar{w}}{Dt} - \frac{D\bar{w}}{Dt} \right) & \approx \nabla^2 \bar{w} \frac{\dot{\theta}}{Re} + \frac{\partial^2 \bar{w}}{\partial z^2} \frac{\dot{\theta}}{Re} , \\
\left( \frac{D\bar{b}}{Dt} - \frac{D\bar{b}}{Dt} \right) & \approx \nabla^2 \bar{b} \frac{\dot{\theta}}{PrRe} + \frac{\partial^2 \bar{b}}{\partial z^2} \frac{\dot{\theta}}{PrRe} .
\end{align*}
\]

2.19.1.1 Reynolds-Number Limited Eddy Viscosity

One way of ensuring that the gridscale is sufficiently viscous (i.e., resolved) is to choose the eddy viscosity \(A_h\) so that the gridscale horizontal Reynolds number based on this eddy viscosity, \(Re_h\), is \(O(1)\). That is, if the gridscale is to be viscous, then the viscosity should be chosen to make the viscous terms as large as the advective ones. Bryan et al. (1975) [BMP75] notes that a computational mode is squelched by using \(Re_h < 2\).

MITgcm users can select horizontal eddy viscosities based on \(Re_h\) using two methods. 1) The user may estimate the velocity scale expected from the calculation and grid spacing and set \(viscAh\) to satisfy \(Re_h < 2\). 2) The user may use \(viscAhReMax\), which ensures that the viscosity is always chosen so that \(Re_h < viscAhReMax\). This last option should be used with caution, however, since it effectively implies that viscous terms are fixed in magnitude relative to advective terms. While it may be a useful method for specifying a minimum viscosity with little effort, tests Bryan et al. (1975) [BMP75] have shown that setting \(viscAhReMax = 2\) often tends to increase the viscosity substantially over other more 'physical' parameterizations below, especially in regions where gradients of velocity are small (and thus turbulence may be weak), so perhaps a more liberal value should be used, e.g. \(viscAhReMax = 10\).

While it is certainly necessary that viscosity be active at the gridscale, the wavelength where dissipation of energy or enstrophy occurs is not necessarily \(L = A_h / U\). In fact, it is by ensuring that either the dissipation of energy in a 3-d turbulent cascade (Smagorinsky) or dissipation of enstrophy in a 2-d turbulent cascade (Leith) is resolved that these parameterizations derive their physical meaning.

2.19.1.2 Vertical Eddy Viscosities

Vertical eddy viscosities are often chosen in a more subjective way, as model stability is not usually as sensitive to vertical viscosity. Usually the ‘observed’ value from finescale measurements is used (e.g. \(viscAr \approx 1 \times 10^{-4} m^2/s\)). However, Smagorinsky (1993) [Sma93] notes that the Smagorinsky parameterization of isotropic turbulence implies a value of the vertical viscosity as well as the horizontal viscosity (see below).
2.19.1.3 Smagorinsky Viscosity

Some suggest (see Smagorinsky 1963 [Sma63]; Smagorinsky 1993 [Sma93]) choosing a viscosity that depends on the resolved motions. Thus, the overall viscous operator has a nonlinear dependence on velocity. Smagorinsky chose his form of viscosity by considering Kolmogorov’s ideas about the energy spectrum of 3-d isotropic turbulence.

Kolmogorov supposed that energy is injected into the flow at large scales (small $k$) and is ‘cascaded’ or transferred conservatively by nonlinear processes to smaller and smaller scales until it is dissipated near the viscous scale. By setting the energy flux through a particular wavenumber $k$, $\epsilon$, to be a constant in $k$, there is only one combination of viscosity and energy flux that has the units of length, the Kolmogorov wavelength. It is $L_\epsilon(\nu) \propto \pi \epsilon^{-1/4} \nu^{3/4}$ (the $\pi$ stems from conversion from wavenumber to wavelength). To ensure that this viscous scale is resolved in a numerical model, the gridscale should be decreased until $L_\epsilon(A_h) > L$ (so-called Direct Numerical Simulation, or DNS). Alternatively, an eddy viscosity can be used and the corresponding Kolmogorov length can be made larger than the gridscale, $L_\epsilon(A_h) \propto \pi \epsilon^{-1/4} A_h^{3/4}$ (for Large Eddy Simulation or LES).

There are two methods of ensuring that the Kolmogorov length is resolved in MITgcm. 1) The user can estimate the flux of energy through spectral space for a given simulation and adjust grid spacing or $\text{viscAh}$ to ensure that $L_\epsilon(A_h) > L$; 2) The user may use the approach of Smagorinsky with $\text{viscC2Smag}$, which estimates the energy flux at every grid point, and adjusts the viscosity accordingly.

Smagorinsky formed the energy equation from the momentumequations by dotting them with velocity. There are some complications when using the hydrostatic approximation as described by Smagorinsky (1993) [Sma93]. The positive definite energy dissipation by horizontal viscosity in a hydrostatic flow is $\nu D^2$, where $D$ is the deformation rate at the viscous scale. According to Kolmogorov’s theory, this should be a good approximation to the energy flux at any wavenumber $\epsilon \approx \nu D^2$. Kolmogorov and Smagorinsky noted that using an eddy viscosity that exceeds the molecular value $\nu$ should ensure that the energy flux through viscous scale set by the eddy viscosity is the same as it would have been had we resolved all the way to the true viscous scale. That is, $\epsilon \approx A_h^{\text{Smag}} D^2$. If we use this approximation to estimate the Kolmogorov viscous length, then

$$L_\epsilon(A_h^{\text{Smag}}) \propto \pi \epsilon^{-1/4} A_h^{3/4} \approx \pi(A_h^{\text{Smag}} D^2)^{1/4} A_h^{3/4} = \pi A_h^{1/2} D^{1/2}$$

To make $L_\epsilon(A_h^{\text{Smag}})$ scale with the gridscale, then

$$A_h^{\text{Smag}} = \left( \frac{\text{viscC2Smag}}{\pi} \right)^2 L^2 |D|$$

Where the deformation rate appropriate for hydrostatic flows with shallow-water scaling is

$$|D| = \sqrt{\left( \frac{\partial \tilde{u}}{\partial x} - \frac{\partial \tilde{v}}{\partial y} \right)^2 + \left( \frac{\partial \tilde{u}}{\partial y} + \frac{\partial \tilde{v}}{\partial x} \right)^2}$$

The coefficient $\text{viscC2Smag}$ is what an MITgcm user sets, and it replaces the proportionality in the Kolmogorov length with an equality. Others (Griffies and Hallberg, 2000 [GH00]) suggest values of $\text{viscC2Smag}$ from 2.2 to 4 for oceanic problems. Smagorinsky (1993) [Sma93] shows that values from 0.2 to 0.9 have been used in atmospheric modeling.

Smagorinsky (1993) [Sma93] shows that a corresponding vertical viscosity should be used:

$$A_v^{\text{Smag}} = \left( \frac{\text{viscC2Smag}}{\pi} \right)^2 H^2 \sqrt{\left( \frac{\partial \tilde{u}}{\partial z} \right)^2 + \left( \frac{\partial \tilde{v}}{\partial z} \right)^2}$$

This vertical viscosity is currently not implemented in MITgcm.
2.19.1.4 Leith Viscosity

Leith (1968, 1996) [Lei68] [Lei96] notes that 2-D turbulence is quite different from 3-D. In 2-D turbulence, energy cascades to larger scales, so there is no concern about resolving the scales of energy dissipation. Instead, another quantity, enstrophy, (which is the vertical component of vorticity squared) is conserved in 2-D turbulence, and it cascades to smaller scales where it is dissipated.

Following a similar argument to that above about energy flux, the enstrophy flux is estimated to be equal to the positive-definite gridscale dissipation rate of enstrophy \( \eta \approx A_{h\text{Leith}}^2 |\nabla \omega_3|^2 \). By dimensional analysis, the enstrophy-dissipation scale is \( L_\eta(A_{h\text{Leith}}) \propto \pi A_{h\text{Leith}}^{1/2} \eta^{-1/6} \). Thus, the Leith-estimated length scale of enstrophy-dissipation and the resulting eddy viscosity are

\[
L_\eta(A_{h\text{Leith}}) \propto \pi A_{h\text{Leith}}^{1/2} \eta^{-1/6} = \pi A_{h\text{Leith}}^{1/3} |\nabla \omega_3|^{-1/3} \tag{2.162}
\]

\[
A_{h\text{Leith}} = \left( \frac{\text{viscC2Leith}}{\pi} \right)^{\frac{3}{6}} L^3 |\nabla \omega_3| \tag{2.163}
\]

\[
|\nabla \omega_3| \equiv \sqrt{\left( \frac{\partial}{\partial x} \left( \frac{\partial \tilde{u}}{\partial x} - \frac{\partial \tilde{v}}{\partial y} \right) \right)^2 + \left( \frac{\partial}{\partial y} \left( \frac{\partial \tilde{u}}{\partial x} - \frac{\partial \tilde{v}}{\partial y} \right) \right)^2} \tag{2.164}
\]

The runtime flag useFullLeith controls whether or not to calculate the full gradients for the Leith viscosity (.TRUE.) or to use an approximation (.FALSE.). The only reason to set useFullLeith = .FALSE. is if your simulation fails when computing the gradients. This can occur when using the cubed sphere and other complex grids.

2.19.1.5 Modified Leith Viscosity

The argument above for the Leith viscosity parameterization uses concepts from purely 2-dimensional turbulence, where the horizontal flow field is assumed to be non-divergent. However, oceanic flows are only quasi-two-dimensional. While the barotropic flow, or the flow within isopycnal layers may behave nearly as two-dimensional turbulence, there is a possibility that these flows will be divergent. In a high-resolution numerical model, these flows may be substantially divergent near the grid scale, and in fact, numerical instabilities exist which are only horizontally divergent and have little vertical vorticity. This causes a difficulty with the Leith viscosity, which can only respond to buildup of vorticity at the grid scale.

MITgcm offers two options for dealing with this problem. 1) The Smagorinsky viscosity can be used instead of Leith, or in conjunction with Leith – a purely divergent flow does cause an increase in Smagorinsky viscosity; 2) The viscC2LeithD parameter can be set. This is a damping specifically targeting purely divergent instabilities near the gridscale. The combined viscosity has the form:

\[
A_{h\text{Leith}} = L^3 \sqrt{\left( \frac{\text{viscC2Leith}}{\pi} \right)^6 |\nabla \omega_3|^2 + \left( \frac{\text{viscC2LeithD}}{\pi} \right)^6 |\nabla \nabla \cdot \tilde{u}_h|^2} \tag{2.165}
\]

\[
|\nabla \nabla \cdot \tilde{u}_h| \equiv \sqrt{\left( \frac{\partial}{\partial x} \left( \frac{\partial \tilde{u}}{\partial x} + \frac{\partial \tilde{v}}{\partial y} \right) \right)^2 + \left( \frac{\partial}{\partial y} \left( \frac{\partial \tilde{u}}{\partial x} + \frac{\partial \tilde{v}}{\partial y} \right) \right)^2} \tag{2.166}
\]

Whether there is any physical rationale for this correction is unclear, but the numerical consequences are good. The divergence in flows with the grid scale larger or comparable to the Rossby radius is typically much smaller than the vorticity, so this adjustment only rarely adjusts the viscosity if viscC2LeithD = viscC2Leith. However, the rare regions where this viscosity acts are often the locations for the largest values of vertical velocity in the domain. Since the CFL condition on vertical velocity is often what sets the maximum timestep, this viscosity may substantially increase the allowable timestep without severely compromising the verity of the simulation. Tests have shown that in some calculations, a timestep three times larger was allowed when viscC2LeithD = viscC2Leith.
2.19.1.6 Quasi-Geostrophic Leith Viscosity

A variant of Leith viscosity can be derived for quasi-geostrophic dynamics. This leads to a slightly different equation for the viscosity that includes a contribution from quasi-geostrophic vortex stretching (Bachman et al. 2017 [BFKP17]). The viscosity is given by

$$\nu_* = \left(\frac{\Lambda \Delta s}{\pi}\right)^3 \left| \nabla_h (f \hat{z}) + \nabla_h (\nabla \times \mathbf{v}_{h*}) + \partial_z \frac{f}{N^2} \nabla_h b \right|$$

where $\Lambda$ is a tunable parameter of $\mathcal{O}(1)$, $\Delta s = \sqrt{\Delta x \Delta y}$ is the grid scale, $f \hat{z}$ is the vertical component of the Coriolis parameter, $\mathbf{v}_{h*}$ is the horizontal velocity, $N^2$ is the Brunt-Väisälä frequency, and $b$ is the buoyancy.

However, the viscosity given by (2.167) does not constrain purely divergent motions. As such, a small $\mathcal{O}(\epsilon)$ correction is added

$$\nu_* = \left(\frac{\Lambda \Delta s}{\pi}\right)^3 \left| \nabla_h (f \hat{z}) + \nabla_h (\nabla \times \mathbf{v}_{h*}) + \partial_z \frac{f}{N^2} \nabla_h b \right| + \left| \nabla [\nabla \cdot \mathbf{v}_h] \right|^2$$

This form is, however, numerically awkward; as the Brunt-Väisälä Frequency becomes very small in regions of weak or vanishing stratification, the vortex stretching term becomes very large. The resulting large viscosities can lead to numerical instabilities. Bachman et al. (2017) [BFKP17] present two limiting forms for the viscosity based on flow parameters such as $Fr_*$, the Froude number, and $Ro_*$, the Rossby number. The second of which,

$$\nu_* = \left(\frac{\Lambda \Delta s}{\pi}\right)^3 \sqrt{\min \left( \left| \nabla_h q_{2s} + \partial_z \frac{f^2}{N^2} \nabla_h b \right|, \left(1 + \frac{Fr_*^2}{Ro_*^2} + Fr_*^4 \right) \left| \nabla_h q_{2s} \right| \right)^2 + \left| \nabla [\nabla \cdot \mathbf{v}_h] \right|^2},$$

has been implemented and is active when #define ALLOW_LEITH_QG is included in a copy of MOM_COMMON_OPTIONS.h in a code mods directory (specified through -mods command line option in genmake2).

LeithQG viscosity is designed to work best in simulations that resolve some mesoscale features. In simulations that are too coarse to permit eddies or fine enough to resolve submesoscale features, it should fail gracefully. The non-dimensional parameter viscC2LeithQG corresponds to $\Lambda$ in the above equations and scales the viscosity; the recommended value is 1.

There is no reason to use the quasi-geostrophic form of Leith at the same time as either standard Leith or modified Leith. Therefore, the model will not run if non-zero values have been set for these coefficients; the model will stop during the configuration check. LeithQG can be used regardless of the setting for useFullLeith. Just as for the other forms of Leith viscosity, this flag determines whether or not the full gradients are used. The simplified gradients were originally intended for use on complex grids, but have been shown to produce better kinetic energy spectra even on very straightforward grids.

To add the LeithQG viscosity to the GMRedi coefficient, as was done in some of the simulations in Bachman et al. (2017) [BFKP17], #define ALLOW_LEITH_QG must be specified, as described above. In addition to this, the compile-time flag ALLOW_GM_LEITH_QG must also be defined in a (-mods) copy of GMREDI_OPTIONS.h when the model is compiled, and the runtime parameter GM_useLeithQG set to .TRUE. in data.gmredi. This will use the value of viscC2LeithQG specified in the data input file to compute the coefficient.
2.19.1.7 Courant–Freidrichs–Lewy Constraint on Viscosity

Whatever viscosities are used in the model, the choice is constrained by gridscale and timestep by the Courant–Freidrichs–Lewy (CFL) constraint on stability:

\[ A_h < \frac{L^2}{4\Delta t} \]
\[ A_4 \leq \frac{L^4}{32\Delta t} \]

The viscosities may be automatically limited to be no greater than these values in MITgcm by specifying \( \text{viscAhGridMax} < 1 \) and \( \text{viscA4GridMax} < 1 \). Similarly-scaled minimum values of viscosities are provided by \( \text{viscAhGridMin} \) and \( \text{viscA4GridMin} \), which if used, should be set to values \( \ll 1 \). \( L \) is roughly the gridscale (see below).

Following Griffies and Hallberg (2000) [GH00], we note that there is a factor of \( \Delta x^2/8 \) difference between the harmonic and biharmonic viscosities. Thus, whenever a non-dimensional harmonic coefficient is used in the MITgcm (e.g. \( \text{viscAhGridMax} < 1 \)), the biharmonic equivalent is scaled so that the same non-dimensional value can be used (e.g. \( \text{viscA4GridMax} < 1 \)).

2.19.1.8 Biharmonic Viscosity

Holland (1978) [Hol78] suggested that eddy viscosities ought to be focused on the dynamics at the grid scale, as larger motions would be 'resolved'. To enhance the scale selectivity of the viscous operator, he suggested a biharmonic eddy viscosity instead of a harmonic (or Laplacian) viscosity:

\[
\left( \frac{D\hat{u}}{Dt} - \frac{D\hat{u}}{Dt} \right) \approx -\nabla_h \hat{u} \frac{\text{Re}_h}{\text{Re}_v} + \frac{\partial^2 \hat{u}}{\partial z^2} \frac{\text{Re}_v}{\text{Re}_v}
\]

Holland (1978) [Hol78] proposed that if one scales the biharmonic viscosity by stability considerations, then the biharmonic viscous terms will be similarly active to harmonic viscous terms at the gridscale of the model, but much less active on larger scale motions. Similarly, a biharmonic diffusivity can be used for less diffusive flows.

In practice, biharmonic viscosity and diffusivity allow a less viscous, yet numerically stable, simulation than harmonic viscosity and diffusivity. However, there is no physical rationale for such operators being of leading order, and more boundary conditions must be specified than for the harmonic operators. If one considers the approximations of (2.155) - (2.158) and (2.170) - (2.171) to be terms in the Taylor series expansions of the eddy terms as functions of the large-scale gradient, then one can argue that both harmonic and biharmonic terms would occur in the series, and the only question is the choice of coefficients. Using biharmonic viscosity alone implies that one zeros the first non-vanishing term in the Taylor series, which is unsupported by any fluid theory or observation.

Nonetheless, MITgcm supports a plethora of biharmonic viscosities and diffusivities, which are controlled with parameters named similarly to the harmonic viscosities and diffusivities with the substitution \( h \rightarrow 4 \) in the MITgcm parameter name. MITgcm also supports biharmonic Leith and Smagorinsky viscosities:

\[
A_{4Smag} = \left( \frac{\text{viscC4Smag}}{\pi} \right)^2 \frac{L^4}{8|D|}
\]
\[ A_{4\text{Leith}} = \frac{L^5}{8} \sqrt{\left(\frac{\text{viscC4Leith}}{\pi}\right)^6 |\nabla \omega^3|^2 + \left(\frac{\text{viscC4LeithD}}{\pi}\right)^6 |\nabla \nabla \cdot \mathbf{u}_h|^2} \]  \hspace{1cm} (2.172)

However, it should be noted that unlike the harmonic forms, the biharmonic scaling does not easily relate to whether energy-dissipation or enstrophy-dissipation scales are resolved. If similar arguments are used to estimate these scales and scale them to the gridscale, the resulting biharmonic viscosities should be:

\[ A_{4\text{Smag}} = \left(\frac{\text{viscC4Smag}}{\pi}\right)^5 L^5 |\nabla^2 \mathbf{u}_h| \]  \hspace{1cm} (2.173)

\[ A_{4\text{Leith}} = L^6 \sqrt{\left(\frac{\text{viscC4Leith}}{\pi}\right)^{12} |\nabla^2 \omega^3|^2 + \left(\frac{\text{viscC4LeithD}}{\pi}\right)^{12} |\nabla^2 \nabla \cdot \mathbf{u}_h|^2} \]  \hspace{1cm} (2.174)

Thus, the biharmonic scaling suggested by Griffies and Hallberg (2000) \[GH00\] implies:

\[ |D| \propto L |\nabla^2 \mathbf{u}_h| \]
\[ |\nabla \omega^3| \propto L |\nabla^2 \omega^3| \]

It is not at all clear that these assumptions ought to hold. Only the Griffies and Hallberg (2000) \[GH00\] forms are currently implemented in MITgcm.

### 2.19.1.9 Selection of Length Scale

Above, the length scale of the grid has been denoted \( L \). However, in strongly anisotropic grids, \( L_x \) and \( L_y \) will be quite different in some locations. In that case, the CFL condition suggests that the minimum of \( L_x \) and \( L_y \) be used. On the other hand, other viscosities which involve whether a particular wavelength is ‘resolved’ might be better suited to use the maximum of \( L_x \) and \( L_y \). Currently, MITgcm uses useAreaViscLength to select between two options. If false, the square root of the harmonic mean of \( L_x^2 \) and \( L_y^2 \) is used for all viscosities, which is closer to the minimum and occurs naturally in the CFL constraint. If useAreaViscLength is true, then the square root of the area of the grid cell is used.

### 2.19.2 Mercator, Nondimensional Equations

The rotating, incompressible, Boussinesq equations of motion (Gill, 1982) \[Gil82\] on a sphere can be written in Mercator projection about a latitude \( \theta_0 \) and geopotential height \( z = r - r_0 \). The nondimensional form of these equations is:

\[ Ro \frac{D\tilde{\mathbf{u}}}{Dt} + \frac{\tilde{u} \sin \theta}{\sin \theta_0} + M_{Ro} \frac{\partial \pi}{\partial x} + \frac{\lambda Fr^2 M_{Ro} \cos \theta}{\mu \sin \theta_0} \frac{w}{r/H} = -\frac{Fr^2 M_{Ro} \tilde{u}w}{r/H} + \frac{Ro\lambda \cdot \nabla^2 \mathbf{u}}{Re} \]  \hspace{1cm} (2.175)

\[ Ro \frac{D\tilde{v}}{Dt} + \frac{\tilde{v} \sin \theta}{\sin \theta_0} + M_{Ro} \frac{\partial \pi}{\partial y} = -\frac{\mu Ro \tan \theta (\tilde{u}^2 + \tilde{v}^2)}{r/L} - \frac{Fr^2 M_{Ro} \tilde{vw}}{r/H} + \frac{Ro\lambda \cdot \nabla^2 \mathbf{u}}{Re} \]  \hspace{1cm} (2.176)

\[ Fr^2 \lambda^2 \frac{Dw}{Dt} - b + \frac{\partial \pi}{\partial z} - \frac{\lambda \cot \theta_0 \tilde{u}}{M_{Ro}} = \frac{\lambda \mu^2 (\tilde{u}^2 + \tilde{v}^2)}{M_{Ro} (r/L)} + \frac{Fr^2 \lambda^2 \hat{z} \cdot \nabla^2 \mathbf{u}}{Re} \]  \hspace{1cm} (2.177)

\[ \frac{Db}{Dt} + w = \frac{\nabla^2 b}{Pr Re} \]
\[ \mu^2 \left( \frac{\partial \tilde{u}}{\partial x} + \frac{\partial \tilde{v}}{\partial y} \right) + \frac{\partial w}{\partial z} = 0 \]  \hspace{1cm} (2.178)

Where

\[ \mu \equiv \frac{\cos \theta_0}{\cos \theta}, \quad \tilde{u} = \frac{u^*}{V \mu}, \quad \tilde{v} = \frac{v^*}{V \mu} \]
\[ f_0 \equiv 2\Omega \sin \theta_0, \quad \frac{D}{Dt} \equiv \mu^2 \left( \frac{\partial \bar{u}}{\partial x} + \frac{\partial \bar{v}}{\partial y} \right) + \frac{Fr^2 M Ro}{Ro} w \frac{\partial}{\partial z} \]

\[ x \equiv \frac{r}{L} \phi \cos \theta_0, \quad y \equiv \frac{r}{L} \int_{\theta_0}^{\theta} \frac{\cos \theta_0 \, d\theta'}{\cos \theta'}, \quad z \equiv \frac{r - r_0}{L} \]

\[ t^* = \frac{t}{V}, \quad b^* = b \frac{f_0 M Ro}{\lambda} \]

\[ \pi^* = \pi V f_0 L M Ro, \quad w^* = w V \frac{Fr^2 \lambda M Ro}{Ro} \]

\[ Ro \equiv \frac{V}{f_0 L}, \quad M Ro \equiv \max[1, Ro] \]

\[ Fr \equiv \frac{V}{N \lambda L}, \quad Re \equiv \frac{V L}{\nu}, \quad Pr \equiv \frac{\nu}{\kappa} \]

Dimensional variables are denoted by an asterisk where necessary. If we filter over a grid scale typical for ocean models:

1m < L < 100km
0.0001 < \lambda < 1
0.001 m/s < V < 1 m/s
\( f_0 < 0.0001 \) s\(^{-1}\)
0.01 s\(^{-1}\) < N < 0.0001 s\(^{-1}\)

these equations are very well approximated by

\[ \frac{Ro}{t} \frac{D \bar{u}}{Dt} - \frac{\bar{v} \sin \theta_0}{\sin \theta_0} + M Ro \frac{\partial \pi}{\partial x} = -\frac{\lambda Fr^2 M Ro \cos \theta}{\mu \sin \theta_0} w + \frac{Ro \nabla^2 \bar{u}}{Re} \] (2.179)

\[ \frac{Ro}{t} \frac{D \bar{v}}{Dt} + \frac{\bar{u} \sin \theta_0}{\sin \theta_0} + M Ro \frac{\partial \pi}{\partial y} = \frac{Ro \nabla^2 \bar{v}}{Re} \] (2.180)

\[ Fr^2 \lambda^2 \frac{D w}{Dt} - b + \frac{\partial \pi}{\partial z} = \frac{\lambda \cot \theta_0 \bar{u}}{M Ro} + \frac{Fr^2 \lambda^2 \nabla^2 w}{Re} \] (2.181)

\[ \frac{Db}{Dt} + w = \frac{\nabla^2 b}{Pr Re} \] (2.182)

\[ \mu^2 \left( \frac{\partial \bar{u}}{\partial x} + \frac{\partial \bar{v}}{\partial y} \right) + \frac{\partial w}{\partial z} = 0 \] (2.183)

\[ \nabla^2 \approx \left( \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} + \frac{\partial^2}{\partial z^2} \right) \]

Neglecting the non-frictional terms on the right-hand side is usually called the ‘traditional’ approximation. It is appropriate, with either large aspect ratio or far from the tropics. This approximation is used here, as it does not affect the form of the eddy stresses which is the main topic. The frictional terms are preserved in this approximate form for later comparison with eddy stresses.
CHAPTER
THREE

GETTING STARTED WITH MITGCM

This chapter is divided into two main parts. The first part, which is covered in sections Section 3.1 through Section 3.6, contains information about how to download, build and run MITgcm. We believe the best way to familiarize yourself with the model is to run one of the tutorial examples provided in the MITgcm repository (see Section 4), so would suggest newer MITgcm users jump there following a read-through of the first part of this chapter. Information is also provided in this chapter on how to customize the code when you are ready to try implementing the configuration you have in mind, in the second part (Section 3.8). The code and algorithm are described more fully in Section 2 and Section 6 and chapters thereafter.

In this chapter and others (e.g., chapter Contributing to the MITgcm), for arguments where the user is expected to replace the text with a user-chosen name, userid, etc., our convention is to show these as upper-case text surrounded by « », such as «USER_MUST_REPLACE_TEXT_HERE». The « and » characters are NOT typed when the text is replaced.

3.1 Where to find information

There is a web-archived support mailing list for the model that you can email at MITgcm-support@mitgcm.org once you have subscribed.

To sign up (subscribe) for the mailing list (highly recommended), click here

To browse through the support archive, click here

3.2 Obtaining the code

The MITgcm code and documentation are under continuous development and we generally recommend that one downloads the latest version of the code. You will need to decide if you want to work in a “git-aware” environment (Method 1) or with a one-time “stagnant” download (Method 2). We generally recommend method 1, as it is more flexible and allows your version of the code to be regularly updated as MITgcm developers check in bug fixes and new features. However, this typically requires at minimum a rudimentary understanding of git in order to make it worth one’s while.

Periodically we release an official checkpoint (or “tag”). We recommend one download the latest code, unless there are reasons for obtaining a specific checkpoint (e.g. duplicating older results, collaborating with someone using an older release, etc.)
3.2.1 Method 1

This section describes how to download git-aware copies of the repository. In a terminal window, cd to the directory where you want your code to reside. Type:

```
% git clone https://github.com/MITgcm/MITgcm.git
```

This will download the latest available code. If you now want to revert this code to a specific checkpoint release, first cd into the MITgcm directory you just downloaded, then type `git checkout checkpoint«XXX»` where «XXX» is the checkpoint version.

Alternatively, if you prefer to use ssh keys (say for example, you have a firewall which won’t allow a https download), type:

```
% git clone git@github.com:MITgcm/MITgcm.git
```

You will need a GitHub account for this, and will have to generate a ssh key though your GitHub account user settings. The fully git-aware download is over several hundred MB, which is considerable if one has limited internet download speed. In comparison, the one-time download zip file (Method 2, below) is order 100 MB. However, one can obtain a truncated, yet still git-aware copy of the current code by adding the option `--depth=1` to the git clone command above; all files will be present, but it will not include the full git history. However, the repository can be updated going forward.

3.2.2 Method 2

This section describes how to do a one-time download of MITgcm, NOT git-aware. In a terminal window, cd to the directory where you want your code to reside. To obtain the current code, type:

```
% wget https://github.com/MITgcm/MITgcm/archive/master.zip
```

For specific checkpoint release XXX, instead type:

```
% wget https://github.com/MITgcm/MITgcm/archive/checkpoint«XXX».zip
```

3.3 Updating the code

There are several different approaches one can use to obtain updates to MITgcm; which is best for you depends a bit on how you intend to use MITgcm and your knowledge of git (and/or willingness to learn). Below we outline three suggested update pathways:

1. **Fresh Download of MITgcm**

   This approach is the most simple, and virtually foolproof. Whether you downloaded the code from a static zip file (Method 2) or used the git clone command (Method 1), create a new directory and repeat this procedure to download a current copy of MITgcm. Say for example you are starting a new research project, this would be a great time to grab the most recent code repository and keep this new work entirely separate from any past simulations. This approach requires no understanding of git, and you are free to make changes to any files in the MIT repo tree (although we generally recommend that you avoid doing so, instead working in new subdirectories or on separate scratch disks as described here, for example).

2. **Using git pull to update the (unmodified) MITgcm repo tree**
If you have downloaded the code through a git clone command (Method 1 above), you can incorporate any changes to the source code (including any changes to any files in the MITgcm repository, new packages or analysis routines, etc.) that may have occurred since your original download. There is a simple command to bring all code in the repository to a ‘current release’ state. From the MITgcm top directory or any of its subdirectories, type:

```
% git pull
```

and all files will be updated to match the current state of the code repository, as it exists at GitHub. (Note: if you plan to contribute to MITgcm and followed the steps to download the code as described in Section 5, you will need to type `git pull upstream` instead.)

This update pathway is ideal if you are in the midst of a project and you want to incorporate new MITgcm features into your executable(s), or take advantage of recently added analysis utilities, etc. After the git pull, any changes in model source code and include files will be updated, so you can repeat the build procedure (Section 3.5) and you will include all these new features in your new executable.

Be forewarned, this will only work if you have not modified ANY of the files in the MITgcm repository (adding new files is ok; also, all verification run subdirectories `build` and `run` are also ignored by git). If you have modified files and the `git pull` fails with errors, there is no easy fix other than to learn something about git (continue reading...)

3. Fully embracing the power of git!

Git offers many tools to help organize and track changes in your work. For example, one might keep separate projects on different branches, and update the code separately (using `git pull`) on these separate branches. You can even make changes to code in the MIT repo tree; when git then tries to update code from upstream (see Figure 5.1), it will notify you about possible conflicts and even merge the code changes together if it can. You can also use `git commit` to help you track what you are modifying in your simulations over time. If you’re planning to submit a pull request to include your changes, you should read the contributing guide in Section 5, and we suggest you do this model development in a separate, fresh copy of the code. See Section 5.2 for more information and how to use git effectively to manage your workflow.

### 3.4 Model and directory structure

The “numerical” model is contained within a execution environment support wrapper. This wrapper is designed to provide a general framework for grid-point models; MITgcm is a specific numerical model that makes use of this framework (see Section 6.2 for additional detail). Under this structure, the model is split into execution environment support code and conventional numerical model code. The execution environment support code is held under the `eesupp` directory. The grid point model code is held under the `model` directory. Code execution actually starts in the `eesupp` routines and not in the `model` routines. For this reason the top-level main.F is in the `eesupp/src` directory. In general, end-users should not need to worry about the wrapper support code. The top-level routine for the numerical part of the code is in `model/src/the_model_main.F`. Here is a brief description of the directory structure of the model under the root tree.

- **model**: this directory contains the main source code. Also subdivided into two subdirectories: `model/inc` (includes files) and `model/src` (source code).
- **eesupp**: contains the execution environment source code. Also subdivided into two subdirectories: `eesupp/inc` and `eesupp/src`.
- **pkg**: contains the source code for the packages. Each package corresponds to a subdirectory. For example, `pkg/gmredi` contains the code related to the Gent-McWilliams/Redi scheme, `pkg/seaice` the code for a dynamic seaice model which can be coupled to the ocean model. The packages are described in detail in Section 8 and Section 9.
- **doc**: contains MITgcm documentation in reStructured Text (rst) format.
• tools: this directory contains various useful tools. For example, genmake2 is a script written in bash that should be used to generate your Makefile. The subdirectory tools/build_options contains ‘optfiles’ with the compiler options for many different compilers and machines that can run MITgcm (see Section 3.5.2.2). This directory also contains subdirectories tools/adjoint_options and tools/OAD_support that are used to generate the tangent linear and adjoint model (see details in Section 7).

• utils: this directory contains various utilities. The utils/matlab subdirectory contains matlab scripts for reading model output directly into matlab. The subdirectory utils/python contains similar routines for python. utils/scripts contains C-shell post-processing scripts for joining processor-based and tiled-based model output.

• verification: this directory contains the model examples. See Section 4.

• jobs: contains sample job scripts for running MITgcm.

• lsopt: Line search code used for optimization.

• optim: Interface between MITgcm and line search code.

3.5 Building the model

3.5.1 Quickstart Guide

To compile the code, we use the make program. This uses a file (Makefile) that allows us to pre-process source files, specify compiler and optimization options and also figures out any file dependencies. We supply a script (genmake2), described in section Section 3.5.2, that automatically generates the Makefile for you. You then need to build the dependencies and compile the code (Section 3.5.3).

As an example, assume that you want to build and run experiment verification/exp2. Let’s build the code in verification/exp2/build:

```
% cd verification/exp2/build
```

First, generate the Makefile:

```
% ../../../tools/genmake2 -mods ../code -optfile «/PATH/TO/OPTFILE»
```

The -mods command line option tells genmake2 to override model source code with any files in the subdirectory ../code (here, you need to configure the size of the model domain by overriding MITgcm’s default SIZE.h with an edited copy ../code/SIZE.h containing the specific domain size for exp2).

The -optfile command line option tells genmake2 to run the specified optfile, a bash shell script, during genmake2’s execution. An optfile typically contains definitions of environment variables, paths, compiler options, and anything else that needs to be set in order to compile on your local computer system or cluster with your specific Fortran compiler. As an example, we might replace «/PATH/TO/OPTFILE» with ../../../tools/build_options/linux_amd64_ifort11 for use with the Intel Fortran compiler (version 11 and above) on a linux x86_64 platform. This and many other optfiles for common systems and Fortran compilers are located in tools/build_options.

-mods, -optfile, and many additional genmake2 command line options are described more fully in Section 3.5.2.1. Detailed instructions on building with MPI are given in Section 3.5.4.

Once a Makefile has been generated, we create the dependencies with the command:

```
% make depend
```

It is important to note that the make depend stage will occasionally produce warnings or errors if the dependency parsing tool is unable to find all of the necessary header files (e.g., netcdf.inc, or worse, say it cannot find a Fortran compiler in your path). In some cases you may need to obtain help from your system administrator to locate these files.
Next, one can compile the code using:

```
% make
```

Assuming no errors occurred, the `make` command creates an executable called `mitgcmuv`.

Now you are ready to run the model. General instructions for doing so are given in section Section 3.6.

## 3.5.2 Generating a Makefile using genmake2

A shell script called `genmake2` for generating a `Makefile` is included as part of MITgcm. Typically `genmake2` is used in a sequence of steps as shown below:

```
% ../../../tools/genmake2 -mods ../code -optfile «/PATH/TO/OPTFILE»
% make depend
% make
```

The first step above creates a unix-style `Makefile`. The `Makefile` is used by `make` to specify how to compile the MITgcm source files (for more detailed descriptions of what the `make` tools are, and how they are used, see here).

This section describes details and capabilities of `genmake2`, located in the `tools` directory. `genmake2` is a shell script written to work in `bash` (and with all “sh”–compatible shells including Bourne shells). Like many unix tools, there is a help option that is invoked thru `genmake2 -h`. `genmake2` parses information from the following sources, in this order:

1. Command-line options (see Section 3.5.2.1)
2. A `genmake_local` file if one is found in the current directory. This is a `bash` shell script that is executed prior to the `optfile` (see step #3), used in some special model configurations and/or to set some options that can affect which lines of the `optfile` are executed. For example, this `genmake_local` file is required for a special setup, building a 'MITgcm coupler' executable; in a more typical setup, one will not require a `genmake_local` file.
3. An “options file” a.k.a. `optfile` (a `bash` shell script) specified by the command-line option `-optfile «/PATH/TO/OPTFILE»`, as mentioned briefly in Section 3.5.1 and described in detail in Section 3.5.2.2.
4. A `packages.conf` file (if one is found) with the specific list of packages to compile (see Section 8.1.1). The search path for file `packages.conf` is first the current directory, and then each of the `-mods` directories in the given order (as described here).

When you run the `genmake2` script, typical output might be as follows:

```
% ../../tools/genmake2 -mods ./code -optfile ../../../tools/build_options/linux_amd64_gfortran
GENMAKE :

A program for GENerating MAKEfiles for the MITgcm project.

For a quick list of options, use "genmake2 -h"
or for more detail see the documentation, section "Building the model"
(under "Getting Started") at: https://mitgcm.readthedocs.io/

=== Processing options files and arguments ===
getting local config information: none found
Warning: ROOTDIR was not specified ; try using a local copy of MITgcm found at "../..
getting OPTFILE information:
  using OPTFILE="../../../tools/build_options/linux_amd64_gfortran"
getting AD_OPTFILE information:
```

(continues on next page)
using AD_OPTFILE="tools/adjoint_options/adjoint_default"

check Fortran Compiler... pass (set FC_CHECK=5/5)
check makedepend (local: 0, system: 1, 1)

=== Checking system libraries ===
Do we have the system() command using gfortran... yes
Do we have the fdate() command using gfortran... yes
Do we have the etime() command using gfortran... c,r: yes (SbR)
Can we call simple C routines (here, "cloc()") using gfortran... yes
Can we unlimit the stack size using gfortran... yes
Can we register a signal handler using gfortran... yes
Can we use stat() through C calls... yes
Can we create NetCDF-enabled binaries... yes
    skip check for LAPACK Libs
Can we call FLUSH intrinsic subroutine... yes

=== Setting defaults ===
Adding MODS directories: ../code
Making source files in eesupp from templates
Making source files in pkg/exch2 from templates
Making source files in pkg/regrid from templates

=== Determining package settings ===
getting package dependency info from ../../../pkg/pkg_depend
getting package groups info from ../../../pkg/pkg_groups
checking list of packages to compile:
    using PKG_LIST="../code/packages.conf"
    before group expansion packages are: oceanic -kpp -gmredi cd_code
    replacing "oceanic" with: gfd gmredi kpp
    replacing "gfd" with: mom_common mom_fluxform mom_vecinv generic_advdiff debug
    mdsio rw monitor
    after group expansion packages are: mom_common mom_fluxform mom_vecinv generic_advdiff debug mdsio rw monitor gmredi kpp -kpp -gmredi cd_code
applying DISABLE settings
applying ENABLE settings
packages are: cd_code debug generic_advdiff mdsio mom_common mom_fluxform mom_vecinv monitor rw
applying package dependency rules
packages are: cd_code debug generic_advdiff mdsio mom_common mom_fluxform mom_vecinv monitor rw
Adding STANDARDDIRS='eesupp model'
Searching for *OPTIONS.h files in order to warn about the presence
of "#define "-type statements that are no longer allowed:
    found CPP_EEOPTIONS="tools/eesupp/inc/CPP_EEOPTIONS.h"
    found CPP_OPTIONS="tools/model/inc/CPP_OPTIONS.h"

Creating the list of files for the adjoint compiler.

=== Creating the Makefile ===
setting INCLUDES
Determining the list of source and include files
Writing makefile: Makefile
Add the source list for AD code generation
Making list of "exceptions" that need ".p" files
Making list of NOOPTFILES
Add rules for links
Adding makedepend marker

=== Done ===
original 'Makefile' generated successfully
=> next steps:
  > make depend
  > make (<<-- to generate executable)

In the above, notice:

• we did not specify ROOTDIR, i.e., a path to your MITgcm repository, but here we are building code from within the repository (specifically, in one of the verification subdirectory experiments). As such, genmake2 was smart enough to locate all necessary files on its own. To specify a remote ROOTDIR, see here.

• we specified the optfile linux_amd64_gfortran based on the computer system and Fortran compiler we used (here, a linux 64-bit machine with gfortran installed).

• genmake2 did some simple checking on availability of certain system libraries; all were found (except LAPACK, which was not checked since it is not needed here). NetCDF only requires a ‘yes’ if you want to write netCDF output; more specifically, a ‘no’ response to “Can we create NetCDF-enabled binaries” will disable including pkg/mnc and switch to output plain binary files. While the makefile can still be built with other ‘no’ responses, sometimes this will foretell errors during the make depend or make commands.

• any .F or .h files in the -mods directory ../code will also be compiled, overriding any MITgcm repository versions of files, if they exist.

• a handful of packages are being used in this build; see Section 8.1.1 for more detail about how to enable and disable packages.

• genmake2 terminated without error (note output at end after === Done ===), generating Makefile and a log file genmake.log. As mentioned, this does not guarantee that your setup will compile properly, but if there are errors during make depend or make, these error messages and/or the standard output from genmake2 or genmake. log may provide clues as to the problem. If instead genmake2 finishes with a warning message Warning: FORTRAN compiler test failed, this means that genmake2 is unable to locate the Fortran compiler or pass a trivial “hello world” Fortran compilation test. In this case, you should see genmake. log for errors and/or seek assistance from your system administrator; these tests need to pass in order to proceed to the make steps.

### 3.5.2.1 Command-line options:

genmake2 supports a number of helpful command-line options. A complete list of these options can be obtained by:

```
% genmake2 -h
```

The most important command-line options are:

- **-optfile */PATH/TO/OPTFILE** (or shorter: -of ) specifies the optfile that should be used for a particular build.

  If no optfile is specified through the command line, genmake2 will try to make a reasonable guess from the list provided in tools/build_options. The method used for making this guess is to first determine the combination of operating system and hardware and then find a working Fortran compiler within the user’s path. When these three items have been identified, genmake2 will try to find an optfile that has a matching name. See Section 3.5.2.2.
mods '«DIR1 DIR2 DIR3 ...»' specifies a list of directories containing "modifications". These directories contain files with names that may (or may not) exist in the main MITgcm source tree but will be overridden by any identically-named sources within the mods directories. Note the quotes around the list of directories, necessary given multiple arguments.

The order of precedence for versions of files with identical names:

- "mods" directories in the order given (e.g., will use copy of file located in DIR1 instead of DIR2)
- Packages either explicitly specified or included by default
- Packages included due to package dependencies
- The "standard dirs" (which may have been specified by the -standarddirs option below)

-rootdir «/PATH/TO/MITGCMDIR» specify the location of the MITgcm repository top directory (ROOTDIR). By default, genmake2 will try to find this location by looking in parent directories from where genmake2 is executed (up to 5 directory levels above the current directory).

In the quickstart example above (Section 3.5.1) we built the executable in the build directory of the experiment. Below, we show how to configure and compile the code on a scratch disk, without having to copy the entire source tree. The only requirement is that you have genmake2 in your $PATH, or you know the absolute path to genmake2. In general, one can compile the code in any given directory by following this procedure. Assuming the model source is in ~/MITgcm, then the following commands will build the model in /scratch/exp2-run1:

```
% cd /scratch/exp2-run1
% ~/MITgcm/tools/genmake2 -rootdir ~/MITgcm -mods ~/MITgcm/verification/exp2/code
% make depend
% make
```

As an alternative to specifying the MITgcm repository location through the -rootdir command-line option, genmake2 recognizes the environment variable $MITGCM_ROOTDIR.

-standarddirs «/PATH/TO/STANDARDDIR» specify a path to the standard MITgcm directories for source and includes files. By default, model and eesupp directories (src and inc) are the "standard dirs". This command can be used to reset these default standard directories, or instead NOT include either model or eesupp as done in some specialized configurations.

-oad generates a makefile for an OpenAD build (see Section 7.5)

-adoptfile «/PATH/TO.FILE» (or shorter: -adof) specifies the "adjoint" or automatic differentiation options file to be used. The file is analogous to the optfile defined above but it specifies information for the AD build process. See Section 7.2.3.4.

The default file is located in tools/adjoint_options/adjoint_default and it defines the “TAF” and “TAMC” compiler options.

-mpi enables certain MPI features (using CPP #define) within the code and is necessary for MPI builds (see Section 3.5.4).

-omp enables OpenMP code and compiler flag OMPFLAG (see Section 3.5.5).

-ieee use IEEE numerics (requires support in optfile). This option is typically a good choice if one wants to compare output from different machines running the same code. Note using IEEE disables all compiler optimizations.

-devel use IEEE numerics (requires support in optfile) and add additional compiler options to check array bounds and add other additional warning and debugging flags.

-make «/PATH/TO/GMAKE» due to the poor handling of soft-links and other bugs common with the make versions provided by commercial unix vendors, GNU make (sometimes called gmake) may be preferred. This option provides a means for specifying the make executable to be used.
While it is possible to use `genmake2` command-line options to set the Fortran or C compiler name (-fc and -cc respectively), we generally recommend setting these through an optfile, as discussed in Section 3.5.2.2. Other `genmake2` options are available to enable performance/timing analyses, etc.; see `genmake2 -h` for more info.

### 3.5.2.2 Optfiles in tools/build_options directory:

The purpose of the optfiles is to provide all the compilation options for particular “platforms” (where “platform” roughly means the combination of the hardware and the compiler) and code configurations. Given the combinations of possible compilers and library dependencies (e.g., MPI or netCDF) there may be numerous optfiles available for a single machine. The naming scheme for the majority of the optfiles shipped with the code is `OS_HARDWARE_COMPILER` where

- **OS** is the name of the operating system (generally the lower-case output of a `linux` `uname` command)
- **HARDWARE** is a string that describes the CPU type and corresponds to output from a `uname -m` command. Some common CPU types:
  - `amd64` use this code for x86_64 systems (most common, including AMD and Intel 64-bit CPUs)
  - `ia64` is for Intel IA64 systems (eg. Itanium, Itanium2)
  - `ppc` is for (old) Mac PowerPC systems
- **COMPILER** is the compiler name (generally, the name of the Fortran compiler executable). MITgcm is primarily written in FORTRAN 77. Compiling the code requires a FORTRAN 77 compiler. Any more recent compiler which is backwards compatible with FORTRAN 77 can also be used; for example, the model will build successfully with a Fortran 90 or Fortran 95 compiler. A C99 compatible compiler is also need, together with a C preprocessor. Some optional packages make use of Fortran 90 constructs (either free-form formatting, or dynamic memory allocation); as such, setups which use these packages require a Fortran 90 or later compiler build.

There are existing optfiles that work with many common hardware/compiler configurations; we first suggest you peruse the list in `tools/build_options` and try to find your platform/compiler configuration. These are the most common:

- `linux_amd64_gfortran`
- `linux_amd64_ifort11`
- `linux_amd64_ifort+impi`
- `linux_amd64_pgf77`

The above optfiles are all for linux x86_64 (64-bit) systems, utilized in many large high-performance computing centers. All of the above will work with single-threaded, MPI, or shared memory (OpenMP) code configurations. `gfortran` is GNU Fortran, `ifort` is Intel Fortran, `pgf77` is PGI Fortran (formerly known as “The Portland Group”). Note in the above list there are two `ifort` optfiles: `linux_amd64_ifort+impi` is for a specific case of using `ifort` with the Intel MPI library (a.k.a. `impi`), which requires special define statements in the optfile (in contrast with Open MPI or MVAPICH2 libraries; see Section 3.5.4). Note that both `ifort` optfiles require `ifort` version 11 or higher. Many clusters nowadays use `environment modules`, which allows one to easily choose which compiler to use through `module load <MODULENAME>`, automatically configuring your environment for a specific compiler choice (type `echo $PATH` to see where `genmake2` will look for compilers and system software).

In most cases, your platform configuration will be included in the available optfiles list and will result in a usable `Makefile` being generated. If you are unsure which optfile is correct for your configuration, you can try not specifying an optfile; on some systems the `genmake2` program will be able to automatically recognize the hardware, find a compiler and other tools within the user’s path, and then make a best guess as to an appropriate optfile from the list in the `tools/build_options` directory. However, for some platforms and code configurations, new optfiles must be written. To create a new optfile, it is generally best to start with one of the defaults and modify it to suit your needs. Like `genmake2`, the optfiles are all written in bash (or using a simple `sh`–compatible syntax). While nearly all environment variables used within `genmake2` may be specified in the optfiles, the critical ones that should be defined are:
FC  the Fortran compiler (executable) to use on .F files, e.g., ifort or gfortran, or if using MPI, the mpi-wrapper equivalent, e.g., mpif77

F90C  the Fortran compiler to use on .F90 files (only necessary if your setup includes a package which contains .F90 source code)

CC  similarly, the C compiler to use, e.g., icc or gcc, or if using MPI, the mpi-wrapper equivalent, e.g., mpicc

DEFINES  command-line options passed to the compiler

CPP  the C preprocessor to use, and any necessary command-line options, e.g. cpp -traditional -P

CFLAGS, FFLAGS  command-line compiler flags required for your C and Fortran compilers, respectively, to compile and execute properly. See your C and Fortran compiler documentation for specific options and syntax.

FOPTIM  command-line optimization Fortran compiler settings. See your Fortran compiler documentation for specific options and syntax.

NOOPTFLAGS  command-line settings for special files that should not be optimized using the FOPTIM flags

NOOPTFILES  list of source code files that should be compiled using NOOPTFLAGS settings

INCLUDES  path for additional files (e.g., netcdf.inc, mpi.f.h) to include in the compilation using the command-line `-I` option

INCLUDEDIRS  path for additional files to be included in the compilation

LIBS  path for additional library files that need to be linked to generate the final executable, e.g., libnetcdf.a

For example, an excerpt from an optfile which specifies several of these variables (here, for the linux-amd64 architecture using the PGI Fortran compiler) is as follows:

```
if test "x$MPI" = xtrue ; then
    CC=mpicc
    FC=mpif77
    F90C=mpif90
else
    CC=pgcc
    FC=pgf77
    F90C=pgf90
fi

DEFINES="-DWORDLENGTH=4"
if test "x$ALWAYS_USE_F90" = x1 ; then
    FC=$F90C
else
    DEFINES="$DEFINES -DNML_EXTENDED_F77"
fi

CPP='cpp -traditional -P'
F90FIXEDFORMAT=' -Mfixed'
EXTENDED_SRC_FLAG=' -Mextend'
GET_FC_VERSION=' -V'
OMPFLAG=' -mp'

NOOPTFLAGS=' -O0'
NOOPTFILES=''

FFLAGS="$FFLAGS -byteswapio -Ktrap=fp"
#- might want to use '-r8' for fizhi pkg:
```
The above list of environment variables typically specified in an optfile is by no means complete; additional variables may be required for your specific setup and/or your specific Fortran (or C) compiler.

If you write an optfile for an unrepresented machine or compiler, you are strongly encouraged to submit the optfile to the MITgcm project for inclusion. MITgcm developers are willing to provide help writing or modifying optfiles. Please submit the file through the GitHub issue tracker or email the MITgcm-support@mitgcm.org list.

Instructions on how to use optfiles to build MPI-enabled executables is presented in Section 3.5.4.

3.5.3 make commands

Following a successful build of Makefile, type make depend. This command modifies the Makefile by attaching a (usually, long) list of files upon which other files depend. The purpose of this is to reduce re-compilation if and when you start to modify the code. The make depend command also creates local links for all source files from the source directories (see “-mods” description in Section 3.5.2.1), so that all source files to be used are visible from the local build directory, either as hardcopy or as symbolic link.

IMPORTANT NOTE: Editing the source code files in the build directory will not edit a local copy (since these are just links) but will edit the original files in model/src (or model/inc) or in the specified -mods directory. While the latter might be what you intend, editing the master copy in model/src is usually NOT what is intended and may cause grief somewhere down the road. Rather, if you need to add to the list of modified source code files, place a copy of the file(s) to edit in the -mods directory, make the edits to these -mods directory files, go back to the build directory and type make Clean, and then re-build the makefile (these latter steps critical or the makefile will not link to this newly edited file).

The final make invokes the C preprocessor to produce the “little f” files (*.f and *.f90) and then compiles them to object code using the specified Fortran compiler and options. The C preprocessor step converts a number of CPP macros and #ifdef statements to actual Fortran and expands C-style #include statements to incorporate header files into the “little f” files. CPP style macros and #ifdef statements are used to support generating different compile code for different model configurations. The result of the build process is an executable with the name mitgcmuv.

Additional make “targets” are defined within the makefile to aid in the production of adjoint (Section 7.2.2) and other versions of MITgcm.

On computers with multiple processor cores, the build process can often be sped up appreciably using the command:

```bash
% make -j 2
```

where the “2” can be replaced with a number that corresponds to the number of cores (or discrete CPUs) available.

In addition, there are several housekeeping make clean options that might be useful:

- make clean removes files that make generates (e.g., *.o and *.f files)
• *make Clean* removes files and links generated by *make* and *make depend*; strongly recommended for “un-clean” directories which may contain the (perhaps partial) results of previous builds

• *make CLEAN* removes pretty much everything, including any executables and output from *genmake2*

### 3.5.4 Building with MPI

Building MITgcm to use MPI libraries can be complicated due to the variety of different MPI implementations available, their dependencies or interactions with different compilers, and their often ad-hoc locations within file systems. For these reasons, it's generally a good idea to start by finding and reading the documentation for your machine(s) and, if necessary, seeking help from your local systems administrator.

The steps for building MITgcm with MPI support are:

1. Make sure you have MPI libraries installed on your computer system or cluster. Different Fortran compilers (and different versions of a specific compiler) will generally require a custom version (of a MPI library) built specifically for it. On environment module-enabled clusters, one typically must first load a Fortran compiler, then specific MPI libraries for that compiler will become available to load. If libraries are not installed, MPI implementations and related tools are available including:
   - Open MPI
   - MVAPICH2
   - MPICH
   - Intel MPI

   Ask your systems administrator for assistance in installing these libraries.

2. Determine the location of your MPI library “wrapper” Fortran compiler, e.g., *mpif77* or *mpifort* etc. which will be used instead of the name of the fortran compiler (*gfortran*, *ifort*, *pgi77* etc.) to compile your code. Often the directory in which these wrappers are located will be automatically added to your $PATH environment variable when you perform a module load «SOME_MPI_MODULE»; thus, you will not need to do anything beyond the module load itself. If you are on a cluster that does not support environment modules, you may have to manually add this directory to your path, e.g., type `PATH=$PATH:«ADD_ADDITIONAL_PATH_TO_MPI_WRAPPER_HERE»` in a bash shell.

3. Determine the location of the includes file *mpif.h* and any other MPI-related includes files. Often these files will be located in a subdirectory off the main MPI library include/. In all optsfiles in tools/build_options, it is assumed environment variable $MPI_INC_DIR specifies this location; $MPI_INC_DIR should be set in your terminal session prior to generating a Makefile.

4. Determine how many processors (i.e., CPU cores) you will be using in your run, and modify your configuration’s SIZE.h (located in a “modified code” directory, as specified in your genmake2 command-line). In SIZE.h, you will need to set variables nPx*nPy to match the number of processors you will specify in your run script’s MITgcm execution statement (i.e., typically *mpirun* or some similar command, see Section 3.6.1). Note that MITgcm does not use dynamic memory allocation (a feature of Fortran 90, not FORTRAN 77), so all array sizes, and hence the number of processors to be used in your MPI run, must be specified at compile-time in addition to run-time. More information about the MITgcm WRAPPER, domain decomposition, and how to configure SIZE.h can be found in Section 6.3.

5. Build the code with the *genmake2 -mpi* option using commands such as:

```
% ../../../tools/genmake2 -mods=./code -mpi -of=«/PATH/TO/OPTFILE»
% make depend
% make
```
3.5.5 Building with OpenMP

Unlike MPI, which requires installation of additional software support libraries, using shared memory (OpenMP) for multi-threaded executable builds can be accomplished simply through the genmake2 command-line option `-omp`:

```
% ../../../tools/genmake2 -mods=../code -omp -of=«/PATH/TO/OPTFILE»
% make depend
% make
```

While the most common optfiles specified in Section 3.5.2.2 include support for the `-omp` option, some optfiles in `tools/build_options` do not include support for multi-threaded executable builds. Before using one of the less common optfiles, check whether `OMPFLAG` is defined.

Note that one does not need to specify the number of threads until runtime (see Section 3.6.2). However, the default maximum number of threads in MITgcm is set to a (low) value of 4, so if you plan on more you will need to change this value in `eesupp/inc/EEPARAMS.h` in your modified code directory.

3.6 Running the model

If compilation finished successfully (Section 3.5) then an executable called `mitgcmuv` will now exist in the local (build) directory.

To run the model as a single process (i.e., not in parallel) simply type (assuming you are still in the build directory):

```
% cd ../run
% ln -s ../input/* .
% cp ../build/mitgcmuv .
% ./mitgcmuv
```

Here, we are making a link to all the support data files (in `../input/`) needed by the MITgcm for this experiment, and then copying the executable from the the build directory. The `./` in the last step is a safe-guard to make sure you use the local executable in case you have others that might exist in your `PATH`. The above command will spew out many lines of text output to your screen. This output contains details such as parameter values as well as diagnostics such as mean kinetic energy, largest CFL number, etc. It is worth keeping this text output with the binary output so we normally re-direct the `stdout` stream as follows:

```
% ./mitgcmuv > output.txt
```

In the event that the model encounters an error and stops, it is very helpful to include the last few line of this `output.txt` file along with the (stderr) error message within any bug reports.

For the example experiment in `verification/exp2`, an example of the output is kept in `verification/exp2/results/output.txt` for comparison. You can compare your `output.txt` with the corresponding one for that experiment to check that your set-up indeed works. Congratulations!
3.6.1 Running with MPI

Run the code with the appropriate MPI “run” or “exec” program provided with your particular implementation of MPI. Typical MPI packages such as Open MPI will use something like:

```
% mpirun -np 4 ./mitgcmuv
```

Slightly more complicated scripts may be needed for many machines since execution of the code may be controlled by both the MPI library and a job scheduling and queueing system such as Slurm, PBS/TORQUE, LoadLeveler, or any of a number of similar tools. See your local cluster documentation or system administrator for the specific syntax required to run on your computing facility.

3.6.2 Running with OpenMP

Assuming the executable `mitgcmuv` was built with OpenMP (see Section 3.5.5), the syntax to run a multi-threaded simulation is the same as running single-threaded (see Section 3.6), except that the following additional steps are required beforehand:

1. Environment variables for the number of threads and the stacksize need to be set prior to executing the model. The exact names of these environment variables differ by Fortran compiler, but are typically some variant of `OMP_NUM_THREADS` and `OMP_STACKSIZE`, respectively. For the latter, in your run script we recommend adding the line `export OMP_STACKSIZE=400M` (or for a C shell-variant, `setenv OMP_STACKSIZE 400M`). If this stacksize setting is insufficient, MITgcm will crash, in which case a larger number can be used. Similarly, `OMP_NUM_THREADS` should be set to the exact number of threads you require.

2. In file `eedata` you will need to change namelist parameters `nTx` and `nTy` to reflect the number of threads in x and y, respectively (for a single-threaded run, `nTx = nTy = 1`). The value of `nTx * nTy` must equal the value of environment variable `OMP_NUM_THREADS` (or its name-equivalent for your Fortan compiler) or MITgcm will terminate during its initialization with an error message.

MITgcm will take the number of tiles used in the model (as specified in `SIZE.h`) and the number of threads (`nTx` and `nTy` from file `eedata`), and in running will spread the tiles out evenly across the threads. This is done independently for x and y. As such, the number of tiles in x (variable `nSx` as defined in `SIZE.h`) must divide evenly by the number of threads in x (namelist parameter `nTx`), and similarly for `nSy` and `nTy`, else MITgcm will terminate on initialization. More information about the MITgcm WRAPPER, domain decomposition, and how to configure `SIZE.h` can be found in Section 6.3.
3.6.3 Output files

The model produces various output files and, when using pkg/mnc (i.e., netCDF), sometimes even directories. Depending upon the I/O package(s) selected at compile time (either pkg/mdsio, pkg/mnc, or both as determined by packages.conf) and the run-time flags set (in data.pkg), the following output may appear. More complete information describing output files and model diagnostics is described in Section 9.

3.6.3.1 Raw binary output files

The “traditional” output files are generated by the pkg/mdsio (see Section 9.2). The pkg/mdsio model data are written according to a “meta/data” file format. Each variable is associated with two files with suffix names .data and .meta. The .data file contains the data written in binary form (big endian by default). The .meta file is a “header” file that contains information about the size and the structure of the .data file. This way of organizing the output is particularly useful when running multi-processors calculations.

At a minimum, the instantaneous “state” of the model is written out, which is made of the following files:

- **U.00000nIter** - zonal component of velocity field (m/s and positive eastward).
- **V.00000nIter** - meridional component of velocity field (m/s and positive northward).
- **W.00000nIter** - vertical component of velocity field (ocean: m/s and positive upward, atmosphere: Pa/s and positive towards increasing pressure i.e., downward).
- **T.00000nIter** - potential temperature (ocean: °C, atmosphere: K).
- **S.00000nIter** - ocean: salinity (g/kg), atmosphere: water vapor (g/kg).
- **Eta.00000nIter** - ocean: surface elevation (m), atmosphere: surface pressure anomaly (Pa).

The chain 00000nIter consists of ten figures that specify the iteration number at which the output is written out. For example, U.0000000300 is the zonal velocity at iteration 300.

In addition, a “pickup” or “checkpoint” file called:

- **pickup.00000nIter**

is written out. This file represents the state of the model in a condensed form and is used for restarting the integration (at the specific iteration number). Some additional parameterizations and packages also produce separate pickup files, e.g.,

- **pickup_cd.00000nIter** if the C-D scheme is used (see C_D Scheme)
- **pickup_seaice.00000nIter** if the seaice package is turned on (see SEAICE Package)
- **pickup_ptracers.00000nIter** if passive tracers are included in the simulation (see PTRACERS Package)

Rolling checkpoint files are the same as the pickup files but are named differently. Their name contain the chain ckptA or ckptB instead of 00000nIter. They can be used to restart the model but are overwritten every other time they are output to save disk space during long integrations.
3.6.3.2 NetCDF output files

pkg/mnc is a set of routines written to read, write, and append netCDF files. Unlike the pkg/mdsio output, the pkg/mnc–generated output is usually placed within a subdirectory with a name such as mnc_output_ (by default, netCDF tries to append, rather than overwrite, existing files, so a unique output directory is helpful for each separate run).

The pkg/mnc output files are all in the “self-describing” netCDF format and can thus be browsed and/or plotted using tools such as:

- ncdump is a utility which is typically included with every netCDF install, and converts the netCDF binaries into formatted ASCII text files.
- ncview is a very convenient and quick way to plot netCDF data and it runs on most platforms. Panoply is a similar alternative.
- MATLAB, GrADS, IDL and other common post-processing environments provide built-in netCDF interfaces.

3.6.4 Looking at the output

3.6.4.1 MATLAB

Raw binary output

The repository includes a few MATLAB utilities to read binary output files written in the /pkg/mdsio format. The MATLAB scripts are located in the directory utils/matlab under the root tree. The script utils/matlab/rdmds.m reads the data. Look at the comments inside the script to see how to use it.

Some examples of reading and visualizing some output in MATLAB:

```
% matlab
>> H=rdmds('Depth');
>> contourf(H);colorbar;
>> title('Depth of fluid as used by model');

>> eta=rdmds('Eta',10);
>> imagesc(eta);axis ij;colorbar;
>> title('Surface height at iter=10');

>> [eta,iters,M]=rdmds('Eta',NaN); % this will read all dumped iterations
>> % iter numbers put in variable 'iters'; 'M' is a character string w/metadat
>> for n=1:length(iters); imagesc(eta(:,:,n));axis ij;colorbar;pause(.5);end
```

Typing help rdmds in MATLAB will pull up further information on how to use the rdmds utility.
NetCDF output

Similar scripts for netCDF output (e.g., utils/matlab/rdmnc.m) are available and they are described in Section 9.3.

3.6.4.2 Python

Install the MITgcmutils python package following the instructions in Section 11.1.

Raw binary output

The following example shows how to load in some data:

```python
# python
from MITgcmutils import mds
Eta = mds.rdmds('Eta', itrs=10)
```

For more information about this function and its options, see the API docs, `MITgcmutils.mds.rdmds()`.

NetCDF output

The netCDF output is currently produced with one file per processor. This means the individual tiles need to be stitched together to create a single netCDF file that spans the model domain. The script `utils/python/MITgcmutils/scripts/gluemncbig` can do this efficiently from the command line. If you have installed the MITgcmutils package, a copy of `gluemncbig` should be on your path. For usage information, see Section 11.1.9.

The following example shows how to use the `xarray` python package to read the resulting netCDF file into Python:

```python
# python
import xarray as xr
Eta = xr.open_dataset('Eta.nc')
```

3.7 Customizing the Model Configuration - Code Parameters and Compilation Options

3.7.1 Model Array Dimensions

MITgcm’s array dimensions need to be configured for each unique model domain. The size of each tile (in dimensions \(x\), \(y\), and vertical coordinate \(r\)) the “overlap” region of each tile (in \(x\) and \(y\)), the number of tiles in the \(x\) and \(y\) dimensions, and the number of processes (using MPI) in the \(x\) and \(y\) dimensions all need to be specified in `SIZE.h`. From these parameters, global domain-size variables \(Nx\), \(Ny\) are computed by the model. See a more technical discussion of `SIZE.h` parameters in in Section 6.3.1, and a detailed explanation of an example `SIZE.h` setup in tutorial Baroclinic Ocean Gyre.
### Parameters Table

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sNx</td>
<td>30</td>
<td>number of points in ( x ) dimension in a single tile</td>
</tr>
<tr>
<td>sNy</td>
<td>15</td>
<td>number of points in ( y ) dimension in a single tile</td>
</tr>
<tr>
<td>Nr</td>
<td>4</td>
<td>number of points in ( r ) dimension</td>
</tr>
<tr>
<td>OLx</td>
<td>2</td>
<td>number of &quot;overlap&quot; points in ( x ) dimension for a tile</td>
</tr>
<tr>
<td>OLy</td>
<td>2</td>
<td>number of &quot;overlap&quot; points in ( y ) dimension for a tile</td>
</tr>
<tr>
<td>nSx</td>
<td>2</td>
<td>number of tile per process in ( x ) dimension</td>
</tr>
<tr>
<td>nSy</td>
<td>4</td>
<td>number of tile per process in ( y ) dimension</td>
</tr>
<tr>
<td>nPx</td>
<td>1</td>
<td>number of processes in ( x ) dimension</td>
</tr>
<tr>
<td>nPy</td>
<td>1</td>
<td>number of processes in ( y ) dimension</td>
</tr>
</tbody>
</table>

Note the repository version of \( SIZE.h \) includes several lines of text at the top that will halt compilation with errors. Thus, to use MITgcm you will need to copy \( SIZE.h \) to a code modification directory and make edits, including deleting or commenting out the offending lines of text.

### 3.7.2 C Preprocessor Options

The CPP flags relative to the “numerical model” part of the code are defined and set in the file \( CPP\_OPTIONS.h \) in the directory \( \text{model/inc/} \). In the parameter tables in Section 3.8 we have noted CPP options that need to be changed from the default to enable specific runtime parameter to be used properly. Also note many of the options below are for less-common situations or are somewhat obscure, so newer users of the MITgcm are encouraged to jump to Section 3.8 where more basic runtime parameters are discussed.

<table>
<thead>
<tr>
<th>CPP Flag Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SHORTWAVE_HEATING</td>
<td>#undef</td>
<td>provide separate shortwave heating file, allowing shortwave to penetrate below surface layer</td>
</tr>
<tr>
<td>ALLOW_GEOTHERMAL_FLUX</td>
<td>#undef</td>
<td>include code for applying geothermal heat flux at the bottom of the ocean</td>
</tr>
<tr>
<td>ALLOW_FRICTION_HEATING</td>
<td>#undef</td>
<td>include code to allow heating due to friction (and momentum dissipation)</td>
</tr>
<tr>
<td>ALLOW_ADDFLUID</td>
<td>#undef</td>
<td>allow mass source or sink of fluid in the interior (3D generalization of oceanic real-fresh water flux)</td>
</tr>
<tr>
<td>ATMOSPHERIC_LOADING</td>
<td>#define</td>
<td>include code for atmospheric pressure-loading (and seaice-loading) on ocean surface</td>
</tr>
<tr>
<td>ALLOW_BALANCE_FLUXES</td>
<td>#undef</td>
<td>include balancing surface forcing fluxes code</td>
</tr>
<tr>
<td>ALLOW_BALANCE_RELAX</td>
<td>#undef</td>
<td>include balancing surface forcing relaxation code</td>
</tr>
<tr>
<td>CHECK_SALINITY_FOR_NEGATIVE_VALUES</td>
<td>#undef</td>
<td>include code checking for negative salinity</td>
</tr>
<tr>
<td>EXCLUDE_FFIELDS_LOAD</td>
<td>#undef</td>
<td>exclude external forcing-fields load; code allows reading and simple linear time interpolation of oceanic forcing fields, if no specific pkg (e.g., pkg/exf) is used to compute them</td>
</tr>
<tr>
<td>INCLUDE_PHIHYD_CALCULATION_CODE</td>
<td>#define</td>
<td>include code to calculate ( \phi_{hyd} )</td>
</tr>
<tr>
<td>INCLUDE_CONVECT_CALL</td>
<td>#define</td>
<td>include code for convective adjustment mixing algorithm</td>
</tr>
<tr>
<td>INCLUDE_CALC_DIFFUSIVITY_CALL</td>
<td>#define</td>
<td>include codes that calculates (tracer) diffusivities and viscosities</td>
</tr>
<tr>
<td>ALLOW_3D_DIFFKR</td>
<td>#undef</td>
<td>allow full 3D specification of vertical diffusivity</td>
</tr>
<tr>
<td>ALLOW_BL79_LAT_VARY</td>
<td>#undef</td>
<td>allow latitudinally varying Bryan and Lewis 1979 [BL79] vertical diffusivity</td>
</tr>
</tbody>
</table>

continues on next page
<table>
<thead>
<tr>
<th>CPP Flag Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXCLUDE_PCELL_MIX_CODE</td>
<td>#undef</td>
<td>exclude code for partial-cell effect (physical or enhanced) in vertical mixing; this allows accounting for partial-cell in vertical viscosity and diffusion, either from grid-spacing reduction effect or as artificially enhanced mixing near surface &amp; bottom for too thin grid-cell</td>
</tr>
<tr>
<td>ALLOW_Smag_3D.Diffusivity</td>
<td>#undef</td>
<td>include code for isotropic 3-D Smagorinsky diffusivity for tracers (viscosity scaled by constant Prandtl number)</td>
</tr>
<tr>
<td>ALLOW_Solve4_PS_AND_Drag</td>
<td>#undef</td>
<td>include code for combined surface pressure and drag implicit solver</td>
</tr>
<tr>
<td>INCLUDE_ImplVertAdv_Code</td>
<td>#define</td>
<td>include code for implicit vertical advection</td>
</tr>
<tr>
<td>ALLOW_Adamsbashforth_3</td>
<td>#undef</td>
<td>include code for Adams-Bashforth 3rd-order</td>
</tr>
<tr>
<td>ALLOW_Qhyd_Stagger_TS</td>
<td>#undef</td>
<td>include code for quasi-hydrostatic stagger time-step Adams-Bashforth code</td>
</tr>
<tr>
<td>Exact_CONSERV</td>
<td>#define</td>
<td>include code for “exact conservation” of fluid in free-surface formulation (recompute divergence after pressure solver)</td>
</tr>
<tr>
<td>NonLin_FRSURF</td>
<td>#undef</td>
<td>allow the use of non-linear free-surface formulation; implies that grid-cell thickness (hFactors) varies with time</td>
</tr>
<tr>
<td>ALLOW_NONHYDROSTATIC</td>
<td>#undef</td>
<td>include non-hydrostatic and 3D pressure solver codes</td>
</tr>
<tr>
<td>ALLOW_EDDYPsi</td>
<td>#undef</td>
<td>include GM-like eddy stress in momentum code (untested, not recommended)</td>
</tr>
<tr>
<td>ALLOW_Cg2D_NSA</td>
<td>#undef</td>
<td>use non-self-adjoint (NSA) conjugate-gradient solver</td>
</tr>
<tr>
<td>ALLOW_SRCG</td>
<td>#define</td>
<td>include code for single reduction conjugate gradient solver</td>
</tr>
<tr>
<td>Solve_Diagonal_LOWMEMORY</td>
<td>#undef</td>
<td>low memory footprint (not suitable for AD) choice for implicit solver routines solve_*diagonal.F</td>
</tr>
<tr>
<td>Solve_Diagonal_Kinner</td>
<td>#undef</td>
<td>choice for implicit solver routines solve_*diagonal.F suitable for AD</td>
</tr>
<tr>
<td>CoSiNeMETH_III</td>
<td>#define</td>
<td>selects implementation form of $\cos \phi$ scaling of bi-harmonic term for viscosity (note, CPP option for tracer diffusivity set independently in GAD_OPTIONS.h)</td>
</tr>
<tr>
<td>Isotropic_Cos_Scaling</td>
<td>#undef</td>
<td>selects isotropic scaling of harmonic and bi-harmonic viscous terms when using the $\cos \phi$ scaling (note, CPP option for tracer diffusivity set independently in GAD_OPTIONS.h)</td>
</tr>
</tbody>
</table>

By default, MITgcm includes several core packages, i.e., these packages are enabled during genmake2 execution if a file packages.conf is not found. See Section 8.1.1 for more information about packages.conf, and see pkg/pkg_groups for more information about default packages and package groups. These default packages are as follows:

- pkg/mom_common
- pkg/mom_fluxform
- pkg/mom_vecinv
- pkg/generic_advdiff
Additional CPP options that affect the model core code are set in files `{PKG}_OPTIONS.h located in these packages’ directories. Similarly, optional (non-default) packages also include package-specific CPP options that must be set in files `{PKG}_OPTIONS.h.

### 3.7.3 Preprocessor Execution Environment Options

Most MITgcm users can skip this section; many of these flags were intended for very specific platform environments, and not meant to be changed for more general environments (an exception being if you are using a coupled setup, see below).

The file `CPP_EEOPTIONS.h` in the directory `eesupp/inc/` contains a number of CPP flags related to the execution environment where the model will run. Below we describe the subset of user-editable CPP flags:

<table>
<thead>
<tr>
<th>CPP Flag Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GLOBAL_SUM_ORDER_TILES</td>
<td>#define</td>
<td>always cumulate tile local-sum in the same order by applying MPI allreduce to array of tiles</td>
</tr>
<tr>
<td>CG2D_SINGLECPU_SUM</td>
<td>#undef</td>
<td>alternative way of doing global sum on a single CPU to eliminate tiling-dependent roundoff errors</td>
</tr>
<tr>
<td>SINGLE_DISK_IO</td>
<td>#undef</td>
<td>to write STDOUT, STDERR and scratch files from process 0 only</td>
</tr>
<tr>
<td>USE_FORTRAN_SCRATCH_FILES</td>
<td>#undef</td>
<td>flag to turn on old default of opening scratch files with the STATUS=’SCRATCH’ option</td>
</tr>
<tr>
<td>COMPONENT_MODULE</td>
<td>#undef</td>
<td>control use of communication with other components, i.e., sets component to work with a coupler interface</td>
</tr>
<tr>
<td>DISCONNECTED_TILES</td>
<td>#undef</td>
<td>use disconnected tiles (no exchange between tiles, just fill-in edges assuming locally periodic subdomain)</td>
</tr>
<tr>
<td>REAL4_IS_SLOW</td>
<td>#define</td>
<td>if undefined, force _RS variables to be declared as real*4</td>
</tr>
</tbody>
</table>

The default setting of `#define GLOBAL_SUM_ORDER_TILES` in `CPP_EEOPTIONS.h` provides a way to achieve numerically reproducible global sums for a given tile domain decomposition. As implemented however, this approach will increase the volume of network traffic in a way that scales with the total number of tiles. Profiling has shown that letting the code fall through to a baseline approach that simply uses `MPI_Allreduce()` can provide significantly improved performance for certain simulations\(^1\). The fall-through approach is activated by `#undef GLOBAL_SUM_ORDER_TILES`.

In order to get bit-wise reproducible results between different tile domain decompositions (e.g., single tile on single processor versus multiple tiles either on single or multiple processors), one can choose to `#define` option `CG2D_SINGLECPU_SUM` to use the MUCH slower `global_sum_singlecpu.F` for the key part of MITgcm algorithm.

---

\(^1\) One example is the llc_540 case located at `https://github.com/MITgcm-contrib/llc_hires/tree/master/llc_540`. This case was run on the Pleiades computer for 20 simulated days using 767 and 2919 MPI ranks. At 767 ranks, the fall-through approach provided a throughput of to 799.0 simulated days per calendar day (dd/d) while the default approach gave 781.0. The profiler showed the speedup was directly attributable to spending less time in MPI_Allreduce. The volume of memory traffic associated with MPI_Allreduce dropped by 3 orders (22.456T -> 32.596G). At 2819 MPI ranks the fall-through approach gave a throughput of 1300 dd/d while the default approach gave 800.0 dd/d. Put another way, this case did not scale at all from 767p to 2819p unless the fall-though approach was utilized. The profiler showed the speedup was directly attributable to spending less time in MPI_Allreduce. The volume of memory traffic associated with MPI_Allreduce dropped by 3 orders (303.70T ->121.08G ).
CG2D that relies on global sum. This option is not only much slower but also requires a large volume of communications so it is practically unusable for a large set-up; furthermore, it does not address reproducibility when global sum is used outside CG2D, e.g., in non-hydrostatic simulations.

In a default multi-processor configuration, each process opens and reads its own set of namelist files and open and writes its own standard output. This can be slow or even problematic for very large processor counts. Defining the CPP-flag SINGLE_DISK_IO suppresses this behavior and lets only the master process (process 0) read namelist files and writes a standard output stream. This may seem advantageous, because it reduces the amount of seemingly redundant output, but use this option with caution and only when absolutely confident that the setup is working since any message (error/warning/print) from any processor $\neq 0$ will be lost.

The way the namelist files are read requires temporary (scratch) files in the initialization phase. By default, the MITgcm does not use intrinsic Fortran scratch files (STATUS='scratch') because they can lead to conflicts in multi-processor simulations on some HPC-platforms, when the processors do not open scratch files with reserved names. However, the implemented default scheme for the scratch files can be slow for large processor counts. If this is a problem in a given configuration, defining the CPP-flag USE_FORTRAN_SCRATCH_FILES may help by making the code use the intrinsic Fortran scratch files.

The CPP-flag COMPONENT_MODULE needs to be set to #define only for builds in which the MITgcm executable (for either an oceanic or atmospheric simulation) is configured to communicate with a coupler. This coupler can be a specially configured build of MITgcm itself; see, for example, verification experiment cpl_aim+ocn.

The CPP-flag DISCONNECTED_TILES should not be #define unless one wants to run simultaneously several small, single-tile ensemble members from a single process, as each tile will be disconnected from the others and considered locally as a doubly periodic patch.

MITgcm _RS variables are forced to be declared as real*4 if CPP-flag REAL4_IS_SLOW to is set to #undef in CPP_EEOPTIONS.h (_RS is a macro used in declaring real variables that, in principle, do not require double precision). However, this option is not recommended except for computational benchmarking or for testing the trade-off between memory footprint and model precision. And even for these specialized tests, there is no need to edit CPP_EEOPTIONS.h since this feature can be activated using the genmake2 command line option -use_r4, as done in some regression tests (see testing results page tests with optfile suffix .use_r4).

3.8 Customizing the Model Configuration - Runtime Parameters

When you are ready to run the model in the configuration you want, the most straightforward approach is to use and adapt the setup of a tutorial or verification experiment (described in Section 4) that is the closest to your configuration. Then, the amount of setup will be minimized. In this section, we document the complete list of MITgcm model namelist runtime parameters set in file data, which needs to be located in the directory where you will run the model. Model parameters are defined and declared in the file PARAMS.h and their default values are generally set in the routine set_defaults.F, otherwise when initialized in the routine ini_parms.F. Section 3.8.9 documents the “execution environment” namelist parameters in file edata, which must also reside in the current run directory. Note that runtime parameters used by (non-default) MITgcm packages are not documented here but rather in Section 8 and Section 9, and prescribed in package-specific data.${pkg} namelist files which are read in via package-specific ${pkg}_readparms.F where ${pkg} is the package name (see Section 8.1.1).

In what follows, model parameters are grouped into categories related to configuration/computational domain, algorithmic parameters, equations solved in the model, parameters related to model forcing, and simulation controls. The tables below specify the namelist parameter name, the namelist parameter group in data (and eedata in Section 3.8.9), the default value, and a short description of its function. Runtime parameters that require non-default CPP options to be set prior to compilation (see Section 3.7) for proper use are noted.
3.8.1 Parameters: Configuration, Computational Domain, Geometry, and Time-Discretization

3.8.1.1 Model Configuration

`buoyancyRelation` is set to `OCEANIC` by default, which employs a \( z \)-coordinate vertical axis. To simulate an ocean using pressure coordinates in the vertical, set it to `OCEANICP`. For atmospheric simulations, `buoyancyRelation` needs to be set to `ATMOSPHERIC`, which also uses pressure as the vertical coordinate. The default model configuration is hydrostatic; to run a non-hydrostatic simulation, set the logical variable `nonHydrostatic` to `TRUE`.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Group</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>buoyancyRelation</code></td>
<td>PARM01</td>
<td>OCEANIC</td>
<td>buoyancy relation (<code>OCEANIC, OCEANICP, or ATMOSPHERIC</code>)</td>
</tr>
<tr>
<td><code>quasiHydrostatic</code></td>
<td>PARM01</td>
<td>FALSE</td>
<td>quasi-hydrostatic formulation on/off flag</td>
</tr>
<tr>
<td><code>rhoRefFile</code></td>
<td>PARM01</td>
<td>' '</td>
<td>filename for reference density profile (kg/m(^3)); activates anelastic form of model</td>
</tr>
<tr>
<td><code>nonHydrostatic</code></td>
<td>PARM01</td>
<td>FALSE</td>
<td>non-hydrostatic formulation on/off flag; requires <code>#define ALLOW_NONHYDROSTATIC</code></td>
</tr>
</tbody>
</table>

3.8.1.2 Grid

Four different grids are available: Cartesian, spherical polar, cylindrical, and curvilinear (which includes the cubed sphere). The grid is set through the logical variables `usingCartesianGrid`, `usingSphericalPolarGrid`, `usingCylindricalGrid`, and `usingCurvilinearGrid`. Note that the cylindrical grid is designed for modeling a rotating tank, so that \( x \) is the azimuthal direction, \( y \) is the radial direction, and \( r \) is vertical coordinate (see tutorial `rotating tank`).

The variable `xgOrigin` sets the position of the western most gridcell face in the \( x \) dimension (Cartesian, meters; spherical and cylindrical, degrees). For a Cartesian or spherical grid, the southern boundary is defined through the variable `ygOrigin` which corresponds to the latitude of the southern most gridcell face (Cartesian, meters; spherical, degrees). For a cylindrical grid, a positive `ygOrigin` (m) adds an inner cylindrical boundary at the center of the tank. The resolution along the \( x \) and \( y \) directions is controlled by the 1-D arrays `delX` (meters for a Cartesian grid, degrees otherwise) and `delY` (meters for Cartesian and cylindrical grids, degrees spherical). On a spherical polar grid, you might decide to set the variable `cosPower` which is set to 0 by default and which represents \( n \) in \((\cos \phi)^n\), the power of cosine of latitude to multiply horizontal viscosity and tracer diffusivity. The vertical grid spacing is set through the 1-D array `delR` (\( z \)-coordinates: in meters; \( p \)-coordinates, in Pa). Using a curvilinear grid requires complete specification of all horizontal MITgcm grid variables, either through a default filename (link to new doc section) or as specified by `horizGridFile`.

The variable `seaLev_Z` represents the standard position of sea level, in meters. This is typically set to 0 m for the ocean (default value). If instead pressure is used as the vertical coordinate, the pressure at the top (of the atmosphere or ocean) is set through `top_Pres`, typically 0 Pa. As such, these variables are analogous to `xgOrigin` and `ygOrigin` to define the vertical grid axis. But they also are used for a second purpose: in a \( z \)-coordinate setup, `top_Pres` sets a reference top pressure (required in a non-linear equation of state computation, for example); note that 1 bar (i.e., typical Earth atmospheric sea-level pressure) is added already, so the default is 0 Pa. Similarly, for a \( p \)-coordinate setup, `seaLev_Z` is used to set a reference geopotential (after gravity scaling) at the top of the ocean or bottom of the atmosphere.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Group</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>usingCartesianGrid</code></td>
<td>PARM04</td>
<td>TRUE</td>
<td>use Cartesian grid/coordinates on/off flag</td>
</tr>
<tr>
<td><code>usingSphericalPolarGrid</code></td>
<td>PARM04</td>
<td>FALSE</td>
<td>use spherical grid/coordinates on/off flag</td>
</tr>
<tr>
<td><code>usingCylindricalGrid</code></td>
<td>PARM04</td>
<td>FALSE</td>
<td>use cylindrical grid/coordinates on/off flag</td>
</tr>
<tr>
<td><code>usingCurvilinearGrid</code></td>
<td>PARM04</td>
<td>FALSE</td>
<td>use curvilinear grid/coordinates on/off flag</td>
</tr>
<tr>
<td><code>xgOrigin</code></td>
<td>PARM04</td>
<td>0.0</td>
<td>west edge ( x )-axis origin (Cartesian: m; spherical and cylindrical: degrees longitude)</td>
</tr>
</tbody>
</table>
Table 3.3 – continued from previous page

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Group</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ygOrigin</td>
<td>PARM04</td>
<td>0.0</td>
<td>South edge $y$-axis origin (Cartesian and cylindrical: m; spherical: degrees latitude)</td>
</tr>
<tr>
<td>dxSpacing</td>
<td>PARM04</td>
<td>unset</td>
<td>$x$-axis uniform grid spacing, separation between cell faces (Cartesian: m; spherical and cylindrical: degrees)</td>
</tr>
<tr>
<td>delX</td>
<td>PARM04</td>
<td>dxSpacing</td>
<td>1D array of $x$-axis grid spacing, separation between cell faces (Cartesian: m; spherical and cylindrical: degrees)</td>
</tr>
<tr>
<td>delXFile</td>
<td>PARM04</td>
<td>' '</td>
<td>filename containing 1D array of $x$-axis grid spacing</td>
</tr>
<tr>
<td>dySpacing</td>
<td>PARM04</td>
<td>unset</td>
<td>$y$-axis uniform grid spacing, separation between cell faces (Cartesian and cylindrical: m; spherical: degrees)</td>
</tr>
<tr>
<td>delY</td>
<td>PARM04</td>
<td>dySpacing</td>
<td>1D array of $x$-axis grid spacing, separation between cell faces (Cartesian and cylindrical: m; spherical: degrees)</td>
</tr>
<tr>
<td>delYFile</td>
<td>PARM04</td>
<td>' '</td>
<td>filename containing 1D array of $y$-axis grid spacing</td>
</tr>
<tr>
<td>cosPower</td>
<td>PARM01</td>
<td>0.0</td>
<td>power law $n$ in $(\cos \phi)^n$ factor for horizontal (harmonic or biharmonic) viscosity and tracer diffusivity (spherical polar)</td>
</tr>
<tr>
<td>delR</td>
<td>PARM04</td>
<td>computed using delR</td>
<td>vertical grid spacing 1D array ($r$ unit)</td>
</tr>
<tr>
<td>delRc</td>
<td>PARM04</td>
<td>computed using delR</td>
<td>vertical cell center spacing 1D array ($r$ unit)</td>
</tr>
<tr>
<td>delRFile</td>
<td>PARM04</td>
<td>' '</td>
<td>filename for vertical grid spacing 1D array ($r$ unit)</td>
</tr>
<tr>
<td>delRcFile</td>
<td>PARM04</td>
<td>' '</td>
<td>filename for vertical cell center spacing 1D array ($r$ unit)</td>
</tr>
<tr>
<td>rSphere</td>
<td>PARM04</td>
<td>6.37E+06</td>
<td>radius of sphere for spherical polar or curvilinear grid (m)</td>
</tr>
<tr>
<td>seaLev_Z</td>
<td>PARM04</td>
<td>0.0</td>
<td>reference height of sea level (m)</td>
</tr>
<tr>
<td>top_Pres</td>
<td>PARM04</td>
<td>0.0</td>
<td>top pressure ($p$-coordinates) or top reference pressure ($z$-coordinates) (Pa)</td>
</tr>
<tr>
<td>selectFindRoSurf</td>
<td>PARM01</td>
<td>0</td>
<td>select method to determine surface reference pressure from orography (atmos.-only)</td>
</tr>
<tr>
<td>horizGridFile</td>
<td>PARM04</td>
<td>' '</td>
<td>filename containing full set of horizontal grid variables (curvilinear)</td>
</tr>
<tr>
<td>radius_fromHorizGrid</td>
<td>PARM04</td>
<td>rSphere</td>
<td>radius of sphere used in input curvilinear horizontal grid file (m)</td>
</tr>
<tr>
<td>phiEuler</td>
<td>PARM04</td>
<td>0.0</td>
<td>Euler angle, rotation about original $z$-axis (spherical polar) (degrees)</td>
</tr>
<tr>
<td>thetaEuler</td>
<td>PARM04</td>
<td>0.0</td>
<td>Euler angle, rotation about new $x$-axis (spherical polar) (degrees)</td>
</tr>
<tr>
<td>psiEuler</td>
<td>PARM04</td>
<td>0.0</td>
<td>Euler angle, rotation about new $z$-axis (spherical polar) (degrees)</td>
</tr>
</tbody>
</table>

3.8.1.3 Topography - Full and Partial Cells

For the ocean, the topography is read from a file that contains a 2-D ($x$, $y$) map of bathymetry, in meters for $z$-coordinates, in pascals for $p$-coordinates. The bathymetry is specified by entering the vertical position of the ocean floor relative to the surface, so by convention in $z$-coordinates bathymetry is specified as negative numbers (“depth” is defined as positive-definite) whereas in $p$-coordinates bathymetry data is positive. The file name is represented by the variable bathyFile. See our introductory tutorial setup Section 4.1 for additional details on the file format. Note no changes are required in the model source code to represent enclosed, periodic, or double periodic domains: periodicity is assumed by default and is suppressed by setting the depths to 0 m for the cells at the limits of the computational domain.

To use the partial cell capability, the variable hFacMin needs to be set to a value between 0.0 and 1.0 (it is set to 1.0 by default) corresponding to the minimum fractional size of a gridcell. For example, if a gridcell is 500 m thick and...
hFacMin is set to 0.1, the minimum thickness for a “thin-cell” for this specific gridcell is 50 m. Thus, if the specified bathymetry depth were to fall exactly in the middle of this 500m thick gridcell, the initial model variable $hFacC(x, y, r)$ would be set to 0.5. If the specified bathymetry depth fell within the top 50m of this gridcell (i.e., less than hFacMin), the model bathymetry would snap to the nearest legal value (i.e., initial $hFacC(x, y, r)$ would be equal to 0.0 or 0.1 depending if the depth was within 0-25 m or 25-50 m, respectively). Also note while specified bathymetry bottom depths (or pressures) need not coincide with the model’s levels as deduced from delR, any depth falling below the model’s defined vertical axis is truncated.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Group</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bathyFile</td>
<td>PARM05</td>
<td>' '</td>
<td>filename for 2D bathymetry (ocean) (z-coor.: m, negative; p-coor.: Pa, positive)</td>
</tr>
<tr>
<td>topoFile</td>
<td>PARM05</td>
<td>' '</td>
<td>filename for 2D surface topography (atmosphere) (m)</td>
</tr>
<tr>
<td>addWwallFile</td>
<td>PARM05</td>
<td>' '</td>
<td>filename for 2D western cell-edge “thin-wall”</td>
</tr>
<tr>
<td>addSwallFile</td>
<td>PARM05</td>
<td>' '</td>
<td>filename for 2D southern cell-edge “thin-wall”</td>
</tr>
<tr>
<td>hFacMin</td>
<td>PARM01</td>
<td>1.0E+00</td>
<td>minimum fraction size of a cell</td>
</tr>
<tr>
<td>hFacMinDr</td>
<td>PARM01</td>
<td>1.0E+00</td>
<td>minimum dimensional size of a cell ([r] unit)</td>
</tr>
<tr>
<td>hFacInf</td>
<td>PARM01</td>
<td>2.0E-01</td>
<td>lower threshold fraction for surface cell; for non-linear free surface only, see parameter nonlinFreeSurf</td>
</tr>
<tr>
<td>hFacSup</td>
<td>PARM01</td>
<td>2.0E+00</td>
<td>upper threshold fraction for surface cell; for non-linear free surface, only see parameter nonlinFreeSurf</td>
</tr>
<tr>
<td>useMin4hFacEdges</td>
<td>PARM04</td>
<td>FALSE</td>
<td>set hFacW, hFacS as minimum of adjacent hFacC on/off flag</td>
</tr>
<tr>
<td>pCellMix_select</td>
<td>PARM04</td>
<td>0</td>
<td>option/factor to enhance mixing at the surface or bottom (0-99)</td>
</tr>
<tr>
<td>pCellMix_maxFac</td>
<td>PARM04</td>
<td>1.0E+04</td>
<td>maximum enhanced mixing factor for too thin partial-cell (non-dim.)</td>
</tr>
<tr>
<td>pCellMix_delR</td>
<td>PARM04</td>
<td>0.0</td>
<td>thickness criteria for too thin partial-cell ([r] unit)</td>
</tr>
</tbody>
</table>

### 3.8.1.4 Physical Constants

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Group</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rhoConst</td>
<td>PARM01</td>
<td>rhoNil</td>
<td>vertically constant reference density (Boussinesq) (kg/m$^3$)</td>
</tr>
<tr>
<td>gravity</td>
<td>PARM01</td>
<td>9.81E+00</td>
<td>gravitational acceleration (m/s$^2$)</td>
</tr>
<tr>
<td>gravityFile</td>
<td>PARM01</td>
<td>' '</td>
<td>filename for 1D gravity vertical profile (m/s$^2$)</td>
</tr>
<tr>
<td>gBaro</td>
<td>PARM01</td>
<td>gravity</td>
<td>gravity constant in barotropic equation (m/s$^2$)</td>
</tr>
</tbody>
</table>

### 3.8.1.5 Rotation

For a Cartesian or cylindrical grid, the Coriolis parameter $f$ is set through the variables $f_0$ (in $s^{-1}$) and $\beta$ ($\frac{\partial f}{\partial y}$; in $m^{-1}s^{-1}$), which corresponds to a Coriolis parameter $f = f_0 + \beta y$ (the so-called $\beta$-plane).
### 3.8.1.6 Free Surface

The logical variables `rigidLid` and `implicitFreeSurface` specify your choice for ocean upper boundary (or lower boundary if using p-coordinates); set one to `.TRUE.` and the other to `.FALSE.`. These settings affect the calculations of surface pressure (for the ocean) or surface geopotential (for the atmosphere); see Section 3.8.2.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Group</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>implicitFreeSurface</td>
<td>PARM01</td>
<td>TRUE</td>
<td>implicit free surface on/off flag</td>
</tr>
<tr>
<td>rigidLid</td>
<td>PARM01</td>
<td>FALSE</td>
<td>rigid lid on/off flag</td>
</tr>
<tr>
<td>useRealFreshWater-Flux</td>
<td>PARM01</td>
<td>FALSE</td>
<td>use true E-P-R freshwater flux (changes free surface/sea level) on/off flag</td>
</tr>
<tr>
<td>implicSurfPress</td>
<td>PARM01</td>
<td>1.0E+00</td>
<td>implicit fraction of the surface pressure gradient (0-1)</td>
</tr>
<tr>
<td>implicDiv2Dflow</td>
<td>PARM01</td>
<td>1.0E+00</td>
<td>implicit fraction of the barotropic flow divergence (0-1)</td>
</tr>
<tr>
<td>implicitNHPress</td>
<td>PARM01</td>
<td>implicSurfPress</td>
<td>implicit fraction of the non-hydrostatic pressure gradient (0-1); for non-hydrostatic only, see parameter <code>nonHydrostatic</code></td>
</tr>
<tr>
<td>nonlinFreeSurf</td>
<td>PARM01</td>
<td>0</td>
<td>non-linear free surface options (-1,0,1,2,3; see Table 2.1); requires #define NONLIN_FRSURF</td>
</tr>
<tr>
<td>select_rStar</td>
<td>PARM01</td>
<td>0</td>
<td>vertical coordinate option</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• 0: use r</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• &gt;0: use $r^*$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>see Table 2.1; requires #define NONLIN_FRSURF</td>
</tr>
<tr>
<td>selectNHfreeSurf</td>
<td>PARM01</td>
<td>0</td>
<td>non-hydrostatic free surface formulation option</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• 0: don’t use</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• &gt;0: use</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>requires non-hydrostatic formulation, see parameter <code>non-Hydrostatic</code></td>
</tr>
<tr>
<td>exactConserv</td>
<td>PARM01</td>
<td>FALSE</td>
<td>exact total volume conservation (recompute divergence after pressure solver) on/off flag</td>
</tr>
</tbody>
</table>
3.8.1.7 Time-Discretization

The time steps are set through the real variables deltaTMom and deltaTtracer (in seconds) which represent the time step for the momentum and tracer equations, respectively (or you can prescribe a single time step value for all parameters using deltaT). The model “clock” is defined by the variable deltaTClock (in seconds) which determines the I/O frequencies and is used in tagging output. Time in the model is thus computed as:

\[ \text{model time} = \text{baseTime} + \text{iteration number} \times \text{deltaTClock} \]

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Group</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>deltaT</td>
<td>PARM03</td>
<td>0.0</td>
<td>default value used for model time step parameters (s)</td>
</tr>
<tr>
<td>deltaTClock</td>
<td>PARM03</td>
<td>deltaT</td>
<td>timestep used for model clock (s): used for I/O frequency and tagging output</td>
</tr>
<tr>
<td>deltaTmom</td>
<td>PARM03</td>
<td>deltaT</td>
<td>momentum equation timestep (s)</td>
</tr>
<tr>
<td>deltaTtracer</td>
<td>PARM03</td>
<td>deltaT</td>
<td>tracer equation timestep (s)</td>
</tr>
<tr>
<td>dTtracerLev</td>
<td>PARM03</td>
<td>deltaT</td>
<td>tracer equation timestep specified at each vertical level (s)</td>
</tr>
<tr>
<td>deltaTtreesurf</td>
<td>PARM03</td>
<td>deltaTmom</td>
<td>free-surface equation timestep (s)</td>
</tr>
<tr>
<td>baseTime</td>
<td>PARM03</td>
<td>0.0</td>
<td>model base time corresponding to iteration 0 (s)</td>
</tr>
</tbody>
</table>

3.8.2 Parameters: Main Algorithmic Parameters

3.8.2.1 Pressure Solver

By default, a hydrostatic simulation is assumed and a 2-D elliptic equation is used to invert the pressure field. If using a non-hydrostatic configuration, the pressure field is inverted through a 3-D elliptic equation (note this capability is not yet available for the atmosphere). The parameters controlling the behavior of the elliptic solvers are the variables cg2dMaxIters and cg2dTargetResidual for the 2-D case and cg3dMaxIters and cg3dTargetResidual for the 3-D case.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Group</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cg2dMaxIters</td>
<td>PARM02</td>
<td>150</td>
<td>upper limit on 2D conjugate gradient solver iterations</td>
</tr>
<tr>
<td>cg2dTargetResidual</td>
<td>PARM02</td>
<td>1.0E-07</td>
<td>2D conjugate gradient target residual (non-dim. due to RHS normalization)</td>
</tr>
<tr>
<td>cg2dTargetResWunit</td>
<td>PARM02</td>
<td>-1.0E+00</td>
<td>2D conjugate gradient target residual ( (\hat{T}) ) units; &lt;0: use RHS normalization, i.e., cg2dTargetResidual instead</td>
</tr>
<tr>
<td>cg2dPreCondFreq</td>
<td>PARM02</td>
<td>1</td>
<td>frequency (in number of iterations) for updating cg2d preconditioner; for non-linear free surface only, see parameter nonlinFreeSurf</td>
</tr>
<tr>
<td>cg2dUseMinResSol</td>
<td>PARM02</td>
<td>0 unless flat-bottom, Cartesian</td>
<td>0: use last-iteration/converged cg2d solution</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1: use solver minimum-residual solution</td>
</tr>
<tr>
<td>cg3dMaxIters</td>
<td>PARM02</td>
<td>150</td>
<td>upper limit on 3D conjugate gradient solver iterations; requires #define ALLOW_NONHYDROSTATIC</td>
</tr>
<tr>
<td>cg3dTargetResidual</td>
<td>PARM02</td>
<td>1.0E-07</td>
<td>3D conjugate gradient target residual (non-dim. due to RHS normalization); requires #define ALLOW_NONHYDROSTATIC</td>
</tr>
</tbody>
</table>

continues on next page
### 3.8.2.2 Time-Stepping Algorithm

The Adams-Bashforth stabilizing parameter is set through the variable abEps (dimensionless). The stagger baroclinic time stepping algorithm can be activated by setting the logical variable staggerTimeStep to .TRUE.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Group</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>abEps</td>
<td>PARM03</td>
<td>1.0E-02</td>
<td>Adams-Bashforth-2 stabilizing weight (non-dim.)</td>
</tr>
<tr>
<td>alph_AB</td>
<td>PARM03</td>
<td>0.5E-00</td>
<td>Adams-Bashforth-3 primary factor (non-dim.); requires #define ALLOW_ADAMSBASHFORTH_3</td>
</tr>
<tr>
<td>beta_AB</td>
<td>PARM03</td>
<td>5/12</td>
<td>Adams-Bashforth-3 secondary factor (non-dim.); requires #define ALLOW_ADAMSBASHFORTH_3</td>
</tr>
<tr>
<td>staggerTimeStep</td>
<td>PARM01</td>
<td>FALSE</td>
<td>use staggered time stepping (thermodynamic vs. flow variables) on/off flag</td>
</tr>
<tr>
<td>multiDimAdvection</td>
<td>PARM01</td>
<td>TRUE</td>
<td>use multi-dim. advection algorithm in schemes where non multi-dim. is possible on/off flag</td>
</tr>
<tr>
<td>implicitIntGravWave</td>
<td>PARM01</td>
<td>FALSE</td>
<td>treat internal gravity waves implicitly on/off flag; requires #define ALLOW_NONHYDROSTATIC</td>
</tr>
</tbody>
</table>

### 3.8.3 Parameters: Equation of State

The form of the equation of state is controlled by the model configuration and eosType.

For the atmosphere, eosType must be set to IDEALGAS.

For the ocean, several forms of the equation of state are available:

- For a linear approximation, set eosType to LINEAR, and you will need to specify the thermal and haline expansion coefficients, represented by the variables tAlpha (in K⁻¹) and sBeta (in (g/kg)⁻¹). Because the model equations are written in terms of perturbations, a reference thermodynamic state needs to be specified. This is done through the 1-D arrays tRef and sRef. tRef specifies the reference potential temperature profile (in °C for the ocean and K for the atmosphere) starting from the level k=1. Similarly, sRef specifies the reference salinity profile (in g/kg) for the ocean or the reference specific humidity profile (in g/kg) for the atmosphere.
- MITgcm offers several approximations to the full (oceanic) non-linear equation of state that can be selected as eosType:
"POLYNOMIAL": This approximation is based on the Knudsen formula (see Bryan and Cox 1972 [BC72]). For this option you need to generate a file of polynomial coefficients called POLY3.COEFFS. To do this, use the program utils/knudsen2/knudsen2.f under the model tree (a Makefile is available in the same directory; you will need to edit the number and the values of the vertical levels in knudsen2.f so that they match those of your configuration).

"UNESCO": The UNESCO equation of state formula (IES80) of Fofonoff and Millard (1983) [FRM83]. This equation of state assumes in-situ temperature, which is not a model variable; its use is therefore discouraged.

"JMD95Z": A modified UNESCO formula by Jackett and McDougall (1995) [JM95], which uses the model variable potential temperature as input. The ‘Z’ indicates that this equation of state uses a horizontally and temporally constant pressure $p_0 = -g \rho_0 z$.

"JMD95P": A modified UNESCO formula by Jackett and McDougall (1995) [JM95], which uses the model variable potential temperature as input. The ‘P’ indicates that this equation of state uses the actual hydrostatic pressure of the last time step. Lagging the pressure in this way requires an additional pickup file for restarts.

"MDJWF": A more accurate and less expensive equation of state than UNESCO by McDougall et al. (2003) [MJWF03], also using the model variable potential temperature as input. It also requires lagging the pressure and therefore an additional pickup file for restarts.

"TEOS10": TEOS-10 is based on a Gibbs function formulation from which all thermodynamic properties of seawater (density, enthalpy, entropy sound speed, etc.) can be derived in a thermodynamically consistent manner; see http://www.teos-10.org. See IOC et al. (2010) [ISI10], McDougall and Parker (2011) [MB11], and Roquet et al. (2015) [RMB15] for implementation details. It also requires lagging the pressure and therefore an additional pickup file for restarts. Note at this time a full implementation of TEOS10 (i.e., ocean variables of conservative temperature and practical salinity, including consideration of surface forcings) has not been implemented; also note the original 48-term polynomial term is used, not the newer, preferred 75-term polynomial.

For these non-linear approximations, neither a reference profile of temperature or salinity is required, except for a setup where implicitIntGravWave is set to .TRUE. or selectP_inEOS_Zc=1.

Note that for simplicity, salinity is expressed as a ratio in g/kg (thus effectively unitless) regardless of the choice of equation of state, despite “Practical Salinity” not precisely equal to salinity expressed as a dissolved mass fraction. If TEOS-10 is selected, the model variable salt can be interpreted as “Absolute Salinity”. See Millero (2010) [Mil10] and Pawlowicz (2013) [Paw13] for detailed discussion of salinity measurements, and why being expressed as g/kg is preferred, in the context of the ocean equation of state.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Group</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>eosType</td>
<td>PARM01</td>
<td>LINEAR</td>
<td>equation of state form</td>
</tr>
<tr>
<td>tRef</td>
<td>PARM01</td>
<td>20.0°C (ocn) or 300.0 K (atm)</td>
<td>1D vertical reference temperature profile (°C or K)</td>
</tr>
<tr>
<td>tRefFile</td>
<td>PARM01</td>
<td>*' '</td>
<td>filename for reference temperature profile (°C or K)</td>
</tr>
<tr>
<td>thetaConst</td>
<td>PARM01</td>
<td>tRef(k=1)</td>
<td>vertically constant reference temp. for atmosphere $p^*$ coordinates (°K); for ocean, specify instead of tRef or tRefFile for vertically constant reference temp. (°C)</td>
</tr>
<tr>
<td>sRef</td>
<td>PARM01</td>
<td>30.0 (g/kg) (ocn) or 0.0 (atm)</td>
<td>1D vertical reference salinity profile (g/kg)</td>
</tr>
<tr>
<td>sRefFile</td>
<td>PARM01</td>
<td>*' '</td>
<td>filename for reference salinity profile (g/kg)</td>
</tr>
</tbody>
</table>

continues on next page
Table 3.7 – continued from previous page

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Group</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
</table>
| selectP_inEOS_Zc        | PARM01  | depends on eosType | select which pressure to use in EOS for \( z \)-coor.  
|                         |         |               |  
|                         |         |               | • 0: use \(-g𝜌c_z\)  
|                         |         |               | • 1: use \(p_{ref} = -\int -g𝜌(T_{ref}, S_{ref}, p_{ref})dz\)  
|                         |         |               | • 2: hydrostatic dynamical pressure  
|                         |         |               | • 3: use full hyd.+non-hyd. pressure for MD95P, UNESCO, MDJWF, TEOS10 default=2, otherwise default =0 |
| rhonil                  | PARM01  | 9.998E+02     | reference density for linear EOS (kg/m\(^3\)) |
| tAlpha                  | PARM01  | 2.0E-04       | linear EOS thermal expansion coefficient (1/°C) |
| sBeta                   | PARM01  | 7.4E-04       | linear EOS haline contraction coefficient ((g/kg)\(^{-1}\)) |

3.8.3.1 Thermodynamic Constants

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Group</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HeatCapacity_Cp</td>
<td>PARM01</td>
<td>3.994E+03</td>
<td>specific heat capacity ( C_p ) (ocean) (J/kg/K)</td>
</tr>
<tr>
<td>celsius2K</td>
<td>PARM01</td>
<td>2.7315E+02</td>
<td>conversion constant °C to Kelvin</td>
</tr>
<tr>
<td>atm_Cp</td>
<td>PARM01</td>
<td>1.004E+03</td>
<td>specific heat capacity ( C_p ) dry air at const. press. (J/kg/K)</td>
</tr>
<tr>
<td>atm_Rd</td>
<td>PARM01</td>
<td>atm_Cp*(2/7)</td>
<td>gas constant for dry air (J/kg/K)</td>
</tr>
<tr>
<td>atm_Rq</td>
<td>PARM01</td>
<td>0.0</td>
<td>water vapor specific volume anomaly relative to dry air (g/kg)</td>
</tr>
<tr>
<td>atm_Po</td>
<td>PARM01</td>
<td>1.0E+05</td>
<td>atmosphere standard reference pressure (for potential temp. defn.) (Pa)</td>
</tr>
</tbody>
</table>

3.8.4 Parameters: Momentum Equations

3.8.4.1 Configuration

There are a few logical variables that allow you to turn on/off various terms in the momentum equation. These variables are called momViscosity, momAdvection, useCoriolis, momStepping, metricTerms, and momPressureForcing and by default are set to .TRUE.. Vertical diffusive fluxes of momentum can be computed implicitly by setting the logical variable implicitViscosity to .TRUE.. The details relevant to both the momentum flux-form and the vector-invariant form of the equations and the various (momentum) advection schemes are covered in Section 2.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Group</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>momStepping</td>
<td>PARM01</td>
<td>TRUE</td>
<td>momentum equation time-stepping on/off flag</td>
</tr>
<tr>
<td>momViscosity</td>
<td>PARM01</td>
<td>TRUE</td>
<td>momentum friction terms on/off flag</td>
</tr>
<tr>
<td>momAdvection</td>
<td>PARM01</td>
<td>TRUE</td>
<td>advection of momentum on/off flag</td>
</tr>
<tr>
<td>momPressureForcing</td>
<td>PARM01</td>
<td>TRUE</td>
<td>pressure term in momentum equation on/off flag</td>
</tr>
<tr>
<td>metricTerms</td>
<td>PARM01</td>
<td>TRUE</td>
<td>include metric terms (spherical polar, momentum flux-form) on/off flag</td>
</tr>
<tr>
<td>useNHMTerms</td>
<td>PARM01</td>
<td>FALSE</td>
<td>use “non-hydrostatic form” of metric terms on/off flag; (see Section 2.14.4; note these terms are non-zero in many model configurations beside non-hydrostatic)</td>
</tr>
<tr>
<td>momImplVertAdv</td>
<td>PARM01</td>
<td>FALSE</td>
<td>momentum implicit vertical advection on/off flag; requires #define INCLUDE_IMPLVERTADV_CODE</td>
</tr>
<tr>
<td>implicitViscosity</td>
<td>PARM01</td>
<td>FALSE</td>
<td>implicit vertical viscosity on/off flag</td>
</tr>
<tr>
<td>interViscAr_pCell</td>
<td>PARM04</td>
<td>FALSE</td>
<td>account for partial-cell in interior vertical viscosity on/off flag</td>
</tr>
</tbody>
</table>

continues on next page
Table 3.8 – continued from previous page

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Group</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>momDissip_In_AB</td>
<td>PARM03</td>
<td>TRUE</td>
<td>use Adams-Bashforth time stepping for dissipation tendency</td>
</tr>
<tr>
<td>useCoriolis</td>
<td>PARM01</td>
<td>TRUE</td>
<td>include Coriolis terms on/off flag</td>
</tr>
<tr>
<td>use3dCoriolis</td>
<td>PARM01</td>
<td>TRUE</td>
<td>include cos $\phi$ Coriolis terms on/off flag</td>
</tr>
<tr>
<td>selectCoriScheme</td>
<td>PARM01</td>
<td>0</td>
<td>Coriolis scheme selector</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• 0: original scheme</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• 1: wet-point averaging method</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• 2: Flux-Form: energy conserving; Vector-Inv: hFac weighted average</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• 3: Flux-Form: energy conserving using wet-point method; Vector-Inv: energy conserving with hFac weight</td>
</tr>
<tr>
<td>vectorInvariantMomentum</td>
<td>PARM01</td>
<td>FALSE</td>
<td>use vector-invariant form of momentum equations flag</td>
</tr>
<tr>
<td>useJamartMomAdv</td>
<td>PARM01</td>
<td>FALSE</td>
<td>use Jamart wetpoints method for relative vorticity advection (vector invariant form) on/off flag</td>
</tr>
<tr>
<td>selectVortScheme</td>
<td>PARM01</td>
<td>1</td>
<td>vorticity scheme (vector invariant form) options</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• 0,1: enstrophy conserving forms</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• 2: energy conserving form</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• 3: energy and enstrophy conserving form</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>see Sadourny 1975 [Sad75] and Burridge &amp; Haseler 1977 [BH77]</td>
</tr>
<tr>
<td>upwindVorticity</td>
<td>PARM01</td>
<td>FALSE</td>
<td>bias interpolation of vorticity in the Coriolis term (vector invariant form) on/off flag</td>
</tr>
<tr>
<td>useAbsVorticity</td>
<td>PARM01</td>
<td>FALSE</td>
<td>use $f + \zeta$ in Coriolis terms (vector invariant form) on/off flag</td>
</tr>
<tr>
<td>highOrderVorticity</td>
<td>PARM01</td>
<td>FALSE</td>
<td>use 3rd/4th order interpolation of vorticity (vector invariant form) on/off flag</td>
</tr>
<tr>
<td>upwindShear</td>
<td>PARM01</td>
<td>FALSE</td>
<td>use 1st order upwind for vertical advection (vector invariant form) on/off flag</td>
</tr>
<tr>
<td>selectKEscheme</td>
<td>PARM01</td>
<td>0</td>
<td>kinetic energy computation in Bernoulli function (vector invariant form) options</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• 0: standard form</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• 1: area-weighted standard form</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• 2: as 0 but account for partial cells</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• 3: as 1 w/partial cells</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>see mom_calc_ke.F</td>
</tr>
</tbody>
</table>

3.8.4.2 Initialization

The initial horizontal velocity components can be specified from binary files uVelInitFile and vVelInitFile. These files should contain 3-D data ordered in an $(x, y, r)$ fashion with $k=1$ as the first vertical level (surface level). If no file names are provided, the velocity is initialized to zero. The initial vertical velocity is always derived from the horizontal velocity using the continuity equation. In the case of a restart (from the end of a previous simulation), the velocity field is read from a pickup file (see Section 3.8.7) and the initial velocity files are ignored.
3.8.4.3 General Dissipation Scheme

The lateral eddy viscosity coefficient is specified through the variable \( \text{viscAh} \) (in \( \text{m}^2\text{s}^{-1} \)). The vertical eddy viscosity coefficient is specified through the variable \( \text{viscAr} \) (in \( [r]^2\text{s}^{-1} \), where \([r]\) is the dimension of the vertical coordinate). In addition, biharmonic mixing can be added as well through the variable \( \text{viscA4} \) (in \( \text{m}^4\text{s}^{-1} \)).

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Group</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{viscAh} )</td>
<td>PARM01</td>
<td>0.0</td>
<td>lateral eddy viscosity (( \text{m}^2\text{s} ))</td>
</tr>
<tr>
<td>( \text{viscAhD} )</td>
<td>PARM01</td>
<td>( \text{viscAh} )</td>
<td>lateral eddy viscosity acts on divergence part (( \text{m}^2\text{s} ))</td>
</tr>
<tr>
<td>( \text{viscAhZ} )</td>
<td>PARM01</td>
<td>( \text{viscAh} )</td>
<td>lateral eddy viscosity acts on vorticity part (( \zeta ) points) (( \text{m}^2\text{s} ))</td>
</tr>
<tr>
<td>( \text{viscAhW} )</td>
<td>PARM01</td>
<td>( \text{viscAhD} )</td>
<td>lateral eddy viscosity for mixing vertical momentum (non-hydrostatic form) (( \text{m}^2\text{s} )); for non-hydrostatic only, see parameter ( \text{nonHydrostatic} )</td>
</tr>
<tr>
<td>( \text{viscAhDfile} )</td>
<td>PARM05</td>
<td>' '</td>
<td>filename for 3D specification of lateral eddy viscosity (divergence part) (( \text{m}^2\text{s} )); requires #define ALLOW_3D_VISCAH in pkg/mom_common/MOM_COMMON_OPTIONS.h</td>
</tr>
<tr>
<td>( \text{viscAhZfile} )</td>
<td>PARM05</td>
<td>' '</td>
<td>filename for 3D specification of lateral eddy viscosity (vorticity part, ( \zeta ) points); requires #define ALLOW_3D_VISCAH in pkg/mom_common/MOM_COMMON_OPTIONS.h</td>
</tr>
<tr>
<td>( \text{viscAhGrid} )</td>
<td>PARM01</td>
<td>0.0</td>
<td>grid-dependent lateral eddy viscosity (non-dim.)</td>
</tr>
<tr>
<td>( \text{viscAhMax} )</td>
<td>PARM01</td>
<td>1.0E+21</td>
<td>maximum lateral eddy viscosity (( \text{m}^2\text{s} ))</td>
</tr>
<tr>
<td>( \text{viscAhGridMax} )</td>
<td>PARM01</td>
<td>1.0E+21</td>
<td>maximum lateral eddy (grid-dependent) viscosity (non-dim.)</td>
</tr>
<tr>
<td>( \text{viscAhGridMin} )</td>
<td>PARM01</td>
<td>0.0</td>
<td>minimum lateral eddy (grid-dependent) viscosity (non-dim.)</td>
</tr>
<tr>
<td>( \text{viscAhReMax} )</td>
<td>PARM01</td>
<td>0.0</td>
<td>minimum lateral eddy viscosity based on Reynolds number (non-dim.)</td>
</tr>
<tr>
<td>( \text{viscC2leith} )</td>
<td>PARM01</td>
<td>0.0</td>
<td>Leith harmonic viscosity factor (vorticity part, ( \zeta ) points) (non-dim.)</td>
</tr>
<tr>
<td>( \text{viscC2leithD} )</td>
<td>PARM01</td>
<td>0.0</td>
<td>Leith harmonic viscosity factor (divergence part) (non-dim.)</td>
</tr>
<tr>
<td>( \text{viscC2LeithQG} )</td>
<td>PARM01</td>
<td>0.0</td>
<td>Quasi-geostrophic Leith viscosity factor (non-dim.)</td>
</tr>
<tr>
<td>( \text{viscC2smag} )</td>
<td>PARM01</td>
<td>0.0</td>
<td>Smagorinsky harmonic viscosity factor (non-dim.)</td>
</tr>
<tr>
<td>( \text{viscA4} )</td>
<td>PARM01</td>
<td>0.0</td>
<td>lateral biharmonic viscosity (( \text{m}^2\text{s} ))</td>
</tr>
<tr>
<td>( \text{viscA4D} )</td>
<td>PARM01</td>
<td>( \text{viscA4} )</td>
<td>lateral biharmonic viscosity (divergence part) (( \text{m}^2\text{s} ))</td>
</tr>
<tr>
<td>( \text{viscA4Z} )</td>
<td>PARM01</td>
<td>( \text{viscA4} )</td>
<td>lateral biharmonic viscosity (vorticity part, ( \zeta ) points) (( \text{m}^2\text{s} ))</td>
</tr>
<tr>
<td>( \text{viscA4W} )</td>
<td>PARM01</td>
<td>( \text{viscA4D} )</td>
<td>lateral biharmonic viscosity for mixing vertical momentum (non-hydrostatic form) (( \text{m}^2\text{s} )); for non-hydrostatic only, see parameter ( \text{nonHydrostatic} )</td>
</tr>
<tr>
<td>( \text{viscA4Dfile} )</td>
<td>PARM05</td>
<td>' '</td>
<td>filename for 3D specification of lateral biharmonic viscosity (divergence part) (( \text{m}^2\text{s} )); requires #define ALLOW_3D_VISCA4 in pkg/mom_common/MOM_COMMON_OPTIONS.h</td>
</tr>
<tr>
<td>( \text{viscA4Zfile} )</td>
<td>PARM05</td>
<td>' '</td>
<td>filename for 3D specification of lateral biharmonic viscosity (vorticity part, ( \zeta ) points); requires #define ALLOW_3D_VISCA4 in pkg/mom_common/MOM_COMMON_OPTIONS.h</td>
</tr>
<tr>
<td>( \text{viscA4Grid} )</td>
<td>PARM01</td>
<td>0.0</td>
<td>grid-dependent biharmonic viscosity (non-dim.)</td>
</tr>
<tr>
<td>( \text{viscA4Max} )</td>
<td>PARM01</td>
<td>1.0E+21</td>
<td>maximum biharmonic viscosity (( \text{m}^2\text{s} ))</td>
</tr>
</tbody>
</table>
### Table 3.9 – continued from previous page

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Group</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>viscA4GridMax</td>
<td>PARM01</td>
<td>1.0E+21</td>
<td>maximum biharmonic (grid-dependent) viscosity (non-dim.)</td>
</tr>
<tr>
<td>viscA4GridMin</td>
<td>PARM01</td>
<td>0.0</td>
<td>minimum biharmonic (grid-dependent) viscosity (non-dim.)</td>
</tr>
<tr>
<td>viscA4ReMax</td>
<td>PARM01</td>
<td>0.0</td>
<td>minimum biharmonic viscosity based on Reynolds number (non-dim.)</td>
</tr>
<tr>
<td>viscC4leith</td>
<td>PARM01</td>
<td>0.0</td>
<td>Leith biharmonic viscosity factor (vorticity part, (\zeta) points) (non-dim.)</td>
</tr>
<tr>
<td>viscC4leithD</td>
<td>PARM01</td>
<td>0.0</td>
<td>Leith biharmonic viscosity factor (divergence part) (non-dim.)</td>
</tr>
<tr>
<td>viscC4smag</td>
<td>PARM01</td>
<td>0.0</td>
<td>Smagorinsky biharmonic viscosity factor (non-dim.)</td>
</tr>
<tr>
<td>useFullLeith</td>
<td>PARM01</td>
<td>FALSE</td>
<td>use full form of Leith viscosities on/off flag</td>
</tr>
<tr>
<td>useSmag3D</td>
<td>PARM01</td>
<td>FALSE</td>
<td>use isotropic 3D Smagorinsky harmonic viscosities flag; requires #define ALLOW_SMAG_3D in pkg/mom_common/MOM_COMMON_OPTIONS.h</td>
</tr>
<tr>
<td>smag3D_coeff</td>
<td>PARM01</td>
<td>1.0E-02</td>
<td>isotropic 3D Smagorinsky coefficient (non-dim.); requires #define ALLOW_SMAG_3D in pkg/mom_common/MOM_COMMON_OPTIONS.h</td>
</tr>
<tr>
<td>useStrainTensionVisc</td>
<td>PARM01</td>
<td>FALSE</td>
<td>flag to use strain-tension form of viscous operator</td>
</tr>
<tr>
<td>useAreaViscLength</td>
<td>PARM01</td>
<td>FALSE</td>
<td>flag to use area for viscous (L^2) instead of harmonic mean of (L_x^2, L_y^2)</td>
</tr>
<tr>
<td>viscAr</td>
<td>PARM01</td>
<td>0.0</td>
<td>vertical eddy viscosity (([r]^2/s))</td>
</tr>
<tr>
<td>viscArNr</td>
<td>PARM01</td>
<td>0.0</td>
<td>vertical profile of vertical eddy viscosity (([r]^2/s))</td>
</tr>
<tr>
<td>pCellMix_viscAr</td>
<td>PARM04</td>
<td>viscArNr</td>
<td>vertical viscosity for too thin partial-cell (([r]^2/s))</td>
</tr>
</tbody>
</table>

#### 3.8.4.4 Sidewall/Bottom Dissipation

Slip or no-slip conditions at lateral and bottom boundaries are specified through the logical variables no_slip_sides and no_slip_bottom. If set to .FALSE., free-slip boundary conditions are applied. If no-slip boundary conditions are applied at the bottom, a bottom drag can be applied as well. Two forms are available: linear (set the variable bottomDragLinear in \([r]/s\)) and quadratic (set the variable bottomDragQuadratic, \([r]/m\)).

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Group</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>no_slip_sides</td>
<td>PARM01</td>
<td>TRUE</td>
<td>viscous BCs: no-slip sides on/off flag</td>
</tr>
<tr>
<td>sideDragFactor</td>
<td>PARM01</td>
<td>2.0E+00</td>
<td>side-drag scaling factor (2.0: full drag) (non-dim.)</td>
</tr>
<tr>
<td>no_slip_bottom</td>
<td>PARM01</td>
<td>TRUE</td>
<td>viscous BCs: no-slip bottom on/off flag</td>
</tr>
<tr>
<td>bottomDragLinear</td>
<td>PARM01</td>
<td>0.0</td>
<td>linear bottom-drag coefficient (([r]/s))</td>
</tr>
<tr>
<td>bottomDragQuadratic</td>
<td>PARM01</td>
<td>0.0</td>
<td>quadratic bottom-drag coefficient (([r]/m))</td>
</tr>
<tr>
<td>selectBotDragQuadr</td>
<td>PARM01</td>
<td>-1</td>
<td>select quadratic bottom drag discretization option</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• -1: not used</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• 0: average KE from grid center to (u, v) location</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• 1: use local velocity norm @ (u, v) location</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• 2: as 1 with wet-point averaging of other velocity component</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>if bottomDragQuadratic (\neq 0), then default is 0</td>
</tr>
<tr>
<td>selectImplicitDrag</td>
<td>PARM01</td>
<td>0</td>
<td>top/bottom drag implicit treatment options</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• 0: fully explicit</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• 1: implicit on provisional velocity, i.e., before (\nabla \eta) increment</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• 2: fully implicit</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>if =2, requires #define ALLOW_SOLVE4_PS_AND_DRAG</td>
</tr>
<tr>
<td>bottomVisc_pCell</td>
<td>PARM01</td>
<td>FALSE</td>
<td>account for partial-cell in bottom viscosity (using no_slip_bottom = .TRUE.) on/off flag</td>
</tr>
</tbody>
</table>
3.8.5 Parameters: Tracer Equations

This section covers the tracer equations, i.e., the potential temperature equation and the salinity (for the ocean) or specific humidity (for the atmosphere) equation.

3.8.5.1 Configuration

The logical variables tempAdvection and tempStepping allow you to turn on/off terms in the temperature equation (similarly for salinity or specific humidity with variables saltAdvection etc.). These variables all default to a value of .TRUE.. The vertical diffusive fluxes can be computed implicitly by setting the logical variable implicitDiffusion to .TRUE..

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Group</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tempStepping</td>
<td>PARM01</td>
<td>TRUE</td>
<td>temperature equation time-stepping on/off flag</td>
</tr>
<tr>
<td>tempAdvection</td>
<td>PARM01</td>
<td>TRUE</td>
<td>advection of temperature on/off flag</td>
</tr>
<tr>
<td>tempAdvScheme</td>
<td>PARM01</td>
<td>2</td>
<td>temperature horizontal advection scheme selector (see Table 2.2)</td>
</tr>
<tr>
<td>tempVertAdvScheme</td>
<td>PARM01</td>
<td>tempAdvScheme</td>
<td>temperature vertical advection scheme selector (see Table 2.2)</td>
</tr>
<tr>
<td>tempImplVertAdv</td>
<td>PARM01</td>
<td>FALSE</td>
<td>temperature implicit vertical advection on/off flag</td>
</tr>
<tr>
<td>addFrictionHeating</td>
<td>PARM01</td>
<td>FALSE</td>
<td>include frictional heating in temperature equation on/off flag; requires #define ALLOW_FRICTION_HEATING</td>
</tr>
<tr>
<td>temp_stayPositive</td>
<td>PARM01</td>
<td>FALSE</td>
<td>use Smolarkiewicz hack to ensure temperature stays positive on/off flag; requires #define GAD_SMOLARKIEWICZ_HACK in pkg/generic_advdiff/GAD_OPTIONS.h</td>
</tr>
<tr>
<td>saltStepping</td>
<td>PARM01</td>
<td>TRUE</td>
<td>salinity equation time-stepping on/off flag</td>
</tr>
<tr>
<td>saltAdvection</td>
<td>PARM01</td>
<td>TRUE</td>
<td>advection of salinity on/off flag</td>
</tr>
<tr>
<td>saltAdvScheme</td>
<td>PARM01</td>
<td>2</td>
<td>salinity horizontal advection scheme selector (see Table 2.2)</td>
</tr>
<tr>
<td>saltVertAdvScheme</td>
<td>PARM01</td>
<td>saltAdvScheme</td>
<td>salinity vertical advection scheme selector (see Table 2.2)</td>
</tr>
<tr>
<td>saltImplVertAdv</td>
<td>PARM01</td>
<td>FALSE</td>
<td>salinity implicit vertical advection on/off flag</td>
</tr>
<tr>
<td>salt_stayPositive</td>
<td>PARM01</td>
<td>FALSE</td>
<td>use Smolarkiewicz hack to ensure salinity stays positive on/off flag; requires #define GAD_SMOLARKIEWICZ_HACK in pkg/generic_advdiff/GAD_OPTIONS.h</td>
</tr>
<tr>
<td>implicitDiffusion</td>
<td>PARM01</td>
<td>FALSE</td>
<td>implicit vertical diffusion on/off flag</td>
</tr>
<tr>
<td>interDiffKr_pCell</td>
<td>PARM04</td>
<td>FALSE</td>
<td>account for partial-cell in interior vertical diffusion on/off flag</td>
</tr>
<tr>
<td>linFSConserveTr</td>
<td>PARM01</td>
<td>TRUE</td>
<td>correct source/sink of tracer due to use of linear free surface on/off flag</td>
</tr>
<tr>
<td>doAB_onGtGs</td>
<td>PARM03</td>
<td>TRUE</td>
<td>apply Adams-Bashforth on tendencies (rather than on T,S) on/off flag</td>
</tr>
</tbody>
</table>
3.8.5.2 Initialization

The initial tracer data can be contained in the binary files `hydrogThetaFile` and `hydrogSaltFile`. These files should contain 3-D data ordered in an \((x, y, r)\) fashion with \(k=1\) as the first vertical level. If no file names are provided, the tracers are then initialized with the values of \(tRef\) and \(sRef\) discussed in Section 3.8.3. In this case, the initial tracer data are uniform in \(x\) and \(y\) for each depth level.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Group</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hydrogThetaFile</td>
<td>PARM05</td>
<td>' '</td>
<td>filename for 3D specification of initial potential temperature (°C)</td>
</tr>
<tr>
<td>hydrogSaltFile</td>
<td>PARM05</td>
<td>' '</td>
<td>filename for 3D specification of initial salinity (g/kg)</td>
</tr>
<tr>
<td>maskIniTemp</td>
<td>PARM05</td>
<td>TRUE</td>
<td>apply (center-point) mask to initial hydrographic theta data on/off flag</td>
</tr>
<tr>
<td>maskIniSalt</td>
<td>PARM05</td>
<td>TRUE</td>
<td>apply (center-point) mask to initial hydrographic salinity on/off flag</td>
</tr>
<tr>
<td>checkIniTemp</td>
<td>PARM05</td>
<td>TRUE</td>
<td>check if initial theta (at wet-point) identically zero on/off flag</td>
</tr>
<tr>
<td>checkIniSalt</td>
<td>PARM05</td>
<td>TRUE</td>
<td>check if initial salinity (at wet-point) identically zero on/off flag</td>
</tr>
</tbody>
</table>

3.8.5.3 Tracer Diffusivities

Lateral eddy diffusivities for temperature and salinity/specific humidity are specified through the variables `diffKhT` and `diffKhS` (in \(m^2/s\)). Vertical eddy diffusivities are specified through the variables `diffKrT` and `diffKrS`. In addition, biharmonic diffusivities can be specified as well through the coefficients `diffK4T` and `diffK4S` (in \(m^4/s\)). The Gent and McWilliams parameterization for advection and mixing of oceanic tracers is described in Section 8.4.1.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Group</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>diffKhT</td>
<td>PARM01</td>
<td>0.0</td>
<td>Laplacian diffusivity of heat laterally ((m^2/s))</td>
</tr>
<tr>
<td>diffK4T</td>
<td>PARM01</td>
<td>0.0</td>
<td>biharmonic diffusivity of heat laterally ((m^4/s))</td>
</tr>
<tr>
<td>diffKrT</td>
<td>PARM01</td>
<td>0.0</td>
<td>Laplacian diffusivity of heat vertically ((m^2/s))</td>
</tr>
<tr>
<td>diffK4T</td>
<td>PARM01</td>
<td>0.0</td>
<td>biharmonic diffusivity of heat vertically ((m^4/s))</td>
</tr>
<tr>
<td>diffKrNrT</td>
<td>PARM01</td>
<td>0.0 at (k=)top vertical profile of vertical diffusivity of temperature ((m^2/s))</td>
<td></td>
</tr>
<tr>
<td>pCellMix_diffKr</td>
<td>PARM04</td>
<td>diffKrNr</td>
<td>vertical diffusivity for too thin partial-cell ((r^2/s))</td>
</tr>
<tr>
<td>diffKhS</td>
<td>PARM01</td>
<td>0.0</td>
<td>Laplacian diffusivity of salt laterally ((m^2/s))</td>
</tr>
<tr>
<td>diffK4S</td>
<td>PARM01</td>
<td>0.0</td>
<td>biharmonic diffusivity of salt laterally ((m^4/s))</td>
</tr>
<tr>
<td>diffKrS</td>
<td>PARM01</td>
<td>0.0</td>
<td>Laplacian diffusivity of salt vertically ((m^2/s))</td>
</tr>
<tr>
<td>diffK4S</td>
<td>PARM01</td>
<td>0.0</td>
<td>biharmonic diffusivity of salt vertically ((m^4/s))</td>
</tr>
<tr>
<td>diffKrNrS</td>
<td>PARM01</td>
<td>0.0 at (k=)top vertical profile of vertical diffusivity of salt ((m^2/s))</td>
<td></td>
</tr>
<tr>
<td>diffKrFile</td>
<td>PARM05</td>
<td>' '</td>
<td>filename for 3D specification of vertical diffusivity ((m^2/s)); requires #define ALLOW_3D_DIFFKR</td>
</tr>
<tr>
<td>diffKrBL79surf</td>
<td>PARM01</td>
<td>0.0</td>
<td>surface diffusivity for Bryan &amp; Lewis 1979 [BL79] ((m^2/s))</td>
</tr>
<tr>
<td>diffKrBL79deep</td>
<td>PARM01</td>
<td>0.0</td>
<td>deep diffusivity for Bryan &amp; Lewis 1979 [BL79] ((m^2/s))</td>
</tr>
<tr>
<td>diffKrBL79scl</td>
<td>PARM01</td>
<td>2.0E+02</td>
<td>depth scale for Bryan &amp; Lewis 1979 [BL79] (m)</td>
</tr>
<tr>
<td>diffKrBL79Ho</td>
<td>PARM01</td>
<td>-2.0E+03</td>
<td>turning depth for Bryan &amp; Lewis 1979 [BL79] (m)</td>
</tr>
<tr>
<td>diffKrBLEQsurf</td>
<td>PARM01</td>
<td>0.0</td>
<td>same as <code>diffKrBL79surf</code> but at equator; requires #define ALLOW_BL79_LAT_VARY</td>
</tr>
<tr>
<td>diffKrBLEQdeep</td>
<td>PARM01</td>
<td>0.0</td>
<td>same as <code>diffKrBL79deep</code> but at equator; requires #define ALLOW_BL79_LAT_VARY</td>
</tr>
<tr>
<td>diffKrBLEQscl</td>
<td>PARM01</td>
<td>2.0E+02</td>
<td>same as <code>diffKrBL79scl</code> but at equator; requires #define ALLOW_BL79_LAT_VARY</td>
</tr>
<tr>
<td>diffKrBLEQHo</td>
<td>PARM01</td>
<td>-2.0E+03</td>
<td>same as <code>diffKrBL79Ho</code> but at equator; requires #define ALLOW_BL79_LAT_VARY</td>
</tr>
</tbody>
</table>

continues on next page
Table 3.11 – continued from previous page

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Group</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BL79LatVary</td>
<td>PARM01</td>
<td>3.0E+01</td>
<td>transition from diffKrBLEQ to diffKrBL79 parms at this latitude; requires #define ALLOW_BL79_LAT_VARY</td>
</tr>
</tbody>
</table>

3.8.5.4 Ocean Convection

In addition to specific packages that parameterize ocean convection, two main model options are available. To use the first option, a convective adjustment scheme, you need to set the variable cadjFreq, the frequency (in seconds) with which the adjustment algorithm is called, to a non-zero value (note, if cadjFreq set to a negative value by the user, the model will set it to the model clock time step). The second option is to parameterize convection with implicit vertical diffusion. To do this, set the logical variable implicitDiffusion to .TRUE., and the real variable ivdc_kappa (in m^2/s) to an appropriate tracer vertical diffusivity value for mixing due to static instabilities (typically, several orders of magnitude above the background vertical diffusivity). Note that cadjFreq and ivdc_kappa cannot both have non-zero value.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Group</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ivdc_kappa</td>
<td>PARM01</td>
<td>0.0</td>
<td>implicit vertical diffusivity for convection (m^2/s)</td>
</tr>
<tr>
<td>cAdjFreq</td>
<td>PARM03</td>
<td>0</td>
<td>frequency of convective adj. scheme; &lt;0: sets value to deltaTclock (s)</td>
</tr>
<tr>
<td>hMixCriteria</td>
<td>PARM01</td>
<td>-0.8E+00</td>
<td>• &lt;0: specifies ΔT (°C) to define ML depth where Δρ = ΔT * dρ/dT occurs</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• &gt;1: define ML depth where local strat. exceeds mean strat. by this factor (non-dim.)</td>
</tr>
<tr>
<td>hMixSmooth</td>
<td>PARM01</td>
<td>0.0</td>
<td>use this fraction of neighboring points (for smoothing) in ML calculation (0-1; 0: no smoothing)</td>
</tr>
</tbody>
</table>

3.8.6 Parameters: Model Forcing

The forcing options that can be prescribed through runtime parameters in data are easy to use but somewhat limited in scope. More complex forcing setups are possible with optional packages such as pkg/exf or pkg/rbcs, in which case most or all of the parameters in this section can simply be left at their default value.

3.8.6.1 Momentum Forcing

This section only applies to the ocean. You need to generate wind-stress data into two files zonalWindFile and meridWindFile corresponding to the zonal and meridional components of the wind stress, respectively (if you want the stress to be along the direction of only one of the model horizontal axes, you only need to generate one file). The format of the files is similar to the bathymetry file. The zonal (meridional) stress data are assumed to be in pascals and located at U-points (V-points). See the MATLAB program gendata.m in the input directories of verification for several tutorial example (e.g. gendata.m in the barotropic gyre tutorial) to see how simple analytical wind forcing data are generated for the case study experiments.
### 3.8.6.2 Tracer Forcing

A combination of flux data and relaxation terms can be used for driving the tracer equations. For potential temperature, heat flux data (in W/m²) can be stored in the 2-D binary file `surfQnetFile`. Alternatively or in addition, the forcing can be specified through a relaxation term. The SST data to which the model surface temperatures are restored are stored in the 2-D binary file `ThetaClimFile`. The corresponding relaxation time scale coefficient is set through the variable `tauThetaClimRelax` (in seconds). The same procedure applies for salinity with the variable names `EmPmRfile`, `saltClimFile`, and `tauSaltClimRelax` for freshwater flux (in m/s) and surface salinity (in g/kg) data files and relaxation timescale coefficient (in seconds), respectively.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Group</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tempForcing</td>
<td>PARM01</td>
<td>TRUE</td>
<td>external forcing of temperature forcing on/off flag</td>
</tr>
<tr>
<td>surfQnetFile</td>
<td>PARM05</td>
<td>' '</td>
<td>filename for 2D specification of net total heat flux (W/m²)</td>
</tr>
<tr>
<td>surfQswFile</td>
<td>PARM05</td>
<td>' '</td>
<td>filename for 2D specification of net shortwave flux (W/m²); requires #define SHORTWAVE_HEATING</td>
</tr>
<tr>
<td>tauThetaClimRelax</td>
<td>PARM03</td>
<td>0.0</td>
<td>temperature (surface) relaxation time scale (s)</td>
</tr>
<tr>
<td>lambdaThetaFile</td>
<td>PARM05</td>
<td>' '</td>
<td>filename for 2D specification of inverse temperature (surface) relaxation time scale (1/s)</td>
</tr>
<tr>
<td>ThetaClimFile</td>
<td>PARM05</td>
<td>' '</td>
<td>filename for specification of (surface) temperature relaxation values (°C)</td>
</tr>
<tr>
<td>balanceThetaClimRelax</td>
<td>PARM01</td>
<td>FALSE</td>
<td>subtract global mean heat flux due to temp. relaxation flux every time step on/off flag; requires #define ALLOW_BALANCE_RELAX</td>
</tr>
<tr>
<td>balanceQnet</td>
<td>PARM01</td>
<td>FALSE</td>
<td>subtract global mean Qnet every time step on/off flag; requires #define ALLOW_BALANCE_FLUXES</td>
</tr>
<tr>
<td>geothermalFile</td>
<td>PARM05</td>
<td>' '</td>
<td>filename for 2D specification of geothermal heating flux through bottom (W/m²); requires #define ALLOW_GEOTHERMAL_FLUX</td>
</tr>
<tr>
<td>temp_EvPrRn</td>
<td>PARM01</td>
<td>UNSET</td>
<td>temperature of rain and evaporated water (unset, use local temp.) (°C)</td>
</tr>
<tr>
<td>allowFreezing</td>
<td>PARM01</td>
<td>FALSE</td>
<td>limit (ocean) temperature at surface to &gt;= -1.9°C</td>
</tr>
<tr>
<td>saltForcing</td>
<td>PARM01</td>
<td>TRUE</td>
<td>external forcing of salinity forcing on/off flag</td>
</tr>
<tr>
<td>convertFW2Salt</td>
<td>PARM01</td>
<td>3.5E+01</td>
<td>salinity used to convert freshwater flux to salt flux (-1: use local S) (g/kg) (note default is -1 if useRealFreshWaterFlux= .TRUE.)</td>
</tr>
<tr>
<td>rhoConstFresh</td>
<td>PARM01</td>
<td>rhoConst</td>
<td>constant reference density for fresh water (rain) (kg/m³)</td>
</tr>
<tr>
<td>EmPmRFile</td>
<td>PARM05</td>
<td>' '</td>
<td>filename for 2D specification of net freshwater flux (m/s)</td>
</tr>
</tbody>
</table>

continues on next page
### 3.8.6.3 Periodic Forcing

To prescribe time-dependent periodic forcing, concatenate successive time records into a single file ordered in a $(x, y, t)$ fashion and set the following variables: `periodicExternalForcing` to `.TRUE.`, `externForcingPeriod` to the period (in seconds between two records in input files) with which the forcing varies (e.g., 1 month), and `externForcingCycle` to the repeat time (in seconds) of the forcing (e.g., 1 year; note `externForcingCycle` must be a multiple of `externForcingPeriod`). With these variables specified, the model will interpolate the forcing linearly at each iteration.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Group</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>periodicExternalForcing</td>
<td>PARM03</td>
<td>FALSE</td>
<td>allow time-dependent periodic forcing on/off flag</td>
</tr>
<tr>
<td>externForcingPeriod</td>
<td>PARM03</td>
<td>0.0</td>
<td>period over which forcing varies (e.g. monthly) (s)</td>
</tr>
<tr>
<td>externForcingCycle</td>
<td>PARM03</td>
<td>0.0</td>
<td>period over which the forcing cycle repeats (e.g. one year) (s)</td>
</tr>
</tbody>
</table>

---

**Table 3.12 – continued from previous page**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Group</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>saltFluxFile</td>
<td>PARM05</td>
<td>' '</td>
<td>filename for 2D specification of salt flux (from seaice) ((g/kg) \cdot kg/m^2/s)</td>
</tr>
<tr>
<td>tauSaltClimRelax</td>
<td>PARM03</td>
<td>0.0</td>
<td>salinity (surface) relaxation time scale (s)</td>
</tr>
<tr>
<td>lambdaSaltFile</td>
<td>PARM05</td>
<td>' '</td>
<td>filename for 2D specification of inverse salinity (surface) relaxation time scale (1/s)</td>
</tr>
<tr>
<td>saltClimFile</td>
<td>PARM05</td>
<td>' '</td>
<td>filename for specification of (surface) salinity relaxation values ((g/kg))</td>
</tr>
<tr>
<td>balanceSaltClimRelax</td>
<td>PARM01</td>
<td>FALSE</td>
<td>subtract global mean flux due to salt relaxation every time step on/off flag</td>
</tr>
<tr>
<td>balanceEmPmR</td>
<td>PARM01</td>
<td>FALSE</td>
<td>subtract global mean EmPmR every time step on/off flag; requires #define ALLOW_BALANCE_FLUXES</td>
</tr>
<tr>
<td>salt_EvPrRn</td>
<td>PARM01</td>
<td>0.0</td>
<td>salinity of rain and evaporated water ((g/kg))</td>
</tr>
<tr>
<td>selectAddFluid</td>
<td>PARM01</td>
<td>0</td>
<td>add fluid to ocean interior options ((-1, 0: \text{off}, or } 1); requires #define ALLOW_ADDFLUID</td>
</tr>
<tr>
<td>temp_addMass</td>
<td>PARM01</td>
<td>temp_EvPrRn</td>
<td>temp. of added or removed (interior) water ((^\circ C)); requires #define ALLOW_ADDFLUID</td>
</tr>
<tr>
<td>salt_addMass</td>
<td>PARM01</td>
<td>salt_EvPrRn</td>
<td>salinity of added or removed (interior) water ((^\circ C)); requires #define ALLOW_ADDFLUID</td>
</tr>
<tr>
<td>addMassFile</td>
<td>PARM05</td>
<td>' '</td>
<td>filename for 3D specification of mass source/sink (+=source, kg/s); requires #define ALLOW_ADDFLUID</td>
</tr>
<tr>
<td>balancePrintMean</td>
<td>PARM01</td>
<td>FALSE</td>
<td>print subtracted balancing means to STDOUT on/off flag; requires #define ALLOW_BALANCE_FLUXES and/or #define ALLOW_BALANCE_RELAX</td>
</tr>
<tr>
<td>latBandClimRelax</td>
<td>PARM03</td>
<td>whole domain</td>
<td>relaxation to ((T,S)) climatology equatorward of this latitude band is applied</td>
</tr>
<tr>
<td>tracForcingOutAB</td>
<td>PARM03</td>
<td>0</td>
<td>1: take (T, S, ) and pTracer forcing out of Adams-Bashforth time stepping</td>
</tr>
</tbody>
</table>

---
3.8.7 Parameters: Simulation Controls

3.8.7.1 Run Start and Duration

The beginning of a simulation is set by specifying a start time (in seconds) through the real variable startTime or by specifying an initial iteration number through the integer variable nIter0. If these variables are set to non-zero values, the model will look for a "pickup" file (by default, pickup.0000nIter0) to restart the integration. The end of a simulation is set through the real variable endTime (in seconds). Alternatively, one can instead specify the number of time steps to execute through the integer variable nTimeSteps. Iterations are referenced to deltaTClock, i.e., each iteration is deltaTClock seconds of model time.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Group</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nIter0</td>
<td>PARM03</td>
<td>0</td>
<td>starting timestep iteration number for this integration</td>
</tr>
<tr>
<td>nTimeSteps</td>
<td>PARM03</td>
<td>0</td>
<td>number of (model clock) timesteps to execute</td>
</tr>
<tr>
<td>nEndIter</td>
<td>PARM03</td>
<td>0</td>
<td>run ending timestep iteration number (alternate way to prescribe nTimeSteps)</td>
</tr>
<tr>
<td>startTime</td>
<td>PARM03</td>
<td>baseTime</td>
<td>run start time for this integration (s) (alternate way to prescribe nIter0)</td>
</tr>
<tr>
<td>endTime</td>
<td>PARM03</td>
<td>0.0</td>
<td>run ending time (s) (with startTime, alternate way to prescribe nTimeSteps)</td>
</tr>
</tbody>
</table>

3.8.7.2 Input/Output Files

The precision with which to read binary data is controlled by the integer variable readBinaryPrec, which can take the value 32 (single precision) or 64 (double precision). Similarly, the precision with which to write binary data is controlled by the integer variable writeBinaryPrec. By default, MITgcm writes output (snapshots, diagnostics, and pickups) separately for individual tiles, leaving it to the user to reassemble these into global files, if needed (scripts are available in utils/). There are two options however to have the model do this for you. Setting globalFiles to .TRUE. should always work in a single process setup (including multi-threaded processes), but for MPI runs this will depend on the platform – it requires simultaneous write access to a common file (permissible in typical Lustre setups, but not on all filesystems). Alternatively, one can set useSingleCpuIO to .TRUE. to generate global files, which should always work, but requires additional mpi-passing of data and may result in slower execution.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Group</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>globalFiles</td>
<td>PARM01</td>
<td>FALSE</td>
<td>write output “global” (i.e. not per tile) files on/off flag</td>
</tr>
<tr>
<td>useSingleCpuIO</td>
<td>PARM01</td>
<td>FALSE</td>
<td>only master MPI process does I/O (producing global output files)</td>
</tr>
<tr>
<td>the_run_name</td>
<td>PARM05</td>
<td>' '</td>
<td>string identifying the name of the model “run” for meta files</td>
</tr>
<tr>
<td>readBinaryPrec</td>
<td>PARM01</td>
<td>32</td>
<td>precision used for reading binary files (32 or 64)</td>
</tr>
<tr>
<td>writeBinaryPrec</td>
<td>PARM01</td>
<td>32</td>
<td>precision used for writing binary files (32 or 64)</td>
</tr>
<tr>
<td>outputTypesInclusive</td>
<td>PARM03</td>
<td>FALSE</td>
<td>allows writing of output files in multiple formats (i.e. pkg/mdsio and pkg/mnc)</td>
</tr>
<tr>
<td>rwSuffixType</td>
<td>PARM03</td>
<td>0</td>
<td>controls the format of the pkg/mdsio binary file “suffix”</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• 0: use iteration number (myIter, I10.10)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• 1: 100*myTime</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• 2: myTime</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• 3: myTime/360</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• 4: myTime/3600</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>where myTime is model time in seconds</td>
</tr>
<tr>
<td>mdsioLocalDir</td>
<td>PARM05</td>
<td>' '</td>
<td>if not blank, read-write output tiled files from/to this directory name (+four-digit processor-rank code)</td>
</tr>
</tbody>
</table>
3.8.7.3 Frequency/Amount of Output

The frequency (in seconds) with which output is written to disk needs to be specified. `dumpFreq` controls the frequency with which the instantaneous state of the model is written. `monitorFreq` controls the frequency with which monitor output is dumped to the standard output file(s). The frequency of output is referenced to `deltaTClock`.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Group</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>dumpFreq</code></td>
<td>PARM03</td>
<td>0.0</td>
<td>interval to write model state/snapshot data (s)</td>
</tr>
<tr>
<td><code>dumpInitAndLast</code></td>
<td>PARM03</td>
<td>TRUE</td>
<td>write out initial and last iteration model state on/off flag</td>
</tr>
<tr>
<td><code>diagFreq</code></td>
<td>PARM03</td>
<td>0.0</td>
<td>interval to write additional intermediate (debugging cg2d/3d) output (s)</td>
</tr>
<tr>
<td><code>monitorFreq</code></td>
<td>PARM03</td>
<td>lowest of other output *Freq parms</td>
<td></td>
</tr>
<tr>
<td><code>monitorSelect</code></td>
<td>PARM03</td>
<td>2 (3 if fluid is water)</td>
<td></td>
</tr>
<tr>
<td><code>debugLevel</code></td>
<td>PARM01</td>
<td>depends on debug-Mode</td>
<td></td>
</tr>
<tr>
<td><code>plotLevel</code></td>
<td>PARM01</td>
<td><code>debugLevel</code></td>
<td></td>
</tr>
</tbody>
</table>

3.8.7.4 Restart/Pickup Files

`chkPtFreq` and `pchkPtFreq` control the output frequency of rolling and permanent pickup (a.k.a. checkpoint) files, respectively. These frequencies are referenced to `deltaTClock`.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Group</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>pChkPtFreq</code></td>
<td>PARM03</td>
<td>0.0</td>
<td>permanent restart/pickup checkpoint file write interval (s)</td>
</tr>
<tr>
<td><code>chkPtFreq</code></td>
<td>PARM03</td>
<td>0.0</td>
<td>rolling restart/pickup checkpoint file write interval (s)</td>
</tr>
<tr>
<td><code>pickupSuff</code></td>
<td>PARM03</td>
<td>' '</td>
<td>force run to use pickups (even if <code>nIter0 =0</code>) and read files with this suffix (10 char. max)</td>
</tr>
<tr>
<td><code>pickupStrictlyMatch</code></td>
<td>PARM03</td>
<td>TRUE</td>
<td>force pickup (meta) file formats to exactly match (or terminate with error) on/off flag</td>
</tr>
<tr>
<td><code>writePickupAtEnd</code></td>
<td>PARM03</td>
<td>FALSE</td>
<td>write a (rolling) pickup file at run completion on/off flag</td>
</tr>
<tr>
<td><code>usePickupBeforeC54</code></td>
<td>PARM01</td>
<td>FALSE</td>
<td>initialize run using old pickup format from code prior to checkpoint54a</td>
</tr>
<tr>
<td><code>startFromPickupAB2</code></td>
<td>PARM03</td>
<td>FALSE</td>
<td>using Adams-Bashforth-3, start using Adams-Bashforth-2 pickup format; requires #define ALLOW_ADAMSBASHFORTH_3</td>
</tr>
</tbody>
</table>

3.8.8 Parameters Used In Optional Packages

Some optional packages were not written with package-specific namelist parameters in a `data.${pkg}` file; or for historical and/or other reasons, several package-specific namelist parameters remain in `data`. 

3.8. Customizing the Model Configuration - Runtime Parameters
3.8.8.1 C-D Scheme

If you run at a sufficiently coarse resolution, you might choose to enable the C-D scheme for the computation of the Coriolis terms. The variable tauCD, which represents the C-D scheme coupling timescale (in seconds) needs to be set.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Group</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>useCDscheme</td>
<td>PARM01</td>
<td>FALSE</td>
<td>use C-D scheme for Coriolis terms on/off flag</td>
</tr>
<tr>
<td>tauCD</td>
<td>PARM03</td>
<td>deltaTmom</td>
<td>C-D scheme coupling timescale (s)</td>
</tr>
<tr>
<td>rCD</td>
<td>PARM03</td>
<td>1 - deltaTmom/tauCD</td>
<td>C-D scheme normalized coupling parameter (non-dim.)</td>
</tr>
<tr>
<td>epsAB_CD</td>
<td>PARM03</td>
<td>abEps</td>
<td>Adams-Bashforth-2 stabilizing weight used in C-D scheme</td>
</tr>
</tbody>
</table>

3.8.8.2 Automatic Differentiation

(package pkg/autodiff; see Section 7)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Group</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nTimeSteps_l2</td>
<td>PARM03</td>
<td>4</td>
<td>number of inner timesteps to execute per timestep</td>
</tr>
<tr>
<td>adjdumpFreq</td>
<td>PARM03</td>
<td>0.0</td>
<td>interval to write model state/snapshot data adjoint run (s)</td>
</tr>
<tr>
<td>adjMonitorFreq</td>
<td>PARM03</td>
<td>0.0</td>
<td>interval to write monitor output adjoint run (s)</td>
</tr>
<tr>
<td>adTapeDir</td>
<td>PARM05</td>
<td>' '</td>
<td>if not blank, read-write checkpointing files from/to this directory name</td>
</tr>
</tbody>
</table>

3.8.9 Execution Environment Parameters

If running multi-threaded (i.e., using shared memory/OpenMP), you will need to set nTn and/or nTy so that nTn*nTy is the total number of threads (per process).

The parameter useCubedSphereExchange needs to be changed to .TRUE. if you are using any type of grid composed of interconnected individual faces, including the cubed sphere topology or a lat-lon cap grid. See (needs section to be written).

Note that setting flag debugMode to .TRUE. activates a separate set of debugging print statements than parameter debugLevel (see Section 3.8.7.3). The latter controls print statements that monitor model activity (such as opening files, etc.), whereas the former produces a more coding-oriented set of print statements (e.g., entering and exiting subroutines, etc.)
### 3.9 MITgcm Input Data File Format

MITgcm input files for grid-related data (e.g., `delXFile`), forcing fields (e.g., `tauThetaClimRelax`), parameter fields (e.g., `viscAhZfile`), etc. are assumed to be in "flat" or "unblocked" binary format. For historical reasons, MITgcm files use big-endian byte ordering, NOT little-endian which is the more common default for today’s computers. Thus, some care is required to create MITgcm-readable input files.

- Using MATLAB: When writing binary files, MATLAB’s `fopen` command includes a MACHINEFORMAT option `'b'` which instructs MATLAB to read or write using big-endian byte ordering. 2-D arrays should be index-ordered in MATLAB as `(x, y)` and 3-D arrays as `(x, y, z); data is ordered from low to high in each index, with `x` varying most rapidly.

An example to create a bathymetry file of single-precision, floating point values (from tutorial *Barotropic Ocean Gyre*, a simple enclosed, flat-bottom domain) is as follows:

```matlab
ieee = 'b'; % big-endian format
accuracy = 'float32'; % this is single-precision (='real*4')

Ho=5000; % ocean depth in meters
nx=62; % number of gridpoints in x-direction
ny=62; % number of gridpoints in y-direction

% Flat bottom at z = -Ho
h = -Ho * ones(nx, ny);

% Walls (surrounding domain)
h([1 end], :) = 0; % set ocean depth to zero at east and west walls
h(:, [1 end]) = 0; % set ocean depth to zero at south and north walls

% save as single-precision (float32) with big-endian byte ordering
fid = fopen('bathy.bin', 'w', ieee);
fwrite(fid, h, accuracy);
fclose(fid);
```

To read this bathymetry file back into MATLAB, reshaped back to `(nx, ny)`: 

### Parameter Table

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Group</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>useCubedSphereExchange</td>
<td>EEPARMS</td>
<td>FALSE</td>
<td>use cubed-sphere topology domain on/off flag</td>
</tr>
<tr>
<td>nTx</td>
<td>EEPARMS</td>
<td>1</td>
<td>number of threads in the <code>x</code> direction</td>
</tr>
<tr>
<td>nTy</td>
<td>EEPARMS</td>
<td>1</td>
<td>number of threads in the <code>y</code> direction</td>
</tr>
<tr>
<td>useCoupler</td>
<td>EEPARMS</td>
<td>FALSE</td>
<td>communicate with other model components through a coupler on/off flag</td>
</tr>
<tr>
<td>useSETRLSTK</td>
<td>EEPARMS</td>
<td>FALSE</td>
<td>call C routine to set environment stacksize to ‘unlimited’</td>
</tr>
<tr>
<td>useSIGREG</td>
<td>EEPARMS</td>
<td>FALSE</td>
<td>enable signal handler to receive signal to terminate run cleanly on/off flag</td>
</tr>
<tr>
<td>debugMode</td>
<td>EEPARMS</td>
<td>FALSE</td>
<td>print additional debugging messages; also “flush” STD-OUT file unit after each print</td>
</tr>
<tr>
<td>printMapIncludesZeros</td>
<td>EEPARMS</td>
<td>FALSE</td>
<td>text map plots of fields should ignore exact zero values on/off flag</td>
</tr>
<tr>
<td>maxLengthPrt1D</td>
<td>EEPARMS</td>
<td>65</td>
<td>maximum number of 1D array elements to print to standard output</td>
</tr>
</tbody>
</table>
```c
fid = fopen('bathy.bin', 'r', ieee);
h = reshape(fread(fid, inf, accuracy), nx, ny);
fclose(fid);
```

- **Using Python:**

  A python version of the above script to create a bathymetry file is as follows:

  ```python
  import numpy as np
  Ho = 5000 # ocean depth in meters
  nx = 62 # number of gridpoints in x-direction
  ny = 62 # number of gridpoints in y-direction
  
  # Flat bottom at z = -Ho
  h = -Ho * np.ones((ny, nx))

  # Walls (surrounding domain)
  h[:, [0,-1]] = 0 # set ocean depth to zero at east and west walls
  h[[0,-1], :] = 0 # set ocean depth to zero at south and north walls

  # save as single-precision (NumPy type float32) with big-endian byte ordering
  h.astype('>f4').tofile('bathy.bin')
  
  The dtype specification '>f4' above instructs Python to write the file with big-endian byte ordering (specifically, due to the '>') as single-precision real numbers (due to the 'f4' which is NumPy float32 or equivalently, Fortran real*4 format).

  To read this bathymetry file back into Python, reshaped back to (ny, nx):

  ```python
  h = np.fromfile('bathy.bin', '>f4').reshape(ny, nx)
  ```

  where again the dtype spec instructs Python to read a big-endian file of single-precision, floating point values.

  Note that 2-D and 3-D arrays should be index-ordered as (y, x) and (z, y, x), respectively, to be written in proper ordering for MITgcm.

  A more complicated example of using Python to generate input data is provided in verification/tutorial_baroclinic_gyre/input/gendata.py.

- **Using Fortran:** To create flat binary files in Fortran, open with syntax OPEN(..., ACCESS='DIRECT', ...) (i.e., NOT ACCESS='SEQUENTIAL' which includes additional metadata). By default Fortran will use the local computer system's native byte ordering for reading and writing binary files, which for most systems will be little-endian. One therefore has two options: after creating a binary file in Fortran, use MATLAB or Python (or some other utility) to read in and swap the bytes in the process of writing a new file; or, determine if your local Fortran has a compiler flag to control byte-ordering of binary files. Similar to MATLAB, 2-D and 3-D arrays in Fortran should be index-ordered as (x, y) and (x, y, z), respectively.

  Using NetCDF format for input files is only partially implemented at present in MITgcm, and use is thus discouraged. Input files are by default single-precision real numbers (32-bit, real*4), but can be switched to double precision by setting namelist parameter readBinaryPrec (PARM01 in file data) to a value of 64.
The full MITgcm distribution comes with a set of pre-configured numerical experiments. Some of these example experiments are tests of individual parts of the model code, but many are fully fledged numerical simulations. Full tutorials exist for a few of the examples, and are documented in sections Section 4.1 - Section 4.14. The other examples follow the same general structure as the tutorial examples, see below. All example experiments are located in subdirectories under the directory verification. A list of additional experiments (i.e., not documented as full tutorials), with brief description, is provided in Section 4.15 and Section 4.16.

Each example experiment directory has the following subdirectories:

- **code**: contains code specific to the example. At a minimum, this directory includes the following files:
  - `code/packages.conf`: declares the list of packages or package groups to be used. If not included, the default set of packages is located in `pkg/pkg_groups`. Package groups are simply convenient collections of commonly used packages which are defined in `pkg/pkg_groups` (see *Using MITgcm Packages*). Some packages may require other packages or may require their absence (that is, they are incompatible) and these package dependencies are listed in `pkg/pkg_depend`.
  - `code/SIZE.h`: declares the size of underlying computational grid. This file is compiled instead of the MITgcm repository version `model/inc/SIZE.h`.
  - The `code/` directory may include other files and subroutines specific to the experiment, i.e., containing changes from the standard repository version. For example, some experiments contains CPP header options files to enable or disable some parts of the code at compile time; the most common ones would be `model/inc/CPP_OPTIONS.h` for core model options and «PKG»_OPTIONS.h for individual packages.

- **input**: contains the input data files required to run the example. At a minimum, the `input` directory contains the following files:
  - `input/data`: this file, written as a namelist, specifies the main parameters for the experiment.
  - `input/data.pkg`: contains parameters relative to the packages used in the experiment.
  - `input/eedata`: this file contains “execution environment” data. This consists of a specification of the number of threads to use in $x$ and $y$. For multi-threaded execution, these will be set to numbers greater than 1.
  - Forcing and topography file(s), as well as files describing the initial state of the experiment and any other supporting data. Required support files vary from experiment to experiment, depending on the setup.

- **results**: this directory contains the output file `output.txt` produced by the simulation example. This file is useful for comparison with your own output when you run the experiment.

- **build**: this directory is initially empty and should be used to compile the model and generate the executable.

- **run**: this directory is initially empty and should be used to run the executable. From the (empty) run directory, link files from `input` using the command `ln -s ../input/* .`, then execute the file `../input/prepare_run` if it exists. If you are running one of the experiment variations, i.e., using `input.<OTHER>`, first
link files from input.«OTHER» (running ..../input.«OTHER»/prepare_run if it exists) and next link files from input (and run ..../input/prepare_run). Following this procedure, file links from input.«OTHER» will NOT be overwritten by identically named files in input.

4.1 Barotropic Ocean Gyre

(in directory verification/tutorial_barotropic_gyre/)

This example experiment demonstrates using the MITgcm to simulate a barotropic, wind-forced, ocean gyre circulation. The experiment is a numerical rendition of the gyre circulation problem described analytically by Stommel in 1948 [Sto48] and Munk in 1950 [Mun50], and numerically in Bryan (1963) [Bry63]. Note this tutorial assumes a basic familiarity with ocean dynamics and geophysical fluid dynamics; readers new to the field may which to consult one of the standard texts on these subjects, such as Vallis (2017) [Val17] or Cushman-Roisin and Beckers (2011) [CRB11].

In this experiment the model is configured to represent a rectangular enclosed box of fluid, \(1200 \times 1200\) km in lateral extent. The fluid depth \(H = 5\) km. The fluid is forced by a zonal wind stress, \(\tau_z\), that varies sinusoidally in the north-south direction and is constant in time. Topologically the grid is Cartesian and the Coriolis parameter \(f\) is defined according to a mid-latitudes beta-plane equation

\[
f(y) = f_0 + \beta y
\]

where \(y\) is the distance along the ‘north-south’ axis of the simulated domain. For this experiment \(f_0\) is set to \(10^{-4} \text{s}^{-1}\) and \(\beta = 10^{-11} \text{s}^{-1} \text{m}^{-1}\).

The sinusoidal wind-stress variations are defined according to

\[
\tau_x(y) = -\tau_0 \cos\left(\frac{\pi y}{L_y}\right)
\]

where \(L_y\) is the lateral domain extent and \(\tau_0\) is set to 0.1N m\(^{-2}\).

Figure 4.1 summarizes the configuration simulated.

4.1.1 Equations Solved

The model is configured in hydrostatic form (the MITgcm default). The implicit free surface form of the pressure equation described in Marshall et al. (1997) [MHPA97] is employed. A horizontal Laplacian operator \(\nabla_h^2\) provides viscous dissipation. The wind-stress momentum input is added to the momentum equation for the ‘zonal flow’, \(u\). This effectively yields an active set of equations for this configuration as follows:

\[
\frac{Du}{Dt} - f v + g \frac{\partial \eta}{\partial x} - A_h \nabla_h^2 u = \frac{\tau_x}{\rho_c H} \tag{4.1}
\]

\[
\frac{Dv}{Dt} + f u + g \frac{\partial \eta}{\partial y} - A_h \nabla_h^2 v = 0 \tag{4.2}
\]

\[
\frac{\partial \eta}{\partial t} + \nabla_h \cdot H \vec{u} = 0 \tag{4.3}
\]

where \(u\) and \(v\) are the \(x\) and \(y\) components of the flow vector \(\vec{u}\), \(\eta\) is the free surface height, \(A_h\) the horizontal Laplacian viscosity, \(\rho_c\) is the fluid density, and \(g\) the acceleration due to gravity.
4.1.2 Discrete Numerical Configuration

The domain is discretized with a uniform grid spacing in the horizontal set to $\Delta x = \Delta y = 20$ km, so that there are sixty ocean grid cells in the $x$ and $y$ directions. The numerical domain includes a border row of “land” cell surrounding the ocean cells, so the numerical grid size is $62 \times 62$ (if these land cells were not included, the domain would be periodic in both the $x$ and $y$ directions).

Vertically the model is configured using a single layer in depth, $\Delta z$, of 5000 m.

4.1.2.1 Numerical Stability Criteria

Let’s start with our choice for the model’s time step. To minimize the amount of required computational resources, typically one opts for as large a time step as possible while keeping the model solution stable. The advective Courant–Friedrichs–Lewy (CFL) condition (see Adcroft 1995 [Adc95]) for an extreme maximum horizontal flow speed is:

$$S_a = 2 \left( \frac{|u| \Delta t}{\Delta x} \right) < 0.5 \text{ for stability} \quad (4.4)$$

The 2 factor on the left is because we have a 2-D problem (in contrast with the more familiar 1-D canonical stability analysis); the right hand side is 0.5 due to our default use of Adams-Bashforth2 (see Section 2.5) rather than the more familiar value of 1 that one would obtain using a forward Euler scheme. In our configuration, let’s assume our solution will achieve a maximum $|u| = 1 \text{ ms}^{-1}$ (in reality, current speeds in our solution will be much smaller). To keep $\Delta t$ safely below the stability threshold, let’s choose $\Delta t = 1200$ s (= 20 minutes), which results in $S_a = 0.12$.

The numerical stability for inertial oscillations using Adams-Bashforth II (Adcroft 1995 [Adc95])

$$S_i = f \Delta t < 0.5 \text{ for stability} \quad (4.5)$$

evaluates to 0.12 for our choice of $\Delta t$, which is below the stability threshold.

There are two general rules in choosing a horizontal Laplacian eddy viscosity $A_h$:
• the resulting Munk layer width should be at least as large (preferably, larger) than the lateral grid spacing;
• the viscosity should be sufficiently small that the model is stable for horizontal friction, given the time step.

Let’s use this first rule to make our choice for \( A_h \), and check this value using the second rule. The theoretical Munk boundary layer width (as defined by the solution zero-crossing, see Pedlosky 1987 [Ped87]) is given by:

\[
M_w = \frac{2\pi}{\sqrt{3}} \left( \frac{A_h}{\beta} \right)^{\frac{1}{3}}
\] (4.6)

For our configuration we will choose to resolve a boundary layer of \( \approx 100 \) km, or roughly across five grid cells, so we set \( A_h = 400 \) m\(^2\) s\(^{-1}\) (more precisely, this sets the full width at \( M_w = 124 \) km). This choice ensures that the frictional boundary layer is well resolved.

Given our choice of \( \Delta t \), the stability parameter for the horizontal Laplacian friction (Adcroft 1995 [Adc95])

\[
S_l = 2 \left( \frac{4 A_h \Delta t}{\Delta x^2} \right) < 0.6 \text{ for stability}
\] (4.7)

evaluates to 0.0096, which is well below the stability threshold. As in (4.4) the above criteria is for a 2D problem using Adams-Bashforth2 time stepping, with the 0.6 value on the right replacing the more familiar 1 that is obtained using a forward Euler scheme.

See Section 2.5 for additional details on Adams-Bashforth time-stepping and numerical stability criteria.

### 4.1.3 Configuration

The model configuration for this experiment resides under the directory `verification/tutorial_barotropic_gyre/`. The experiment files

- `verification/tutorial_barotropic_gyre/code/SIZE.h`
- `verification/tutorial_barotropic_gyre/input/data`
- `verification/tutorial_barotropic_gyre/input/data.pkg`
- `verification/tutorial_barotropic_gyre/input/eedata`
- `verification/tutorial_barotropic_gyre/input/bathy.bin`
- `verification/tutorial_barotropic_gyre/input/windx_cosy.bin`

contain the code customizations and parameter settings for this experiment. Below we describe these customizations in detail.

Note: MITgcm’s defaults are configured to simulate an ocean rather than an atmosphere, with vertical \( z \)-coordinates. To model the ocean using pressure coordinates using MITgcm, additional parameter changes are required; see tutorial `ocean_in_p`. To switch parameters to model an atmosphere, see tutorial `Held_Suarez`.

#### 4.1.3.1 Compile-time Configuration

**File code/SIZE.h**
Listing 4.1: verification/tutorial_barotropic_gyre/code/SIZE.h

```c
CBOP
C !ROUTINE: SIZE.h
C !INTERFACE:
C include SIZE.h
C !DESCRIPTION: 
C *==========================================================*
C | SIZE.h Declare size of underlying computational grid.
C *==========================================================*
C |
C | The design here supports a three-dimensional model grid
C | with indices I,J and K. The three-dimensional domain
C | is comprised of nPx*nSx blocks (or tiles) of size sNx
C | along the first (left-most index) axis, nPy*nSy blocks
C | of size sNy along the second axis and one block of size
C | Nr along the vertical (third) axis.
C | Blocks/tiles have overlap regions of size OLx and OLy
C | along the dimensions that are subdivided.
C *==========================================================*
C 
C Voodoo numbers controlling data layout:
C sNx :: Number of X points in tile.
C sNy :: Number of Y points in tile.
C OLx :: Tile overlap extent in X.
C OLy :: Tile overlap extent in Y.
C nSx :: Number of tiles per process in X.
C nSy :: Number of tiles per process in Y.
C nPx :: Number of processes to use in X.
C nPy :: Number of processes to use in Y.
C Nx :: Number of points in X for the full domain.
C Ny :: Number of points in Y for the full domain.
C Nr :: Number of points in vertical direction.
CEOP
INTEGER sNx
INTEGER sNy
INTEGER OLx
INTEGER OLy
INTEGER nSx
INTEGER nSy
INTEGER nPx
INTEGER nPy
INTEGER Nx
INTEGER Ny
INTEGER Nr
PARAMETER ( 
  & sNx = 62,
  & sNy = 62,
  & OLx = 2,
  & OLy = 2,
  & nSx = 1,
  & nSy = 1,
  & nPx = 1,
(continues on next page)
```
Here we show a modified model/inc source code file, customizing MITgcm’s array sizes to our model domain. This file must be uniquely configured for any model setup; using the MITgcm default model/inc/SIZE.h will in fact cause a compilation error. Note that MITgcm’s storage arrays are allocated as static variables (hence their size must be declared in the source code), in contrast to some model codes which declare array sizes dynamically, i.e., through runtime (namelist) parameter settings.

For this first tutorial, our setup and run environment is the most simple possible: we run on a single process (i.e., NOT MPI and NOT multi-threaded) using a single model “tile.” For a more complete explanation of the parameter choices to use multiple tiles, see the tutorial Baroclinic Gyre.

- These lines set parameters sNx and sNy, the number of grid points in the x and y directions, respectively.

```
&  sNx = 62,
&  sNy = 62,
```

- These lines set parameters OLx and OLy in the x and y directions, respectively. These values are the overlap extent of a model tile, the purpose of which will be explained in later tutorials. Here, we simply specify the required minimum value (2) in both x and y.

```
&  OLx = 2,
&  OLy = 2,
```

- These lines set parameters nSx, nSy, nPx, and nPy, the number of model tiles and the number of processes in the x and y directions, respectively. As discussed above, in this tutorial we configure a single model tile on a single process, so these parameters are all set to the value one.

```
&  nSx = 1,
&  nSy = 1,
&  nPx = 1,
&  nPy = 1,
```

- This line sets parameter Nr, the number of points in the vertical dimension. Here we use just a single vertical level.

```
&  Nr = 1)
```

- Note these lines summarize the horizontal size of the model domain (NOT to be edited).

```
&  Nx = sNx*nSx*nPx,
&  Ny = sNy*nSy*nPy,
```
Further information and examples about how to configure `model/inc/SIZE.h` are given in Section 6.3.1.

### 4.1.3.2 Run-time Configuration

**File input/data**

Listing 4.2: verification/tutorial_barotropic_gyre/input/data

```plaintext
# Model parameters
# Continuous equation parameters
&PARM01
  viscAh=4.E2,
  f0=1.E-4,
  beta=1.E-11,
  rhoConst=1000.,
  gBaro=9.81,
  rigidLid=.FALSE.,
  implicitFreeSurface=.TRUE.,
# momAdvection=.FALSE.,
  tempStepping=.FALSE.,
  saltStepping=.FALSE.,
&

# Elliptic solver parameters
&PARM02
cg2dTargetResidual=1.E-7,
cg2dMaxIters=1000,
&

# Time stepping parameters
&PARM03
nIter0=0,
nTimeSteps=10,
deltaT=1200.0,
pChkptFreq=3110400.0,
chkptFreq=15552000.0,
dumpFreq=15552000.0,
monitorFreq=1200.,
monitorSelect=2,
#-for longer run (3.0 yr):
# nTimeSteps=77760,
# monitorFreq=864000.,
&

# Gridding parameters
&PARM04
usingCartesianGrid=.TRUE.,
de1X=62*20.E3,
de1Y=62*20.E3,
xgOrigin=-20.E3,
ygOrigin=-20.E3,
de1R=5000.,
```

(continues on next page)
# Input datasets

```fortran
&
# Input datasets
&PARM05
bathyFile='bathy.bin'
zonalWindFile='windx_cosy.bin',
#zonalWindFile='windx_siny.bin',
meridWindFile=,
&
```

This file, reproduced completely above, specifies the main parameters for the experiment. The parameters that are significant for this configuration (shown with line numbers to left) are as follows.

**PARM01 - Continuous equation parameters**

- This line sets parameter \( \text{viscAh} \), the horizontal Laplacian viscosity, to 400 m\(^2\) s\(^{-1}\).
  ```fortran
  viscAh=4.E2,
  ```

- These lines set \( f_0 \) and \( \beta \) (the Coriolis parameter \( f_0 \) and the gradient of the Coriolis parameter \( \beta \)) for our beta-plane to \( 1 \times 10^{-4} \) s\(^{-1}\) and \( 1 \times 10^{-11} \) m\(^{-1}\) s\(^{-1}\), respectively.
  ```fortran
  f0=1.E-4,
  beta=1.E-11,
  ```

- This line sets parameter \( \text{rhoConst} \), the Boussinesq reference density \( \rho_c \) in (4.1), to 1000 kg/m\(^3\).
  ```fortran
  rhoConst=1000.,
  ```

- This line sets parameter \( gBaro \), the acceleration due to gravity \( g \) (in the free surface terms in (4.1) and (4.2)), to 9.81 m/s\(^2\). This is the MITgcm default value, i.e., the value used if this line were not included in data. One might alter this parameter for a reduced gravity model, or to simulate a different planet, for example.
  ```fortran
  gBaro=9.81,
  ```

- These lines set parameters \( \text{rigidLid} \) and \( \text{implicitFreeSurface} \) in order to suppress the rigid lid formulation of the surface pressure inverter and activate the implicit free surface formulation.
  ```fortran
  rigidLid=.FALSE.,
  implicitFreeSurface=.TRUE.,
  ```

- This line sets parameter \( \text{momAdvection} \) to suppress the (non-linear) momentum of advection terms in the momentum equations. However, note the # in column 1: this “comments out” the line, so using the above data file verbatim will in fact include the momentum advection terms (i.e., MITgcm default for this parameter is TRUE). We’ll explore the linearized solution (i.e., by removing the leading #) in Section 4.1.5. Note the ability to comment out a line in a namelist file is not part of standard Fortran, but this feature is implemented for all MITgcm namelist files.
  ```fortran
  # momAdvection=.FALSE.,
  ```

- These lines set parameters \( \text{tempStepping} \) and \( \text{saltStepping} \) to suppress MITgcm’s forward time integration of temperature and salt in the tracer equations, as these prognostic variables are not relevant for the model solution in this configuration. By default, MITgcm solves equations governing these two (active) tracers; later tutorials will
demonstrate how additional passive tracers can be included in the solution. The advantage of NOT solving the temperature and salinity equations is to eliminate many unnecessary computations. In most typical configurations however, one will want the model to compute a solution for $T$ and $S$, which typically comprises the majority of MITgcm’s processing time.

```
12 tempStepping=.FALSE.,
13 saltStepping=.FALSE.,
```

**PARM02 - Elliptic solver parameters**

- The first line sets the tolerance (parameter cg2dTargetResidual) that the 2-D conjugate gradient solver, the iterative method used in the pressure method algorithm, will use to test for convergence. The second line sets parameter cg2dMaxIters, the maximum number of iterations. The solver will iterate until the residual falls below this target value (here, set to $1 \times 10^{-7}$) or until this maximum number of solver iterations is reached (here, set to a maximum 1000 iterations). Typically, the solver will converge in far fewer than 1000 iterations, but it does not hurt to allow for a large number. The chosen value for the target residual happens to be the MITgcm default, and will serve well in most model configurations.

```
18 cg2dTargetResidual=1.E-7,
19 cg2dMaxIters=1000,
```

**PARM03 - Time stepping parameters**

- This line sets the starting (integer) iteration number for the run. Here we set the value to zero, which starts the model from a new, initialized state. If nIter0 is non-zero, the model would require appropriate pickup files (i.e., restart files) in order to continue integration of an existing run.

```
24 nIter0=0,
```

- This line sets parameter nTimeSteps, the (integer) number of timesteps the model will integrate forward. Below, we have set this to integrate for just 10 time steps, for MITgcm automated testing purposes (Section 5.5). To integrate the solution to near steady state, uncomment the line further down where we set the value to 77760 time steps. When you make this change, be sure to also uncomment the next line that sets monitorFreq (see below).

```
25 nTimeSteps=10,
```

- This line sets parameter deltaT, the timestep used in stepping forward the model, to 1200 seconds. In combination with the larger value of nTimeSteps mentioned above, we have effectively set the model to integrate forward for $77760 \times 1200 \text{ s} = 3.0 \text{ years}$ (based on 360-day years), long enough for the solution to approach equilibrium.

```
26 deltaT=1200.0,
```

- These lines control the frequency at which restart (a.k.a. pickup) files are dumped by MITgcm. Here the value of pChkptFreq is set to 31,104,000 seconds (=1.0 years) of model time; this controls the frequency of “permanent” checkpoint pickup files. With permanent files, the model’s iteration number is part of the file name (as the filename “suffix”; see Section 4.1.4.2) in order to save it as a labelled, permanent, pickup state. The value of ChkptFreq is set to 15,552,000 seconds (=0.5 years); the pickup files written at this frequency but will NOT

---

4.1. Barotropic Ocean Gyre 149
include the iteration number in the filename, instead toggling between ckptA and ckptB in the filename, and thus these files will be overwritten with new data every 2 × 15,552,000 seconds. Temporary checkpoint files can be written more frequently without requiring additional disk space, for example to peruse (or re-run) the model state prior to an instability, or restart following a computer crash, etc. Either type of checkpoint file can be used to restart the model.

\[
\begin{align*}
pChkptFreq = & 31104000.0, \\
chkptFreq = & 15552000.0, \\
dumpFreq = & 15552000.0, \\
\end{align*}
\]

- This line sets parameter dumpFreq, frequency of writing model state snapshot diagnostics (of relevance in this setup: variables \(u\), \(v\), and \(\eta\)). Here, we opt for a snapshot of model state every 15,552,000 seconds (=0.5 years), or after every 12960 time steps of integration.

\[
\begin{align*}
\text{monitorFreq} = & 1200., \\
\text{monitorSelect} = & 2, \\
dumpFreq = & 15552000.0, \\
\end{align*}
\]

- These lines are set to dump monitor output (see Section 9.4) every 1200 seconds (i.e., every time step) to standard output. While this monitor frequency is needed for MITgcm automated testing, this is too much output for our tutorial run. Comment out this line and uncomment the line where monitorFreq is set to 864,000 seconds, i.e., output every 10 days. Parameter monitorSelect is set to 2 here to reduce output of non-applicable quantities for this simple example.

\[
\begin{align*}
\text{monitorFreq} = & 1200., \\
\text{monitorSelect} = & 2, \\
dumpFreq = & 15552000.0, \\
\end{align*}
\]

**PARM04 - Gridding parameters**

- This line sets parameter usingCartesianGrid, which specifies that the simulation will use a Cartesian coordinate system.

\[
\text{usingCartesianGrid} = .TRUE.,
\]

- These lines set the horizontal grid spacing of the model grid, as vectors \(\text{delX}\) and \(\text{delY}\) (i.e., \(\Delta x\) and \(\Delta y\) respectively). This syntax indicates that we specify 62 values in both the \(x\) and \(y\) directions, which matches the domain size as specified in SIZE.h. Grid spacing is set to \(20 \times 10^3 \text{ m} (=20 \text{ km})\).

\[
\begin{align*}
\text{delX} = & 62 \times 20. \text{E3}, \\
\text{delY} = & 62 \times 20. \text{E3}, \\
\end{align*}
\]

- The cartesian grid default origin is (0,0) so here we set the origin with parameters \(\text{xgOrigin}\) and \(\text{ygOrigin}\) to (-20000,-20000), accounting for the bordering solid wall. The centers of the grid boxes will thus be at -10 km, 10 km, 30 km, 50 km, ..., in both \(x\) and \(y\) directions.

\[
\begin{align*}
\text{xgOrigin} = & -20. \text{E3}, \\
\text{ygOrigin} = & -20. \text{E3}, \\
\end{align*}
\]

- This line sets parameter \(\text{delR}\), the vertical grid spacing in the \(z\)-coordinate (i.e., \(\Delta z\)), to 5000 m.

\[
\text{delR} = 5000.,
\]
PARM05 - Input datasets

- This line sets parameter `bathyFile`, the name of the bathymetry file. See Section 4.1.3.2 for information about the file format.

```plaintext
bathyFile='bathy.bin'
```

- These lines specify the names of the files from which the surface wind stress is read. There is a separate file for the $x$-direction (`zonalWindFile`) and the $y$-direction (`meridWindFile`). Note, here we have left the latter parameter blank, as there is no meridional wind stress forcing in our example.

```plaintext
zonalWindFile='windx_cosy.bin',
#zonalWindFile='windx_siny.bin',
meridWindFile=,
```

File input/data.pkg

Listing 4.3: verification/tutorial_barotropic_gyre/input/data.pkg

```plaintext
# Packages
&PACKAGES
&
```

This file does not set any namelist parameters, yet is necessary to run – only standard packages (i.e., those compiled in MITgcm by default) are required for this setup, so no other customization is necessary. We will demonstrate how to include additional packages in other tutorial experiments.

File input/eedata
This file uses standard default values (i.e., MITgcm default is single-threaded) and does not contain customizations for this experiment.

File input/bathy.bin

This file is a 2-D \((x, y)\) map of bottom bathymetry, specified as the \(z\)-coordinate of the solid bottom boundary. Here, the value is set to -5000 m everywhere except along the N, S, E, and W edges of the array, where the value is set to 0 (i.e., “land”). As discussed in Section 4.1.2, the domain in MITgcm is assumed doubly periodic (i.e., periodic in both \(x\)- and \(y\)-directions), so boundary walls are necessary to set up our enclosed box domain. The matlab program verification/tutorial_barotropic_gyre/input/gendata.m was used to generate this bathymetry file (alternatively, see python equivalent gendata.py). By default, this file is assumed to contain 32-bit (single precision) binary numbers. See Section 3.9 for additional information on MITgcm input data file format specifications.

File input/windx_cosy.bin

Similar to file input/bathy.bin, this file is a 2-D \((x, y)\) map of \(\tau_x\) wind stress values, formatted in the same manner. The units are \(\text{Nm}^{-2}\). Although \(\tau_x\) is only a function of \(y\) in this experiment, this file must still define a complete 2-D map in order to be compatible with the standard code for loading forcing fields in MITgcm. The matlab program verification/tutorial_barotropic_gyre/input/gendata.m was used to generate this wind stress file (alternatively, see python equivalent gendata.py). To run the barotropic jet variation of this tutorial example (see Figure 4.4), you will in fact need to run one of these programs to generate the file input/windx_siny.bin.
4.1.4.1 Standard output

Your run’s standard output file should be similar to verification/tutorial_barotropic_gyre/results/output.txt. The standard output is essentially a log file of the model run. The following information is included (in rough order):

- startup information including MITgcm checkpoint release number and other execution environment information, and a list of activated packages (including all default packages, as well as optional packages).
- the text from all data.* and other critical files (in our example here, eedata, SIZE.h, data, and data.pkg).
- information about the grid and bathymetry, including dumps of all grid variables (only if Cartesian or spherical polar coordinates used, as is the case here).
- all runtime parameter choices used by the model, including all model defaults as well as user-specified parameters.
- monitor statistics at regular intervals (as specified by parameter monitorFreq in data. See Section 9.4).
- output from the 2-D conjugate gradient solver. More specifically, statistics from the right-hand side of the elliptic equation – for our linear free-surface, see eq. (2.15) – are dumped for every model time step. If the model solution blows up, these statistics will increase to infinity, so one can see exactly when the problem occurred (i.e., to aid in debugging). Additional solver variables, such as number of iterations and residual, are included with the monitor statistics.
- a summary of end-of-run execution information, including user-, wall- and system-time elapsed during execution, and tile communication statistics. These statistics are provided for the overall run, and also broken down by time spent in various subroutines.

Different setups using non-standard packages and/or different parameter choices will include additional or different output as part of the standard output. It is also possible to select more or less output by changing the parameter debugLevel in data; see (missing doc for pkg debug).

STDERR.0000 - if errors (or warnings) occurred during the run, helpful warning and/or error message(s) would appear in this file.

4.1.4.2 Other output files

In addition to raw binary data files with .data extension, each binary file has a corresponding .meta file. These plain-text files include information about the array size, precision (i.e., float32 or float64), and if relevant, time information and/or a list of these fields included in the binary file. The .meta files are used by MITgcm utils when binary data are read.

The following output files are generated:

Grid Data: see Section 2.11 for definitions and description of the Arakawa C-grid staggering of model variables.

- XC, YC - grid cell center point locations
- XG, YG - locations of grid cell vertices
- RC, RF - vertical cell center and cell faces positions
- DXC, DYC - grid cell center point separations (Figure 2.8 b)
- DXG, DYG - separation of grid cell vertices (Figure 2.8 a)
• DRC, DRF - separation of vertical cell centers and faces, respectively

• RAC, RAS, RAW, RAZ - areas of the grid “tracer cells”, “southern cells”, “western cells” and “vorticity cells”, respectively (Figure 2.8)

• hFacC, hFacS, hFacW - fractions of the grid cell in the vertical which are “open” as defined in the center and on the southern and western boundaries, respectively. These variables effectively contain the configuration bathymetric (or topographic) information.

• Depth - bathymetry depths

All these files contain 2-D($x, y$) data except RC, RF, DRC, DRF, which are 1-D($z$), and hFacC, hFacS, hFacW, which contain 3D($x, y, z$) data. Units for the grid files depends on one’s choice of model grid; here, they are all in given in meters (or $m^2$ for areas).

All the 2-D grid data files contain .001.001 in their filename, e.g., DXC.001.001.data – this is the tile number in .XXX.YYY format. Here, we have just a single tile in both x and y, so both tile numbers are 001. Using multiple tiles, the default is that the local tile grid information would be output separately for each tile (as an example, see the baroclinic gyre tutorial, which is set up using multiple tiles), producing multiple files for each 2-D grid variable.

State Variable Snapshot Data:

Eta.0000000000.001.001.data, Eta.0000000000.001.001.meta - this is a binary data snapshot of model dynamic variable etaN (the free-surface height) and its meta file, respectively. Note the tile number is included in the filename, as is the iteration number 0000000000, which is simply the time step (the iteration number here is referred to as the “suffix” in MITgcm parlance; there are options to change this suffix to something other than iteration number). In other words, this is a dump of the free-surface height from the initialized state, iteration 0; if you load up this data file, you will see it is all zeroes. More interesting is the free-surface height after some time steps have occurred. Snapshots are written according to our parameter choice dumpFreq, here set to 15,552,000 seconds, which is every 12960 time steps. We will examine the model solutions in Section 4.1.5. The free-surface height is a 2-D($x, y$) field.

Snapshot files exist for other prognostic model variables, in particular filenames starting with U (uVel), V (uVel), T (theta), and S (salt); given our setup, these latter two fields remain uniform in space and time, thus not very interesting until we explore a baroclinic gyre setup in tutorial_baroclinic_gyre. These are all 3-D($x, y, z$) fields. The format for the file names is similar to the free-surface height files. Also dumped are snapshots of diagnosed vertical velocity W (wVel) (note that in non-hydrostatic simulations, W is a fully prognostic model variable).

Checkpoint Files:

The following pickup files are generated:

- pickup.0000025920.001.001.data, pickup.0000025920.001.001.meta, etc. - written at frequency set by pChkptFreq

- pickup.ckptA.001.001.data, pickup.ckptA.001.001.meta, pickup.ckptB.001.001.data, pickup.ckptB.001.001.meta - written at frequency set by ChkptFreq

Other Model Output Data: Model output related to reference density and hydrostatic pressure, in files Rhoref, PHrefC, PHrefF, PH, and PHL, is discussed in depth here in tutorial Baroclinic Ocean Gyre (as these data are not terribly interesting in this single-layer setup).
4.1.5 Model Solution

After running the model for 77,760 time steps (3.0 years), the solution is near equilibrium. Given an approximate timescale of one month for barotropic Rossby waves to cross our model domain, one might expect the solution to require several years to achieve an equilibrium state. The model solution of free-surface height $\eta$ (proportional to streamfunction) at $t = 3.0$ years is shown in Figure 4.2. For further details on this solution, particularly examining the effect of the non-linear terms with increasing Reynolds number, the reader is referred to Pedlosky (1987) [Ped87] section 5.11.

![MITgcm Barotropic Gyre Free-Surface Height (m)](image)

Figure 4.2: MITgcm solution to the barotropic gyre example after $t = 3.0$ years of model integration. Free surface height is shown in meters.

Using matlab for example, visualizing output using the `utils/matlab/rdmds.m` utility to load the binary data in `Eta.0000077760.001.001.data` is as simple as:

```matlab
addpath ../../../utils/matlab/
XC=rdmds('XC');
YC=rdmds('YC');
Eta=rdmds('Eta', 77760);
contourf(XC/1000, YC/1000, Eta, [-0.04:.01:.04])
colorbar
colormap((flipud(hot)))
set(gca, 'XLim', [0 1200])
set(gca, 'YLim', [0 1200])
```

or using python (you will need to install the MITgcmutils package, see Section 11.1):
from MITgcmutils import mds
import numpy as np
import matplotlib.pyplot as plt

XC = mds.rdmds('XC')
YC = mds.rdmds('YC')
Eta = mds.rdmds('Eta', 77760)

plt.contourf(XC/1000, YC/1000, Eta, np.linspace(-0.02, 0.05, 8), cmap='hot_r')
plt.colorbar()
plt.show()

(for a more involved example with detailed explanations how to read in model output, perform calculations using these data, and make plots, see tutorial Baroclinic Ocean Gyre)

Let's simplify the example by considering the linear problem where we neglect the advection of momentum terms. In other words, replace \( \frac{\partial u}{\partial t} \) and \( \frac{\partial v}{\partial t} \) with \( \frac{\partial u}{\partial t} \) and \( \frac{\partial v}{\partial t} \), respectively, in (4.1) and (4.2). To do so, we uncomment (i.e., remove the leading #) in the line # momAdvection=.FALSE., in file data and re-run the model. Any existing output files will be overwritten.

For the linearized equations, the Munk layer (equilibrium) analytical solution is given by:

\[
\eta(x,y) = \frac{\tau_o}{\rho g H} \cdot \left[ 1 - \frac{x}{L_x} \right] \cdot \frac{\pi}{L_y} \cdot \sin\left(\frac{\pi y}{L_y}\right) \cdot \left[ 1 - \exp\left(\frac{-x}{2\delta_m}\right) \cdot \left( \cos\left(\frac{\sqrt{3}x}{2\delta_m}\right) + \frac{1}{\sqrt{3}} \sin\left(\frac{\sqrt{3}x}{2\delta_m}\right) \right) \right]
\]

where \( \delta_m \equiv \left( \frac{2\beta}{\sqrt{3}} \right)^{1/3} \). Figure 4.3 displays the MITgcm output after switching off momentum advection vs. the analytical solution to the linearized equations. Success!

![MITgcm Barotropic Gyre Linearized Solution vs. Analytical Solution](image)

Figure 4.3: Comparison of free surface height for the near-equilibrium MITgcm solution (\( t = 3.0 \) years) with momentum advection switched off (left) and the analytical equilibrium solution to the linearized equation (right).

Finally, let's examine one additional simulation where we change the cosine profile of wind stress forcing to a sine profile. First, run the matlab script verification/tutorial_barotropic_gyre/input/gendata.m to generate the alternate sine profile wind stress, and place a copy in your run directory. Then, in file data, replace the line zonalWindFile='windx_cosy.bin', with zonalWindFile='windx_siny.bin'.
Figure 4.4: MITgcm equilibrium solution to the barotropic setup with alternate sine profile of wind stress forcing, producing a barotropic jet.
The free surface solution given this forcing is shown in Figure 4.4. Two “half gyres” are separated by a strong jet. We’ll look more at the solution to this “barotropic jet” setup in later tutorial examples.

### 4.2 Baroclinic Ocean Gyre

(in directory: verification/tutorial_baroclinic_gyre)

This section describes an example experiment using MITgcm to simulate a baroclinic, wind and buoyancy-forced, double-gyre ocean circulation. Unlike tutorial Barotropic Ocean Gyre, which used a Cartesian grid and a single vertical layer, here the grid employs spherical polar coordinates with 15 vertical layers. The configuration is similar to the double-gyre setup first solved numerically in Cox and Bryan (1984) [CB84]: the model is configured to represent an enclosed sector of fluid on a sphere, spanning the tropics to mid-latitudes, $60^\circ \times 60^\circ$ in lateral extent. The fluid is 1.8 km deep and is forced by a zonal wind stress which is constant in time, $\tau_\lambda$, varying sinusoidally in the north-south direction. The Coriolis parameter, $f$, is defined according to latitude $\varphi$

$$f(\varphi) = 2\Omega \sin(\varphi)$$

with the rotation rate, $\Omega$ set to $\frac{2\pi}{86164}$ s$^{-1}$ (i.e., corresponding to the standard Earth rotation rate). The sinusoidal wind-stress variations are defined according to

$$\tau_\lambda(\varphi) = -\tau_0 \cos \left( \frac{2\pi \varphi - \varphi_0}{L_\varphi} \right)$$

where $L_\varphi$ is the lateral domain extent ($60^\circ$), $\varphi_0$ is set to $15^\circ$N and $\tau_0$ is 0.1 N m$^{-2}$. Figure 4.5 summarizes the configuration simulated. As indicated by the axes in the lower left of the figure, the model code works internally in a locally orthogonal coordinate $(x, y, z)$. For this experiment description the local orthogonal model coordinate $(x, y, z)$ is synonymous with the coordinates $(\lambda, \varphi, r)$ shown in Figure 1.20. Initially the fluid is stratified with a reference potential temperature profile that varies from $\theta = 30^\circ$C in the surface layer to $\theta = 2^\circ$C in the bottom layer. The equation of state used in this experiment is linear:

$$\rho = \rho_0 (1 - \alpha_\theta \theta')$$

which is implemented in the model as a density anomaly equation

$$\rho' = -\rho_0 \alpha_\theta \theta'$$

with $\rho_0 = 999.8$ kg m$^{-3}$ and $\alpha_\theta = 2 \times 10^{-4}$ K$^{-1}$. Given the linear equation of state, in this configuration the model state variable for temperature is equivalent to either in-situ temperature, $T$, or potential temperature, $\theta$. For consistency with later examples, in which the equation of state is non-linear, here we use the variable $\theta$ to represent temperature.

Temperature is restored in the surface layer to a linear profile:

$$F_\theta = -\frac{1}{\tau_\theta} (\theta - \theta^*)$$

where the relaxation timescale $\tau_\theta = 30$ days and $\theta_{max} = 30^\circ$ C, $\theta_{min} = 0^\circ$ C.
4.2.1 Equations solved

For this problem the implicit free surface, HPE form of the equations (see Section 1.3.4.2; Section 2.4) described in Marshall et al. (1997) [MHPA97] are employed. The flow is three-dimensional with just temperature, \( \theta \), as an active tracer. The viscous and diffusive terms provides viscous dissipation and a diffusive sub-grid scale closure for the momentum and temperature equations, respectively. A wind-stress momentum forcing is added to the momentum equation for the zonal flow, \( u \). Other terms in the model are explicitly switched off for this experiment configuration (see Section 4.2.3). This yields an active set of equations solved in this configuration, written in spherical polar coordinates as follows:

\[
\frac{Du}{Dt} - f v - \frac{uv}{a} \tan \varphi + \frac{1}{\rho_e a \cos \varphi} \frac{\partial p'}{\partial \lambda} + \nabla_h \cdot (-A_h \nabla_h u) + \frac{\partial}{\partial z} \left( -A_z \frac{\partial u}{\partial z} \right) = F_u \tag{4.11}
\]

\[
\frac{Dv}{Dt} + f u + \frac{u^2}{a} \tan \varphi + \frac{1}{\rho_e a} \frac{\partial p'}{\partial \varphi} + \nabla_h \cdot (-A_h \nabla_h v) + \frac{\partial}{\partial z} \left( -A_z \frac{\partial v}{\partial z} \right) = F_v \tag{4.12}
\]

\[
\frac{\partial \eta}{\partial t} + \frac{1}{a \cos \varphi} \left( \frac{\partial H \frac{\partial \tilde{u}}{\partial \lambda} + \partial H \tilde{v} \cos \varphi}{\partial \varphi} \right) = 0 \tag{4.13}
\]

\[
\frac{D\theta}{Dt} + \nabla_h \cdot (-\kappa_h \nabla_h \theta) + \frac{\partial}{\partial z} \left( -\kappa_z \frac{\partial \theta}{\partial z} \right) = F_\theta \tag{4.14}
\]

\[
p' = g \rho_e \eta + \int_0^z g p' \, dz \tag{4.15}
\]

where \( u \) and \( v \) are the components of the horizontal flow vector \( \vec{u} \) on the sphere \((u = \dot{\lambda}, v = \dot{\varphi})\), \( a \) is the distance from the center of the Earth, \( \rho_e \) is a fluid density (which appears in the momentum equations, and can be set differently than \( \rho_0 \) in (4.9)), \( A_h \) and \( A_v \) are horizontal and vertical viscosity, and \( \kappa_h \) and \( \kappa_v \) are horizontal and vertical diffusivity.
respectively. The terms \( H \hat{u} \) and \( H \hat{v} \) are the components of the vertical integral term given in equation (1.35) and explained in more detail in Section 2.4. However, for the problem presented here, the continuity relation (4.13) differs from the general form given in Section 2.4, equation (2.10) because the source terms \( \mathcal{P} - \mathcal{E} + \mathcal{R} \) are all zero.

The forcing terms \( \mathcal{F}_u, \mathcal{F}_v, \) and \( \mathcal{F}_\theta \) are applied as source terms in the model surface layer and are zero in the interior. The windstress forcing, \( \mathcal{F}_u \) and \( \mathcal{F}_v \), is applied in the zonal and meridional momentum equations, respectively; in this configuration, \( \mathcal{F}_u = -\frac{\tau_x}{\rho \Delta z} \) (where \( \Delta z_s \) is the depth of the surface model gridcell), and \( \mathcal{F}_v = 0 \). Similarly, \( \mathcal{F}_\theta \) is applied in the temperature equation, as given by (4.10).

In (4.15) the pressure field, \( \rho' \), is separated into a barotropic part due to variations in sea-surface height, \( \eta \), and a hydrostatic part due to variations in density, \( \rho' \), integrated through the water column. Note the \( g \) in the first term on the right hand side is MITgcm parameter \( \bar{g} \) whereas in the second term \( g \) is parameter \( \text{gravity} \); allowing for different gravity constants here is useful, for example, if one wanted to slow down external gravity waves.

In the momentum equations, lateral and vertical boundary conditions for the \( \nabla \) gravity constants here is useful, for example, if one wanted to slow down external gravity waves.

In the momentum equations, lateral and vertical boundary conditions for the \( \nabla \) gravity constants here is useful, for example, if one wanted to slow down external gravity waves.

\( \nabla \) Barotropic Ocean Gyre

Thus, using the advective CFL condition, first assuming our solution will achieve maximum horizontal advection \( |c_{\text{max}}| \sim 1 \text{ m s}^{-1} \)

\[ S_a = 2 \left( \frac{|c_{\text{max}}| \Delta t}{\Delta x} \right) < 0.5 \text{ for stability} \quad (4.16) \]

we choose the same time step as in tutorial \( \text{Barotropic Ocean Gyre}, \Delta t = 1200 \text{ s} (= 20 \text{ minutes}) \), resulting in \( S_a = 0.08 \). Also note this time step is stable for propagation of internal gravity waves: approximating the propagation speed as...
\sqrt{g' h} \text{ where } g' \text{ is reduced gravity (our maximum } \Delta \rho \text{ using our linear equation of state is } \rho_0 \alpha_\rho \Delta \theta = 6 \text{ kg/m}^3 \text{) and } h \text{ is the upper layer depth (we'll assume 150 m), produces an estimated propagation speed generally less than } |c_{\text{max}}| = 3 \text{ ms}^{-1} \text{ (see Adcroft 1995 [Adc95] or Gill 1982 [Gil82]), thus still comfortably below the threshold.}

Using our chosen value of } \Delta t, \text{ numerical stability for inertial oscillations using Adams-Bashforth II}

\[ S_t = f \Delta t < 0.5 \text{ for stability} \quad (4.17) \]

evaluates to 0.17 for the largest } f \text{ value in our domain (1.4 } \times 10^{-4} \text{ s}^{-1}, \text{ below the stability threshold.}

To choose a horizontal Laplacian eddy viscosity } A_h, \text{ note that the largest } \Delta x \text{ value in our domain (i.e., in the south) is } \approx 110 \text{ km. With the Munk boundary width as follows,}

\[ M_w = \frac{2\pi}{\sqrt{3}} \left( \frac{A_h \Delta t}{\beta} \right)^{\frac{1}{3}} \quad (4.18) \]

in order to have a well resolved boundary current in the subtropical gyre we will set } A_h = 5000 \text{ m}^2 \text{ s}^{-1}. \text{ This results in a boundary current resolved across two to three grid cells in the southern portion of the domain.}

Given that our choice for } A_h \text{ in this experiment is an order of magnitude larger than in tutorial } \text{ Barotropic Ocean Gyre}, \text{ let’s re-examine the stability of horizontal Laplacian friction:}

\[ S_{th} = 2 \left( 4 \frac{A_h \Delta t}{\Delta x^2} \right) < 0.6 \text{ for stability} \quad (4.19) \]

evaluates to 0.057 for our smallest } \Delta x, \text{ which is below the stability threshold. Note this same stability test also applies to horizontal Laplacian diffusion of tracers, with } \kappa_h \text{ replacing } A_h, \text{ but we will choose } \kappa_h \ll A_h \text{ so this should not pose any stability issues.}

Finally, stability of vertical diffusion of momentum:

\[ S_{tv} = 4 \frac{A_v \Delta t}{\Delta z^2} < 0.6 \text{ for stability} \quad (4.20) \]

Here we will choose } A_v = 1 \times 10^{-2} \text{ m}^2 \text{ s}^{-1}, \text{ so } S_{tv} \text{ evaluates to 0.02 for our minimum } \Delta z, \text{ well below the stability threshold. Note if we were to use Adams Bashforth II for diffusion of tracers the same check would apply, with } \kappa_v \text{ replacing } A_v. \text{ However, we will instead choose an implicit scheme for computing vertical diffusion of tracers (see Section 4.2.3.2), which is unconditionally stable.}

### 4.2.3 Configuration

The model configuration for this experiment resides under the directory verification/tutorial_baroclinic_gyre/.

The experiment files

- verification/tutorial_baroclinic_gyre/code/packages.conf
- verification/tutorial_baroclinic_gyre/code/SIZE.h
- verification/tutorial_baroclinic_gyre/code/DIAGNOSTICS_SIZE.h
- verification/tutorial_baroclinic_gyre/input/data
- verification/tutorial_baroclinic_gyre/input/data.pkg
- verification/tutorial_baroclinic_gyre/input/data.mnc
- verification/tutorial_baroclinic_gyre/input/data.diagnostics
- verification/tutorial_baroclinic_gyre/input/eedata
- verification/tutorial_baroclinic_gyre/input/bathy.bin

### 4.2. Baroclinic Ocean Gyre
• verification/tutorial_baroclinic_gyre/input/windx_cosy.bin
• verification/tutorial_baroclinic_gyre/input/SST_relax.bin

contain the code customizations, parameter settings, and input data files for this experiment. Below we describe these customizations in detail.

4.2.3.1 Compile-time Configuration

File code/packages.conf

Listing 4.5: verification/tutorial_baroclinic_gyre/code/packages.conf

```plaintext
--- list of packages (or group of packages) to compile for this experiment:
gfd
diagnostics
mnc
```

Here we specify which MITgcm packages we want to include in our configuration. gfd is a pre-defined “package group” (see Using MITgcm Packages) of standard packages necessary for most setups; it is also the default compiled packages setting and the minimum set of packages necessary for GFD-type setups. In addition to package group gfd we include two additional packages (individual packages, not package groups), mnc and diagnostics. Package mnc is required for output to be dumped in netCDF format. Package diagnostics allows one to choose output from a extensive list of model diagnostics, and specify output frequency, with multiple time averaging or snapshot options available. Without this package enabled, output is limited to a small number of snapshot output fields. Subsequent tutorial experiments will explore the use of packages which expand the physical and scientific capabilities of MITgcm, e.g., such as physical parameterizations or modeling capabilities for tracers, ice, etc., that are not compiled unless specified.

File code/SIZE.h

Listing 4.6: verification/tutorial_baroclinic_gyre/code/SIZE.h

```plaintext
CBOP
C !ROUTINE: SIZE.h
C !INTERFACE:
C include SIZE.h
C !DESCRIPTION: \bv
C *==========================================================*
C | SIZE.h Declare size of underlying computational grid.
C *==========================================================*
C | The design here supports a three-dimensional model grid
C | with indices I,J and K. The three-dimensional domain
C | is comprised of nPx*nSx blocks (or tiles) of size sNx
C | along the first (left-most index) axis, nPy*nSy blocks
C | of size sNy along the second axis and one block of size
C | Nr along the vertical (third) axis.
C | Blocks/tiles have overlap regions of size OLx and OLy
C | along the dimensions that are subdivided.
C *==========================================================*
C \ev
C
C Voodoo numbers controlling data layout:
```
For this second tutorial, we will break the model domain into multiple tiles. Although initially we will run the model on a single processor, a multi-tiled setup is required when we demonstrate how to run the model using either MPI or using multiple threads.

The following lines calculate the horizontal size of the global model domain (NOT to be edited). Our values for SIZE.h parameters below must multiply so that our horizontal model domain is $62 \times 62$:

```fortran
& Nx = sNx*nSx*nPx,
& Ny = sNy*nSy*nPy,
& Nr = 15)
```

### 4.2. Baroclinic Ocean Gyre
Now let’s look at all individual \texttt{SIZE.h} parameter settings.

- Although our model domain is 62\(\times\)62, here we specify the size of a single tile to be one-half that in both \(x\) and \(y\). Thus, the model requires four of these tiles to cover the full ocean sector domain (see below, where we set \(nSx\) and \(nSy\)). Note that the grid can only be subdivided into tiles in the horizontal dimensions, not in the vertical.

\begin{verbatim}
& sNx = 31,
& sNy = 31,
\end{verbatim}

- As in tutorial \textit{Barotropic Ocean Gyre}, here we set the overlap extent of a model tile to the value 2 in both \(x\) and \(y\). In other words, although our model tiles are sized 31\(\times\)31, in MITgcm array storage there are an additional 2 border rows surrounding each tile which contain model data from neighboring tiles. Some horizontal advection schemes and other parameter and setup choices require a larger overlap setting (see Table 2.2). In our configuration, we are using a second-order center-differences advection scheme (the MITgcm default) which does not require setting a overlap beyond the MITgcm minimum 2.

\begin{verbatim}
& OLx = 2,
& OLy = 2,
\end{verbatim}

- These lines set parameters \(nSx\) and \(nSy\), the number of model tiles in the \(x\) and \(y\) directions, respectively, which execute on a single process. Initially, we will run the model on a single core, thus both \(nSx\) and \(nSy\) are set to 2 so that all 2\(\times\)2 = 4 tiles are integrated forward in time.

\begin{verbatim}
& nSx = 2,
& nSy = 2,
\end{verbatim}

- These lines set parameters \(nPx\) and \(nPy\), the number of processes to use in the \(x\) and \(y\) directions, respectively. As noted, initially we will run using a single process, so for now these parameters are both set to 1.

\begin{verbatim}
& nPx = 1,
& nPy = 1,
\end{verbatim}

- Here we tell the model we are using 15 vertical levels.

\begin{verbatim}
& Nr = 15)
\end{verbatim}

\textbf{File code/DIAGNOSTICS\_SIZE.h}

\begin{verbatim}
C Diagnostics Array Dimension
C ----------------------------------------
C ndiagMax :: maximum total number of available diagnostics
C numlists :: maximum number of diagnostics list (in data.diagnostics)
C numperlist :: maximum number of active diagnostics per list (data.diagnostics)
C numLevels :: maximum number of levels to write (data.diagnostics)
C numDiags :: maximum size of the storage array for active 2D/3D diagnostics
C nRegions :: maximum number of regions (statistics-diagnostics)
C sizRegMsk :: maximum size of the regional-mask (statistics-diagnostics)
C nStats :: maximum number of statistics (e.g.: aver,min,max ...)
C diagSt_size:: maximum size of the storage array for statistics-diagnostics
C Note : may need to increase "numDiags" when using several 2D/3D diagnostics,
C and "diagSt_size" (statistics-diags) since values here are deliberately small.
\end{verbatim}

(continues on next page)
In the default version /pkg/diagnostics/DIAGNOSTICS_SIZE.h the storage array for diagnostics is purposely set quite small, in other words forcing the user to assess how many diagnostics will be computed and thus choose an appropriate size for a storage array. In the above file we have modified the value of parameter `numDiags`:

```
PARAMETER( numDiags = 20*Nr )
```

from its default value 1*Nr, which would only allow a single 3-D diagnostic to be computed and saved, to 20*Nr, which will permit up to some combination of up to 20 3-D diagnostics or 300 2-D diagnostic fields.

### 4.2.3.2 Run-time Configuration

#### File input/data

Listing 4.8: verification/tutorial_baroclinic_gyre/input/data

- # Model parameters
- # Continuous equation parameters
- &PARM01
- viscaH=5000.,
- viscsR=1.E-2,
- no_slip_sides=.TRUE.,
- no_slip_bottom=.FALSE.,
- diffKt=1000.,
- diffKr=1.E-5,
- ivd_c_kappa=1.,
- implicitDiffusion=.TRUE.,
- eosType='LINEAR',
- tAlpha=2,E-4,
- sBeta=0.,
- rhoNiL=999.8,
- gravity=9.81,
- rigidLid=.FALSE.,
- implicitFreeSurface=.TRUE.,
- exactConserv=.TRUE.,

(continues on next page)
Parameters for this configuration are set as follows.
PARM01 - Continuous equation parameters

• These lines set parameters \( \text{viscAh} \) and \( \text{viscAr} \), the horizontal and vertical Laplacian viscosities respectively, to 5000 m\(^2\) s\(^{-1}\) and \(1 \times 10^{-2}\) m\(^2\) s\(^{-1}\). Note the subscript \( r \) is used for the vertical, reflecting MITgcm’s generic \( r \)-vertical coordinate capability (i.e., the model is capable of using either a \( z \)-coordinate or a \( p \)-coordinate system).

\[
\begin{align*}
\text{viscAh} &= 5000., \\
\text{viscAr} &= 1.E-2,
\end{align*}
\]

• These lines set parameters \( \text{no_slip_sides} \) and \( \text{no_slip_bottom} \), to specify the boundary conditions for momentum on the model domain sidewalls and bottom. Parameter \( \text{no_slip_sides} \) is set to \( \text{.TRUE.} \), i.e., no-slip lateral boundary conditions (the default), which will yield a Munk (1950) [Mun50] western boundary solution. Parameter \( \text{no_slip_bottom} \) is set to \( \text{.FALSE.} \), i.e., free-slip bottom boundary condition (default is true). If instead of a Munk layer we desired a Stommel (1948) [Sto48] western boundary layer solution, we would opt for free-slip lateral boundary conditions and no-slip conditions along the bottom.

\[
\begin{align*}
\text{no_slip_sides} &= \text{.TRUE.}, \\
\text{no_slip_bottom} &= \text{.FALSE.},
\end{align*}
\]

• These lines set parameters \( \text{diffKhT} \) and \( \text{diffKrT} \), the horizontal and vertical Laplacian temperature diffusivities respectively, to 1000 m\(^2\) s\(^{-1}\) and \(1 \times 10^{-5}\) m\(^2\) s\(^{-1}\). The boundary condition on this operator is zero-flux at all boundaries.

\[
\begin{align*}
\text{diffKhT} &= 1000., \\
\text{diffKrT} &= 1.E-5,
\end{align*}
\]

• By default, MITgcm does not apply any parameterization to mix statically unstable columns of water. In a coarse resolution, hydrostatic configuration, typically such a parameterization is desired. We recommend a scheme which simply applies (presumably, large) vertical diffusivity between statically unstable grid cells in the vertical. This vertical diffusivity is set by parameter \( \text{ivdc_kappa} \), which here we set to 1.0 m\(^2\) s\(^{-1}\). This scheme requires that \( \text{implicitDiffusion} \) is set to \( \text{.TRUE.} \) (see Section 2.6; more specifically, applying a large vertical diffusivity to represent convective mixing requires the use of an implicit time-stepping method for vertical diffusion, rather than Adams Bashforth II). Alternatively, a traditional convective adjustment scheme is available; this can be activated through the \( \text{cAdjFreq} \) parameter, see Section 3.8.5.4.

\[
\begin{align*}
\text{ivdc_kappa} &= 1., \\
\text{implicitDiffusion} &= \text{.TRUE.},
\end{align*}
\]

• The following parameters tell the model to use a linear equation of state. Note a list of \( \text{Nr} (=15, \text{from SIZE.h}) \) potential temperature values in °C is specified for parameter \( \text{tRef} \), ordered from surface to depth. \( \text{tRef} \) is used for two purposes here. First, anomalies in density are computed using this reference \( \theta, \theta'(x,y,z) = \theta(x,y,z) - \theta_{ref}(z) \); see use in (4.8) and (4.9). Second, the model will use these reference temperatures for its initial state, as we are not providing a pickup file nor specifying an initial temperature hydrographic file (in later tutorials we will demonstrate how to do so). For each depth level the initial and reference profiles will be uniform in \( x \) and \( y \). Note when checking static stability or computing \( N^2 \), the density gradient resulting from these specified reference levels is added to \( \partial \rho' / \partial z \) from (4.9). Finally, we set the thermal expansion coefficient \( \alpha_\theta (\text{tAlpha}) \) as used in (4.8) and (4.9), while setting the haline contraction coefficient \( \text{sBeta} \) to zero (see (4.8), which omits a salinity contribution to the linear equation of state; like tutorial \( \text{Barotropic Ocean Gyre} \), salinity is not included as a tracer in this very idealized model setup).

\[
\begin{align*}
\text{eosType} &= \text{’LINEAR’}, \\
\text{tRef} &= 30., 27., 24., 21., 18., 15., 13., 11., 9., 7., 6., 5., 4., 3., 2., \\
\text{tAlpha} &= 2.E-4, \\
\text{sBeta} &= 0.,
\end{align*}
\]

4.2. Baroclinic Ocean Gyre
• This line sets parameter $\rho_0$ (rhoNil) to 999.8 kg/m$^3$, the surface reference density for our linear equation of state, i.e., the density of water at $t_{\text{Ref}}(k=1)$. This value will also be used as $\rho_c$ (parameter rhoConst) in (4.11)-(4.15), lacking a separate explicit assignment of rhoConst in data. Note this value is the model default value for rhoNil.

```
 rhoNil=999.8,
```

• This line sets parameter gravity, the acceleration due to gravity $g$ in (4.15), and this value will also be used to set gBaro, the barotropic (i.e., free surface-related) gravity parameter which we set in tutorial Barotropic Ocean Gyre. This is the MITgcm default value.

```
gravity=9.81,
```

• These lines set parameters which prescribe the linearized free surface formulation, similar to tutorial Barotropic Ocean Gyre. Note we have added parameter exactConserv, set to .TRUE.: this instructs the model to recompute divergence after the pressure solver step, ensuring volume conservation of the free surface solution (the model default is NOT to recompute divergence, but given the small numerical cost, we typically recommend doing so).

```
rigidLid=.FALSE.,
imPLICITFreeSurface=.TRUE.,
extactConserv=.TRUE.,
```

• As in tutorial Barotropic Ocean Gyre, we suppress MITgcm’s forward time integration of salt in the tracer equations.

```
saltStepping=.FALSE.,
```

**PARM02 - Elliptic solver parameters**

These parameters are unchanged from tutorial Barotropic Ocean Gyre.

**PARM03 - Time stepping parameters**

• In tutorial Barotropic Ocean Gyre we specified a starting iteration number nIter0 and a number of time steps to integrate, nTimeSteps. Here we opt to use another approach to control run start and duration: we set a startTime and endTime, both in units of seconds. Given a starting time of 0.0, the model starts from rest using specified initial values of temperature (here, as previously noted, from the tRef parameter) rather than attempting to restart from a saved checkpoint file. The specified value for endTime, 12000.0 seconds is equivalent to 10 time steps, set for testing purposes. To integrate over a longer, more physically relevant period of time, uncomment the endTime and monitorFreq lines located near the end of this parameter block. Note, for simplicity, our units for these time choices assume a 360-day “year” and 30-day “month” (although lacking a seasonal cycle in our forcing, defining a “year” is immaterial; we will demonstrate how to apply time-varying forcings in later tutorials).

```
startTime=0.,
endTime=12000.,
```

```
# for longer run (100 yrs)
# endTime  = 3110400000.,
# monitorFreq=2592000.,
```

• Remaining time stepping parameter choices (specifically, $\Delta t$, checkpoint frequency, output frequency, and monitor settings) are described in tutorial Barotropic Ocean Gyre; refer to the description here.
The parameter `tauThetaClimRelax` sets the time scale, in seconds, for restoring potential temperature in the model’s top surface layer (see (4.10)). Our choice here of 2,592,000 seconds is equal to 30 days.

```
deltaT=1200.,
pChkptFreq=622080000.,
chkptFreq=155520000.,
dumpFreq=31104000.,
monitorFreq=1200.,
monitorSelect=2,
```

### PARM04 - Gridding parameters

- This line sets parameter `usingSphericalPolarGrid`, which specifies that the simulation will use spherical polar coordinates (and affects the interpretation of other grid coordinate parameters).

```
usingSphericalPolarGrid=.TRUE.,
```

- These lines set the horizontal grid spacing, as vectors `delX` and `delY` (i.e., $\Delta x$ and $\Delta y$ respectively), with units of degrees as dictated by our choice `usingSphericalPolarGrid`. As before, this syntax indicates that we specify 62 values in both the $x$ and $y$ directions, which matches the global domain size as specified in `SIZE.h`. Our ocean sector domain starts at 0° longitude and 15° N; accounting for a surrounding land row of cells, we thus set the origin in longitude to −1.0° and in latitude to 14.0°. Again note that our origin specifies the southern and western edges of the gridcell, not the cell center location. Setting the origin in latitude is critical given that it affects the Coriolis parameter $f$ (which appears in (4.11) and (4.12)); the default value for `ygOrigin` is 0.0°. Note that setting `xgOrigin` is optional, given that absolute longitude does not appear in the equation discretization.

```
delX=62*1.,
delY=62*1.,
xgOrigin=-1.,
ygOrigin=14.,
```

- This line sets parameter `delR`, the vertical grid spacing in the $z$-coordinate (i.e., $\Delta z$), to a vector of 15 depths (in meters), from 50 m in the surface layer to a bottom layer depth of 190 m. The sum of these specified depths equals 1800 m, the full depth $H$ of our idealized ocean sector.

```
delR=50.,60.,70.,80.,90.,100.,110.,120.,130.,140.,150.,160.,170.,180.,190.,
```

### PARM05 - Input datasets

- Similar to tutorial `Barotropic Ocean Gyre`, these lines specify filenames for bathymetry and surface wind stress forcing files.

```
bathyFile='bathy.bin',
zonalWindFile='windx_cosy.bin',
```

- This line specifies parameter `thetaClimFile`, the filename for the (2-D) restoring temperature field.

```
thetaClimFile='SST_relax.bin',
```

4.2. Baroclinic Ocean Gyre
Here we activate two MITgcm packages that are not included with the model by default: package mnc (see Section 9.3) specifies that model output should be written in netCDF format, and package diagnostics (see Section 9.1) allows user-selectable diagnostic output. The boolean parameters set are useMNC and useDiagnostics, respectively. Note these add-on packages also need to be specified when the model is compiled, see Section 4.2.3.1. Apart from these two additional packages, only standard packages (i.e., those compiled in MITgcm by default) are required for this setup.

This file sets parameters which affect package pkg/mnc behavior; in fact, with pkg/mnc enabled, it is required (many packages look for file data."PACKAGENAME" and will terminate if not present). By setting the parameter monitor_mnc to .FALSE., we are specifying NOT to create separate netCDF output files for pkg/monitor output, but rather to include this monitor output in the standard output file (see Section 4.1.4). See Section 9.3.1.2 for a complete listing of pkg/mnc namelist parameters and their default settings.

Unlike raw binary output, which overwrites any existing files, when using mnc output the model will create new directories if the parameters mnc_use_outdir and mnc_outdir_str are set, as above; the model will append a 4-digit number to mnc_outdir_str, starting at 0001, incrementing as needed if existing directories already exist. If these parameters are NOT set, the model will terminate with an error if one attempts to overwrite an existing .nc file (in other words, to re-run in an previous run directory, one must delete all *.nc files before restarting). Note that our subdirectory name choice mnc_test_ is required by MITgcm automated testing protocols, and can be changed to something more mnemonic, if desired.

In general, it is good practice to write diagnostic output into subdirectories, to keep the top run directory less “cluttered”; some unix file systems do not respond well when very large numbers of files are produced, which can occur in setups divided into many tiles and/or when many diagnostics are selected for output.
# Diagnostic Package Choices

#-------------------
# dumpAtLast (logical): always write output at the end of simulation (default=F)
# diag_mnc (logical): write to NetCDF files (default=useMNC)
#--for each output-stream:
#  fileName(n) : prefix of the output file name (max 80c long) for outp.stream n
#  frequency(n):< 0 : write snap-shot output every |frequency| seconds
#  > 0 : write time-average output every frequency seconds
#  timePhase(n) : write at time = timePhase + multiple of |frequency|
#  averagingFreq : frequency (in s) for periodic averaging interval
#  averagingPhase : phase (in s) for periodic averaging interval
#  repeatCycle : number of averaging intervals in 1 cycle
#  levels(:,n) : list of levels to write to file (Notes: declared as REAL)
#  when this entry is missing, select all common levels of this list
#  fields(:,n) : list of selected diagnostics fields (8.c) in outp.stream n
#  (see "available_diagnostics.log" file for the full list of diags)
#  missing_value(n) : missing value for real-type fields in output file "n"
#  fileFlags(n) : specific code (8c string) for output file "n"
#-------------------

&DIAGNOSTICS_LIST

fields(1:3,1) = 'ETAN','TRELAX','MXLDEPTH',
fileName(1) = 'surfDiag',
frequency(1) = 31104000.,

fields(1:5,2) = 'THETA','PHIHYD','UVEL','VVEL','WVEL',
# did not specify levels => all levels are selected
fileName(2) = 'dynDiag',
frequency(2) = 31104000.,
&

#-------------------
# Parameter for Diagnostics of per level statistics:
#-------------------
# diagSt_mnc (logical): write stat-diags to NetCDF files (default=diag_mnc)
# diagSt_regMaskFile : file containing the region-mask to read-in
# nSetRegMskFile : number of region-mask sets within the region-mask file
# set_regMask(i) : region-mask set-index that identifies the region "i"
# val_regMask(i) : region "i" identifier value in the region mask
#--for each output-stream:
#  stat_fName(n) : prefix of the output file name (max 80c long) for outp.stream n
#  stat_freq(n):< 0 : write snap-shot output every |stat_freq| seconds
#  > 0 : write time-average output every stat_freq seconds
#  stat_phase(n) : write at time = stat_phase + multiple of |stat_freq|
#  stat_region(:,n) : list of "regions" (default: 1 region only=global)
#  stat_fields(:,n) : list of selected diagnostics fields (8.c) in outp.stream n
#  (see "available_diagnostics.log" file for the full list of diags)
#-------------------

&DIAG_STATIS_PARMS

(continues on next page)
The above lines tell MITgcm that our first list will consist of three diagnostic variables:

- **ETAN** - the linearized free surface height (m)
- **TRELAX** - the heat flux entering the ocean due to surface temperature relaxation (W/m²)
- **MXLDEPTH** - the depth of the mixed layer (m), as defined here by a given magnitude decrease in density from the surface (we’ll use the model default for ∆ρ)

Note that all these diagnostic fields are 2-D output. **2-D and 3-D diagnostics CANNOT be mixed in a diagnostics list.** These variables are specified in parameter fields: the first index is specified as 1:«NUMBER_OF_DIAGS», the second index designates this for diagnostics list 1. Next, the output filename for diagnostics list 1 is specified in variable fileName. Finally, for this list we specify variable frequency to provide time-averaged output every 31,104,000 seconds, i.e., once per year. Had we entered a negative value for frequency, MITgcm would have instead written snapshot data at this interval. Next, we set up a second diagnostics list for several 3-D diagnostics.

The diagnostics in list 2 are:

- **THETA** - potential temperature (°C)
- **PHIHYD** - hydrostatic pressure potential anomaly (m²/s²)
- **UVEL**, **VVEL**, **WVEL** - the zonal, meridional, and vertical velocity components respectively (m/s)

Here we did not specify parameter levels, so all depth levels will be included in the output. An example of syntax to limit which depths are output is `levels(1:5,2) = 1., 2., 3.,`, which would dump just the top three levels. We again specify an output file name via parameter fileName, and specify a time-average period of one year through parameter frequency.

```plaintext
stat_fields(1:2,1) = 'THETA', 'TRELAX',
stat_fName(1) = 'dynStDiag',
stat_freq(1) = 2592000.,
&

DIAGNOSTICS_LIST - Diagnostic Package Choices

In this section we specify what diagnostics we want to compute, how frequently to compute them, and the name of output files. Multiple diagnostic fields can be grouped into individual files (i.e., an individual output file here is associated with a ‘list’ of diagnostics).

```plaintext
fields(1:3,1) = 'ETAN', 'TRELAX', 'MXLDEPTH',
fileName(1) = 'surfDiag',
frequency(1) = 31104000.,
```
DIAG_STATIS_PARMS - Diagnostic Per Level Statistics

It is also possible to request output statistics averaged for global mean and by level average (for 3-D diagnostics) over the full domain, and/or for a pre-defined \((x, y)\) region of the model grid. The statistics computed for each diagnostic are as follows:

- (area weighted) mean (in both space and time, if time-averaged frequency is selected)
- (area weighted) standard deviation
- minimum value
- maximum value
- volume of the area used in the calculation (multiplied by the number of time steps if time-averaged).

While these statistics could in theory also be calculated (by the user) from 2-D and 3-D DIAGNOSTICS_LIST output, the advantage is that much higher frequency statistical output can be achieved without filling up copious amounts of disk space.

Options for namelist DIAG_STATIS_PARMS are set as follows:

```
stat_fields(1:2,1) = 'THETA', 'TRELAX',
stat_fName(1) = 'dynStDiag',
stat_freq(1) = 2592000.,
```

The syntax here is analogous with DIAGNOSTICS_LIST namelist parameters, except the parameter names begin with stat (here, stat_fields, stat_fName, stat_freq). Frequency can be set to snapshot or time-averaged output, and multiple lists of diagnostics (i.e., separate output files) can be specified. The only major difference from DIAGNOSTICS_LIST syntax is that 2-D and 3-D diagnostics can be mixed in a list. As noted, it is possible to select limited horizontal regions of interest, in addition to the full domain calculation.

File input/eedata

Listing 4.12: verification/tutorial_baroclinic_gyre/input/eedata

```
# Example "eedata" file
# Lines beginning ":" are comments
# nTx :: No. threads per process in X
# nTy :: No. threads per process in Y
# debugMode :: print debug msg (sequence of S/R calls)
&EEPARMS
nTx=1,
nTy=1,
&
# Note: Some systems use & as the namelist terminator (as shown here).
# Other systems use a / character.
```

As shown, this file is configured for a single-threaded run, but will be modified later in this tutorial for a multi-threaded setup (Section 4.2.6).

4.2. Baroclinic Ocean Gyre
Files input/bathy.bin, input/windx.cosy.bin

The purpose and format of these files is similar to tutorial Barotropic Ocean Gyre, and were generated by matlab script verification/tutorial_baroclinic_gyre/input/gendata.m (alternatively, python script gendata.py). See Section 3.9 for additional information on MITgcm input data file format specifications.

File input/SST_relax.bin

This file specifies a 2-D(\(x, y\)) map of surface relaxation temperature values, as generated by verification/tutorial_baroclinic_gyre/input/gendata.m or gendata.py.

4.2.4 Building and running the model

To build and run the model on a single processor, follow the procedure outlined in Section 4.1.4. To run the model for a longer period (i.e., to obtain a reasonable solution; for testing purposes, by default the model is set to run only a few time steps) uncomment the lines in data which specify larger numbers for parameters endTime and monitorFreq. This will run the model for 100 years, which will likely take several hours on a single processor (depending on your computer specs); below we also give instructions for running the model in parallel either using MPI or multi-threaded (OpenMP), which will cut down run time significantly.

4.2.4.1 Output Files

As in tutorial Barotropic Ocean Gyre, standard output is produced (redirected into file output.txt as specified in Section 4.1.4); like before, this file includes model startup information, parameters, etc. (see Section 4.1.4.1). And because we set monitor_mnc = .FALSE. in data.mnc, our standard output file will include all monitor statistics output. Note monitor statistics and cg2d information are evaluated over the global domain, despite the bifurcation of the grid into four separate tiles. As before, the file STDERR.0000 will contain a log of any run-time errors.

With pkg/mnc compiled and activated in data.pkg, other output is in netCDF format: grid information, snapshot output specified in data, diagnostics output specified in data.diagnostics and separate files containing hydrostatic pressure data (see below). There are two notable differences from standard binary output. Recall that we specified that the grid was subdivided into four separate tiles (in SIZE.h); instead of a .XXX.YYY. file naming scheme for different tiles (as discussed here), with pkg/nmc the file names contain .t«nnn». where «nnn» is the tile number. Secondly, model data from multiple time snapshots (or periods) is included in a single file. Although an iteration number is still part of the file name (here, 0000000000), this is the iteration number at the start of the run (instead of marking the specific iteration number for the data contained in the file, as the case for standard binary output). Note that if you dump data frequently, standard binary can produce huge quantities of separate files, whereas using netCDF will greatly reduce the number of files. On the other hand, the netCDF files created can instead become quite large.

To more easily process and plot our results as a single array over the full domain, we will first reassemble the individual tiles into new netCDF format global data files. To accomplish this, we will make use of utility script utils/python/MITgcmutils/scripts/gluemncbig. From the output run (top) directory, type:

```
% ln -s ../../../utils/python/MITgcmutils/scripts/gluemncbig .
% ./gluemncbig -o grid.nc mnc_test_*/grid.t*.nc
% ./gluemncbig -o state.nc mnc_test_*/state*.t*.nc
% ./gluemncbig -o dynDiag.nc mnc_test_*/dynDiag*.t*.nc
% ./gluemncbig -o surfDiag.nc mnc_test_*/surfDiag*.t*.nc
% ./gluemncbig -o phiHyd.nc mnc_test_*/phiHyd*.t*.nc
% ./gluemncbig -o phiHydLow.nc mnc_test_*/phiHydLow*.t*.nc
% ln -s mnc_test_0001/dynStDiag.0000000000.t001.nc dynStDiag.nc
```
For help using this utility, type glueMncBig --help (note, this utility requires python). The files grid.nc, state.nc, etc. are concatenated from the separate t001, t002, t003, t004 files into global grid files of horizontal dimension 62 x 62. glueMncBig is a fairly intelligent script, and by inserting the wildcards in the path/filename specification, it will grab the most recent run (in case you have started up runs multiple times in this directory, thus having mnc_test_0001, mnc_test_0002, etc. directories present; see Section 4.2.3.2). Note that the last line above is simply making a link to a file in the mnc_test_0001 output subdirectory; this is the statistical-dynamical diagnostics output, which is already assembled over the global domain (and also note here we are required to be specific which mnc_test_ directory to link from). For convenience we simply place the link at the top level of the run directory, where the other assembled .nc files are saved by glueMncBig.

Let’s proceed through the netcdf output that is produced.

- **grid.nc** - includes all the model grid variables used by MITgcm. This includes the grid cell center points and separation (XC, YC, dXc, dYc), corner point locations and separation (XG, YG, dXg, dYg), the separation between velocity points (dYU, dXV), vertical coordinate location and separation (RC, RF, dRC, dRF), grid cell areas (rA, rAw, rAS, rAZ), and bathymetry information (Depth, HFacC, HFacW, HFacS). See Section 2.11 for definitions and description of the C grid staggering of these variables. There are also grid variables in vector form that are not used in the MITgcm source code (X, Y, Xp1, Yp1, Z, Zp1, Zu, Zl); see description in grid.nc. The variables named p1 include an additional data point and are dimensioned +1 larger than the standard array size; for example, Xp1 is the longitude of the gridcell left corner, and includes an extra data point for the last gridcell’s right corner longitude.

- **state.nc** - includes snapshots of state variables U, V, W, Temp, S, and Eta at model times T in seconds (variable iter(T) stores the model iteration corresponding with these model times). Also included are vector forms of grid variables X, Y, Z, Xp1, Yp1, and Zl. As mentioned, in model output-by-tile files, e.g., state.00000000000.t001.nc, the iteration number 00000000000 is the parameter nIter0 for the model run (recall, we initialized our model with nIter0 = 0). Snapshots of model state are written for model iterations 0, 25920, 51840, … according to our data file parameter choice dumpFreq (dumpFreq/deltaT = 25920).

- **surfDiag.nc** - includes output diagnostics as specified from list 1 in data.diagnostics. Here we specified that list 1 include 2-D diagnostics ETAN, TRELAX, and MXLDEPTH. Also includes an array of model times corresponding to the end of the time-average period, the iteration number corresponding to these model times, and vector forms of grid variables which describe these data. A Z index is included in the output arrays, even though its dimension is one (given that this list contains only 2-D fields).

- **dynDiag.nc** - similar to surfDiag.nc except this file contains the time-averaged 3-D diagnostics we specified in list 2 of data.diagnostics: THETA, PHIHYD, UVEL, VVEL, WVEL.

- **dynStDiag.nc** - includes output statistical-dynamical diagnostics as specified in the DIAG_STATIS_PARS section of data.diagnostics. Like surfDiag.nc it also includes an array of model times and corresponding iteration numbers for each time-average period end. Output variables are 3-D: (time, region, depth). In data.diagnostics, we have not defined any additional regions (and by default only global output is produced, “region 1”). Depth-integrated statistics are computed (in which case the depth subscript has a range of one element; this is also the case for surface diagnostics such as TRELAX), but output is also tabulated at each depth for some variables (i.e., the depth subscript will range from 1 to Nr).

- **phiHyd.nc, phiHydLow.nc** - these files contain a snapshot 3-D field of hydrostatic pressure potential anomaly \( p'/\rho_c \) (see Section 1.3.6) and a snapshot 2-D field of bottom hydrostatic pressure potential anomaly, respectively. These are technically not MITgcm state variables, as they are computed during the time step (normal snapshot state variables are dumped after the time step), ergo they are not included in file state.nc. Like state.nc output however these fields are written at interval according to dumpFreq, except are not written out at time nIter0 (i.e., have one time record fewer than state.nc). Also note when writing standard binary output, these filenames begin as PH and PHL respectively.

The hydrostatic pressure potential anomaly \( \phi' \) is computed as follows:

\[
\phi' = \frac{1}{\rho_c} \left( \rho_c g \eta + \int_0^z (\rho - \rho_0) g dz \right)
\]
following (4.8), (4.9) and (4.15). Note that with the linear free surface approximation, the contribution of the free
surface position $\eta$ to $\phi'$ involves the constant density $\rho_c$ and not the density anomaly $\rho'$, in contrast with contributions
from below $z = 0$.

Several additional files are output in standard binary format. These are:

RhoRef.data, RhoRef.meta - this is a 1-D ($k=1...Nr$) array of reference density, defined as:

$$\rho_{ref}(k) = \rho_0 (1 - \alpha_0(\theta_{ref}(k) - \theta_{ref}(1)))$$

PHrefC.data, PHrefC.meta, PHrefF.data, PHrefF.meta - these are 1-D ($k=1...Nr$ for PHrefC and $k=1...Nr+1$ for PHrefF) arrays containing a reference hydrostatic “pressure potential” $\phi = p/\rho_c$ (see Section 1.3.6). Using a linear equation of state, PHrefC is simply $\rho_c g |z|/\rho_c$, with output computed at the midpoint of each vertical cell, whereas PHrefF is computed at the surface and bottom of each vertical cell. Note that these quantities are not especially useful when using a linear equation of state (to compute the full hydrostatic pressure potential, one would use RhoRef and integrate downward, and add phiHyd, rather than use these fields), but are of greater utility using a non-linear equation of state.

pickup.ckptA.001.001.data, pickup.ckptA.001.001.meta, pickup.0000518400.001.001.data, pickup.0000518400.001.001.meta etc. - as described in detail in tutorial Barotropic Gyre, these are temporary and permanent checkpoint files, output in binary format. Note that separate checkpoint files are written for each model tile.

And finally, because we are using the diagnostics package, upon startup the file available_diagnostics.log will be generated. This (plain text) file contains a list of all diagnostics available for output in this setup, including a description of each diagnostic and its units, and the number of levels for which the diagnostic is available (i.e., 2-D or 3-D field). This list of available diagnostics will change based on what packages are included in the setup. For example, if your setup includes a seaice package, many seaice diagnostics will be listed in available_diagnostics.log that are not available for the tutorial Baroclinic Gyre setup.

### 4.2.5 Running with MPI

In the verification/tutorial_baroclinic_gyre/code directory there is a alternate file verification/tutorial_baroclinic_gyre/code/SIZE.h_mpi. Overwrite verification/tutorial_baroclinic_gyre/code/SIZE.h with this file and re-compile the model from scratch (the most simple approach is to create a new build directory build_mpi; if instead you wanted to re-compile in your existing build directory, you should make CLEAN first, which will delete any existing files and dependencies you had created previously):

```
% ../../../tools/genmake2 -mods ../code -mpi -of="/PATH/TO/OPTFILE"
% make depend
% make
```

Note we have added the option -mpi to the genmake2 command that generates the makefile. A successful build requires MPI libraries installed on your system, and you may need to add to your $PATH environment variable and/or set environment variable $MPI INC_DIR (for more details, see Section 3.5.4). If there is a problem finding MPI libraries, genmake2 output will complain.

Several lines in verification/tutorial_baroclinic_gyre/code/SIZE.h_mpi are different from the standard version. First, we change nSx and nSy to 1, so that each process integrates the model for a single tile.

```plaintext
& nSx = 1,
& nSy = 1,
```

Next, we change nPx and nPy so that we use two processes in each dimension, for a total of $2 \times 2 = 4$ processes. Effectively, we have subdivided the model grid into four separate tiles, and the model equations are solved in parallel on four separate processes (presumably, on a unique physical processor or core). Because of the overlap regions (i.e.,
gridpoints along the tile edges are duplicated in two or more tiles), and limitations in the transfer speed of data between processes, the model will not run \(4 \times\) faster, but should be at least \(2-3 \times\) faster than running on a single process.

Finally, to run the model (from your run directory), using four processes running in parallel:

\[
\begin{align*}
\& nPx = 2, \\
\& nPy = 2,
\end{align*}
\]

On some systems the MPI run command (and the subsequent command-line option \(-np\)) might be something other than \texttt{mpirun}; ask your local system administrator. When using a large HPC cluster, prior steps might be required to allocate four processor cores to your job, and/or it might be necessary to write this command within a batch scheduler script; again, check with your local system documentation or system administrator. If four cores are not available when you execute the above \texttt{mpirun} command, an error will occur.

When running in parallel, \texttt{pkg/mnc} output will create separate output subdirectories for each process, assuming option \texttt{mnc_use_outdir} is set to \texttt{TRUE} (here, by specifying \texttt{-np 4} four directories will be created, one for each tile – \texttt{mnc_test_00001} through \texttt{mnc_test_00004} – the first time the model is run). The (global) statistical-dynamical diagnostics output file will be written in only the first of these directories. The \texttt{gluemncbig} steps outlined \textit{above} remain unchanged (if in doubt, one can always tell \texttt{gluemncbig} which specific directories to read, e.g., in bash \texttt{mnc_test_{0009..0012}} will grab only directories 0009, 0010, 0011, 0012). Also note it is no longer necessary to redirect standard output to a file such as \texttt{output.txt}; rather, separate \texttt{STDOUT.xxxx} and \texttt{STDERR.xxxx} files are created by each process, where \texttt{xxxx} is the process number (starting from \texttt{0000}). Other than some additional MPI-related information, the standard output content is similar to that from the single-process run.

### 4.2.6 Running with OpenMP

To run multi-threaded (using shared memory, \texttt{OpenMP}), the original \texttt{SIZE.h} file is used. In our example, for compatibility with MITgcm testing protocols, we will run using two separate threads, but the user should feel free to experiment using four threads if their local machine contains four cores. Like the previous section we must first re-compile the executable from scratch, using a special command line option (for this configuration, \texttt{-omp}). However it is not necessary to specify how many threads at compile-time (unlike MPI, which requires specific processor count information to be set in \texttt{SIZE.h}). Create and navigate into a new build directory \texttt{build_openmp} and type:

\[
\begin{align*}
% &% ../../../tools/genmake2 -mods ../code -omp -of=«/PATH/TO/OPTFILE» \\
% &% make depend \\
% &% make
\end{align*}
\]

In a run directory, overwrite the contents of \texttt{eedata} with file \texttt{verification/tutorial_baroclinic_gyre/input/eedata.mth}. The parameter \texttt{nTy} is changed; we now specify to use two threads across the \(y\)-domain. Since our model domain is subdivided into four tiles, each thread will now integrate two tiles in the \(x\)-domain. Alternatively, to run a multi-threaded example using four threads, both lines should be set to 2.

\[
\begin{align*}
nTx=1, \\
nTy=2,
\end{align*}
\]

To run the model, we first need to set two \textit{environment variables}, before invoking the executable:

\[
\begin{align*}
% &% export OMP_STACKSIZE=400M \\
% &% export OMP_NUM_THREADS=2 \\
% &% ./build_openmp/mitgcmuv >output.txt
\end{align*}
\]

Your system’s \textit{environment variables} may differ from above; see Section 3.6.2 and/or ask your system administrator (also note, above is \texttt{bash} shell syntax; different syntax is required for \texttt{C shell}). The important point to note is that we
must tell the operating system environment how many threads will be used, prior to running the executable. The total number of threads set in OMP_NUM_THREADS must match nTx * nTy as specified in file edata. Moreover, the model domain must be subdivided into sufficient number of tiles in SIZE.h through the choices of nSx and nSy: the number of tiles (nSx * nSy) must be equal to or greater than the number of threads. More specifically, nSx must be equal to or an integer multiple of nTx, and nSy must be equal to or an integer multiple of nTy.

Also note that at this time, pkg/mnc is automatically disabled for multi-threaded setups, so output is dumped in standard binary format (i.e., using pkg/msdio). You will receive a gentle warning message if you run this multi-threaded setup and keep useMNC set to .TRUE. in data.pkg. The full filenames for grid variables (e.g., XC, YC, etc.), snapshot output (e.g., Eta, T, PHL) and pkg/diagnostics output (e.g., surfDiag, oceStDiag, etc.) include a suffix that contains the time iteration number and tile identification (tile 001 includes .001.001 in the filename, tile 002 .002.001, tile 003 .001.002, and tile 004 .002.002). Unfortunately there is no analogous script to utilities/MITgcmutils/scripts/gluemncbig to concatenate raw binary files, but it is relatively straightforward to do so in matlab (reading in files using utilities/matlab/rdmds.m), or equally simple in python – or, one could simply set globalFiles to .TRUE. and the model will output global files for you (note, this global option is not available for pkg/mnc output). One additional difference between pkg/msdio and pkg/mnc is that Diagnostics Per Level Statistics are written in plain text, not binary, with pkg/msdio.

4.2.7 Model solution

In this section, we will examine details of the model solution, using monthly and annual mean time average data provided in diagnostics files dynStDiag.nc, dynDiag.nc, and surfDiag.nc. See companion matlab or python (or python using xarray) script which shows the code to read output netCDF files and create figures shown in this section. Our ocean sector model is forced mechanically by wind stress and thermodynamically though temperature relaxation at the surface. As such, we expect our solution to not only exhibit wind-driven gyres in the upper layers, but also include a deep, overturning circulation. Our focus in this section will be on the former; this component of the solution at the surface. As such, we expect our solution to not only exhibit wind-driven gyres in the upper layers, but also include a deep, overturning circulation. Our focus in this section will be on the former; this component of the solution equilibrates on a time scale of decades, more or less, whereas the deep cell depends on a slower, diffusive timescale. We will begin by examining some of our Diagnostics Per Level Statistics output, to assess how close we are to equilibration at different ocean model levels. Recall we’ve requested these statistics to be computed monthly.

Load diagnostics TRELAX_ave, THETA_lv_avg, and THETA_lv_std from file dynStDiag.nc. In Figure 4.6a we plot the global model surface mean heat flux (TRELAX_ave) as a function of time. At the beginning of the run, we observe that the ocean is cooling dramatically; this is mainly because our ocean surface layer is initialized to a uniform 30°C (as specified here), which results in very strong relaxation initially in the northern portion of ocean model, where the restoring temperature is just above 0°C. (As an aside comment, such large initialization shocks are often best avoided if possible, as they may cause model instability, which may necessitate smaller time steps at model onset and/or more realistic initial conditions.) However, this initial burst of cooling quickly diminishes over the first decade of integration, as the surface layer approaches temperature values close to the specified profile; see Figure 4.6b where the mean temperature at surface, thermocline, and abyssal depth are plotted as a function of time. Note that while the total heat flux shows that the global heat content is slowly decreasing, even after 100 years, the temperature of the deepest water is slowly warming. In Figure 4.6c we plot standard deviation of temperature (by level) over time. Given that each level is initialized at uniform temperature, initially the standard deviation is zero, but should tend to level off at some non-zero value over time, as the solution at each depth equilibrates. Not surprisingly, the largest gradients in temperature exist at the surface, whereas in the abyss the differences in temperature are quite small. In summary, we conclude that while the surface appears to approach equilibrium rapidly, even after 100 years there are changes occurring in deep circulation, presumably related to the meridional overturning circulation. We leave it as an exercise to the reader to integrate the solution further and/or examine and calculate the meridional overturning circulation strength over time.

Next, let’s examine the effect of wind stress on the ocean’s upper layers. Given the orientation of the wind stress and its variation over a full sine wave as shown in Figure 4.5 (crudely mimicking easterlies in the tropics, mid-latitude westerlies, and polar easterlies), we anticipate a double-gyre solution, with a subtropical gyre and a subpolar gyre. Let’s begin by examining the free surface solution (load diagnostics ETAN and TRELAX from file surfDiag.nc). In Figure 4.7 we show contours of free surface height (ETAN; this is what we plotted in our barotropic gyre tutorial solution).
Figure 4.6: a) Surface heat flux due to temperature restoring, negative values indicate heat flux out of the ocean; b) and c) potential temperature mean and standard deviation by level, respectively.
overlaying a 2-D color plot of TRELAX (blue is where heat is released from the ocean, red where heat enters the ocean), averaged over year 100. Note that a subtropical gyre is readily apparent, as suggested by geostrophic currents in balance with the free surface elevation (not shown, but the reader is encouraged to load diagnostics UVEL and VVEL and plot the circulation at various levels). Heat is entering the ocean mainly along the southern boundary, where upwelling of cold water is occurring, but also along the boundary current between 50°N and 65°N, where we would expect southward flow (i.e., advecting water that is colder than the local restoring temperature). Heat is exiting the ocean where the western boundary current transports warm water northward, before turning eastward into the basin at 40°N, and also weakly throughout the higher latitude bands, where deeper mixed layers occur (not shown, but variations in mixed layer depth can be easily visualized by loading diagnostic MXLDEPTH).

So what happened to our model solution subpolar gyre? Let's compute depth-integrated velocity $U_{bt}, V_{bt}$ (units: m² s⁻¹) and use it calculate the barotropic transport streamfunction:

$$U_{bt} = -\frac{\partial \Psi}{\partial y}, \quad V_{bt} = \frac{\partial \Psi}{\partial x}$$

Compute $U_{bt}$ by summing the diagnostic UVEL multiplied by gridcell depth (grid.nc variable drF, i.e., the separation between gridcell faces in the vertical). Now do a cumulative sum of $-U_{bt}$ times the gridcell spacing the in the $y$ direction (you will need to load grid.nc variable dyG, the separation between gridcell faces in $y$). A plot of the resulting $\Psi$ field is shown in Figure 4.8. Note one could also cumulative sum $V_{bt}$ times the grid spacing in the $x$-direction and obtain a similar result.

When velocities are integrated over depth, the subpolar gyre is readily apparent, as might be expected given our wind stress forcing profile. The pattern in Figure 4.8 in fact resembles the double-gyre free surface solution we observed in Figure 4.4 from tutorial Barotropic Ocean Gyre, when our model grid was only a single layer in the vertical.
Is the magnitude of $\Psi$ we obtain in our solution reasonable? To check this, consider the Sverdrup transport:

$$\rho u_{bt} = \hat{k} \cdot \nabla \times \vec{\tau}$$

If we plug in a typical mid-latitude value for $\beta$ ($2 \times 10^{-11}$ m$^{-1}$ s$^{-1}$) and note that $\tau$ varies by 0.1 Nm$^{-2}$ over 15° latitude, and multiply by the width of our ocean sector, we obtain an estimate of approximately 20 Sv. This estimate agrees reasonably well with the strength of the circulation in Figure 4.8.

Finally, let’s examine the model solution potential temperature field averaged over year 100. Read in diagnostic THETA from the file dynDiag.nc. Figure 4.9a shows a plan view of temperature at 220 m depth (vertical level k=4). Figure 4.9b shows a slice in the $xz$ plane at 28.5°N ($y$-dimension j=15), through the center of the subtropical gyre.

The dynamics of the subtropical gyre are governed by Ventilated Thermocline Theory (see, for example, Pedlosky (1996) [Ped96] or Vallis (2017) [Val17]). Note the presence of warm “mode water” on the western side of the basin; the contours of the warm water in the southern half of the sector crudely align with the free surface heights we observed in Figure 4.8. In Figure 4.9b note the presence of a thermocline, i.e., the bunching up of the contours between 200 m and 400 m depth, with weak stratification below the thermocline. What sets the penetration depth of the subtropical gyre? Following a simple advective scaling argument (see Vallis (2017) [Val17] or Cushman-Roisin and Beckers (2011) [CRB11]; this scaling is obtained via thermal wind and the linearized barotropic vorticity equation), the depth of the thermocline $h$ should scale as:

$$h = \left( \frac{w_e f^2 L_x}{\beta \Delta b} \right)^2 = \left( \frac{\tau/L_y f L_x}{\beta \rho'} \right)^2$$

where $w_e$ is a representative value for Ekman pumping, $\Delta b = g \rho' / \rho_0$ is the variation in buoyancy across the gyre, and $L_x$ and $L_y$ are length scales in the $x$ and $y$ directions, respectively. Plugging in applicable values at 30°N, we obtain an estimate for $h$ of 200 m, which agrees quite well with that observed in Figure 4.9b.
4.3 Southern Ocean Reentrant Channel Example

This example experiment simulates flow through a reentrant channel, crudely mimicking the Antarctic Circumpolar Current. The fluid is forced by a zonal wind stress, $\tau_x$, that varies sinusoidally in the north-south direction and is constant in time, and by temperature relaxation at the surface and northern boundary. The grid is Cartesian and the Coriolis parameter $f$ is defined according to a mid-latitude beta-plane equation $f(y) = f_0 + \beta y$; here we choose $f_0 < 0$ to place our domain in the Southern Hemisphere. A linear EOS is used with density only depending on $T$, and there is no sea ice.

Although important aspects of the of the Southern Ocean and Antarctic Circumpolar Current were realized in the early 20th Century (e.g., Sverdrup 1933 [Sve33]), understanding this system has been a major research focus in recent decades. Many significant breakthroughs in understanding its dynamics, role in the global ocean circulation, and role in the climate system have been achieved (e.g., Marshall and Radko 2003 [MR03]; Olbers and Visbeck 2004 [OV04]; Marshall and Speer 2012 [MS12]; Nikurashin and Vallis 2012 [NV12]; Armour et al. 2016 [AMS+16]; Sallée 2018 [Sal18]). Much of this understanding came about using simple, idealized reentrant channel models in the spirit of the model described in this tutorial. The configuration here is fairly close to that employed in Abernathy et al. (2011) [AMF11] (using the MITgcm) with some important differences, such as our introduction of a deep north-south ridge.

We assume the reader is familiar with a basic MITgcm setup, as introduced in tutorial Barotropic Ocean Gyre and tutorial Baroclinic Ocean Gyre. Although the setup here is again quite idealized, we introduce many new features and capabilities of MITgcm. Novel aspects include using MITgcm packages to augment the physical modeling capabilities, discussion of partial cells to represent topography, and an introduction to the layers diagnostics package (/pkg/layers). Our initial focus is on running and comparing coarse-resolution solutions with and without activating the Gent-McWilliams (“GM”) (1990) [GM90] mesoscale eddy parameterization (/pkg/gmredi). As first noted in Danabasoglu et al. (1994) [DMG94], use of GM in coarse resolution models improves global temperature distribution, poleward and surface heat fluxes, and locations of deep-water formation (see also the Gent 2011 [Gen11] perspective on two decades GM usage in ocean models). At the end of this tutorial, we will describe how to increase resolution to an eddy-permitting regime, detailing the few necessary changes in code and parameters, and examine this high-resolution solution. In our discussion, our focus will be on highlighting how the representation of mesoscale eddies plays a significant role in governing the equilibrium state.
Below we describe the idealized configuration in detail (see Figure 4.10). The sinusoidal wind-stress variations are defined thus:

\[ \tau_x(y) = \tau_0 \sin \left( \frac{y}{2L_y} \pi \right), \]

where \( L_y \) is the lateral domain extent and \( \tau_0 \) is set to 0.2 N m\(^{-2}\). Surface temperature restoring varies linearly from 10 °C at the northern boundary to -2 °C at the southern end. A wall is placed at the southern boundary of our domain, thus our setup is only reentrant in the east-west direction. Because MITgcm assumes a periodic domain in both the east-west and north-south directions, our southern wall effectively functions as a wall at the northern boundary as well. The full water column in the northern boundary is a “sponge layer”; relaxing temperature though the full water column will partially constrain the stratification, and in the eddy-permitting solution will absorb any eddies reaching the northern boundary (truly acting as a “sponge”). As shown in Figure 4.10, a north-south ridge runs through the bottom topography, which is otherwise flat with a depth \( H \) of 3980 m. A sloping notch cuts through the middle of the ridge; in the latitude band where the notch exists, potential vorticity \( f/H \) contours are unblocked, which permits a vigorous zonal barotropic jet. Shaved cells are used to represent the topography.

![Figure 4.10: Schematic of simulation domain, bottom topography, and wind-stress forcing function for the idealized reentrant channel numerical setup. A full-depth solid wall at \( y = 0 \) is not shown; because MITgcm is also periodic in the north-south direction, this acts as a wall on the north boundary.](image)

Figure 4.10: Schematic of simulation domain, bottom topography, and wind-stress forcing function for the idealized reentrant channel numerical setup. A full-depth solid wall at \( y = 0 \) is not shown; because MITgcm is also periodic in the north-south direction, this acts as a wall on the north boundary.

Similar to both tutorial Barotropic Ocean Gyre and tutorial Baroclinic Ocean Gyre, we use a linear equation of state which is a function of temperature only (temperature is our only model tracer field). Figure 4.11 shows initial conditions in temperature at the northern and southern end of the domain. Initial temperature decreases exponentially from the

4.3. Southern Ocean Reentrant Channel Example
relaxation SST profile to -2°C at depth \( H \). Note that this same northern boundary profile is used to restore temperature in the model’s sponge layer, as discussed above.

Figure 4.11: Initial conditions in temperature at the northern and southern boundaries. Note this same northern boundary profile is used as relaxation temperature in the model’s sponge layer.

### 4.3.1 Equations Solved

The active set of equations solved is identical to those employed in tutorial *Baroclinic Ocean Gyre* (i.e., hydrostatic with an implicit linearized free surface), except here we use standard Cartesian geometry rather than spherical polar coordinates:

\[
\begin{align*}
\frac{Du}{Dt} - f v + \frac{1}{\rho_c} \frac{\partial p'}{\partial x} + \nabla_h \cdot (-A_h \nabla_h u) + \frac{\partial}{\partial z} \left( -A_z \frac{\partial u}{\partial z} \right) &= F_u \\
\frac{Dv}{Dt} + f u + \frac{1}{\rho_c} \frac{\partial p'}{\partial y} + \nabla_h \cdot (-A_h \nabla_h v) + \frac{\partial}{\partial z} \left( -A_z \frac{\partial v}{\partial z} \right) &= F_v \\
\frac{\partial \eta}{\partial t} + \nabla_h \cdot \left( H \vec{u} \right) &= 0 \\
\frac{D\theta}{Dt} + \nabla_h \cdot (-\kappa_h \nabla_h \theta) + \frac{\partial}{\partial z} \left( -\kappa_z \frac{\partial \theta}{\partial z} \right) &= F_\theta \\
p' &= g \rho_c \eta + \int_0^\alpha g \rho' dz
\end{align*}
\]

Forcing term \( F_u \) is applied as a source term in the model surface layer and zero in the interior, and source term \( F_v \) is zero everywhere. The forcing term \( F_\theta \) is applied as temperature relaxation in the surface layer and throughout the full depth in the two northern-most rows (in the coarse resolution setup) of the model domain.
4.3.2 Discrete Numerical Configuration

The coarse-resolution domain is discretized with a uniform Cartesian grid spacing in the horizontal set to \( \Delta x = \Delta y = 50 \) km, so that there are 20 grid cells in the \( x \) direction and 40 in the \( y \) direction. There are 49 levels in the vertical, ranging from 5.5 m depth at the surface to 149 m at depth. An “optimal grid” vertical spacing here was generated using the hyperbolic tangent method of Stewart et al. (2017) [SHG+17], implemented in Python at https://github.com/kialstewart/vertical_grid_for_ocean_models, based on input parameters of ocean depth (4000 m), minimum (surface) depth (5 m), and maximum depth (150 m). In ocean modeling, it is generally advantageous to have finer resolution in the upper ocean (as was also done previously in tutorial Baroclinic Ocean Gyre), but note that the transition to deeper layers should be done gradually, in the interests of solution fidelity and stability. Although our topography is idealized, the topography is not \textit{a priori} discretized to levels matching the vertical grid, and we make use of MITgcm’s ability to represent “partial cells” (see Section 2.11.6).

Otherwise, the numerical configuration is similar to that of tutorial Baroclinic Ocean Gyre, with an important difference: we use a high-order advection scheme (“7th order one-step method w/ limiter”, tempAdvScheme parameter code 7) for potential temperature instead of center second-ordered differences (which is used in tutorials Barotropic Ocean Gyre and Baroclinic Ocean Gyre and is the model default). This will enable us to use the same numerical scheme in both coarse-resolution and eddy-permitting simulations. Note that this advection scheme does NOT use Adams-Bashforth time stepping for potential temperature, instead using its own time stepping scheme. The fixed flux form of the momentum equations are solved, as described in Section 2.14, with an implicit linear free surface (Section 2.4). Laplacian diffusion of tracers and momentum is employed. The pressure forces that drive the fluid motions, \( \frac{\partial p'}{\partial x} \) and \( \frac{\partial p'}{\partial y} \), are found by summing pressure due to surface elevation \( \eta \) and the hydrostatic pressure, as discussed in Section 4.2.1. The sea-surface height is found by solving implicitly the 2-D (elliptic) surface pressure equation (see Section 2.4).

Additional changes in the numerical configuration for the eddy-permitting simulation are discussed in Section 4.3.5.2.

4.3.2.1 Numerical Stability Criteria

The numerical considerations behind our setup are not trivial. We do not wish the thermocline to be diffused away by numerics. Accordingly, we employ a vertical diffusivity acting on temperature typical of background values observed in the ocean, \( 1 \times 10^{-5} \text{ m}^2 \text{ s}^{-1} \). We now examine numerical stability criteria to help choose and assess parameters for our coarse resolution study: parameters used in the eddy-permitting setup are discussed in Section 4.3.5.2.

We anticipate development of a large barotropic flow through the notch in the topographic ridge which will have implications for the length of the timestep we will be able to use. Let us consider the advective CFL condition (4.24) and the stability of inertial oscillations (4.25):

\[
S_a = 2 \left( \frac{|c_{max}| \Delta t}{\Delta x} \right) < 0.5 \text{ for stability} \tag{4.24}
\]

\[
S_i = f \Delta t < 0.5 \text{ for stability} \tag{4.25}
\]

where \( |c_{max}| \) is the maximum horizontal velocity. We anticipate \( |c_{max}| \) of order \( \sim 1 \text{ ms}^{-1} \). Note that barotropic currents of this speed over a jet of order \( \sim 100 \) km in lateral scale will result in a barotropic flow of the order of hundreds of Sverdups. At a resolution of 50 km, (4.24) then implies that the timestep must be less than 12000 s and (4.25) implies a timestep less than 3500 s. Here we make a conservative choice of \( \Delta t = 1000 \) s to keep \( f \Delta t \) under 0.20.

How shall we set the horizontal viscosity? From the numerical stability criteria:

\[
S_l = 4 A_h \Delta t \left( \frac{1}{\Delta x^2} + \frac{1}{\Delta y^2} \right) < 1.0 \text{ for stability} \tag{4.26}
\]

Note that the threshold in (4.26) is \( < 1.0 \) instead of \( < 0.6 \) due to our specification (in input/data) that momentum dissipation NOT be solved using Adams-Bashforth, as discussed \textit{below}. With \( \Delta t = 1000 \) s, we can choose \( A_h \) to be as
large as order $1 \times 10^5$ m$^2$ s$^{-1}$. However, such a value would result in a very viscous solution. We anticipate a boundary current along the deep ridge and sloping notch on a scale given by Munk scaling:

$$M_w = \frac{2\pi}{\sqrt{3}} \left( \frac{A_h}{\beta} \right)^{\frac{1}{3}}.$$  

We can set $A_h$ to as low as 100 m$^2$ s$^{-1}$ and still comfortably resolve the Munk boundary layer on our grid. However, guided by an ensemble of runs exploring parameter space, we found the solution with $A_h = 100$ s$^{-1}$, while stable, was rather noisy. As a compromise, a value of $A_h = 2000$ m$^2$ s$^{-1}$ reduced solution noise whilst also controlling the strength of the barotropic current. This is the value used here. Also note with this choice $A_h/\Delta x$ gives a velocity scaling of 4 cm/s, a reasonable value.

Regarding the vertical viscosity, we choose to solve this term implicitly (Euler backward time-stepping) by setting implicitViscosity to .TRUE. in input/data, which results in no additional stability constraint on the model timestep (see Section 2.6). Otherwise, given that our vertical resolution is quite fine near the surface (approximately 5 m), the following stability criteria would have applied:

$$S_{lv} = \frac{A_v \Delta t}{\Delta z} < 1.0$$  

which effectively would limit our choice for $A_v$ to very small values. For simplicity, and given that away from the equator coarse resolution models are typically not very sensitive to the value of vertical viscosity, we pick a constant value of $A_v = 3 \times 10^{-3}$ m$^2$ s$^{-1}$ over the full domain, somewhere in between (in geometric mean sense) typical values found in the mixed layer ($\sim 10^{-2}$) and in the deep ocean ($\sim 10^{-4}$) (Roach et al. 2015 [RPBR15]) Note this implicit scheme is also used for vertical diffusion of tracers, for which it can also be used to represent convective adjustment (again, because it is unconditionally stable regardless of diffusivity value).

### 4.3.3 Configuration

The model configuration for this experiment resides under the directory verification/tutorial_reentrant_channel/. The experiment files

- verification/tutorial_reentrant_channel/code/SIZE.h
- verification/tutorial_reentrant_channel/code/LAYERS_SIZE.h
- verification/tutorial_reentrant_channel/code/DIAGNOSTICS_SIZE.h
- verification/tutorial_reentrant_channel/input/data
- verification/tutorial_reentrant_channel/input/data.pkg
- verification/tutorial_reentrant_channel/input/data.gmredi
- verification/tutorial_reentrant_channel/input/data.rbc
- verification/tutorial_reentrant_channel/input/data.layers
- verification/tutorial_reentrant_channel/input/data.diagnostics
- verification/tutorial_reentrant_channel/input/eedata
- verification/tutorial_reentrant_channel/input/bathy.50km.bin
- verification/tutorial_reentrant_channel/input/zonal_wind.50km.bin
- verification/tutorial_reentrant_channel/input/T_surf.50km.bin
- verification/tutorial_reentrant_channel/input/temperature.50km.bin
- verification/tutorial_reentrant_channel/input/T_relax_mask.50km.bin

contain the code customizations and parameter settings for this experiment. Below we describe these customizations in detail.
4.3.3.1 Compile-time Configuration

File code/packages.conf

Listing 4.13: verification/tutorial_reentrant_channel/code/packages.conf

```plaintext
#-- list of packages (or group of packages) to compile for this experiment:
gfd
gmredi
rbcs
layers
diagnostics
```

In addition to the pre-defined standard package gfd, we define four additional packages.

- Package `pkg/gmredi` (see *GMREDI: Gent-McWilliams/Redi SGS Eddy Parameterization*): This implements the Gent and McWilliams parameterization (as first described in Gent and McWilliams 1990 [GM90]) of geostrophic eddies. This mixes along sloping neutral surfaces (here, just $T$ surfaces). It is used instead of large prescribed diffusivities aligned in the horizontal plane (parameter `diffKh`). In Section 4.3.5.1 we will illustrate the marked improvement in the solution resulting from the use of this parameterization.

- Package `pkg/rbcs` (see *RBCS Package*): The default MITgcm code library permits relaxation boundary conditions only at the ocean surface; in the setup here, we relax temperature over the full-depth $xz$ plane along our domain’s northern border. By including the `pkg/rbcs` code library in our model build, we can relax selected fields (tracers or horizontal velocities) in any 3-D location.

We also include two packages which augment MITgcm’s diagnostic capabilities.

- Package `pkg/layers`: This calculates the thickness and transport of layers of specified density (or temperature, or salinity; here, temperature and density are aligned because of our simple equation of state). Further explanation of `pkg/layers` parameter options and output is given below.

- Package `pkg/diagnostics`: This selects which fields to output, and at what frequencies. This was introduced in tutorial *Baroclinic Ocean Gyre*.

File code/SIZE.h

Listing 4.14: verification/tutorial_reentrant_channel/code/SIZE.h

```plaintext
CBOP
C !ROUTINE: SIZE.h
C !INTERFACE:
C include SIZE.h
C !DESCRIPTION: v
C *==========================================================*
C | SIZE.h Declare size of underlying computational grid.
C *==========================================================*
C | The design here supports a three-dimensional model grid
C | with indices I,J and K. The three-dimensional domain
C | is comprised of nPx*nSx blocks (or tiles) of size sNx
C | along the first (left-most index) axis, nPy*nSy blocks
C | of size sNy along the second axis and one block of size
C | Nr along the vertical (third) axis.
C | Blocks/tiles have overlap regions of size OLx and OLy
```

(continues on next page)
Our model tile size is defined above to be $20 \times 10$ gridpoints, so four tiles (i.e., $nSy = 4$) are required to span the full domain in $y$. Note that our overlap sizes ($OLx$, $OLy$) are set to 4 in this tutorial, as required by our choice of advection scheme (see discussion in Section 4.3.2.1 and Table 2.2 from which this required overlap can be obtained); in tutorial Baroclinic Ocean Gyre this was set to 2, which is the minimum required for the default center second-ordered advection scheme.
differences scheme. For this setup we will specify a reasonably high resolution in the vertical, using 49 levels.

**File code/LAYERS_SIZE.h**

Listing 4.15: verification/tutorial_reentrant_channel/code/LAYERS_SIZE.h

```c
C ======================================================================
C * Compiled-in size options for the LAYERS package *
C
C - Just as you have to define Nr in SIZE.h, you must define the number
C of vertical layers for isopycnal averaging so that the proper array
C sizes can be declared in the LAYERS.h header file.
C
C - Variables -
C Nlayers :: the number of isopycnal layers (must match data.layers)
C FineGridFact :: how many fine-grid cells per dF cell
C FineGridMax :: the number of points in the finer vertical grid
C used for interpolation
C layers_maxNum :: max number of tracer fields used for layer averaging

INTEGER Nlayers, FineGridFact, FineGridMax, layers_maxNum
PARAMETER( Nlayers = 37 )
PARAMETER( FineGridFact = 10 )
PARAMETER( FineGridMax = Nr * FineGridFact )
PARAMETER( layers_maxNum = 1 )
```

As noted above in this file’s comments, we must set the discrete number of layers to use in our diagnostic calculations. The model default is 20 layers. Here we set `PARAMETER(Nlayers = 37)` and so choose 37 layers. In making this choice, one needs to ensure sufficiently fine layer bounds in the density (or temperature) range of interest, while also possible to specify fairly coarse bounds in other density ranges. The specific temperatures defining layer bounds will be prescribed in `input/data.layers`.

**File code/DIAGNOSTICS_SIZE.h**

Listing 4.16: verification/tutorial_reentrant_channel/code/DIAGNOSTICS_SIZE.h

```c
C Diagnostics Array Dimension
C ---------------------------
C ndiagMax :: maximum total number of available diagnostics
C numlists :: maximum number of diagnostics list (in data.diagnostics)
C numperlist :: maximum number of active diagnostics per list (data.diagnostics)
C numLevels :: maximum number of levels to write (data.diagnostics)
C numDiags :: maximum size of the storage array for active 2D/3D diagnostics
C nRegions :: maximum number of regions (statistics-diagnostics)
C sizRegMsk :: maximum size of the regional-mask (statistics-diagnostics)
C nStats :: maximum number of statistics (e.g.: aver,min,max ...)
C diagSt_size:: maximum size of the storage array for statistics-diagnostics
C Note: may need to increase "numDiags" when using several 2D/3D diagnostics,
C and "diagSt_size" (statistics-diags) since values here are deliberately small.

INTEGER ndiagMax
INTEGER numlists, numperlist, numLevels
INTEGER numDiags
```

(continues on next page)
Here the parameter `numDiags` has been changed to allow a combination of up to 35 3-D diagnostic fields or 1715 (=35*49) 2-D fields.

### 4.3.3.2 Run-time Configuration

#### File input/data

```bash
# Model parameters
# Continuous equation parameters
&PARM01

# Viscosity
viscAh=2000.,
viscAr=3.E-3,
imPLICITViscosity=.TRUE.,

# Diffusivity and convection
diffKhT=0.,
diffKrT=1.E-5,
ivdc_kappa=1.,
imPLICITDiffusion=.TRUE.,

# Coriolis parameter
f0=-1.363e-4,
beta=1.313e-11,
selectCoriScheme=1,

# Density and equation of state
# Temp only active tracer, no salinity
rhoConst=1035.,
rhoNil=1035.,
eosType='LINEAR',
tAlpha=2.0E-4,
sBeta=0.0E-4,
tRef=49*5.,
saltStepping=.FALSE.,

# activate partial cells
hFacMinDr=5.,
hFacMin=0.1,

# free surface parameters
rigidLid=.FALSE.,
imPLICITFreeSurface=.TRUE.,
exactConserv=.TRUE.,

# advection scheme
tempAdvScheme=7,
```

(continues on next page)
staggerTimeStep=.TRUE.,
#---------------------------------------
# for non-GM coarse run, set horizontal diffusivity non-zero:
# diffKhT=1000.,
#
# for eddy-permitting run, uncomment the following:
# viscC2Leith = 1.,
# useFullLeith=.TRUE.,
# viscAhGridMax = 0.5,
# useSingleCpuIO=.TRUE.,
#- and comment out above statement viscAh=2000.,
&

# Elliptic solver parameters
&PARM02
cg2dTargetResidual=1.E-7,
cg2dMaxIters=1000,
&

# Time stepping parameters
&PARM03
nIter0=0,
nTimeSteps=10,
deltaT=1000.0,
pChkptFreq=31104000.0,
chkptFreq=15552000.0,
dumpFreq=0,
monitorFreq=1200.,
monitorSelect=2,
tauThetaClimRelax=864000.,
momDissip_In_AB=.FALSE.,
#---------------------------------------
#- change monitor frequency for longer run:
# monitorFreq=864000.,
#
#- nTimesteps for 30 yrs, coarse:
# nTimeSteps=933120,
#
#- nTimesteps for 30 yrs, eddy-permitting:
# nTimeSteps=3732480,
#
#- for eddy-permitting run, also need to change timestep:
# deltaT=250.0,
&

# Gridding parameters
&PARM04
usingCartesianGrid=.TRUE.,
delX=20*50.E3,
delY=40*50.E3,
delR= 5.48716549, 6.19462098, 6.99291201, 7.89353689,
8.90937723, 10.05483267, 11.34595414, 12.80056778,
This file, reproduced in its entirety above, specifies the main parameters for the experiment. Parameters for this configuration (shown with line numbers to left) are as follows.

**PARAM01 - Continuous equation parameters**

- These lines set the horizontal and vertical Laplacian viscosities. As in earlier tutorials, we use a spatially uniform value for viscosity in both the horizontal and vertical. We set viscosity to be solved implicitly, using the backward method, as discussed in Section 4.3.2.1.

```plaintext
viscAh=2000.,
viscAr=3.E-3,
implicitViscosity=.TRUE.,
```

- These lines set the horizontal and vertical diffusivities. In the standard (coarse resolution) configuration the Gent-McWilliams parameterization (pkg/gmredi) is activated, and we set the horizontal diffusivity to zero (which is the default value). Similar to tutorial Baroclinic Ocean Gyre, we set a large vertical diffusivity (ivdc_kappa) for mixing unstable water columns, which requires implicit numerical treatment of vertical diffusion.

```plaintext
diffKhT=0.,
diffKrT=1.E-5,
```

(continues on next page)
• The first two lines below set the model’s Coriolis parameters ($f_0$ and $\beta$) to values representative of the latitude band encompassing the Antarctic Circumpolar Current. In the last line we set the model to use the Jamart and Ozer (1986) [JO86] wet-points averaging method, in lieu of the model default (see Section 2.14.2; parameter options here are given in Section 3.8.4). The method affects the discretization of the Coriolis terms in the momentum equations. In this setup – as we will show, the jet is dominated by barotropic potential vorticity conservation – it turns out the solution is rather sensitive to this discretization (particularly adjacent to topography). We tested both the default and wet-points methods, and found the wet-points method closer to the eddy-permitting solution, where obviously the discretization of the Coriolis term is better resolved.

\begin{verbatim}
ivdc_kappa=1.,
implicitDiffusion=.TRUE.,
\end{verbatim}

• These lines set parameters related to the density and equation of state. Here we choose the same value for the Boussinesq reference density $\rho_{\text{Const}}$ as our value $\rho_{\text{Nil}}$, for the linear equation of state. To keep things simple, as well as speed up model run-time, we limit ourselves to a single tracer, temperature, and tell the model not to step salinity forward in time or include salinity in the equation of state. Also note we use a uniform reference temperature ($t_{\text{Ref}}$) throughout the water column. We will be specifying a file for initial conditions of temperature in our simulation, and so $t_{\text{Ref}}$ will not be used for this purpose (as it was in tutorial Baroclinic Ocean Gyre). Thus, $t_{\text{Ref}}$ is only employed here as a reference to compute density anomalies. In principle, one could define $t_{\text{Ref}}$ to a more representative array of values at each level, but for most applications any gain in numerical accuracy is small, and a single representative value suffices.

\begin{verbatim}
f0=-1.363e-4,
beta=1.313e-11,
selectCoriScheme=1,
\end{verbatim}

\begin{verbatim}
rhoConst=1035.,
rhoNil=1035.,
eosType='LINEAR',
tAlpha=2.E-4,
sBeta =0.E-4,
tRef= 49*5.,
saltStepping=.FALSE.,
\end{verbatim}

• These lines activate the use of partial cells, as described in Section 2.11.6. $h_{\text{FacMin}}$=0.1 permits partial cells that are as small as 10% of the full cell depth, but with $h_{\text{FacMinDr}}$=5.0 m this partial cell must also be at least 5 m in depth. Note that the model default of $h_{\text{FacMin}}$=1.0 disables partial cells, i.e., values from a specified bathymetry file are rounded up or down to match grid depth interface levels (model variable $rF$). See also Section 3.8.1.3 for general information on using these parameters and below for additional information about partial cells in this setup.

\begin{verbatim}
hFacMinDr=5.,
hFacMin=0.1,
\end{verbatim}

• These lines activate the implicit free surface formulation (Section 2.4) with the exact conservation option enabled, similar to tutorial Baroclinic Ocean Gyre.

\begin{verbatim}
rigidLid=.FALSE.,
implicitFreeSurface=.TRUE.,
exactConserv=.TRUE.,
\end{verbatim}

• This instructs the model to use a 7th order monotonicity-preserving advection scheme (code 7) – basically, a higher-order, more accurate, less noisy advection scheme – instead of the center-differences, 2nd order model default scheme (code 2). The downside here is additional computations, costly if running with many tracers, and

\section*{4.3. Southern Ocean Reentrant Channel Example}
a larger necessary overlap size in `SIZE.h`, which may get costly if one parallelizes the model into many small tiles. We will use the same scheme for both coarse and eddy-permitting resolutions; using the higher-order scheme is particularly helpful in the high resolution setup. When using non-Adams-Bashforth advection schemes (see Table 2.2), the flag `staggerTimeStep` should be set to `.TRUE.`.

```plaintext
34 tempAdvScheme=7,
35 staggerTimeStep=.TRUE.,
```

**PARM02 - Elliptic solver parameters**

These parameters are unchanged from tutorials *Barotropic Ocean Gyre* and *Baroclinic Ocean Gyre*.

**PARM03 - Time stepping parameters**

- For testing purposes the tutorial is set to integrate 10 time steps, but uncomment the line further down in the file setting `nTimeSteps` to integrate the solution for 30 years.

```plaintext
56 nIter0=0,
57 nTimeSteps=10,
```

```plaintext
# nTimeSteps=933120,
```

- Remaining time stepping parameters are as described in earlier tutorials. See Section 4.3.2.1 for a discussion on our choice of `deltaT`.

```plaintext
58 deltaT=1000.0,
59 pChkptFreq=31104000.0,
60 chkptFreq=15552000.0,
61 dumpFreq=0,
62 monitorFreq=1200.0,
63 monitorSelect=2,
```

- As in tutorial *Baroclinic Ocean Gyre* we set the timescale, in seconds, for relaxing potential temperature in the model’s top layer (note: relaxation timescale for the northern boundary sidewalls is set in `data.rbecs`, not here). Our choice of 864,000 seconds is equal to 10 days.

```plaintext
64 tauThetaClimRelax=864000.0,
```

- This instructs the model to NOT apply Adams-Bashforth scheme to the viscosity tendency and other dissipation terms (such as side grad and bottom drag) in the momentum equations (the default is to use Adams-Bashforth for all terms); instead, dissipation is computed using an explicit, forward, first-order scheme. For our coarse resolution setup with uniform harmonic viscosity, this setting is not strictly necessary (and does not noticeably change results). However, for our eddy-permitting run we will use a difference scheme for setting viscosity, and for stability requires this setting.

```plaintext
65 momDissip_In_AB=.FALSE.,
```
PARM04 - Gridding parameters

- We specify a Cartesian coordinate system with 20 gridpoints in $x$ and 40 gridpoints in $y$, with (default) origin (0,0).

  \begin{verbatim}
  usingCartesianGrid=.TRUE.,
  delX=20*50.E3,
  delY=40*50.E3,
  \end{verbatim}

- We set the vertical grid spacing for 49 vertical levels, ranging from thickness of approximately 5.5 m at the surface to 149 m at depth. When varying cell thickness in this manner, one must be careful that vertical grid spacing varies smoothly with depth; see Section 4.3.2 for details on how this specific grid spacing was generated.

\begin{verbatim}
delR=  5.48716549,  6.19462098,  6.99291201,  7.89353689,
   8.90937723, 10.05483267, 11.34595414, 12.80056778,
  14.43837763, 16.28102917, 18.35210877, 20.67704362,
  23.28285446, 26.19769810, 29.45012046, 33.06792588,
  37.07656002, 41.49691200, 46.34278644, 51.61592052,
  57.30518684, 63.37960847, 69.78661289, 76.44996107,
  83.28285446, 90.13003112, 96.89898027,103.44631852,
 109.65099217,115.41222750,120.65692923,125.34295968,
129.45821977,133.01641219,136.05088105,138.60793752,
140.74074276,142.50436556,143.95220912,145.13372400,
146.40131516,148.72455653,148.98310489,149.18968055,
149.35485820,
\end{verbatim}

PARM05 - Input datasets

- The following lines set file names for the bathymetry, zonal wind forcing, and climatological surface temperature relaxation files (these files are all 2-D fields, see below)

\begin{verbatim}
bathyFile='bathy.50km.bin'
zonalWindFile='zonal_wind.50km.bin',
thetaClimFile='SST_relax.50km.bin',
\end{verbatim}

- This last line specifies the name of the 3-D file containing initial conditions for temperature (as noted above, $t_{Ref}$ values specified in namelist PARM01 are NOT used for the initial state).

\begin{verbatim}
hydrogThetaFile='temperature.50km.bin',
\end{verbatim}

File input/data.pkg

Listing 4.18: verification/tutorial_reentrant_channel/input/data.pkg

\begin{verbatim}
# Packages
&PACKAGES
useGMRedi=.TRUE.,
useRBCS=.TRUE.,
useLayers=.TRUE.,
useDiagnostics=.TRUE.,
\end{verbatim}

(continues on next page)
# for non-GM or eddy-permitting run, deactivate GMRedi package:
# useGMRedi=.FALSE.,
&

• These first two lines affect the model physics packages we’ve included in our build, pkg/gmredi and pkg/rbcs. In
our standard configuration, we will activate both (but in an second run, we will opt to NOT activate pkg/gmredi).

useGMRedi=.TRUE.,
useRBCS=.TRUE.,

• These lines instruct the model to activate both diagnostics packages we’ve included in our build, pkg/layers and
pkg/diagnostics.

useLayers=.TRUE.,
useDiagnostics=.TRUE.,

File input/data.gmredi

Listing 4.19: verification/tutorial_reentrant_channel/input.GM/data.gmredi

# GM-Redi package parameters:
# GM_background_K: GM diffusion coefficient
# GM_taper_scheme: slope clipping or one of the tapering schemes
&GM_PARM01
GM_background_K = 1000.,
GM_taper_scheme = 'dm95',
GM_AdvForm = .TRUE.,
&

Note that this file is ignored with pkg/gmredi disabled (in input/data.pkg, useGMRedi=.FALSE.), but must be present
when enabled. Parameter choices are as follows.

• Parameter background_K sets the Gent-McWilliams “thickness diffusivity”, which determines the strength of the
parameterized geostrophic eddies in flattening sloping isopycnals surfaces. By default, this parameter is also used
as diffusivity for the Redi component of the parameterization, which diffuses tracers along isoneutral surfaces.
It is possible to set the Redi diffusivity to a separate value from the thickness diffusivity by setting parameter
GM_isopycK in the above list. However, in this setup with a single tracer determining density, it would not serve
any purpose because diffusion of temperature along surfaces of constant temperature has no impact.

GM_background_K = 1000.,

• By default, pkg/gmredi does not select a tapering scheme (see Section 8.4.1.5); however, for best results, one
should be selected. Here we choose the tapering approach described in Danabasoglu and McWilliams (1995)
[DJCM95]. Additional choices for the tapering scheme (or alternatively, the more simple slope clipping ap-
proach), and why such a scheme is necessary, are described in the GMRedi package documentation.

GM_taper_scheme = 'dm95',

• We select the advective or “bolus” form of the parameterization, which specifies that GM fluxes are parameterized
into a bolus advective transport, rather than implemented as a “skewflux” transport via added terms in the
diffusion tensor (see Griffies 1998 [Gri98]). The skewflux form is the package default. Analytically, these forms are identical, but in practice are discretized differently. For instance, the bolus form will, by default, advect tracers with combined eulerian and bolus transport (i.e. residual transport) which then inherits the higher order precision of the selected advection scheme 7. This can lead to noticeably different solutions in some setups (anecdotally, particularly where you have steeply sloping isopycnals near boundaries). For diagnostic purposes, the bolus form permits a straightforward calculation of the actual advective transport (from the GM part), whereas obtaining this transport using the skewflux form is less straightforward due to discretization issues.

File input/data.rbcs

Listing 4.20: verification/tutorial_reentrant_channel/input/data.rbcs

```plaintext
# RBCS package parameters:
&RBCS_PARM01
useRBCtemp=.TRUE.,
tauRelaxT=864000.,
relaxMaskFile='T_relax_mask.50km.bin'
relaxTFile='temperature.50km.bin',
-----------------------------
#- for eddy-permitting run, use files generated by gendata_5km.m:
# relaxMaskFile='T_relax_mask.5km.bin'
# relaxTFile='temperature.5km.bin',
&
# RBCS for pTracers (read this namelist only when ptracers pkg is compiled)
&RBCS_PARM02
&
```

Setting parameter useRBCtemp to .TRUE. instructs pkg/rbcs that we will be restoring temperature (and by default, it will not restore salinity, nor velocity, nor any other passive tracers). tauRelaxT sets the relaxation timescale for 3-D temperature restoring to 864,000 s or 10 days. The remaining two parameters are a filename for a 3-D mask of gridpoint locations to restore (relaxMaskFile), and a filename for a 3-D field of restoring temperature values (relaxTFile). See below for further description of these fields.

File input/data.layers

Listing 4.21: verification/tutorial_reentrant_channel/input/data.layers

```plaintext
&LAYERS_PARM01
layers_name(1) = 'TH',
layers_bounds(1:38,1) = 2.00, -1.75, -1.50, -1.25,
-1.00, -0.75, -0.50, -0.25,
0.00, 0.25, 0.50, 0.75,
1.00, 1.25, 1.50, 1.75,
2.00, 2.25, 2.50, 2.75,
3.00, 3.25, 3.50, 3.75,
4.00, 4.25, 4.50, 5.0,
5.5, 6.0, 6.5, 7.0,
7.5, 8.0, 8.5, 9.0,
(continues on next page)
```

4.3. Southern Ocean Reentrant Channel Example
pkg/layers consists of online calculations which separate water masses into specified layers, either by temperature, salinity, or density. Note that parameters here include an array index of 1; it is possible to diagnose layers in both temperature and salinity simultaneously, for example, in which case one would add a second set of parameters with array index 2. Even though layers_maxNum is set to 1 (i.e., only allows a for single layers coordinate) in LAYERS_SIZE.h, the index is still required.

- The parameter layers_name is set to 'TH' which specifies temperature as our layers coordinate.

```
layers_name(1) = 'TH',
```

- Parameter layers_bounds specifies the discretization of the layers coordinate system; we span from the lowest possible model temperature (i.e., the coldest restoring temperature at the surface or northern boundary, -2 °C) to the warmest model temperature (i.e., the warmest restoring temperature, 10 °C). The number of values here must be Nlayers + 1, as specified in LAYERS_SIZE.h. Here, Nlayers is set to 37, so we have 38 discrete layers_bounds). pkg/layers will not complain if the discretization does not span the full range of existing water in the model ocean; it will simply ignore water masses (and their transport) that fall outside the specified range in layers_bounds (this will make it impossible however to close the layer volume budget). Also note that the range must be monotonically increasing, even if this results in a layers coordinate k=1:Nlayers that proceeds in the opposite sense as the depth coordinate (i.e., the k=1 layers coordinate is at the ocean bottom, whereas the k=1 depth coordinate refers to the ocean surface layer).

```
layers_bounds(1:38,1) = -2.00, -1.75, -1.50, -1.25,
-1.00, -0.75, -0.50, -0.25,
 0.00, 0.25, 0.50, 0.75,
 1.00, 1.25, 1.50, 1.75,
 2.00, 2.25, 2.50, 2.75,
 3.00, 3.25, 3.50, 3.75,
 4.00, 4.25, 4.50, 5.0,
 5.5, 6.0, 6.5, 7.0,
 7.5, 8.0, 8.5, 9.0,
 9.5, 10.0,
```

### File input/data.diagnostics

Listing 4.22: verification/tutorial_reentrant_channel/input/data.diagnostics

```
# Diagnostic Package Choices
#----------------------
# dumpAtLast (logical): always write output at the end of simulation (default=F)
# diag_mnc  (logical): write to NetCDF files (default=useMNC)
#--for each output-stream:
# fileName(n) : prefix of the output file name (max 80c long) for outp.stream n
# frequency(n):< 0 : write snap-shot output every |frequency| seconds
# > 0 : write time-average output every frequency seconds
# timePhase(n) : write at time = timePhase + multiple of |frequency|
# averagingFreq : frequency (in s) for periodic averaging interval
# averagingPhase : phase (in s) for periodic averaging interval
# repeatCycle : number of averaging intervals in 1 cycle
```

(continues on next page)
13  # levels(:,n) : list of levels to write to file (Notes: declared as REAL)
14  # when this entry is missing, select all common levels of this list
15  # fields(:,n) : list of selected diagnostics fields (8.c) in outp.stream n
16  # (see "available_diagnostics.log" file for the full list of diags)
17  # missing_value(n) : missing value for real-type fields in output file "n"
18  # fileFlags(n) : specific code (8c string) for output file "n"
19  #---------------------
20  &DIAGNOSTICS_LIST
21  # write pkg diagnostics output to separate subdirectory
22  diagMdsDir = 'Diags'
23  # 2D diagnostics
24  fields(1:3,1) = 'TRELAX ', 'MXLDEPTH', 'ETAN ',
25  frequency(1) = 31104000.,
26  filename(1) = '2D_diags',
27  # 3D state variables
28  fields(1:5,2) = 'THETA ', 'VVEL ', 'UVEL ',
29     'WVEL ', 'CONVADJ ',
30  frequency(2) = 31104000.,
31  filename(2) = 'state',
32  # Heat budget terms
33  fields(1:7,3) = 'ADVx_TH ', 'ADVy_TH ', 'ADVr_TH ',
34     'DFxE_TH ', 'DFyE_TH ', 'DFrI_TH ',
35     'DFrE_TH ',
36  frequency(3) = 31104000.,
37  filename(3) = 'heat_3D',
38  # Residual mean flow - Layers Package
39  fields(1:3,4) = 'LaVH1TH ', 'LaHs1TH ', 'LaVa1TH '
40  frequency(4) = 31104000.,
41  fileName(4) = 'layDiag',
42  # GM diagnostics
43  # - Note: comment out this diagnostics list below if useGMRedi=.FALSE.
44  # or you will get warnings messages in STDERR
45  fields(1:2,5) = 'GM_PsiX ', 'GM_PsiY ',
46  frequency(5) = 31104000.,
47  filename(5) = 'GM_diags',
48  #-------------------
49  # Eddy-permitting run, diagnose vorticity (not computed when using uniform Ah)
50  # fields(1:2,6) = 'momVort3', 'momHDiv ',
51  # frequency(6) = 31104000.,
52  # filename(6) = 'state_vort',
53  &
54  #-------------------
55  # Parameter for Diagnostics of per level statistics:
56  #-------------------
57  # diagSt_mnc (logical): write stat-diags to NetCDF files (default=diag_mnc)

(continues on next page)
# diagSt_regMaskFile : file containing the region-mask to read-in
# nSetRegMskFile : number of region-mask sets within the region-mask file
# set_regMask(i) : region-mask set-index that identifies the region "i"
# val_regMask(i) : region "i" identifier value in the region mask
#--for each output-stream:
# stat_fName(n) : prefix of the output file name (max 80c long) for outp.stream n
# stat_freq(n)< 0 : write snap-shot output every |stat_freq| seconds
# > 0 : write time-average output every stat_freq seconds
# stat_phase(n) : write at time = stat_phase + multiple of |stat_freq|
# stat_region(:,n) : list of "regions" (default: 1 region only=global)
# stat_fields(:,n) : list of selected diagnostics fields (8.c) in outp.stream n
# (see "available_diagnostics.log" file for the full list of diags)

&DIAG_STATIS_PARMS
stat_fName(1) = 'dynStDiag',
stat_freq(1) = 864000.,
stat_fields(1:2,1) = 'THETA ','TRELAX ',

&DIAGNOSTICS_LIST - Diagnostic Package Choices
See tutorial Baroclinic Ocean Gyre for a detailed explanation of parameter settings to customize data.diagnostics to a desired set of output diagnostics.

We have divided the output diagnostics into several separate lists (recall, 2-D output fields cannot be mixed with 3-D fields!!!) The first two lists are quite similar to what used in tutorial Baroclinic Ocean Gyre: specifically, several key 2-D diagnostics are in one file (surface restoring heat flux, mixed layer depth, and free surface height), and several 3-D diagnostics and state variables in another (theta, velocity components, convective adjustment index).

In diagnostics list 3, we specify horizontal advective heat fluxes (ADVx_TH and ADVy_TH in x and y directions, respectively), vertical advective heat flux (ADVr_TH), horizontal diffusive heat fluxes (DFxE_TH and DFyE_TH), and vertical diffusive heat flux (DFrI_TH and DFrE_TH). Note the latter is broken into separate implicit and explicit components, respectively, the latter of which will only be non-zero if pkg/gmredi activated. Although we will not examine these 3-D diagnostics below when describing the model solution, the zonal terms are needed to compute zonally-averaged meridional heat transport, and all terms needed for a diagnostic attempt at reconciling a heat budget of the model solution.

fields(1:7,3) = 'ADVx_TH ','ADVy_TH ','ADVr_TH ',
'DFxE_TH ','DFyE_TH ','DFrI_TH ',
'DFrE_TH ',
frequency(3) = 31104000.,
filename(3) = 'heat_3D',

In diagnostics list 4, we specify several pkg/layers diagnostics. In our setup we use a linear equation of state based solely on temperature, so we will diagnose layers of temperature in the model solution, as shown in Figure 4.12.

fields(1:3,4) = 'LaVH1TH ','LaHs1TH ','LaVa1TH '
frequency(4) = 31104000.,
fileName(4) = 'layDiag',

Diagnostic LaVH1TH is the integrated meridional mass transport in the layer: here we request an annual mean time average (via the frequency parameter setting), which will effectively output the quantity $\bar{v} h$ (m$^2$ s$^{-1}$). LaHs1TH is the layer thickness $h$ (m) calculated at “v” points (see Section 2.11.4). LaVa1TH is the layer average meridional velocity $v$.
These diagnostics are all 3-D fields, albeit the vertical dimension here is the layer discretization in temperature space, which was defined in data.layers. See Section 4.3.5.1 for examples using these diagnostics to calculate the residual circulation and the meridional overturning circulation in density coordinates.

**DIAG_STATIS_PARMS - Diagnostic Per Level Statistics**

Here we specify statistical diagnostics of potential temperature and surface relaxation heat flux, output every ten days, to assess how well the model has equilibrated. See tutorial Baroclinic Ocean Gyre for a more complete description of syntax and output produced by these diagnostics.

**File input/eedata**

This file uses standard default values (single-threaded) and does not contain customizations for this experiment.

**File input/bathy.50km.bin**

This is a 2-D(x, y) map of bottom bathymetry, as generated by the MATLAB program verification/tutorial_reentrant_channel/input/gendata.50km.m or the Python script verification/tutorial_reentrant_channel/input/gendata.50km.py. Input files are 32-bit single precision, by default. Our bathymetry file has active ocean grid cells along both the eastern and western boundaries (i.e., no land points or walls are present along either boundary), and thus our model will be fully zonally reentrant. While our northern boundary also consists entirely of active ocean points, we prescribe a wall along the southern end of our model domain, therefore the model is NOT meridionally reentrant.

Unlike in previous examples, where the bathymetry was discretized to match depths of defined vertical grid faces (rF, see Figure 2.9), we have a more complicated bottom bathymetry as defined using a sine function for our bottom ridge. The model default in such case is to round the bathymetry up or down to the nearest allowed vertical cell face level. However, the model permits the use of “partial cells” (sometimes also referred to as “shaved” or “lopped” cells), which can provide dramatic improvements in model solution (see Adcroft et al. 1997 [AHM97]). Here, we activate partial cells though parameter choices hFacMin and hFacMinDr in input/data, as discussed above. The fraction of a vertical cell that contains fluid is represented in the 3-D output variable hFacC, which will have a value of 0.0 beneath the ocean floor (and at land points), 1.0 at an active full-depth ocean cell, and a number between hFacMin and 1.0 for a partial ocean cell. As such, hFacC is often quite useful as a “mask” when computing diagnostics using model output.

As an example, consider horizontal location (10,15) in our setup here, located in our bottom ridge along the sloping notch. In our bathymetry file, the vertical level is specified as -2382.3 m. This falls between vertical faces located at -2360.1 and -2504.0 [these are grid variable rF(39:40)]. Thus, this grid cell will be included in the active ocean domain as a thin, yet legal, partial cell: hFacC(10,15,39)=0.154.

---

**4.3. Southern Ocean Reentrant Channel Example**
File \texttt{input/zonal\_wind.50km.bin}, \texttt{input/SST\_relax.50km.bin}

These files are 2-D($x, y$) maps of zonal wind stress $\tau_x$ (Nm$^{-2}$) and surface relaxation temperature (°C), as generated by program \texttt{verification/tutorial\_reentrant\_channel/input/gendata.50km.m} or \texttt{verification/tutorial\_reentrant\_channel/input/gendata.50km.py}. Note that a 2-D($x, y$) file is expected even though as specified, both $\tau_x$ and SST field are only $f(y)$.

File \texttt{input/temperature.50km.bin}

This file specifies a 3-D($x, y, z$) map of temperature (°C), as generated by \texttt{verification/tutorial\_reentrant\_channel/input/gendata.50km.m} or \texttt{verification/tutorial\_reentrant\_channel/input/gendata.50km.py} (see Figure 4.11). Note again a 3-D($x, y, z$) file is expected despite temperature begin only $f(y, z)$. This file is used here for two purposes: first, as specified in \texttt{input/data}, these values are used for temperature initial conditions; secondly, this file was also specified in \texttt{input/data.rbc} as a 3-D field used for temperature relaxation purposes.

File \texttt{input/T\_relax\_mask.50km.bin}

This file specifies a 3-D($x, y, z$) mask, as required by /pkg/rbcs to inform the model which gridpoints to relax. These values should be between 0.0 and 1.0, with 0.0 for no restoring, 1.0 for full restoring, with fractional values as a multiplicative factor to effectively weaken restoring at that location (see Section 8.3.2). Here, we select a value of 1.0 along the model northern wall for all sub-surface depths (relaxation at the surface is specified using \texttt{input/SST\_relax.50km.bin}, otherwise you would be restoring the surface layer twice), and use a fractional value for the $xz$ plane of grid cells just south of the northern border (see verification/tutorial\_reentrant\_channel/input/gendata.50km.m or verification/tutorial\_reentrant\_channel/input/gendata.50km.py).

### 4.3.4 Building and running the model

This model can be built and run using the standard procedure described in Section 3.5 and Section 3.6. (see also README).

For testing purposes the model is set to run 10 time steps. For a reasonable solution, we suggest running for 30 years, which requires changing \texttt{nTimeSteps} to 933120. When making this edit, also change \texttt{monitorFreq} to something more reasonable, say 10 days (=864000s). Using a single processor core, it should take 12 hours or so to run 30 years; to speed this up using MPI, re-compile using \texttt{nPy=4}, and \texttt{nSy=1}, in SIZE.h and recompile with the -mpi flag (see Section 3.6.1 for instructions how to run using MPI, here you will be using 4 cores). As an exercise, see if you can speed it up further using additional processor cores, e.g., by decreasing the tile size in $x$ and increasing \texttt{nPx}.

As configured, the model runs with pkg/gmredi activated, i.e., \texttt{useGMRedi=.TRUE.} in data.pkg. In Section 4.3.5.1 we will also examine a model solution using old-fashioned large horizontal diffusion with pkg/gmredi deactivated. The same executable can be used for the non-GM run. Set \texttt{useGMRedi=.FALSE.} in data.pkg, and also set \texttt{diffKhT=1000}, in data namelist PARM01. Also, comment out the lines for diagnostics list 5 in data.diagnostics or you will get (non-fatal) warning messages in STDERR.

In Section 4.3.5.2 we will present results with the resolution increased by an order of magnitude, eddy-permitting. Additional required changes to the code and parameters are discussed.
4.3.5 Model Solution

Our primary focus in this section is physical interpretation of the model solution, not how to generate plots from MITgcm output, and thus in parallel we strongly recommend carefully going through our Python analysis code, documented in Jupyter Notebook format, see verification/tutorial_reentrant_channel/analysis/py_notebook.ipynb. This notebook reads in grid data, discusses (and plots) the setup and forcing data files in additional detail, and generates figures shown in the tutorial. MATLAB analysis code to generate tutorial output figures is available at verification/tutorial_reentrant_channel/analysis/matlab_plots.m.

4.3.5.1 Coarse Resolution Solution

Before examining the circulation and temperature structure of the solution, let’s first assess whether the solution is approaching a quasi-equilibrium state after 30 years of integration. Typically, one might expect a solution given this setup to equilibrate over a timescale of a hundred years or more, given the depth of the domain and the prescribed weak vertical diffusivity. As in tutorial Baroclinic Ocean Gyre, we will make use of the ‘Diagnostic Per Level Statistics’ to assess equilibrium; specifically, we will look at the change in surface (restoring) heat flux over time, as well as the potential temperature field. In this tutorial we use standard native Fortan (binary) output files (using pkg/mdsio) rather than netCDF output (as done in tutorial Baroclinic Ocean Gyre). Important note: when using pkg/mdsio, the statistical diagnostics output is written in plain text, NOT binary format. An advantage is that this permits a simple unix cat or more command to display the file to the terminal window as integration proceeds, i.e., for a quick check that results look reasonable. The disadvantage however is that some additional parsing is required (when using MATLAB) to generate some plots using these data. Making use of MITgcm shell script utils/scripts/extract_StD, in a terminal window (in the run directory) type

```
% ../../../utils/scripts/extract_StD dynStDiag.0000000000.txt STATDIAGS dat
```

where dynStDiag.0000000000.txt is the name of our statistical diagnostics output file, STATDIAGS is a name we chose for files generated by running the script, with extension dat. This shell script extracts data into the following (plain text) files:

- STATDIAGS_head.dat - header file containing metadata
- STATDIAGS_Iter.dat - list of iteration numbers for which statdiags dumped
- STATDIAGS_THETA.dat - statdiags for field THETA (diagnostic field specified in input/data.diagnostics)
- STATDIAGS_TRELAX.dat - statdiags for field TRELAX (diagnostic field specified in input/data.diagnostics)

The files STATDIAGS_Iter.dat and STATDIAGS_«DIAGNAME».dat are simple column(s) of data that can be loaded or read in as an array of numbers using any basic analysis tool. Here we will make use of another MITgcm utility, utils/matlab/Read_StD.m, which uses MATLAB to make life a bit more simple for reading in all statistical diagnostic data. In a MATLAB session type

```
>> [nIter,regList,time,stdiagout,listFlds,listK]=read_StD('STATDIAGS','dat','all_flds');
```

where

- nIter = number of iterations (i.e., time records) dumped
- regList = list of region numbers (=0 here, as we did not define any regions, by default global output only)
- time(:,1) = iteration numbers ; time(:,2) = time in simulation (seconds)
- listFlds = list of fields dumped
- listK = for each field, lists number of k levels dumped
- stdiagout = 5 dimensional output array ( kLev, time_rec, region_rec, [ave,std,min,max,vol], fld_rec ) where kLev=1 is depth-average, kLev=2:50 is for depths rC(1:49)
A function to parse statistical diagnostics MITgcm output is also available in the python package *MITgcmutils*. Executing the python command

```python
stdiags_bylev, stdiags_2D, iters = readstats('dynStDiag.0000000000.txt')
```

will load up the level-by-level statistical diagnostics into `stdiags_bylev` (e.g., `stdiags_bylev['THETA'][:,:,0]` is the time series for top level average temperature), `stdiags_2D` given column-integrated or 2-D fields (e.g., `stdiags_2D['TRELAX'][:,:,0]` is the time series for surface heat flux), and `iters` is iteration number for the time series (e.g. `iters['TRELAX']` is a series of iteration numbers for the THETA diagnostic, the user is left to convert into time units). See the *MITgcmutils* documentation for more information.

On the left side of Figure 4.13 we show time series of global surface heat flux. In the first decade there is rapid adjustment, with a much slower trend in both mean and standard deviation in years 10-30. In the mean there remains a significant heat flux into the ocean in the run without GM (solid), whereas with GM (dashed) the net heat uptake is also positive, but smaller. The panels on the right show potential temperature at the surface, mid-level (270 m) and at depth. Note in particular the warming trend at depth in the run without GM. The SST series display a much less obvious trend (as might be expected given rapid restoring of SST). Examining these results, we see that after 30 years our run is not at full equilibrium, presumably due to the long timescale for vertical diffusion. And, we infer that less surface heating is penetrating to depth in the GM solution. This difference is also obvious in Figure 4.14 where we plot zonal mean temperature: note the deeper thermocline in the left panel (without GM), in addition to the deeper mixed layer (and warmer surface) in the southern half of the model domain. The differences in convective adjustment are remarkable, as shown in Figure 4.15; here we plot a plan view of diagnostic CONVADJ, which is the fraction of the time steps a grid cell is convectively unstable, at 92 m depth. Note that at this depth, convection is limited to grid cells near the southern boundary in the GM run, whereas a significant portion of the domain is convecting in the non-GM run: as discussed in Gent (2011) [Gen11], the Deacon cell advects cold water northward at the surface, resulting in unstable water columns and excessively deep mixed layers. Clearly, the temperature structure of the model solution is sensitive to our mesoscale eddy parameterization (we will explore this further).

Figure 4.16 shows the barotropic streamfunction without GM (left) and with GM (right). The pattern is quite similar in both simulations, characterized by a jet centered in the latitude bands with the deep notch, with some deflection to the south after the jet squeezes through the notch. There is a balance between negative relative vorticity, as the jet curves northward through the notch and then southward again, and increasing $f$ to the north (from the beta-plane) such that barotropic potential vorticity is conserved. North of the notch, we see in Figure 4.14 the ocean is much more stratified, with dynamics presumably more baroclinic.

Figure 4.17 shows the Eulerian meridional overturning circulation for the non-GM run (left) and GM run (right). Again, they appear quite similar; what we are observing here is known as a “Deacon Cell” (Deacon 1937 [Dea37]; Bryan 1991 [Bry91]) forced by surface Ekman transport to the north (see also Döös and Webb 1994 [DW94], Speer et al. 2000 [SS00]), with downwelling in the northern half of the basin and upwelling in the south. The magnitude of this cell, on the order of 1-2 Sverdrups, may not seem very impressive, but it is important to consider our zonal domain spans only about 1/20th of the 60th parallel south; scaled up, the magnitude of this cell is quite large. Some local recirculation occurs in the latitude bands where the ridge slopes down to the center of the deep notch. The centers of these recirculations occur in the bottom 2000 m, where stratification is quite weak, so much of water recirculated here falls within a very narrow density class. The deep ridge effectively creates east-west sidewalls at depth, thus able to support an overturning in thermal wind balance, whereas no sidewalls exist in the upper portion of the water column. There is little overturning associated with the deep jet flowing through the flat bottom of the notch.

Also worth noting is that we see some evidence of noise (jaggedy contours) in Figure 4.17, despite our rather large choice of $A_h=2000$ m$^2$ s$^{-1}$ for (uniform) horizontal viscosity and our higher-order advective scheme. These noise artifacts increase fairly dramatically for smaller choices of $A_h$, although we tested the solution remains stable for $A_h$ decreased by an order of magnitude.
Figure 4.13: Left: time series of area-integrated heat flux into the surface ocean (blue) and its standard deviation (magenta). Right: area-mean temperature at the surface (top, cyan), in the thermocline (middle, green), and at depth (bottom, red). In all panels, solid curves show non-GM run, dashed curves include GM.

Figure 4.14: Zonal-mean temperature (shaded) and zonal-mean mixed layer depth (black line) averaged over simulation year 30. Left plot is from non-GM run, right using GM.
Figure 4.15: Convective adjustment index: 0 = never convectively unstable during year 30, 1 = always convectively unstable. Left plot is from non-GM run, right using GM.
Figure 4.16: Barotropic streamfunction averaged over simulation year 30. Left plot is from non-GM run, right using GM. Contour interval is 20 Sv.

Figure 4.17: Eulerian meridional overturning circulation (shaded) averaged over simulation year 30. Left plot is from non-GM run, right using GM. Contour interval is 0.5 Sv.
When using pkg/gmredi, it is often desirable to diagnose an eddy bolus velocity, or a bolus transport, in order to compute the residual circulation (Ferrari 2003 [FP03]), the Lagrangian transport in the ocean (i.e., which effects tracer transport; see, for example, Wolfe 2014 [Wol14]). Unfortunately the bolus velocity is not directly available from MITgcm, but must be computed from other GM diagnostics, which differ if the skew flux or bolus/advective form of GM is selected. Here we choose the later form in data.gmredi (GM_AdvForm = .TRUE.), for which a bolus streamfunction diagnostic is available, thus the bolus velocity can be readily computed (see matlab_plots.m; obtaining the bolus velocity, for reasons of gridding, is a bit more straightforward using the advective form). In Figure 4.18 we’ve computed and added the bolus velocity to the Eulerian velocity. We see that the upper meridional overturning cell has weakened in magnitude, particularly in the northern half of the domain. The eddy parameterization will attempt to flatten sloping isopycnals seen in Figure 4.14, creating a bolus overturning circulation in the opposite sense to the Deacon Cell. The magnitude of the GM thickness diffusion effectively controls the strength of the eddy transport; here we observed only partial cancellation of the Deacon Cell shown in Figure 4.17. In global ocean general circulation models, an observation of near-cancellation in the Southern Ocean Deacon Cell when the GM parameterization was used was first reported in Danabasoglu et al. (1994) [DMG94].

![Figure 4.18: Meridional overturning circulation (shaded) from GM simulation including bolus advective transport, averaged over simulation year 30. Contour interval is 0.5 Sv.](image)

Now let’s use pkg/layers output to examine the residual meridional overturning circulation, shown in Figure 4.19. We integrate the time- and zonal-mean transport in isopycnal layers (see Figure 4.12) to obtain a streamfunction in density coordinates. See Abernathy et al. (2011) [AMF11] for a more detailed explanation of this calculation; this approach is the tried-and-true method to diagnose the residual circulation in an eddy-permitting regime, as required when we run this setup at higher resolution (Section 4.3.5.2). Note that pkg/layers automatically includes bolus transport from pkg/gmredi in its calculations, assuming GM is used. With temperature as the ordinate in Figure 4.19, vertical flows reflect diabatic processes. The green dashed lines represent the maximum and minimum SST for a given latitude band, thus representing upper layer circulation within this band. On the left side, without GM, we again see a robust Deacon cell, with a strong diabatic component, presumably due to horizontal diffusion occurring across sloping isopycnals (i.e. the so-called “Veronis effect”, see Veronis (1975) [Ver75] as well as other numerous papers prior to the widespread adoption of the GM parameterization in ocean models). [As an aside, it is for lack of a better name that we label this left plot of Figure 4.19, lacking either eddies or GM, as the residual circulation, as indeed it is identical to the Eulerian circulation in density coordinates]. On the right side, with GM, the Deacon cell is much weaker due to partial cancellation from the bolus circulation, as noted earlier, but also note that interior contours of streamfunction run roughly horizontal in the plot. We see some evidence of a deep cell in the lowest temperature classes, less obvious in the Eulerian MOC Figure 4.17. One might ask: what happened to the deep recirculating cells seen in Figure 4.18? Recall that our discretization of temperature layers is fairly crude, 0.25 K in the coldest temperatures, and presumably much of this recirculation is “lost” as recirculation within a single density class. If this deep circulation were of interest,
one could simply re-run the model with finer resolution at depth (perhaps increasing the number of layers used, which requires changing LAYERS_SIZE.h and recompiling).

![Residual MOC (Sv): w/o GM](image1)

![Residual MOC (Sv): w/GM](image2)

Figure 4.19: Residual meridional overturning circulation (shaded) as computed in density (i.e., temperature) coordinates, averaged over simulation year 30. Contour interval is 0.5 Sv. Green dashed curves show maximum and minimum SST in each latitude band. Left plot is from non-GM run, right using GM.

Finally, let’s convert the residual circulation shown in Figure 4.19 back into depth coordinates, see Figure 4.20. Solid lines now display contours of zonal mean temperature. On the left, consistent with previous analyses, we see a small, upper ocean counter-clockwise circulation in the southern sector, where deep mixed layers occur (Figure 4.14), with the dominant feature again being the (clockwise) Deacon cell. In contrast, using GM, we see a weak residual clockwise cell aligned along temperature surfaces in the thermocline, with a weak deep counter-clockwise cell aligned with the coldest temperature contour (i.e., the deep cell seen in Figure 4.19).

4.3.5.2 Eddy Permitting Solution
In this section we discuss a model solution with the horizontal grid space reduced from 50 km to 5 km, which is sufficiently resolved to permit eddies to form (see above, which shows SST, surface relative vorticity, and surface current speed, left to right, toward the end of the 30-year simulation). Vertical resolution is unchanged. While we provide instructions on how to compile and run in this new configuration, it will require parallelizing (using MPI) on at least a hundred processor cores or else a 30-year integration will take on the order of a month or longer – in other words, this requires a large cluster or high-performance computing (HPC) facility to run efficiently.

Running with higher resolution requires re-compiling the code after changing the tile size and number of processors, see code/SIZE.h_eddy (as configured here, for 100 processors; for faster results change the tile size and use 200 or even 400 processors). Note we will NOT enable pkg/gmredi in our eddy run, so it can be eliminated from the list in packages.conf\(^1\) (make sure to set useGMRedi=.FALSE. in data.pkg).

In conjunction with the change in code/SIZE.h_eddy, uncomment these lines in PARM04 in data:

\begin{verbatim}
\begin{verbatim}
delX=200*5.E3,
delY=400*5.E3,
\end{verbatim}
\end{verbatim}

to specify 5 km resolution in 200 × 400 grid cells in \(x\) and \(y\). New files for bathymetry, forcing fields, and initial temperature can be generated using the MATLAB program verification/tutorial_reentrant_channel/input/gendata_5km.m or Python script verification/tutorial_reentrant_channel/input/gendata.5km.py (don’t forget to change the filenames in data.rbcs and PARM05 in data).

Running at higher resolution requires a smaller time step for stability. Revisiting Section 4.3.2.1, to maintain advective stability (CFL condition, (4.24)) one could simply decrease the time step by the same factor of 10 decrease as \(\Delta x\) – stability of inertial oscillations is no longer a limiting factor, given a smaller \(\Delta t\) in (4.25) – but to speed things up we’d like to keep \(\Delta t\) as large as possible. With a rich eddying solution, however, is it clear that horizontal velocity will remain order \(\sim 1\) ms\(^{-1}\)? As a compromise, we suggest setting parameter DeltaT=250. (seconds) in data, which we found to be stable. For this choice, a 30-year integration requires setting nTimeSteps=3732480.

\(^1\) Note it is not strictly necessary to remove pkg/gmredi from your high-resolution build – however, if kept in the list of packages included in packages.conf, it then becomes necessary to deactivate in data.pkg for this run by setting useGMRedi=.FALSE.. If by chance you set a use«PKG» flag to .TRUE. in data.pkg but have not included the package in the build, the model will terminate with error on startup. But you can always set a use«PKG» flag to .FALSE. whether or not the package is included in the build.

Figure 4.20: Residual meridional overturning circulation (shaded) as computed in density coordinates and converted back into (zonal mean) depth coordinates, averaged over simulation year 30. Black lines show zonal mean temperature, contour interval 1 °C. Left plot is from non-GM run, right using GM.
While it would be possible to decrease (spatially uniform) harmonic viscosity to a more appropriate value for this resolution, or perhaps use bi-harmonic viscosity (see Section 2.14.5), we will make use of one of the nonlinear viscosity schemes described in Section 2.19, geared toward large eddy simulations, where viscosity is a function of the resolved motion. Here, we employ the Leith viscosity (Leith 1968, Leith 1996 [Lei68] [Lei96]). Set the following parameters in PARM01 of data:

```
viscC2Leith = 1.,
useFullLeith=.TRUE.,
viscAhGridMax = 0.5,
```

(and comment out the line `viscAh = 2000.`). \( \text{viscC2Leith} \) is a scaling coefficient which we set to 1.0, \( \text{useFullLeith} = \text{.TRUE.} \) uses unapproximated gradients in the Leith formulation (see Section 2.19.1.4). Parameter \( \text{viscAhGridMax} \) places a maximum limit on the Leith viscosity so that the CFL condition is obeyed (see Section 2.19.1.7 and (4.26) in discussion of Numerical Stability Criteria). The values of \( \text{viscAh} \) that the Leith scheme generates in this solution generally range from order 1 m\(^2\) s\(^{-1}\) in regions of weak flow to over 100 m\(^2\) s\(^{-1}\) in jets. Note that while it would have been possible to use the Leith scheme in the 50 km resolution setup, the scheme was not really designed to be used at such a large \( \Delta x \), and the \( A_h \) it generates about an order of magnitude below the constant \( A_h = 2000 \) m\(^2\) s\(^{-1}\) employed in the coarse model runs, resulting in a very noisy solution.

Finally, we suggest adding the parameter \( \text{useSingleCpuIO} = \text{.TRUE.} \) in PARM01 of data. This will produce global output files generated by the master MPI processor, rather than a copious amount of single-tile files (each processor dumping output for its specific sub-domain).

To compare the eddying solution with the coarse-resolution simulations, we need to take a fairly long time average; even in annual means there is noticeably variability in the solution. Figure 4.21 through Figure 4.23 plot similar figures as Figure 4.14-Figure 4.20, showing a time mean over the last five years of the simulation.

![Figure 4.21: Left: Zonal-mean temperature (shaded) and zonal-mean mixed layer depth (black line) from eddying simulation averaged over years 26-30. Right: Eulerian meridional overturning circulation (shaded) from eddying simulation averaged over years 26-30. Contour interval is 0.5 Sv.](image)

In general, our coarse resolution solutions are not a bad likeness of the (time mean) eddying solution, particularly when we use pkg/gmredi to parameterize mesoscale eddies. More detailed comments comparing these solutions are as follows:

- The superiority of the GM solution is clear in the plot of zonal mean temperature (Figure 4.21 left panel vs. Figure 4.14) and the residual overturning circulation (Figure 4.23 vs. Figure 4.19 and Figure 4.20). Differences
Figure 4.22: Barotropic streamfunction from eddying simulation averaged over years 26-30. Contour interval is 20 Sv.
Figure 4.23: Left: Residual meridional overturning circulation (shaded) as computed in density (i.e., temperature) coordinates, from eddying simulation averaged over years 26-30. Contour interval is 0.5 Sv. Green dashed curve shows maximum and minimum (instantaneous) SST in each latitude band. Right: Residual meridional overturning circulation (shaded) as computed in density coordinates and converted back into depth coordinates, from eddying simulation averaged over years 26-30. Black lines show zonal mean temperature, contour interval 1 °C.

among the Eulerian MOC plots (Figure 4.21 right panel vs. Figure 4.17) are less obvious, but note that in the more stratified northern section of the domain, the eddying MOC looks more like the coarse “Eulerian + Bolus” GM solution (Figure 4.18). However, these two fields are not expected to be equal, since the eddying MOC calculated by layers also includes a stationary eddy component (Viebahn and Eden 2012 [VE12]; Dufour et al. 2012 [DSZ+12]).

• A large anticyclonic barotropic vortex is present away from the topographic ridge as shown in a plot of the barotropic streamfunction (Figure 4.22; recall, our domain is located in the Southern Hemisphere, so anticyclonic is counter-clockwise). As such, the flow passing through the deep notch is somewhat less than obtained in the coarse solution (Figure 4.16). Yet, similar constraints on barotropic potential vorticity conservation lead to a similar overall pattern.

• Examining the residual circulation generated from pkg/layers diagnostics (see Figure 4.23 vs. Figure 4.19, Figure 4.20), the non-GM solution seems quite poor, which would certainly have implications on tracer transport had any additional tracers been included in the simulation. In the GM solution, eddies seem to only partially cancel the cell forced by northward Ekman transport (Deacon Cell). In the eddying solution, the residual circulation is oriented in the opposite sense: eddy fluxes resulting from baroclinic instability due to the northern sponge layer (stratification) overwhelms the Deacon Cell. This would seem to suggest than our parameterization of eddies by GM, or more specifically, our choice for parameter GM_background_K of 1000 m$^2$s$^{-1}$, may be too low, at least for this idealized setup! Parameterizing eddies in the Southern Ocean is a topical research question, but some studies suggest this value of GM thickness diffusivity may indeed be low for values in the Southern Ocean (e.g., Ferriera et al. 2005 [FMH05]). A weak residual deep cell, oriented with rising flow along the sponge layer, is also present. Note that the area enclosed by the dashed green lines in Figure 4.23 is quite large, due to episodic large deviations in SST associated with eddies.

• As might be suggested by the orientation of the residual MOC, in the eddying solution temperature relaxation in the sponge layer is associated with heat gain in the thermocline. In the coarse runs, however, the sponge layer is effectively cooling, particularly in the non-GM run. Although at present there is no diagnostic available in pkg/rbcs which directly tabulates these fluxes, computing them is quite simple: the heat flux (in watts) into a grid cell in the sponge layer is computed as $\rho C_p V_y * (\theta(i,j,k) - \theta_{rbc}(i,j,k)) * M_{rbc}$ where $C_p$ is HeatCapacity_Cp (3994.0)
\[ J \text{ kg}^{-1} \text{K}^{-1} \text{ by default), } V_g \text{ is the grid cell volume (} rA(i,j) \text{ * drF(k) * hFacC(i,j,k)); see Section 4.3.3.2 for definition of } hFacC, \theta(i,j,k) \text{ is gridpoint potential temperature (} \circC, \theta(i,j,k)_{rel} \text{ is gridpoint relaxation potential temperature (} \circC, \text{ as prescribed in file input/temperature.5km.bin or input/temperature.50km.bin), } \tau_T \text{ is the restoring timescale } \text{tauRelaxT (as set in data.rbcs to 864,000 seconds or 10 days), and } M_{rel} \text{ is a 3-D restoring mask (values between 0.0 and 1.0 as discussed above) as specified in file T_relax_mask.5km.bin or T_relax_mask.50km.bin.} \]

4.4 Ocean Gyre Advection Schemes

(in directory: verification/tutorial_advection_in_gyre/)

This set of examples is based on the barotropic and baroclinic gyre MITgcm configurations, that are described in Section 4.1 and Section 4.2. The examples in this section explain how to introduce a passive tracer into the flow field of the barotropic and baroclinic gyre setups and looks at how the time evolution of the passive tracer depends on the advection or transport scheme that is selected for the tracer.

Passive tracers are useful in many numerical experiments. In some cases tracers are used to track flow pathways, for example in Dutay et al. (2002) [DBD+02] a passive tracer is used to track pathways of CFC-11 in 13 global ocean models, using a numerical configuration similar to the example described in Section 4.13. In other cases tracers are used as a way to infer bulk mixing coefficients for a turbulent flow field, for example in Marshall et al. (2006) [MSJH06] a tracer is used to infer eddy mixing coefficients in the Antarctic Circumpolar Current region. Typically, in biogeochemical and ecological simulations large numbers of tracers are used that carry the concentrations of biological nutrients and concentrations of biological species. When using tracers for these and other purposes it is useful to have a feel for the role that the advection scheme employed plays in determining properties of the tracer distribution. In particular, in a discrete numerical model, tracer advection only approximates the continuum behavior in space and time and different advection schemes introduce different approximations so that the resulting tracer distributions vary. In the following text we illustrate how to use the different advection schemes available in MITgcm, and discuss which properties are well represented by each scheme. The advection schemes selections also apply to active tracers (e.g., \( T \) and \( S \)) and the character of the schemes also affects their distributions and behavior.

4.4.1 Advection and tracer transport

In general, the tracer problem we want to solve can be written

\[
\frac{\partial C}{\partial t} = -U \cdot \nabla C + S
\]  

(4.29)

where \( C \) is the tracer concentration in a model cell, \( U = (u, v, w) \) is the model 3-D flow field. In (4.29), \( S \) represents source, sink and tendency terms not associated with advective transport. Example of terms in \( S \) include (i) air-sea fluxes for a dissolved gas, (ii) biological grazing and growth terms (for a biogeochemical problem) or (iii) convective mixing and other sub-grid parameterizations of mixing. In this section we are primarily concerned with

1. how to introduce the tracer term, \( C \), into an integration
2. the different discretized forms of the \( -U \cdot \nabla C \) term that are available
4.4.2 Introducing a tracer into the flow

The MITgcm ptracers package (see section Section 8.3.3 for a more complete discussion of the ptracers package and section Section 8.1.1 for a general introduction to MITgcm packages) provides pre-coded support for a simple passive tracer with an initial distribution at simulation time \( t = 0 \) of \( C_0(x, y, z) \). The steps required to use this capability are

1. **Activating the ptracers package.** This simply requires adding the line `ptracers` to the file `code/packages.conf`.

2. **Setting an initial tracer distribution.**

Once the two steps above are complete we can proceed to examine how the tracer we have created is carried by the flow field and what properties of the tracer distribution are preserved under different advection schemes.

4.4.3 Selecting an advection scheme

- flags in `input/data` and `input/data.ptracers`
- overlap width
- `#define` CPP option `PTRACERS_ALLOW_DYN_STATE` in `code/PTRACERS_OPTIONS.h` as required for SOM case

4.4.4 Comparison of different advection schemes

1. Conservation
2. Dispersion
3. Diffusion
4. Positive definite

![Dye evolving in a double gyre with different advection schemes. The figure shows the dye concentration one year after injection into a single grid cell near the left boundary.](image)

Figure 4.24: Dye evolving in a double gyre with different advection schemes. The figure shows the dye concentration one year after injection into a single grid cell near the left boundary.
4.5 Global Ocean Simulation

This example experiment demonstrates using the MITgcm to simulate the planetary ocean circulation. The simulation is configured with realistic geography and bathymetry on a $4^\circ \times 4^\circ$ spherical polar grid. Fifteen levels are used in the vertical, ranging in thickness from 50 m at the surface to 690 m at depth, giving a maximum model depth of 5200 m. Different time-steps are used to accelerate the convergence to equilibrium (see Bryan 1984 [Bry84]) so that, at this resolution, the configuration can be integrated forward for thousands of years on a single processor desktop computer.

4.5.1 Overview

The model is forced with climatological wind stress data from Trenberth (1990) [TOL90] and NCEP surface flux data from Kalnay et al. (1996) [KKK+96]. Climatological data (Levitus and Boyer 1994a,b [LB94a, LB94b]) is used to initialize the model hydrography. Levitus and Boyer seasonal climatology data is also used throughout the calculation to provide additional air-sea fluxes. These fluxes are combined with the NCEP climatological estimates of surface heat flux, resulting in a mixed boundary condition of the style described in Haney (1971) [Han71]. Altogether, this yields the following forcing applied in the model surface layer.

\[
\mathcal{F}_u = \frac{\tau_x}{\rho_0 \Delta z_s}
\]
\[
\mathcal{F}_v = \frac{\tau_y}{\rho_0 \Delta z_s}
\]
\[
\mathcal{F}_\theta = -\lambda_\theta (\theta - \theta^*) - \frac{1}{C_p \rho_0 \Delta z_s} \mathcal{Q}
\]
\[
\mathcal{F}_s = -\lambda_s (S - S^*) + \frac{S_0}{\Delta z_s} (\mathcal{E} - \mathcal{P} - \mathcal{R})
\]
where $\mathcal{F}_u, \mathcal{F}_v, \mathcal{F}_\theta, \mathcal{F}_s$ are the forcing terms in the zonal and meridional momentum and in the potential temperature and salinity equations respectively. The term $\Delta z$ represents the top ocean layer thickness in meters. It is used in conjunction with a reference density, $\rho_0$ (here set to 999.8 kg m$^{-3}$), a reference salinity, $S_0$ (here set to 35 ppt), and a specific heat capacity, $C_p$ (here set to 4000 J kg$^{-1}$ K$^{-1}$), to convert input dataset values into time tendencies of potential temperature (with units of °C s$^{-1}$), salinity (with units ppt s$^{-1}$) and velocity (with units m s$^{-2}$). The externally supplied forcing fields used in this experiment are $\tau_x, \tau_y, \mathcal{E} - \mathcal{P} - \mathcal{R}$. The wind stress fields ($\tau_x, \tau_y$) have units of N m$^{-2}$. The temperature forcing fields ($\mathcal{E} - \mathcal{P} - \mathcal{R}$) have units of °C s$^{-1}$, and salinity forcing fields ($S^*, \mathcal{E} - \mathcal{P} - \mathcal{R}$) have units of ppt and m s$^{-1}$ respectively. The source files and procedures for ingesting this data into the simulation are described in the experiment configuration discussion in section Section 4.5.3.

### 4.5.2 Discrete Numerical Configuration

The model is configured in hydrostatic form. The domain is discretized with a uniform grid spacing in latitude and longitude on the sphere $\Delta \phi = \Delta \lambda = 4^\circ$, so that there are 90 grid cells in the zonal and 40 in the meridional direction. The internal model coordinate variables $x$ and $y$ are initialized according to

\[
\begin{align*}
x &= r \cos(\phi), & \Delta x &= r \cos(\Delta \phi) \\
y &= r \lambda, & \Delta y &= r \Delta \lambda
\end{align*}
\]

Arctic polar regions are not included in this experiment. Meridionally the model extends from $80^\circ$S to $80^\circ$N. Vertically the model is configured with fifteen layers with the following thicknesses:

- $\Delta z_1 = 50$ m
- $\Delta z_2 = 70$ m
- $\Delta z_3 = 100$ m
- $\Delta z_4 = 140$ m
- $\Delta z_5 = 190$ m
- $\Delta z_6 = 240$ m
- $\Delta z_7 = 290$ m
- $\Delta z_8 = 340$ m
- $\Delta z_9 = 390$ m
- $\Delta z_{10} = 440$ m
- $\Delta z_{11} = 490$ m
- $\Delta z_{12} = 540$ m
- $\Delta z_{13} = 590$ m
- $\Delta z_{14} = 640$ m
- $\Delta z_{15} = 690$ m

(here the numeric subscript indicates the model level index number, $k$) to give a total depth, $H$, of 5200 m. The implicit free surface form of the pressure equation described in Marshall et al. (1997) [MHPA97] is employed. A Laplacian operator, $\nabla^2$, provides viscous dissipation. Thermal and haline diffusion is also represented by a Laplacian operator.

Wind-stress forcing is added to the momentum equations in (4.34) for both the zonal flow $u$ and the meridional flow $v$, according to equations (4.30) and (4.31). Thermodynamic forcing inputs are added to the equations in (4.35) for potential temperature, $\theta$, and salinity, $S$, according to equations (4.32) and (4.33). This produces a set of equations solved in this configuration as follows:

\[
\begin{align*}
\frac{D u}{D t} - fu + \frac{1}{\rho} \frac{\partial p'}{\partial x} - \nabla_h \cdot A_h \nabla_h u - \frac{\partial}{\partial z} A_z \frac{\partial u}{\partial z} &= \left\{ \begin{array}{ll} \mathcal{F}_u & \text{(surface)} \\ 0 & \text{(interior)} \end{array} \right. \\
\frac{D v}{D t} + fu + \frac{1}{\rho} \frac{\partial p'}{\partial y} - \nabla_h \cdot A_h \nabla_h v - \frac{\partial}{\partial z} A_z \frac{\partial v}{\partial z} &= \left\{ \begin{array}{ll} \mathcal{F}_v & \text{(surface)} \\ 0 & \text{(interior)} \end{array} \right.
\end{align*}
\]  

(4.34)

### 4.5. Global Ocean Simulation
\[
\frac{\partial \eta}{\partial t} + \nabla h \cdot \vec{u} = 0
\]

\[
\frac{D\theta}{Dt} - \nabla h \cdot K h \nabla \theta - \frac{\partial}{\partial z} \Gamma(K_z) \frac{\partial \theta}{\partial z} = \begin{cases} F_\theta & \text{(surface)} \\ 0 & \text{(interior)} \end{cases}
\]

\[
\frac{D\theta}{Dt} - \nabla h \cdot K h \nabla \theta - \frac{\partial}{\partial z} \Gamma(K_z) \frac{\partial \theta}{\partial z} = \begin{cases} F_s & \text{(surface)} \\ 0 & \text{(interior)} \end{cases}
\]

(4.35)

where \(u = \frac{Dx}{Dt} = r \cos(\phi) \frac{D\lambda}{Dt}\) and \(v = \frac{Dy}{Dt} = r \frac{D\phi}{Dt}\) are the zonal and meridional components of the flow vector, \(\vec{u}\), on the sphere. As described in Section 2, the time evolution of potential temperature \(\theta\) equation is solved prognostically.

\[
g \rho_0 \eta + \int_{-z}^{0} \rho' dz = p'
\]

The total pressure \(p\) is diagnosed by summing pressure due to surface elevation \(\eta\) and the hydrostatic pressure.

### 4.5.2.1 Numerical Stability Criteria

The Laplacian dissipation coefficient, \(A_h\), is set to \(5 \times 10^5 \text{ m s}^{-1}\). This value is chosen to yield a Munk layer width (see Adcroft 1995 [Adc95]),

\[
M_w = \pi \left( \frac{A_h}{\beta} \right)^{\frac{3}{4}}
\]

of \(~600 \text{ km}\). This is greater than the model resolution in low-latitudes, \(\Delta x \approx 400 \text{ km}\), ensuring that the frictional boundary layer is adequately resolved.

The model is stepped forward with a time step \(\Delta t_\theta = 24 \text{ hours}\) for thermodynamic variables and \(\Delta t_v = 30 \text{ minutes}\) for momentum terms. With this time step, the stability parameter to the horizontal Laplacian friction (Adcroft 1995 [Adc95])

\[
S_l = 4 \frac{A_h \Delta t_v}{\Delta x^2}
\]

evaluates to 0.6 at a latitude of \(\phi = 80^\circ\), which is above the 0.3 upper limit for stability, but the zonal grid spacing \(\Delta x\) is smallest at \(\phi = 80^\circ\) where \(\Delta x = r \cos(\phi) \Delta \phi \approx 77 \text{ km}\) and the stability criterion is already met one grid cell equatorwards (at \(\phi = 76^\circ\)).

The vertical dissipation coefficient, \(A_z\), is set to \(1 \times 10^{-3} \text{ m}^2 \text{s}^{-1}\). The associated stability limit

\[
S_l = 4 \frac{A_z \Delta t_v}{\Delta z^2}
\]

evaluates to 0.0029 for the smallest model level spacing (\(\Delta z_1 = 50 \text{ m}\)) which is well below the upper stability limit.

The numerical stability for inertial oscillations (Adcroft 1995 [Adc95])

\[
S_i = f^2 \Delta t_v^2
\]

evaluates to 0.07 for \(f = 2 \omega \sin(80^\circ) = 1.43 \times 10^{-4} \text{ s}^{-1}\), which is below the \(S_i < 1\) upper limit for stability.

The advective CFL (Adcroft 1995 [Adc95]) for a extreme maximum horizontal flow speed of \(|\vec{u}| = 2 \text{ m s}^{-1}\)

\[
S_a = \frac{|\vec{u}| \Delta t_v}{\Delta x}
\]

(4.40)

evaluates to \(5 \times 10^{-2}\). This is well below the stability limit of 0.5.

The stability parameter for internal gravity waves propagating with a maximum speed of \(c_g = 10 \text{ m s}^{-1}\) (Adcroft 1995 [Adc95])

\[
S_c = \frac{c_g \Delta t_v}{\Delta x}
\]

(4.41)

evaluates to \(2.3 \times 10^{-1}\). This is close to the linear stability limit of 0.5.
4.5.3 Experiment Configuration

The experiment files

- verification/tutorial_global_oce_latlon/input/data
- verification/tutorial_global_oce_latlon/input/data.pkg
- verification/tutorial_global_oce_latlon/input/eedata
- verification/tutorial_global_oce_latlon/input/trenberth_taux.bin
- verification/tutorial_global_oce_latlon/input/trenberth_tauy.bin
- verification/tutorial_global_oce_latlon/input/lev_s.bin
- verification/tutorial_global_oce_latlon/input/lev_t.bin
- verification/tutorial_global_oce_latlon/input/lev_sss.bin
- verification/tutorial_global_oce_latlon/input/lev_sst.bin
- verification/tutorial_global_oce_latlon/input/bathymetry.bin
- verification/tutorial_global_oce_latlon/code/SIZE.h

contain the code customizations and parameter settings for these experiments. Below we describe the customizations to these files associated with this experiment.

4.5.3.1 Driving Datasets

Figure 4.26—Figure 4.31 show the relaxation temperature ($\theta^*$) and salinity ($S^*$) fields, the wind stress components ($\tau_x$ and $\tau_y$), the heat flux ($Q$) and the net fresh water flux ($\mathcal{E} - \mathcal{P} - \mathcal{R}$) used in equations (4.30)-(4.33). The figures also indicate the lateral extent and coastline used in the experiment. Figure (—missing figure — ) shows the depth contours of the model domain.

4.5.3.2 File input/data

Listing 4.23: verification/tutorial_global_oce_latlon/input/data

```plaintext
# Model parameters
#
# Continuous equation parameters
&PARM01
tRef = 15*20.,
sRef = 15*35.,
viscAr=1.E-3,
viscAh=5.E5,
diffKhT=0.,
diffKrT=3.E-5,
diffKs=0.,
diffKs=3.E-5,
rhoConst=1035.,
rhoConstFresh=1000.,
```

(continues on next page)
Figure 4.26: Annual mean of relaxation temperature (°C)

Figure 4.27: Annual mean of relaxation salinity (g/kg)
Figure 4.28: Annual mean of zonal wind stress component (N m$^{-2}$)

Figure 4.29: Annual mean of meridional wind stress component (N m$^{-2}$)
Figure 4.30: Annual mean heat flux (W m$^{-2}$)

Figure 4.31: Annual mean freshwater flux (Evaporation-Precipitation) (m s$^{-1}$)
eosType = 'JMD95Z',
ivdc_kappa=100.,
imPLICITDIFFUSION=.TRUE.,
allowFreezing=.TRUE.,
extactConserv=.TRUE.,
useRealFreshWaterFlux=.TRUE.,
useCDScheme=.TRUE.,
# turn on looped cells
hFacMin=.05,
hFacMindr=50.,
# set precision of data files
readBinaryPrec=32,
&

# Elliptic solver parameters
&PARM02
cg2dMaxIters=500,
cg2dTargetResidual=1.E-13,
&

# Time stepping parameters
&PARM03
nIter0= 0,
nTimeSteps = 20,
# 100 years of integration will yield a reasonable flow field
# startTime = 0.,
# endTime = 3110400000.,
deltaTmom = 1800.,
tauCD = 321428.,
deltaTtracer = 86400.,
deltaTClock = 86400.,
deltaTfreesurf = 86400.,
abEps = 0.1,
pChkptFreq = 1728000.,
dumpFreq = 8640000.,
taveFreq = 8640000.,
monitorFreq=1.,
# 2 months restoring timescale for temperature
tauThetaClimRelax= 5184000.,
# 6 months restoring timescale for salinity
tauSaltClimRelax = 15552000.,
periodicExternalForcing=.TRUE.,
externForcingPeriod=2592000.,
externForcingCycle=31104000.,
&

# Gridding parameters
&PARM04
usingSphericalPolarGrid=.TRUE.,
de1R= 50., 70., 100., 140., 190.,
  240., 290., 340., 390., 440.,
  490., 540., 590., 640., 690.,
(continues on next page)
ygOrigin=-80.,
dySpacing=4.,
dxSpacing=4.,
&

# Input datasets
&PARM05
bathyFile= 'bathymetry.bin',
hydrogThetaFile= 'lev_t.bin',
hydrogSaltFile= 'lev_s.bin',
meridWindFile= 'trenberth_taux.bin',
meridWindFile= 'trenberth_tauy.bin',
thetaClimFile= 'lev_sst.bin',
saltClimFile= 'lev_sss.bin',
surfQnetFile= 'ncep_qnet.bin',
the_run_name= 'global_oce_latlon',
# fresh water flux is turned on, comment next line to it turn off
# (maybe better with surface salinity restoring)
EmPmRFile= 'ncep_emp.bin',
&

This file specifies the main parameters for the experiment. The parameters that are significant for this configuration are

- Lines 7-8,

```plaintext
tRef= 15*20.,
sRef= 15*35.,
```

set reference values for potential temperature and salinity at each model level in units of °C and ppt. The entries are ordered from surface to depth. Density is calculated from anomalies at each level evaluated with respect to the reference values set here.

- Line 9,

```plaintext
viscAr=1.E-3,
```

this line sets the vertical Laplacian dissipation coefficient to $1 \times 10^{-3}$ m$^2$ s$^{-1}$. Boundary conditions for this operator are specified later.

- Line 10,

```plaintext
viscAh=5.E5,
```

this line sets the horizontal Laplacian frictional dissipation coefficient to $5 \times 10^5$ m$^2$ s$^{-1}$. Boundary conditions for this operator are specified later.

- Lines 11, 13,

```plaintext
diffKhT=0.,
diffKhS=0.,
```

set the horizontal diffusion coefficient for temperature and salinity to 0, since pkg/gmredi is used.

- Lines 12, 14,
diffKrT=3.E-5,  
diffKrS=3.E-5,

set the vertical diffusion coefficient for temperature and salinity to $3 \times 10^{-5}$ m$^2$ s$^{-1}$. The boundary condition on this operator is $\frac{\partial}{\partial z} = 0$ at both the upper and lower boundaries.

- Lines 15-17,

\begin{verbatim}
rhoConst=1035.,  
rhoConstFresh=1000.,  
eosType = 'JMD95Z',
\end{verbatim}

set the reference densities for sea water and fresh water, and selects the equation of state (Jackett and McDougall 1995 [JM95]).

- Lines 18-19,

\begin{verbatim}
ivdc_kappa=100.,  
implicitDiffusion=.TRUE.,
\end{verbatim}

specify an “implicit diffusion” scheme with increased vertical diffusivity of 100 m$^2$/s in case of instable stratification.

- Line 28,

\begin{verbatim}
readBinaryPrec=32,
\end{verbatim}

Sets format for reading binary input datasets containing model fields to use 32-bit representation for floating-point numbers.

- Line 33,

\begin{verbatim}
cg2dMaxIters=500,
\end{verbatim}

Sets maximum number of iterations the two-dimensional, conjugate gradient solver will use, irrespective of convergence criteria being met.

- Line 34,

\begin{verbatim}
cg2dTargetResidual=1.E-13,
\end{verbatim}

Sets the tolerance which the 2-D conjugate gradient solver will use to test for convergence in (2.15) to $1 \times 10^{-13}$. Solver will iterate until tolerance falls below this value or until the maximum number of solver iterations is reached.

- Line 39,

\begin{verbatim}
nIter0=0,
\end{verbatim}

Sets the starting time for the model internal time counter. When set to non-zero this option implicitly requests a checkpoint file be read for initial state. By default the checkpoint file is named according to the integer number of time step value nIter0. The internal time counter works in seconds. Alternatively, startTime can be set.

- Line 40,

\begin{verbatim}
nTimeSteps=20,
\end{verbatim}
Sets the time step number at which this simulation will terminate. At the end of a simulation a checkpoint file is automatically written so that a numerical experiment can consist of multiple stages. Alternatively `endTime` can be set.

- Line 44,

```plaintext
deltaTmom=1800.,
```

Sets the timestep $\Delta t$ used in the momentum equations to 30 minutes. See Section 2.2.

- Line 45,

```plaintext
tauCD=321428.,
```

Sets the D-grid to C-grid coupling time scale $\tau_{CD}$ used in the momentum equations.

- Lines 46-48,

```plaintext
deltaTtracer=86400.,
deltaTClock = 86400.,
deltaTfreesurf= 86400.5,
```

Sets the default timestep, $\Delta t_\theta$, for tracer equations and implicit free surface equations to 24 hours. See Section 2.2.

- Line 76,

```plaintext
bathyFile='bathymetry.bin'
```

This line specifies the name of the file from which the domain bathymetry is read. This file is a 2-D ($x, y$) map of depths. This file is assumed to contain 32-bit binary numbers giving the depth of the model at each grid cell, ordered with the $x$ coordinate varying fastest. The points are ordered from low coordinate to high coordinate for both axes. The units and orientation of the depths in this file are the same as used in the MITgcm code. In this experiment, a depth of 0 m indicates a solid wall and a depth of <0 m indicates open ocean.

- Lines 79-80,

```plaintext
zonalWindFile='trenberth_taux.bin'
meridWindFile='trenberth_tauy.bin'
```

These lines specify the names of the files from which the $x$- and $y$- direction surface wind stress is read. These files are also 3-D ($x, y, time$) maps and are enumerated and formatted in the same manner as the bathymetry file.

Other lines in the file `input/data` are standard values that are described in the Section 3.8.

### 4.5.3.3 File `input/data.pkg`

This file uses standard default values and does not contain customizations for this experiment.
4.5.3.4 File input/eedata

This file uses standard default values and does not contain customizations for this experiment.

4.5.3.5 Files input/trenberth_taux.bin and input/trenberth_tauy.bin

The input/trenberth_taux.bin and input/trenberth_tauy.bin files specify 3-D \((x, y, time)\) maps of wind stress \((\tau_x, \tau_y)\), based on values from Treberth et al. (1990) [TOL90]. The units are \(\text{N m}^{-2}\).

4.5.3.6 File input/bathymetry.bin

The input/bathymetry.bin file specifies a 2-D \((x, y)\) map of depth values. For this experiment values range between 0 and -5200 m, and have been derived from ETOPO5. The file contains a raw binary stream of data that is enumerated in the same way as standard MITgcm 2-D horizontal arrays.

4.5.3.7 File code/SIZE.h

Listing 4.24: verification/tutorial_global_oce_latlon/code/SIZE.h

```
CBOP
C    !ROUTINE: SIZE.h
C    !INTERFACE:
C    !include SIZE.h
C    !DESCRIPTION: \bv
C     *=================================================================
C     | SIZE.h Declare size of underlying computational grid.
C     *=================================================================
C     | The design here supports a three-dimensional model grid
C     | with indices I,J and K. The three-dimensional domain
C     | is comprised of nPx*nSx blocks (or tiles) of size sNx
C     | along the first (left-most index) axis, nPy*nSy blocks
C     | of size sNy along the second axis and one block of size
C     | Nr along the vertical (third) axis.
C     | Blocks/tiles have overlap regions of size OLx and OLy
C     | along the dimensions that are subdivided.
C     *=================================================================
C     \ev
C
C    Voodoo numbers controlling data layout:
C    sNx :: Number of X points in tile.
C    sNy :: Number of Y points in tile.
C    OLx :: Tile overlap extent in X.
C    OLy :: Tile overlap extent in Y.
C    nSx :: Number of tiles per process in X.
C    nSy :: Number of tiles per process in Y.
C    nPx :: Number of processes to use in X.
C    nPy :: Number of processes to use in Y.
C    Nx :: Number of points in X for the full domain.
C    Ny :: Number of points in Y for the full domain.
C    Nr :: Number of points in vertical direction.
CEOP
```

(continues on next page)
Four lines are customized in this file for the current experiment

- Line 45,

```fortran
sNx=45,
```

this line sets the number of grid points of each tile (or sub-domain) along the \( x \)-coordinate axis.

- Line 46,

```fortran
sNy=40,
```

this line sets the number of grid points of each tile (or sub-domain) along the \( y \)-coordinate axis.

- Lines 49,51,

```fortran
nSx=2,
nPx=1,
```

these lines set, respectively, the number of tiles per process and the number of processes along the \( x \)-coordinate axis. Therefore, the total number of grid points along the \( x \)-coordinate axis corresponding to the full domain extent is \( N_x = sNx \times nSx \times nPx = 90 \).
4.6 Global Ocean Simulation in Pressure Coordinates

(in directory: verification/tutorial_global_oce_in_p/)

This example experiment demonstrates using MITgcm to simulate the planetary ocean circulation in pressure coordinates, that is, without making the Boussinesq approximations. The simulation is configured as a near copy of tutorial_global_oce_latlon (Section 4.5), with realistic geography and bathymetry on a 4° × 4° spherical polar grid. Fifteen levels are used in the vertical, ranging in thickness from 50.4089 dbar ≈ 50 m at the surface to 710.33 dbar ≈ 690 m at depth, giving a maximum model depth of 5302.3122 dbar ≈ 5200 m. At this resolution, the configuration can be integrated forward for thousands of years on a single processor desktop computer.

4.6.1 Overview

The model is forced with climatological wind stress data from Trenberth (1990) [TOL90] and surface flux data from Jiang et al. (1999) [JSMR99]. Climatological data (Levitus and Boyer 1994a,b [LB94a, LB94b]) is used to initialize the model hydrography. Levitus and Boyer seasonal climatology data is also used throughout the calculation to provide additional air-sea fluxes. These fluxes are combined with the Jiang et al. climatological estimates of surface heat flux, resulting in a mixed boundary condition of the style described in Haney (1971) [Han71]. Altogether, this yields the following forcing applied in the model surface layer.

\[ F_u = g \frac{\tau_x}{\Delta p_s} \]  
\[ F_v = g \frac{\tau_y}{\Delta p_s} \]  
\[ F_\theta = -g \lambda_0 (\theta - \theta^*) - \frac{1}{C_p \Delta p_s} Q \]  
\[ F_s = +g \rho_{FW} \frac{S}{\rho \Delta p_s} (E - P - R) \]

where \( F_u, F_v, F_\theta, F_s \) are the forcing terms in the zonal and meridional momentum and in the potential temperature and salinity equations respectively. The term \( \Delta p_s \) represents the top ocean layer thickness in Pa. It is used in conjunction with a reference density, \( \rho_{FW} \) (here set to 999.8 kg m\(^{-3}\)), the surface salinity, \( S \), and a specific heat capacity, \( C_p \) (here set to 4000 J kg\(^{-1}\) K\(^{-1}\)), to convert input dataset values into time tendencies of potential temperature (with units of °C), salinity (with units ppt) and velocity (with units m s\(^{-1}\)). The externally supplied forcing fields used in this experiment are \( \tau_x, \tau_y, \theta^*, Q, E - P - R \). The wind stress fields \( (\tau_x, \tau_y) \) have units of N m\(^{-2}\). The temperature forcing fields \( (\theta^* \text{ and } Q) \) have units of °C and W m\(^{-2}\) respectively. The salinity forcing fields \( (E - P - R) \) has units of m s\(^{-1}\) respectively. The source files and procedures for ingesting these data into the simulation are described in the experiment configuration discussion in section Section 4.5.3.
4.6.2 Discrete Numerical Configuration

Due to the pressure coordinate, the model can only be hydrostatic (de Szoeke and Samelson 2002 [dSS02]). The domain is discretized with a uniform grid spacing in latitude and longitude on the sphere \( \Delta \phi = \Delta \lambda = 4^\circ \), so that there are 90 grid cells in the zonal and 40 in the meridional direction. The internal model coordinate variables \( x \) and \( y \) are initialized according to

\[
    x = r \cos(\phi), \quad \Delta x = r \cos(\Delta \phi) \\
y = r\lambda, \quad \Delta y = r\Delta \lambda
\]

Arctic polar regions are not included in this experiment. Meridionally the model extends from 80\(^\circ\)S to 80\(^\circ\)N. Vertically the model is configured with fifteen layers with the following thicknesses:

\[
\begin{align*}
\Delta p_1 & = 710330.720021 \text{ Pa} \\
\Delta p_2 & = 6570548.440790 \text{ Pa} \\
\Delta p_3 & = 6041670.010249 \text{ Pa} \\
\Delta p_4 & = 5516436.666057 \text{ Pa} \\
\Delta p_5 & = 4994602.034410 \text{ Pa} \\
\Delta p_6 & = 4475903.435290 \text{ Pa} \\
\Delta p_7 & = 3960063.245801 \text{ Pa} \\
\Delta p_8 & = 3446790.312651 \text{ Pa} \\
\Delta p_9 & = 2935781.405664 \text{ Pa} \\
\Delta p_{10} & = 2426722.705046 \text{ Pa} \\
\Delta p_{11} & = 1919291.315988 \text{ Pa} \\
\Delta p_{12} & = 1413156.804970 \text{ Pa} \\
\Delta p_{13} & = 1008846.750166 \text{ Pa} \\
\Delta p_{14} & = 705919.025481 \text{ Pa} \\
\Delta p_{15} & = 504089.693499 \text{ Pa}
\end{align*}
\]

(here the numeric subscript indicates the model level index number, \( k \); note that the surface layer has the highest index number 15) to give a total depth, \( H \), of -5200 m. In pressure, this is \( p_0 = 53023122.566084 \text{ Pa} \). The implicit free surface form of the pressure equation described in Marshall et al. (1997) [MHPA97] with the nonlinear extension by Campin et al. (2004) [CAHM04] is employed. A Laplacian operator, \( \nabla^2 \), provides viscous dissipation. Thermal and haline diffusion is also represented by a Laplacian operator.

Wind-stress forcing is added to the momentum equations in (4.46) for both the zonal flow, \( u \) and the meridional flow \( v \), according to equations (4.42) and (4.43). Thermodynamic forcing inputs are added to the equations in (4.47) for potential temperature, \( \theta \), and salinity, \( S \), according to equations (4.44) and (4.45). This produces a set of equations solved in this configuration as follows:

\[
\begin{align*}
\frac{Du}{Dt} - f v + \frac{1}{\rho} \frac{\partial \Phi'}{\partial x} - \nabla_h \cdot A_h \nabla_h u - (g p_0)^2 \frac{\partial}{\partial p} A_r \frac{\partial u}{\partial p} &= \begin{cases} F_u \text{ (surface)} \\
0 \text{ (interior)} \end{cases} \\
\frac{Dv}{Dt} + f u + \frac{1}{\rho} \frac{\partial \Phi'}{\partial y} - \nabla_h \cdot A_h \nabla_h v - (g p_0)^2 \frac{\partial}{\partial p} A_r \frac{\partial v}{\partial p} &= \begin{cases} F_v \text{ (surface)} \\
0 \text{ (interior)} \end{cases} \\
\frac{\partial p_h}{\partial t} + \nabla_h \cdot \vec{u} &= 0 \\
\frac{D\theta}{Dt} - \nabla_h \cdot K_h \nabla_h \theta - (g p_0)^2 \frac{\partial}{\partial p} \Gamma(K_r) \frac{\partial \theta}{\partial p} &= \begin{cases} F_\theta \text{ (surface)} \\
0 \text{ (interior)} \end{cases} \\
\frac{Ds}{Dt} - \nabla_h \cdot K_h \nabla_h s - (g p_0)^2 \frac{\partial}{\partial p} \Gamma(K_r) \frac{\partial S}{\partial p} &= \begin{cases} F_s \text{ (surface)} \\
0 \text{ (interior)} \end{cases}
\end{align*}
\]
\[
\Phi^{(0)}_H + \alpha_0 p_b + \int_0^P \alpha' d\rho = \Phi' 
\]

where \( u = \frac{Dx}{Dt} = r \cos(\phi) \frac{D\lambda}{Dt} \) and \( v = \frac{Dy}{Dt} = r \frac{D\phi}{Dt} \) are the zonal and meridional components of the flow vector, \( \vec{u} \), on the sphere. As described in Section 2, the time evolution of potential temperature \( \theta \) equation is solved prognostically. The full geopotential height \( \Phi \) is diagnosed by summing the geopotential height anomalies \( \Phi' \) due to bottom pressure \( p_b \) and density variations. The integration of the hydrostatic equation is started at the bottom of the domain. The condition of \( p = 0 \) at the sea surface requires a time-independent integration constant for the height anomaly due to density variations \( \Phi^{(0)}_H \), which is provided as an input field.

### 4.6.3 Experiment Configuration

The experiment files

- verification/tutorial_global_oce_in_p/input/data
- verification/tutorial_global_oce_in_p/input/data.pkg
- verification/tutorial_global_oce_in_p/input/eedata
- verification/tutorial_global_oce_in_p/input/topog.bin
- verification/tutorial_global_oce_in_p/input/deltageopotjmd95.bin
- verification/tutorial_global_oce_in_p/input/lev_s.bin
- verification/tutorial_global_oce_in_p/input/lev_t.bin
- verification/tutorial_global_oce_in_p/input/trenberth_taux.bin
- verification/tutorial_global_oce_in_p/input/trenberth_tauy.bin
- verification/tutorial_global_oce_in_p/input/lev_sst.bin
- verification/tutorial_global_oce_in_p/input/shi_qnet.bin
- verification/tutorial_global_oce_in_p/input/shi_empmr.bin
- verification/tutorial_global_oce_in_p/code/CPP_OPTIONS.h
- verification/tutorial_global_oce_in_p/code/SIZE.h

contain the code customizations and parameter settings for these experiments. Below we describe the customizations to these files associated with this experiment.

#### 4.6.3.1 Driving Datasets

Figure 4.32-Figure 4.37 show the relaxation temperature \( (\theta^*) \) and salinity \( (S^*) \) fields, the wind stress components \( (\tau_x \) and \( \tau_y) \), the heat flux \( (Q) \) and the net fresh water flux \( (\mathcal{E} - \mathcal{P} - \mathcal{R}) \) used in equations (4.42) - (4.45). The figures also indicate the lateral extent and coastline used in the experiment. Figure 4.38 shows the depth contours of the model domain.
Figure 4.32: Annual mean of relaxation temperature (°C)

Figure 4.33: Annual mean of relaxation salinity (g/kg)
Figure 4.34: Annual mean of zonal wind stress component (N m$^{-2}$)

Figure 4.35: Annual mean of meridional wind stress component (N m$^{-2}$)
Figure 4.36: Annual mean heat flux (W m$^{-2}$)

Figure 4.37: Annual mean freshwater flux (Evaporation-Precipitation) (m s$^{-1}$)
4.6.3.2 File input/data

Listing 4.25: verification/tutorial_global_oce_oce_in_p/input/data

```plaintext
# ====================
# | Model parameters |
# ====================
#
# Continuous equation parameters
&PARM01
tRef = 15*20.,
sRef = 15*35.,
viscAh =3.E5,
no_slip_sides=.TRUE.,
viscAr =1.721611620915750e5,
#viscAz =1.67E-3,
no_slip_bottom=.FALSE.,
diffKhT=1.E3,
diffKrT=5.15452581125000e3,
#diffKzT=0.5E-4,
diffKhS=1.E3,
diffKrS=5.15452581125000e3,
#diffKzS=0.5E-4,
cosPower=0.5,
imPLICITDiffusion=.TRUE.,
ivdc_kappa=1.03090516225000e9,
```

(continues on next page)
# ivdc_kappa=10.0,
gavity=9.81,
rhoConst=1035.0,
rhoConstFresh=1000.0,
buoyancyRelation='OCEANICP',
integr_GeoPot=1,
eosType='JMD95P',
useNHTerms=.TRUE.,
implicitFreeSurface=.TRUE.,
exactConserv=.TRUE.,
nonlinFreeSurf=4,
hFacInf=0.2,
hFacSup=2.0,
#- to check that it has no impact:
doResetHFactors=.TRUE.,
#useRealFreshWaterFlux=.TRUE.,
readBinaryPrec=64,
writeBinaryPrec=64,

# Elliptic solver parameters
&PARM02
cg2dMaxIters=200,
cg2dTargetResidual=1.E-9,

# Time stepping parameters
&PARM03
startTime = 0.,
endTime = 3456000.,
# after 100 years of intergration, one gets a reasonable flow field
#endTime = 3110400000.,
deltaTMom = 1200.0,
deltaTtracer = 172800.0,
deltaTFreeSurf = 172800.0,
deltaTClock = 172800.0,
abEps = 0.1,
pChkptFreq = 3110400000.,
dumpFreq = 3110400000.,
taveFreq = 3110400000.,
monitorFreq = 1.,
periodicExternalForcing=.TRUE.,
externForcingPeriod=2592000.,
externForcingCycle=31104000.,
tauThetaClimRelax=5184000.0,

# Gridding parameters
&PARM04
usingSphericalPolarGrid=.TRUE.,
dxSpacing=4.,
dySpacing=4.,
This file specifies the main parameters for the experiment. The parameters that are significant for this configuration are

- Line 9–10,

```fortran
viscAh=3.E5,
no_slip_sides=.TRUE.
```

these lines set the horizontal Laplacian frictional dissipation coefficient to $3 \times 10^5$ m$^2$ s$^{-1}$ and specify a no-slip boundary condition for this operator, i.e., $u = 0$ along boundaries in $y$ and $v = 0$ along boundaries in $x$.

- Lines 11-13,

```fortran
viscAr =1.721611620915750e5,
#viscAz =1.67E-3,
no_slip_bottom=.FALSE.,
```

These lines set the vertical Laplacian frictional dissipation coefficient to $1.721611620915750 \times 10^5$ Pa$^2$ s$^{-1}$, which corresponds to $1.67 \times 10^{-3}$ m$^2$ s$^{-1}$ in the commented line, and specify a free slip boundary condition for this operator, i.e., $\partial u/\partial p = \partial v/\partial p = 0$ at $p = p_b^0$, where $p_b^0$ is the local bottom pressure of the domain at rest. Note that the factor $(g\rho)^2$ needs to be included in this value.

- Line 14,

```fortran
diffKhT=1.E3,
```

this line sets the horizontal diffusion coefficient for temperature to 1000 m$^2$ s$^{-1}$. The boundary condition on this operator is $\partial u/\partial x = \partial v/\partial y = 0$ on all boundaries.

- Line 15–16,
diffKzT=0.5E-4,

this line sets the vertical diffusion coefficient for temperature to \(5.15452581125 \times 10^3\) Pa \(\text{s}^{-1}\), which corresponds to \(5 \times 10^{-4}\) m\(^2\) s\(^{-1}\) in the commented line. Note that the factor \((g \rho)^2\) needs to be included in this value. The boundary condition on this operator is \(\frac{\partial}{\partial p} = 0\) at both the upper and lower boundaries.

• Line 17–19,

These lines set the diffusion coefficients for salinity to the same value as for temperature.

• Line 21–23,

Select implicit diffusion as a convection scheme and set coefficient for implicit vertical diffusion to \(1.030905162225 \times 10^9\) Pa \(\text{s}^{-1}\), which corresponds to \(10\) m\(^2\) s\(^{-1}\).

• Line 24,

This line sets the gravitational acceleration coefficient to 9.81 m \(\text{s}^{-1}\).

• Line 25,

sets the reference density of sea water to 1035 kg m\(^{-3}\).

• Line 29,

Selects the full equation of state according to Jackett and McDougall (1995) [JM95]. Note that the only other sensible choice here could be the equation of state by McDougall et al. (2003) [MJWF03], MDJFW. Other model choices for equations of state do not make sense in this configuration.

• Line 28-29,

Selects the barotropic pressure equation to be the implicit free surface formulation.

• Line 32,

Select a more accurate conservation of properties at the surface layer by including the horizontal velocity divergence to update the free surface.
nonlinFreeSurf=3,
hFacInf=0.2,
hFacSup=2.0,

Select the nonlinear free surface formulation and set lower and upper limits for the free surface excursions.

- Line 39–40,

readBinaryPrec=64,
writeBinaryPrec=64,

Sets format for reading binary input datasets and writing binary output datasets containing model fields to use 64-bit representation for floating-point numbers.

- Line 45,

cg2dMaxIters=200,

Sets maximum number of iterations the 2-D conjugate gradient solver will use, **irrespective of convergence criteria being met**.

- Line 46,

cg2dTargetResidual=1.E-13,

Sets the tolerance which the 2-D conjugate gradient solver will use to test for convergence in (2.15) to $1 \times 10^{-9}$. Solver will iterate until tolerance falls below this value or until the maximum number of solver iterations is reached.

- Line 51,

startTime=0,

Sets the starting time for the model internal time counter. When set to non-zero, this option implicitly requests a checkpoint file be read for initial state. By default the checkpoint file is named according to the integer number of time steps in the startTime value. The internal time counter works in seconds.

- Line 52–54,

deltaTmom = 1200.0,
deltaTtracer = 172800.0,
deltaTfreesurf = 172800.0,

Sets the timestep $\delta t_v$ used in the momentum equations to 20 minutes and the timesteps $\delta t_{\theta}$ in the tracer equations and $\delta t_{\eta}$ in the implicit free surface equation to 48 hours. See Section 2.2.

- Line 60,
\texttt{pChkptFreq = 3110400000.,}

write a pickup file every 100 years of integration.

- Line 61–63,

\begin{verbatim}
dumpFreq = 3110400000.,
taveFreq = 3110400000.,
monitorFreq = 1.,
\end{verbatim}

write model output and time-averaged model output every 100 years, and monitor statistics every model time step (this is set for testing purposes; change to a larger number for long integrations).

- Line 64–66,

\begin{verbatim}
periodicExternalForcing=.TRUE.,
externForcingPeriod=2592000.,
externForcingCycle=31104000.,
\end{verbatim}

Allow periodic external forcing: set one month forcing period during which a single time slice of data is valid, and the repeat cycle to one year. Thus, external forcing files will contain twelve periods of forcing data.

- Line 67,

\texttt{tauThetaClimRelax=5184000.0,}

Set the restoring timescale to 2 months.

- Line 59,

\texttt{abEps=0.1,}

Adams-Bashforth factor (see Section 2.5).

- Line 72,

\texttt{usingSphericalPolarGrid=.TRUE.,}

Select spherical grid.

- Line 73–74,

\begin{verbatim}
dXspacing=4.,
dYspacing=4.,
\end{verbatim}

Set the horizontal grid spacing in degrees spherical distance.

- Line 77–81,

\texttt{delR=7103300.720021, ...}

set the layer thickness in pressure units, starting with the bottom layer.

- Line 87–96,

\begin{verbatim}
bathyFile='topog.box'
ploadFile='deltageopotjmd95.bin'
hydrogThetaFile='lev_t.bin',
\end{verbatim}

(continues on next page)
hydrogSaltFile = 'lev_s.bin',
zonalWindFile = 'trenberth_taux.bin',
meridWindFile = 'trenberth_tauy.bin',
thetaClimFile = 'lev_sst.bin',
surfQFile = 'shi_qnet.bin',
EmPmRFile = 'shi_empmr.bin',

These lines specify the names of the files holding the bathymetry data set, the time-independent geopotential height anomaly at the bottom, initial conditions of temperature and salinity, wind stress forcing fields, sea surface temperature climatology, heat flux, and fresh water flux (evaporation minus precipitation minus runoff) at the surface. See file descriptions in section Section 4.6.3.

Other lines in the file input/data are standard values that are described in the Section 3.8.

4.6.3.3 File input/data.pkg

This file uses standard default values and does not contain customizations for this experiment.

4.6.3.4 File input/eedata

This file uses standard default values and does not contain customizations for this experiment.

4.6.3.5 File input/topog.bin

This file is a 2-D \((x, y)\) map of depths. This file is assumed to contain 64-bit binary numbers giving the depth of the model at each grid cell, ordered with the \(x\) coordinate varying fastest. The points are ordered from low coordinate to high coordinate for both axes. The units and orientation of the depths in this file are the same as used in the MITgcm code (Pa for this experiment). In this experiment, a depth of 0 Pa indicates a land point (wall) and a depth of >0 Pa indicates open ocean.

4.6.3.6 File input/deltageopotjmd95.box

The file contains twelve identical 2-D maps \((x, y)\) of geopotential height anomaly at the bottom at rest. The values have been obtained by vertically integrating the hydrostatic equation with the initial density field (using input/lev_t.bin and input/lev_s.bin). This file has to be consistent with the temperature and salinity field at rest and the choice of equation of state!

4.6.3.7 Files input/lev_t.bin and input/lev_s.bin

The files input/lev_t.bin and input/lev_s.bin specify the initial conditions for temperature and salinity for every grid point in a 3-D array \((x, y, z)\). The data are obtain by interpolating monthly mean values using Levitus and Boyer (1994a,b) [LB94a, LB94b] for January onto the model grid. Keep in mind that the first index corresponds to the bottom layer and highest index to the surface layer.
4.6.3.8 Files input/trenberth_taux.bin and input/trenberth_tauy.bin

The files input/trenberth_taux.bin and input/trenberth_tauy.bin contain twelve 2-D \((x, y)\) maps of zonal and meridional wind stress values, \(\tau_x\) and \(\tau_y\), respectively, in 3-D arrays \((x, y, t)\). These are monthly mean values from Trenberth et al. (1990) [TOL90], units of N m\(^{-2}\).

4.6.3.9 File input/lev_sst.bin

The file input/lev_sst.bin contains twelve monthly surface temperature climatologies from Levitus and Boyer (1994b) [LB94b] in a 3-D arrays \((x, y, t)\).

4.6.3.10 Files input/shi_qnet.bin and input/shi_empmr.bin

The files input/shi_qnet.bin and input/shi_empmr.bin contain twelve monthly surface fluxes of heat (qnet) and freshwater (empmr) from Jiang et al. (1999) [JSMR99] in 3-D arrays \((x, y, t)\). Both fluxes are normalized so that the total forcing over one year results in no net flux into the ocean (note, the freshwater flux is actually constant in time).

4.6.3.11 File code/SIZE.h

The file code/SIZE.h is identical to that described in tutorial global ocean simulation, for more specifics see Section 4.5.3.7.

4.6.3.12 File code/CPP_OPTIONS.h

This file uses standard default values except for:

- \#define ATMOSPHERIC_LOADING
  enables pressure loading which is abused to include the initial geopotential height anomaly.

- \#define EXACT_CONSERV
  enables more accurate conservation properties to include the horizontal mass divergence in the free surface.

- \#define NONLIN_FRSURF
  enables the nonlinear free surface.

4.7 Held-Suarez Atmosphere

(in directory: verification/tutorial_held_suarez_cs/)

This example illustrates the use of the MITgcm as an atmospheric GCM, using simple Held and Suarez (1994) [HS94] forcing to simulate atmospheric dynamics on global scale. The set-up uses the rescaled pressure coordinate \((p^*)\) of Adcroft and Campin (2004) [AC04] in the vertical direction, with 20 equally-spaced levels, and the conformal cube-sphere grid (C32) described in Adcroft et al. (2004) [ACHM04].
4.7.1 Overview

This example demonstrates using the MITgcm to simulate the planetary atmospheric circulation, with flat orography and simplified forcing. In particular, only dry air processes are considered and radiation effects are represented by a simple Newtonian cooling. Thus, this example does not rely on any particular atmospheric physics package. This kind of simplified atmospheric simulation has been widely used in GFD-type experiments and in intercomparison projects of AGCM dynamical cores (Held and Suarez 1994 [HS94]).

The horizontal grid is obtained from the projection of a uniform gridded cube to the sphere. Each of the 6 faces has the same resolution, with \( 32 \times 32 \) grid points. The equator coincides with a grid line and crosses through the middle in 4 of the 6 faces, leaving 2 faces for the northern and southern polar regions. This curvilinear grid requires the use of the 2nd generation exchange topology (pkg/exch2) to connect tile and face edges, but without any limitation on the number of processors.

The use of the \( p^\ast \) coordinate with 20 equally spaced levels (\( 20 \times 50 \) mb, from \( p^\ast = 1000 \) mb to 0 at the top of the atmosphere) follows the choice of Held and Suarez (1994) [HS94]. Note that without topography, the \( p^\ast \) coordinate and the normalized pressure coordinate (\( \sigma_p \)) coincide exactly. Both viscosity and diffusivity are set to zero here, but an 8th order Shapiro (1970) [Sha70] filter is applied to both momentum and potential temperature, to remove selectively grid scale noise. Apart from the horizontal grid, this experiment is made very similar to the grid-point model case used in the Held and Suarez (1994) [HS94] study.

At this resolution, the configuration can be integrated forward for many years on a single processor desktop computer.

4.7.2 Forcing

The model is forced by relaxation to a radiative equilibrium temperature from Held and Suarez (1994) [HS94]. A linear frictional drag (Rayleigh damping) is applied in the lower part of the atmosphere and accounts for surface friction and momentum dissipation in the boundary layer. Altogether, this yields the following forcing from Held and Suarez (1994) [HS94] that is applied to the fluid:

\[
\vec{F}_v = -k_v(p)\vec{v}_h
\]

\[
F_\theta = -k_\theta(\varphi,p)[\theta - \theta_{eq}(\varphi,p)]
\]

where \( F_v, F_\theta \) are the forcing terms in the zonal and meridional momentum and in the potential temperature equations, respectively. The term \( k_v \) in (4.48) applies a Rayleigh damping that is active within the planetary boundary layer. It is defined so as to decay as pressure decreases according to

\[
k_v = k_f \max[0, (p^\ast/P_s^0 - \sigma_b)/(1 - \sigma_b)]
\]

\[
\sigma_b = 0.7 \text{ and } k_f = 1/86400 \text{ s}^{-1}
\]

where \( p^\ast \) is the pressure level of the cell center and \( P_s^0 \) is the pressure at the base of the atmospheric column, which is constant and uniform here (\( = 10^5 \) Pa), in the absence of topography.

The equilibrium temperature \( \theta_{eq} \) and relaxation time scale \( k_\theta \) are set to:

\[
\theta_{eq}(\varphi, p^\ast) = \max\{200(P_s^0/p^\ast)^\kappa, 315 - \Delta T_y \sin^2(\varphi) - \Delta \theta_z \cos^2(\varphi) \log(p^\ast/P_s^0)\}
\]

\[
k_\theta(\varphi, p^\ast) = k_a + (k_s - k_a) \cos^4(\varphi) \max[0, (p^\ast/P_s^0 - \sigma_b)/(1 - \sigma_b)]
\]

with:

\[
\Delta T_y = 60 \text{ K, } k_a = 1/(40 \cdot 86400) \text{ s}^{-1}
\]

\[
\Delta \theta_z = 10 \text{ K, } k_s = 1/(4 \cdot 86400) \text{ s}^{-1}
\]

Initial conditions correspond to a resting state with horizontally uniform stratified fluid. The initial temperature profile is simply the horizontal average of the radiative equilibrium temperature.

4.7. Held-Suarez Atmosphere
4.7.3 Set-up description

The model is configured in hydrostatic form, using non-Boussinesq $p^*$ coordinate. The vertical resolution is uniform, $\Delta p^* = 50 \times 10^2 \text{ Pa}$, with 20 levels, from $p^* = 10^5 \text{ Pa}$ to 0 at the top. The domain is discretized using the C32 cube-sphere grid (see Adcroft et al. 2004 [ACHM04]) that covers the whole sphere with a relatively uniform grid spacing. The resolution at the equator or along the Greenwich meridian is similar to a 128 $\times$ 64 equally spaced longitude-latitude grid, but requires 25% less grid points. Grid spacing and grid-point location are not computed by the model, but instead read from files.

The vector-invariant form of the momentum equation (see Section 2.15) is used so that no explicit metrics are necessary. Applying the vector-invariant discretization to the atmospheric equations (1.59), and adding the forcing terms (4.48), (4.49) on the right-hand-side, leads to the set of equations that are solved in this configuration:

\[
\frac{\partial \vec{v}_h}{\partial t} + (f + \zeta) \hat{k} \times \vec{v}_h + \nabla_p (\text{KE}) + \omega \frac{\partial \vec{v}_h}{\partial p} + \nabla_p \Phi' = -k_v \vec{v}_h \quad (4.50)
\]

\[
\frac{\partial \Phi'}{\partial p} + \frac{\partial \Pi}{\partial p} \Phi' = 0
\]

\[
\nabla_p \cdot \vec{v}_h + \frac{\partial \omega}{\partial p} = 0
\]

\[
\frac{\partial \theta}{\partial t} + \nabla_p \cdot (\theta \vec{v}_h) + \frac{\partial (\theta \omega)}{\partial p} = -k_v [\theta - \theta_{eq}]
\]

where $\vec{v}_h$ and $\omega = \frac{D_p}{D_t}$ are the horizontal velocity vector and the vertical velocity in pressure coordinate, $\zeta$ is the relative vorticity and $f$ the Coriolis parameter, $\hat{k}$ is the vertical unity vector, KE is the kinetic energy, $\Phi$ is the geopotential, and $\Pi$ the Exner function ($\Pi = C_p \left( \frac{p}{p_c} \right)^\kappa$ with $p_c = 10^5 \text{ Pa}$). Primed variables correspond to anomaly from the resting, uniformly stratified state.

As described in Section 2, the continuity equation is integrated vertically, to give a prognostic equation for the surface pressure $p_s$:

\[
\frac{\partial p_s}{\partial t} + \nabla_h \cdot \int_0^{p_s} \vec{v}_h dp = 0
\]

The implicit free surface form of the pressure equation described in Marshall et al. (1997) [MHPA97] is employed to solve for $p_s$; Vertically integrating the hydrostatic balance gives the geopotential $\Phi'$ and allows one to step forward the momentum equation (4.50). The potential temperature, $\theta$, is stepped forward using the new velocity field (see Section 2.8).

4.7.3.1 Numerical Stability Criteria

The numerical stability for inertial oscillations (see Adcroft 1995 [Adc95]):

\[
S_i = f^2 \Delta t^2
\]

(4.51)

evaluates to $4 \times 10^{-3}$ at the poles, for $f = 2\Omega \sin(\pi/2) = 1.45 \times 10^{-4} \text{ s}^{-1}$, which is well below the $S_i < 1$ upper limit for stability. The advective CFL (Adcroft 1995 [Adc95]) for a extreme maximum horizontal flow speed of $|\vec{u}| = 90 \text{ m/s}$ and the smallest horizontal grid spacing $\Delta x = 1.1 \times 10^5 \text{ m}$:

\[
S_a = \frac{|\vec{u}| \Delta t}{\Delta x}
\]

(4.52)

evaluates to 0.37, which is close to the stability limit of 0.5. The stability parameter for internal gravity waves propagating with a maximum speed of $c_g = 100 \text{ m/s}$ (Adcroft 1995 [Adc95])

\[
S_c = \frac{c_g \Delta t}{\Delta x}
\]

(4.53)

evaluates to $4 \times 10^{-1}$. This is close to the linear stability limit of 0.5.
4.7.4 Experiment Configuration

The model configuration for this experiment resides under the directory verification/tutorial_held_suarez_cs/. The experiment files

- verification/tutorial_held_suarez_cs/input/data
- verification/tutorial_held_suarez_cs/input/data.pkg
- verification/tutorial_held_suarez_cs/input/data.shap
- verification/tutorial_held_suarez_cs/input/eedata
- verification/tutorial_held_suarez_cs/code/packages.conf
- verification/tutorial_held_suarez_cs/code/CPP_OPTIONS.h
- verification/tutorial_held_suarez_cs/code/SIZE.h
- verification/tutorial_held_suarez_cs/code/DIAGNOSTICS_SIZE.h
- verification/tutorial_held_suarez_cs/code/apply_forcing.F,

contain the code customizations and parameter settings for these experiments. Below we describe the customizations to these files associated with this experiment.

4.7.4.1 File input/data

Listing 4.26: verification/tutorial_held_suarez_cs/input/data

```plaintext
# ================
# | Model parameters |
# ================
#
# Continuous equation parameters
&PARM01
  tRef=295.2, 295.5, 295.9, 296.3, 296.7, 297.1, 297.6, 298.1, 298.7, 299.3,
  300.0, 300.7, 301.9, 304.1, 308.0, 315.1, 329.5, 362.3, 419.2, 573.8,
  sRef=20*0.0,
  no_slip_sides=.FALSE.,
  no_slip_bottom=.FALSE.,
  buoyancyRelation='ATMOSPHERIC',
  eosType='IDEALG',
  rotationPeriod=86400.,
  implicitFreeSurface=.TRUE.,
  exactConserv=.TRUE.,
  nonlinFreeSurf=4,
  select_rStar=2,
  hFacInf=0.2,
  hFacSup=2.0,
  uniformLin_PhiSurf=.FALSE.,
  #hFacMin=0.2,
  saltStepping=.FALSE.,
  momViscosity=.FALSE.,
  vectorInvariantMomentum=.TRUE.,
  staggerTimeStep=.TRUE.,
  readBinaryPrec=64,
```

(continues on next page)
writeBinaryPrec=64,
&
# Elliptic solver parameters
&PARM02
cg2dMaxIters=200,
#cg2dTargetResidual=1.E-12,
cg2dTargetResWunit=1.E-17,
&
# Time stepping parameters
&PARM03
deltaT=450.,
#nIter0=276480,
startTime=124416000.,
#- run for 1 year (192.iterations x 450.s = 1.day, 360*192=69120):
#nTimeSteps=69120,
#forcing_In_AB=.FALSE.,
tracForcingOutAB=1,
abEps=0.1,
pChkptFreq=31104000.,
chkptFreq=2592000.,
dumpFreq=2592000.,
#monitorFreq=43200.,
taveFreq=0.,
#- to run a short test (2.h):
 nTimeSteps=16,
 monitorFreq=1.,
&
# Gridding parameters
&PARM04
usingCurvilinearGrid=.TRUE.,
horizGridFile='grid_cs32',
radius_fromHorizGrid=6370.E3,
delR=20*50.E2,
&
# Input datasets
&PARM05
topoFile='topo.cs.bin',
&
This file specifies the main parameters for the experiment. The parameters that are significant for this configuration are:

- Lines 7-8.
  
<table>
<thead>
<tr>
<th>tRef</th>
</tr>
</thead>
<tbody>
<tr>
<td>295.2</td>
</tr>
<tr>
<td>295.5</td>
</tr>
<tr>
<td>295.9</td>
</tr>
<tr>
<td>296.3</td>
</tr>
<tr>
<td>296.7</td>
</tr>
<tr>
<td>297.1</td>
</tr>
<tr>
<td>297.6</td>
</tr>
<tr>
<td>298.1</td>
</tr>
<tr>
<td>298.7</td>
</tr>
<tr>
<td>299.3</td>
</tr>
<tr>
<td>300.0</td>
</tr>
<tr>
<td>300.7</td>
</tr>
<tr>
<td>301.9</td>
</tr>
<tr>
<td>304.1</td>
</tr>
<tr>
<td>308.0</td>
</tr>
<tr>
<td>315.1</td>
</tr>
<tr>
<td>329.5</td>
</tr>
<tr>
<td>362.3</td>
</tr>
<tr>
<td>419.2</td>
</tr>
<tr>
<td>573.8</td>
</tr>
</tbody>
</table>

set reference values for potential temperature (in kelvins) at each model level. The entries are ordered like model level, from surface up to the top. Density is calculated from anomalies at each level evaluated with respect to the
reference values set here.

- Line 10,

```fortran
no_slip_sides=.FALSE.,
```

this line selects a free-slip lateral boundary condition for the horizontal Laplacian friction operator, e.g., \( \frac{\partial u}{\partial y} = 0 \) along boundaries in \( y \) and \( \frac{\partial v}{\partial x} = 0 \) along boundaries in \( x \).

- Line 11,

```fortran
no_slip_bottom=.FALSE.,
```

this line selects a free-slip boundary condition at the top, in the vertical Laplacian friction operator, e.g., \( \frac{\partial u}{\partial p} = 0 \).

- Line 12,

```fortran
buoyancyRelation='ATMOSPHERIC',
```

this line sets the type of fluid and the type of vertical coordinate to use, which, in this case, is air with a pressure-like coordinate (\( \rho \) or \( p^* \)).

- Line 13,

```fortran
eosType='IDEALG',
```

Selects the ideal gas equation of state.

- Line 15,

```fortran
implicitFreeSurface=.TRUE.,
```

Selects the way the barotropic equation is solved, here using the implicit free-surface formulation.

- Line 16,

```fortran
exactConserv=.TRUE.,
```

Explicitly calculate (again) the surface pressure changes from the divergence of the vertically integrated horizontal flow, after the implicit free surface solver and filters are applied.

- Lines 17-18,

```fortran
nonlinFreeSurf=4,
select_rStar=2,
```

Select the non-linear free surface formulation, using \( r^* \) vertical coordinate (here \( p^* \)). Note that, except for the default (= 0), other values of those two parameters are only permitted for testing/debugging purpose.

- Line 21,

```fortran
uniformLin_PhiSurf=.FALSE.,
```

Select the linear relation between surface geopotential anomaly and surface pressure anomaly to be evaluated from \( \frac{\partial \Phi}{\partial p} = \frac{1}{\rho(\theta_{Re})} \) (see Section 2.10.2). Note that using the default (= .TRUE.), the constant \( 1/\rho_0 \) is used instead, and is not necessarily consistent with other parts of the geopotential that rely on \( \theta_{Re,f} \).

- Line 23-24,
saltStepping=.FALSE.,
momViscosity=.FALSE.,

Do not step forward water vapor and do not compute viscous terms. This saves computer time.

- Line 25,
  
  vectorInvariantMomentum=.TRUE.,

  Select the vector-invariant form to solve the momentum equation.

- Line 26,
  
  staggerTimeStep=.TRUE.,

  Select the staggered time-stepping (rather than synchronous time stepping).

- Lines 27-28,
  
  readBinaryPrec=64,
  writeBinaryPrec=64,

  Sets format for reading binary input datasets and writing output fields to use 64-bit representation for floating-point numbers.

- Line 33,
  
  cg2dMaxIters=200,

  Sets maximum number of iterations the 2-D conjugate gradient solver will use. **irrespective of convergence criteria being met.**

- Line 35,
  
  cg2dTargetResWunit=1.E-17,

  Sets the tolerance (in units of \( \omega \)) which the 2-D conjugate gradient solver will use to test for convergence in equation (2.15) to \( 1 \times 10^{-17} \) Pa/s. Solver will iterate until tolerance falls below this value or until the maximum number of solver iterations is reached.

- Line 40,
  
  deltaT=450.,

  Sets the timestep \( \Delta t \) used in the model to 450 seconds (= 1/8 hour).

- Line 42,
  
  startTime=124416000.,

  Sets the starting time, in seconds, for the model time counter. A non-zero starting time requires the initial state read from a pickup file. By default the pickup file is named according to the integer number (nIter0) of time steps in the startTime value (nIter0 = startTime / deltaT).

- Line 44,
  
  #nTimeSteps=69120,
A commented out setting for the length of the simulation (in number of timesteps) that corresponds to 1-year simulation.

- Lines 54-55,

```plaintext
nTimeSteps=16,
monitorFreq=1.,
```

Sets the length of the simulation (in number of timesteps) and the frequency (in seconds) for “monitor” output to 16 iterations and 1 seconds respectively. This choice corresponds to a short simulation test.

- Line 48,

```plaintext
pChkptFreq=31104000.,
```

Sets the time interval, in seconds, between 2 consecutive “permanent” pickups (“permanent checkpoint frequency”) that are used to restart the simulation, to 1 year.

- Line 48,

```plaintext
chkptFreq=2592000.,
```

Sets the time interval, in seconds, between two consecutive “temporary” pickups (“checkpoint frequency”) to one month. The “temporary” pickup file name is alternatively “ckptA” and “ckptB”; these pickups (as opposed to the permanent ones) are designed to be over-written by the model as the simulation progresses.

- Line 50,

```plaintext
dumpFreq=2592000.,
```

Set the frequency (in seconds) for the snapshot output to 1 month.

- Line 51,

```plaintext
#monitorFreq=43200.,
```

A commented out line setting the frequency (in seconds) for the “monitor” output to 12 h. This frequency fits better with the longer simulation of one year.

- Line 60,

```plaintext
usingCurvilinearGrid=.TRUE.,
```

Set the horizontal type of grid to curvilinear grid.

- Line 61,

```plaintext
horizGridFile='grid_cs32',
```

Set the root for the grid file name to grid_cs32. The grid-file names are derived from the root, adding a suffix with the face number (e.g., .face001.bin, .face002.bin ...)

- Lines 63,

```plaintext
delR=20*50.E2,
```

This line sets the increments in pressure units to 20 equally thick levels of $50 \times 10^2$ Pa each. This defines the origin (interface $k = 1$) of the vertical pressure axis, with decreasing pressure as the level index $k$ increases.

- Line 68,
This commented out line would set the file name of a 2-D orography file, in units of meters, to topo.cs.bin.

Other lines in the file input/data are standard values that are described in Section 3..

### 4.7.4.2 File input/data.pkg

```
# Packages
&Packages
useSHAP_FILT=.TRUE.,
useDiagnostics=.TRUE.,
#useMNC=.TRUE.,
&
```

This file specifies the additional packages that the model uses for the experiment. Note that some packages are used by default (e.g., pkg/generic_advdiff) and some others are selected according to parameter in input/data.pkg (e.g., pkg/mom_vecinv). The additional packages that are used for this configuration are

- **Line 3**, `useSHAP_FILT=.TRUE.,`

  This line selects the Shapiro filter (Shapiro 1970 [Sha70]) (pkg/shap_filt) to be used in this experiment.

- **Line 4**, `useDiagnostics=.TRUE.,`

  This line selects pkg/diagnostics to be used in this experiment.

- **Line 5**, `#useMNC=.TRUE.,`

  This line would select pkg/mnc for I/O but is commented out.

### 4.7.4.3 File input/data.shap

```
# Shapiro Filter parameters
&SHAP_PARM01
  shap_filt_uvStar=.FALSE.,
  shap_filt_TrStagg=.TRUE.,
  Shap_funct=2,
  nShapT=0,
  nShapUV=4,
  #nShapTrPhys=0,
  nShapUVPhys=4,
  #Shap_TrLength=140000.,
  #Shap_uvLength=110000.,
```

(continues on next page)
This file specifies the parameters that the model uses for the Shapiro filter package (Shapiro 1970 [Sha70]), see Section 2.18. The parameters that are significant for this configuration are:

- Line 5,

```
Shap_funct=2,
```

This line selects which Shapiro filter function to use, here S2, for this experiment (see Section 2.18).

- Lines 6-7,

```
nShapT=0,  
nShapUV=4,
```

These lines select the order of the Shapiro filter for active tracers ($\theta$ and $q$) and momentum ($u$, $v$) respectively. In this case, no filter is applied to active tracers. Regarding the momentum, this sets the integer parameter $n$ to 4, in the equations of Section 2.18, which corresponds to a 8th order filter.

- Line 9,

```
nShapUVPhys=4,
```

This line selects the order of the physical space filter (filter function S2g, see Section 2.18) that applies to $u$, $v$. The difference $nShapUV - nShapUVPhys$ corresponds to the order of the computational filter (filter function S2c, see Section 2.18).

- Lines 12-13,

```
#Shap_Trtau=5400.,  
#Shap_uvtau=1800.,
```

These commented lines would have set the time scale of the filter (in seconds), for $\theta$, $q$ and for $u$, $v$ respectively, to 5400 s (90 min) and 1800 s (30 min). Without explicitly setting those timescales, the default is used, which corresponds to the model timestep.

### 4.7.4.4 File input/eedata

Listing 4.29: verification/tutorial_held_suarez_cs/input/eedata

```
# Example "eedata" file
# Lines beginning "#" are comments
# nTx - No. threads per process in X
# nTy - No. threads per process in Y
&EEPARAMS
  useCubedSphereExchange=.TRUE.,
# Activate one line below to support 2, 3 or 6 way multi-threading
  nTx=2,
  nTx=3,
  nTx=6,
```

(continues on next page)
# Note: Some systems use & as the namelist terminator. Other systems use a / character (as shown here).

This file uses standard default values except line 6:

```plaintext
useCubedSphereExchange=.TRUE.,
```

This line selects the cubed-sphere specific exchanges to connect tiles and faces edges.

## 4.7.4.5 File code/SIZE.h

Listing 4.30: verification/tutorial_held_suarez_cs/code/SIZE.h

```plaintext
CBOP
C !ROUTINE: SIZE.h
C !INTERFACE:
C include SIZE.h
C !DESCRIPTION: v
C | SIZE.h Declare size of underlying computational grid.
C | *==========================================================*
C | The design here supports a three-dimensional model grid
C | with indices I,J and K. The three-dimensional domain
C | is comprised of nPx*nSx blocks (or tiles) of size sNx
C | along the first (left-most index) axis, nPy*nSy blocks
C | of size sNy along the second axis and one block of size
C | Nr along the vertical (third) axis.
C | Blocks/tiles have overlap regions of size OLx and OLy
C | along the dimensions that are subdivided.
C | *==========================================================*
C \ev
C
C Voodoo numbers controlling data layout:
C sNx :: Number of X points in tile.
C sNy :: Number of Y points in tile.
C OLx :: Tile overlap extent in X.
C OLy :: Tile overlap extent in Y.
C nSx :: Number of tiles per process in X.
C nSy :: Number of tiles per process in Y.
C nPx :: Number of processes to use in X.
C nPy :: Number of processes to use in Y.
C Nx :: Number of points in X for the full domain.
C Ny :: Number of points in Y for the full domain.
C Nr :: Number of points in vertical direction.
CEOP
    INTEGER sNx
    INTEGER sNy
    INTEGER OLx
    INTEGER OLy
```

(continues on next page)
Four lines are customized in this file for the current experiment

- Line 45,

```fortran
sNx=32,
```

sets the lateral domain extent in grid points along the $x$-direction, for one face.

- Line 46,

```fortran
sNy=32,
```

sets the lateral domain extent in grid points along the $y$-direction, for one face.

- Line 49,

```fortran
nSx=6,
```

sets the number of tiles in the $x$-direction, for the model domain decomposition. In this simple case (single processor, with one tile per face), this number corresponds to the total number of faces.

- Line 55,

```fortran
Nr=20,
```

sets the vertical domain extent in grid points.
4.7.4.6 File code/packages.conf

Listing 4.31: verification/tutorial_held_suarez_cs/input/code/packages.conf

```plaintext
#-- list of packages (or group of packages) to compile for this experiment:
exch2
gfd
shap_filt
diagnostics
mnc
```

This file specifies the packages that are compiled and made available for this experiment. The additional packages that are used for this configuration are

- Line 1,
  `gfd`
  This line selects the standard set of packages that are used by default.
- Line 2,
  `shap_filt`
  This line makes the Shapiro filter package available for this experiment.
- Line 3,
  `exch2`
  This line selects `pkg/exch2` to be compiled and used in this experiment. Note that at present, no such parameter `useEXCH2` exists and therefore this package is always used when it is compiled.
- Line 4,
  `diagnostics`
  This line selects `pkg/diagnostics` to be compiled, and makes it available for this experiment.
- Line 5,
  `mnc`
  This line selects the `pkg/mnc` to be compiled, and makes it available for this experiment.

4.7.4.7 File code/CPP_OPTIONS.h

This file uses the standard default except for:

```plaintext
#define NONLIN_FRSURF
```

This line enables the non-linear free-surface part of the code, which is required for the $p^*$ coordinate formulation.
4.7.4.8 Other Files

Other files relevant to this experiment are

- code/apply_forcing.F
- input/grid_cs32.face00[n].bin, with \( n = 1, 2, 3, 4, 5, 6 \)

contain the code customizations and binary input files for this experiment. The file apply_forcing.F contains four subroutines that calculate the forcing terms (i.e., right-hand side terms) in the momentum equation (4.48), EXTERNAL_FORCING_U and EXTERNAL_FORCING_V and in the potential temperature equation (4.49), EXTERNAL_FORCING_T. The water-vapor forcing subroutine (EXTERNAL_FORCING_S) is left empty for this experiment. The grid-files input/grid_cs32.face00[n].bin, with \( n = 1, 2, 3, 4, 5, 6 \), are binary files (direct-access, big-endian 64 bit reals) that contains all the cubed-sphere grid lengths, areas and grid-point positions, with one file per face. Each file contains 18 2-D arrays (dimension 33 \( \times \) 33) that correspond to the model variables: \( XC, YC, DXF, DYF, RA, XG, YG, DXV, DYU, RAZ, DXC, DYC, RAW, RAS, DXG, DYG, AngleCS, AngleSN \) (see model/inc/GRID.h)

4.8 Deep Convection

(in directory: verification/tutorial_deep_convection/)

Figure 4.39: Schematic of simulation domain for the surface driven convection experiment. The domain is doubly periodic with an initially uniform temperature of 20 °C.

This experiment, Figure 4.39, showcasing MITgcm’s non-hydrostatic capability, was designed to explore the temporal and spatial characteristics of convection plumes as they might exist during a period of oceanic deep convection. It is

- non-hydrostatic
- doubly-periodic with cubic geometry
discretized with 50 m resolution in $x, y, z$
- Cartesian
- on an $f$-plane
- using a linear equation of state

4.8.1 Overview

The model domain consists of an approximately 3 km square by 1 km deep box of initially unstratified, resting fluid. The domain is doubly periodic.

The experiment has 20 levels in the vertical, each of equal thickness $\Delta z = 50$ m (the horizontal resolution is also 50 m). The fluid is initially unstratified with a uniform reference potential temperature $\theta = 20$ °C. The equation of state used in this experiment is linear

$$\rho = \rho_0 (1 - \alpha_\theta \theta')$$  \hfill (4.54)

which is implemented in the model as a density anomaly equation

$$\rho' = -\rho_0 \alpha_\theta \theta'$$  \hfill (4.55)

with $\rho_0 = 1000$ kg m$^{-3}$ and $\alpha_\theta = 2 \times 10^{-4}$ degrees$^{-1}$. Integrated forward in this configuration, the model state variable $\theta$ is equivalent to either in-situ temperature, $T$, or potential temperature, $\theta$. For consistency with other examples, in which the equation of state is non-linear, we use $\theta$ to represent temperature here. This is the quantity that is carried in the model core equations.

As the fluid in the surface layer is cooled (at a mean rate of 800 W m$^{-2}$), it becomes convectively unstable and overturns, at first close to the grid-scale, but, as the flow matures, on larger scales (Figure 4.40 and Figure 4.41), under the influence of rotation ($f_o = 10^{-4}$ s$^{-1}$).

Model parameters are specified in file input/data. The grid dimensions are prescribed in code/SIZE.h. The forcing (file input/Qsurf.bin) is specified in a binary data file generated using the Matlab script input/gendata.m.

4.8.2 Equations solved

The model is configured in non-hydrostatic form, that is, all terms in the Navier Stokes equations are retained and the pressure field is found, subject to appropriate boundary conditions, through inversion of a 3-D elliptic equation.

The implicit free surface form of the pressure equation described in Marshall et al. (1997) [MHPA97] is employed. A horizontal Laplacian operator $\nabla_h^2$ provides viscous dissipation. The thermodynamic forcing appears as a sink in the equation for potential temperature $\theta$. This produces a set of equations solved in this configuration as follows:

\[
\begin{align*}
\frac{Du}{Dt} - fu &+ \frac{1}{\rho} \frac{\partial \rho'}{\partial x} - \nabla_h \cdot A_h \nabla_h u - \frac{\partial}{\partial z} A_z \frac{\partial u}{\partial z} = \begin{cases} 0 \text{ (surface)} \\ 0 \text{ (interior)} \end{cases} \\
\frac{Dv}{Dt} + fu &+ \frac{1}{\rho} \frac{\partial \rho'}{\partial y} - \nabla_h \cdot A_h \nabla_h v - \frac{\partial}{\partial z} A_z \frac{\partial v}{\partial z} = \begin{cases} 0 \text{ (surface)} \\ 0 \text{ (interior)} \end{cases} \\
\frac{Dw}{Dt} - g \frac{\rho'}{\rho} &+ \frac{1}{\rho} \frac{\partial \rho'}{\partial z} - \nabla_h \cdot A_h \nabla_h w - \frac{\partial}{\partial z} A_z \frac{\partial w}{\partial z} = \begin{cases} 0 \text{ (surface)} \\ 0 \text{ (interior)} \end{cases} \\
\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} + \frac{\partial w}{\partial z} &= 0 \\
\frac{D\theta}{Dt} - \nabla_h \cdot K_h \nabla_h \theta &- \frac{\partial}{\partial z} K_z \frac{\partial \theta}{\partial z} = \begin{cases} F_\theta \text{ (surface)} \\ 0 \text{ (interior)} \end{cases}
\end{align*}
\]
Figure 4.40: Vertical section

Figure 4.41: Surface section

4.8. Deep Convection
where \( u = \frac{Dx}{Dt} \), \( v = \frac{Dy}{Dt} \) and \( w = \frac{Dz}{Dt} \) are the components of the flow vector in directions \( x, y \) and \( z \). The pressure is diagnosed through inversion (subject to appropriate boundary conditions) of a 3-D elliptic equation derived from the divergence of the momentum equations and continuity (see Section 1.3.6).

### 4.8.3 Discrete numerical configuration

The domain is discretized with a uniform grid spacing in each direction. There are 64 grid cells in directions \( x \) and \( y \) and 20 vertical levels thus the domain comprises a total of just over 80,000 gridpoints.

### 4.8.4 Numerical stability criteria and other considerations

For a heat flux of 800 Wm\(^2\) and a rotation rate of \(10^{-4}\) s\(^{-1}\) the plume-scale can be expected to be a few hundred meters guiding our choice of grid resolution. This in turn restricts the timestep we can take. It is also desirable to minimize the level of diffusion and viscosity we apply.

For this class of problem it is generally the advective time-scale which restricts the timestep.

For an extreme maximum flow speed of \(|\vec{u}| = 1\) m s\(^{-1}\), at a resolution of 50 m, the implied maximum timestep for stability, \(\delta t_u\) is

\[
\delta t_u = \frac{\Delta x}{|\vec{u}|} = 50 \text{s}
\]

The choice of \(\delta t = 10\) s is a safe 20 percent of this maximum.

Interpreted in terms of a mixing-length hypothesis, a magnitude of Laplacian diffusion coefficient \(\kappa_h ( = \kappa_v ) = 0.1\) m\(^2\) s\(^{-1}\) is consistent with an eddy velocity of 2 mm s\(^{-1}\) correlated over 50 m.

### 4.8.5 Experiment configuration

The model configuration for this experiment resides under the directory verification/convection/. The experiment files

- code/CPP_OPTIONS.h
- code/SIZE.h
- input/data
- input/data.pkg
- input/eedata
- input/Qsurf.bin,

contain the code customizations and parameter settings for this experiment. Below we describe these experiment-specific customizations.

#### 4.8.5.1 File code/CPP_OPTIONS.h

This file uses standard default values and does not contain customizations for this experiment.
4.8.5.2 File code/SIZE.h

Listing 4.32: verification/tutorial_deep_convection/code/SIZE.h

CBOP
C    !ROUTINE: SIZE.h
C    !INTERFACE:
C    include SIZE.h
C    !DESCRIPTION:
C    *==========================================================*
C    | SIZE.h Declare size of underlying computational grid.
C    *==========================================================*
C    | The design here supports a three-dimensional model grid
C    | with indices I,J and K. The three-dimensional domain
C    | is comprised of nPx*nSx blocks (or tiles) of size sNx
C    | along the first (left-most index) axis, nPy*nSy blocks
C    | of size sNy along the second axis and one block of size
C    | Nr along the vertical (third) axis.
C    | Blocks/tiles have overlap regions of size OLx and OLy
C    | along the dimensions that are subdivided.
C    *==========================================================*
C    
C    Voodoo numbers controlling data layout:
C    sNx :: Number of X points in tile.
C    sNy :: Number of Y points in tile.
C    OLx :: Tile overlap extent in X.
C    OLy :: Tile overlap extent in Y.
C    nSx :: Number of tiles per process in X.
C    nSy :: Number of tiles per process in Y.
C    nPx :: Number of processes to use in X.
C    nPy :: Number of processes to use in Y.
C    Nx :: Number of points in X for the full domain.
C    Ny :: Number of points in Y for the full domain.
C    Nr :: Number of points in vertical direction.
C    *==========================================================*
C    
C    INTEGER sNx
C    INTEGER sNy
C    INTEGER OLx
C    INTEGER OLy
C    INTEGER nSx
C    INTEGER nSy
C    INTEGER nPx
C    INTEGER nPy
C    INTEGER Nx
C    INTEGER Ny
C    INTEGER Nr
C    PARAMETER ( sNx = 50,
C    &   sNy = 50,
C    &   OLx = 2,
C    &   OLy = 2,
C    &   nSx = 2,}

(continues on next page)
Three lines are customized in this file. These prescribe the domain grid dimensions.

- **Line 45.**

  ```plaintext
  sNx=50,
  ```

  this line sets the lateral domain extent in grid points for the axis aligned with the $x$-coordinate.

- **Line 46.**

  ```plaintext
  sNy=50,
  ```

  this line sets the lateral domain extent in grid points for the axis aligned with the $y$-coordinate.

- **Line 55.**

  ```plaintext
  Nr=50,
  ```

  this line sets the vertical domain extent in grid points.

### 4.8.5.3 File input/data

Listing 4.33: verification/tutorial_deep_convection/input/data

```plaintext
# | Model parameters |
# ===============
# Continuous equation parameters
&PARM01
  tRef=20*20.,
  sRef=20*35.,
  viscAh=4.E-2,
  viscAz=4.E-2,
  no_slip_sides=.FALSE.,
  no_slip_bottom=.FALSE.,
  diffKhT=4.E-2,
```
diffKzT=4.E-2,
f0=1.E-4,
beta=0.E-11,
tAlpha=2.0E-4,
sBeta =0.,
gravity=10.,
rhoConst=1000.,
rhoNil=1000.,
heatCapacity_Cp=4000.,
#rigidLid=.TRUE.,
implicitFreeSurface=.TRUE.,
#exactConserv=.TRUE.,
eosType='LINEAR',
nonHydrostatic=.TRUE.,
saltStepping=.FALSE.,
&

# Elliptic solver parameters
&PARM02
cg2dMaxIters=1000,
cg2dTargetResidual=1.E-9,
cg3dMaxIters=100,
cg3dTargetResidual=1.E-9,
&

# Time stepping parameters
&PARM03
nIter0=0,
endTime=43200.,
nTimeSteps=3,
deltaT=20.,
abEps=0.1,
pChkptFreq=43200.,
chkptFreq=7200.,
dumpFreq=1800.,
monitorFreq=600.,
monitorSelect=1,
monitorFreq=1.,
&

# Gridding parameters
&PARM04
usingCartesianGrid=.TRUE.,
dXspacing=20.,
dYspacing=20.,
delZ=50*20.,
&

# Input datasets
&PARM05
surfQnetFile= 'Qnet_p32.bin',
hyd rogThetaFile= 'T.120mn.bin',

(continued from previous page)
This file specifies the main parameters for the experiment. The parameters that are significant for this configuration are:

- **Line 7**, 
  \[ t_{\text{Ref}} = 20 \times 20.0, \]
  this line sets the initial and reference values of potential temperature at each model level in units of °C. Here the value is arbitrary since, in this case, the flow evolves independently of the absolute magnitude of the reference temperature. For each depth level the initial and reference profiles will be uniform in \( x \) and \( y \).

- **Line 8**, 
  \[ s_{\text{Ref}} = 20 \times 35.0, \]
  this line sets the initial and reference values of salinity at each model level in units of ppt. In this case salinity is set to an (arbitrary) uniform value of 35.0 ppt. However since, in this example, density is independent of salinity, an appropriately defined initial salinity could provide a useful passive tracer. For each depth level the initial and reference profiles will be uniform in \( x \) and \( y \).

- **Line 9**, 
  \[ \text{viscAh} = 0.1, \]
  this line sets the horizontal Laplacian dissipation coefficient to 0.1 \( \text{m}^2\text{s}^{-1} \). Boundary conditions for this operator are specified later.

- **Line 10**, 
  \[ \text{viscAz} = 0.1, \]
  this line sets the vertical Laplacian frictional dissipation coefficient to 0.1 \( \text{m}^2\text{s}^{-1} \). Boundary conditions for this operator are specified later.

- **Line 11**, 
  \[ \text{no_slip_sides} = \text{.FALSE.} \]
  this line selects a free-slip lateral boundary condition for the horizontal Laplacian friction operator e.g., \( \frac{\partial u}{\partial y} = 0 \) along boundaries in \( y \) and \( \frac{\partial v}{\partial x} = 0 \) along boundaries in \( x \).

- **Lines 12**, 
  \[ \text{no_slip_bottom} = \text{.TRUE.} \]
  this line selects a no-slip boundary condition for the bottom boundary condition in the vertical Laplacian friction operator e.g., \( u = v = 0 \) at \( z = -H \), where \( H \) is the local depth of the domain.

- **Line 13**, 
  \[ \text{diffKhT} = 0.1, \]
  this line sets the horizontal diffusion coefficient for temperature to 0.1 \( \text{m}^2\text{s}^{-1} \). The boundary condition on this operator is \( \frac{\partial}{\partial x} = \frac{\partial}{\partial y} = 0 \) at all boundaries.
• Line 14.

```
diffKzT=0.1,
```
this line sets the vertical diffusion coefficient for temperature to 0.1 m$^2$s$^{-1}$. The boundary condition on this operator is $\frac{\partial}{\partial z} = 0$ on all boundaries.

• Line 15.

```
f0=1E-4,
```
this line sets the Coriolis parameter to $1 \times 10^{-4}$ s$^{-1}$. Since $\beta = 0.0$ this value is then adopted throughout the domain.

• Line 16.

```
beta=0.E-11,
```
this line sets the the variation of Coriolis parameter with latitude to 0.

• Line 17.

```
tAlpha=2.E-4,
```
This line sets the thermal expansion coefficient for the fluid to $2 \times 10^{-4}$ °C$^{-1}$.

• Line 18.

```
sBeta=0,
```
This line sets the saline expansion coefficient for the fluid to 0, consistent with salt’s passive role in this example.

• Line 23-24.

```
rigidLid=.FALSE.,
implicitFreeSurface=.TRUE.,
```
Selects the barotropic pressure equation to be the implicit free surface formulation.

• Line 26.

```
eosType='LINEAR',
```
Selects the linear form of the equation of state.

• Line 27.

```
nonHydrostatic=.TRUE.,
```
Selects for non-hydrostatic code.

• Line 33.

```
cg2dMaxIters=1000,
```
Inactive - the pressure field in a non-hydrostatic simulation is inverted through a 3-D elliptic equation.

• Line 34.
Inactive - the pressure field in a non-hydrostatic simulation is inverted through a 3-D elliptic equation.

- Line 35,

\[
\text{cg3dMaxIters=40},
\]

This line sets the maximum number of iterations the 3-D conjugate gradient solver will use to 40, \textit{irrespective of the convergence criteria being met}.

- Line 36,

\[
\text{cg3dTargetResidual=1.E-9},
\]

Sets the tolerance which the 3-D conjugate gradient solver will use to test for convergence in equation (2.61) to \(1 \times 10^{-9}\). The solver will iterate until the tolerance falls below this value or until the maximum number of solver iterations is reached.

- Line 43,

\[
\text{nTimeSteps=8640.},
\]

Sets the number of timesteps at which this simulation will terminate (in this case 8640 timesteps or 1 day or \(\delta t = 10\) s). At the end of a simulation a checkpoint file is automatically written so that a numerical experiment can consist of multiple stages.

- Line 44,

\[
\text{deltaT=10},
\]

Sets the timestep \(\delta t\) to 10 s.

- Line 57,

\[
\text{dXspacing=50.0},
\]

Sets horizontal (\(x\)-direction) grid interval to 50 m.

- Line 58,

\[
\text{dYspacing=50.0},
\]

Sets horizontal (\(y\)-direction) grid interval to 50 m.

- Line 59,

\[
\text{delZ=20*50.0},
\]

Sets vertical grid spacing to 50 m. Must be consistent with \texttt{code/SIZE.h}. Here, 20 corresponds to the number of vertical levels.

- Line 64,

\[
\text{surfQfile='Qsurf.bin'}
\]

This line specifies the name of the file from which the surface heat flux is read. This file is a 2-D \((x, y)\) map. It is assumed to contain 64-bit binary numbers giving the value of \(Q\) (W m\(^{2}\)) to be applied in each surface grid cell, ordered with the \(x\) coordinate varying fastest. The points are ordered from low coordinate to high coordinate.
for both axes. The matlab program `input/gendata.m` shows how to generate the surface heat flux file used in the example.

### 4.8.5.4 File `input/data.pkg`

This file uses standard default values and does not contain customizations for this experiment.

### 4.8.5.5 File `input/eedata`

This file uses standard default values and does not contain customizations for this experiment.

### 4.8.5.6 File `input/Qsurf.bin`

The file `input/Qsurf.bin` specifies a 2-D \((x, y)\) map of heat flux values where \(Q = Q_0 \times (0.5 + \text{random number between 0 and 1})\).

In the example \(Q_0 = 800 \text{ W m}^{-2}\) so that values of \(Q\) lie in the range 400 to 1200 \text{ W m}^{-2}\) with a mean of \(Q_0\). Although the flux models a loss, because it is directed upwards, according to the model’s sign convention, \(Q\) is positive.

### 4.9 Gravity Plume On a Continental Slope

(in directory: verification/tutorial_plume_on_slope/)

![Temperature after 23 hours of cooling](image)

Figure 4.42: Temperature after 23 hours of cooling. The cold dense water is mixed with ambient water as it accelerates down the slope and hence is warmer than the unmixed plume.)
An important test of any ocean model is the ability to represent the flow of dense fluid down a slope. One example of such a flow is a non-rotating gravity plume on a continental slope, forced by a limited area of surface cooling above a continental shelf. Because the flow is non-rotating, a two dimensional model can be used in the across slope direction. The experiment is non-hydrostatic and uses open-boundaries to radiate transients at the deep water end. (Dense flow down a slope can also be forced by a dense inflow prescribed on the continental shelf; this configuration is being implemented by the DOME (Dynamics of Overflow Mixing and Entrainment) collaboration to compare solutions in different models).

The fluid is initially unstratified. The surface buoyancy loss $B_0$ (dimensions of $L^2T^{-3}$) over a cross-shelf distance $R$ causes vertical convective mixing and modifies the density of the fluid by an amount

$$\Delta \rho = \frac{B_0 \rho_0 t}{gH}$$

where $H$ is the depth of the shelf, $g$ is the acceleration due to gravity, $t$ is time since onset of cooling and $\rho_0$ is the reference density. Dense fluid slumps under gravity, with a flow speed close to the gravity wave speed:

$$U \sim \sqrt{gH} \sim \sqrt{\frac{g\Delta \rho H}{\rho_0}} \sim \sqrt{B_0 t}$$

A steady state is rapidly established in which the buoyancy flux out of the cooling region is balanced by the surface buoyancy loss. Then

$$U \sim (B_0 R)^{1/3} ; \Delta \rho \sim \frac{\rho_0}{gH} (B_0 R)^{2/3}$$

The Froude number of the flow on the shelf is close to unity (but in practice slightly less than unity, giving subcritical flow). When the flow reaches the slope, it accelerates, so that it may become supercritical (provided the slope angle $\alpha$ is steep enough). In this case, a hydraulic control is established at the shelf break. On the slope, where the Froude number is greater than one, and gradient Richardson number (defined as $Ri \sim g' h^*/U^2$ where $h^*$ is the thickness of the interface between dense and ambient fluid) is reduced below 1/4, Kelvin-Helmholtz instability is possible, and leads to entrainment of ambient fluid into the plume, modifying the density, and hence the acceleration down the slope. Kelvin-Helmholtz instability is suppressed at low Reynolds and Peclet numbers given by

$$Re \sim \frac{U h}{\nu} \sim \left(\frac{B_0 R}{\nu}\right)^{1/3} ; Pe = Re Pr$$

where $h$ is the depth of the dense fluid on the slope. Hence this experiment is carried out in the high $Re$, $Pe$ regime. A further constraint is that the convective heat flux must be much greater than the diffusive heat flux (Nusselt number $>> 1$). Then

$$Nu = \frac{U h^*}{\kappa} >> 1$$

Finally, since we have assumed that the convective mixing on the shelf occurs in a much shorter time than the horizontal equilibration, this implies $H/R << 1$.

Hence to summarize the important non-dimensional parameters, and the limits we are considering:

$$\frac{H}{R} << 1 ; Re >> 1 ; Pe >> 1 ; Nu >> 1 ; Ri < 1/4$$

In addition we are assuming that the slope is steep enough to provide sufficient acceleration to the gravity plume, but nonetheless much less that 1:1, since many Kelvin-Helmholtz billows appear on the slope, implying horizontal length scale of the slope $>>$ the depth of the dense fluid.
4.9.1 Configuration

The topography, spatial grid, forcing and initial conditions are all specified in binary data files generated using matlab script verification/tutorial_plume_on_slope/input/gendata.m and detailed in Section 4.9.2. Other model parameters are specified in input/data and input/data.obcs and detailed in Section 4.9.4.

4.9.2 Binary input data

![Horizontal grid spacing](image)

Figure 4.43: Horizontal grid spacing, $\Delta x$, in the across-slope direction for the gravity plume experiment.

The domain is 200 m deep and 6.4 km across. Uniform resolution of $60 \times 3^{1/3}$ m is used in the vertical and variable
Figure 4.44: Topography, $h(x)$, used for the gravity plume experiment.
Figure 4.45: Upward surface heat flux, $Q(x)$, used as forcing in the gravity plume experiment.
resolution of the form shown in Figure 4.43 with 320 points is used in the horizontal. The formula for $\Delta x$ is:

$$\Delta x(i) = \Delta x_1 + (\Delta x_2 - \Delta x_1)(1 + \tanh \left(\frac{i - i_s}{w}\right))/2$$

where

$$N_x = 320$$
$$L_x = 6400 \text{ (m)}$$
$$\Delta x_1 = \frac{2L_x}{3N_x} \text{ (m)}$$
$$\Delta x_2 = \frac{L_x/2}{N_x - L_x/(2\Delta x_1)} \text{ (m)}$$
$$i_s = \frac{L_x}{2\Delta x_1}$$
$$w = 40$$

Here, $\Delta x_1$ is the resolution on the shelf, $\Delta x_2$ is the resolution in deep water and $N_x$ is the number of points in the horizontal.

The topography, shown in Figure 4.44, is given by:

$$H(x) = -H_o + (H_o - h_s)(1 + \tanh \left(\frac{x - x_s}{L_s}\right))/2$$

where

$$H_o = 200 \text{ (m)}$$
$$h_s = 40 \text{ (m)}$$
$$x_s = 1500 + L_x/2 \text{ (m)}$$
$$L_s = \frac{(H_o - h_s)}{2s} \text{ (m)}$$
$$s = 0.15$$

Here, $s$ is the maximum slope, $H_o$ is the maximum depth, $h_s$ is the shelf depth, $x_s$ is the lateral position of the shelf-break and $L_s$ is the length-scale of the slope.

The forcing is through heat loss over the shelf, shown in Figure 4.45 and takes the form of a fixed flux with profile:

$$Q(x) = Q_o(1 + \tanh \left(\frac{x - x_q}{L_q}\right))/2$$

where

$$Q_o = 200 \text{ (W m}^{-2})$$
$$x_q = 2500 + L_x/2 \text{ (m)}$$
$$L_q = 100 \text{ (m)}$$

Here, $Q_o$ is the maximum heat flux, $x_q$ is the position of the cut-off, and $L_q$ is the width of the cut-off.

The initial temperature field is unstratified but with random perturbations, to induce convection early on in the run. The random perturbation are calculated in computational space and because of the variable resolution introduce some spatial correlations, but this does not matter for this experiment. The perturbations have range $0 - 0.01 \, ^\circ\text{K}$. 

270 Chapter 4. MITgcm Tutorial Example Experiments
4.9.3 Code configuration

The computational domain (number of gridpoints) is specified in code/SIZE.h and is configured as a single tile of dimensions $320 \times 1 \times 60$.

To compile the model code for this experiment, the non-hydrostatic algorithm needs to be enabled, and the open-boundaries package (pkg/obcs) is required:

- Non-hydrostatic terms and algorithm are enabled with \texttt{#define ALLOW\_NONHYDROSTATIC} in code/CPP\_OPTIONS.h and activated with \texttt{nonHydrostatic = .TRUE.}, in namelist PARM01 of input/data.
- Open boundaries are enabled by adding line \texttt{obcs} to package configuration file code/packages.conf and activated via \texttt{useOBCS = .TRUE.}, in namelist PACKAGES of input/data.pkg.

4.9.4 Model parameters

Table 4.1: Model parameters used in the gravity plume experiment.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$g$</td>
<td>$9.81$ m s$^{-2}$</td>
<td>acceleration due to gravity</td>
</tr>
<tr>
<td>$\rho_o$</td>
<td>$999.8$ kg m$^{-3}$</td>
<td>reference density</td>
</tr>
<tr>
<td>$\alpha$</td>
<td>$2 \times 10^{-4}$ K$^{-1}$</td>
<td>expansion coefficient</td>
</tr>
<tr>
<td>$A_h$</td>
<td>$1 \times 10^{-2}$ m$^2$ s$^{-1}$</td>
<td>horizontal viscosity</td>
</tr>
<tr>
<td>$A_v$</td>
<td>$1 \times 10^{-3}$ m$^2$ s$^{-1}$</td>
<td>vertical viscosity</td>
</tr>
<tr>
<td>$\kappa_h$</td>
<td>$0$ m$^2$ s$^{-1}$</td>
<td>(explicit) horizontal diffusion</td>
</tr>
<tr>
<td>$\kappa_v$</td>
<td>$0$ m$^2$ s$^{-1}$</td>
<td>(explicit) vertical diffusion</td>
</tr>
<tr>
<td>$\Delta t$</td>
<td>$20$ s</td>
<td>time step</td>
</tr>
<tr>
<td>$\Delta z$</td>
<td>$3.33333$ m</td>
<td>vertical grid spacing</td>
</tr>
<tr>
<td>$\Delta x$</td>
<td>$13.3333 - 39.5$ m</td>
<td>horizontal grid spacing</td>
</tr>
</tbody>
</table>

The model parameters (Table 4.1) are specified in input/data and if not assume the default values as defined in Section 3.8. A linear equation of state is used, eosType = 'LINEAR', but only temperature is active, $sBeta = 0.E-11$. For the given heat flux, $Q_o$, the buoyancy forcing is $B_o = \frac{g\alpha Q_o}{\rho_o c_p} \sim 10^{-7}$ m$^2$ s$^{-3}$. Using $R = 10^3$ m, the shelf width, this gives a velocity scale of $U \sim 5 \times 10^{-2}$ m s$^{-1}$ for the initial front but will accelerate by an order of magnitude over the slope. The temperature anomaly will be of order $\Delta \theta \sim 3 \times 10^{-2}$ K. The viscosity is constant and gives a Reynolds number of 100, using $h = 20$ m for the initial front and will be an order magnitude bigger over the slope. There is no explicit diffusion but a non-linear advection scheme is used for temperature which adds enough diffusion so as to keep the model stable. The time-step is set to $20$ s and gives Courant number order one when the flow reaches the bottom of the slope.

4.10 Biogeochemistry Simulation

(in directory: verification/tutorial_global_oce_biogeo/)

4.10. Biogeochemistry Simulation
4.10.1 Overview

This model overlays the dissolved inorganic carbon biogeochemistry model (pkg/dic) over a 2.8° global physical model. The physical model has 15 levels, and is forced with a climatological annual cycle of surface wind stresses (Trenberth et al. 1989 [TOL89]), surface heat and freshwater fluxes (Jiang et al. 1999 [JSR99]) with additional relaxation toward climatological sea surface temperature and salinity (Levitus and Boyer (1994a,b) [LB94a, LB94b]). It uses the Gent and McWilliams (1990) [GM90] eddy parameterization scheme, has an implicit free-surface, implicit vertical diffusion and uses the convective adjustment scheme.

The biogeochemical model considers the coupled cycles of carbon, oxygen, phosphorus and alkalinity. A simplified parameterization of biological production is used, limited by the availability of light and phosphate. A fraction of this productivity enters the dissolved organic pool pool, which has an e-folding timescale for remineralization of 6 months (following Yamanaka and Tajika 1997 [YT97]). The remaining fraction of this productivity is instantaneously exported as particulate to depth (Yamanaka and Tajika 1997 [YT97]) where it is remineralized according to the empirical power law relationship determined by Martin et al. (1987) [MK87]. The fate of carbon is linked to that of phosphorus by the Redfield ratio. Carbonate chemistry is explicitly solved (see Follow et al. 2006 [FID06]) and the air-sea exchange of CO$_2$ is parameterized with a uniform gas transfer coefficient following Wanninkhof (1992) [Wan92]. Oxygen is also linked to phosphorus by the Redfield ratio, and oxygen air-sea exchange also follows Wanninkhof (1992) [Wan92]. For more details see Dutkiewicz et al. (2005) [DSSaP05].

The example setup described here shows the physical model after 5900 years of spin-up and the biogeochemistry after 2900 years of spin-up. The biogeochemistry is at a pre-industrial steady-state (atmospheric ppmv is kept at 278). Five tracers are resolved: dissolved inorganic carbon ($DIC$), alkalinity ($ALK$), phosphate ($PO4$), dissolved organic phosphorus ($DOP$) and dissolved oxygen ($O2$).

![Figure 4.46: Modeled annual mean air-sea CO$_2$ flux (mol C m$^{-2}$ y$^{-1}$) for pre-industrial steady-state. Positive indicates flux of CO$_2$ from ocean to the atmosphere (out-gassing), contour interval is 1 mol C m$^{-2}$ y$^{-1}$]
4.10.2 Equations Solved

The physical ocean model velocity and diffusivities are used to redistribute the 5 tracers within the ocean. Additional redistribution comes from chemical and biological sources and sinks. For any tracer \( A \):

\[
\frac{\partial A}{\partial t} = -\nabla \cdot (\vec{u}^* A) + \nabla \cdot (K \nabla A) + S_A
\]

where \( \vec{u}^* \) is the transformed Eulerian mean circulation (which includes Eulerian and eddy-induced advection), \( K \) is the mixing tensor, and \( S_A \) are the sources and sinks due to biological and chemical processes.

The sources and sinks are:

\[
\begin{align*}
S_{\text{DIC}} &= F_{CO_2} + V_{CO_2} + r_{C:P} S_{PO_4} + J_{Ca} \\
S_{\text{ALK}} &= V_{ALK} - r_{N:P} S_{PO_4} + 2J_{Ca} \\
S_{PO_4} &= -f_{DOP} J_{\text{prod}} - \frac{\partial F_p}{\partial z} + \kappa_{\text{remin}} [DOP] \\
S_{DOP} &= f_{DOP} J_{\text{prod}} - \kappa_{\text{remin}} [DOP] \\
S_{O_2} &= \begin{cases} -r_{O:P} S_{PO_4} & \text{if } O_2 > O_{2\text{crit}} \\ 0 & \text{if } O_2 < O_{2\text{crit}} \end{cases}
\end{align*}
\]

where:

- \( F_{CO_2} \) is the flux of \( CO_2 \) from the ocean to the atmosphere
- \( V_{CO_2} \) is “virtual flux” due to changes in \( DIC \) due to the surface freshwater fluxes
- \( r_{C:P} \) is Redfield ratio of carbon to phosphorus
- \( J_{Ca} \) includes carbon removed from surface due to calcium carbonate formation and subsequent cumulation of the downward flux of \( CaCO_3 \)
- \( V_{ALK} \) is “virtual flux” due to changes in alkalinity due to the surface freshwater fluxes
- \( r_{N:P} \) Redfield ratio is nitrogen to phosphorus
- \( f_{DOP} \) is fraction of productivity that remains suspended in the water column as dissolved organic phosphorus
- \( J_{\text{prod}} \) is the net community productivity
- \( \frac{\partial F_p}{\partial z} \) is the accumulation of remineralized phosphorus with depth
- \( \kappa_{\text{remin}} \) is rate with which \( DOP \) remineralizes back to \( PO_4 \)
- \( F_{O_2} \) is air-sea flux of oxygen
- \( r_{O:P} \) is Redfield ratio of oxygen to phosphorus
- \( O_{2\text{crit}} \) is a critical level below which oxygen consumption if halted

These terms (for the first four tracers) are described more in Dutkiewicz et al. (2005) [DSSaPS05] and by McKinley et al. (2004) [MFM04] for the terms relating to oxygen.
4.10.3 Code configuration

The modifications to the code (in verification/tutorial_global_oce_biogeo/code) are:

- `code/SIZE.h`: which dictates the size of the model domain (128x64x15).
- `code/PTRACERS_SIZE.h`: which dictates how many tracers to assign how many tracers will be used (here, 5).
- `code/DIAGNOSTICS_SIZE.h`: assigns size information for the diagnostics package.
- `code/packages.conf`: which dictates which packages will be compiled in this version of the model - among the many that are used for the physical part of the model, this also includes pkg/ptracers, pkg/gchem, and pkg/dic which allow the biogeochemical part of this setup to function.

The input fields needed for this run (in verification/tutorial_global_oce_biogeo/input) are:

- `input/data`: specifies the main parameters for the experiment. Some parameters that may be useful to know: nTimeSteps number timesteps model will run, change to 720 to run for a year taveFreq frequency with which time averages are done, change to 31104000 for annual averages.
- `input/data.diagnostics`: specifies details of diagnostic pkg output
- `input/data.gchem`: specifies details needed in the biogeochemistry model run
- `input/data.gmredi`: specifies details for the GM parameterization
- `input/data.pkg`: set true or false for various packages to be used
- `input/data.ptracers`: details of the tracers to be used, including makes, diffusivity information and (if needed) initial files. Of particular importance is the PTRACERS_numInUse which states how many tracers are used, and PTRACERS_Iter0 which states at which timestep the biogeochemistry model tracers were initialized.
- `bathy.bin`: bathymetry data file
- `input/eedata`: This file uses standard default values and does not contain customizations for this experiment.
- `fice.bin`: ice data file, needed for the biogeochemistry
- `lev_monthly_salt.bin`: SSS values which model relaxes toward
- `lev_monthly_temp.bin`: SST values which model relaxes toward
- `pickup_0005184000.data`: variable and tendency values need to restart the physical part of the model
- `pickup_cd_0005184000.data`: variable and tendency values need to restart the cd pkg
- `pickup_ptracers_0005184000.data`: variable and tendency values need to restart the the biogeochemistry part of the model
- `shi_empmr_year.bin`: freshwater forcing data file
- `shi_qnet.bin`: heat flux forcing data file
- `sillev1.bin`: silica data file, need for the biogeochemistry
- `tren_speed.bin`: wind speed data file, needed for the biogeochemistry
- `tren_taux.bin`: meridional wind stress data file
- `tren_tauy.bin`: zonal wind stress data file
4.10.4 Running the example

As the model is set up to run in the verification experiment, it only runs for 4 timesteps (2 days) and outputs data at the end of this short run. For a more informative run, you will need to run longer. As set up, this model starts from a pre-spun up state and initializes physical fields and the biogeochemical tracers from the pickup files.

Physical data (e.g., S,T, velocities etc) will be output as for any regular ocean run. The biogeochemical output are:

- tracer snapshots: look in input/data.ptracers to see which number matches which type of tracer (e.g., ptracer01 is DIC).
- tracer time averages
- specific DIC diagnostics: these are averaged over taveFreq (set in input/data) and are specific to pkg/dic (currently are only available in binary format):
  - DIC_Biotave: 3-D biological community productivity (mol P m$^{-3}$ s$^{-1}$)
  - DIC_Cartave: 3-D tendencies due to calcium carbonate cycle (mol C m$^{-3}$ s$^{-1}$)
  - DIC_fluxCO2ave: 2-D air-sea flux of CO$_2$ (mol C m$^{-2}$ s$^{-1}$)
  - DIC_pCO2tave: 2-D partial pressure of CO$_2$ in surface layer
  - DIC_pHtave: 2-D pH in surface layer
  - DIC_Surtave: 2-D surface tendency of DIC due to air-sea flux and virtual flux (mol C m$^{-3}$ s$^{-1}$)
  - DIC_SurOtave: 2-D tendency due to air-sea flux of O$_2$ (mol O m$^{-3}$ s$^{-1}$)

4.11 Global Ocean State Estimation

(in directory: verification/tutorial_global_oce_optim/)

4.11.1 Overview

This experiment illustrates the optimization capacity of the MITgcm: here, a high level description.

In this tutorial, a very simple case is used to illustrate the optimization capacity of the MITgcm. Using an ocean configuration with realistic geography and bathymetry on a 4 x 4° spherical polar grid, we estimate a time-independent surface heat flux adjustment $Q_{netm}$ that attempts to bring the model climatology into consistency with observations (Levitus and Boyer (1994a,b) [LB94a, LB94b]).

This adjustment $Q_{netm}$ (a 2-D field only function of longitude and latitude) is the control variable of an optimization problem. It is inferred by an iterative procedure using an ‘adjoint technique’ and a least-squares method (see, for example, Stammer et al. (2002) [SWG+02] and Ferriera et al. (2005) [FMH05].

The ocean model is run forward in time and the quality of the solution is determined by a cost function, $J_1$, a measure of the departure of the model climatology from observations:

$$J_1 = \frac{1}{N} \sum_{i=1}^{N} \left[ \frac{T_i - T_i^{lev}}{\sigma_i^T} \right]^2$$

(4.56)

where $T_i$ and $T_i^{lev}$ are, respectively, the model and observed potential temperature at each grid point $i$. The differences are weighted by an $a$ priori uncertainty $\sigma_i^T$ on observations (as provided by Levitus and Boyer (1994a) [LB94a]). The error $\sigma_i^T$ is only a function of depth and varies from 0.5 K at the surface to 0.05 K at the bottom of the ocean, mainly reflecting the decreasing temperature variance with depth (see Figure 4.47a). A value of $J_1$ of order 1 means that the model is, on average, within observational uncertainties.
The cost function also places constraints on the adjustment to insure it is “reasonable”, i.e., of order of the uncertainties on the observed surface heat flux:

\[ J_2 = \frac{1}{N} \sum_{i=1}^{N} \left( \frac{Q_{\text{netm}}}{\sigma_i} \right)^2 \]

where \( \sigma_i^Q \) are the *a priori* errors on the observed heat flux as estimated by Stammer et al. (2002) [SWG+02] from 30% of local root-mean-square variability of the NCEP forcing field (see Figure 4.47b).

The total cost function is defined as \( J = \lambda_1 J_1 + \lambda_2 J_2 \) where \( \lambda_1 \) and \( \lambda_2 \) are weights controlling the relative contribution of the two components. The adjoint model then yields the sensitivities \( \partial J/\partial Q_{\text{netm}} \) of \( J \) relative to the 2-D fields \( Q_{\text{netm}} \). Using a line-searching algorithm (Gilbert and Lemaréchal 1989 [GLemarechal89]), \( Q_{\text{netm}} \) is adjusted then in the sense to reduce \( J \) — the procedure is repeated until convergence.

Figure 4.48 shows the results of such an optimization. The model is started from rest and from January-mean temperature and salinity initial conditions taken from the Levitus dataset. The experiment is run a year and the averaged temperature over the whole run (i.e., annual mean) is used in the cost function (4.56) to evaluate the model\(^1\). Only the top 2 levels are used. The first guess \( Q_{\text{netm}} \) is chosen to be zero. The weights \( \lambda_1 \) and \( \lambda_2 \) are set to 1 and 2, respectively. The total cost function converges after 15 iterations, decreasing from 6.1 to 2.7 (the temperature contribution decreases from 6.1 to 1.8 while the heat flux one increases from 0 to 0.42). The right panels of Figure 4.48 illustrate the evolution of the temperature error at the surface from iteration 0 to iteration 15. Unsurprisingly, the largest errors at iteration 0 (up to 6 °C, top left panels) are found in the Western boundary currents. After optimization, the departure of the model temperature from observations is reduced to 1 °C or less almost everywhere except in the Pacific equatorial cold tongue. Comparison of the initial temperature error (top, right) and heat flux adjustment (bottom, left) shows that the system

---

\(^1\) Because of the daily automatic testing, the experiment as found in the repository is set-up with a very small number of time-steps. To reproduce the results shown here, one needs to set \( \text{TimeSteps} = 360 \) and \( \text{lastinterval} = 31104000 \) (both corresponding to a year, see Section 4.11.3.2 for further details).
basically increased the heat flux out of the ocean where temperatures were too warm and vice-versa. Obviously, heat flux uncertainties are not solely responsible for temperature errors, and the heat flux adjustment partly compensates the poor representation of narrow currents (Western boundary currents, equatorial currents) at $4 \times 4^\circ$ resolution. This is allowed by the large \textit{a priori} error on the heat flux Figure 4.47. The Pacific cold tongue is a counter example: there, heat fluxes uncertainties are fairly small (about 20 W m$^{-2}$), and a large temperature errors remains after optimization.

Figure 4.48: Initial annual mean surface heat flux (top right in W m$^{-2}$) and adjustment obtained at iteration 15 (bottom right). Averaged difference between model and observed potential temperatures at the surface (in $^\circ$C) before optimization (iteration 0, top right) and after optimization (iteration 15, bottom right). Contour intervals for heat flux and temperature are 25 W m$^{-2}$ and 1 $^\circ$C, respectively. A positive flux is out of the ocean.
4.11.2 Implementation of the control variable and the cost function

One of the goals of this tutorial is to illustrate how to implement a new control variable. Most of this is fairly generic and is done in pkg/ctrl and pkg/cost. The modifications can be tracked by the CPP option ALLOW_HFLUXM_CONTROL or the comment cHFLUXM_CONTROL. The more specific modifications required for the experiment are found in verification/tutorial_global_oce_optim/code_ad. Here follows a brief description of the implementation.

4.11.2.1 The control variable

The adjustment $Q^\text{netm}$ is activated by setting `#define ALLOW_HFLUXM_CONTROL` in code_ad/CTRL_OPTIONS.h.

It is first implemented as a “normal” forcing variable. It is defined in model/inc/FFIELDS.h, initialized to zero in model/src/ini_forcing.F, and then used in model/src/external_forcing_surf.F. $Q^\text{netm}$ is made a control variable in pkg/ctrl by modifying the following subroutines:

- pkg/ctrl/ctrl_init.F where $Q^\text{netm}$ is defined as the control variable number 24,
- pkg/ctrl/ctrl_pack.F which writes, at the end of each iteration, the sensitivity of the cost function $\partial J / \partial Q^\text{netm}$ in to a file to be used by the line-search algorithm,
- pkg/ctrl/ctrl_unpack.F which reads, at the start of each iteration, the updated adjustment as provided by the line-search algorithm,
- pkg/ctrl/ctrl_map_forcing.F in which the updated adjustment is added to the first guess $Q^\text{netm}$.

Note also some minor changes in pkg/ctrl.h, pkg/ctrl/readparms.F, and pkg/ctrl_dummy.h (xx_hfluxm_file, fname_hfluxm, xx_hfluxm_dummy).

4.11.2.2 Cost functions

The cost functions are implemented using pkg/cost.

- The temperature cost function $J_1$ which measures the drift of the mean model temperature from the Levitus climatology is implemented in /verification/tutorial_global_oce_optim/code_ad/cost_temp.F. It is activated by `#define ALLOW_COST_TEMP` in code_ad/COST_OPTIONS.h. It requires the mean temperature of the model which is obtained by accumulating the temperature in pkg/cost/cost_tile.F (called at each time step). The value of the cost function is stored in objf_temp and its weight $\lambda_1$ in mult_temp.
- The heat flux cost function, penalizing the departure of the surface heat flux from observations is implemented in /verification/tutorial_global_oce_optim/code_ad/cost_hflux.F, and activated by `#define ALLOW_COST_HFLUXM` in code_ad/COST_OPTIONS.h. The value of the cost function is stored in objf_hfluxm and its weight $\lambda_2$ in mult_hflux.
- The subroutine pkg/cost/cost_final.F calls the cost function subroutines and makes the (weighted) sum of the various contributions.
- The various weights used in the cost functions are read in /verification/tutorial_global_oce_optim/code_ad/cost_weights.F. The weight of the cost functions are read in pkg/cost/readparms.F from the input file verification/tutorial_global_oce_optim/input_ad/data.cost.
4.11.3 Code Configuration

The experiment files in verification/tutorial_global_oce_optim/code_ad/ and verification/tutorial_global_oce_optim/input_ad/ contain the code customizations and parameter settings. Most of them are identical to those used in the Global Ocean (experiment verification/tutorial_global_oce_latlon/). Below, we describe some of the customizations required for this experiment.

4.11.3.1 Compilation-time customizations in code_ad

In code_ad/CTRL_OPTIONS.h:

• #define ALLOW_ECCO_OPTIMIZATION

4.11.3.2 Running-time customizations in input_ad

• input_ad/data: note the smaller cg2dTargetResidual than in the forward-only experiment,
• input_ad/data.optim specifies the iteration number,
• input_ad/data.ctrl is used, in particular, to specify the name of the sensitivity and adjustment files associated to a control variable,
• input_ad/data.cost: parameters of the cost functions, in particular lastinterval specifies the length of time-averaging for the model temperature to be used in the cost function (4.56),
• input_ad/data.pkg: note that the Gradient Check package is turned on by default (useGrdchk = .TRUE.),
• Err_hflux.bin and Err_levitus_15layer.bin are the files containing the heat flux and potential temperature uncertainties, respectively.

4.11.4 Compiling

The optimization experiment requires two executables: 1) the MITgcm and its adjoint (mitgcmuv_ad) and 2) the line-search algorithm (optim.x).

4.11.4.1 Compilation of MITgcm and its adjoint: mitgcmuv_ad

Before compiling, first note that in the directory code_ad, two files must be updated:

• code_ad/code_ad_diff.list which lists new subroutines to be compiled by the TAF software (code_ad/cost_temp.F and code_ad/cost_hflux.F),
• the file code_ad/ad_optfile.local provides a list of the control variables and the name of cost function to the TAF software.

Then, in the directory build, type:

```bash
% ../../../tools/genmake2 -mods=../code_ad -adof=../code_ad/ad_optfile.local
% make depend
% make adall
```

to generate the MITgcm executable mitgcmuv_ad.
4.11.4.2 Compilation of the line-search algorithm: optim.x

This is done from the directories lsopt/ and optim/ (found in the top MITgcm directory). In lsopt/, unzip the bash1 library adapted to your platform (see lsopt/README), and change the Makefile accordingly. Compile with:

```bash
% make all
```

(more details in lsopt/lsopt_doc.txt)

In optim/, the path of the directory where mitgcm_ad was compiled must be specified in the Makefile in the variable INCLUDEDIRS. The file name of the control variable (here, xx_hfluxm_file) must be added to the namelist read by optim/optim_numbmod.F. Then use

```bash
% make depend
```

and

```bash
% make
```

to generate the line-search executable optim.x.

4.11.5 Running the estimation

Make a new subdirectory input_ad/OPTIM. Copy the mitgcmuv_ad executable to input_ad and optim.x to this subdirectory. cd into input_ad/. The first iteration is somewhat particular and is best done “by hand” while the following iterations can be run automatically (see below). Check that the iteration number is set to 0 in input_ad/data.optim and run MITgcm:

```bash
% ./mitgcmuv_ad
```

The output files adxx_hfluxm.0000000000.* and xx_hfluxm.0000000000.* contain the sensitivity of the cost function to \( Q_{netm} \) and the adjustment to \( Q_{netm} \) (zero at the first iteration), respectively. Two other files called costhflux_tut_MITgcm.opt0000 and ctrlhflux_tut_MITgcm.opt0000 are also generated. They essentially contain the same information as the adxx_.hfluxm* and xx_.hfluxm* files, but in a compressed format. These two files are the only ones involved in the communication between the adjoint model mitgcm_ad and the line-search algorithm optim.x. Only at the first iteration, are they both generated by mitgcmuv_ad. Subsequently, costhflux_tut_MITgcm.opt \( n \) is an output of the adjoint model at iteration \( n \) and an input of the line-search. The latter returns an updated adjustment in ctrlhflux_tut_MITgcm.opt \( n + 1 \) to be used as an input of the adjoint model at iteration \( n + 1 \).

At the first iteration, move costhflux_tut_MITgcm.opt0000 and ctrlhflux_tut_MITgcm.opt0000 to input_ad/OPTIM, move into this directory and link input_ad/data.optim and input_ad/data.ctrl locally:

```bash
% cd OPTIM/
% ln -s ../data.optim .
% ln -s ../data.ctrl .
```

The target cost function \( f_{min} \) needs to be specified in input_ad/data.optim: as a rule of thumb, it should be about 0.95-0.90 times the value of the cost function at the first iteration. This value is only used at the first iteration and does not need to be updated afterward. However, it implicitly specifies the “pace” at which the cost function is going down (if you are lucky and it does indeed diminish!).

Once this is done, run the line-search algorithm:

```bash
% ./optim.x
```
4.12 Adjoint Sensitivity Analysis for Tracer Injection

(in directory: verification/tutorial_tracer_adjsens/)

MITgcm has been adapted to enable AD using TAMC or TAF. The present description is specific to the use of TAMC or TAF as AD tool. The following sections describe the steps which are necessary to generate a tangent linear or adjoint model of MITgcm. We take as an example the sensitivity of carbon sequestration in the ocean. The AD-relevant hooks in the code are outlined in Section 7.2 and Section 7.2.4.4.

4.12.1 Overview of the experiment

We describe an adjoint sensitivity analysis of out-gassing from the ocean into the atmosphere of a carbon-like tracer injected into the ocean interior (see Hill et al. 2004 [HBFM04]).

4.12.1.1 Passive tracer equation

For this work, MITgcm was augmented with a thermodynamically inactive tracer, $C$. Tracer residing in the ocean model surface layer is out-gassed according to a relaxation time scale, $\mu$. Within the ocean interior, the tracer is passively advected by the ocean model currents. The full equation for the time evolution

$$\frac{\partial C}{\partial t} = -U \cdot \nabla C - \mu C + \Gamma(C) + S$$

(4.57)

also includes a source term $S$. This term represents interior sources of $C$ such as would arise due to direct injection. The velocity term, $U$, is the sum of the model Eulerian circulation and an eddy-induced velocity, the latter parameterized according to Gent/McWilliams (Gent and McWilliams 1990 [GM90]; Gent et al. (1995) [GWMM95]). The convection function, $\Gamma$, mixes $C$ vertically wherever the fluid is locally statically unstable.

The out-gassing time scale, $\mu$, in (4.57) is set so that $1/\mu \sim 1$ year for the surface ocean and $\mu = 0$ elsewhere. With this value, (4.57) is valid as a prognostic equation for small perturbations in oceanic carbon concentrations. This configuration provides a powerful tool for examining the impact of large-scale ocean circulation on $CO_2$ out-gassing due to interior injections. As source we choose a constant in time injection of $S = 1$ mol s$^{-1}$.

4.12.1.2 Model configuration

The model configuration employed has a constant $4^\circ \times 4^\circ$ resolution horizontal grid and realistic geography and bathymetry. Twenty vertical layers are used with vertical spacing ranging from 50 m near the surface to 815 m at depth. Driven to steady-state by climatological wind-stress, heat and fresh-water forcing, the model reproduces well known large-scale features of the ocean general circulation.
4.12.1.3 Out-gassing cost function

To quantify and understand out-gassing due to injections of $C$ in (4.57), we define a cost function $\mathcal{J}$ that measures the total amount of tracer out-gassed at each timestep:

$$
\mathcal{J}(t = T) = \int_{t=0}^{t=T} \int_{A} \mu C \, dA \, dt
$$

(4.58) integrates the out-gassing term, $\mu C$, from (4.57) over the entire ocean surface area, $A$, and accumulates it up to time $T$. Physically, $\mathcal{J}$ can be thought of as representing the amount of CO$_2$ that our model predicts would be out-gassed following an injection at rate $S$. The sensitivity of $\mathcal{J}$ to the spatial location of $S$, $\frac{\partial \mathcal{J}}{\partial S}$, can be used to identify regions from which circulation would cause CO$_2$ to rapidly out-gas following injection and regions in which CO$_2$ injections would remain effectively sequestered within the ocean.

4.12.2 Code configuration

The code customization routines are in verification/tutorial_tracer_adjsens/code_ad:

- verification/tutorial_tracer_adjsens/code_ad/COST_OPTIONS.h
- verification/tutorial_tracer_adjsens/code_ad/CTRL_OPTIONS.h
- verification/tutorial_tracer_adjsens/code_ad/CPP_OPTIONS.h
- verification/tutorial_tracer_adjsens/code_ad/AUTODIFF_OPTIONS.h
- verification/tutorial_tracer_adjsens/code_ad/CTRL_SIZE.h
- verification/tutorial_tracer_adjsens/code_ad/GAD_OPTIONS.h
- verification/tutorial_tracer_adjsens/code_ad/GMREDI_OPTIONS.h
- verification/tutorial_tracer_adjsens/code_ad/SIZE.h
- verification/tutorial_tracer_adjsens/code_ad/tamc.h
- verification/tutorial_tracer_adjsens/code_ad/ctrl_map_ini_genarr.F
- verification/tutorial_tracer_adjsens/code_ad/ptracers_forcing_surf.F
- verification/tutorial_tracer_adjsens/code_ad/packages.conf

The runtime flag and parameters settings are contained in verification/tutorial_tracer_adjsens/input/ and verification/tutorial_tracer_adjsens/input_ad/, together with the forcing fields and and restart files:

- verification/tutorial_tracer_adjsens/input/ad
- verification/tutorial_tracer_adjsens/input/ad/data
- verification/tutorial_tracer_adjsens/input/ad/data.cost
- verification/tutorial_tracer_adjsens/input/ad/data.ctrl
- verification/tutorial_tracer_adjsens/input/ad/data.gmredi
- verification/tutorial_tracer_adjsens/input/ad/data.grdchk
- verification/tutorial_tracer_adjsens/input/ad/data.optim
- verification/tutorial_tracer_adjsens/input/ad/data.pkg
- verification/tutorial_tracer_adjsens/input/ad/data.ptracers
- verification/tutorial_tracer_adjsens/input/ad/eedata
- verification/tutorial_tracer_adjsens/input/ad/topog.bin
Below we describe the customizations of this files which are specific to this experiment.

### 4.12.2.1 File `code_ad/COST_OPTIONS.h` /

This file contains package-specific CPP-options (see Section 7.2.4).

### 4.12.2.2 File `code_ad/CTRL_OPTIONS.h` /

This file contains package-specific CPP-options (see Section 10.3).

### 4.12.2.3 File `code_ad/CPP_OPTIONS.h` /

This file contains model-specific CPP options (see Section 3.7). Most options are related to the forward model setup. They are identical to the global steady circulation setup of `verification/global_ocean.90x40x15/`. The three options specific to this experiment are as follows. 

- **#define ALLOW_PASSIVE_TRACER**: enables the code to carry through the advection/diffusion of a passive tracer along the model integration.
- **#define ALLOW_MIT_ADJOINT_RUN**: enables the inclusion of some AD-related fields concerning initialization, link between control variables and forward model variables, and the call to the top-level forward/adjoint subroutine `adthe_main_loop.F` instead of `model/src/the_main_loop.F`. 
- **#define ALLOW_GRADIENT_CHECK**: enables the gradient check package. After computing the unperturbed cost function and its gradient, a series of computations are performed for which:
  - an element of the control vector is perturbed
  - the cost function w.r.t. the perturbed element is computed
  - the difference between the perturbed and unperturbed cost function is computed to compute the finite difference gradient
  - the finite difference gradient is compared with the adjoint-generated gradient.

The gradient check package is further described in Section 7.3.

### 4.12.2.4 File `ECCO_OPTIONS.h`

The CPP options of several AD-related packages are grouped in this file:

- Overall ECCO-related execution modus:
  These determine whether a pure forward run, a sensitivity run or an iteration of optimization is performed. These options are not needed in the present context.
- Adjoint support package: `pkg/autodiff/`
  This package contains hand-written adjoint code such as active file handling, flow directives for files which must not be differentiated, and TMC-specific header files. 
  - **#define ALLOW_AUTODIFF_TAMC**: defines TMC-related features in the code.
  - **#define ALLOW_TAMC_CHECKPOINTING**: enables the checkpointing feature of TMC (see Section 7.1.3). In the present example a 3-level checkpointing is implemented. The code contains the relevant store directives, common block and tape initializations, storing key computation, and loop index handling. The checkpointing length at each level is defined in file `code_ad/tamc.h`, see below. The out
and intermediate loop directives are contained in the files pkg/autodiff/checkpoint_lev3_directives.h, pkg/autodiff/checkpoint_lev2_directives.h. \texttt{#define ALLOW_AUTODIFF_MONITOR} enables the monitoring of intermediate adjoint variables (see Section 7.2.5.4). \texttt{#define ALLOW_DIVIDED_ADJOINT} enables adjoint dump and restart (see Section 7.4).

- **Cost function package: pkg/cost/**
  This package contains all relevant routines for initializing, accumulating and finalizing the cost function (see Section 7.2.4). \texttt{#define ALLOW_COST} enables all general aspects of the cost function handling, in particular the hooks in the forward code for initializing, accumulating and finalizing the cost function. \texttt{#define ALLOW_COST_TRACER} includes the call to the cost function for this particular experiment, eqn. (4.58).

- **Control variable package: pkg/ctrl/**
  This package contains all relevant routines for the handling of the control vector. Each control variable can be enabled/disabled with its own flag:

  \begin{verbatim}
  #define ALLOW_THETA0_CONTROL initial temperature
  #define ALLOW_SALT0_CONTROL initial salinity
  #define ALLOW_TR10_CONTROL initial passive tracer concentration
  #define ALLOW_TAUU0_CONTROL zonal wind stress
  #define ALLOW_TAUV0_CONTROL meridional wind stress
  #define ALLOW_SFLUX0_CONTROL freshwater flux
  #define ALLOW_HFLUX0_CONTROL heat flux
  #define ALLOW_DIFFKR_CONTROL diapycnal diffusivity
  #undef ALLOW_KAPGM_CONTROL isopycnal diffusivity
  \end{verbatim}

### 4.12.2.5 File SIZE.h

Listing 4.34: verification/tutorial_global_oce_latlon/code/SIZE.h

```
CBOP
C  !ROUTINE: SIZE.h
C  !INTERFACE:
C  include SIZE.h
C  !DESCRIPTION: \bv
C  *==========================================================*
C  | SIZE.h Declare size of underlying computational grid.
C  *==========================================================*
C  | The design here supports a three-dimensional model grid
C  | with indices I,J and K. The three-dimensional domain
C  | is comprised of nPx*nSx blocks (or tiles) of size sNx
C  | along the first (left-most index) axis, nPy*nSy blocks
C  | of size sNy along the second axis and one block of size
C  | Nr along the vertical (third) axis.
C  | Blocks/tiles have overlap regions of size OLx and OLy
C  | along the dimensions that are subdivided.
C  *==========================================================*
C  \ev
C
C  Voodoo numbers controlling data layout:
C  sNx :: Number of X points in tile.
C  sNy :: Number of Y points in tile.
C  OLx :: Tile overlap extent in X.
(continues on next page)
```
The file contains the grid point dimensions of the forward model. It is identical to the verification/exp2/.
4.12.2.6 File /pkg/autodiff/adcommon.h

This file contains common blocks of some adjoint variables that are generated by TAMC. The common blocks are used by the adjoint support routine /pkg/autodiff/addummy_in_stepping.F which needs to access those variables:

<table>
<thead>
<tr>
<th>Common Block</th>
<th>Related File</th>
</tr>
</thead>
<tbody>
<tr>
<td>/addynvars_r/</td>
<td>model/inc/DYNVARS.h</td>
</tr>
<tr>
<td>/addynvars_cd/</td>
<td>model/inc/DYNVARS.h</td>
</tr>
<tr>
<td>/addynvars_diffr/</td>
<td>model/inc/DYNVARS.h</td>
</tr>
<tr>
<td>/addynvars_kapr/</td>
<td>model/inc/DYNVARS.h</td>
</tr>
<tr>
<td>/adtr1_r/</td>
<td>TR1.h</td>
</tr>
<tr>
<td>/adffields/</td>
<td>model/inc/FFIELDS.h</td>
</tr>
</tbody>
</table>

Note that if the structure of the common block changes in the above header files of the forward code, the structure of the adjoint common blocks will change accordingly. Thus, one must make sure that the structure of the adjoint common block in the hand-written file /pkg/autodiff/adcommon.h complies with the automatically generated adjoint common blocks in adjoint_model.F. The header file is enabled via the CPP-option ALLOW_AUTODIFF_MONITOR.

4.12.2.7 File code_ad/tamc.h

This routine contains the dimensions for TAMC checkpointing and some indices relevant for storing ky computations.

- #ifdef ALLOW_TAMC_CHECKPOINTING

  3-level checkpointing is enabled, i.e., the time stepping is divided into three different levels (see Section 7.1.3). The model state of the outermost (nchklev_3) and the intermediate (nchklev_2) time stepping loop are stored to file (handled in model/src/the_main_loop.F). The innermost loop (nchklev_1) avoids I/O by storing all required variables to common blocks. This storing may also be necessary if no checkpointing is chosen (nonlinear functions, if-statements, iterative loops, ...). In the present example the dimensions are chosen as follows:

  nchklev_1 = 36
  nchklev_2 = 30
  nchklev_3 = 60

  To guarantee that the checkpointing intervals span the entire integration period the following relation must be satisfied:

  nchklev_1 * nchklev_2 * nchklev_3 \geq nTimeSteps

  where nTimeSteps is either specified in input_ad/data or computed via:

  nTimeSteps = (endTime - startTime) / deltaTClock.

- #undef ALLOW_TAMC_CHECKPOINTING

  No checkpointing is enabled. In this case the relevant counter is nchklev_0. Similar to above, the following relation has to be satisfied:

  nchklev_0 \geq nTimeSteps
The following parameters may be worth describing: isbyte, maxpass.

### 4.12.2.8 File `makefile`

This file contains all relevant parameter flags and lists to run TAMC or TAF. It is assumed that TAMC is available to you, either locally, being installed on your network, or remotely through the 'TAMC Utility'. TAMC is called with the command `tamc` followed by a number of options. They are described in detail in the TAMC manual (Giering 1999 [Gie99]). Here we briefly discuss the main flags used in the `makefile`. The standard output for TAF is written to file `taf.log`.

**TAMC:**

- `-input «variable names»`
- `-output «variable name»`
- `-i4 -r4 ...`
- `-toplevel «S/R name»`
- `-reverse «file names»`

**TAF:**

- `-input «variable names»`
- `-output «variable name»`
- `-i4 -r4 ...`
- `-toplevel «S/R name»`
- `-reverse «file names»`
- `-flow taf_flow.log`
- `-nonew_arg`

- `-toplevel «S/R name»`
  Name of the toplevel routine, with respect to which the control flow analysis is performed.

- `input «variable names»`
  List of independent variables $u$ with respect to which the dependent variable $J$ is differentiated.

- `-output «variable name»`
  Dependent variable $J$ which is to be differentiated.

- `-reverse «file names»`
  Adjoint code is generated to compute the sensitivity of an independent variable w.r.t. many dependent variables. In the discussion of Section 7 the generated adjoint top-level routine computes the product of the transposed Jacobian matrix $M^T$ times the gradient vector $\nabla_v J$. «file names» refers to the list of files .f which are to be analyzed by TAMC. This list is generally smaller than the full list of code to be compiled. The files not contained are either above the top-level routine (some initializations), or are deliberately hidden from TAMC, either because hand-written adjoint routines exist, or the routines must not (or don't have to) be differentiated. For each routine which is part of the flow tree of the top-level routine, but deliberately hidden from TAMC (or for each package which contains such routines), a corresponding file .flow exists containing flow directives for TAMC.

- `-i4 -r4`

- `-flow taf_flow.log`
  Will cause TAF to produce a flow listing file named `taf_flow.log` in which the set of active and passive variables are identified for each subroutine.

- `-nonew_arg`
  The default in the order of the parameter list of adjoint routines has changed. Before TAF 1.3 the default was compatible with the TAMC-generated list. As of TAF 1.3 the order of adjoint routine parameter lists is no longer compatible with TAMC. To restore compatibility when using TAF 1.3 and higher, this argument is needed. It is currently crucial to use since all hand-written adjoint routines refer to the TAMC default.
File `input/topog.bin`

Contains 2-D bathymetry information.

Files `input/windx.bin`, `input/windy.bin`, `input/salt.bin`, `input/theta.bin`, `input/SSS.bin`, `input/SST.bin`

These contain the initial values of salinity and potential temperature (`salt.bin`, `theta.bin`), surface boundary values (surface wind stresses `windx.bin`, `windy.bin`), and surface restoring fields (`SSS.bin`, `SST.bin`).

### 4.12.3 Compiling the model and its adjoint

The build process of the adjoint model is slightly more complex than that of compiling the forward code. The main reason is that the adjoint code generation requires a specific list of routines that are to be differentiated (as opposed to the automatic generation of a list of files to be compiled by `genmake2`). This list excludes routines that don’t have to be or must not be differentiated. For some of the latter routines, flow directives may be necessary, a list of which has to be given as well. For this reason, a separate `makefile` is currently maintained in the directory `adjoint/`. This `makefile` is responsible for the adjoint code generation.

In the following, we describe the build process step by step, assuming you are in the directory `bin/`. A summary of steps to follow is given at the end.

#### 4.12.3.1 Adjoint code generation and compilation – step by step

1. `ln -s ../verification/???/code/.genmakerc .
   
   Link your customized `genmake` options, header files, and modified code to the compile directory.

2. `../tools/genmake -makefile`
   
   Generate your `Makefile` (see Section 3.5.2).

3. `make depend`
   
   Dependency analysis for the CPP pre-compiler (see Section 3.5.1).

4. `cd ../adjoint`
   
   `make adtaf` or `make adtamc`
   
   Depending on whether you have TAF or TAMC at your disposal, you’ll choose `adtaf` or `adtamc` as your make target for the `makefile` in the directory `adjoint/`. Several things happen at this stage.

   • `make adrestore`
     
     `make ftlrestore`
     
     The initial template files `adjoint_model.F` and `tangentlinear_model.F` in `pkg/autodiff` which are part of the compiling list created by `genmake2` are restored.

   • `make depend, make small_f`
     
     The bin/ directory is brought up to date, i.e., for recent changes in header or source code . [Fh], corresponding .f routines are generated or re-generated. Note that here, only CPP pre-compiling is performed; no object code .o is generated as yet. Pre-compiling is necessary for TAMC to see the full code.

   • `make allcode`
     
     All Fortran routines .f in bin/ are concatenated into a single file called `tmc_code.f`.

   • `make admodeltaf/admodeltamc`

---

Chapter 4. MITgcm Tutorial Example Experiments
Adjoint code is generated by TAMC or TAF. The adjoint code is written to the file \texttt{tamc\_code\_ad.f}. It contains all adjoint routines of the forward routines concatenated in \texttt{tamc\_code.f}. For a given forward routine subroutine routinename the adjoint routine is named \texttt{adsubroutine routinename} by default (that default can be changed via the flag \texttt{-admark «markname}}). Furthermore, it may contain modified code which incorporates the translation of adjoint store directives into specific Fortran code. For a given forward routines subroutine routinename the modified routine is named \texttt{mdsubroutine routinename}. TAMC or TAF info is written to file \texttt{tamc\_code.prot} or \texttt{taf.log}, respectively.

- \textbf{make adchange}
  
  The multi-threading capability of MITgcm requires a slight change in the parameter list of some routines that are related to to active file handling. This post-processing invokes the sed script \texttt{tools/adjoint\_sed} to insert the threading counter \texttt{myThId} into the parameter list of those subroutines. The resulting code is written to file \texttt{tamc\_code\_sed\_ad.f} and appended to the file \texttt{adjoint\_model.F}. This concludes the adjoint code generation.

5. cd ../bin
   make

   The file \texttt{adjoint\_model.F} now contains the full adjoint code. All routines are now compiled.

N.B.: The targets \texttt{make adtaf/adtamc} now comprise a series of targets that in previous versions had to be invoked separately. This was probably preferable at a more experimental stage, but has now been dropped in favor of a more straightforward build process.

\textbf{Adjoint code generation and compilation – summary}

\begin{verbatim}
cd bin
ln -s ../verification/my_experiment/code/.genmakerc .
ln -s ../verification/my_experiment/code/*.[Fh] .
./tools/genmake -makefile
make depend
cd ../adjoint
make adtaf <OR: make adtamc>
    contains the targets:
    adrestore small_f allcode admodeltaf/admodeltamc adchange
cd ../bin
make
\end{verbatim}
4.13 Offline Experiments

(in directory: verification/tutorial_cfc_offline/)

This document describes two experiments using the offline form of the MITgcm.

4.13.1 Overview

The first experiment demonstrates use of the offline form of the MITgcm to study advection of a passive tracer. Time-averaged flow-fields and mixing coefficients, deriving from a prior online run, are re-used leaving only the tracer equation to be integrated. This first experiment’s run configuration is specified in directory verification/tutorial_cfc_offline/input_tutorial.

Figure — missing figure — shows a movie of tracer being advected using the offline package of the MITgcm. In the top panel the frames of the movie show the monthly surface evolution of an initially local source of passive tracer. In the lower panel, the frames of the movie show the changing monthly surface evolution where the initial tracer field had a global distribution.

The second experiment, a more complicated example exploring contamination of the global ocean through surface exposure to CFCs during the last century, is described after this more simple first example. The run configuration for this second experiment is specified in directory verification/tutorial_cfc_offline/input.

4.13.2 Time-stepping of tracers

See Section 2.16 and Section 2.17 for details of available tracer time-stepping schemes and their characteristics.

4.13.3 Code Configuration

The experiment files

- verification/tutorial_cfc_offline/input_tutorial/data
- verification/tutorial_cfc_offline/input_tutorial/data.off
- verification/tutorial_cfc_offline/input_tutorial/data.pkg
- verification/tutorial_cfc_offline/input_tutorial/data.ptracers
- verification/tutorial_cfc_offline/input_tutorial/eedata
- verification/tutorial_cfc_offline/code/packages.conf
- verification/tutorial_cfc_offline/code/PTRACERS_SIZE.h
- verification/tutorial_cfc_offline/code/GMREDI_OPTIONS.h
- verification/tutorial_cfc_offline/code/SIZE.h

contain the code customizations and parameter settings required to run the example. In addition the following binary data files are required:

- input/depth_g77.bin
- pickup_ptracers.0004269600, pickup_ptracers.0004269600.meta
- binary files in verification/tutorial_cfc_offline/input/input_off
4.13.3.1 File input_tutorial/data

Listing 4.35: verification/tutorial_cfc_offline/input_tutorial/data

```plaintext
# ====================
# | Model parameters |
# ====================
#
# Continuous equation parameters
&PARM01
implicitDiffusion=.TRUE.,
&
#
# Elliptic solver parameters
&PARM02
cg2dMaxIters=1000,
cg2dTargetResidual=1.E-13,
&
#
# Time stepping parameters
&PARM03
nIter0 = 4269600,
nTimeSteps = 4,
deltaTtracer= 43200.0,
deltaTClock = 43200.0,
pChkptFreq=3110400000.,
chkptFreq= 3110400000.,
dumpFreq= 31104000.,
taveFreq= 31104000.,
monitorFreq= 1.,
periodicExternalForcing=.TRUE.,
externForcingPeriod=2592000.,
externForcingCycle=31104000.,
&
#
# Gridding parameters
&PARM04
usingSphericalPolarGrid=.TRUE.,
delR= 50., 70., 100., 140., 190.,
  240., 290., 340., 390., 440.,
  490., 540., 590., 640., 690.,
ygOrigin=-90.,
dxSpacing=2.8125,
dySpacing=2.8125,
&
#
# Input datasets
&PARM05
bathyFile= 'depth_g77.bin',
&
```

This file specifies the main parameters for the experiment.

- Lines 18-19,
\[ n_{\text{Iter}0} = 4269600, \]
\[ n_{\text{TimeSteps}} = 4, \]
\n\texttt{nIter0} and \texttt{nTimeSteps} control the start time and the length of the run (in timesteps). Given at \texttt{nIter0} is non-zero, the model requires appropriate pickup files to be present in the run directory. For testing purposes, the model has been prescribed to run for 4 timesteps; for a longer run, increase \texttt{nTimeSteps}.

- Line 20,
\[ \texttt{deltaTtracer} = 43200.0, \]
\nde\texttt{taTtracer} is the tracer timestep in seconds, in this case, 12 hours (43200 seconds = 12 hours). Note that \texttt{deltaTtracer} must be specified in \texttt{input_tutorial/data} as well as specified in \texttt{deltaToffline} in \texttt{input_tutorial/data.off}.

- Line 21,
\[ \texttt{deltaTClock} = 43200.0, \]
\nWhen using the MITgcm in offline mode, \texttt{deltaTClock} (an internal model counter) should be made equal to the value assigned to \texttt{deltaTtracer}.

- Line 27,
\[ \texttt{periodicExternalForcing} = \texttt{.TRUE.}, \]
\n\texttt{periodicExternalForcing} is a flag telling the model whether to cyclically re-use forcing data where there is external forcing (see Section 4.13.5 below). Where there is no external forcing, as here, but where there is to be cyclic re-use of the offline flow and mixing fields, \texttt{periodicExternalForcing} must be assigned the value \texttt{.TRUE.}.

- Line 28,
\[ \texttt{externForcingPeriod} = 2592000.0, \]
\n\texttt{externForcingPeriod} specifies the period of the external forcing data in seconds. In the absence of external forcing, as in this example, it must be made equal to the value of \texttt{externForcingPeriod} in \texttt{input_tutorial/data.off}, in this case, monthly (2592000 seconds = 1 month).

- Line 29,
\[ \texttt{externForcingCycle} = 31104000.0, \]
\n\texttt{externForcingCycle} specifies the duration of the external forcing data cycle in seconds. In the absence of external forcing, as in this example, it must be made equal to the value of \texttt{externForcingCycle} in \texttt{input_tutorial/data.off}, in this case, the cycle is one year (31104000 seconds = 1 year).

- Line 34,
\[ \texttt{usingSphericalPolarGrid} = \texttt{.TRUE.}, \]
\nThis line requests that the simulation be performed in a spherical polar coordinate system. It affects the interpretation of grid input parameters and causes the grid generation routines to initialize an internal grid based on spherical polar geometry.

- Lines 35-37,
\[ \texttt{delR} = 50., 70., 100., 140., 190., \]
\[ 240., 290., 340., 390., 440., \]
\[ 490., 540., 590., 640., 690., \]
This line sets the vertical grid spacing between each \( z \)-coordinate line in the discrete grid. Here the total model depth is 5200 m.

- Line 38,

\[ \text{ygOrigin}=-90.0, \]

This line sets the southern boundary of the modeled domain to -90° latitude N (= 90° S). This value affects both the generation of the locally orthogonal grid that the model uses internally and affects the initialization of the Coriolis force. Note: it is not required to set a longitude boundary, since the absolute longitude does not alter the kernel equation discretization.

- Line 39,

\[ \text{dxSpacing}=2.8125, \]

This line sets the horizontal grid spacing between each \( y \)-coordinate line in the discrete grid to 2.8125° in longitude.

- Line 40,

\[ \text{dySpacing}=2.8125, \]

This line sets the vertical grid spacing between each \( x \)-coordinate line in the discrete grid to 2.8125° in latitude.

- Line 45,

\[ \text{bathyFile='depth_g77.bin'}, \]

This line specifies the name of the file from which the domain bathymetry is read. This file contains a 2-D \((x, y)\) map of (assumed 64-bit) binary numbers giving the depth of the model at each grid cell, ordered with the \( x \) coordinate varying fastest. The points are ordered from low coordinate to high coordinate for both axes. The units and orientation of the depths in this file are the same as used in the MITgcm code. In this experiment, a depth of 0 m indicates land.

### 4.13.3.2 File input_tutorial/data.off

Listing 4.36: verification/tutorial_cfc_offline/input_tutorial/data.off

```plaintext
&OFFLINE_PARM01
UvelFile= 'input_off/uVeltave',
VvelFile= 'input_off/vVeltave',
WvelFile= 'input_off/wVeltave',
GMwxFile= 'input_off/GM_Kwx-T',
GMwyFile= 'input_off/GM_Kwy-T',
GMwzFile= 'input_off/GM_Kwz-T',
ConvFile= 'input_off/Convtave',
&

&OFFLINE_PARM02
offlineIter0=4248000,
deltaToffline=43200.,
offlineForcingPeriod=2592000.,
offlineForcingCycle=31104000.,
&
```

## 4.13. Offline Experiments
input_tutorial/data.off provides the MITgcm offline package with package specific parameters. Specifically, it contains the location (relative to the run directory) and prefix of files describing the flow field \( (U\text{velFile}, V\text{velFile}, W\text{velFile}) \) and the corresponding convective mixing coefficients \( (\text{ConvFile}) \) which together prescribe the 3-D, time varying dynamic system within which the offline model will advect the tracer.

- Lines 2-4, 8
  
  \[
  \begin{align*}
  U\text{velFile} &= '\cdots/input/input\_off/u\text{Veltave}', \\
  V\text{velFile} &= '\cdots/input/input\_off/v\text{Veltave}', \\
  W\text{velFile} &= '\cdots/input/input\_off/w\text{Veltave}', \\
  \text{ConvFile} &= '\cdots/input/input\_off/\text{Convtave}'
  \end{align*}
  \]

In the example the offline data is located in the sub-directory verification/tutorial_cfc_offline/input/input_off. In this directory are fields describing the velocity and convective mixing histories of a prior forward integration of the MITgcm, required for the offline package. Based on the values of \( \Delta t\_\text{offline} \), \( \text{offlineForcingPeriod} \) and \( \text{offlineForcingCycle} \) specified in verification/tutorial_cfc_offline/input/input_off, since \( \text{offlineForcingCycle} \) corresponds to twelve forcing periods \( \text{offlineForcingPeriod} \) and since \( \text{offlineIter}0 \) is zero, there needs to be twelve \( u\text{Veltave} \), twelve \( v\text{Veltave} \), twelve \( w\text{Veltave} \) and twelve \( \text{Convtave} \) files each having a 10 digit sequence identifier between 0000000001 to 0000000012, that is, a total of 48 files.

- Line 12,
  
  \[\text{offlineIter}0 = 4248000,\]

  \( \text{offlineIter}0 \), here specified to be 4248000 timesteps, corresponds to the timestep at which the tracer model is initialized. Note that \( \text{offlineIter}0 \) and \( n\text{Iter}0 \) (set in input_tutorial/data) need not be the same.

- Line 13,
  
  \[\Delta t\_\text{offline} = 43200.,\]

  \( \Delta t\_\text{offline} \) sets the timestep associated with the offline model data in seconds, here 12 hours (43200 seconds = 12 hours).

- Line 14,
  
  \[\text{offlineForcingPeriod} = 43200.,\]

  \( \text{offlineForcingPeriod} \) sets the forcing period associated with the offline model data in seconds.

- Line 15,
  
  \[\text{offlineForcingCycle} = 518400.,\]

  \( \text{offlineForcingCycle} \) sets the forcing cycle length associated with the offline model data in seconds. In this example the offline forcing cycle is 6 days, or twelve offline forcing periods. Together \( \Delta t\_\text{offline}, \text{offlineForcingPeriod} \) and \( \text{offlineForcingCycle} \) determine the value of the ten digit sequencing tag the model expects files in input_tutorial/data.off to have.
4.13.3.3 File input_tutorial/data.pkg

This file specifies which MITgcm packages are to be used.

- Line 4,

```
usePTRACERS=.TRUE.,
```

usePTRACERS is a flag invoking pkg/ptracers which is responsible for the advection of the tracer within the model.

4.13.3.4 File input_tutorial/data.ptracers

This file provides the MITgcm ptracers package with package specific parameters, prescribing the nature of the tracer/tracers as well as the variables associated with their advection.

- Line 2,

```
PTRACERS_numInUse=2,
```

PTRACERS_numInUse tells the model how many separate tracers are to be advected, in this case 2. Note: The value of PTRACERS_numInUse must agree with the value specified in code/PTRACERS_SIZE.h (see below).
PTRACERS_Iter0= 4248000,

PTRACERS_Iter0 specifies the iteration at which the tracer is to be introduced.

- Lines 6 and 13,

PTRACERS_advScheme(1)=77,

PTRACERS_advScheme(n) identifies which advection scheme will be used for tracer n, where n is the number of the tracer up to PTRACERS_numInUse. See Section 2.17 to identify the numerical codes used to specify different advection schemes (e.g. centered 2nd order, 3rd order upwind) as well as details of each.

- Lines 7 and 14,

PTRACERS_diffKh(1)=1.E3,

PTRACERS_diffKh(n) is the horizontal diffusion coefficient for tracer n, where n is the number of the tracer up to PTRACERS_numInUse.

- Lines 8 and 15,

PTRACERS_diffKr(1)=5.E-5,

PTRACERS_diffKr(n) is the vertical diffusion coefficient for tracer n, where n is the number of the tracer up to PTRACERS_numInUse.

- Lines 11 and 18,

PTRACERS_initialFile(1)=’ ’,

PTRACERS_initialFile(n) identifies the initial tracer field to be associated with tracer n, where n is the number of the tracer up to PTRACERS_numInUse. Note that no initial file is specified here.

Note input_tutorial/data.ptracers requires a set of entries for each tracer.

4.13.3.5 File input_tutorial/eedata

This file uses standard default values and does not contain customizations for this experiment.

4.13.3.6 File code/packages.conf

Listing 4.39: verification/tutorial_cfc_offline/code/packages.conf

```plaintext
#-- list of packages (or group of packages) to compile for this experiment:
gfd
-mom_common
-mom_fluxform
-mom_vecinv
gredi
offline
ptracers
gchem
cfc
timeave
```

This file is used to invoke the model components required for a particular implementation of the MITgcm.
4.13.7 File code/PTRACERS_SIZE.h

Listing 4.40: verification/tutorial_cfc_offline/code/PTRACERS_SIZE.h

```c
#ifdef ALLOW_PTRACERS

C !ROUTINE: PTRACERS_SIZE.h
C !INTERFACE:
C #include PTRACERS_SIZE.h

C !DESCRIPTION:
C Contains passive tracer array size (number of tracers).

C PTRACERS_num defines how many passive tracers are allocated/exist.
C and is set here (default 1)

C Number of tracers
INTEGER PTRACERS_num
PARAMETER(PTRACERS_num = 2 )

#endif /* ALLOW_PTRACERS */
```

- Line 16,

```
PARAMETER(PTRACERS_num = 2 )
```

This line sets the parameters PTRACERS_num (the number of tracers to be integrated) to 2 (in agreement with input_tutorial/data.ptracers).

4.13.8 File code/SIZE.h

Listing 4.41: verification/tutorial_cfc_offline/code/SIZE.h

```c
C !ROUTINE: SIZE.h
C !INTERFACE:
C include SIZE.h
C !DESCRIPTION:
C *==========================================================*
C | SIZE.h Declare size of underlying computational grid.
C *==========================================================*
C | The design here supports a three-dimensional model grid
C | with indices I,J and K. The three-dimensional domain
C | is comprised of nPx*nSx blocks (or tiles) of size sNx
C | along the first (left-most index) axis, nPy*nSy blocks
C | of size sWy along the second axis and one block of size
C | Nr along the vertical (third) axis.
C | Blocks/tiles have overlap regions of size OLx and OLy
C | along the dimensions that are subdivided.
C *==================================================================*
```

(continues on next page)
Voodoo numbers controlling data layout:

- `sNx`: Number of X points in tile.
- `sNy`: Number of Y points in tile.
- `OLx`: Tile overlap extent in X.
- `OLy`: Tile overlap extent in Y.
- `nSx`: Number of tiles per process in X.
- `nSy`: Number of tiles per process in Y.
- `nPx`: Number of processes to use in X.
- `nPy`: Number of processes to use in Y.
- `Nx`: Number of points in X for the full domain.
- `Ny`: Number of points in Y for the full domain.
- `Nr`: Number of points in vertical direction.

```c
INTEGER sNx
INTEGER sNy
INTEGER OLx
INTEGER OLy
INTEGER nSx
INTEGER nSy
INTEGER nPx
INTEGER nPy
INTEGER Nx
INTEGER Ny
INTEGER Nr
PARAMETER ( sNx = 64,
              sNy = 32,
              OLx = 4,
              OLy = 4,
              nSx = 2,
              nSy = 2,
              nPx = 1,
              nPy = 1,
              Nx = sNx*nSx*nPx,
              Ny = sNy*nSy*nPy,
              Nr = 15)
```

- `MAX_OLX`: Set to the maximum overlap region size of any array that will be exchanged. Controls the sizing of exchange routine buffers.

```c
INTEGER MAX_OLX
INTEGER MAX_OLY
PARAMETER ( MAX_OLX = OLx,
             MAX_OLY = OLy )
```

Several lines are customized in this file for the current experiment:
• Line 45,
\[ sNx=64, \]
this line sets the lateral domain extent in grid points for the axis aligned with the \( x \)-coordinate.

• Line 46,
\[ sNy=64, \]
this line sets the lateral domain extent in grid points for the axis aligned with the \( y \)-coordinate.

• Line 55,
\[ Nr=15, \]
this line sets the vertical domain extent in grid points.

### 4.13.4 Running the Experiment

In your run directory, as per usual, a copy of all files from the input directory (here, input_tutorial/) are required. In addition, you will also need to copy .data and .meta files from directory input/input_off.

### 4.13.5 A more complicated example

The previous example demonstrated simple advection of a passive tracer using the offline form of the MITgcm. Now we present a more complicated example in which the model is used to explore contamination of the global ocean through surface exposure to CFCs during the last century. In invoking packages pkg/gchem, pkg/gmredi and pkg/cfc it provides a starting point and template for more complicated offline modeling, involving as it does surface forcing through wind and ice fields, more sophisticated mixing, and a time-varying forcing function.

The run configuration for this experiment resides under the directory verification/tutorial_cfc_offline/input/ (the code configuration is the same as in the first example, so the same model executable can be used, i.e., no need to re-compile). The files

• verification/tutorial_cfc_offline/input/data
• verification/tutorial_cfc_offline/input/data.off
• verification/tutorial_cfc_offline/input/data.pkg
• verification/tutorial_cfc_offline/input/data.ptracers
• verification/tutorial_cfc_offline/input/data.gmredi
• verification/tutorial_cfc_offline/input/data.gchem
• verification/tutorial_cfc_offline/input/data.cfc
• verification/tutorial_cfc_offline/input/eedata

contain all the parameter settings required.
4.13.5.1 File input/data

Listing 4.42: verification/tutorial_cfc_offline/input/data

```plaintext
# ====================
# | Model parameters |
# ====================
#
# Continuous equation parameters
&PARM01
tRef=15*20.,
sRef=15*35.,
viscA4=0.,
viscAh=5.E5,
diffKhT=0.E3,
diffKhS=0.E3,
viscAr=1.E-3,
diffKrT=5.E-5,
diffKrS=5.E-5,
gravity=9.81,
rhoConst=1035.,
rigidLid=.FALSE.,
imPLICITFreeSurface=.TRUE.,
eosType='POLY3',
imPLICITDiffusion=.TRUE.,
imPLICITViscosity=.TRUE.,
ivdc_kappa=100.,
multiDimAdvection = .FALSE.
#allowFreezing=.TRUE.,
useCDscheme=.FALSE.,
&
#
# Elliptic solver parameters
&PARM02
cg2dMaxIters=1000,
cg2dTargetResidual=1.E-13,
&
#
# Time stepping parameters
&PARM03
nIter0 = 4269600,
nTimeSteps = 4,
# 100 years starting from a spinup of 5900 years:
#startTime = 1.835136E+11,
#endTime = 1.866240E+11,
deltaTmom = 900.0,
#tauCD = 321428.,
deltaTtracer= 43200.0,
deltaTClock = 43200.0,
abEps = 0.1,
#cAdjFreq = -1,
pChkptFreq=3110400000.,
#chkptFreq=3110400000.,
```

(continues on next page)
50 dumpFreq= 31104000.,
51 taveFreq= 31104000.,
52 #monitorFreq= 4853865600.,
53 monitorFreq= 2592000.,
54 #tauThetaClimRelax = 5184000.0,
55 #tauSaltClimRelax = 7776000.0,
56 periodicExternalForcing=.TRUE.,
57 externForcingPeriod=2592000.,
58 externForcingCycle=31104000.,
59 &
60 # Gridding parameters
61 &PARM04
62 usingSphericalPolarGrid=.TRUE.,
63 delR= 50., 70., 100., 140., 190.,
64 240., 290., 340., 390., 440.,
65 490., 540., 590., 640., 690.,
66 ygOrigin=-90.,
67 dxSpacing=2.8125,
68 dySpacing=2.8125,
69 &
70 # Input datasets
71 &PARM05
72 bathyFile= 'depth_g77.bin',
73 #hydrogThetaFile= 'lev_clim_temp.bin',
74 #hydrogSaltFile= 'lev_clim_salt.bin',
75 #zonalWindFile= 'tren_taux.bin',
76 #meridWindFile= 'tren_tauy.bin',
77 #thetaClimFile= 'lev_monthly_temp.bin',
78 #saltClimFile= 'lev_monthly_salt.bin',
79 #surfQnetFile= 'shi_qnet.bin',
80 #EmPmRFile= 'shi_empmr_year.bin',
81 the_run_name= 'Testing CFC and OFFLINE code',
82 &

A single line must be added (under PARM01, line 21) from the previous example

&PARM01
implicitDiffusion=.TRUE.,
&

When pkg/gmredi is used, the flag implicitDiffusion must be assigned the value .TRUE.

In this example the starting timestep nIter0 is set to 4269600 requiring model access to pickup files with the suffix 0004269600. The model will run for 4 timesteps (nTimeSteps = 4). In this case the frequencies with which permanent and rolling checkpoints (pChkptFreq and chkptFreq) have been set is sufficiently long to ensure that only one from the last timestep will be written. This is also true of the values that have been assigned to the frequency with which dumps are written (dumpFreq) and time averaging (taveFreq) is performed. However, since the model always dumps the state of the model when it stops without error, a dump will be written with suffix 0004269604 upon completion.

4.13. Offline Experiments
### 4.13.5.2 File input/data.off

Listing 4.43: verification/tutorial_cfc_offline/input/data.off

```plaintext
&OFFLINE_PARM01
    UvelFile= '../input/input_off/uVeltave',
    VvelFile= '../input/input_off/vVeltave',
    WvelFile= '../input/input_off/wVeltave',
    GMwxFile= '../input/input_off/GM_Kwx-T',
    GMwyFile= '../input/input_off/GM_Kwy-T',
    GMwzFile= '../input/input_off/GM_Kwz-T',
    ConvFile= '../input/input_off/Convtave',
    SaltFile= '../input/input_off/Stave',
    ThetFile= '../input/input_off/Ttave',
    # SFluxFile= '../input/input_off/sFluxtave',
    # HFluxFile= '

&OFFLINE_PARM02
    offlineIter0=4248000,
    deltaToffline=43200.,
    offlineForcingPeriod=2592000.,
    offlineForcingCycle=31104000.,
    &
```

This file specifies the prefixes and locations of additional input files required to run the offline model. Note that directory `input/input_off` contains only as many offline files as are required to successfully run for 4 timesteps. Where the GMREDI scheme was used in the forward run, as here, package GMREDI must again be invoked when running offline. In this example, tracer is specified as having been introduced with a non-zero starttime, at timestep 4248000.

### 4.13.5.3 File input/data.pkg

Chapter 4. MITgcm Tutorial Example Experiments
This file specifies which MITgcm packages are to be used. It now invokes additional packages pkg/gmredi and pkg/gchem.

### 4.13.5.4 File input/data.ptracers

This file specifies parameters associated with the CFC11 and CFC12 tracer fields advected in this example.

- **Line 3,**
  ```plaintext
  PTRACERS_Iter0= 4248000,
  ```

In this example the tracers were introduced at iteration 4248000.
• Lines 12 and 21,

\[
\text{PTRACERS\_diffK}\text{h}(n)=0.E3,
\]

Since package GMREDI is being used, regular horizontal diffusion is set to zero.

• Lines 14-15 and 23-24,

\[
\text{PTRACERS\_useGMRedi}(n)=.TRUE.,
\text{PTRACERS\_useKPP}(n)=.FALSE.,
\]

Setting flag PTRACERS\_useGMRedi(n) to .TRUE. identifies that /pkg/gmredi is to be used. Setting flag PTRACERS\_useKPP(n) to .FALSE. explicitly turns off KPP mixing.

• Lines 16 and 25,

\[
\text{PTRACERS\_initialFile}(n)=''
\]

Since this is a ‘pickup’ run the initial tracer files PTRACERS\_initialFile are not needed. The model will obtain the tracer state from pickup_ptracers.0004269600.data

### 4.13.5.5 File input/data.gchem

Listing 4.46: verification/tutorial\_cfc\_offline/input/data.gchem

```plaintext
&GCHEM\_PARM01
useCFC=.TRUE.,
nsubtime=1,
&
```

This file specifies the parameters used in /pkg/gchem.

### 4.13.5.6 File input/data.gmredi

Listing 4.47: verification/tutorial\_cfc\_offline/input/data.gmredi

```plaintext
# from MOM
# GM\_background\_K: isopycnal diffusion coefficient
# GM\_maxSlope: max slope of isopycnals
# GM\_Scrit: transition for scaling diffusion coefficient
# GM\_Sd: half width scaling for diffusion coefficient
# real background diff: horizontal diffusion

# ifdef GM\_VISBECK\_VARIABLE\_K, include following in GM\_PARM01
# GM\_Visbeck\_alpha = 0.,
# GM\_Visbeck\_length = 2.e+5,
# GM\_Visbeck\_depth = 1.e+3,
# GM\_Visbeck\_maxval\_K= 2.5e+3,

&GM\_PARM01
GM\_background\_K = 1.e+3,
GM\_taper\_scheme = 'gkw91',
GM\_maxSlope = 1.e-2,
```

(continues on next page)
This file specifies parameters required for /pkg/gmredi.

4.13.5.7 File input/cfc1112.atm

This is a ASCII data file containing the CFC source functions over the northern and southern hemispheres annually from 1931 through 1998.

4.13.5.8 Running the Experiment

The model is run as before.

4.14 Rotating Tank

((in directory: verification/tutorial_rotating_tank/)

This example configuration demonstrates using the MITgcm to simulate a laboratory demonstration using a differentially heated rotating annulus of water. The simulation is configured for a laboratory scale on a $3^\circ \times 1$ cm cylindrical grid with 29 vertical levels of 0.5 cm each. This is a typical laboratory setup for illustrating principles of GFD, as well as for a laboratory data assimilation project.

example illustration from GFD lab here

4.14.1 Equations Solved

4.14.2 Discrete Numerical Configuration

The domain is discretized with a uniform cylindrical grid spacing in the horizontal set to $\Delta a = 1$ cm and $\Delta \phi = 3^\circ$, so that there are 120 grid cells in the azimuthal direction and 31 grid cells in the radial, representing a tank 62 cm in diameter. The bathymetry file sets the depth=0 in the nine lowest radial rows to represent the central of the annulus. Vertically the model is configured with 29 layers of uniform 0.5 cm thickness.

something about heat flux
4.14.3 Code Configuration

The model configuration for this experiment resides under the directory verification/tutorial_rotating_tank/. The experiment files

- verification/tutorial_rotating_tank/input/data
- verification/tutorial_rotating_tank/input/data.pkg
- verification/tutorial_rotating_tank/input/eedata
- verification/tutorial_rotating_tank/input/bathyPolR.bin
- verification/tutorial_rotating_tank/input/thetaPolR.bin
- verification/tutorial_rotating_tank/code/CPP_OPTIONS.h
- verification/tutorial_rotating_tank/code/SIZE.h

contain the code customizations and parameter settings for this experiments. Below we describe the customizations to these files associated with this experiment.

### 4.14.3.1 File input/data

#### Listing 4.48: verification/tutorial_rotating_tank/input/data

```
# ================
# | Model parameters |
# ================

# Continuous equation parameters
&PARM01
  tRef=29*20.0,
  sRef=29*35.0,
  viscAh=5.0E-6,
  viscAz=5.0E-6,
  no_slip_sides=.FALSE.,
  no_slip_bottom=.FALSE.,
  diffKhT=2.5E-6,
  diffKzT=2.5E-6,
  diffKhS=1.0E-6,
  diffKzS=1.0E-6,
  f0=0.5,
  eosType='LINEAR',
  sBeta =0.,
  gravity=9.81,
  rhoConst=1000.0,
  rhoNil=1000.0,
  #heatCapacity_Cp=3900.0,
  rigidLid=.TRUE.,
  implicitFreeSurface=.FALSE.,
  nonHydrostatic=.TRUE.,
  readBinaryPrec=32,
&

# Elliptic solver parameters
```

(continues on next page)
This file specifies the main parameters for the experiment. The parameters that are significant for this configuration are

- Lines 9-10,

\[
\text{viscAb}=5.0\times10^{-6},
\text{viscAz}=5.0\times10^{-6},
\]

These lines set the Laplacian friction coefficient in the horizontal and vertical, respectively. Note that they are several orders of magnitude smaller than the other examples due to the small scale of this example.

- Lines 13-16,

\[
\text{diffKhT}=2.5\times10^{-6},
\text{diffKzT}=2.5\times10^{-6},
\text{diffKhS}=1.0\times10^{-6},
\text{diffKzS}=1.0\times10^{-6},
\]

4.14. Rotating Tank
These lines set horizontal and vertical diffusion coefficients for temperature and salinity. Similar to the friction coefficients, the values are a couple of orders of magnitude less than most configurations.

- Line 17,

\[ f_0 = 0.5, \]

this line sets the Coriolis term, and represents a tank spinning at about 2.4 rpm.

- Lines 24 and 25,

\[
\text{rigidLid} = \text{.TRUE.}, \\
\text{implicitFreeSurface} = \text{.FALSE.},
\]

These lines activate the rigid lid formulation of the surface pressure inverter and suppress the implicit free surface form of the pressure inverter.

- Line 40,

\[ nIter = 0, \]

This line indicates that the experiment should start from \( t = 0 \) and implicitly suppresses searching for checkpoint files associated with restarting an numerical integration from a previously saved state. Instead, the file thetaPolR.bin will be loaded to initialized the temperature fields as indicated below, and other variables will be initialized to their defaults.

- Line 43,

\[ \text{deltaT} = 0.1, \]

This line sets the integration timestep to 0.1 s. This is an unusually small value among the examples due to the small physical scale of the experiment. Using the ensemble Kalman filter to produce input fields can necessitate even shorter timesteps.

- Line 54,

\[ \text{usingCylindricalGrid} = \text{.TRUE.}, \]

This line requests that the simulation be performed in a cylindrical coordinate system.

- Line 55,

\[ \text{dXspacing} = 3, \]

This line sets the azimuthal grid spacing between each \( x \)-coordinate line in the discrete grid. The syntax indicates that the discrete grid should be comprised of 120 grid lines each separated by 3°.

- Line 56,

\[ \text{dYspacing} = 0.01, \]

This line sets the radial cylindrical grid spacing between each \( a \)-coordinate line in the discrete grid to 1 cm.

- Line 57,

\[ \text{delZ} = 29 \times 0.005, \]

This line sets the vertical grid spacing between each of 29 \( z \)-coordinate lines in the discrete grid to 0.005 m (= 5 mm).
• Line 64,

```python
bathyFile='bathyPolR.bin',
```

This line specifies the name of the file from which the domain ‘bathymetry’ (i.e., tank depth) is read. This file is a 2-D \((a, \phi)\) map of depths. This file is assumed to contain 64-bit binary numbers giving the depth of the model at each grid cell, ordered with the \(\phi\) coordinate varying fastest. The points are ordered from low coordinate to high coordinate for both axes. The units and orientation of the depths in this file are the same as used in the MITgcm code. In this experiment, a depth of 0 m indicates an area outside of the tank and a depth of -0.145 m indicates the tank itself.

• Line 63,

```python
hydrogThetaFile='thetaPol.bin',
```

This line specifies the name of the file from which the initial values of temperature are read. This file is a 3-D \((x, y, z)\) map and is enumerated and formatted in the same manner as the bathymetry file.

• Lines 65 and 66

```python
tCylIn = 0.,
tCylOut = 20.,
```

These line specify the temperatures in degrees Celsius of the interior and exterior walls of the tank – typically taken to be icewater on the inside and room temperature on the outside.

Other lines in the file `verification/tutorial_rotating_tank/input/data` are standard values that are described in Section 3.8.

### 4.14.3.2 File - input/data.pkg

This file uses standard default values and does not contain customizations for this experiment.

### 4.14.3.3 File - input/eedata

This file uses standard default values and does not contain customizations for this experiment.

### 4.14.3.4 File input/thetaPolR.bin

This file specifies a 3-D \((x, y, z)\) map of initial values of \(\theta\) in degrees Celsius. This particular experiment is set to random values around 20 °C to provide initial perturbations.

### 4.14.3.5 File input/bathyPolR.bin

This file specifies a 2-D \((x, y)\) map of depth values. For this experiment values are either 0 m or -\(\text{delZ}\) m, corresponding respectively to outside or inside of the tank. The file contains a raw binary stream of data that is enumerated in the same way as standard MITgcm 2-D, horizontal arrays.
4.14.3.6 File code/SIZE.h

Listing 4.49: verification/tutorial_rotating_tank/code/SIZE.h

```
CBOP
C    !ROUTINE: SIZE.h
C    !INTERFACE:
C    include SIZE.h
C    !DESCRIPTION: 
C    *==========================================================*
C     | SIZE.h Declare size of underlying computational grid.
C    *==========================================================*
C    | The design here supports a three-dimensional model grid
C    | with indices I,J and K. The three-dimensional domain
C    | is comprised of nPx*nSx blocks (or tiles) of size sNx
C    | along the first (left-most index) axis, nPy*nSy blocks
C    | of size sNy along the second axis and one block of size
C    | Nr along the vertical (third) axis.
C    | Blocks/tiles have overlap regions of size OLx and OLy
C    | along the dimensions that are subdivided.
C    *==========================================================*
C    \ev

C Voodoo numbers controlling data layout:
C sNx :: Number of X points in tile.
C sNy :: Number of Y points in tile.
C OLx :: Tile overlap extent in X.
C OLy :: Tile overlap extent in Y.
C nSx :: Number of tiles per process in X.
C nSy :: Number of tiles per process in Y.
C nPx :: Number of processes to use in X.
C nPy :: Number of processes to use in Y.
C Nx :: Number of points in X for the full domain.
C Ny :: Number of points in Y for the full domain.
C Nr :: Number of points in vertical direction.
CEOP

    INTEGER sNx
    INTEGER sNy
    INTEGER OLx
    INTEGER OLy
    INTEGER nSx
    INTEGER nSy
    INTEGER nPx
    INTEGER nPy
    INTEGER Nx
    INTEGER Ny
    INTEGER Nr
    PARAMETER (
        & sNx = 30,
        & sNy = 23,
        & OLx = 3,
        & OLy = 3,
        & nSx = 4,
    )
```
Two lines are customized in this file for the current experiment

- Line 45,

```fortran
sNx=120,
```

this line sets the lateral domain extent in grid points for the axis aligned with the $x$-coordinate.

- Line 46,

```fortran
sNy=31,
```

this line sets the lateral domain extent in grid points for the axis aligned with the $y$-coordinate.

### 4.14.3.7 File code/CPP_OPTIONS.h

This file uses standard default values and does not contain customizations for this experiment.

### 4.15 Additional Example Experiments: Forward Model Setups

For many experiments, additional information is provided in a README file located in the respective experiment’s subdirectory.

1. 1D_ocean_ice_column - Oceanic column with seaice on top.
2. adjustment.128x64x1 - Barotropic adjustment problem on latitude-longitude grid with 128x64 grid points (2.8° resolution).
3. adjustment.cs-32x32x1 - Barotropic adjustment problem on cube sphere grid with 32x32 points per face (roughly 2.8° resolution). Also contains a non-linear free-surface adjustment version (input.nlfs).
4. advect_cs - 2-D passive advection test on cube sphere grid (32x32 grid points per face, roughly 2.8° resolution).
5. advect_xy - 2-D (horizontal plane) passive advection test on Cartesian grid. Also contains an additional setup using Adams-Bashforth 3 (input.ab3_c4).
6. advect_xz - 2-D (vertical plane) passive advection test on Cartesian grid. Also contains an additional setup using non-linear free-surface with divergent barotropic flow and implicit vertical advection (input.nlfs), and a setup using piecewise quartic (“mono” and “weno” limiter) advection schemes (input.pqm).
7. aim.5l_Equatorial_Channel - 5-level intermediate atmospheric physics, 3-D equatorial channel configuration.
8. aim.5l_LatLon - 5-level intermediate atmospheric physics, global configuration, on latitude-longitude grid with 128x64x5 grid points (2.8° resolution).
9. aim.5l_cs - 5-level intermediate atmospheric physics, global configuration on cube sphere grid (32x32 grid points per face, roughly 2.8° resolution). Also contains an additional setup with a slab-ocean and thermodynamic sea ice (input.thSI).
10. cfc_example - Global ocean with online computation and advection of CFC11 and CFC12.
11. cheapAML_box - Example using cheap atmospheric mixed layer (cheapaml) package.
12. cpl_aim+ocn - Coupled ocean-atmosphere realistic configuration on cubed-sphere cs32 horizontal grid, using intermediate atmospheric physics (pkg/aim_v23) thermodynamic seaice (pkg/thsice) and land packages.
13. deep_anelastic - Convection simulation on a giant planet: relaxes both the Boussinesq approximation (anelastic) and the thin atmosphere approximation (deep atmosphere).
14. dome - Idealized 3-D test of a density-driven bottom current (Denmark Overflow Mixing and Entrainment experiment).
15. exp2 - Old version of the global ocean experiment (no GM, no partial-cells). Also contains an additional setup with rigid lid (input.rigidLid).
16. exp4 - Flow over a Gaussian bump in open-water or channel with open boundaries. Also contains an additional setup using non-linear free-surface (input.nlfs), and a setup using Stevens (1990) [Ste90] boundary conditions (input.stevens).
17. fizhi-cs-32x32x40 - Global atmospheric simulation with realistic topography, 40 vertical levels, a cubed sphere grid and the full atmospheric physics package.
18. fizhi-cs-aqualev20 - Global atmospheric simulation on an aqua planet with full atmospheric physics. Run is perpetual March with an analytical SST distribution. This is the configuration used for the APE (Aqua Planet Experiment) participation experiment.
19. fizhi-gridalt-hs - Global atmospheric simulation Held-Suarez (1994) [HS94] forcing, with the physical forcing and the dynamical forcing running on different vertical grids.
20. flt_example - Example using float package.
21. front_relax - Relaxation of an 2-D (y − z) ocean thermal front (test of Gent and McWilliams scheme). Also contains additional setups:
   • using the boundary-value problem method (Ferrari et al. 2010 [FGNV10]) (input.bvp).
   • with mixed-layer eddy parameterization (Ferrari and McWilliams 2008 [FMCD08]) (input.mxl).
22. global_ocean.90x40x15 - Global ocean simulation at 4°x4° resolution. Similar to tutorial_global_oce_latlon, but using z* coordinates with quasi-non-hydrostatic and non-hydrostatic metric terms. This experiment illustrates the use of sbo package. Also contains additional setups:
   • using down-slope package (input.dwnslp)
   • using package ggl90 scheme (Gaspar et al. 1990 [GGL90]) with parameterized tidal and wind energy input into vertical mixing (input.idemix).
23. global_ocean.cs32x15 - Global ocean experiment on the cubed sphere grid. Also contains additional setups:
   • non-hydrostatic with biharmonic viscosity (input.viscA4)
   • using thermodynamic sea ice and bulk force (input.thsice)
   • using both thermodynamic (pkg/thsice) and dynamic (pkg/seaice) sea ice packages with exf package (input.icedyn)
• using thermodynamic and dynamic (pkg/seaice) sea ice with exf package package (input.seaice).

24. global_ocean_ebm - Global ocean experiment on a lat-lon grid coupled to a zonally averaged atmospheric energy balance model. Similar to global_ocean.90x40x15 experiment.

25. global_oce_biogeo_bling - Global ocean biogeochemistry simulation, based on Biogeochemistry Simulation but using package bling instead of the DIC package.

26. global_with_exf - Global ocean experiment (at 4°x4") on a lat-lon grid using the exf package with exf interpolation. Similar to tutorial_global_oce_latlon experiment. Also contains a secondary setup with yearly exf fields (input.yearly).

27. halfpipe_streamice - Example using package streamice.

28. hs94.128x64x5 - 3-D atmosphere dynamics on lat-lon grid, using Held and Suarez (1994) [HS94] forcing.

29. hs94.1x64x5 - Zonal averaged atmosphere dynamics using Held and Suarez (1994) [HS94] forcing.

30. hs94.cs-32x32x5 - 3-D atmosphere dynamics using Held and Suarez (1994) [HS94] forcing on the cubed sphere, similar to tutorial_held_suarez_cs experiment but using linear free-surface and only 5 levels. Also contains an additional setup with implicit internal gravity waves treatment and Adams-Bashforth 3 (input.impIGW).

31. ideal_2D_oce - Idealized 2-D global ocean simulation on an aqua planet.

32. internal_wave - Ocean internal wave forced by open boundary conditions. Also contains an additional setup using pkg/k110 (see Section 8.4.5, Klymak and Legg 2010 [KL10]) (input.k110).

33. inverted_barometer - Simple test of atmospheric pressure loading with radially symmetric Bessel-function geometry in a quadratic domain.

34. isomip - ISOMIP-like setup (Ice Shelf Ocean Model Intercomparison Project experiment 0) including ice-shelf cavities (pkg/shelfice). Also contains additional setups:
   • with “htd” (Hellmer’s thermodynamics, Hellmer 1989 [HO89]) (input.htd).
   • using package icefront (input.icefront)
   • using package OBCS enabled to balance surface mass (freshwater and ice shelf mass flux) input through open boundaries (input.obcs).

35. lab_sea - Regional (2°x2°) Labrador Sea simulation on a lat-lon grid using pkg/seaice. Also contains additional setups:
   • using the simple “free-drift” assumption for sea ice (input.fd)
   • using aEVP dynamics (instead of LSR solver) and Hibler and Bryan (1987) [HB87] sea ice ocean stress (input.bb87)
   • using package salt_plume (input.salt_plume).

36. matrix_example - Test of experimental method to accelerate convergence towards equilibrium.

37. MLAdjust - Simple tests of different viscosity formulations in a zonally reentrant, flat-bottom channel. Contains additional setups; see verification/MLAdjust/README for a listing of different viscosity settings in these experiments:
   • input.A4FlxF
   • input.AhFlxF
   • input.AhVrDv
   • input.AhStTn
   • input.QGLeith
   • input.QGLthGM.

4.15. Additional Example Experiments: Forward Model Setups
38. natl_box - Eastern subtropical North Atlantic with KPP scheme. Contains additional setup with added tracers (pkg/tracers) using the package longstep to speed up integration time (input.longstep).

39. offline_exf_seaice - Sea ice on top of oceanic surface layer in an idealized channel. Forcing is computed by bulk-formulae (pkg/exf) with temperature relaxation to prescribed SST (i.e., no momentum time-stepping in ocean, so ocean is “offline”, not to be confused with pkg/offline). Also contains additional setups:
   - sea ice dynamics-only using JFNK solver and (pkg/seaice) advection (input.dyn_jfnk)
   - sea ice dynamics-only using LSR solver and (pkg/seaice) advection (input.dyn_lsr)
   - sea ice thermodynamics-only using (pkg/seaice) (input.thermo)
   - sea ice thermodynamics-only using (pkg/thsice) (input.thsice).

40. seaice_itd - Seaice example using ice thickness distribution (ITD); otherwise very similar to offline_exf_seaice. Also contains additional setups; see verification/seaice_itd/README for details of these setups:
   - input.thermo
   - input.lipscomb07.

41. seaice_obcs - Similar to lab_sea (input.salt_plume) experiment with only a fraction of the domain and open boundary conditions derived from lab_sea experiment. Also contains additional setups:
   - includes relaxation of seaice variables (input.seaiceSponge)
   - includes tidal velocity forcing (input.tides).

42. shelfice_2d_remesh - Simple experiment to test (pkg/shelfice) vertical remeshing code in 2-D idealized-geometry setup.

43. short_surf_wave - Short surface wave adjustment (non-hydrostatic) in homogeneous 2-D vertical section (x – z).

44. so_box_biogeo - Open boundary Southern Ocean box around Drake Passage, using same model parameters and forcing as experiment tutorial_global_oce_biogeo from which initial conditions and open boundary conditions have been extracted. Also contains additional setup using the SolveSAPHE algorithm (Munhoven 2013 [Mun13]) to determine oceanic pH (input.saphe).

45. solid-body.cs-32x32x1 - Solid body rotation test for cube sphere grid.

46. tutorial_deep_convection - Experiment as described in Section 4.8, also contains an additional setup using the Smagorinsky (1963) [Sma63] viscosity scheme (input.smag3d).

47. vermix - Simple test in a small domain (3 columns) for ocean vertical mixing schemes. The standard setup (input) uses the KPP scheme Large et al. (1994) [LMD94]. Also contains additional setups:
   - with double diffusion scheme from KPP (input.dd)
   - with package ggl90 scheme (Gaspar et al. 1990 [GGL90]) scheme (input.ggl90)
   - with Mellor and Yamada (1982) [MY82] level 2 (pkg/my82) scheme (input.my82)
   - with Paluszkiewicz and Romea (1997) [PR97] (pkg/opps) scheme (input.opps)
4.16 Additional Example Experiments: Adjoint Model Setups

Unless stated otherwise, the physical setup of the adjoint run is identical to the forward run, see description above. TAF adjoint setups require building with directory `code_ad` with input directory `input_ad`, whereas OpenAD requires directories `coad_oad` and `input_oad` respectively.

1. **1D_ocean_ice_column** - Based on standard forward experiment, TAF adjoint setup, uses package `ecco`.
2. **bottom_ctrl_5x5** - TAF adjoint test using the bottom topography as the control parameter, uses package `ecco`.
3. **global_ocean.90x40x15** - Based on standard forward experiment, TAF and OpenAD adjoint setups. Also contains additional TAF adjoint setups:
   - with bottom drag as a control (`input_ad.bottomdrag`)
   - with $\kappa_{GM}$ as a control (`input_ad.kapgm`)
   - with $\kappa_{Redi}$ as a control (`input_ad.kapredi`).
4. **global_ocean.cs32x15** - Based on standard forward experiment, TAF adjoint setup. Also contains additional TAF adjoint setups:
   - using thermodynamic-dynamic sea ice (`input_ad.seaice`).
   - same as above but without adjoint sea ice dynamics (`input_ad.seaice_dynmix`).
   - using thermodynamic sea ice from pkg/thsice (`input_ad.thsice`).
5. **global_ocean_ebm** - Based on standard forward experiment, TAF adjoint setup.
6. **global_with_exf** - Based on standard forward experiment, TAF adjoint setup.
7. **halfpipe_streamice** - Based on standard forward experiment, TAF and OpenAD adjoint setups.
8. **hs94.1x64x5** - Based on standard forward experiment, TAF and OpenAD adjoint setups.
9. **isomip** - Based on standard forward experiment, TAF and OpenAD adjoint setups. Also contains additional TAF adjoint setup with “htd” (Hellmer’s thermodynamics, Hellmer 1989 [HO89]) (`input_ad.htd`).
10. **lab_sea** - Based on standard forward experiment, TAF adjoint setup, uses package `ecco` and divided adjoint (DIVA). Also contains additional TAF adjoint setups:
    - without seaice dynamics (`input_ad.noseaicedyn`).
    - without seaice altogether (`input_ad.noseaice`).
11. **obcs_ctrl** - Adjoint test using open boundary conditions as control parameters, uses package `ecco`.
12. **offline_exf_seaice** - Based on standard forward experiment, TAF adjoint setup. Also contains additional TAF adjoint setup with sea ice thermodynamics-only using pkg/thsice (`input_ad.thsice`).
13. **OpenAD** - Simple adjoint experiment (used also to test OpenAD compiler), TAF and OpenAD adjoint setups. Also contains additional OpenAD adjoint setups:
    - using package ggl90 (`input_oad.ggl90`).
    - using package kpp (`input_oad.kpp`).
14. **tutorial_dic** - TAF adjoint setup of offline form of passive tracers coupled to the dissolved inorganic carbon biogeochemistry model (currently NOT documented as a tutorial experiment).
15. **tutorial_global_oce** - Based on forward experiment described in Section 4.10, TAF and OpenAD adjoint setups.
17. tutorial_tracer_adjsens - Based on adjoint experiment described in Section 4.12, contains an additional TAF setup using Second Order Moment (SOM) advection scheme (input_ad.som81).
The MITgcm is an open source project that relies on the participation of its users, and we welcome contributions. This chapter sets out how you can contribute to the MITgcm.

5.1 Bugs and feature requests

If you think you’ve found a bug, the first thing to check that you’re using the latest version of the model. If the bug is still in the latest version, then think about how you might fix it and file a ticket in the GitHub issue tracker. Please include as much detail as possible. At a minimum your ticket should include:

- what the bug does;
- the location of the bug: file name and line number(s); and
- any suggestions you have for how it might be fixed.

To request a new feature, or guidance on how to implement it yourself, please open a ticket with the following details:

- a clear explanation of what the feature will do; and
- a summary of the equations to be solved.

5.2 Using Git and Github

To contribute to the source code of the model you will need to fork the repository and place a pull request on GitHub. The two following sections describe this process in different levels of detail. If you are unfamiliar with git, you may wish to skip the quickstart guide and use the detailed instructions. All contributions to the source code are expected to conform with the Coding style guide. Contributions to the manual should follow the same procedure and conform with Section 5.6.

5.2.1 Quickstart Guide

1. Fork the project on GitHub (using the fork button).
2. Create a local clone (we strongly suggest keeping a separate repository for development work):
   ```
   % git clone https://github.com/«GITHUB_USERNAME»/MITgcm.git
   ```
3. Move into your local clone directory (cd MITgcm) and and set up a remote that points to the original:
   ```
   % git remote add upstream https://github.com/MITgcm/MITgcm.git
   ```
4. Make a new branch from upstream/master (name it something appropriate, such as ‘bugfix’ or ‘newfeature’ etc.) and make edits on this branch:

```
% git fetch upstream
% git checkout -b «YOUR_NEWBRANCH_NAME» upstream/master
```

5. When edits are done, do all git add’s and git commit’s. In the commit message, make a succinct (<70 char) summary of your changes. If you need more space to describe your changes, you can leave a blank line and type a longer description, or break your commit into multiple smaller commits. Reference any outstanding issues addressed using the syntax #«ISSUE_NUMBER».

6. Push the edited branch to the origin remote (i.e. your fork) on GitHub:

```
% git push -u origin «YOUR_NEWBRANCH_NAME»
```

7. On GitHub, go to your fork and hit the compare and pull request (PR) button, provide the requested information about your PR (in particular, a non-trivial change to the model requires a suggested addition to doc/tag-index) and wait for the MITgcm head developers to review your proposed changes. In general the MITgcm code reviewers try to respond to a new PR within a week. The reviewers may accept the PR as is, or may request edits and changes. Occasionally the review team will reject changes that are not sufficiently aligned with and do not fit with the code structure. The review team is always happy to discuss their decisions, but wants to avoid people investing extensive effort in code that has a fundamental design flaw. The current review team is Jean-Michel Campin, Ed Doddridge, Chris Hill, Oliver Jahn, and Jeff Scott.

If you want to update your code branch before submitting a PR (or any point in development), follow the recipe below. It will ensure that your GitHub repo stays up to date with the main repository. Note again that your edits should always be to your development branch, not the master branch.

```
% git checkout master
% git pull upstream master
% git push origin master
% git checkout «YOUR_NEWBRANCH_NAME»
% git merge master
```

If you prefer, you can rebase rather than merge in the final step above; just be careful regarding your rebase syntax!

### 5.2.2 Detailed guide for those less familiar with Git and GitHub

What is Git? Git is a version control software tool used to help coordinate work among the many MITgcm model contributors. Version control is a management system to track changes in code over time, not only facilitating ongoing changes to code, but also as a means to check differences and/or obtain code from any past time in the project history. Without such a tool, keeping track of bug fixes and new features submitted by the global network of MITgcm contributors would be virtually impossible. If you are familiar with the older form of version control used by the MITgcm (CVS), there are many similarities, but we now take advantage of the modern capabilities offered by Git.

Git itself is open source linux software (typically included with any new linux installation, check with your sys-admin if it seems to be missing) that is necessary for tracking changes in files, etc. through your local computer’s terminal session. All Git-related terminal commands are of the form `git «arguments»`. Important functions include syncing or updating your code library, adding files to a collection of files with edits, and commands to “finalize” these changes for sending back to the MITgcm maintainers. There are numerous other Git command-line tools to help along the way (see man pages via `man git`).

The most common git commands are:

- `git clone` download (clone) a repository to your local machine
- `git status` obtain information about the local git repository
• **git diff** highlight differences between the current version of a file and the version from the most recent commit
• **git add** stage a file, or changes to a file, so that they are ready for **git commit**
• **git commit** create a commit. A commit is a snapshot of the repository with an associated message that describes the changes.

What is GitHub then? GitHub is a website that has three major purposes: 1) Code Viewer: through your browser, you can view all source code and all changes to such over time; 2) “Pull Requests”: facilitates the process whereby code developers submit changes to the primary MITgcm maintainers; 3) the “Cloud”: GitHub functions as a cloud server to store different copies of the code. The utility of #1 is fairly obvious. For #2 and #3, without GitHub, one might envision making a big tarball of edited files and emailing the maintainers for inclusion in the main repository. Instead, GitHub effectively does something like this for you in a much more elegant way. Note unlike using (linux terminal command) git, GitHub commands are NOT typed in a terminal, but are typically invoked by hitting a button on the web interface, or clicking on a webpage link etc. To contribute edits to MITgcm, you need to obtain a github account. It’s free; do this first if you don’t have one already.

Before you start working with git, make sure you identify yourself. From your terminal, type:

```
% git config --global user.email «your_email@example.edu»
% git config --global user.name «'John Doe'»
```

(note the required quotes around your name). You should also personalize your profile associated with your GitHub account.

There are many online tutorials to using Git and GitHub (see for example https://akrabat.com/the-beginners-guide-to-contributing-to-a-github-project); here, we are just communicating the basics necessary to submit code changes to the MITgcm. Spending some time learning the more advanced features of Git will likely pay off in the long run, and not just for MITgcm contributions, as you are likely to encounter it in all sorts of different projects.

To better understand this process, Figure 5.1 shows a conceptual map of the Git setup. Note three copies of the code: the main MITgcm repository source code “upstream” (i.e., owned by the MITgcm maintainers) in the GitHub cloud, a copy of the repository “origin” owned by you, also residing in the GitHub cloud, and a local copy on your personal computer or compute cluster (where you intend to compile and run). The Git and GitHub commands to create this setup are explained more fully below.

One other aspect of Git that requires some explanation to the uninitiated: your local linux copy of the code repository can contain different “branches”, each branch being a different copy of the code repository (this can occur in all git-aware directories). When you switch branches, basic unix commands such as ls or cat will show a different set of files specific to current branch. In other words, Git interacts with your local file system so that edits or newly created files only appear in the current branch, i.e., such changes do not appear in any other branches. So if you swore you made some changes to a particular file, and now it appears those changes have vanished, first check which branch you are on (**git status** is a useful command here), all is probably not lost. NOTE: for a file to be “assigned” to a specific Git branch, Git must first be “made aware” of the file, which occurs after a **git add** and **git commit** (see below). Prior to this, the file will appear in the current folder independently, i.e., regardless of which git branch you are on.

A detailed explanation of steps for contributing MITgcm repository edits:

1. On GitHub, create a local copy of the repository in your GitHub cloud user space: from the main repository (https://github.com/MITgcm/MITgcm) hit the **Fork** button. As mentioned, your GitHub copy “origin” is necessary to streamline the collaborative development process – you need to create a place for your edits in the GitHub cloud, for developers to peruse.

2. Download the code onto your local computer using the **git clone** command. Even if you previously downloaded the code through a “git-aware” method (i.e., a git clone command, see Section 3.2.1), we STRONGLY SUGGEST you download a fresh repository, to a separate disk location, for your development work (keeping your research work separate). Type:
Figure 5.1: A conceptual map of the GitHub setup. Git terminal commands are shown in red, GitHub commands are shown in green.
from your terminal (technically, here you are copying the forked “origin” version from the cloud, not the “upstream” version, but these will be identical at this point).

3. Move into the local clone directory on your computer:

```bash
% cd MITgcm
```

We need to set up a remote that points to the main repository:

```bash
% git remote add upstream https://github.com/MITgcm/MITgcm.git
```

This means that we now have two “remotes” of the project. A remote is just a pointer to a repository not on your computer, i.e., in the GitHub cloud, one pointing to your GitHub user space (“origin”), and this new remote pointing to the original (“upstream”). You can read and write into your “origin” version (since it belongs to you, in the cloud), but not into the “upstream” version. This command just sets up this remote, which is needed in step #4 – no actual file manipulation is done at this point. If in doubt, the command `git remote -v` will list what remotes have been set up.

4. Next make a new branch.

```bash
% git fetch upstream
% git checkout -b «YOUR_NEWBRANCH_NAME» upstream/master
```

You will make edits on this new branch, to keep these new edits completely separate from all files on the master branch. The first command `git fetch upstream` makes sure your new branch is the latest code from the main repository; as such, you can redo step 4 at any time to start additional, separate development projects (on a separate, new branch). Note that this second command above not only creates this new branch, from the `upstream/master` branch, it also switches you onto this newly created branch. Naming the branch something descriptive like ‘newfeature’ or ‘bugfix’ (preferably, be even more descriptive) is helpful.

5. Doing stuff! This usually comes in one of three flavors:

i) cosmetic changes, formatting, documentation, etc.;
ii) fixing bug(s), or any change to the code which results in different numerical output; or
iii) adding a feature or new package.

To do this you should:

- edit the relevant file(s) and/or create new files. Refer to *Coding style guide* for details on expected documentation standards and code style requirements. Of course, changes should be thoroughly tested to ensure they compile and run successfully!
- type `git add «FILENAME1» «FILENAME2» ...` to stage the file(s) ready for a commit command (note both existing and brand new files need to be added). “Stage” effectively means to notify Git of the the list of files you plan to “commit” for changes into the version tracking system. Note you can change other files and NOT have them sent to model developers; only staged files will be sent. You can repeat this `git add` command as many times as you like and it will continue to augment the list of files. `git diff` and `git status` are useful commands to see what you have done so far.
- use `git commit` to commit the files. This is the first step in bundling a collection of files together to be sent off to the MITgcm maintainers. When you enter this command, an editor window will pop up. On the top line, type a succinct (<70 character) summary of what these changes accomplished. If your commit is non-trivial and additional explanation is required, leave a blank line and then type a longer description of why the action in this
commit was appropriate etc. It is good practice to link with known issues using the syntax `#ISSUE_NUMBER` in either the summary line or detailed comment. Note that all the changes do not have to be handled in a single commit (i.e. you can git add some files, do a commit, than continue anew by adding different files, do another commit etc.); the git `commit` command itself does not (yet) submit anything to maintainers.

- if you are fixing a more involved bug or adding a new feature, such that many changes are required, it is preferable to break your contribution into multiple commits (each documented separately) rather than submitting one massive commit; each commit should encompass a single conceptual change to the code base, regardless of how many files it touches. This will allow the MITgcm maintainers to more easily understand your proposed changes and will expedite the review process.

When your changes are tested and documented, continue on to step #6, but read all of step #6 and #7 before proceeding; you might want to do an optional “bring my development branch up to date” sequence of steps before step #6.

6. Now we “push” our modified branch with committed changes onto the origin remote in the GitHub cloud. This effectively updates your GitHub cloud copy of the MITgcm repo to reflect the wonderful changes you are contributing.

```bash
% git push -u origin «YOUR_NEWBRANCH_NAME»
```

Some time might elapse during step #5, as you make and test your edits, during which continuing development occurs in the main MITgcm repository. In contrast with some models that opt for static, major releases, the MITgcm is in a constant state of improvement and development. It is very possible that some of your edits occur to files that have also been modified by others. Your local clone however will not know anything about any changes that may have occurred to the MITgcm repo in the cloud, which may cause an issue in step #7 below, when one of three things will occur:

- the files you have modified in your development have NOT been modified in the main repo during this elapsed time, thus git will have no conflicts in trying to update (i.e. merge) your changes into the main repo.
- during the elapsed time, the files you have modified have also been edited/updated in the main repo, but you edited different places in these files than those edits to the main repo, such that git is smart enough to be able to merge these edits without conflict.
- during the elapsed time, the files you have modified have also been edited/updated in the main repo, but git is not smart enough to know how to deal with this conflict (it will notify you of this problem during step #7).

One option is to NOT attempt to bring your development code branch up to date, instead simply proceed with steps #6 and #7 and let the maintainers assess and resolve any conflict(s), should such occur (there is a checkbox ‘Allow edits by maintainers’ that is checked by default when you do step #7). If very little time elapsed during step #5, such conflict is less likely. However, if step #5 takes on the order of months, we do suggest you follow this recipe below to update the code and merge yourself. And/or during the development process, you might have reasons to bring the latest changes in the main repo into your development branch, and thus might opt to follow these same steps.

Development branch code update recipe:

```bash
% git checkout master
% git pull upstream master
% git push origin master
% git checkout «YOUR_NEWBRANCH_NAME»
% git merge master
```

This first command switches you from your development branch to the master branch. The second command above will synchronize your local master branch with the main MITgcm repository master branch (i.e. “pull” any new changes that might have occurred in the upstream repository into your local clone). Note you should not have made any changes to your clone’s master branch; in other words, prior to the pull, master should be a stagnant copy of the code from the day you performed step #1 above. The git `push` command does the opposite of pull, so in the third step you are synchronizing your GitHub cloud copy (“origin”) master branch to your local clone’s master branch (which you just updated). Then, switch back to your development branch via the second git `checkout` command. Finally, the last command will merge any changes into your development branch. If conflicts occur that git cannot resolve, git will
provide you a list of the problematic file names, and in these files, areas of conflict will be demarcated. You will need to edit these files at these problem spots (while removing git’s demarcation text), then do a `git add «FILENAME»` for each of these files, followed by a final `git commit` to finish off the merger.

Some additional `git diff` commands to help sort out file changes, in case you want to assess the scope of development changes, are as follows. `git diff master upstream/master` will show you all differences between your local master branch and the main MITgcm repo, i.e., so you can peruse what parallel MITgcm changes have occurred while you were doing your development (this assumes you have not yet updated your clone’s master branch). You can check for differences on individual files via `git diff master upstream/master «FILENAME»`. If you want to see all differences in files you have modified during your development, the command is `git diff master`. Similarly, to see a combined list of both your changes and those occurring to the main repo, `git diff upstream/master`.

Aside comment: if you are familiar with git, you might realize there is an alternate way to merge, using the “rebase” syntax. If you know what you are doing, feel free to use this command instead of our suggested merge command above.

7. Finally create a “pull request” (a.k.a. “PR”; in other words, you are requesting that the maintainers pull your changes into the main code repository). In GitHub, go to the fork of the project that you made ([https://github.com/«GITHUB_USERNAME»/MITgcm.git](https://github.com/«GITHUB_USERNAME»/MITgcm.git)). There is a button for “Compare and Pull” in your newly created branch. Click the button! Now you can add a final succinct summary description of what you’ve done in your commit(s), flag up any issues, and respond to the remaining questions on the PR template form. If you have made non-trivial changes to the code or documentation, we will note this in the MITgcm change log, doc/tag-index. Please suggest how to note your changes in doc/tag-index; we will not accept the PR if this field is left blank. The maintainers will now be notified and be able to peruse your changes! In general, the maintainers will try to respond to a new PR within a week. While the PR remains open, you can go back to step #5 and make additional edits, `git adds`, `git commits`, and then redo step #6; such changes will be added to the PR (and maintainers re-notified), no need to redo step #7.

Your pull request remains open until either the maintainers fully accept and merge your code changes into the main repository, or decide to reject your changes. Occasionally, the review team will reject changes that are not sufficiently aligned with and do not fit with the code structure; the review team is always happy to discuss their decisions, but wants to avoid people investing extensive additional effort in code that has a fundamental design flaw. But much more likely than outright rejection, you will instead be asked to respond to feedback, modify your code changes in some way, and/or clean up your code to better satisfy our style requirements, etc., and the pull request will remain open. In some cases, the maintainers might take initiative to make some changes to your pull request (such changes can then be incorporated back into your local branch simply by typing `git pull` from your branch), but more typically you will be asked to undertake the majority of the necessary changes.

It is possible for other users (besides the maintainers) to examine or even download your pull request; see Reviewing pull requests.

The current review team is Jean-Michel Campin, Ed Doddridge, Chris Hill, Oliver Jahn, and Jeff Scott.
5.3 Coding style guide

Detailed instructions or link to be added.

5.4 Creating MITgcm packages

Optional parts of code are separated from the MITgcm core driver code and organized into packages. The packaging structure provides a mechanism for maintaining suites of code, specific to particular classes of problem, in a way that is cleanly separated from the generic fluid dynamical engine. An overview of available MITgcm packages is presented in Section 8, as illustrated in Figure 8.1. An overview of how to include and use MITgcm packages in your setup is presented in Section 8.1.1, with specific details on using existing packages spread throughout Section 8, Section 9, and Section 10. This sub-section includes information necessary to create your own package for use with MITgcm.

The MITgcm packaging structure is described below using generic package names ${pkg}$. A concrete example of a package is the code for implementing GM/Redi mixing: this code uses the package names ${{PKG} = \text{GMREDI}}$, ${{pkg} = \text{gmredi}}$, and ${{Pkg} = \text{gmRedi}}$.

5.4.1 Package structure

- Compile-time state: Given that each package is allowed to be compiled or not (e.g., all ${{pkg}}$ listed in packages.conf are compiled, see Section 8.1.1.1), genmake2 keeps track of each package's compile-state in PACKAGES_CONFIG.h with CPP option ALLOW_${pkg} being defined (#define) or not (#undef). Therefore, in the MITgcm core code (or code from other included packages), calls to package-specific subroutines and package-specific header file #include statements must be protected within #ifdef ALLOW_${pkg} ... ... #endif /* ALLOW_${pkg} */ (see below) to ensure that the model compiles when this ${{pkg}}$ is not compiled.

- Run-time state: The core driver part of the model can check for a run-time on/off switch of individual package(s) through the Fortran logical flag use${Pkg}$. The information is loaded from a global package setup file called data.pkg. Note a use${Pkg}$ flag is NOT used within the package-local subroutine code (i.e., ${{pkg}_«DO_SOMETHING».F}$ package source code).

- Each package gets its runtime configuration parameters from a file named data.${pkg}. Package runtime configuration options are imported into a common block held in a header file called ${{PKG}.h}$. Note in some packages, the header file ${{PKG}.h}$ is split into ${{PKG}_PARAMS.h}$, which contains the package parameters, and ${{PKG}_VARS.h}$ for the field arrays. The ${{PKG}.h}$ header file(s) can be imported by other packages to check dependencies and requirements from other packages (see Section 5.4.2).

In order for a package's run-time state use${Pkg}$ to be set to true (i.e., “on”), the code build must have its compile-time state ALLOW_${PKG} defined (i.e., “included”), else mitgcmuv will terminate (cleanly) during initialization. A package's run-time state is not permitted to change during a model run.

Every call to a package routine from outside the package requires a check on BOTH compile-time and run-time states:

```c
#include "PACKAGES_CONFIG.h"
#include "CPP_OPTIONS.h"

#ifdef ALLOW_${PKG}
# include "$PKG\_PARAMS.h"
#else
#endif
```
Within an individual package, the header file `${PKG}_OPTIONS.h` is used to set CPP flags specific to that package. This header file should include `PACKAGES_CONFIG.h` and `CPP_OPTIONS.h`, as shown in this example:

```c
#ifdef ALLOW_${PKG}
    IF ( use${Pkg} ) THEN
      .
      .
      CALL ${PKG}_DO_SOMETHING(...)
      .
      ENDIF
#endif
#endif
```

See for example `GMREDI_OPTIONS.h`.

### 5.4.2 Package boot sequence

All packages follow a required "boot" sequence outlined here:

```c
S/R PACKAGES_BOOT()

S/R PACKAGES_READPARMS()
#ifdef ALLOW_${PKG}
    IF ( use${Pkg} ) CALL ${PKG}_READPARMS( retCode )
#endif

S/R PACKAGES_INIT_FIXED()
#ifdef ALLOW_${PKG}
    IF ( use${Pkg} ) CALL ${PKG}_INIT_FIXED( retCode )
#endif

S/R PACKAGES_CHECK()
#ifdef ALLOW_${PKG}
    IF ( use${Pkg} ) CALL ${PKG}_CHECK( retCode )
#endif
```

(continues on next page)
else
    IF ( use${Pkg} ) CALL PACKAGES_CHECK_ERROR( '${PKG}' )
#endif

S/R PACKAGES_INIT_VARIABLES()

#ifdef ALLOW_${PKG}
    IF ( use${Pkg} ) CALL ${PKG}_INIT_VARIA( )
#endif

• **PACKAGES_BOOT()** determines the logical state of all use${Pkg} variables, as defined in the file data.pkg.

• **${PKG}_READPARMS()** is responsible for reading in the package parameters file data.${pkg} and storing the package parameters in ${PKG}.h (or in ${PKG}_PARAMS.h). ${PKG}_READPARMS is called in S/R packages_readparms.F, which in turn is called from S/R initialise_fixed.F.

• **${PKG}_INIT_FIXED()** is responsible for completing the internal setup of a package, including adding any package-specific variables available for output in pkg/diagnostics (done in S/R ${PKG}_DIAGNOSTICS_INIT). ${PKG}_INIT_FIXED is called in S/R packages_init_fixed.F, which in turn is called from S/R initialise_fixed.F. Note: some packages instead use CALL ${PKG}_INITIALISE (or the old form CALL ${PKG}_INIT).

• **${PKG}_CHECK()** is responsible for validating basic package setup and inter-package dependencies. ${PKG}_CHECK can also import parameters from other packages that it may need to check; this is accomplished through header files ${PKG}.h. (It is assumed that parameters owned by other packages will not be reset during ${PKG}_CHECK !!!) ${PKG}_CHECK is called in S/R packages_check.F, which in turn is called from S/R initialise_fixed.F.

• **${PKG}_INIT_VARIA()** is responsible for initialization of all package variables, called after the core model state has been completely initialized but before the core model timestepping starts. This routine calls ${PKG}_READ_PICKUP, where any package variables required to restart the model will be read from a pickup file. ${PKG}_INIT_VARIA is called in packages_init_variables.F, which in turn is called from S/R initialise_varia.F. Note: the name ${PKG}_INIT_VARIA is not yet standardized across all packages; one can find other S/R names such as ${PKG}_INI_VARS or ${PKG}_INIT_VARIABLES or ${PKG}_INIT.

### 5.4.3 Package S/R calls

Calls to package subroutines within the core code timestepping loop can vary. Below we show an example of calls to do calculations, generate output and dump the package state (for pickup):

```plaintext
S/R DO_OCEANIC_PHYS()
#ifdef ALLOW_${PKG}
    IF ( use${Pkg} ) CALL ${PKG}_DO_SOMETHING( )
#endif

S/R DO_THE_MODEL_IO()
#ifdef ALLOW_${PKG}
    IF ( use${Pkg} ) CALL ${PKG}_OUTPUT( )
#endif

S/R PACKAGES_WRITE_PICKUP()
#ifdef ALLOW_${PKG}
    IF ( use${Pkg} ) CALL ${PKG}_WRITE_PICKUP( )
#endif
```
• \_${PKG}_DO_SOMETHING()\_ refers to any local package source code file, which may be called from any
model/src routine (or, from any subroutine in another package). An specific example would be the S/R call
gmredi_calc_tensor.F from within the core S/R model/src/do_oceanic_phys.F.

• \_${PKG}_OUTPUT()\_ is responsible for writing time-average fields to output files (although the cumulat-
ing step is done within other package subroutines). May also call other output routines (e.g., CALL
\_${PKG}_MONITOR) and write snapshot fields that are held in common blocks. Other temporary fields are
directly dumped to file where they are available. Note that pkg/diagnostics output of \_${PKG} variables is
generated in pkg/diagnostics subroutines. \_${PKG}_OUTPUT() is called in S/R do_the_model_io.F NOTE:
1) the S/R \_${PKG}_DIAGS is used in some packages but is being replaced by \_${PKG}_OUTPUT to avoid
confusion with pkg/diagnostics functionality. 2) the output part is not yet in a standard form.

• \_${PKG}_WRITE_PICKUP()\_ is responsible for writing a package pickup file, used in packages where such is
necessary for a restart. \_${PKG}_WRITE_PICKUP is called in packages_write_pickup.F which in turn is
called from the_model_main.F.

Note: In general, subroutines in one package (pkgA) that only contains code which is connected to a 2nd package
(pkgB) will be named pkgA_pkgB_something.F (e.g., gmredi_diagnostics_init.F).

5.4.4 Package “mypackage”

In order to simply creating the infrastructure required for a new package, we have created pkg/mypackage as essen-
tially an existing package (i.e., all package variables defined, proper boot sequence, output generated) that does not do
anything. Thus, we suggest you start with this “blank” package’s code infrastructure and add your new package func-
tionality to it, perusing the existing mypackage routines and editing as necessary, rather than creating a new package
from scratch.

5.5 MITgcm code testing protocols

verification directory includes many examples intended for regression testing (some of which are tutorial experiments
presented in detail in Section 4). Each one of these test-experiment directories contains “known-good” standard output
files (see Section 5.5.2.1) along with all the input (including both code and data files) required for their re-calculation.
Also included in verification is the shell script testreport to perform regression tests.

5.5.1 Test-experiment directory content

Each test-experiment directory («TESTDIR», see verification for the full list of choices) contains several standard
subdirectories and files which testreport recognizes and uses when running a regression test. The directories and
files that testreport uses are different for a forward test and an adjoint test (testreport -adm, see Section 5.5.2)
and some test-experiments are set-up for only one type of regression test whereas others allow both types of tests
(forward and adjoint). Also some test-experiments allow, using the same MITgcm executable, multiple tests using
different parameters and input files, with a primary input set-up (e.g., input/ or input_ad/) and corresponding results
(e.g., results/ output.txt or results/ output_adm.txt) and with one or several secondary inputs (e.g., input.
«OTHER»/ or input_ad. «OTHER»/) and corresponding results (e.g., results/ output.«OTHER». txt or results/
output_adm.«OTHER».txt).

directory «TESTDIR»/code/ Contains the test-experiment specific source code (i.e., files that have been modified
from the standard MITgcm repository version) used to build the MITgcm executable (mitgcmuv) for forward-
test (using genmake2 -mods=../code).

It can also contain specific source files with the suffix _mpi to be used in place of the corresponding file (without
suffix) for an MPI test (see Section 5.5.2). The presence or absence of SIZE.h_mpi determines whether or not an
MPI test on this test-experiment is performed or skipped. Note that the original code/SIZE.h_mpi is not directly
used as SIZE.h to build an MPI-executable; instead, a local copy build/SIZE.h.mpi is derived from code/
SIZE.h_mpi by adjusting the number of processors (nPx, nPy) according to «NUMBER_OF_PROCS» (see
Section 5.5.2, testreport - MPI); then it is linked to SIZE.h (ln -s SIZE.h.mpi SIZE.h) before building
the MPI-executable.

directory «TESTDIR»/code_ad/ Contains the test-experiment specific source code used to build the MITgcm ex-
ecutable (mitgcmuv_ad) for adjoint-test (using genmake2 -mods=../code_ad). It can also contain specific
source files with the suffix _mpi (see above).

directory «TESTDIR»/build/ Directory where testreport will build the MITgcm executable for forward and adjoint
tests. It is initially empty except in some cases will contain an experiment specific genmake_local file (see
Section 3.5.2).

directory TESTDIR/input/ Contains the input and parameter files used to run the primary forward test of this test-
experiment.

It can also contain specific parameter files with the suffix .mpi to be used in place of the corresponding file
(without suffix) for MPI tests, or with suffix .mth to be used for multi-threaded tests (see Section 5.5.2). The
presence or absence of eedata.mth determines whether or not a multi-threaded test on this test-experiment is
performed or skipped, respectively.

To save disk space and reduce downloading time, multiple copies of the same input file are avoided by using
a shell script prepare_run. When such a script is found in TESTDIR/input/, testreport runs this script in
directory TESTDIR/run/ after linking all the input files from TESTDIR/input/.

directory «TESTDIR»/input_ad/ Contains the input and parameter files used to run the primary adjoint test of this
test-experiment. It can also contain specific parameter files with the suffix .mpi and shell script prepare_run
as described above.

directory «TESTDIR»/input."OTHER"/ Contains the input and parameter files used to run the secondary OTHER
forward test of this test-experiment. It can also contain specific parameter files with suffix .mpi or .mth and
shell script prepare_run (see above).

The presence or absence the file eedata.mth determines whether or not a secondary multi-threaded test on this
test-experiment is performed or skipped.

directory «TESTDIR»/input_ad."OTHER"/ Contains the input and parameter files used to run the secondary
OTHER adjoint test of this test-experiment. It can also contain specific parameter files with the suffix .mpi
and shell script prepare_run (see above).

directory «TESTDIR»/results/ Contains reference standard output used for test comparison. results/output.txt
and results/output_adm.txt, respectively, correspond to primary forward and adjoint test run on the refer-
ence platform (currently villon.mit.edu) on one processor (no MPI, single thread) using the reference compiler
(currently the GNU Fortran compiler gfortran). The presence of these output files determines whether or not
testreport is testing or skipping this test-experiment. Reference standard output for secondary tests (results/
output."OTHER".txt or results/output_adm."OTHER".txt) are also expected here.

directory «TESTDIR»/run/ Initially empty directory where testreport will run the MITgcm executable for primary
forward and adjoint tests.

Symbolic links (using command ln -s) are made for input and parameter files (from ../input/ or from ../
/input_ad/) and for MITgcm executable (from ../build/) before the run proceeds. The sequence of links
(function linkdata within shell script testreport) for a forward test is:

- link and rename or remove links to special files with suffix .mpi or .mth from ../input/
- link files from ../input/
- execute ../input/prepare_run (if it exists)

The sequence for an adjoint test is similar, with ../input_ad/ replacing ../input/.
directory «TESTDIR»/tr_run.«OTHER»/ Directory created by testreport to run the MITgcm executable for secondary “OTHER” forward or adjoint tests.

The sequence of links for a forward secondary test is:

- link and rename or remove links to special files with suffix .mpi or .mth from ../input.OTHER/
- link files from ../input.OTHER/
- execute ../input.OTHER/prepare_run (if it exists)
- link files from ../input/
- execute ../input/prepare_run (if it exists)

The sequence for an adjoint test is similar, with ../input_ad.OTHER/ and ../input_ad/ replacing ../input.OTHER/ and ../input/.

5.5.2 The testreport utility

The shell script testreport, which was written to work with genmake2, can be used to build different versions of MITgcm code, run the various examples, and compare the output. On some systems, the testreport script can be run with a command line as simple as:

```bash
% cd verification
% ./testreport -optfile ../tools/build_options/linux_amd64_gfortran
```

The testreport script accepts a number of command-line options which can be listed using the -help option. The most important ones are:

-ieee (default) / -fast If allowed by the compiler (as defined in the specified optfile), use IEEE arithmetic (genmake2 -ieee). In contrast, -fast uses the optfile default for compiler flags.

-devel Use optfile development flags (assumes specified in optfile).

-optfile «/PATH/FILENAME» (or -optfile '«/PATH/F1» «/PATH/F2» ...') This specifies a list of “options files” that will be passed to genmake2. If multiple options files are used (for example, to test different compilers or different sets of options for the same compiler), then each options file will be used with each of the test directories.

-tdir «TESTDIR» (or -tdir '«TDIR1» «TDIR2» ...') This option specifies the test directory or list of test directories that should be used. Each of these entries should exactly match (note: they are case sensitive!) the names of directories in verification. If this option is omitted, then all directories that are properly formatted (that is, containing an input subdirectory and a results/output.txt file) will be used.

-skipdir «TESTDIR» (or -skipdir '«TDIR1» «TDIR2» ...') This option specifies a test directory or list of test directories to skip. The default is to test ALL directories in verification.

-MPI «NUMBER_OF_PROCS» (or -mpi) If the necessary file «TESTDIR»/code/SIZE.h_mpi exists, then use it (and all TESTDIR/code/*_mpi files) for an MPI-enabled run. The option -MPI followed by the maximum number of processors enables to build and run each test-experiment using different numbers of MPI processors (specific number chosen by: multiple of nPx*nPy from «TESTDIR»/code/SIZE.h_mpi and not larger than «NUMBER_OF_PROCS»). The short option (-mpi) can only be used to build and run on 2 MPI processors (equivalent to -MPI 2).

Note that the use of MPI typically requires a special command option (see “-command” below) to invoke the MPI executable.

-command='«SOME COMMANDS TO RUN»’ For some tests, particularly MPI runs, a specific command might be needed to run the executable. This option allows a more general command (or shell script) to be invoked.

5.5. MITgcm code testing protocols
The default here is for «SOME COMMANDS TO RUN» to be replaced by `mpirun -np TR_NPROC mitgcmuv`. If on your system you require something other than `mpirun`, you will need to use the option and specify your computer’s syntax. Because the number of MPI processors varies according to each test-experiment, the keyword `TR_NPROC` will be replaced by its effective value, the actual number of MPI processors needed to run the current test-experiment.

`-mth` Compile with `gmake2 -omp` and run with multiple threads (using `eedata.mth`).

`-adm` Compile and test the adjoint suite of verification runs using TAF.

`-clean «NUMBER»` Set matching criteria to «NUMBER» of significant digits (default is 10 digits).

Additional testreport options are available to pass options to `gmake2` (called during testreport execution) as well as additional options to skip specific steps of the testreport shell script. See `testreport -help` for a detailed list.

In the verification/ directory, the testreport script will create an output directory «tr_NAME_DATE_N», with your computer hostname substituted for NAME, the current date for DATE, followed by a suffix number N to distinguish from previous testreport output directories. Unless you specify otherwise using the `--tdir` or `--skipdir` options described above, all sub-directories (i.e., TESTDIR experiments) in verification will be tested. testreport writes progress to the screen (stdout) and reports into the «tr_NAME_DATE_N/TESTDIR» sub-directories as it runs. In particular, one can find, in each TESTDIR subdirectory, a `summary.txt` file in addition to log and/or error file(s) (depending how the run failed, if this occurred). `summary.txt` contains information about the run and a comparison of the current output with “reference output” (see below for information on how this reference output is generated). The test comparison involves several output model variables. By default, for a forward test, these are the 2D solver initial residual cg2d_init_res and 3D state variables (T, S, U, V) from pkg/monitor output; by default for an adjoint test, the cost-function and gradient-check. However, some test-experiments use some package-specific variables from pkg/monitor according to the file «TESTDIR*/input/[«ad]*/[«OTHER*/]tr_checklist specification. Note that at this time, the only variables that are compared by testreport are those dumped in standard output via pkg/monitor, not output produced by pkg/diagnostics. Monitor output produced from ALL run time steps are compared to assess significant digit match; the worst match is reported. At the end of the testing process, a composite `summary.txt` file is generated in the top «tr_NAME_DATE_N» directory as a compact, combined version of the `summary.txt` files located in all TESTDIR sub-directories (a slightly more condensed version of this information is also written to file `tr_out.txt` in the top verification/ directory; note this file is overwritten upon subsequent testreport runs). Figure 5.2 shows an excerpt from the composite `summary.txt`, created by running the full testreport suite (in the example here, on a linux cluster, using gfortran):

The four columns on the left are build/run results (successful=Y, unsuccessful=N). Explanation of these columns is as follows:

- **Gen2**: did gmake2 build the makefile for this experiment without error?
- **Dpnd**: did the `make depend` for this experiment complete without error?
- **Make**: did the `make` successfully generate a `mitgcmuv` executable for this experiment?
- **Run**: did execution of this experiment startup and complete successfully?

The next set of columns shows the number of significant digits matched from the monitor output “cg2d”, “min”, “max”, “mean”, and “s d” (standard deviation) for variables T, S, U, and V (see column headings), as compared with the reference output. **NOTE:** these column heading labels are for the default list of variables, even if different variables are specified in a `tr_checklist` file (for reference, the list of actual variables tested for a specific TESTDIR experiment is output near the end of the file `summary.txt` appearing in the specific TESTDIR experiment directory). For some experiments, additional variables are tested, as shown in “PTR 01”, “PTR 02” sets of columns; testreport will detect if tracers are active in a given experiment and check digit match on their concentration values. A match to near-full machine precision is 15-16 digits; this generally will occur when a similar type of computer, similar operating system, and similar version of Fortran compiler are used for the test. Otherwise, different round-off can occur, and due to the chaotic nature of ocean and climate models, fewer digits (typically, 10-13 digits) are matched. A match of 22 digits generally is due to output being exactly 0.0. In some experiments, some variables may not be used or meaningful, which causes the ‘0’ and ‘4’ match results in several of the adjustment experiments above.
Figure 5.2: Example output from testreport summary.txt
While the significant digit match for many variables is tested and displayed in `summary.txt`, only one of these is used to assess pass/fail (output to the right of the match test results) – the number bracketed by > and <. For example, see above for experiment `advect_cs` the pass/fail test occurs on variable “T: s d” (i.e., standard deviation of potential temperature), the first variable in the list specified in `verification/advect_cs/input/tr_checklist`. By default (i.e., if no file `tr_checklist` is present), pass/fail is assessed on the cg2d monitor output. See the `testreport` script for a list of permissible variables to test and a guide to their abbreviations. See `tr_checklist` files in the input subdirectories of several TESTDIR experiments (e.g., `verification/advect_xz/input/tr_checklist`) for examples of syntax (note, a + after a variable in a `tr_checklist` file is shorthand to compare the mean, minimum, maximum, and standard deviation for the variable).

### 5.5.2.1 Reference Output

Reference output is currently generated using the Linux server `villon.mit.edu` which employs an Intel Haswell processor running Ubuntu 18.04.3 LTS. For each verification experiment in the MITgcm repository, this reference output is stored in the file `<TESTDIR>/results/output.txt`, which is the standard output generated by running `testreport` (using a single process) on `villon.mit.edu` using the gfortran (GNU Fortran) compiler version 7.4.0.

Using a different gfortran version (or a different Fortran compiler entirely), and/or running with MPI, a different operating system, or a different processor (cpu) type will generally result in output that differs to machine precision. The greater the number of such differences between your platform and this reference platform, typically the fewer digits of matching output precision.

### 5.5.3 The do_tst_2+2 utility

The shell script `tools/do_tst_2+2` can be used to check the accuracy of the restart procedure. For each experiment that has been run through testreport, `do_tst_2+2` executes three additional short runs using the `tools/tst2+2` script. The first run makes use of the pickup files output from the run executed by `testreport` to restart and run for four time steps, writing pickup files upon completion. The second run is similar except only two time steps are executed, writing pickup files. The third run restarts from the end of the second run, executing two additional time steps, writing pickup files upon completion. In order to successfully pass `do_tst_2+2`, not only must all three runs execute and complete successfully, but the pickups generated at the end the first run must be identical to the pickup files from the end of the third run. Note that a prerequisite to running `do_tst_2+2` is running `testreport`, both to build the executables used by `do_tst_2+2`, and to generate the pickup files from which `do_tst_2+2` begins execution.

The `tools/do_tst_2+2` script should be called from the `verification/` directory, e.g.:

```
% cd verification
% ../tools/do_tst_2+2
```

The `do_tst_2+2` script accepts a number of command-line options which can be listed using the `--help` option. The most important ones are:

- `-t <TESTDIR>` Similar to `testreport` option `-tdir`, specifies the test directory or list of test directories that should be used. If omitted, the test is attempted in all sub-directories.

- `-skd <TESTDIR>` Similar to `testreport` option `--skipdir`, specifies a test directory or list of test directories to skip.

- `-mpi` Run the tests using MPI; requires the prerequisite `testreport` run to have been executed with the `-mpi` or `-MPI <NUMBER_OF_PROCS>` flag. No argument is necessary, as the `do_tst_2+2` script will determine the correct number of processes to use for your executable.

- `-clean` Clean up any output generated from the `do_tst_2+2`. This step is necessary if one wants to do additional `testreport` runs from these directories.

Upon completion, `do_tst_2+2` will generate a file `tst_2+2_out.txt` in the `verification/` directory which summarizes the results. The top half of the file includes information from the composite `summary.txt` file from the prerequisite...
testreport run. In the bottom half, new results from each verification experiment are given: each line starts with four Y/N indicators indicating if pickups from the testreport run were available, and whether runs 1, 2 and 3, completely successfully, respectively, followed by a pass or fail from the output pickup file comparison test, followed by the TEST- DIR experiment name. In each «TESTDIR»/run subdirectory do_tst_2+2 also creates a log file tst_2+2_out.log which contains additional information. During do_tst_2+2 execution a separate directory of summary information, including log files for all failed tests, is created in an output directory «rs_NAME_DATE_N» similar to the syntax for the testreport output directory name. Note however this directory is deleted by default upon do_tst_2+2 completion, but can be saved by adding the do_tst_2+2 command line option -a NONE.

5.5.4 Daily Testing of MITgcm

On a daily basis, MITgcm runs a full suite of testreport (i.e., forward and adjoint runs, single process, single-threaded and mpi) on an array of different clusters, running using different operating systems, testing several different Fortran compilers. The reference machine villon.mit.edu is one of such daily test machines. When changes in output occur from previous runs, even if as minor as changes in numeric output to machine precision, MITgcm maintainers are automatically notified.

Links to summary results from the daily testing are posted at http://mitgcm.org/public/testing.html.

5.5.5 Required Testing for MITgcm Code Contributors

5.5.5.1 Using testreport to check your new code

Before submitting your pull request for approval, if you have made any changes to MITgcm code, however trivial, you MUST complete the following:

- Run testreport (on all experiments) on an unmodified master branch of MITgcm. We suggest using the -devel option and gfortran (typically installed in most linux environments) although neither is strictly necessary for this test. Depending how different your platform is from our reference machine setup, typically most tests will pass but some match tests may fail; it is possible one or more experiments might not even build or run successfully. But even if there are multiple experiment fails or unsuccessful builds or runs, do not despair, the purpose at this stage is simply to generate a reference report on your local platform using the master code. It may take one or more hours for testreport to complete.

- Save a copy of this summary output from running testreport on the master branch: from the verification directory, type `cp tr_out.txt tr_out_master.txt`. The file tr_out.txt is simply a condensed version of the composite summary.txt file located in the «tr_NAME_DATE_N» directory. Note we are not making this file “git-aware”, as we have no desire to check this into the repo, so we are using an old-fashioned copy to save the output here for later comparison.

- Switch to your pull request branch, and repeat the testreport sequence using the same options.

- From the verification directory, type `diff tr_out_master.txt tr_out.txt` which will report any differences in testreport output from the above tests. If no differences occur (other than timestamp-related), see below if you are required to do a do_tst_2+2 test; otherwise, you are clear for submitting your pull request.

Differences might occur due to one or more of the following reasons:

- Your modified code no longer builds properly in one or more experiments. This is likely due to a Fortran syntax error; examine output and log files in the failed experiment TESTDIR to identify and fix the problem.

- The run in the modified code branch terminates due to a numerical exception error. This too requires further investigation into the cause of the error, and a remedy, before the pull request should be submitted.

- You have made changes which require changes to input parameters (e.g., renaming a namelist parameter, changing the units or function of an input parameter, etc.) This by definition is a “breaking change”, which must be noted when completing the PR template – but should not deter you from submitting your PR. Ultimately, you and the
maintainers will likely have to make changes to one or more verification experiments, but as a first step we will want to review your PR.

- You have made algorithmic changes which change model output in some or all setups; this too is a “breaking change” that should be noted in the PR template. As usual recourse, if the PR is accepted, the maintainers will re-generate reference output and push to the affected «TESTDIR»/results/ directories when the PR is merged.

Most typically, running testreport using a single process is a sufficient test. However, any code changes which call MITgcm routines (such as eesupp/src/global_sum.F) employing low-level MPI-directives should run testreport with the -mpi option enabled.

5.5.5.2 Using do_tst_2+2 to check your new code

If you make any kind of algorithmic change to the code, or modify anything related to generating or reading pickup files, you are also required to also complete a do_tst_2+2. Again, run the test on both the unmodified master branch and your pull request branch (after you have run testreport on both branches). Verify that the output tst_2+2_out.txt file is identical between branches, similar to the above procedure for the file tr_out.txt. If the files differ, attempt to identify and fix what is causing the problem.

5.5.5.3 Automatic testing with GitHub Actions

Once your PR is submitted onto GitHub, the continuous integration service GitHub Actions runs additional tests on your PR submission. On the ‘Pull request’ tab in GitHub (https://github.com/MITgcm/MITgcm/pulls), find your pull request; initially you will see a yellow circle to the right of your PR title, indicating testing in progress. Eventually this will change to a green checkmark (pass) or a red X (fail). If you get a red X, click the X and then click on ‘Details’ to list specifics tests that failed; these can be clicked to produce a screenshot with error messages.

Note that GitHub Actions builds documentation (both html and latex) in addition to code testing, so if you have introduced syntax errors into the documentation files, these will be flagged at this stage. Follow the same procedure as above to identify the error messages so the problem(s) can be fixed. Make any appropriate edits to your pull request, re-git add and re-git commit any newly modified files, re-git push. Anytime changes are pushed to the PR, GitHub Actions will re-run its tests.

The maintainers will not review your PR until all GitHub Actions tests pass.

5.6 Contributing to the manual

Whether you are simply correcting typos or describing undocumented packages, we welcome all contributions to the manual. The following information will help you make sure that your contribution is consistent with the style of the MITgcm documentation. (We know that not all of the current documentation follows these guidelines - we’re working on it)

The manual is written in rst format, which is short for ReStructuredText directives. rst offers many wonderful features: it automatically does much of the formatting for you, it is reasonably well documented on the web (e.g., primers available here and here), it can accept raw latex syntax and track equation labelling for you, in addition to numerous other useful features. On the down side however, it can be very fussy about formatting, requiring exact spacing and indenting, and seemingly innocuous things such as blank spaces at ends of lines can wreak havoc. We suggest looking at the existing rst files in the manual to see exactly how something is formatted, along with the syntax guidelines specified in this section, prior to writing and formatting your own manual text.

The manual can be viewed either of two ways: interactively (i.e., web-based), as hosted by read-the-docs (https://readthedocs.org/), requiring an html format build, or downloaded as a pdf file. When you have completed your documentation edits, you should double check both versions are to your satisfaction, particularly noting that figure sizing and placement may be rendered differently in the pdf build. See Building the manual for detail.
5.6.1 Section headings

- Chapter headings - these are the main headings with integer numbers - underlined with ****
- Section headings - headings with number format X.Y - underlined with ====
- Subsection headings - headings with number format X.Y.Z - underlined with ----
- Subsubsection headings - headings with number format X.Y.Z.A - underlined with ~~~~
- Paragraph headings - headings with no numbers - underlined with ^^^^

N.B. all underlinings should be the same length as the heading. If they are too short an error will be produced.

5.6.2 Internal document references

rst allows internal referencing of figures, tables, section headings, and equations, i.e. clickable links that bring the reader to the respective figure etc. in the manual. To be referenced, a unique label is required. To reference figures, tables, or section headings by number, the rst (inline) directive is :numref:`«LABELNAME»`. For example, this syntax would write out Figure XX on a line (assuming «LABELNAME» referred to a figure), and when clicked, would relocate your position in the manual to figure XX. Section headings can also be referenced so that the name is written out instead of the section number, instead using this directive :ref:`«LABELNAME»`

Equation references have a slightly different inline syntax: :eq:`«LABELNAME»` will produce a clickable equation number reference, surrounded by parentheses.

For instructions how to assign a label to tables and figures, see below. To label a section heading, labels go above the section heading they refer to, with the format `.. _«LABELNAME»:`. Note the necessary leading underscore. You can also place a clickable link to any spot in the text (e.g., mid-section), using this same syntax to make the label, using the syntax :ref:`«SOME TEXT TO CLICK ON» <«LABELNAME»>` for the link.

5.6.3 Citations

In the text, references should be given using the standard “Author(s) (Year)” shorthand followed by a link to the full reference in the manual bibliography. This link is accomplished using the syntax :cite:`«BIB_REFERENCE»`; this will produce clickable text, usually some variation on the authors’ initials or names, surrounded by brackets.

Full references are specified in the file doc/manual_references.bib using standard BibTeX format. Even if unfamiliar with BibTeX, it is relatively easy to add a new reference by simply examining other entries. Furthermore, most publishers provide a means to download BibTeX formatted references directly from their website. Note this file is in approximate alphabetic order by author name. For all new references added to the manual, please include a DOI or a URL in addition to journal name, volume and other standard reference information. An example JGR journal article reference is reproduced below; note the «BIB_REFERENCE» here is “bryan:79” so the syntax in the rst file format would be “Bryan and Lewis (1979) :cite:`bryan:79`, which will appear in the manual as Bryan and Lewis (1979) [BL79].

@Article{bryan:79,  author = {Bryan, K. and L.J. Lewis},  title = {A water mass model of the world ocean},  journal = jgr,  volume = 84,  number = {C5},  pages = {2503–2517},  doi = {10.1029/JC084iC05p02503},
year = 1979,
}

5.6.4 Other embedded links

Hyperlinks: to reference a (clickable) URL, simply enter the full URL. If you want to have a different, clickable text link instead of displaying the full URL, the syntax is `«CLICKABLE TEXT» <«URL»>` (the `<` and `>` are literal characters, and note the trailing underscore). For this kind of link, the clickable text has to be unique for each URL. If you would like to use non-unique text (like ‘click here’), you should use an ‘anonymous reference’ with a double trailing underscore: `«CLICKABLE TEXT» <«URL»>`.

File references: to create a link to pull up MITgcm code (or any file in the repo) in a code browser window, the syntax is :filelink:`PATH/FIENAME`. If you want to have a different text link to click on (e.g., say you didn’t want to display the full path), the syntax is :filelink:`CLICKABLE TEXT <PATH/FIENAME>` (again, the `<` and `>` are literal characters). The top directory here is https://github.com/MITgcm/MITgcm, so if for example you wanted to pop open the file dynamics.F from the main model source directory, you would specify model/src/dynamics.F in place of `PATH/FIENAME`.

Variable references: to create a link to bring up a webpage displaying all MITgcm repo references to a particular variable name (for this purpose we are using the LXR Cross Referencer), the syntax is :varlink:`NAME_OF_VARIABLE`. This will work on CPP options as well as FORTRAN identifiers (e.g., common block names, subroutine names).

5.6.5 Symbolic Notation

Inline math is done with :math:`«LATEX_HERE»`

Separate equations, which will be typeset on their own lines, are produced with:

```
.. math::
   «LATEX_HERE»
   :label: «EQN_LABEL_HERE»
```

Labelled separate equations are assigned an equation number, which may be referenced elsewhere in the document (see Section 5.6.2). Omitting the :label: above will still produce an equation on its own line, except without an equation label. Note that using latex formatting \begin{aligned} ... \end{aligned} across multiple lines of equations will not work in conjunction with unique equation labels for each separate line (any embedded formatting & characters will cause errors too). Latex alignment will work however if you assign a single label for the multiple lines of equations.

There is a software tool ‘universal document converter’ named pandoc that we have found helpful in converting raw latex documents into rst format. To convert a .tex file into .rst, from a terminal window type:

```
% pandoc -f latex -t rst -o «OUTPUT_FILENAME».rst «INPUT_FILENAME».tex
```

Additional conversion options are available, for example if you have your equations or text in another format; see the pandoc documentation.

Note however we have found that a fair amount of clean-up is still required after conversion, particularly regarding latex equations/labels (pandoc has the unfortunate tendency to add extra spaces, sometimes confusing the rst :math: directive, other times creating issues with indentation).
5.6.6 Figures

The syntax to insert a figure is as follows:

```
.. figure:: «PATHNAME/FILENAME».*
   :width: 80%
   :align: center
   :alt: «TEXT DESCRIPTION OF FIGURE HERE»
   :name: «MY_FIGURE_NAME»
```

The figure caption goes here as a single line of text.

```
figure:: The figure file is located in subdirectory pathname above; in practice, we have located figure files in subdirectories figs off each manual chapter subdirectory. The wild-card * is used here so that different file formats can be used in the build process. For vector graphic images, save a pdf for the pdf build plus a svg file for the html build. For bitmapped images, gif, png, or jpeg formats can be used for both builds, no wild-card necessary, just substitute the actual extension (see here for more info on compatible formats). [Note: A repository for figure source .eps needs to be created]
   :width: used to scale the size of the figure, here specified as 80% scaling factor (check sizing in both the pdf and html builds, as you may need to adjust the figure size within the pdf file independently).
   :align: can be right, center, or left.
   :name: use this name when you refer to the figure in the text, i.e. :numref:`«MY_FIGURE_NAME»`.
```

Note the indentation and line spacing employed above.

5.6.7 Tables

There are two syntaxes for tables in reStructuredText. Grid tables are more flexible but cumbersome to create. Simple tables are easy to create but limited (no row spans, etc.). The raw rst syntax is shown first, then the output.

Grid Table Example:

```
+------------+------------+-----------+
| Header 1   | Header 2   | Header 3  |
+============+============+===========+
| body row 1 | column 2   | column 3  |
+------------+------------+-----------+
| body row 2 | Cells may span columns. |
+------------+------------------------+
| body row 3 | Cells may | - Cells |
| span rows. | - contain |        |
| body row 4 | - blocks. |          |
+------------+------------------------+
```

Simple Table Example:

```
<table>
<thead>
<tr>
<th>Header 1</th>
<th>Header 2</th>
<th>Header 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>body row 1</td>
<td>column 2</td>
<td>column 3</td>
</tr>
<tr>
<td>body row 2</td>
<td>Cells may span columns.</td>
<td></td>
</tr>
</tbody>
</table>
| body row 3     | Cells may span rows. |• Cells
| body row 4     |               |• contain
```

5.6. Contributing to the manual 337
5.6.8 Other text blocks

Conventionally, we have used the rst ‘inline literal’ syntax around any literal computer text (commands, labels, literal computer syntax etc.) Surrounding text with double back-quotes `` results in output html like this.

To set several lines apart in an whitespace box, e.g. useful for showing lines in from a terminal session, rst uses :: to set off a ‘literal block’. For example:

::

    % unix_command_foo
    % unix_command_fum

(note the :: would not appear in the output html or pdf) A splasher way to outline a block, including a box label, is to employ what is termed in rst as an ‘admonition block’. In the manual these are used to show calling trees and for describing subroutine inputs and outputs. An example of a subroutine input/output block is as follows:

This is an admonition block showing subroutine in/out syntax

.. admonition:: SUBROUTINE_NAME
   :class: note

   | var1: VAR1 (WHERE_VAR1_DEFINED.h) 
   | var2: VAR1 (WHERE_VAR2_DEFINED.h ) 
   | var3: VAR1 (WHERE_VAR3_DEFINED.h )

An example of a subroutine in/out admonition box in the documentation is here.
An example of a calling tree in the documentation is here.

To show text from a separate file (e.g., to show lines of code, show comments from a Fortran file, show a parameter file etc.), use the `literalinclude` directive. Example usage is shown here:

```
.. literalinclude:: «FILE_TO_SHOW»
   :start-at: String indicating where to start grabbing text
   :end-at: String indicating where to stop grabbing text
```

Unlike the `:filelink:` and `:varlink:` directives, which assume a file path starting at the top of the MITgcm repository, one must specify the path relative to the current directory of the file (for example, from the doc directory, it would require `../..` at the start of the file path to specify the base directory of the MITgcm repository). Note one can instead use `:start-after:` and `:end-before:` to get text from the file between (not including) those lines. If one omits the `start-at` or `start-after`, etc. options the whole file is shown. More details for this directive can be found here. Example usage in this documentation is here, where the lines to generate this are:

```
.. literalinclude:: ../../model/src/the_model_main.F
   :start-at: C Invocation from WRAPPER level...
   :end-at: C         :: events.
```

### 5.6.9 Other style conventions

Units should be typeset in normal text, with a space between a numeric value and the unit, and exponents added with the `:sup:` command.

```
9.8 m/s\ :sup:`2`
```

will produce 9.8 m/s\(^2\). If the exponent is negative use two dashes -- to make the minus sign sufficiently long. The backslash removes the space between the unit and the exponent. Similarly, for subscripts the command is `:sub:`.

Alternatively, latex `:math:` directives (see above) may also be used to display units, using the `\text{}` syntax to display non-italic characters.

Line length: as recommended in the sphinx style guide, lines of raw rst text should be kept to fewer than 80 characters (this restriction does not apply to tables, URLs, etc. where a line break might cause difficulty).

- Todo: determine how to break up sections into smaller files

### 5.6.10 Building the manual

Once you’ve made your changes to the manual, you should build it locally to verify that it works as expected. To do this you will need a working python installation with the following packages installed:

- sphinx
- sphinxcontrib-bibtex
- sphinxcontrib-programoutput
- sphinx_rtd_theme
- numpy

These packages can be installed from the Python Package Index using pip. If you have an existing python installation using Anaconda or one of its variants (e.g., miniconda), we recommend that you can create (and use) a clean environment with the required packages like this:
If you don’t yet have a python installation on your computer, we recommend following the Anaconda installation procedure, then following the recipe above. You do not need to learn python to build the manual; just note you should type conda activate mitgcm_build_the_docs in a shell when starting up a manual editing session, and conda deactivate when you finish (also note you only need to perform the conda create ... step above when you initially follow the recipe). This will maintain a clean, separate python virtual environment for manual compilation and won’t interfere with your python setup should you decide to learn python in the future.

Once these modules are installed you can build the html version of the manual by running make html in the doc directory.

To build the pdf version of the manual you will also need a working version of LaTeX that includes several packages that are not always found in minimal LaTeX installations. The command to build the pdf version is make latexpdf, which should also be run in the doc directory.

5.7 Reviewing pull requests

The only people with write access to the main repository are a small number of core MITgcm developers. They are the people that will eventually merge your pull requests. However, before your PR gets merged, it will undergo the automated testing on Travis-CI, and it will be assessed by the MITgcm community.

Everyone can review and comment on pull requests. Even if you are not one of the core developers you can still comment on a pull request.

The simplest way to examine a pull request is to use GitHub. You can look at changes made to files (GitHub will show you a standard linux diff for each file changed), read though commit messages, and/or peruse any comments the MITgcm community has made regarding this pull request.

If you are reviewing changes to the documentation, most likely you will also want to review the rendered manual in html format. While this is not available at GitHub, you can view html builds based on the pull request documentation using this link at readthedocs.org. Here you will need to click on the appropriate pull request (as labeled by the pull request number), then click on “View docs” (not the green button near the top of the page, but the text in the middle of the page on the right side).

Finally, if you want to test pull requests locally (i.e., to compile or run the code), you should download the pull request branch. You can do this either by cloning the branch from the pull request:

```bash
git clone -b «THEIR_DEVELOPMENT_BRANCHNAME» https://github.com/«THEIR_GITHUB_USERNAME»/MITgcm.git
```

where «THEIR_GITHUB_USERNAME» is replaced by the username of the person proposing the pull request, and «THEIR_DEVELOPMENT_BRANCHNAME» is the branch from the pull request.

Alternatively, you can add the repository of the user proposing the pull request as a remote to your existing local repository. Navigate to your local repository and type

```bash
git remote add «THEIR_GITHUB_USERNAME» https://github.com/«THEIR_GITHUB_USERNAME»/MITgcm.git
```
where «THEIR_GITHUB_USERNAME» is replaced by the user name of the person who has made the pull request. Then download their pull request changes

```
    git fetch «THEIR_GITHUB_USERNAME»
```

and switch to the desired branch

```
    git checkout --track «THEIR_GITHUB_USERNAME»/«THEIR_DEVELOPMENT_BRANCHNAME»
```

You now have a local copy of the code from the pull request and can run tests locally. If you have write access to the main repository you can push fixes or changes directly to the pull request.

None of these steps, apart from pushing fixes back to the pull request, require write access to either the main repository or the repository of the person proposing the pull request. This means that anyone can review pull requests. However, unless you are one of the core developers you won’t be able to directly push changes. You will instead have to make a comment describing any problems you find.
CHAPTER

SIX

SOFTWARE ARCHITECTURE

This chapter focuses on describing the WRAPPER environment within which both the core numerics and the pluggable packages operate. The description presented here is intended to be a detailed exposition and contains significant background material, as well as advanced details on working with the WRAPPER. The tutorial examples in this manual (see Section 4) contain more succinct, step-by-step instructions on running basic numerical experiments, of various types, both sequentially and in parallel. For many projects, simply starting from an example code and adapting it to suit a particular situation will be all that is required. The first part of this chapter discusses the MITgcm architecture at an abstract level. In the second part of the chapter we described practical details of the MITgcm implementation and the current tools and operating system features that are employed.

6.1 Overall architectural goals

Broadly, the goals of the software architecture employed in MITgcm are three-fold:

• To be able to study a very broad range of interesting and challenging rotating fluids problems;
• The model code should be readily targeted to a wide range of platforms; and
• On any given platform, performance should be comparable to an implementation developed and specialized specifically for that platform.

These points are summarized in Figure 6.1, which conveys the goals of the MITgcm design. The goals lead to a software architecture which at the broadest level can be viewed as consisting of:

1. A core set of numerical and support code. This is discussed in detail in Section 2.
2. A scheme for supporting optional “pluggable” packages (containing for example mixed-layer schemes, biogeochemical schemes, atmospheric physics). These packages are used both to overlay alternate dynamics and to introduce specialized physical content onto the core numerical code. An overview of the package scheme is given at the start of Section 8.
3. A support framework called WRAPPER (Wrappable Application Parallel Programming Environment Resource), within which the core numerics and pluggable packages operate.

This chapter focuses on describing the WRAPPER environment under which both the core numerics and the pluggable packages function. The description presented here is intended to be a detailed exposition and contains significant background material, as well as advanced details on working with the WRAPPER. The “Getting Started” chapter of this manual (Section 3) contains more succinct, step-by-step instructions on running basic numerical experiments both sequentially and in parallel. For many projects simply starting from an example code and adapting it to suit a particular situation will be all that is required.
Wide range of target hardware

Deskside clusters ↔ Terascale MPP
PVP clusters

Wide span of applications

Process studies ↔ Coupled climate experiments

Figure 6.1: The MITgcm architecture is designed to allow simulation of a wide range of physical problems on a wide range of hardware. The computational resource requirements of the applications targeted range from around $10^7$ bytes (~10 megabytes) of memory to $10^{11}$ bytes (~100 gigabytes). Arithmetic operation counts for the applications of interest range from $10^9$ floating point operations to more than $10^{17}$ floating point operations.

6.2 WRAPPER

A significant element of the software architecture utilized in MITgcm is a software superstructure and substructure collectively called the WRAPPER (Wrappable Application Parallel Programming Environment Resource). All numerical and support code in MITgcm is written to “fit” within the WRAPPER infrastructure. Writing code to fit within the WRAPPER means that coding has to follow certain, relatively straightforward, rules and conventions (these are discussed further in Section 6.3.1).

The approach taken by the WRAPPER is illustrated in Figure 6.2, which shows how the WRAPPER serves to insulate code that fits within it from architectural differences between hardware platforms and operating systems. This allows numerical code to be easily retargeted.

6.2.1 Target hardware

The WRAPPER is designed to target as broad as possible a range of computer systems. The original development of the WRAPPER took place on a multi-processor, CRAY Y-MP system. On that system, numerical code performance and scaling under the WRAPPER was in excess of that of an implementation that was tightly bound to the CRAY system’s proprietary multi-tasking and micro-tasking approach. Later developments have been carried out on uniprocessor and multiprocessor Sun systems with both uniform memory access (UMA) and non-uniform memory access (NUMA) designs. Significant work has also been undertaken on x86 cluster systems, Alpha processor based clustered SMP systems, and on cache-coherent NUMA (CC-NUMA) systems such as Silicon Graphics Altix systems. The MITgcm code, operating within the WRAPPER, is also routinely used on large scale MPP systems (for example, Cray T3E and IBM SP systems). In all cases, numerical code, operating within the WRAPPER, performs and scales very competitively with equivalent numerical code that has been modified to contain native optimizations for a particular system (see Hoe et al. 1999) [HHA99].
Figure 6.2: Numerical code is written to fit within a software support infrastructure called WRAPPER. The WRAPPER is portable and can be specialized for a wide range of specific target hardware and programming environments, without impacting numerical code that fits within the WRAPPER. Codes that fit within the WRAPPER can generally be made to run as fast on a particular platform as codes specially optimized for that platform.
6.2.2 Supporting hardware neutrality

The different systems mentioned in Section 6.2.1 can be categorized in many different ways. For example, one common distinction is between shared-memory parallel systems (SMP and PVP) and distributed memory parallel systems (for example x86 clusters and large MPP systems). This is one example of a difference between compute platforms that can impact an application. Another common distinction is between vector processing systems with highly specialized CPUs and memory subsystems and commodity microprocessor based systems. There are numerous other differences, especially in relation to how parallel execution is supported. To capture the essential differences between different platforms the WRAPPER uses a machine model.

6.2.3 WRAPPER machine model

Applications using the WRAPPER are not written to target just one particular machine (for example an IBM SP2) or just one particular family or class of machines (for example Parallel Vector Processor Systems). Instead the WRAPPER provides applications with an abstract machine model. The machine model is very general; however, it can easily be specialized to fit, in a computationally efficient manner, any computer architecture currently available to the scientific computing community.

6.2.4 Machine model parallelism

Codes operating under the WRAPPER target an abstract machine that is assumed to consist of one or more logical processors that can compute concurrently. Computational work is divided among the logical processors by allocating “ownership” to each processor of a certain set (or sets) of calculations. Each set of calculations owned by a particular processor is associated with a specific region of the physical space that is being simulated, and only one processor will be associated with each such region (domain decomposition).

In a strict sense the logical processors over which work is divided do not need to correspond to physical processors. It is perfectly possible to execute a configuration decomposed for multiple logical processors on a single physical processor. This helps ensure that numerical code that is written to fit within the WRAPPER will parallelize with no additional effort. It is also useful for debugging purposes. Generally, however, the computational domain will be subdivided over multiple logical processors in order to then bind those logical processors to physical processor resources that can compute in parallel.

6.2.4.1 Tiles

Computationally, the data structures (e.g., arrays, scalar variables, etc.) that hold the simulated state are associated with each region of physical space and are allocated to a particular logical processor. We refer to these data structures as being owned by the processor to which their associated region of physical space has been allocated. Individual regions that are allocated to processors are called tiles. A processor can own more than one tile. Figure 6.3 shows a physical domain being mapped to a set of logical processors, with each processor owning a single region of the domain (a single tile). Except for periods of communication and coordination, each processor computes autonomously, working only with data from the tile that the processor owns. If instead multiple tiles were allotted to a single processor, each of these tiles would be computed on independently of the other allotted tiles, in a sequential fashion.
Figure 6.3: The WRAPPER provides support for one and two dimensional decompositions of grid-point domains. The figure shows a hypothetical domain of total size $N_x N_y N_z$. This hypothetical domain is decomposed in two-dimensions along the $N_x$ and $N_y$ directions. The resulting tiles are owned by different processors. The owning processors perform the arithmetic operations associated with a tile. Although not illustrated here, a single processor can own several tiles. Whenever a processor wishes to transfer data between tiles or communicate with other processors it calls a WRAPPER supplied function.

### 6.2.4.2 Tile layout

Tiles consist of an interior region and an overlap region. The overlap region of a tile corresponds to the interior region of an adjacent tile. In Figure 6.4 each tile would own the region within the black square and hold duplicate information for overlap regions extending into the tiles to the north, south, east and west. During computational phases a processor will reference data in an overlap region whenever it requires values that lie outside the domain it owns. Periodically processors will make calls to WRAPPER functions to communicate data between tiles, in order to keep the overlap regions up to date (see Section 6.2.6). The WRAPPER functions can use a variety of different mechanisms to communicate data between tiles.

Figure 6.4: A global grid subdivided into tiles. Tiles contain a interior region and an overlap region. Overlap regions are periodically updated from neighboring tiles.
6.2.5 Communication mechanisms

Logical processors are assumed to be able to exchange information between tiles (and between each other) using at least one of two possible mechanisms, shared memory or distributed memory communication. The WRAPPER assumes that communication will use one of these two styles. The underlying hardware and operating system support for the style used is not specified and can vary from system to system.

6.2.5.1 Shared memory communication

Under this mode of communication, data transfers are assumed to be possible using direct addressing of regions of memory. In the WRAPPER shared memory communication model, simple writes to an array can be made to be visible to other CPUs at the application code level. So, as shown below, if one CPU (CPU1) writes the value 8 to element 3 of array a, then other CPUs (here, CPU2) will be able to see the value 8 when they read from a(3). This provides a very low latency and high bandwidth communication mechanism. Thus, in this way one CPU can communicate information to another CPU by assigning a particular value to a particular memory location.

<table>
<thead>
<tr>
<th>CPU1</th>
<th>CPU2</th>
</tr>
</thead>
<tbody>
<tr>
<td>===</td>
<td>===</td>
</tr>
<tr>
<td>a(3) = 8</td>
<td>WHILE ( a(3) .NE. 8 )</td>
</tr>
<tr>
<td></td>
<td>WAIT</td>
</tr>
<tr>
<td></td>
<td>END WHILE</td>
</tr>
</tbody>
</table>

Under shared communication independent CPUs are operating on the exact same global address space at the application level. This is the model of memory access that is supported at the basic system design level in “shared-memory” systems such as PVP systems, SMP systems, and on distributed shared memory systems (e.g., SGI Origin, SGI Altix, and some AMD Opteron systems). On such systems the WRAPPER will generally use simple read and write statements to access directly application data structures when communicating between CPUs.

In a system where assignments statements map directly to hardware instructions that transport data between CPU and memory banks, this can be a very efficient mechanism for communication. In such case multiple CPUs can communicate simply by reading and writing to agreed locations and following a few basic rules. The latency of this sort of communication is generally not that much higher than the hardware latency of other memory accesses on the system. The bandwidth available between CPUs communicating in this way can be close to the bandwidth of the systems main-memory interconnect. This can make this method of communication very efficient provided it is used appropriately.

Memory consistency

When using shared memory communication between multiple processors, the WRAPPER level shields user applications from certain counter-intuitive system behaviors. In particular, one issue the WRAPPER layer must deal with is a systems memory model. In general the order of reads and writes expressed by the textual order of an application code may not be the ordering of instructions executed by the processor performing the application. The processor performing the application instructions will always operate so that, for the application instructions the processor is executing, any reordering is not apparent. However, machines are often designed so that reordering of instructions is not hidden from other second processors. This means that, in general, even on a shared memory system two processors can observe inconsistent memory values.

The issue of memory consistency between multiple processors is discussed at length in many computer science papers. From a practical point of view, in order to deal with this issue, shared memory machines all provide some mechanism to enforce memory consistency when it is needed. The exact mechanism employed will vary between systems. For communication using shared memory, the WRAPPER provides a place to invoke the appropriate mechanism to ensure memory consistency for a particular platform.
Cache effects and false sharing

Shared-memory machines often have local-to-processor memory caches which contain mirrored copies of main memory. Automatic cache-coherence protocols are used to maintain consistency between caches on different processors. These cache-coherence protocols typically enforce consistency between regions of memory with large granularity (typically 128 or 256 byte chunks). The coherency protocols employed can be expensive relative to other memory accesses and so care is taken in the WRAPPER (by padding synchronization structures appropriately) to avoid unnecessary coherence traffic.

Operating system support for shared memory

Applications running under multiple threads within a single process can use shared memory communication. In this case all the memory locations in an application are potentially visible to all the compute threads. Multiple threads operating within a single process is the standard mechanism for supporting shared memory that the WRAPPER utilizes. Configuring and launching code to run in multi-threaded mode on specific platforms is discussed in Section 6.3.2.1. However, on many systems, potentially very efficient mechanisms for using shared memory communication between multiple processes (in contrast to multiple threads within a single process) also exist. In most cases this works by making a limited region of memory shared between processes. The MMAP and IPC facilities in UNIX systems provide this capability as do vendor specific tools like LAPI and IMC. Extensions exist for the WRAPPER that allow these mechanisms to be used for shared memory communication. However, these mechanisms are not distributed with the default WRAPPER sources, because of their proprietary nature.

6.2.5.2 Distributed memory communication

Under this mode of communication there is no mechanism, at the application code level, for directly addressing regions of memory owned and visible to another CPU. Instead a communication library must be used, as illustrated below. If one CPU (here, CPU1) writes the value 8 to element 3 of array a, then at least one of CPU1 and/or CPU2 will need to call a function in the API of the communication library to communicate data from a tile that it owns to a tile that another CPU owns. By default the WRAPPER binds to the MPI communication library for this style of communication (see https://computing.llnl.gov/tutorials/mpi/ for more information about the MPI Standard).

<table>
<thead>
<tr>
<th>CPU1</th>
<th>CPU2</th>
</tr>
</thead>
<tbody>
<tr>
<td>a(3) = 8</td>
<td>WHILE ( a(3) .NE. 8 )</td>
</tr>
<tr>
<td>CALL SEND( CPU2,a(3) )</td>
<td>CALL RECV( CPU1,a(3) )</td>
</tr>
<tr>
<td></td>
<td>END WHILE</td>
</tr>
</tbody>
</table>

Many parallel systems are not constructed in a way where it is possible or practical for an application to use shared memory for communication. For cluster systems consisting of individual computers connected by a fast network, there is no notion of shared memory at the system level. For this sort of system the WRAPPER provides support for communication based on a bespoke communication library. The default communication library used is MPI. It is relatively straightforward to implement bindings to optimized platform specific communication libraries. For example the work described in Hoe et al. (1999) [HHA99] substituted standard MPI communication for a highly optimized library.
6.2.6 Communication primitives

Optimized communication support is assumed to be potentially available for a small number of communication operations. It is also assumed that communication performance optimizations can be achieved by optimizing a small number of communication primitives. Three optimizable primitives are provided by the WRAPPER.

Figure 6.5: Three performance critical parallel primitives are provided by the WRAPPER. These primitives are always used to communicate data between tiles. The figure shows four tiles. The curved arrows indicate exchange primitives which transfer data between the overlap regions at tile edges and interior regions for nearest-neighbor tiles. The straight arrows symbolize global sum operations which connect all tiles. The global sum operation provides both a key arithmetic primitive and can serve as a synchronization primitive. A third barrier primitive is also provided, which behaves much like the global sum primitive.

- **EXCHANGE** This operation is used to transfer data between interior and overlap regions of neighboring tiles. A number of different forms of this operation are supported. These different forms handle:
  - Data type differences. Sixty-four bit and thirty-two bit fields may be handled separately.
  - Bindings to different communication methods. Exchange primitives select between using shared memory or distributed memory communication.
  - Transformation operations required when transporting data between different grid regions. Transferring data between faces of a cube-sphere grid, for example, involves a rotation of vector components.
  - Forward and reverse mode computations. Derivative calculations require tangent linear and adjoint forms of the exchange primitives.

- **GLOBAL SUM** The global sum operation is a central arithmetic operation for the pressure inversion phase of the MITgcm algorithm. For certain configurations, scaling can be highly sensitive to the performance of the global sum primitive. This operation is a collective operation involving all tiles of the simulated domain. Different forms of the global sum primitive exist for handling:
  - Data type differences. Sixty-four bit and thirty-two bit fields may be handled separately.
  - Bindings to different communication methods. Exchange primitives select between using shared memory or distributed memory communication.
  - Forward and reverse mode computations. Derivative calculations require tangent linear and adjoint forms of the exchange primitives.
• **BARRIER** The WRAPPER provides a global synchronization function called barrier. This is used to synchronize computations over all tiles. The **BARRIER** and **GLOBAL SUM** primitives have much in common and in some cases use the same underlying code.

### 6.2.7 Memory architecture

The WRAPPER machine model is aimed to target efficient systems with highly pipelined memory architectures and systems with deep memory hierarchies that favor memory reuse. This is achieved by supporting a flexible tiling strategy as shown in Figure 6.6. Within a CPU, computations are carried out sequentially on each tile in turn. By reshaping tiles according to the target platform it is possible to automatically tune code to improve memory performance. On a vector machine a given domain might be subdivided into a few long, thin regions. On a commodity microprocessor based system, however, the same region could be simulated use many more smaller sub-domains.

![Figure 6.6: The tiling strategy that the WRAPPER supports allows tiles to be shaped to suit the underlying system memory architecture. Compact tiles that lead to greater memory reuse can be used on cache based systems (upper half of figure) with deep memory hierarchies, whereas long tiles with large inner loops can be used to exploit vector systems having highly pipelined memory systems.](image)
6.2.8 Summary

Following the discussion above, the machine model that the WRAPPER presents to an application has the following characteristics:

- The machine consists of one or more logical processors.
- Each processor operates on tiles that it owns.
- A processor may own more than one tile.
- Processors may compute concurrently.
- Exchange of information between tiles is handled by the machine (WRAPPER) not by the application.

Behind the scenes this allows the WRAPPER to adapt the machine model functions to exploit hardware on which:

- Processors may be able to communicate very efficiently with each other using shared memory.
- An alternative communication mechanism based on a relatively simple interprocess communication API may be required.
- Shared memory may not necessarily obey sequential consistency, however some mechanism will exist for enforcing memory consistency.
- Memory consistency that is enforced at the hardware level may be expensive. Unnecessary triggering of consistency protocols should be avoided.
- Memory access patterns may need to be either repetitive or highly pipelined for optimum hardware performance.

This generic model, summarized in Figure 6.7, captures the essential hardware ingredients of almost all successful scientific computer systems designed in the last 50 years.

Figure 6.7: Summary of the WRAPPER machine model.
6.3 Using the WRAPPER

In order to support maximum portability the WRAPPER is implemented primarily in sequential Fortran 77. At a practical level the key steps provided by the WRAPPER are:

1. specifying how a domain will be decomposed
2. starting a code in either sequential or parallel modes of operations
3. controlling communication between tiles and between concurrently computing CPUs.

This section describes the details of each of these operations. Section 6.3.1 explains the way a domain is decomposed (or composed) is expressed. Section 6.3.2 describes practical details of running codes in various different parallel modes on contemporary computer systems. Section 6.3.3 explains the internal information that the WRAPPER uses to control how information is communicated between tiles.

6.3.1 Specifying a domain decomposition

At its heart, much of the WRAPPER works only in terms of a collection of tiles which are interconnected to each other. This is also true of application code operating within the WRAPPER. Application code is written as a series of compute operations, each of which operates on a single tile. If application code needs to perform operations involving data associated with another tile, it uses a WRAPPER function to obtain that data. The specification of how a global domain is constructed from tiles or alternatively how a global domain is decomposed into tiles is made in the file SIZE.h. This file defines the following parameters:

```
File: model/inc/SIZE.h
Parameter: sNx, sNx
Parameter: OLx, OLy
Parameter: nSx, nSy
Parameter: nPx, nPy
```

Together these parameters define a tiling decomposition of the style shown in Figure 6.8. The parameters sNx and sNx define the size of an individual tile. The parameters OLx and OLy define the maximum size of the overlap extent. This must be set to the maximum width of the computation stencil that the numerical code finite-difference operations require between overlap region updates. The maximum overlap required by any of the operations in the MITgcm code distributed at this time is four grid points (some of the higher-order advection schemes require a large overlap region). Code modifications and enhancements that involve adding wide finite-difference stencils may require increasing OLx and OLy. Setting OLx and OLy to a too large value will decrease code performance (because redundant computations will be performed), however it will not cause any other problems.

The parameters nSx and nSy specify the number of tiles that will be created within a single process. Each of these tiles will have internal dimensions of sNx and sNy. If, when the code is executed, these tiles are allocated to different threads of a process that are then bound to different physical processors (see the multi-threaded execution discussion in Section 6.3.2), then computation will be performed concurrently on each tile. However, it is also possible to run the same decomposition within a process running a single thread on a single processor. In this case the tiles will be computed over sequentially. If the decomposition is run in a single process running multiple threads but attached to a single physical processor, then, in general, the computation for different tiles will be interleaved by system level software. This too is a valid mode of operation.

The parameters sNx, sNy, OLx, OLy, nSx and nSy are used extensively by numerical code. The settings of sNx, sNy, OLx, and OLy are used to form the loop ranges for many numerical calculations and to provide dimensions for arrays...
Figure 6.8: The three level domain decomposition hierarchy employed by the WRAPPER. A domain is composed of tiles. Multiple tiles can be allocated to a single process. Multiple processes can exist, each with multiple tiles. Tiles within a process can be spread over multiple compute threads.
holding numerical state. The nSx and nSy are used in conjunction with the thread number parameter myThid. Much of the numerical code operating within the WRAPPER takes the form:

```
DO  bj=myByLo(myThid), myByHi(myThid)
   DO  bi=myBxLo(myThid), myBxHi(myThid)
      : a block of computations ranging
         over 1,sNx +/- OLx and 1,sNy +/- OLy grid points
      :
   ENDDO
ENDDO
```

communication code to sum a number or maybe update
tile overlap regions

```
DO  bj=myByLo(myThid), myByHi(myThid)
   DO  bi=myBxLo(myThid), myBxHi(myThid)
      : another block of computations ranging
         over 1,sNx +/- OLx and 1,sNy +/- OLy grid points
      :
   ENDDO
ENDDO
```

The variables myBxLo(myThid), myBxHi(myThid), myByLo(myThid) and myByHi(myThid) set the bounds of the loops in bi and bj in this schematic. These variables specify the subset of the tiles in the range 1, nSx and 1, nSy1 that the logical processor bound to thread number myThid owns. The thread number variable myThid ranges from 1 to the total number of threads requested at execution time. For each value of myThid the loop scheme above will step sequentially through the tiles owned by that thread. However, different threads will have different ranges of tiles assigned to them, so that separate threads can compute iterations of the bi, bj loop concurrently. Within a bi, bj loop, computation is performed concurrently over as many processes and threads as there are physical processors available to compute.

An exception to the use of bi and bj in loops arises in the exchange routines used when the exch2 package is used with the cubed sphere. In this case bj is generally set to 1 and the loop runs from 1, bi. Within the loop bi is used to retrieve the tile number, which is then used to reference exchange parameters.

The amount of computation that can be embedded in a single loop over bi and bj varies for different parts of the MITgcm algorithm. Consider a code extract from the 2-D implicit elliptic solver:

```
REAL*8  cg2d_r(1-OLx:sNx+OLx,1-OLy:sNy+OLy,nSx,nSy)
REAL*8  err
      :
      :
      err = 0.
DO  bj=myByLo(myThid), myByHi(myThid)
   DO  bi=myBxLo(myThid), myBxHi(myThid)
      DO  J=1,sNy
          DO  I=1,sNx
             err = err + cg2d_r(I,J,bi,bj)*cg2d_r(I,J,bi,bj)
           ENDDO
      ENDDO
ENDDO
```

(continues on next page)
This portion of the code computes the $L_2$-Norm of a vector whose elements are held in the array `cg2d_r`, writing the final result to scalar variable `err`. Notice that under the WRAPPER, arrays such as `cg2d_r` have two extra trailing dimensions. These right most indices are tile indexes. Different threads with a single process operate on different ranges of tile index, as controlled by the settings of `myByLo(myThid)`, `myByHi(myThid)`, `myBxLo(myThid)` and `myBxHi(myThid)`. Because the $L_2$-Norm requires a global reduction, the `bi`, `bj` loop above only contains one statement. This computation phase is then followed by a communication phase in which all threads and processes must participate. However, in other areas of the MITgcm, code entries subsections of code are within a single `bi`, `bj` loop.

The final decomposition parameters are `nPx` and `nPy`. These parameters are used to indicate to the WRAPPER level how many processes (each with `nSx`×`nSy` tiles) will be used for this simulation. This information is needed during initialization and during I/O phases. However, unlike the variables `sNx`, `sNy`, `OLx`, `OLy`, `nSx` and `nSy` the values of `nPx` and `nPy` are absent from the core numerical and support code.

### 6.3.1.1 Examples of SIZE.h specifications

The following different SIZE.h parameter setting illustrate how to interpret the values of `sNx`, `sNy`, `OLx`, `OLy`, `nSx`, `nSy`, `nPx` and `nPy`.

1. **PARAMETER**
   ```
   & sNx = 90,
   & sNy = 40,
   & OLx = 3,
   & OLy = 3,
   & nSx = 1,
   & nSy = 1,
   & nPx = 1,
   & nPy = 1)
   ```
   This sets up a single tile with x-dimension of ninety grid points, y-dimension of forty grid points, and x and y overlaps of three grid points each.

2. **PARAMETER**
   ```
   & sNx = 45,
   & sNy = 20,
   & OLx = 3,
   & OLy = 3,
   & nSx = 1,
   & nSy = 1,
   & nPx = 2,
   & nPy = 2)
   ```
   This sets up tiles with x-dimension of forty-five grid points, y-dimension of twenty grid points, and x and y overlaps of three grid points each. There are four tiles allocated to four separate processes (nPx=2, nPy=2) and arranged so that the global domain size is again ninety grid points in x and forty grid points in y. In general the formula for global grid size (held in model variables `Nx` and `Ny`) is
\[ \text{Nx} = s\text{Nx} \times n\text{Sx} \times n\text{Px} \]
\[ \text{Ny} = s\text{Ny} \times n\text{Sy} \times n\text{Py} \]

3. PARAMETER (
\& s\text{Nx} = 90,
\& s\text{Ny} = 10,
\& O\text{Lx} = 3,
\& O\text{Ly} = 3,
\& n\text{Sx} = 1,
\& n\text{Sy} = 2,
\& n\text{Px} = 1,
\& n\text{Py} = 2)

This sets up tiles with \( x \)-dimension of ninety grid points, \( y \)-dimension of ten grid points, and \( x \) and \( y \) overlaps of three grid points each. There are four tiles allocated to two separate processes (\( n\text{Py} = 2 \)) each of which has two separate sub-domains \( n\text{Sy} = 2 \). The global domain size is again ninety grid points in \( x \) and forty grid points in \( y \). The two sub-domains in each process will be computed sequentially if they are given to a single thread within a single process. Alternatively if the code is invoked with multiple threads per process the two domains in \( y \) may be computed concurrently.

4. PARAMETER (
\& s\text{Nx} = 32,
\& s\text{Ny} = 32,
\& O\text{Lx} = 3,
\& O\text{Ly} = 3,
\& n\text{Sx} = 6,
\& n\text{Sy} = 1,
\& n\text{Px} = 1,
\& n\text{Py} = 1)

This sets up tiles with \( x \)-dimension of thirty-two grid points, \( y \)-dimension of thirty-two grid points, and \( x \) and \( y \) overlaps of three grid points each. There are six tiles allocated to six separate logical processors (\( n\text{Sx} = 6 \)). This set of values can be used for a cube sphere calculation. Each tile of size 32 \( \times \) 32 represents a face of the cube. Initializing the tile connectivity correctly (see Section 6.3.3.3. allows the rotations associated with moving between the six cube faces to be embedded within the tile-tile communication code.

### 6.3.2 Starting the code

When code is started under the WRAPPER, execution begins in a main routine `eesupp/src/main.F` that is owned by the WRAPPER. Control is transferred to the application through a routine called `model/src/the_model_main.F` once the WRAPPER has initialized correctly and has created the necessary variables to support subsequent calls to communication routines by the application code. The main stages of the WRAPPER startup calling sequence are as follows:

```
MAIN |
| --EEBOOT :: WRAPPER initialization |
| | |
| | -- EEBOOT_MINIMAL :: Minimal startup. Just enough to |
| | allow basic I/O. |
| | -- EEINTRO_MSG :: Write startup greeting. |
| |
| | -- EESET_PARMS :: Set WRAPPER parameters |
```

(continues on next page)
The steps above precede transfer of control to application code, which occurs in the procedure `the_main_model.F`

### 6.3.2.1 Multi-threaded execution

Prior to transferring control to the procedure `the_main_model.F` the WRAPPER may cause several coarse grain threads to be initialized. The routine `the_main_model.F` is called once for each thread and is passed a single stack argument which is the thread number, stored in the `myThid`. In addition to specifying a decomposition with multiple tiles per process (see Section 6.3.1) configuring and starting a code to run using multiple threads requires the following steps.

**Compilation**

First the code must be compiled with appropriate multi-threading directives active in the file `eesupp/src/main.F` and with appropriate compiler flags to request multi-threading support. The header files `eesupp/inc/MAIN_PDIRECTIVES1.h` and `eesupp/inc/MAIN_PDIRECTIVES2.h` contain directives compatible with compilers for Sun, Compaq, SGI, Hewlett-Packard SMP systems and CRAY PVP systems. These directives can be activated by using compile time directives `-DTARGET_SUN`, `-DTARGET_DEC`, `-DTARGET_SGI`, `-DTARGET_HP` or `-DTARGET_CRAY_VECTOR` respectively. Compiler options for invoking multi-threaded compilation vary from system to system and from compiler to compiler. The options will be described in the individual compiler documentation. For the Fortran compiler from Sun the following options are needed to correctly compile multi-threaded code

```
-stackvar -explicitpar -vpara -noautopar
```

These options are specific to the Sun compiler. Other compilers will use different syntax that will be described in their documentation. The effect of these options is as follows:

1. `-stackvar` Causes all local variables to be allocated in stack storage. This is necessary for local variables to ensure that they are private to their thread. Note, when using this option it may be necessary to override the default limit on stack-size that the operating system assigns to a process. This can normally be done by changing the settings of the command shell’s `stack-size`. However, on some systems changing this limit will require privileged administrator access to modify system parameters.

2. `-explicitpar` Requests that multiple threads be spawned in response to explicit directives in the application code. These directives are inserted with syntax appropriate to the particular target platform when, for example, the `-DTARGET_SUN` flag is selected.

3. `-vpara` This causes the compiler to describe the multi-threaded configuration it is creating. This is not required but it can be useful when troubleshooting.
4. **-noautopar** This inhibits any automatic multi-threaded parallelization the compiler may otherwise generate.

An example of valid settings for the *eedata* file for a domain with two subdomains in \( y \) and running with two threads is shown below

\[
\text{nTx=1, nTy=2}
\]

This set of values will cause computations to stay within a single thread when moving across the \( nSx \) sub-domains. In the \( y \)-direction, however, sub-domains will be split equally between two threads.

Despite its appealing programming model, multi-threaded execution remains less common than multi-process execution (described in Section 6.3.2.2). One major reason for this is that many system libraries are still not “thread-safe”. This means that, for example, on some systems it is not safe to call system routines to perform I/O when running in multi-threaded mode (except, perhaps, in a limited set of circumstances). Another reason is that support for multi-threaded programming models varies between systems.

### 6.3.2.2 Multi-process execution

Multi-process execution is more ubiquitous than multi-threaded execution. In order to run code in a multi-process configuration, a decomposition specification (see Section 6.3.1) is given (in which at least one of the parameters \( nPx \) or \( nPy \) will be greater than one). Then, as for multi-threaded operation, appropriate compile time and run time steps must be taken.

#### Compilation

Multi-process execution under the WRAPPER assumes that portable, MPI libraries are available for controlling the start-up of multiple processes. The MPI libraries are not required, although they are usually used, for performance critical communication. However, in order to simplify the task of controlling and coordinating the start up of a large number (hundreds and possibly even thousands) of copies of the same program, MPI is used. The calls to the MPI multi-process startup routines must be activated at compile time. Currently MPI libraries are invoked by specifying the appropriate options file with the `-of` flag when running the genmake2 script, which generates the Makefile for compiling and linking MITgcm. (Previously this was done by setting the \texttt{ALLOW\_USE\_MPI} and \texttt{ALWAYS\_USE\_MPI} flags in the \texttt{CPP\_EEOPTIONS.h} file.) More detailed information about the use of genmake2 for specifying local compiler flags is located in Section 3.5.2.

#### Execution

The mechanics of starting a program in multi-process mode under MPI is not standardized. Documentation associated with the distribution of MPI installed on a system will describe how to start a program using that distribution. For the open-source MPICH system, the MITgcm program can be started using a command such as

\[
\text{mpirun -np 64 -machinefile mf ./mitgcmuv}
\]

In this example the text `-np 64` specifies the number of processes that will be created. The numeric value 64 must be equal to (or greater than) the product of the processor grid settings of \( nPx \) and \( nPy \) in the file \texttt{SIZE.h}. The option `-machinefile mf` specifies that a text file called \texttt{mf} will be read to get a list of processor names on which the sixty-four processes will execute. The syntax of this file is specified by the MPI distribution.
6.3.2.3 Environment variables

On some systems multi-threaded execution also requires the setting of a special environment variable. On many machines this variable is called PARALLEL and its values should be set to the number of parallel threads required. Generally the help or manual pages associated with the multi-threaded compiler on a machine will explain how to set the required environment variables.

6.3.2.4 Runtime input parameters

Finally the file eedata needs to be configured to indicate the number of threads to be used in the x and y directions:

```
# Example "eedata" file
# Lines beginning "#" are comments
# nTx - No. threads per process in X
# nTy - No. threads per process in Y
&EEPARMS
  nTx=1,
  nTy=1,
&
```

The product of nTx and nTy must be equal to the number of threads spawned, i.e., the setting of the environment variable PARALLEL. The value of nTx must subdivide the number of sub-domains in x (nSx) exactly. The value of nTy must subdivide the number of sub-domains in y (nSy) exactly. The multi-process startup of the MITgcm executable mitgcmuv is controlled by the routines eeboot_minimal.F and ini_procs.F. The first routine performs basic steps required to make sure each process is started and has a textual output stream associated with it. By default two output files are opened for each process with names STDOUT.NNNN and STDERR.NNNN. The NNNNN part of the name is filled in with the process number so that process number 0 will create output files STDOUT.0000 and STDERR.0000, process number 1 will create output files STDOUT.0001 and STDERR.0001, etc. These files are used for reporting status and configuration information and for reporting error conditions on a process-by-process basis. The eeboot_minimal.F procedure also sets the variables myProcId and MPI_COMM_MODEL. These variables are related to processor identification and are used later in the routine ini_procs.F to allocate tiles to processes.

Allocation of processes to tiles is controlled by the routine ini_procs.F. For each process this routine sets the variables myXGlobalLo and myYGlobalLo. These variables specify, in index space, the coordinates of the southernmost and westernmost corner of the southernmost and westernmost tile owned by this process. The variables pidW, pidE, pidS and pidN are also set in this routine. These are used to identify processes holding tiles to the west, east, south and north of a given process. These values are stored in global storage in the header file EESUPPORT.h for use by communication routines. The above does not hold when the exch2 package is used. The exch2 package sets its own parameters to specify the global indices of tiles and their relationships to each other. See the documentation on the exch2 package for details.

6.3.3 Controlling communication

The WRAPPER maintains internal information that is used for communication operations and can be customized for different platforms. This section describes the information that is held and used.

1. **Tile-tile connectivity information** For each tile the WRAPPER sets a flag that sets the tile number to the north, south, east and west of that tile. This number is unique over all tiles in a configuration. Except when using the cubed sphere and the exch2 package, the number is held in the variables tileNo (this holds the tiles own number), tileNoN, tileNoS, tileNoE and tileNoW. A parameter is also stored with each tile that specifies the type of communication that is used between tiles. This information is held in the variables tileCommModeN, tileCommModeS, tileCommModeE and tileCommModeW. This latter set of variables can take one of the following values COMM_NONE, COMM_MSG, COMM_PUT and COMM_GET. A value of COMM_NONE is used to indicate that a tile has no neighbor to communicate with on a particular face. A value of COMM_MSG is used to indicate that some form of distributed memory communication is required to communicate between these tile faces (see Section 6.2.5.2).
A value of `COMM_PUT` or `COMM_GET` is used to indicate forms of shared memory communication (see Section 6.2.5.1). The `COMM_PUT` value indicates that a CPU should communicate by writing to data structures owned by another CPU. A `COMM_GET` value indicates that a CPU should communicate by reading from data structures owned by another CPU. These flags affect the behavior of the WRAPPER exchange primitive (see Figure 6.5). The routine `ini_communication_patterns.F` is responsible for setting the communication mode values for each tile.

When using the cubed sphere configuration with the `exch2 package`, the relationships between tiles and their communication methods are set by the `exch2 package` and stored in different variables. See the `exch2 package` documentation for details.

### 2. MP directives

The WRAPPER transfers control to numerical application code through the routine `the_model_main.F`. This routine is called in a way that allows for it to be invoked by several threads. Support for this is based on either multi-processing (MP) compiler directives or specific calls to multi-threading libraries (e.g., POSIX threads). Most commercially available Fortran compilers support the generation of code to spawn multiple threads through some form of compiler directives. Compiler directives are generally more convenient than writing code to explicitly spawn threads. On some systems, compiler directives may be the only method available. The WRAPPER is distributed with template MP directives for a number of systems.

These directives are inserted into the code just before and after the transfer of control to numerical algorithm code through the routine `the_model_main.F`. An example of the code that performs this process for a Silicon Graphics system is as follows:

```c
C--
C-- Parallel directives for MIPS Pro Fortran compiler
C--
C Parallel compiler directives for SGI with IRIX
C$PAR PARALLEL DO
C$PAR& CHUNK=1,MP_SCHEDTYPE=INTERLEAVE,
C$PAR& SHARE(nThreads),LOCAL(myThid,I)
C
DO I=1,nThreads
   myThid = I

C-- Invoke nThreads instances of the numerical model
CALL THE_MODEL_MAIN(myThid)
ENDDO

Prior to transferring control to the procedure `the_model_main.F` the WRAPPER may use MP directives to spawn multiple threads. This code is extracted from the files `main.F` and `eesupp/inc/MAIN_PDIRECTIVES1.h`. The variable `nThreads` specifies how many instances of the routine `the_model_main.F` will be created. The value of `nThreads` is set in the routine `ini_threading_environment.F`. The value is set equal to the the product of the parameters `nTx` and `nTy` that are read from the file `eedata`. If the value of `nThreads` is inconsistent with the number of threads requested from the operating system (for example by using an environment variable as described in Section 6.3.2.1) then usually an error will be reported by the routine `check_threads.F`.  

---

6.3. Using the WRAPPER 361
3. **memsync flags** As discussed in Section 6.2.5.1, a low-level system function may be need to force memory consistency on some shared memory systems. The routine `memsync.F` is used for this purpose. This routine should not need modifying and the information below is only provided for completeness. A logical parameter `exchNeedsMemSync` set in the routine `ini_communication_patterns.F` controls whether the `memsync.F` primitive is called. In general this routine is only used for multi-threaded execution. The code that goes into the `memsync.F` routine is specific to the compiler and processor used. In some cases, it must be written using a short code snippet of assembly language. For an Ultra Sparc system the following code snippet is used

```
asm("membar #LoadStore|#StoreStore");
```

For an Alpha based system the equivalent code reads

```
asm("mb");
```

while on an x86 system the following code is required

```
asm("lock; addl $0,0(%%esp): : :"memory")
```

4. **Cache line size** As discussed in Section 6.2.5.1, multi-threaded codes explicitly avoid penalties associated with excessive coherence traffic on an SMP system. To do this the shared memory data structures used by the `global_sum.F`, `global_max.F` and `barrier.F` routines are padded. The variables that control the padding are set in the header file `EEPARAMS.h`. These variables are called `cacheLineSize`, `lShare1`, `lShare4` and `lShare8`. The default values should not normally need changing.

5. **_BARRIER** This is a CPP macro that is expanded to a call to a routine which synchronizes all the logical processors running under the WRAPPER. Using a macro here preserves flexibility to insert a specialized call in-line into application code. By default this resolves to calling the procedure `barrier.F`. The default setting for the `_BARRIER` macro is given in the file `CPP_EEMACROS.h`.

6. **_GSUM** This is a CPP macro that is expanded to a call to a routine which sums up a floating point number over all the logical processors running under the WRAPPER. Using a macro here provides extra flexibility to insert a specialized call in-line into application code. By default this resolves to calling the procedure `GLOBAL_SUM_R8()` for 64-bit floating point operands or `GLOBAL_SUM_R4()` for 32-bit floating point operand (located in file `global_sum.F`). The default setting for the `_GSUM` macro is given in the file `CPP_EEMACROS.h`. The `_GSUM` macro is a performance critical operation, especially for large processor count, small tile size configurations. The custom communication example discussed in Section 6.3.3.2 shows how the macro is used to invoke a custom global sum routine for a specific set of hardware.

7. **_EXCH** The `_EXCH` CPP macro is used to update tile overlap regions. It is qualified by a suffix indicating whether overlap updates are for two-dimensional (`_EXCH_XY`) or three dimensional (`_EXCH_XYZ`) physical fields and whether fields are 32-bit floating point (`_EXCH_XY_R4`, `_EXCH_XYZ_R4`) or 64-bit floating point (`_EXCH_XY_R8`, `_EXCH_XYZ_R8`). The macro mappings are defined in the header file `CPP_EEMACROS.h`. As with `_GSUM`, the
_EXCH operation plays a crucial role in scaling to small tile, large logical and physical processor count configurations. The example in Section 6.3.3.2 discusses defining an optimized and specialized form on the _EXCH operation.

The _EXCH operation is also central to supporting grids such as the cube-sphere grid. In this class of grid a rotation may be required between tiles. Aligning the coordinate requiring rotation with the tile decomposition allows the coordinate transformation to be embedded within a custom form of the _EXCH primitive. In these cases _EXCH is mapped to exch2 routines, as detailed in the exch2 package documentation.

8. **Reverse Mode** The communication primitives _EXCH and _GSUM both employ hand-written adjoint forms (or reverse mode) forms. These reverse mode forms can be found in the source code directory pkg/autodiff. For the global sum primitive the reverse mode form calls are to GLOBAL_ADSUM_R4() and GLOBAL_ADSUM_R8() (located in file global_sum_ad.F). The reverse mode form of the exchange primitives are found in routines prefixed ADEXCH. The exchange routines make calls to the same low-level communication primitives as the forward mode operations. However, the routine argument theSimulationMode is set to the value REVERSE_SIMULATION. This signifies to the low-level routines that the adjoint forms of the appropriate communication operation should be performed.

9. **MAX_NO_THREADS** The variable MAX_NO_THREADS is used to indicate the maximum number of OS threads that a code will use. This value defaults to thirty-two and is set in the file EEPARAMS.h. For single threaded execution it can be reduced to one if required. The value is largely private to the WRAPPER and application code will not normally reference the value, except in the following scenario.

For certain physical parametrization schemes it is necessary to have a substantial number of work arrays. Where these arrays are allocated in heap storage (for example COMMON blocks) multi-threaded execution will require multiple instances of the COMMON block data. This can be achieved using a Fortran 90 module construct. However, if this mechanism is unavailable then the work arrays can be extended with dimensions using the tile dimensioning scheme of nSx and nSy (as described in Section 6.3.1). However, if the configuration being specified involves many more tiles than OS threads then it can save memory resources to reduce the variable MAX_NO_THREADS to be equal to the actual number of threads that will be used and to declare the physical parameterization work arrays with a single MAX_NO_THREADS extra dimension. An example of this is given in the verification experiment verification/aim.5l_cs. Here the default setting of MAX_NO_THREADS is altered to

```
INTEGER MAX_NO_THREADS
PARAMETER ( MAX_NO_THREADS = 6 )
```

and several work arrays for storing intermediate calculations are created with declarations of the form.

```
common /FORCIN/ sst1(ngp,MAX_NO_THREADS)
```

This declaration scheme is not used widely, because most global data is used for permanent, not temporary, storage of state information. In the case of permanent state information this approach cannot be used because there has to be enough storage allocated for all tiles. However, the technique can sometimes be a useful scheme for reducing memory requirements in complex physical parameterizations.
6.3.3.1 Specializing the Communication Code

The isolation of performance critical communication primitives and the subdivision of the simulation domain into tiles is a powerful tool. Here we show how it can be used to improve application performance and how it can be used to adapt to new gridding approaches.

6.3.3.2 JAM example

On some platforms a big performance boost can be obtained by binding the communication routines \_EXCH and \_GSUM to specialized native libraries (for example, the shmem library on CRAY T3E systems). The LETS_MAKE_JAM CPP flag is used as an illustration of a specialized communication configuration that substitutes for standard, portable forms of \_EXCH and \_GSUM. It affects three source files eboot.F, CPP_EEMACROS.h and cg2d.F. When the flag is defined is has the following effects.

- An extra phase is included at boot time to initialize the custom communications library (see ini_jam.F).
- The \_GSUM and \_EXCH macro definitions are replaced with calls to custom routines (see gsum_jam.F and exch_jam.F)
- A highly specialized form of the exchange operator (optimized for overlap regions of width one) is substituted into the elliptic solver routine cg2d.F.

Developing specialized code for other libraries follows a similar pattern.

6.3.3.3 Cube sphere communication

Actual \_EXCH routine code is generated automatically from a series of template files, for example exch2_rx1_cube.template. This is done to allow a large number of variations of the exchange process to be maintained. One set of variations supports the cube sphere grid. Support for a cube sphere grid in MITgcm is based on having each face of the cube as a separate tile or tiles. The exchange routines are then able to absorb much of the detailed rotation and reorientation required when moving around the cube grid. The set of \_EXCH routines that contain the word cube in their name perform these transformations. They are invoked when the run-time logical parameter useCubedSphereExchange is set .TRUE.. To facilitate the transformations on a staggered C-grid, exchange operations are defined separately for both vector and scalar quantities and for grid-centered and for grid-face and grid-corner quantities. Three sets of exchange routines are defined. Routines with names of the form exch2_rx are used to exchange cell centered scalar quantities. Routines with names of the form exch2_uv_rx are used to exchange vector quantities located at the C-grid velocity points. The vector quantities exchanged by the exch_uv_rx routines can either be signed (for example velocity components) or un-signed (for example grid-cell separations). Routines with names of the form exch_z_rx are used to exchange quantities at the C-grid vorticity point locations.

6.4 MITgcm execution under WRAPPER

Fitting together the WRAPPER elements, package elements and MITgcm core equation elements of the source code produces the calling sequence shown below.
6.4.1 Annotated call tree for MITgcm and WRAPPER

WRAPPER layer.

```
MAIN
|-- EEBOOT :: WRAPPER initialization
| |-- EEBOOT_MINIMAL :: Minimal startup. Just enough to allow basic I/O.
| |-- EEINTRO_MSG :: Write startup greeting.
| |-- EESET_PARMS :: Set WRAPPER parameters
| |-- EEWRITE_EENV :: Print WRAPPER parameter settings
| |.-- INI_PROCS :: Associate processes with grid regions.
| | |.-- INI_THREADING_ENVIRONMENT :: Associate threads with grid regions.
| | | | |-- INI_COMMUNICATION_PATTERNS :: Initialize between tile communication data structures
| | ||-- CHECK_THREADS :: Validate multiple thread start up.
| ||-- THE_MODEL_MAIN :: Numerical code top-level driver routine

Core equations plus packages.
```

C Invocation from WRAPPER level...
C
C | THE_MODEL_MAIN :: Primary driver for the MITgcm algorithm
C | | :: Called from WRAPPER level numerical code invocation routine. On entry
C | | :: to THE_MODEL_MAIN separate thread and separate processes will have been established.
C | | :: Each thread and process will have a unique ID
C | | :: but as yet it will not be associated with a specific region in decomposed discrete space.
C | | INITIALISE_FIXED :: Set fixed model arrays such as topography,
C | | | :: grid, solver matrices etc..
C | | INI_PARMS :: Routine to set kernel model parameters.
C | | | :: Kernel parameters are read from file "data"
C | | | :: in directory in which code executes.
C | | PACKAGES_BOOT :: Start up the optional package environment.
C | | | :: Runtime selection of active packages.
C | | CPL_IMPORT_CPLPARMS :: Import coupling parameters from/to the coupler layer
C | | PACKAGES_READPARMS :: Read each package input parameter file
```

(continues on next page)
$(\text{continued from previous page})$

C | | - ${\text{PKG}}$\text{-READPARMS}

C | | - SET_PARMS :: Finalise model parameter setting (if fct of pkg usage)

C | | -INI_MODEL_IO :: Initialise Input/Output setting

C | | | - MNC_INIT :: Initialise MITgcm NetCDF interface (MNC)(see pkg/mnc)
C | | | - MNC_CW_INIT :: Initialise MNC grid and variable types (see pkg/mnc)
C | | | - MON_INIT :: Initialises monitor package (see pkg/monitor)

C | | -INI_GRID :: Control grid array (vert. and horiz.) initialisation.

C | | | Grid arrays are held and described in GRID.h.
C | | | | - LOAD_GRID_SPACING :: Load grid spacing (vector) from files
C | | | | -INI_VERTICAL_GRID :: Set up vertical grid and coordinate
C | | | | | (calculate grid from kernel parameters).
C | | | | -INI_SPHERICAL_POLAR_GRID :: Spherical polar horiz. grid setting
C | | | | | (calculate grid from kernel parameters).
C | | | | -INI_CURVILINEAR_GRID :: General orthogonal, structured horiz. grid
C | | | | | initialisation; input from raw grid files
C | | | | | (LONC.bin, LATC.bin, DXF.bin, ... ) or per
C | | | | | face file: horizGridFile(.faceXXX.bin)
C | | | | -INI_CYLINDER_GRID :: Cylindrical horiz. grid setting

C | | - LOAD_REF_FILES :: Read-in reference vertical profiles (T,S,Rho)
C | | -INI_EOS :: Initialise Equation Of State (EOS) coefficients
C | | -SET_REF_STATE :: Set reference pressure/geopotential, reference
C | | | stratiﬁcation (for implicit IGW), vertical
C | | | velocity scaling factor and anelastic ref. density
C | | -SET_GRID_FACTORS :: Set grid factors (fct of k) for deep-atmosphere

C | | -INI_DEPTHS :: Read (from "bathyFile") or set bathymetry/orography.
C | | -INI_MASKS_ETC :: Derive horizontal and vertical cell fractions and
C | | | land masking for solid-fluid boundaries.

C | | - PACKAGES_INIT_FIXED :: do all packages fixed-initialisation setting
C | | | - ${\text{PKG}}$\text{-INIT_FIXED}

C | | -INI_GLOBAL_DOMAIN :: Initialise domain related (global) quantities.
C | | -INI_LINEAR_PHISURF :: Set ref. surface $\Phi_{surf}$

C | | | -INI_Cori :: Set coriolis term. zero, f-plane, beta-plane,
C | | | | sphere options are coded.
C | | | -INI_CG2D :: 2D conjugate grad solver initialisation.
C | | | -INI_CG3D :: 3D conjugate grad solver initialisation.

C | | | -CONFIG_SUMMARY :: Provide synopsis of kernel setup. Includes
C | | | | annotated table of kernel parameter settings.

C | | | -PACKAGES_CHECK :: call each package configuration checking S/R
C | | | | - ${\text{PKG}}$\text{-CHECK}

C | | | -CONFIG_CHECK :: Check config and parameter consistency.

(continues on next page)
C | | WRITE_GRID :: write grid fields to output files
C | | CPL_EXCH_CONFIGS :: exchange config with coupler-interface
C | | CTRL_UNPACK :: Control vector support package. see pkg/ctrl
C | | COST_DEPENDENT_INIT :: ( see pkg/cost )
C | | ADTHE>Main_LOOP :: Derivative evaluating form of main time stepping loop
C ! :: Automatically generated by TAMC/TAF.
C | | THE_MAIN_LOOP :: Main timestepping loop routine.
C | | INITIALISE_VARIA :: Set the initial conditions for time evolving fields
C #ifdef ALLOW_AUTODIFF
C | | INI_DEPTH :: Initialise all Non-Lin Free-Surf arrays (SURFACE.h)
C | | DYNVARS :: Initialise to zero all DYNVARS.h arrays
C | | NH_VARS :: Initialise to zero all NH_VARS.h arrays
C | | FFIELDS :: Initialise forcing fields in FFIELDS.h to zero
C | | FIELDS :: Control initialising model fields to non-zero
C | | VEL :: Initialize 3D flow field.
C | | THETA :: Set model initial temperature field.
C | | SALT :: Set model initial salinity field.
C | | PSURF :: Set model initial free-surface height/pressure.
C | | READ_PICKUP :: Read in main model pickup files to restart a run.
C | | MIXING :: Initialise diapycnal diffusivity.
C | | TAUEDDY_INIT_VARIA :: Initialise eddy (bolus) streamfunction
C | | FORCING :: Set model initial forcing fields, either
C | | :: set in-line or from file as shown here:
C | | READ_FLD_XY_RS(zonalWindFile)
C | | READ_FLD_XY_RS(meridWindFile)
C | | READ_FLD_XY_RS(surfQnetFile)
C | | READ_FLD_XY_RS(EmPmRfile)
C | | READ_FLD_XY_RS(thetaClimFile)
C | | READ_FLD_XY_RS(saltClimFile)
C | | READ_FLD_XY_RS(surfQswFile)
C | | AUTODIFF_INIT_VARIA :: (see pkg/autodiff )
C | | PACKAGES_INIT_VARIABLES :: Does initialisation of time evolving
C | | | $PKG_INIT_VARIA :: package data.
C | | COST_INIT_VARIA :: ( see pkg/cost )

(continues on next page)
C | |-CONVECTIVE_ADJUSTMENT_INI :: Apply conv. adjustment to initial state
C | |-CALC_R_STAR :: Calculate the new level thickness factor (r* coord)
C | |-UPDATE_R_STAR :: Update the level thickness fraction (r* coord).
C | |-UPDATE_SIGMA :: Update the level thickness fraction (sigma-coord).
C | |-CALC_SURF_DR :: Calculate the new surface level thickness.
C | |-UPDATE_SURF_DR :: Update the surface-level thickness fraction.
C | |-UPDATE_R_STAR :: Calculate the new level thickness factor (r* coord)
C | |-CALC_SURF_DR :: Calculate the new surface level thickness.
C | |-STATE_SUMMARY :: Summarise model prognostic variables.
C | |-MONITOR :: Monitor state (see pkg/monitor)
C | |-DO_STATEVARS_TAVE :: Time averaging package ( see pkg/timeave ).
C | |-TIMEAVE_STATVARS :: Accumulate main model state variables
C | |-PTRACERS_TIMEAVE :: Accumulate passive tracers variables
C | |-DO_THE_MODEL_IO :: Controlling routine for IO
C | |-WRITE_STATE :: Write model state variables.
C | |-TIMEAVE_STATVARS_WRITE :: Write Time averaged output (see pkg/timeave)
C | |-FIZHI_WRITE_STATE :: Write Fizhi pkg output (see pkg/fizhi)
C | |-AIM_WRITE_TAVE :: Write AIM pkg output (see pkg/aim_v23)
C | |-LAND_OUTPUT :: Write Land pkg output (see pkg/land)
C | |-OBCS_OUTPUT :: Write OBCS pkg output (see pkg/obcs)
C | |-GMREDI_OUTPUT :: Write GM-Redi pkg output (see pkg/gmredi)
C | |-KPP_OUTPUT :: Write KPP pkg output (see pkg/kpp)
C | |-PP81_OUTPUT :: Write PP81 pkg output (see pkg/pp81)
C | |-KL10_OUTPUT :: Write KL10 pkg output (see pkg/kl10)
C | |-MY82_OUTPUT :: Write MY82 pkg output (see pkg/my82)
C | |-OPPS_OUTPUT :: Write OPPS pkg output (see pkg/opps)
C | |-GGL90_OUTPUT :: Write GGL90 pkg output (see pkg/ggl90)
C | |-SBO_CALC :: Compute SBO diagnostics (see pkg/sbo)
C | |-SBO_OUTPUT :: Write SBO pkg output (see pkg/sbo)
C | |-SEAICE_OUTPUT :: Write SeaIce pkg output (see pkg/seaice)
C | |-SHELFICE_OUTPUT :: Write ShelfIce pkg output (see pkg/shelfice)
C | |-BULKF_OUTPUT :: Write Bulk-Force output (see pkg/bulkf_force)
C | |-THSICE_OUTPUT :: Write ThSice pkg output (see pkg/thsice)
C | |-PTRACERS_OUTPUT :: Write pTracers pkg output (see pkg/ptracers)
C | |-MATRIX_OUTPUT :: Write Matrix pkg output (see pkg/matrix)
C | |-GCHEN_OUTPUT :: Write Geochemistry pkg output (see pkg/gchem)
C | |-CPL_OUTPUT :: Write Coupler-Interface output (see pkg/atm_compon_interf, pkg/ocn_compon_interf)
C | | | |-LAYERS_CALC :: Calculate layers diagnostics (see pkg/layers)
C | | | |-LAYERS_OUTPUT :: Write Layers pkg output (see pkg/layers)
C | | | |-DIAGNOSTICS_WRITE :: Write pkg/diagnostics output
C |

C====|>| ****************************
C====|>| BEGIN MAIN TIMESTEPPING LOOP
C====|>| ****************************
C | |-COST_AVERAGESFIELDS :: time-averaged Cost function terms (see pkg/cost)
C | |-PROFILES_INLOOP :: ( see pkg/profiles )
C | /
C | |-MAIN_DO_LOOP :: Open-AD case: Main timestepping loop routine
C | | otherwise: just call FORWARD_STEP
C | /
C\ | |-FORWARD_STEP :: Step forward a time-step ( AT LAST !!! )
C\ | |
C\ | |-AUTODIFF_INADMODE_UNSET :: Set/reset some adjoint flags
C\ | |-RESET_NLFS_VARS :: Reset some Non-Lin Free-Surf vars (Adjoint)
C\ | |-UPDATE_R_STAR :: Reset r-star factor variables (Adjoint)
C\ | |-UPDATE_SURF_DR :: Reset NLFS surface thickness vars (Adjoint)
C\ | |
C\ | |-PTRACERS_SWITCH_ONOFF :: Set/reset pTracers time-stepping switch
C\ | |-DIAGNOSTICS_SWITCH_ONOFF :: Activate/de-activate diagnostics
C\ | |-DO_STATEVARS_DIAGS ( 0 ) :: fill-up state variable diagnostics
C\ | |
C\ | |-NEST_CHILD_SETMEMO :: Nesting interface
C\ | |-NEST_PARENT_IO_1 :: Nesting interface
C\ |
C\ | |-LOAD_FIELDS_DRIVER :: Control loading of input fields from files
C\ |
C\ | |-BULKF_FORCING :: Calculate surface forcing (see pkg/bulk_force)
C\ | |-CHEAPAML :: Cheap AML driver ( see pkg/cheapaml )
C\ | |-CTRL_MAP_FORCING :: Control vector support package. (see pkg/ctrl)
C\ | |-DUMMY_IN_STEPPING :: Autodiff package ( pkg/autodiff ).
C\ |
C\ | |-CPL_EXPORT_MY_DATA :: Send coupling fields to coupler
C\ | |-CPL_IMPORT_EXTERNAL_DATA :: Receive coupling fields from coupler
C\ |
C\ | |-OASIS_PUT :: Oasis coupler interface
C\ | |-OASIS_GET :: Oasis coupler interface
C\ |
C\ | |-EBM_DRIVER :: Calculate EBM type atmospheric forcing (see pkg/ebm)
C\ |
C\ | |-DO_ATMOSPHERIC_PHYS :: Atmospheric physics computation
C\ |
C\ | |-UPDATE_OCEAN_EXPORTS :: ( see pkg/fizhi )
C\ | |-UPDATE_EARTH_EXPORTS :: ( see pkg/fizhi )
C\ | |-UPDATE_CHEMISTRY_EXPORTS :: ( see pkg/fizhi )
C\ | |-FIZHI_WRAPPER :: ( see pkg/fizhi )
C\ | |-STEP_FIZHI_FG :: ( see pkg/fizhi )
C\ | |-FIZHI_UPDATE_TIME :: ( see pkg/fizhi )
C\ |
C\ | |-ATM_PHYS_DRIVER :: ( see pkg/atm_phys )

(continues on next page)
C/ | | | |-AIM_DO_PHYSICS :: ( see pkg/aim_v23 )
C/ | | | |-DO_OCEANIC_PHYS :: Oceanic (& seaice) physics computation
C/ | | | |-OBCS_CALC :: Open boundary. package (see pkg/obcs).
C/ | | | |-FRAZIL_CALC_RHS :: Compute FRAZIL tendencies ( see pkg/frazil )
C/ | | | |-THSICE_MAIN :: Thermodynamic sea-ice driver (see pkg/thsice)
C/ | | | |-SEAICE_MODEL :: Sea-ice model driver (see pkg/seaice )
C/ | | | |-SEAICE_COST_SENSI :: Sea-ice cost-function (see pkg/seaice )
C/ | | | |-SHELFICE_THERMODYNAMICS :: Compute ShelfIce thermo (pkg/shelfice)
C/ | | | |-ICEFRONT_THERMODYNAMICS :: Compute IceFront thermo (pkg/icefront)

C/ | | | |-SALT_PLUME_DO_EXCH :: (see pkg/salt_plume )
C/ | | | |-FREEZE_SURFACE :: Prevent SST to fall below TFreeze
C/ | | | |-OCN_APPLY_IMPORT :: Apply imported fields from coupler
C/ | | | |-EXTERNAL_FORCING_SURF:: Compute appropriately dimensioned
C/ | | | | :: surface forcing terms.
C/ | | | |-FIND_RHO_2D @ p(k) :: Calculate \([\rho(T,S,p)-\rho_0]\) of a slice
C/ | | | |-FIND_RHO_2D @ p(k-1) :: Calculate \([\rho(T,S,p)-\rho_0]\) of a slice
C/ | | | |-GRAD_SIGMA :: Calculate isoneutral gradients
C/ | | | |-CALC_IVDC :: Set Implicit Vertical Diffusivity for Convection
C/ | | | |-CALC_OCE_MXLAYER :: Diagnose Oceanic Mixed Layer depth

C/ | | | |-SALT_PLUME_CALC_DEPTH :: (see pkg/salt_plume )
C/ | | | |-SALT_PLUME_VOLFRAC :: (see pkg/salt_plume )
C/ | | | |-SALT_PLUME_APPLY (Temp) :: (see pkg/salt_plume )
C/ | | | |-SALT_PLUME_APPLY (Salt) :: (see pkg/salt_plume )
C/ | | | |-SALT_PLUME_FORCING_SURF :: (see pkg/salt_plume )
C/ | | | |-KPP_CALC :: Compute KPP vertical mixing ( see pkg/kpp )
C/ | | | |-PP81_CALC :: Compute PP81 vertical mixing ( see pkg/pp81 )
C/ | | | |-KL10_CALC :: Compute KL10 vertical mixing ( see pkg/kl10 )
C/ | | | |-MY82_CALC :: Compute MY82 vertical mixing ( see pkg/my82 )
C/ | | | |-GGL90_CALC :: Compute GGL90 vertical mixing (see pkg/ggl90)
C/ | | | |-GMREDI_CALC_TENSOR :: Compute GM-Redi tensor ( see pkg/gmredi )
C/ | | | |-DWNSLP_CALC_FLOW :: Compute Down-Slope flow (see pkg/down_slope)
C/ | | | |-BBL_CALC_RHS :: Compute BBL tendencies ( see pkg/bbl )
C/ | | | |-MYPACKAGE_CALC_RHS :: Compute mypackage tendencies (pkg/mypackage)

C/ | | | |-GMREDI_DO_EXCH :: ( see pkg/gmredi )
C/ | | | |-KPP_DO_EXCH :: ( see pkg/kpp )
C/ | | | |-DIAGS_RHO_G :: Compute some density related diagnostics
C/ | | | |-DIAGS_OCEANIC_SURF_FLUX :: Diagnose oceanic surface fluxes
C/ | | | |-SALT_PLUME_DIAGNOSTICS_FILL :: (see pkg/salt_plume )
C/ | | | |-ECCO_PHYS :: ( see pkg/ecco )

C/ | | | |-STREAMICE_TIMESTEP :: ( see pkg/streamice )
C/ | | | |-GCHEM_CALC_TENDENCY :: geochemistry driver routine (see pkg/gchem)
C/ | | | |-LONGSTEP_AVERAGE :: Averaging state vars ( see pkg/longstep )

(continues on next page)
6.4. MITgcm execution under WRAPPER

(continued from previous page)

C/  |  |-LONGSTEP_THERMODYNAMICS :: Step forward tracers (see pkg/longstep)
C/  |  
C/  |  |-THERMODYNAMICS       :: theta, salt + tracer equations driver.
C/  |  |  (synchronous time-stepping case)
C/  |  |  |-CALC_WSURF_TR        :: Compute T & S Linear-Free-Surf correction
C/  |  |  |-PTRACERS_CALC_WSURF_TR :: Compute Tracers Linear-Free-Surf correct.
C/  |  |
C/  |  |-GMREDI_RESIDUAL_FLOW :: Get the flow field used to advect tracers
C/  |  |
C/  |  |-TEMP_INTEGRATE       :: Step forward Prognostic Eq for Temperature.
C/  |  |
C/  |  |  |-ADAMS_BASHFORTH3  :: Extrapolate tracer forward in time (AB-3)
C/  |  |  |-ADAMS_BASHFORTH2  :: Extrapolate tracer forward in time (AB-2)
C/  |  |
C/  |  |-GAD_SOM_ADVECT       :: Second Order Moment (SOM) advection
C/  |  |
C/  |  |-GAD_ADVECTION        :: Generalised advection driver (multi-dim advection case) (see pkg/gad).
C/  |  |
C/  |  |-CALC_ADV_FLOW        :: set 3-D flow field to advect tracer
C/  |  |
C/  |  |-APPLY_FORCING_T      :: Problem specific forcing for temperature.
C/  |  |
C/  |  |-GAD_CALC_RHS         :: Calculate Advection-Diffusion tendency terms
C/  |  |
C/  |  |  |-ADAMS_BASHFORTH3  :: Extrapolate tendency forward in time (AB-3)
C/  |  |  |-ADAMS_BASHFORTH2  :: Extrapolate tendency forward in time (AB-2)
C/  |  |
C/  |  |  |-FREESURF_RESCALE_G :: Re-scale Gt for free-surface height.
C/  |  |
C/  |  |-DWNSLP_APPLY         :: Add pkg/down_slope tendency
C/  |  |
C/  |  |-TIMESTEP_TRACER      :: Step tracer field forward in time
C/  |  |
C/  |  |-GAD IMPLICIT_R       :: Solve vertical implicit Advect-Diffus. eqn.
C/  |  |
C/  |  |-IMPLDIFF             :: Solve vertical implicit diffusion equation.
C/  |  |
C/  |  |-CYCLE_AB_TRACER      :: Cycle time-stepping arrays for tracer field
C/  |  |
C/  |  |-CYCLE_TRACER         :: Cycle time-stepping arrays for tracer field
C/  |
C/  |  |-SALT_INTEGRATE      :: Step forward Prognostic Eq for Salinity.
C/  |  |  same sequence of calls as in TEMP_INTEGRATE
C/  |
C/  |  |-PTRACERS_INTEGRATE :: Integrate other tracer(s) (see pkg/ptracers).
C/  |  |  same sequence of calls as in TEMP_INTEGRATE
C/  |
C/  |  |-OBCS APPLY PTRACER   :: Open boundary package for pTracers
C/  |
C/  |  |-OBCS APPLY TS       :: Open boundary package (see pkg/obcs).
C/  |
C/  |  |-LONGSTEP_AVERAGE    :: Averaging state vars (see pkg/longstep)
C/  |
C/  |  |-LONGSTEP THERMODYNAMICS :: Step forward tracers (see pkg/longstep)
C/  |
C/  |  |-DO STAGGER FIELDS EXCHANGES :: Update overlap regions of arrays
C/  |  |  Theta & Salt (implicit IGW case)
C/  |
C/  |  |-DYNAMICS            :: Momentum equations driver.
C/  |
C/  |  |-CALC GRAD PHI SURF :: Calculate the gradient of the surface

(continues on next page)
Potential anomaly.

- **CALC_VISCOSITY** :: Calculate net vertical viscosity
- **MOM_CALC_3D_STRAIN** :: Calculates the strain tensor of 3D flow field
- **OBCS_COPY_UV_N** :: for Stevens boundary Conditions (see pkg/obcs)
- **CALC_PHI_HYD** :: Integrate the hydrostatic relation.
- **MOM_FLUXFORM** :: Flux Form momentum eqn. (pkg/mom_fluxform)
- **MOM_VECINV** :: Vector Invariant momentum eqn (pkg/mom_vecinv)
- **MOM_CALC_SMAG_3D** :: Calculate Smagorinsky 3D (harmonic) viscosities
- **MOM_UV_SMAG_3D** :: Calculate U,V mom. tendency due to Smag 3D Visc
- **TIMESTEP** :: Step horizontal momentum fields forward in time
- **MOM_U_IMPLICIT_R** :: Solve implicitly vertical Adv-Diffus equation.
- **IMPLDIFF** :: Solve vertical implicit diffusion equation.
- **OBCS_SAVE_UV_N** :: for Stevens boundary Conditions (see pkg/obcs)
- **OBCS_APPLY_UV** :: Apply Open boundary Conditions to provisional U,V
- **IMPLDIFF** :: (CD-Scheme) Solve vertical impl. diffus. eqn
- **MOM_GW** :: Vert. momentum tendency terms (Non-Hydrostatic)
- **MOM_W_SMAG_3D** :: Calculate W mom. tendency due to Smag 3D Visc
- **TIMESTEP_WVEL** :: Step vert mom forward in time (Non-Hydrostatic)
- **MNC_UPDATE_TIME** :: Update MNC time record (see pkg/mnc)
- **UPDATE_R_STAR** :: Update the level thickness fraction (r* coord).
- **UPDATE_SIGMA** :: Update the level thickness fraction (sigma-coord).
- **UPDATE_R_STAR** :: Update the level thickness fraction.
- **UPDATE_SURF_DR** :: Update the surface-level thickness fraction.
- **UPDATE_CG2D** :: Update 2D conjugate grad. for Free-Surf.
- **SHAP_FILT_APPLY_UV** :: Apply Shapiro Filter to provisional velocity
- **ZONAL_FILT_APPLY_UV** :: Apply Zonal Filter to provisional velocity
- **SOLVE_FOR_PRESSURE** :: Find surface pressure.
- **CALC_DIV_GHAT** :: Form the RHS of the surface pressure eqn.
- **CG2D** :: Two-dim pre-con. conjugate-gradient.
- **PREF.CG3D** :: Finish to set the RHS of the 3-D pressure eqn.
- **CG3D** :: Three-dim pre-con. conjugate-gradient solver.
- **POST.CG3D** :: finalise solution of NH and Free-Surf pressure
- **MOMENTUM_CORRECTION_STEP** :: Finalise momentum stepping
- **CALC_GRAD_PHI_SURF** :: Return DDx and DDy of surface pressure
- **CORRECTION_STEP** :: Pressure correction to momentum
- **OBCS_APPLY_UV** :: Open boundary package (see pkg/obcs).
- **SHAP_FILT_APPLY_UV** :: Apply Shapiro Filter to latest velocity
- **ZONAL_FILT_APPLY_UV** :: Apply Zonal Filter to latest velocity
- **INTEGR_CONTINUITY** :: Integrate continuity equation (see above)
- **CALC_R_STAR** :: Calculate the new level thickness factor (r* coord)
- **CALC_SURF_DR** :: Calculate the new surface level thickness.
C/ | | |-DO_STAGGER_FIELDS_EXCHANGES :: Update overlap regions of arrays
C/ | | uVel,vVel & wVel (stagger-time-step case)
C/ | |
C/ | | |-DO_STATEVARS_DIAGS ( 1 ) :: fill-up state variable diagnostics
C/ | |
C/ | | |-THERMODYNAMICS :: theta, salt + tracer Eq. driver (see above).
C/ | |
C/ | | (staggered time-stepping case)
C/ | |
C/ | | |-TRACERS_CORRECTION_STEP :: Finalise tracer stepping:
C/ | | :: apply filter, conv.adjustment
C/ | | |-TRACERS_IIGW_CORRECTION :: apply Implicit IGW adjustment to T & S
C/ | | |-SHAP_FILT_APPLY_TS :: Apply Shapiro Filter to latest T & S
C/ | | |-ZONAL_FILT_APPLY_TS :: Apply Zonal Filter to latest T & S
C/ | | | |-PTRACERS_ZONAL_FILT_APPLY :: Apply Zonal Filter to pTracers
C/ | | |-SALT_FILL :: Fill up negative Salt
C/ | | |-OPPS_INTERFACE :: ( see pkg/opps )
C/ | | |-CONVective_ADJUSTMENT :: Apply convective adjustment
C/ | | |-MATRIX_STORE_TENDENCY_IMP :: ( see pkg/matrix )
C/ | |
C/ | | |-LONGSTEP_AVERAGE :: Averaging state vars ( see pkg/longstep )
C/ | | |-LONGSTEP_THERMODYNAMICS :: Step forward tracers ( see pkg/longstep )
C/ | |
C/ | | |-GChem_forcing_SEP :: Tracer forcing for gchem pkg (if tracer
C/ | | | dependent tendencies calculated separately)
C/ | |
C/ | | |-DO_FIELDS_BLOCKING_EXCHANGES :: Sync up overlap regions.
C/ | |
C/ | | |-DO_STATEVARS_DIAGS ( 2 ) :: fill-up state variable diagnostics
C/ | |
C/ | | |-GRIDALT_UPDATE :: ( see pkg/gridalt )
C/ | | |-STEP_FIZHI_CORR :: ( see pkg/fizhi )
C/ | |
C/ | | |-FLT_MAIN :: Step forward Floats (see pkg/flt)
C/ | |
C/ | | |-DO_STATEVARS_TAVE :: Time averaging package (see above)
C/ | |
C/ | | |-NEST_PARENT_IO_2 :: Nesting interface
C/ | | |-NEST_CHILD_TRANSP :: Nesting interface
C/ | |
C/ | | |-MONITOR :: Monitor package (pkg/monitor).
C/ | |
C/ | | |-COST_TILE :: ( see pkg/cost )
C/ | |
C/ | | |-DO_THE_MODEL_IO :: Controlling routine for IO (see above)
C/ | |
C/ | | |-PTRACERS_RESET :: Re-initialize PTRACERS ( see pkg/ptracers )
C/ | |
C/ | | |-DO_WRITE_PICKUP :: Controlling routine for writing files to restart
C/ | | |-PACKAGES_WRITE_PICKUP :: Write pickup files for each package
C/ | | |- :: which needs it to restart
C/ | | |-GAD_WRITE_PICKUP :: Write Generic AdvDiff pickups for SOM
C/ | | |- :: advection scheme (pkg/generic_advdiff)

(continues on next page)
C/| | | | |-CD_CODE_WRITE_PICKUP :: Write CD-code pickups (see pkg/cd_code)
C/| | | | |-OBCS_WRITE_PICKUP :: Write OBCS pickups (see pkg/obcs)
C/| | | | |-GGL90_WRITE_PICKUP :: Write GGL90 pickups (see pkg/ggl90)
C/| | | | |-BBL_WRITE_PICKUP :: Write BBL pickups (see pkg/bbl)
C/| | | | |-CHEAPAML_WRITE_PICKUP :: Write CheapAML pickups (pkg/cheapaml)
C/| | | | |-FLT_WRITE_PICKUP :: Write Floats pickups (see pkg/flt)
C/| | | | |-PTRACERS_WRITE_PICKUP :: Write pTracers pickups (pkg.ptracers)
C/| | | | |-GCHEM_WRITE_PICKUP :: Write Geo-Chem pickups (see pkg/gchem)
C/| | | | |-SEAICE_WRITE_PICKUP :: Write SeaIce pickups (see pkg/seaice)
C/| | | | |-STREAMICE_WRITE_PICKUP :: Write StreamIce pickups (pkg/streamice)
C/| | | | |-SHELFICE_WRITE_PICKUP :: Write ShelfIce pickups (pkg/shelfice)
C/| | | | |-THSICE_WRITE_PICKUP :: Write ThsIce pickups (see pkg/thsice)
C/| | | | |-LAND_WRITE_PICKUP :: Write Land pickups (see pkg/land)
C/| | | | |-ATM_PHYS_WRITE_PICKUP :: Write Atm-Phys pickups (pkg/atm_phys)
C/| | | | |-FIZHI_WRITE_PICKUP :: Write Fizhi pickups (see pkg/fizhi)
C/| | | | |-FIZHI_WRITE_VEGTILES :: Write Fizhi VegTiles (see pkg/fizhi)
C/| | | | |-FIZHI_WRITE_DATETIME :: Write Fizhi DateTime (see pkg/fizhi)
C/| | | | |-CTRL_PACK :: Control vector support package (see pkg/ctrl)
C/| | | | |-GRDCHK_MAIN :: Gradient check package (see pkg/grdchk)
C/| | | | |-TIMER_PRINTALL :: Computational timing summary
C/| | | | |-COMM_STATS :: Summarise inter-proc and inter-thread communication
C/ | | | | |-WRITE_PICKUP :: Write main model pickup files.
C/ | | | | |-AUTODIFF_INADMODE_SET :: Set/reset some adjoint flags
C/| | | | | **********************************************
C/| | | | | END MAIN TIMESTEPPING LOOP
C/| | | | | **********************************************
C/ |
C | |-COST_AVERAGESFIELDS :: Time-averaged Cost function terms (see pkg/cost)
C/ | |-PROFILES_INLOOP :: ( see pkg/profiles )
C/ | |-COST_FINAL :: Cost function package (see pkg/cost)
C/ |
C | |-CTRL_PACK :: Control vector support package (see pkg/ctrl)
C |
C | |-GRDCHK_MAIN :: Gradient check package (see pkg/grdchk)
C |
C | |-TIMER_PRINTALL :: Computational timing summary
C |
C | |-COMM_STATS :: Summarise inter-proc and inter-thread communication
C | :: events.
6.4.2 Measuring and Characterizing Performance

TO BE DONE (CNH)

6.4.3 Estimating Resource Requirements

TO BE DONE (CNH)

6.4.3.1 Atlantic 1/6 degree example

6.4.3.2 Dry Run testing

6.4.3.3 Adjoint Resource Requirements

6.4.3.4 State Estimation Environment Resources
Automatic differentiation (AD), also referred to as algorithmic (or, more loosely, computational) differentiation, involves automatically deriving code to calculate partial derivatives from an existing fully non-linear prognostic code (see Griewank and Walther, 2008 [GW08]). A software tool is used that parses and transforms source files according to a set of linguistic and mathematical rules. AD tools are like source-to-source translators in that they parse a program code as input and produce a new program code as output (we restrict our discussion to source-to-source tools, ignoring operator-overloading tools). However, unlike a pure source-to-source translation, the output program represents a new algorithm, such as the evaluation of the Jacobian, the Hessian, or higher derivative operators. In principle, a variety of derived algorithms can be generated automatically in this way.

MITgcm has been adapted for use with the Tangent linear and Adjoint Model Compiler (TAMC) and its successor TAF (Transformation of Algorithms in Fortran), developed by Ralf Giering (Giering and Kaminski, 1998 [GK98], Giering, 2000 [Gie00]). The first application of the adjoint of MITgcm for sensitivity studies was published by Marotzke et al. (1999) [MGZ+99]. Stammer et al. (1997, 2002) [SWG+97] [SWG+02] use MITgcm and its adjoint for ocean state estimation studies. In the following we shall refer to TAMC and TAF synonymously, except were explicitly stated otherwise.

As of mid-2007 we are also able to generate fairly efficient adjoint code of the MITgcm using a new, open-source AD tool, called OpenAD (see Naumann, 2006 [NUH+06] and Utke et al., 2008 [UNF+08]). This enables us for the first time to compare adjoint models generated from different AD tools, providing an additional accuracy check, complementary to finite-difference gradient checks. OpenAD and its application to MITgcm is described in detail in Section 7.5.

The AD tool exploits the chain rule for computing the first derivative of a function with respect to a set of input variables. Treating a given forward code as a composition of operations – each line representing a compositional element, the chain rule is rigorously applied to the code, line by line. The resulting tangent linear or adjoint code, then, may be thought of as the composition in forward or reverse order, respectively, of the Jacobian matrices of the forward code’s compositional elements.

### 7.1 Some basic algebra

Let $\mathcal{M}$ be a general nonlinear, model, i.e., a mapping from the $m$-dimensional space $U \subset \mathbb{R}^m$ of input variables $\vec{u} = (u_1, \ldots, u_m)$ (model parameters, initial conditions, boundary conditions such as forcing functions) to the $n$-dimensional space $V \subset \mathbb{R}^n$ of model output variable $\vec{v} = (v_1, \ldots, v_n)$ (model state, model diagnostics, objective function, ...) under consideration:

$$\mathcal{M} : U \longrightarrow V$$

$$\vec{u} \mapsto \vec{v} = \mathcal{M}(\vec{u})$$

(7.1)
The vectors $\vec{u} \in U$ and $\vec{v} \in V$ may be represented with respect to some given basis vectors $\text{span}(U) = \{\vec{e}_i\}_{i=1,...,m}$ and $\text{span}(V) = \{\vec{f}_j\}_{j=1,...,n}$ as

$$\vec{u} = \sum_{i=1}^{m} u_i \vec{e}_i, \quad \vec{v} = \sum_{j=1}^{n} v_j \vec{f}_j$$

Two routes may be followed to determine the sensitivity of the output variable $\vec{v}$ to its input $\vec{u}$.

### 7.1.1 Forward or direct sensitivity

Consider a perturbation to the input variables $\delta \vec{u}$ (typically a single component $\delta u_i \vec{e}_i$). Their effect on the output may be obtained via the linear approximation of the model $\mathcal{M}$ in terms of its Jacobian matrix $M$, evaluated in the point $u^{(0)}$ according to

$$\delta \vec{v} = M|_{\vec{u}^{(0)}} \delta \vec{u}$$

(7.2) is the tangent linear model (TLM). In contrast to the full nonlinear model $\mathcal{M}$, the operator $M$ is just a matrix which can readily be used to find the forward sensitivity of $\vec{v}$ to perturbations in $\vec{u}$, but if there are very many input variables ($\gg O(10^6)$ for large-scale oceanographic application), it quickly becomes prohibitive to proceed directly as in (7.2), if the impact of each component $e_i$ is to be assessed.

### 7.1.2 Reverse or adjoint sensitivity

Let us consider the special case of a scalar objective function $\mathcal{J}(\vec{v})$ of the model output (e.g., the total meridional heat transport, the total uptake of CO$_2$ in the Southern Ocean over a time interval, or a measure of some model-to-data misfit)

$$\mathcal{J} : \quad U \rightarrow V \rightarrow \mathbb{R}$$

(7.4)

The perturbation of $\mathcal{J}$ around a fixed point $\mathcal{J}_0$,

$$\mathcal{J} = \mathcal{J}_0 + \delta \mathcal{J}$$

can be expressed in both bases of $\vec{u}$ and $\vec{v}$ with respect to their corresponding inner product $\langle \cdot , \cdot \rangle$

$$\mathcal{J} = \mathcal{J}|_{\vec{u}^{(0)}} + \langle \nabla_u \mathcal{J}^T|_{\vec{u}^{(0)}} , \delta \vec{u} \rangle + O(\delta \vec{u}^2)$$

$$= \mathcal{J}|_{\vec{v}^{(0)}} + \langle \nabla_v \mathcal{J}^T|_{\vec{v}^{(0)}} , \delta \vec{v} \rangle + O(\delta \vec{v}^2)$$

(7.5)

(note, that the gradient $\nabla f$ is a co-vector, therefore its transpose is required in the above inner product). Then, using the representation of $\delta \mathcal{J} = \langle \nabla_v \mathcal{J}^T , \delta \vec{v} \rangle$, the definition of an adjoint operator $A^*$ of a given operator $A$,

$$\langle A^* \vec{x} , \vec{y} \rangle = \langle \vec{x} , A \vec{y} \rangle$$

which for finite-dimensional vector spaces is just the transpose of $A$,

$$A^* = A^T$$
and from (7.2), (7.5), we note that (omitting |’s):
\[
\delta \mathcal{J} = \langle \nabla_v \mathcal{J}^T, \delta \vec{v} \rangle = \langle \nabla_v \mathcal{J}^T, M \delta \vec{u} \rangle = \langle M^T \nabla_v \mathcal{J}^T, \delta \vec{u} \rangle \tag{7.6}
\]
With the identity (7.5), we then find that the gradient \( \nabla_v \mathcal{J} \) can be readily inferred by invoking the adjoint \( M^* \) of the tangent linear model \( M \)
\[
\nabla_v \mathcal{J}^T|_{\vec{u}} = M^T|_{\vec{u}} \cdot \nabla_v \mathcal{J}^T|_{\vec{v}} = M^T|_{\vec{u}} \cdot \delta \vec{v}^* = \delta \vec{u}^* \tag{7.7}
\]
(7.7) is the adjoint model (ADM), in which \( M^T \) is the adjoint (here, the transpose) of the tangent linear operator \( M \), \( \delta \vec{v}^* \) the adjoint variable of the model state \( \vec{v} \), and \( \delta \vec{u}^* \) the adjoint variable of the control variable \( \vec{u} \).

The reverse nature of the adjoint calculation can be readily seen as follows. Consider a model integration which consists of \( \Lambda \) consecutive operations \( \mathcal{M}_\Lambda(\mathcal{M}_{\Lambda-1}(\ldots(\mathcal{M}_1(M_0(\vec{u}))))), \) where the \( \mathcal{M}’s \) could be the elementary steps, i.e., single lines in the code of the model, or successive time steps of the model integration, starting at step 0 and moving up to step \( \Lambda \), with intermediate \( \mathcal{M}_\Lambda(\vec{u}) = \vec{v}(\Lambda+1) \) and final \( \mathcal{M}_\Lambda(\vec{u}) = \vec{v}_* \). Let \( \mathcal{J} \) be a cost function which explicitly depends on the final state \( \vec{v}_* \) only (this restriction is for clarity reasons only). \( \mathcal{J}(\vec{u}) \) may be decomposed according to:
\[
\mathcal{J}(\vec{M}(\vec{u})) = \mathcal{J}(\mathcal{M}_\Lambda(\mathcal{M}_{\Lambda-1}(\ldots(\mathcal{M}_1(M_0(\vec{u})))))) \tag{7.8}
\]
Then, according to the chain rule, the forward calculation reads, in terms of the Jacobi matrices (we’ve omitted the |’s which, nevertheless are important to the aspect of tangent linearity; note also that by definition \( \langle \nabla_v \mathcal{J}^T, \delta \vec{v} \rangle = \nabla_v \mathcal{J} \cdot \delta \vec{v} \))
\[
\nabla_v \mathcal{J}(M(\delta \vec{u})) = \nabla_v \mathcal{J} \cdot M_{\Lambda} \cdot \ldots \cdot M_{\Lambda} \cdot \ldots \cdot M_1 \cdot M_0 \cdot \delta \vec{u} = \nabla_v \mathcal{J} \cdot \delta \vec{v} \tag{7.9}
\]
whereas in reverse mode we have
\[
M^T(\nabla_v \mathcal{J}^T) = M_0^T \cdot M_1^T \cdot \ldots \cdot M_{\Lambda}^T \cdot \ldots \cdot M_1^T \cdot M_0^T \cdot \nabla_v \mathcal{J}^T
= M_0^T \cdot M_1^T \cdot \ldots \cdot \nabla_v(\lambda) \mathcal{J}^T
= \nabla_v \mathcal{J}^T \tag{7.10}
\]
clearly expressing the reverse nature of the calculation. (7.10) is at the heart of automatic adjoint compilers. If the intermediate steps \( \lambda \) in (7.8) – (7.10) represent the model state (forward or adjoint) at each intermediate time step as noted above, then correspondingly, \( M^T(\delta \vec{v}(\lambda)^* \cdot \delta \vec{v}^* \) for the adjoint variables. It thus becomes evident that the adjoint calculation also yields the adjoint of each model state component \( \vec{v}(\lambda) \) at each intermediate step \( \lambda \), namely
\[
\nabla_v(\lambda) \mathcal{J}^T|_{\vec{v}(\lambda)} = M_\Lambda^T|_{\vec{v}(\lambda)} \cdot \ldots \cdot M_1^T|_{\vec{v}(\lambda)} \cdot \delta \vec{v}^* = \delta \vec{v}(\lambda)^* \tag{7.11}
\]
in close analogy to (7.7) we note in passing that the \( \delta \vec{v}(\lambda)^* \) are the Lagrange multipliers of the model equations which determine \( \vec{v}(\lambda) \).

In components, (7.7) reads as follows. Let
\[
\delta \vec{u} = \begin{pmatrix} \delta u_1, \ldots, \delta u_m \end{pmatrix}^T, \quad \delta \vec{u}^* = \nabla_u \mathcal{J}^T = \begin{pmatrix} \frac{\partial \mathcal{J}}{\partial u_1}, \ldots, \frac{\partial \mathcal{J}}{\partial u_m} \end{pmatrix}^T
\]
\[
\delta \vec{v} = \begin{pmatrix} \delta v_1, \ldots, \delta v_n \end{pmatrix}^T, \quad \delta \vec{v}^* = \nabla_v \mathcal{J}^T = \begin{pmatrix} \frac{\partial \mathcal{J}}{\partial v_1}, \ldots, \frac{\partial \mathcal{J}}{\partial v_n} \end{pmatrix}^T
\]
denote the perturbations in \(\vec{u}\) and \(\vec{v}\), respectively, and their adjoint variables; further

\[
M = \begin{pmatrix}
\frac{\partial M_1}{\partial u_1} & \cdots & \frac{\partial M_1}{\partial u_m} \\
\vdots & \ddots & \vdots \\
\frac{\partial M_n}{\partial u_1} & \cdots & \frac{\partial M_n}{\partial u_m}
\end{pmatrix}
\]

is the Jacobi matrix of \(\mathcal{M}\) (an \(n \times m\) matrix) such that \(\delta \vec{v} = M \cdot \delta \vec{u}\), or

\[
\delta v_j = \sum_{i=1}^{m} M_{ji} \delta u_i = \sum_{i=1}^{m} \frac{\partial M_j}{\partial u_i} \delta u_i
\]

Then (7.7) takes the form

\[
\delta u_i^* = \sum_{j=1}^{n} M_{ji} \delta v_j^* = \sum_{j=1}^{n} \frac{\partial M_j}{\partial u_i} \delta v_j^*
\]

or

\[
\begin{pmatrix}
\frac{\partial}{\partial u_1} \mathcal{J} \\ \vdots \\ \frac{\partial}{\partial u_m} \mathcal{J}
\end{pmatrix}_{\vec{u}(0)} = \begin{pmatrix}
\frac{\partial M_1}{\partial u_1} |_{\vec{u}(0)} & \cdots & \frac{\partial M_1}{\partial u_m} |_{\vec{u}(0)} \\
\vdots & \ddots & \vdots \\
\frac{\partial M_n}{\partial u_1} |_{\vec{u}(0)} & \cdots & \frac{\partial M_n}{\partial u_m} |_{\vec{u}(0)}
\end{pmatrix} \cdot \begin{pmatrix}
\frac{\partial}{\partial v_1} \mathcal{J} \\ \vdots \\ \frac{\partial}{\partial v_n} \mathcal{J}
\end{pmatrix}_{\vec{v}}
\]

Furthermore, the adjoint \(\delta v_j^{(\lambda)}\) of any intermediate state \(v_j^{(\lambda)}\) may be obtained, using the intermediate Jacobian (an \(n_{\lambda+1} \times n_{\lambda}\) matrix)

\[
M_{\lambda} = \begin{pmatrix}
\frac{\partial (M_{\lambda})_1}{\partial v_1^{(\lambda)}} & \cdots & \frac{\partial (M_{\lambda})_1}{\partial v_{n_{\lambda}}^{(\lambda)}} \\
\vdots & \ddots & \vdots \\
\frac{\partial (M_{\lambda})_{n_{\lambda}+1}}{\partial v_1^{(\lambda)}} & \cdots & \frac{\partial (M_{\lambda})_{n_{\lambda}+1}}{\partial v_{n_{\lambda}}^{(\lambda)}}
\end{pmatrix}
\]

and the shorthand notation for the adjoint variables \(\delta v_j^{(\lambda)}\)

\[
\begin{pmatrix}
\delta v_1^{(\lambda)} \\
\vdots \\
\delta v_{n_{\lambda}}^{(\lambda)}
\end{pmatrix} = \begin{pmatrix}
\frac{\partial (M_{\lambda})_1}{\partial v_1^{(\lambda)}} & \cdots & \frac{\partial (M_{\lambda})_{n_{\lambda}+1}}{\partial v_1^{(\lambda)}} \\
\vdots & \ddots & \vdots \\
\frac{\partial (M_{\lambda})_1}{\partial v_{n_{\lambda}}^{(\lambda)}} & \cdots & \frac{\partial (M_{\lambda})_{n_{\lambda}+1}}{\partial v_{n_{\lambda}}^{(\lambda)}}
\end{pmatrix} \cdot \begin{pmatrix}
\frac{\partial}{\partial v_1} \mathcal{J} \\ \vdots \\ \frac{\partial}{\partial v_n} \mathcal{J}
\end{pmatrix}_{\vec{v}} 
\]

(7.9) and (7.10) are perhaps clearest in showing the advantage of the reverse over the forward mode if the gradient \(\nabla_u \mathcal{J}\), i.e., the sensitivity of the cost function \(\mathcal{J}\) with respect to all input variables \(u\) (or the sensitivity of the cost function with respect to all intermediate states \(\vec{v}^{(\lambda)}\)) are sought. In order to be able to solve for each component of the gradient \(\partial \mathcal{J}/\partial u_i\) in (7.9) a forward calculation has to be performed for each component separately, i.e., \(\delta \vec{u} = \delta u_i \vec{e}_i\) for the \(i\)-th forward calculation. Then, (7.9) represents the projection of \(\nabla_u \mathcal{J}\) onto the \(i\)-th component. The full gradient
is retrieved from the $m$ forward calculations. In contrast, (7.10) yields the full gradient $\nabla_u \mathcal{J}$ (and all intermediate gradients $\nabla_v \mathcal{J}$) within a single reverse calculation.

Note, that if $\mathcal{J}$ is a vector-valued function of dimension $l > 1$, (7.10) has to be modified according to

$$M^T \left( \nabla_v \mathcal{J}^T \left( \delta \mathcal{J} \right) \right) = \nabla_u \mathcal{J}^T \cdot \delta \mathcal{J}$$

where now $\delta \mathcal{J} \in \mathbb{R}^l$ is a vector of dimension $l$. In this case $l$ reverse simulations have to be performed for each $\delta J_k$, $k = 1, \ldots, l$. Then, the reverse mode is more efficient as long as $l < n$, otherwise the forward mode is preferable.

A detailed analysis of the underlying numerical operations shows that the computation of $\nabla_u \mathcal{J}$ in this way requires about two to five times the computation of the cost function. Alternatively, the gradient vector could be approximated by finite differences, requiring $m$ computations of the perturbed cost function.

To conclude, we give two examples of commonly used types of cost functions:

**7.1.2.1 Example 1:** $\mathcal{J} = v_j(T)$

The cost function consists of the $j$-th component of the model state $\vec{v}$ at time $T$. Then $\nabla_v \mathcal{J}^T = \vec{f}_j$ is just the $j$-th unit vector. The $\nabla_u \mathcal{J}^T$ is the projection of the adjoint operator onto the $j$-th component $f_j$,

$$\nabla_u \mathcal{J}^T = M^T \cdot \nabla_v \mathcal{J}^T = \sum_i M_{ij}^T \vec{e}_i$$

**7.1.2.2 Example 2:** $\mathcal{J} = \langle \mathcal{H}(\vec{v}) - \vec{d}, \mathcal{H}(\vec{v}) - \vec{d} \rangle$

The cost function represents the quadratic model vs. data misfit. Here, $\vec{d}$ is the data vector and $\mathcal{H}$ represents the operator which maps the model state space onto the data space. Then, $\nabla_v \mathcal{J}$ takes the form

$$\nabla_v \mathcal{J}^T = 2 \mathcal{H} \cdot \left( \mathcal{H}(\vec{v}) - \vec{d} \right)$$

$$= 2 \sum_j \left\{ \sum_k \frac{\partial \mathcal{H}_k}{\partial v_j} (\mathcal{H}_k(\vec{v}) - d_k) \right\} \vec{f}_j$$

where $H_{kj} = \partial \mathcal{H}_k / \partial v_j$ is the Jacobi matrix of the data projection operator. Thus, the gradient $\nabla_u \mathcal{J}$ is given by the adjoint operator, driven by the model vs. data misfit:

$$\nabla_u \mathcal{J}^T = 2 M^T \cdot \mathcal{H} \cdot \left( \mathcal{H}(\vec{v}) - \vec{d} \right)$$

**7.1.3 Storing vs. recomputation in reverse mode**

We note an important aspect of the forward vs. reverse mode calculation. Because of the local character of the derivative (a derivative is defined with respect to a point along the trajectory), the intermediate results of the model trajectory $\vec{v}(\lambda + 1) = \mathcal{M}_\lambda(v(\lambda))$ may be required to evaluate the intermediate Jacobian $M_{\lambda}|_{\vec{v}(\lambda)} \delta \vec{v}(\lambda)$. This is the case for example for nonlinear expressions (momentum advection, nonlinear equation of state), and state-dependent conditional statements (parameterization schemes). In the forward mode, the intermediate results are required in the same order as computed by the full forward model $\mathcal{M}$, but in the reverse mode they are required in the reverse order. Thus, in the reverse mode the trajectory of the forward model integration $\mathcal{M}$ has to be stored to be available in the reverse calculation. Alternatively, the complete model state up to the point of evaluation has to be recomputed whenever its value is required.

A method to balance the amount of recomputations vs. storage requirements is called checkpointing (e.g., Griewank, 1992 [Gri92], Restrepo et al., 1998 [RLG98]). It is depicted in Figure 7.1 for a 3-level checkpointing (as an example, we give explicit numbers for a 3-day integration with a 1-hourly timestep in square brackets).
Figure 7.1: Schematic view of intermediate dump and restart for 3-level checkpointing.
In a first step, the model trajectory is subdivided into $n_{lev3}$ subsections [$n_{lev3}$=3 1-day intervals], with the label $lev3$ for this outermost loop. The model is then integrated along the full trajectory, and the model state stored to disk only at every $k_{lev3}$-th timestep [i.e. 3 times, at $i = 0, 1, 2$ corresponding to $k_{lev3} = 0, 24, 48$]. In addition, the cost function is computed, if needed.

In a second step each subsection itself is divided into $n_{lev2}$ subsections [$n_{lev2}$=4 6-hour intervals per subsection]. The model picks up at the last outermost dumped state $v_{k_{lev3}}$ and is integrated forward in time along the last subsection, with the label $lev2$ for this intermediate loop. The model state is now stored to disk at every $k_{lev2}$-th timestep [i.e. 4 times, at $i = 0, 1, 2, 3$ corresponding to $k_{lev2} = 48, 54, 60, 66$].

Finally, the model picks up at the last intermediate dump state $v_{k_{lev2}}$ and is integrated forward in time along the last subsection, with the label $lev1$ for this intermediate loop. Within this sub-subsection only, parts of the model state are stored to memory at every timestep [i.e. every hour $i = 0, ..., 5$ corresponding to $k_{lev1} = 66, 67, ..., 71$]. The final state $v_n = v_{k_{lev1}}$ is reached and the model state of all preceding timesteps along the last innermost subsection are available, enabling integration backwards in time along the last subsection. The adjoint can thus be computed along this last subsection $k_{lev2}$.

This procedure is repeated consecutively for each previous subsection $k_{lev2}^{n-1}, ..., k_{lev2}^1$ carrying the adjoint computation to the initial time of the subsection $k_{lev3}^n$. Then, the procedure is repeated for the previous subsection $k_{lev3}^{n-1}$ carrying the adjoint computation to the initial time $k_{lev3}^{n-1}$.

For the full model trajectory of $n_{lev3} \cdot n_{lev2} \cdot n_{lev1}$ timesteps the required storing of the model state was significantly reduced to $n_{lev2} + n_{lev1}$ to disk and roughly $n_{lev1}$ to memory (i.e., for the 3-day integration with a total of 72 timesteps the model state was stored 7 times to disk and roughly 6 times to memory). This saving in memory comes at a cost of a required 3 full forward integrations of the model (one for each checkpointing level). The optimal balance of storage vs. recomputation certainly depends on the computing resources available and may be adjusted by adjusting the partitioning among the $n_{lev3}, n_{lev2}, n_{lev1}$.

### 7.2 TLM and ADM generation in general

In this section we describe in a general fashion the parts of the code that are relevant for automatic differentiation using the software tool TAF. Modifications to use OpenAD are described in Section 7.5.

The basic flow is as follows:

```plaintext
the_model_main
  |--- initialise_fixed
  |--- #ifdef ALLOW_ADJOINT_RUN
  |     |--- ctrl_unpack
  |     |--- adthe_main_loop
  |     |     |--- initialise_varia
  |     |     |--- ctrl_map_forcing
  |     |     |--- do iloop = 1, nTimeSteps
  |     |     |     |--- forward_step
  |     |     |--- cost_tile
  |     |     |--- end do
  |     |--- cost_final

(continues on next page)
```
If CPP option ALLOW_AUTODIFF_TAMC is defined, the driver routine the_model_main.F, instead of calling the_model_loop.F, invokes the adjoint of this routine, adthe_main_loop.F (case #define ALLOW_ADJOINT_RUN, or the tangent linear of this routine g_the_main_loop.F (case #define ALLOW_TANGENTIAL_LINEAR_RUN), which are the toplevel routines in terms of automatic differentiation. The routines adthe_main_loop.F or g_the_main_loop.F are generated by TAF. It contains both the forward integration of the full model, the cost function calculation, any additional storing that is required for efficient checkpointing, and the reverse integration of the adjoint model.

[DESCRIBE IN A SEPARATE SECTION THE WORKING OF THE TLM]

The above structure of adthe_main_loop.F has been strongly simplified to focus on the essentials; in particular, no checkpointing procedures are shown here. Prior to the call of adthe_main_loop.F, the routine ctrl_unpack.F is invoked to unpack the control vector or initialize the control variables. Following the call of adthe_main_loop.F, the routine ctrl_pack.F is invoked to pack the control vector (cf. Section 7.2.5). If gradient checks are to be performed, the option #define ALLOW_GRDCHK is chosen. In this case the driver routine grdchk_main.F is called after the gradient has been computed via the adjoint (cf. Section 7.3).

### 7.2.1 General setup

In order to configure AD-related setups the following packages need to be enabled:

- pkg/autodiff
- pkg/ctrl
- pkg/cost
- pkg/grdchk

The packages are enabled by adding them to your experiment-specific configuration file packages.conf (see Section ??).
The following AD-specific CPP option files need to be customized:

- **ECCO_CPPOPTIONS.h** This header file collects CPP options for pkg/autodiff, pkg/cost, pkg/ctrl as well as AD-unrelated options for the external forcing package pkg/exf. (NOTE: These options are not set in their package-specific headers such as COST_OPTIONS.h, but are instead collected in the single header file ECCO_CPPOPTIONS.h. The package-specific header files serve as simple placeholders at this point.)
- **tamc.h** This header configures the splitting of the time stepping loop with respect to the 3-level checkpointing (see section ???).

### 7.2.2 Building the AD code using TAF

The build process of an AD code is very similar to building the forward model. However, depending on which AD code one wishes to generate, and on which AD tool is available (TAF or TAMC), the following make targets are available:

<table>
<thead>
<tr>
<th>AD-target</th>
<th>output</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>«MODE» «TOOL»</td>
<td>«MODE» «TOOL» output.f</td>
<td>generates code for «MODE» using «TOOL»</td>
</tr>
<tr>
<td>«MODE» «TOOL»</td>
<td>«MODE» «TOOL» output.f</td>
<td>generates code for «MODE» using «TOOL»</td>
</tr>
<tr>
<td>«MODE» «TOOL»</td>
<td>«MODE» «TOOL» output.f</td>
<td>includes make dependencies on .F .h</td>
</tr>
<tr>
<td>«MODE» «TOOL»</td>
<td>«MODE» «TOOL» output.f</td>
<td>i.e. input for «TOOL» may be re-generated</td>
</tr>
<tr>
<td>«MODE» all</td>
<td>mitgcmuv «MODE»</td>
<td>generates code for «MODE» using «TOOL» and compiles all code</td>
</tr>
</tbody>
</table>

Here, the following placeholders are used:

- **«TOOL»**
  - TAF
  - TAMC
- **«MODE»**
  - ad generates the adjoint model (ADM)
  - ftl generates the tangent linear model (TLM)
  - svd generates both ADM and TLM for singular value decomposition (SVD) type calculations

For example, to generate the adjoint model using TAF after routines (.F) or headers (.h) have been modified, but without compilation, type `make adtaf`; or, to generate the tangent linear model using TAMC without re-generating the input code, type `make ftltamconly`.

A typical full build process to generate the ADM via TAF would look like follows:

```bash
% mkdir build
% cd build
% ../../../tools/genmake2 -mods=../code_ad
% make depend
% make adall
```
7.2.3 The AD build process in detail

The make «MODE» all target consists of the following procedures:

1. A header file AD_CONFIG.h is generated which contains a CPP option on which code ought to be generated. Depending on the make target, the contents is one of the following:
   - #define ALLOW_ADJOINT_RUN
   - #define ALLOW_TANGENTLINEAR_RUN
   - #define ALLOW_ECCO_OPTIMIZATION

2. A single file «MODE» _ input_code . f is concatenated consisting of all . f files that are part of the list AD_FILES and all . flow files that are part of the list AD_FLOW_FILES.

3. The AD tool is invoked with the «MODE» _ TOOL _ FLAGS. The default AD tool flags in genmake2 can be overwritten by a tools/adjoint_options file (similar to the platform-specific tools/build_options, see Section 3.5.2.2). The AD tool writes the resulting AD code into the file «MODE» _ input_code_ad . f.

4. A short sed script tools/adjoint_sed is applied to «MODE» _ input_code_ad . f to reinstate myThid into the CALL argument list of active file I/O. The result is written to file «MODE» _ TOOL _ output . f.

5. All routines are compiled and an executable is generated.

7.2.3.1 The list AD_FILES and . list files

Not all routines are presented to the AD tool. Routines typically hidden are diagnostics routines which do not influence the cost function, but may create artificial flow dependencies such as I/O of active variables.

genmake2 generates a list (or variable) AD_FILES which contains all routines that are shown to the AD tool. This list is put together from all files with suffix .list that genmake2 finds in its search directories. The list file for the core MITgcm routines is model/src/model_ad_diff . list Note that no wrapper routine is shown to TAF. These are either not visible at all to the AD code, or hand-written AD code is available (see next section).

Each package directory contains its package-specific list file «PKG» _ ad_diff . list. For example, pkg/ptracers contains the file ptracers_ad_diff . list. Thus, enabling a package will automatically extend the AD_FILES list of genmake2 to incorporate the package-specific routines. Note that you will need to regenerate the makefile if you enable a package (e.g., by adding it to packages . conf) and a Makefile already exists.

7.2.3.2 The list AD_FLOW_FILES and . flow files

TAMC and TAF can evaluate user-specified directives that start with a specific syntax (CADJ, C$TAF, !$TAF). The main categories of directives are STORE directives and FLOW directives. Here, we are concerned with flow directives, store directives are treated elsewhere.

Flow directives enable the AD tool to evaluate how it should treat routines that are 'hidden' by the user, i.e. routines which are not contained in the AD_FILES list (see previous section), but which are called in part of the code that the AD tool does see. The flow directive tell the AD tool:

- which subroutine arguments are input/output
- which subroutine arguments are active
- which subroutine arguments are required to compute the cost
- which subroutine arguments are dependent
The syntax for the flow directives can be found in the AD tool manuals. 

genmake2 generates a list (or variable) AD_FLOW_FILES which contains all files with suffix.flow that it finds in its search directories. The flow directives for the core MITgcm routines of eesupp/src/ and model/src/ reside in pkg/autodiff/. This directory also contains hand-written adjoint code for the MITgcm WRAPPER (Section 6.2).

Flow directives for package-specific routines are contained in the corresponding package directories in the file «PKG»_ad.flow, e.g., ptracers-specific directives are in ptracers_ad.flow.

### 7.2.3.3 Store directives for 3-level checkpointing

The storing that is required at each period of the 3-level checkpointing is controlled by three top-level headers.

```fortran
    do ilev_3 = 1, nchklev_3
        include '$checkpoint_lev3.h$
        do ilev_2 = 1, nchklev_2
            include '$checkpoint_lev2.h$
            do ilev_1 = 1, nchklev_1
                include '$checkpoint_lev1.h$
            end do
        end do
    end do
```

All files checkpoint_lev?.h are contained in directory pkg/autodiff/.

### 7.2.3.4 Changing the default AD tool flags: ad_options files

### 7.2.3.5 Hand-written adjoint code

### 7.2.4 The cost function (dependent variable)

The cost function $J$ is referred to as the dependent variable. It is a function of the input variables $\vec{u}$ via the composition $J(\vec{u}) = J(M(\vec{u}))$. The input are referred to as the independent variables or control variables. All aspects relevant to the treatment of the cost function $J$ (parameter setting, initialization, accumulation, final evaluation), are controlled by the package pkg/cost. The aspects relevant to the treatment of the independent variables are controlled by the package pkg/ctrl and will be treated in the next section.

```fortran
    the_model_main
    |   |-- initialise_fixed
    |   |   |-- packages_readparms
    |   |   |-- cost_readparms
    |   |   o
    |   |-- the_main_loop
    |   ...   |-- initialise_varia
    (continues on next page)
7.2.4.1 Enabling the package

pkg/cost is enabled by adding the line `cost` to your file `packages.conf` (see Section ???).

In general the following packages ought to be enabled simultaneously: `pkg/autodiff`, `pkg/ctrl`, and `pkg/cost`. The basic CPP option to enable the cost function is `ALLOW_COST`. Each specific cost function contribution has its own option. For the present example the option is `ALLOW_COST_TRACER`. All cost-specific options are set in `ECCO_CPPOPTIONS.h` Since the cost function is usually used in conjunction with automatic differentiation, the CPP option `ALLOW_ADJOINT_RUN` (file `CPP_OPTIONS.h`) and `ALLOW_AUTODIFF_TAMC` (file `ECCO_CPPOPTIONS.h`) should be defined.

7.2.4.2 Initialization

The initialization of pkg/cost is readily enabled as soon as the CPP option `ALLOW_COST` is defined.

- The S/R `cost_readparms.F` reads runtime flags and parameters from file `data.cost`. For the present example the only relevant parameter read is `mult_tracer`. This multiplier enables different cost function contributions to be switched on (`= 1`) or off (`= 0`) at runtime. For more complex cost functions which involve model vs. data misfits, the corresponding data filenames and data specifications (start date and time, period, ...) are read in this S/R.
- The S/R `cost_init_varia.F` initializes the different cost function contributions. The contribution for the present example is `objf_tracer` which is defined on each tile (bi,bj).

7.2.4.3 Accumulation

The 'driver' routine `cost_tile.F` is called at the end of each time step. Within this 'driver' routine, S/R are called for each of the chosen cost function contributions. In the present example (ALLOW_COST_TRACER), S/R `cost_tracer.F` is called. It accumulates `objf_tracer` according to eqn. (ref:ask-the-author).
7.2.4.4 Finalize all contributions

At the end of the forward integration S/R cost_final.F is called. It accumulates the total cost function $f_c$ from each contribution and sums over all tiles:

$$\mathcal{J} = f_c = \text{mult tracer} \sum_{i}^{nSx} \sum_{j}^{nSy} \text{objf_tracer}(b_i, b_j) + ...$$

The total cost function $f_c$ will be the 'dependent' variable in the argument list for TAF, i.e.,

```
taf -output 'fc' ...
```

*************
the_main_loop
*************

|--- initialise_varia

|...

|   |-- packages_init_varia

|   |...

|   | |-- #ifdef ALLOW_ADJOINT_RUN

|   |   |-- call ctrl_map_ini

|   |   |-- call cost_ini

|   |   #endif

|   |...

|   | o

|...

|--- #ifdef ALLOW_ADJOINT_RUN

|   |-- call ctrl_map_forcing

|   #endif

...

|--- #ifdef ALLOW_TAMC_CHECKPOINTING

|   do ilev_3 = 1, nchklev_3

|   do ilev_2 = 1, nchklev_2

|   do ilev_1 = 1, nchklev_1

|   iloop = (ilev_3-1)*nchklev_2*nchklev_1 +

|     (ilev_2-1)*nchklev_1 + ilev_1

| #else

|   do iloop = 1, nTimeSteps

| #endif

|--- call forward_step


(continues on next page)
7.2.5 The control variables (independent variables)

The control variables are a subset of the model input (initial conditions, boundary conditions, model parameters). Here we identify them with the variable \( \vec{u} \). All intermediate variables whose derivative with respect to control variables do not vanish are called active variables. All subroutines whose derivative with respect to the control variables don’t vanish are called active routines. Read and write operations from and to file can be viewed as variable assignments. Therefore, files to which active variables are written and from which active variables are read are called active files. All aspects relevant to the treatment of the control variables (parameter setting, initialization, perturbation) are controlled by the package pkg/ctrl.
7.2.5.1 genmake2 and CPP options

Package pkg/ctrl is enabled by adding the line ctrl to your file packages.conf. Each control variable is enabled via its own CPP option in ECCO_CPPOPTIONS.h.

7.2.5.2 Initialization

- The S/R ctrl_readparms.F reads runtime flags and parameters from file data.ctrl. For the present example the file contains the file names of each control variable that is used. In addition, the number of wet points for each control variable and the net dimension of the space of control variables (counting wet points only) nvarlength is determined. Masks for wet points for each tile (bi,bj) and vertical layer k are generated for the three relevant categories on the C-grid: nWetCtile for tracer fields, nWetWtile for zonal velocity fields, nWetStile for meridional velocity fields.

- Two important issues related to the handling of the control variables in MITgcm need to be addressed. First, in order to save memory, the control variable arrays are not kept in memory, but rather read from file and added to the initial fields during the model initialization phase. Similarly, the corresponding adjoint fields which represent the gradient of the cost function with respect to the control variables are written to file at the end of the adjoint integration. Second, in addition to the files holding the 2-D and 3-D control variables and the corresponding cost gradients, a 1-D control vector and gradient vector are written to file. They contain only the wet points of the control variables and the corresponding gradient. This leads to a significant data compression. Furthermore, an option is available (ALLOW_NONDIMENSIONAL_CONTROL_IO) to non-dimensionalize the control and gradient vector, which otherwise would contain different pieces of different magnitudes and units. Finally, the control and gradient vector can be passed to a minimization routine if an update of the control variables is sought as part of a minimization exercise.

The files holding fields and vectors of the control variables and gradient are generated and initialized in S/R ctrl_unpack.F.

7.2.5.3 Perturbation of the independent variables

The dependency flow for differentiation with respect to the controls starts with adding a perturbation onto the input variable, thus defining the independent or control variables for TAF. Three types of controls may be considered:

- Consider as an example the initial tracer distribution pTracer as control variable. After pTracer has been initialized in ptracers_init_varia.F (dynamical variables such as temperature and salinity are initialized in ini_fields.F), a perturbation anomaly is added to the field in S/R ctrl_map_ini.F:

\[
\begin{align*}
  u &= u[0] + \Delta u \\
  tr1(...) &= tr1_{ini}(...) + xx_{tr1}(...) 
\end{align*}
\]  

(7.11)

\(xx_{tr1}\) is a 3-D global array holding the perturbation. In the case of a simple sensitivity study this array is identical to zero. However, its specification is essential in the context of automatic differentiation since TAF treats the corresponding line in the code symbolically when determining the differentiation chain and its origin. Thus, the variable names are part of the argument list when calling TAF:

```bash
taf -input 'xx_tr1 ...' ...
```

Now, as mentioned above, MITgcm avoids maintaining an array for each control variable by reading the perturbation to a temporary array from file. To ensure the symbolic link to be recognized by TAF, a scalar dummy variable \(xx_{tr1_dummy}\) is introduced and an 'active read' routine of the adjoint support package pkg/autodiff is invoked. The read-procedure is tagged with the variable \(xx_{tr1_dummy}\) enabling TAF to recognize the initialization of the perturbation. The modified call of TAF thus reads
and the modified operation (to perturb) in the code takes on the form

```plaintext
call active_read.xyz(
  ..., tmpfld3d, ..., xx_tr1_dummy, ...)

tr1(...) = tr1(...) + tmpfld3d(...)
```

Note that reading an active variable corresponds to a variable assignment. Its derivative corresponds to a write statement of the adjoint variable, followed by a reset. The 'active file' routines have been designed to support active read and corresponding adjoint active write operations (and vice versa).

- The handling of boundary values as control variables proceeds exactly analogous to the initial values with the symbolic perturbation taking place in S/R ctrl_map_forcing.F. Note however an important difference: Since the boundary values are time dependent with a new forcing field applied at each time step, the general problem may be thought of as a new control variable at each time step (or, if the perturbation is averaged over a certain period, at each $N$ timesteps), i.e.,

$$u_{\text{forcing}} = \{ u_{\text{forcing}}(t_n) \}_{n=1,...,n\text{TimeSteps}}$$

In the current example an equilibrium state is considered, and only an initial perturbation to surface forcing is applied with respect to the equilibrium state. A time dependent treatment of the surface forcing is implemented in the ECCO environment, involving the calendar (pkg/cal) and external forcing (pkg/ext) packages.

- This routine is not yet implemented, but would proceed along the same lines as the initial value sensitivity. The mixing parameters diffkr and kapgm are currently added as controls in ctrl_map_ini.F.

### 7.2.5.4 Output of adjoint variables and gradient

Several ways exist to generate output of adjoint fields.

- In `ctrl_map_ini.F`, `ctrl_map_forcing.F`:
  - The control variable fields `xx\_\`...`\`: before the forward integration, the control variables are read from file \`xx\_\`...`\ and added to the model field.
  - The adjoint variable fields `adxx\_\`...`\`, i.e., the gradient \(\nabla uJ\) for each control variable: after the adjoint integration the corresponding adjoint variables are written to `adxx\_\`...`\`.

- In `ctrl_unpack.F`, `ctrl_pack.F`:
  - The control vector `vector_ctrl`: at the very beginning of the model initialization, the updated compressed control vector is read (or initialized) and distributed to 2-D and 3-D control variable fields.
  - The gradient vector `vector_grad`: at the very end of the adjoint integration, the 2-D and 3-D adjoint variables are read, compressed to a single vector and written to file.

- In addition to writing the gradient at the end of the forward/adjoint integration, many more adjoint variables of the model state at intermediate times can be written using S/R `addummy_in_stepping.F`. The procedure is enabled using via the CPP-option `ALLOW_AUTODIFF_MONITOR` (file ECCO_CPPOPTIONS.h). To be part of the adjoint code, the corresponding S/R `dummy_in_stepping.F` has to be called in the forward model (S/R `the_main_loop.F`) at the appropriate place. The adjoint common blocks are extracted from the adjoint code via the header file `adcommon.h`.

  `dummy_in_stepping.F` is essentially empty, the corresponding adjoint routine is hand-written rather than generated automatically. Appropriate flow directives (`dummy_in_stepping.flow`) ensure that TAMC does not automatically generate `addummy_in_stepping.F` by trying to differentiate `dummy_in_stepping.F`, but instead refers to the hand-written routine.
dummy_in_stepping.F is called in the forward code at the beginning of each timestep, before the call to model/src/dynamics.F, thus ensuring that addummy_in_stepping.F is called at the end of each timestep in the adjoint calculation, after the call to addummy_in_dynamics.F.

addummy_in_stepping.F includes the header files adcommon.h. This header file is also hand-written. It contains the common blocks addynvars_r, addynvars_cd, addynvars_diffkr, addynvars_kapgm, adtr1_r, adfields, which have been extracted from the adjoint code to enable access to the adjoint variables.

**WARNING:** If the structure of the common blocks dynvars_r, dynvars_cd, etc., changes similar changes will occur in the adjoint common blocks. Therefore, consistency between the TAMC-generated common blocks and those in adcommon.h have to be checked.

### 7.2.5.5 Control variable handling for optimization applications

In optimization mode the cost function \( J(u) \) is sought to be minimized with respect to a set of control variables \( \delta J = 0 \), in an iterative manner. The gradient \( \nabla_u J|_{u[k]} \), together with the value of the cost function itself \( J(u[k]) \) at iteration step \( k \) serve as input to a minimization routine (e.g. quasi-Newton method, conjugate gradient, ...) (Gilbert and Lemaréchal, 1989 [GLemarechal89]) to compute an update in the control variable for iteration step \( k + 1 \):

\[
\begin{align*}
u[k+1] &= u[0] + \Delta u[k+1] \\
\text{satisfying} \quad J(u[k+1]) < J(u[k])
\end{align*}
\]

\( u[k+1] \) then serves as input for a forward/adjoint run to determine \( J \) and \( \nabla_u J \) at iteration step \( k + 1 \). Figure 7.2 sketches the flow between forward/adjoint model and the minimization routine.

The routines ctrl_unpack.F and ctrl_pack.F provide the link between the model and the minimization routine. As described in Section ref:ask-the-author the ctrl_unpack.F and ctrl_pack.F routines read and write control and gradient vectors which are compressed to contain only wet points, in addition to the full 2-D and 3-D fields. The corresponding I/O flow is shown in Figure 7.3:

---

**Figure 7.2:** Flow between the forward/adjoint model and the minimization routine.

The routines ctrl_unpack.F and ctrl_pack.F provide the link between the model and the minimization routine. As described in Section ref:ask-the-author the ctrl_unpack.F and ctrl_pack.F routines read and write control and gradient vectors which are compressed to contain only wet points, in addition to the full 2-D and 3-D fields. The corresponding I/O flow is shown in Figure 7.3:

---

**Figure 7.3:** Flow of data between the forward/adjoint model and the minimization routine.
vector_ctrl_<k>  \\
ctrl_unpack  \\
xx_theta0...<k>  \\
xx_salt0...<k>  \\
\vdots  \\
read  \\
forward integration  \\
↓  \\
adxx_theta0...<k>  \\
adxx_salt0...<k>  \\
\vdots  \\
write  \\
adjoint integration  \\
↓  \\
ctrl_pack  \\
vector_grad_<k>

Figure 7.3: Flow chart showing I/O in the forward/adjoint model.

ctrl_unpack.F reads the updated control vector vector_ctrl_<k>. It distributes the different control variables to 2-D and 3-D files xx_«...»<k>. At the start of the forward integration the control variables are read from xx_«...»<k> and added to the field. Correspondingly, at the end of the adjoint integration the adjoint fields are written to adxx_«...»<k>, again via the active file routines. Finally, ctrl_pack.F collects all adjoint files and writes them to the compressed vector file vector_grad_<k>.

NOTE: These options are not set in their package-specific headers such as COST_OPTIONS.h, but are instead collected in the single header file ECCO_CPPOPTIONS.h. The package-specific header files serve as simple placeholders at this point.

7.3 The gradient check package

An indispensable test to validate the gradient computed via the adjoint is a comparison against finite difference gradients. The gradient check package pkg/grdchk enables such tests in a straightforward and easy manner. The driver routine grdchk_main.F is called from the_model_main.F after the gradient has been computed via the adjoint model (cf. flow chart ???).

The gradient check proceeds as follows: The $i$–th component of the gradient $(\nabla u J^T)_i$ is compared with the following finite-difference gradient:

$$(\nabla u J^T)_i \quad \text{vs.} \quad \frac{\partial J}{\partial u_i} = \frac{J(u_i + \epsilon) - J(u_i)}{\epsilon}$$

A gradient check at point $u_i$ may generally considered to be successful if the deviation of the ratio between the adjoint and the finite difference gradient from unity is less than 1 percent,

$$1 - \frac{(\text{grad} J)_i(\text{adjoint})}{(\text{grad} J)_i(\text{finite difference})} < 1\%$$
7.3.1 Code description

7.3.2 Code configuration

The relevant CPP precompile options are set in the following files:

• CPP_OPTIONS.h - Together with the flag ALLOW_ADJOINT_RUN, define the flag ALLOW_GRADIENT_CHECK.

The relevant runtime flags are set in the files:

• data.pkg - Set useGrdchk = .TRUE.

• data.grdchk
  - grdchk_eps
  - nbeg
  - nstep
  - nend
  - grdchkvarindex

```
the_model_main
|-- ctrl_unpack
|-- adthe_main_loop - unperturbed cost function and
|-- ctrl_pack adjoint gradient are computed here
|--
|-- grdchk_main
| |-- grdchk_init
| |-- do icomp=... - loop over control vector elements
|   |-- grdchk_loc - determine location of icomp on grid
|   |-- grdchk_getxx - get control vector component from file
|   |-- grdchk_getadxx - get gradient component calculated
|   |   | via adjoint
|   |-- the_main_loop - forward run and cost evaluation
|   |   | with perturbed control vector element
|   |-- calculate ratio of adj. vs. finite difference gradient
|   |-- grdchk_setxx - Reset control vector element
|   |-- grdchk_print - print results
```
7.4 Adjoint dump & restart – divided adjoint (DIVA)

Authors: Patrick Heimbach & Geoffrey Gebbie, 07-Mar-2003*

*NOTE: THIS SECTION IS SUBJECT TO CHANGE. IT REFERS TO TAF-1.4.26.

Previous TAF versions are incomplete and have problems with both TAF options -pure and -mpi.

The code which is tuned to the DIVA implementation of this TAF version is checkpoint50 (MITgcm) and ecco_c50_e28 (ECCO).

7.4.1 Introduction

Most high performance computing (HPC) centers require the use of batch jobs for code execution. Limits in maximum available CPU time and memory may prevent the adjoint code execution from fitting into any of the available queues. This presents a serious limit for large long time adjoint ocean and climate model integrations. The MITgcm itself enables the split of the total model integration into sub-intervals through standard dump/restart of/from the full model state. For a similar procedure to run in reverse mode, the adjoint model requires, in addition to the model state, the adjoint model state, i.e., all variables with derivative information which are needed in an adjoint restart. This adjoint dump & restart is also termed 'divided adjoint (DIVA)'.

For this to work in conjunction with automatic differentiation, an AD tool needs to perform the following tasks:

1. identify an adjoint state, i.e., those sensitivities whose accumulation is interrupted by a dump/restart and which influence the outcome of the gradient. Ideally, this state consists of
   - the adjoint of the model state,
   - the adjoint of other intermediate results (such as control variables, cost function contributions, etc.)
   - bookkeeping indices (such as loop indices, etc.)

2. generate code for storing and reading adjoint state variables

3. generate code for bookkeeping , i.e., maintaining a file with index information

4. generate a suitable adjoint loop to propagate adjoint values for dump/restart with a minimum overhead of adjoint intermediate values.

TAF (but not TAMC!) generates adjoint code which performs the above specified tasks. It is closely tied to the adjoint multi-level checkpointing. The adjoint state is dumped (and restarted) at each step of the outermost checkpointing level and adjoint integration is performed over one outermost checkpointing interval. Prior to the adjoint computations, a full forward sweep is performed to generate the outermost (forward state) tapes and to calculate the cost function. In the current implementation, the forward sweep is immediately followed by the first adjoint leg. Thus, in theory, the following steps are performed (automatically)

- **1st model call:** This is the case if file costfinal does not exist. S/R mdthe_main_loop.f (generated by TAF) is called.
  1. calculate forward trajectory and dump model state after each outermost checkpointing interval to files tapelev3
  2. calculate cost function fc and write it to file costfinal

- **2nd and all remaining model calls:** This is the case if file costfinal does exist. S/R adthe_main_loop.f (generated by TAF) is called.
  1. (forward run and cost function call is avoided since all values are known)
    - if 1st adjoint leg: create index file divided.ctrl which contains info on current checkpointing index ilev3
– if not \( i \)-th adjoint leg: adjoint picks up at \( \text{ilev}3 = n\text{lev}3 - i + 1 \) and runs to \( n\text{lev}3 - i \)
2. perform adjoint leg from \( n\text{lev}3 - i + 1 \) to \( n\text{lev}3 - i \)
3. dump adjoint state to file \text{snapshot}
4. dump index file \text{divided.ctrl} for next adjoint leg
5. in the last step the gradient is written.

A few modifications were performed in the forward code, obvious ones such as adding the corresponding TAF-directive at the appropriate place, and less obvious ones (avoid some re-initializations, when in an intermediate adjoint integration interval).

[For TAF-1.4.20 a number of hand-modifications were necessary to compensate for TAF bugs. Since we refer to TAF-1.4.26 onwards, these modifications are not documented here].

### 7.4.2 Recipe 1: single processor

1. In \text{ECCO_CPPOPTIONS.h} set:
   - \#define ALLOW_DIVIDED_ADJOINT
   - \#undef ALLOW_DIVIDED_ADJOINT\_MPI

2. Generate adjoint code. Using the TAF option \(-\text{pure}\), two codes are generated:
   - \text{mdthe}\_main\_loop\_f: Is responsible for the forward trajectory, storing of outermost checkpoint levels to file, computation of cost function, and storing of cost function to file (1st step).
   - \text{adthe}\_main\_loop\_f: Is responsible for computing one adjoint leg, dump adjoint state to file and write index info to file (2nd and consecutive steps).

   for adjoint code generation, e.g., add \(-\text{pure}\) to TAF option list

   ```make adtaf```

   - One modification needs to be made to adjoint codes in S/R \text{adecco}\_the\_main\_loop\_f (generated by TAF):

     There’s a remaining issue with the \(-\text{pure}\) option. The call \text{ad...} between call \text{ad...} and the read of the \text{snapshot} file should be called only in the first adjoint leg between \( n\text{lev}3 \) and \( n\text{lev}3 - 1 \). In the ecco-branch, the following lines should be bracketed by an \text{if (idivbeg .GE. nchklev\_3) then}, thus:

     ```
     ... 
     xx\_psbar\_mean\_dummy = onetape\_xx\_psbar\_mean\_dummy\_3h(1)
     xx\_tbar\_mean\_dummy = onetape\_xx\_tbar\_mean\_dummy\_4h(1)
     xx\_sbar\_mean\_dummy = onetape\_xx\_sbar\_mean\_dummy\_5h(1)
     call barrier( mytid )
     cAdd(
       if (idivbeg .GE. nchklev\_3) then
       cAdd)
     call adcost\_final( mytid )
     call barrier( mytid )
     call adcost\_sst( mytid )
     call adcost\_ssh( mytid )
     call adcost\_hyd( mytid )
     call adcost\_averagesfields( mytime,myiter,mytid )
     ``` (continues on next page)
call barrier( mythid )
cAdd(   endif
    cAdd)

C-----------------------------------------------
C read snapshot
C-----------------------------------------------
if (idivbeg .lt. nchklev_3) then
    open(unit=77,file='snapshot',status='old',form='unformatted',
    $iostat=iers)
...

For the main code, in all likelihood the block which needs to be bracketed consists of adcost_final.f (generated by TAF) only.

• Now the code can be copied as usual to adjoint_model.F and then be compiled:

    make adchange

    then compile

7.4.3 Recipe 2: multi processor (MPI)

1. On the machine where you execute the code (most likely not the machine where you run TAF) find the includes directory for MPI containing mpif.h. Either copy mpif.h to the machine where you generate the .f files before TAF-ing, or add the path to the includes directory to your genmake2 platform setup, TAF needs some MPI parameter settings (essentially mpi_comm_world and mpi_integer) to incorporate those in the adjoint code.

2. In ECCO_CPPOPTIONS.h set
   • #define ALLOW_DIVIDED_ADJOINT
   • #define ALLOW_DIVIDED_ADJOINT_MPI

   This will include the header file mpif.h into the top level routine for TAF.

3. Add the TAF option -mpi to the TAF argument list in the makefile.

4. Follow the same steps as in Recipe 1.

7.5 Adjoint code generation using OpenAD

Authors: Jean Utke, Patrick Heimbach and Chris Hill
7.5.1 Introduction

The development of OpenAD was initiated as part of the ACTS (Adjoint Compiler Technology & Standards) project funded by the NSF Information Technology Research (ITR) program. The main goals for OpenAD initially defined for the ACTS project are:

1. develop a flexible, modular, open source tool that can generate adjoint codes of numerical simulation programs,
2. establish a platform for easy implementation and testing of source transformation algorithms via a language-independent abstract intermediate representation,
3. support for source code written in C and Fortan, and
4. generate efficient tangent linear and adjoint for the MIT general circulation model.

OpenAD’s homepage is at http://www-unix.mcs.anl.gov/OpenAD. A development WIKI is at http://wiki.mcs.anl.gov/OpenAD/index.php/Main_Page. From the WIKI’s main page, click on Handling GCM for various aspects pertaining to differentiating the MITgcm with OpenAD.

7.5.2 Downloading and installing OpenAD

The OpenAD webpage has a detailed description on how to download and build OpenAD. From its homepage, please click on Binaries. You may either download pre-built binaries for quick trial, or follow the detailed build process described at http://www.mcs.anl.gov/OpenAD/access.shtml.

7.5.3 Building MITgcm adjoint with OpenAD

17-January-2008

OpenAD was successfully built on head node of itrda.acesgrid.org, for following system:

```
> uname -a
Linux itrda 2.6.22.2-42.fc6 #1 SMP Wed Aug 15 12:34:26 EDT 2007 i686 i686 i386 GNU/Linux

> cat /proc/version
Linux version 2.6.22.2-42.fc6 (brewbuilder@hs20-bc2-4.build.redhat.com) (gcc version 4.1.2 20070626 (Red Hat 4.1.2-13)) #1 SMP Wed Aug 15 12:34:26 EDT 2007

> module load ifc/9.1.036 icc/9.1.042
```

Head of MITgcm branch (checkpoint59m with some modifications) was used for building adjoint code. Following routing needed special care (revert to revision 1.1): http://wwwcvs.mitgcm.org/viewvc/MITgcm/MITgcm_contrib/heimbach/OpenAD/OAD_support/active_module.f90?hideattic=0&view=markup.

7.5.4 Building the MITgcm adjoint using an OpenAD Singularity container

The MITgcm adjoint can also be built using a Singularity container. You will need Singularity, version 3.X. A container with OpenAD can be downloaded from the Sylabs Cloud:

```
singularity pull library://jahn/default/openad:latest
```

To use it, supply the path to the downloaded container to genmake2,

---

1 A big thank you to Dan Goldberg for supplying the definition file for the Singularity container!
If your build directory is on a remotely mounted file system (mounted at /mountpoint), you may have to add an option for mounting it in the container:

```
../../..tools/genmake2 -oad -oadsingularity /path/to/openad_latest.sif ...
```

The `oadsingularity` option is also supported by testreport, Section 5.5.2. Note that the path to the container has to be either absolute or relative to the build directory.
In this chapter and in the following chapter, the MITgcm ‘packages’ are described. While you can carry out many experiments with MITgcm by starting from case studies in section Section 4, configuring a brand new experiment or making major changes to an experimental configuration requires some knowledge of the packages that make up the full MITgcm code. Packages are used in MITgcm to help organize and layer various code building blocks that are assembled and selected to perform a specific experiment. Each of the specific experiments described in section Section 4 uses a particular combination of packages.

Figure 8.1 shows the full set of packages that are available. As shown in the figure packages are classified into different groupings that layer on top of each other. The top layer packages are generally specialized to specific simulation types. In this layer there are packages that deal with biogeochemical processes, ocean interior and boundary layer processes, atmospheric processes, sea-ice, coupled simulations and state estimation. Below this layer are a set of general purpose numerical and computational packages. The general purpose numerical packages provide code for kernel numerical algorithms that apply to many different simulation types. Similarly, the general purpose computational packages implement non-numerical algorithms that provide parallelism, I/O and time-keeping functions that are used in many different scenarios.

The following sections describe the packages shown in Figure 8.1. Section Section 8.1.1 describes the general procedure for using any package in MITgcm. Sections Section 8 to Section 10 layout the algorithms implemented in specific packages and describe how to use the individual packages. A brief synopsis of the function of each package is given in Figure 8.1. Organizationally package code is assigned a separate subdirectory in the MITgcm code distribution (within the source code directory pkg). The name of this subdirectory is used as the package name in Figure 8.1.

8.1 Overview

8.1.1 Using MITgcm Packages

The set of packages that will be used within a particular model can be configured using a combination of both “compile–time” and “run–time” options. Compile–time options are those used to select which packages will be “compiled in” or implemented within the program. Packages excluded at compile time are completely absent from the executable program(s) and thus cannot be later activated by any set of subsequent run–time options.

Here we use the following shorthand for various forms of package names, i.e. that appear in package-related filenames, parameters etc.: all upper case ${PKG}$, all lower case ${pkg}$, and mixed case ${Pkg}$. For example, for pkg/gmredi these are GMREDI, gmredi, and gmRedi respectively.
Figure 8.1: Hierarchy of code layers that are assembled to make up an MITgcm simulation. Conceptually (and in terms of code organization) MITgcm consists of several layers. At the base is a layer of core software that provides a basic numerical and computational foundation for MITgcm simulations. This layer is shown marked Foundation Code at the bottom of the figure and corresponds to code in the italicised subdirectories on the figure. This layer is not organized into packages. All code above the foundation layer is organized as packages. Much of the code in MITgcm is contained in packages which serve as a useful way of organizing and layering the different levels of functionality that make up the full MITgcm software distribution. The figure shows the different packages in MITgcm as boxes containing bold face upper case names. Directly above the foundation layer are two layers of general purpose infrastructure software that consist of computational and numerical packages. These general purpose packages can be applied to both online and offline simulations and are used in many different physical simulation types. Above these layers are more specialized packages.
8.1.1.1 Package Inclusion/Exclusion

There are numerous ways that one can specify compile–time package inclusion or exclusion and they are all implemented by the `genmake2` program which was previously described in Section Section 3.5. The options are as follows:

1. Setting the `genmake2` options `-enable PKG` and/or `-disable PKG` specifies inclusion or exclusion. This method is intended as a convenient way to perform a single (perhaps for a quick test) compilation.

2. By creating a text file with the name `packages.conf` in either the local build directory or the `-mods=DIR` directory, one can specify a list of packages (one package per line, with `'#'` as the comment character) to be included. Since the `packages.conf` file can be saved, this is the preferred method for setting and recording (for future reference) the package configuration.

3. For convenience, a list of “standard” package groups is contained in the `pkg/pkg_groups` file. By selecting one of the package group names in the `packages.conf` file, one automatically obtains all packages in that group.

4. By default (that is, if a `packages.conf` file is not found), the `genmake2` program will use the package group default “default_pkg_list” as defined in `pkg/pkg_groups` file.

5. To help prevent users from creating unusable package groups, the `genmake2` program will parse the contents of the `pkg/pkg_depend` file to determine:
   - whether any two requested packages cannot be simultaneously included (eg. `seaice` and `thsice` are mutually exclusive),
   - whether additional packages must be included in order to satisfy package dependencies (eg. `rw` depends upon functionality within the `mdsio` package), and
   - whether the set of all requested packages is compatible with the dependencies (and producing an error if they aren’t).

   Thus, as a result of the dependencies, additional packages may be added to those originally requested.

8.1.1.2 Package Activation

For run–time package control, MITgcm uses flags set through a `data.pkg` file. While some packages (eg. `debug`, `mnc`, `exch2`) may have their own usage conventions, most follow a simple flag naming convention of the form:

```
usePackageName=.TRUE.
```

where the `usePackageName` variable can activate or disable the package at runtime. As mentioned previously, packages must be included in order to be activated. Generally, such mistakes will be detected and reported as errors by the code. However, users should still be aware of the dependency.

8.1.1.3 Package Coding Standards

The following sections describe how to modify and/or create new MITgcm packages.
Packages are Not Libraries

To a beginner, the MITgcm packages may resemble libraries as used in myriad software projects. While future versions are likely to implement packages as libraries (perhaps using FORTRAN90/95 syntax) the current packages (FORTRAN77) are not based upon any concept of libraries.

File Inclusion Rules

Instead, packages should be viewed only as directories containing “sets of source files” that are built using some simple mechanisms provided by `genmake2`. Conceptually, the build process adds files as they are found and proceeds according to the following rules:

1. `genmake2` locates a “core” or main set of source files (the `-standarddirs` option sets these locations and the default value contains the directories `eesupp` and `model`).
2. `genmake2` then finds additional source files by inspecting the contents of each of the package directories:
   1. As the new files are found, they are added to a list of source files.
   2. If there is a file name “collision” (that is, if one of the files in a package has the same name as one of the files previously encountered) then the file within the newer (more recently visited) package will superseed (or “hide”) any previous file(s) with the same name.
   3. Packages are visited (and thus files discovered) in the order that the packages are enabled within `genmake2`. Thus, the files in PackB may superseed the files in PackA if PackA is enabled before PackB. Thus, package ordering can be significant! For this reason, `genmake2` honors the order in which packages are specified.

These rules were adopted since they provide a relatively simple means for rapidly including (or “hiding”) existing files with modified versions.

Conditional Compilation and `PACKAGES_CONFIG.h`

Given that packages are simply groups of files that may be added or removed to form a whole, one may wonder how linking (that is, FORTRAN symbol resolution) is handled. This is the second way that `genmake2` supports the concept of packages. Basically, `genmake2` creates a `Makefile` that, in turn, is able to create a file called `PACKAGES_CONFIG.h` that contains a set of C pre-processor (or “CPP”) directives such as:

```
#undef ALLOW_KPP
#undef ALLOW_LAND
...#define ALLOW_GENERIC_ADVDIFF
#define ALLOW_MDSIO
...```

These CPP symbols are then used throughout the code to conditionally isolate variable definitions, function calls, or any other code that depends upon the presence or absence of any particular package.

An example illustrating the use of these defines is:

```
#ifndef ALLOW_GMREDI
   IF (useGMRedi) CALL GMREDI_CALC_DIFF(
      I bi,bj,iMin,iMax,jMin,jMax,K,
      I maskUp,
      O KappaRT,KappaRS,
```

(continues on next page)
I myThid)
#endif

which is included from the file and shows how both the compile–time ALLOW_GMREDI flag and the run–time useGMRedi are nested.

There are some benefits to using the technique described here. The first is that code snippets or subroutines associated with packages can be placed or called from almost anywhere else within the code. The second benefit is related to memory footprint and performance. Since unused code can be removed, there is no performance penalty due to unnecessary memory allocation, unused function calls, or extra run-time IF (...) conditions. The major problems with this approach are the potentially difficult-to-read and difficult-to-debug code caused by an overuse of CPP statements. So while it can be done, developers should exercise some discipline and avoid unnecessarily “smearing” their package implementation details across numerous files.

Package Startup or Boot Sequence

Calls to package routines within the core code timestepping loop can vary. However, all packages should follow a required “boot” sequence outlined here:

1. S/R PACKAGES_BOOT()
   :
   CALL OPEN_COPY_DATA_FILE( 'data.pkg', 'PACKAGES_BOOT', ... )

2. S/R PACKAGES_READPARMS()
   :
   #ifdef ALLOW_${PKG}
   if ( use${Pkg} )
   & CALL ${PKG}_READPARMS( retCode )
   #endif

3. S/R PACKAGES_INIT_FIXED()
   :
   #ifdef ALLOW_${PKG}
   if ( use${Pkg} )
   & CALL ${PKG}_INIT_FIXED( retCode )
   #endif

4. S/R PACKAGES_CHECK()
   :
   #ifdef ALLOW_${PKG}
   if ( use${Pkg} )
   & CALL ${PKG}_CHECK( retCode )
   #else
   if ( use${Pkg} )
   & CALL PACKAGES_CHECK_ERROR('${PKG}')
   #endif

5. S/R PACKAGES_INIT_VARIABLES()
   :
   #ifdef ALLOW_${PKG}
   if ( use${Pkg} )
Adding a package to PARAMS.h and packages_boot()

An MITgcm package directory contains all the code needed for that package apart from one variable for each package. This variable is the use${Pkg} flag. This flag, which is of type logical, must be declared in the shared header file PARAMS.h in the PARM_PACKAGES block. This convention is used to support a single runtime control file data.pkg which is read by the startup routine packages_boot() and that sets a flag controlling the runtime use of a package. This routine needs to be able to read the flags for packages that were not built at compile time. Therefore when adding a new package, in addition to creating the per-package directory in the pkg/ subdirectory a developer should add a use${Pkg} flag to PARAMS.h and a use${Pkg} entry to the packages_boot() PACKAGES namelist. The only other package specific code that should appear outside the individual package directory are calls to the specific package API.

8.2 Packages Related to Hydrodynamical Kernel

8.2.1 Generic Advection/Diffusion

The generic_advdiff package contains high-level subroutines to solve the advection-diffusion equation of any tracer, either active (potential temperature, salinity or water vapor) or passive (see pkg/ptracer). (see also Section 2.16 and Section 2.17).

8.2.1.1 Introduction

Package “generic_advdiff” provides a common set of routines for calculating advective/diffusive fluxes for tracers (cell centered quantities on a C-grid).

Many different advection schemes are available: the standard centered second order, centered fourth order and upwind biased third order schemes are known as linear methods and require some stable time-stepping method such as Adams-Bashforth. Alternatives such as flux-limited schemes are stable in the forward sense and are best combined with the multi-dimensional method provided in gad_advection.
8.2.1.2 Key subroutines, parameters and files

There are two high-level routines:

• **GAD_CALC_RHS** calculates all fluxes at time level “n” and is used for the standard linear schemes. This must be used in conjunction with Adams–Bashforth time stepping. Diffusive and parameterized fluxes are always calculated here.

• **GAD_ADVECTION** calculates just the advective fluxes using the non-linear schemes and can not be used in conjunction with Adams–Bashforth time stepping.

<table>
<thead>
<tr>
<th>CPP Flag Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>COSINEMETH_III</td>
<td>#define</td>
<td>sets the implementation form of $\cos \varphi$ scaling of bi-harmonic terms for tracer diffusivity (note, in pkg/generic_advdiff routines the definition set here overrides whether this is defined in model/inc/CPP_OPTIONS.h, where the setting affects viscous term calculations)</td>
</tr>
<tr>
<td>ISOTROPIC_COS_SCALING</td>
<td>#undef</td>
<td>selects isotropic scaling of harmonic and bi-harmonic terms when using the $\cos \varphi$ scaling (note, in pkg/generic_advdiff routines the definition set here overrides whether this is defined in model/inc/CPP_OPTIONS.h, where the setting affects viscous term calculations)</td>
</tr>
<tr>
<td>DISABLE_MULTIDIM_ADVECTION</td>
<td>#undef</td>
<td>disables compilation of multi-dim. advection code</td>
</tr>
<tr>
<td>GAD_MULTIDIM_COMPRESSIBLE</td>
<td>#undef</td>
<td>use compressible flow method for multi-dim advection instead of older, less accurate method; note option has no effect on SOM advection which always uses compressible flow method</td>
</tr>
<tr>
<td>GAD_ALLOW_TS_SOM_ADV</td>
<td>#undef</td>
<td>enable the use of 2nd-order moment advection scheme (Prather 1986 [Pra86]) for temp. and salinity</td>
</tr>
<tr>
<td>GAD_SMOLARKIEWICZ_HACK</td>
<td>#undef</td>
<td>enables hack to get rid of negatives caused by Redi, see Smolarkiewicz (1989) [Smo89] (for ptracers, except temp and salinity)</td>
</tr>
</tbody>
</table>

8.2.1.3 GAD Diagnostics

<table>
<thead>
<tr>
<th>&lt;-Name-&gt;</th>
<th>Levs</th>
<th>&lt;-parsing code-&gt;</th>
<th>&lt;- Units --&gt;</th>
<th>&lt;- Tile (max=80c)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADVr_TH</td>
<td>15</td>
<td>WM LR</td>
<td>degC.m^3/s</td>
<td>Vertical</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Temperature</td>
<td></td>
<td>Adveotive Flux of Pot.</td>
</tr>
<tr>
<td>ADVx_TH</td>
<td>15</td>
<td>UU 087MR</td>
<td>degC.m^3/s</td>
<td>Zonal</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Temperature</td>
<td></td>
<td>Adveotive Flux of Pot.</td>
</tr>
<tr>
<td>ADVy_TH</td>
<td>15</td>
<td>VV 086MR</td>
<td>degC.m^3/s</td>
<td>Meridional</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Temperature</td>
<td></td>
<td>Adveotive Flux of Pot.</td>
</tr>
<tr>
<td>DFrE_TH</td>
<td>15</td>
<td>WM LR</td>
<td>degC.m^3/s</td>
<td>Vertical</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Temperature</td>
<td></td>
<td>Diffusive part</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(Explicit part)</td>
<td></td>
<td>Flux of Pot.</td>
</tr>
<tr>
<td>DIFx_TH</td>
<td>15</td>
<td>UU 090MR</td>
<td>degC.m^3/s</td>
<td>Zonal</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Temperature</td>
<td></td>
<td>Diffusive Flux of Pot.</td>
</tr>
<tr>
<td>DIFy_TH</td>
<td>15</td>
<td>VV 089MR</td>
<td>degC.m^3/s</td>
<td>Meridional</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Temperature</td>
<td></td>
<td>Diffusive Flux of Pot.</td>
</tr>
</tbody>
</table>

(continues on next page)
### Fluid Transport Terms

<table>
<thead>
<tr>
<th>Term</th>
<th>CPP/ISO</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DFrI_TH</td>
<td>15 WM LR</td>
<td></td>
<td>Vertical Diffusive Flux of Pot. Temperature</td>
</tr>
<tr>
<td>DFrII_TH</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADVr_SLT</td>
<td>15 WM LR</td>
<td></td>
<td>Vertical Advection Flux of Salinity</td>
</tr>
<tr>
<td>ADVx_SLT</td>
<td>15 UU 094MR</td>
<td></td>
<td>Zonal Advection Flux of Salinity</td>
</tr>
<tr>
<td>ADVy_SLT</td>
<td>15 VV 093MR</td>
<td></td>
<td>Meridional Advection Flux of Salinity</td>
</tr>
<tr>
<td>DFre_SLT</td>
<td>15 WM LR</td>
<td></td>
<td>Vertical Diffusive Flux of Salinity (Explicit)</td>
</tr>
<tr>
<td>DIFx_SLT</td>
<td>15 UU 097MR</td>
<td></td>
<td>Zonal Diffusive Flux of Salinity</td>
</tr>
<tr>
<td>DIFy_SLT</td>
<td>15 VV 096MR</td>
<td></td>
<td>Meridional Diffusive Flux of Salinity</td>
</tr>
</tbody>
</table>

#### Experiments and Tutorials that use GAD

- Baroclinic gyre experiment, in tutorial_baroclinic_gyre verification directory.
- Tracer Sensitivity tutorial, in tutorial_tracer_adjsens verification directory.

### 8.2.2 Momentum Packages

<table>
<thead>
<tr>
<th>CPP Flag Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALLOW_SMAG_3D</td>
<td>#undef</td>
<td>allow isotropic 3D Smagorinsky viscosity</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(MOM_COMMON_OPTIONS.h)</td>
</tr>
<tr>
<td>ALLOW_3D_VISCAH</td>
<td>#undef</td>
<td>allow full 3D specification of horizontal Laplacian</td>
</tr>
<tr>
<td></td>
<td></td>
<td>viscosity (MOM_COMMON_OPTIONS.h)</td>
</tr>
<tr>
<td>ALLOW_3D_VISCA4</td>
<td>#undef</td>
<td>allow full 3D specification of horizontal biharmonic</td>
</tr>
<tr>
<td></td>
<td></td>
<td>viscosity (MOM_COMMON_OPTIONS.h)</td>
</tr>
<tr>
<td>MOM_BOUNDARY_CONSERVE</td>
<td>#undef</td>
<td>conserve $u$, $v$ momentum next to a step (vertical plane) or a coastline edge (horizontal plane)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(MOM_FLUXFORM_OPTIONS.h)</td>
</tr>
</tbody>
</table>

### 8.2.3 Shapiro Filter

(in directory: pkg/shap_filt/)

#### 8.2.3.1 Key subroutines, parameters and files

Implementation of filter is described in Section 2.18.
8.2.3.2 Experiments and tutorials that use shap filter

- Held Suarez tutorial, in verification/tutorial_held_suarez_cs.
- Other Held Suarez verification experiments (hs94.128x64x5, hs94.1x64x5, hs94.cs-32x32x5)
- AIM verification experiments (aim.5l_cs, aim.5l_Equatorial_Channel, aim.5l_LatLon)
- Fizhi verification experiments (fizhi-cs-32x32x40, fizhi-cs-aqualev20, fizhi-gridalt-hs)

8.2.4 FFT Filtering Code

(in directory: pkg/zonal_filt/)

8.2.4.1 Key subroutines, parameters and files

8.2.4.2 Experiments and tutorials that use zonal filter

- Held Suarez verification experiment (hs94.128x64x5)
- AIM verification experiment (aim.5l_LatLon)

8.2.5 exch2: Extended Cubed Sphere Topology

(in directory: pkg/exch2/)

8.2.5.1 Introduction

The exch2 package extends the original cubed sphere topology configuration to allow more flexible domain decomposition and parallelization. Cube faces (also called subdomains) may be divided into any number of tiles that divide evenly into the grid point dimensions of the subdomain. Furthermore, the tiles can run on separate processors individually or in groups, which provides for manual compile-time load balancing across a relatively arbitrary number of processors.

The exchange parameters are declared in W2_EXCH_TOPOLOGY.h and assigned in w2_e2setup.F. The validity of the cube topology depends on the SIZE.h file as detailed below. The default files provided in the release configure a cubed sphere topology of six tiles, one per subdomain, each with 32 × 32 grid points, with all tiles running on a single processor. Both files are generated by Matlab scripts in utils/exch2/matlab-topology-generator; see Section 8.2.5.3 for details on creating alternate topologies. Pregenerated examples of these files with alternate topologies are provided under utils/exch2/code-mods along with the appropriate SIZE.h file for single-processor execution.

8.2.5.2 Invoking exch2

To use exch2 with the cubed sphere, the following conditions must be met:

- The exch2 package is included when gennake2 is run. The easiest way to do this is to add the line exch2 to the packages.conf file – see Section Building the model for general details.

- An example of W2_EXCH2_TOPOLOGY.h and w2_e2setup.F must reside in a directory containing files symbolically linked by the gennake2 script. The safest place to put these is the directory indicated in the -mods=DIR command line modifier (typically ./code), or the build directory. The default versions of these files reside in pkg/exch2 and are linked automatically if no other versions exist elsewhere in the build path, but they should be left untouched to avoid breaking configurations other than the one you intend to modify.
• Files containing grid parameters, named tile00$n$.mitgrid where n=(1:6) (one per subdomain), must be in the working directory when the MITgcm executable is run. These files are provided in the example experiments for cubed sphere configurations with $32 \times 32$ cube sides – please contact MITgcm support if you want to generate files for other configurations.

• As always when compiling MITgcm, the file SIZE.h must be placed where genmake2 will find it. In particular for exch2, the domain decomposition specified in SIZE.h must correspond with the particular configuration's topology specified in W2_EXCH2_TOPOLOGY.h and w2_e2setup.F. Domain decomposition issues particular to exch2 are addressed in Section Generating Topology Files for exch2 and exch2, SIZE.h, and Multiprocessing a more general background on the subject relevant to MITgcm is presented in Section Using the WRAPPER.

At the time of this writing the following examples use exch2 and may be used for guidance:

• verification/adjust_nlfs.cs-32x32x1
• verification/adjustment.cs-32x32x1
• verification/aim.5l_cs
• verification/global_ocean.cs32x15
• verification/hs94.cs-32x32x5

8.2.5.3 Generating Topology Files for exch2

Alternate cubed sphere topologies may be created using the Matlab scripts in utils/exch2/matlab-topology-generator. Running the m-file driver.m from the Matlab prompt (there are no parameters to pass) generates exch2 topology files W2_EXCH2_TOPOLOGY.h and w2_e2setup.F in the working directory and displays a figure of the topology via Matlab – Figure 8.4, Figure 8.3, and Figure 8.2 are examples of the generated diagrams. The other m-files in the directory are subroutines called from driver.m and should not be run "bare" except for development purposes.

The parameters that determine the dimensions and topology of the generated configuration are $n_r$, $n_b$, $n_g$, $t_{nx}$ and $t_{ny}$, and all are assigned early in the script.

The first three determine the height and width of the subdomains and hence the size of the overall domain. Each one determines the number of grid points, and therefore the resolution, along the subdomain sides in a "great circle" around each the three spatial axes of the cube. At the time of this writing MITgcm requires these three parameters to be equal, but they provide for future releases to accomodate different resolutions around the axes to allow subdomains with differing resolutions.

The parameters $t_{nx}$ and $t_{ny}$ determine the width and height of the tiles into which the subdomains are decomposed, and must evenly divide the integer assigned to $n_r$, $n_b$ and $n_g$. The result is a rectangular tiling of the subdomain. Figure 8.2 shows one possible topology for a twenty-four-tile cube, and Figure 8.4 shows one for six tiles.

Tiles can be selected from the topology to be omitted from being allocated memory and processors. This tuning is useful in ocean modeling for omitting tiles that fall entirely on land. The tiles omitted are specified in the file blanklist.txt by their tile number in the topology, separated by a newline.
Figure 8.2: Plot of a cubed sphere topology with a $32 \times 192$ domain divided into six $32 \times 32$ subdomains, each of which is divided into eight tiles of width $t_{nx}=16$ and height $t_{ny}=8$ for a total of forty-eight tiles. The colored borders of the subdomains represent the parameters $nr$ (red), $ng$ (green), and $nb$ (blue). This tiling is used in the example verification/adjustment.cs-32x32x1/ with the option (blanklist.txt) to remove the land-only 4 tiles (11,12,13,14) which are filled in red on the plot.

Figure 8.3: Plot of a non-square cubed sphere topology with 6 subdomains of different size ($nr=90, ng=360, nb=90$), divided into one to four tiles each ($t_{nx}=90$, $t_{ny}=90$), resulting in a total of 18 tiles.
Figure 8.4: Plot of a cubed sphere topology with a $32 \times 192$ domain divided into six $32 \times 32$ subdomains with one tile each ($tnx=32$, $tny=32$). This is the default configuration.

### 8.2.5.4 exch2, SIZE.h, and Multiprocessing

Once the topology configuration files are created, each Fortran `PARAMETER` in `SIZE.h` must be configured to match. Section 6.3 provides a general description of domain decomposition within MITgcm and its relation to `SIZE.h`. The current section specifies constraints that the exch2 package imposes and describes how to enable parallel execution with MPI.

As in the general case, the parameters `sNx` and `sNy` define the size of the individual tiles, and so must be assigned the same respective values as `$tnx$` and `$tny$` in `driver.m`.

The halo width parameters $OLx$ and $OLy$ have no special bearing on exch2 and may be assigned as in the general case. The same holds for $Nr$, the number of vertical levels in the model.

The parameters $nSx$, $nSy$, $nPx$, and $nPy$ relate to the number of tiles and how they are distributed on processors. When using exch2, the tiles are stored in the $x$ dimension, and so $nSy = 1$ in all cases. Since the tiles as configured by exch2 cannot be split up across processors without regenerating the topology, $nPy = 1$ as well.

The number of tiles MITgcm allocates and how they are distributed between processors depends on $nPx$ and $nSx$. $nSx$ is the number of tiles per processor and $nPx$ is the number of processors. The total number of tiles in the topology minus those listed in `blanklist.txt` must equal $nSx*nPx$. Note that in order to obtain maximum usage from a given number of processors in some cases, this restriction might entail sharing a processor with a tile that would otherwise be excluded because it is topographically outside of the domain and therefore in `blanklist.txt`. For example, suppose you have five processors and a domain decomposition of thirty-six tiles that allows you to exclude seven tiles. To evenly distribute the remaining twenty-nine tiles among five processors, you would have to run one "dummy" tile to make an even six tiles per processor. Such dummy tiles are not listed in `blanklist.txt`.

The following is an example of `SIZE.h` for the six-tile configuration illustrated in Figure 8.4 running on one processor:

```plaintext
PARAMETER (
(continues on next page)
```
The following is an example for the forty-eight-tile topology in Figure 8.2 running on six processors:

```plaintext
PARAMETER (  
&    sNx = 16,  
&    sNy = 8,  
&    OLx = 2,  
&    OLy = 2,  
&    nSx = 8,  
&    nSy = 1,  
&    nPx = 6,  
&    nPy = 1,  
&    Nx = sNx*nSx*nPx,  
&    Ny = sNy*nSy*nPy,  
&    Nr = 5)
```

### 8.2.5.5 Key Variables

The descriptions of the variables are divided up into scalars, one-dimensional arrays indexed to the tile number, and two and three-dimensional arrays indexed to tile number and neighboring tile. This division reflects the functionality of these variables: The scalars are common to every part of the topology, the tile-indexed arrays to individual tiles, and the arrays indexed by tile and neighbor to relationships between tiles and their neighbors.

**Scalars:**

The number of tiles in a particular topology is set with the parameter `exch2_nTiles`, and the maximum number of neighbors of any tiles by `W2_maxNeighbours`. These parameters are used for defining the size of the various one and two dimensional arrays that store tile parameters indexed to the tile number and are assigned in the files generated by `driver.m`.

The scalar parameters `exch2_domain_nxt` and `exch2_domain_nyt` express the number of tiles in the x and y global indices. For example, the default setup of six tiles (Figure 8.4) has `exch2_domain_nxt=6` and `exch2_domain_nyt=1`. A topology of forty-eight tiles, eight per subdomain (as in Figure 8.2), will have `exch2_domain_nxt=12` and `exch2_domain_nyt=4`. Note that these parameters express the tile layout in order to allow global data files that are tile-layout-neutral. They have no bearing on the internal storage of the arrays. The tiles are stored internally in a range from `bi = (1:exch2_nTiles)` in the x axis, and the y axis variable `bj` is assumed to equal 1 throughout the package.
Arrays indexed to tile number:

The following arrays are of length `exch2_nTiles` and are indexed to the tile number, which is indicated in the diagrams with the notation `tn`. The indices are omitted in the descriptions.

The arrays `exch2_tnx` and `exch2_tny` express the x and y dimensions of each tile. At present for each tile `exch2_tnx` = `sNx` and `exch2_tny` = `sNy`, as assigned in `SIZE.h` and described in Section 8.2.5.4. Future releases of MITgcm may allow varying tile sizes.

The arrays `exch2_tbase` and `exch2_tbasey` determine the tiles’ Cartesian origin within a subdomain and locate the edges of different tiles relative to each other. As an example, in the default six-tile topology (Figure 8.4) each index in these arrays is set to 0 since a tile occupies its entire subdomain. The twenty-four-tile case discussed above will have values of 0 or 16, depending on the quadrant of the tile within the subdomain. The elements of the arrays `exch2_txglobal` and `exch2_tyglobal` are similar to `exch2_tbase` and `exch2_tbasey`, but locate the tile edges within the global address space, similar to that used by global output and input files.

The array `exch2_myFace` contains the number of the subdomain of each tile, in a range (1:6) in the case of the standard cube topology and indicated by `fn` in Figures 8.4 and 8.2. `exch2_nNeighbours` contains a count of the neighboring tiles each tile has, and sets the bounds for looping over neighboring tiles. `exch2_tProc` holds the process rank of each tile, and is used in interprocess communication.

The arrays `exch2_isWedge`, `exch2_isEedge`, `exch2_isSedge`, and `exch2_isNedge` are set to 1 if the indexed tile lies on the edge of its subdomain, 0 if not. The values are used within the topology generator to determine the orientation of neighboring tiles, and to indicate whether a tile lies on the corner of a subdomain. The latter case requires special exchange and numerical handling for the singularities at the eight corners of the cube.

Arrays Indexed to Tile Number and Neighbor:

The following arrays have vectors of length `W2_maxNeighbours` and `exch2_nTiles` and describe the orientations between the the tiles.

The array `exch2_neighbourId(a,T)` holds the tile number `Tn` for each of the tile number `T`’s neighboring tiles `a`. The neighbor tiles are indexed 1:`exch2_nNeighbours(T)` in the order right to left on the north then south edges, and then top to bottom on the east then west edges.

The `exch2_opposingSend_record(a,T)` array holds the index `b` of the element in `exch2_neighbourId(b,Tn)` that holds the tile number `T`, given `Tn=exch2_neighbourId(a,T)`. In other words,

```
exch2_neighbourId( exch2_opposingSend_record(a,T),
                    exch2_neighbourId(a,T) ) = T
```

This provides a back-reference from the neighbor tiles.

The arrays `exch2_pi` and `exch2 pj` specify the transformations of indices in exchanges between the neighboring tiles. These transformations are necessary in exchanges between subdomains because a horizontal dimension in one subdomain may map to other horizontal dimension in an adjacent subdomain, and may also have its indexing reversed. This swapping arises from the “folding” of two-dimensional arrays into a three-dimensional cube.

The dimensions of `exch2_pi(t,N,T)` and `exch2 pj(t,N,T)` are the neighbor ID `N` and the tile number `T` as explained above, plus a vector of length 2 containing transformation factors `t`. The first element of the transformation vector holds the factor to multiply the index in the same dimension, and the second element holds the the same for the orthogonal dimension. To clarify, `exch2_pi(1,N,T)` holds the mapping of the x axis index of tile T to the x axis of tile T’s neighbor N, and `exch2 pj(2,N,T)` holds the mapping of T’s x index to the neighbor N’s y index.

One of the two elements of `exch2_pi` or `exch2 pj` for a given tile T and neighbor N will be 0, reflecting the fact that the two axes are orthogonal. The other element will be 1 or -1, depending on whether the axes are indexed in the same or opposite directions. For example, the transform vector of the arrays for all tile neighbors on the same subdomain will...
be \((1,0)\), since all tiles on the same subdomain are oriented identically. An axis that corresponds to the orthogonal dimension with the same index direction in a particular tile-neighbor orientation will have \((0,1)\). Those with the opposite index direction will have \((0,-1)\) in order to reverse the ordering.

The arrays \texttt{exch2\_oi}, \texttt{exch2\_oj}, \texttt{exch2\_oi\_f}, and \texttt{exch2\_oj\_f} are indexed to tile number and neighbor and specify the relative offset within the subdomain of the array index of a variable going from a neighboring tile \(N\) to a local tile \(T\). Consider \(T=1\) in the six-tile topology (Figure 8.4), where

| \texttt{exch2\_oi(1,1)} | \texttt{33} |
| \texttt{exch2\_oi(2,1)} | \texttt{0} |
| \texttt{exch2\_oi(3,1)} | \texttt{32} |
| \texttt{exch2\_oi(4,1)} | \texttt{-32} |

The simplest case is \texttt{exch2\_oi(2,1)}, the southern neighbor, which is \(Tn=6\). The axes of \(T\) and \(Tn\) have the same orientation and their \(x\) axes have the same origin, and so an exchange between the two requires no changes to the \(x\) index. For the western neighbor (\(Tn=5\)), \texttt{code\_oi(3,1)=32} since the \(x=0\) vector on \(T\) corresponds to the \(y=32\) vector on \(Tn\). The eastern edge of \(T\) shows the reverse case (\texttt{exch2\_oi(4,1)=-32}), where \(x=32\) on \(T\) exchanges with \(x=0\) on \(Tn=2\).

The most interesting case, where \texttt{exch2\_oi(1,1)=33} and \(Tn=3\), involves a reversal of indices. As in every case, the offset \texttt{exch2\_oi} is added to the original \(x\) index of \(T\) multiplied by the transformation factor \texttt{exch2\_pi(t,N,T)}. Here \texttt{exch2\_pi(1,1,1)=0} since the \(x\) axis of \(T\) is orthogonal to the \(x\) axis of \(Tn\). \texttt{exch2\_pi(2,1,1)=-1} since the \(x\) axis of \(T\) corresponds to the \(y\) axis of \(Tn\), but the index is reversed. The result is that the index of the northern edge of \(T\), which runs \((1:32)\), is transformed to \((-1:-32)\). \texttt{exch2\_oi(1,1)} is then added to this range to get back \((32:1)\) – the index of the \(y\) axis of \(Tn\) relative to \(T\). This transformation may seem overly convoluted for the six-tile case, but it is necessary to provide a general solution for various topologies.

Finally, \texttt{exch2\_itlo\_c}, \texttt{exch2\_ithi\_c}, \texttt{exch2\_itlo\_c} and \texttt{exch2\_ithi\_c} hold the location and index bounds of the edge segment of the neighbor tile \(N\)’s subdomain that gets exchanged with the local tile \(T\). To take the example of tile \(T=2\) in the forty-eight-tile topology (Figure 8.2):

| \texttt{exch2\_itlo\_c(4,2)} | \texttt{17} |
| \texttt{exch2\_ithi\_c(4,2)} | \texttt{17} |
| \texttt{exch2\_jtlo\_c(4,2)} | \texttt{0} |
| \texttt{exch2\_jthi\_c(4,2)} | \texttt{33} |

Here \(N=4\), indicating the western neighbor, which is \(Tn=1\). \(Tn\) resides on the same subdomain as \(T\), so the tiles have the same orientation and the same \(x\) and \(y\) axes. The \(x\) axis is orthogonal to the western edge and the tile is 16 points wide, so \texttt{exch2\_itlo\_c} and \texttt{exch2\_ithi\_c} indicate the column beyond \(Tn\)’s eastern edge, in that tile’s halo region. Since the border of the tiles extends through the entire height of the subdomain, the \(y\) axis bounds \texttt{exch2\_jtlo\_c} to \texttt{exch2\_jthi\_c} cover the height of \((1:32)\), plus 1 in either direction to cover part of the halo.

For the north edge of the same tile \(T=2\) where \(N=1\) and the neighbor tile is \(Tn=5\):

| \texttt{exch2\_itlo\_c(1,2)} | \texttt{0} |
| \texttt{exch2\_ithi\_c(1,2)} | \texttt{0} |
| \texttt{exch2\_jtlo\_c(1,2)} | \texttt{0} |
| \texttt{exch2\_jthi\_c(1,2)} | \texttt{17} |

\(T\)’s northern edge is parallel to the \(x\) axis, but since \(Tn\)’s \(y\) axis corresponds to \(T\)’s \(x\) axis, \(T\)’s northern edge exchanges with \(Tn\)’s western edge. The western edge of the tiles corresponds to the lower bound of the \(x\) axis, so \texttt{exch2\_itlo\_c} and \texttt{exch2\_ithi\_c} are 0, in the western halo region of \(Tn\). The range of \texttt{exch2\_jtlo\_c} and \texttt{exch2\_jthi\_c} correspond to the width of \(T\)’s northern edge, expanded by one into the halo.

8.2. Packages Related to Hydrodynamical Kernel
8.2.5.6 Key Routines

Most of the subroutines particular to exch2 handle the exchanges themselves and are of the same format as those described in *Cube sphere communication*. Like the original routines, they are written as templates which the local Makefile converts from RX into RL and RS forms.

The interfaces with the core model subroutines are `EXCH_UV_XY_RX`, `EXCH_UV_XYZ_RX` and `EXCH_XY_RX`. They override the standard exchange routines when `genmake2` is run with `exch2` option. They in turn call the local exch2 subroutines `EXCH2_UV_XY_RX` and `EXCH2_UV_XYZ_RX` for two and three-dimensional vector quantities, and `EXCH2_XY_RX` and `EXCH2_XYZ_RX` for two and three-dimensional scalar quantities. These subroutines set the dimensions of the area to be exchanged, call `EXCH2_RX1_CUBE` for scalars and `EXCH2_RX2_CUBE` for vectors, and then handle the singularities at the cube corners.

The separate scalar and vector forms of `EXCH2_RX1_CUBE` and `EXCH2_RX2_CUBE` reflect that the vector-handling subroutine needs to pass both the $u$ and $v$ components of the physical vectors. This swapping arises from the topological folding discussed above, where the $x$ and $y$ axes get swapped in some cases, and is not an issue with the scalar case. These subroutines call `EXCH2_SEND_RX1` and `EXCH2_SEND_RX2`, which do most of the work using the variables discussed above.

8.2.5.7 Experiments and tutorials that use exch2

- Held Suarez tutorial, in `verification/tutorial_held_suarez_cs` verification directory.

8.2.6 Gridalt - Alternate Grid Package

8.2.6.1 Introduction

The gridalt package [Mol09] is designed to allow different components of MITgcm to be run using horizontal and/or vertical grids which are different from the main model grid. The gridalt routines handle the definition of the all the various alternative grid(s) and the mappings between them and the MITgcm grid. The implementation of the gridalt package which allows the high end atmospheric physics (fizhi) to be run on a high resolution and quasi terrain-following vertical grid is documented here. The package has also (with some user modifications) been used for other calculations within the GCM.

The rationale for implementing the atmospheric physics on a high resolution vertical grid involves the fact that the MITgcm $p^*$ (or any pressure-type) coordinate cannot maintain the vertical resolution near the surface as the bottom topography rises above sea level. The vertical length scales near the ground are small and can vary on small time scales, and the vertical grid must be adequate to resolve them. Many studies with both regional and global atmospheric models have demonstrated the improvements in the simulations when the vertical resolution near the surface is increased ( ). Some of the benefit of increased resolution near the surface is realized by employing the higher resolution for the computation of the forcing due to turbulent and convective processes in the atmosphere.

The parameterizations of atmospheric subgrid scale processes are all essentially one-dimensional in nature, and the computation of the terms in the equations of motion due to these processes can be performed for the air column over one grid point at a time. The vertical grid on which these computations take place can therefore be entirely independant of the grid on which the equations of motion are integrated, and the 'tendency' terms can be interpolated to the vertical grid on which the equations of motion are integrated. A modified $p^*$ coordinate, which adjusts to the local terrain and adds additional levels between the lower levels of the existing $p^*$ grid (and perhaps between the levels near the tropopause as well), is implemented. The vertical discretization is different for each grid point, although it consist of the same number of levels. Additional 'sponge' levels aloft are added when needed. The levels of the physics grid are constrained to fit exactly into the existing $p^*$ grid, simplifying the mapping between the two vertical coordinates. This is illustrated as follows:
The algorithm presented here retains the state variables on the high resolution ‘physics’ grid as well as on the coarser resolution ‘dynamics’ grid, and ensures that the two estimates of the state ‘agree’ on the coarse resolution grid. It would have been possible to implement a technique in which the tendencies due to atmospheric physics are computed on the high resolution grid and the state variables are retained at low resolution only. This, however, for the case of the turbulence parameterization, would mean that the turbulent kinetic energy source terms, and all the turbulence terms that are written in terms of gradients of the mean flow, cannot really be computed making use of the fine structure in the vertical.

### 8.2.6.2 Equations on Both Grids

In addition to computing the physical forcing terms of the momentum, thermodynamic and humidity equations on the modified (higher resolution) grid, the higher resolution structure of the atmosphere (the boundary layer) is retained between physics calculations. This necessitates a second set of evolution equations for the atmospheric state variables on the modified grid. If the equation for the evolution of $U$ on $p^*$ can be expressed as:

$$\frac{\partial U}{\partial t} \bigg|_{p^*} = \frac{\partial U}{\partial t} \bigg|_{p^*}^{\text{dynamics}} + \frac{\partial U}{\partial t} \bigg|_{p^*}^{\text{physics}}$$

where the physics forcing terms on $p^*$ have been mapped from the modified grid, then an additional equation to govern the evolution of $U$ (for example) on the modified grid is written:

$$\frac{\partial U}{\partial t} \bigg|_{p^* \text{m}} = \frac{\partial U}{\partial t} \bigg|_{p^* \text{m}}^{\text{dynamics}} + \frac{\partial U}{\partial t} \bigg|_{p^* \text{m}}^{\text{physics}} + \gamma (U \big|_{p^*} - U \big|_{p^* \text{m}})$$

where $p^* \text{m}$ refers to the modified higher resolution grid, and the dynamics forcing terms have been mapped from $p^*$ space. The last term on the RHS is a relaxation term, meant to constrain the state variables on the modified vertical grid to ‘track’ the state variables on the $p^*$ grid on some time scale, governed by $\gamma$. In the present implementation, $\gamma = 1$, requiring an immediate agreement between the two ‘states’.
8.2.6.3 Time stepping Sequence

If we write $T_{\text{phys}}$ as the temperature (or any other state variable) on the high resolution physics grid, and $T_{\text{dyn}}$ as the temperature on the coarse vertical resolution dynamics grid, then:

1. Compute the tendency due to physics processes.
2. Advance the physics state: $T^{n+1}_{\text{phys}}(l) = T^n_{\text{phys}}(l) + \delta T_{\text{phys}}$.
3. Interpolate the physics tendency to the dynamics grid, and advance the dynamics state by physics and dynamics tendencies: $T^{n+1}_{\text{dyn}}(L) = T^n_{\text{dyn}}(L) + \delta T_{\text{dyn}}(L) + [\delta T_{\text{phys}}(l)](L)$.
4. Interpolate the dynamics tendency to the physics grid, and update the physics grid due to dynamics tendencies: $T^{n+1}_{\text{phys}}(l) = T^{n+1}_{\text{phys}}(l) + \delta T_{\text{dyn}}(L)(l)$.
5. Apply correction term to physics state to account for divergence from dynamics state: $T^{n+1}_{\text{phys}}(l) = T^{n+1}_{\text{phys}}(l) + \gamma \{ T_{\text{dyn}}(L) - \{ T_{\text{phys}}(l) \} (L) \} (l)$. Where $\gamma = 1$ here.

8.2.6.4 Interpolation

In order to minimize the correction terms for the state variables on the alternative, higher resolution grid, the vertical interpolation scheme must be constructed so that a dynamics-to-physics interpolation can be exactly reversed with a physics-to-dynamics mapping. The simple scheme employed to achieve this is:

Coarse to fine: For all physics layers $l$ in dynamics layer $L$, $T_{\text{phys}}(l) = \{ T_{\text{dyn}}(L) \} = T_{\text{dyn}}(L)$.

Fine to coarse: For all physics layers $l$ in dynamics layer $L$, $T_{\text{dyn}}(L) = \{ T_{\text{phys}}(l) \} = \int T_{\text{phys}} dp$.

Where $\{ \}$ is defined as the dynamics-to-physics operator and $\{ \}$ is the physics-to-dynamics operator, $T$ stands for any state variable, and the subscripts $\text{phys}$ and $\text{dyn}$ stand for variables on the physics and dynamics grids, respectively.

8.2.6.5 Key subroutines, parameters and files

One of the central elements of the gridalt package is the routine which is called from subroutine gridalt_initialise to define the grid to be used for the high end physics calculations. Routine make_phys_grid passes back the parameters which define the grid, ultimately stored in the common block gridalt_mapping.

```fortran
subroutine make_phys_grid(drF,hfacC,im1,im2,jm1,jm2,Nr,
   . Nsx,Nsy,i1,i2,j1,j2,bi,bj,Nrphys,Lbot,dpphys,numlevphys,nlperdyn)
```

Purpose: Define the grid that will be used to run the high-end atmospheric physics.

Algorithm: Fit additional levels of some (~) known thickness in between existing levels of the grid used for the dynamics

Need: Information about the dynamics grid vertical spacing

Input: drF - delta r (p*) edge-to-edge

hfacC - fraction of grid box above topography

im1, im2 - beginning and ending i - dimensions

jm1, jm2 - beginning and ending j - dimensions

Nr - number of levels in dynamics grid

Nsx,Nsy, i1,i2,j1,j2,bi,bj, Nrphys, Lbot, dpphys, numlevphys, nlperdyn

(continues on next page)
8.2. Packages Related to Hydrodynamical Kernel 419

In the case of the grid used to compute the atmospheric physical forcing (Fizhi: High-end Atmospheric Physics), the locations of the grid points move in time with the MITgcm $p^*$ coordinate, and subroutine gridalt_update is called during the run to update the locations of the grid points:

```fortran
subroutine gridalt_update(myThid)
C******************************************************************************
C Purpose: Update the pressure thicknesses of the layers of the alternative vertical grid (used now for atmospheric physics).  
C******************************************************************************
C Calculate: ddphys - new delta r ($p^*$) edge-to-edge of physics grid  
C using ddphys0 (initial value) and rstarfacC
C******************************************************************************
```

The gridalt package also supplies utility routines which perform the mappings from one grid to the other. These routines are called from the code which computes the fields on the alternative (fizhi) grid.

```fortran
subroutine dyn2phys(qdyn,pedyn,im1,im2,jm1,jm2,lmdyn,Nsx,Nsy,  
    . idim1,idim2,jdim1,jdim2,bi,bj,windphy,pephy,Lbot,lmphy,nlperdyn,  
    . flg,qphy)
C******************************************************************************
C Purpose:  
C To interpolate an arbitrary quantity from the 'dynamics' eta ($p^*$) grid to the higher resolution physics grid  
C Algorithm:  
C Routine works one layer (edge to edge pressure) at a time.  
C Dynamics -> Physics retains the dynamics layer mean value,  
C weights the field either with the profile of the physics grid wind speed (for U and V fields), or uniformly (T and Q)  
C Input:  
C qdyn..... [im,jm,lmdyn] Arbitrary Quantity on Input Grid  
C pedyn..... [im,jm,lmdyn+1] Pressures at bottom edges of input levels  
C im1,2 ... Limits for Longitude Dimension of Input  
C jm1,2 ... Limits for Latitude Dimension of Input
C******************************************************************************
```
And similarly, gridalt contains subroutine phys2dyn.

### 8.2.6.6 Gridalt Diagnostics

<table>
<thead>
<tr>
<th>&lt;-Name-&gt;</th>
<th>Levs</th>
<th>&lt;-parsing code-&gt;</th>
<th>&lt;- Units -&gt;</th>
<th>&lt;- Tile (max=80c)</th>
</tr>
</thead>
<tbody>
<tr>
<td>DPPHYS</td>
<td>20</td>
<td>SM ML</td>
<td>Pascal</td>
<td>Pressure Thickness of Layers on Fizhi_Grid</td>
</tr>
</tbody>
</table>

### 8.2.6.7 Dos and donts

### 8.2.6.8 Gridalt Reference

### 8.2.6.9 Experiments and tutorials that use gridalt

- Fizhi experiment, in verification/fizhi-cs-32x32x10 verification directory

### 8.3 General purpose numerical infrastructure packages

#### 8.3.1 OBCS: Open boundary conditions for regional modeling

Authors: Alistair Adcroft, Patrick Heimbach, Samar Katiwala, Martin Losch
8.3.1.1 Introduction

The OBCS-package (pkg/obcs) is fundamental to regional ocean modeling with the MITgcm, but there are so many details to be considered in regional ocean modeling that this package cannot accommodate all imaginable and possible options. Therefore, for a regional simulation with very particular details it is recommended to familiarize oneself not only with the compile-time and run-time options of this package, but also with the code itself. In many cases it will be necessary to adapt the obcs-code (in particular S/R OBCS_CALC) to the application in question; in these cases pkg/obcs (together with the pkg/rbcs, see Section 8.3.2) is a very useful infrastructure for implementing special regional models.

8.3.1.2 OBCS configuration and compiling

As with all MITgcm packages, OBCS can be turned on or off at compile-time

- using the packages.conf file by adding obcs to it
- or using genmake2 adding -enable=obcs or -disable=obcs switches
- Required packages and CPP options:
  - Two alternatives are available for prescribing open boundary values, which differ in the way how OB’s are treated in time:
    * Simple time-management (e.g., constant in time, or cyclic with fixed frequency) is provided through S/R OBCS_FIELDS_LOAD
    * More sophisticated ‘real-time’ (i.e. calendar time) management is available through S/R OBCS_PRESCRIBE_READ
  - The latter case requires packages pkg/cal and pkg/exf to be enabled.

Parts of the OBCS code can be enabled or disabled at compile-time via CPP preprocessor flags. These options are set in OBCS_OPTIONS.h. Table 8.1 summarizes these options.

<table>
<thead>
<tr>
<th>CPP option</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALLOW_OBCS_NORTH</td>
<td>#define</td>
<td>enable Northern OB</td>
</tr>
<tr>
<td>ALLOW_OBCS_SOUTH</td>
<td>#define</td>
<td>enable Southern OB</td>
</tr>
<tr>
<td>ALLOW_OBCS_EAST</td>
<td>#define</td>
<td>enable Eastern OB</td>
</tr>
<tr>
<td>ALLOW_OBCS_WEST</td>
<td>#define</td>
<td>enable Western OB</td>
</tr>
<tr>
<td>ALLOW_OBCS_PRESCRIBE</td>
<td>#define</td>
<td>enable code for prescribing OB’s</td>
</tr>
<tr>
<td>ALLOW_OBCS_Sponge</td>
<td>#undef</td>
<td>enable sponge layer code</td>
</tr>
<tr>
<td>ALLOW_OBCS_BALANCE</td>
<td>#define</td>
<td>enable code for balancing transports through OB’s</td>
</tr>
<tr>
<td>ALLOW_ORLANSKI</td>
<td>#define</td>
<td>enable Orlanski radiation conditions at OB’s</td>
</tr>
<tr>
<td>ALLOW_OBCS_STEVENS</td>
<td>#undef</td>
<td>enable Stevens (1990) boundary conditions at OB’s</td>
</tr>
<tr>
<td>ALLOW_OBCS_SEAICE_Sponge</td>
<td>#undef</td>
<td>Include hooks to sponge layer treatment of pkg/seaice variables</td>
</tr>
<tr>
<td>ALLOW_OBCS_TIDES</td>
<td>#undef</td>
<td>Add tidal contributions to normal OB flow (At the moment tidal forcing is applied only to “normal” flow)</td>
</tr>
</tbody>
</table>

8.3. General purpose numerical infrastructure packages
8.3.1.3 Run-time parameters

Run-time parameters are set in files data.pkg, data.obcs, and data.exf if 'real-time' prescription is requested (i.e., pkg/exf enabled). These parameter files are read in S/Rs PACKAGES_READPARMS, OBCS_READPARMS, and EXF_READPARMS, respectively. Run-time parameters may be broken into three categories:

1. switching on/off the package at runtime
2. OBCS package flags and parameters
3. additional timing flags in data.exf if selected.

Enabling the package

The OBCS package is switched on at runtime by setting useOBCS = .TRUE. in data.pkg.

Package flags and parameters

Table 8.2 summarizes the runtime flags that are set in data.obcs and their default values.
Table 8.2: OBCS runtime parameters

<table>
<thead>
<tr>
<th>Flag/parameter</th>
<th>default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OB_Jnorth</td>
<td>0</td>
<td>Nx-vector of J-indices (w.r.t. Ny) of Northern OB at each I-position (w.r.t. Nx)</td>
</tr>
<tr>
<td>OB_Jsouth</td>
<td>0</td>
<td>Nx-vector of J-indices (w.r.t. Ny) of Southern OB at each I-position (w.r.t. Nx)</td>
</tr>
<tr>
<td>OB_Ieast</td>
<td>0</td>
<td>Ny-vector of I-indices (w.r.t. Nx) of Eastern OB at each J-position (w.r.t. Ny)</td>
</tr>
<tr>
<td>OB_Iwest</td>
<td>0</td>
<td>Ny-vector of I-indices (w.r.t. Nx) of Western OB at each J-position (w.r.t. Ny)</td>
</tr>
<tr>
<td>useOBCSsprescribe</td>
<td>FALSE</td>
<td></td>
</tr>
<tr>
<td>useOBCSsponge</td>
<td>FALSE</td>
<td></td>
</tr>
<tr>
<td>useOBCSbalance</td>
<td>FALSE</td>
<td></td>
</tr>
<tr>
<td>OBCS_balanceFacN,</td>
<td>1</td>
<td>Factor(s) determining the details of the balancing code</td>
</tr>
<tr>
<td>OBCS_balanceFacS,</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OBCS_balanceFacE,</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OBCS_balanceFacW</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OBCSBalanceSurf</td>
<td>FALSE</td>
<td>include surface mass flux in balance</td>
</tr>
<tr>
<td>useOrlanskiNorth, useOrlan-</td>
<td>FALSE</td>
<td>Turn on Orlanski boundary conditions for individual boundary.</td>
</tr>
<tr>
<td>skiSouth, useOrlanskiEast,</td>
<td></td>
<td></td>
</tr>
<tr>
<td>useOrlanskiWest</td>
<td></td>
<td></td>
</tr>
<tr>
<td>useStevensNorth, useStevensSouth, useStevensEast, useStevensWest</td>
<td>FALSE</td>
<td>Turn on Stevens boundary conditions for individual boundary</td>
</tr>
<tr>
<td>OBxyFile</td>
<td></td>
<td>File name of OB field: X: N(orth), S(outh), E(ast), W(est) y: t(temperature), s(salinity), eta (sea surface height), u(-velocity), v(-velocity), w(-velocity), a (sea ice area), h (sea ice thickness), sn (snow thickness), sl (sea ice salinity)</td>
</tr>
<tr>
<td>Orlanski Parameters</td>
<td></td>
<td></td>
</tr>
<tr>
<td>cvelTimeScale</td>
<td>2000.0</td>
<td>Averaging period for phase speed (seconds)</td>
</tr>
<tr>
<td>cMAX</td>
<td>0.45</td>
<td>Maximum allowable phase speed-CFL for AB-II (m/s)</td>
</tr>
<tr>
<td>CFIX</td>
<td>0.8</td>
<td>Fixed boundary phase speed (m/s)</td>
</tr>
<tr>
<td>useFixedCEast</td>
<td>FALSE</td>
<td></td>
</tr>
<tr>
<td>useFixedCWest</td>
<td>FALSE</td>
<td></td>
</tr>
<tr>
<td>Sponge layer parameters</td>
<td></td>
<td></td>
</tr>
<tr>
<td>spongeThickness</td>
<td>0</td>
<td>sponge layer thickness (in grid points)</td>
</tr>
<tr>
<td>Urelaxobcsinner</td>
<td>0.0</td>
<td>relaxation time scale at the innermost sponge layer point of a meridional OB (s)</td>
</tr>
<tr>
<td>Vrelaxobcsinner</td>
<td>0.0</td>
<td>relaxation time scale at the innermost sponge layer point of a zonal OB (s)</td>
</tr>
<tr>
<td>Urelaxobcsbound</td>
<td>0.0</td>
<td>relaxation time scale at the outermost sponge layer point of a meridional OB (s)</td>
</tr>
<tr>
<td>Vrelaxobcsbound</td>
<td>0.0</td>
<td>relaxation time scale at the outermost sponge layer point of a zonal OB (s)</td>
</tr>
<tr>
<td>Stevens parameters</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TrelaxStevens SrelaxStevens</td>
<td>0</td>
<td>Relaxation time scale for temperature/salinity (s)</td>
</tr>
<tr>
<td>useStevensPhaseVel</td>
<td>TRUE</td>
<td></td>
</tr>
<tr>
<td>useStevensAdvection</td>
<td>TRUE</td>
<td></td>
</tr>
</tbody>
</table>

8.3. General purpose numerical infrastructure packages  423
8.3.1.4 Defining open boundary positions

There are up to four open boundaries (OBs): Northern, Southern, Eastern, and Western. All OB locations are specified by their absolute meridional (Northern/Southern) or zonal (Eastern/Western) indices. Thus, for each zonal position \( i = 1 \ldots N_x \), a meridional index \( j \) specifies the Northern/Southern OB position, and for each meridional position \( j = 1 \ldots N_y \), a zonal index \( i \) specifies the Eastern/Western OB position. For Northern/Southern OB this defines an \( N_x \)-dimensional “row” array \( OB_{Jnorth}(Nx) \) / \( OB_{Jsouth}(Nx) \) and an \( N_y \)-dimensional “column” array \( OB_{Ieast}(Ny) \) / \( OB_{Iwest}(Ny) \). Positions determined in this way allows Northern/Southern OBs to be at variable \( j \) (or \( y \)) positions and Eastern/Western OBs at variable \( i \) (or \( x \)) positions. Here indices refer to tracer points on the C-grid. A zero (0) element in \( OB_{I...} \) / \( OB_{J...} \) means there is no corresponding OB in that column/row. By default all elements in \( OB_{I...} \) / \( OB_{J...} \) are zero. For a Northern/Southern OB, the OB V-point is to the South/North. For an Eastern/Western OB, the OB U-point is to the West/East. For example

\[ OB_{Jnorth}(3)=34 \] means that:
- \( T(3,34) \) is a OB point
- \( U(3,34) \) is a OB point
- \( V(3,34) \) is a OB point

\[ OB_{Jsouth}(3)=1 \] means that:
- \( T(3,1) \) is a OB point
- \( U(3,1) \) is a OB point
- \( V(3,2) \) is a OB point

\[ OB_{Ieast}(10)=69 \] means that:
- \( T(69,10) \) is a OB point
- \( U(69,10) \) is a OB point
- \( V(69,10) \) is a OB point

\[ OB_{Iwest}(10)=1 \] means that:
- \( T(1,10) \) is a OB point
- \( U(2,10) \) is a OB point
- \( V(1,10) \) is a OB point

For convenience, negative values for \( OB_{Jnorth} \) / \( OB_{Ieast} \) refer to points relative to the Northern/Eastern edges of the model, e.g. \( OB_{Jnorth}(3)=-1 \) means that the point \((3,Ny)\) is a northern OB and \( OB_{Ieast}(3)=-5 \) means that the point \((3,Nx-5)\) is an eastern OB.

Simple examples

For a model grid with \( N_x \times N_y = 120 \times 144 \) horizontal grid points with four open boundaries along the four edges of the domain, the simplest way of specifying the boundary points:

\[
\begin{align*}
OB_{Ieast} &= 144*-1, \\
# or OB_{Ieast} &= 144*120, \\
OB_{Iwest} &= 144*1, \\
OB_{Jnorth} &= 120*-1, \\
# or OB_{Jnorth} &= 120*144, \\
OB_{Jsouth} &= 120*1,
\end{align*}
\]
When the boundaries are in single rows or columns as in the above example, the same can be achieved with the convenient parameters `OB_singleInorth / OB_singleJsouth / OB_singleIeast / OB_singleIwest`:

```
OB_singleIeast  = -1,
OB_singleIwest  = 1,
OB_singleJnorth = -1,
OB_singleJsouth = 1,
```

If only the first 50 grid points of the southern boundary are boundary points:

```
OB_Jsouth(1:50) = 50*1,
```

### A more complex example

Open boundaries are not restricted to single rows or columns. Each OB can be distributed in different rows and columns resulting in OBs consisting of the combination of different types of open boundaries (i.e., N, S, E and W). Figure 8.6 displays such an OB located on the left-bottom corner of a domain. Note there are five boundary points defined by southern and western boundaries. In particular, there are five southern boundary (blue lines) and two western boundaries points (red lines). For the boundary displayed in Figure 8.6 and the same dimensions as in the previous example (i.e. 120 × 144 grid points), the namelist looks like this:

```
OB_Iwest   = 1*0,1*5,142*0,
OB_Jsouth = 2*3,3*2,115*0,
```

![Figure 8.6: Example boundary with more than one row. The dark grey, light grey, and white boxes are points outside the domain, OB points, and ocean points, respectively. The black dots mark the OB index to write into the namelist.](image)

For an even more complicated open boundary geometry, e.g., delimiting a concave interior domain (OB_Ieast ≤ OB_Iwest), one might need to also specify the interior domain through an additional input file `insideOBmaskFile` for the interior mask (1 inside, 0 outside).
8.3.1.5 Equations and key routines

OBCS_READPARMS:

Set OB positions through arrays OB_Inorth(Nx), OB_Isouth(Nx), OB_Ieast(Ny), OB_Iwest(Ny) and runtime flags (see Table Table 8.2).

OBCS_CALC:

Top-level routine for filling values to be applied at OB for $T, S, U, V, \eta$ into corresponding “slice” arrays $(x, z)$ $(y, z)$ for each OB: OB[N/S/E/W][t/s/u/v]; e.g. for the salinity array at the Southern OB, the array name is OBSs. Values filled are either

- constant vertical $T, S$ profiles as specified in file data (tRef(Nr), sRef(Nr)) with zero velocities $U, V$
- $T, S, U, V$ values determined via Orlanski radiation conditions (see below)
- prescribed time-constant or time-varying fields (see below).
- prescribed boundary fields to compute Stevens boundary conditions.

ORLANSKI:

Orlanski radiation conditions [Orl76] examples can be found in example configurations verification/dome and verification/tutorial_plume_on_slope (as described in detail in Section 4.9).

OBCS_PRESCRIBE_READ:

When useOBCSprecribe = .TRUE. the model tries to read temperature, salinity, u- and v-velocities from files specified in the runtime parameters OB[N/S/E/W][t/s/u/v]File. These files are the usual IEEE, big-endian files with dimensions of a section along an open boundary:

- For North/South boundary files the dimensions are $(N_x \times N_r \times$ time levels), for East/West boundary files the dimensions are $(N_y \times N_r \times$ time levels).
- If a non-linear free surface is used (Section 2.10.2), additional files OB[N/S/E/W]etaFile for the sea surface height $\eta$ with dimension $(N_x/y \times$ time levels) may be specified.
- If non-hydrostatic dynamics are used (Section 2.9), additional files OB[N/S/E/W]wFile for the vertical velocity $w$ with dimensions $(N_x/y \times N_r \times$ time levels) can be specified.
- If useSEAICE = .TRUE. then additional files OB[N/S/E/W][a,h,sl,sn,uice,vice] for sea ice area, thickness (HEFF), seaice salinity, snow and ice velocities $(N_x/y \times$ time levels) can be specified.

As in external_fields_load.F or as done in pkg/exf, the code reads two time levels for each variable, e.g., OBNu0 and OBNu1, and interpolates linearly between these time levels to obtain the value OBNu at the current model time (step). When pkg/exf is used, the time levels are controlled for each boundary separately in the same way as the pkg/exf fields in data.exf, namelist EXF_NML_OBCS. The run-time flags follow the above naming conventions, e.g., for the western boundary the corresponding flags are OBCSWstartdate1, OBCSWstartdate2 and OBCSWperiod. Sea-ice boundary values are controlled separately with siobWstartdate1, siobWstartdate2 and siobWperiod. When pkg/exf is not used the time levels are controlled by the runtime flags externForcingPeriod and externForcingCycle in data; see verification/exp4/input/data for an example.
OBCS_CALC_STEVENS:

The boundary conditions following [Ste90] require the vertically averaged normal velocity (originally specified as a stream function along the open boundary) \( \bar{u}_{ob} \) and the tracer fields \( \chi_{ob} \) (note: passive tracers are currently not implemented and the code stops when package ptracers is used together with this option). Currently the code vertically averages the normal velocity as specified in OB[E,W]u or OB[N,S]v. From these prescribed values the code computes the boundary values for the next timestep \( n + 1 \) as follows (as an example, we use the notation for an eastern or western boundary):

- \( u^{n+1}(y, z) = \bar{u}_{ob}(y) + (u')^n(y, z) \) where \( (u')^n \) is the deviation from the vertically averaged velocity at timestep \( n \) on the boundary. \( (u')^n \) is computed in the previous time step \( n \) from the intermediate velocity \( u^* \) prior to the correction step (see Section 2.2 equation (2.12)). (This velocity is not available at the beginning of the next time step \( n + 1 \), when S/Rs OBCS_CALC and OBCS_CALC_STEVENS are called, therefore it needs to be saved in S/R DYNAMICS by calling S/R OBCS_SAVE_UV_N and also stored in a separate restart files pickup_stevens[N/S/E/W].$\{iteration\}.data)

- If \( u^{n+1} \) is directed into the model domain, the boudary value for tracer \( \chi \) is restored to the prescribed values:
  \[
  \chi^{n+1} = \chi^n + \frac{\Delta t}{\tau_\chi} (\chi_{ob} - \chi^n)
  \]
  where \( \tau_\chi \) is the relaxation time scale (either TrelaxStevens or SrelaxStevens). The new \( \chi^{n+1} \) is then subject to the advection by \( u^{n+1} \).

- If \( u^{n+1} \) is directed out of the model domain, the tracer \( \chi^{n+1} \) on the boundary at timestep \( n + 1 \) is estimated from advection out of the domain with \( u^{n+1} + c \), where \( c \) is a phase velocity estimated as \( \frac{1}{2} \frac{\partial \chi}{\partial t} / \frac{\partial \chi}{\partial x} \). The numerical scheme is (as an example for an eastern boundary):
  \[
  \chi^{n+1}_{i_b,j,k} = \chi^n_{i_b,j,k} + \Delta t (u^{n+1} + c)_{i_b,j,k} \frac{\chi^n_{i_b,j,k} - \chi^n_{i_b-1,j,k}}{\Delta x_{i_b,j}} \text{ if } u^{n+1}_{i_b,j,k} > 0
  \]
  where \( i_b \) is the boundary index. For test purposes, the phase velocity contribution or the entire advection can be turned off by setting the corresponding parameters useStevensPhaseVel and useStevensAdvection to .FALSE..

See [Ste90] for details. With this boundary condition specifying the exact net transport across the open boundary is simple, so that balancing the flow with (S/R OBCS_BALANCE_FLOW see next paragraph) is usually not necessary.

Special cases where the current implementation is not complete:

- When you use the non-linear free surface option (parameter nonlinFreeSurf > 1), the current implementation just assumes that the gradient normal to the open boundary is zero \( (\frac{\partial \eta}{\partial n} = 0) \). Although this is inconsistent with geostrophic dynamics and the possibility to specify a non-zero tangent velocity together with Stevens BCs for normal velocities, it seems to work. Recommendation: Always specify zero tangential velocities with Stevens BCs.

- There is no code for passive tracers, just a commented template in S/R OBCS_CALC_STEVENS. This means that passive tracers can be specified independently and are fluxed with the velocities that the Stevens BCs compute, but without the restoring term.

- There are no specific Stevens BCs for sea ice, e.g., pkg/seaice. The model uses the default boundary conditions for the sea ice packages.

8.3. General purpose numerical infrastructure packages 427
OBCS_BALANCE_FLOW:

When turned on (CPP option ALLOW_OBCS_BALANCE defined in OBCS_OPTIONS.h and useOBCSbalance set to .TRUE. in data.obcs/OBCS_PARM01), this routine balances the net flow across the open boundaries. By default the net flow across the boundaries is computed and all normal velocities on boundaries are adjusted to obtain zero net inflow.

This behavior can be controlled with the runtime flags OBCS_balanceFacN, OBCS_balanceFacS, OBCS_balanceFacE, and OBCS_balanceFacW. The values of these flags determine how the net inflow is redistributed as small correction velocities between the individual sections. A value -1 balances an individual boundary, values >0 determine the relative size of the correction. For example, the values

OBCS_balanceFacE = 1.,
OBCS_balanceFacW = -1.,
OBCS_balanceFacN = 2.,
OBCS_balanceFacS = 0.,

make the model

- correct Western OBWu by substracting a uniform velocity to ensure zero net transport through the Western open boundary;
- correct Eastern and Northern normal flow, with the Northern velocity correction two times larger than the Eastern correction, but not the Southern normal flow, to ensure that the total inflow through East, Northern, and Southern open boundary is balanced.

The old method of balancing the net flow for all sections individually can be recovered by setting all flags to -1. Then the normal velocities across each of the four boundaries are modified separately, so that the net volume transport across each boundary is zero. For example, for the western boundary at \(i = i_b\), the modified velocity is:

\[
 u(y, z) - \int_{	ext{western boundary}} u dy dz \approx OBNu(jk) - \sum_{jk} OBNu(jk)h_w(i_b,jk)\Delta y_G(i_b,j)\Delta z(k).
\]

This also ensures a net total inflow of zero through all boundaries, but this combination of flags is not useful if you want to simulate, for example, a sector of the Southern Ocean with a strong ACC entering through the western and leaving through the eastern boundary, because the value of -1 for these flags will make sure that the strong inflow is removed. Clearly, global balancing with \(OBCS_balanceFacE/W/N/S \geq 0\) is the preferred method.

Setting runtime parameter OBCSbalanceSurf to .TRUE., the surface mass flux contribution, say, from surface freshwater flux \(EmPmR\) is included in the balancing scheme.

OBCS_APPLY_+:

OBCS_Sponge:

The sponge layer code (turned on with CPP option ALLOW_OBCS_Sponge and run-time parameter useOBCSSponge) adds a relaxation term to the right-hand-side of the momentum and tracer equations. The variables are relaxed towards the boundary values with a relaxation time scale that increases linearly with distance from the boundary

\[
 G^{(sponge)}_\chi = \frac{-\chi - [(L - \delta L)\chi_{BC} + \delta L\chi_l]/L}{[(L - \delta L)\tau_b + \delta L\tau_l]/L} = \frac{-\chi - [(1 - l)\chi_{BC} + l\chi_l]}{[(1 - l)\tau_b + l\tau_l]}
\]

where \(\chi\) is the model variable (U/V/T/S) in the interior, \(\chi_{BC}\) the boundary value, \(L\) the thickness of the sponge layer (runtime parameter spongeThickness in number of grid points), \(\delta L \in [0, L]\) \((\delta L = \frac{\Delta L}{L} = l \in [0, 1])\) the distance from the boundary (also in grid points), and \(\tau_b\) (runtime parameters Urelaxobcsbound and Vrelaxobcsbound) and \(\tau_l\) (runtime...
parameters $U_{\text{relaxobcsinner}}$ and $V_{\text{relaxobcsinner}}$) the relaxation time scales on the boundary and at the interior termination of the sponge layer. The parameters $U_{\text{relaxobcsbound}}$ and $U_{\text{relaxobcsinner}}$ set the relaxation time scales for the Eastern and Western boundaries, $V_{\text{relaxobcsbound}}$ and $V_{\text{relaxobcsinner}}$ for the Northern and Southern boundaries.

**OB’s with nonlinear free surface**

**OB’s with sea ice**

### 8.3.1.6 Flow chart

```plaintext
C !CALLING SEQUENCE:
C ...
```

### 8.3.1.7 OBCS diagnostics

Diagnostics output is available via the diagnostics package (see Section 9.1). Available output fields are summarized below:

<table>
<thead>
<tr>
<th>Name</th>
<th>Levs</th>
<th>grid</th>
<th>Units</th>
<th>Tile (max=80c)</th>
</tr>
</thead>
</table>

### 8.3.1.8 Experiments and tutorials that use obcs

In the directory `verification` the following experiments use `pkg/obcs`:

- `exp4`: box with 4 open boundaries, simulating flow over a Gaussian bump based on also tests Stevens-boundary conditions;
- `dome`: based on the project “Dynamics of Overflow Mixing and Entrainment” uses Orlanski-BCs;
- `internal_wave`: uses a heavily modified S/R OBCS_CALC
- `seaice_obcs`: simple example how to use the sea-ice related code based on `lab_sea`;
- Tutorial `Gravity Plume On a Continental Slope`: uses Orlanski-BCs.

### 8.3.2 RBCS Package

#### 8.3.2.1 Introduction

`pkg/rbcs` provides the flexibility to relax fields (temperature, salinity, tracers, horizontal velocities) in any 3-D location, thus can be used to implement a sponge layer, or a “source” anywhere in the domain, among other uses.

For a field $(T)$ at every grid point the tendency is modified so that:

$$\frac{dT}{dt} = \frac{dT}{dt} - \frac{M_{rbc}}{\tau_T}(T - T_{rbc})$$

where $M_{rbc}$ is a 3-D mask (no time dependence) between 0 and 1. Values are relaxed to a field given by $T_{rbc}$, which can be set to vary in time. Where $M_{rbc}(x, y, z) = 1.0$, the relaxing timescale is $\tau_T$; where $M_{rbc}(x, y, z) = 0.0$, there is no relaxing. If $M_{rbc}(x, y, z) = 0.5$ then relaxation occurs but at a timescale of $2\tau_T$. 

---

**8.3. General purpose numerical infrastructure packages**
A separate mask can be used for $\theta$, $S$ and ptracers and each of these can be relaxed or not and can have its own timescale $\tau_r$. These are set in data.rbcs (see below).

### 8.3.2.2 Key subroutines and parameters

The only compile-time parameter you are likely to have to change is in RBCS_SIZE.h, the number of masks, PARAMETER( maskLEN = 3 ) by default, see below.

Table 8.3 summarizes the runtime flags that are set in data.rbcs, and their default values.

<table>
<thead>
<tr>
<th>Flag/Parameter</th>
<th>Group</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rbcsForcingPeriod</td>
<td>PARM01</td>
<td>0.0</td>
<td>time interval between forcing fields (in seconds), zero means constant-in-time forcing</td>
</tr>
<tr>
<td>rbcsForcingCycle</td>
<td>PARM01</td>
<td>0.0</td>
<td>repeat cycle of forcing fields (in seconds), zero means non-cyclic forcing</td>
</tr>
<tr>
<td>rbcsForcingOffset</td>
<td>PARM01</td>
<td>0.0</td>
<td>time offset of forcing fields (in seconds), relative to time averages starting at $t = 0$, i.e., the first forcing record/file is placed at $(\text{rbcsForcingOffset} + \text{rbcsForcingPeriod})/2$; see below for examples</td>
</tr>
<tr>
<td>rbcsSingleTimeFiles</td>
<td>PARM01</td>
<td>FALSE</td>
<td>if .TRUE., forcing fields are given one file per rbcsForcingPeriod</td>
</tr>
<tr>
<td>deltaTrbcs</td>
<td>PARM01</td>
<td>deltaT-clock</td>
<td>time step used to compute the iteration numbers for rbcsSingleTimeFiles = .TRUE.</td>
</tr>
<tr>
<td>rbcsVanishingTime</td>
<td>PARM01</td>
<td>0.0</td>
<td>if rbcsVanishingTime &gt; 0, the relaxation strength reduces linearly to vanish at myTime == rbcsVanishingTime</td>
</tr>
<tr>
<td>rbcsIter0</td>
<td>PARM01</td>
<td>0</td>
<td>shift in iteration numbers used to label files if rbcsSingleTimeFiles = .TRUE. (see below for examples)</td>
</tr>
<tr>
<td>useRBCtemp, useRBCsalt, useRBCuVel, useRBCvVel</td>
<td>PARM01</td>
<td>FALSE</td>
<td>whether to use RBCS for T/S/U/V</td>
</tr>
<tr>
<td>tauRelaxT, tauRelaxS, tauRelaxU, tauRelaxV</td>
<td>PARM01</td>
<td>0.0</td>
<td>timescales (in seconds) for relaxing T/S/U/V ($\tau_T$ in equation above); required if the corresponding useRBCxxx is .TRUE.</td>
</tr>
<tr>
<td>relaxMaskFile (irbc)</td>
<td>PARM01</td>
<td>' '</td>
<td>filename of 3-D file with mask ($M_{rbc}$ above), requires a file for each irbc (1=temperature, 2=salinity, 3=ptracer1, 4=ptracer2, etc); if maskLEN is less than the number of tracers, then relaxMaskFile(maskLEN) is used for all remaining tracers</td>
</tr>
<tr>
<td>relaxMaskUFile, relaxMaskVFile</td>
<td>PARM01</td>
<td>' '</td>
<td>filename of 3-D file with mask for U/V.</td>
</tr>
<tr>
<td>relaxTFile, relaxSFile, relaxUFile, relaxVFile</td>
<td>PARM01</td>
<td>' '</td>
<td>name of file with 3-D relaxation field values ($T_{rbc}$ in equation above); if rbcsSingleTimeFiles = .FALSE., it must have one record for each forcing period, otherwise there must be a separate file for each period with a 10-digit iteration number appended to the file name (see Table 8.4 and examples below)</td>
</tr>
<tr>
<td>useRBCpTrNum (iTrc)</td>
<td>PARM02</td>
<td>FALSE</td>
<td>if .TRUE., use RBCS for the corresponding passive tracer</td>
</tr>
<tr>
<td>tauRelaxPTR (iTrc)</td>
<td>PARM02</td>
<td>0.0</td>
<td>relaxing timescale (seconds) for the corresponding ptracer</td>
</tr>
<tr>
<td>relaxPtracerFile (iTrc)</td>
<td>PARM02</td>
<td>' '</td>
<td>file with relaxation field for the corresponding ptracer</td>
</tr>
</tbody>
</table>
8.3.2.3 Timing of relaxation forcing fields

For constant-in-time relaxation, set `rbcsForcingPeriod = 0`. For time-varying relaxation, Table 8.4 illustrates the relation between model time and forcing fields (either records in one big file or, for `rbcsSingleTimeFiles = .TRUE.`, individual files labeled with an iteration number). With `rbcsSingleTimeFiles = .TRUE.`, this is the same as in the offline package, except that the forcing offset is in seconds.

Table 8.4: Timing of RBCS relaxation fields

<table>
<thead>
<tr>
<th>rbcsSingleTimeFiles</th>
<th><code>c = 0</code></th>
<th><code>c ≠ 0</code></th>
<th>record</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><code>model time</code></td>
<td><code>file number</code></td>
<td><code>file number</code></td>
</tr>
<tr>
<td><code>T</code></td>
<td><code>t_0</code></td>
<td><code>i_0</code></td>
<td><code>i_0 + c / Δt_{rbcs}</code></td>
</tr>
<tr>
<td></td>
<td><code>t_0 + p / 2</code></td>
<td><code>t_0 + p / Δt_{rbcs}</code></td>
<td><code>t_0 + p / Δt_{rbcs}</code></td>
</tr>
<tr>
<td></td>
<td><code>t_0 + p + p / 2</code></td>
<td><code>t_0 + 2p / Δt_{rbcs}</code></td>
<td><code>t_0 + 2p / Δt_{rbcs}</code></td>
</tr>
<tr>
<td></td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td></td>
<td><code>t_0 + c - p / 2</code></td>
<td>...</td>
<td><code>i_0 + c / Δt_{rbcs}</code></td>
</tr>
<tr>
<td></td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

where

- `p = rbcsForcingPeriod`
- `c = rbcsForcingCycle`
- `t_0 = rbcsForcingOffset`
- `i_0 = rbcsIter0`
- `Δt_{rbcs} = deltaTrbcs`

8.3.2.4 Example 1: forcing with time averages starting at `t = 0`

**Cyclic data in a single file**

Set `rbcsSingleTimeFiles = .FALSE.` and `rbcsForcingOffset = 0`, and the model will start by interpolating the last and first records of rbcs data, placed at `−p / 2` and `p / 2`, respectively, as appropriate for fields averaged over the time intervals `[−p, 0]` and `[0, p]`.

**Non-cyclic data, multiple files**

Set `rbcsForcingCycle = 0` and `rbcsSingleTimeFiles = .TRUE.`. With `rbcsForcingOffset = 0`, `rbcsIter0 = 0` and `deltaTrbcs = rbcsForcingPeriod`, the model would then start by interpolating data from files `relax*File.0000000000.data` and `relax*File.0000000001.data`, ..., again placed at `−p / 2` and `p / 2`.

8.3. General purpose numerical infrastructure packages
8.3.2.5 Example 2: forcing with snapshots starting at \( t = 0 \)

**Cyclic data in a single file**

Set \( \text{rbcsSingleTimeFiles} = \text{.FALSE.} \) and \( \text{rbcsForcingOffset} = -p/2 \), and the model will start forcing with the first record at \( t = 0 \).

**Non-cyclic data, multiple files**

Set \( \text{rbcsForcingCycle} = 0 \) and \( \text{rbcsSingleTimeFiles} = \text{.TRUE.} \). In this case, it is more natural to set \( \text{rbcsForcingOffset} = +p/2 \). With \( \text{rbcsIter0} = 0 \) and \( \text{deltaTrbc} = \text{rbcsForcingPeriod} \), the model would then start with data from files \( \text{relax}\*File.0000000000.data \) at \( t = 0 \). It would then proceed to interpolate between this file and files \( \text{relax}\*File.0000000001.data \) at \( t = \text{rbcsForcingPeriod} \).

8.3.2.6 Do’s and Don’ts

8.3.2.7 Reference Material

8.3.2.8 Experiments and tutorials that use rbcs

In the directory, the following experiments use pkg/rbcs:

- tutorial *Southern Ocean Reentrant Channel* makes use of pkg/rbcs at its northern boundary as a sponge layer;
- exp4 simulating flow over a Gaussian bump based on [AHM97], demonstrating relaxation of a ptracer field.

8.3.3 PTRACERS Package

8.3.3.1 Introduction

This is a “passive” tracer package. Passive here means that the tracers don’t affect the density of the water (as opposed to temperature and salinity) so no not actively affect the physics of the ocean. Tracers are initialized, advected, diffused and various outputs are taken care of in this package. For methods to add additional sources and sinks of tracers use the *gchem Package*.

Can use up tp 3843 tracers. But can not use the *diagnostics package* with more than about 90 tracers. Use *utils/matlab/ioLb2num.m* and *num2ioLb.m* to find correspondence between tracer number and tracer designation in the code for more than 99 tracers (since tracers only have two digit designations).

8.3.3.2 Equations

8.3.3.3 Key subroutines and parameters

The only code you should have to modify is: PTRACERS_SIZE.h where you need to set in the number of tracers to be used in the experiment: PTRACERS_num.

Run time parameters set in data.ptracers:

- \( \text{PTRACERS_Iter0} \) which is the integer timestep when the tracer experiment is initialized. If \( \text{nIter0} = \text{PTRACERS_Iter0} \) then the tracers are initialized to zero or from initial files. If \( \text{nIter0} > \text{PTRACERS_Iter0} \) then tracers (and previous timestep tendency terms) are read in from a the ptracers pickup file. Note that tracers of zeros will be carried around if \( \text{nIter0} < \text{PTRACERS_Iter0} \).
• **PTRACERS_numInUse**: number of tracers to be used in the run (needs to be \( \leq \) PTRACERS_num set in PTRACERS_SIZE.h)

• **PTRACERS_dumpFreq**: defaults to dumpFreq (set in data)

• **PTRACERS_taveFreq**: defaults to taveFreq (set in data)

• **PTRACERS_monitorFreq**: defaults to monitorFreq (set in data)

• **PTRACERS_timeave_mnc**: needs useMNC, timeave_mnc, default to false

• **PTRACERS_snapshot_mnc**: needs useMNC, snapshot_mnc, default to false

• **PTRACERS_monitor_mnc**: needs useMNC, monitor_mnc, default to false

• **PTRACERS_pickup_write_mnc**: needs useMNC, pickup_write_mnc, default to false

• **PTRACERS_pickup_read_mnc**: needs useMNC, pickup_read_mnc, default to false

• **PTRACERS_useRecords**: defaults to false. If true, will write all tracers in a single file, otherwise each tracer in a separate file.

The following can be set for each tracer (tracer number iTrc):

• **PTRACERS_advScheme** (iTrc): will default to saltAdvScheme (set in data). For other options see Table MITgcm Advection Schemes.

• **PTRACERS_ImplVertAdv** (iTrc): implicit vertical advection flag, defaults to false.

• **PTRACERS_diffKh** (iTrc): horizontal Laplacian Diffusivity, defaults to diffKhS (set in data).

• **PTRACERS_diffK4** (iTrc): Biharmonic Diffusivity, defaults to diffK4S (set in data).

• **PTRACERS_diffKr** (iTrc): vertical diffusion, defaults to un-set.

• **PTRACERS_diffKrNr** (k,iTrc): level specific vertical diffusion, defaults to diffKrNrS. Will be set to PTRACERS_diffKr if this is set.

• **PTRACERS_ref** (k,iTrc): reference tracer value for each level k, defaults to 0. Currently only used for dilution/concentration of tracers at surface if PTRACERS_EvPrRn (iTrc) is set and convertFW2Salt (set in data) is set to something other than -1 (note default is convertFW2Salt = 35).

• **PTRACERS_EvPrRn** (iTrc): tracer concentration in freshwater. Needed for calculation of dilution/concentration in surface layer due to freshwater addition/evaporation. Defaults to un-set in which case no dilution/concentration occurs.

• **PTRACERS_useGMRedi** (iTrc): apply GM or not. Defaults to useGMRedi.

• **PTRACERS_useKPP** (iTrc): apply KPP or not. Defaults to useKPP.

• **PTRACERS_initialFile** (iTrc): file with initial tracer concentration. Will be used if PTRACERS_Iter0 = nIter0. Default is no name, in which case tracer is initialised as zero. If PTRACERS_Iter0 < nIter0, then tracer concentration will come from pickup_ptracer.

• **PTRACERS_names** (iTrc): tracer name. Needed for netcdf. Defaults to nothing.

• **PTRACERS_long_names** (iTrc): optional name in long form of tracer.

• **PTRACERS_units** (iTrc): optional units of tracer.
### 8.3.3.4 PTRACERS Diagnostics

Note that beyond 99 ptracers, diagnostics will be labeled with letters in addition to numbers, e.g., the diagnostic for the 100th ptracer is TRAC0a, etc.

<table>
<thead>
<tr>
<th>Name</th>
<th>Code</th>
<th>Units</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRAC01</td>
<td>SMR</td>
<td>mol/m^3</td>
<td>Dissolved Inorganic Carbon (DIC) [mol C/m^3]</td>
</tr>
<tr>
<td>UTRAC01</td>
<td>UUr</td>
<td>mol/m^3.m/s</td>
<td>Zonal Mass-Weighted Transp of DIC</td>
</tr>
<tr>
<td>VTRAC01</td>
<td>VUr</td>
<td>mol/m^3.m/s</td>
<td>Merid Mass-Weighted Transp of DIC</td>
</tr>
<tr>
<td>WTRAC01</td>
<td>WM</td>
<td>mol/m^3.m/s</td>
<td>Vert Mass-Weighted Transp of DIC</td>
</tr>
<tr>
<td>ForcTr01</td>
<td>SMR</td>
<td>mol/m^3/s</td>
<td>DIC forcing tendency</td>
</tr>
<tr>
<td>AB_gTr01</td>
<td>SMR</td>
<td>mol/m^3/s</td>
<td>DIC tendency from Adams-Bashforth</td>
</tr>
<tr>
<td>Tp_gTr01</td>
<td>SMR</td>
<td>mol/m^3/s</td>
<td>DIC total transport tendency (before gchem_forcing_)</td>
</tr>
<tr>
<td>ADVrTr01</td>
<td>WM</td>
<td>mol/m^3.m^3/s</td>
<td>Vertical Advective Flux of DIC</td>
</tr>
<tr>
<td>ADVxTr01</td>
<td>UU</td>
<td>mol/m^3.m^3/s</td>
<td>Zonal Advective Flux of DIC</td>
</tr>
<tr>
<td>ADVyTr01</td>
<td>VV</td>
<td>mol/m^3.m^3/s</td>
<td>Meridional Advective Flux of DIC</td>
</tr>
<tr>
<td>DFrETr01</td>
<td>WM</td>
<td>mol/m^3.m^3/s</td>
<td>Vertical Diffusive Flux of DIC (Explicit part)</td>
</tr>
<tr>
<td>DFxETr01</td>
<td>UU</td>
<td>mol/m^3.m^3/s</td>
<td>Zonal Diffusive Flux of DIC</td>
</tr>
<tr>
<td>DFyETr01</td>
<td>VV</td>
<td>mol/m^3.m^3/s</td>
<td>Meridional Diffusive Flux of DIC</td>
</tr>
<tr>
<td>DFrITr01</td>
<td>WM</td>
<td>mol/m^3.m^3/s</td>
<td>Vertical Diffusive Flux of DIC (Implicit part)</td>
</tr>
<tr>
<td>TRAC02</td>
<td>SMR</td>
<td>mol/m^3</td>
<td>Alkalinity (Alk) [mol eq/m^3] concentration</td>
</tr>
<tr>
<td>UTRAC02</td>
<td>UUr</td>
<td>mol/m^3.m/s</td>
<td>Zonal Mass-Weighted Transp of Alk</td>
</tr>
<tr>
<td>VTRAC02</td>
<td>VUr</td>
<td>mol/m^3.m/s</td>
<td>Merid Mass-Weighted Transp of Alk</td>
</tr>
<tr>
<td>WTRAC02</td>
<td>WM</td>
<td>mol/m^3.m/s</td>
<td>Vert Mass-Weighted Transp of Alk</td>
</tr>
<tr>
<td>ForcTr02</td>
<td>SMR</td>
<td>mol/m^3/s</td>
<td>Alk forcing tendency</td>
</tr>
<tr>
<td>AB_gTr02</td>
<td>SMR</td>
<td>mol/m^3/s</td>
<td>Alk tendency from Adams-Bashforth</td>
</tr>
<tr>
<td>Tp_gTr02</td>
<td>SMR</td>
<td>mol/m^3/s</td>
<td>Alk total transport tendency (before gchem_forcing_)</td>
</tr>
<tr>
<td>ADVrTr02</td>
<td>WM</td>
<td>mol/m^3.m^3/s</td>
<td>Vertical Advective Flux of Alk</td>
</tr>
<tr>
<td>ADVxTr02</td>
<td>UU</td>
<td>mol/m^3.m^3/s</td>
<td>Zonal Advective Flux of Alk</td>
</tr>
<tr>
<td>ADVyTr02</td>
<td>VV</td>
<td>mol/m^3.m^3/s</td>
<td>Meridional Advective Flux of Alk</td>
</tr>
<tr>
<td>DFrETr02</td>
<td>WM</td>
<td>mol/m^3.m^3/s</td>
<td>Vertical Diffusive Flux of Alk (Explicit part)</td>
</tr>
<tr>
<td>DFxETr02</td>
<td>UU</td>
<td>mol/m^3.m^3/s</td>
<td>Zonal Diffusive Flux of Alk</td>
</tr>
<tr>
<td>DFyETr02</td>
<td>VV</td>
<td>mol/m^3.m^3/s</td>
<td>Meridional Diffusive Flux of Alk</td>
</tr>
<tr>
<td>DFrITr02</td>
<td>WM</td>
<td>mol/m^3.m^3/s</td>
<td>Vertical Diffusive Flux of Alk (Implicit part)</td>
</tr>
<tr>
<td>TRAC03</td>
<td>SMR</td>
<td>mol/m^3</td>
<td>Phosphate (PO4) [mol P/m^3] concentration</td>
</tr>
<tr>
<td>UTRAC03</td>
<td>UUr</td>
<td>mol/m^3.m/s</td>
<td>Zonal Mass-Weighted Transp of PO4</td>
</tr>
<tr>
<td>VTRAC03</td>
<td>VUr</td>
<td>mol/m^3.m/s</td>
<td>Merid Mass-Weighted Transp of PO4</td>
</tr>
<tr>
<td>WTRAC03</td>
<td>WM</td>
<td>mol/m^3.m/s</td>
<td>Vert Mass-Weighted Transp of PO4</td>
</tr>
<tr>
<td>ForcTr03</td>
<td>SMR</td>
<td>mol/m^3/s</td>
<td>PO4 forcing tendency</td>
</tr>
<tr>
<td>AB_gTr03</td>
<td>SMR</td>
<td>mol/m^3/s</td>
<td>PO4 tendency from Adams-Bashforth</td>
</tr>
<tr>
<td>Tp_gTr03</td>
<td>SMR</td>
<td>mol/m^3/s</td>
<td>PO4 total transport tendency (before gchem_forcing_)</td>
</tr>
</tbody>
</table>

(continues on next page)
8.3.3.5 Do’s and Don’ts

8.3.3.6 Reference Material

8.4 Ocean Packages

8.4.1 GMREDI: Gent-McWilliams/Redi SGS Eddy Parameterization

There are two parts to the Redi/GMparameterization of geostrophic eddies. The first, the Redi scheme [Red82], aims to mix tracer properties along isentropes (neutral surfaces) by means of a diffusion operator oriented along the local isentropic surface. The second part, GM [GM90, GWMM95], adiabatically re-arranges tracers through an advective flux where the advecting flow is a function of slope of the isentropic surfaces.

The first GCM implementation of the Redi scheme was by [Cox87] in the GFDL ocean circulation model. The original approach failed to distinguish between isopycnals and surfaces of locally referenced potential density (now called neutral surfaces) which are proper isentropes for the ocean. As will be discussed later, it also appears that the Cox scheme is susceptible to a computational mode. Due to this mode, the Cox scheme requires a background lateral diffusion to be present to conserve the integrity of the model fields.
The GM parameterization was then added to the GFDL code in the form of a non-divergent bolus velocity. The method defines two stream-functions expressed in terms of the isoneutral slopes subject to the boundary condition of zero value on upper and lower boundaries. The horizontal bolus velocities are then the vertical derivative of these functions. Here in lies a problem highlighted by [GGP+98]: the bolus velocities involve multiple derivatives on the potential density field, which can consequently give rise to noise. Griffies et al. point out that the GM bolus fluxes can be identically written as a skew flux which involves fewer differential operators. Further, combining the skew flux formulation and Redi scheme, substantial cancellations take place to the point that the horizontal fluxes are unmodified from the lateral diffusion parameterization.

8.4.1.1 Redi scheme: Isopycnal diffusion

The Redi scheme diffuses tracers along isopycnals and introduces a term in the tendency (rhs) of such a tracer (here $\tau$) of the form:

$$\nabla \cdot \kappa_p \mathbf{K}_{\text{Redi}} \nabla \tau$$

where $\kappa_p$ is the along isopycnal diffusivity and $\mathbf{K}_{\text{Redi}}$ is a rank 2 tensor that projects the gradient of $\tau$ onto the isopycnal surface. The unapproximated projection tensor is:

$$\mathbf{K}_{\text{Redi}} = \frac{1}{1 + |\mathbf{S}|^2} \begin{pmatrix} 1 + S_x^2 & -S_x S_y & S_x \\ -S_x S_y & 1 + S_y^2 & S_y \\ S_x & S_y & |\mathbf{S}|^2 \end{pmatrix}$$

Here, $S_x = -\partial_x \sigma / \partial_z \sigma$ and $S_y = -\partial_y \sigma / \partial_z \sigma$ are the components of the isoneutral slope.

The first point to note is that a typical slope in the ocean interior is small, say of the order $10^{-4}$. A maximum slope might be of order $10^{-2}$ and only exceeds such in unstratified regions where the slope is ill defined. It is therefore justifiable, and customary, to make the small slope approximation, $|\mathbf{S}| << 1$. The Redi projection tensor then becomes:

$$\mathbf{K}_{\text{Redi}} = \begin{pmatrix} 1 & 0 & S_x \\ 0 & 1 & S_y \\ S_x & S_y & |\mathbf{S}|^2 \end{pmatrix}$$

8.4.1.2 GM parameterization

The GM parameterization aims to represent the “advective” or “transport” effect of geostrophic eddies by means of a “bolus” velocity, $u^*$. The divergence of this advective flux is added to the tracer tendency equation (on the rhs):

$$-\nabla \cdot \tau u^*$$

The bolus velocity $u^*$ is defined as the rotational of a streamfunction $\mathbf{F}^*=(F_x^*, F_y^*, 0)$:

$$u^* = \nabla \times \mathbf{F}^* = \begin{pmatrix} -\partial_z F_y^* \\ +\partial_z F_x^* \\ \partial_x F_y^* - \partial_y F_x^* \end{pmatrix},$$

and thus is automatically non-divergent. In the GM parameterization, the streamfunction is specified in terms of the isoneutral slopes $S_x$ and $S_y$:

$$F_x^* = -\kappa_{GM} S_y$$
$$F_y^* = \kappa_{GM} S_x$$

with boundary conditions $F_x^* = F_y^* = 0$ on upper and lower boundaries. In the end, the bolus transport in the GM parameterization is given by:

$$u^* = \begin{pmatrix} u^* \\ v^* \\ w^* \end{pmatrix} = \begin{pmatrix} -\partial_z (\kappa_{GM} S_x) \\ -\partial_z (\kappa_{GM} S_y) \\ \partial_x (\kappa_{GM} S_x) + \partial_y (\kappa_{GM} S_y) \end{pmatrix}$$
This is the form of the GM parameterization as applied by Donabasaglu, 1997, in MOM versions 1 and 2.

Note that in the MITgcm, the variables containing the GM bolus streamfunction are:

\[
\begin{pmatrix}
G_{M_P s i X} \\
G_{M_P s i Y}
\end{pmatrix} = \begin{pmatrix}
\kappa_{GM} S_x \\
\kappa_{GM} S_y
\end{pmatrix} = \begin{pmatrix}
F_y^* \\
-F_x^*
\end{pmatrix}.
\]

### 8.4.1.3 Griffies Skew Flux

[Gri98] notes that the discretisation of bolus velocities involves multiple layers of differencing and interpolation that potentially lead to noisy fields and computational modes. He pointed out that the bolus flux can be re-written in terms of a non-divergent flux and a skew-flux:

\[
\begin{pmatrix}
\sigma_x \\
\sigma_y
\end{pmatrix} = \begin{pmatrix}
\kappa_{GM} S_x \sigma_x \\
\kappa_{GM} S_y \sigma_y
\end{pmatrix} + \begin{pmatrix}
\kappa_{GM} S_x \sigma_x \\
\kappa_{GM} S_y \sigma_y
\end{pmatrix}
\]

The first vector is non-divergent and thus has no effect on the tracer field and can be dropped. The remaining flux can be written:

\[
u^\tau = -\kappa_{GM} K_{GM} \nabla \sigma
\]

where

\[
K_{GM} = \begin{pmatrix}
0 & 0 & -S_x \\
0 & 0 & -S_y \\
S_x & S_y & 0
\end{pmatrix}
\]

is an anti-symmetric tensor.

This formulation of the GM parameterization involves fewer derivatives than the original and also involves only terms that already appear in the Redi mixing scheme. Indeed, a somewhat fortunate cancellation becomes apparent when we use the GM parameterization in conjunction with the Redi isolinear mixing scheme:

\[
\kappa_{\rho} K_{Redi} \nabla \sigma - u^\tau = (\kappa_{\rho} K_{Redi} + \kappa_{GM} K_{GM}) \nabla \sigma
\]

In the instance that \(\kappa_{GM} = \kappa_{\rho}\) then

\[
\kappa_{\rho} K_{Redi} + \kappa_{GM} K_{GM} = \kappa_{\rho} \begin{pmatrix}
1 & 0 & 0 \\
0 & 1 & 0 \\
2S_x & 2S_y & |S|^2
\end{pmatrix}
\]

which differs from the variable Laplacian diffusion tensor by only two non-zero elements in the \(z\)-row.

### Subroutine

S/R GMREDI_CALC_TENSOR (pkg/gmredi/gmredi_calc_tensor.F)

\(\sigma_x\): SlopeX (argument on entry)

\(\sigma_y\): SlopeY (argument on entry)

\(\sigma_z\): SlopeY (argument)

\(S_x\): SlopeX (argument on exit)

\(S_y\): SlopeY (argument on exit)
8.4.1.4 Variable $\kappa_{GM}$

[VMHS97] suggest making the eddy coefficient, $\kappa_{GM}$, a function of the Eady growth rate, $|f|/\sqrt{R_i}$. The formula involves a non-dimensional constant, $\alpha$, and a length-scale $L$:

$$\kappa_{GM} = \alpha L^2 \frac{|f|}{\sqrt{R_i}}$$

where the Eady growth rate has been depth averaged (indicated by the over-line). A local Richardson number is defined $R_i = N^2/(\partial u/\partial z)^2$ which, when combined with thermal wind gives:

$$\frac{1}{R_i} = \frac{(\partial u/\partial z)^2}{N^2} = \frac{\left(\frac{g}{\rho_o} |\nabla \sigma| \right)^2}{N^2} = \frac{M^4}{|f|^2 N^2}$$

where $M^2$ is defined $M^2 = \frac{g}{\rho_o} |\nabla \sigma|$. Substituting into the formula for $\kappa_{GM}$ gives:

$$\kappa_{GM} = \alpha L^2 \frac{M^2}{N} = \alpha L^2 \frac{M^2 N^{-2}}{N} = \alpha L^2 \frac{|\nabla \sigma| N^2}{N^2}$$

8.4.1.5 Tapering and stability

Experience with the GFDL model showed that the GM scheme has to be matched to the convective parameterization. This was originally expressed in connection with the introduction of the KPP boundary layer scheme [LMD94] but in fact, as subsequent experience with the MIT model has found, is necessary for any convective parameterization.

Subroutine

S/R GMREDI_SLOPE_LIMIT (pkg/gmredi/gmredi_slope_limit.F)

$\sigma_x, s_x$: SlopeX (argument)
$\sigma_y, s_y$: SlopeY (argument)
$\sigma_z$: dSigmadRReal (argument)
$z^*_{\sigma}$: dRdSigmaLd (argument)

8.4.1.6 Slope clipping

Deep convection sites and the mixed layer are indicated by homogenized, unstable or nearly unstable stratification. The slopes in such regions can be either infinite, very large with a sign reversal or simply very large. From a numerical point of view, large slopes lead to large variations in the tensor elements (implying large bolus flow) and can be numerically unstable. This was first recognized by [Cox87] who implemented “slope clipping” in the isopycnal mixing tensor. Here, the slope magnitude is simply restricted by an upper limit:

$$|\nabla \sigma| = \sqrt{\sigma_x^2 + \sigma_y^2}$$

$$S_{lim} = \frac{|\nabla \sigma|}{S_{max}}$$

where $S_{max}$ is a parameter

$$\sigma^*_{z} = \min(\sigma_z, S_{lim})$$

$$[s_x, s_y] = -\frac{[\sigma_x, \sigma_y]}{\sigma^*_{z}}$$
Figure 8.7: Taper functions used in GKW91 and DM95.

Figure 8.8: Effective slope as a function of ‘true’ slope using Cox slope clipping, GKW91 limiting and DM95 limiting.
Notice that this algorithm assumes stable stratification through the “min” function. In the case where the fluid is well stratified ($\sigma_z < S_{lim}$) then the slopes evaluate to:

$$[s_x, s_y] = \left[ \frac{\sigma_x, \sigma_y}{\sigma_z} \right]$$

while in the limited regions ($\sigma_z > S_{lim}$) the slopes become:

$$[s_x, s_y] = \left[ \frac{\sigma_x, \sigma_y}{|\nabla \sigma|/S_{max}} \right]$$

so that the slope magnitude is limited $\sqrt{s_x^2 + s_y^2} = S_{max}$.

The slope clipping scheme is activated in the model by setting `GM_taper_scheme = 'clipping'` in `data.gmredi`.

Even using slope clipping, it is normally the case that the vertical diffusion term (with coefficient $\kappa_{\rho}K_{33} = \kappa_{\rho}S_{max}^2$) is large and must be time-stepped using an implicit procedure (see section on discretisation and code later). Fig. [fig-mixedlayer] shows the mixed layer depth resulting from a) using the GM scheme with clipping and b) no GM scheme (horizontal diffusion). The classic result of dramatically reduced mixed layers is evident. Indeed, the deep convection sites to just one or two points each and are much shallower than we might prefer. This, it turns out, is due to the over zealous re-stratification due to the bolus transport parameterization. Limiting the slopes also breaks the adiabatic nature of the GM/Redi parameterization, re-introducing diabatic fluxes in regions where the limiting is in effect.

### 8.4.1.7 Tapering: Gerdes, Koberle and Willebrand, Clim. Dyn. 1991

The tapering scheme used in [GKW91] addressed two issues with the clipping method: the introduction of large vertical fluxes in addition to convective adjustment fluxes is avoided by tapering the GM/Redi slopes back to zero in low-stratification regions; the adjustment of slopes is replaced by a tapering of the entire GM/Redi tensor. This means the direction of fluxes is unaffected as the amplitude is scaled.

The scheme inserts a tapering function, $f_1(S)$, in front of the GM/Redi tensor:

$$f_1(S) = \min \left[ 1, \left( \frac{S_{max}}{|S|} \right)^2 \right]$$

where $S_{max}$ is the maximum slope you want allowed. Where the slopes, $|S| < S_{max}$ then $f_1(S) = 1$ and the tensor is un-tapered but where $|S| \geq S_{max}$ then $f_1(S)$ scales down the tensor so that the effective vertical diffusivity term $\kappa f_1(S)|S|^2 = \kappa S_{max}^2$.

The GKW91 tapering scheme is activated in the model by setting `GM_taper_scheme = 'gkw91'` in `data.gmredi`.

### 8.4.1.8 Tapering: Danabasoglu and McWilliams, J. Clim. 1995

The tapering scheme used by followed a similar procedure but used a different tapering function, $f_1(S)$:

$$f_1(S) = \frac{1}{2} \left( 1 + \tanh \left[ \frac{S_c - |S|}{S_d} \right] \right)$$

where $S_c = 0.004$ is a cut-off slope and $S_d = 0.001$ is a scale over which the slopes are smoothly tapered. Functionally, the operates in the same way as the GKW91 scheme but has a substantially lower cut-off, turning off the GM/Redi SGS parameterization for weaker slopes.

The DM95 tapering scheme is activated in the model by setting `GM_taper_scheme = 'dm95'` in `data.gmredi`. 

Chapter 8. Packages I - Physical Parameterizations
8.4.1.9 Tapering: Large, Danabasoglu and Doney, JPO 1997

The tapering used in [LDDM97] is based on the DM95 tapering scheme, but also tapers the scheme with an additional function of height, \( f_2(z) \), so that the GM/Redi SGS fluxes are reduced near the surface:

\[
f_2(z) = \frac{1}{2} \left( 1 + \sin(\pi \frac{z}{D} - \frac{\pi}{2}) \right)
\]

where \( D = L_\rho |S| \) is a depth-scale and \( L_\rho = c/f \) with \( c = 2 \) m s\(^{-1}\). This tapering with height was introduced to fix some spurious interaction with the mixed-layer KPP parameterization.

The LDD97 tapering scheme is activated in the model by setting \texttt{GM\_taper\_scheme = 'ldd97'} in \texttt{data.gmredi}.

8.4.1.10 Package Reference

```markdown
<table>
<thead>
<tr>
<th>&lt;-Name-&gt;</th>
<th>Levs</th>
<th>&lt;-parsing code-&gt;</th>
<th>&lt;- Units --&gt;</th>
<th>&lt;- Tile (max=80c)</th>
</tr>
</thead>
<tbody>
<tr>
<td>GM_VisbK</td>
<td>1</td>
<td>SM P M1</td>
<td>m^2/s</td>
<td>Mixing coefficient from Visbeck etal parameterization</td>
</tr>
<tr>
<td>GM_Kux</td>
<td>15</td>
<td>UU P 177MR</td>
<td>m^2/s</td>
<td>K_11 element (U.point, X.dir) of GM-Redi tensor</td>
</tr>
<tr>
<td>GM_Kvy</td>
<td>15</td>
<td>VV P 176MR</td>
<td>m^2/s</td>
<td>K_22 element (V.point, Y.dir) of GM-Redi tensor</td>
</tr>
<tr>
<td>GM_Kuz</td>
<td>15</td>
<td>UU 179MR</td>
<td>m^2/s</td>
<td>K_13 element (U.point, Z.dir) of GM-Redi tensor</td>
</tr>
<tr>
<td>GM_Kvz</td>
<td>15</td>
<td>VV 178MR</td>
<td>m^2/s</td>
<td>K_23 element (V.point, Z.dir) of GM-Redi tensor</td>
</tr>
<tr>
<td>GM_Kwx</td>
<td>15</td>
<td>UM 181LR</td>
<td>m^2/s</td>
<td>K_31 element (W.point, X.dir) of GM-Redi tensor</td>
</tr>
<tr>
<td>GM_Kwy</td>
<td>15</td>
<td>VM 180LR</td>
<td>m^2/s</td>
<td>K_32 element (W.point, Y.dir) of GM-Redi tensor</td>
</tr>
<tr>
<td>GM_Kwz</td>
<td>15</td>
<td>WM P LR</td>
<td>m^2/s</td>
<td>K_33 element (W.point, Z.dir) of GM-Redi tensor</td>
</tr>
<tr>
<td>GM_PsiX</td>
<td>15</td>
<td>UU 184LR</td>
<td>m^2/s</td>
<td>GM Bolus transport stream-function : X component</td>
</tr>
<tr>
<td>GM_PsiY</td>
<td>15</td>
<td>VV 183LR</td>
<td>m^2/s</td>
<td>GM Bolus transport stream-function : Y component</td>
</tr>
<tr>
<td>GM_KuzTz</td>
<td>15</td>
<td>UU 186MR</td>
<td>degC.m^3/s</td>
<td>Redi Off-diagonal Temperature flux: X component</td>
</tr>
<tr>
<td>GM_KvzTz</td>
<td>15</td>
<td>VV 185MR</td>
<td>degC.m^3/s</td>
<td>Redi Off-diagonal Temperature flux: Y component</td>
</tr>
</tbody>
</table>
```

8.4.1.11 Experiments and tutorials that use gmredi

- Global Ocean tutorial, in tutorial_global_oce_latlon verification directory, described in section [sec:eg-global]
- Front Relax experiment, in front_relax verification directory.
- Ideal 2D Ocean experiment, in ideal_2D_oce verification directory.
8.4.2 KPP: Nonlocal K-Profile Parameterization for Vertical Mixing

Authors: Dimitris Menemenlis and Patrick Heimbach

8.4.2.1 Introduction

The nonlocal K-Profile Parameterization (KPP) scheme of Large et al. (1994) \cite{LMD94} unifies the treatment of a variety of unresolved processes involved in vertical mixing. To consider it as one mixing scheme is, in the view of the authors, somewhat misleading since it consists of several entities to deal with distinct mixing processes in the ocean’s surface boundary layer, and the interior:

1. mixing in the interior is governed by shear instability (modeled as function of the local gradient Richardson number), internal wave activity (assumed constant), and double-diffusion (not implemented here).
2. a boundary layer depth \( h \) or \( h_{bl} \) is determined at each grid point, based on a critical value of turbulent processes parameterized by a bulk Richardson number;
3. mixing is strongly enhanced in the boundary layer under the stabilizing or destabilizing influence of surface forcing (buoyancy and momentum) enabling boundary layer properties to penetrate well into the thermocline; mixing is represented through a polynomial profile whose coefficients are determined subject to several contraints;
4. the boundary-layer profile is made to agree with similarity theory of turbulence and is matched, in the asymptotic sense (function and derivative agree at the boundary), to the interior thus fixing the polynomial coefficients; matching allows for some fraction of the boundary layer mixing to affect the interior, and vice versa;
5. a “non-local” term \( \hat{\gamma} \) or \( \hat{h} \) which is independent of the vertical property gradient further enhances mixing where the water column is unstable.

The scheme has been extensively compared to observations (see e.g. Large et al. 1997 \cite{LDDM97}) and is now common in many ocean models.

The current code originates in the NCAR NCOM 1-D code and was kindly provided by Bill Large and Jan Morzel. It has been adapted first to the MITgcm vector code and subsequently to the current parallel code. Adjustment were mainly in conjunction with WRAPPER requirements (domain decomposition and threading capability), to enable automatic differentiation of tangent linear and adjoint code via TAMC.

The following sections will describe the KPP package configuration and compiling (Section 8.4.2.2), the settings and choices of runtime parameters (Section 8.4.2.3), more detailed description of equations to which these parameters relate (Section 8.4.2.4), and key subroutines where they are used (Section 8.4.2.5), and diagnostics output of KPP-derived diffusivities, viscosities and boundary-layer/mixed-layer depths (Section 8.4.2.6).

8.4.2.2 KPP configuration and compiling

As with all MITgcm packages, KPP can be turned on or off at compile time

- using the packages.conf file by adding kpp to it,
- or using genmake2 adding -enable=kpp or -disable=kpp switches
- Required packages and CPP options: No additional packages are required, but the MITgcm kernel flag enabling the penetration of shortwave radiation below the surface layer needs to be set in CPP_OPTIONS.h as follows:

  //define SHORTWAVE_HEATING

(see Section 8.1.1)

Parts of the KPP code can be enabled or disabled at compile time via CPP preprocessor flags. These options are set in KPP_OPTIONS.h. Table Table 8.5 summarizes them.
Table 8.5: CPP flags for KPP

<table>
<thead>
<tr>
<th>CPP option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>_KPP_RL</td>
<td></td>
</tr>
<tr>
<td>FRUGAL_KPP</td>
<td></td>
</tr>
<tr>
<td>KPP_SMOOTH_SHSQ</td>
<td></td>
</tr>
<tr>
<td>KPP_SMOOTH_DVSQ</td>
<td></td>
</tr>
<tr>
<td>KPP_SMOOTH_DENS</td>
<td></td>
</tr>
<tr>
<td>KPP_SMOOTH_VISC</td>
<td></td>
</tr>
<tr>
<td>KPP_SMOOTH_DIFF</td>
<td></td>
</tr>
<tr>
<td>KPP_ESTIMATE_UREF</td>
<td></td>
</tr>
<tr>
<td>INCLUDE_DIAGNOSTICS_INTERFACE_CODE</td>
<td></td>
</tr>
<tr>
<td>KPP_GHAT</td>
<td></td>
</tr>
<tr>
<td>EXCLUDE_KPP_SHEAR_MIX</td>
<td></td>
</tr>
</tbody>
</table>

8.4.2.3 Run-time parameters

Run-time parameters are set in files data.pkg and data.kpp which are read in kpp_readparms.F. Run-time parameters may be broken into 3 categories: (i) switching on/off the package at runtime, (ii) required MITgcm flags, (iii) package flags and parameters.

Enabling the package

The KPP package is switched on at runtime by setting useKPP = .TRUE. in data.pkg.

Required MITgcm flags

The following flags/parameters of the MITgcm dynamical kernel need to be set in conjunction with KPP:

| implicitViscosity = .TRUE. | enable implicit vertical viscosity |
| implicitDiffusion = .TRUE. | enable implicit vertical diffusion |

Package flags and parameters

Table 8.6 summarizes the runtime flags that are set in data.pkg, and their default values.

Table 8.6: Runtime flags for KPP

<table>
<thead>
<tr>
<th>Flag/parameter</th>
<th>default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>kpp_freq</td>
<td>deltaTClock</td>
<td>Recomputation frequency for KPP fields</td>
</tr>
<tr>
<td>kpp_dumpFreq</td>
<td>dumpFreq</td>
<td>Dump frequency of KPP field snapshots</td>
</tr>
<tr>
<td>kpp_taveFreq</td>
<td>taveFreq</td>
<td>Averaging and dump frequency of KPP fields</td>
</tr>
<tr>
<td>KPPmixingMaps</td>
<td>.FALSE.</td>
<td>include KPP diagnostic maps in STDOUT</td>
</tr>
</tbody>
</table>

continues on next page
<table>
<thead>
<tr>
<th>Flag/parameter</th>
<th>default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>KPPwriteState</td>
<td>.FALSE.</td>
<td>write KPP state to file</td>
</tr>
<tr>
<td>KPP\textunderscore ghat\textunderscore UseTotalDiffus</td>
<td>.FALSE.</td>
<td>if .T. compute non-local term using total vertical diffusivity if .F. use KPP vertical diffusivity</td>
</tr>
</tbody>
</table>

### General KPP parameters

<table>
<thead>
<tr>
<th>Flag/parameter</th>
<th>default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>minKPPhbl</td>
<td>delRc(1)</td>
<td>Minimum boundary layer depth</td>
</tr>
<tr>
<td>epsilon</td>
<td>0.1</td>
<td>nondimensional extent of the surface layer</td>
</tr>
<tr>
<td>vonk</td>
<td>0.4</td>
<td>von Karman constant</td>
</tr>
<tr>
<td>dB\textunderscore dz</td>
<td>5.2E-5 s\textsuperscript -2</td>
<td>maximum dB/dz in mixed layer hMix</td>
</tr>
<tr>
<td>concs</td>
<td>98.96</td>
<td></td>
</tr>
<tr>
<td>concv</td>
<td>1.8</td>
<td></td>
</tr>
</tbody>
</table>

### Boundary layer parameters (S/R bldepth)

<table>
<thead>
<tr>
<th>Flag/parameter</th>
<th>default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ricr</td>
<td>0.3</td>
<td>critical bulk Richardson number</td>
</tr>
<tr>
<td>cekman</td>
<td>0.7</td>
<td>coefficient for Ekman depth</td>
</tr>
<tr>
<td>cmonob</td>
<td>1.0</td>
<td>coefficient for Monin-Obukhov depth</td>
</tr>
<tr>
<td>concv</td>
<td>1.8</td>
<td>ratio of interior to entrainment depth buoyancy frequency</td>
</tr>
<tr>
<td>hbf</td>
<td>1.0</td>
<td>fraction of depth to which absorbed solar radiation contributes to surface buoyancy forcing</td>
</tr>
<tr>
<td>Vtc</td>
<td></td>
<td>non-dim. coeff. for velocity scale of turbulent velocity shear (= function of concv,concs,epsilon,vonk,Ricr)</td>
</tr>
</tbody>
</table>

### Boundary layer mixing parameters (S/R blmix)

<table>
<thead>
<tr>
<th>Flag/parameter</th>
<th>default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cstar</td>
<td>10.</td>
<td>proportionality coefficient for non-local transport</td>
</tr>
<tr>
<td>cg</td>
<td></td>
<td>non-dimensional coefficient for counter-gradient term (= function of cstar,vonk,concs,epsilon)</td>
</tr>
</tbody>
</table>

### Interior mixing parameters (S/R Ri\textunderscore iwmix)

<table>
<thead>
<tr>
<th>Flag/parameter</th>
<th>default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Riinfty</td>
<td>0.7</td>
<td>gradient Richardson number limit for shear instability</td>
</tr>
<tr>
<td>BVDQcon</td>
<td>-0.2E-4 s\textsuperscript -2</td>
<td>Brunt-Väisälä squared</td>
</tr>
<tr>
<td>difm0</td>
<td>0.005 m\textsuperscript 2 s\textsuperscript -1</td>
<td>viscosity max. due to shear instability</td>
</tr>
<tr>
<td>difs0</td>
<td>0.005 m\textsuperscript 2 s</td>
<td>tracer diffusivity max. due to shear instability</td>
</tr>
<tr>
<td>dift0</td>
<td>0.005 m\textsuperscript 2 s</td>
<td>heat diffusivity max. due to shear instability</td>
</tr>
<tr>
<td>difmcon</td>
<td>0.1</td>
<td>viscosity due to convective instability</td>
</tr>
<tr>
<td>difscon</td>
<td>0.1</td>
<td>tracer diffusivity due to convective instability</td>
</tr>
<tr>
<td>diftcon</td>
<td>0.1</td>
<td>heat diffusivity due to convective instability</td>
</tr>
</tbody>
</table>

continues on next page
8.4.2.4 Equations and key routines

We restrict ourselves to writing out only the essential equations that relate to main processes and parameters mentioned above. We closely follow the notation of Large et al. (1994) \[LMD94\].

**KPP_CALC:**

Top-level routine.

**KPP_MIX:**

Intermediate-level routine

**BLMIX: Mixing in the boundary layer**

The vertical fluxes $\overline{\tau}$ of momentum and tracer properties $X$ is composed of a gradient-flux term (proportional to the vertical property divergence $\partial_z X$), and a “nonlocal” term $\gamma_x$ that enhances the gradient-flux mixing coefficient $K_x$

$$\overline{\tau}(d) = -K_x \left( \frac{\partial X}{\partial z} - \gamma_x \right)$$

- **Boundary layer mixing profile** It is expressed as the product of the boundary layer depth $h$, a depth-dependent turbulent velocity scale $w_x(\sigma)$ and a non-dimensional shape function $G(\sigma)$

$$K_x(\sigma) = h w_x(\sigma) G(\sigma)$$

with dimensionless vertical coordinate $\sigma = d/h$. For details of $w_x(\sigma)$ and $G(\sigma)$ we refer to Large et al. (1994) \[LMD94\].

- **Nonlocal mixing term** The nonlocal transport term $\gamma$ is nonzero only for tracers in unstable (convective) forcing conditions. Thus, depending on the stability parameter $\zeta = d/L$ (with depth $d$, Monin-Obukhov length scale $L$) it has the following form:

$$\begin{align*}
\gamma_x &= 0 \quad \zeta \geq 0 \\
\gamma_m &= 0 \\
\gamma_s &= C_s \frac{\overline{\tau}}{w_x(\sigma) h} \quad \zeta < 0 \\
\gamma_\theta &= C_\theta \frac{\overline{\theta}}{w_x(\sigma) h}
\end{align*}$$

In practice, the routine performs the following tasks:

1. compute velocity scales at hbl
2. find the interior viscosities and derivatives at hbl
3. compute turbulent velocity scales on the interfaces
4. compute the dimensionless shape functions at the interfaces
5. compute boundary layer diffusivities at the interfaces
6. compute nonlocal transport term
7. find diffusivities at kbl-1 grid level

RI_IWMIX: Mixing in the interior

Compute interior viscosity and diffusivity coefficients due to
- shear instability (dependent on a local gradient Richardson number),
- to background internal wave activity, and
- to static instability (local Richardson number < 0).

TO BE CONTINUED.

BLDEPTH: Boundary layer depth calculation:

The oceanic planetary boundary layer depth, hbl, is determined as the shallowest depth where the bulk Richardson number is equal to the critical value, Ricr.

Bulk Richardson numbers are evaluated by computing velocity and buoyancy differences between values at zgrid(kl) < 0 and surface reference values. In this configuration, the reference values are equal to the values in the surface layer. When using a very fine vertical grid, these values should be computed as the vertical average of velocity and buoyancy from the surface down to epsilon*zgrid(kl).

When the bulk Richardson number at k exceeds Ricr, hbl is linearly interpolated between grid levels zgrid(k) and zgrid(k-1).

The water column and the surface forcing are diagnosed for stable/ustable forcing conditions, and where hbl is relative to grid points (caseA), so that conditional branches can be avoided in later subroutines.

TO BE CONTINUED.

KPP_CALC_DIFF_T/_S, KPP_CALC_VISC:

Add contribution to net diffusivity/viscosity from KPP diffusivity/viscosity.

TO BE CONTINUED.
KPP_TRANSPORT_T/_S/_PTR:

Add non local KPP transport term (ghat) to diffusive temperature/salinity/passive tracer flux. The nonlocal transport term is nonzero only for scalars in unstable (convective) forcing conditions.

TO BE CONTINUED.

Implicit time integration

TO BE CONTINUED.

Penetration of shortwave radiation

TO BE CONTINUED.

8.4.2.5 Flow chart

```
C !CALLING SEQUENCE:
C ...
C kpp_calc (TOP LEVEL ROUTINE)
C |-- statekpp: o compute all EOS/density-related arrays
C | o uses S/R FIND_ALPHA, FIND_BETA, FIND_RHO
C |-- kppmix
C | |--- ri_iwmix (compute interior mixing coefficients due to constant
C | | internal wave activity, static instability,
C | | and local shear instability).
C | |--- bldepth (diagnose boundary layer depth)
C | |--- blmix (compute boundary layer diffusivities)
C | |--- enhance (enhance diffusivity at interface kbl - 1)
C | o
C |-- swfrac
C o
```

8.4.2.6 KPP diagnostics

Diagnostics output is available via the diagnostics package (see Section [sec:pkg:diagnostics]). Available output fields are summarized here:

<table>
<thead>
<tr>
<th>Name</th>
<th>Levs</th>
<th>grid</th>
<th>Units</th>
<th>Tile (max=80c)</th>
</tr>
</thead>
<tbody>
<tr>
<td>KPPviscA</td>
<td>23</td>
<td>SM</td>
<td>m²/s</td>
<td></td>
</tr>
<tr>
<td>KPPdiffS</td>
<td>23</td>
<td>SM</td>
<td>m²/s</td>
<td></td>
</tr>
</tbody>
</table>

(continues on next page)
8.4.2.7 Reference experiments

lab_sea:

natl_box:

8.4.2.8 References

8.4.2.9 Experiments and tutorials that use kpp

- Labrador Sea experiment, in lab_sea verification directory

8.4.3 GGL90: a TKE vertical mixing scheme

(in directory: pkg/ggl90/)

8.4.3.1 Key subroutines, parameters and files

see [GGL90]

8.4.3.2 Experiments and tutorials that use GGL90

- Vertical mixing verification experiment (vermix/input.ggl90)

8.4.4 OPPS: Ocean Penetrative Plume Scheme

(in directory: pkg/opps/)

8.4.4.1 Key subroutines, parameters and files

See [PR97]
8.4.4.2 Experiments and tutorials that use OPPS

- Vertical mixing verification experiment (vermix/input.opps)

8.4.5 KL10: Vertical Mixing Due to Breaking Internal Waves

(in directory: pkg/kl10/)

Authors: Jody M. Klymak

8.4.5.1 Introduction

The [KL10] parameterization for breaking internal waves is meant to represent mixing in the ocean “interior” due to convective instability. Many mixing schemes in the presence of unstable stratification simply turn on an arbitrarily large diffusivity and viscosity in the overturning region. This assumes the fluid completely mixes, which is probably not a terrible assumption, but it also makes estimating the turbulence dissipation rate in the overturning region meaningless.

The KL10 scheme overcomes this limitation by estimating the viscosity and diffusivity from a combination of the Ozmidov relation and the Osborn relation, assuming a turbulent Prandtl number of one. The Ozmidov relation says that outer scale of turbulence in an overturn will scale with the strength of the turbulence \( \epsilon \), and the stratification \( N \), as

\[
L_O^2 \approx \epsilon N^{-3}.
\]  

(8.1)

The Osborn relation relates the strength of the dissipation to the vertical diffusivity as

\[
K_v = \Gamma \epsilon N^{-2},
\]

where \( \Gamma \approx 0.2 \) is the mixing ratio of buoyancy flux to thermal dissipation due to the turbulence. Combining the two gives us

\[
K_v \approx \Gamma L_O^2 N.
\]

The ocean turbulence community often approximates the Ozmidov scale by the root-mean-square of the Thorpe displacement, \( \delta_z \), in an overturn [Tho77]. The Thorpe displacement is the distance one would have to move a water parcel for the water column to be stable, and is readily measured in a measured profile by sorting the profile and tracking how far each parcel moves during the sorting procedure. This method gives an imperfect estimate of the turbulence, but it has been found to agree on average over a large range of overturns [Mou96, SG94, WG94].

The algorithm coded here is a slight simplification of the usual Thorpe method for estimating turbulence in overturning regions. Usually, overturns are identified and \( N \) is averaged over the overturn. Here, instead we estimate

\[
K_v(z) \approx \Gamma \delta_z^2 N_s(z).
\]

where \( N_s(z) \) is the local sorted stratification. This saves complexity in the code and adds a slight inaccuracy, but we don’t believe is biased.

We assume a turbulent Prandtl number of 1, so \( A_v = K_v \).

We also calculate and output a turbulent dissipation from this scheme. We do not simply evaluate the overturns for \( \epsilon \) using (8.1). Instead we compute the vertical shear terms that the viscosity is acting on:

\[
\epsilon_v = A_v \left( \left( \frac{\partial u}{\partial z} \right)^2 + \left( \frac{\partial v}{\partial z} \right)^2 \right).
\]

There are straightforward caveats to this approach, covered in [KL10].
• If your resolution is too low to resolve the breaking internal waves, you won’t have any turbulence.

• If the model resolution is too high, the estimates of $\epsilon_v$ will start to be exaggerated, particularly if the run in non-hydrostatic. That is because there will be significant shear at small scales that represents the turbulence being parameterized in the scheme. At very high resolutions direct numerical simulation or more sophisticated large-eddy schemes should be used.

• We find that grid cells of approximately 10 to 1 aspect ratio are a good rule of thumb for achieving good results are usual oceanic scales. For a site like the Hawaiian Ridge, and Luzon Strait, this means 10-m vertical resolution and approximately 100-m horizontal. The 10-m resolution can be relaxed if the stratification drops, and we often WKB-stretch the grid spacing with depth.

• The dissipation estimate is useful for pinpointing the location of turbulence, but again, is grid size dependent to some extent, and should be treated with a grain of salt. It will also not include any numerical dissipation such as you may find with higher order advection schemes.

8.4.5.2 KL10 configuration and compiling

As with all MITgcm packages, KL10 can be turned on or off at compile time

• using the packages.conf file by adding kl10 to it,

• or using genmake2 adding -enable=kl10 or -disable=kl10 switches

• Required packages and CPP options: No additional packages are required.

KL10 has no compile-time options (KL10_OPTIONS.h is empty).

8.4.5.3 Run-time parameters

Run-time parameters are set in files data.pkg and data.kl10 which are read in kl10_readparms.F. Run-time parameters may be broken into 3 categories: (i) switching on/off the package at runtime, (ii) required MITgcm flags, (iii) package flags and parameters.

Enabling the package

The KL10 package is switched on at runtime by setting useKL10 = .TRUE. in data.pkg.

Required MITgcm flags

The following flags/parameters of the MITgcm dynamical kernel need to be set in conjunction with KL10:

| implicitViscosity = .TRUE. | enable implicit vertical viscosity |
| implicitDiffusion = .TRUE. | enable implicit vertical diffusion |
Package flags and parameters

Table 8.7 summarizes the runtime flags that are set in data.kl10, and their default values.

<table>
<thead>
<tr>
<th>Flag/parameter</th>
<th>default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>KLviscMax</td>
<td>300 m² s⁻¹</td>
<td>Maximum viscosity the scheme will ever give (useful for stability)</td>
</tr>
<tr>
<td>KLtaveFreq</td>
<td>taveFreq</td>
<td>Averaging and dump frequency of KL10 fields</td>
</tr>
<tr>
<td>KLwriteState</td>
<td>.FALSE.</td>
<td>write KL10 state to file</td>
</tr>
</tbody>
</table>

8.4.5.4 Equations and key routines

**KL10_CALC:**

Top-level routine. Calculates viscosity and diffusivity on the grid cell centers. Note that the runtime parameters viscAz and diffKzT act as minimum viscosity and diffusivities. So if there are no overturns (or they are weak) then these will be returned.

**KL10_CALC_VISC:**

Calculates viscosity on the W and S grid faces for U and V respectively.

**KL10_CALC_DIFF:**

Calculates the added diffusion from KL10.

8.4.5.5 KL10 diagnostics

Diagnostics output is available via the diagnostics package (see Section [sec:pkg:diagnostics]). Available output fields are summarized here:

```
<-Name-|Levs|grid|-- Units -->|-- Tile (max=80c)
-----------------------------------------------
KLviscAr|Nr |SM |m²/s |KL10 vertical eddy viscosity coefficient
KLdiffKr|Nr |SM |m²/s |Vertical diffusion coefficient for salt, temperature, & tracers
KLeps   |Nr |SM |m³/s³|Turbulence dissipation estimate.
```
8.4.5.6 References


8.4.5.7 Experiments and tutorials that use KL10

- Modified Internal Wave experiment, in internal_wave verification directory

8.4.6 BULK_FORCE: Bulk Formula Package

author: Stephanie Dutkiewicz

Instead of forcing the model with heat and fresh water flux data, this package calculates these fluxes using the changing sea surface temperature. We need to read in some atmospheric data: *air temperature, air humidity, down shortwave radiation, down longwave radiation, precipitation, wind speed*. The current setup also reads in *wind stress*, but this can be changed so that the stresses are calculated from the wind speed.

The current setup requires that there is the thermodynamic-seaice package (*pkg/thsice*, also referred below as seaice) is also used. It would be useful though to have it also setup to run with some very simple parametrization of the sea ice. The heat and fresh water fluxes are calculated in *bulkf_forcing.F* called from *forward_step.F*. These fluxes are used over open water, fluxes over seaice are recalculated in the sea-ice package. Before the call to *bulkf_forcing.F* we call *bulkf_fields_load.F* to find the current atmospheric conditions. The only other changes to the model code come from the initializing and writing diagnostics of these fluxes.

8.4.6.1 subroutine BULKF_FIELDS_LOAD

Here we find the atmospheric data needed for the bulk formula calculations. These are read in at periodic intervals and values are interpolated to the current time. The data file names come from *data.blk*. The values that can be read in are: air temperature, air humidity, precipitation, down solar radiation, down long wave radiation, zonal and meridional wind speeds, total wind speed, net heat flux, net freshwater forcing, cloud cover, snow fall, zonal and meridional wind stresses, and SST and SSS used for relaxation terms. Not all these files are necessary or used. For instance cloud cover and snow fall are not used in the current bulk formula calculation. If total wind speed is not supplied, wind speed is calculate from the zonal and meridional components. If wind stresses are not read in, then the stresses are calculated from the wind speed. Net heat flux and net freshwater can be read in and used over open ocean instead of the bulk formula calculations (but over seaice the bulk formula is always used). This is “hardwired” into *bulkf_forcing* and the “ch” in the variable names suggests that this is “cheating”. SST and SSS need to be read in if there is any relaxation used.

8.4.6.2 subroutine BULKF_FORCING

In *bulkf_forcing.F*, we calculate heat and fresh water fluxes (and wind stress, if necessary) for each grid cell. First we determine if the grid cell is open water or seaice and this information is carried by *iceornot*. There is a provision here for a different designation if there is snow cover (but currently this does not make any difference). We then call *bulkf_formula_lanl.F* which provides values for: up long wave radiation, latent and sensible heat fluxes, the derivative of these three with respect to surface temperature, wind stress, evaporation. Net long wave radiation is calculated from the combination of the down long wave read in and the up long wave calculated.

We then find the albedo of the surface - with a call to *sfc_albedo* if there is sea-ice (see the seaice package for information on the subroutine). If the grid cell is open ocean the albedo is set as 0.1. Note that this is a parameter that can be used to tune the results. The net short wave radiation is then the down shortwave radiation minus the amount reflected.
If the wind stress needed to be calculated in `bulkf_formula_lanl.F`, it was calculated to grid cell center points, so in `bulkf_forcing.F` we regrid to \( u \) and \( v \) points. We let the model know if it has read in stresses or calculated stresses by the switch `readwindstress` which is can be set in data.blk, and defaults to .TRUE..

We then calculate \( \text{Qnet} \) and \( \text{EmPmR} \) that will be used as the fluxes over the open ocean. There is a provision for using runoff. If we are “cheating” and using observed fluxes over the open ocean, then there is a provision here to use read in \( \text{Qnet} \) and \( \text{EmPmR} \).

The final call is to calculate averages of the terms found in this subroutine.

### 8.4.6.3 subroutine BULKF_FORMULA_LANL

This is the main program of the package where the heat fluxes and freshwater fluxes over ice and open water are calculated. Note that this subroutine is also called from the seacice package during the iterations to find the ice surface temperature.

Latent heat \((L)\) used in this subroutine depends on the state of the surface: vaporization for open water, fusion and vaporization for ice surfaces. Air temperature is converted from Celsius to Kelvin. If there is no wind speed \((u_s)\) given, then the wind speed is calculated from the zonal and meridional components.

We calculate the virtual temperature:

\[
T_o = T_{air}(1 + \gamma q_{air})
\]

where \( T_{air} \) is the air temperature at \( h_T \), \( q_{air} \) is humidity at \( h_q \) and \( \gamma \) is a constant.

The saturated vapor pressure is calculate (QQ ref):

\[
q_{sat} = \frac{a}{p_o} e^{L(b - \frac{T_{surf}}{T_{surf}})}
\]

where \( a, b, c \) are constants, \( T_{surf} \) is surface temperature and \( p_o \) is the surface pressure.

The two values crucial for the bulk formula calculations are the difference between air at sea surface and sea surface temperature:

\[
\Delta T = T_{air} - T_{surf} + \alpha h_T
\]

where \( \alpha \) is adiabatic lapse rate and \( h_T \) is the height where the air temperature was taken; and the difference between the air humidity and the saturated humidity

\[
\Delta q = q_{air} - q_{sat}.
\]

We then calculate the turbulent exchange coefficients following Bryan et al (1996) and the numerical scheme of Hunke and Lipscombe (1998). We estimate initial values for the exchange coefficients, \( c_u, c_T \) and \( c_q \) as

\[
\kappa \frac{ln(z_{ref}/z_{rou})}{z_{ref}/z_{rou}}
\]

where \( \kappa \) is the Von Karman constant, \( z_{ref} \) is a reference height and \( z_{rou} \) is a roughness length scale which could be a function of type of surface, but is here set as a constant. Turbulent scales are:

\[
\begin{align*}
  u^* &= c_u u_s \\
  T^* &= c_T \Delta T \\
  q^* &= c_q \Delta q
\end{align*}
\]

We find the “integrated flux profile” for momentum and stability if there are stable QQ conditions \((\Upsilon > 0)\):

\[
\psi_m = \psi_s = -5 \Upsilon
\]
and for unstable QQ conditions ($\Upsilon < 0$):

$$\psi_m = 2\ln(0.5(1 + \chi)) + ln(0.5(1 + \chi^2)) - 2\tan^{-1} \chi + \pi/2$$

$$\psi_s = 2\ln(0.5(1 + \chi^2))$$

where

$$\Upsilon = \frac{\kappa g z_{ref}}{u^*} \left( \frac{T^*}{T_o} + \frac{q^*}{1/\gamma + q_a} \right)$$

and $\chi = (1 - 16\Upsilon)^{1/2}$.

The coefficients are updated through 5 iterations as:

$$c_u = \frac{\hat{c}_u}{1 + \hat{c}_u(\lambda - \psi_{uu})/\kappa}$$

$$c_T = \frac{\hat{c}_T}{1 + \hat{c}_T(\lambda - \psi_s)/\kappa}$$

$$c_q = \frac{\hat{c}_q}{\hat{c}_T}$$

where $\lambda = \ln(h_T/z_{ref})$.

We can then find the bulk formula heat fluxes:

Sensible heat flux:

$$Q_s = \rho_{air} c_{p_{air}} u_s c_u c_T \Delta T$$

Latent heat flux:

$$Q_l = \rho_{air} L u_s c_u c_q \Delta q$$

Up long wave radiation

$$Q_{lw}^{up} = \epsilon \sigma T_{sr_f}^4$$

where $\epsilon$ is emissivity (which can be different for open ocean, ice and snow), $\sigma$ is Stefan-Boltzman constant.

We calculate the derivatives of the three above functions with respect to surface temperature

$$\frac{dQ_s}{dT} = \rho_{air} c_{p_{air}} u_s c_u c_T$$

$$\frac{dQ_l}{dT} = \rho_{air} L^2 u_s c_u c_q c$$

$$\frac{dQ_{lw}^{up}}{dT} = 4\epsilon \sigma T_{sr_f}^3$$

And total derivative $\frac{dQ_o}{dT} = \frac{dQ_s}{dT} + \frac{dQ_l}{dT} + \frac{dQ_{lw}^{up}}{dT}$.

If we do not read in the wind stress, it is calculated here.

### 8.4.6.4 Initializing subroutines

bulkf_init.F: Set bulkf variables to zero.
bulkf_readparms.F: Reads data.blk
8.4.6.5 Diagnostic subroutines

bulkf_ave.F: Keeps track of means of the bulkf variables
bulkf_diags.F: Finds averages and writes out diagnostics

8.4.6.6 Common Blocks

BULKF.h: BULKF Variables, data file names, and logicals readwindstress and readsurface
BULKF_DIAGS.h: matrices for diagnostics: averages of fields from bulkf_diags.F
BULKF_ICE_CONSTANTS.h: all the parameters needed by the ice model and in the bulkf formula calculations.

8.4.6.7 Input file DATA.ICE

We read in the file names of atmospheric data used in the bulk formula calculations. Here we can also set the logicals: readwindstress if we read in the wind stress rather than calculate it from the wind speed; and readsurface to read in the surface temperature and salinity if these will be used as part of a relaxing term.

8.4.6.8 Important Notes

1. heat fluxes have different signs in the ocean and ice models.
2. StartIceModel must be changed in data.ice: 1 (if starting from no ice), 0 (if using pickup.ic file).

8.4.6.9 References


Hunke, E.C and W.H. Lipscomb, circa 2001: CICE: the Los Alamos Sea Ice Model Documentation and Software User’s Manual. LACC-98-16v.2. (note: this documentation is no longer available as CICE has progressed to a very different version 3)

8.4.6.10 Experiments and tutorials that use bulk_force

- Global ocean experiment in global_ocean.cs32x15 verification directory, input from input.thsice directory.

8.4.7 EXF: The external forcing package

Authors: Patrick Heimbach and Dimitris Menemenlis
8.4.7.1 Introduction

The external forcing package, in conjunction with the calendar package (cal), enables the handling of real-time (or “model-time”) forcing fields of differing temporal forcing patterns. It comprises climatological restoring and relaxation. Bulk formulae are implemented to convert atmospheric fields to surface fluxes. An interpolation routine provides on-the-fly interpolation of forcing fields an arbitrary grid onto the model grid.

CPP options enable or disable different aspects of the package (Section 8.4.7.2). Runtime options, flags, filenames and field-related dates/times are set in data.exf (Section 8.4.7.3). A description of key subroutines is given in Section 8.4.7.6. Input fields, units and sign conventions are summarized in Section 8.4.7.5, and available diagnostics output is listed in Section 8.4.7.7.

8.4.7.2 EXF configuration, compiling & running

Compile-time options

As with all MITgcm packages, EXF can be turned on or off at compile time

- using the packages.conf file by adding exf to it,
- or using genmake2 adding -enable=exf or -disable=exf switches
- required packages and CPP options: EXF (only) requires the calendar package cal to be enabled if the julian calendar will be used with the data; no additional CPP options are required.

(see Section 3.5).

Parts of the EXF code can be enabled or disabled at compile time via CPP preprocessor flags. These options are set in either EXF_OPTIONS.h or in ECCO_CPPOPTIONS.h. Table 8.8 summarizes these options.

Table 8.8: EXF CPP options

<table>
<thead>
<tr>
<th>CPP option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXF_VERBOSE</td>
<td>verbose mode (recommended only for testing)</td>
</tr>
<tr>
<td>ALLOW_ATM_TEMP</td>
<td>compute heat/freshwater fluxes from atmos. state input</td>
</tr>
<tr>
<td>ALLOW_ATM_WIND</td>
<td>compute wind stress from wind speed input</td>
</tr>
<tr>
<td>ALLOW_BULKFORMULAE</td>
<td>is used if ALLOW_ATM_TEMP or ALLOW_ATM_WIND is enabled</td>
</tr>
<tr>
<td>EXF_READ_EVAP</td>
<td>read evaporation instead of computing it</td>
</tr>
<tr>
<td>ALLOW_RUNOFF</td>
<td>read time-constant river/glacier run-off field</td>
</tr>
<tr>
<td>ALLOW_DOWNWARD_RADIATION</td>
<td>compute net from downward or downward from net radiation</td>
</tr>
<tr>
<td>USE_EXF_INTERPOLATION</td>
<td>enable on-the-fly bilinear or bicubic interpolation of input fields</td>
</tr>
<tr>
<td></td>
<td>used in conjunction with relaxation to prescribed (climatological) fields</td>
</tr>
<tr>
<td>ALLOW_CLIMSST_RELAXATION</td>
<td>relaxation to 2-D SST climatology</td>
</tr>
<tr>
<td>ALLOW_CLIMSSS_RELAXATION</td>
<td>relaxation to 2-D SSS climatology</td>
</tr>
<tr>
<td></td>
<td>these are set outside of EXF in CPP_OPTIONS.h</td>
</tr>
<tr>
<td>SHORTWAVE_HEATING</td>
<td>enable shortwave radiation</td>
</tr>
<tr>
<td>ATMOSPHERIC_LOADING</td>
<td>enable surface pressure forcing</td>
</tr>
</tbody>
</table>
8.4.7.3 Run-time parameters

Run-time parameters are set in files data.pkg and data.exf which is read in exf_readparms.F. Run-time parameters may be broken into 3 categories: (i) switching on/off the package at runtime, (ii) general flags and parameters, and (iii) attributes for each forcing and climatological field.

Enabling the package

A package is switched on/off at runtime by setting (e.g. for EXF) useEXF = .TRUE. in data.pkg.

General flags and parameters

<table>
<thead>
<tr>
<th>Flag/parameter</th>
<th>default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>useExfCheckRange</td>
<td>.TRUE.</td>
<td>check range of input fields and stop if out of range</td>
</tr>
<tr>
<td>useExfYearlyFields</td>
<td>.FALSE.</td>
<td>append current year postfix of form _YYYY on filename</td>
</tr>
<tr>
<td>twoDigitYear</td>
<td>.FALSE.</td>
<td>instead of appending _YYYY append YY</td>
</tr>
<tr>
<td>repeatPeriod</td>
<td>0.0</td>
<td>&gt; 0: cycle through all input fields at the same period (in seconds)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>= 0: use period assigned to each field</td>
</tr>
<tr>
<td>exf_offset_atemp</td>
<td>0.0</td>
<td>set to 273.16 to convert from deg. Kelvin (assumed input) to Celsius</td>
</tr>
<tr>
<td>windstressmax</td>
<td>2.0</td>
<td>max. allowed wind stress N m$^{-2}$</td>
</tr>
<tr>
<td>exf_albedo</td>
<td>0.1</td>
<td>surface albedo used to compute downward vs. net radiative fluxes</td>
</tr>
<tr>
<td>climtempfreeze</td>
<td>-1.9</td>
<td>???</td>
</tr>
<tr>
<td>ocean_emissivity</td>
<td></td>
<td>longwave ocean-surface emissivity</td>
</tr>
<tr>
<td>ice_emissivity</td>
<td></td>
<td>longwave seaice emissivity</td>
</tr>
<tr>
<td>snow_emissivity</td>
<td></td>
<td>longwave snow emissivity</td>
</tr>
<tr>
<td>exf_iceCd</td>
<td>1.63E-3</td>
<td>drag coefficient over sea-ice</td>
</tr>
<tr>
<td>exf_iceCe</td>
<td>1.63E-3</td>
<td>evaporation transfer coeff. over sea-ice</td>
</tr>
<tr>
<td>exf_iceCh</td>
<td>1.63E-3</td>
<td>sensible heat transfer coeff. over sea-ice</td>
</tr>
<tr>
<td>exf_scal_BulkCdn</td>
<td>1.0</td>
<td>overall scaling of neutral drag coeff.</td>
</tr>
<tr>
<td>useStabilityFct_overIce</td>
<td>.FALSE.</td>
<td>compute turbulent transfer coeff. over sea-ice</td>
</tr>
<tr>
<td>readStressOnAgrid</td>
<td>.FALSE.</td>
<td>read wind-stress located on model-grid, A-grid point</td>
</tr>
<tr>
<td>readStressOnCgrid</td>
<td>.FALSE.</td>
<td>read wind-stress located on model-grid, C-grid point</td>
</tr>
<tr>
<td>useRelativeWind</td>
<td>.FALSE.</td>
<td>subtract [U/V]VEL or [U/V]ICE from U/V]WIND before computing [U/V]STRESS</td>
</tr>
<tr>
<td>zref</td>
<td>10.0</td>
<td>reference height</td>
</tr>
<tr>
<td>hu</td>
<td>10.0</td>
<td>height of mean wind</td>
</tr>
<tr>
<td>ht</td>
<td>2.0</td>
<td>height of mean temperature and rel. humidity</td>
</tr>
<tr>
<td>umin</td>
<td>0.5</td>
<td>minimum absolute wind speed for computing Cd</td>
</tr>
<tr>
<td>atmrho</td>
<td>1.2</td>
<td>mean atmospheric density [kg/m$^3$]</td>
</tr>
<tr>
<td>atmcp</td>
<td>1005.0</td>
<td>mean atmospheric specific heat [J/kg/K]</td>
</tr>
<tr>
<td>cdrag_[n]</td>
<td>???</td>
<td>n = 1,2,3; parameters for drag coeff. function</td>
</tr>
<tr>
<td>cstanton_[n]</td>
<td>???</td>
<td>n = 1,2; parameters for Stanton number function</td>
</tr>
<tr>
<td>cdalton</td>
<td>???</td>
<td>parameter for Dalton number function</td>
</tr>
<tr>
<td>flamb</td>
<td>25000000.0</td>
<td>latent heat of evaporation [J/kg]</td>
</tr>
<tr>
<td>flami</td>
<td>334000.0</td>
<td>latent heat of melting of pure ice [J/kg]</td>
</tr>
<tr>
<td>zolmin</td>
<td>-100.0</td>
<td>minimum stability parameter</td>
</tr>
<tr>
<td>cvapor_fac</td>
<td>640380.0</td>
<td></td>
</tr>
<tr>
<td>cvapor_exp</td>
<td>5107.4</td>
<td></td>
</tr>
<tr>
<td>cvapor_fac_ice</td>
<td>11637800.0</td>
<td></td>
</tr>
</tbody>
</table>

continues on next page
Table 8.9 – continued from previous page

<table>
<thead>
<tr>
<th>Flag/parameter</th>
<th>default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cvapor_fac_ice</td>
<td>5897.8</td>
<td></td>
</tr>
<tr>
<td>humid_fac</td>
<td>0.606</td>
<td>parameter for virtual temperature calculation</td>
</tr>
<tr>
<td>gamma_blk</td>
<td>0.010</td>
<td>adiabatic lapse rate</td>
</tr>
<tr>
<td>saltsat</td>
<td>0.980</td>
<td>reduction of saturation vapor pressure over salt-water</td>
</tr>
<tr>
<td>psim_fac</td>
<td>5.0</td>
<td></td>
</tr>
<tr>
<td>exf_monFreq</td>
<td>monitorFreq</td>
<td>output frequency [s]</td>
</tr>
<tr>
<td>exf_iprec</td>
<td>32</td>
<td>precision of input fields (32-bit or 64-bit)</td>
</tr>
<tr>
<td>exf_yftype</td>
<td>‘RL’</td>
<td>precision of arrays (‘RL’ vs. ‘RS’)</td>
</tr>
</tbody>
</table>

Field attributes

All EXF fields are listed in Section 8.4.7.5. Each field has a number of attributes which can be customized. They are summarized in Table 8.10. To obtain an attribute for a specific field, e.g. uwind prepend the field name to the listed attribute, e.g. for attribute period this yields uwindperiod:

```
field & attribute → parameter
e.g. uwind & period → uwindperiod
```

Table 8.10: EXF runtime attributes Note there is one exception for the default of atempconst = celsius2K = 273.16

<table>
<thead>
<tr>
<th>attribute</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>field file</td>
<td>‘ ’</td>
<td>filename; if left empty no file will be read; const will be used instead</td>
</tr>
<tr>
<td>field const</td>
<td>0.0</td>
<td>constant that will be used if no file is read</td>
</tr>
<tr>
<td>field startdate1</td>
<td>0.0</td>
<td>format: YYYYMMDD; start year (YYYY), month (MM), day (YY) of field to determine record number</td>
</tr>
<tr>
<td>field startdate2</td>
<td>0.0</td>
<td>format: HHMMSS; start hour (HH), minute (MM), second(SS) of field to determine record number</td>
</tr>
<tr>
<td>field period</td>
<td>0.0</td>
<td>interval in seconds between two records; the special value -12 means 12 repeating (calendar) monthly records; the special value -1 means non-repeating (calendar) monthly records (see below)</td>
</tr>
<tr>
<td>exf_inscal_field</td>
<td></td>
<td>optional rescaling of input fields to comply with EXF units</td>
</tr>
<tr>
<td>exf_outscal_field</td>
<td></td>
<td>optional rescaling of EXF fields when mapped onto MITgcm fields</td>
</tr>
</tbody>
</table>

For fieldperiod=-1, the records in the forcing file represent averages over calendar months. If useExfYearlyFields = .TRUE., each yearly file must have 12 records, starting with January. For useExfYearlyFields = .FALSE., a single file starting with the month given by fieldstartdate1 is required.
Example configuration

The following block is taken from the data.exf file of the verification experiment global_with_exf/. It defines attributes for the heat flux variable hflux:

<table>
<thead>
<tr>
<th>hfluxfile</th>
<th>'ncep_qnet.bin',</th>
</tr>
</thead>
<tbody>
<tr>
<td>hfluxstartdate1</td>
<td>19920101,</td>
</tr>
<tr>
<td>hfluxstartdate2</td>
<td>000000,</td>
</tr>
<tr>
<td>hfluxperiod</td>
<td>2592000.0,</td>
</tr>
<tr>
<td>hflux_lon0</td>
<td>2</td>
</tr>
<tr>
<td>hflux_lon_inc</td>
<td>4</td>
</tr>
<tr>
<td>hflux_lat0</td>
<td>-78</td>
</tr>
<tr>
<td>hflux_lat_inc</td>
<td>39*4</td>
</tr>
<tr>
<td>hflux_nlon</td>
<td>90</td>
</tr>
<tr>
<td>hflux_nlat</td>
<td>40</td>
</tr>
</tbody>
</table>

EXF will read a file of name 'ncep_qnet.bin'. Its first record represents January 1st, 1992 at 00:00 UTC. Next record is 2592000 seconds (or 30 days) later. Note that the first record read and used by the EXF package corresponds to the value 'startDate1' set in data.cal. Therefore if you want to start the EXF forcing from later in the 'ncep_qnet.bin' file, it suffices to specify startDate1 in data.cal as a date later than 19920101 (for example, startDate1 = 19940101, for starting January 1st, 1994). For this to work, 'ncep_qnet.bin' must have at least 2 years of data because in this configuration EXF will read 2 years into the file to find the 1994 starting value. Interpolation on-the-fly is used (in the present case trivially on the same grid, but included nevertheless for illustration), and input field grid starting coordinates and increments are supplied as well.

8.4.7.4 EXF bulk formulae

T.B.D. (cross-ref. to parameter list table)

8.4.7.5 EXF input fields and units

The following list is taken from the header file EXF_FIELDS.h. It comprises all EXF input fields.

Output fields which EXF provides to the MITgcm are fields fu, fv, Qnet, Qsw, EmPmR, and pload. They are defined in FFIELDS.h.

<table>
<thead>
<tr>
<th>field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ustress</td>
<td>Zonal surface wind stress in N/m^2</td>
</tr>
<tr>
<td>vstress</td>
<td>Meridional surface wind stress in N/m^2</td>
</tr>
</tbody>
</table>

(continues on next page)
c        | Southwest C-grid V point
|        | Input field

| hs        | sensible heat flux into ocean in W/m^2
| > 0 for increase in theta (ocean warming)

| hl        | latent heat flux into ocean in W/m^2
| > 0 for increase in theta (ocean warming)

| hflux     | Net upward surface heat flux in W/m^2
| (including shortwave)
| hflux = latent + sensible + lwflux + swflux
| > 0 for decrease in theta (ocean cooling)
| Typical range: -250 < hflux < 600
| Southwest C-grid tracer point
| Input field

| sflux     | Net upward freshwater flux in m/s
| sflux = evap - precip - runoff
| > 0 for increase in salt (ocean salinity)
| Typical range: -1e-7 < sflux < 1e-7
| Southwest C-grid tracer point
| Input field

| swflux    | Net upward shortwave radiation in W/m^2
| swflux = - ( swdown - ice and snow absorption - reflected )
| > 0 for decrease in theta (ocean cooling)
| Typical range: -350 < swflux < 0
| Southwest C-grid tracer point
| Input field

| uwind     | Surface (10-m) zonal wind velocity in m/s
| > 0 for increase in uVel, which is west to
east for cartesian and spherical polar grids
| Typical range: -10 < uwind < 10
| Southwest C-grid U point
| Input or input/output field

| vwind     | Surface (10-m) meridional wind velocity in m/s
| > 0 for increase in vVel, which is south to
north for cartesian and spherical polar grids
| Typical range: -10 < vwind < 10
| Southwest C-grid V point
| Input or input/output field

| wspeed    | Surface (10-m) wind speed in m/s
| >= 0 sqrt(u^2+v^2)
| Typical range: 0 < wspeed < 10
| Input or input/output field

| atemp     | Surface (2-m) air temperature in deg K
| Typical range: 200 < atemp < 300
c | Southwest C-grid tracer point
| Input or input/output field

-----------------------------
c aqh :: Surface (2m) specific humidity in kg/kg
| Typical range: 0 < aqh < 0.02
| Southwest C-grid tracer point
| Input or input/output field

-----------------------------
c lwflux :: Net upward longwave radiation in W/m^2
| lwflux = - ( lwdown - ice and snow absorption - emitted )
| > 0 for decrease in theta (ocean cooling)
| Typical range: -20 < lwflux < 170
| Southwest C-grid tracer point
| Input field

-----------------------------
c evap :: Evaporation in m/s
| > 0 for increase in salt (ocean salinity)
| Typical range: 0 < evap < 2.5e-7
| Southwest C-grid tracer point
| Input, input/output, or output field

-----------------------------
c precip :: Precipitation in m/s
| > 0 for decrease in salt (ocean salinity)
| Typical range: 0 < precip < 5e-7
| Southwest C-grid tracer point
| Input or input/output field

-----------------------------
c snowprecip :: snow in m/s
| > 0 for decrease in salt (ocean salinity)
| Typical range: 0 < precip < 5e-7
| Input or input/output field

-----------------------------
c runoff :: River and glacier runoff in m/s
| > 0 for decrease in salt (ocean salinity)
| Typical range: 0 < runoff < 5e-7
| Southwest C-grid tracer point
| Input or input/output field

-----------------------------
c swdown :: Downward shortwave radiation in W/m^2
| > 0 for increase in theta (ocean warming)
| Typical range: 0 < swdown < 450
| Southwest C-grid tracer point
| Input/output field

-----------------------------
c ldown :: Downward longwave radiation in W/m^2
| > 0 for increase in theta (ocean warming)
| Typical range: 50 < ldown < 450
| Southwest C-grid tracer point
| Input/output field

-----------------------------
c apressure :: Atmospheric pressure field in N/m^2
| Typical range: 88000 < apressure < 108000

(continues on next page)
8.4.7.6 Key subroutines

Top-level routine: exf_getforcing.F

```fortran
!CALLING SEQUENCE:
... 
c  exf_getforcing (TOP LEVEL ROUTINE)
c  |  
c  |-- exf_getclim (get climatological fields used e.g. for relax.)
c  |  |  |  |  |-- exf_set_climsst (relax. to 2-D SST field)
c  |  |  |  |  |-- exf_set_climsss (relax. to 2-D SSS field)
c  |  |  o 
c  |  |-- exf_getffields <- this one does almost everything
    |  |   1. reads in fields, either flux or atmos. state,
c  |  |   depending on CPP options (for each variable two fields
    |  |   consecutive in time are read in and interpolated onto
    |  |   current time step).
c  |  |   2. If forcing is atmos. state and control is atmos. state,
c  |  |   then the control variable anomalies are read here via ctrl_get_gen
    |  |   (atemp, aqh, precip, swflux, swdown, uwind, vwind).
c  |  |   If forcing and control are fluxes, then
    |  |   controls are added later.
c  |  |  o 
c  |-- exf_radiation
    |  |   Compute net or downwelling radiative fluxes via
    |  |   Stefan-Boltzmann law in case only one is known.
c  |  |  o 
c  |-- exf_wind
    |  |   Computes wind speed and stresses, if required.
c  |  |  o 
c  |-- exf_bulkformulae
    |  |   Compute air-sea buoyancy fluxes from
    |  |   atmospheric state following Large and Pond, JPO, 1981/82
    |  |  o 
c  |  |  |-- < hflux is sum of sensible, latent, longwave rad. >
c  |  |  |-- < sflux is sum of evap. minus precip. minus runoff >
c  |  |  |-- exf_getsurfacefluxes
    |  |   If forcing and control is flux, then the
    |  |   control vector anomalies are read here via ctrl_get_gen
    |  |   (hflux, sflux, ustress, vstress)
c  |  |  |-- < update tile edges here >
```

(continues on next page)
Radiation calculation: exf_radiation.F

Wind speed and stress calculation: exf_wind.F

Bulk formula: exf_bulkformulae.F

Generic I/O: exf_set_gen.F

Interpolation: exf_interp.F

Header routines

8.4.7.7 EXF diagnostics

Diagnostics output is available via the diagnostics package (see Section 9.1). Available output fields are summarized below.

<table>
<thead>
<tr>
<th>&lt;Name&gt;</th>
<th>Levs</th>
<th>grid</th>
<th>Units</th>
<th>Tile (max=80c)</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXFhs</td>
<td>1</td>
<td>SM</td>
<td>W/m²</td>
<td>Sensible heat flux into ocean, &gt;0 increases theta</td>
</tr>
<tr>
<td>EXFhl</td>
<td>1</td>
<td>SM</td>
<td>W/m²</td>
<td>Latent heat flux into ocean, &gt;0 increases theta</td>
</tr>
<tr>
<td>EXFlwnet</td>
<td>1</td>
<td>SM</td>
<td>W/m²</td>
<td>Net upward longwave radiation, &gt;0 decreases theta</td>
</tr>
<tr>
<td>EXFswnet</td>
<td>1</td>
<td>SM</td>
<td>W/m²</td>
<td>Net upward shortwave radiation, &gt;0 increases theta</td>
</tr>
<tr>
<td>EXFlwdn</td>
<td>1</td>
<td>SM</td>
<td>W/m²</td>
<td>Downward longwave radiation, &gt;0 decreases theta</td>
</tr>
<tr>
<td>EXFswnet</td>
<td>1</td>
<td>SM</td>
<td>W/m²</td>
<td>Downward shortwave radiation, &gt;0 increases theta</td>
</tr>
<tr>
<td>EXFqnet</td>
<td>1</td>
<td>SM</td>
<td>W/m²</td>
<td>Net upward heat flux (turb+rad), &gt;0 decreases theta</td>
</tr>
<tr>
<td>EXFtaux</td>
<td>1</td>
<td>SU</td>
<td>N/m²</td>
<td>zonal surface wind stress, &gt;0 increases uVel</td>
</tr>
<tr>
<td>EXFtau</td>
<td>1</td>
<td>SV</td>
<td>N/m²</td>
<td>meridional surface wind stress, &gt;0 increases vVel</td>
</tr>
<tr>
<td>EXFuwind</td>
<td>1</td>
<td>SM</td>
<td>m/s</td>
<td>zonal 10-m wind speed, &gt;0 increases uVel</td>
</tr>
<tr>
<td>EXFvwind</td>
<td>1</td>
<td>SM</td>
<td>m/s</td>
<td>meridional 10-m wind speed, &gt;0 increases uVel</td>
</tr>
<tr>
<td>EXFspeee</td>
<td>1</td>
<td>SM</td>
<td>m/s</td>
<td>10-m wind speed modulus ( &gt;= 0 )</td>
</tr>
<tr>
<td>EXFatemp</td>
<td>1</td>
<td>SM</td>
<td>degK</td>
<td>surface (2-m) air temperature</td>
</tr>
<tr>
<td>EXFaq</td>
<td>1</td>
<td>SM</td>
<td>kg/kg</td>
<td>surface (2-m) specific humidity</td>
</tr>
<tr>
<td>EXFevap</td>
<td>1</td>
<td>SM</td>
<td>m/s</td>
<td>evaporation, &gt; 0 increases salinity</td>
</tr>
</tbody>
</table>

(continues on next page)
8.4.7.8 References

8.4.7.9 Experiments and tutorials that use exf

- Global Ocean experiment, in global_with_exf verification directory
- Labrador Sea experiment, in lab_sea verification directory

8.4.8 CAL: The calendar package

Authors: Christian Eckert and Patrick Heimbach

This calendar tool was originally intended to enable the use of absolute dates (Gregorian Calendar dates) in MITgcm. There is, however, a fair number of routines that can be used independently of the main MITgcm executable. After some minor modifications the whole package can be used either as a stand-alone calendar or in connection with any dynamical model that needs calendar dates. Some straightforward extensions are still pending e.g. the availability of the Julian Calendar, to be able to resolve fractions of a second, and to have a time-step that is longer than one day.

8.4.8.1 Basic assumptions for the calendar tool

It is assumed that the SMALLEST TIME INTERVAL to be resolved is ONE SECOND.

Further assumptions are that there is an INTEGER NUMBER OF MODEL STEPS EACH DAY, and that AT LEAST ONE STEP EACH DAY is made.

Not each individual routine depends on these assumptions; there are only a few places where they enter.

8.4.8.2 Format of calendar dates

In this calendar tool a complete date specification is defined as the following integer array:

```
c integer date(4)
c ( yyyyymmdd, hhmmss, leap_year, dayofweek )
c date(1) = yyyyymmdd <-- Year-Month-Day
c date(2) = hhmmss <-- Hours-Minutes-Seconds
c date(3) = leap_year <-- Leap Year/No Leap Year
c date(4) = dayofweek <-- Day of the Week
c leap_year is either equal to 1 (normal year)
c or equal to 2 (leap year)
c dayofweek has a range of 1 to 7.
```
In case the Gregorian Calendar is used, the first day of the week is Friday, since day of the Gregorian Calendar was Friday, 15 Oct. 1582. As a date array this date would be specified as

```
c refdate(1) = 15821015
c refdate(2) = 0
nc refdate(3) = 1
c refdate(4) = 1
```

### 8.4.8.3 Calendar dates and time intervals

Subtracting calendar dates yields time intervals. Time intervals have the following format:

```
c integer datediff(4)
c
c datediff(1) = # Days
nc datediff(2) = hhmmss
nc datediff(3) = 0
nc datediff(4) = -1
```

Such time intervals can be added to or can be subtracted from calendar dates. Time intervals can be added to and be subtracted from each other.

### 8.4.8.4 Using the calendar together with MITgcm

Each routine has as an argument the thread number that it is belonging to, even if this number is not used in the routine itself.

In order to include the calendar tool into the MITgcm setup the MITgcm subroutine “initialise.F” or the routine “initialise_fixed.F”, depending on the MITgcm release, has to be modified in the following way:

```
c #ifdef ALLOW_CALENDAR
nc C-- Initialise the calendar package.
c #ifdef USE_CAL_NENDITER
nc CALL cal_Init(
c I startTime,
c I endTime,
c I deltaTclock,
c I nIter0,
c I nEndIter,
c I nTimeSteps,
c I myThid
&
nc #else
nc CALL cal_Init(
c I startTime,
c I endTime,
c I deltaTclock,
c I nIter0,
c I nTimeSteps,
c I myThid
&
nc #endif
```

(continues on next page)
It is useful to have the CPP flag ALLOW_CAL in order to switch from the usual MITgcm setup to the one that includes the calendar tool. The CPP flag USE_CAL has been introduced in order to enable the use of the calendar for MITgcm releases earlier than checkpoint 25 which do not have the global variable *nEndIter*.

### 8.4.8.5 The individual calendars

Simple model calendar:

This calendar can be used by defining

```c
TheCalendar='model'
```

in the calendar’s data file “data.cal”.

In this case a year is assumed to have 360 days. The model year is divided into 12 months with 30 days each.

Gregorian Calendar:

This calendar can be used by defining

```c
TheCalendar='gregorian'
```

in the calendar’s data file “data.cal”.

### 8.4.8.6 Short routine description

<table>
<thead>
<tr>
<th>Routine</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>cal_Init</code></td>
<td>Initialise the calendar. This is the interface to MITgcm.</td>
</tr>
<tr>
<td><code>cal_Set</code></td>
<td>Sets the calendar according to the user specifications.</td>
</tr>
<tr>
<td><code>cal_GetDate</code></td>
<td>Given the model's current timestep or the model's current time return the corresponding calendar date.</td>
</tr>
<tr>
<td><code>cal_FullDate</code></td>
<td>Complete a date specification (leap year and day of the week).</td>
</tr>
<tr>
<td><code>cal_IsLeap</code></td>
<td>Determine whether a given year is a leap year.</td>
</tr>
<tr>
<td><code>cal_TimePassed</code></td>
<td>Determine the time passed between two dates.</td>
</tr>
<tr>
<td><code>cal_AddTime</code></td>
<td>Add a time interval either to a time interval or to a date.</td>
</tr>
<tr>
<td><code>cal_TimeInterval</code></td>
<td>Given a time interval return the corresponding date array.</td>
</tr>
<tr>
<td><code>cal_SubDates</code></td>
<td>Determine the time interval between two dates</td>
</tr>
</tbody>
</table>
or between two time intervals.

- **cal_ConvDate**: Decompose a date array or a time interval array into its components.

- **cal_CopyDate**: Copy a date array or a time interval array to another array.

- **cal_CompDates**: Compare two calendar dates or time intervals.

- **cal_ToSeconds**: Given a time interval array return the number of seconds.

- **cal_WeekDay**: Return the weekday as a string given the calendar date.

- **cal_NumInts**: Return the number of time intervals between two given dates.

- **cal_StepsPerDay**: Given an iteration number or the current integration time return the number of time steps to integrate in the current calendar day.

- **cal_DaysPerMonth**: Given an iteration number or the current integration time return the number of days to integrate in this calendar month.

- **cal_MonthsPerYear**: Given an iteration number or the current integration time return the number of months to integrate in the current calendar year.

- **cal_StepsForDay**: Given the integration day return the number of steps to be integrated, the first step, and the last step in the day specified. The first and the last step refer to the total number of steps (1, ..., cal_IntSteps).

- **cal_DaysForMonth**: Given the integration month return the number of days to be integrated, the first day, and the last day in the month specified. The first and the last day refer to the total number of steps (1, ..., cal_IntDays).

- **cal_MonthsForYear**: Given the integration year return the number of months to be integrated, the first month, and the last month in the year specified. The first and the last month refer to the total number of steps (1, ..., cal_IntMonths).

- **cal_Intsteps**: Return the number of calendar years that are affected by the current integration.
8.4.8.7 Experiments and tutorials that use cal

- Global ocean experiment in global_with_exf verification directory.
- Labrador Sea experiment in lab_sea verification directory.

8.5 Atmosphere Packages

8.5.1 Atmospheric Intermediate Physics: AIM

Note: The following document below describes the aim_v23 package that is based on the version v23 of the SPEEDY code ()

8.5.1.1 Key subroutines, parameters and files

8.5.1.2 AIM Diagnostics

| <-Name-> | Levs | <-parsing code-> | <- Units --> | <- Tile (max=80c) -->
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>DIABT</td>
<td>5</td>
<td>SM ML</td>
<td>K/s</td>
<td>Pot. Temp. Tendency (Mass-Weighted)</td>
</tr>
</tbody>
</table>

from Diabatic Processes
## 8.5. Atmosphere Packages

<table>
<thead>
<tr>
<th>Variable</th>
<th>Scale</th>
<th>Metric</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DIABQ</td>
<td>5</td>
<td>SM ML</td>
<td>g/kg/s</td>
</tr>
<tr>
<td>RADS</td>
<td>5</td>
<td>SM ML</td>
<td>K/s</td>
</tr>
<tr>
<td>RADL</td>
<td>5</td>
<td>SM ML</td>
<td>K/s</td>
</tr>
<tr>
<td>DTCNV</td>
<td>5</td>
<td>SM MR</td>
<td>K/s</td>
</tr>
<tr>
<td>TURBT</td>
<td>5</td>
<td>SM ML</td>
<td>K/s</td>
</tr>
<tr>
<td>DTLS</td>
<td>5</td>
<td>SM ML</td>
<td>K/s</td>
</tr>
<tr>
<td>DQCONV</td>
<td>5</td>
<td>SM MR</td>
<td>g/kg/s</td>
</tr>
<tr>
<td>TURBQ</td>
<td>5</td>
<td>SM ML</td>
<td>g/kg/s</td>
</tr>
<tr>
<td>DQLS</td>
<td>5</td>
<td>SM ML</td>
<td>g/kg/s</td>
</tr>
<tr>
<td>TSR</td>
<td>1</td>
<td>SM P U1</td>
<td>W/m²</td>
</tr>
<tr>
<td>OLR</td>
<td>1</td>
<td>SM P U1</td>
<td>W/m²</td>
</tr>
<tr>
<td>RADSWG</td>
<td>1</td>
<td>SM P L1</td>
<td>W/m²</td>
</tr>
<tr>
<td>RADLWG</td>
<td>1</td>
<td>SM L1</td>
<td>W/m²</td>
</tr>
<tr>
<td>HFLUX</td>
<td>1</td>
<td>SM L1</td>
<td>W/m²</td>
</tr>
<tr>
<td>EVAP</td>
<td>1</td>
<td>SM L1</td>
<td>g/m²/s</td>
</tr>
<tr>
<td>PRECON</td>
<td>1</td>
<td>SM P L1</td>
<td>g/m²/s</td>
</tr>
<tr>
<td>PRECLS</td>
<td>1</td>
<td>SM M1</td>
<td>g/m²/s</td>
</tr>
<tr>
<td>CLDFRC</td>
<td>1</td>
<td>SM P M1</td>
<td>0-1</td>
</tr>
<tr>
<td>CLDPRS</td>
<td>1</td>
<td>SM PC167M1</td>
<td>0-1</td>
</tr>
<tr>
<td>CLDMAS</td>
<td>5</td>
<td>SM P LL</td>
<td>kg/m²/s</td>
</tr>
<tr>
<td>DRAG</td>
<td>5</td>
<td>SM P LL</td>
<td>kg/m²/s</td>
</tr>
<tr>
<td>WINDS</td>
<td>1</td>
<td>SM P L1</td>
<td>m/s</td>
</tr>
<tr>
<td>TS</td>
<td>1</td>
<td>SM L1</td>
<td>K</td>
</tr>
<tr>
<td>QS</td>
<td>1</td>
<td>SM P L1</td>
<td>g/kg</td>
</tr>
<tr>
<td>ENPREC</td>
<td>1</td>
<td>SM M1</td>
<td>W/m²</td>
</tr>
<tr>
<td>ALBVISDF</td>
<td>1</td>
<td>SM P L1</td>
<td>0-1</td>
</tr>
<tr>
<td>DWNLWG</td>
<td>1</td>
<td>SM P L1</td>
<td>W/m²</td>
</tr>
<tr>
<td>SWCLR</td>
<td>5</td>
<td>SM ML</td>
<td>K/s</td>
</tr>
<tr>
<td>LWCLR</td>
<td>5</td>
<td>SM ML</td>
<td>K/s</td>
</tr>
<tr>
<td>TSRCLR</td>
<td>1</td>
<td>SM P U1</td>
<td>W/m²</td>
</tr>
<tr>
<td>OLRCLR</td>
<td>1</td>
<td>SM P U1</td>
<td>W/m²</td>
</tr>
<tr>
<td>SWGCLR</td>
<td>1</td>
<td>SM P L1</td>
<td>W/m²</td>
</tr>
<tr>
<td>LWGCLR</td>
<td>1</td>
<td>SM L1</td>
<td>W/m²</td>
</tr>
</tbody>
</table>
8.5.3 Experiments and tutorials that use aim

- Global atmosphere experiment in aim.51_cs verification directory.

8.5.2 Land package

8.5.2.1 Introduction

This package provides a simple land model based on Rong Zhang [e-mail: roz@gfdl.noaa.gov] 2 layers model (see documentation below).

It is primarily implemented for AIM (_v23) atmospheric physics but could be adapted to work with a different atmospheric physics. Two subroutines (aim_aim2land.F aim_land2aim.F in pkg/aim_v23) are used as interface with AIM physics.

Number of layers is a parameter (land_nLev in LAND_SIZE.h) and can be changed.

Note on Land Model date: June 1999 author: Rong Zhang

8.5.2.2 Equations and Key Parameters

This is a simple 2-layer land model. The top layer depth $z_1 = 0.1m$, the second layer depth $z_2 = 4m$.

Let $T_{g1}$, $T_{g2}$ be the temperature of each layer, $W_1, W_2$ be the soil moisture of each layer. The field capacity $f_1, f_2$ are the maximum water amount in each layer, so $W_1$ is the ratio of available water to field capacity. $f_i = \gamma z_i, \gamma = 0.24$ is the field capapcity per meter soil, so $f_1 = 0.024m, f_2 = 0.96m$.

The land temperature is determined by total surface downward heat flux $F$,

$$z_1 C_1 \frac{dT_{g1}}{dt} = F - \lambda \frac{T_{g1} - T_{g2}}{(z_1 + z_2)/2}$$

$$z_2 C_2 \frac{dT_{g2}}{dt} = \lambda \frac{T_{g1} - T_{g2}}{(z_1 + z_2)/2}$$

here $C_1, C_2$ are the heat capacity of each layer , $\lambda$=0.42Wm$^{-1}$K$^{-1}\cdot$

$$C_1 = C_w W_1 \gamma + C_s$$

$$C_2 = C_w W_2 \gamma + C_s$$

$C_w, C_s$ are the heat capacity of water and dry soil respectively. $C_w = 4.2 \times 10^6 Jm^{-3}K^{-1}, C_s = 1.13 \times 10^6 Jm^{-3}K^{-1}.$

The soil moisture is determined by precipitation $P(m/s)$,surface evaporation $E(m/s)$ and runoff $R(m/s)$.

$$\frac{dW_1}{dt} = \frac{P - E - R}{f_1} + \frac{W_2 - W_1}{\tau}$$
\( \tau = 2 \text{ days} \) is the time constant for diffusion of moisture between layers.

\[
dW_2 \frac{dt}{\tau} = \frac{f_1}{f_2} (W_1 - W_2)
\]

In the code, \( R = 0 \) gives better result, \( W_1, W_2 \) are set to be within \([0, 1]\). If \( W_1 \) is greater than 1, then let \( \delta W_1 = W_1 - 1, W_1 = 1 \) and \( W_2 = W_2 + \rho \delta W_1 \frac{f_1}{f_2} \), i.e. the runoff of top layer is put into second layer. \( p = 0.5 \) is the fraction of top layer runoff that is put into second layer.

The time step is 1 hour, it takes several years to reach equilibrium offline.

### 8.5.2.3 Land diagnostics

<table>
<thead>
<tr>
<th>Name</th>
<th>Levs</th>
<th>Parsing Code</th>
<th>Units</th>
<th>Tile (max=80c)</th>
</tr>
</thead>
<tbody>
<tr>
<td>GrdSurfT</td>
<td>1</td>
<td>SM Lg</td>
<td>degC</td>
<td>Surface Temperature over land</td>
</tr>
<tr>
<td>GrdTemp</td>
<td>2</td>
<td>SM MG</td>
<td>degC</td>
<td>Ground Temperature at each level</td>
</tr>
<tr>
<td>GrdEnth</td>
<td>2</td>
<td>SM MG</td>
<td>J/m^3</td>
<td>Ground Enthalpy at each level</td>
</tr>
<tr>
<td>GrdWater</td>
<td>2</td>
<td>SM P MG</td>
<td>0-1</td>
<td>Ground Water (vs Field Capacity)</td>
</tr>
<tr>
<td>LdSnowH</td>
<td>1</td>
<td>SM P Lg</td>
<td>m</td>
<td>Snow Thickness over land</td>
</tr>
<tr>
<td>LdSnowAge</td>
<td>1</td>
<td>SM P Lg</td>
<td>s</td>
<td>Snow Age over land</td>
</tr>
<tr>
<td>RUNOFF</td>
<td>1</td>
<td>SM L1</td>
<td>m/s</td>
<td>Run-Off per surface unit</td>
</tr>
<tr>
<td>EnRunOff</td>
<td>1</td>
<td>SM L1</td>
<td>W/m^2</td>
<td>Energy flux associated with run-Off</td>
</tr>
<tr>
<td>landHFlx</td>
<td>1</td>
<td>SM Lg</td>
<td>W/m^2</td>
<td>Net surface downward Heat flux over land</td>
</tr>
<tr>
<td>landPmE</td>
<td>1</td>
<td>SM Lg</td>
<td>kg/m^2/s</td>
<td>Precipitation minus Evaporation over land</td>
</tr>
<tr>
<td>ldEnFxPr</td>
<td>1</td>
<td>SM Lg</td>
<td>W/m^2</td>
<td>Energy flux (over land) associated with Precip (snow,rain)</td>
</tr>
</tbody>
</table>

### 8.5.2.4 References


### 8.5.2.5 Experiments and tutorials that use land

- Global atmosphere experiment in aim.51_cs verification directory.

### 8.5.3 Fizhi: High-end Atmospheric Physics

#### 8.5.3.1 Introduction

The fizhi (high-end atmospheric physics) package includes a collection of state-of-the-art physical parameterizations for atmospheric radiation, cumulus convection, atmospheric boundary layer turbulence, and land surface processes. The collection of atmospheric physics parameterizations were originally used together as part of the GEOS-3 (Goddard Earth Observing System-3) GCM developed at the NASA/Goddard Global Modelling and Assimilation Office (GMAO).
8.5.3.2 Equations

Moist Convective Processes:

Sub-grid and Large-scale Convection

Sub-grid scale cumulus convection is parameterized using the Relaxed Arakawa Schubert (RAS) scheme of [MS92], which is a linearized Arakawa Schubert type scheme. RAS predicts the mass flux from an ensemble of clouds. Each subensemble is identified by its entrainment rate and level of neutral buoyancy which are determined by the grid-scale properties.

The thermodynamic variables that are used in RAS to describe the grid scale vertical profile are the dry static energy, \( s = c_p T + gz \), and the moist static energy, \( h = c_p T + gz + Lq \). The conceptual model behind RAS depicts each subensemble as a rising plume cloud, entraining mass from the environment during ascent, and detraining all cloud air at the level of neutral buoyancy. RAS assumes that the normalized cloud mass flux, \( \eta \), normalized by the cloud base mass flux, is a linear function of height, expressed as:

\[
\frac{\partial \eta(z)}{\partial z} = \lambda \quad \text{or} \quad \frac{\partial \eta(P^c)}{\partial P^c} = \frac{-c_p \theta \lambda}{g}
\]

where we have used the hydrostatic equation written in the form:

\[
\frac{\partial z}{\partial P^c} = \frac{-c_p \theta}{g}
\]

The entrainment parameter, \( \lambda \), characterizes a particular subensemble based on its detrainment level, and is obtained by assuming that the level of detrainment is the level of neutral buoyancy, i.e., the level at which the moist static energy of the cloud, \( h_c \), is equal to the saturation moist static energy of the environment, \( h^* \). Following [MS92], \( \lambda \) may be written as

\[
\lambda = \frac{h_B - h_D^*}{c_p \int_{P_D}^{P_B} \frac{\theta (h_B^* - h)}{P^c} dP^c},
\]

where the subscript \( B \) refers to cloud base, and the subscript \( D \) refers to the detrainment level.

The convective instability is measured in terms of the cloud work function \( A \), defined as the rate of change of cumulus kinetic energy. The cloud work function is related to the buoyancy, or the difference between the moist static energy in the cloud and in the environment:

\[
A = \int_{P_D}^{P_B} \frac{\eta}{1 + \gamma} \left[ \frac{h_c - h^*}{P^c} \right] dP^c
\]

where \( \gamma = \frac{L}{c_p \partial q/\partial T} \) obtained from the Clausius Clapeyron equation, and the subscript \( c \) refers to the value inside the cloud.

To determine the cloud base mass flux, the rate of change of \( A \) in time due to dissipation by the clouds is assumed to approximately balance the rate of change of \( A \) due to the generation by the large scale. This is the quasi-equilibrium assumption, and results in an expression for \( m_B \):

\[
m_B = -\left. \frac{dA}{dt} \right|_{ls} K
\]

where \( K \) is the cloud kernel, defined as the rate of change of the cloud work function per unit cloud base mass flux, and is currently obtained by analytically differentiating the expression for \( A \) in time. The rate of change of \( A \) due to the generation by the large scale can be written as the difference between the current \( A(t + \Delta t) \) and its equilibrated value after the previous convective time step \( A(t) \), divided by the time step. \( A(t) \) is approximated as some critical \( A_{crit} \), computed by Lord (1982) from in situ observations.
The predicted convective mass fluxes are used to solve grid-scale temperature and moisture budget equations to determine the impact of convection on the large scale fields of temperature (through latent heating and compensating subsidence) and moisture (through precipitation and detrainment):

\[ \frac{\partial \theta}{\partial t} \bigg|_c = \alpha \frac{m_B}{c_p P_k \eta} \frac{\partial s}{\partial p} \]

and

\[ \frac{\partial q}{\partial t} \bigg|_c = \alpha \frac{m_B}{L \eta} (\frac{\partial h}{\partial p} - \frac{\partial s}{\partial p}) \]

where \( \theta = \frac{T}{T_e}, P = (p/p_0), \) and \( \alpha \) is the relaxation parameter.

As an approximation to a full interaction between the different allowable subensembles, many clouds are simulated frequently, each modifying the large scale environment some fraction \( \alpha \) of the total adjustment. The parameterization thereby “relaxes” the large scale environment towards equilibrium.

In addition to the RAS cumulus convection scheme, the fizhi package employs a Kessler-type scheme for the re-evaporation of falling rain [SM88], which correspondingly adjusts the temperature assuming \( h \) is conserved. RAS in its current formulation assumes that all cloud water is deposited into the detrainment level as rain. All of the rain is available for re-evaporation, which begins in the level below detrainment. The scheme accounts for some microphysics such as the rainfall intensity, the drop size distribution, as well as the temperature, pressure and relative humidity of the surrounding air. The fraction of the moisture deficit in any model layer into which the rain may re-evaporate is controlled by a free parameter, which allows for a relatively efficient re-evaporation of liquid precipitate and larger rainout for frozen precipitation.

Due to the increased vertical resolution near the surface, the lowest model layers are averaged to provide a 50 mb thick sub-cloud layer for RAS. Each time RAS is invoked (every ten simulated minutes), a number of randomly chosen subensembles are checked for the possibility of convection, from just above cloud base to 10 mb.

Supersaturation or large-scale precipitation is initiated in the fizhi package whenever the relative humidity in any grid-box exceeds a critical value, currently 100 %. The large-scale precipitation re-evaporates during descent to partially saturate lower layers in a process identical to the re-evaporation of convective rain.

**Cloud Formation**

Convective and large-scale cloud fractions which are used for cloud-radiative interactions are determined diagnostically as part of the cumulus and large-scale parameterizations. Convective cloud fractions produced by RAS are proportional to the detrained liquid water amount given by

\[ F_{RAS} = \min \left[ \frac{L_{RAS}}{l_c}, 1.0 \right] \]

where \( l_c \) is an assigned critical value equal to 1.25 g/kg. A memory is associated with convective clouds defined by:

\[ F^n_{RAS} = \min \left[ F_{RAS} + (1 - \frac{\Delta t_{RAS}}{\tau}) F^{n-1}_{RAS}, 1.0 \right] \]

where \( F_{RAS} \) is the instantaneous cloud fraction and \( F^{n-1}_{RAS} \) is the cloud fraction from the previous RAS timestep. The memory coefficient is computed using a RAS cloud timescale, \( \tau \), equal to 1 hour. RAS cloud fractions are cleared when they fall below 5 %.

Large-scale cloudiness is defined, following Slingo and Ritter (1985), as a function of relative humidity:

\[ F_{LS} = \min \left[ \left( \frac{RH - RH_e}{1 - RH_e} \right)^2, 1.0 \right] \]
where

\[ \text{RH}_c = 1 - s(1-s)(2+2s) \frac{r}{s} = \frac{p}{p_{\text{surf}}} , \quad \text{RH}_{\text{min}} = 0.75 = 0.573285 . \]

These cloud fractions are suppressed, however, in regions where the convective sub-cloud layer is conditionally unstable. The functional form of \( \text{RH}_c \) is shown in Figure 8.9.

![Figure 8.9: Critical Relative Humidity for Clouds.](image)

The total cloud fraction in a grid box is determined by the larger of the two cloud fractions:

\[ F_{\text{CLD}} = \max [F_{\text{RAS}}, F_{\text{LS}}] . \]

Finally, cloud fractions are time-averaged between calls to the radiation packages.

Radiation:

The parameterization of radiative heating in the fizhi package includes effects from both shortwave and longwave...
Radiative fluxes are calculated at each model edge-level in both up and down directions. The heating rates/cooling rates are then obtained from the vertical divergence of the net radiative fluxes. The net flux is

\[ F = F^\uparrow - F^\downarrow \]

where \( F \) is the net flux, \( F^\uparrow \) is the upward flux and \( F^\downarrow \) is the downward flux.

The heating rate due to the divergence of the radiative flux is given by

\[ \frac{\partial p c_p T}{\partial t} = -\frac{\partial F}{\partial z} \]

or

\[ \frac{\partial T}{\partial t} = \frac{g}{c_p \pi} \frac{\partial F}{\partial \sigma} \]

where \( g \) is the acceleration due to gravity and \( c_p \) is the heat capacity of air at constant pressure.

The time tendency for Longwave Radiation is updated every 3 hours. The time tendency for Shortwave Radiation is updated once every three hours assuming a normalized incident solar radiation, and subsequently modified at every model time step by the true incident radiation. The solar constant value used in the package is equal to 1365 W/m² and a CO₂ mixing ratio of 330 ppm. For the ozone mixing ratio, monthly mean zonally averaged climatological values specified as a function of latitude and height [RSG87] are linearly interpolated to the current time.

### Shortwave Radiation

The shortwave radiation package used in the package computes solar radiative heating due to the absorption by water vapor, ozone, carbon dioxide, oxygen, clouds, and aerosols and due to the scattering by clouds, aerosols, and gases. The shortwave radiative processes are described by [Cho90, Cho92]. This shortwave package uses the Delta-Eddington approximation to compute the bulk scattering properties of a single layer following King and Harshvardhan (JAS, 1986). The transmittance and reflectance of diffuse radiation follow the procedures of Sagan and Pollock (JGR, 1967) and [LH74].

Highly accurate heating rate calculations are obtained through the use of an optimal grouping strategy of spectral bands. By grouping the UV and visible regions as indicated in Table 8.11, the Rayleigh scattering and the ozone absorption of solar radiation can be accurately computed in the ultraviolet region and the photosynthetically active radiation (PAR) region. The computation of solar flux in the infrared region is performed with a broadband parameterization using the spectrum regions shown in Table 8.12. The solar radiation algorithm used in the fizhi package can be applied not only for climate studies but also for studies on the photolysis in the upper atmosphere and the photosynthesis in the biosphere.
Table 8.11: UV and Visible Spectral Regions used in shortwave radiation package.

<table>
<thead>
<tr>
<th>UV and Visible Spectral Regions</th>
<th>Band</th>
<th>Wavelength (micron)</th>
</tr>
</thead>
<tbody>
<tr>
<td>UV-C</td>
<td>1.</td>
<td>.175 -.225</td>
</tr>
<tr>
<td></td>
<td>2.</td>
<td>.225 -.245</td>
</tr>
<tr>
<td></td>
<td>3.</td>
<td>.245 -.260</td>
</tr>
<tr>
<td>UV-B</td>
<td>4.</td>
<td>.280 -.295</td>
</tr>
<tr>
<td></td>
<td>5.</td>
<td>.295 -.310</td>
</tr>
<tr>
<td></td>
<td>6.</td>
<td>.310 -.320</td>
</tr>
<tr>
<td>UV-A</td>
<td>7.</td>
<td>.320 -.400</td>
</tr>
<tr>
<td>PAR</td>
<td>8.</td>
<td>.400 -.700</td>
</tr>
</tbody>
</table>

Table 8.12: Infrared Spectral Regions used in shortwave radiation package.

<table>
<thead>
<tr>
<th>Infrared Spectral Regions</th>
<th>Band</th>
<th>Wavenumber (cm(^{-1}))</th>
<th>Wavelength (micron)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>1000-4400</td>
<td>2.27-10.0</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>4400-8200</td>
<td>1.22-2.27</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>8200-14300</td>
<td>0.70-1.22</td>
</tr>
</tbody>
</table>

Within the shortwave radiation package, both ice and liquid cloud particles are allowed to co-exist in any of the model layers. Two sets of cloud parameters are used, one for ice particles and the other for liquid particles. Cloud parameters are defined as the cloud optical thickness and the effective cloud particle size. In the fizhi package, the effective radius for water droplets is given as 10 microns, while 65 microns is used for ice particles. The absorption due to aerosols is currently set to zero.

To simplify calculations in a cloudy atmosphere, clouds are grouped into low \((p > 700 \text{ mb})\), middle \((700 \text{ mb} \geq p > 400 \text{ mb})\), and high \((p < 400 \text{ mb})\) cloud regions. Within each of the three regions, clouds are assumed maximally overlapped, and the cloud cover of the group is the maximum cloud cover of all the layers in the group. The optical thickness of a given layer is then scaled for both the direct (as a function of the solar zenith angle) and diffuse beam radiation so that the grouped layer reflectance is the same as the original reflectance. The solar flux is computed for each of eight cloud realizations possible within this low/middle/high classification, and appropriately averaged to produce the net solar flux.
Longwave Radiation

The longwave radiation package used in the fizhi package is thoroughly described by [CS94]. As described in that document, IR fluxes are computed due to absorption by water vapor, carbon dioxide, and ozone. The spectral bands together with their absorbers and parameterization methods, configured for the fizhi package, are shown in Table 8.13.

Table 8.13: IR Spectral Bands, Absorbers, and Parameterization Method

<table>
<thead>
<tr>
<th>IR Spectral Bands</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Band</td>
<td>Spectral Range (cm⁻¹)</td>
</tr>
<tr>
<td>1</td>
<td>0-340</td>
</tr>
<tr>
<td>2</td>
<td>340-540</td>
</tr>
<tr>
<td>3a</td>
<td>540-620</td>
</tr>
<tr>
<td>3b</td>
<td>620-720</td>
</tr>
<tr>
<td>3b</td>
<td>720-800</td>
</tr>
<tr>
<td>4</td>
<td>800-980</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>980-1100</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>1100-1380</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>1380-1900</td>
</tr>
<tr>
<td>8</td>
<td>1900-3000</td>
</tr>
</tbody>
</table>

K: k-distribution method with linear pressure scaling  
T: Table look-up with temperature and pressure scaling  
S: One-parameter temperature scaling

The longwave radiation package accurately computes cooling rates for the middle and lower atmosphere from 0.01 mb to the surface. Errors are < 0.4 C day⁻¹ in cooling rates and < 1% in fluxes. From Chou and Suarez, it is estimated that the total effect of neglecting all minor absorption bands and the effects of minor infrared absorbers such as nitrous oxide (N:math: _2O), methane (CH:math: _4), and the chlorofluorocarbons (CFCs), is an underestimate of ≈ 5 W/m² in the downward flux at the surface and an overestimate of ≈ 3 W/m² in the upward flux at the top of the atmosphere.

Similar to the procedure used in the shortwave radiation package, clouds are grouped into three regions categorized as low/middle/high. The net clear line-of-site probability (P) between any two levels, p₁ and p₂ (p₂ > p₁), assuming randomly overlapped cloud groups, is simply the product of the probabilities within each group:

\[ P_{net} = P_{low} \times P_{mid} \times P_{hi}. \]

Since all clouds within a group are assumed maximally overlapped, the clear line-of-site probability within a group is given by:

\[ P_{group} = 1 - F_{max}, \]

where \( F_{max} \) is the maximum cloud fraction encountered between \( p_1 \) and \( p_2 \) within that group. For groups and/or levels outside the range of \( p_1 \) and \( p_2 \), a clear line-of-site probability equal to 1 is assigned.
Cloud-Radiation Interaction

The cloud fractions and diagnosed cloud liquid water produced by moist processes within the fizhi package are used in the radiation packages to produce cloud-radiative forcing. The cloud optical thickness associated with large-scale cloudiness is made proportional to the diagnosed large-scale liquid water, $\ell$, detrained due to super-saturation. Two values are used corresponding to cloud ice particles and water droplets. The range of optical thickness for these clouds is given as

$$0.0002 \leq \tau_{\text{ice}}(mb^{-1}) \leq 0.002 \quad \text{for} \quad 0 \leq \ell \leq 2 \ \text{mg/kg},$$

$$0.02 \leq \tau_{\text{h2o}}(mb^{-1}) \leq 0.2 \quad \text{for} \quad 0 \leq \ell \leq 10 \ \text{mg/kg}.$$  

The partitioning, $\alpha$, between ice particles and water droplets is achieved through a linear scaling in temperature:

$$0 \leq \alpha \leq 1 \quad \text{for} \quad 233.15 \leq T \leq 253.15.$$  

The resulting optical depth associated with large-scale cloudiness is given as

$$\tau_{\text{LS}} = \alpha \tau_{\text{h2o}} + (1 - \alpha)\tau_{\text{ice}}.$$  

The optical thickness associated with sub-grid scale convective clouds produced by RAS is given as

$$\tau_{\text{RAS}} = 0.16 \ \text{mb}^{-1}.$$  

The total optical depth in a given model layer is computed as a weighted average between the large-scale and sub-grid scale optical depths, normalized by the total cloud fraction in the layer:

$$\tau = \left( \frac{F_{\text{RAS}} \tau_{\text{RAS}} + F_{\text{LS}} \tau_{\text{LS}}}{F_{\text{RAS}} + F_{\text{LS}}} \right) \Delta p,$$

where $F_{\text{RAS}}$ and $F_{\text{LS}}$ are the time-averaged cloud fractions associated with RAS and large-scale processes described in Section [sec:fizhi:clouds]. The optical thickness for the longwave radiative feedback is assumed to be 75% of these values.

The entire Moist Convective Processes Module is called with a frequency of 10 minutes. The cloud fraction values are time-averaged over the period between Radiation calls (every 3 hours). Therefore, in a time-averaged sense, both convective and large-scale cloudiness can exist in a given grid-box.

Turbulence

Turbulence is parameterized in the fizhi package to account for its contribution to the vertical exchange of heat, moisture, and momentum. The turbulence scheme is invoked every 30 minutes, and employs a backward-implicit iterative time scheme with an internal time step of 5 minutes. The tendencies of atmospheric state variables due to turbulent diffusion are calculated using the diffusion equations:

$$\frac{\partial u}{\partial t_{\text{turb}}} = \frac{\partial}{\partial z}(-u'w') = \frac{\partial}{\partial z}(K_m \frac{\partial u}{\partial z}),$$

$$\frac{\partial v}{\partial t_{\text{turb}}} = \frac{\partial}{\partial z}(-v'w') = \frac{\partial}{\partial z}(K_m \frac{\partial v}{\partial z}),$$

$$\frac{\partial T}{\partial t} = P^e \frac{\partial \theta}{\partial t_{\text{turb}}} = P^e \frac{\partial}{\partial z}(-w'\theta') = P^e \frac{\partial}{\partial z}(K_h \frac{\partial \theta}{\partial z}),$$

$$\frac{\partial q}{\partial t_{\text{turb}}} = \frac{\partial}{\partial z}(-w'q') = \frac{\partial}{\partial z}(K_h \frac{\partial q}{\partial z}).$$
Within the atmosphere, the time evolution of second turbulent moments is explicitly modeled by representing the third moments in terms of the first and second moments. This approach is known as a second-order closure modeling. To simplify and streamline the computation of the second moments, the level 2.5 assumption of Mellor and Yamada (1974) and [Yam77] is employed, in which only the turbulent kinetic energy (TKE),

\[
\frac{1}{2}q^2 = u'^2 + v'^2 + w'^2,
\]

is solved prognostically and the other second moments are solved diagnostically. The prognostic equation for TKE allows the scheme to simulate some of the transient and diffusive effects in the turbulence. The TKE budget equation is solved numerically using an implicit backward computation of the terms linear in \(q^2\) and is written:

\[
\frac{d}{dt}\left(\frac{1}{2}q^2\right) + \frac{\partial}{\partial z}\left(\frac{5}{3}\lambda_1 q\frac{\partial}{\partial z}\left(\frac{1}{2}q^2\right)\right) = -u'w\frac{\partial U}{\partial z} - v'w\frac{\partial V}{\partial z} + \frac{g}{\Theta_0}w'\theta_v - \frac{q^3}{\Lambda_1},
\]

where \(q\) is the turbulent velocity, \(u', v', w'\) and \(\theta'\) are the fluctuating parts of the velocity components and potential temperature, \(U\) and \(V\) are the mean velocity components, \(\Theta_0^{-1}\) is the coefficient of thermal expansion, and \(\lambda_1\) and \(\Lambda_1\) are constant multiples of the master length scale, \(\ell\), which is designed to be a characteristic measure of the vertical structure of the turbulent layers.

The first term on the left-hand side represents the time rate of change of TKE, and the second term is a representation of the triple correlation, or turbulent transport term. The first three terms on the right-hand side represent the sources of TKE due to shear and buoyancy, and the last term on the right hand side is the dissipation of TKE.

In the level 2.5 approach, the vertical fluxes of the scalars \(\theta\) and \(q\) and the wind components \(u\) and \(v\) are expressed in terms of the diffusion coefficients \(K_h\) and \(K_m\), respectively. In the statistically realizable level 2.5 turbulence scheme of [HL88], these diffusion coefficients are expressed as

\[
K_h = \begin{cases} 
q \ell S_H(G_M, G_H) & \text{for decaying turbulence} \\
\frac{q^2}{\ell} S_H(G_{M_e}, G_{H_e}) & \text{for growing turbulence}
\end{cases}
\]

and

\[
K_m = \begin{cases} 
q \ell S_M(G_M, G_H) & \text{for decaying turbulence} \\
\frac{q^2}{\ell} S_M(G_{M_e}, G_{H_e}) & \text{for growing turbulence}
\end{cases}
\]

where the subscript \(e\) refers to the value under conditions of local equilibrium (obtained from the Level 2.0 Model), \(\ell\) is the master length scale related to the vertical structure of the atmosphere, and \(S_M\) and \(S_H\) are functions of \(G_H\) and \(G_M\), the dimensionless buoyancy and wind shear parameters, respectively. Both \(G_H\) and \(G_M\), and their equilibrium values \(G_{H_e}\) and \(G_{M_e}\), are functions of the Richardson number:

\[
RI = \frac{g_0 \frac{\partial \theta_e}{\partial z}}{(\frac{\partial u}{\partial z})^2 + (\frac{\partial v}{\partial z})^2} = \frac{C_p \frac{\partial \theta}{\partial z}}{(\frac{\partial u}{\partial z})^2 + (\frac{\partial v}{\partial z})^2}.
\]

Negative values indicate unstable buoyancy and shear, small positive values (\(< 0.2\)) indicate dominantly unstable shear, and large positive values indicate dominantly stable stratification.

Turbulent eddy diffusion coefficients of momentum, heat and moisture in the surface layer, which corresponds to the lowest GCM level (see — missing table —), are calculated using stability-dependant functions based on Monin-Obukhov theory:

\[
K_m(\text{surface}) = C_u \times u_s = C_D W_s
\]

and

\[
K_h(\text{surface}) = C_t \times u_s = C_H W_s
\]

where \(u_s = C_u W_s\) is the surface friction velocity, \(C_D\) is termed the surface drag coefficient, \(C_H\) the heat transfer coefficient, and \(W_s\) is the magnitude of the surface layer wind.
$C_u$ is the dimensionless exchange coefficient for momentum from the surface layer similarity functions:

$$C_u = \frac{u_*}{W_s} = \frac{k}{\psi_m}$$

where $k$ is the Von Karman constant and $\psi_m$ is the surface layer non-dimensional wind shear given by

$$\psi_m = \int_{\zeta_0}^{\zeta} \frac{\phi_m}{\zeta} d\zeta.$$  

Here $\zeta$ is the non-dimensional stability parameter, and $\phi_m$ is the similarity function of $\zeta$ which expresses the stability dependence of the momentum gradient. The functional form of $\phi_m$ is specified differently for stable and unstable layers.

$C_t$ is the dimensionless exchange coefficient for heat and moisture from the surface layer similarity functions:

$$C_t = -\frac{(w' \theta')}{u_* \Delta \theta} = -\frac{(w' q')}{u_* \Delta q} = \frac{k}{(\psi_h + \psi_g)}$$

where $\psi_h$ is the surface layer non-dimensional temperature gradient given by

$$\psi_h = \int_{\zeta_0}^{\zeta} \frac{\phi_h}{\zeta} d\zeta.$$  

Here $\phi_h$ is the similarity function of $\zeta$, which expresses the stability dependence of the temperature and moisture gradients, and is specified differently for stable and unstable layers according to [HS95].

$\psi_g$ is the non-dimensional temperature or moisture gradient in the viscous sublayer, which is the most laminar region between the surface and the tops of the roughness elements, in which temperature and moisture gradients can be quite large. Based on [YK74]:

$$\psi_g = \frac{0.55(P_r^{2/3} - 0.2)}{\nu^{1/2}}(h_0 u_* - h_{0,ref} u_{*ref})^{1/2}$$

where $Pr$ is the Prandtl number for air, $\nu$ is the molecular viscosity, $z_0$ is the surface roughness length, and the subscript $ref$ refers to a reference value. $h_0 = 30 z_0$ with a maximum value over land of 0.01.

The surface roughness length over oceans is a function of the surface-stress velocity,

$$z_0 = c_1 u_*^3 + c_2 u_*^2 + c_3 u_* + c_4 + \frac{c_5}{u_*}$$

where the constants are chosen to interpolate between the reciprocal relation of [Kon75] for weak winds, and the piecewise linear relation of [LP81] for moderate to large winds. Roughness lengths over land are specified from the climatology of [DS89].

For an unstable surface layer, the stability functions, chosen to interpolate between the condition of small values of $\beta$ and the convective limit, are the KEYPS function [Pan73] for momentum, and its generalization for heat and moisture:

$$\phi_m^4 - 18 \zeta \phi_m^3 = 1 \quad ; \quad \phi_h^2 - 18 \zeta \phi_h^3 = 1$$

The function for heat and moisture assures non-vanishing heat and moisture fluxes as the wind speed approaches zero.

For a stable surface layer, the stability functions are the observationally based functions of [Cla70], slightly modified for the momentum flux:

$$\phi_m = \frac{1 + 5\zeta_1}{1 + 0.00794\zeta_1 (1 + 5\zeta_1)} \quad ; \quad \phi_h = \frac{1 + 5\zeta_1}{1 + 0.00794\zeta_1 (1 + 5\zeta_1)}.$$  

The moisture flux also depends on a specified evapotranspiration coefficient, set to unity over oceans and dependant on the climatological ground wetness over land.

Once all the diffusion coefficients are calculated, the diffusion equations are solved numerically using an implicit backward operator.
Atmospheric Boundary Layer

The depth of the atmospheric boundary layer (ABL) is diagnosed by the parameterization as the level at which the turbulent kinetic energy is reduced to a tenth of its maximum near surface value. The vertical structure of the ABL is explicitly resolved by the lowest few (3-8) model layers.

Surface Energy Budget

The ground temperature equation is solved as part of the turbulence package using a backward implicit time differencing scheme:

\[ C_g \frac{\partial T_g}{\partial t} = R_{sw} - R_{lw} + Q_{ice} - H - LE \]

where \( R_{sw} \) is the net surface downward shortwave radiative flux and \( R_{lw} \) is the net surface upward longwave radiative flux.

\( H \) is the upward sensible heat flux, given by:

\[ H = P^s \rho c_p C_H \omega_s (\theta_{surface} - \theta_{LAY}) \quad \text{where:} \quad C_H = C_u C_t \]

where \( \rho \) is the atmospheric density at the surface, \( c_p \) is the specific heat of air at constant pressure, and \( \theta \) represents the potential temperature of the surface and of the lowest \( \sigma \)-level, respectively.

The upward latent heat flux, \( LE \), is given by

\[ LE = \rho \beta L C_H \omega_s (q_{surface} - q_{LAY}) \quad \text{where:} \quad C_H = C_u C_t \]

where \( \beta \) is the fraction of the potential evapotranspiration actually evaporated, \( L \) is the latent heat of evaporation, and \( q_{surface} \) and \( q_{LAY} \) are the specific humidity of the surface and of the lowest \( \sigma \)-level, respectively.

The heat conduction through sea ice, \( Q_{ice} \), is given by

\[ Q_{ice} = \frac{C_{ti} H_i}{T_i - T_g} \]

where \( C_{ti} \) is the thermal conductivity of ice, \( H_i \) is the ice thickness, assumed to be 3 m where sea ice is present, \( T_i \) is 273 degrees Kelvin, and \( T_g \) is the surface temperature of the ice.

\( C_g \) is the total heat capacity of the ground, obtained by solving a heat diffusion equation for the penetration of the diurnal cycle into the ground (\( \lambda \)), and is given by:

\[ C_g = \sqrt{\frac{\lambda C_s}{2 \omega}} = \left( \frac{0.386 + 0.536 W + 0.15 W^2}{0.86400} \right) \times 10^{-3} \times \frac{86400}{2\pi}. \]

Here, the thermal conductivity, \( \lambda \), is equal to \( 2 \times 10^{-3} \) \( \text{ly cm sec}^{-1} K^{-1} \), the angular velocity of the earth, \( \omega \), is written as 86400 \( \text{sec/day} \) divided by \( 2\pi \) \( \text{radians/day} \), and the expression for \( C_s \), the heat capacity per unit volume at the surface, is a function of the ground wetness, \( W \).

Land Surface Processes:
Surface Type

The fizhi package surface Types are designated using the Koster-Suarez [KS91, KS92] Land Surface Model (LSM) mosaic philosophy which allows multiple “tiles”, or multiple surface types, in any one grid cell. The Koster-Suarez LSM surface type classifications are shown in Table 8.14. The surface types and the percent of the grid cell occupied by any surface type were derived from the surface classification of [DT94], and information about the location of permanent ice was obtained from the classifications of [DS89]. The surface type map for a 1° grid is shown in Figure 8.10. The determination of the land or sea category of surface type was made from NCAR’s 10 minute by 10 minute Navy topography dataset, which includes information about the percentage of water-cover at any point. The data were averaged to the model’s grid resolutions, and any grid-box whose averaged water percentage was ≥ 60% was defined as a water point. The Land-Water designation was further modified subjectively to ensure sufficient representation from small but isolated land and water regions.

Table 8.14: Surface Type Designation

<table>
<thead>
<tr>
<th>Type</th>
<th>Vegetation Designation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Broadleaf Evergreen Trees</td>
</tr>
<tr>
<td>2</td>
<td>Broadleaf Deciduous Trees</td>
</tr>
<tr>
<td>3</td>
<td>Needleleaf Trees</td>
</tr>
<tr>
<td>4</td>
<td>Ground Cover</td>
</tr>
<tr>
<td>5</td>
<td>Broadleaf Shrubs</td>
</tr>
<tr>
<td>6</td>
<td>Dwarf Trees (Tundra)</td>
</tr>
<tr>
<td>7</td>
<td>Bare Soil</td>
</tr>
<tr>
<td>8</td>
<td>Desert (Bright)</td>
</tr>
<tr>
<td>9</td>
<td>Glacier</td>
</tr>
<tr>
<td>10</td>
<td>Desert (Dark)</td>
</tr>
<tr>
<td>100</td>
<td>Ocean</td>
</tr>
</tbody>
</table>

Surface Roughness

The surface roughness length over oceans is computed iteratively with the wind stress by the surface layer parameterization [HS95]. It employs an interpolation between the functions of [LP81] for high winds and of [Kon75] for weak winds.

Albedo

The surface albedo computation, described in , employs the “two stream” approximation used in Sellers’ (1987) Simple Biosphere (SiB) Model which distinguishes between the direct and diffuse albedos in the visible and in the near infrared spectral ranges. The albedos are functions of the observed leaf area index (a description of the relative orientation of the leaves to the sun), the greenness fraction, the vegetation type, and the solar zenith angle. Modifications are made to account for the presence of snow, and its depth relative to the height of the vegetation elements.
Figure 8.10: Surface type combinations
Gravity Wave Drag

The fizhi package employs the gravity wave drag scheme of [ZSL95]. This scheme is a modified version of Vernekar et al. (1992), which was based on Alpert et al. (1988) and Helfand et al. (1987). In this version, the gravity wave stress at the surface is based on that derived by Pierrehumbert (1986) and is given by:

\[
|\vec{\tau}_{sfc}| = \rho U^3 \frac{N^2}{N^*} \left( \frac{F_r^2}{1 + F_r^2} \right),
\]

where \( F_r = N h / U \) is the Froude number, \( N \) is the Brunt - Väisälä frequency, \( U \) is the surface wind speed, \( h \) is the standard deviation of the sub-grid scale orography, and \( N^* \) is the wavelength of the monochromatic gravity wave in the direction of the low-level wind. A modification introduced by Zhou et al. allows for the momentum flux to escape through the top of the model, although this effect is small for the current 70-level model. The subgrid scale standard deviation is defined by \( h \), and is not allowed to exceed 400 m.

The effects of using this scheme within a GCM are shown in [TS96]. Experiments using the gravity wave drag parameterization yielded significant and beneficial impacts on both the time-mean flow and the transient statistics of the a GCM climatology, and have eliminated most of the worst dynamically driven biases in the a GCM simulation. An examination of the angular momentum budget during climate runs indicates that the resulting gravity wave torque is similar to the data-driven torque produced by a data assimilation which was performed without gravity wave drag. It was shown that the inclusion of gravity wave drag results in large changes in both the mean flow and in eddy fluxes. The result is a more accurate simulation of surface stress (through a reduction in the surface wind strength), of mountain torque (through a redistribution of mean sea-level pressure), and of momentum convergence (through a reduction in the flux of westerly momentum by transient flow eddies).

Boundary Conditions and other Input Data:

Required fields which are not explicitly predicted or diagnosed during model execution must either be prescribed internally or obtained from external data sets. In the fizhi package these fields include: sea surface temperature, sea ice extent, surface geopotential variance, vegetation index, and the radiation-related background levels of: ozone, carbon dioxide, and stratospheric moisture.

Boundary condition data sets are available at the model’s resolutions for either climatological or yearly varying conditions. Any frequency of boundary condition data can be used in the fizhi package; however, the current selection of data is summarized in Table 8.15. The time mean values are interpolated during each model timestep to the current time.

Table 8.15: Boundary conditions and other input data used in the fizhi package. Also noted are the current years and frequencies available.

<table>
<thead>
<tr>
<th>Fizhi Input Datasets</th>
<th>Frequency</th>
<th>Availability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sea Ice Extent</td>
<td>monthly</td>
<td>1979-current, climatology</td>
</tr>
<tr>
<td>Sea Ice Extent</td>
<td>weekly</td>
<td>1982-current, climatology</td>
</tr>
<tr>
<td>Sea Surface Temperature</td>
<td>monthly</td>
<td>1979-current, climatology</td>
</tr>
<tr>
<td>Sea Surface Temperature</td>
<td>weekly</td>
<td>1982-current, climatology</td>
</tr>
<tr>
<td>Zonally Averaged Upper-Level Moisture</td>
<td>monthly</td>
<td>climatology</td>
</tr>
<tr>
<td>Zonally Averaged Ozone Concentration</td>
<td>monthly</td>
<td>climatology</td>
</tr>
</tbody>
</table>
Topography and Topography Variance

Surface geopotential heights are provided from an averaging of the Navy 10 minute by 10 minute dataset supplied by the National Center for Atmospheric Research (NCAR) to the model’s grid resolution. The original topography is first rotated to the proper grid-orientation which is being run, and then averages the data to the model resolution.

The standard deviation of the subgrid-scale topography is computed by interpolating the 10 minute data to the model’s resolution and re-interpolating back to the 10 minute by 10 minute resolution. The sub-grid scale variance is constructed based on this smoothed dataset.

Upper Level Moisture

The fizhi package uses climatological water vapor data above 100 mb from the Stratospheric Aerosol and Gas Experiment (SAGE) as input into the model’s radiation packages. The SAGE data is archived as monthly zonal means at 5° latitudinal resolution. The data is interpolated to the model’s grid location and current time, and blended with the GCM’s moisture data. Below 300 mb, the model’s moisture data is used. Above 100 mb, the SAGE data is used. Between 100 and 300 mb, a linear interpolation (in pressure) is performed using the data from SAGE and the GCM.

8.5.3.3 Fizhi Diagnostics

Fizhi Diagnostic Menu: [sec:pkg:fizhi:diagnostics]

<table>
<thead>
<tr>
<th>NAME</th>
<th>UNITS</th>
<th>LEVELS</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>UFLUX</td>
<td>N m$^{-2}$</td>
<td>1</td>
<td>Surface U-Wind Stress on the atmosphere</td>
</tr>
<tr>
<td>VFLUX</td>
<td>N m$^{-2}$</td>
<td>1</td>
<td>Surface V-Wind Stress on the atmosphere</td>
</tr>
<tr>
<td>HFLUX</td>
<td>W m$^{-2}$</td>
<td>1</td>
<td>Surface Flux of Sensible Heat</td>
</tr>
<tr>
<td>EFLUX</td>
<td>W m$^{-2}$</td>
<td>1</td>
<td>Surface Flux of Latent Heat</td>
</tr>
<tr>
<td>QICE</td>
<td>W m$^{-2}$</td>
<td>1</td>
<td>Heat Conduction through Sea-Ice</td>
</tr>
<tr>
<td>RADLWG</td>
<td>W m$^{-2}$</td>
<td>1</td>
<td>Net upward LW flux at the ground</td>
</tr>
<tr>
<td>RADSWG</td>
<td>W m$^{-2}$</td>
<td>1</td>
<td>Net downward SW flux at the ground</td>
</tr>
<tr>
<td>RI</td>
<td>dimensionless</td>
<td>Nrphys</td>
<td>Richardson Number</td>
</tr>
<tr>
<td>CT</td>
<td>dimensionless</td>
<td>1</td>
<td>Surface Drag coefficient for T and Q</td>
</tr>
<tr>
<td>CU</td>
<td>dimensionless</td>
<td>1</td>
<td>Surface Drag coefficient for U and V</td>
</tr>
<tr>
<td>ET</td>
<td>m$^{2}$ s$^{-1}$</td>
<td>Nrphys</td>
<td>Diffusivity coefficient for T and Q</td>
</tr>
<tr>
<td>EU</td>
<td>m$^{2}$ s$^{-1}$</td>
<td>Nrphys</td>
<td>Diffusivity coefficient for U and V</td>
</tr>
<tr>
<td>TURBU</td>
<td>m s$^{-1}$ day$^{-1}$</td>
<td>Nrphys</td>
<td>U-Momentum Changes due to Turbulence</td>
</tr>
<tr>
<td>TURBV</td>
<td>m s$^{-1}$ day$^{-1}$</td>
<td>Nrphys</td>
<td>V-Momentum Changes due to Turbulence</td>
</tr>
<tr>
<td>TURBT</td>
<td>deg day$^{-1}$</td>
<td>Nrphys</td>
<td>Temperature Changes due to Turbulence</td>
</tr>
<tr>
<td>TURBQ</td>
<td>g/kg/day</td>
<td>Nrphys</td>
<td>Specific Humidity Changes due to Turbulence</td>
</tr>
<tr>
<td>MOISTT</td>
<td>deg day$^{-1}$</td>
<td>Nrphys</td>
<td>Temperature Changes due to Moist Processes</td>
</tr>
<tr>
<td>MOISTQ</td>
<td>g/kg/day</td>
<td>Nrphys</td>
<td>Specific Humidity Changes due to Moist Processes</td>
</tr>
<tr>
<td>RADLW</td>
<td>deg day$^{-1}$</td>
<td>Nrphys</td>
<td>Net Longwave heating rate for each level</td>
</tr>
<tr>
<td>RADSW</td>
<td>deg day$^{-1}$</td>
<td>Nrphys</td>
<td>Net Shortwave heating rate for each level</td>
</tr>
<tr>
<td>PREACC</td>
<td>mm/day</td>
<td>1</td>
<td>Total Precipitation</td>
</tr>
<tr>
<td>PRECON</td>
<td>mm/day</td>
<td>1</td>
<td>Convective Precipitation</td>
</tr>
<tr>
<td>TUFLUX</td>
<td>N m$^{-2}$</td>
<td>Nrphys</td>
<td>Turbulent Flux of U-Momentum</td>
</tr>
<tr>
<td>TVFLUX</td>
<td>N m$^{-2}$</td>
<td>Nrphys</td>
<td>Turbulent Flux of V-Momentum</td>
</tr>
<tr>
<td>TTFLUX</td>
<td>W m$^{-2}$</td>
<td>Nrphys</td>
<td>Turbulent Flux of Sensible Heat</td>
</tr>
<tr>
<td>NAME</td>
<td>UNITS</td>
<td>LEVELS</td>
<td>DESCRIPTION</td>
</tr>
<tr>
<td>----------</td>
<td>-----------</td>
<td>--------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>TQFLUX</td>
<td>W m(^{-2})</td>
<td>Nrphys</td>
<td>Turbulent Flux of Latent Heat</td>
</tr>
<tr>
<td>CN</td>
<td>dimensionless</td>
<td>1</td>
<td>Neutral Drag Coefficient</td>
</tr>
<tr>
<td>WINDS</td>
<td>m s(^{-1})</td>
<td>1</td>
<td>Surface Wind Speed</td>
</tr>
<tr>
<td>DTSRF</td>
<td>deg</td>
<td>1</td>
<td>Air/Surface virtual temperature difference</td>
</tr>
<tr>
<td>TG</td>
<td>deg</td>
<td>1</td>
<td>Ground temperature</td>
</tr>
<tr>
<td>TS</td>
<td>deg</td>
<td>1</td>
<td>Surface air temperature (Adiabatic from lowest model layer)</td>
</tr>
<tr>
<td>DTG</td>
<td>deg</td>
<td>1</td>
<td>Ground temperature adjustment</td>
</tr>
<tr>
<td>TGRLW</td>
<td>deg</td>
<td>1</td>
<td>Instantaneous ground temperature used as input to the Longwave radiation subroutine</td>
</tr>
<tr>
<td>ST4</td>
<td>W m(^{-2})</td>
<td>1</td>
<td>Upward Longwave flux at the ground ((\sigma T^4))</td>
</tr>
<tr>
<td>OLR</td>
<td>W m(^{-2})</td>
<td>1</td>
<td>Net upward Longwave flux at the top of the model</td>
</tr>
<tr>
<td>OLRCLR</td>
<td>W m(^{-2})</td>
<td>1</td>
<td>Net upward clearsky Longwave flux at the top of the model</td>
</tr>
<tr>
<td>LWGCLR</td>
<td>W m(^{-2})</td>
<td>1</td>
<td>Net upward clearsky Longwave flux at the ground</td>
</tr>
<tr>
<td>LWCLR</td>
<td>deg day(^{-1})</td>
<td>Nrphys</td>
<td>Net clearsky Longwave heating rate for each level</td>
</tr>
<tr>
<td>TLW</td>
<td>deg</td>
<td>Nrphys</td>
<td>Instantaneous temperature used as input to the Longwave radiation subroutine</td>
</tr>
<tr>
<td>SHLW</td>
<td>g g(^{-1})</td>
<td>Nrphys</td>
<td>Instantaneous specific humidity used as input to the Longwave radiation subroutine</td>
</tr>
<tr>
<td>OZLW</td>
<td>g g(^{-1})</td>
<td>Nrphys</td>
<td>Instantaneous ozone used as input to the Longwave radiation subroutine</td>
</tr>
<tr>
<td>CLMOLW</td>
<td>0 − 1</td>
<td>Nrphys</td>
<td>Maximum overlap cloud fraction used in the Longwave radiation subroutine</td>
</tr>
<tr>
<td>CLDTOT</td>
<td>0 − 1</td>
<td>Nrphys</td>
<td>Total cloud fraction used in the Longwave and Shortwave radiation subroutines</td>
</tr>
<tr>
<td>LGWDOWN</td>
<td>W m(^{-2})</td>
<td>1</td>
<td>Downwelling Longwave radiation at the ground</td>
</tr>
<tr>
<td>QWDT</td>
<td>deg day(^{-1})</td>
<td>Nrphys</td>
<td>Temperature tendency due to Gravity Wave Drag</td>
</tr>
<tr>
<td>RADSWT</td>
<td>W m(^{-2})</td>
<td>1</td>
<td>Incident Shortwave radiation at the top of the atmosphere</td>
</tr>
<tr>
<td>TAUCLD</td>
<td>per 100 mb</td>
<td>Nrphys</td>
<td>Counted Cloud Optical Depth (non-dimensional) per 100 mb</td>
</tr>
<tr>
<td>TAUCLDC</td>
<td>Number</td>
<td>Nrphys</td>
<td>Cloud Optical Depth Counter</td>
</tr>
<tr>
<td>NAME</td>
<td>UNITS</td>
<td>LEVELS</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>---------</td>
<td>--------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>CLDLOW</td>
<td>0-1</td>
<td>Nrphys</td>
<td>Low-Level (1000-700 hPa) Cloud Fraction (0-1)</td>
</tr>
<tr>
<td>EVAP</td>
<td>mm/day</td>
<td>1</td>
<td>Surface evaporation</td>
</tr>
<tr>
<td>DPDT</td>
<td>hPa/day</td>
<td>1</td>
<td>Surface Pressure time-tendency</td>
</tr>
<tr>
<td>UAVE</td>
<td>m/sec</td>
<td>Nrphys</td>
<td>Average U-Wind</td>
</tr>
<tr>
<td>VAVE</td>
<td>m/sec</td>
<td>Nrphys</td>
<td>Average V-Wind</td>
</tr>
<tr>
<td>TAVE</td>
<td>deg</td>
<td>Nrphys</td>
<td>Average Temperature</td>
</tr>
<tr>
<td>QAVE</td>
<td>g/kg</td>
<td>Nrphys</td>
<td>Average Specific Humidity</td>
</tr>
<tr>
<td>OMEGA</td>
<td>hPa/day</td>
<td>Nrphys</td>
<td>Vertical Velocity</td>
</tr>
<tr>
<td>DUDT</td>
<td>m/sec/day</td>
<td>Nrphys</td>
<td>Total U-Wind tendency</td>
</tr>
<tr>
<td>DVDT</td>
<td>m/sec/day</td>
<td>Nrphys</td>
<td>Total V-Wind tendency</td>
</tr>
<tr>
<td>DTDT</td>
<td>deg/day</td>
<td>Nrphys</td>
<td>Total Temperature tendency</td>
</tr>
<tr>
<td>DQDT</td>
<td>g/kg/day</td>
<td>Nrphys</td>
<td>Total Specific Humidity tendency</td>
</tr>
<tr>
<td>VORT</td>
<td>$10^3[-4]/sec$</td>
<td>Nrphys</td>
<td>Relative Vorticity</td>
</tr>
<tr>
<td>DTLS</td>
<td>deg/day</td>
<td>Nrphys</td>
<td>Temperature tendency due to Stratiform Cloud Formation</td>
</tr>
<tr>
<td>DQLS</td>
<td>g/kg/day</td>
<td>Nrphys</td>
<td>Specific Humidity tendency due to Stratiform Cloud Formation</td>
</tr>
<tr>
<td>USTAR</td>
<td>m/sec</td>
<td>1</td>
<td>Surface USTAR wind</td>
</tr>
<tr>
<td>Z0</td>
<td>m</td>
<td>1</td>
<td>Surface roughness</td>
</tr>
<tr>
<td>FRQTRB</td>
<td>0-1</td>
<td>Nrphys-1</td>
<td>Frequency of Turbulence</td>
</tr>
<tr>
<td>PBL</td>
<td>mb</td>
<td>1</td>
<td>Planetary Boundary Layer depth</td>
</tr>
<tr>
<td>SWCLR</td>
<td>deg/day</td>
<td>Nrphys</td>
<td>Net clearsky Shortwave heating rate for each level</td>
</tr>
<tr>
<td>OSR</td>
<td>W m$^{-2}$</td>
<td>1</td>
<td>Net downward Shortwave flux at the top of the model</td>
</tr>
<tr>
<td>OSRCLR</td>
<td>W m$^{-2}$</td>
<td>1</td>
<td>Net downward clearsky Shortwave flux at the top of the model</td>
</tr>
<tr>
<td>CLDMAS</td>
<td>kg / m$^2$</td>
<td>Nrphys</td>
<td>Convective cloud mass flux</td>
</tr>
<tr>
<td>UAVE</td>
<td>m/sec</td>
<td>Nrphys</td>
<td>Time-averaged U-Wind</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>NAME</th>
<th>UNITS</th>
<th>LEVELS</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>VAVE</td>
<td>m/sec</td>
<td>Nrphys</td>
<td>Time-averaged U-Wind</td>
</tr>
<tr>
<td>TAVE</td>
<td>deg</td>
<td>Nrphys</td>
<td>Time-averaged Temperature</td>
</tr>
<tr>
<td>QAVE</td>
<td>g/g</td>
<td>Nrphys</td>
<td>Time-averaged Specific Humidity</td>
</tr>
<tr>
<td>RFT</td>
<td>deg/day</td>
<td>Nrphys</td>
<td>Temperature tendency due Rayleigh Friction</td>
</tr>
<tr>
<td>PS</td>
<td>mb</td>
<td>1</td>
<td>Surface Pressure</td>
</tr>
<tr>
<td>QQAVE</td>
<td>(m/sec)$^2$</td>
<td>Nrphys</td>
<td>Time-averaged Turbulent Kinetic Energy</td>
</tr>
<tr>
<td>SWGCLR</td>
<td>W m$^{-2}$</td>
<td>1</td>
<td>Net downward clearsky Shortwave flux at the ground</td>
</tr>
<tr>
<td>PAVE</td>
<td>mb</td>
<td>1</td>
<td>Time-averaged Surface Pressure</td>
</tr>
<tr>
<td>DIABU</td>
<td>m/sec/day</td>
<td>Nrphys</td>
<td>Total Diabatic forcing on U-Wind</td>
</tr>
<tr>
<td>DIABV</td>
<td>m/sec/day</td>
<td>Nrphys</td>
<td>Total Diabatic forcing on V-Wind</td>
</tr>
<tr>
<td>DIABT</td>
<td>deg/day</td>
<td>Nrphys</td>
<td>Total Diabatic forcing on Temperature</td>
</tr>
<tr>
<td>DIAQQ</td>
<td>g/kg/day</td>
<td>Nrphys</td>
<td>Total Diabatic forcing on Specific Humidity</td>
</tr>
<tr>
<td>RFU</td>
<td>m/sec/day</td>
<td>Nrphys</td>
<td>U-Wind tendency due to Rayleigh Friction</td>
</tr>
<tr>
<td>RFV</td>
<td>m/sec/day</td>
<td>Nrphys</td>
<td>V-Wind tendency due to Rayleigh Friction</td>
</tr>
<tr>
<td>GWDU</td>
<td>m/sec/day</td>
<td>Nrphys</td>
<td>U-Wind tendency due to Gravity Wave Drag</td>
</tr>
<tr>
<td>GWDUS</td>
<td>N m$^{-2}$</td>
<td>1</td>
<td>U-Wind Gravity Wave Drag Stress at Surface</td>
</tr>
<tr>
<td>GWDVS</td>
<td>N m$^{-2}$</td>
<td>1</td>
<td>V-Wind Gravity Wave Drag Stress at Surface</td>
</tr>
<tr>
<td>GWDUT</td>
<td>N m$^{-2}$</td>
<td>1</td>
<td>U-Wind Gravity Wave Drag Stress at Top</td>
</tr>
<tr>
<td>GWDVT</td>
<td>N m$^{-2}$</td>
<td>1</td>
<td>V-Wind Gravity Wave Drag Stress at Top</td>
</tr>
<tr>
<td>LZRAD</td>
<td>mg/kg</td>
<td>Nrphys</td>
<td>Estimated Cloud Liquid Water used in Radiation</td>
</tr>
</tbody>
</table>
### NAME | UNITS | LEVELS | DESCRIPTION
--- | --- | --- | ---
SLP | mb | 1 | Time-averaged Sea-level Pressure
CLDFRC | 0-1 | 1 | Total Cloud Fraction
TPW | gm cm⁻² | 1 | Precipitable water
U2M | m/sec | 1 | U-Wind at 2 meters
V2M | m/sec | 1 | V-Wind at 2 meters
T2M | deg | 1 | Temperature at 2 meters
Q2M | g/kg | 1 | Specific Humidity at 2 meters
U10M | m/sec | 1 | U-Wind at 10 meters
V10M | m/sec | 1 | V-Wind at 10 meters
T10M | deg | 1 | Temperature at 10 meters
Q10M | g/kg | 1 | Specific Humidity at 10 meters
DTRAIN | kg m⁻² | Nrphys | Detrainment Cloud Mass Flux
QFILL | g/kg/day | Nrphys | Filling of negative specific humidity
DTCONV | deg/sec | Nr | Temp Change due to Convection
DQCONV | g/kg/sec | Nr | Specific Humidity Change due to Convection
RELHUM | percent | Nr | Relative Humidity
PRECLS | g/m²/sec | 1 | Large Scale Precipitation
ENPREC | J/g | 1 | Energy of Precipitation (snow, rain Temp)

### 8.5.3.4 Fizhi Diagnostic Description

In this section we list and describe the diagnostic quantities available within the GCM. The diagnostics are listed in the order that they appear in the Diagnostic Menu, Section [sec:pkg:fizhi:diagnostics]. In all cases, each diagnostic as currently archived on the output datasets is time-averaged over its diagnostic output frequency:

\[
\text{DIAGNOSTIC} = \frac{1}{TTOT} \sum_{t=1}^{TTOT} \text{diag}(t)
\]

where \( TTOT = \frac{\text{NQDIAG}}{\Delta t} \), \( \text{NQDIAG} \) is the output frequency of the diagnostic, and \( \Delta t \) is the timestep over which the diagnostic is updated.

### Surface Zonal Wind Stress on the Atmosphere (\textit{Newton/m}²)

The zonal wind stress is the turbulent flux of zonal momentum from the surface.

\[
\text{UFLUX} = -\rho C_D W_s u \quad \text{where:} \quad C_D = C_u^2
\]

where \( \rho \) = the atmospheric density at the surface, \( C_D \) is the surface drag coefficient, \( C_u \) is the dimensionless surface exchange coefficient for momentum (see diagnostic number 10), \( W_s \) is the magnitude of the surface layer wind, and \( u \) is the zonal wind in the lowest model layer.
Surface Meridional Wind Stress on the Atmosphere \((\text{Newton/m}^2)\)

The meridional wind stress is the turbulent flux of meridional momentum from the surface.

\[
\text{VFLUX} = -\rho C_D W_s v \quad \text{where} \quad C_D = C_u^2
\]

where \(\rho\) is the atmospheric density at the surface, \(C_D\) is the surface drag coefficient, \(C_u\) is the dimensionless surface exchange coefficient for momentum (see diagnostic number 10), \(W_s\) is the magnitude of the surface layer wind, and \(v\) is the meridional wind in the lowest model layer.

Surface Flux of Sensible Heat \((\text{W m}^{-2})\)

The turbulent flux of sensible heat from the surface to the atmosphere is a function of the gradient of virtual potential temperature and the eddy exchange coefficient:

\[
\text{HFLUX} = \rho c_p C_H W_s (\theta_{\text{surface}} - \theta_{\text{Nephy}}) \quad \text{where} \quad C_H = C_u C_t
\]

where \(\rho\) is the atmospheric density at the surface, \(c_p\) is the specific heat of air, \(C_H\) is the dimensionless surface heat transfer coefficient, \(W_s\) is the magnitude of the surface layer wind, \(C_u\) is the dimensionless surface exchange coefficient for momentum (see diagnostic number 10), \(C_t\) is the dimensionless surface exchange coefficient for heat and moisture (see diagnostic number 9), and \(\theta\) is the potential temperature at the surface and at the bottom model level.

Surface Flux of Latent Heat \((\text{Watts/m}^2)\)

The turbulent flux of latent heat from the surface to the atmosphere is a function of the gradient of moisture, the potential evapotranspiration fraction and the eddy exchange coefficient:

\[
\text{EFLUX} = \rho \beta L C_H W_s (q_{\text{surface}} - q_{\text{Nephy}}) \quad \text{where} \quad C_H = C_u C_t
\]

where \(\rho\) is the atmospheric density at the surface, \(\beta\) is the fraction of the potential evapotranspiration actually evaporated, \(L\) is the latent heat of evaporation, \(C_H\) is the dimensionless surface heat transfer coefficient, \(W_s\) is the magnitude of the surface layer wind, \(C_u\) is the dimensionless surface exchange coefficient for momentum (see diagnostic number 10), \(C_t\) is the dimensionless surface exchange coefficient for heat and moisture (see diagnostic number 9), and \(q_{\text{surface}}\) and \(q_{\text{Nephy}}\) are the specific humidity at the surface and at the bottom model level, respectively.

Heat Conduction Through Sea Ice \((\text{Watts/m}^2)\)

Over sea ice there is an additional source of energy at the surface due to the heat conduction from the relatively warm ocean through the sea ice. The heat conduction through sea ice represents an additional energy source term for the ground temperature equation.

\[
\text{QICE} = \frac{C_{ti}}{H_i} (T_i - T_g)
\]

where \(C_{ti}\) is the thermal conductivity of ice, \(H_i\) is the ice thickness, assumed to be 3 m where sea ice is present, \(T_i\) is 273 degrees Kelvin, and \(T_g\) is the temperature of the sea ice.

NOTE: QICE is not available through model version 5.3, but is available in subsequent versions.
Net upward Longwave Flux at the surface ($Watts/m^2$)

\[
\text{RADLWG} = F_{\text{LW,Nrphys+1}}^{\text{Net}} - F_{\text{LW,Nrphys+1}}^{\downarrow}
\]

where Nrphys+1 indicates the lowest model edge-level, or \( p = p_{\text{srf}} \). \( F_{\text{LW}}^{\uparrow} \) is the upward Longwave flux and \( F_{\text{LW}}^{\downarrow} \) is the downward Longwave flux.

Net downward shortwave Flux at the surface ($Watts/m^2$)

\[
\text{RADSWG} = F_{\text{SW,Nrphys+1}}^{\text{Net}} - F_{\text{SW,Nrphys+1}}^{\uparrow}
\]

where Nrphys+1 indicates the lowest model edge-level, or \( p = p_{\text{srf}} \). \( F_{\text{SW}}^{\downarrow} \) is the downward Shortwave flux and \( F_{\text{SW}}^{\uparrow} \) is the upward Shortwave flux.

Richardson number (dimensionless)

The non-dimensional stability indicator is the ratio of the buoyancy to the shear:

\[
\text{RI} = \frac{\frac{\partial \theta}{\partial z}}{\left(\frac{\partial \theta}{\partial z}\right)^2 + \left(\frac{\partial u}{\partial z}\right)^2} = \frac{\frac{\partial \theta}{\partial z}}{\left(\frac{\partial \theta}{\partial z}\right)^2 + \left(\frac{\partial u}{\partial z}\right)^2}
\]

where we used the hydrostatic equation:

\[
\frac{\partial \phi}{\partial \bar{p}} = c_p \frac{\partial \theta}{\partial z}
\]

Negative values indicate unstable buoyancy AND shear, small positive values (< 0.4) indicate dominantly unstable shear, and large positive values indicate dominantly stable stratification.

**CT - Surface Exchange Coefficient for Temperature and Moisture (dimensionless)**

The surface exchange coefficient is obtained from the similarity functions for the stability dependant flux profile relationships:

\[
\text{CT} = -\frac{(w'\theta')}{u_* \Delta \theta} = -\frac{(w'q')}{u_* \Delta q} = \frac{k}{(\psi_h + \psi_g)}
\]

where \( \psi_h \) is the surface layer non-dimensional temperature change and \( \psi_g \) is the viscous sublayer non-dimensional temperature or moisture change:

\[
\psi_h = \int_{\zeta_0}^{\zeta} \frac{\phi_h}{\zeta} \, d\zeta \quad \text{and} \quad \psi_g = \frac{0.55(P_r^{2/3} - 0.2)}{\nu^{1/2}} (h_0 u_* - h_{0,ref} u_{*,ref})^{1/2}
\]

and: \( h_0 = 30 z_0 \) with a maximum value over land of 0.01

\( \phi_h \) is the similarity function of \( \zeta \), which expresses the stability dependance of the temperature and moisture gradients, specified differently for stable and unstable layers according to \( \zeta \). \( k \) is the Von Karman constant, \( \zeta \) is the non-dimensional stability parameter, \( Pr \) is the Prandtl number for air, \( \nu \) is the molecular viscosity, \( z_0 \) is the surface roughness length, \( u_* \) is the surface stress velocity (see diagnostic number 67), and the subscript ref refers to a reference value.
CU - Surface Exchange Coefficient for Momentum (dimensionless)

The surface exchange coefficient is obtained from the similarity functions for the stability dependant flux profile relationships:

\[ CU = \frac{u_*}{W_s} = \frac{k}{\psi_m} \]

where \( \psi_m \) is the surface layer non-dimensional wind shear:

\[ \psi_m = \int_0^\zeta \frac{\phi_m}{\zeta} d\zeta \]

\( \phi_m \) is the similarity function of \( \zeta \), which expresses the stability dependance of the temperature and moisture gradients, specified differently for stable and unstable layers according to. \( k \) is the Von Karman constant, \( \zeta \) is the non-dimensional stability parameter, \( u_* \) is the surface stress velocity (see diagnostic number 67), and \( W_s \) is the magnitude of the surface layer wind.

ET - Diffusivity Coefficient for Temperature and Moisture (m^2/sec)

In the level 2.5 version of the Mellor-Yamada (1974) hierarchy, the turbulent heat or moisture flux for the atmosphere above the surface layer can be expressed as a turbulent diffusion coefficient \( K_h \) times the negative of the gradient of potential temperature or moisture. In the [HL88] adaptation of this closure, \( K_h \) takes the form:

\[ ET = K_h = -q \frac{\partial T}{\partial z} = \left\{ \begin{array}{ll} q \ell S_H(G_M, G_H) & \text{for decaying turbulence} \\ \frac{q_e}{q_e} \ell S_H(G_M, G_H_e) & \text{for growing turbulence} \end{array} \right. \]

where \( q \) is the turbulent velocity, or \( \sqrt{2 \times \text{turbulent kinetic energy}} \), \( q_e \) is the turbulence velocity derived from the more simple level 2.0 model, which describes equilibrium turbulence, \( \ell \) is the master length scale related to the layer depth, \( S_H \) is a function of \( G_H \) and \( G_M \), the dimensionless buoyancy and wind shear parameters, respectively, or a function of \( G_{H_e} \) and \( G_{M_e} \), the equilibrium dimensionless buoyancy and wind shear parameters. Both \( G_H \) and \( G_M \), and their equilibrium values \( G_{H_e} \) and \( G_{M_e} \), are functions of the Richardson number.

For the detailed equations and derivations of the modified level 2.5 closure scheme, see [HL88].

In the surface layer, \( ET \) is the exchange coefficient for heat and moisture, in units of \( m/sec \), given by:

\[ ET_{\text{NrPhys}} = C_t u_* = C_H W_s \]

where \( C_t \) is the dimensionless exchange coefficient for heat and moisture from the surface layer similarity functions (see diagnostic number 9), \( u_* \) is the surface friction velocity (see diagnostic number 67), \( C_H \) is the heat transfer coefficient, and \( W_s \) is the magnitude of the surface layer wind.

EU - Diffusivity Coefficient for Momentum (m^2/sec)

In the level 2.5 version of the Mellor-Yamada (1974) hierarchy, the turbulent heat momentum flux for the atmosphere above the surface layer can be expressed as a turbulent diffusion coefficient \( K_m \) times the negative of the gradient of the u-wind. In the [HL88] adaptation of this closure, \( K_m \) takes the form:

\[ EU = K_m = -\left( \frac{u'u'}{\partial v} \right) = \left\{ \begin{array}{ll} q \ell S_M(G_M, G_H) & \text{for decaying turbulence} \\ \frac{q_e}{q_e} \ell S_M(G_{M_e}, G_{H_e}) & \text{for growing turbulence} \end{array} \right. \]

where \( q \) is the turbulent velocity, or \( \sqrt{2 \times \text{turbulent kinetic energy}} \), \( q_e \) is the turbulence velocity derived from the more simple level 2.0 model, which describes equilibrium turbulence, \( \ell \) is the master length scale related to the layer
depth, \( S_M \) is a function of \( G_H \) and \( G_M \), the dimensionless buoyancy and wind shear parameters, respectively, or a function of \( G_{H_e} \) and \( G_{M_e} \), the equilibrium dimensionless buoyancy and wind shear parameters. Both \( G_H \) and \( G_M \), and their equilibrium values \( G_{H_e} \) and \( G_{M_e} \), are functions of the Richardson number.

For the detailed equations and derivations of the modified level 2.5 closure scheme, see [HL88].

In the surface layer, \( EU \) is the exchange coefficient for momentum, in units of \( m/sec \), given by:

\[
EU_{Nrphys} = C_u \cdot u_s = C_D W_s
\]

where \( C_u \) is the dimensionless exchange coefficient for momentum from the surface layer similarity functions (see diagnostic number 10), \( u_s \) is the surface friction velocity (see diagnostic number 67), \( C_D \) is the surface drag coefficient, and \( W_s \) is the magnitude of the surface layer wind.

**TURBU - Zonal U-Momentum changes due to Turbulence (m/sec/day)**

The tendency of U-Momentum due to turbulence is written:

\[
TURBU = \frac{\partial u}{\partial t_{turb}} = \frac{\partial}{\partial z}(-u'w') = \frac{\partial}{\partial z}(K_m \frac{\partial u}{\partial z})
\]

The Helfand and Labraga level 2.5 scheme models the turbulent flux of u-momentum in terms of \( K_m \), and the equation has the form of a diffusion equation.

**TURBV - Meridional V-Momentum changes due to Turbulence (m/sec/day)**

The tendency of V-Momentum due to turbulence is written:

\[
TURBV = \frac{\partial v}{\partial t_{turb}} = \frac{\partial}{\partial z}(-v'w') = \frac{\partial}{\partial z}(K_m \frac{\partial v}{\partial z})
\]

The Helfand and Labraga level 2.5 scheme models the turbulent flux of v-momentum in terms of \( K_m \), and the equation has the form of a diffusion equation.

**TURBT - Temperature changes due to Turbulence (deg/day)**

The tendency of temperature due to turbulence is written:

\[
TURBT = \frac{\partial T}{\partial t} = P^e \frac{\partial \theta}{\partial t_{turb}} = P^e \frac{\partial}{\partial z}(-w'\theta') = P^e \frac{\partial}{\partial z}(K_h \frac{\partial \theta_e}{\partial z})
\]

The Helfand and Labraga level 2.5 scheme models the turbulent flux of temperature in terms of \( K_h \), and the equation has the form of a diffusion equation.

**TURBQ - Specific Humidity changes due to Turbulence (g/kg/day)**

The tendency of specific humidity due to turbulence is written:

\[
TURBQ = \frac{\partial q}{\partial t_{turb}} = \frac{\partial}{\partial z}(-w'q') = \frac{\partial}{\partial z}(K_h \frac{\partial q}{\partial z})
\]

The Helfand and Labraga level 2.5 scheme models the turbulent flux of temperature in terms of \( K_h \), and the equation has the form of a diffusion equation.
MOISTT - Temperature Changes Due to Moist Processes (deg/day)

\[
\text{MOISTT} = \left. \frac{\partial T}{\partial t} \right|_c + \left. \frac{\partial T}{\partial t} \right|_{ls}
\]

where:

\[
\left. \frac{\partial T}{\partial t} \right|_c = R \sum_i \left( \alpha \frac{m_B}{c_p} \Gamma_s \right)_i \quad \text{and} \quad \left. \frac{\partial T}{\partial t} \right|_{ls} = \frac{L}{c_p} (q^* - q)
\]

The subscript \( c \) refers to convective processes, while the subscript \( ls \) refers to large scale precipitation processes, or supersaturation rain. The summation refers to contributions from each cloud type called by RAS. The dry static energy is given as \( s \), the convective cloud base mass flux is given as \( m_B \), and the cloud entrainment is given as \( \eta \), which are explicitly defined in Section 8.5.3.2, the description of the convective parameterization. The fractional adjustment, or relaxation parameter, for each cloud type is given as \( \alpha \), while \( R \) is the rain re-evaporation adjustment.

MOISTQ - Specific Humidity Changes Due to Moist Processes (g/kg/day)

\[
\text{MOISTQ} = \left. \frac{\partial q}{\partial t} \right|_c + \left. \frac{\partial q}{\partial t} \right|_{ls}
\]

where:

\[
\left. \frac{\partial q}{\partial t} \right|_c = R \sum_i \left( \alpha \frac{m_B}{L} \left( \Gamma_h - \Gamma_s \right) \right)_i \quad \text{and} \quad \left. \frac{\partial q}{\partial t} \right|_{ls} = (q^* - q)
\]

and

\[
\Gamma_s = g \eta \frac{\partial s}{\partial p} \quad \text{and} \quad \Gamma_h = g \eta \frac{\partial h}{\partial p}
\]

The subscript \( c \) refers to convective processes, while the subscript \( ls \) refers to large scale precipitation processes, or supersaturation rain. The summation refers to contributions from each cloud type called by RAS. The dry static energy is given as \( s \), the moist static energy is given as \( h \), the convective cloud base mass flux is given as \( m_B \), and the cloud entrainment is given as \( \eta \), which are explicitly defined in Section 8.5.3.2, the description of the convective parameterization. The fractional adjustment, or relaxation parameter, for each cloud type is given as \( \alpha \), while \( R \) is the rain re-evaporation adjustment.

RADLW - Heating Rate due to Longwave Radiation (deg/day)

The net longwave heating rate is calculated as the vertical divergence of the net terrestrial radiative fluxes. Both the clear-sky and cloudy-sky longwave fluxes are computed within the longwave routine. The subroutine calculates the clear-sky flux, \( F_{\text{clear sky}} \), first. For a given cloud fraction, the clear line-of-sight probability \( C(p, p') \) is computed from the current level pressure \( p \) to the model top pressure, \( p' = p_{\text{top}} \), and the model surface pressure, \( p' = p_{\text{surf}} \), for the upward and downward radiative fluxes. (see Section [sec:fizhi:radcloud]). The cloudy-sky flux is then obtained as:

\[
F_{\text{LW}} = C(p, p') \cdot F_{\text{clear sky}}
\]
Finally, the net longwave heating rate is calculated as the vertical divergence of the net terrestrial radiative fluxes:

$$\frac{\partial \rho c_p T}{\partial t} = -\frac{\partial}{\partial z} F_{NET}^{LW},$$

or

$$\text{RADLW} = \frac{g}{c_p \rho} \frac{\partial}{\partial \sigma} F_{NET}^{LW}.$$

where $g$ is the acceleration due to gravity, $c_p$ is the heat capacity of air at constant pressure, and

$$F_{NET}^{LW} = F_{LW}^{\uparrow} - F_{LW}^{\downarrow}.$$

### RADSW - Heating Rate due to Shortwave Radiation (deg/day)

The net Shortwave heating rate is calculated as the vertical divergence of the net solar radiative fluxes. The clear-sky and cloudy-sky shortwave fluxes are calculated separately. For the clear-sky case, the shortwave fluxes and heating rates are computed with both CLMO (maximum overlap cloud fraction) and CLRO (random overlap cloud fraction) set to zero (see Section [sec:fizhi:radcloud]). The shortwave routine is then called a second time, for the cloudy-sky case, with the true time-averaged cloud fractions CLMO and CLRO being used. In all cases, a normalized incident shortwave flux is used as input at the top of the atmosphere.

The heating rate due to Shortwave Radiation under cloudy skies is defined as:

$$\frac{\partial \rho c_p T}{\partial t} = -\frac{\partial}{\partial z} F(\text{cloudy})_{SW}^{NET} \cdot \text{RADSWT},$$

or

$$\text{RADSW} = \frac{g}{c_p \rho} \frac{\partial}{\partial \sigma} F(\text{cloudy})_{SW}^{NET} \cdot \text{RADSWT}.$$

where $g$ is the acceleration due to gravity, $c_p$ is the heat capacity of air at constant pressure, RADSWT is the true incident shortwave radiation at the top of the atmosphere (See Diagnostic #48), and

$$F(\text{cloudy})_{SW}^{NET} = F(\text{cloudy})_{SW}^{\uparrow} - F(\text{cloudy})_{SW}^{\downarrow}.$$

### PREACC - Total (Large-scale + Convective) Accumulated Precipitation (mm/day)

For a change in specific humidity due to moist processes, $\Delta q_{moist}$, the vertical integral or total precipitable amount is given by:

$$\text{PREACC} = \int_{\text{surf}}^{\text{top}} \rho \Delta q_{moist} \, dz = -\int_{\text{surf}}^{\text{top}} \Delta q_{moist} \, \frac{dp}{g} = \frac{1}{g} \int_{0}^{1} \Delta q_{moist} \, dp$$

A precipitation rate is defined as the vertically integrated moisture adjustment per Moist Processes time step, scaled to mm/day.

### PRECON - Convective Precipition (mm/day)

For a change in specific humidity due to sub-grid scale cumulus convective processes, $\Delta q_{cum}$, the vertical integral or total precipitable amount is given by:

$$\text{PRECON} = \int_{\text{surf}}^{\text{top}} \rho \Delta q_{cum} \, dz = -\int_{\text{surf}}^{\text{top}} \Delta q_{cum} \, \frac{dp}{g} = \frac{1}{g} \int_{0}^{1} \Delta q_{cum} \, dp$$

A precipitation rate is defined as the vertically integrated moisture adjustment per Moist Processes time step, scaled to mm/day.
TUFLUX - Turbulent Flux of U-Momentum (Newton/m^2)

The turbulent flux of u-momentum is calculated for *diagnostic purposes only* from the eddy coefficient for momentum:

\[ \text{TUFLUX} = \rho (u'w') = \rho (-K_m \frac{\partial U}{\partial z}) \]

where \( \rho \) is the air density, and \( K_m \) is the eddy coefficient.

TVFLUX - Turbulent Flux of V-Momentum (Newton/m^2)

The turbulent flux of v-momentum is calculated for *diagnostic purposes only* from the eddy coefficient for momentum:

\[ \text{TVFLUX} = \rho (v'w') = \rho (-K_m \frac{\partial V}{\partial z}) \]

where \( \rho \) is the air density, and \( K_m \) is the eddy coefficient.

TTFLUX - Turbulent Flux of Sensible Heat (Watts/m^2)

The turbulent flux of sensible heat is calculated for *diagnostic purposes only* from the eddy coefficient for heat and moisture:

\[ \text{TTFLUX} = c_p \rho P^s (w'\theta') = c_p \rho P^s (-K_h \frac{\partial \theta}{\partial z}) \]

where \( \rho \) is the air density, and \( K_h \) is the eddy coefficient.

TQFLUX - Turbulent Flux of Latent Heat (Watts/m^2)

The turbulent flux of latent heat is calculated for *diagnostic purposes only* from the eddy coefficient for heat and moisture:

\[ \text{TQFLUX} = L \rho (w'q') = L \rho (-K_h \frac{\partial q}{\partial z}) \]

where \( \rho \) is the air density, and \( K_h \) is the eddy coefficient.

CN - Neutral Drag Coefficient (dimensionless)

The drag coefficient for momentum obtained by assuming a neutrally stable surface layer:

\[ \text{CN} = \frac{k}{\ln \left( \frac{h}{z_0} \right)} \]

where \( k \) is the Von Karman constant, \( h \) is the height of the surface layer, and \( z_0 \) is the surface roughness.

NOTE: CN is not available through model version 5.3, but is available in subsequent versions.
WINDS - Surface Wind Speed (meter/sec)

The surface wind speed is calculated for the last internal turbulence time step:

\[
\text{WINDS} = \sqrt{u_N^{2} + v_N^{2}}
\]

where the subscript \( N_{rphys} \) refers to the lowest model level.

The air/surface virtual temperature difference measures the stability of the surface layer:

\[
\text{DTSRF} = (\theta_{vN_{rphys+1}} - \theta v_{N_{rphys}}) P_{surf}^{s}
\]

where

\[
\theta_{vN_{rphys+1}} = T_g P_{surf}^{s} (1 + 0.609 q_{N_{rphys+1}})
\]

and

\[
q_{N_{rphys+1}} = q_{N_{rphys}} + \beta (q^*(T_g, P_s) - q_{N_{rphys}})
\]

\( \beta \) is the surface potential evapotranspiration coefficient (\( \beta = 1 \) over oceans), \( q^*(T_g, P_s) \) is the saturation specific humidity at the ground temperature and surface pressure, level \( N_{rphys} \) refers to the lowest model level and level \( N_{rphys} + 1 \) refers to the surface.

TG - Ground Temperature (deg K)

The ground temperature equation is solved as part of the turbulence package using a backward implicit time differencing scheme:

\[
\text{TG is obtained from: } C_g \frac{\partial T_g}{\partial t} = R_{sw} - R_{lw} + Q_{ice} - H - LE
\]

where \( R_{sw} \) is the net surface downward shortwave radiative flux, \( R_{lw} \) is the net surface upward longwave radiative flux, \( Q_{ice} \) is the heat conduction through sea ice, \( H \) is the upward sensible heat flux, \( LE \) is the upward latent heat flux, and \( C_g \) is the total heat capacity of the ground. \( C_g \) is obtained by solving a heat diffusion equation for the penetration of the diurnal cycle into the ground (), and is given by:

\[
C_g = \sqrt{\frac{\lambda C_s}{2\omega}} = \sqrt{\frac{(0.386 + 0.536 W + 0.15 W^2)2 \times 10^{-3} 86400}{2\pi}} .
\]

Here, the thermal conductivity, \( \lambda \), is equal to \( 2 \times 10^{-3} \frac{\text{J m cm}}{\text{sec K}} \), the angular velocity of the earth, \( \omega \), is written as \( 86400 \text{ sec/day} \) divided by \( 2\pi \text{ radians/day} \), and the expression for \( C_s \), the heat capacity per unit volume at the surface, is a function of the ground wetness, \( W \).

TS - Surface Temperature (deg K)

The surface temperature estimate is made by assuming that the model’s lowest layer is well-mixed, and therefore that \( \theta \) is constant in that layer. The surface temperature is therefore:

\[
\text{TS} = \theta_{N_{rphys}} P_{surf}^{s}
\]
**DTG - Surface Temperature Adjustment (deg K)**

The change in surface temperature from one turbulence time step to the next, solved using the Ground Temperature Equation (see diagnostic number 30) is calculated:

\[ DTG = T_g^n - T_g^{n-1} \]

where superscript \( n \) refers to the new, updated time level, and the superscript \( n - 1 \) refers to the value at the previous turbulence time level.

**QG - Ground Specific Humidity (g/kg)**

The ground specific humidity is obtained by interpolating between the specific humidity at the lowest model level and the specific humidity of a saturated ground. The interpolation is performed using the potential evapotranspiration function:

\[ QG = q_{Nphys+1} = q_{Nphys} + \beta(q^*(T_g, P_s) - q_{Nphys}) \]

where \( \beta \) is the surface potential evapotranspiration coefficient (\( \beta = 1 \) over oceans), and \( q^*(T_g, P_s) \) is the saturation specific humidity at the ground temperature and surface pressure.

**QS - Saturation Surface Specific Humidity (g/kg)**

The surface saturation specific humidity is the saturation specific humidity at the ground temperature and surface pressure:

\[ QS = q^*(T_g, P_s) \]

**TGRLW - Instantaneous ground temperature used as input to the Longwave radiation subroutine (deg)**

\[ TGRLW = T_g(\lambda, \phi, n) \]

where \( T_g \) is the model ground temperature at the current time step \( n \).

**ST4 - Upward Longwave flux at the surface (Watts/m^2)**

\[ ST4 = \sigma T^4 \]

where \( \sigma \) is the Stefan-Boltzmann constant and \( T \) is the temperature.
OLR - Net upward Longwave flux at \( p = p_{\text{top}} \) (Watts/m\(^2\))

\[
\text{OLR} = F_{\text{NET}}^{\text{LW,top}}
\]

where top indicates the top of the first model layer. In the GCM, \( p_{\text{top}} = 0.0 \text{ mb} \).

OLRCLR - Net upward clearsky Longwave flux at \( p = p_{\text{top}} \) (Watts/m\(^2\))

\[
\text{OLRCLR} = F_{\text{NET}}^{\text{LW,top}}(\text{clearsky})
\]

where top indicates the top of the first model layer. In the GCM, \( p_{\text{top}} = 0.0 \text{ mb} \).

LWGCLR - Net upward clearsky Longwave flux at the surface (Watts/m\(^2\))

\[
\text{LWGCLR} = F_{\text{NET}}^{\text{LW, Nrphys+1}}(\text{clearsky}) \uparrow - F_{\text{NET}}^{\text{LW, Nrphys+1}}(\text{clearsky}) \downarrow
\]

where \( \text{Nrphys+1} \) indicates the lowest model edge-level, or \( p = p_{\text{surf}} \). \( F_{\text{NET}}^{\text{LW, Nrphys+1}}(\text{clearsky}) \uparrow \) is the upward clearsky Longwave flux and \( F_{\text{NET}}^{\text{LW, Nrphys+1}}(\text{clearsky}) \downarrow \) is the downward clearsky Longwave flux.

LWCLR - Heating Rate due to Clearsky Longwave Radiation (deg/day)

The net longwave heating rate is calculated as the vertical divergence of the net terrestrial radiative fluxes. Both the clear-sky and cloudy-sky longwave fluxes are computed within the longwave routine. The subroutine calculates the clear-sky flux, \( F_{\text{LW}}^{\text{clearsky}} \), first. For a given cloud fraction, the clear line-of-sight probability \( C(p, p') \) is computed from the current level pressure \( p \) to the model top pressure, \( p' = p_{\text{top}} \), and the model surface pressure, \( p' = p_{\text{surf}} \), for the upward and downward radiative fluxes. (see Section [sec:fizhi:radcloud]). The cloudy-sky flux is then obtained as:

\[
F_{\text{LW}} = C(p, p') \cdot F_{\text{LW}}^{\text{clearsky}},
\]

Thus, LWCLR is defined as the net longwave heating rate due to the vertical divergence of the clear-sky longwave radiative flux:

\[
\frac{\partial p c_p T}{\partial t}^{\text{clearsky}} = - \frac{\partial}{\partial z} F_{\text{LW}}^{\text{NET}}(\text{clearsky}),
\]

or

\[
\text{LWCLR} = g \frac{\partial}{\partial \sigma} F_{\text{LW}}^{\text{NET}}(\text{clearsky}).
\]

where \( g \) is the acceleration due to gravity, \( c_p \) is the heat capacity of air at constant pressure, and

\[
F_{\text{LW}}^{\text{NET}}(\text{clearsky}) = F_{\text{LW}}^{\uparrow}(\text{clearsky}) - F_{\text{LW}}^{\downarrow}(\text{clearsky})
\]
TLW - Instantaneous temperature used as input to the Longwave radiation subroutine (deg)

\[ TLW = T(\lambda, \phi, level, n) \]

where \( T \) is the model temperature at the current time step \( n \).

SHLW - Instantaneous specific humidity used as input to the Longwave radiation subroutine (kg/kg)

\[ SHLW = q(\lambda, \phi, level, n) \]

where \( q \) is the model specific humidity at the current time step \( n \).

OZLW - Instantaneous ozone used as input to the Longwave radiation subroutine (kg/kg)

\[ OZLW = OZ(\lambda, \phi, level, n) \]

where \( OZ \) is the interpolated ozone data set from the climatological monthly mean zonally averaged ozone data set.

CLMOLW - Maximum Overlap cloud fraction used in LW Radiation (0-1)

CLMOLW is the time-averaged maximum overlap cloud fraction that has been filled by the Relaxed Arakawa/Schubert Convection scheme and will be used in the Longwave Radiation algorithm. These are convective clouds whose radiative characteristics are assumed to be correlated in the vertical. For a complete description of cloud/radiative interactions, see Section [sec:fizhi:radcloud].

\[ CLMOLW = CLMO_{RAS,LW}(\lambda, \phi, level) \]

CLDTOT - Total cloud fraction used in LW and SW Radiation (0-1)

CLDTOT is the time-averaged total cloud fraction that has been filled by the Relaxed Arakawa/Schubert and Large-scale Convection schemes and will be used in the Longwave and Shortwave Radiation packages. For a complete description of cloud/radiative interactions, see Section [sec:fizhi:radcloud].

\[ CLDTOT = F_{RAS} + F_{LS} \]

where \( F_{RAS} \) is the time-averaged cloud fraction due to sub-grid scale convection, and \( F_{LS} \) is the time-averaged cloud fraction due to precipitating and non-precipitating large-scale moist processes.
CLMOSW - Maximum Overlap cloud fraction used in SW Radiation (0-1)

**CLMOSW** is the time-averaged maximum overlap cloud fraction that has been filled by the Relaxed Arakawa/Schubert Convection scheme and will be used in the Shortwave Radiation algorithm. These are convective clouds whose radiative characteristics are assumed to be correlated in the vertical. For a complete description of cloud/radiative interactions, see Section [sec:fizhi:radcloud].

\[
\text{CLMOSW} = CLM_{RAS,SW}(\lambda, \phi, \text{level})
\]

CLROSW - Random Overlap cloud fraction used in SW Radiation (0-1)

**CLROSW** is the time-averaged random overlap cloud fraction that has been filled by the Relaxed Arakawa/Schubert and Large-scale Convection schemes and will be used in the Shortwave Radiation algorithm. These are convective and large-scale clouds whose radiative characteristics are not assumed to be correlated in the vertical. For a complete description of cloud/radiative interactions, see Section [sec:fizhi:radcloud].

\[
\text{CLROSW} = CLR_{RAS,LS,SW}(\lambda, \phi, \text{level})
\]

RADSWT - Incident Shortwave radiation at the top of the atmosphere (Watts/m^2)

\[
\text{RADSWT} = \frac{S_0}{R_a} \cdot \cos \phi_z
\]

where \( S_0 \) is the extra-terrestrial solar constant, \( R_a \) is the earth-sun distance in Astronomical Units, and \( \cos \phi_z \) is the cosine of the zenith angle. It should be noted that RADSWT, as well as OSR and OSRCLR, are calculated at the top of the atmosphere (p=0 mb). However, the OLR and OLRCLR diagnostics are currently calculated at \( p = p_{top} \) (0.0 mb for the GCM).

EVAP - Surface Evaporation (mm/day)

The surface evaporation is a function of the gradient of moisture, the potential evapotranspiration fraction and the eddy exchange coefficient:

\[
\text{EVAP} = \rho \beta K_h (q_{surface} - q_{Nphys})
\]

where \( \rho \) is the atmospheric density at the surface, \( \beta \) is the fraction of the potential evapotranspiration actually evaporated (\( \beta = 1 \) over oceans), \( K_h \) is the turbulent eddy exchange coefficient for heat and moisture at the surface in \( m/sec \) and \( q_{surface} \) and \( q_{Nphys} \) are the specific humidity at the surface (see diagnostic number 34) and at the bottom model level, respectively.

DUDT - Total Zonal U-Wind Tendency (m/sec/day)

**DUDT** is the total time-tendency of the Zonal U-Wind due to Hydrodynamic, Diabatic, and Analysis forcing.

\[
\text{DUDT} = \frac{\partial u}{\partial t} \text{Dynamics} + \frac{\partial u}{\partial t} \text{Moist} + \frac{\partial u}{\partial t} \text{Turbulence} + \frac{\partial u}{\partial t} \text{Analysis}
\]
DVDT - Total Zonal V-Wind Tendency (m/sec/day)

**DVDT** is the total time-tendency of the Meridional V-Wind due to Hydrodynamic, Diabatic, and Analysis forcing.

\[
\text{DVDT} = \frac{\partial v}{\partial t}^{\text{Dynamics}} + \frac{\partial v}{\partial t}^{\text{Moist}} + \frac{\partial v}{\partial t}^{\text{Turbulence}} + \frac{\partial v}{\partial t}^{\text{Analysis}}
\]

DTDT - Total Temperature Tendency (deg/day)

**DTDT** is the total time-tendency of Temperature due to Hydrodynamic, Diabatic, and Analysis forcing.

\[
\text{DTDT} = \frac{\partial T}{\partial t}^{\text{Dynamics}} + \frac{\partial T}{\partial t}^{\text{MoistProcesses}} + \frac{\partial T}{\partial t}^{\text{ShortwaveRadiation}} + \frac{\partial T}{\partial t}^{\text{LongwaveRadiation}} + \frac{\partial T}{\partial t}^{\text{Turbulence}} + \frac{\partial T}{\partial t}^{\text{Analysis}}
\]

DQDT - Total Specific Humidity Tendency (g/kg/day)

**DQDT** is the total time-tendency of Specific Humidity due to Hydrodynamic, Diabatic, and Analysis forcing.

\[
\text{DQDT} = \frac{\partial q}{\partial t}^{\text{Dynamics}} + \frac{\partial q}{\partial t}^{\text{MoistProcesses}} + \frac{\partial q}{\partial t}^{\text{Turbulence}} + \frac{\partial q}{\partial t}^{\text{Analysis}}
\]

USTAR - Surface-Stress Velocity (m/sec)

The surface stress velocity, or the friction velocity, is the wind speed at the surface layer top impeded by the surface drag:

\[
\text{USTAR} = C_u W_s \quad \text{where} \quad C_u = \frac{k}{\psi_m}
\]

\(C_u\) is the non-dimensional surface drag coefficient (see diagnostic number 10), and \(W_s\) is the surface wind speed (see diagnostic number 28).

Z0 - Surface Roughness Length (m)

Over the land surface, the surface roughness length is interpolated to the local time from the monthly mean data of \(Z0\). Over the ocean, the roughness length is a function of the surface-stress velocity, \(u_s\).

\[
Z0 = c_1 u_s^3 + c_2 u_s^2 + c_3 u_s + c_4 + c_5 u_s
\]

where the constants are chosen to interpolate between the reciprocal relation of for weak winds, and the piecewise linear relation of for moderate to large winds.
FRQTRB - Frequency of Turbulence (0-1)

The fraction of time when turbulence is present is defined as the fraction of time when the turbulent kinetic energy exceeds some minimum value, defined here to be $0.005 \text{ m}^2/\text{sec}^2$. When this criterion is met, a counter is incremented. The fraction over the averaging interval is reported.

PBL - Planetary Boundary Layer Depth (mb)

The depth of the PBL is defined by the turbulence parameterization to be the depth at which the turbulent kinetic energy reduces to ten percent of its surface value.

$$\text{PBL} = P_{PBL} - P_{surface}$$

where $P_{PBL}$ is the pressure in mb at which the turbulent kinetic energy reaches one tenth of its surface value, and $P_s$ is the surface pressure.

SWCLR - Clear sky Heating Rate due to Shortwave Radiation (deg/day)

The net Shortwave heating rate is calculated as the vertical divergence of the net solar radiative fluxes. The clear-sky and cloudy-sky shortwave fluxes are calculated separately. For the clear-sky case, the shortwave fluxes and heating rates are computed with both CLMO (maximum overlap cloud fraction) and CLRO (random overlap cloud fraction) set to zero (see Section [sec:fizhi:radcloud]). The shortwave routine is then called a second time, for the cloudy-sky case, with the true time-averaged cloud fractions CLMO and CLRO being used. In all cases, a normalized incident shortwave flux is used as input at the top of the atmosphere.

The heating rate due to Shortwave Radiation under clear skies is defined as:

$$\frac{\partial pc_p T}{\partial t} = - \frac{\partial}{\partial z} F(\text{clear})_{\text{SW}}^{N^E T} \cdot \text{RADSWT},$$

or

$$\text{SWCLR} = \frac{g}{c_p} \frac{\partial}{\partial p} F(\text{clear})_{\text{SW}}^{N^E T} \cdot \text{RADSWT},$$

where $g$ is the acceleration due to gravity, $c_p$ is the heat capacity of air at constant pressure, RADSWT is the true incident shortwave radiation at the top of the atmosphere (See Diagnostic #48), and

$$F(\text{clear})_{\text{SW}}^{N^E T} = F(\text{clear})_{\text{SW}}^{\uparrow} - F(\text{clear})_{\text{SW}}^{\downarrow}$$

OSR - Net upward Shortwave flux at the top of the model (Watts/m^2)

$$\text{OSR} = F_{\text{SW, top}}^{N^E T}$$

where top indicates the top of the first model layer used in the shortwave radiation routine. In the GCM, $p_{SW_{top}} = 0$ mb.
OSRCLR - Net upward clearsky Shortwave flux at the top of the model (Watts/m^2)

\[
\text{OSRCLR} = F(\text{clearsky})^{\text{NET}}_{\text{SW, top}}
\]

where top indicates the top of the first model layer used in the shortwave radiation routine. In the GCM, \( p_{\text{SW, top}} = 0 \) mb.

CLDMAS - Convective Cloud Mass Flux (kg/m^2)

The amount of cloud mass moved per RAS timestep from all convective clouds is written:

\[
\text{CLDMAS} = \eta m_B
\]

where \( \eta \) is the entrainment, normalized by the cloud base mass flux, and \( m_B \) is the cloud base mass flux. \( m_B \) and \( \eta \) are defined explicitly in Section 8.5.3.2, the description of the convective parameterization.

UAVE - Time-Averaged Zonal U-Wind (m/sec)

The diagnostic \( \text{UAVE} \) is simply the time-averaged Zonal U-Wind over the \( \text{NUAVE} \) output frequency. This is contrasted to the instantaneous Zonal U-Wind which is archived on the Prognostic Output data stream.

\[
\text{UAVE} = u(\lambda, \phi, level, t)
\]

Note, \( \text{UAVE} \) is computed and stored on the staggered C-grid.

VAVE - Time-Averaged Meridional V-Wind (m/sec)

The diagnostic \( \text{VAVE} \) is simply the time-averaged Meridional V-Wind over the \( \text{NVAVE} \) output frequency. This is contrasted to the instantaneous Meridional V-Wind which is archived on the Prognostic Output data stream.

\[
\text{VAVE} = v(\lambda, \phi, level, t)
\]

Note, \( \text{VAVE} \) is computed and stored on the staggered C-grid.

TAVE - Time-Averaged Temperature (Kelvin)

The diagnostic \( \text{TAVE} \) is simply the time-averaged Temperature over the \( \text{NTAVE} \) output frequency. This is contrasted to the instantaneous Temperature which is archived on the Prognostic Output data stream.

\[
\text{TAVE} = T(\lambda, \phi, level, t)
\]

QAVE - Time-Averaged Specific Humidity (g/kg)

The diagnostic \( \text{QAVE} \) is simply the time-averaged Specific Humidity over the \( \text{NQAVE} \) output frequency. This is contrasted to the instantaneous Specific Humidity which is archived on the Prognostic Output data stream.

\[
\text{QAVE} = q(\lambda, \phi, level, t)
\]
PAVE - Time-Averaged Surface Pressure - PTOP (mb)

The diagnostic PAVE is simply the time-averaged Surface Pressure - PTOP over the NPAVE output frequency. This is contrasted to the instantaneous Surface Pressure - PTOP which is archived on the Prognostic Output data stream.

\[
PAVE = \pi(\lambda, \phi, \text{level}, t) = p_s(\lambda, \phi, \text{level}, t) - p_T
\]

QQAVE - Time-Averaged Turbulent Kinetic Energy (m/sec)^2

The diagnostic QQAVE is simply the time-averaged prognostic Turbulent Kinetic Energy produced by the GCM Turbulence parameterization over the NQQAVE output frequency. This is contrasted to the instantaneous Turbulent Kinetic Energy which is archived on the Prognostic Output data stream.

\[
QQAVE = qq(\lambda, \phi, \text{level}, t)
\]

Note, QQAVE is computed and stored at the “mass-point” locations on the staggered C-grid.

SWGCLR - Net downward clearsky Shortwave flux at the surface (Watts/m^2)

\[
SWGCLR = F(clearsky)_{SW,Nrphys+1}^{Net} = F(clearsky)_{SW,Nrphys+1}^\downarrow - F(clearsky)_{SW,Nrphys+1}^\uparrow
\]

where Nrphys+1 indicates the lowest model edge-level, or \( p = p_{surf} \). \( F(clearsky)_{SW}^{\downarrow} \) is the downward clearsky Shortwave flux and \( F(clearsky)_{SW}^{\uparrow} \) is the upward clearsky Shortwave flux.

DIABU - Total Diabatic Zonal U-Wind Tendency (m/sec/day)

DIABU is the total time-tendency of the Zonal U-Wind due to Diabatic processes and the Analysis forcing.

\[
DIABU = \frac{\partial u}{\partial t}_{\text{Moist}} + \frac{\partial u}{\partial t}_{\text{Turbulence}} + \frac{\partial u}{\partial t}_{\text{Analysis}}
\]

DIABV - Total Diabatic Meridional V-Wind Tendency (m/sec/day)

DIABV is the total time-tendency of the Meridional V-Wind due to Diabatic processes and the Analysis forcing.

\[
DIABV = \frac{\partial v}{\partial t}_{\text{Moist}} + \frac{\partial v}{\partial t}_{\text{Turbulence}} + \frac{\partial v}{\partial t}_{\text{Analysis}}
\]

DIABT Total Diabatic Temperature Tendency (deg/day)

DIABT is the total time-tendency of Temperature due to Diabatic processes and the Analysis forcing.

\[
DIABT = \frac{\partial T}{\partial t}_{\text{MoistProcesses}} + \frac{\partial T}{\partial t}_{\text{ShortwaveRadiation}} + \frac{\partial T}{\partial t}_{\text{LongwaveRadiation}} + \frac{\partial T}{\partial t}_{\text{Turbulence}} + \frac{\partial T}{\partial t}_{\text{Analysis}}
\]
If we define the time-tendency of Temperature due to Diabatic processes as

$$\frac{\partial T}{\partial t}_{\text{Diabatic}} = \frac{\partial T}{\partial t}_{\text{MoistProcesses}} + \frac{\partial T}{\partial t}_{\text{ShortwaveRadiation}} + \frac{\partial T}{\partial t}_{\text{LongwaveRadiation}} + \frac{\partial T}{\partial t}_{\text{Turbulence}}$$

then, since there are no surface pressure changes due to Diabatic processes, we may write

$$\frac{\partial T}{\partial t}_{\text{Diabatic}} = \frac{p}{\pi} \frac{\partial \pi \theta}{\partial t}_{\text{Diabatic}}$$

where $\theta = T/p^\kappa$. Thus, DIABT may be written as

$$\text{DIABT} = \frac{p}{\pi} \left( \frac{\partial \pi \theta}{\partial t}_{\text{Diabatic}} + \frac{\partial \pi \theta}{\partial t}_{\text{Analysis}} \right)$$

**DIABQ - Total Diabatic Specific Humidity Tendency (g/kg/day)**

**DIABQ** is the total time-tendency of Specific Humidity due to Diabatic processes and the Analysis forcing.

$$\text{DIABQ} = \frac{\partial q}{\partial t}_{\text{MoistProcesses}} + \frac{\partial q}{\partial t}_{\text{Turbulence}} + \frac{\partial q}{\partial t}_{\text{Analysis}}$$

If we define the time-tendency of Specific Humidity due to Diabatic processes as

$$\frac{\partial q}{\partial t}_{\text{Diabatic}} = \frac{\partial q}{\partial t}_{\text{MoistProcesses}} + \frac{\partial q}{\partial t}_{\text{Turbulence}}$$

then, since there are no surface pressure changes due to Diabatic processes, we may write

$$\frac{\partial q}{\partial t}_{\text{Diabatic}} = \frac{1}{\pi} \frac{\partial \pi q}{\partial t}_{\text{Diabatic}}$$

Thus, **DIABQ** may be written as

$$\text{DIABQ} = \frac{1}{\pi} \left( \frac{\partial \pi q}{\partial t}_{\text{Diabatic}} + \frac{\partial \pi q}{\partial t}_{\text{Analysis}} \right)$$

**VINTUQ - Vertically Integrated Moisture Flux (m/sec g/kg)**

The vertically integrated moisture flux due to the zonal u-wind is obtained by integrating $uq$ over the depth of the atmosphere at each model timestep, and dividing by the total mass of the column.

$$\text{VINTUQ} = \frac{\int_{\text{surf}}^{\text{top}} uq \rho dz}{\int_{\text{surf}}^{\text{top}} \rho dz}$$

Using $\rho \delta z = -\frac{\delta p}{g} = -\frac{1}{g} \delta p$, we have

$$\text{VINTUQ} = \int_{0}^{1} uq dp$$
VINTVQ - Vertically Integrated Moisture Flux (m/sec g/kg)

The vertically integrated moisture flux due to the meridional v-wind is obtained by integrating \( vq \) over the depth of the atmosphere at each model timestep, and dividing by the total mass of the column.

\[
VINTVQ = \frac{\int_{surf}^{top} vq \rho \, dz}{\int_{surf}^{top} \rho \, dz}
\]

Using \( \rho \delta z = -\frac{\delta p}{g} = -\frac{1}{g} \delta p \), we have

\[
VINTVQ = \int_0^1 vq dp
\]

VINTUT - Vertically Integrated Heat Flux (m/sec deg)

The vertically integrated heat flux due to the zonal u-wind is obtained by integrating \( uT \) over the depth of the atmosphere at each model timestep, and dividing by the total mass of the column.

\[
VINTUT = \frac{\int_{surf}^{top} uT \rho \, dz}{\int_{surf}^{top} \rho \, dz}
\]

Or,

\[
VINTUT = \int_0^1 uT dp
\]

VINTVT - Vertically Integrated Heat Flux (m/sec deg)

The vertically integrated heat flux due to the meridional v-wind is obtained by integrating \( vT \) over the depth of the atmosphere at each model timestep, and dividing by the total mass of the column.

\[
VINTVT = \frac{\int_{surf}^{top} vT \rho \, dz}{\int_{surf}^{top} \rho \, dz}
\]

Using \( \rho \delta z = -\frac{\delta p}{g} \), we have

\[
VINTVT = \int_0^1 vT dp
\]

CLDFRC - Total 2-Dimensional Cloud Fraction (0-1)

If we define the time-averaged random and maximum overlapped cloudiness as CLRO and CLMO respectively, then the probability of clear sky associated with random overlapped clouds at any level is (1-CLRO) while the probability of clear sky associated with maximum overlapped clouds at any level is (1-CLMO). The total clear sky probability is given by (1-CLRO)*(1-CLMO), thus the total cloud fraction at each level may be obtained by 1-(1-CLRO)*(1-CLMO).

At any given level, we may define the clear line-of-site probability by appropriately accounting for the maximum and random overlap cloudiness. The clear line-of-site probability is defined to be equal to the product of the clear line-of-site probabilities associated with random and maximum overlap cloudiness. The clear line-of-site probability \( C(p, p') \)
associated with maximum overlap clouds, from the current pressure \( p \) to the model top pressure, \( p' = p_{top} \), or the model surface pressure, \( p' = p_{surf} \), is simply 1.0 minus the largest maximum overlap cloud value along the line-of-site, ie.

\[
1 - MAX_{p'}^p (CLMO_p)
\]

Thus, even in the time-averaged sense it is assumed that the maximum overlap clouds are correlated in the vertical. The clear line-of-site probability associated with random overlap clouds is defined to be the product of the clear sky probabilities at each level along the line-of-site, ie.

\[
\prod_{p} (1 - CLRO_{p})
\]

The total cloud fraction at a given level associated with a line-of-site calculation is given by

\[
1 - \left( 1 - MAX_{p'}^p [CLMO_p] \right) \prod_{p} (1 - CLRO_{p})
\]

The 2-dimensional net cloud fraction as seen from the top of the atmosphere is given by

\[
CLDFRC = 1 - \left( 1 - MAX_{l=1}^{Nrphys} [CLMO_{l}] \right) \prod_{l=1}^{Nrphys} (1 - CLRO_{l})
\]

For a complete description of cloud/radiative interactions, see Section [sec:fizhi:radcloud].

**QINT - Total Precipitable Water (gm/cm^2)**

The Total Precipitable Water is defined as the vertical integral of the specific humidity, given by:

\[
QINT = \int_{surf}^{top} \rho q dz = \frac{\pi}{g} \int_{0}^{1} q dp
\]

where we have used the hydrostatic relation \( \rho \delta z = -\frac{\delta p}{g} \).

**U2M Zonal U-Wind at 2 Meter Depth (m/sec)**

The u-wind at the 2-meter depth is determined from the similarity theory:

\[
U2M = \frac{u_s}{k} \psi_{m2m} \frac{u_{sl}}{W_s} = \frac{\psi_{m2m} u_{sl}}{\psi_{m2sl}}
\]

where \( \psi_{m}(2m) \) is the non-dimensional wind shear at two meters, and the subscript sl refers to the height of the top of the surface layer. If the roughness height is above two meters, \( U2M \) is undefined.

**V2M - Meridional V-Wind at 2 Meter Depth (m/sec)**

The v-wind at the 2-meter depth is a determined from the similarity theory:

\[
V2M = \frac{u_s}{k} \psi_{m2m} \frac{v_{sl}}{W_s} = \frac{\psi_{m2m} v_{sl}}{\psi_{m2sl}}
\]

where \( \psi_{m}(2m) \) is the non-dimensional wind shear at two meters, and the subscript sl refers to the height of the top of the surface layer. If the roughness height is above two meters, \( V2M \) is undefined.
T2M - Temperature at 2 Meter Depth (deg K)

The temperature at the 2-meter depth is determined from the similarity theory:

\[
T_{2M} = P^c \left( \frac{\theta_s}{k} (\psi_{h2m} + \psi_g) + \theta_{surf} \right) = P^c (\theta_{surf} + \frac{\psi_{h2m} + \psi_g}{\psi_{hsl} + \psi_g} (\theta_{sl} - \theta_{surf}))
\]

where:

\[
\theta_s = \frac{(w' q')}{u_*}
\]

where \(\psi_{h}(2m)\) is the non-dimensional temperature gradient at two meters, \(\psi_g\) is the non-dimensional temperature gradient in the viscous sublayer, and the subscript \(sl\) refers to the height of the top of the surface layer. If the roughness height is above two meters, \(T_{2M}\) is undefined.

Q2M - Specific Humidity at 2 Meter Depth (g/kg)

The specific humidity at the 2-meter depth is determined from the similarity theory:

\[
Q_{2M} = P^c \left( \frac{q_s}{q_*} k (\psi_{h2m} + \psi_g) + q_{surf} \right) = P^c (q_{surf} + \frac{\psi_{h2m} + \psi_g}{\psi_{hsl} + \psi_g} (q_{sl} - q_{surf}))
\]

where:

\[
q_s = \frac{(w' q')}{u_*}
\]

where \(\psi_{h}(2m)\) is the non-dimensional temperature gradient at two meters, \(\psi_g\) is the non-dimensional temperature gradient in the viscous sublayer, and the subscript \(sl\) refers to the height of the top of the surface layer. If the roughness height is above two meters, \(Q_{2M}\) is undefined.

U10M - Zonal U-Wind at 10 Meter Depth (m/sec)

The u-wind at the 10-meter depth is an interpolation between the surface wind and the model lowest level wind using the ratio of the non-dimensional wind shear at the two levels:

\[
U_{10M} = \frac{u_*}{k} \psi_{m10m} \frac{u_{sl}}{W_s} = \frac{\psi_{m10m}}{\psi_{m_{sl}}} u_{sl}
\]

where \(\psi_{m}(10m)\) is the non-dimensional wind shear at ten meters, and the subscript \(sl\) refers to the height of the top of the surface layer.

V10M - Meridional V-Wind at 10 Meter Depth (m/sec)

The v-wind at the 10-meter depth is an interpolation between the surface wind and the model lowest level wind using the ratio of the non-dimensional wind shear at the two levels:

\[
V_{10M} = \frac{v_*}{k} \psi_{m10m} \frac{v_{sl}}{W_s} = \frac{\psi_{m10m}}{\psi_{m_{sl}}} v_{sl}
\]

where \(\psi_{m}(10m)\) is the non-dimensional wind shear at ten meters, and the subscript \(sl\) refers to the height of the top of the surface layer.
T10M - Temperature at 10 Meter Depth (deg K)

The temperature at the 10-meter depth is an interpolation between the surface potential temperature and the model
lowest level potential temperature using the ratio of the non-dimensional temperature gradient at the two levels:

\[
T_{10M} = P^c \left( \frac{\theta_s}{\psi_{h10m} + \psi_g} + \theta_{surf} \right) = P^c \left( \theta_{surf} + \frac{\psi_{h10m} + \psi_g}{\psi_{hsl} + \psi_g} (\theta_{sl} - \theta_{surf}) \right)
\]

where:

\[
\theta_s = -\frac{\langle w' \theta' \rangle}{u_*}
\]

where \( \psi_{h(10m)} \) is the non-dimensional temperature gradient at two meters, \( \psi_g \) is the non-dimensional temperature
gradient in the viscous sublayer, and the subscript \( sl \) refers to the height of the top of the surface layer.

Q10M - Specific Humidity at 10 Meter Depth (g/kg)

The specific humidity at the 10-meter depth is an interpolation between the surface specific humidity and the model
lowest level specific humidity using the ratio of the non-dimensional temperature gradient at the two levels:

\[
Q_{10M} = P^c \left( \frac{q_s}{\psi_{h10m} + \psi_g} + q_{surf} \right) = P^c \left( q_{surf} + \frac{\psi_{h10m} + \psi_g}{\psi_{hsl} + \psi_g} (q_{sl} - q_{surf}) \right)
\]

where:

\[
q_s = -\frac{\langle w' q' \rangle}{u_*}
\]

where \( \psi_{h(10m)} \) is the non-dimensional temperature gradient at two meters, \( \psi_g \) is the non-dimensional temperature
gradient in the viscous sublayer, and the subscript \( sl \) refers to the height of the top of the surface layer.

DTRAIN - Cloud Detrainment Mass Flux (kg/m^2)

The amount of cloud mass moved per RAS timestep at the cloud detrainment level is written:

\[
DTRAIN = \eta_{r_D} m_B
\]

where \( r_D \) is the detrainment level, \( m_B \) is the cloud base mass flux, and \( \eta \) is the entrainment, defined in Section 8.5.3.2.

QFILL - Filling of negative Specific Humidity (g/kg/day)

Due to computational errors associated with the numerical scheme used for the advection of moisture, negative values
of specific humidity may be generated. The specific humidity is checked for negative values after every dynamics
timestep. If negative values have been produced, a filling algorithm is invoked which redistributes moisture from
below. Diagnostic QFILL is equal to the net filling needed to eliminate negative specific humidity, scaled to a per-day
rate:

\[
QFILL = q_{final}^{n+1} - q_{initial}^{n+1}
\]

where

\[
q^{n+1} = (\pi q)^{n+1}/\pi^{n+1}
\]
8.5.3.5 Key subroutines, parameters and files

8.5.3.6 Dos and don’ts

8.5.3.7 Fizhi Reference

8.5.3.8 Experiments and tutorials that use fizhi

- Global atmosphere experiment with realistic SST and topography in fizhi-cs-32x32x10 verification directory.
- Global atmosphere aqua planet experiment in fizhi-cs-aqualev20 verification directory.

8.6 Ice and Sea Ice Packages

8.6.1 THSICE: The Thermodynamic Sea Ice Package

**Important note:** This document has been written by Stephanie Dutkiewicz and describes an earlier implementation of the sea-ice package. This needs to be updated to reflect the recent changes (JMC).

This thermodynamic ice model is based on the 3-layer model by Winton (2000), and the energy-conserving LANL CICE model (Bitz and Lipscomb, 1999). The model considers two equally thick ice layers; the upper layer has a variable specific heat resulting from brine pockets, the lower layer has a fixed heat capacity. A zero heat capacity snow layer lies above the ice. Heat fluxes at the top and bottom surfaces are used to calculate the change in ice and snow layer thickness. Grid cells of the ocean model are either fully covered in ice or are open water. There is a provision to parametrize ice fraction (and leads) in this package. Modifications are discussed in small font following the subroutine descriptions.

8.6.1.1 Key parameters and Routines

The ice model is called from `thermodynamics.F`, subroutine `ice_forcing.F` is called in place of `external_forcing_surf.F`. In `ice_forcing.F`, we calculate the freezing potential of the ocean model surface layer of water:

$$ \text{frzmlt} = (T_f - \text{SST}) \frac{c_{sw} \rho_{sw} \Delta z}{\Delta t} $$

where $c_{sw}$ is seawater heat capacity, $\rho_{sw}$ is the seawater density, $\Delta z$ is the ocean model upper layer thickness and $\Delta t$ is the model (tracer) timestep. The freezing temperature, $T_f = \mu S$ is a function of the salinity.

1. Provided there is no ice present and `frzmlt` is less than 0, the surface tendencies of wind, heat and freshwater are calculated as usual (ie. as in `external_forcing_surf.F`).
2. If there is ice present in the grid cell we call the main ice model routine `ice_therm.F` (see below). Output from this routine gives net heat and freshwater flux affecting the top of the ocean.

Subroutine `ice_forcing.F` uses these values to find the sea surface tendencies in grid cells. When there is ice present, the surface stress tendencies are set to zero; the ice model is purely thermodynamic and the effect of ice motion on the sea-surface is not examined.

Relaxation of surface $T$ and $S$ is only allowed equatorward of `relaxlat` (see `DATA.ICE` below), and no relaxation is allowed under the ice at any latitude.

(Note that there is provision for allowing grid cells to have both open water and seaice; if `compact` is between 0 and 1)
subroutine ICE_FREEZE

This routine is called from thermodynamics.F after the new temperature calculation, calc_gt.F, but before calc_gs.F. In ice_freeze.F, any ocean upper layer grid cell with no ice cover, but with temperature below freezing, \( T_f = \mu S \) has ice initialized. We calculate frzmlt from all the grid cells in the water column that have a temperature less than freezing. In this routine, any water below the surface that is below freezing is set to \( T_f \). A call to ice_start.F is made if \( \text{frzmlt} \) > 0, and salinity tendency is updated for brine release.

(There is a provision for fractional ice: In the case where the grid cell has less ice coverage than icemaskmax we allow ice_start.F to be called)

subroutine ICE_START

The energy available from freezing the sea surface is brought into this routine as esurp. The enthalpy of the 2 layers of any new ice is calculated as:

\[
q_1 = -c_i T_f + L_i
\]

\[
q_2 = -c_f T_{\text{mlt}} + c_i (T_{\text{mlt}} - T_f) + L_i (1 - \frac{T_{\text{mlt}}}{T_f})
\]

where \( c_f \) is specific heat of liquid fresh water, \( c_i \) is the specific heat of fresh ice, \( L_i \) is latent heat of freezing, \( \rho_i \) is density of ice and \( T_{\text{mlt}} \) is melting temperature of ice with salinity of 1. The height of a new layer of ice is

\[
h_{\text{inew}} = \frac{\text{esurp} \Delta t}{q_{0av}}
\]

where \( q_{0av} = -\frac{\rho_i}{2} (q_1 + q_2) \).

The surface skin temperature \( T_s \) and ice temperatures \( T_1, T_2 \) and the sea surface temperature are set at \( T_f \).

(There is provision for fractional ice: new ice is formed over open water; the first freezing in the cell must have a height of himin0; this determines the ice fraction compact. If there is already ice in the grid cell, the new ice must have the same height and the new ice fraction is

\[
i_f = (1 - \hat{i}_f) \frac{h_{\text{inew}}}{h_i}
\]

where \( \hat{i}_f \) is ice fraction from previous timestep and \( h_i \) is current ice height. Snow is redistributed over the new ice fraction. The ice fraction is not allowed to become larger than icemaskmax and if the ice height is above himin then freezing energy comes from the full grid cell, ice growth does not occur under orginal ice due to freezing water.)

subroutine ICE_THERM

The main subroutine of this package is ice_therm.F where the ice temperatures are calculated and the changes in ice and snow thicknesses are determined. Output provides the net heat and fresh water fluxes that force the top layer of the ocean model.

If the current ice height is less than himin then the ice layer is set to zero and the ocean model upper layer temperature is allowed to drop lower than its freezing temperature; and atmospheric fluxes are allowed to effect the grid cell. If the ice height is greater than himin we proceed with the ice model calculation.

We follow the procedure of Winton (1999) – see equations 3 to 21 – to calculate the surface and internal ice temperatures. The surface temperature is found from the balance of the flux at the surface \( F_s \), the shortwave heat flux absorbed by the ice, fswnit, and the upward conduction of heat through the snow and/or ice, \( F_u \). We linearize \( F_s \) about the surface temperature, \( T_s \), at the previous timestep (where ‘indicates the value at the previous timestep):

\[
F_s(T_s) = F_s(T_s^{'}) + \frac{\partial F_s(T_s^{'})}{\partial T_s}(T_s - T_s^{'})
\]
where,

\[ F_s = F_{\text{sensible}} + F_{\text{latent}} + F_{\text{downlongwave}} + F_{\text{uplongwave}} + (1 - \alpha)F_{\text{shortwave}} \]

and

\[ \frac{dF_s}{dT} = \frac{dF_{\text{sensible}}}{dT} + \frac{dF_{\text{latent}}}{dT} + \frac{dF_{\text{uplongwave}}}{dT}. \]

\( F_s \) and \( \frac{dF_s}{dT} \) are currently calculated from the BULKF package described separately, but could also be provided by an atmospheric model. The surface albedo is calculated from the ice height and/or surface temperature (see below, stf_albedo.F) and the shortwave flux absorbed in the ice is

\[ \text{fswint} = (1 - e^{\kappa_i h_i})(1 - \alpha)F_{\text{shortwave}} \]

where \( \kappa_i \) is bulk extinction coefficient.

The conductive flux to the surface is

\[ F_u = K_{1/2}(T_1 - T_s) \]

where \( K_{1/2} \) is the effective conductive coupling of the snow-ice layer between the surface and the mid-point of the upper layer of ice:

\[ K_{1/2} = \frac{4 K_i K_s}{K_s h_i + 4 K_i h_s} \]

\( K_i \) and \( K_s \) are constant thermal conductivities of sea ice and snow.

From the above equations we can develop a system of equations to find the skin surface temperature, \( T_s \) and the two ice layer temperatures (see Winton, 1999, for details). We solve these equations iteratively until the change in \( T_s \) is small. When the surface temperature is greater than the melting temperature of the surface, the temperatures are recalculated setting \( T_s \) to 0. The enthalpy of the ice layers are calculated in order to keep track of the energy in the ice model. Enthalpy is defined, here, as the energy required to melt a unit mass of sea ice with temperature \( T \). For the upper layer (1) with brine pockets and the lower fresh layer (2):

\[ q_1 = -c_f T_f + c_i(T_f - T) + L_i(1 - \frac{T_f}{T}) \]
\[ q_2 = -c_i T + L_i \]

where \( c_f \) is specific heat of liquid fresh water, \( c_i \) is the specific heat of fresh ice, and \( L_i \) is latent heat of melting fresh ice.

From the new ice temperatures, we can calculate the energy flux at the surface available for melting (if \( T_s = 0 \)) and the energy at the ocean-ice interface for either melting or freezing.

\[ E_{\text{top}} = (F_s - K_{1/2}(T_s - T_1)) \Delta t \]
\[ E_{\text{bot}} = \left( \frac{4K_i(T_2 - T_f)}{h_i} - F_b \right) \Delta t \]

where \( F_b \) is the heat flux at the ice bottom due to the sea surface temperature variations from freezing. If \( T_{\text{sst}} \) is above freezing, \( F_b = c_{sw} \rho_{\text{sw}}\gamma(T_{\text{sst}} - T_f)u^* \), \( \gamma \) is the heat transfer coefficient and \( u^* = QQ \) is frictional velocity between ice and water. If \( T_{\text{sst}} \) is below freezing, \( F_b = (T_f - T_{\text{sst}})c_f \rho_f \Delta z/\Delta t \) and set \( T_{\text{sst}} \) to \( T_f \). We also include the energy from lower layers that drop below freezing, and set those layers to \( T_f \).

If \( E_{\text{top}} > 0 \) we melt snow from the surface, if all the snow is melted and there is energy left, we melt the ice. If the ice is all gone and there is still energy left, we apply the left over energy to heating the ocean model upper layer (See Winton, 1999, equations 27-29). Similarly if \( E_{\text{bot}} > 0 \) we melt ice from the bottom. If all the ice is melted, the snow is melted (with energy from the ocean model upper layer if necessary). If \( E_{\text{bot}} < 0 \) we grow ice at the bottom

\[ \Delta h_i = -\frac{E_{\text{bot}}}{(q_{\text{bot}}P_i)} \]
where \( q_{\text{bot}} = -c_i T_f + L_i \) is the enthalpy of the new ice. The enthalpy of the second ice layer, \( q_2 \) needs to be modified:

\[
q_2 = \frac{\hat{h}_i/2 \hat{q}_2 + \Delta h_i q_{\text{bot}}}{\hat{h}_i/2 + \Delta h_i}
\]

If there is a ice layer and the overlying air temperature is below 0°C then any precipitation, \( P \) joins the snow layer:

\[
\Delta h_s = -P \frac{\rho_f}{\rho_s} \Delta t,
\]

\( \rho_f \) and \( \rho_s \) are the fresh water and snow densities. Any evaporation, similarly, removes snow or ice from the surface. We also calculate the snow age here, in case it is needed for the surface albedo calculation (see \texttt{srf_albedo.F} below).

For practical reasons we limit the ice growth to \( \text{hilim} \) and snow is limited to \( \text{hslim} \). We converts any ice and/or snow above these limits back to water, maintaining the salt balance. Note however, that heat is not conserved in this conversion; sea surface temperatures below the ice are not recalculated.

If the snow/ice interface is below the waterline, snow is converted to ice (see Winton, 1999, equations 35 and 36). The subroutine \texttt{new_layers_winton.F}, described below, repartitions the ice into equal thickness layers while conserving energy.

The subroutine \texttt{ice_therm.F} now calculates the heat and fresh water fluxes affecting the ocean model surface layer. The heat flux:

\[
q_{\text{net}} = f_{\text{swocn}} - F_b - \frac{\text{esurp}}{\Delta t}
\]

is composed of the shortwave flux that has passed through the ice layer and is absorbed by the water, \( f_{\text{swocn}} = Q Q \), the ocean flux to the ice \( F_b \), and the surplus energy left over from the melting, \( \text{esurp} \). The fresh water flux is determined from the amount of fresh water and salt in the ice/snow system before and after the timestep.

(There is a provision for fractional ice: If ice height is above \( \text{hihig} \) then all energy from freezing at sea surface is used only in the open water apart of the cell (ie. \( F_b \) will only have the conduction term). The melt energy is partitioned by \texttt{frac_energy} between melting ice height and ice extent. However, once ice height drops below \( \text{himont} \) then all energy melts ice extent.)

**subroutine SFC_ALBEDO**

The routine \texttt{ice_therm.F} calls this routine to determine the surface albedo. There are two calculations provided here:

1. from LANL CICE model

\[
\alpha = f_x \alpha_s + (1 - f_x) (\alpha_{i_{\text{min}}} + (\alpha_{i_{\text{max}}} - \alpha_{i_{\text{min}}})(1 - e^{-h_i/h_s}))
\]

where \( f_x \) is 1 if there is snow, 0 if not; the snow albedo, \( \alpha_s \), has two values depending on whether \( T_s < 0 \) or not; \( \alpha_{i_{\text{min}}} \) and \( \alpha_{i_{\text{max}}} \) are ice albedos for thin melting ice, and thick bare ice respectively, and \( h_{i_s} \) is a scale height.

2. From GISS model (Hansen et al 1983)

\[
\alpha = \alpha_i e^{-h_s/h_{i_s}} + \alpha_s (1 - e^{-h_s/h_{i_s}})
\]

where \( \alpha_i \) is a constant albedo for bare ice, \( h_{i_s} \) is a scale height and \( \alpha_s \) is a variable snow albedo.

\[
\alpha_s = \alpha_1 + \alpha_2 e^{-\lambda_a a_s}
\]

where \( \alpha_1 \) is a constant, \( \alpha_2 \) depends on \( T_{s_s} \), \( a_s \) is the snow age, and \( \lambda_a \) is a scale frequency. The snow age is calculated in \texttt{ice_therm.F} and is given in equation 41 in Hansen et al (1983).
subroutine NEW_LAYERS_WINTON

The subroutine new_layers_winton.F repartitions the ice into equal thickness layers while conserving energy. We pass to this subroutine, the ice layer enthalpies after melting/growth and the new height of the ice layers. The ending layer height should be half the sum of the new ice heights from ice_therm.F. The enthalpies of the ice layers are adjusted accordingly to maintain total energy in the ice model. If layer 2 height is greater than layer 1 height then layer 2 gives ice to layer 1 and:

\[ q_1 = f_1 q_1 + (1 - f_1) q_2 \]

where \( f_1 \) is the fraction of the new to old upper layer heights. \( T_1 \) will therefore also have changed. Similarly for when ice layer height 2 is less than layer 1 height, except here we need to to be careful that the new \( T_2 \) does not fall below the melting temperature.

Initializing subroutines

ice_init.F: Set ice variables to zero, or reads in pickup information from pickup.ic (which was written out in checkpoint.F)

ice_readparms.F: Reads data.ice

Diagnostic subroutines

ice_ave.F: Keeps track of means of the ice variables

ice_diags.F: Finds averages and writes out diagnostics

Common Blocks

ICE.h: Ice Variables, also relaxlat and startIceModel

ICE_DIAGS.h: matrices for diagnostics: averages of fields from ice_diags.F

BULKF_ICE_CONSTANTS.h (in BULKF package): all the parameters need by the ice model

Input file DATA.ICE

Here we need to set StartIceModel: which is 1 if the model starts from no ice; and 0 if there is a pickup file with the ice matrices (pickup.ic) which is read in ice_init.F and written out in checkpoint.F. The parameter relaxlat defines the latitude poleward of which there is no relaxing of surface \( T \) or \( S \) to observations. This avoids the relaxation forcing the ice model at these high latitudes.

(Note: hicemin is set to 0 here. If the provision for allowing grid cells to have both open water and seaice is ever implemented, this would be greater than 0)
8.6.1.2 Important Notes

1. heat fluxes have different signs in the ocean and ice models.
2. StartIceModel must be changed in data.ice: 1 (if starting from no ice), 0 (if using pickup.ic file).

8.6.1.3 THSICE Diagnostics

| <-Name-> | Levs | <-parsing code-> | <-- Units --> | <- Tile (max=80c) |
|----------|------|------------------|--------------|-----------------
| SI_Fract | 1    | SM P M1          | 0-1          | Sea-Ice fraction [0-1] |
| SI_Thick | 1    | SM PC197M1       | m            | Sea-Ice thickness (area weighted average) |
| SI_SnowH | 1    | SM PC197M1       | m            | Snow thickness over Sea-Ice (area weighted) |
| SI_Tsrf  | 1    | SM C197M1        | degC         | Surface Temperature over Sea-Ice (area weighted) |
| SI_Tice1 | 1    | SM C197M1        | degC         | Sea-Ice Temperature, 1st layer (area weighted) |
| SI_Tice2 | 1    | SM C197M1        | degC         | Sea-Ice Temperature, 2nd layer (area weighted) |
| SI_Qice1 | 1    | SM C198M1        | J/kg         | Sea-Ice enthalpy, 1st layer (mass weighted) |
| SI_Qice2 | 1    | SM C198M1        | J/kg         | Sea-Ice enthalpy, 2nd layer (mass weighted) |
| SIalbedo | 1    | SM PC197M1       | 0-1          | Sea-Ice Albedo [0-1] (area weighted average) |
| SIsnwAge | 1    | SM P M1          | s            | snow age over Sea-Ice |
| SIsnwPrc | 1    | SM C197M1        | kg/m^2/s     | snow precip. (+=dw) over Sea-Ice (area weighted) |
| SIflxAtm | 1    | SM M1            | W/m^2        | net heat flux from the Atmosphere (+=dw) |
| SIfrwAtm | 1    | SM M1            | kg/m^2/s     | fresh-water flux to the Atmosphere (+=up) |
| SIflxOc  | 1    | SM M1            | W/m^2        | heat flux out of the ocean (+=up) |
| SIfrwOc  | 1    | SM M1            | m/s          | fresh-water flux out of the ocean (+=up) |
| SIsaltFx | 1    | SM M1            | (g/kg).kg/m^2 | salt flux out of the ocean (+=up) |
| SItOcMxL | 1    | SM M1            | degC         | ocean mixed layer temperature |
| SIsOcMxL | 1    | SM P M1          | g/kg         | ocean mixed layer salinity |

8.6.1.4 References


Hunke, E.C and W.H. Lipscomb, circa 2001: CICE: the Los Alamos Sea Ice Model Documentation and Software User’s Manual. LACC-98-16v.2. (note: this documentation is no longer available as CICE has progressed to a very different version 3)


8.6. Ice and Sea Ice Packages
8.6.1.5 Experiments and tutorials that use thsice

- Global atmosphere experiment in aim.5l_cs verification directory, input from input.thsice directory.
- Global ocean experiment in global_ocean.cs32x15 verification directory, input from input.thsice directory.

8.6.2 SEAICE Package

Authors: Martin Losch, Dimitris Menemenlis, An Nguyen, Jean-Michel Campin, Patrick Heimbach, Chris Hill, Jinlun Zhang, and Damien Ringeisen

8.6.2.1 Introduction

Package seaice provides a dynamic and thermodynamic interactive sea ice model.

CPP options enable or disable different aspects of the package (Section 8.6.2.2). Run-time options, flags, filenames and field-related dates/times are set in data.seaice (Section 8.6.2.3). A description of key subroutines is given in Section 8.6.2.6. Available diagnostics output is listed in Section 8.6.2.7.

8.6.2.2 SEAICE configuration and compiling

Compile-time options

As with all MITgcm packages, SEAICE can be turned on or off at compile time (see Section 3.5)

- using the packages.conf file by adding seaice to it
- or using genmake2 adding -enable=seaice or -disable=seaice switches
- required packages and CPP options: seaice requires the external forcing package pkg/exf to be enabled; no additional CPP options are required.

Parts of the seaice code can be enabled or disabled at compile time via CPP preprocessor flags. These options are set in SEAICE_OPTIONS.h. Table 8.16 summarizes the most important ones. For more options see SEAICE_OPTIONS.h.
### Table 8.16: Some of the most relevant CPP preprocessor flags in the seaice package.

<table>
<thead>
<tr>
<th>CPP option</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SEAICE_DEBUG</td>
<td>#undef</td>
<td>enhance STDOUT for debugging</td>
</tr>
<tr>
<td>SEAICE_ALLOW_DYNAMICS</td>
<td>#define</td>
<td>sea ice dynamics code</td>
</tr>
<tr>
<td>SEAICE_CGRID</td>
<td>#define</td>
<td>LSR solver on C-grid (rather than original B-grid)</td>
</tr>
<tr>
<td>SEAICE_ALLOW_EVP</td>
<td>#define</td>
<td>enable use of EVP rheology solver</td>
</tr>
<tr>
<td>SEAICE_ALLOW_JFNK</td>
<td>#define</td>
<td>enable use of JFNK rheology solver</td>
</tr>
<tr>
<td>SEAICE_ALLOW_KRYLOV</td>
<td>#define</td>
<td>enable use of Krylov rheology solver</td>
</tr>
<tr>
<td>SEAICE_ALLOW_TEM</td>
<td>#undef</td>
<td>enable use of the truncated ellipse method (TEM) and coulombic yield curve</td>
</tr>
<tr>
<td>SEAICE_ALLOW_MCS</td>
<td>#undef</td>
<td>enable use of Mohr-Coulomb yield curve with shear flow rule</td>
</tr>
<tr>
<td>SEAICE_ALLOW_MCE</td>
<td>#undef</td>
<td>enable use of Mohr-Coulomb yield curve with elliptical plastic potential</td>
</tr>
<tr>
<td>SEAICE_LSR_ZEBRA</td>
<td>#undef</td>
<td>use a coloring method for LSR solver</td>
</tr>
<tr>
<td>SEAICE_EXTERNAL_FLUXES</td>
<td>#define</td>
<td>use pkg/ext-computed fluxes as starting point</td>
</tr>
<tr>
<td>SEAICE_ZETA_SMOOTHREG</td>
<td>#define</td>
<td>use differentiable regularization for viscosities</td>
</tr>
<tr>
<td>SEAICE_DELTA_SMOOTHREG</td>
<td>#undef</td>
<td>use differentiable regularization for $1/\Delta$</td>
</tr>
<tr>
<td>SEAICE_ALLOW_TD</td>
<td>#undef</td>
<td>enable use of teardrop and parabolic Lens yield curves with normal flow rules</td>
</tr>
<tr>
<td>SEAICE_ITD</td>
<td>#undef</td>
<td>run with dynamical sea Ice Thickness Distribution (ITD)</td>
</tr>
<tr>
<td>SEAICE_VARIABLE_SALINITY</td>
<td>#undef</td>
<td>enable sea ice with variable salinity</td>
</tr>
<tr>
<td>SEAICE_CAP_ICELOAD</td>
<td>#undef</td>
<td>enable to limit seaice load (siceLoad) on the sea surface</td>
</tr>
<tr>
<td>ALLOW_SITRACER</td>
<td>#undef</td>
<td>enable sea ice tracer package</td>
</tr>
<tr>
<td>SEAICE_BICE_STRESS</td>
<td>#undef</td>
<td>B-grid only for backward compatibility: turn on ice-stress on ocean</td>
</tr>
<tr>
<td>EXPLICIT_SSH_SLOPE</td>
<td>#undef</td>
<td>B-grid only for backward compatibility: use ETAN for tilt computations rather than geostrophic velocities</td>
</tr>
</tbody>
</table>

### 8.6.2.3 Run-time parameters

Run-time parameters (see Table 8.17) are set in data.seaice (read in pkg/seaice/seaice_readparms.F).

**Enabling the package**

seaice package is switched on/off at run-time by setting useSEAICE = .TRUE., in data.pkg.

**General flags and parameters**

Table 8.17 lists most run-time parameters.

<table>
<thead>
<tr>
<th>Name</th>
<th>Default value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SEAICE_writeState</td>
<td>FALSE</td>
<td>write sea ice state to file</td>
</tr>
<tr>
<td>SEAICE_useDYNAMICS</td>
<td>TRUE</td>
<td>use dynamics</td>
</tr>
<tr>
<td>SEAICE_useJFNK</td>
<td>FALSE</td>
<td>use the JFNK-solver</td>
</tr>
</tbody>
</table>

continues on next page
<table>
<thead>
<tr>
<th>Name</th>
<th>Default value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SEAICEuseTEM</td>
<td>FALSE</td>
<td>use truncated ellipse method or coulombic yield curve</td>
</tr>
<tr>
<td>SEAICEuseMCS</td>
<td>FALSE</td>
<td>use the Mohr-Coulomb yield curve with shear flow rule</td>
</tr>
<tr>
<td>SEAICEuseMCE</td>
<td>FALSE</td>
<td>use the Mohr-Coulomb yield curve with elliptical plastic potential</td>
</tr>
<tr>
<td>SEAICEuseTD</td>
<td>FALSE</td>
<td>use the teardrop yield curve with normal flow rule</td>
</tr>
<tr>
<td>SEAICEusePL</td>
<td>FALSE</td>
<td>use the parabolic Lens yield curve with normal flow rule</td>
</tr>
<tr>
<td>SEAICEuseStrImpCpl</td>
<td>FALSE</td>
<td>use strength implicit coupling in LSR/JFNK</td>
</tr>
<tr>
<td>SEAICEuseMetricTerms</td>
<td>TRUE</td>
<td>use metric terms in dynamics</td>
</tr>
<tr>
<td>SEAICEuseEVPpickup</td>
<td>TRUE</td>
<td>use EVP pickups</td>
</tr>
<tr>
<td>SEAICEuseFluxForm</td>
<td>TRUE</td>
<td>use flux form for 2nd central difference advection scheme</td>
</tr>
<tr>
<td>SEAICErestoreUnderIce</td>
<td>FALSE</td>
<td>enable restoring to climatology under ice</td>
</tr>
<tr>
<td>SEAICEupdateOceanStress</td>
<td>TRUE</td>
<td>update ocean surface stress accounting for sea ice cover</td>
</tr>
<tr>
<td>SEAICEscaleSurfStress</td>
<td>TRUE</td>
<td>scale atmosphere and ocean-surface stress on ice by concentration (AREA)</td>
</tr>
<tr>
<td>SEAICEaddSnowMass</td>
<td>TRUE</td>
<td>in computing seaiceMass, add snow contribution</td>
</tr>
<tr>
<td>useHB87stressCoupling</td>
<td>FALSE</td>
<td>turn on ice-ocean stress coupling following</td>
</tr>
<tr>
<td>usePW79thermodynamics</td>
<td>TRUE</td>
<td>flag to turn off zero-layer-thermodynamics for testing</td>
</tr>
<tr>
<td>SEAICEadvHelf</td>
<td>TRUE</td>
<td>flag to turn off advection of scalar variable HEFF</td>
</tr>
<tr>
<td>SEAICEadvArea</td>
<td>TRUE</td>
<td>flag to turn off advection of scalar variable AREA</td>
</tr>
<tr>
<td>SEAICEadvSnow</td>
<td>TRUE</td>
<td>flag to turn off advection of scalar variable HSNOW</td>
</tr>
<tr>
<td>SEAICEadvSalt</td>
<td>TRUE</td>
<td>flag to turn off advection of scalar variable HSALT</td>
</tr>
<tr>
<td>SEAICEadvScheme</td>
<td>77</td>
<td>set advection scheme for seaice scalar state variables</td>
</tr>
<tr>
<td>SEAICEuseFlooding</td>
<td>TRUE</td>
<td>use flood-freeze algorithm</td>
</tr>
<tr>
<td>SEAICE_no_slip</td>
<td>FALSE</td>
<td>use no-slip boundary conditions instead of free-slip</td>
</tr>
<tr>
<td>SEAICE_deltaTterm</td>
<td>dTracerLev (1)</td>
<td>time step for seaice thermodynamics (s)</td>
</tr>
<tr>
<td>SEAICE_deltaTdyn</td>
<td>dTracerLev (1)</td>
<td>time step for seaice dynamics (s)</td>
</tr>
<tr>
<td>SEAICE_deltaTevp</td>
<td>0.0</td>
<td>EVP sub-cycling time step (s); values &gt; 0 turn on EVP</td>
</tr>
<tr>
<td>SEAICEuseEVPstar</td>
<td>TRUE</td>
<td>use modified EVP* instead of EVP, following [LKT+12]</td>
</tr>
<tr>
<td>SEAICEuseEVPrev</td>
<td>TRUE</td>
<td>“revisited form” variation on EVP*, following [BFLM13]</td>
</tr>
<tr>
<td>SEAICEnEVPstarSteps</td>
<td>unset</td>
<td>number of modified EVP* iterations</td>
</tr>
<tr>
<td>SEAICE_evpAlpha</td>
<td>unset</td>
<td>EVP* parameter (non-dim.), to replace 2*SEAICE_evpTauRelax/SEAICE_deltaTevp</td>
</tr>
<tr>
<td>SEAICE_evpBeta</td>
<td>unset</td>
<td>EVP* parameter (non-dim.), to replace SEACIEnEVPstarSteps/SEAICE_deltaTevp</td>
</tr>
<tr>
<td>SEAICEaEVPcCoeff</td>
<td>unset</td>
<td>largest stabilized frequency for adaptive EVP (non-dim.)</td>
</tr>
<tr>
<td>SEAICEaEVPcStar</td>
<td>4.0</td>
<td>aEVP multiple of stability factor (non-dim.), see [KDL16] ( \alpha \beta = c^* \gamma )</td>
</tr>
<tr>
<td>SEAICEaEVPalphaMin</td>
<td>5.0</td>
<td>aEVP lower limit of alpha and beta (non-dim.), see [KDL16]</td>
</tr>
<tr>
<td>SEAICE_elasticParm</td>
<td>0.333333333</td>
<td>EVP parameter ( E_0 ) (non-dim.), sets relaxation timescale</td>
</tr>
<tr>
<td>SEAICE_evpTauRelax</td>
<td>dTracerLev (1) * SEAICE_elasticParm</td>
<td>relaxation time scale ( T ) for EVP waves (s)</td>
</tr>
<tr>
<td>SEAICE_OLx</td>
<td>OLx - 2</td>
<td>overlap for LSR-solver or preconditioner, ( x )-dimension</td>
</tr>
<tr>
<td>SEAICE_OLy</td>
<td>OLY - 2</td>
<td>overlap for LSR-solver or preconditioner, ( y )-dimension</td>
</tr>
<tr>
<td>SEAICE_nonLinIterMax</td>
<td>2/10</td>
<td>maximum number of non-linear (outer loop) iterations (LSR/JFNK)</td>
</tr>
<tr>
<td>SEAICElinearIterMax</td>
<td>1500/10</td>
<td>maximum number of linear iterations (LSR/JFNK)</td>
</tr>
<tr>
<td>SEAICE_JFNK_lsIter</td>
<td>(off)</td>
<td>start line search after “lsIter” Newton iterations</td>
</tr>
<tr>
<td>Name</td>
<td>Default value</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------</td>
<td>---------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>SEAICE_JFNK_lsLmax</td>
<td>4</td>
<td>maximum number of line search steps</td>
</tr>
<tr>
<td>SEAICE_JFNK_lsGamma</td>
<td>0.5</td>
<td>line search step size parameter</td>
</tr>
<tr>
<td>JFNkGamma_lin_min</td>
<td>0.10</td>
<td>minimum tolerance parameter for linear JFNK solver</td>
</tr>
<tr>
<td>JFNkGamma_lin_max</td>
<td>0.99</td>
<td>maximum tolerance parameter for linear JFNK solver</td>
</tr>
<tr>
<td>JFNKres_tFac</td>
<td>unset</td>
<td>tolerance parameter for FGMRES residual</td>
</tr>
<tr>
<td>SEAICE_JFNKepsilon</td>
<td>1.0E-06</td>
<td>step size for the FD-gradient in s/r seaice_jacvec</td>
</tr>
<tr>
<td>SEAICE_dumpFreq</td>
<td>dumpFreq</td>
<td>dump frequency (s)</td>
</tr>
<tr>
<td>SEAICE_dump_mdsio</td>
<td>TRUE</td>
<td>write snapshot using /pkg/mdsio</td>
</tr>
<tr>
<td>SEAICE_dump_mnc</td>
<td>FALSE</td>
<td>write snapshot using /pkg/mnc</td>
</tr>
<tr>
<td>OCEAN_drag</td>
<td>1.0E-03</td>
<td>air-ocean drag coefficient (non-dim.)</td>
</tr>
<tr>
<td>SEAICE_waterDrag</td>
<td>5.5E-03</td>
<td>water-ice drag coefficient (non-dim.)</td>
</tr>
<tr>
<td>SEAICE_dryIceAlb</td>
<td>0.75</td>
<td>winter sea ice albedo</td>
</tr>
<tr>
<td>SEAICE_wetIceAlb</td>
<td>0.66</td>
<td>summer sea ice albedo</td>
</tr>
<tr>
<td>SEAICE_drySnowAlb</td>
<td>0.84</td>
<td>dry snow albedo</td>
</tr>
<tr>
<td>SEAICE_wetSnowAlb</td>
<td>0.70</td>
<td>wet snow albedo</td>
</tr>
<tr>
<td>SEAICE_waterAlbedo</td>
<td>0.10</td>
<td>water albedo (not used if #define SEAICE_EXTERNAL_FLUXES)</td>
</tr>
<tr>
<td>SEAICE_strength</td>
<td>2.75E+04</td>
<td>sea ice strength constant $P^*$ (N/m²)</td>
</tr>
<tr>
<td>SEAICE_cStar</td>
<td>20.0</td>
<td>sea ice strength constant $C^*$ (non-dim.)</td>
</tr>
<tr>
<td>SEAICE_eccen</td>
<td>2.0</td>
<td>VP rheology ellipse aspect ratio $e$</td>
</tr>
<tr>
<td>SEAICE_eccfr</td>
<td>= SEAICE_eccen</td>
<td>sea ice plastic potential ellipse aspect ratio $c_{GC}$</td>
</tr>
<tr>
<td>SEAICE_emcMU</td>
<td>1.0</td>
<td>slope of the Mohr-Coulomb yield curve</td>
</tr>
<tr>
<td>SEAICEpressReplFac</td>
<td>1.0</td>
<td>use replacement pressure (0.0-1.0)</td>
</tr>
<tr>
<td>SEAICE_tensilFac</td>
<td>0.0</td>
<td>tensile factor for the yield curve</td>
</tr>
<tr>
<td>SEAICE_rhoAir</td>
<td>1.3 (or pkg/exf value)</td>
<td>density of air (kg/m³)</td>
</tr>
<tr>
<td>SEAICE_cpAir</td>
<td>1004.0</td>
<td>specific heat of air (J/kg/K)</td>
</tr>
<tr>
<td>SEAICE_lhEvap</td>
<td>2.5E+06</td>
<td>latent heat of evaporation (J/kg)</td>
</tr>
<tr>
<td>SEAICE_lhFusion</td>
<td>3.34E+05</td>
<td>latent heat of fusion (J/kg)</td>
</tr>
<tr>
<td>SEAICE_dalton</td>
<td>1.75E-03</td>
<td>ice-ocean transfer coefficient for latent and sensible heat (non-dim.)</td>
</tr>
<tr>
<td>useMaykutSatVapPoly</td>
<td>FALSE</td>
<td>use Maykut polynomial to compute saturation vapor pressure</td>
</tr>
<tr>
<td>SEAICE_iceConduct</td>
<td>2.16560E+00</td>
<td>sea ice conductivity (W m⁻¹ K⁻¹)</td>
</tr>
<tr>
<td>SEAICE_snowConduct</td>
<td>3.10000E-01</td>
<td>snow conductivity (W m⁻¹ K⁻¹)</td>
</tr>
<tr>
<td>SEAICE_emissivity</td>
<td>0.970018</td>
<td>longwave ocean surface emissivity (non-dim.)</td>
</tr>
<tr>
<td>SEAICE_snowThick</td>
<td>0.15</td>
<td>cutoff snow thickness to use snow albedo (m)</td>
</tr>
<tr>
<td>SEAICE_shortwave</td>
<td>0.30</td>
<td>ice penetration shortwave radiation factor (non-dim.)</td>
</tr>
<tr>
<td>SEAICE_saltFrac</td>
<td>0.0</td>
<td>salinity newly formed ice (as fraction of ocean surface salinity)</td>
</tr>
<tr>
<td>SEAICE_frazilFrac</td>
<td>1.0 (or computed from other parms)</td>
<td>frazil to sea ice conversion rate, as fraction (relative to the local freezing point of sea ice water)</td>
</tr>
</tbody>
</table>

continues on next page
<table>
<thead>
<tr>
<th>Name</th>
<th>Default value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SEAICEstressFactor</td>
<td>1.0</td>
<td>scaling factor for ice area in computing total ocean stress (non-dim.)</td>
</tr>
<tr>
<td>HeffFile</td>
<td>unset</td>
<td>filename for initial sea ice eff. thickness field HEFF (m)</td>
</tr>
<tr>
<td>AreaFile</td>
<td>unset</td>
<td>filename for initial fraction sea ice cover AREA (non-dim.)</td>
</tr>
<tr>
<td>HsnowFile</td>
<td>unset</td>
<td>filename for initial eff. snow thickness field HSNOW (m)</td>
</tr>
<tr>
<td>HsaltFile</td>
<td>unset</td>
<td>filename for initial eff. sea ice salinity field HSALT (g/m²)</td>
</tr>
<tr>
<td>LSR_ERROR</td>
<td>1.0E-05</td>
<td>sets accuracy of LSR solver</td>
</tr>
<tr>
<td>DIFF1</td>
<td>0.0</td>
<td>parameter used in advect.F</td>
</tr>
<tr>
<td>HO</td>
<td>0.5</td>
<td>lead closing parameter $h_0$ (m); demarcation thickness between thick and thin ice which determines partition between vertical and lateral ice growth</td>
</tr>
<tr>
<td>MIN_ATEMP</td>
<td>-50.0</td>
<td>minimum air temperature (°C)</td>
</tr>
<tr>
<td>MIN_LWDOWN</td>
<td>60.0</td>
<td>minimum downward longwave (W/m²)</td>
</tr>
<tr>
<td>MIN_TICE</td>
<td>-50.0</td>
<td>minimum ice temperature (°C)</td>
</tr>
<tr>
<td>MAX_TICE</td>
<td>10</td>
<td>number of iterations for ice surface temperature solution</td>
</tr>
<tr>
<td>SEAICE_EPS</td>
<td>1.0E-10</td>
<td>a “small number” used in various routines</td>
</tr>
<tr>
<td>SEAICE_area_reg</td>
<td>1.0E-5</td>
<td>minimum concentration to regularize ice thickness</td>
</tr>
<tr>
<td>SEAICE_hice_reg</td>
<td>0.05</td>
<td>minimum ice thickness (m) for regularization</td>
</tr>
<tr>
<td>SEAICE_multDim</td>
<td>1</td>
<td>number of ice categories for thermodynamics</td>
</tr>
<tr>
<td>SEAICE_useMultDimSnow</td>
<td>TRUE</td>
<td>use same fixed pdf for snow as for multi-thickness-category ice</td>
</tr>
</tbody>
</table>

The following dynamical ice thickness distribution and ridging parameters in Table 8.18 are only active with #define SEAICE_ITD. All parameters are non-dimensional unless indicated.

<table>
<thead>
<tr>
<th>Name</th>
<th>Default value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>useHibler79IceStrength</td>
<td>TRUE</td>
<td>use [Hib79] ice strength; do not use [Rot75] with #define SEAICE_ITD</td>
</tr>
<tr>
<td>SEAICEsimpleRidging</td>
<td>TRUE</td>
<td>use simple ridging a la [Hib79]</td>
</tr>
<tr>
<td>SEAICE_cf</td>
<td>17.0</td>
<td>scaling parameter of [Rot75] ice strength parameterization</td>
</tr>
<tr>
<td>SEAICEpartFunc</td>
<td>0</td>
<td>use partition function of [TRMC75]</td>
</tr>
<tr>
<td>SEAICEredistFunc</td>
<td>0</td>
<td>use redistribution function of [Hib80]</td>
</tr>
<tr>
<td>SEAICEridgingIterMax</td>
<td>10</td>
<td>maximum number of ridging sweeps</td>
</tr>
<tr>
<td>SEAICEshearParm</td>
<td>0.5</td>
<td>fraction of shear to be used for ridging</td>
</tr>
<tr>
<td>SEAICEgStar</td>
<td>0.15</td>
<td>max. ice conc. that participates in ridging [TRMC75]</td>
</tr>
<tr>
<td>SEAICEhStar</td>
<td>25.0</td>
<td>ridging parameter for [TRMC75], [LMHJ07]</td>
</tr>
<tr>
<td>SEAICEeStar</td>
<td>0.05</td>
<td>similar to SEAICEgStar for [LMHJ07] participation function</td>
</tr>
<tr>
<td>SEAICEemuRidging</td>
<td>3.0</td>
<td>similar to SEAICEhStar for [LMHJ07] ridging function</td>
</tr>
<tr>
<td>SEAICEmaxRaft</td>
<td>1.0</td>
<td>regularization parameter for rafting</td>
</tr>
<tr>
<td>SEAICEsnowFracRidge</td>
<td>0.5</td>
<td>fraction of snow that remains on ridged ice</td>
</tr>
<tr>
<td>SEAICEuseLinRemapITD</td>
<td>TRUE</td>
<td>use linear remapping scheme of [Lip01]</td>
</tr>
<tr>
<td>Hlimit</td>
<td>unset</td>
<td>nITD+1-array of ice thickness category limits (m)</td>
</tr>
<tr>
<td>Hlimit_c1</td>
<td>3.0, 15.0, 3.0</td>
<td>when Hlimit is not set, then these parameters determine Hlimit from a simple function following [Lip01]</td>
</tr>
</tbody>
</table>
8.6.2.4 Description

The MITgcm sea ice model is based on a variant of the viscous-plastic (VP) dynamic-thermodynamic sea ice model (Zhang and Hibler 1997 [ZH97]) first introduced in Hibler (1979) and Hibler (1980) [Hib79, Hib80]. In order to adapt this model to the requirements of coupled ice-ocean state estimation, many important aspects of the original code have been modified and improved, see Losch et al. (2010) [LMC+10]:

- the code has been rewritten for an Arakawa C-grid, both B- and C-grid variants are available; the C-grid code allows for no-slip and free-slip lateral boundary conditions;
- three different solution methods for solving the nonlinear momentum equations have been adopted: LSOR (Zhang and Hibler 1997 [ZH97]), EVP (Hunke and Dukowicz 1997 [HD97]), JFNK (Lemieux et al. 2010 [LTSedlavec+10], Losch et al. 2014 [LFLV14]);
- ice-ocean stress can be formulated as in Hibler and Bryan (1987) [HB87] or as in Campin et al. (2008) [CMF08];
- ice variables are advected by sophisticated, conservative advection schemes with flux limiting;
- growth and melt parameterizations have been refined and extended in order to allow for more stable automatic differentiation of the code.

The sea ice model is tightly coupled to the ocean component of the MITgcm. Heat, fresh water fluxes and surface stresses are computed from the atmospheric state and, by default, modified by the ice model at every time step.

The ice dynamics models that are most widely used for large-scale climate studies are the viscous-plastic (VP) model (Hibler 1979 [Hib79]), the cavitating fluid (CF) model (Flato and Hibler 1992 [FWDH92]), and the elastic-viscous-plastic (EVP) model (Hunke and Dukowicz 1997 [HD97]). Compared to the VP model, the CF model does not allow ice shear in calculating ice motion, stress, and deformation. EVP models approximate VP by adding an elastic term to the equations for easier adaptation to parallel computers. Because of its higher accuracy in plastic solution and relatively simpler formulation, compared to the EVP model, we decided to use the VP model as the default dynamic component of our ice model. To do this we extended the line successive over relaxation (LSOR) method of Zhang and Hibler (1997) [ZH97] for use in a parallel configuration. An EVP model and a free-drift implementation can be selected with run-time flags.

Compatibility with ice-thermodynamics package pkg/thsice

By default pkg/seaice includes the original so-called zero-layer thermodynamics with a snow cover as in the appendix of Semtner (1976) [Sem76]. The zero-layer thermodynamic model assumes that ice does not store heat and, therefore, tends to exaggerate the seasonal variability in ice thickness. This exaggeration can be significantly reduced by using Winton’s (Winton 2000 [Win00]) three-layer thermodynamic model that permits heat storage in ice.

The Winton (2000) sea-ice thermodynamics have been ported to MITgcm; they currently reside under pkg/thsice, described in Section 8.6.1. It is fully compatible with the packages seaice and exf. When turned on together with seaice, the zero-layer thermodynamics are replaced by the Winton thermodynamics. In order to use package seaice with the thermodynamics of pkg/thsice, compile both packages and turn both package on in data.pkg; see an example in verification/global_ocean.cs32x15/input.icedyn. Note, that once thsice is turned on, the variables and diagnostics associated to the default thermodynamics are meaningless, and the diagnostics of thsice must be used instead.
**Surface forcing**

The sea ice model requires the following input fields: 10 m winds, 2 m air temperature and specific humidity, downward longwave and shortwave radiations, precipitation, evaporation, and river and glacier runoff. The sea ice model also requires surface temperature from the ocean model and the top level horizontal velocity. Output fields are surface wind stress, evaporation minus precipitation minus runoff, net surface heat flux, and net shortwave flux. The sea-ice model is global: in ice-free regions bulk formulae (by default computed in package exf) are used to estimate oceanic forcing from the atmospheric fields.

**Dynamics**

The momentum equation of the sea-ice model is

\[
m \frac{Du}{Dt} = -mfk \times u + \tau_{\text{air}} + \tau_{\text{ocean}} - m\nabla \phi(0) + F
\]  

(8.2)

where \( m = m_i + m_s \) is the ice and snow mass per unit area; \( u = ui + vj \) is the ice velocity vector; \( i, j, \text{ and } k \) are unit vectors in the \( x, y, \text{ and } z \) directions, respectively; \( f \) is the Coriolis parameter; \( \tau_{\text{air}} \) and \( \tau_{\text{ocean}} \) are the wind-ice and ocean-ice stresses, respectively; \( g \) is the gravity acceleration; \( \nabla \phi(0) \) is the gradient (or tilt) of the sea surface height; \( \phi(0) = g\eta + p_a/\rho_0 + mg/\rho_0 \) is the sea surface height potential in response to ocean dynamics (\( g\eta \)), to atmospheric pressure loading (\( p_a/\rho_0 \), where \( \rho_0 \) is a reference density) and a term due to snow and ice loading; and \( F = \nabla \cdot \sigma \) is the divergence of the internal ice stress tensor \( \sigma_{ij} \). Advection of sea-ice momentum is neglected. The wind and ice-ocean stress terms are given by

\[
\tau_{\text{air}} = \rho_{\text{air}}C_{\text{air}}|U_{\text{air}} - u|R_{\text{air}}(U_{\text{air}} - u) \\
\tau_{\text{ocean}} = \rho_{\text{ocean}}C_{\text{ocean}}|U_{\text{ocean}} - u|R_{\text{ocean}}(U_{\text{ocean}} - u)
\]

where \( U_{\text{air/ocean}} \) are the surface winds of the atmosphere and surface currents of the ocean, respectively; \( C_{\text{air/ocean}} \) are air and ocean drag coefficients; \( \rho_{\text{air/ocean}} \) are reference densities; and \( R_{\text{air/ocean}} \) are rotation matrices that act on the wind/current vectors.

**Viscous-Plastic (VP) Rheology**

For an isotropic system the stress tensor \( \sigma_{ij} (i,j = 1,2) \) can be related to the ice strain rate and strength by a nonlinear viscous-plastic (VP) constitutive law:

\[
\sigma_{ij} = 2\eta(\dot{\epsilon}_{ij}, P)\dot{\epsilon}_{ij} + [\zeta(\dot{\epsilon}_{ij}, P) - \eta(\dot{\epsilon}_{ij}, P)] \dot{\epsilon}_{kk} \delta_{ij} - \frac{P}{2} \delta_{ij}
\]  

(8.3)

The ice strain rate is given by

\[
\dot{\epsilon}_{ij} = \frac{1}{2} \left( \frac{\partial U_i}{\partial x_j} + \frac{\partial U_j}{\partial x_i} \right)
\]

The maximum ice pressure \( P_{\text{max}} \) (variable \( \text{PRESS0} \) in the code), a measure of ice strength, depends on both thickness \( h \) and compactness (concentration) \( c \):

\[
P_{\text{max}} = P^* c h \exp\{-C^* \cdot (1 - c)\},
\]

(8.4)

with the constants \( P^* \) (run-time parameter \( \text{SEAICE_strength} \)) and \( C^* \) (run-time parameter \( \text{SEAICE_cStar} \)). By default, \( P \) (variable \( \text{PRESS} \) in the code) is the replacement pressure

\[
P = (1 - k_4) P_{\text{max}} \left( (1 - f_r) + f_r \frac{\Delta}{\Delta_{\text{reg}}} \right)
\]

(8.5)
where $f_r$ is run-time parameter `SEAICEpressReplFac` (default = 1.0), and $\Delta_{reg}$ is a regularized form of $\Delta = \left[ (\dot{\epsilon}_{11} + \dot{\epsilon}_{22})^2 + e^{-2} (\dot{\epsilon}_{11} - \dot{\epsilon}_{22})^2 + \dot{\epsilon}_{12}^2 \right]^{\frac{1}{2}}$, for example $\Delta_{reg} = \max(\Delta, \Delta_{min})$.

The tensile strength factor $k_t$ (run-time parameter `SEAICE_tensilFac`) determines the ice tensile strength $T = k_t \cdot P_{max}$, as defined by König Beatty and Holland (2010) [KBH10]. `SEAICE_tensilFac` is zero by default.

Different VP rheologies can be used to model sea ice dynamics. The different rheologies are characterized by different definitions of the bulk and shear viscosities $\zeta$ and $\eta$ in (8.3). The following Table 8.19 is a summary of the available choices with recommended (sensible) parameter values. All the rheologies presented here depend on the ice strength $P$ (8.5).

Table 8.19: Overview over available sea ice viscous-plastic rheologies

<table>
<thead>
<tr>
<th>Name</th>
<th>CPP flags</th>
<th>Run-time flags (recommended value)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elliptical yield curve with normal flow rule</td>
<td>None (default)</td>
<td>• <code>SEAICE_eccen</code> (2.0)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• <code>SEAICE_tensilFac</code> (= 0.0)</td>
</tr>
<tr>
<td>Elliptical yield curve with non-normal flow rule</td>
<td>None</td>
<td>• <code>SEAICE_eccen</code> (2.0)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• <code>SEAICE_eccfr</code> (&lt; 2.0)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• <code>SEAICE_tensilFac</code> (= 0.0)</td>
</tr>
<tr>
<td>Truncated ellipse method (TEM) for elliptical yield curve</td>
<td><code>SEAICE_ALLOW_TEM</code></td>
<td>• <code>SEAICEuseTEM</code> (= .TRUE.)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• <code>SEAICE_eccen</code> (1.4)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• <code>SEAICE_eccfr</code> (&lt; 1.4)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• <code>SEAICE_tensilFac</code> (= 0.05)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• <code>SEAICEEmcMU</code> (= 0.6 to 0.8)</td>
</tr>
<tr>
<td>Mohr-Coulomb yield curve with elliptical plastic potential</td>
<td><code>SEAICE_ALLOW_MCE</code></td>
<td>• <code>SEAICEuseMCE</code> (= .TRUE.)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• <code>SEAICE_eccen</code> (1.4)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• <code>SEAICE_eccfr</code> (&lt; 1.4)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• <code>SEAICE_tensilFac</code> (= 0.05)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• <code>SEAICEEmcMU</code> (= 0.6 to 0.8)</td>
</tr>
</tbody>
</table>

continues on next page
Table 8.19 – continued from previous page

<table>
<thead>
<tr>
<th>Name</th>
<th>CPP flags</th>
<th>Run-time flags (recommended value)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mohr-Coulomb yield curve with shear flow rule</td>
<td>SEAICE_ALLOW_MCS</td>
<td>• SEAICE_useMCS (= .TRUE.)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• SEAICE_tensilFac (= 0.05)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• SEAICE_emcMU (= 0.6 to 0.8)</td>
</tr>
<tr>
<td>Teardrop yield curve with normal flow rule</td>
<td>SEAICE_ALLOW_TD</td>
<td>• SEAICE_useTD (= .TRUE.)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• SEAICE_tensilFac (= 0.025)</td>
</tr>
<tr>
<td>Parabolic lens yield curve with normal flow rule</td>
<td>SEAICE_ALLOW_TD</td>
<td>• SEAICE_usePL (= .TRUE.)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• SEAICE_tensilFac (= 0.025)</td>
</tr>
</tbody>
</table>

Note: With the exception of the default rheology and the TEM (with SEAICE_emcMU : \( \mu = 1.0 \)), these rheologies are not implemented in EVP (Section 8.6.2.4).

Elliptical yield curve with normal flow rule

The default rheology in the sea ice module of the MITgcm implements the widely used elliptical yield curve with a normal flow rule [Hib79]. For this yield curve, the nonlinear bulk and shear viscosities \( \zeta \) and \( \eta \) are functions of ice strain rate invariants and ice strength such that the principal components of the stress lie on an elliptical yield curve with the ratio of major to minor axis \( e = 2.0 \) (run-time parameter SEAICE_eccen); they are given by:

\[
\zeta = \min \left( \frac{(1 + k_t)P_{\text{max}}}{2 \max(\Delta, \Delta_{\text{min}})}, \zeta_{\text{max}} \right)
\]

\[
\eta = \frac{\zeta}{e^2}
\]

with the abbreviation

\[
\Delta = \left[ (\dot{\varepsilon}_{11} + \dot{\varepsilon}_{22})^2 + e^{-2} \left( (\dot{\varepsilon}_{11} - \dot{\varepsilon}_{22})^2 + \dot{\varepsilon}_{12}^2 \right) \right]^{\frac{1}{2}}
\]

The bulk viscosities are bounded above by imposing both a minimum \( \Delta_{\text{min}} \) (for numerical reasons, run-time parameter SEAICE_deltaMin is set to a default value of \( 10^{-10} \) s\(^{-1} \), the value of SEAICE_EPS) and a maximum \( \zeta_{\text{max}} = P_{\text{max}}/(2\Delta^*) \), where \( \Delta^* = (2 \times 10^4/5 \times 10^{12}) \) s\(^{-1} = 2 \times 10^{-9} \) s\(^{-1} \). Obviously, this corresponds to regularizing \( \Delta \) with the typical value of SEAICE_deltaMin = 2 \times 10^{-9}. Clearly, some of this regularization is redundant. (There is also the option of bounding \( \zeta \) from below by setting run-time parameter SEAICE_zetaMin > 0, but this is generally not recommended). For stress tensor computation the replacement pressure \( P = 2 \Delta \zeta \) is used so that the stress state always lies on the elliptical yield curve by definition.
Defining the CPP-flag `SEAICE_ZETA_SMOOTHREG` in `SEAICE_OPTIONS.h` before compiling replaces the method for bounding $\zeta$ by a smooth (differentiable) expression:

$$\zeta = \zeta_{\text{max}} \tanh \left( \frac{(1 + k_t) P_{\text{max}}}{2 \min(\Delta, \Delta_{\text{min}}) \zeta_{\text{max}}} \right)$$

$$= \frac{(1 + k_t) P_{\text{max}}}{2 \Delta^*} \tanh \left( \frac{\Delta^*}{\min(\Delta, \Delta_{\text{min}})} \right)$$

(8.7)

where $\Delta_{\text{min}} = 10^{-20}$ s$^{-1}$ should be chosen to avoid divisions by zero.

In this default formulation the yield curve does not allow isotropic tensile stress, that is, sea ice can be “pulled apart” without any effort. Setting the parameter $k_t$ (`SEAICE_tensilFac`) to a small value larger than zero, extends the yield curve into a region where the divergence of the stress $\sigma_{11} + \sigma_{22} > 0$ to allow some tensile stress.

Besides this commonly used default rheology, a number of a alternative rheologies are implemented. Some of these are experimental and should be used with caution.

### Elliptical yield curve with non-normal flow rule

Defining the run-time parameter `SEAICE_eccfr` with a value different from `SEAICE_eccen` allows one to use an elliptical yield curve with a non-normal flow rule as described in Ringeisen et al. (2020) [RTL20]. In this case the viscosities are functions of $e_F$ (`SEAICE_eccen`) and $e_G$ (`SEAICE_eccfr`):

$$\zeta = \frac{P_{\text{max}}(1 + k_t)}{2 \Delta}$$

$$\eta = \frac{\zeta}{e_G^2} = \frac{P_{\text{max}}(1 + k_t)}{2 e_G^2 \Delta}$$

with the abbreviation

$$\Delta = \sqrt{(\dot{\epsilon}_{11} - \dot{\epsilon}_{22})^2 + \frac{e_F^2}{e_G^2} ((\dot{\epsilon}_{11} - \dot{\epsilon}_{22})^2 + 4 \dot{\epsilon}_{12}^2)}.$$  

Note that if $e_G = e_F = e$, these formulae reduce to the normal flow rule.

### Truncated ellipse method (TEM) for elliptical yield curve

In the so-called truncated ellipse method, the shear viscosity $\eta$ is capped to suppress any tensile stress:

$$\eta = \min \left( \frac{\zeta}{e^2}, \frac{(1 + k_t) P_{\text{max}} - \zeta (\dot{\epsilon}_{11} + \dot{\epsilon}_{22})}{\sqrt{\max(\Delta_{\text{min}}^*(\dot{\epsilon}_{11} - \dot{\epsilon}_{22})^2 + 4 \dot{\epsilon}_{12}^2)}} \right).$$

(8.8)

To enable this method, set `#define SEAICE_ALLOW_TEM` in `SEAICE_OPTIONS.h` and turn it on with `SEAICEuseTEM = .TRUE.,` in `data.seaice`. This parameter combination implies the default of `SEAICEcMU = 1.0`.

Instead of an ellipse that is truncated by constant slope coulombic limbs, this yield curve can also be seen as a Mohr-Coulomb yield curve with elliptical yield curve that is truncated for high $P$ by an ellipse. As a consequence, the Mohr-Coulomb slope $SEAICEcMU$ can be set in `data.seaice` to values $\neq 1.0$. This defines a coulombic yield curve similar to the ones shown in Hibler and Schulson (2000) [HS00] and Ringeisen et al. (2019) [RLTNH19].

For this rheology, it is recommended to use a non-zero tensile strength, so set `SEAICE_tensilFac = k_t > 0` in `data.seaice`, e.g., $= 0.05$ or $5\%$.  

8.6. Ice and Sea Ice Packages 525
Mohr-Coulomb yield curve with elliptical plastic potential

To use a Mohr-Coulomb rheology, set \#define SEAICE_ALLOW_MCE in SEAICE_OPTIONS.h and SEAICEuseMCE = .TRUE., in data.seaice. This Mohr-Coulomb yield curve uses an elliptical plastic potential to define the flow rule. The slope of the Mohr-Coulomb yield curve is defined by SEAICEmcMU in data.seaice, and the plastic potential ellipse aspect ratio is set by SEAICE_eccfr in data.seaice. For details of this rheology, see https://doi.org/10.26092/elib/380, Chapter 2.

For this rheology, it is recommended to use a non-zero tensile strength, so set SEAICE_tensilFac > 0 in data.seaice, e.g., = 0.05 or 5%.

Mohr-Coulomb yield curve with shear flow rule

To use the specific Mohr-Coulomb rheology as defined first by Ip et al. (1991) [IHF91], set \#define SEAICE.Allow_MCS in SEAICE_OPTIONS.h and SEAICEuseMCS = .TRUE., in data.seaice. The slope of the Mohr-Coulomb yield curve is defined by SEAICEmcMU in data.seaice. For details of this rheology, including the tensile strength, see https://doi.org/10.26092/elib/380, Chapter 2.

For this rheology, it is recommended to use a non-zero tensile strength, so set SEAICE_tensilFac > 0 in data.seaice, e.g., = 0.05 or 5%.

WARNING: This rheology is known to be unstable. Use with caution!

Teardrop yield curve with normal flow rule

The teardrop rheology was first described in Zhang and Rothrock (2005) [ZR05]. Here we implement a slightly modified version (See https://doi.org/10.26092/elib/380, Chapter 2).

To use this rheology, set \#define SEAICE.Allow_TEARDROP in SEAICE_OPTIONS.h and SEAICEuseTD = .TRUE., in data.seaice. The size of the yield curve can be modified by changing the tensile strength, using SEAICE_tensFac in data.seaice.

For this rheology, it is recommended to use a non-zero tensile strength, so set SEAICE_tensilFac > 0 in data.seaice, e.g., = 0.025 or 2.5%.

Parabolic lens yield curve with normal flow rule

The parabolic lens rheology was first described in Zhang and Rothrock (2005) [ZR05]. Here we implement a slightly modified version (See https://doi.org/10.26092/elib/380, Chapter 2).

To use this rheology, set \#define SEAICE.Allow_TEARDROP in SEAICE_OPTIONS.h and SEAICEusePL = .TRUE., in data.seaice. The size of the yield curve can be modified by changing the tensile strength, using SEAICE_tensFac in data.seaice.

For this rheology, it is recommended to use a non-zero tensile strength, so set SEAICE_tensilFac > 0 in data.seaice, e.g., = 0.025 or 2.5%.
LSR and JFNK solver

In matrix notation, the discretized momentum equations can be written as

$$A(x) \cdot x = b(x). \quad (8.9)$$

The solution vector $x$ consists of the two velocity components $u$ and $v$ that contain the velocity variables at all grid points and at one time level. The standard (and default) method for solving Eq. (8.9) in the sea ice component of MITgcm is an iterative Picard solver: in the $k$-th iteration a linearized form $A(x^{k-1}) \cdot x^k = b(x^{k-1})$ is solved (in the case of MITgcm it is a Line Successive (over) Relaxation (LSR) algorithm). Picard solvers converge slowly, but in practice the iteration is generally terminated after only a few nonlinear steps and the calculation continues with the next time level. This method is the default method in MITgcm. The number of nonlinear iteration steps or pseudo-time steps can be controlled by the run-time parameter SEAICE_JFNKmax. This parameter’s default is 2, but using a number of at least 10 is recommended for better solutions that are converged at least in an energy norm sense (Zhang and Hibler 1997) [ZH97].

In order to overcome the poor convergence of the Picard solver, Lemieux et al. (2010) [LTSedlavcek+10] introduced a Jacobian-free Newton-Krylov solver for the sea ice momentum equations. This solver is also implemented in MITgcm (see Losch et al. 2014 [LFLV14]). The Newton method transforms minimizing the residual $F(x) = A(x) \cdot x - b(x)$ to finding the roots of a multivariate Taylor expansion of the residual $F$ around the previous $(k-1)$ estimate $x^{k-1}$:

$$F(x^{k-1} + \delta x^k) = F(x^{k-1}) + F'(x^{k-1}) \cdot \delta x^k \quad (8.10)$$

with the Jacobian $J \equiv F'$. The root $F(x^{k-1} + \delta x^k) = 0$ is found by solving

$$J(x^{k-1}) \cdot \delta x^k = -F(x^{k-1}) \quad (8.11)$$

for $\delta x^k$. The next $(k$-th) estimate is given by $x^k = x^{k-1} + (1 - \gamma_{LS})^l \cdot \delta x^k$.

By default $l = 0$, but in order to avoid overshoots, the step size factor $(1 - \gamma_{LS})^l$ with $\gamma_{LS} < 1$ can be iteratively reduced in a line search with $l = 0, 1, 2, \ldots$ until $\|F(x^k)\| < \|F(x^{k-1})\|$, where $\|\cdot\| = \int dx^2$ is the $L_2$-norm. The line search starts after SEAICE_JFNK_lsIter nonlinear Newton iterations (off by default) to allow for full Newton steps at the beginning of the iteration. If the line search is turned on by setting SEAICE_JFNK_lsIter to a non-negative value in data.seaice, by default, the line search with $\gamma_{LS} = \frac{1}{2}$ (runtime parameter SEAICE_JFNK_lsGamma) is stopped after $L_{max} = 4$ (runtime parameter SEAICE_JFNK_lsLmax) steps.

Forming the Jacobian $J$ explicitly is often avoided as “too error prone and time consuming”. Instead, Krylov methods only require the action of $J$ on an arbitrary vector $w$ and hence allow a matrix free algorithm for solving (8.11). The action of $J$ can be approximated by a first-order Taylor series expansion:

$$J(x^{k-1}) \cdot w \approx \frac{F(x^{k-1} + \epsilon w) - F(x^{k-1})}{\epsilon} \quad (8.12)$$

or computed exactly with the help of automatic differentiation (AD) tools. SEAICE_JFNKepsilon sets the step size $\epsilon$.

We use the Flexible Generalized Minimum RESidual (FMGRES) method with right-hand side preconditioning to solve (8.11) iteratively starting from a first guess of $\delta x_0^k = 0$. For the preconditioning matrix $P$ we choose a simplified form of the system matrix $A(x^{k-1})$ where $x^{k-1}$ is the estimate of the previous Newton step $k-1$. The transformed equation (8.11) becomes

$$J(x^{k-1}) \cdot P^{-1} \cdot \delta z = -F(x^{k-1}), \quad \text{with} \quad \delta z = P \cdot \delta x^k \quad (8.13)$$

The Krylov method iteratively improves the approximate solution to (8.13) in subspace $(r_0, JP^{-1}r_0, (JP^{-1})^2r_0, \ldots, (JP^{-1})^mr_0)$ with increasing $m$; $r_0 = -F(x^{k-1}) - J(x^{k-1}) \cdot \delta x_0^k$ is the initial residual of (8.11); $r_0 = -F(x^{k-1})$ with the first guess $\delta x_0^k = 0$. We allow a Krylov subspace of dimension $m = 50$ and we do allow restarts for more than 50 Krylov iterations. The preconditioning operation involves applying $P^{-1}$ to the basis vectors $v_0, v_1, v_2, \ldots, v_m$ of the Krylov subspace. This operation is approximated by solving the linear system $P \cdot w = v_i$. Because $P \approx A(x^{k-1})$, the
Elastic-Viscous-Plastic (EVP) Dynamics

Hunke and Dukowicz (1997) [HD97] introduced an elastic contribution to the strain rate in order to regularize (8.3) in such a way that the resulting elastic-viscous-plastic (EVP) and VP models are identical at steady state,

$$\frac{1}{E} \frac{\partial \sigma_{ij}}{\partial t} + \frac{\eta - \zeta}{4\eta} \sigma_{kk} \delta_{ij} + \frac{P}{4\zeta} \delta_{ij} = \dot{\varepsilon}_{ij},$$  (8.15)

The EVP model uses an explicit time stepping scheme with a short timestep. According to the recommendation in Hunke and Dukowicz (1997) [HD97], the EVP-model should be stepped forward in time 120 times (SEAICE_deltaTevp = SEAICE_deltaTdyn /120) within the physical ocean model time step (although this parameter is under debate), to allow for elastic waves to disappear. Because the scheme does not require a matrix inversion it is fast in spite of the small internal timestep and simple to implement on parallel computers. For completeness, we repeat the equations for the components of the stress tensor \(\sigma_1 = \sigma_{11} + \sigma_{22}, \sigma_2 = \sigma_{11} - \sigma_{22},\) and \(\sigma_{12}\). Introducing the divergence \(D_D = \dot{\varepsilon}_{11} + \dot{\varepsilon}_{22}\), and the horizontal tension and shearing strain rates, \(D_T = \dot{\varepsilon}_{11} - \dot{\varepsilon}_{22}\) and \(D_S = 2\dot{\varepsilon}_{12}\), respectively, and using the above abbreviations, the equations (8.15) can be written as:

$$\frac{\partial \sigma_1}{\partial t} + \frac{\sigma_1}{2T} + \frac{P}{2T \Delta} D_D = \frac{P}{2T \Delta} D_D,$$  (8.16)

$$\frac{\partial \sigma_2}{\partial t} + \frac{\sigma_2^2}{2T} = \frac{P}{2T \Delta} D_T,$$  (8.17)

$$\frac{\partial \sigma_{12}}{\partial t} + \frac{\sigma_{12}^2}{2T} = \frac{P}{4T \Delta} D_S,$$  (8.18)

Here, the elastic parameter \(E\) is redefined in terms of a damping timescale \(T\) for elastic waves

$$E = \frac{\zeta}{T}$$
\( T = E_0 \Delta t \) with the tunable parameter \( E_0 < 1 \) and the external (long) time step \( \Delta t \). \( E_0 = \frac{1}{3} \) is the default value in the code and close to what Hunke and Dukowicz (1997) recommend.

We do not recommend to use the EVP solver in its original form. Instead, use mEVP or aEVP instead (see Section 8.6.2.4). If you really need to use the original EVP solver, make sure that both \#define SEAICE_CGRID and \#define SEAICE_ALLOW_EVP are set in SEAICE_OPTIONS.h (both are defined by default). By default, the runtime parameters \texttt{SEAICEuseEVPPstar} and \texttt{SEAICEuseEVPP} are set to \texttt{.FALSE.}. The solver is turned on by setting \texttt{SEAICE_CGRID} to a value larger than zero. The choice of this time step is under debate. Hunke and Dukowicz (1997) \cite{HD97} recommend order 120 time steps within one model time step \( \Delta t \) (deltaTmom). One can also choose order 120 time steps within the forcing time scale, but then we recommend adjusting the damping accordingly, by setting either \texttt{SEAICE_elasticParm} (\( E_0 \)), so that \( E_0 \Delta t = \) forcing time scale, or directly \texttt{SEAICE_evpTauRelax} (\( T \)) to the forcing time scale. \texttt{(NOTE: with the improved EVP variants of the next section, the above recommendations are obsolete. Use mEVP or aEVP instead.)}

**More stable variants of Elastic-Viscous-Plastic Dynamics: EVP*, mEVP, and aEVP**

The genuine EVP scheme appears to give noisy solutions (see Hunke 2001, Lemieux et al. 2012, Bouillon et al. 2013 \cite{BFLM13, Hun01, LKT+12}). This has led to a modified EVP or EVP* (Lemieux et al. 2012, Bouillon et al. 2013, Kimmritz et al. 2015 \cite{BFLM13, KDL15, LKT+12}); here, we refer to these variants by modified EVP (mEVP) and adaptive EVP (aEVP). The main idea is to modify the “natural” time-discretization of the momentum equations:

\[
\frac{D \mathbf{u}}{Dt} \approx m \frac{\mathbf{u}^{p+1} - \mathbf{u}^p}{\Delta t} + \beta \frac{\mathbf{u}^{p+1} - \mathbf{u}^p}{\Delta t_{EVP}} \tag{8.19}
\]

where \( n \) is the previous time step index, and \( p \) is the previous sub-cycling index. The extra “intertial” term \( m (\mathbf{u}^{p+1} - \mathbf{u}^p)/\Delta t \) allows the definition of a residual \(|\mathbf{u}^{p+1} - \mathbf{u}^p|\) that, as \( \mathbf{u}^{p+1} \to \mathbf{u}^{p+1} \), converges to 0. In this way EVP can be re-interpreted as a pure iterative solver where the sub-cycling has no association with time-relation (through \( \Delta t_{EVP} \)). With the setting of \texttt{SEAICEuseEVPPstar} to \texttt{.TRUE.} (default), this form of EVP is used. Using the terminology of Kimmritz et al. 2015 \cite{KDL15}, the evolution equations of stress \( \sigma_{ij} \) and momentum \( \mathbf{u} \) can be written as:

\[
\sigma_{ij}^{p+1} = \sigma_{ij}^p + \frac{1}{\alpha} \left( \sigma_{ij}(\mathbf{u}^p) - \sigma_{ij}^p \right), \tag{8.20}
\]

\[
\mathbf{u}^{p+1} = \mathbf{u}^p + \frac{1}{\beta} \left( \frac{\Delta t}{m} \nabla \cdot \sigma^{p+1} + \frac{\Delta t}{m} \mathbf{R}^p + \mathbf{u}_n - \mathbf{u}^p \right) \tag{8.21}
\]

\( \mathbf{R} \) contains all terms in the momentum equations except for the rheology terms and the time derivative; \( \alpha \) and \( \beta \) are free parameters (\texttt{SEAICE_evpAlpha}, \texttt{SEAICE_evpBeta}) that replace the time stepping parameters \texttt{SEAICE_deltatEvpp} (\( \Delta t_{EVP} \)), \texttt{SEAICE_elasticParm} (\( E_0 \)), or \texttt{SEAICE_evpTauRelax} (\( T \)). \( \alpha \) and \( \beta \) determine the speed of convergence and the stability. Usually, it makes sense to use \( \alpha = \beta \), and \texttt{SEAICEenEvppstarSteps} \( \gg \) (\( \alpha \), \( \beta \)) \cite{KDL15}. Currently, there is no termination criterion and the number of mEVP iterations is fixed to \texttt{SEAICEenEvppstarSteps}.

In order to use mEVP in MITgcm, compile with both \#define \texttt{SEAICE_CGRID} and \#define \texttt{SEAICE_ALLOW_EVP} in SEAICE_OPTIONS.h (default) and make sure that \texttt{SEAICEuseEvppstar} = \texttt{.TRUE.}, \texttt{(default)} in data.seaice. By default \texttt{SEAICEuseEVPP} is set to \texttt{.TRUE.} and the actual form of equations (8.20) and (8.21) is used with fewer implicit terms and the factor of \( c^2 \) dropped in the stress equations (8.17) and (8.18). Although this modifies the original EVP equations, it turns out to improve convergence (Bouillon et al. 2013 \cite{BFLM13}).

The aEVP scheme is an enhanced variant of mEVP (Kimmritz et al. 2016 \cite{KDL16}), where the value of \( \alpha \) is set dynamically based on the stability criterion

\[
\alpha = \beta = \max \left( c \pi \sqrt{c \frac{\zeta}{A_c} \frac{\Delta t}{\text{max}(m, 10^{-4} \text{ kg})}}, \alpha_{\text{min}} \right) \tag{8.22}
\]

**8.6. Ice and Sea Ice Packages**
with the grid cell area $A_c$ and the ice and snow mass $m$. This choice sacrifices speed of convergence for stability with the result that aEVP converges quickly to VP where $\alpha$ can be small and more slowly in areas where the equations are stiff. In practice, aEVP leads to an overall better convergence than mEVP (Kimmritz et al. 2016 [KDL16]). To use aEVP in MITgcm set $SEAICEaEVPcoeff = \hat{c}$ (see (8.22); default is unset); this also sets the default values of $SEAICEaEVPcStar (c = 4)$ and $SEAICEaEVPalphaMin (\alpha_{min} = 5)$. Good convergence has been obtained with these values (Kimmritz et al. 2016 [KDL16]):

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$SEAICEaEVPcoeff$</td>
<td>$0.5$</td>
</tr>
<tr>
<td>$SEAICEaEVPstarSteps$</td>
<td>$500$</td>
</tr>
</tbody>
</table>

Because of the C-grid staggering of velocities and stresses, mEVP may not converge as successfully as in Kimmritz et al. (2015) [KDL15], see also Kimmritz et al. (2016) [KDL16]. Convergence at very high resolution (order 5 km) has not yet been studied.

Ice-Ocean stress

Moving sea ice exerts a stress on the ocean which is the opposite of the stress $\tau_{ocean}$ in (8.2). This stress is applied directly to the surface layer of the ocean model. An alternative ocean stress formulation is given by Hibler and Bryan (1987) [HB87]. Rather than applying $\tau_{ocean}$ directly, the stress is derived from integrating over the ice thickness to the bottom of the oceanic surface layer. In the resulting equation for the combined ocean-ice momentum, the interfacial stress cancels and the total stress appears as the sum of wind stress and divergence of internal ice stresses: $\delta(z)(\tau_{air} + F)/\rho_0$, see also Eq. (2) of Hibler and Bryan (1987) [HB87]. The disadvantage of this formulation is that now the velocity in the surface layer of the ocean that is used to advect tracers, is really an average over the ocean surface velocity and the ice velocity leading to an inconsistency as the ice temperature and salinity are different from the oceanic variables. To turn on the stress formulation of Hibler and Bryan (1987) [HB87], set $useHB87StressCoupling = .TRUE.$, in data.seaice.

Finite-volume discretization of the stress tensor divergence

On an Arakawa C grid, ice thickness and concentration and thus ice strength $P$ and bulk and shear viscosities $\zeta$ and $\eta$ are naturally defined a C-points in the center of the grid cell. Discretization requires only averaging of $\zeta$ and $\eta$ to vorticity or Z-points (or $\zeta$-points, but here we use $\bar{Z}$ in order avoid confusion with the bulk viscosity) at the bottom left corner of the cell to give $\bar{\zeta}$ and $\bar{\eta}$. In the following, the superscripts indicate location at Z or C points, distance across the cell (F), along the cell edge (G), between u-points (U), v-points (V), and C-points (C). The control volumes of the $u$- and $v$-equations in the grid cell at indices $(i, j)$ are $A_{i,j}^u$ and $A_{i,j}^v$, respectively. With these definitions (which follow
the model code documentation except that \( \zeta \)-points have been renamed to Z-points), the strain rates are discretized as:

\[
\begin{align*}
\dot{\varepsilon}_{11} &= \partial_1 u_1 + k_2 u_2 \\
\Rightarrow (\varepsilon_{11})_{i,j}^C &= \frac{u_{i+1,j} - u_{i,j}}{\Delta x_{i,j}^C} + k_2 \frac{u_{i,j+1} + u_{i,j}}{2} \\
\dot{\varepsilon}_{22} &= \partial_2 u_2 + k_1 u_1 \\
\Rightarrow (\varepsilon_{22})_{i,j}^C &= \frac{v_{i,j+1} - v_{i,j}}{\Delta y_{i,j}^C} + k_1 \frac{v_{i,j+1} + v_{i,j}}{2}
\end{align*}
\]

\[
\begin{align*}
\dot{\varepsilon}_{12} &= \dot{\varepsilon}_{21} = \frac{1}{2} \left( \partial_1 u_2 + \partial_2 u_1 - k_1 u_2 - k_2 u_1 \right)
\Rightarrow (\varepsilon_{12})_{i,j}^Z &= \frac{1}{2} \left( \frac{v_{i,j} - v_{i-1,j}}{\Delta x_{i,j}^Z} + \frac{u_{i,j} - u_{i,j-1}}{\Delta y_{i,j}^Z} \\
&\quad - k_1 \frac{v_{i,j} + v_{i-1,j}}{2} - k_2 \frac{u_{i,j} + u_{i,j-1}}{2} \right)
\end{align*}
\]

so that the diagonal terms of the strain rate tensor are naturally defined at C-points and the symmetric off-diagonal term at Z-points. No-slip boundary conditions \((u_{i,j-1} + u_{i,j} = 0)\) and \((v_{i-1,j} + v_{i,j} = 0)\) across boundaries) are implemented via “ghost-points”; for free slip boundary conditions \((\varepsilon_{12})^Z = 0\) on boundaries.

For a spherical polar grid, the coefficients of the metric terms are \(k_1 = 0\) and \(k_2 = -\tan \phi / a\), with the spherical radius \(a\) and the latitude \(\phi\); \(\Delta x_1 = \Delta x = a \cos \phi \Delta \lambda\), and \(\Delta x_2 = \Delta y = a \Delta \phi\). For a general orthogonal curvilinear grid, \(k_1\) and \(k_2\) can be approximated by finite differences of the cell widths:

\[
\begin{align*}
k_{1,i,j}^C &= \frac{1}{\Delta y_{i,j}^C} \frac{\Delta y_{i+1,j}^G - \Delta y_{i,j}^G}{\Delta x_{i,j}^C} \\
k_{2,i,j}^C &= \frac{1}{\Delta x_{i,j}^C} \frac{\Delta x_{i,j+1}^G - \Delta x_{i,j}^G}{\Delta y_{i,j}^C} \\
k_{1,i,j}^Z &= \frac{1}{\Delta y_{i,j}^Z} \frac{\Delta y_{i,j}^C - \Delta y_{i-1,j}^C}{\Delta x_{i,j}^Z} \\
k_{2,i,j}^Z &= \frac{1}{\Delta x_{i,j}^Z} \frac{\Delta x_{i,j}^C - \Delta x_{i,j-1}^C}{\Delta y_{i,j}^Z}
\end{align*}
\]

The stress tensor is given by the constitutive viscous-plastic relation \(\sigma_{\alpha\beta} = 2\eta \dot{\varepsilon}_{\alpha\beta} + [\zeta - \eta] \dot{\varepsilon}_{\gamma\gamma} - P/2 \delta_{\alpha\beta}\). The stress tensor divergence \((\nabla \sigma)_{\alpha} = \partial_{\beta} \sigma_{\alpha\beta}\), is discretized in finite volumes. This conveniently avoids dealing with further metric terms, as these are “hidden” in the differential cell widths. For the \(u\)-equation \((\alpha = 1)\) we have:

\[
(\nabla \sigma)_1 : \frac{1}{A_{i,j}^w} \int_{cell} (\partial_1 \sigma_{11} + \partial_2 \sigma_{21}) \, dx_1 \, dx_2 \\
= \frac{1}{A_{i,j}^w} \left\{ \int_{x_2}^{x_2 + \Delta x_2} \int_{x_1}^{x_1 + \Delta x_1} \sigma_{11} \, dx_2 \, dx_1 + \int_{x_1}^{x_1 + \Delta x_1} \sigma_{21} \, dx_1 \int_{x_2}^{x_2 + \Delta x_2} \right\} \\
\approx \frac{1}{A_{i,j}^w} \left\{ \Delta x_2 \sigma_{11} \left|_{x_1}^{x_1 + \Delta x_1} \right. + \Delta x_1 \sigma_{21} \left|_{x_2}^{x_2 + \Delta x_2} \right. \right\} \\
= \frac{1}{A_{i,j}^w} \left\{ (\Delta x_2 \sigma_{11})_{i,j} - (\Delta x_2 \sigma_{11})_{i-1,j} \\
+ (\Delta x_1 \sigma_{21})_{i,j+1} - (\Delta x_1 \sigma_{21})_{i,j} \right\}
\]
\[
\begin{align*}
(\Delta x_2 \sigma_{12})^{C}_{i,j} &= \Delta y_{i,j}^F (\zeta + \eta)_{i,j}^\sigma \frac{u_{i+1,j} - u_{i,j}}{\Delta x_{i,j}^F} \\
&\quad + \Delta y_{i,j}^F (\zeta + \eta)_{i,j}^\sigma k_{2,i,j}^F \frac{v_{i,j+1} + v_{i,j}}{2} \\
&\quad + \Delta y_{i,j}^F (\zeta - \eta)_{i,j}^\sigma \frac{v_{i,j+1} - v_{i,j}}{\Delta y_{i,j}^F} \\
&\quad + \Delta y_{i,j}^F (\zeta - \eta)_{i,j}^\sigma k_{1,i,j}^F \frac{u_{i+1,j} + u_{i,j}}{2} \\
&\quad - \Delta y_{i,j}^F \frac{P}{2}
\end{align*}
\]

Similarly, we have for the \(v\)-equation \((\alpha = 2)\):

\[
(\nabla \sigma)_v = \frac{1}{A_{i,j}} \int_{\text{cell}} (\partial_1 \sigma_{12} + \partial_2 \sigma_{22}) \, dx_1 \, dx_2
\]

with

\[
(\Delta x_1 \sigma_{12})^{Z}_{i,j} = \Delta y_{i,j}^U \pi_{i,j}^Z \frac{u_{i,j} - u_{i,j-1}}{\Delta y_{i,j}^U} + \Delta y_{i,j}^U \pi_{i,j}^Z \frac{v_{i,j} - v_{i,j-1}}{\Delta x_{i,j}^U}
\]

\[
(\Delta x_2 \sigma_{22})^{C}_{i,j} = \Delta x_{i,j}^F (\zeta - \eta)_{i,j}^\sigma \frac{u_{i+1,j} - u_{i,j}}{\Delta x_{i,j}^F} \\
+ \Delta x_{i,j}^F (\zeta - \eta)_{i,j}^\sigma k_{2,i,j}^F \frac{v_{i,j+1} + v_{i,j}}{2} \\
+ \Delta x_{i,j}^F (\zeta + \eta)_{i,j}^\sigma \frac{v_{i,j+1} - v_{i,j}}{\Delta y_{i,j}^F} \\
+ \Delta x_{i,j}^F (\zeta + \eta)_{i,j}^\sigma k_{1,i,j}^F \frac{u_{i+1,j} + u_{i,j}}{2} \\
- \Delta x_{i,j}^F \frac{P}{2}
\]
Again, no-slip boundary conditions are realized via ghost points and $u_{i,j-1} + u_{i,j} = 0$ and $v_{i-1,j} + v_{i,j} = 0$ across boundaries. For free-slip boundary conditions the lateral stress is set to zeros. In analogy to $(\epsilon_{12})^2 = 0$ on boundaries, we set $\sigma_{\epsilon_{12}}^2 = 0$, or equivalently $\eta_{\epsilon_{12}}^2 = 0$, on boundaries.

**Thermodynamics**

**NOTE: THIS SECTION IS STILL NOT COMPLETE**

In its original formulation the sea ice model uses simple 0-layer thermodynamics following the appendix of Semtner (1976) [Sem76]. This formulation neglects storage of heat, that is, the heat capacity of ice is zero, and all internal heat sources so that the heat equation reduces to a constant conductive heat flux. This constant upward conductive heat flux together with a constant ice conductivity implies a linear temperature profile. The boundary conditions for the heat equations are: at the bottom of the ice $T_{b} = T_{fr}$ (freezing point temperature of sea water), and at the surface: $Q_{top} = \frac{\eta}{\sigma_{\epsilon} T_{b}} = (K/h)(T_0 - T_{fr})$, where $K$ is the ice conductivity, $h$ the ice thickness, and $T_0 - T_{fr}$ the difference between the ice surface temperature and the water temperature at the bottom of the ice (at the freezing point). The surface heat flux $Q_{top}$ is computed in a similar way to that of Parkinson and Washington (1979) [PW79] and Manabe et al. (1979) [MBS79]. The resulting equation for surface temperature is

\[ \frac{K}{h}(T_0 - T_{fr}) = Q_{SW}(1 - \text{albedo}) + \epsilon Q_{LW} - Q_{LW'}(T_0) + Q_{LH}(T_0) + Q_{SH}(T_0), \]

(8.23)

where $\epsilon$ is the emissivity of the surface (snow or ice), $Q_{SW}$ the downwelling shortwave and longwave radiation to be prescribed, and $Q_{LW'} = \epsilon \sigma_B T_0^{4}$ the emitted long wave radiation with the Stefan-Boltzmann constant $\sigma_B$. With explicit expressions in $T_0$ for the turbulent fluxes of latent and sensible heat

\[ Q_{LH} = \rho_{air} C_E (\Lambda_v + \Lambda_f) |U_{air}| (q_{air} - q_{sat}(T_0)) \]

\[ Q_{SH} = \rho_{air} c_p C_E |U_{air}| [T_{10m} - T_0], \]

(8.23) can be solved for math:$T_0$ with an iterative :Ralphson-Newton method, which usually converges very quickly in less than 10 :iterations. In these equations, $\rho_{air}$ is the air density :parameter SEAICE_rhoeAir, $C_E$ is the ice-ocean transfer :coefficient for sensible and latent heat :parameter SEAICE_dalton, $\Lambda_v$ and $\Lambda_f$ are the latent heat of vaporization and :fusion, respectively :parameters SEAICE_lhEvap and :SEAICE_lhFusion, and $c_p$ is the specific heat of air :parameter SEAICE_cpAir. For the latent heat $Q_{LH}$ an :choice can be made between the old polynomial expression for saturation :humidity $q_{sat}(T_0)$ (by setting :useMaykutSatVapPoly to .TRUE.) and the default exponential :relation approximation that is more accurate at low temperatures.

In the zero-layer model of Semtner (1976) [Sem76], the conductive heat flux depends strongly on the ice thickness $h$. However, the ice thickness in the model represents a mean over a potentially very heterogeneous thickness distribution. In order to parameterize a sub-grid scale distribution for heat flux computations, the mean ice thickness $h$ is split into $N$ thickness categories $H_n$ that are equally distributed between $2h$ and a minimum imposed ice thickness of 5 cm by $H_n = \frac{2n-1}{2} h$ for $n \in [1, N]$. The heat fluxes computed for each thickness category are area-averaged to give the total heat flux (see Hibler 1984 [Hib84]). To use this thickness category parameterization set SEAICE_multDim to the number of desired categories in data.seaice (7 is a good guess, for anything larger than 7 modify SEAICE_SIZE.h). Note that this requires different restart files and switching this flag on in the middle of an integration is not advised. As an alternative to the flat distribution, the run-time parameter SEAICE_pdf (1D-array of length nITD) can be used to prescribe an arbitrary distribution of ice thicknesses, for example derived from observed distributions (Castro-Morales et al. 2014 [CMKL+14]). In order to include the ice thickness distribution also for snow, set SEAICE_useMultDimSnow to .TRUE. (this is the default); only then, the parameterization of always having a fraction of thin ice is efficient and generally thicker ice is produced (see Castro-Morales et al. 2014 [CMKL+14]).

The atmospheric heat flux is balanced by an oceanic heat flux from below. The oceanic flux is proportional to $\rho c_p (T_w - T_{fr})$ where $\rho$ and $c_p$ are the density and heat capacity of sea water and $T_{fr}$ is the local freezing point temperature that is a function of salinity. This flux is not assumed to instantaneously melt or create ice, but a time scale
of three days (run-time parameter SEAICE_gamma_t) is used to relax $T_w$ to the freezing point. The parameterization of lateral and vertical growth of sea ice follows that of Hibler (1979) and Hibler (1980) [Hib79, Hib80]; the so-called lead closing parameter $h_0$ (run-time parameter HO) has a default value of 0.5 meters.

On top of the ice there is a layer of snow that modifies the heat flux and the albedo (Zhang et al. 1998 [ZWDHSR98]). Snow modifies the effective conductivity according to

$$
\frac{K}{h} \rightarrow \frac{h_s}{K_s} + \frac{h}{K},
$$

where $K_s$ is the conductivity of snow and $h_s$ the snow thickness. If enough snow accumulates so that its weight submerges the ice and the snow is flooded, a simple mass conserving parameterization of snowice formation (a flood-freeze algorithm following Archimedes’ principle) turns snow into ice until the ice surface is back at $z = 0$ (see Leppäranta 1983 [Lepparanta83]). The flood-freeze algorithm is turned on with run-time parameter SEAICEuseFloodng set to .TRUE..

### Advection of thermodynamic variables

Effective ice thickness (ice volume per unit area, $c \cdot h$), concentration $c$ and effective snow thickness ($c \cdot h_s$) are advected by ice velocities:

$$
\frac{\partial X}{\partial t} = -\nabla \cdot (u \cdot X) + \Gamma_X + D_X
$$

where $\Gamma_X$ are the thermodynamic source terms and $D_X$ the diffusive terms for quantities $X = (c \cdot h), c, (c \cdot h_s)$. From the various advection schemes that are available in MITgcm, we recommend flux-limited schemes to preserve sharp gradients and edges that are typical of sea ice distributions and to rule out unphysical over- and undershoots (negative thickness or concentration). These schemes conserve volume and horizontal area and are unconditionally stable, so that we can set $D_X = 0$. Run-time flags: SEAICEadvScheme (default=77, is a 2nd-order flux limited scheme), DIFF1 = $D_X/\Delta x$ (default=0).

The MITgcm sea ice model provides the option to use the thermodynamics model of Winton (2000) [Win00], which in turn is based on the 3-layer model of Semtner (1976) [Sem76] which treats brine content by means of enthalpy conservation; the corresponding package thsice is described in section Section 8.6.1. This scheme requires additional state variables, namely the enthalpy of the two ice layers (instead of effective ice salinity), to be advected by ice velocities. The internal sea ice temperature is inferred from ice enthalpy. To avoid unphysical (negative) values for thickness and concentration, a positive 2nd-order advection scheme with a SuperBee flux limiter (Roe 1985 [Roe85]) should be used to advect all sea-ice-related quantities of the Winton (2000) [Win00] thermodynamic model (run-time flag thSIceAdvScheme = 77 and thSIce_diffK = $D_X = 0$ in data.ice, defaults are 0). Because of the nonlinearity of the advection scheme, care must be taken in advecting these quantities: when simply using ice velocity to advect enthalpy, the total energy (i.e., the volume integral of enthalpy) is not conserved. Alternatively, one can advect the energy content (i.e., product of ice-volume and enthalpy) but then false enthalpy extrema can occur, which then leads to unrealistic ice temperature. In the currently implemented solution, the sea-ice mass flux is used to advect the enthalpy in order to ensure conservation of enthalpy and to prevent false enthalpy extrema.

### Dynamical Ice Thickness Distribution (ITD)

The ice thickness distribution model used by MITgcm follows the implementation in the Los Alamos sea ice model CICE (https://github.com/CICE-Consortium/CICE). There are two parts to it that are closely connected: the participation and ridging functions that determine which thickness classes take part in ridging and which thickness classes receive ice during ridging based on Thorndike et al. (1975) [TRMC75], and the ice strength parameterization by Rothrock (1975) [Rot75] which uses this information. The following description is slightly modified from Ungermann et al. (2017) [UTML17]. Verification experiment seaice_itd uses the ITD model.
Distribution, participation and redistribution functions in ridging

When `SEAICE_ITD` is defined in `SEAICE_OPTIONS.h`, the ice thickness is described by the ice thickness distribution \( g(h, x, t) \) for the subgrid-scale (see Thorndike et al. 1975 [TRMC75]), a probability density function for thickness \( h \) following the evolution equation

\[
\frac{\partial g}{\partial t} = -\nabla \cdot (u g) - \frac{\partial}{\partial h} (fg) + \Psi.
\] (8.25)

Here \( f = \frac{dh}{dt} \) is the thermodynamic growth rate and \( \Psi \) a function describing the mechanical redistribution of sea ice during ridging or lead opening.

The mechanical redistribution function \( \Psi \) generates open water in divergent motion and creates ridged ice during convergent motion. The ridging process depends on total strain rate and on the ratio between shear (run-time parameter `SEAICEshearParm`) and divergent strain. In the single category model, ridge formation is treated implicitly by limiting the ice concentration to a maximum of one (see Hibler 1979 [Hib79]), so that further volume increase in convergent motion leads to thicker ice. (This is also the default for ITD models; to change from the default, set run-time parameter `SEAICEsimpleRidging = .FALSE., in data.seaice`). For the ITD model, the ridging mode in convergence

\[
\omega_r(h) = \frac{-a(h) + n(h)}{N}
\]

gives the effective change for the ice volume with thickness between \( h \) and \( h + dh \) as the normalized difference between the ice \( n(h) \) generated by ridging and the ice \( a(h) \) participating in ridging.

The participation function \( a(h) = b(h)g(h) \) can be computed either following Thorndike et al. (1975) [TRMC75] (run-time parameter `SEAICEpartFunc =0`) or Lipscomb et al. (2007) [LHMJ07] (`SEAICEpartFunc =1`), and similarly the ridging function \( n(h) \) can be computed following Hilber (1980) [Hib80] (run-time parameter `SEAICEredistFunc =0`) or Lipscomb et al. (2007) [LHMJ07] (`SEAICEredistFunc =1`). As an example, we show here the functions that Lipscomb et al. (2007) [LHMJ07] suggested to avoid noise in the solutions. These functions are smooth and avoid non-differentiable discontinuities, but so far we did not find any noise issues as in Lipscomb et al. (2007) [LHMJ07].

With `SEAICEpartFunc = 1`, in `data.seaice`, the participation function with the relative amount of ice of thickness \( h \) weighted by an exponential function

\[
b(h) = b_0 \exp[-G(h)/a^*]
\]

where \( G(h) = \int_0^h g(h)dh \) is the cumulative thickness distribution function, \( b_0 \) a normalization factor, and \( a^* \) (SEAICEaStar) the exponential constant that determines which relative amount of thicker and thinner ice take part in ridging.

With `SEAICEredistFunc = 1`, in `data.seaice`, the ice generated by ridging is calculated as

\[
n(h) = \int_0^\infty a(h_1)\gamma(h_1, h)dh_1
\]

where the density function \( \gamma(h_1, h) \) of resulting thickness \( h \) for ridged ice with an original thickness of \( h_1 \) is taken as

\[
\gamma(h_1, h) = \frac{1}{k\lambda} \exp \left[ \frac{-(h - h_{\text{min}})}{\lambda} \right]
\]

for \( h \geq h_{\text{min}} \), with \( \gamma(h_1, h) = 0 \) for \( h < h_{\text{min}} \). In this parameterization, the normalization factor \( k = h_{\text{min}} + \lambda \), the e-folding scale \( \lambda = \mu h_1^{1/2} \) and the minimum ridge thickness \( h_{\text{min}} = \min(2h_1, h_1 + h_{\text{raft}}) \) all depend on the original thickness \( h_1 \). The maximal ice thickness allowed to raft \( h_{\text{raft}} \) is constant (SEAICEmaxRaft, default =1 m) and \( \mu \) (SEAICEmuRidging) is a tunable parameter.

In the numerical model these equations are discretized into a set of \( n \) (nITD defined in `SEAICE_SIZE.h`) thickness categories employing the delta function scheme of Bitz et al. (2001) [BHWE01]. For each thickness category in an
ITD configuration, the volume conservation equation (8.24) is evaluated using the heat flux with the category-specific values for ice and snow thickness, so there are no conceptual differences in the thermodynamics between the single category and ITD configurations. The only difference is that in the thinnest category the creation of new ice of thickness $H_0$ (run-time parameter $H_0$) is possible, all other categories are limited to basal growth. The conservation of ice area is replaced by the evolution equation of the ITD (8.25) that is discretized in thickness space with $n + 1$ category limits given by run-time parameter $H_{\text{limit}}$. If $H_{\text{limit}}$ is not set in data.seaice, a simple recursive formula following Lipscomb (2001) [Lip01] is used to compute $H_{\text{limit}}$:

$$H_{\text{limit}}(k) = H_{\text{limit}}(k - 1) + \frac{c_1}{n} + \frac{c_1 c_2}{n} \left[ 1 + \tanh \left( \frac{k - 1}{n} - 1 \right) \right]$$

with $H_{\text{limit}}(0) = 0$ m and $H_{\text{limit}}(n) = 999.9$ m. The three constants are the run-time parameters $H_{\text{limit}_c1}$, $H_{\text{limit}_c2}$, and $H_{\text{limit}_c3}$. The total ice concentration and volume can then be calculated by summing up the values for each category.

**Ice strength parameterization**

In the default approach of equation (8.4), the ice strength is parameterized following Hibler (1979) [Hib79] and $P$ depends only on average ice concentration and thickness per grid cell and the constant ice strength parameters $P^*$ (SEAICE_strength) and $C^*$ (SEAICE_cStar). With an ice thickness distribution, it is possible to use a different parameterization following Rothrock (1975) [Rot75]

$$P = C_f C_p \int_0^\infty h^2 \omega_r(h) dh$$

(8.26)

by considering the production of potential energy and the frictional energy loss in ridging. The physical constant $C_p = \rho_i (\rho_w - \rho_i) g / (2 \rho_w)$ is a combination of the gravitational acceleration $g$ and the densities $\rho_i$, $\rho_w$ of ice and water, and $C_f$ (SEAICE_cf) is a scaling factor relating the amount of work against gravity necessary for ridging to the amount of work against friction. To calculate the integral, this parameterization needs information about the ITD in each grid cell, while the default parameterization (8.4) can be used for both ITD and single thickness category models. In contrast to (8.4), which is based on the plausible assumption that thick and compact ice is stronger than thin and loose drifting ice, this parameterization (8.26) clearly contains the more physical assumptions about energy conservation. For that reason alone this parameterization is often considered to be more physically realistic than (8.4), but in practice, the success is not so clear (Ungermann et al. 2007 [UTML17]). Ergo, the default is to use (8.4); set useHibler79IceStrength = .FALSE., in data.seaice to change this behavior.

### 8.6.2.5 Known issues and work-arounds

- An often encountered problem in long simulations with sea ice models is (local) perpetually increasing sea ice (plus snow) height; this is problematic when using a non-linear free surface and useRealFreshWaterFlux set to .TRUE., because the mass of the sea ice places a load on the sea surface, which if too large, can cause the surface cells of the model to become too thin so that the model eventually stops with an error message. Usually this problem occurs because of dynamical ice growth (i.e., convergence and ridging of ice) or simply too much net precipitation with insufficient summer surface melting. If the problem is dynamical in nature (e.g., caused by ridging in a deep inlet), the first step to try is to turn off the replacement pressure method (SEAICEpressReplFac = 0; in Section 8.6.2.4); turning this off provides resistance against additional growth due to further ridging, because the ice pressure $P$ is no longer reduced as $\Delta \to 0$ in nearly motionless thick ice (8.5). If this does not solve the problem, a somewhat more radical yet effective approach is simply to cap the sea ice load on the free surface by defining the CPP option SEAICE_CAP_ICELOAD. This option effectively limits the sea ice load (variable sliceLoad) to a mass of 1/5 of the the top grid cell depth. If desired, this limit can be changed in routine seaice_growth.F where variable heffTooHeavy is assigned.
8.6.2.6 Key subroutines

Top-level routine: pkg/seaice/seaice_model.F

```c
!
CALLING SEQUENCE:
...
seaice_model (TOP LEVEL ROUTINE)
|
|-- #ifdef SEAICE_CGRID
|-- SEAICE_DYNSOLVER
| | | |-- < compute proxy for geostrophic velocity >
| | | | |-- < set up mass per unit area and Coriolis terms >
| | | | |-- < dynamic masking of areas with no ice >
| | | #ELSE
| | | DYNSOLVER
| | #ENDIF
| |-- if ( useOBCS )
| | | OBCS_APPLY_UVICE
| |-- if ( SEAICEadvHeff .OR. SEAICEadvArea .OR. SEAICEadvSnow .OR. SEAICEadvSalt )
| | | SEAICE_ADVDIFF
| | | | SEAICE_REG_RIDGE
| | |-- if ( usePW79thermodynamics )
| | | SEAICE_GROWTH
| | |-- if ( useOBCS )
| | | if ( SEAICEadvHeff ) OBCS_APPLY_HEFF
| | | if ( SEAICEadvArea ) OBCS_APPLY_AREA
| | | if ( SEAICEadvSALT ) OBCS_APPLY_HSALT
| | | if ( SEAICEadvSNOW ) OBCS_APPLY_HSNOW
| | |-- < do various exchanges >
| | |-- < do additional diagnostics >
```
### 8.6.2.7 SEAICE diagnostics

Diagnostics output is available via the diagnostics package (see Section 9.1). Available output fields are summarized in the following table:

<table>
<thead>
<tr>
<th>Name</th>
<th>Grid</th>
<th>Units</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sIceLoad</td>
<td>SM</td>
<td>U1</td>
<td>kg/m²</td>
</tr>
</tbody>
</table>

#### SEA ICE STATE:

<table>
<thead>
<tr>
<th>Name</th>
<th>Grid</th>
<th>Units</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIarea</td>
<td>SM</td>
<td>M1</td>
<td>m²/m²</td>
</tr>
<tr>
<td>SIheff</td>
<td>SM</td>
<td>M1</td>
<td>m</td>
</tr>
<tr>
<td>SIhsnow</td>
<td>SM</td>
<td>M1</td>
<td>m</td>
</tr>
<tr>
<td>SIhsalt</td>
<td>SM</td>
<td>M1</td>
<td>g/m²</td>
</tr>
<tr>
<td>SIuice</td>
<td>UU</td>
<td>M1</td>
<td>m/s</td>
</tr>
<tr>
<td>SIvice</td>
<td>VV</td>
<td>M1</td>
<td>m/s</td>
</tr>
</tbody>
</table>

#### ATMOSPHERIC STATE AS SEEN BY SEA ICE:

<table>
<thead>
<tr>
<th>Name</th>
<th>Grid</th>
<th>Units</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SItices</td>
<td>SM</td>
<td>C</td>
<td>M1/K</td>
</tr>
<tr>
<td>SIuwind</td>
<td>UM</td>
<td>U1</td>
<td>m/s</td>
</tr>
<tr>
<td>SIvwind</td>
<td>VM</td>
<td>U1</td>
<td>m/s</td>
</tr>
<tr>
<td>_uVel</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SIsmPrcp</td>
<td>SM</td>
<td>U1</td>
<td>kg/m²</td>
</tr>
</tbody>
</table>

#### FLUXES ACROSS ICE-OCEAN INTERFACE (ATMOS to OCEAN FOR ICE-FREE REGIONS):

<table>
<thead>
<tr>
<th>Name</th>
<th>Grid</th>
<th>Units</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIfu</td>
<td>UU</td>
<td>U1</td>
<td>N/m²</td>
</tr>
<tr>
<td>SIfv</td>
<td>VV</td>
<td>U1</td>
<td>N/m²</td>
</tr>
<tr>
<td>_vVel</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SIqnet</td>
<td>SM</td>
<td>U1</td>
<td>W/m²</td>
</tr>
<tr>
<td>SIqsw</td>
<td>SM</td>
<td>U1</td>
<td>W/m²</td>
</tr>
<tr>
<td>SIempmr</td>
<td>SM</td>
<td>U1</td>
<td>kg/m²</td>
</tr>
<tr>
<td>SIqneto</td>
<td>SM</td>
<td>U1</td>
<td>W/m²</td>
</tr>
<tr>
<td>SIqneti</td>
<td>SM</td>
<td>U1</td>
<td>W/m²</td>
</tr>
</tbody>
</table>

#### FLUXES ACROSS ATMOSPHERE-ICE INTERFACE (ATMOS to OCEAN FOR ICE-FREE REGIONS):

<table>
<thead>
<tr>
<th>Name</th>
<th>Grid</th>
<th>Units</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIatmQnt</td>
<td>SM</td>
<td>U1</td>
<td>W/m²</td>
</tr>
<tr>
<td>SIatmFW</td>
<td>SM</td>
<td>U1</td>
<td>kg/m²</td>
</tr>
<tr>
<td>SIfwSubl</td>
<td>SM</td>
<td>U1</td>
<td>kg/m²</td>
</tr>
</tbody>
</table>

#### THERMODYNAMIC DIAGNOSTICS:

<table>
<thead>
<tr>
<th>Name</th>
<th>Grid</th>
<th>Units</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIareaPR</td>
<td>SM</td>
<td>M1</td>
<td>m²/m²</td>
</tr>
<tr>
<td>SIareaPT</td>
<td>SM</td>
<td>M1</td>
<td>m²/m²</td>
</tr>
<tr>
<td>SIheffPT</td>
<td>SM</td>
<td>M1</td>
<td>m</td>
</tr>
<tr>
<td>SIhsnoPT</td>
<td>SM</td>
<td>M1</td>
<td>m</td>
</tr>
<tr>
<td>SIAqB0CN</td>
<td>SM</td>
<td>M1</td>
<td>m/s</td>
</tr>
<tr>
<td>SIAqBATC</td>
<td>SM</td>
<td>M1</td>
<td>m/s</td>
</tr>
</tbody>
</table>
### 8.6. Ice and Sea Ice Packages

| SIaQbATO|SM | M1|m/s | Potential HEFF rate of change by open ocn atm flux |
| SIdHbOCN|SM | M1|m/s | HEFF rate of change by ocean ice flux |
| SIdSbATC|SM | M1|m/s | HSNOW rate of change by atm flux over sea ice |
| SIdSbOCN|SM | M1|m/s | HSNOW rate of change by ocean ice flux |
| SIdHbATC|SM | M1|m/s | HEFF rate of change by atm flux over sea ice |
| SIdHbATO|SM | M1|m/s | HEFF rate of change by open ocn atm flux |
| SIdHbFLO|SM | M1|m/s | HEFF rate of change by flooding snow |
| SIdAbATO|SM | M1|m^2/m^2/s | Potential AREA rate of change by open ocn atm flux |
| SIdAbATC|SM | M1|m^2/m^2/s | Potential AREA rate of change by atm flux over sea ice |
| SIdAbOCN|SM | M1|m^2/m^2/s | Potential AREA rate of change by ocean ice flux |
| SIdA |SM | M1|m^2/m^2/s | AREA rate of change (net) |

#### DYNAMIC/RHEOLOGY DIAGNOSTICS:

| SIpress |SM | M1|N/m | SEAICE strength (with upper and lower limit) |
| SIZeta |SM | M1|kg/s | SEAICE nonlinear bulk viscosity |
| SIeta |SM | M1|kg/s | SEAICE nonlinear shear viscosity |
| SISig1 |SM | M1|no units | SEAICE normalized principle stress, component one |
| SISig2 |SM | M1|no units | SEAICE normalized principle stress, component two |
| SIShear |SM | M1|1/s | SEAICE shear deformation rate |
| SISDelta |SM | M1|1/s | SEAICE Delta deformation rate |
| SItensil |SM | M1|N/m | SEAICE maximal tensile strength |

#### ADVECTIVE/DIFFUSIVE FLUXES OF SEA ICE variables:

| ADVxHEFF|UU | M1|m.m^2/s | Zonal Advevtive Flux of eff ice thickn |
| ADVyHEFF|VV | M1|m.m^2/s | Meridional Advevtive Flux of eff ice thickn |
| SIVheff|UU | M1|m^2/s | Zonal Transport of eff ice thickn (centered) |
| SIVheff|VV | M1|m^2/s | Meridional Transport of eff ice thickn (centered) |
| DFxEHEFF|UU | M1|m^2/s | Zonal Diffusive Flux of eff ice thickn |
| DFyEHEFF|VV | M1|m^2/s | Meridional Diffusive Flux of eff ice thickn |
| ADVxAREA|UU | M1|m.^2/m.^2/s | Zonal Advevtive Flux of fract area |
| ADVyAREA|VV | M1|m.^2/m.^2/s | Meridional Advevtive Flux of fract area |
| DFxEAREA|UU | M1|m.^2/m.^2/s | Zonal Diffusive Flux of fract area |
| DFyEAREA|VV | M1|m.^2/m.^2/s | Meridional Diffusive Flux of fract area |
| ADVxSNOW|UU | M1|m.m^2/s | Zonal Advevtive Flux of eff snow thickn |
| ADVySNOW|VV | M1|m.m^2/s | Meridional Advevtive Flux of eff snow thickn |
| DFxEESNOW|UU | M1|m.m^2/s | Zonal Diffusive Flux of eff snow thickn |
| DFyEESNOW|VV | M1|m.m^2/s | Meridional Diffusive Flux of eff snow thickn |
| ADVxSSLT|UU | M1|(g/kg).m^2/s | Zonal Advevtive Flux of seaice salinity |
| ADVySSLT|VV | M1|(g/kg).m^2/s | Meridional Advevtive Flux of seaice salinity |
| DFxESSLT|UU | M1|(g/kg).m^2/s | Zonal Diffusive Flux of seaice salinity |
| DFyESSLT|VV | M1|(g/kg).m^2/s | Meridional Diffusive Flux of seaice salinity |
8.6.2.8 Experiments and tutorials that use seaice

- verification/lab_sea: Labrador Sea experiment
- verification/seaice_obcs, based on lab_sea
- verification/online_exf_seaice, idealized topography in a zonally re-entrant channel, tests solvers and rheologies
- verification/seaice_itd, based on offline_exf_seaice, tests ice thickness distribution
- verification/global_ocean.cs32x15, global cubed-sphere-experiment with combinations of pkg/seaice and pkg/thsice
- verification/1D_ocean_ice_column, just thermodynamics

8.6.3 SHELFICE Package

Authors: Martin Losch, Jean-Michel Campin

8.6.3.1 Introduction

pkg/shelfice provides a thermodynamic model for basal melting underneath floating ice shelves.

CPP options enable or disable different aspects of the package (Section 8.6.3.2). Run-time options, flags, filenames and field-related dates/times are described in Section 8.6.3.3. A description of key subroutines is given in Section 8.6.3.5. Available diagnostics output is listed in Section 8.6.3.6.

8.6.3.2 SHELFICE configuration

As with all MITgcm packages, pkg/shelfice can be turned on or off at compile time:

- using the packages.conf file by adding shelfice to it,
- or using genmake2 adding -enable=shelfice or disable=shelfice switches

pkg/shelfice does not require any additional packages, but it will only work with conventional vertical z-coordinates (pressure coordinates are not implemented). If you use it together with vertical mixing schemes, be aware that non-local parameterizations are turned off, e.g., such as pkg/kpp.

Parts of the pkg/shelfice code can be enabled or disabled at compile time via CPP preprocessor flags. These options are set in SHELFICE_OPTIONS.h:

<table>
<thead>
<tr>
<th>Table 8.20: Compile-time parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPP Flag Name</td>
</tr>
<tr>
<td>------------------------</td>
</tr>
<tr>
<td>ALLOW_SHELFICE_DEBUG</td>
</tr>
<tr>
<td>ALLOW_ISOMIP_TD</td>
</tr>
<tr>
<td>SHI_ALLOW_GAMMAFRICT</td>
</tr>
</tbody>
</table>
8.6.3.3 SHELFICE run-time parameters

pkg/shelfice is switched on/off at run time by setting useSHELFICE to .TRUE. in file data.pkg. Run-time parameters are set in file data.shelfice (read in pkg/shelfice/shelfice_readparms.F), as listed below.

The data file specifying under-ice topography of ice shelves (SHELFICEtopoFile) is in meters; upwards is positive, and as for the bathymetry files, negative values are required for topography below the sea-level. The data file for the pressure load anomaly at the bottom of the ice shelves SHELFICEloadAnomalyFile is in pressure units (Pa). This field is absolutely required to avoid large excursions of the free surface during initial adjustment processes, obtained by integrating an approximate density from the surface at \( z = 0 \) down to the bottom of the last fully dry cell within the ice shelf, see (8.30). Note however the file SHELFICEloadAnomalyFile must not be \( p_{\text{top}} \), but \( p_{\text{top}} - g \sum_{k'=1}^{n-1} \rho_c \Delta z_{k'} \), with \( \rho_c = \text{rhoConst} \), so that in the absence of a \( \rho^* \) that is different from \( \rho_c \), the anomaly is zero.

Table 8.21: Run-time parameters and default values; all parameters are in namelist group SHELFICE_PARM01

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>useISOMIPTD</td>
<td>FALSE</td>
<td>use simplified ISOMIP thermodynamics on/off flag</td>
</tr>
<tr>
<td>SHELFICEconserve</td>
<td>FALSE</td>
<td>use conservative form of temperature boundary conditions on/off flag</td>
</tr>
<tr>
<td>SHELFICEboundaryLayer</td>
<td>FALSE</td>
<td>use simple boundary layer mixing parameterization on/off flag</td>
</tr>
<tr>
<td>SHI_withBL_realFWflux</td>
<td>FALSE</td>
<td>with SHELFICEboundaryLayer, allow to use real-FW flux</td>
</tr>
<tr>
<td>SHELFICEloadAnomalyFile</td>
<td>' '</td>
<td>initial geopotential anomaly</td>
</tr>
<tr>
<td>SHELFICEtopoFile</td>
<td>' '</td>
<td>filename for under-ice topography of ice shelves</td>
</tr>
<tr>
<td>SHELFICEmassFile</td>
<td>' '</td>
<td>filename for mass of ice shelves</td>
</tr>
<tr>
<td>SHELFICEMassDynTendFile</td>
<td>' '</td>
<td>filename for mass tendency of ice shelves</td>
</tr>
<tr>
<td>SHELFICETransCoeffTFile</td>
<td>' '</td>
<td>filename for spatially varying transfer coefficients</td>
</tr>
<tr>
<td>SHELFICElatentHeat</td>
<td>334.0E+03</td>
<td>latent heat of fusion (J/kg)</td>
</tr>
<tr>
<td>SHELFICEHeatCapacity_Cp</td>
<td>2000.0E+00</td>
<td>specific heat capacity of ice (J/kg/K)</td>
</tr>
<tr>
<td>rhoShelfIce</td>
<td>917.0E+00</td>
<td>(constant) mean density of ice shelf (kg/m³)</td>
</tr>
<tr>
<td>SHELFICEheatTransCoeff</td>
<td>1.0E-04</td>
<td>transfer coefficient (exchange velocity) for temperature (m/s)</td>
</tr>
<tr>
<td>SHELFICEsaltTransCoeff</td>
<td>SHELFICEsaltToHeatRatio * SHELFICEheatTransCoeff</td>
<td>transfer coefficient (exchange velocity) for salinity (m/s)</td>
</tr>
<tr>
<td>SHELFICEsaltToHeatRatio</td>
<td>5.05E-03</td>
<td>ratio of salinity to temperature transfer coefficients (non-dim.)</td>
</tr>
<tr>
<td>SHELFICEkappa</td>
<td>1.54E-06</td>
<td>temperature diffusion coefficient of the ice shelf (m²/s)</td>
</tr>
<tr>
<td>SHELFICEthetaSurface</td>
<td>-20.0E+00</td>
<td>(constant) surface temperature above the ice shelf (°C)</td>
</tr>
<tr>
<td>no_slip_shelfice</td>
<td>no_slip_bottom</td>
<td>slip along bottom of ice shelf on/off flag</td>
</tr>
<tr>
<td>SHELFICEDragLinear</td>
<td>bottomDragLinear</td>
<td>linear drag coefficient at bottom ice shelf (m/s)</td>
</tr>
<tr>
<td>SHELFICEDragQuadratic</td>
<td>bottomDragQuadratic</td>
<td>quadratic drag coefficient at bottom ice shelf (non-dim.)</td>
</tr>
<tr>
<td>SHELFICEselectDragQuadradr</td>
<td>-1</td>
<td>select form of quadratic drag coefficient (non-dim.)</td>
</tr>
<tr>
<td>SHELFICEMassStepping</td>
<td>FALSE</td>
<td>recalculate ice shelf mass at every time step</td>
</tr>
</tbody>
</table>

continues on next page
Table 8.21 – continued from previous page

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SHELFICEDynMassOnly</td>
<td>FALSE</td>
<td>if SHELFICEmassStepping = TRUE, exclude freshwater flux contribution</td>
</tr>
<tr>
<td>SHELFICEadvDiffHeatFlux</td>
<td>FALSE</td>
<td>use advective-diffusive heat flux into ice shelf instead of default diffusive heat flux</td>
</tr>
<tr>
<td>SHELFICEuseGammaFRICT</td>
<td>FALSE</td>
<td>use velocity dependent exchange coefficients (Holland and Jenkins 1999 [HJ99])</td>
</tr>
<tr>
<td>SHELFICE_oldCalcUStar</td>
<td>FALSE</td>
<td>use old uStar averaging expression</td>
</tr>
<tr>
<td>SHELFICE_writeState</td>
<td>FALSE</td>
<td>write ice shelf state to file on/off flag</td>
</tr>
<tr>
<td>SHELFICE_dumpFreq</td>
<td>dumpFreq</td>
<td>dump frequency (s)</td>
</tr>
<tr>
<td>SHELFICE_dump_mnc</td>
<td>snapshot_mnc</td>
<td>write snapshot using MNC on/off flag</td>
</tr>
</tbody>
</table>

8.6.3.4 SHELFICE description

In the light of isomorphic equations for pressure and height coordinates, the ice shelf topography on top of the water column has a similar role as (and in the language of Marshall et al. (2004) [MAC+04], is isomorph to) the orography and the pressure boundary conditions at the bottom of the fluid for atmospheric and oceanic models in pressure coordinates. The total pressure \( p_{\text{tot}} \) in the ocean can be divided into the pressure at the top of the water column \( p_{\text{top}} \), the hydrostatic pressure and the non-hydrostatic pressure contribution \( p_{NH} \):

\[
p_{\text{tot}} = p_{\text{top}} + \int_{z}^{\eta-h} g \rho \, dz + p_{NH} \quad (8.27)
\]

with the gravitational acceleration \( g \), the density \( \rho \), the vertical coordinate \( z \) (positive upwards), and the dynamic sea-surface height \( \eta \). For the open ocean, \( p_{\text{top}} = p_a \) (atmospheric pressure) and \( h = 0 \). Underneath an ice-shelf that is assumed to be floating in isostatic equilibrium, \( p_{\text{top}} \) at the top of the water column is the atmospheric pressure \( p_a \) plus the weight of the ice-shelf. It is this weight of the ice-shelf that has to be provided as a boundary condition at the top of the water column (in run-time parameter SHELFICE_loadAnomalyFile). The weight is conveniently computed by integrating a density profile \( \rho^* \), that is constant in time and corresponds to the sea-water replaced by ice, from \( z = 0 \) to a “reference” ice-shelf draft at \( z = -h \) (Beckmann et al. (1999) [BHT99]), so that

\[
p_{\text{top}} = p_a + \int_{-h}^{0} g \rho^* \, dz \quad (8.28)
\]

Underneath the ice shelf, the “sea-surface height” \( \eta \) is the deviation from the “reference” ice-shelf draft \( h \). During a model integration, \( \eta \) adjusts so that the isostatic equilibrium is maintained for sufficiently slow and large scale motion.

In MITgcm, the total pressure anomaly \( p'_{\text{tot}} \) which is used for pressure gradient computations is defined by subtracting a purely depth dependent contribution \( -g \rho_c z \) using constant reference density \( \rho_c \) from \( p_{\text{tot}} \). (8.27) becomes

\[
p'_{\text{tot}} = p_{\text{top}} - g \rho_c (z + h) + g \rho_c \eta + \int_{z}^{\eta-h} g(\rho - \rho_c) \, dz + p_{NH} \quad (8.29)
\]

and after rearranging

\[
p'_{\text{tot}} = p'_{\text{top}} + g \rho_c \eta + \int_{z}^{\eta-h} g(\rho - \rho_c) \, dz + p_{NH}
\]

with \( p'_{\text{tot}} = p_{\text{tot}} + g \rho_c \, z \) and \( p'_{\text{top}} = p_{\text{top}} - g \rho_c \, h \). The non-hydrostatic pressure contribution \( p_{NH} \) is neglected in the following.

In practice, the ice shelf contribution to \( p_{\text{top}} \) is computed by integrating (8.28) from \( z = 0 \) to the bottom of the last fully dry cell within the ice shelf:

\[
p_{\text{top}} = g \sum_{k'=1}^{n-1} \rho_k^* \Delta z_k' + p_a \quad (8.30)
\]
where $n$ is the vertical index of the first (at least partially) “wet” cell and $\Delta z_{k'}$ is the thickness of the $k'$-th layer (counting downwards). The pressure anomaly for evaluating the pressure gradient is computed in the center of the “wet” cell $k$ as

$$p_k' = p_{\text{top}}' + g \rho_n \eta + g \sum_{k'=n}^{k} \left( \rho_{k'} - \rho_c \right) \Delta z_{k'} \left( 1 + \frac{H(k' - k)}{2} \right)$$

(8.31)

where $H(k' - k) = 1$ for $k' < k$ and 0 otherwise.

Figure 8.11: Schematic of a vertical section of the grid at the base of an ice shelf. Grid lines are thin; the thick line is the model’s representation of the ice shelf-water interface. Plus signs mark the position of pressure points for pressure gradient computations. The letters A, B, and C mark specific grid cells for reference. $h_k$ is the fractional cell thickness so that $h_k \Delta z_k$ is the actual cell thickness.

Setting SHELFICEboundaryLayer = .TRUE. introduces a simple boundary layer that reduces the potential noise problem at the cost of increased vertical mixing. For this purpose the water temperature at the $k$-th layer abutting ice shelf topography for use in the heat flux parameterizations is computed as a mean temperature $\bar{\theta}_k$ over a boundary layer of the same thickness as the layer thickness $\Delta z_k$:

$$\bar{\theta}_k = \theta_k h_k + \theta_{k+1} (1 - h_k)$$

(8.32)

where $h_k \in [0, 1]$ is the fractional layer thickness of the $k$-th layer (see Figure 8.11). The original contributions due to ice shelf-ocean interaction $g_{\theta}$ to the total tendency terms $G_{\theta}$ in the time-stepping equation $\theta^{n+1} = f(\theta^n, \Delta t, G^n_{\theta})$ are

$$g_{\theta,k} = \frac{Q}{\rho_c c_p h_k \Delta z_k}$$

and $g_{\theta,k+1} = 0$

(8.33)

for layers $k$ and $k + 1$ ($c_p$ is the heat capacity). Averaging these terms over a layer thickness $\Delta z_k$ (e.g., extending from the ice shelf base down to the dashed line in cell C) and applying the averaged tendency to cell A (in layer $k$) and to the appropriate fraction of cells C (in layer $k + 1$) yields

$$g_{\theta,k}^* = \frac{Q}{\rho_c c_p \Delta z_k}$$

(8.34)

and

$$g_{\theta,k+1}^* = \frac{Q}{\rho_c c_p \Delta z_{k+1}} \frac{\Delta z_k (1 - h_k)}{\Delta z_{k+1}}$$

(8.35)
(8.35) describes averaging over the part of the grid cell \( k + 1 \) that is part of the boundary layer with tendency \( g_{b,k}^* \) and the part with no tendency. Salinity is treated in the same way. The momentum equations are not modified.

### Three-equations thermodynamics

Freezing and melting form a boundary layer between the ice shelf and the ocean that is represented in the model by an infinitesimal layer used to calculate the exchanges between the ocean and the ice. Melting and freezing at the boundary between saline water and ice imply a freshwater mass flux \( q \) (\( q < 0 \) for melting); the relevant heat fluxes into and out of the boundary layer therefore include a diffusive flux through the ice, the latent heat flux due to melting and freezing, and the advective heat that is carried by the freshwater mass flux. There is a salt flux carried by the mass flux if the ice has a non-zero salinity \( S_I \). Further, the position of the interface between ice and ocean changes because of \( q \), so that, say, in the case of melting the volume of sea water increases. As a consequence of these fluxes, salinity and temperature are affected.

The turbulent tracer exchanges between the infinitesimal boundary layer and the ocean are expressed as diffusive fluxes. Following Jenkins et al. (2001) [JHH01], the boundary conditions for a tracer take into account that this boundary may not move with the ice ocean interface (for example, in a linear free surface model). The implied upward freshwater flux \( q \) is therefore included in the boundary conditions for the temperature and salinity equation as an advective flux.

The boundary conditions for tracer \( X = S, T \) (tracer \( X \) stands for either in-situ temperature \( T \) or salinity \( S \), located at the first interior ocean grid point) in the ocean are expressed as the sum of advective and diffusive fluxes

\[
F_X = (\rho_c \gamma_X - q)(X_b - X) \tag{8.36}
\]

where the diffusive flux has been parameterized as a turbulent exchange \( \rho_c \gamma_X (X_b - X) \) following Holland and Jenkins (1999) [HJ99] or Jenkins et al. (2001) [JHH01]. \( X_b \) indicates the tracer in the boundary layer, \( \rho_c \) the density of seawater (parameter rhoConst), and \( \gamma_X \) is the turbulent exchange (or transfer) coefficient (parameters SHELFICEheatTransCoeff and SHELFICESaltTransCoeff), in units of an exchange velocity. In-situ temperature, computed locally from tracer potential temperature, \( \theta \), is required here to accurately compute the in-situ freezing point of seawater in order to determine ice melt.

The tracer budget for the infinitesimal boundary layer takes the general form:

\[
\rho_l K_{I,X} \frac{\partial X_I}{\partial z} \bigg|_b = \rho_c \gamma_X (X_b - X) - q(X_b - X) \tag{8.37}
\]

where the LHS represents diffusive flux from the ice evaluated at the interface between the infinitesimal boundary layer and the ice, and the RHS represents the turbulent and advective exchanges between the infinitesimal layer and the ocean and the advective exchange between the boundary layer and the ice (\( qX_I \), this flux will be zero if the ice contains no tracer: \( X_I = 0 \)). The temperature in the boundary layer (\( T_b \)) is taken to be at the freezing point, which is a function of pressure and salinity, \( \rho_l \) is ice density (rhoShelfIce), and \( K_{I,X} \) the appropriate ice diffusivity.

For any material tracer such as salinity, the LHS in (8.37) vanishes (no material diffusion into the ice), while for temperature, the term \( q(T_b - T_I) \) vanishes, because both the boundary layer and the ice are at the freezing point. Instead, the latent heat of freezing is included as an additional term to take into account the conversion of ice to water:

\[
\rho_l c_{p,I} \kappa_{I,T} \frac{\partial T_I}{\partial z} \bigg|_b = c_p \rho_c \gamma_T (T_b - T) + Lq. \tag{8.38}
\]

where \( \kappa_{I,T} \) is the thermal ice diffusivity (SHELFICEkappa), \( L \) is the latent heat of fusion (SHELFICElatentHeat), \( c_p \) is the specific heat capacity of water (HeatCapacity_Cp), and \( c_{p,I} \) the heat capacity of the ice shelf (SHELFICEHeatCapacity_Cp). A reasonable choice for \( \gamma_T \) (SHELFICEheatTransCoeff), the turbulent exchange coefficient of temperature, is discussed in Holland and Jenkins (1999) [HJ99] (see Section 8.6.3.4). The temperature at the interface \( T_b \) is assumed to be the in-situ freezing point temperature of sea-water \( T_f \), which is computed from a linear equation of state:

\[
T_f = (0.0901 - 0.0575 S_b) - 7.61 \times 10^{-4} p_b. \tag{8.39}
\]
where $T_f$ is given in °C and $p_b$ is in dBar. In (8.38), the diffusive heat flux at the ice-ocean interface can be approximated by assuming a linear temperature profile in the ice and approximating the vertical derivative of temperature in the ice as the difference between the ice surface and ice bottom temperatures divided by the ice thickness, so that the left-hand-side of (8.38) becomes

$$\rho_f c_{p, I} \kappa_{I, T} \frac{\partial T_I}{\partial z} \bigg|_{S_b} = \frac{\rho_f c_{p, I} \kappa_{I, T}}{h} (T_S - T_b)$$

(8.40)

where $T_S$ the (surface) temperature of the ice shelf (SHELFICEthetaSurface) and $h$ is the ice-shelf draft. Alternatively, assuming that the ice is “advected” vertically as implied by the meltflux $q$, the diffusive flux can be approximated as $\min(q, 0) c_{p, I} (T_S - T_b)$ (runtime flag SHELFICEadvDifHeatFlux; see Holland and Jenkins, 1999 [HJ99] for details).

From the salt budget, the salt flux across the shelf ice-ocean interface is equal to the salt flux due to melting and freezing:

$$\rho_c \gamma_S (S - S_b) = -q (S_b - S_I)$$

(8.41)

where $\gamma_S = \text{SHELFICEsaltToHeatRatio} \cdot \gamma_T$ is the turbulent salinity exchange coefficient. Note, it is assumed that $\kappa_{I, S} = 0$; moreover, the salinity of the ice shelf is generally neglected ($S_I = 0$).

The budget equations for temperature (8.40) and salinity (8.41), together with the freezing point temperature of seawater (8.39), form the so-called three-equation-model (e.g., Hellmer and Olbers (1989) [HO89], Jenkins et al. (2001) [JHH01]). These equations are solved to obtain $S_b, T_b, q$, which are then substituted into (8.36) to obtain the boundary conditions for the temperature and salinity equations of the ocean model. Note that with $S_I = 0$ and (8.41), the boundary flux for salinity becomes $F_S = q S$, which is the flux that is necessary to account for the dilution of salinity in the case of melting.

The three-equation-model tends to yield smaller melt rates than the simpler formulation of the ISOMIP protocol (Section 8.6.3.4) because the freshwater flux (due to melting) decreases the salinity, which in turn raises the freezing point temperature and thus leads to less melting at the interface. For a simpler thermodynamics model where $S_b$ is not computed explicitly, for example as in the ISOMIP protocol, (8.36) cannot be applied directly. In this case (8.41) can be used with (8.36) to obtain:

$$F_X = q (S - S_I)$$

This formulation can be used for all cases for which (8.41) is valid. Further, in this formulation it is obvious that melting ($q < 0$) leads to a reduction of salinity.

The default value of SHELFICEconserve = .FALSE. removes the contribution $q (X_b - X)$ from (8.36), making the boundary conditions non-conservative.

### Solving the three-equations system

There has been some confusion about the three-equations system, so we document the solution in the code here: We use (8.39) $T_b = a_0 S_b + \epsilon_4$ to eliminate $T_b$ from (8.38) with (8.40) and find an expression for the freshwater flux $q$:

$$-L q = \epsilon_1 (T - a_0 S_b - \epsilon_4) + \epsilon_3 (T_S - a_0 S_b - \epsilon_4)$$

$$\Leftrightarrow L q = a_0 (\epsilon_1 + \epsilon_3) S_b + \epsilon_q$$

(8.42)

to be substituted into (8.41):

$$\epsilon_2 (S - S_b) = -L q (S_b - S_I) = -a_0 (\epsilon_1 + \epsilon_3) S_b + \epsilon_q (S_b - S_I)$$

$$\Leftrightarrow 0 = a_0 (\epsilon_1 + \epsilon_3) S_b^2 + \{\epsilon_q - \epsilon_2 - a_0 (\epsilon_1 + \epsilon_3) S_I\} S_b + \epsilon_2 S - \epsilon_q S_I$$

where the abbreviations $\epsilon_1 = c_p \rho_c \gamma_T, \epsilon_2 = \rho_c \gamma_S \epsilon_3 = \rho_c \gamma_T, \epsilon_4 = b_0 p + c_0, \epsilon_3 (\epsilon_4 - T) + \epsilon_3 (\epsilon_4 - T_S)$ have been introduced. The quadratic equation in $S_b$ is solved and the smaller non-negative root is used. In the MITgcm
code, the ice shelf salinity $S_I$ is always zero and the quadratic equation simplifies to

$$0 = a_0 (\epsilon_1 + \epsilon_3) S^2_b + (\epsilon_q - \epsilon_2) S_b + \epsilon_2 S$$

$$S_b = \frac{\epsilon_2 - \epsilon_q + \sqrt{(\epsilon_q - \epsilon_2)^2 - 4 a_0 (\epsilon_1 + \epsilon_3) \epsilon_2}}{2 a_0 (\epsilon_1 + \epsilon_3)}$$

With $S_b$, the boundary layer temperature $T_b$ and the melt rate $q$ are known through (8.39) and (8.42).

**ISOMIP thermodynamics**

A simpler formulation follows the ISOMIP protocol. The freezing and melting in the boundary layer between ice shelf and ocean is parameterized following Grosfeld et al. (1997) [GGD97]. In this formulation (8.38) reduces to

$$c_p \rho_c \gamma T (T - T_b) = -Lq$$

and the fresh water flux $q$ is computed from

$$q = -\frac{c_p \rho_c \gamma T (T - T_b)}{L}$$

In order to use this formulation, set runtime parameter useISOMIPTD = .TRUE. in data.shelfice.

**Exchange coefficients**

The default exchange coefficients $\gamma T/S$ are constant and set by the run-time parameters SHELFICEheatTransCoeff and SHELFICESaltTransCoeff (see Table 8.21). If SHELFICEuseGammaFrict = .TRUE., exchange coefficients are computed from drag laws and friction velocities estimated from ocean speeds following Holland and Jenkins (1999) [HJ99]. This computation can be modified using runtime parameters and the user is referred to pkg/shelfice/shelfice_readparms.F for details.

**Remark**

The shelfice package and experiments demonstrating its strengths and weaknesses are also described in Losch (2008) [Los08]. Unfortunately, the description of the thermodynamics in the appendix of Losch (2008) is wrong.

**8.6.3.5 Key subroutines**

The main routine is shelfice_thermodynamics.F but note that /pkg/shelfice routines are also called when solving the momentum equations.

```plaintext
C !CALLING SEQUENCE:  
C ...  
C |-FORWARD_STEP :: Step forward a time-step ( AT LAST !!! )  
C ...  
C | |-DO_OCEANIC_PHY :: Control oceanic physics and parameterization  
C ...  
C | | |-SHELFICE_THERMODYNAMICS :: main routine for thermodynamics with diagnostics  
C ...  
C | | |-THERMODYNAMICS :: theta, salt + tracer equations driver.  
```

(continues on next page)
8.6.3.6 SHELFICE diagnostics

Diagnostics output is available via the diagnostics package (see Section 9). Available output fields are summarized as follows:

<table>
<thead>
<tr>
<th>Name</th>
<th>Levs</th>
<th>grid</th>
<th>Units</th>
<th>Tile (max=80c)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SHIfwFlx</td>
<td>1</td>
<td>SM</td>
<td>kg/m^2/s</td>
<td>Ice shelf fresh water flux (positive upward)</td>
</tr>
<tr>
<td>SHIhtFlx</td>
<td>1</td>
<td>SM</td>
<td>W/m^2</td>
<td>Ice shelf heat flux (positive upward)</td>
</tr>
<tr>
<td>SHIUDrag</td>
<td>30</td>
<td>UU</td>
<td>m/s^2</td>
<td>U momentum tendency from ice shelf drag</td>
</tr>
<tr>
<td>SHIVDrag</td>
<td>30</td>
<td>VV</td>
<td>m/s^2</td>
<td>V momentum tendency from ice shelf drag</td>
</tr>
<tr>
<td>SHIForcT</td>
<td>1</td>
<td>SM</td>
<td>W/m^2</td>
<td>Ice shelf forcing for theta, &gt;0 increases theta</td>
</tr>
<tr>
<td>SHIForcS</td>
<td>1</td>
<td>SM</td>
<td>g/m^2/s</td>
<td>Ice shelf forcing for salt, &gt;0 increases salt</td>
</tr>
</tbody>
</table>

8.6.3.7 Experiments and tutorials that use shelfice

See the verification experiment isomip for example usage of pkg/shelfice.

8.6.4 SHELFICE Remeshing

Author: James Jordan and Daniel Goldberg
8.6.4.1 Introduction

This code works the nonlinear free surface feature of the ocean model to allow continuous updating of the ice shelf draft in a mass-, salt- and heat-conservative fashion. Note ‘Shelfice Remeshing’ is not a separate physical package as such, but works in conjunction with pkg/shelfice, and all code is contained within pkg/shelfice. However, as a separate line of development its application and use warrants its own entry in the documentation. By using pkg/streamice at the same time, remeshing can allow synchronous coupling between ocean and ice shelf.

8.6.4.2 REMESHING configuration and compiling

Compile-time options

Shelfice remeshing requires that pkg/shelfice be enabled, which is done by adding shelfice to packages.conf (see Section Section 3.5). Nonlinear free surface is required, which is enabled by adding #define NONLIN_FRSURF to CPP_OPTIONS.h. Additionally,

- #define ALLOW_SHELFICE_REMESHING must be added to SHELFICE_OPTIONS.h;
- If SHI_ALLOW_GAMMAFRICT is defined in SHELFICE_OPTIONS.h we recommend also setting run-time parameter SHI_withBL_uStarTopDz to .true., which will limit spurious features in the melt rate as explained in Section 8.6.4.5.

8.6.4.3 Run-time parameters

Table 8.22 lists run-time parameters in data.shelfice relevant to shelfice remeshing. In addition, nonlinFreeSurf=4 should be set, and select_rstar should be zero (the model default) in file data.

<table>
<thead>
<tr>
<th>Name</th>
<th>Default value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SHI_withBL_realFWflux</td>
<td>FALSE</td>
<td>Necessary for mass/volume-conservative freezing/melting when SHELFICEboundaryLayer = .true.</td>
</tr>
<tr>
<td>SHI_withBL_uStarTopDz</td>
<td>FALSE</td>
<td>With SHELFICEboundaryLayer = .true. compute $u^*$ from uVel,vVel averaged over top $\Delta z$ thickness</td>
</tr>
<tr>
<td>SHELFICEmassFile</td>
<td>' '</td>
<td>Initialization file for ice shelf mass (kg m$^{-2}$)</td>
</tr>
<tr>
<td>SHELFICEMassStepping</td>
<td>FALSE</td>
<td>Enables ice mass to change in time</td>
</tr>
<tr>
<td>SHELFICEMassDynTendFile</td>
<td>' '</td>
<td>Input file to specify non-thermodynamic ice mass change rate in kg/s (overridden when pkg/streamice enabled)</td>
</tr>
<tr>
<td>SHELFICEDynMassOnly</td>
<td>FALSE</td>
<td>Update shelficeMass via non-thermodynamic change only (overridden when pkg/streamice enabled)</td>
</tr>
<tr>
<td>SHELFICERemeshFrequency</td>
<td>2592000</td>
<td>Frequency of remeshing operation (seconds)</td>
</tr>
<tr>
<td>SHELFICESplitThreshold</td>
<td>1.25</td>
<td>Maximum allowed hFacC for a cell</td>
</tr>
<tr>
<td>SHELFICEMergeThreshold</td>
<td>0.26</td>
<td>Minimum allowed hFacC for a cell</td>
</tr>
</tbody>
</table>
8.6.4.4 Description

When pkg/shelfIce is enabled, the elevation of the free surface in a grid cell is determined by the mass of the ice shelf in that cell, see Jordan et al. (2018) [JHG+18]. In general use of shelfIce this mass is held constant, but if it is allowed to change the free surface adjusts if implicitFreeSurface = .true. through adjustment of the thickness of the top-level cell (Figure 8.12). If nonlinFreeSurf=4 these changes are fully accounted for in the ocean dynamics and tracer transport. However:

- in the case of ice thinning, the numerical approximation to the governing equations will break down if the top-level cell becomes too thick, and
- in the case of ice thickening, the top-level cell thickness cannot become negative or the model will fail.

Remeshing addresses these issues. At predefined intervals (set by SHELFICERemeshFrequency), the code checks every column and flags those where the top-level cell is too thick ($h_c$ is larger than SHELFICESplitThreshold) or where it is too thin ($h_c$ is smaller than SHELFICEMergeThreshold). In the former case, the cell is split into two cells as described in Figure 8.13. In the latter case, the top cells is “merged” with the one below it. In both cases, splitting and merging conserves mass, heat, and salt. Momentum is conserved where possible but this is more difficult because velocities live at cell edges.

Ice shelf basal melt and freezing add and remove water from the ocean. useRealFreshWaterFlux = .true. will cause the ocean volume to be updated - unless SHELFICEboundaryLayer = .true. which inputs heat and salt fluxes over a distance $\Delta z$ from the ice-ocean interface, instead of inputting directly into the top cell (Figure 8.12). In this case, an additional option SHI_withBL_realFWflux = .true. can be set in data.shelfIce to allow volume conservation.

![Figure 8.12: Schematic representation of (a) reference ice-shelf depth, d, vertical position of the ice-ocean interface, $z_{surf}$, and the distance between the two, $\eta$, and (b) the extent of the ice-shelf boundary layer used to calculate velocities, $Bv$ (red), and tracers, $B_x$ (blue), used in the melt rate calculation. The model grid is represented by dashed lines with the actual size of the cells represented by the solid lines.](image-url)
8.6.4.5 Alternate boundary layer formulation

If SHELFICEboundaryLayer = .true., then salt and temperature are averaged over a distance $\Delta z$ from the ice-ocean interface in order to calculate melt rates, as described in Losch (2008) [Los08]. When SHI ALLOW GAMMAFRICT is defined and SHELFICEuseGammaFrict = .true., near-ice velocities are used to calculate exchange coefficients of heat and salt, which can lead to spurious features where there is a change in the level of the top fluid cell. In the default formulation velocities (or rather square velocities) are first averaged horizontally from cell faces to cell centers, and then vertically over a distance $\Delta z$. The run-time parameter SHI withBL uStarTopDz = .true. reverses this order: velocities are first averaged vertically, and then horizontally. In some cases this has been found to give less spurious variability, but either can be used.

8.6.4.6 Coupling with pkg/streamice

Shelfice remeshing can be used on its own (i.e. without coupling to an ice sheet model), with the effects of ice dynamics specified via SHELFICEMassDynTendFile. Alternatively it can be coupled to the pkg/streamice. To allow this, pkg/streamice must be enabled. Please see the verification/shelfice_2d_remesh for an example. (Documentation on SHELFICE is under construction)

8.6.4.7 Diagnostics

In addition to the diagnostics from pkg/SHELFICE, remeshing adds one additional diagnostic: SHIRshe1, the “reference” elevation of the ice shelf base ($d$ in Figure 8.12).

8.6.4.8 Experiments that use Remeshing

- verification/shelfice_2d_remesh
8.6.5 STREAMICE Package

Author: Daniel Goldberg

8.6.5.1 Introduction

Package STREAMICE provides a dynamic land ice model for MITgcm. It was created primarily to develop a TAF- and OpenAD-generated ice model adjoint and to provide synchronous ice-ocean coupling through the SHELFICE package. It solves a set of dynamic equations appropriate for floating ice-shelf flow as well as ice-stream and slower ice-sheet flow. It has been tested at the scale of one or several ice streams, but has not been tested at the continental scale.

8.6.5.2 STREAMICE configuration

Compile-time options

pkg/streamice can be included on at compile time in the packages.conf file by adding a line streamice (see Section 8.1.1).

Parts of the pkg/streamice code can be enabled or disabled at compile time via CPP flags. These options are set in STREAMICE_OPTIONS.h.

Table 8.23: CPP flags used by pkg/streamice.

<table>
<thead>
<tr>
<th>CPP Flag Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>STREAMICE_CONSTRUCT_MATRIX</td>
<td>#define</td>
<td>explicit construction of matrix for Picard iteration for velocity</td>
</tr>
<tr>
<td>STREAMICE_HYBRID_STRESS</td>
<td>#undef</td>
<td>use L1L2 formulation for stress balance (default shallow shelf approx.)</td>
</tr>
<tr>
<td>USE_ALT_RLOW</td>
<td>#undef</td>
<td>use package array for rLow rather than model</td>
</tr>
<tr>
<td>STREAMICE_GEOM_FILE_SETUP</td>
<td>#undef</td>
<td>use files rather than parameters in STREAMICE_PARM03 to configure boundaries</td>
</tr>
<tr>
<td>ALLOW_PETSC</td>
<td>#undef</td>
<td>enable interface to PETSc for velocity solver matrix solve</td>
</tr>
<tr>
<td>STREAMICE_COULOMB_SLIDING</td>
<td>#undef</td>
<td>enable basal sliding of the form (8.50)</td>
</tr>
</tbody>
</table>

Enabling the package

Once it has been compiled, pkg/streamice is switched on/off at run-time by setting useSTREAMICE to .TRUE. in file data.pkg.

Runtime parameters: general flags and parameters

Run-time parameters are set in file data.streamice (read in streamice_readparms.F). General pkg/streamice parameters are set under STREAMICE_PARM01 as described in Table 8.24.

Table 8.24: Run-time parameters and default values (defined under STREAMICE_PARM01 namelist)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>streamice_density</td>
<td>910</td>
<td>the (uniform) density of land ice (kg/m^3)</td>
</tr>
<tr>
<td>Parameter</td>
<td>Default</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------------</td>
<td>---------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>streamice_density_ocean_avg</td>
<td>1024</td>
<td>the (uniform) density of ocean (kg/m^3)</td>
</tr>
<tr>
<td>n_glen</td>
<td>3</td>
<td>Glen’s Flow Law exponent (non-dim.)</td>
</tr>
<tr>
<td>eps_glen_min</td>
<td>1e-12</td>
<td>minimum strain rate in Glen’s Law (ε₀, yr⁻¹)</td>
</tr>
<tr>
<td>eps_u_min</td>
<td>1e-6</td>
<td>minimum speed in nonlinear sliding law (u₀, m/yr)</td>
</tr>
<tr>
<td>n_basal_friction</td>
<td>0</td>
<td>exponent in nonlinear sliding law (non-dim.)</td>
</tr>
<tr>
<td>streamice_cg_tol</td>
<td>1e-6</td>
<td>tolerance of conjugate gradient of linear solve of Picard iteration for velocity</td>
</tr>
<tr>
<td>streamice_lower_cg_tol</td>
<td>TRUE</td>
<td>lower CG tolerance when nonlinear residual decreases by fixed factor</td>
</tr>
<tr>
<td>streamice_max_cg_iter</td>
<td>2000</td>
<td>maximum iterations in linear solve</td>
</tr>
<tr>
<td>streamice_maxciter_cpl</td>
<td>0</td>
<td>as above when coupled with pkg/shelfice</td>
</tr>
<tr>
<td>streamice_nonlin_tol</td>
<td>1e-6</td>
<td>tolerance of nonlinear residual for velocity (relative to initial)</td>
</tr>
<tr>
<td>streamice_maxnliter_cpl</td>
<td>0</td>
<td>as above when coupled with pkg/shelfice</td>
</tr>
<tr>
<td>streamice_nonlin_tol_tol</td>
<td>1e-6</td>
<td>tolerance of relative change for velocity iteration (relative to magnitude)</td>
</tr>
<tr>
<td>streamice_err_norm</td>
<td>0</td>
<td>type of norm evaluated for error (p in p-norm; 0 is ∞)</td>
</tr>
<tr>
<td>streamice_chkfixedptconvergence</td>
<td>FALSE</td>
<td>terminate velocity iteration based on relative change per iteration</td>
</tr>
<tr>
<td>streamice_chkresidconvergence</td>
<td>TRUE</td>
<td>terminate velocity iteration based on residual</td>
</tr>
<tr>
<td>streamicethickInit</td>
<td>FILE</td>
<td>thickness initialization file, in meters (rather than parameters in STREAMICE_PARM03)</td>
</tr>
<tr>
<td>streamicethickFile</td>
<td>' '</td>
<td>file to initialize streamicethickFile</td>
</tr>
<tr>
<td>streamice_move_front</td>
<td>FALSE</td>
<td>allow ice shelf front to advance</td>
</tr>
<tr>
<td>streamice_calve_to_mask</td>
<td>FALSE</td>
<td>if streamice_move_front TRUE do not allow to advance beyond streamice_calve_mask</td>
</tr>
<tr>
<td>streamice_calveMaskFile</td>
<td>' '</td>
<td>file to initialize streamice_calve_mask</td>
</tr>
<tr>
<td>streamice_criteria_only</td>
<td>FALSE</td>
<td>do not update ice thickness (velocity solve only)</td>
</tr>
<tr>
<td>streamice_CFL_factor</td>
<td>0.5</td>
<td>CFL factor which determine maximum time step for thickness subcycling</td>
</tr>
<tr>
<td>streamice_adjDump</td>
<td>0.0</td>
<td>frequency (s) of writing of adjoint fields to file (TAF only)</td>
</tr>
<tr>
<td>streamicebasalTracConfig</td>
<td>UNIFORM</td>
<td>method by which to initialize basal traction (FILE or UNIFORM)</td>
</tr>
<tr>
<td>streamicebasalTracFile</td>
<td>' '</td>
<td>basal trac initialization file (see Units of input files for units)</td>
</tr>
<tr>
<td>C_basal_fric_const</td>
<td>31.71</td>
<td>uniform basal traction value (see Units of input files for units)</td>
</tr>
<tr>
<td>streamiceGlenConstConfig</td>
<td>UNIFORM</td>
<td>method by which to initialize Glen’s constant (FILE or UNIFORM)</td>
</tr>
<tr>
<td>streamiceGlenConstFile</td>
<td>' '</td>
<td>Glen’s constant initialization file (see Units of input files for units)</td>
</tr>
<tr>
<td>B_glen_isothermal</td>
<td>9.461e-18</td>
<td>uniform Glen’s constant value (see Units of input files for units)</td>
</tr>
<tr>
<td>streamiceBdotFile</td>
<td>' '</td>
<td>file to initialize time-indep melt rate (m/yr)</td>
</tr>
<tr>
<td>streamiceBdotTimeDepFile</td>
<td>' '</td>
<td>file to initialize time-varying melt rate (m/yr), based on streamice_forcing_period</td>
</tr>
<tr>
<td>streamiceTopogFile</td>
<td>' '</td>
<td>topography initialization file (m); requires #define USE_ALT_RLOW</td>
</tr>
<tr>
<td>streamiceHmaskFile</td>
<td>' '</td>
<td>streamice_hmask initialization file; requires #define STREAMICE_GEOM_FILE_SETUP</td>
</tr>
<tr>
<td>streamiceuFaceBdryFile</td>
<td>' '</td>
<td>streamice STREAMICE_ufacemask_bdry initialization file; requires #define STREAMICE_GEOM_FILE_SETUP</td>
</tr>
<tr>
<td>streamicevFaceBdryFile</td>
<td>' '</td>
<td>streamice STREAMICE_vfacemask_bdry initialization file; requires #define STREAMICE_GEOM_FILE_SETUP</td>
</tr>
</tbody>
</table>
### Configuring domain through files

The **STREAMICE_GEOM_FILE_SETUP** CPP option allows versatility in defining the domain. With this option, the array `streamice_hmask` must be initialized through a file (`streamiceHmaskFile`) as must `streamice_ufacemask_bdry` and `streamice_vfacemask_bdry` (through `streamiceuFaceBdryFile` and `streamicevFaceBdryFile`) as well as `u_flux_bdry_SI` and `v_flux_bdry_SI`, volume flux at the boundaries, where appropriate (through `streamiceuMassFluxFile` and `streamicevMassFluxFile`). Thickness must be initialized through a file as well (`streamicethickFile`); `streamice_hmask` is set to zero where ice thickness is zero, and boundaries between in-domain and out-of-domain cells (according to `streamice_hmask`) are no-slip by default.

When using this option, it is important that for all internal boundaries, `streamice_ufacemask_bdry` and `streamice_vfacemask_bdry` are -1 (this will not be the case if `streamiceuFaceBdryFile` and `streamicevFaceBdryFile` are undefined).

In fact, if `streamice_hmask` is configured correctly, `streamice_ufacemask_bdry` and `streamice_vfacemask_bdry` can be set uniformly to -1, UNLESS there are no-stress or flux-condition boundaries in the domain. Where `streamice_ufacemask_bdry` and `streamice_vfacemask_bdry` are set to -1, they will be overridden at (a) boundaries where `streamice_hmask` changes from 1 to -1 (which become no-slip boundaries), and (b) boundaries where `streamice_hmask` changes from 1 to 0 (which become calving front boundaries).

#### Table 8.24 – continued from previous page

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>streamiceuMassFluxFile</td>
<td>’ ’</td>
<td>mass flux at u-faces init. file (m²/yr); requires #define STREAM-ICE_GEOM_FILE_SETUP</td>
</tr>
<tr>
<td>streamicevMassFluxFile</td>
<td>’ ’</td>
<td>mass flux at v-faces init. file (m²/yr); requires #define STREAM-ICE_GEOM_FILE_SETUP</td>
</tr>
<tr>
<td>streamiceuFluxTimeDepFile</td>
<td>’ ’</td>
<td>time-depend. mass flux at u-faces file (m²/yr); requires #define STREAM-ICE_GEOM_FILE_SETUP</td>
</tr>
<tr>
<td>streamicevFluxTimeDepFile</td>
<td>’ ’</td>
<td>time-depend. mass flux at v-faces file (m²/yr); requires #define STREAM-ICE_GEOM_FILE_SETUP</td>
</tr>
<tr>
<td>streamiceuNormalStressFile</td>
<td>’ ’</td>
<td>calving front normal stress parm along u-faces (non-dim.; see Boundary Stresses)</td>
</tr>
<tr>
<td>streamicevNormalStressFile</td>
<td>’ ’</td>
<td>calving front normal stress parm along v-faces (non-dim.; see Boundary Stresses)</td>
</tr>
<tr>
<td>streamiceuShearStressFile</td>
<td>’ ’</td>
<td>calving front normal stress parm along u-faces (non-dim.; see Boundary Stresses)</td>
</tr>
<tr>
<td>streamicevShearStressFile</td>
<td>’ ’</td>
<td>calving front normal stress parm along v-faces (non-dim.; see Boundary Stresses)</td>
</tr>
<tr>
<td>streamiceuNormalTimeDepFile</td>
<td>’ ’</td>
<td>time-dependent version of streamiceuNormalStressFile</td>
</tr>
<tr>
<td>streamicevNormalTimeDepFile</td>
<td>’ ’</td>
<td>time-dependent version of streamicevNormalStressFile</td>
</tr>
<tr>
<td>streamiceuShearTimeDepFile</td>
<td>’ ’</td>
<td>time-dependent version of streamiceuShearStressFile</td>
</tr>
<tr>
<td>streamicevShearTimeDepFile</td>
<td>’ ’</td>
<td>time-dependent version of streamicevShearStressFile</td>
</tr>
<tr>
<td>streamice_adot_uniform</td>
<td>0</td>
<td>file input frequency for streamice time-dependent forcing fields (s)</td>
</tr>
<tr>
<td>streamice_forcing_period</td>
<td>0</td>
<td>thickness range parameter in basal traction smoothing across grounding line (m)</td>
</tr>
<tr>
<td>streamice_allow_reg_coulomb</td>
<td>FALSE</td>
<td>use regularized Coulomb sliding (8.50). Requires STREAMICE_COULOMB_SLIDING CPP option.</td>
</tr>
<tr>
<td>STREAMICE_vel_ext</td>
<td>FALSE</td>
<td>over-ride velocity calculation with binary file</td>
</tr>
<tr>
<td>STREAMICE_uvel_ext_file</td>
<td>FALSE</td>
<td>over-ride velocity calculation with binary file, with velocities applied directly to C-grid.</td>
</tr>
<tr>
<td>STREAMICE_vvel_ext_file</td>
<td>FALSE</td>
<td>file to initialise x-velocity component (m/a)</td>
</tr>
<tr>
<td>STREAMICE_vvel_ext_file</td>
<td>FALSE</td>
<td>file to initialise y-velocity component (m/a)</td>
</tr>
</tbody>
</table>
An example of domain configuration through files can be found in verification/halfpipe_streamice. By default, verification/halfpipe_streamice is compiled with STREAMICE_GEOM_FILE_SETUP undefined, but the user can modify this option. The file verification/halfpipe_streamice/input/data.streamice_geomSetup represents an alternative version of verification/halfpipe_streamice/input/data.streamice in which the appropriate binary files are specified.

### Configuring domain through parameters

For a very specific type of domain the boundary conditions and initial thickness can be set via parameters in data.streamice. Such a domain will be rectangular. In order to use this option, the STREAMICE_GEOM_FILE_SETUP CPP flag should be undefined.

There are different boundary condition types (denoted within the parameter names) that can be set:

- **noflow**: $x$- and $y$-velocity will be zero along this boundary.
- **nostress**: velocity normal to boundary will be zero; there will be no tangential stress along the boundary.
- **fluxbdry**: a mass volume flux is specified along this boundary, which becomes a boundary condition for the thickness advection equation (see *Equations Solved*). Velocities will be zero. The corresponding parameters flux_bdry_val_NORTH, flux_bdry_val_SOUTH, flux_bdry_val_EAST and flux_bdry_val_WEST then set the values.
- **CFBC**: calving front boundary condition, a Neumann condition based on ice thickness and bed depth, is imposed at this boundary (see *Equations Solved*).

Note the above only apply if there is dynamic ice in the cells at the boundary in question. The boundary conditions are then set by specifying the above conditions over ranges of each (north/south/east/west) boundary. The division of each boundary should be exhaustive and the ranges should not overlap. Parameters to initialize boundary conditions (defined under STREAMICE_PARM03 namelist) are listed in Table 8.25.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>min_x_noflow_NORTH</td>
<td>0</td>
<td>western limit of no-flow region on northern boundary (m)</td>
</tr>
<tr>
<td>max_x_noflow_NORTH</td>
<td>0</td>
<td>eastern limit of no-flow region on northern boundary (m)</td>
</tr>
<tr>
<td>min_x_noflow_SOUTH</td>
<td>0</td>
<td>western limit of no-flow region on southern boundary (m)</td>
</tr>
<tr>
<td>max_x_noflow_SOUTH</td>
<td>0</td>
<td>eastern limit of no-flow region on southern boundary (m)</td>
</tr>
<tr>
<td>min_x_noflow_EAST</td>
<td>0</td>
<td>southern limit of no-flow region on eastern boundary (m)</td>
</tr>
<tr>
<td>max_x_noflow_EAST</td>
<td>0</td>
<td>northern limit of no-flow region on eastern boundary (m)</td>
</tr>
<tr>
<td>min_x_noflow_WEST</td>
<td>0</td>
<td>southern limit of no-flow region on western boundary (m)</td>
</tr>
<tr>
<td>max_x_noflow_WEST</td>
<td>0</td>
<td>northern limit of no-flow region on western boundary (m)</td>
</tr>
<tr>
<td>min_x_nostress_NORTH</td>
<td>0</td>
<td>western limit of no-stress region on northern boundary (m)</td>
</tr>
<tr>
<td>max_x_nostress_NORTH</td>
<td>0</td>
<td>eastern limit of no-stress region on northern boundary (m)</td>
</tr>
<tr>
<td>min_x_nostress_SOUTH</td>
<td>0</td>
<td>western limit of no-stress region on southern boundary (m)</td>
</tr>
<tr>
<td>max_x_nostress_SOUTH</td>
<td>0</td>
<td>eastern limit of no-stress region on southern boundary (m)</td>
</tr>
<tr>
<td>min_x_nostress_EAST</td>
<td>0</td>
<td>southern limit of no-stress region on eastern boundary (m)</td>
</tr>
<tr>
<td>max_x_nostress_EAST</td>
<td>0</td>
<td>northern limit of no-stress region on eastern boundary (m)</td>
</tr>
<tr>
<td>min_x_nostress_WEST</td>
<td>0</td>
<td>southern limit of no-stress region on western boundary (m)</td>
</tr>
<tr>
<td>max_x_nostress_WEST</td>
<td>0</td>
<td>northern limit of no-stress region on western boundary (m)</td>
</tr>
<tr>
<td>min_x_fluxbdry_NORTH</td>
<td>0</td>
<td>western limit of flux-boundary region on northern boundary (m)</td>
</tr>
<tr>
<td>max_x_fluxbdry_NORTH</td>
<td>0</td>
<td>eastern limit of flux-boundary region on northern boundary (m)</td>
</tr>
<tr>
<td>min_x_fluxbdry_SOUTH</td>
<td>0</td>
<td>western limit of flux-boundary region on southern boundary (m)</td>
</tr>
<tr>
<td>max_x_fluxbdry_SOUTH</td>
<td>0</td>
<td>eastern limit of flux-boundary region on southern boundary (m)</td>
</tr>
<tr>
<td>min_x_fluxbdry_EAST</td>
<td>0</td>
<td>southern limit of flux-boundary region on eastern boundary (m)</td>
</tr>
<tr>
<td>max_y_fluxbdry_EAST</td>
<td>0</td>
<td>northern limit of flux-boundary region on eastern boundary (m)</td>
</tr>
</tbody>
</table>

continues on next page
Table 8.25 – continued from previous page

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>max_y_fluxbdry_EAST</td>
<td>0</td>
<td>northern limit of flux-boundary region on eastern boundary (m)</td>
</tr>
<tr>
<td>min_y_fluxbdry_WEST</td>
<td>0</td>
<td>southern limit of flux-boundary region on western boundary (m)</td>
</tr>
<tr>
<td>max_y_fluxbdry_WEST</td>
<td>0</td>
<td>northern limit of flux-boundary region on western boundary (m)</td>
</tr>
<tr>
<td>min_x_CFBC_NORTH</td>
<td>0</td>
<td>western limit of calving front condition region on northern boundary (m)</td>
</tr>
<tr>
<td>max_x_CFBC_NORTH</td>
<td>0</td>
<td>eastern limit of calving front condition region on northern boundary (m)</td>
</tr>
<tr>
<td>min_x_CFBC_SOUTH</td>
<td>0</td>
<td>western limit of calving front condition region on southern boundary (m)</td>
</tr>
<tr>
<td>max_x_CFBC_SOUTH</td>
<td>0</td>
<td>eastern limit of calving front condition region on southern boundary (m)</td>
</tr>
<tr>
<td>min_y_CFBC_EAST</td>
<td>0</td>
<td>southern limit of calving front condition region on eastern boundary (m)</td>
</tr>
<tr>
<td>max_y_CFBC_EAST</td>
<td>0</td>
<td>northern limit of calving front condition region on eastern boundary (m)</td>
</tr>
<tr>
<td>min_y_CFBC_WEST</td>
<td>0</td>
<td>southern limit of calving front condition region on western boundary (m)</td>
</tr>
<tr>
<td>max_y_CFBC_WEST</td>
<td>0</td>
<td>northern limit of calving front condition region on western boundary (m)</td>
</tr>
<tr>
<td>flux_bdry_val_SOUTH</td>
<td>0</td>
<td>volume flux per width entering at flux-boundary on southern boundary (m²/a)</td>
</tr>
<tr>
<td>flux_bdry_val_NORTH</td>
<td>0</td>
<td>volume flux per width entering at flux-boundary on southern boundary (m²/a)</td>
</tr>
<tr>
<td>flux_bdry_val_EAST</td>
<td>0</td>
<td>volume flux per width entering at flux-boundary on southern boundary (m²/a)</td>
</tr>
<tr>
<td>flux_bdry_val_WEST</td>
<td>0</td>
<td>volume flux per width entering at flux-boundary on southern boundary (m²/a)</td>
</tr>
</tbody>
</table>

8.6.5.3 Description

Equations Solved

The model solves for 3 dynamic variables: $x$-velocity ($u$), $y$-velocity ($v$), and thickness ($h$). There is also a variable that tracks coverage of fractional cells, discussed in Ice front advance.

By default the model solves the “shallow shelf approximation” (SSA) for velocity. The SSA is appropriate for floating ice (ice shelf) or ice flowing over a low-friction bed (e.g., Macayeal (1989) [Mac89]). The SSA consists of the $x$-momentum balance:

$$
\partial_x (hc(u\partial_x + 2v\partial_y)) + \partial_y (2hu\partial_x) - \tau_{bx} = \rho gh \frac{\partial s}{\partial x}
$$

the $y$-momentum balance:

$$
\partial_x (2hu\partial_x) + \partial_y (h(u\partial_y + 2v\partial_x)) - \tau_{by} = \rho gh \frac{\partial s}{\partial y}
$$

where $\rho$ is ice density, $g$ is gravitational acceleration, and $s$ is surface elevation. $\nu$, $\tau_{bx}$, and $\tau_{by}$ are ice viscosity, basal drag, and the strain rate tensor, respectively, all explained below.

From the velocity field, thickness evolves according to the continuity equation:

$$
h_t + \nabla \cdot (hu\partial) = \dot{a} - \dot{b}
$$

Where $\dot{b}$ is a basal mass balance (e.g., melting due to contact with the ocean), positive where there is melting. This is a field that can be specified through a file. At the moment surface mass balance $\dot{a}$ can only be set as uniform. Where ice is grounded, surface elevation is given by

$$
s = R + h
$$

where $R(x, y)$ is the bathymetry, and the basal elevation $b$ is equal to $R$. If ice is floating, then the assumption of hydrostasy and constant density gives

$$
s = (1 - \frac{\rho}{\rho_w})h_z
$$

8.6. Ice and Sea Ice Packages

555
where \( \rho_w \) is a representative ocean density, and \( b = -\langle \rho/\rho_w \rangle h \). Again by hydrostasy, floatation is assumed wherever

\[
h \leq -\frac{\rho_w}{\rho} R
\]

is satisfied. Floatation criteria is stored in `float_frac_streamice`, equal to 1 where ice is grounded, and equal to 0 where ice is floating.

The strain rates \( \varepsilon_{ij} \) are generalized to the case of orthogonal curvilinear coordinates, to include the “metric” terms that arise when casting the equations of motion on a sphere or projection on to a sphere (see *Finite-volume discretization of the stress tensor divergence*). Thus

\[
\dot{\varepsilon}_{xx} = u_x + k_1 v,
\]

\[
\dot{\varepsilon}_{yy} = v_y + k_1 u,
\]

\[
\dot{\varepsilon}_{xy} = \frac{1}{2}(u_y + v_x) + k_1 u + k_2 v.
\]

\( \nu \) has the form arising from Glen’s law

\[
\nu = \frac{1}{2} A^{-\frac{1}{2}} \left( \dot{\varepsilon}_{xx}^2 + \dot{\varepsilon}_{yy}^2 + \dot{\varepsilon}_{xy}^2 \dot{\varepsilon}_{yy} + \dot{\varepsilon}_{xy}^2 + \varepsilon_{\text{min}}^2 \right)^{\frac{1}{2m}}
\]

(8.48) though the form is slightly different if a hybrid formulation is used.

Whether \( \tau_b \) is nonzero depends on whether the floatation condition is satisfied. Currently this is determined simply on an instantaneous cell-by-cell basis (unless subgrid interpolation is used), as is the surface elevation \( s \), but possibly this should be rethought if the effects of tides are to be considered. \( \tau_b \) has the form

\[
\tau_b = C(\bar{u}|u|^2 + u_{\text{min}}^2 \frac{m^{-1}}{2} \bar{u}^2)
\]

(8.49) again, the form is slightly different if a hybrid formulation is to be used, and the velocity refers to sliding velocity (\( u_b \)). An alternative to the above “power law” sliding parameterization can be used by defining the `STREAM-ICE_COULOB_SLIDING` CPP option and setting `streamice_allow_reg_coulomb` to `.TRUE.`.

\[
\tau_b = C \frac{|u|^m N}{2 \left[ C^{1/m} |u| + (0.5N)^{1/m} \right]^{1/m} u_{\text{min}}^{-1} \bar{u}}
\]

(8.50)

where \( u \) is shorthand for the regularized norm in (8.49) (or for \( u_b \) if a hybrid formulation is used). \( m \) is the same exponent as in (8.49). \( N \) is effective pressure:

\[
N = \rho g (h - h_f),
\]

(8.51) with \( h_f \) the floatation thickness

\[
h_f = \max \left( 0, -\frac{\rho_w}{\rho} R \right),
\]

where \( R \) is bed elevation. This formulation was used in the MISMIP+ intercomparison tests [ADCD+16]. (8.51) assumes complete hydraulic connectivity to the ocean throughout the domain, which is likely only true within a few tens of kilometers of the grounding line. With this sliding relation, Coulomb sliding is predominant near the grounding line, with the yield strength proportional to height above floatation. Further inland sliding transitions to the power law relation in (8.49).

The momentum equations are solved together with appropriate boundary conditions, discussed below. In the case of a calving front boundary condition (CFBC), the boundary condition has the following form:

\[
(h \nu(4\dot{\varepsilon}_{xx} + 2\dot{\varepsilon}_{yy})/n_x + (2h \nu \dot{\varepsilon}_{xy}) n_y = \frac{1}{2} g \left( \rho h^2 - \rho_w b^2 \right) n_x
\]

(8.52)

\[
(2h \nu \dot{\varepsilon}_{xy}) n_x + (h \nu(4\dot{\varepsilon}_{yy} + 2\dot{\varepsilon}_{xx})) n_y = \frac{1}{2} g \left( \rho h^2 - \rho_w b^2 \right) n_y.
\]

(8.53)

Here \( \vec{n} \) is the normal to the boundary, and \( b \) is ice base.
Hybrid SIA-SSA stress balance

The SSA does not take vertical shear stress or strain rates (e.g., $\sigma_{xz}, \partial u/\partial z$) into account. Although there are other terms in the stress tensor, studies have found that in all but a few cases, vertical shear and longitudinal stresses (represented by the SSA) are sufficient to represent glaciological flow. pkg/streamice can allow for representation of vertical shear, although the approximation is made that longitudinal stresses are depth-independent. The stress balance is referred to as “hybrid” because it is a joining of the SSA and the “shallow ice approximation” (SIA), which accounts only for vertical shear. Such hybrid formulations have been shown to be valid over a larger range of conditions than SSA (Goldberg 2011) [Gol11].

In the hybrid formulation, $\mathbf{\Pi}$ and $\mathbf{\Sigma}$, the depth-averaged $x-$ and $y-$velocities, replace $u$ and $v$ in (8.45), (8.46), and (8.47), and gradients such as $u_x$ are replaced by $(\mathbf{\Pi})_x$. Viscosity becomes

$$\nu = \frac{1}{2} A^{-\frac{1}{n}} \left( \dot{\varepsilon}_{xx}^2 + \dot{\varepsilon}_{yy}^2 + \dot{\varepsilon}_{xy} \dot{\varepsilon}_{yx} + \dot{\varepsilon}_{x}^2 + \frac{1}{4} u_x^2 + \frac{1}{4} v_z^2 + \dot{\varepsilon}_{min}^2 \right)^{\frac{1}{2n}}$$

In the formulation for $\tau_b$, $u_b$, the horizontal velocity at $u_b$ is used instead. The details are given in Goldberg (2011) [Gol11].

Ice front advance

By default all mass flux across calving boundaries is considered lost. However, it is possible to account for this flux and potential advance of the ice shelf front. If streamice_move_front is TRUE, then a partial-area formulation is used.

The algorithm is based on Albrecht et al. (2011) [AMH+11]. In this scheme, for empty or partial cells adjacent to a calving front, a reference thickness $h_{ref}$ is found, defined as an average over the thickness of all neighboring cells that flow into the cell. The total volume input over a time step is added to the volume of ice already in the cell, whose partial area coverage is then updated based on the volume and reference thickness. If the area coverage reaches 100% in a time step, then the additional volume is cascaded into adjacent empty or partial cells.

If streamice_calve_to_mask is TRUE, this sets a limit to how far the front can advance, even if advance is allowed. The front will not advance into cells where the array streamice_calve_mask is not equal to 1. This mask must be set through a binary input file to allow the front to advance past its initial position.

No calving parameterization is implemented in pkg/streamice. However, front advancement is a precursor for such a development to be added.

Units of input files

The inputs for basal traction (streamicebasalTracFile, C_basal_fric_const) and ice stiffness (streamiceGlenConstFile, B_glen_isothermal) require specific units. For ice stiffness ($A$ in (8.48)), $B = A^{-1/n}$ is specified; or, more accurately, its square root $A^{-1/(2n)}$ is specified (this is to ensure positivity of $B$ by squaring the input). The units of streamiceGlenConstFile and B_glen_isothermal are Pa$^{1/2}$ yr$^{1/(2n)}$ where $n$ is n_glen.

streamicebasalTracFile and C_basal_fric_const initialize the basal traction ($C$ in (8.49)). Again $C^{1/2}$ is directly specified rather than $C$ to ensure positivity. The units are Pa$^{1/2}$ (m yr$^{-1}$)$^{n_b}$ where $n_b$ is n_basal_friction.
8.6.5.4 Numerical Details

The moment balance is solved via iteration on viscosity (Goldberg 2011 [Gol11]). At each iteration, a linear elliptic differential equation is solved via a finite-element method using bilinear basis functions. The velocity solution “lives” on cell corners, while thickness “lives” at cell centers (Figure 8.14). The cell-centered thickness is then evolved using a second-order slope-limited finite-volume scheme, with the velocity field from the previous solve. To represent the flow of floating ice, basal stress terms are multiplied by an array float_frac_streamice, a cell-centered array which determines where ice meets the floatation condition.

The computational domain of pkg/streamice (which may be smaller than the array/grid as defined by SIZE.h and GRID.h) is determined by a number of mask arrays within pkg/streamice. They are

- **hmask** (streamice_hmask): equal to 1 (ice-covered), 0 (open ocean), 2 (partly-covered), or -1 (out of domain)
- **umask** (streamice_umask): equal to 1 (an “active” velocity node), 3 (a Dirichlet node), or 0 (zero velocity)
- **vmask** (streamice_vmask): similar to umask

- **ufacemaskbdry** (streamice ufacemask_bdry): equal to -1 (interior face), 0 (no-slip), 1 (no-stress), 2 (calving stress front), or 4 (flux input boundary); when 4, then u_flux_bdry_SI must be initialized, through binary or parameter file
- **vfacemaskbdry** (streamice_vfacemask_bdry): similar to ufacemaskbdry

*hmask* is defined at cell centers, like *h*. *umask* and *vmask* are defined at cell nodes, like velocities. *ufacemaskbdry* and *vfacemaskbdry* are defined at cell faces, like velocities in a C-grid - but unless one sets #define STREAMICE_GEOM_FILE_SETUP in STREAMICE_OPTIONS.h, the values are only relevant at the boundaries of the grid.
Figure 8.15: Hypothetical configuration, detailing the meaning of thickness and velocity masks and their role in controlling boundary conditions.
The values of \( u_{\text{mask}} \) and \( v_{\text{mask}} \) determine which nodal values of \( u \) and \( v \) are involved in the solve for velocities. These masks are not configured directly by the user, but are re-initialized based on \textit{streamice_hmask}, \textit{streamice_ufacemask_bdry} and \textit{streamice_vfacemask_bdry} at each time step. Figure 8.15 demonstrates how these values are set in various cells.

With \( u_{\text{mask}} \) and \( v_{\text{mask}} \) appropriately initialized, subroutine \textit{streamice_vel_solve.F} can proceed rather generally. Contributions are only evaluated if \( h_{\text{mask}} = 1 \) in a given cell, and a given nodal basis function is only considered if \( u_{\text{mask}} = 1 \) or \( v_{\text{mask}} = 1 \) at that node.

### 8.6.5.5 Additional Features

#### PETSc

There is an option to use PETSc for the matrix solve component of the velocity solve, and this has been observed to give a 3- or 4-fold improvement in performance over the inbuilt conjugate gradient solver in a number of cases. To use this option, the CPP option \texttt{ALLOW_PETSC} must be defined, and MITgcm must be compiled with the \texttt{-mpi} flag (see Section 3.5.4). However, often a system-specific installation of PETSc is required. If you wish to use PETSc with \texttt{pkg/streamice}, please contact the author.

#### Boundary Stresses

The calving front boundary conditions (8.52) and (8.53) are intended for ice fronts bordering open ocean. However, there may be reasons to apply different Neumann conditions at these locations, e.g., one might want to represent force associated with ice melange, or to represent parts of the ice shelf that are not resolved, as in Goldberg et al. (2015) [GHJS15]. The user can then modify these boundary conditions in the form

\[
(h \nu (4 \dot{\varepsilon}_{xx} + 2 \dot{\varepsilon}_{yy})) n_x + (2 h \nu \dot{\varepsilon}_{xy}) n_y = \frac{1}{2} g \left( \rho h^2 - \rho_w b^2 \right) n_x + \sigma n_x + \tau n_y
\]

\[
(2 h \nu \dot{\varepsilon}_{xy}) n_x + (h \nu (4 \dot{\varepsilon}_{yy} + 2 \dot{\varepsilon}_{xx})) n_y = \frac{1}{2} g \left( \rho h^2 - \rho_w b^2 \right) n_y + \sigma n_y + \tau n_x
\]

In these equations, \( \sigma \) and \( \tau \) represent normal and shear stresses at the boundaries of cells. They are not specified directly, but through coefficients \( \gamma_{\sigma} \) and \( \gamma_{\tau} \):

\[
\sigma = \frac{1}{2} g \left( \rho h^2 - \rho_w b^2 \right) \gamma_{\sigma}
\]

\[
\tau = \frac{1}{2} g \left( \rho h^2 - \rho_w b^2 \right) \gamma_{\tau}
\]

\( \gamma_{\sigma} \) is specified through \textit{streamiceuNormalStressFile}, \textit{streamicevNormalStressFile}, \textit{streamiceuNormalTimeDepFile}, \textit{streamicevNormalTimeDepFile} and \( \gamma_{\tau} \) is specified through \textit{streamiceuShearStressFile}, \textit{streamicevShearStressFile}, \textit{streamiceuShearTimeDepFile}, and \textit{streamicevShearTimeDepFile}. Within the file names, the \( u \) and \( v \) determine whether the values are specified along horizontal \( (u-) \) faces and vertical \( (v-) \) faces. The values will only have an effect if they are specified along calving front boundaries (see Configuring domain through files).

### 8.6.5.6 Adjoint

The STREAMICE package is adjointable using both TAF (Goldberg et al. 2013 [GH13]) and OpenAD (Goldberg et al. 2016 [GNHU16]). In OpenAD, the fixed-point method of [Chr94] is implemented, greatly reducing the memory requirements and also improving performance when PETSc is used.

Verification experiments with both OpenAD and TAF are located in the \texttt{verification/halfpipe_streamice} (see below).
8.6.5.7 Key Subroutines

Top-level routine: streamice_timestep.F (called from model/src/do_oceanic_phys.F)

```
CALLING SEQUENCE
...
streamice_timestep (called from DO_OCEANIC_PHYS)
|-- #ifdef ALLOW_STREAMICE_TIMEDEP_FORCING
|-- STREAMICE_FIELDS_LOAD
| #endif
|--#if (defined (ALLOW_STREAMICE_OAD_FP))
|-- STREAMICE_VEL_SOLVE_OPENAD
| #else
|-- STREAMICE_VEL_SOLVE
| | -- STREAMICE_DRIVING_STRESS
| | [ITERATE ON FOLLOWING]
||-- STREAMICE_VISC_BETA
| ||-- STREAMICE_VISC_BETA_HYBRID
| | #ifdef ALLOW_PETSC
| | STREAMICE_VISC_BETA_PETSC
| | #endif
| ||-- #ifdef STREAMICE_HYBRID_STRESS
| | STREAMICE_VISC_BETA_HYBRID
| | #else
| | STREAMICE_VISC_BETA
| | #endif
|-- STREAMICE_ADVECT_THICKNESS
| | |-- STREAMICE_ADV_FRONT
|-- STREAMICE_UPD_FFRAC_UNCOUPLED
``` 

8.6.5.8 STREAMICE diagnostics

Diagnostics output is available via the diagnostics package (Packages II - Diagnostics and I/O). Available output fields are summarized in the following table:

<table>
<thead>
<tr>
<th>&lt;-Name-&gt;</th>
<th>Levs</th>
<th>mate</th>
<th>&lt;- code -&gt;</th>
<th>&lt;- Units -&gt;</th>
<th>&lt;- Tile (max=80c)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SI_Uvel</td>
<td>1</td>
<td></td>
<td>UZ</td>
<td>L1</td>
<td>m/a</td>
</tr>
<tr>
<td>SI_Vvel</td>
<td>1</td>
<td></td>
<td>VZ</td>
<td>L1</td>
<td>m/a</td>
</tr>
<tr>
<td>SI_Thick</td>
<td>1</td>
<td></td>
<td>SM</td>
<td>L1</td>
<td>m</td>
</tr>
</tbody>
</table>

(continues on next page)
8.6.5.9 Experiments and tutorials that use streamice

The verification/halfpipe_streamice experiment uses pkg/streamice.

8.7 Biogeochemistry Packages

8.7.1 GCHEM Package

8.7.1.1 Introduction

This package has been developed as interface to the PTRACERS package. The purpose is to provide a structure where various (any) tracer experiments can be added to the code. For instance there are placeholders for routines to read in parameters needed for any tracer experiments, a routine to read in extra fields required for the tracer code, routines for either external forcing or internal interactions between tracers and routines for additional diagnostics relating to the tracers. Note that the gchem package itself is only a means to call the subroutines used by specific biogeochemical experiments, and does not “do” anything on its own.

There are two examples: cfc which looks at 2 tracers with a simple external forcing and dic with 4, 5 or 6 tracers whose tendency terms are related to one another. We will discuss these here only as how they provide examples to use this package.

8.7.1.2 Key subroutines and parameters

FRAMEWORK

The different biogeochemistry frameworks (e.g. cfc of dic) are specified in the packages_conf file. GCHEM_OPTIONS.h includes the compiler options to be used in any experiment. An important compiler option is #define GCHEM_SEPARATE_FORCING which determined how and when the tracer forcing is applied (see discussion on Forcing below). See section on dic for some additional flags that can be set for that experiment.

There are further runtime parameters set in data.gchem and kept in common block GCHEM.h. These runtime options include:

- Parameters to set the timing for periodic forcing files to be loaded are: gchem_ForcingPeriod, gchem_ForcingCycle. The former is how often to load, the latter is how often to cycle through those fields (eg. period couple be monthly and cycle one year). This is used in dic and cfc, with gchem_ForcingPeriod=0 meaning no periodic forcing.
- nsubtime is the integer number of extra timesteps required by the tracer experiment. This will give a timestep of deltaTtracer/nsubtime for the dependencies between tracers. The default is one.
- File names - these are several filenames than can be read in for external fields needed in the tracer forcing - for instance wind speed is needed in both DIC and CFC packages to calculate the air-sea exchange of gases. Not all file names will be used for every tracer experiment.
- `gchem_int_` are variable names for run-time set integer numbers. (Currently 1 through 5).
- `gchem_rl_` are variable names for run-time set real numbers. (Currently 1 through 5).
- Note that the old `tIter0` has been replaced by `PTRACERS_Iter0` which is set in `data.ptracers` instead.

**INITIALIZATION**

The values set at runtime in `data.gchem` are read in using `gchem_readparms.F` which is called from `packages_readparms.F`. This will include any external forcing files that will be needed by the tracer experiment. There are two routine used to initialize parameters and fields needed by the experiment packages. These are `gchem_init_fixed.F` which is called from `packages_init_fixed.F`, and `gchem_init_vari.F` called from `packages_init_variable.F`. The first should be used to call a subroutine specific to the tracer experiment which sets fixed parameters, the second should call a subroutine specific to the tracer experiment which sets (or initializes) time fields that will vary with time.

**LOADING FIELDS**

External forcing fields used by the tracer experiment are read in by a subroutine (specific to the tracer experiment) called from `gchem_fields_load.F`. This latter is called from `forward_step.F`.

**FORCING**

Tracer fields are advected-and-diffused by the ptracer package. Additional changes (e.g. surface forcing or interactions between tracers) to these fields are taken care of by the gchem interface. For tracers that are essentially passive (e.g. CFC’s) but may have some surface boundary conditions this can easily be done within the regular tracer timestep. In this case `gchem_calc_tendency.F` is called from `forward_step.F`, where the reactive (as opposed to the advective diffusive) tendencies are computed. These tendencies, stored on the 3D field `gchemTendency`, are added to the passive tracer tendencies `gPtr` in `gchem_add_tendency.F`, which is called from `ptracers_forcing.F`. For tracers with more complicated dependencies on each other, and especially tracers which require a smaller timestep than `deltaTtracer`, it will be easier to use `gchem_forcing_sep.F` which is called from `forward_step.F`. There is a compiler option set in `GCHEM_OPTIONS.h` that determines which method is used: `#define GCHEM_SEPARATE_FORCING` does the latter where tracers are forced separately from the advection-diffusion code, and `#undef GCHEM_SEPARATE_FORCING` includes the forcing in the regular timestepping.

**DIAGNOSTICS**

This package also uses the passive tracer routine `ptracers_monitor.F` which prints out tracer statistics as often as the model dynamic statistic diagnostics (dynsys) are written (or as prescribed by the runtime flag `PTRACERS_monitorFreq`, set in `data.ptracers`). There is also a placeholder for any tracer experiment specific diagnostics to be calculated and printed to files. This is done in `gchem_diags.F`. For instance the time average CO2 air-sea fluxes, and sea surface pH (among others) are written out by `dic_biotic_diags.F` which is called from `gchem_diags.F`.

### 8.7.1.3 GCHEM Diagnostics

These diagnostics are particularly for the `dic` package.

<table>
<thead>
<tr>
<th>&lt;-Name-&gt;</th>
<th>Levs</th>
<th>&lt;-parsing code-&gt;</th>
<th>&lt;- Units --&gt;</th>
<th>&lt;- Tile (max=80c)</th>
</tr>
</thead>
<tbody>
<tr>
<td>DICBIOA</td>
<td>15</td>
<td>SM P MR</td>
<td>mol/m3/sec</td>
<td>Biological Productivity (mol/m3/s)</td>
</tr>
<tr>
<td>DICARB</td>
<td>15</td>
<td>SM P MR</td>
<td>mol eq/m3/sec</td>
<td>Carbonate chg-biol prod and remin (mol/eq/m3/s)</td>
</tr>
<tr>
<td>DICTFLX</td>
<td>1</td>
<td>SM P L1</td>
<td>mol/m3/sec</td>
<td>Tendency of DIC due to air-sea exch (mol/m3/s)</td>
</tr>
</tbody>
</table>

(continues on next page)
8.7.1.4 Do’s and Don’ts

The pkg tracer is required with use with this pkg. Also, as usual, the runtime flag useGCHEM must be set to .TRUE. in data.pkg. By itself, gchem pkg will read in data.gchem and will write out gchem diagnostics. It requires tracer experiment specific calls to do anything else (for instance the calls to dic and cfc pkgs).

8.7.1.5 Reference Material

8.7.1.6 Experiments and tutorials that use gchem

- Global Ocean biogeochemical tutorial, in tutorial_global_oce_biogeo verification directory, described in section [sec:eg-biogeochem_tutorial] uses gchem and dic
- Global Ocean cfc tutorial, in tutorial_cfc_offline verification directory, uses gchem and cfc (and offline) described in [sec:eg-offline-cfc]
- Global Ocean online cfc example in cfc_example verification directory, uses gchem and cfc

8.7.2 DIC Package

8.7.2.1 Introduction

This is one of the biogeochemical packages handled from the pkg gchem. The main purpose of this package is to consider the cycling of carbon in the ocean. It also looks at the cycling of phosphorous and potentially oxygen and iron. There are four standard tracers \( DIC, ALK, PO4, DOP \) and also possibly \( O2 \) and \( Fe \). The air-sea exchange of \( CO2 \) and \( O2 \) are handled as in the OCMIP experiments (reference). The export of biological matter is computed as a function of available light and \( PO4 \) (and \( Fe \)). This export is remineralized at depth according to a Martin curve (again, this is the same as in the OCMIP experiments). There is also a representation of the carbonate flux handled as in the OCMIP experiments. The air-sea exchange on \( CO2 \) is affected by temperature, salinity and the pH of the surface waters. The pH is determined following the method of Follows et al. For more details of the equations see section [sec:eg-biogeochem_tutorial].

8.7.2.2 Key subroutines and parameters

INITIALIZATION

\( DIC_ABIOTIC.h \) contains the common block for the parameters and fields needed to calculate the air-sea flux of \( CO2 \) and \( O2 \). The fixed parameters are set in \( dic_abiotic_param \) which is called from gchem_init_fixed.F. The parameters needed for the biotic part of the calculations are initialized in \( dic_biotic_param \) and are stored in \( DIC_BIOTIC.h \). The first guess of pH is calculated in \( dic_surfforcing_init.F \).

LOADING FIELDS

---

### Table

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DICOFLX</td>
<td>mol/m3/sec</td>
<td>Tendency of ( O2 ) due to air-sea exch</td>
</tr>
<tr>
<td>DICCFLX</td>
<td>mol/m2/sec</td>
<td>Flux of ( CO2 ) - air-sea exch</td>
</tr>
<tr>
<td>DICPCO2</td>
<td>atm</td>
<td>Partial Pressure of ( CO2 )</td>
</tr>
<tr>
<td>DICPHAV</td>
<td>dimensionless</td>
<td>pH</td>
</tr>
</tbody>
</table>

---
The air-sea exchange of CO₂ and O₂ need wind, atmospheric pressure (although the current version has this hardwired to 1), and sea-ice coverage. The calculation of pH needs silica fields. These fields are read in in `dic_fields_load.F`. These fields are initialized to zero in `dic_ini_forcing.F`. The fields for interpolating are in common block in `DIC_LOAD.h`.

**FORCING**

The tracers are advected-diffused in `ptracers_integrate.F`. The updated tracers are passed to `dic_biotic_forcing.F` where the effects of the air-sea exchange and biological activity and remineralization are calculated and the tracers are updated for a second time. Below we discuss the subroutines called from `dic_biotic_forcing.F`.

Air-sea exchange of CO₂ is calculated in `dic_surrforcing`. Air-Sea Exchange of CO₂ depends on T,S and pH. The determination of pH is done in `carbon_chem.F`. There are three subroutines in this file: `carbon_coeffs` which determines the coefficients for the carbon chemistry equations; `calc_pco2` which calculates the pH using a Newton-Raphson method; and `calc_pco2_approx` which uses the much more efficient method of Follows et al. The latter is hard-wired into this package, the former is kept here for completeness.

Biological productivity is determined following Dutkiewicz et al. (2005) and is calculated in `bio_export.F` The light in each latitude band is calculate in `insol.F`, unless using one of the flags listed below. The formation of hard tissue (carbonate) is linked to the biological productivity and has an effect on the alkalinity - the flux of carbonate is calculated in `car_flux.F`, unless using the flag listed below for the Friis et al (2006) scheme. The flux of phosphate to depth where it instantly remineralized is calculated in `phos_flux.F`.

The dilution or concentration of carbon and alkalinity by the addition or subtraction of freshwater is important to their surface patterns. These “virtual” fluxes can be calculated by the model in several ways. The older scheme is done following OCMIP protocols (see more in Dutkiewicz et al 2005), in the subroutines `dic_surrforcing.F` and `alk_surrforcing.F`. To use this you need to set in `GCHEM_OPTIONS.h`: #define ALLOW_OLDVIRTUALFLUX. But this can also be done by the ptracers pkg if this is undefined. You will then need to set the concentration of the tracer in rainwater and potentially a reference tracer value in data.ptracer (PTRACERS_EvPrRn, and PTRACERS_ref respectively).

Oxygen air-sea exchange is calculated in `o2_surrforcing.F`.

Iron chemistry (the amount of free iron) is taken care of in `fe_chem.F`.

**DIAGNOSTICS**

Averages of air-sea exchanges, biological productivity, carbonate activity and pH are calculated. These are initialized to zero in `dic_biotic_init` and are stored in common block in `DIC_BIOTIC.h`.

**COMPILE TIME FLAGS**

These are set in `GCHEM_OPTIONS.h`:

DIC_BIOTIC: needs to be set for dic to work properly (should be fixed sometime).

ALLOW_O2: include the tracer oxygen.

ALLOW_FE: include the tracer iron. Note you will need an iron dust file set in data.gchem in this case.

MINFE: limit the iron, assuming precipitation of any excess free iron.

CAR_DISS: use the calcium carbonate scheme of Friis et al 2006.

ALLOW_OLDVIRTUALFLUX: use the old OCMIP style virtual flux for alkalinity adn carbon (rather than doing it through pkg/ptracers).
READ_PAR: read the light (photosynthetically available radiation) from a file set in data.gchem.
USE_QSW: use the numbers from QSW to be the PAR. Note that a file for Qsw must be supplied in data, or Qsw must be supplied by an atmospheric model.
If the above two flags are not set, the model calculates PAR in insol.F as a function of latitude and year day.
USE_QSW_UNDERICE: if using a sea ice model, or if the Qsw variable has the seaice fraction already taken into account, this flag must be set.
AD_SAFE: will use a tanh function instead of a max function - this is better if using the adjoint
DIC_NO_NEG: will include some failsafes in case any of the variables become negative. (This is advisable).
ALLOW_DIC_COST: was used for calculating cost function (but hasn’t been updated or maintained, so not sure if it works still)

8.7.2.3 Do’s and Don’ts

This package must be run with both ptracers and gchem enabled. It is set up for at least 4 tracers, but there is the provision for oxygen and iron. Note the flags above.

8.7.2.4 Reference Material


8.7.2.5 Experiments and tutorials that use dic

• Global Ocean tutorial, in tutorial_global_oce_biogeo verification directory, described in section [sec:eg-biogeochem_tutorial]
CHAPTER NINE

PACKAGES II - DIAGNOSTICS AND I/O

MITgcm includes several packages related to input and output during a model integration. The packages described in this chapter are related to the choice of input/output fields and their on-disk format.

9.1 pkg/diagnostics – A Flexible Infrastructure

9.1.1 Introduction

This section of the documentation describes the diagnostics package (pkg/diagnostics) available within MITgcm. A large selection of model diagnostics is available for output. In addition to the diagnostic quantities pre-defined within MITgcm, there exists the option, in any code setup, to define a new diagnostic quantity and include it as part of the diagnostic output with the addition of a single subroutine call in the routine where the field is computed. As a matter of philosophy, no diagnostic is enabled as default, thus each user must specify the exact diagnostic information required for an experiment. This is accomplished by enabling the specific diagnostics of interest from the list of available diagnostics. Additional diagnostic quantities, defined within different MITgcm packages, are available and are listed in the diagnostic list subsection of the manual section associated with each relevant package. Instructions for enabling diagnostic output and defining new diagnostic quantities are found in Section 9.1.4 of this document.

Once a diagnostic is enabled, MITgcm will continually increment an array specifically allocated for that diagnostic whenever the appropriate quantity is computed. A counter is defined which records how many times each diagnostic quantity has been incremented. Several special diagnostics are included in the list of available diagnostics. Quantities referred to as “counter diagnostics” are defined for selected diagnostics which record the frequency at which a diagnostic is incremented separately for each model grid location. Quantities referred to as “user diagnostics” are included to facilitate defining new diagnostics for a particular experiment.

9.1.2 Equations

Not relevant.

9.1.3 Key Subroutines and Parameters

There are several utilities within MITgcm available to users to enable, disable, clear, write and retrieve model diagnostics, and may be called from any routine. The available utilities and the CALL sequences are listed below.

diagnostics_addtolist.F: This routine is the underlying interface routine for defining a new permanent diagnostic in the main model or in a package. The calling sequence is:
CALL DIAGNOSTICS_ADDTOLIST ( 
  O diagNum, 
  I diagName, diagCode, diagUnits, diagTitle, diagMate, 
  I myThid ) 

where:
  diagNum = diagnostic Id number - Output from routine 
  diagName = name of diagnostic to declare 
  diagCode = parser code for this diagnostic 
  diagUnits = field units for this diagnostic 
  diagTitle = field description for this diagnostic 
  diagMate = diagnostic mate number 
  myThid = my Thread Id number 

diagnostics_fill.F: This is the main user interface routine to the diagnostics package. This routine will increment the specified diagnostic quantity with a field sent through the argument list.

CALL DIAGNOSTICS_FILL( 
  I inpFld, diagName, 
  I kLev, nLevs, bibjFlg, bi, bj, myThid ) 

where:
  inpFld = Field to increment diagnostics array 
  diagName = diagnostic identifier name (8 characters long) 
  kLev = Integer flag for vertical levels: 
    > 0 (any integer): WHICH single level to increment in qdiag. 
    0,-1 to increment "nLevs" levels in qdiag, 
    0 : fill-in in the same order as the input array 
    -1: fill-in in reverse order. 
  nLevs = indicates Number of levels of the input field array 
    (whether to fill-in all the levels (kLev<1) or just one (kLev>0)) 
  bibjFlg = Integer flag to indicate instructions for bi bj loop 
    = 0 indicates that the bi-bj loop must be done here 
    = 1 indicates that the bi-bj loop is done OUTSIDE 
    = 2 indicates that the bi-bj loop is done OUTSIDE 
    AND that we have been sent a local array (with overlap regions) 
      (local array here means that it has no bi-bj dimensions) 
    = 3 indicates that the bi-bj loop is done OUTSIDE 
      AND that we have been sent a local array 
      AND that the array has no overlap region (interior only) 
      NOTE - bibjFlg can be NEGATIVE to indicate not to increment counter 
  bi = X-direction tile number - used for bibjFlg=1-3 
  bj = Y-direction tile number - used for bibjFlg=1-3 
  myThid = my thread Id number 

diagnostics_scale_fill.F: This is a possible alternative routine to diagnostics_fill.F which performs the same functions and has an additional option to scale the field before filling or raise the field to a power before filling.

CALL DIAGNOSTICS_SCALE_FILL( 
  I inpFld, scaleFact, power, diagName, 
  I kLev, nLevs, bibjFlg, bi, bj, myThid ) 

(continues on next page)
where all the arguments are the same as for DIAGNOSTICS_FILL with the addition of:

- `scaleFact` = Scaling factor to apply to the input field product
- `power` = Integer power to which to raise the input field (after scaling)

**diagnostics_fract_fill.F**: This is a specific alternative routine to `diagnostics_scale_fill.F` for the case of a diagnostics which is associated to a fraction-weight factor (referred to as the diagnostics “counter-mate”). This fraction-weight field is expected to vary during the simulation and is provided as argument to `diagnostics_fract_fill.F` in order to perform fraction-weighted time-average diagnostics. Note that the fraction-weight field has to correspond to the diagnostics counter-mate which has to be filled independently with a call to `diagnostics_fill.F`.

```fortran
CALL DIAGNOSTICS_FRACT_FILL(
    inpFld, fractFld, scaleFact, power, diagName,
    kLev, nLevs, bibjFlg, bi, bj, myThid )
```

where all the arguments are the same as for `DIAGNOSTICS_SCALE_FILL` with the addition of:

- `fractFld` = fraction used for weighted average diagnostics

**diagnostics_is_on.F**: Function call to inquire whether a diagnostic is active and should be incremented. Useful when there is a computation that must be done locally before a call to `diagnostics_fill.F`. The call sequence:

```fortran
flag = DIAGNOSTICS_IS_ON( diagName, myThid )
```

where:

- `diagName` = diagnostic identifier name (8 characters long)
- `myThid` = my thread Id number

**diagnostics_count.F**: This subroutine increments the diagnostics counter only. In general, the diagnostics counter is incremented at the same time as the diagnostics is filled, by calling `diagnostics_fill.F`. However, there are few cases where the counter is not incremented during the filling (e.g., when the filling is done level per level but level 1 is skipped) and needs to be done explicitly with a call to subroutine `diagnostics_count.F`. The call sequence is:

```fortran
CALL DIAGNOSTICS_COUNT(
    diagName, bi, bj, myThid )
```

where:

- `diagName` = name of diagnostic to increment the counter
- `bi` = X-direction tile number, or 0 if called outside `bi,bj` loops
- `bj` = Y-direction tile number, or 0 if called outside `bi,bj` loops
- `myThid` = my thread Id number

The diagnostics are computed at various times and places within MITgcm. Because MITgcm may employ a staggered grid, diagnostics may be computed at grid box centers, corners, or edges, and at the middle or edge in the vertical. Some diagnostics are scalars, while others are components of vectors. An internal array is defined which contains information concerning various grid attributes of each diagnostic. The `gdiag` array (in common block diagnostics in file `DIAGNOSTICS.h`) is internally defined as a character*16 variable, and is equivalenced to a character*1 “parse” array in output in order to extract the grid-attribute information. The `gdiag` array is described in Table 9.1.

9.1. pkg/diagnostics – A Flexible Infrastructure
Table 9.1: Diagnostic Parsing Array

<table>
<thead>
<tr>
<th>Array</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>parse(1)</td>
<td>→ S</td>
<td>scalar diagnostic</td>
</tr>
<tr>
<td></td>
<td>→ U</td>
<td>U-vector component diagnostic</td>
</tr>
<tr>
<td></td>
<td>→ V</td>
<td>V-vector component diagnostic</td>
</tr>
<tr>
<td>parse(2)</td>
<td>→ U</td>
<td>C-grid U-point</td>
</tr>
<tr>
<td></td>
<td>→ V</td>
<td>C-grid V-point</td>
</tr>
<tr>
<td></td>
<td>→ M</td>
<td>C-grid mass point</td>
</tr>
<tr>
<td></td>
<td>→ Z</td>
<td>C-grid vorticity (corner) point</td>
</tr>
<tr>
<td>parse(3)</td>
<td>→ r</td>
<td>used for level-integrated output: cumulate levels</td>
</tr>
<tr>
<td></td>
<td>→ R</td>
<td>same but cumulate product by model level thickness</td>
</tr>
<tr>
<td>parse(4)</td>
<td>→ P</td>
<td>positive definite diagnostic</td>
</tr>
<tr>
<td></td>
<td>→ A</td>
<td>Adjoint variable diagnostic</td>
</tr>
<tr>
<td>parse(5)</td>
<td>→ C</td>
<td>with counter array</td>
</tr>
<tr>
<td></td>
<td>→ P</td>
<td>post-processed (not filled up) from other diags</td>
</tr>
<tr>
<td></td>
<td>→ D</td>
<td>disable diagnostic for output</td>
</tr>
<tr>
<td>parse(6-8)</td>
<td>→</td>
<td>retired, formerly 3-digit mate number</td>
</tr>
<tr>
<td>parse(9)</td>
<td>→ U</td>
<td>model level + ( \frac{1}{2} )</td>
</tr>
<tr>
<td></td>
<td>→ M</td>
<td>model level middle</td>
</tr>
<tr>
<td></td>
<td>→ L</td>
<td>model level - ( \frac{1}{2} )</td>
</tr>
<tr>
<td>parse(10)</td>
<td>→ 0</td>
<td>levels = 0</td>
</tr>
<tr>
<td></td>
<td>→ 1</td>
<td>levels = 1</td>
</tr>
<tr>
<td></td>
<td>→ R</td>
<td>levels = Nr</td>
</tr>
<tr>
<td></td>
<td>→ L</td>
<td>levels = MAX(Nr,NrPhys)</td>
</tr>
<tr>
<td></td>
<td>→ M</td>
<td>levels = MAX(Nr,NrPhys) - 1</td>
</tr>
<tr>
<td></td>
<td>→ G</td>
<td>levels = ground_level number</td>
</tr>
<tr>
<td></td>
<td>→ I</td>
<td>levels = seaice_level number</td>
</tr>
<tr>
<td></td>
<td>→ X</td>
<td>free levels option (need to be set explicitly)</td>
</tr>
</tbody>
</table>

As an example, consider a diagnostic whose associated gdiag parameter is equal to UUR MR. From gdiag we can determine that this diagnostic is a U-vector component located at the C-grid U-point, model mid-level (M) with Nr levels (last R).

In this way, each diagnostic in the model has its attributes (i.e., vector or scalar, C-grid location, etc.) defined internally. The output routines use this information in order to determine what type of transformations need to be performed. Any interpolations are done at the time of output rather than during each model step. In this way the user has flexibility in determining the type of output gridded data.

9.1.4 Usage Notes

9.1.4.1 Using available diagnostics

To use the diagnostics package, other than enabling it in packages.conf and turning the useDiagnostics flag in data.pkg to .TRUE., there are two further steps the user must take to enable the diagnostics package for output of quantities that are already defined in MITgcm under an experiment's configuration of packages. A parameter file data.diagnostics must be supplied in the run directory, and the file DIAGNOSTICS_SIZE.h must be included in the code directory. The steps for defining a new (permanent or experiment-specific temporary) diagnostic quantity will be outlined later.

The namelist in parameter file data.diagnostics will activate a user-defined list of diagnostics quantities to be computed, specify the frequency and type of output, the number of levels, and the name of all the separate output files.
A sample data.diagnostics namelist file:

```plaintext
# Diagnostic Package Choices #-------------------
# dumpAtLast (logical): always write output at the end of simulation (default=F)
# diag_mnc   (logical): write to NetCDF files (default=useMNC)
#--for each output-stream:
# fileName(n) : prefix of the output file name (max 80c long) for outp.stream n
# frequency(n):< 0 : write snap-shot output every |frequency| seconds
# > 0 : write time-average output every frequency seconds
# timePhase(n) : write at time = timePhase + multiple of |frequency|
# averagingFreq : frequency (in s) for periodic averaging interval
# averagingPhase : phase (in s) for periodic averaging interval
# repeatCycle : number of averaging intervals in 1 cycle
# levels(:,n) : list of levels to write to file (Notes: declared as REAL)
# when this entry is missing, select all common levels of this list
# fields(:,n) : list of selected diagnostics fields (8.c) in outp.stream n
# (see "available_diagnostics.log" file for the full list of diags)
# missing_value(n) : missing value for real-type fields in output file "n"
# fileFlags(n) : specific code (8c string) for output file "n"
#-------------------
&DIGONOSTICS_LIST
  fields(1:2,1) = 'UVEL ', 'VVEL ',
  levels(1:5,1) = 1., 2., 3., 4., 5.,
  filename(1) = 'diagout1',
  frequency(1) = 86400.,
  fields(1:2,2) = 'THETA ', 'SALT ',
  filename(2) = 'diagout2',
  fileflags(2) = 'P1 ',
  frequency(2) = 3600.,
&
&DIG_STATIS_PARMS
&
```

In this example, there are two output files that will be generated for each tile and for each output time. The first set of output files has the prefix `diagout1`, does time averaging every 86400. seconds, (frequency is 86400.), and will write fields which are multiple-level fields at output levels 1-5. The names of diagnostics quantities are `UVEL` and `VVEL`. The second set of output files has the prefix `diagout2`, does time averaging every 3600. seconds, includes fields with all levels, and the names of diagnostics quantities are `THETA` and `SALT`.

The user must assure that enough computer memory is allocated for the diagnostics and the output streams selected for a particular experiment. This is accomplished by modifying the file `DIAGNOSTICS_SIZE.h` and including it in the experiment code directory. The parameters that should be checked are called `numDiags`, `numLists`, `numperList`, and `diagSt_size`.

`numDiags` (and `diagSt_size`): All MITgcm diagnostic quantities are stored in the single diagnostic array `gdiag` which is located in the file and has the form:

```plaintext
_RL qdiag(1-0lx,sNx+0lx,1-0lx,sNx+0lx,numDiags,nSx,nSy)
_RL qsdiag(0:nStats,0:nRegions,diagSt_size,nSx,nSy)
COMMON / DIAG_STORE_R / qdiag, qsdiag
```

The first two-dimensions of `diagSt_size` correspond to the horizontal dimension of a given diagnostic, and the third dimension of `diagSt_size` is used to identify diagnostic fields and levels combined. In order to minimize the memory
requirement of the model for diagnostics, the default MITgcm executable is compiled with room for only one horizontal diagnostic array, or with numDiags set to Nr. In order for the user to enable more than one 3-D diagnostic, the size of the diagnostics common must be expanded to accommodate the desired diagnostics. This can be accomplished by manually changing the parameter numDiags in the file. numDiags should be set greater than or equal to the sum of all the diagnostics activated for output each multiplied by the number of levels defined for that diagnostic quantity. For the above example, there are four multiple level fields, which the available diagnostics list (see below) indicates are defined at the MITgcm vertical resolution, Nr. The value of numDiags in DIAGNOSTICS_SIZE.h would therefore be equal to 4*Nr, or, say 40 if Nr=10.

numLists and numperList: The parameter numLists must be set greater than or equal to the number of separate output streams that the user specifies in the namelist file data.diagnostics. The parameter numperList corresponds to the maximum number of diagnostics requested per output streams.

9.1.4.2 Adjoint variables

The diagnostics package can also be used to print adjoint state variables. Using the diagnostics package as opposed to using the standard ‘adjoint dump’ options allows one to take advantage of all the averaging and postprocessing routines available to other diagnostics variables.

Currently, the available adjoint state variables are:

<table>
<thead>
<tr>
<th>ID</th>
<th>Symbol</th>
<th>Title</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>109</td>
<td>ADJetan</td>
<td>Surface height anomaly</td>
<td>Sensitivity to sea surface height anomaly</td>
</tr>
<tr>
<td>110</td>
<td>ADJuvel</td>
<td>Velocity</td>
<td>Sensitivity to zonal velocity</td>
</tr>
<tr>
<td>111</td>
<td>ADJuvel</td>
<td>Velocity</td>
<td>Sensitivity to meridional velocity</td>
</tr>
<tr>
<td>112</td>
<td>ADJuvel</td>
<td>Velocity</td>
<td>Sensitivity to vertical velocity</td>
</tr>
<tr>
<td>113</td>
<td>ADJtheta</td>
<td>Potential temperature</td>
<td>Sensitivity to sea surface temperature</td>
</tr>
<tr>
<td>114</td>
<td>ADJsalt</td>
<td>Salinity</td>
<td>Sensitivity to salinity</td>
</tr>
<tr>
<td>115</td>
<td>ADJtaux</td>
<td>Wind stress</td>
<td>Sensitivity to zonal wind stress</td>
</tr>
<tr>
<td>116</td>
<td>ADJtauy</td>
<td>Wind stress</td>
<td>Sensitivity to meridional wind stress</td>
</tr>
<tr>
<td>117</td>
<td>ADJempmr</td>
<td>Surface Fresh-Water flux into the ocean</td>
<td>Sensitivity to net surface Fresh-Water flux into the ocean</td>
</tr>
<tr>
<td>118</td>
<td>ADJqnet</td>
<td>Heat flux into the ocean</td>
<td>Sensitivity to net heat flux into the ocean</td>
</tr>
<tr>
<td>119</td>
<td>ADJqsw</td>
<td>Short-Wave radiation</td>
<td>Sensitivity to net Short-Wave radiation</td>
</tr>
<tr>
<td>120</td>
<td>ADJsst</td>
<td>Surface Temperature</td>
<td>Sensitivity to Sea Surface Temperature</td>
</tr>
<tr>
<td>121</td>
<td>ADJssss</td>
<td>Surface Salinity</td>
<td>Sensitivity to Sea Surface Salinity</td>
</tr>
<tr>
<td>122</td>
<td>ADJbtdrg</td>
<td>Drag coefficient</td>
<td>Sensitivity to bottom drag coefficient</td>
</tr>
<tr>
<td>123</td>
<td>ADJdifkr</td>
<td>Diffusivity</td>
<td>Sensitivity to vertical diffusivity</td>
</tr>
<tr>
<td>124</td>
<td>ADJepsix</td>
<td>Eddy streamfunction</td>
<td>Sensitivity to zonal eddy streamfunction</td>
</tr>
<tr>
<td>125</td>
<td>ADJepsiy</td>
<td>Eddy streamfunction</td>
<td>Sensitivity to meridional eddy streamfunction</td>
</tr>
</tbody>
</table>
Additionally the packages `gmredi`, `ptracrs`, `exf`, and `seaice` have the following available adjoint diagnostics

<table>
<thead>
<tr>
<th>Code</th>
<th>Function</th>
<th>Variable Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADJkapgm</td>
<td>SMRA</td>
<td>dJ/d[m^2/s]</td>
<td>Sensitivity to GM → Intensity</td>
</tr>
<tr>
<td>ADJkapre</td>
<td>SMRA</td>
<td>dJ/d[m^2/s]</td>
<td>Sensitivity to Redi → Coefficient</td>
</tr>
<tr>
<td>TRAC01</td>
<td>SMR</td>
<td>mol C/m</td>
<td>Dissolved Inorganic Carbon concentration</td>
</tr>
<tr>
<td>ADJptr01</td>
<td>SMRA</td>
<td>dJ/mol C/m</td>
<td>Sensitivity to Dissolved Inorganic Carbon concentration</td>
</tr>
<tr>
<td>ADJustrs</td>
<td>UU A</td>
<td>dJ/(N/m^2)</td>
<td>Sensitivity to zonal surface wind stress</td>
</tr>
<tr>
<td>ADJvstrs</td>
<td>VV A</td>
<td>dJ/(N/m^2)</td>
<td>Sensitivity to merid. surface wind stress</td>
</tr>
<tr>
<td>ADJhflux</td>
<td>SM A</td>
<td>dJ/(W/m^2)</td>
<td>Sensitivity to upward heat flux</td>
</tr>
<tr>
<td>ADJsfux</td>
<td>SM A</td>
<td>dJ/(m/s)</td>
<td>Sensitivity to fresh water flux</td>
</tr>
<tr>
<td>ADJatemp</td>
<td>SM A</td>
<td>dJ/K</td>
<td>Sensitivity to atmos. surface temperature</td>
</tr>
<tr>
<td>ADJpreci</td>
<td>SM A</td>
<td>dJ/(m/s)</td>
<td>Sensitivity to precipitation</td>
</tr>
<tr>
<td>ADJroff</td>
<td>SM A</td>
<td>dJ/(m/s)</td>
<td>Sensitivity to river runoff</td>
</tr>
<tr>
<td>ADJsflux</td>
<td>SM A</td>
<td>dJ/(m/s)</td>
<td>Sensitivity to upward fresh water flux</td>
</tr>
<tr>
<td>ADJuwind</td>
<td>UM A</td>
<td>dJ/d(m/s)</td>
<td>Sensitivity to zonal 10-m wind speed</td>
</tr>
<tr>
<td>ADJvwind</td>
<td>VM A</td>
<td>dJ/d(m/s)</td>
<td>Sensitivity to meridional 10-m wind speed</td>
</tr>
<tr>
<td>ADJclsst</td>
<td>SM A</td>
<td>dJ/K</td>
<td>Sensitivity to restoring SST</td>
</tr>
<tr>
<td>ADJclsss</td>
<td>SM A</td>
<td>dJ/(g/kg)</td>
<td>Sensitivity to restoring SSS</td>
</tr>
<tr>
<td>ADJarea</td>
<td>SM A</td>
<td>dJ/(m^2/m^2)</td>
<td>Sensitivity to seaice fractional ice-cover</td>
</tr>
<tr>
<td>ADJheff</td>
<td>SM A</td>
<td>dJ/dm</td>
<td>Sensitivity to seaice ice thickness</td>
</tr>
<tr>
<td>ADJhsnow</td>
<td>SM A</td>
<td>dJ/dm</td>
<td>Sensitivity to seaice snow thickness</td>
</tr>
<tr>
<td>ADJjuice</td>
<td>UU A</td>
<td>dJ/(m/s)</td>
<td>Sensitivity to zonal ice velocity</td>
</tr>
<tr>
<td>ADJVice</td>
<td>VV A</td>
<td>dJ/(m/s)</td>
<td>Sensitivity to meridional ice velocity</td>
</tr>
</tbody>
</table>
Some notes to the user

1. This feature is currently untested with OpenAD.
2. This feature does not work with the divided adjoint.
3. The sensitivity to sea surface height $ADJ\eta$an is technically one time step ahead of other adjoint diagnostics printed at the same time step number. To be concrete, if $ADJ\eta$an is written via the diagnostics package at every iteration, $n$, then each field will technically correspond to the written iteration number, $n+1$. This is simply due to a technicality about when this variable is printed in relation to the adjoint pressure solve.
4. The diagStats options are not available for these variables.
5. Adjoint variables are recognized by checking the 10 character variable $diagCode$. To add a new adjoint variable, set the 4th position of $diagCode$ to A (notice this is the case for the list of available adjoint variables).

Using pkg/diagnostics for adjoint variables

1. Make sure the following flag is defined in either AUTODIFF_OPTIONS.h or ECCO_CPOPTIONS.h if that is being used.

```c
#define ALLOW_AUTODIFF_MONITOR
```

2. Be sure to increase numLists and numDiags appropriately in DIAGNOSTICS_SIZE.h. Safe values are e.g. 10-20 and 500-1000 respectively.

3. Specify desired variables in data.diagnostics as any other variable, as in the following example or as in this data.diagnostics. Note however, adjoint and forward diagnostic variables cannot be in the same list. That is, a single fields(:, :) list cannot contain both adjoint and forward variables.

```c
&DIAGNOSTICS_LIST
# ---
fields(1:5,1) = 'ADJtheta','ADJsalt ',
                'ADJuvel ','ADJvvel ','ADJwvel '
filename(1) = 'diags/adjState_3d_snaps',
frequency(1)=86400.0,
timePhase(1)=0.0,
#---
fields(1:5,2) = 'ADJtheta','ADJsalt ',
                'ADJuvel ','ADJvvel ','ADJwvel '
filename(2) = 'diags/adjState_3d_avg',
frequency(2)=86400.0,
#---
&
```

Note: the diagnostics package automatically provides a phase shift of frequency/2, so specify timePhase = 0 to match output from adjDumpFreq.
9.1.4.3 Adding new diagnostics to the code

In order to define and include as part of the diagnostic output any field that is desired for a particular experiment, two steps must be taken. The first is to enable the “User Diagnostic” in data.diagnostics. This is accomplished by adding one of the “User Diagnostic” field names (see available diagnostics): UDIAG1 through UDIAG10, for multi-level fields, or SDIAG1 through SDIAG10 for single level fields) to the data.diagnostics namelist in one of the output streams. The second step is to add a call to diagnostics_fill.F from the subroutine in which the quantity desired for diagnostic output is computed.

In order to add a new diagnostic to the permanent set of diagnostics that the main model or any package contains as part of its diagnostics menu, the subroutine diagnostics_addtolist.F should be called during the initialization phase of the main model or package. For the main model, the call should be made from subroutine diagnostics_main_init.F, and for a package, the call should probably be made from from inside the particular package’s init_fixed routine. A typical code sequence to set the input arguments to diagnostics_addtolist.F would look like:

```fortran
  diagName = 'RHOAnoma'
diagTitle = 'Density Anomaly (=Rho-rhoConst)'
diagUnits = 'kg/m^3'
diagCode = 'SMR MR'
CALL DIAGNOSTICS_ADDTOLIST( diagNum,
I   diagName, diagCode, diagUnits, diagTitle, 0, myThid )
```

If the new diagnostic quantity is associated with either a vector pair or a diagnostic counter, the diagMate argument must be provided with the proper index corresponding to the “mate”. The output argument from diagnostics_addtolist.F that is called diagNum here contains a running total of the number of diagnostics defined in the code up to any point during the run. The sequence number for the next two diagnostics defined (the two components of the vector pair, for instance) will be diagNum+1 and diagNum+2. The definition of the first component of the vector pair must fill the “mate” segment of the diagCode as diagnostic number diagNum+2. Since the subroutine increments diagNum, the definition of the second component of the vector fills the “mate” part of diagCode with diagNum. A code sequence for this case would look like:

```fortran
  diagName = 'UVEL '
diagTitle = 'Zonal Component of Velocity (m/s)'
diagUnits = 'm/s'
diagCode = 'UUR MR'
diagMate = diagNum + 2
CALL DIAGNOSTICS_ADDTOLIST( diagNum,
I   diagName, diagCode, diagUnits, diagTitle, diagMate, myThid )

  diagName = 'VVEL '
diagTitle = 'Meridional Component of Velocity (m/s)'
diagUnits = 'm/s'
diagCode = 'VVR MR'
diagMate = diagNum
CALL DIAGNOSTICS_ADDTOLIST( diagNum,
I   diagName, diagCode, diagUnits, diagTitle, diagMate, myThid )
```
9.1.4.4 MITgcm kernel available diagnostics list:

|<-Name->|<- code ->|<<-- Units -->>|<- Tile (max=80c) |
|------------------------------------------|
|SDIAG1 |SM L1|user-defined |User-Defined Surface Diagnostic #1 |
|SDIAG2 |SM L1|user-defined |User-Defined Surface Diagnostic #2 |
|SDIAG3 |SM L1|user-defined |User-Defined Surface Diagnostic #3 |
|SDIAG4 |SM L1|user-defined |User-Defined Surface Diagnostic #4 |
|SDIAG5 |SM L1|user-defined |User-Defined Surface Diagnostic #5 |
|SDIAG6 |SM L1|user-defined |User-Defined Surface Diagnostic #6 |
|SDIAG7 |SU L1|user-defined |User-Defined U.pt Surface Diagnostic #7 |
|SDIAG8 |SV L1|user-defined |User-Defined V.pt Surface Diagnostic #8 |
|SDIAG9 |UU L1|user-defined |User-Defined U.vector Surface Diag. #9 |
|SDIAG10 |VV L1|user-defined |User-Defined V.vector Surface Diag. #10 |
|UDIAG1 |SM MR|user-defined |User-Defined Model-Level Diagnostic #1 |
|UDIAG2 |SM MR|user-defined |User-Defined Model-Level Diagnostic #2 |
|UDIAG3 |SMR MR|user-defined |User-Defined Model-Level Diagnostic #3 |
|UDIAG4 |SMR MR|user-defined |User-Defined Model-Level Diagnostic #4 |
|UDIAG5 |SU MR|user-defined |User-Defined U.pt Model-Level Diag. #5 |
|UDIAG6 |SV MR|user-defined |User-Defined V.pt Model-Level Diag. #6 |
|UDIAG7 |UUR MR|user-defined |User-Defined U.vector Model-Lev Diag.#7 |
|UDIAG8 |VVR MR|user-defined |User-Defined V.vector Model-Lev Diag.#8 |
|UDIAG9 |SM ML|user-defined |User-Defined Phys-Level Diagnostic #9 |
|UDIAG10 |SM ML|user-defined |User-Defined Phys-Level Diagnostic #10 |
|SDIAGC |SM C L1|user-defined |User-Defined Counted Surface Diagnostic |
|SDIAGCC |SM L1|count |User-Defined Surface Diagnostic Counter |
|ETAN |SM M1|m |Surface Height Anomaly |
|ETAN SQ |SM P M1|m^2 |Square of Surface Height Anomaly |
|DETADT2 |SM M1|m^2/s^2 |Square of Surface Height Anomaly Tendency |
|THETA |SMR degC |Potential Temperature |
|SALT |SMR g/kg |Salinity |
|RELUHNM |SMR percent |Relative Humidity |
|SALTanom |SMR g/kg |Salt anomaly (=SALT-35; g/kg) |
|UVEL |UUR m/s |Zonal Component of Velocity (m/s) |
|VVEL |VVR m/s |Meridional Component of Velocity (m/s) |
|WVEL |WM m/s |Vertical Component of Velocity (r_units/s) |
|THETASQ |SMRP degC^2 |Square of Potential Temperature |
|SALTSQ |SMRP (g/kg)^2 |Square of Salinity |
|SALTSQan |SMRP (g/kg)^2 |Square of Salt anomaly (=SALT-35)^2 (g^2/kg^2) |
|UVELSQ |UURP m^2/s^2 |Square of Zonal Comp of Velocity (m^2/s^2) |
|VVELSQ |VVRP m^2/s^2 |Square of Meridional Comp of Velocity (m^2/s^2) |
|WVELSQ |WM P m^2/s^2 |Square of Vertical Comp of Velocity |
|UE_VEL_C UMR m/s |Eastward Velocity (m/s) (cell center) |
|VN_VEL_C VMR m/s |Northward Velocity (m/s) (cell center) |
|UV_VEL_C UMR m^2/s^2 |Product of horizontal Comp of velocity (cell center) |
|UV_VEL_Z UZR m^2/s^2 |Meridional Transport of Zonal Momentum (m^2/s^2) |
|WU_VEL |WU LR m.m/s^2 |Vertical Transport of Zonal Momentum |
|WV_VEL |WV LR m.m/s^2 |Vertical Transport of Meridional Momentum |
|UVELMASS |UUr m/s |Zonal Mass-Weighted Comp of Velocity (m/s) |
|VVELMASS |VVR m/s |Meridional Mass-Weighted Comp of Velocity (m/s) |
|WELMASS WM LR m/s |Vertical Mass-Weighted Comp of Velocity |
|PhiVEL |SMR M1|m^2/s |Horizontal Velocity Potential (m^2/s) |

(continues on next page)
<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PsiVEL</td>
<td>MR</td>
<td>Horizontal Velocity Stream-Function</td>
</tr>
<tr>
<td>UTHMASS</td>
<td>UUr</td>
<td>Zonal Mass-Weight Transp of Pot Temp</td>
</tr>
<tr>
<td>VTHMASS</td>
<td>VVr</td>
<td>Meridional Mass-Weight Transp of Pot Temp</td>
</tr>
<tr>
<td>WTHMASS</td>
<td>WM</td>
<td>Vertical Mass-Weight Transp of Pot Temp (K.m/s)</td>
</tr>
<tr>
<td>USLMASS</td>
<td>UUr</td>
<td>Zonal Mass-Weight Transp of Salinity</td>
</tr>
<tr>
<td>VSLMASS</td>
<td>VVr</td>
<td>Meridional Mass-Weight Transp of Salinity</td>
</tr>
<tr>
<td>WSLMASS</td>
<td>WM</td>
<td>Vertical Mass-Weight Transp of Salinity</td>
</tr>
<tr>
<td>UVELTH</td>
<td>UUR</td>
<td>Zonal Transport of Pot Temp</td>
</tr>
<tr>
<td>VVELTH</td>
<td>VVR</td>
<td>Meridional Transport of Pot Temp</td>
</tr>
<tr>
<td>WVELTH</td>
<td>WM</td>
<td>Vertical Transport of Pot Temp</td>
</tr>
<tr>
<td>UVELSLT</td>
<td>UUR</td>
<td>Zonal Transport of Salinity</td>
</tr>
<tr>
<td>VVELSLT</td>
<td>VVR</td>
<td>Meridional Transport of Salinity</td>
</tr>
<tr>
<td>WVELSLT</td>
<td>WM</td>
<td>Vertical Transport of Salinity</td>
</tr>
<tr>
<td>UVELPHI</td>
<td>UUr</td>
<td>Zonal Mass-Weight Transp of Pressure Pot.(p/rho)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Density Anomaly (=Rho-rhoConst)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Square of Density Anomaly (=Rho-rhoConst)^2)</td>
</tr>
<tr>
<td>URHOMASS</td>
<td>UUr</td>
<td>Zonal Transport of Density</td>
</tr>
<tr>
<td>VRHOMASS</td>
<td>VVr</td>
<td>Meridional Transport of Density</td>
</tr>
<tr>
<td>WRHOMASS</td>
<td>WM</td>
<td>Vertical Transport of Density</td>
</tr>
<tr>
<td>WdRHO_P</td>
<td>WM</td>
<td>Vertical velocity times delta^k(Rho)_at-const-P</td>
</tr>
<tr>
<td>WdRHOdP</td>
<td>WM</td>
<td>Vertical velocity times delta^k(Rho)_at-const-T,S</td>
</tr>
<tr>
<td>PHIHYD</td>
<td>SM</td>
<td>Hydrostatic Pressure Pot.(p/rho) Anomaly</td>
</tr>
<tr>
<td>PHIHYDSQ</td>
<td>SMRP</td>
<td>Square of Hyd. Pressure Pot.(p/rho) Anomaly</td>
</tr>
<tr>
<td>PHIHYDcR</td>
<td>SMR</td>
<td>Hydrostatic Pressure Pot.(p/rho) Anomaly @ const r</td>
</tr>
<tr>
<td>PHI_NH</td>
<td>SMR</td>
<td>Non-Hydrostatic Pressure Pot.(p/rho)</td>
</tr>
<tr>
<td>MXLDEPTH</td>
<td>SM</td>
<td>Mixed-Layer Depth (&gt;0)</td>
</tr>
<tr>
<td>DRHODR</td>
<td>SM</td>
<td>Stratification: d.Sigma/dr (kg/m3/r_unit)</td>
</tr>
<tr>
<td>CONVDJ</td>
<td>SMR</td>
<td>Convective Adjustment Index [0-1]</td>
</tr>
<tr>
<td>oceTAUX</td>
<td>UU</td>
<td>zonal surface wind stress, &gt;0 increases uVel</td>
</tr>
<tr>
<td>oceTAUY</td>
<td>UV</td>
<td>meridional surf. wind stress, &gt;0 increases vVel</td>
</tr>
<tr>
<td>atmPload</td>
<td>SM</td>
<td>Atmospheric pressure loading anomaly (vs surf_pRef)</td>
</tr>
<tr>
<td>sIceLoad</td>
<td>SM</td>
<td>sea-ice loading (in Mass of ice+snow / area unit)</td>
</tr>
<tr>
<td>oceFWflx</td>
<td>SM</td>
<td>net surface Fresh-Water flux into the ocean (+=down), &gt;0 decreases salinity</td>
</tr>
<tr>
<td></td>
<td></td>
<td>net surface Salt flux into the ocean (+=down), &gt;0 increases salinity</td>
</tr>
<tr>
<td></td>
<td></td>
<td>net surface heat flux into the ocean (+=down), &gt;0 increases theta</td>
</tr>
<tr>
<td></td>
<td></td>
<td>net Short-Wave radiation (+=down), &gt;0 increases theta</td>
</tr>
<tr>
<td></td>
<td></td>
<td>heating from freezing of sea-water (allowFreezing=T)</td>
</tr>
<tr>
<td>TRELAX</td>
<td>SM</td>
<td>surface temperature relaxation, &gt;0 increases theta</td>
</tr>
<tr>
<td>SRELAX</td>
<td>SM</td>
<td>surface salinity relaxation, &gt;0 increases salt</td>
</tr>
<tr>
<td>surForcT</td>
<td>SM</td>
<td>model surface forcing for Temperature, &gt;0 increases</td>
</tr>
<tr>
<td>surForcS</td>
<td>SM</td>
<td>model surface forcing for Salinity, &gt;0 increases</td>
</tr>
</tbody>
</table>

(continues on next page)
<table>
<thead>
<tr>
<th>Code</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TFLUX</td>
<td>SM U1 W/m^2</td>
<td>Total heat flux (match heat-content variations), &gt;0 increases theta</td>
</tr>
<tr>
<td>SFLUX</td>
<td>SM U1 g/m^2/s</td>
<td>Total salt flux (match salt-content variations), &gt;0 increases salt</td>
</tr>
<tr>
<td>RCENTER</td>
<td>SM MR m</td>
<td>Cell-Center Height</td>
</tr>
<tr>
<td>RSURF</td>
<td>SM M1 m</td>
<td>Surface Height</td>
</tr>
<tr>
<td>hFactorC</td>
<td>SMr 1</td>
<td>Center cell-thickness fraction [-]</td>
</tr>
<tr>
<td>hFactorW</td>
<td>SUr 1</td>
<td>Western-Edge cell-thickness fraction [-]</td>
</tr>
<tr>
<td>hFactorS</td>
<td>SVr 1</td>
<td>Southern-Edge cell-thickness fraction [-]</td>
</tr>
<tr>
<td>TOTUTEND</td>
<td>UUR m/s/day</td>
<td>Tendency of Zonal Component of Velocity</td>
</tr>
<tr>
<td>TOTVTEND</td>
<td>VVR m/s/day</td>
<td>Tendency of Meridional Component of Velocity</td>
</tr>
<tr>
<td>TOTTTEND</td>
<td>SMR degC/day</td>
<td>Tendency of Potential Temperature</td>
</tr>
<tr>
<td>TOTSTEND</td>
<td>SMR g/kg/day</td>
<td>Tendency of Salinity</td>
</tr>
<tr>
<td>MoistCor</td>
<td>SM MR W/m^2</td>
<td>Heating correction due to moist thermodynamics</td>
</tr>
<tr>
<td>HeatDiss</td>
<td>SM MR W/m^2</td>
<td>Heating from frictional dissipation</td>
</tr>
<tr>
<td>gT_Forc</td>
<td>SMR degC/s</td>
<td>Potential Temp. forcing tendency</td>
</tr>
<tr>
<td>gS_Forc</td>
<td>SMR g/kg/s</td>
<td>Salinity forcing tendency</td>
</tr>
<tr>
<td>AB_gT</td>
<td>SMR degC/s</td>
<td>Potential Temp. tendency from Adams-Bashforth</td>
</tr>
<tr>
<td>AB_gS</td>
<td>SMR g/kg/s</td>
<td>Salinity tendency from Adams-Bashforth</td>
</tr>
<tr>
<td>gTinAB</td>
<td>SMR degC/s</td>
<td>Potential Temp. tendency going in Adams-Bashforth</td>
</tr>
<tr>
<td>gSinAB</td>
<td>SMR g/kg/s</td>
<td>Salinity tendency going in Adams-Bashforth</td>
</tr>
<tr>
<td>gTinU</td>
<td>UUR m/s²</td>
<td>U momentum tendency from Adams-Bashforth</td>
</tr>
<tr>
<td>gTinV</td>
<td>VVR m/s²</td>
<td>V momentum tendency from Adams-Bashforth</td>
</tr>
<tr>
<td>gTinW</td>
<td>WM Lm/s²</td>
<td>W momentum tendency from Adams-Bashforth</td>
</tr>
<tr>
<td>TAUXEDDY</td>
<td>UU LR</td>
<td>Zonal Eddy Stress</td>
</tr>
<tr>
<td>TAUYEDDY</td>
<td>VV LR</td>
<td>Meridional Eddy Stress</td>
</tr>
<tr>
<td>U_EulerM</td>
<td>UUR m/s</td>
<td>Zonal Eulerian-Mean Velocity (m/s)</td>
</tr>
<tr>
<td>V_EulerM</td>
<td>VVR m/s</td>
<td>Meridional Eulerian-Mean Velocity (m/s)</td>
</tr>
<tr>
<td>ADVr_TH</td>
<td>WM degC.m^3/s</td>
<td>Vertical Advective Flux of Pot.Temperature</td>
</tr>
<tr>
<td>ADVx_TH</td>
<td>UU degC.m^3/s</td>
<td>Zonal Advective Flux of Pot.Temperature</td>
</tr>
<tr>
<td>ADVy_TH</td>
<td>VV degC.m^3/s</td>
<td>Meridional Advective Flux of Pot.Temperature</td>
</tr>
<tr>
<td>DFrE_TH</td>
<td>WM degC.m^3/s</td>
<td>Vertical Diffusive Flux of Pot.Temperature</td>
</tr>
<tr>
<td>DFx_E_TH</td>
<td>UU degC.m^3/s</td>
<td>Zonal Diffusive Flux of Pot.Temperature</td>
</tr>
<tr>
<td>DFy_E_TH</td>
<td>VV degC.m^3/s</td>
<td>Meridional Diffusive Flux of Pot.Temperature</td>
</tr>
<tr>
<td>DFrI_TH</td>
<td>WM degC.m^3/s</td>
<td>Vertical Diffusive Flux of Pot.Temperature</td>
</tr>
<tr>
<td>SM_x_TH</td>
<td>UM degC</td>
<td>Pot.Temp. 1st Order Moment Sx</td>
</tr>
<tr>
<td>SM_y_TH</td>
<td>VM degC</td>
<td>Pot.Temp. 1st Order Moment Sy</td>
</tr>
<tr>
<td>SM_z_TH</td>
<td>SM degC</td>
<td>Pot.Temp. 1st Order Moment Szz</td>
</tr>
<tr>
<td>SMxx_TH</td>
<td>UM degC</td>
<td>Pot.Temp. 2nd Order Moment Sxx</td>
</tr>
<tr>
<td>SMyy_TH</td>
<td>VM degC</td>
<td>Pot.Temp. 2nd Order Moment Syy</td>
</tr>
<tr>
<td>SMzz_TH</td>
<td>SM degC</td>
<td>Pot.Temp. 2nd Order Moment Szzz</td>
</tr>
<tr>
<td>SMxy_TH</td>
<td>SM degC</td>
<td>Pot.Temp. 2nd Order Moment Syy</td>
</tr>
<tr>
<td>SMxz_TH</td>
<td>UM degC</td>
<td>Pot.Temp. 2nd Order Moment Sxz</td>
</tr>
<tr>
<td>SMyz_TH</td>
<td>VM degC</td>
<td>Pot.Temp. 2nd Order Moment Syz</td>
</tr>
<tr>
<td>SM_v_TH</td>
<td>SM P (degC)^2</td>
<td>Pot.Temp. sub-grid variance</td>
</tr>
<tr>
<td>ADVr_SLT</td>
<td>WM degC.m^3/s</td>
<td>Vertical Advective Flux of Salinity</td>
</tr>
<tr>
<td>ADVx_SLT</td>
<td>UU g/kg.m³/s</td>
<td>Zonal Advective Flux of Salinity</td>
</tr>
</tbody>
</table>

(continues on next page)
<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADVy_SLT</td>
<td>Meridional Adveactive Flux of Salinity</td>
</tr>
<tr>
<td>DFx_E_SLT</td>
<td>Vertical Diffusive Flux of Salinity (Explicit part)</td>
</tr>
<tr>
<td>DFy_E_SLT</td>
<td>Zonal Diffusive Flux of Salinity</td>
</tr>
<tr>
<td>DFz_I_SLT</td>
<td>Vertical Diffusive Flux of Salinity (Implicit part)</td>
</tr>
<tr>
<td>SALTFILL</td>
<td>Filling of Negative Values of Salinity</td>
</tr>
<tr>
<td>SM_x_SLT</td>
<td>Salinity 1st Order Moment Sx</td>
</tr>
<tr>
<td>SM_y_SLT</td>
<td>Salinity 1st Order Moment Sy</td>
</tr>
<tr>
<td>SM_z_SLT</td>
<td>Salinity 1st Order Moment Sz</td>
</tr>
<tr>
<td>SMxx_SLT</td>
<td>Salinity 2nd Order Moment Sxx</td>
</tr>
<tr>
<td>SMyy_SLT</td>
<td>Salinity 2nd Order Moment Syy</td>
</tr>
<tr>
<td>SMzz_SLT</td>
<td>Salinity 2nd Order Moment Szz</td>
</tr>
<tr>
<td>SMxy_SLT</td>
<td>Salinity 2nd Order Moment Sxy</td>
</tr>
<tr>
<td>SMxz_SLT</td>
<td>Salinity 2nd Order Moment Sxz</td>
</tr>
<tr>
<td>SMyz_SLT</td>
<td>Salinity 2nd Order Moment Syz</td>
</tr>
<tr>
<td>SM_v_SLT</td>
<td>Salinity sub-grid variance</td>
</tr>
<tr>
<td>VISCAC4Z</td>
<td>Harmonic Viscosity Coefficient (2m^4/s) (Zeta Pt)</td>
</tr>
<tr>
<td>VISCACAHD</td>
<td>Harmonic Viscosity Coefficient (2m^4/s) (Div Pt)</td>
</tr>
<tr>
<td>VISCAC4W</td>
<td>Harmonic Viscosity Coefficient (2m^4/s) (W Pt)</td>
</tr>
<tr>
<td>VAHZMAX</td>
<td>CFL-MAX Harm Visc Coefficient (m^2/s) (Zeta Pt)</td>
</tr>
<tr>
<td>VA4ZMAX</td>
<td>CFL-MAX Biharm Visc Coefficient (m^4/s) (Zeta Pt)</td>
</tr>
<tr>
<td>VAHDMAX</td>
<td>CFL-MAX Harm Visc Coefficient (m^2/s) (Div Pt)</td>
</tr>
<tr>
<td>VA4DMAX</td>
<td>CFL-MAX Biharm Visc Coefficient (m^4/s) (Div Pt)</td>
</tr>
<tr>
<td>VAHZMIN</td>
<td>RE-MIN Harm Visc Coefficient (m^2/s) (Zeta Pt)</td>
</tr>
<tr>
<td>VA4ZMIN</td>
<td>RE-MIN Biharm Visc Coefficient (m^4/s) (Zeta Pt)</td>
</tr>
<tr>
<td>VAHDMIN</td>
<td>RE-MIN Harm Visc Coefficient (m^2/s) (Div Pt)</td>
</tr>
<tr>
<td>VA4DMIN</td>
<td>RE-MIN Biharm Visc Coefficient (m^4/s) (Div Pt)</td>
</tr>
<tr>
<td>VAHDLTH</td>
<td>Leith Harm Visc Coefficient (m^2/s) (Zeta Pt)</td>
</tr>
<tr>
<td>VA4DLTH</td>
<td>Leith Biharm Visc Coefficient (m^4/s) (Zeta Pt)</td>
</tr>
<tr>
<td>VAHDLTH</td>
<td>Leith Harm Visc Coefficient (m^2/s) (Div Pt)</td>
</tr>
<tr>
<td>VA4DLTH</td>
<td>Leith Biharm Visc Coefficient (m^4/s) (Div Pt)</td>
</tr>
<tr>
<td>VAHDLTH</td>
<td>LeithD Harm Visc Coefficient (m^2/s) (Zeta Pt)</td>
</tr>
<tr>
<td>VA4DLTH</td>
<td>LeithD Biharm Visc Coefficient (m^4/s) (Zeta Pt)</td>
</tr>
<tr>
<td>VAHDLTHD</td>
<td>LeithD Harm Visc Coefficient (m^2/s) (Div Pt)</td>
</tr>
<tr>
<td>VA4DLMAG</td>
<td>Smagorinsky Harm Visc Coefficient (m^4/s) (Zeta Pt)</td>
</tr>
<tr>
<td>VA4DLMAG</td>
<td>Smagorinsky Biharm Visc Coeff. (m^4/s) (Zeta Pt)</td>
</tr>
<tr>
<td>VA4DLMAG</td>
<td>Smagorinsky Harm Visc Coefficient (m^4/s) (Div Pt)</td>
</tr>
<tr>
<td>VA4DLMAG</td>
<td>Smagorinsky Biharm Visc Coeff. (m^4/s) (Div Pt)</td>
</tr>
<tr>
<td>momKE</td>
<td>Kinetic Energy (in momentum Eq.)</td>
</tr>
<tr>
<td>momHDiv</td>
<td>Horizontal Divergence (in momentum Eq.)</td>
</tr>
<tr>
<td>momVort3</td>
<td>3rd component (vertical) of Vorticity</td>
</tr>
<tr>
<td>Strain</td>
<td>Horizontal Strain of Horizontal Velocities</td>
</tr>
<tr>
<td>Tension</td>
<td>Horizontal Tension of Horizontal Velocities</td>
</tr>
<tr>
<td>Stretch</td>
<td>Vortex stretching from QG Leith dynamic viscosity</td>
</tr>
</tbody>
</table>
USidDrag | UUR | MR | m/s^2 | U momentum tendency from Side Drag
VSidDrag | VVR | MR | m/s^2 | V momentum tendency from Side Drag
Um_Diss | UUR | MR | m/s^2 | U momentum tendency from Dissipation (Explicit part)
Vm_Diss | VVR | MR | m/s^2 | V momentum tendency from Dissipation (Explicit part)
Um_ImplD | UUR | MR | m/s^2 | U momentum tendency from Dissipation (Implicit part)
Vm_ImplD | VVR | MR | m/s^2 | V momentum tendency from Dissipation (Implicit part)
Um_Advec | UUR | MR | m/s^2 | U momentum tendency from Advection terms
Vm_Advec | VVR | MR | m/s^2 | V momentum tendency from Advection terms
Um_Cori | UUR | MR | m/s^2 | U momentum tendency from Coriolis term
Vm_Cori | VVR | MR | m/s^2 | V momentum tendency from Coriolis term
Um_dPhiX | UUR | MR | m/s^2 | U momentum tendency from Pressure/Potential grad
Vm_dPhiY | VVR | MR | m/s^2 | V momentum tendency from Pressure/Potential grad
Um_Ext | UUR | MR | m/s^2 | U momentum tendency from external forcing
Vm_Ext | VVR | MR | m/s^2 | V momentum tendency from external forcing
Um_AdvZ | UUR | MR | m/s^2 | U momentum tendency from Vorticity Advection
Vm_AdvZ | VVR | MR | m/s^2 | V momentum tendency from Vorticity Advection
Um_AdvRe | UUR | MR | m/s^2 | U momentum tendency from vertical Advection
  (Explicit part)
Vm_AdvRe | VVR | MR | m/s^2 | V momentum tendency from vertical Advection
  (Explicit part)
Wm_Diss | WMr | LR | m/s^2 | W momentum tendency from Dissipation
Wm_Advec | WMr | LR | m/s^2 | W momentum tendency from Advection terms
WSidDrag | WMr | LR | m/s^2 | Vertical momentum tendency from Side Drag
botTauX | UU | U1 | N/m^2 | zonal bottom stress, >0 increases uVel
botTauY | VV | U1 | N/m^2 | meridional bottom stress, >0 increases vVel
ADVx_Um | UM | MR | m^4/s^2 | Zonal Advection Flux of U momentum
ADVy_Um | VM | MR | m^4/s^2 | Meridional Advection Flux of U momentum
ADVx_Vm | UZ | MR | m^4/s^2 | Zonal Advection Flux of V momentum
ADVy_Vm | VZ | MR | m^4/s^2 | Meridional Advection Flux of V momentum
VIScX_Um | UM | MR | m^4/s^2 | Zonal Viscous Flux of U momentum
VIScY_Um | VZ | MR | m^4/s^2 | Meridional Viscous Flux of U momentum
VISeE_Um | WU | LR | m^4/s^2 | Vertical Viscous Flux of U momentum (Explicit part)
VISrE_Um | WU | LR | m^4/s^2 | Vertical Viscous Flux of U momentum (Implicit part)
VIScX_Vm | UZ | MR | m^4/s^2 | Zonal Viscous Flux of V momentum
VIScY_Vm | VZ | MR | m^4/s^2 | Meridional Viscous Flux of V momentum
VISrE_Vm | WV | LR | m^4/s^2 | Vertical Viscous Flux of V momentum (Explicit part)
VISrI_Vm | WV | LR | m^4/s^2 | Vertical Viscous Flux of V momentum (Implicit part)

The meaning of the “code” column is explained in Table 9.1. The last character of the code, in particular, determines the number of vertical levels in the diagnostic (of the commonly used codes, “1” represents a 2-D diagnostic, “R” and “L” are multi-level diagnostics).
9.1.4.5 MITgcm packages: available diagnostics lists

For a list of the diagnostic fields available in the different MITgcm packages, follow the link to the available diagnostics listing in the manual section describing the package:

- pkg/aim_v23: available diagnostics
- pkg/exf: available diagnostics
- pkg/gchem: available diagnostics
- pkg/generic_advdiff: available diagnostics
- pkg/gridalt: available diagnostics
- pkg/gmredi: available diagnostics
- pkg/fizhi: available diagnostics
- pkg/kpp: available diagnostics
- pkg/land: available diagnostics
- pkg/mom_common: available diagnostics
- pkg/obcs: available diagnostics
- pkg/thsice: available diagnostics
- pkg/seacie: available diagnostics
- pkg/shap_filt: available diagnostics
- pkg.ptracers: available diagnostics

9.2 Fortran Native I/O: pkg/mdsio and pkg/rw

9.2.1 pkg/mdsio

9.2.1.1 Introduction

pkg/mdsio contains a group of Fortran routines intended as a general interface for reading and writing direct-access ("binary") Fortran files. pkg/mdsio routines are used by pkg/rw.

9.2.1.2 Using pkg/mdsio

pkg/mdsio is geared toward the reading and writing of floating point (Fortran REAL*4 or REAL*8) arrays. It assumes that the in-memory layout of all arrays follows the per-tile MITgcm convention.

```c
C Example of a "2D" array
   _RL anArray(1-OLx:sNx+OLx,1-OLy:sNy+OLy,nSx,nSy)

C Example of a "3D" array
   _RL anArray(1-OLx:sNx+OLx,1-OLy:sNy+OLy,1: Nr,nSx,nSy)
```

where the first two dimensions are spatial or “horizontal” indices that include a “halo” or exchange region (please see Section 6 and Section 8.2.5 which describe domain decomposition), and the remaining indices (Nr, nSx, and nSy) are often present but may or may not be necessary for a specific variable.
In order to write output, `pkg/mdsio` is called with a function such as:

```fortran
CALL MDSWRITEFIELD(fn, prec, lgf, typ, Nr, arr, irec, myIter, myThid)
```

where:
- **fn** is a `CHARACTER` string containing a file “base” name which will then be used to create file names that contain tile and/or model iteration indices
- **prec** is an integer that contains one of two globally defined values (`precFloat64` or `precFloat32`)
- **lgf** is a `LOGICAL` that typically contains the globally defined `globalFile` option which specifies the creation of globally (spatially) concatenated files
- **typ** is a `CHARACTER` string that specifies the type of the variable being written (’RL’ or ’RS’)
- **Nr** is an integer that specifies the number of vertical levels within the variable being written
- **arr** is the variable (array) to be written
- **irec** is the starting record within the output file that will contain the array
- **myIter, myThid** are integers containing, respectively, the current model iteration count and the unique thread ID for the current context of execution

As one can see from the above (generic) example, enough information is made available (through both the argument list and through common blocks) for `pkg/mdsio` to perform the following tasks:

1. open either a per-tile file such as:
   ```fortran
   uVel.0000302400.003.001.data
   ```
   or a “global” file such as
   ```fortran
   uVel.0000302400.data
   ```
2. byte-swap (as necessary) the input array and write its contents (minus any halo information) to the binary file – or to the correct location within the binary file if the `globalfile` option is used, and
3. create an ASCII–text metadata file (same name as the binary but with a `.meta` extension) describing the binary file contents (often, for later use with the MATLAB `rdmds()` utility).

Reading output with `pkg/mdsio` is very similar to writing it. A typical function call is

```fortran
CALL MDSREADFIELD(fn, prec, typ, Nr, arr, irec, myThid)
```

where variables are exactly the same as the `MDSWRITEFIELD` example provided above. It is important to note that the `lgf` argument is missing from the `MDSREADFIELD` function. By default, `pkg/mdsio` will first try to read from an appropriately named global file and, failing that, will try to read from a per-tile file.

### 9.2.1.3 Important considerations

When using `pkg/mdsio`, one should be aware of the following package features and limitations:

- **Byte-swapping:** For the most part, byte-swapping is gracefully handled. All files intended for reading/writing by `pkg/mdsio` should contain big-endian (sometimes called “network byte order”) data. By handling byte-swapping within the model, MITgcm output is more easily ported between different machines, architectures, compilers, etc. Byteswapping can be turned on/off at compile time within `pkg/mdsio` using the `_BYTESWAP0` CPP macro which is usually set within a `genmake2` options file or `optfile` (see Section 3.5.2.2). Additionally, some compilers may have byte-swap options that are speedier or more convenient to use.
• **Data types**: Data types are currently limited to single– or double–precision floating point values. These values can be converted, on-the-fly, from one to the other so that any combination of either single– or double–precision variables can be read from or written to files containing either single– or double–precision data.

• **Array sizes**: Array sizes are limited; pkg/mdsio is very much geared towards the reading/writing of per-tile (that is, domain-decomposed and halo-ed) arrays. Data that cannot be made to “fit” within these assumed sizes can be challenging to read or write with pkg/mdsio.

• **Tiling**: Tiling or domain decomposition is automatically handled by pkg/mdsio for logically rectangular grid topologies (e.g., lat-lon grids) and “standard” cubed sphere topologies. More complicated topologies will probably not be supported. pkg/mdsio can, without any coding changes, read and write to/from files that were run on the same global grid but with different tiling (grid decomposition) schemes. For example, pkg/mdsio can use and/or create identical input/output files for a “C32” cube when the model is run with either 6, 12, or 24 tiles (corresponding to 1, 2 or 4 tiles per cubed sphere face). This is one of the primary advantages that the pkg/mdsio package has over pkg/mnc.

• **Single-CPU I/O**: This option can be specified with the flag useSingleCpuIO = .TRUE. in the PARM01 namelist within the main data file. Single–CPU I/O mode is appropriate for computers (e.g., some SGI systems) where it can either speed overall I/O or solve problems where the operating system or file systems cannot correctly handle multiple threads or MPI processes simultaneously writing to the same file.

• **Meta-data**: Meta-data is written by MITgcm on a per-file basis using a second file with a .meta extension as described above. MITgcm itself does not read the *.meta files, they are there primarily for convenience during post-processing. One should be careful not to delete the meta-data files when using MATLAB post-processing scripts such as rdmds() since it relies upon them.

• **Numerous files**: If one is not careful (e.g., dumping many variables every time step over a long integration), pkg/mdsio will write copious amounts of files. The creation of both a binary (*.data) and ASCII text meta-data (*.meta) file for each output type step exacerbates the issue. Some operating systems do not gracefully handle large numbers (e.g., many thousands to millions) of files within one directory. So care should be taken to split output into smaller groups using subdirectories.

• **Overwriting output**: Overwriting of output is the default behavior for pkg/mdsio. If a model tries to write to a file name that already exists, the older file will be deleted. For this reason, MITgcm users should be careful to move output that they wish to keep into, for instance, subdirectories before performing subsequent runs that may overlap in time or otherwise produce files with identical names (e.g., Monte-Carlo simulations).

• **No “halo” information**: “Halo” information is neither written nor read by pkg/mdsio. Along the horizontal dimensions, all variables are written in an sNx–by–sNy fashion. So, although variables (arrays) may be defined at different locations on Arakawa grids [U (right/left horizontal edges), V (top/bottom horizontal edges), M (mass or cell center), or Z (vorticity or cell corner points)], they are all written using only interior (1:sNx and 1:sNy) values. For quantities defined at U, V, and M points, writing 1:sNx and 1:sNy for every tile is sufficient to ensure that all values are written globally for some grids (e.g., cubed sphere, re-entrant channels, and doubly-periodic rectangular regions). For Z points, failing to write values at the sNx+1 and sNy+1 locations means that, for some tile topologies, not all values are written. For instance, with a cubed sphere topology at least two corner values are “lost” (fail to be written for any tile) if the sNx+1 and sNy+1 values are ignored. If this is an issue, we recommend switching to pkg/mnc, which writes the sNx+1 and sNy+1 grid values for the U, V, and Z locations. Also, pkg/mnc is capable of reading and/or writing entire halo regions and more complicated array shapes which can be helpful when debugging – features that do not exist within pkg/mdsio.

<table>
<thead>
<tr>
<th>CPP Flag Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAFE_IO</td>
<td>#undef</td>
<td>if defined, stops the model from overwriting its own files</td>
</tr>
<tr>
<td>ALLOW_WHIO</td>
<td>#undef</td>
<td>I/O will include tile halos in the files</td>
</tr>
</tbody>
</table>
9.2.2 pkg/rw basic binary I/O utilities

pkg/rw provides a very rudimentary binary I/O capability for quickly writing single record direct-access Fortran binary files. It is primarily used for writing diagnostic output.

9.2.2.1 Introduction

pkg/rw is an interface to the more general pkg/mdsio package. pkg/rw can be used to write or read direct-access Fortran binary files for 2-D XY and 3-D XYZ arrays. The arrays are assumed to have been declared according to the standard MITgcm 2-D or 3-D floating point array type:

```
_C Example of declaring a standard two dimensional "long"
_C floating point type array (the _RL macro is usually
_C mapped to 64-bit floats in most configurations)
 _RL anArray(1-OLx:sNx+OLx,1-OLy:sNy+OLy,nSx,nSy)
```

Each call to a pkg/rw read or write routine will read (or write) to the first record of a file. To write direct access Fortran files with multiple records use the higher-level routines in pkg/mdsio rather than pkg/rw routines. To write self-describing files that contain embedded information describing the variables being written and the spatial and temporal locations of those variables use the pkg/mnc instead (see Section 9.3) which produces netCDF format output.

<table>
<thead>
<tr>
<th>CPP Flag Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RW_SAFE_MFLDS</td>
<td>#define</td>
<td>use READ_MFLDS in “safe” mode (set/check/unset for each file to read); involves more thread synchronization which could slow down multi-threaded run</td>
</tr>
<tr>
<td>RW_DISABLE_SMALL_OVERLAP</td>
<td>#undef</td>
<td>disable writing of small-overlap size array (to reduce memory size since those S/R do a local copy to 3-D full-size overlap array)</td>
</tr>
</tbody>
</table>

9.3 NetCDF I/O: pkg/mnc

Package pkg/mnc is a set of convenience routines written to expedite the process of creating, appending, and reading netCDF files. NetCDF is an increasingly popular self-describing file format intended primarily for scientific data sets. An extensive collection of netCDF documentation, including a user’s guide, tutorial, FAQ, support archive and other information can be obtained from UCAR’s web site http://www.unidata.ucar.edu/software/netcdf.

Since it is a “wrapper” for netCDF, pkg/mnc depends upon the Fortran-77 interface included with the standard NetCDF v3.x library which is often called libnetcdf.a. Please contact your local systems administrators or email mitgcm-support@mitgcm.org for help building and installing netCDF for your particular platform.

Every effort has been made to allow pkg/mnc and pkg/mdsio (see Section 9.2) to peacefully co-exist. In may cases, the model can read one format and write to the other. This side-by-side functionality can be used to, for instance, help convert pickup files or other data sets between the two formats.
9.3.1 Using pkg/mnc

9.3.1.1 pkg/mnc configuration:

As with all MITgcm packages, pkg/mnc can be turned on or off at compile time using the `packages.conf` file or the `genmake2 -enable=mnc` or `-disable=mnc` switches.

While pkg/mnc is likely to work “as is”, there are a few compile–time constants that may need to be increased for simulations that employ large numbers of tiles within each process. Note that the important quantity is the maximum number of tiles per process. Since MPI configurations tend to distribute large numbers of tiles over relatively large numbers of MPI processes, these constants will rarely need to be increased.

If pkg/mnc runs out of space within its “lookup” tables during a simulation, then it will provide an error message along with a recommendation of which parameter to increase. The parameters are all located within `MNC_COMMON.h` and the ones that may need to be increased are:

<table>
<thead>
<tr>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MNC_MAX_ID</td>
<td>1000</td>
<td>IDs for various low-level entities</td>
</tr>
<tr>
<td>MNC_MAX_INFO</td>
<td>400</td>
<td>IDs (mostly for object sizes)</td>
</tr>
<tr>
<td>MNC_CW_MAX_I</td>
<td>150</td>
<td>IDs for the “wrapper” layer</td>
</tr>
</tbody>
</table>

In those rare cases where pkg/mnc “out-of-memory” error messages are encountered, it is a good idea to increase the too-small parameter by a factor of 2–10 in order to avoid wasting time on an iterative compile–test sequence.

9.3.1.2 pkg/mnc Inputs:

Like most MITgcm packages, all of pkg/mnc can be turned on/off at runtime using a single flag in `data.pkg`:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>useMNC</td>
<td>L</td>
<td>.FALSE.</td>
<td>overall MNC ON/OFF switch</td>
</tr>
</tbody>
</table>

One important MNC–related flag is present in the main `data` namelist file in the `PARM03` section:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>outputTypesInclusive</td>
<td>L</td>
<td>.FALSE.</td>
<td>use all available output “types”</td>
</tr>
</tbody>
</table>

which specifies that turning on pkg/mnc for a particular type of output should not simultaneously turn off the default output method as it normally does. Usually, this option is only used for debugging purposes since it is inefficient to write output types using both pkg/mnc and pkg/mdsio or ASCII output. This option can also be helpful when transitioning from pkg/mdsio to pkg/mnc since the output can be readily compared.

For run-time configuration, most of the pkg/mnc–related model parameters are contained within a Fortran namelist file called `data.mnc`. The available parameters currently include:
### Table

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mnc_use_outdir</td>
<td>L</td>
<td>.FALSE.</td>
<td>create a directory for output</td>
</tr>
<tr>
<td>mnc_outdir_str</td>
<td>S</td>
<td>‘mnc_’</td>
<td>output directory name</td>
</tr>
<tr>
<td>mnc_outdir_date</td>
<td>L</td>
<td>.FALSE.</td>
<td>embed date in the outdir name</td>
</tr>
<tr>
<td>mnc_outdir_num</td>
<td>L</td>
<td>.TRUE.</td>
<td>optional</td>
</tr>
<tr>
<td>pickup_write_mnc</td>
<td>L</td>
<td>.TRUE.</td>
<td>use MNC to write pickup files</td>
</tr>
<tr>
<td>pickup_read_mnc</td>
<td>L</td>
<td>.TRUE.</td>
<td>use MNC to read pickup file</td>
</tr>
<tr>
<td>mnc_use_indir</td>
<td>L</td>
<td>.FALSE.</td>
<td>use a directory (path) for input</td>
</tr>
<tr>
<td>mnc_indir_str</td>
<td>S</td>
<td>‘ ’</td>
<td>input directory (or path) name</td>
</tr>
<tr>
<td>snapshot_mnc</td>
<td>L</td>
<td>.TRUE.</td>
<td>write snapshot output w/MNC</td>
</tr>
<tr>
<td>monitor_mnc</td>
<td>L</td>
<td>.TRUE.</td>
<td>write pkg/monitor output w/MNC</td>
</tr>
<tr>
<td>timeave_mnc</td>
<td>L</td>
<td>.TRUE.</td>
<td>write pkg/timeave output w/MNC</td>
</tr>
<tr>
<td>autodiff_mnc</td>
<td>L</td>
<td>.TRUE.</td>
<td>write pkg/autodiff output w/MNC</td>
</tr>
<tr>
<td>mnc_max_fsize</td>
<td>R</td>
<td>2.1e+09</td>
<td>max allowable file size (&lt;2GB)</td>
</tr>
<tr>
<td>mnc_filefreq</td>
<td>R</td>
<td>-1</td>
<td>frequency of new file creation (seconds)</td>
</tr>
<tr>
<td>readgrid_mnc</td>
<td>L</td>
<td>.FALSE.</td>
<td>read grid quantities using MNC</td>
</tr>
<tr>
<td>mnc_echo_gvtypes</td>
<td>L</td>
<td>.FALSE.</td>
<td>list pre-defined “types” (debug)</td>
</tr>
</tbody>
</table>

Unlike the older pkg/mdsio method, pkg/mnc has the ability to create or use existing output directories. If either mnc_outdir_date or mnc_outdir_num is .TRUE., then pkg/mnc will try to create directories on a per process basis for its output. This means that a single directory will be created for a non-MPI run and multiple directories (one per MPI process) will be created for an MPI run. This approach was chosen since it works safely on both shared global file systems (such as NFS and AFS) and on local (per-compute-node) file systems. And if both mnc_outdir_date and mnc_outdir_num are .FALSE., then the pkg/mnc package will assume that the directory specified in mnc_outdir_str already exists and will use it. This allows the user to create and specify directories outside of the model.

For input, pkg/mnc can use a single global input directory. This is a just convenience that allows pkg/mnc to gather all of its input files from a path other than the current working directory. As with pkg/mdsio, the default is to use the current working directory.

The flags snapshot_mnc, monitor_mnc, timeave_mnc, and autodiff_mnc allow the user to turn on pkg/mnc for particular “types” of output. If a type is selected, then pkg/mnc will be used for all output that matches that type. This applies to output from the main model and from all of the optional MITgcm packages. Mostly, the names used here correspond to the names used for the output frequencies in the main data namelist file.

The mnc_max_fsize parameter is a convenience added to help users work around common file size limitations. On many computer systems, either the operating system, the file system(s), and/or the netCDF libraries are unable to handle files greater than two or four gigabytes in size. pkg/mnc is able to work within this limitation by creating new files which grow along the netCDF “unlimited” (usually, time) dimension. The default value for this parameter is just slightly less than 2GB which is safe on virtually all operating systems. Essentially, this feature is a way to intelligently and automatically split files output along the unlimited dimension. On systems that support large file sizes, these splits can be readily concatenated (that is, un-done) using tools such as the NetCDF Operators (with nccat) which is available at http://nco.sourceforge.net.

Another way users can force the splitting of pkg/mnc files along the time dimension is the mnc_filefreq option. With it, files that contain variables with a temporal dimension can be split at regular intervals based solely upon the model time (specified in seconds). For some problems, this can be much more convenient than splitting based upon file size.

Additional pkg/mnc–related parameters may be contained within each package. Please see the individual packages for descriptions of their use of pkg/mnc.
9.3.1.3 pkg/mnc output:

Depending upon the flags used, pkg/mnc will produce zero or more directories containing one or more netCDF files as output. These files are either mostly or entirely compliant with the netCDF “CF” convention (v1.0) and any conformance issues will be fixed over time. The patterns used for file names are:

- ```«BASENAME».«tileNum».nc```
- ```«BASENAME».«nIter».«faceNum».nc```
- ```«BASENAME».«nIter».«tileNum».nc```

and examples are:

- ```grid.t001.nc, grid.t002.nc```
- ```input.0000072000.f001.nc```
- ```state.0000000000.t001.nc, surfDiag.0000036000.t001.nc```

where ```«BASENAME»``` is the name selected to represent a set of variables written together, ```«nIter»``` is the current iteration number as specified in the main data namelist input file and written in a zero-filled 10-digit format, ```«tileNum»``` is a three-or-more-digit zero-filled and t–prefixed tile number, ```«faceNum»``` is a three-or-more-digit zero-filled and f–prefixed face number, and .nc is the file suffix specified by the current netCDF “CF” conventions.

Some example ```«BASENAME»``` values are:

- **grid** contains the variables that describe the various grid constants related to locations, lengths, areas, etc.
- **state** contains the variables output at the dumpFreq time frequency
- **pickup.ckptA, pickup.ckptB** are the “rolling” checkpoint files
- **tave** contains the time-averaged quantities from the main model

All pkg/mnc output is currently done in a “file-per-tile” fashion since most NetCDF v3.x implementations cannot write safely within MPI or multi-threaded environments. This tiling is done in a global fashion and the tile numbers are appended to the base names as described above. Some scripts to manipulate pkg/mnc output are available at ```utils/matlab``` which includes a spatial “assembly” script ```mnc_assembly.m```.

More general manipulations can be performed on netCDF files with the NetCDF Operators (“NCO”) at [http://nco.sourceforge.net](http://nco.sourceforge.net) or with the Climate Data Operators (“CDO”) at [https://code.mpimet.mpg.de/projects/cdo](https://code.mpimet.mpg.de/projects/cdo).

Unlike the older pkg/mdsio routines, pkg/mnc reads and writes variables on different “grids” depending upon their location in the Arakawa C–grid. The following table provides examples:

<table>
<thead>
<tr>
<th>Name</th>
<th>C–grid location</th>
<th># in X</th>
<th># in Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>Temperature</td>
<td>mass</td>
<td>sNx</td>
<td>sNy</td>
</tr>
<tr>
<td>Salinity</td>
<td>mass</td>
<td>sNx</td>
<td>sNy</td>
</tr>
<tr>
<td>U velocity</td>
<td>U</td>
<td>sNx+1</td>
<td>sNy</td>
</tr>
<tr>
<td>V velocity</td>
<td>V</td>
<td>sNx</td>
<td>sNy+1</td>
</tr>
<tr>
<td>Vorticity</td>
<td>vorticity</td>
<td>sNx+1</td>
<td>sNy+1</td>
</tr>
</tbody>
</table>

and the intent is two–fold:

1. For some grid topologies it is impossible to output all quantities using only sNx , sNy arrays for every tile. Two examples of this failure are the missing corners problem for vorticity values on the cubed sphere and the velocity edge values for some open–boundary domains.

2. Writing quantities located on velocity or vorticity points with the above scheme introduces a very small data redundancy. However, any slight inconvenience is easily offset by the ease with which one can, on every individual tile, interpolate these values to mass points without having to perform an “exchange” (or “halo-filling”)
operation to collect the values from neighboring tiles. This makes the most common post-processing operations much easier to implement.

9.3.2 pkg/mnc Troubleshooting

9.3.2.1 Build troubleshooting:

In order to build MITgcm with pkg/mnc enabled, the NetCDF v3.x Fortran-77 (not Fortran-90) library must be available. This library is composed of a single header file (called netcdf.inc) and a single library file (usually called libnetcdf.a) and it must be built with the same compiler with compatible compiler options as the one used to build MITgcm (in other words, while one does not have to build libnetcdf.a with the same exact set of compiler options as MITgcm, one must avoid using some specific different compiler options which are incompatible, i.e., causing a compile-time or run-time error).

For more details concerning the netCDF build and install process, please visit the Getting and Building NetCDF guide which includes an extensive list of known–good netCDF configurations for various platforms.

9.3.2.2 Runtime troubleshooting:

Please be aware of the following:

- As a safety feature, the pkg/mnc does not, by default, allow pre-existing files to be appended to or overwritten. This is in contrast to the older pkg/mdsio which will, without any warning, overwrite existing files. If MITgcm aborts with an error message about the inability to open or write to a netCDF file, please check first whether you are attempting to overwrite files from a previous run.
- The constraints placed upon the “unlimited” (or “record”) dimension inherent with NetCDF v3.x make it very inefficient to put variables written at potentially different intervals within the same file. For this reason, pkg/mnc output is split into groups of files which attempt to reflect the nature of their content.
- On many systems, netCDF has practical file size limits on the order of 2–4GB (the maximum memory addressable with 32-bit pointers or pointer differences) due to a lack of operating system, compiler, and/or library support. The latest revisions of NetCDF v3.x have large file support and, on some operating systems, file sizes are only limited by available disk space.
- There is an 80 character limit to the total length of all file names. This limit includes the directory (or path) since paths and file names are internally appended. Generally, file names will not exceed the limit and paths can usually be shortened using, for example, soft links.
- pkg/mnc does not (yet) provide a mechanism for reading information from a single “global” file as can be done with pkg/mdsio. This is in progress.

9.3.3 pkg/mnc Internals

pkg/mnc is a two-level convenience library (or “wrapper”) for most of the netCDF Fortran API. Its purpose is to streamline the user interface to netCDF by maintaining internal relations (look-up tables) keyed with strings (or names) and entities such as netCDF files, variables, and attributes.

The two levels of pkg/mnc are:

Upper level

The upper level contains information about two kinds of associations:

grid type is lookup table indexed with a grid type name. Each grid type name is associated with a number of dimensions, the dimension sizes (one of which may be unlimited), and starting and ending index
arrays. The intent is to store all the necessary size and shape information for the Fortran arrays containing MITgcm–style “tile” variables (i.e., a central region surrounded by a variably-sized “halo” or exchange region as shown in Figure 6.5 and Figure 6.6).

**variable type** is a lookup table indexed by a variable type name. For each name, the table contains a reference to a grid type for the variable and the names and values of various attributes.

Within the upper level, these associations are not permanently tied to any particular netCDF file. This allows the information to be re-used over multiple file reads and writes.

Lower level

In the lower (or internal) level, associations are stored for netCDF files and many of the entities that they contain including dimensions, variables, and global attributes. All associations are on a per-file basis. Thus, each entity is tied to a unique netCDF file and will be created or destroyed when files are, respectively, opened or closed.

9.3.3.1 pkg/mnc grid–tTypes and variable–types:

As a convenience for users, pkg/mnc includes numerous routines to aid in the writing of data to netCDF format. Probably the biggest convenience is the use of pre-defined “grid types” and “variable types”. These “types” are simply look-up tables that store dimensions, indicies, attributes, and other information that can all be retrieved using a single character string.

The “grid types” are a way of mapping variables within MITgcm to netCDF arrays. Within MITgcm, most spatial variables are defined using 2–D or 3–D arrays with “overlap” regions (see Figure 6.5, a possible vertical index, and Figure 6.6) and tile indicies such as the following “U” velocity:

```
_RL uVel (1-0Lx:sNx+0Lx,1-0Ly:sNy+0Ly,Nr,nSx,nSy)
```

as defined in DYNVARS.h.

The grid type is a character string that encodes the presence and types associated with the four possible dimensions. The character string follows the format:

```
«H0»_«H1»_«H2»_«V»_«T»
```

(note the double underscore between «H2» and «V», and «V» and «T») where the terms «H0», «H1», «H2», «V», «T» can be almost any combination of the following:

<table>
<thead>
<tr>
<th>Horizontal</th>
<th>Vertical</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>H0: location</td>
<td>H1: dimensions</td>
<td>H2: halo</td>
</tr>
<tr>
<td>–</td>
<td>xy</td>
<td>Hn</td>
</tr>
<tr>
<td>U</td>
<td>x</td>
<td>Hy</td>
</tr>
<tr>
<td>V</td>
<td>y</td>
<td>c</td>
</tr>
<tr>
<td>Cen</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cor</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

A example list of all pre-defined combinations is contained in the file pkg/mnc/pre-defined_grids.txt.

The variable type is an association between a variable type name and the following items:

<table>
<thead>
<tr>
<th>Item</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>grid type</td>
<td>defines the in-memory arrangement</td>
</tr>
<tr>
<td>bi,bj dimensions</td>
<td>tiling indices, if present</td>
</tr>
</tbody>
</table>

9.3. NetCDF I/O: pkg/mnc
and is used by the mnc_cw__[R|W] subroutines for reading and writing variables.

### 9.3.3.2 Using pkg/mnc: examples

Writing variables to netCDF files can be accomplished in as few as two function calls. The first function call defines a variable type, associates it with a name (character string), and provides additional information about the indices for the tile (bi,bj) dimensions. The second function call will write the data at, if necessary, the current time level within the model.

Examples of the initialization calls can be found in the file model/src/ini_model_io.F where these function calls:

```c
C     Create MNC definitions for DYNVARS.h variables
      CALL MNC_CW_ADD_VNAME('iter' , '----t', 0,0, myThid)
      CALL MNC_CW_ADD_VATTR_TEXT('iter',1,
&       'long_name', 'iteration_count', myThid)

      CALL MNC_CW_ADD_VNAME('model_time' , '----t', 0,0, myThid)
      CALL MNC_CW_ADD_VATTR_TEXT('model_time',1,
&       'long_name', 'Model Time', myThid)
      CALL MNC_CW_ADD_VATTR_TEXT('model_time',1,'units','s', myThid)

      CALL MNC_CW_ADD_VNAME('U', 'U_xy_Hn__C__t', 4,5, myThid)
      CALL MNC_CW_ADD_VATTR_TEXT('U',1,'units','m/s', myThid)
      CALL MNC_CW_ADD_VATTR_TEXT('U',1,'coordinates','XU YU RC iter', myThid)

      CALL MNC_CW_ADD_VNAME('T', 'Cen_xy_Hn__C__t', 4,5, myThid)
      CALL MNC_CW_ADD_VATTR_TEXT('T',1,'units','degC', myThid)
      CALL MNC_CW_ADD_VATTR_TEXT('T',1,'long_name',
&       'potential_temperature', myThid)
      CALL MNC_CW_ADD_VATTR_TEXT('T',1,
&       'coordinates','XC YC RC iter', myThid)
```

initialize four VNAMEs and add one or more netCDF attributes to each.

The four variables defined above are subsequently written at specific time steps within model/src/write_state.F using the function calls:

```c
C     Write dynvars using the MNC package
      CALL MNC_CW_SET_UDIM('state', -1, myThid)
      CALL MNC_CW_I_W('I','state',0,0,'iter', myIter, myThid)
      CALL MNC_CW_RL_W('D','state',0,0,'model_time',myTime, myThid)
      CALL MNC_CW_RL_W('D','state',0,0,'U', uVel, myThid)
      CALL MNC_CW_RL_W('D','state',0,0,'T', theta, myThid)
```

While it is easiest to write variables within typical 2-D and 3-D fields where all data is known at a given time, it is also possible to write fields where only a portion (e.g., a “slab” or “slice”) is known at a given instant. An example is provided within pkg/mom_vecinv/mom_vecinv.F where an offset vector is used:

```c
IF (useMNC .AND. snapshot_mnc) THEN
      CALL MNC_CW_RL_W_OFFSET('D','mom_vi',bi,bj, 'fV', uCf,
&     offsets, myThid)
      CALL MNC_CW_RL_W_OFFSET('D','mom_vi',bi,bj, 'fU', vCf,
```

(continues on next page)
offssets, myThid)
ENDIF

...to write a 3-D field one depth slice at a time.

Each element in the offset vector corresponds (in order) to the dimensions of the “full” (or virtual) array and specifies which are known at the time of the call. A zero within the offset array means that all values along that dimension are available while a positive integer means that only values along that index of the dimension are available. In all cases, the matrix passed is assumed to start (that is, have an in-memory structure) coinciding with the start of the specified slice. Thus, using this offset array mechanism, a slice can be written along any single dimension or combinations of dimensions.

9.4 Monitor: Simulation State Monitoring Toolkit

9.4.1 Introduction

pkg/monitor is primarily intended as a convenient method for calculating and writing the following statistics:

- minimum
- maximum
- mean
- standard deviation

for spatially distributed fields. By default, pkg/monitor output is sent to the “standard output” channel where it appears as ASCII text containing a %MON string such as this example:

| (PID.TID 0000.0001) %MON time_tsnember | = | 3  |
| (PID.TID 0000.0001) %MON time_secondsf | = | 3.6000000000000E+03 |
| (PID.TID 0000.0001) %MON dynstat_eta_max | = | 1.0025466645951E-03 |
| (PID.TID 0000.0001) %MON dynstat_eta_min | = | -1.0008899950901E-03 |
| (PID.TID 0000.0001) %MON dynstat_eta_mean | = | 2.1037438449350E-14 |
| (PID.TID 0000.0001) %MON dynstat_eta_sd | = | 5.0985228723396E-04 |
| (PID.TID 0000.0001) %MON dynstat_etaDel2 | = | 3.5216706549525E-07 |
| (PID.TID 0000.0001) %MON dynstat_uvel_max | = | 3.7594045977254E-05 |
| (PID.TID 0000.0001) %MON dynstat_uvel_min | = | -2.8264287531564E-05 |
| (PID.TID 0000.0001) %MON dynstat_uvel_mean | = | 9.1369201945671E-06 |
| (PID.TID 0000.0001) %MON dynstat_uvel_sd | = | 1.686843913567E-05 |
| (PID.TID 0000.0001) %MON dynstat_uvelDel2 | = | 8.4315445301916E-08 |

pkg/monitor text can be readily parsed by the testreport script to determine, somewhat crudely but quickly, how similar the output from two experiments are when run on different platforms or before/after code changes.

pkg/monitor output can also be useful for quickly diagnosing practical problems such as CFL limitations, model progress (through iteration counts), and behavior within some packages that use it.
9.4.2 Using pkg/monitor

As with most packages, pkg/monitor can be turned on or off at compile and/or run times using the packages.conf and data.pkg files.

The monitor output can be sent to the standard output channel, to an pkg/mnc–generated file, or to both simultaneously. For pkg/mnc output, the flag monitor_mnc=.TRUE. should be set within the data.mnc file. For output to both ASCII and pkg/mnc, the flag outputTypesInclusive=.TRUE. should be set within the PARM03 section of the main data file. It should be noted that the outputTypesInclusive flag will make ALL kinds of output (that is, everything written by pkg/mdsio, pkg/mnc, and pkg/monitor) simultaneously active so it should be used only with caution — and perhaps only for debugging purposes.

<table>
<thead>
<tr>
<th>CPP Flag Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MONITOR_TEST_HFACZ</td>
<td>#undef</td>
<td>disable use of hFacZ</td>
</tr>
</tbody>
</table>

9.5 Grid Generation

The horizontal discretizations within MITgcm have been written to work with many different grid types including:

- cartesian coordinates
- spherical polar (“latitude-longitude”) coordinates
- general curvilinear orthogonal coordinates

The last of these, especially when combined with the domain decomposition capabilities of MITgcm, allows a great degree of grid flexibility. To date, general curvilinear orthogonal coordinates have been used extensively in conjunction with so-called “cubed sphere” grids. However, it is important to observe that cubed sphere arrangements are only one example of what is possible with domain-decomposed logically rectangular regions each containing curvilinear orthogonal coordinate systems. Much more sophisticated domains can be imagined and constructed.

In order to explore the possibilities of domain-decomposed curvilinear orthogonal coordinate systems, a suite of grid generation software called “SPGrid” (for SPherical Gridding) has been developed. SPGrid is a relatively new facility and papers detailing its algorithms are in preparation. Although SPGrid is new and rapidly developing, it has already demonstrated the ability to generate some useful and interesting grids.

This section provides a very brief introduction to SPGrid and shows some early results. For further information, please contact the MITgcm support list MITgcm-support@mitgcm.org.

9.5.1 Using SPGrid

The SPGrid software is not a single program. Rather, it is a collection of C++ code and MATLAB scripts that can be used as a framework or library for grid generation and manipulation. Currently, grid creation is accomplished by either directly running MATLAB scripts or by writing a C++ “driver” program. The MATLAB scripts are suitable for grids composed of a single “face” (that is, a single logically rectangular region on the surface of a sphere). The C++ driver programs are appropriate for grids composed of multiple connected logically rectangular patches. Each driver program is written to specify the shape and connectivity of tiles and the preferred grid density (that is, the number of grid cells in each logical direction) and edge locations of the cells where they meet the edges of each face. The driver programs pass this information to the SPGrid library, which generates the actual grid and produces the output files that describe it.

Currently, driver programs are available for a few examples including cubes, “lat-lon caps” (cube topologies that have conformal caps at the poles and are exactly lat-lon channels for the remainder of the domain), and some simple “embedded” regions that are meant to be used within typical cubes or traditional lat-lon grids.
To create new grids, one may start with an existing driver program and modify it to describe a domain that has a different arrangement. The number, location, size, and connectivity of grid “faces” (the name used for the logically rectangular regions) can be readily changed. Further, the number of grid cells within faces and the location of the grid cells at the face edges can also be specified.

### 9.5.1.1 SPGrid requirements

The following programs and libraries are required to build and/or run the SPGrid suite:

- **MATLAB** is a run-time requirement since many of the generation algorithms have been written as MATLAB scripts.
- The Geometric Tools Engine (a C++ library) is needed for the main “driver” code.
- The netCDF library is needed for file I/O.
- The Boost serialization library is also used for I/O:
  - a typical Linux/Unix build environment including the make utility (preferably GNU Make) and a C++ compiler (SPGrid was developed with g++ v4.x).

### 9.5.1.2 Obtaining SPGrid

The latest version can be obtained from:

### 9.5.1.3 Building SPGrid

The procedure for building is similar to many open source projects:

```bash
$ tar -xf spgrid-0.9.4.tar.gz
$ cd spgrid-0.9.4
$ export CPPFLAGS="-I/usr/include/netcdf-3"
$ export LDFLAGS="-L/usr/lib/netcdf-3"
$ .configure
$ make
```

where the CPPFLAGS and LDFLAGS environment variables can be edited to reflect the locations of all the necessary dependencies. SPGrid is known to work on Fedora Core Linux (versions 4 and 5) and is likely to work on most any Linux distribution that provides the needed dependencies.

### 9.5.1.4 Running SPGrid

Within the `src` sub-directory, various example driver programs exist. These examples describe small, simple domains and can generate the input files (formatted as either binary *.mitgrid or netCDF) used by MITgcm.

One such example is called `SpF_test_cube_cap` and it can be run with the following sequence of commands:

```bash
$ cd spgrid-0.9.4/src
$ make SpF_test_cube_cap
$ mkdir SpF_test_cube_cap.d
  ( cd SpF_test_cube_cap.d & & ln -s ../scripts/*.m . )
$ ./SpF_test_cube_cap
```

which should create a series of output files:

### 9.5. Grid Generation
where the grid_.mitgrid and grid_.nc files contain the grid information in binary and netCDF formats and the std_topology.nc file contains the information describing the connectivity (both edge–edge and corner–corner contacts) between all the faces.

9.5.2 Example Grids

The following grids are various examples created with SPGrid.

9.6 Pre– and Post–Processing Scripts and Utilities

There are numerous tools for pre-processing data, converting model output and analysis written in MATLAB, Fortran (f77 and f90) and perl. As yet they remain undocumented although many are self-documenting (MATLAB routines have “help” written into them).

Here we’ll summarize what is available but this is an ever growing resource so this may not cover everything that is out there:

9.6.1 Utilities Supplied With the Model

We supply some basic scripts with the model to facilitate conversion or reading of data into analysis software.

9.6.1.1 utils/scripts

In the directory utils/scripts, joinds and joinmds are perl scripts used to joining multi-part files created by MITgcm. Use joinmds. You will only need joinds if you are working with output older than two years (prior to c23).

9.6.1.2 utils/matlab

In the directory utils/matlab you will find several MATLAB scripts (.m files). The principle script is rdmds.m, used for reading the multi-part model output files into MATLAB. Place the scripts in your MATLAB path or change the path appropriately, then at the MATLAB prompt type:

```matlab
>> help rdmds
```

Another useful script scans the terminal output file for pkg/monitor information.

Most other scripts are for working in the curvilinear coordinate systems, and as yet are unpublished and undocumented.
9.6.1.3 pkg/mnc utils

The following scripts and utilities have been written to help manipulate netCDF files:

**Tile Assembly:** A MATLAB script `mnc_assembly.m` is available for spatially “assembling” pkg/mnc output. A convenience wrapper script called `gluemnc.m` is also provided. Please use the MATLAB help facility for more information.

**gmt:** As MITgcm evolves to handle more complicated domains and topologies, a suite of matlab tools is being written to more gracefully handle the model files. This suite is called “gmt” which refers to “generalized model topology” pre-/post-processing. Currently, this directory contains a MATLAB script `gmt/rdnctiles.m` that is able to read netCDF files for any domain. Additional scripts are being created that will work with these fields on a per-tile basis.

9.6.2 Pre-Processing Software

There is a suite of pre-processing software for interpolating bathymetry and forcing data, written by Adcroft and Biastoch. At some point, these will be made available for download. If you are in need of such software, contact one of them.

9.7 Potential Vorticity Matlab Toolbox

Author: Guillaume Maze

9.7.1 Introduction

This section of the documentation describes a MATLAB package that aims to provide useful routines to compute vorticity fields (relative, potential and planetary) and its related components. This is an offline computation. It was developed to be used in mode water studies, so that it comes with other related routines, in particular ones computing surface vertical potential vorticity fluxes.

9.7.2 Equations

9.7.2.1 Potential vorticity

The package computes the three components of the relative vorticity defined by:

$$\omega = \nabla \times U = \begin{pmatrix} \omega_x \\ \omega_y \\ \zeta \end{pmatrix} \approx \begin{pmatrix} -\frac{\partial \omega}{\partial z} \\ -\frac{\partial \omega}{\partial x} - \frac{\partial u}{\partial y} \\ -\frac{\partial \omega}{\partial y} \end{pmatrix}$$

(9.1)

where we omitted the vertical velocity component (as done throughout the package).

The package then computes the potential vorticity as:

$$Q = -\frac{1}{\rho} \omega \cdot \nabla \sigma_\theta$$

$$= -\frac{1}{\rho} \left( \omega_x \frac{\partial \sigma_\theta}{\partial x} + \omega_y \frac{\partial \sigma_\theta}{\partial y} + (f + \zeta) \frac{\partial \sigma_\theta}{\partial z} \right)$$

(9.2)

where $\rho$ is the density, $\sigma_\theta$ is the potential density (both eventually computed by the package) and $f$ is the Coriolis parameter.
The package is also able to compute the simpler planetary vorticity as:

\[ Q_{sp} = -\frac{f \sigma_0}{\rho} \partial_z \]  

(9.3)

### 9.7.2.2 Surface vertical potential vorticity fluxes

These quantities are useful in mode water studies because of the impermeability theorem which states that for a given potential density layer (embedding a mode water), the integrated PV only changes through surface input/output.

Vertical PV fluxes due to frictional and diabatic processes are given by:

\[ J^B_z = -\frac{f}{h} \left( \frac{\alpha Q_{net}}{C_w} - \rho_0 \beta S_{net} \right) \]  

(9.4)

\[ J^F_z = \frac{1}{\rho \delta_c} \vec{k} \times \vec{\tau} \cdot \nabla \sigma_m \]  

(9.5)

These components can be computed with the package. Details on the variables definition and the way these fluxes are derived can be found in Section 9.7.5.

We now give some simple explanations about these fluxes and how they can reduce the PV value of an oceanic potential density layer.

#### Diabatic process

Let's take the PV flux due to surface buoyancy forcing from (9.4) and simplify it as:

\[ J^B_z \approx -\frac{\alpha f}{hC_w} Q_{net} \]

When the net surface heat flux \( Q_{net} \) is upward, i.e., negative and cooling the ocean (buoyancy loss), surface density will increase, triggering mixing which reduces the stratification and then the PV.

- \( Q_{net} < 0 \) (upward, cooling)
- \( J^B_z > 0 \) (upward)
- \(-\rho^{-1} \nabla \cdot J^B_z < 0 \) (PV flux divergence)
- \( PV \downarrow \) where \( Q_{net} < 0 \)

#### Frictional process: “Down-front” wind-stress

Now let’s take the PV flux due to the “wind-driven buoyancy flux” from (9.5) and simplify it as:

\[ J^F_z = \frac{1}{\rho \delta_c} \left( \tau_x \frac{\partial \sigma}{\partial y} - \tau_y \frac{\partial \sigma}{\partial x} \right) \]

\[ \approx \frac{1}{\rho \delta_c} \tau_x \frac{\partial \sigma}{\partial y} \]

When the wind is blowing from the east above the Gulf Stream (a region of high meridional density gradient), it induces an advection of dense water from the northern side of the GS to the southern side through Ekman currents. Then, it induces a “wind-driven” buoyancy lost and mixing which reduces the stratification and the PV.

- \( \vec{k} \times \vec{\tau} \cdot \nabla \sigma > 0 \) ("Down-front" wind)
- \( J^F_z > 0 \) (upward)
- \(-\rho^{-1} \nabla \cdot J^F_z < 0 \) (PV flux divergence)
- \( PV \downarrow \) where \( \vec{k} \times \vec{\tau} \cdot \nabla \sigma > 0 \)
Diabatic versus frictional processes

A recent debate in the community arose about the relative role of these processes. Taking the ratio of (9.4) and (9.5) leads to:

\[
\frac{J_F^z}{J_B^z} = \frac{\frac{1}{\rho_0} \cdot \vec{k} \times \tau \cdot \nabla \sigma}{-\frac{f}{h} \left( \frac{\alpha Q_{\text{net}}}{C_w} - \rho_0 \beta S_{\text{net}} \right)} \approx \frac{Q_{E_k}/\delta_e}{Q_{\text{net}}/h}
\]

where appears the lateral heat flux induced by Ekman currents:

\[
Q_{E_k} = -\frac{C_w}{\alpha \rho f} \cdot \vec{k} \times \tau \cdot \nabla \sigma = \frac{C_w}{\alpha} \delta_e \cdot u_{E_k} \cdot \nabla \sigma
\]

which can be computed with the package. In the aim of comparing both processes, it will be useful to plot surface net and lateral Ekman-induced heat fluxes together with PV fluxes.

9.7.3 Key routines

- **A_compute_potential_density.m**: Compute the potential density field. Requires the potential temperature and salinity (either total or anomalous) and produces one output file with the potential density field (file prefix is SIGMATHETA). The routine uses `utils/matlab/densjmd95.m`, a Matlab counterpart of the MITgcm built-in function to compute the density.

- **B_compute_relative_vorticity.m**: Compute the three components of the relative vorticity defined in (9.1). Requires the two horizontal velocity components and produces three output files with the three components (files prefix are OMEGAX, OMEGAY and ZETA).

- **C_compute_potential_vorticity.m**: Compute the potential vorticity without the negative ratio by the density. Two options are possible in order to compute either the full component (term into parenthesis in (9.2)) or the planetary component \((f \partial_z \sigma_\theta, f \partial_z \theta_\phi, f \partial_z \sigma_\phi\) in (9.3)). Requires the relative vorticity components and the potential density, and produces one output file with the potential vorticity (file prefix is PV for the full term and splPV for the planetary component).

- **D_compute_potential_vorticity.m**: Load the field computed with and divide it by \(-\rho\) to obtain the correct potential vorticity. Require the density field and after loading, overwrite the file with prefix PV or splPV.

- **compute_density.m**: Compute the density \(\rho\) from the potential temperature and the salinity fields.

- **compute_JFz.m**: Compute the surface vertical PV flux due to frictional processes. Requires the wind stress components, density, potential density and Ekman layer depth (all of them, except the wind stress, may be computed with the package), and produces one output file with the PV flux \(J_F^z\) (see (9.5)) and with JFz as a prefix.

- **compute_JBz.m**: Compute the surface vertical PV flux due to diabatic processes as:

\[
J_B^z = -\frac{f}{h} \frac{\alpha Q_{\text{net}}}{C_w}
\]

which is a simplified version of the full expression given in (9.4). Requires the net surface heat flux and the mixed layer depth (of which an estimation can be computed with the package), and produces one output file with the PV flux \(J_B^z\) and with JBz as a prefix.
• **compute_QEk.m**: Compute the horizontal heat flux due to Ekman currents from the PV flux induced by frictional forces as:

\[
Q_{Ek} = -\frac{C_w \delta e}{\alpha f} f z
\]

Requires the PV flux due to frictional forces and the Ekman layer depth, and produces one output with the heat flux and with QEk as a prefix.

• **eg_main_getPV**: A complete example of how to set up a master routine able to compute everything from the package.

### 9.7.4 Technical details

#### 9.7.4.1 File name

A file name is formed by three parameters which need to be set up as global variables in MATLAB before running any routines. They are:

- the prefix, i.e., the variable name (netcdf_UVEL for example). This parameter is specified in the help section of all diagnostic routines.
- netcdf_domain: the geographical domain.
- netcdf_suff: the netcdf extension (nc or cdf for example).

Then, for example, if the calling MATLAB routine had set up:

```matlab
global netcdf_THETA netcdf_SALTanom netcdf_domain netcdf_suff
netcdf_THETA = 'THETA';
netcdf_SALTanom = 'SALT';
netcdf_domain = 'north_atlantic';
netcdf_suff = 'nc';
```

the routine `A_compute_potential_density.m` to compute the potential density field, will look for the files:

```plaintext
THETA.north_atlantic.nc
SALT.north_atlantic.nc
```

and the output file will automatically be: `SIGMATHETA.north_atlantic.nc`.

Otherwise indicated, output file prefix cannot be changed.

#### 9.7.4.2 Path to file

All diagnostic routines look for input files in a subdirectory (relative to the MATLAB routine directory) called `./netcdf-files`, which in turn is supposed to contain subdirectories for each set of fields. For example, computing the potential density for the timestep 12H00 02/03/2005 will require a subdirectory with the potential temperature and salinity files like:

```plaintext
./netcdf-files/200501031200/THETA.north_atlantic.nc
./netcdf-files/200501031200/SALT.north_atlantic.nc
```

The output file `SIGMATHETA.north\_atlantic.nc` will be created in `./netcdf-files/200501031200/`. All diagnostic routines take as argument the name of the timestep subdirectory into `./netcdf-files`.  

598 Chapter 9. Packages II - Diagnostics and I/O
9.7.4.3 Grids

With MITgcm numerical outputs, velocity and tracer fields may not be defined on the same grid. Usually, \texttt{UVEL} and \texttt{VVEL} are defined on a C-grid but when interpolated from a cube-sphere simulation they are defined on a A-grid. When it is needed, routines allow to set up a global variable which define the grid to use.

9.7.5 Notes on the flux form of the PV equation and vertical PV fluxes

9.7.5.1 Flux form of the PV equation

The conservative flux form of the potential vorticity equation is:

\[
\frac{\partial \rho Q}{\partial t} + \nabla \cdot \vec{J} = 0
\]

(9.6)

where the potential vorticity \( Q \) is given by (9.2).

The generalized flux vector of potential vorticity is:

\[
\vec{J} = \rho Q \vec{u} + \vec{N}_Q
\]

which allows to rewrite (9.6) as:

\[
\frac{DQ}{dt} = -\frac{1}{\rho} \nabla \cdot \vec{N}_Q
\]

(9.7)

where the non-advective PV flux \( \vec{N}_Q \) is given by:

\[
\vec{N}_Q = -\frac{\rho_0}{g} B \vec{\omega}_n + \vec{F} \times \nabla \sigma_\theta
\]

(9.8)

Its first component is linked to the buoyancy forcing:

\[
B = -\frac{g}{\rho_0} \frac{D\sigma_\theta}{dt}
\]

and the second one to the non-conservative body forces per unit mass:

\[
\vec{F} = \frac{D\vec{u}}{dt} + 2\Omega \times \vec{u} + \nabla \rho
\]

Note that introducing \( B \) into (9.8) yields:

\[
\vec{N}_Q = \omega_n \frac{D\sigma_\theta}{dt} + \vec{F} \times \nabla \sigma_\theta
\]

9.7.5.2 Determining the PV flux at the ocean’s surface

In the context of mode water study, we are particularly interested in how the PV may be reduced by surface PV fluxes because a mode water is characterized by a low PV value. Considering the volume limited by two \textit{iso} - \( \sigma_{\theta} \), PV flux is limited to surface processes and then vertical component of \( \vec{N}_Q \). It is supposed that \( B \) and \( \vec{F} \) will only be non-zero in the mixed layer (of depth \( h \) and variable density \( \sigma_m \)) exposed to mechanical forcing by the wind and buoyancy fluxes through the ocean’s surface.
Given the assumption of a mechanical forcing confined to a thin surface Ekman layer (of depth $\delta$, eventually computed by the package) and of hydrostatic and geostrophic balances, we can write:

$$\vec{u}_g = \frac{1}{\rho f} \vec{k} \times \nabla p$$

$$\frac{\partial p_m}{\partial t} = -\rho_m g$$

$$\frac{\partial \sigma_m}{\partial t} + \vec{u}_m \cdot \nabla \sigma_m = -\frac{\rho_0}{g} B$$

where:

$$\vec{u}_m = \vec{u}_g + \vec{u}_{Ek} + o(R_o)$$

is the full velocity field composed of the geostrophic current $\vec{u}_g$ and the Ekman drift:

$$\vec{u}_{Ek} = -\frac{1}{\rho f} \vec{k} \times \frac{\partial \tau}{\partial z}$$

(9.11)

(where $\tau$ is the wind stress) and last by other ageostrophic components of $o(R_o)$ which are neglected.

Partitioning the buoyancy forcing as:

$$B = B_g + B_{Ek}$$

(9.12)

and using (9.10) and (9.11), (9.9) becomes:

$$\frac{\partial \sigma_m}{\partial t} + \vec{u}_g \cdot \nabla \sigma_m = -\frac{\rho_0}{g} B_g$$

revealing the “wind-driven buoyancy forcing”:

$$B_{Ek} = \frac{g}{\rho_0} \frac{1}{\rho f} \left( \vec{k} \times \frac{\partial \tau}{\partial z} \right) \cdot \nabla \sigma_m$$

Note that since:

$$\frac{\partial B_g}{\partial z} = \frac{\partial}{\partial z} \left( -\frac{g}{\rho_0} \vec{u}_g \cdot \nabla \sigma_m \right) = -\frac{g}{\rho_0} \frac{\partial \vec{u}_g}{\partial z} \cdot \nabla \sigma_m = 0$$

$B_g$ must be uniform throughout the depth of the mixed layer and then being related to the surface buoyancy flux by integrating (9.12) through the mixed layer:

$$\int_{-h}^{0} B \, dz = h B_g + \int_{-h}^{0} B_{Ek} \, dz = B_{in}$$

(9.13)

where $B_{in}$ is the vertically integrated surface buoyancy (in)flux:

$$B_{in} = \frac{g}{\rho_0} \left( \frac{\alpha Q_{net}}{C_w} - \rho_0 \beta S_{net} \right)$$

(9.14)

with $\alpha \approx 2.5 \times 10^{-4} \, K^{-1}$ the thermal expansion coefficient (computed by the package otherwise), $C_w = 4187\, J\, kg^{-1}\, K^{-1}$ the specific heat of seawater, $Q_{net} \, (W\, m^{-2})$ the net heat surface flux (positive downward, warming the ocean), $\beta \, (g/kg)^{-1}$ the saline contraction coefficient, and $S_{net} = S \cdot (E - P) \, ((g/kg)\, m\, s^{-1})$ the net freshwater surface flux with $S \, (g/kg)$ the surface salinity and $(E - P) \, (m\, s^{-1})$ the fresh water flux.

Introducing the body force in the Ekman layer:

$$F_z = \frac{1}{\rho} \frac{\partial \tau}{\partial z}$$
the vertical component of (9.8) is:

\[
\vec{N}_{Qz} = -\rho_0 \left( B_g + B_{Ek} \right) \omega_z + \frac{1}{\rho} \left( \frac{\partial \tau}{\partial z} \times \nabla \sigma_m \right) \cdot \vec{k}
\]

\[
= -\rho_0 \frac{B_g \omega_z}{g} - \rho_0 \frac{g}{\rho_0 \rho f} \left( \frac{g}{\rho_0} \frac{1}{\rho f} \vec{k} \times \frac{\partial \tau}{\partial z} \cdot \nabla \sigma_m \right) \omega_z + \frac{1}{\rho} \left( \frac{\partial \tau}{\partial z} \times \nabla \sigma_\theta \right) \cdot \vec{k}
\]

\[
= -\rho_0 \frac{B_g \omega_z}{g} + \left( 1 - \frac{\omega_z}{f} \right) \left( \frac{1}{\rho_0 \rho f} \frac{1}{\rho f} \vec{k} \times \frac{\partial \tau}{\partial z} \cdot \nabla \sigma_m \right) \cdot \vec{k}
\]

and given the assumption that \( \omega_z \approx f \), the second term vanishes and we obtain:

\[
\vec{N}_{Qz} = -\rho_0 \frac{f B_g}{g}
\]

Note that the wind-stress forcing does not appear explicitly here but is implicit in \( B_g \) through (9.13): the buoyancy forcing \( B_g \) is determined by the difference between the integrated surface buoyancy flux \( B_{in} \) and the integrated “wind-driven buoyancy forcing”:

\[
B_g = \frac{1}{h} \left( B_{in} - \int_{-h}^{0} B_{Ek} dz \right)
\]

\[
= \frac{1}{h} \frac{g}{\rho_0} \left( \frac{\alpha Q_{net}}{C_w} - \rho_0 \beta S_{net} \right) - \frac{1}{h} \int_{-h}^{0} \frac{g}{\rho_0} \frac{1}{\rho f} \vec{k} \times \frac{\partial \tau}{\partial z} \cdot \nabla \sigma_m dz
\]

\[
= \frac{1}{h} \frac{g}{\rho_0} \left( \frac{\alpha Q_{net}}{C_w} - \rho_0 \beta S_{net} \right) - \frac{g}{\rho_0 \rho f} \vec{k} \times \tau \cdot \nabla \sigma_m
\]

Finally, from (9.8), the vertical surface flux of PV may be written as:

\[
\vec{N}_{Qz} = J_z^B + J_z^F
\]

\[
J_z^B = -\frac{f}{h} \left( \frac{\alpha Q_{net}}{C_w} - \rho_0 \beta S_{net} \right)
\]

\[
J_z^F = \frac{1}{\rho \delta e} \vec{k} \times \tau \cdot \nabla \sigma_m
\]

9.8 pkg/flt – Simulation of float / parcel displacements

9.8.1 Introduction

This section describes the pkg/flt package and is largely based on the original documentation provided by Arne Biastoch and Alistair Adcroft circa 2001. pkg/flt computes float trajectories and simulates the behavior of profiling floats during a model run. Profiling floats (e.g.) Argo typically drift at depth and go back to the surface at pre-defined time intervals. However, pkg/flt can also simulate observing devices such as non-profiling floats or surface drifters.

The package’s core functionalities are operated by the flt_main call in forward_step (see below for details). Checkpointing is supported via flt_write_pickup called in packages_write_pickup.

Time-stepping of float locations is based on a second- or fourth-order Runga-Kutta scheme (Press et al., 1992, Numerical Recipes). Velocities and positions are interpolated between grid points to the simulated device location, and various types of noise can be added the simulated displacements. Spatial interpolation is bilinear close to boundaries and otherwise a polynomial interpolation. Float positions are expressed in local grid index space.
### 9.8.2 Compile-time options in `FLT_OPTIONS.h`

<table>
<thead>
<tr>
<th>CPP Flag Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALLOW_3D_FLT</td>
<td>#define</td>
<td>allow three-dimensional float displacements</td>
</tr>
<tr>
<td>USE_FLT_ALT_NOISE</td>
<td>#define</td>
<td>use alternative method of adding random noise</td>
</tr>
<tr>
<td>ALLOW_FLT_3D_NOISE</td>
<td>#define</td>
<td>add noise also to the vertical velocity of 3D floats</td>
</tr>
<tr>
<td>FLT_SECOND_ORDER_RUNGE_KUTTA</td>
<td>#undef</td>
<td>revert to old second-order Runge-Kutta</td>
</tr>
<tr>
<td>FLT_WITHOUT_X_PERIODICITY</td>
<td>#undef</td>
<td>prevent floats to re-enter the opposite side of a periodic domain</td>
</tr>
<tr>
<td>FLT_WITHOUT_Y_PERIODICITY</td>
<td>#undef</td>
<td>prevent floats to re-enter the opposite side of a periodic domain</td>
</tr>
<tr>
<td>DEVEL_FLT_EXCH2</td>
<td>#undef</td>
<td>allow experimentation with pkg/flt + exch2 despite incomplete implementation</td>
</tr>
</tbody>
</table>

### 9.8.3 Compile-time parameters in `FLT_SIZE.h` include:

**parameter (max_npart_tile = 300)** is the maximum number of floats per tile. Should be smaller than the total number of floats when running on a parallel environment but as small as possible to avoid too large arrays. The model will stop if the number of floats per tile exceeds max_npart_tile at any time.

**parameter (max_npart_exch = 50)** is the maximum number of floats per tile that can be exchanged with other tiles to one side (there are 4 arrays) in one timestep. Should be generally small because only few floats leave the tile exactly at the same time.

### 9.8.4 Run-time options in `data.flt` include:

- **flt_int_traj** is the time interval in seconds to sample float position and dynamic variables (T,S,U,V,Eta). To capture the whole profile cycle of a PALACE float this has to be at least as small as the shortest surface time.

- **flt_int_prof** is the time interval in seconds to sample a whole profile of T,S,U,V (as well as positions and Eta). This has to chosen at least as small as the shortest profiling interval.

- **flt_noise** If FLT_NOISE is defined then this is the amplitude that is added to the advection velocity by the random number generator.

- **flt_file** is the base filename of the float positions without tile information and ending (e.g. `float_pos`)

- **flt_selectTrajOutp** selects variables to output following float trajectories (=0 : none ; =1 : position only ; =2 : +p,u,v,t,s)

- **flt_selectProfOutp** selects variables to output when floats profile (=0 : none ; =1 : position only ; =2 : +p,u,v,t,s)

- **flt_deltaT** is equal to `deltaTClock` by default

**FLT_iter0** is the time step when floats are initialized

**mapIniPos2Index** converts float initial positions to local, fractional indices (.TRUE. by default)

*Notes:*
- **flt_int_prof** is the time between getting profiles, not the the return cycle of the float to the surface. The latter can be specified individually for every float. Because the mechanism for returning to the surface is called in the profiling routine flt_int_prof has to be the minimum of all iup(max_npart). The subsampling of profiles can be done later in the analysis.

*Notes:*
- All profiling intervals have to be an integer multiple of `flt_int_prof`. The profile is always taken over the whole water column. For example, let’s assume that one wants a first set of floats with 5 days profiling interval and 24 hours surface time, and another one with 10 days profiling interval and 12 hours surface time. To capture all of the floats motions, one then would have to set `flt_int_traj=43200` and `flt_int_prof=432000`.

---

602 Chapter 9. Packages II - Diagnostics and I/O
9.8.5 Input Files

If \( nIter0.EQ.FLT\_Iter0 \) then \( flt\_init\_varia \) first looks for a global file (e.g. \( float\_pos.data \)). If that file does not exist, then \( flt\_init\_varia \) looks for local files (e.g. \( float\_pos.001.001.data \), etc.) or for local pickup files that have been generated during a previous model run (e.g. \( pickup\_flt.ckptA.001.001.data \), etc.).

The first line of these input file provides:

- the number of floats on that tile in the first record
- the total number of floats in the sixth record

**Notes:** when using a global file at first-time initialization both fields should be the same.

Afterwards the input files contain one 9-element double-precision record for each float:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>npart</td>
<td>A unique float identifier (1, 2, 3, …)</td>
</tr>
<tr>
<td>tstart</td>
<td>Start date of integration of float (in s)</td>
</tr>
<tr>
<td>- If tstart=-1 floats are integrated right from the beginning</td>
<td></td>
</tr>
<tr>
<td>xpart</td>
<td>X position of float (in units of XC)</td>
</tr>
<tr>
<td>ypart</td>
<td>Y position of float (in units of YC)</td>
</tr>
<tr>
<td>kpart</td>
<td>Actual vertical level of float</td>
</tr>
<tr>
<td>kfloat</td>
<td>Target level of float</td>
</tr>
<tr>
<td>- should be the same as kpart at the beginning</td>
<td></td>
</tr>
<tr>
<td>iup</td>
<td>Flag if the float</td>
</tr>
<tr>
<td>- should profile ( &gt; 0 = return cycle (in s) to surface)</td>
<td></td>
</tr>
<tr>
<td>- remain at depth ( = 0 )</td>
<td></td>
</tr>
<tr>
<td>- is a 3D float ( = -1 )</td>
<td></td>
</tr>
<tr>
<td>- should be advected WITHOUT additional noise ( = -2 ).</td>
<td></td>
</tr>
<tr>
<td>(This implies that the float is non-profiling)</td>
<td></td>
</tr>
<tr>
<td>- is a mooring ( = -3 ), i.e. the float is not advected</td>
<td></td>
</tr>
<tr>
<td>itop</td>
<td>Time of float the surface (in s)</td>
</tr>
<tr>
<td>tend</td>
<td>End date of integration of float (in s)</td>
</tr>
<tr>
<td>- If tend=-1 floats are integrated till the end of the integration</td>
<td></td>
</tr>
</tbody>
</table>

**Notes:** an example how to write a float file (\( write\_float.F \)) is included in the verification experiment documented below.

9.8.6 Output Files

The output consists of 3 sets of local files:

- \( pickup\_flt* \) : last positions of floats that can be used for restart
- \( float\_trajectories* \) : trajectories of floats and actual values at depth
- \( float\_profiles* \) : profiles throughout the whole water column
9.8.7 Verification Experiment

The verification experiment is based on exp4 (flow over a Gaussian in a channel). The two main difference is that an additional wind forcing was introduced to speed up the currents.

A few utilities are included that were supposedly used to prepare input for pkg/flt and / or visualize its output:

extra/cvfloat.F90
extra/cvprofiles.F
extra/write_float.F
input/convert_ini.m
input/read_flt_traj.m

9.8.8 Algorithm details

A summary of what flt_main.F currently does is as follows:

CALL FLT_RUNGA4
   CALL FLT_TRILINEAR
   or CALL FLT_BILINEAR
or CALL FLT_RUNGA2
   CALL FLT_TRILINEAR
   or CALL FLT_BILINEAR
CALL FLT_EXCH2
   CALL EXCH2_SEND_PUT_VEC_RL
   CALL EXCH2_RECV_GET_VEC_RL
or CALL FLT_EXCHG
   CALL EXCH_SEND_PUT_VEC_X_RL
   CALL EXCH_RECV_GET_VEC_X_RL
   CALL EXCH_SEND_PUT_VEC_Y_RL
   CALL EXCH_RECV_GET_VEC_Y_RL
CALL FLT_UP
CALL FLT_DOWN
CALL FLT_TRAJ

A summary of included fortran files is provided inside flt_main.F:

Main Routines:
C
C  o flt_main    - Integrates the floats forward and stores
C       positions and vertical profiles at specific
C       time intervals.
C  o flt_readparms - Read parameter file
C  o flt_init_fixed - Initialise fixed
C  o flt_init_varia - Initialise the floats
C  o flt_restart   - Writes restart data to file (=> renamed: flt_write_pickup)
C
C  Second Level Subroutines:
C
C  o flt_runga2    - Second order Runga-Kutta inetgration (default)
C  o flt_exchg    - Does a new distribution of floats over tiles
C       after every integration step.
C  o flt_up       - moves float to the surface (if flag is set)

(continues on next page)
C and stores profiles to file
C o flt_down - moves float to its target depth (if flag is set)
C o flt_traj - stores positions and data to file
C o flt_interp_linear - contains bilinear interpolation scheme
C o flt_mapping - contains mapping functions & subroutine
C o flt_mdsreadvector - modified mdsreadvector to read files
CHAPTER

TEN

OCEAN STATE ESTIMATION PACKAGES

This chapter describes packages that have been introduced for ocean state estimation purposes and in relation with automatic differentiation (see Automatic Differentiation). Various examples in this chapter rely on two model configurations that can be setup as explained in Test Cases For Estimation Package Capabilities.

10.1 ECCO: model-data comparisons using gridded data sets

Author: Gael Forget

The functionalities implemented in pkg/ecco are: (1) output time-averaged model fields to compare with gridded data sets; (2) compute normalized model-data distances (i.e., cost functions); (3) compute averages and transports (i.e., integrals). The former is achieved as the model runs forwards in time whereas the others occur after time-integration has completed. Following [FCH+15] the total cost function is formulated generically as

\[ J(\bar{u}) = \sum_i \alpha_i \left( \bar{d}_i^T R_i^{-1} \bar{d}_i \right) + \sum_j \beta_j \bar{u}^T \bar{u} \]  

(10.1)

\[ \bar{d}_i = P(\bar{m}_i - \bar{o}_i) \]  

(10.2)

\[ \bar{m}_i = \mathcal{S}\mathcal{D}\mathcal{M}(\bar{v}) \]  

(10.3)

\[ \bar{v} = Q(\bar{u}) \]  

(10.4)

\[ \bar{u} = \mathcal{R}(\bar{u}') \]  

(10.5)

using symbols defined in Table 10.1. Per Equation (10.1) model counterparts (\(\bar{m}_i\)) to observational data (\(\bar{o}_i\)) derive from adjustable model parameters (\(\bar{v}\)) through model dynamics integration (\(\mathcal{M}\)), diagnostic calculations (\(\mathcal{D}\)), and averaging in space and time (\(\mathcal{S}\)). Alternatively \(\mathcal{S}\) stands for subsampling in space and time in the context of Section 10.2 (PROFILES: model-data comparisons at observed locations). Plain model-data misfits (\(\bar{m}_i - \bar{o}_i\)) can be penalized directly in Eq. (10.1) but penalized misfits (\(\bar{d}_i\)) more generally derive from \(\bar{m}_i - \bar{o}_i\) through the generic \(P\) post-processor (Eq. (10.2)). Eqs. (10.4)-(10.5) pertain to model control parameter adjustment capabilities described in Section 10.3 (CTRL: Model Parameter Adjustment Capability).
### Table 10.1: Symbol used in formulating generic cost functions.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \vec{u} )</td>
<td>vector of nondimensional control variables</td>
</tr>
<tr>
<td>( \vec{v} )</td>
<td>vector of dimensional control variables</td>
</tr>
<tr>
<td>( \alpha_i, \beta_j )</td>
<td>misfit and control cost function multipliers (1 by default)</td>
</tr>
<tr>
<td>( R_i )</td>
<td>data error covariance matrix (( R_i^{-1} ) are weights)</td>
</tr>
<tr>
<td>( d_i )</td>
<td>a set of model-data differences</td>
</tr>
<tr>
<td>( \vec{o}_i )</td>
<td>observational data vector</td>
</tr>
<tr>
<td>( \vec{m}_i )</td>
<td>model counterpart to ( \vec{o}_i )</td>
</tr>
<tr>
<td>( P )</td>
<td>post-processing operator (e.g., a smoother)</td>
</tr>
<tr>
<td>( M )</td>
<td>forward model dynamics operator</td>
</tr>
<tr>
<td>( D )</td>
<td>diagnostic computation operator</td>
</tr>
<tr>
<td>( S )</td>
<td>averaging/subsampling operator</td>
</tr>
<tr>
<td>( Q )</td>
<td>Pre-processing operator</td>
</tr>
<tr>
<td>( R )</td>
<td>Pre-conditioning operator</td>
</tr>
</tbody>
</table>

### 10.1.1 Generic Cost Function

The parameters available for configuring generic cost function terms in `data.ecco` are given in Table 10.2 and examples of possible specifications are available in:

- `MITgcm_contrib/verification_other/global_oce_cs32/input/data.ecco`
- `MITgcm_contrib/verification_other/global_oce_cs32/input_ad.sens/data.ecco`
- `MITgcm_contrib/gael/verification/global_oce_llc90/input.ecco_v4/data.ecco`

The gridded observation file name is specified by `gencost_datafile`. Observational time series may be provided as on big file or split into yearly files finishing in '_1992', '_1993', etc. The corresponding \( \vec{m}_i \) physical variable is specified via the `gencost_barfile` root (see Table 10.3). A file named as specified by `gencost_barfile` gets created where averaged fields are written progressively as the model steps forward in time. After the final time step this file is re-read by `cost_generic.F` to compute the corresponding cost function term. If `gencost_outputlevel = 1` and `gencost_name='foo'` then `cost_generic.F` outputs model-data misfit fields (i.e., \( d_i \)) to a file named ‘misfit_foo.data’ for offline analysis and visualization.

In the current implementation, model-data error covariance matrices \( R_i \) omit non-diagonal terms. Specifying \( R_i \) thus boils down to providing uncertainty fields (\( \sigma_i \) such that \( R_i = \sigma_i^2 \)) in a file specified via `gencost_errfile`. By default \( \sigma_i \) is assumed to be time-invariant but a \( \sigma_i \) time series of the same length as the \( \vec{o}_i \) time series can be provided using the `variaweight` option (Table 10.4). By default cost functions are quadratic but \( d_i^T R_i^{-1} d_i \) can be replaced with \( R_i^{-1/2} d_i \) using the `nosumsq` option (Table 10.4).

In principle, any averaging frequency should be possible, but only ‘day’, ‘month’, ‘step’, and ‘const’ are implemented for `gencost_avgperiod`. If two different averaging frequencies are needed for a variable used in multiple cost function terms (e.g., daily and monthly) then an extension starting with '_' should be added to `gencost_barfile` (such as ‘_day’ and ‘_mon’).\(^1\) If two cost function terms use the same variable and frequency, however, then using a common `gencost_barfile` saves disk space.

Climatologies of \( \vec{m}_i \) can be formed from the time series of model averages in order to compare with climatologies of \( \vec{o}_i \) by activating the ‘clim’ option via `gencost_preproc` and setting the corresponding `gencost_preproc_i` integer parameter to the number of records (i.e., a # of months, days, or time steps) per climatological cycle. The generic post-processor \( (P \text{ in Eq. (10.2)}) \) also allows model-data misfits to be, for example, smoothed in space by setting `gencost_posproc` to ‘smooth’ and specifying the smoother parameters via `gencost_posproc_c` and `gencost_posproc_i` (see Table 10.4).

---

\(^1\) `ecco_check` may be missing a test for conflicting names…
Other options associated with the computation of Eq. (10.1) are summarized in Table 10.4 and further discussed below. Multiple gencost_preproc / gencost_posproc options may be specified per cost term.

In general the specification of gencost_name is optional, has no impact on the end-result, and only serves to distinguish between cost function terms amongst the model output (STDOUT.0000, STDERR.0000, costfunction000, misfit*.data). Exceptions listed in Table 10.6 however activate alternative cost function codes (in place of cost_generic.F) described in Section 10.1.3. In this section and in Table 10.3 (unlike in other parts of the manual) ‘zonal’ / ‘meridional’ are to be taken literally and these components are centered (i.e., not at the staggered model velocity points). Preparing gridded velocity data sets for use in cost functions thus boils down to interpolating them to XC / YC.

The gencost_kLev_select option allows the user to select the vertical level of a 3D model field, thereby creating a 2D field out of that slice which is used for the cost computation. For example, drifter velocities correspond to the second depth level of the grid used in ECCOv4, so model velocities are selected from this depth level to compare to the drifter observations. The user can specify this in data.ecco with: gencost_kLev_select (i) = 2, where i is replaced with the index for that cost function term.

### Table 10.2: Run-time parameters used in formulating generic cost functions and defined via *ecco_gencost_nml* namelist in data.ecco. All parameters are vectors of length NGENCOST (the # of available cost terms) except for gencost_preproc* are arrays of size NGENPPROC×NGENCOST (10 × 20 by default; can be changed in ECCO_SIZE.h at compile time).

<table>
<thead>
<tr>
<th>parameter</th>
<th>type</th>
<th>function</th>
</tr>
</thead>
<tbody>
<tr>
<td>gencost_name</td>
<td>character(*)</td>
<td>Name of cost term</td>
</tr>
<tr>
<td>gencost_barfile</td>
<td>character(*)</td>
<td>File to receive model counterpart ( \vec{m}_i ) (See Table 10.3)</td>
</tr>
<tr>
<td>gencost_datafile</td>
<td>character(*)</td>
<td>File containing observational data ( \vec{o}_i )</td>
</tr>
<tr>
<td>gencost_avgperiod</td>
<td>character(5)</td>
<td>Averaging period for ( \vec{o}_i ) and ( \vec{m}_i ) (see text)</td>
</tr>
<tr>
<td>gencost_outputlevel</td>
<td>integer</td>
<td>Greater than 0 will output misfit fields</td>
</tr>
<tr>
<td>gencost_errfile</td>
<td>character(*)</td>
<td>Uncertainty field name (not used in Section 10.1.2)</td>
</tr>
<tr>
<td>gencost_mask</td>
<td>character(*)</td>
<td>Mask file name root (used only in Section 10.1.2)</td>
</tr>
<tr>
<td>mult_gencost</td>
<td>real</td>
<td>Multiplier ( \alpha_i ) (default: 1)</td>
</tr>
<tr>
<td>gencost_preproc</td>
<td>character(*)</td>
<td>Preprocessor names</td>
</tr>
<tr>
<td>gencost_preproc_c</td>
<td>character(*)</td>
<td>Preprocessor character arguments</td>
</tr>
<tr>
<td>gencost_preproc_i</td>
<td>integer(*)</td>
<td>Preprocessor integer arguments</td>
</tr>
<tr>
<td>gencost_preproc_r</td>
<td>real(*)</td>
<td>Preprocessor real arguments</td>
</tr>
<tr>
<td>gencost_posproc</td>
<td>character(*)</td>
<td>Post-processor names</td>
</tr>
<tr>
<td>gencost_posproc_c</td>
<td>character(*)</td>
<td>Post-processor character arguments</td>
</tr>
<tr>
<td>gencost_posproc_i</td>
<td>integer(*)</td>
<td>Post-processor integer arguments</td>
</tr>
<tr>
<td>gencost_posproc_r</td>
<td>real(*)</td>
<td>Post-processor real arguments</td>
</tr>
<tr>
<td>gencost_spmin</td>
<td>real</td>
<td>Data less than this value will be omitted</td>
</tr>
<tr>
<td>gencost_spmax</td>
<td>real</td>
<td>Data greater than this value will be omitted</td>
</tr>
<tr>
<td>gencost_spzero</td>
<td>real</td>
<td>Data points equal to this value will be omitted</td>
</tr>
<tr>
<td>gencost_startdate1</td>
<td>integer</td>
<td>Start date of observations (YYYYMMDD)</td>
</tr>
<tr>
<td>gencost_startdate2</td>
<td>integer</td>
<td>Start date of observations (HHMMSS)</td>
</tr>
<tr>
<td>gencost_is3d</td>
<td>logical</td>
<td>Needs to be true for 3D fields</td>
</tr>
<tr>
<td>gencost_enddate1</td>
<td>integer</td>
<td>Not fully implemented (used only in Section 10.1.3)</td>
</tr>
<tr>
<td>gencost_enddate2</td>
<td>integer</td>
<td>Not fully implemented (used only in Section 10.1.3)</td>
</tr>
<tr>
<td>gencost_kLev_select</td>
<td>integer</td>
<td>Vertical level of a 3D field to create a 2D field for cost computation</td>
</tr>
</tbody>
</table>
Table 10.3: Implemented gencost_barfile options (as of checkpoint 65z) that can be used via cost_generic.F (Section 10.1.1). An extension starting with ‘_’ can be appended at the end of the variable name to distinguish between separate cost function terms. Note: the ‘m_eta’ formula depends on the ATMOSPHERIC_LOADING and ALLOW_PSBAR_STERIC compile-time options and ‘useRealFreshWaterFlux’ run-time parameter.

<table>
<thead>
<tr>
<th>variable name</th>
<th>description</th>
<th>remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>m_eta</td>
<td>sea surface height</td>
<td>free surface + ice + global steric correction</td>
</tr>
<tr>
<td>m_sst</td>
<td>sea surface temperature</td>
<td>first level potential temperature</td>
</tr>
<tr>
<td>m_sss</td>
<td>sea surface salinity</td>
<td>first level salinity</td>
</tr>
<tr>
<td>m_bp</td>
<td>bottom pressure</td>
<td>phiHydLow</td>
</tr>
<tr>
<td>m_siarea</td>
<td>sea-ice area</td>
<td>from pkg/seaice</td>
</tr>
<tr>
<td>m_siheff</td>
<td>sea-ice effective thickness</td>
<td>from pkg/seaice</td>
</tr>
<tr>
<td>m_sihsnow</td>
<td>snow effective thickness</td>
<td>from pkg/seaice</td>
</tr>
<tr>
<td>m_theta</td>
<td>potential temperature</td>
<td>three-dimensional</td>
</tr>
<tr>
<td>m_salt</td>
<td>salinity</td>
<td>three-dimensional</td>
</tr>
<tr>
<td>m_Ue</td>
<td>zonal velocity</td>
<td>three-dimensional</td>
</tr>
<tr>
<td>m_VN</td>
<td>meridional velocity</td>
<td>three-dimensional</td>
</tr>
<tr>
<td>m_ustress</td>
<td>zonal wind stress</td>
<td>from pkg/exf</td>
</tr>
<tr>
<td>m_vstress</td>
<td>meridional wind stress</td>
<td>from pkg/exf</td>
</tr>
<tr>
<td>m_uwind</td>
<td>zonal wind</td>
<td>from pkg/exf</td>
</tr>
<tr>
<td>m_vwind</td>
<td>meridional wind</td>
<td>from pkg/exf</td>
</tr>
<tr>
<td>m_atemp</td>
<td>atmospheric temperature</td>
<td>from pkg/exf</td>
</tr>
<tr>
<td>m_aqh</td>
<td>atmospheric specific humidity</td>
<td>from pkg/exf</td>
</tr>
<tr>
<td>m_precip</td>
<td>precipitation</td>
<td>from pkg/exf</td>
</tr>
<tr>
<td>m_swdown</td>
<td>downward shortwave</td>
<td>from pkg/exf</td>
</tr>
<tr>
<td>m_lwdown</td>
<td>downward longwave</td>
<td>from pkg/exf</td>
</tr>
<tr>
<td>m_wspeed</td>
<td>wind speed</td>
<td>from pkg/exf</td>
</tr>
<tr>
<td>m_diffkr</td>
<td>vertical/diapycnal diffusivity</td>
<td>three-dimensional, constant</td>
</tr>
<tr>
<td>m_kapgm</td>
<td>GM diffusivity</td>
<td>three-dimensional, constant</td>
</tr>
<tr>
<td>m_kapredi</td>
<td>isopycnal diffusivity</td>
<td>three-dimensional, constant</td>
</tr>
<tr>
<td>m_geothermalflux</td>
<td>geothermal heat flux</td>
<td>constant</td>
</tr>
<tr>
<td>m_bottomdrag</td>
<td>bottom drag</td>
<td>constant</td>
</tr>
</tbody>
</table>

Table 10.4: gencost_preproc and gencost_posproc options implemented as of checkpoint 65z. Note: the distinction between gencost_preproc and gencost_posproc seems unclear and may be revisited in the future.

<table>
<thead>
<tr>
<th>name</th>
<th>description</th>
<th>gencost_preproc_i, _r, or _c</th>
</tr>
</thead>
<tbody>
<tr>
<td>gencost_preproc</td>
<td></td>
<td></td>
</tr>
<tr>
<td>clim</td>
<td>Use climatological misfits</td>
<td>integer: no. of records per climatological cycle</td>
</tr>
<tr>
<td>mean</td>
<td>Use time mean of misfits</td>
<td>—</td>
</tr>
<tr>
<td>anom</td>
<td>Use anomalies from time mean</td>
<td>—</td>
</tr>
<tr>
<td>variaweight</td>
<td>Use time-varying weight ( W_i )</td>
<td>—</td>
</tr>
<tr>
<td>nosumsq</td>
<td>Use linear misfits</td>
<td>—</td>
</tr>
<tr>
<td>factor</td>
<td>Multiply ( \bar{m}_i ) by a scaling factor</td>
<td>real: the scaling factor</td>
</tr>
<tr>
<td>offset</td>
<td>subtract mean misfit</td>
<td>—</td>
</tr>
<tr>
<td>mindepth</td>
<td>mask (ignore) misfit above minimum depth</td>
<td>real: minimum water depth (&lt; 0)</td>
</tr>
<tr>
<td>gencost_posproc</td>
<td>Smooth misfits</td>
<td>character: smoothing scale file</td>
</tr>
<tr>
<td>smooth</td>
<td>Smooth misfits</td>
<td>integer: smoother # of time steps</td>
</tr>
</tbody>
</table>
10.1.2 Generic Integral Function

The functionality described in this section is operated by `cost_gencost_boxmean.F`. It is primarily aimed at obtaining a mechanistic understanding of a chosen physical variable via adjoint sensitivity computations (see Automatic Differentiation) as done for example in [FWL+15, HWP+11, MGZ+99]. Thus the quadratic term in Eq. (10.1) $(\vec{d}_i^T R_i^{-1} \vec{d}_i)$ is by default replaced with a $d_i$ scalar that derives from model fields through a generic integral formula (Eq. (10.3)). The specification of `gencost_barfile` again selects the physical variable type. Current valid options to use `cost_gencost_boxmean.F` are reported in Table 10.5. A suffix starting with ‘-‘ can again be appended to `gencost_barfile`.

The integral formula is defined by masks provided via binary files which names are specified via `gencost_mask`. There are two cases: (1) if `gencost_mask = ‘foo_mask’` and `gencost_barfile` is of the ‘m_boxmean*’ type then the model will search for horizontal, vertical, and temporal mask files named `foo_maskC`, `foo_maskK`, and `foo_maskT`; (2) if instead `gencost_barfile` is of the ‘m_horflux_‘ type then the model will search for `foo_maskW`, `foo_maskS`, `foo_maskK`, and `foo_maskT`.

The ‘C’ mask or the ‘W’ / ‘S’ masks are expected to be two-dimensional fields. The ‘K’ and ‘T’ masks (both optional; all 1 by default) are expected to be one-dimensional vectors. The ‘K’ vector length should match Nr. The ‘T’ vector length should match the # of records that the specification of `gencost_avgperiod` implies but there is no restriction on its values. In case #1 (‘m_boxmean*’) the ‘C’ and ‘K’ masks should consists of +1 and 0 values and a volume average will be computed accordingly. In case #2 (‘m_horflux*’) the ‘W’, ‘S’, and ‘K’ masks should consists of +1, -1, and 0 values and an integrated horizontal transport (or overturn) will be computed accordingly.

Table 10.5: Implemented gencost_barfile options (as of checkpoint 67x) that can be used via `cost_gencost_boxmean.F` (Section 10.1.2).

<table>
<thead>
<tr>
<th>variable name</th>
<th>description</th>
<th>remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>m_boxmean_theta</td>
<td>mean of theta over box</td>
<td>specify box</td>
</tr>
<tr>
<td>m_boxmean_salt</td>
<td>mean of salt over box</td>
<td>specify box</td>
</tr>
<tr>
<td>m_boxmean_omega</td>
<td>mean of SSH over box</td>
<td>specify box</td>
</tr>
<tr>
<td>m_boxmean_shifwf</td>
<td>total shelfice freshwater flux over box</td>
<td>specify box</td>
</tr>
<tr>
<td>m_boxmean_shihf</td>
<td>total shelfice heat flux over box</td>
<td>specify box</td>
</tr>
<tr>
<td>m_horflux_vol</td>
<td>volume transport through section</td>
<td>specify transect</td>
</tr>
</tbody>
</table>

10.1.3 Custom Cost Functions

This section (very much a work in progress...) pertains to the special cases of `cost_gencost_bpv4.F`, `cost_gencost_seaicev4.F`, `cost_gencost_sshv4.F`, `cost_gencost_sstv4.F`, `cost_gencost_transp.F`, and `cost_gencost_moc.F`. The `cost_gencost_transp.F` function can be used to compute a transport of volume, heat, or salt through a specified section (non quadratic cost function). To this end one sets `gencost_name = ‘transp*’`, where * is an optional suffix starting with ‘-‘, and set `gencost_barfile` to one of `m_trVol`, `m_trHeat`, and `m_trSalt`.

The `cost_gencost_moc.F` function is similar to transport function, but is intended to compute the meridional overturning streamfunction maximum based on the volumetric transport integrated from the floor to surface, as in Smith and Heimbach (2019) [SH19]. Therefore, this function is intended to work with `gencost_barfile = m_trVol`, and note that the first 3 characters of `gencost_name` must be `moc`, as depicted in Table 10.6. Users can specify a latitude band to compute the MOC with appropriately defined West (‘W’) and South (‘S’) masks as described in Section 10.1.2. As an example see parameter group (3) in this data.ecco file.

Note: the functionality in `cost_gencost_transp.F` is not regularly tested. Users interested in computing volumetric transports through a section are recommended to use the `m_horflux_vol` capabilities described above as it is regularly tested. Users interested in computing heat and salt transport should note the following about `m_trHeat` and `m_trSalt`:

---

2 The quadratic option in fact does not yet exist in `cost_gencost_boxmean.F`...
1. The associated advection scheme with transports may be inconsistent with the model unless `ENUM_CENTERED_2ND` is implemented
2. Bolus velocities are not included
3. Diffusion components are not included

<table>
<thead>
<tr>
<th>name</th>
<th>description</th>
<th>remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>sshv4-mdt</td>
<td>sea surface height</td>
<td>mean dynamic topography (SSH - geod)</td>
</tr>
<tr>
<td>sshv4-tp</td>
<td>sea surface height</td>
<td>Along-Track Topex/Jason SLA (level 3)</td>
</tr>
<tr>
<td>sshv4-ers</td>
<td>sea surface height</td>
<td>Along-Track ERS/Envisat SLA (level 3)</td>
</tr>
<tr>
<td>sshv4-gfo</td>
<td>sea surface height</td>
<td>Along-Track GFO class SLA (level 3)</td>
</tr>
<tr>
<td>sshv4-lsc</td>
<td>sea surface height</td>
<td>Large-Scale SLA (from the above)</td>
</tr>
<tr>
<td>sshv4-gmsl</td>
<td>sea surface height</td>
<td>Global-Mean SLA (from the above)</td>
</tr>
<tr>
<td>bpv4-grace</td>
<td>bottom pressure</td>
<td>GRACE maps (level 4)</td>
</tr>
<tr>
<td>sstv4-amsre</td>
<td>sea surface temperature</td>
<td>Along-Swath SST (level 3)</td>
</tr>
<tr>
<td>sstv4-amsre-lsc</td>
<td>sea surface temperature</td>
<td>Large-Scale SST (from the above)</td>
</tr>
<tr>
<td>si4-cons</td>
<td>sea ice concentration</td>
<td>needs sea-ice adjoint (level 4)</td>
</tr>
<tr>
<td>si4-deconc</td>
<td>model sea ice deficiency</td>
<td>proxy penalty (from the above)</td>
</tr>
<tr>
<td>si4-exconc</td>
<td>model sea ice excess</td>
<td>proxy penalty (from the above)</td>
</tr>
<tr>
<td>transp_trVol</td>
<td>volume transport</td>
<td>specify masks (Section 10.1.2)</td>
</tr>
<tr>
<td>transp_trHeat</td>
<td>heat transport</td>
<td>specify masks (Section 10.1.2)</td>
</tr>
<tr>
<td>transp_trSalt</td>
<td>salt transport</td>
<td>specify masks (Section 10.1.2)</td>
</tr>
<tr>
<td>moc_trVol</td>
<td>meridional ovtr. streamfn. maximum</td>
<td>specify masks (Section 10.1.2)</td>
</tr>
</tbody>
</table>

### 10.1.4 Key Routines


### 10.1.5 Compile Options

TBA ... ALLOW_GENCOST_CONTRIBUTION, ALLOW_GENCOST3D, ALLOW_PSBAR_STERIC, ALLOW_SHALLOW_ALTIMETRY, ALLOW_HIGHLAT_ALTIMETRY, ALLOW_PROFILES_CONTRIBUTION, ALLOW_ECCO_OLD_FC_PRINT, ...

packages required for some functionalities: smooth, profiles, ctrl

### 10.2 PROFILES: model-data comparisons at observed locations

Author: Gael Forget

The purpose of pkg/profiles is to allow sampling of MITgcm runs according to a chosen pathway (after a ship or drifter, along altimeter tracks, etc.), typically leading to easy model-data comparisons. Given input files that contain positions and dates, pkg/profiles will interpolate the model trajectory at the observed location. In particular, pkg/profiles can be used to do model-data comparison online and formulate a least-squares problem (ECCO application).

The pkg/profiles namelist is called data.profiles. In the example below, it includes two input netcdf file names (ARGOIfremer_r8.nc and XBT_v5.nc) that should be linked to the run directory and cost function multipliers that only matter in the context of automatic differentiation (see Automatic Differentiation). The first index is a file number and
the second index (in mult* only) is a variable number. By convention, the variable number is an integer ranging 1 to 6: temperature, salinity, zonal velocity, meridional velocity, sea surface height anomaly, and passive tracer.

The netcdf input file structure is illustrated in the case of XBT_v5.nc To create such files, one can use the MITprof matlab toolbox obtained from https://github.com/gaelforget/MITprof. At run time, each file is scanned to determine which variables are included; these will be interpolated. The (final) output file structure is similar but with interpolated model values in prof_T etc., and it contains model mask variables (e.g. prof_Tmask). The very model output consists of one binary (or netcdf) file per processor. The final netcdf output is to be built from those using netcdf_ecco_recompose.m (offline).

When the k2 option is used (e.g. for cubed sphere runs), the input file is to be completed with interpolation grid points and coefficients computed offline using netcdf_ecco_GenericgridMain.m. Typically, you would first provide the standard namelist and files. After detecting that interpolation information is missing, the model will generate special grid files (profilesXCincl1PointOverlap* etc.) and then stop. You then want to run netcdf_ecco_GenericgridMain.m using the special grid files. This operation could eventually be inlined.

Example: data.profiles

```bash
# # \*****************
# PROFILES cost function
# \*****************
&PROFILES_NML

# profilesfiles(1) = 'ARGOifremer_r8',
mult_profiles(1,1) = 1.,
mult_profiles(1,2) = 1.,
profilesfiles(2) = 'XBT_v5',
mult_profiles(2,1) = 1.,
#
/

Example: XBT_v5.nc

```netcdf XBT_v5 {
dimensions:
iPROF = 278026 ;
iDEPTH = 55 ;
lTXT = 30 ;
variables:
double depth(iDEPTH) ;
depth:units = "meters" ;
double prof_YYYMDDD(iPROF) ;
prof_YYYMDDD:missing_value = -9999. ;
prof_YYYMDDD:long_name = "year (4 digits), month (2 digits), day (2 digits)" ;
double prof_HHMMSS(iPROF) ;
prof_HHMMSS:missing_value = -9999. ;
prof_HHMMSS:long_name = "hour (2 digits), minute (2 digits), second (2 digits)" ;
double prof_lon(iPROF) ;
prof_lon:units = "(degree E)" ;
prof_lon:missing_value = -9999. ;
double prof_lat(iPROF) ;
prof_lat:units = "(degree N)" ;
prof_lat:missing_value = -9999. ;
char prof_descr(iPROF, lTXT) ;
```(continues on next page)
prof_descr:long_name = "profile description" ;
double prof_T(iPROF, iDEPTH) ;
prof_T:long_name = "potential temperature" ;
prof_T:units = "degree Celsius" ;
prof_T:missing_value = -9999. ;
double prof_Tweight(iPROF, iDEPTH) ;
prof_Tweight:long_name = "weights" ;
prof_Tweight:units = "(degree Celsius)-2" ;
prof_Tweight:missing_value = -9999. ;
}

10.3 CTRL: Model Parameter Adjustment Capability

Author: Gael Forget

Package ctrl provides an interface to defining the control variables for an optimization. After defining CPP-flags ALLOW_GENTIM2D_CONTROL, ALLOW_GENARR2D_CONTROL, ALLOW_GENARR3D_CONTROL in CTRL_OPTIONS.h <pkg/ctrl/CTRL_OPTIONS.h, the parameters available for configuring generic cost terms in data.ctrl are given in Table 10.7. The control variables are stored as fields on the model grid in files $ctrlvar, $iternumber.data/meta, and corresponding gradients in ad$ctrlvar.$iternumber.data/meta, where $ctrl is defined in data.ctrl (see Table 10.8 for possible options) and $iternumber is the 10-digit iteration number of the optimization. Further, ctrl maps the gradient fields to a vector that can be handed over to an optimization routine (see Section 10.5) and maps the resulting new control vector to the model grid unless CPP-flag EXCLUDE_CTRL_PACK is defined in CTRL_OPTIONS.h.

Table 10.7: Parameters in ctrl_mnl_genarr namelist in data.ctrl.

<table>
<thead>
<tr>
<th>parameter</th>
<th>type</th>
<th>function</th>
</tr>
</thead>
<tbody>
<tr>
<td>xx_gen*_file</td>
<td>character(*)</td>
<td>Control Name: prefix from Table 10.8 + suffix.</td>
</tr>
<tr>
<td>xx_gen*_weight</td>
<td>character(*)</td>
<td>Weights in the form of $\sigma_{u,j}^{-2}$</td>
</tr>
<tr>
<td>xx_gen*_bounds</td>
<td>real(5)</td>
<td>Apply bounds</td>
</tr>
<tr>
<td>xx_gen*_preproc</td>
<td>character(*)</td>
<td>Control preprocessor(s) (see Table 10.9)</td>
</tr>
<tr>
<td>xx_gen*_preproc_c</td>
<td>character(*)</td>
<td>Preprocessor character arguments (see Table 10.10)</td>
</tr>
<tr>
<td>xx_gen*_preproc_i</td>
<td>integer(*)</td>
<td>Preprocessor integer arguments</td>
</tr>
<tr>
<td>xx_gen*_preproc_r</td>
<td>real(*)</td>
<td>Preprocessor real arguments</td>
</tr>
<tr>
<td>gen*Precond</td>
<td>real</td>
<td>Preconditioning factor (= 1 by default)</td>
</tr>
<tr>
<td>mult_gen*</td>
<td>real</td>
<td>Cost function multiplier $\beta_j$ (= 1 by default)</td>
</tr>
<tr>
<td>xx_gentim2d_period</td>
<td>real</td>
<td>Frequency of adjustments (in seconds)</td>
</tr>
<tr>
<td>xx_gentim2d_startdate1</td>
<td>integer</td>
<td>Adjustment start date</td>
</tr>
<tr>
<td>xx_gentim2d_startdate2</td>
<td>integer</td>
<td>Default: model start date</td>
</tr>
<tr>
<td>xx_gentim2d_cumsum</td>
<td>logical</td>
<td>Accumulate control adjustments</td>
</tr>
<tr>
<td>xx_gentim2d_glosum</td>
<td>logical</td>
<td>Global sum of adjustment (output is still 2D)</td>
</tr>
</tbody>
</table>
Table 10.8: Generic control prefixes implemented as of checkpoint 67x.

<table>
<thead>
<tr>
<th>name</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2D, time-invariant controls</td>
<td></td>
</tr>
<tr>
<td>genarr2d</td>
<td></td>
</tr>
<tr>
<td>xx_etan</td>
<td>initial sea surface height</td>
</tr>
<tr>
<td>xx_bottomdrag</td>
<td>bottom drag</td>
</tr>
<tr>
<td>xx_geothermal</td>
<td>geothermal heat flux</td>
</tr>
<tr>
<td>xx_shicofft</td>
<td>shelfice thermal transfer coefficient (see Section 10.3.1)</td>
</tr>
<tr>
<td>xx_shicofffs</td>
<td>shelfice salinity transfer coefficient (see Section 10.3.1)</td>
</tr>
<tr>
<td>xx_shicdrag</td>
<td>shelfice drag coefficient (see Section 10.3.1)</td>
</tr>
<tr>
<td>xx_depth</td>
<td>bottom topography requires to define ALLOW_DEPTH_CONTROL</td>
</tr>
<tr>
<td>3D, time-invariant controls</td>
<td></td>
</tr>
<tr>
<td>genarr3d</td>
<td></td>
</tr>
<tr>
<td>xx_theta</td>
<td>initial potential temperature</td>
</tr>
<tr>
<td>xx_salt</td>
<td>initial salinity</td>
</tr>
<tr>
<td>xx_uvel</td>
<td>initial zonal velocity</td>
</tr>
<tr>
<td>xx_vvel</td>
<td>initial meridional velocity</td>
</tr>
<tr>
<td>xx_kapgm</td>
<td>GM coefficient</td>
</tr>
<tr>
<td>xx_kapredi</td>
<td>isopycnal diffusivity</td>
</tr>
<tr>
<td>xx_diffkr</td>
<td>diapycnal diffusivity</td>
</tr>
<tr>
<td>2D, time-varying controls</td>
<td></td>
</tr>
<tr>
<td>gentim2D</td>
<td></td>
</tr>
<tr>
<td>xx_atemp</td>
<td>atmospheric temperature</td>
</tr>
<tr>
<td>xx_aqh</td>
<td>atmospheric specific humidity</td>
</tr>
<tr>
<td>xx_swdown</td>
<td>downward shortwave</td>
</tr>
<tr>
<td>xx_lwdown</td>
<td>downward longwave</td>
</tr>
<tr>
<td>xx_precip</td>
<td>precipitation</td>
</tr>
<tr>
<td>xx_runoff</td>
<td>river runoff</td>
</tr>
<tr>
<td>xx_uwind</td>
<td>zonal wind</td>
</tr>
<tr>
<td>xx_vwind</td>
<td>meridional wind</td>
</tr>
<tr>
<td>xx_tauu</td>
<td>zonal wind stress</td>
</tr>
<tr>
<td>xx_tauv</td>
<td>meridional wind stress</td>
</tr>
<tr>
<td>xx_gen_precip</td>
<td>globally averaged precipitation?</td>
</tr>
<tr>
<td>xx_hflux</td>
<td>net heat flux</td>
</tr>
<tr>
<td>xx_sflux</td>
<td>net salt (EmPmR) flux</td>
</tr>
<tr>
<td>xx_shifwflx</td>
<td>shelfice melt rate</td>
</tr>
</tbody>
</table>

Table 10.9: xx_gen????d_preproc options implemented as of checkpoint 67x. Notes: 

- If noscaling is false, the control adjustment is scaled by one on the square root of the weight before being added to the base control variable; if noscaling is true, the control is multiplied by the weight in the cost function itself.

<table>
<thead>
<tr>
<th>name</th>
<th>description</th>
<th>arguments</th>
</tr>
</thead>
<tbody>
<tr>
<td>WC01</td>
<td>Correlation modeling</td>
<td>integer: operator type (default: 1)</td>
</tr>
<tr>
<td>smooth</td>
<td>Smoothing without normalization</td>
<td>integer: operator type (default: 1)</td>
</tr>
<tr>
<td>docycle</td>
<td>Average period replication</td>
<td>integer: cycle length</td>
</tr>
<tr>
<td>replicate</td>
<td>Alias for docycle</td>
<td>(units of xx_gentim2d_period)</td>
</tr>
<tr>
<td>rmcycle</td>
<td>Periodic average subtraction</td>
<td>integer: cycle length</td>
</tr>
<tr>
<td>variaweight</td>
<td>Use time-varying weight</td>
<td>—</td>
</tr>
<tr>
<td>noscaling &quot;</td>
<td>Do not scale with xx_gen* weight</td>
<td>—</td>
</tr>
<tr>
<td>documul</td>
<td>Sets xx_gentim2d_cumsum</td>
<td>—</td>
</tr>
<tr>
<td>doglomean</td>
<td>Sets xx_gentim2d_glosum</td>
<td>—</td>
</tr>
</tbody>
</table>
Table 10.10: \texttt{xx\_gen????d\_preproc\_c} options implemented as of checkpoint 67x.

<table>
<thead>
<tr>
<th>name</th>
<th>description</th>
<th>arguments</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{log10ctrl}</td>
<td>Control adjustments to base 10 logarithm of 2D or 3D array (not available for \texttt{xx_gentim2d}).</td>
<td>See Section 10.3.2</td>
</tr>
</tbody>
</table>

The control problem is non-dimensional by default, as reflected in the omission of weights in control penalties [(\( \bar{u}_i^T \bar{u}_j \) in (10.1)]. Non-dimensional controls (\( \bar{u}_j \)) are scaled to physical units (\( \bar{v}_j \)) through multiplication by the respective uncertainty fields (\( \sigma_{\bar{u}_j} \)), as part of the generic preprocessor \( \mathcal{Q} \) in (10.4). Besides the scaling of \( \bar{u}_j \) to physical units, the preprocessor \( \mathcal{Q} \) can include, for example, spatial correlation modeling (using an implementation of Weaver and Coutier, 2001) by setting \texttt{xx\_gen\*\_preproc = 'WC01'}. Alternatively, setting \texttt{xx\_gen\*\_preproc = 'smooth'} activates the smoothing part of WC01, but omits the normalization. Additionally, bounds for the controls can be specified by setting \texttt{xx\_gen\*\_bounds}. In forward mode, adjustments to the \( i \)-th control are clipped so that they remain between \texttt{xx\_gen\*\_bounds(i,1)} and \texttt{xx\_gen\*\_bounds(i,4)}. If \texttt{xx\_gen\*\_bounds(i,1)} < \texttt{xx\_gen\*\_bounds(i+1,1)} for \( i = 1, 2, 3 \), then the bounds will “emulate a local minimum;” otherwise, the bounds have no effect in adjoint mode.

For the case of time-varying controls, the frequency is specified by \texttt{xx\_gentim2d\_period}. The generic control package interprets special values of \texttt{xx\_gentim2d\_period} in the same way as the \texttt{exf} package: a value of \(-12\) implies cycling monthly fields while a value of \(0\) means that the field is steady. Time varying weights can be provided by specifying the preprocessor \texttt{variaweight}, in which case the \texttt{xx\_gentim2d\_weight} file must contain as many records as the control parameter time series itself (approximately the run length divided by \texttt{xx\_gentim2d\_period}).

The parameter \texttt{mult\_gen\*} sets the multiplier for the corresponding cost function penalty [\( \beta_j \) in (10.1); \( \beta_j = 1 \) by default]. The preconditioner, \( \mathcal{R} \), does not directly appear in the estimation problem, but only serves to push the optimization process in a certain direction in control space; this operator is specified by \texttt{gen\*Precond} (= 1 by default).

Note that control parameters exist for each individual near surface atmospheric state variable, as well as the net heat and salt (\( \text{EmPmR} \)) fluxes. The user must be mindful of control parameter combinations that make sense according to their specific setup, e.g., with the \texttt{EXF} package.

### 10.3.1 Shelfice Control Parameters

The available iceshelf control parameters depend on the form of transfer coefficient used in the simulation.

The adjustments \texttt{xx\_shicoefft} and \texttt{xx\_shicoeffs} are available when the velocity independent form of transfer coefficients is used, by setting \texttt{#undef SHI\_ALLOW\_GAMMAFRICT} in SHELFICE\_OPTIONS.h at compile time (see Table 8.20) and SHELFICEuseGammaFrict = .FALSE. in data.shelfice (see Table 8.21). These parameters provide adjustments to \( \gamma_T \) and/or \( \gamma_S \) directly. If only one of either is used, the value of the other is set based on the control adjustments used together with SHELFICEsaltToHeatRatio, which can be set in data.shelfice. See \texttt{Run-time parameters and default values; all parameters are in namelist group SHELFICE\_PARMO1} for the default.

The adjustment \texttt{xx\_shicdrag} is available in the velocity dependent form of the ice-ocean transfer coefficients, which is specified by \texttt{#define SHI\_ALLOW\_GAMMAFRICT} and \texttt{SHELFICEuseGammaFrict} = .TRUE. at compile time and run time respectively. This parameter provides adjustments to the drag coefficient at the ice ocean boundary, but by default only adjusts the drag coefficient used to compute the thermal and freshwater fluxes, neglecting the momentum contributions. To allow the contribution directly to momentum fluxes, specify \texttt{xx\_genarr2d\_preproc\_c(*,iarr) = 'mom'} in data.ctrl.
10.3.2 Logarithmic Control Parameters

As indicated in Table 10.10, the base-10 logarithm of a control field can be adjusted by specifying the character option genarr*d_preproc_c(k2,iarr) = 'log10ctrl', with k2 and iarr as appropriate, and *d denoting that 2d or 3d are available. As a concrete example, if the control parameter is updating fld2d, then the field will be set as follows:

\[
\text{fld2d}(i,j,bi,bj) = 10^{(\text{log10InitVal} + \text{xx_genarr2d}(i,j,bi,bj,iarr))}
\]

where \text{log10InitVal} is a scalar with a default value of 0, but can be changed by setting gencost_preproc_r(k2,iarr). This is useful in the case where doInitXX=.TRUE.. Concretely, if we had an initial guess for \text{fld2d} = 10^{-4} then one could set the following in data.ctrl:

\begin{align*}
\text{xx_genarr2d}_\text{file}(1) &= \text{xx_fld2d} \\
\text{xx_genarr2d}_\text{weight}(1) &= \text{nonzero_weights.data} \\
\text{xx_genarr2d}_\text{preproc_c}(1,1) &= \text{log10ctrl} \\
\text{xx_genarr2d}_\text{preproc_r}(1,1) &= -4,
\end{align*}

Note that the log10ctrl option can only be used when a weight file is provided, and finally that this log-option cannot be used with \text{xx_gen*\_preproc}(k2,iarr) = 'noscaling'..

10.4 SMOOTH: Smoothing And Covariance Model

Author: Gael Forget

TO BE CONTINUED...

10.5 The line search optimisation algorithm

Author: Patrick Heimbach

10.5.1 General features

The line search algorithm is based on a quasi-Newton variable storage method which was implemented by [GLe-marechal89].

TO BE CONTINUED...

10.5.2 The online vs. offline version

- **Online version**
  
  Every call to simul refers to an execution of the forward and adjoint model. Several iterations of optimization may thus be performed within a single run of the main program (lsopt_top). The following cases may occur:

  - cold start only (no optimization)
  - cold start, followed by one or several iterations of optimization
  - warm start from previous cold start with one or several iterations
  - warm start from previous warm start with one or several iterations

- **Offline version**
Every call to simul refers to a read procedure which reads the result of a forward and adjoint run. Therefore, only one call to simul is allowed, itmax = 0, for cold start itmax = 1, for warm start. Also, at the end, \( x(i+1) \) needs to be computed and saved to be available for the offline model and adjoint run.

In order to achieve minimum difference between the online and offline code \( xdiff(i+1) \) is stored to file at the end of an (offline) iteration, but recomputed identically at the beginning of the next iteration.

### 10.5.3 Number of iterations vs. number of simulations

- `itmax`: controls the max. number of iterations
- `nfunc`: controls the max. number of simulations within one iteration

#### 10.5.3.1 Summary

From one iteration to the next the descent direction changes. Within one iteration more than one forward and adjoint run may be performed. The updated control used as input for these simulations uses the same descent direction, but different step sizes.

#### 10.5.3.2 Description

From one iteration to the next the descent direction \( dd \) changes using the result for the adjoint vector \( gg \) of the previous iteration. In `lsline` the updated control

\[
xdiff(i,1) = xx(i-1) + tact(i-1,1) \cdot dd(i-1)
\]

serves as input for a forward and adjoint model run yielding a new \( gg(i,1) \). In general, the new solution passes the 1st and 2nd Wolfe tests so \( xdiff(i,1) \) represents the solution sought:

\[
xx(i) = xdiff(i,1)
\]

If one of the two tests fails, an inter- or extrapolation is invoked to determine a new step size \( tact(i-1,2) \). If more than one function call is permitted, the new step size is used together with the “old” descent direction \( dd(i-1) \) (i.e. \( dd \) is not updated using the new \( gg(i) \)), to compute a new

\[
xdiff(i,2) = xx(i-1) + tact(i-1,2) \cdot dd(i-1)
\]

that serves as input in a new forward and adjoint run, yielding \( gg(i,2) \). If now, both Wolfe tests are successful, the updated solution is given by

\[
xx(i) = xdiff(i,2) = xx(i-1) + tact(i-1,2) \cdot dd(i-1)
\]

In order to save memory both the fields \( dd \) and \( xdiff \) have a double usage.

- in `lsopt_top`: used as \( x(i) - x(i-1) \) for Hessian update
- in `lsline`: intermediate result for control update \( x = x + tact*dd \)

- in `lsopt_top`, `lsline`: descent vector, \( dd = -gg \) and `hessupd`
- in `dgscale`: intermediate result to compute new preconditioner
10.5.3.3 The parameter file lsopt.par

- **NUPDATE** max. no. of update pairs \((gg(i)-gg(i-1), xx(i)-xx(i-1))\) to be stored in OPWARMD to estimate Hessian [pair of current iter. is stored in \((2*jmax+2, 2*jmax+3)\) jmax must be > 0 to access these entries] Presently NUPDATE must be > 0 (i.e. iteration without reference to previous iterations through OPWARMD has not been tested)

- **EPSX** relative precision on \(xx\) bellow which \(xx\) should not be improved

- **EPSG** relative precision on \(gg\) below which optimization is considered successful

- **IPRINT** controls verbose (>=1) or non-verbose output

- **NUMITER** max. number of iterations of optimisation; NUMTER = 0: cold start only, no optimization

- **ITER_NUM** index of new restart file to be created (not necessarily = NUMITER!)

- **NFUNC** max. no. of simulations per iteration (must be > 0); is used if step size tact is inter-/extrapolated; in this case, if NFUNC > 1, a new simulation is performed with same gradient but “improved” step size

- **FMIN** first guess cost function value (only used as long as first iteration not completed, i.e. for jmax <= 0)

10.5.3.4 OPWARMI, OPWARMD files

Two files retain values of previous iterations which are used in latest iteration to update Hessian:

- **OPWARMI**: contains index settings and scalar variables

<table>
<thead>
<tr>
<th>entry</th>
<th>name</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>n = nn</td>
<td>no. of control variables</td>
</tr>
<tr>
<td>fc</td>
<td>fc = ff</td>
<td>cost value of last iteration</td>
</tr>
<tr>
<td>isize</td>
<td>isize</td>
<td>no. of bytes per record in OPWARMD</td>
</tr>
<tr>
<td>m</td>
<td>m = nupdate</td>
<td>max. no. of updates for Hessian</td>
</tr>
<tr>
<td>jmin, jmax</td>
<td>pointer indices for OPWARMD file (cf. below)</td>
<td></td>
</tr>
<tr>
<td>gnorm0</td>
<td>gnorm0</td>
<td>norm of first (cold start) gradient gg</td>
</tr>
<tr>
<td>iabsiter</td>
<td>total number of iterations with respect to cold start</td>
<td></td>
</tr>
</tbody>
</table>

- **OPWARMD**: contains vectors (control and gradient)

<table>
<thead>
<tr>
<th>entry</th>
<th>name</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>xx(i)</td>
<td>control vector of latest iteration</td>
</tr>
<tr>
<td>2</td>
<td>gg(i)</td>
<td>gradient of latest iteration</td>
</tr>
<tr>
<td>3</td>
<td>xdiff(i),diag</td>
<td>preconditioning vector; (1,...,1) for cold start</td>
</tr>
<tr>
<td>2*jmax+2</td>
<td>gold=g(i)-g(i-1)</td>
<td>for last update (jmax)</td>
</tr>
<tr>
<td>2*jmax+3</td>
<td>xdiff=tact*d=xx(i)-xx (i-1)</td>
<td>for last update (jmax)</td>
</tr>
</tbody>
</table>

Example 1: \(jmin = 1, jmax = 3, mupd = 5\)

```
1 2 3 | 4 5 6 7 8 9 empty   
|____|___|___|___|___|___|   
0    | 1 2 3
```

Example 2: \(jmin = 3, jmax = 7, mupd = 5 \) ---> \(jmax = 2\)

```
1 2 3
```
10.5.3.5 Error handling

lsopt_top

|----- check arguments
|----- CALL INSTORE

|----- determine whether OPWARMI available:
|    * if no: cold start: create OPWARMI
|    * if yes: warm start: read from OPWARMI
|    create or open OPWARMD

|----- check consistency between OPWARMI and model parameters

|----- >>> if COLD start: <<<
|    first simulation with f.g. xx_0; output: first ff_0, gg_0
|    set first preconditioner value xdiff_0 to 1
|    store xx(0), gg(0), xdiff(0) to OPWARMD (first 3 entries)

|----- >>> else: WARM start: <<<
|    read xx(i), gg(i) from OPWARMD (first 2 entries)
|    for first warm start after cold start, i=0

|----- /// if ITMAX > 0: perform optimization (increment loop index i)

(----- save current values of gg(i-1) -> gold(i-1), ff -> fold(i-1)

(----- CALL LSUPDXX

(   |----- >>> if jmax=0 <<<

(   |----- first optimization after cold start:

(   |----- preconditioner estimated via ff_0 - ff_(first guess)

(   |----- dd(i-1) = -gg(i-1)*preco

|   |----- >>> if jmax > 0 <<<

(   |----- dd(i-1) = -gg(i-1)

(   |----- CALL HESSUPD

(   |----- dd(i-1) modified via Hessian approx.

(   |----- >>> if <dd,gg> >= 0 <<<

(   |----- compute step size: tact(i-1)

(   |----- compute update: xdiff(i) = xx(i-1) + tact(i-1)*dd(i-1)

(continues on next page)
| )---- >>> if ifail = 4 <<< ( | )---- CALL OPTLINE / LSLINE | ) | ( goto 1000 | ) | (---- CALL OPTLINE / LSLINE | ) |

... ... ... ...

... ... ...

(---- CALL OPTLINE / LSLINE
| ) |

| ( // loop over simulations
| ) |

( )---- CALL SIMUL
( )

( )---- input: xdiff(i)
( )---- output: ff(i), gg(i)
( )---- >>> if ONLINE <<<
( )

... runs model and adjoint
( )

... reads those values from file
( )

(---- 1st Wolfe test:
( )

ff(i) <= tact*xpara1*<gg(i-1),dd(i-1)>
( )

(---- 2nd Wolfe test:
( )

<gg(i),dd(i-1)> >= xpara2*<gg(i-1),dd(i-1)>
( )

(---- >>> if 1st and 2nd Wolfe tests ok <<<
( )

| 320: update xx: xx(i) = xdiff(i)
( )

| >>> else if 1st Wolfe test not ok <<<
( )

| 500: INTERpolate new tact:
( )

| barr*tact < tact < (1-barr)*tact
( )

| CALL CUBIC
( )

| >>> else if 2nd Wolfe test not ok <<<
( )

| 350: EXTRApolate new tact:
( )

| (1+barmin)*tact < tact < 10*tact
( )

| CALL CUBIC
( )

| (---- >>> if new tact > tmax <<<
( )

| ifail = 7
( )

| (---- >>> if new tact < tmin OR tact*dd < machine precision <<<
( )

| ifail = 8
( )

| (---- >>> else <<<
( )

| update xdiff for new simulation
( )

| \ if nfunc > 1: use inter-/extrapolated tact and xdiff
( )

| for new simulation

(continues on next page)
N.B.: new \( xx \) is thus not based on new \( gg \), but rather on new step size \( tact \).

(---- store new values \( xx(i) \), \( gg(i) \) to OPWARMD (first 2 entries)

(---- compute new pointers \( jmin \), \( jmax \) to include latest values \( gg(i) - gg(i-1) \), \( xx(i) - xx(i-1) \) to Hessian matrix estimate
(---- store \( gg(i) - gg(i-1) \), \( xx(i) - xx(i-1) \) to OPWARMD
( entries \( 2*jmax+2, 2*jmax+3 \)
(---- CALL DGSCALE

|---- |---- call dostore
|---- |---- read preconditioner of previous iteration \( diag(i-1) \)
|---- |---- compute new preconditioner \( diag(i) \), based upon \( diag(i-1) \),
|---- |---- call dostore
|---- |---- write new preconditioner \( diag(i) \) to OPWARMD (3rd entry)

\end\}

end of optimization iteration loop

---- CALL OUTSTORE

|---- |---- store gnorm0, \( ff(i) \), current pointers \( jmin \), \( jmax \), iterabs to OPWARMI

---- if OFFLINE version <<<

\( xx(i+1) \) needs to be computed as input for offline optimization

---- CALL LSUPDXX

|---- compute dd(i), tact(i) -> xdiff(i+1) = x(i) + tact(i)*dd(i)

---- CALL WRITE_CONTROL

|---- write xdiff(i+1) to special file for offline optim.

---- if OFFLINE version <<<

\( xx(i+1) \) needs to be computed as input for offline optimization

---- CALL LSUPDXX

|---- compute dd(i), tact(i) -> xdiff(i+1) = x(i) + tact(i)*dd(i)

---- CALL WRITE_CONTROL

|---- write xdiff(i+1) to special file for offline optim.
The non-MITgcm package `optim_m1qn3` is based on the same quasi-Newton variable storage method (BFGS) [GLemarechal89] as the package in subdirectory `lsopt`, but it uses a reverse communication version of the latest (and probably last) release of the subroutine `m1qn3`. This avoids having to define a dummy subroutine `simul` and also simplifies the code structure. As a consequence this package is simpler to compile and use, because `m1qn3.f` contains all necessary subroutines and only one extra routine (ddot, which was copied from BLAS) is required.

The principle of reverse communication is outlined in this example:

```fortran
external simul_rc
...
reverse = .true.
do while (.true.)
   call m1qn3 (simul_rc,...,x,f,g,...,reverse,indic,...)
   if (reverse) break
   call simul (indic,n,x,f,g)
end while
```

`simul_rc` is an empty “model simulator”, and `simul` generates a new state based on the value of `indic`.

The original `m1qn3` has been modified to work “offline”, i.e. the simulator and the driver of `m1qn3_offline` are separate programs that are called alternatingly from a (shell-)script. This requires that the “state” of `m1qn3` is saved before this program terminates. This state is saved in a single file `OPWARM.optXXX` per simulation, where `XXX` is the simulation number. Communication with the routine, writing and restoring the state of `m1qn3` is achieved via three new common-blocks that are contained in three header files. `simul` is replaced by reading and storing the model state and gradient vectors. Schematically the driver routine `optim_sub` does the following:

```fortran
external simul_rc
...
call optim_readdata( nn, ctrlname, ..., xx ) ! read control vector
call optim_readdata( nn, costname, ..., adxx ) ! read gradient vector
call optim_store_m1qn3( ..., .false. ) ! read state of m1qn3
reverse = .true.
call m1qn3 (simul_rc,...,xx,objf,adxx,...,reverse,indic,...)
call optim_store_m1qn3( ..., .true. ) ! write state of m1qn3
call optim_writedata( nn, ctrlname, ..., xx ) ! write control vector
```

The optimization loop is executed outside of this program within a script.

The code can be obtained at https://github.com/mjlosch/optim_m1qn3. The README contains short instructions how to build and use the code in combination with the verification/tutorial_global_oce_optim experiment. The usage is very similar to the `optim` package.
10.6 Test Cases For Estimation Package Capabilities

First, if you have not done so already, download the model as explained in *Getting Started with MITgcm* via the MITgcm git repository:

```
% git clone https://github.com/MITgcm/MITgcm.git
```

Then, download the setup from the MITgcm verification_other git repository:

```
% cd MITgcm
% git clone https://github.com/MITgcm/verification_other
```

and follow the additional directions provided for global_oce_cs32 or for global_oce_lle90. These model configurations are used for daily regression tests to ensure continued availability of the tested estimation package features discussed in *Ocean State Estimation Packages*. Daily results of these tests, which currently run on the glacier cluster, are reported on https://mitgcm.org/testing-summary. To this end, one sets a crontab job that typically executes the script reported below. The various commands can also be used to run these examples outside of crontab, directly at the command line via the testreport capability.

**Note:** Users are advised against running global_oce_lle90 tests with fewer than 12 cores (96 for adjoint tests) to avoid potential memory overloads. global_oce_lle90 (595M) uses the same LLC90 grid as the production ECCO version 4 setup [FCH+15]. The much coarser resolution global_oce_cs32 (614M) uses the CS32 grid and can run on any modern laptop.

```
% #!/bin/csh -f
% setenv PATH ~/bin:$PATH
% setenv MODULESHOME /usr/share/Modules
% source /usr/share/Modules/init/csh
% module use /usr/share/Modules
% module load openmpi-x86_64
% setenv MPI_INC_DIR $MPI_INCLUDE
%
% cd ~/MITgcm
% #mkdir gitpull.log
% set D=`date +%Y-%m-%d`
% git pull -v > gitpull.log/gitpull.$D.log
%
% cd verification
%
% #ieee case:
% ./.testreport -clean -t 'global_oce_*'
% ./.testreport -of=./tools/build_options/linux_amd64_gfortran -MPI 24 -t 'global_oce_*' -addr username@something.whatever
% ../tools/do_tst_2+2 -t 'global_oce_*' -mpi -exe 'mpirun -np 24 ./mitgcmuv' -a username@something.whatever
%
% #devel case:
% ./.testreport -clean -t 'global_oce_*'
% ./.testreport -of=./tools/build_options/linux_amd64_gfortran -MPI 24 -devel -t 'global_oce_*' -addr username@something.whatever
% ../tools/do_tst_2+2 -t 'global_oce_*' -mpi -exe 'mpirun -np 24 ./mitgcmuv' -a username@something.whatever
```

(continues on next page)
% fast case:
% ./testreport -clean -t 'global_oce_*'
% ./testreport -of=../tools/build_options/linux_amd64_gfortran -MPI 24 -t 'global_oce_*'
% -fast -addr username@something.whatever
% ../tools/do_tst_2+2 -t 'global_oce_*' -mpi -exe 'mpirun -np 24 ./mitgcmuv' -a
% username@something.whatever
%
% #adjoint case:
% ./testreport -clean -t 'global_oce_*'
% ./testreport -of=../tools/build_options/linux_amd64_gfortran -MPI 24 -ad -t 'global_oce_*' -addr username@something.whatever
11.1 MITgcmutils

This Python package includes a number of helpful functions and scripts for dealing with MITgcm output. You can install it from the model repository (in directory utils/python/MITgcmutils) or from the Python Package Index:

```
pip install --user MITgcmutils
```

The following functions are exposed at the package level:

- from module mds: `rdmds()` and `wrmds()`
- from module mnc: `rdmnc()` and `mnc_files()`
- from module ptracers: `iolabel()` and `iolabel2num()`
- from module diagnostics: `readstats()`

The package also includes a standalone script for joining tiled mnc files: `gluemncbig`.

For more functions, see the individual modules:

11.1.1 mds

```
exception MITgcmutils.mds.ParseError
```

```
MITgcmutils.mds.parsmeta(metafile)
```

parses metafile (file object or filename) into a dictionary of lists of floats, ints or strings

```
MITgcmutils.mds.rdmds(fnamearg, itrs=-1, machineformat='b', rec=None, fill_value=0, returnmeta=False,
    astyle=<class 'float'>, region=None, lev=(), useemmap=False, mm=False,
    squeeze=True, verbose=False)
```

Read meta-data files as written by MITgcm.

Call signatures:

```
a = rdmds(fname,...)
a,its,meta = rdmds(fname,...,returnmeta=True)
```

Parameters

- `fname` (string) – name of file to read, without the `.data` or `.meta` suffix. If itrs is given, the iteration number is added to `fname` as well. `fname` may contain shell wildcards, which is useful for tile files organized into directories, e.g.,

```
T = rdmds('prefix*/T', 2880)
```
will read prefix0000/T.000002880.*, prefix0001/T.000002880.*, ... (and any others that
match the wildcard, so be careful how you name things!)

- **itr**s* (int or list of ints or np.NaN or np.Inf) – Iteration number(s). With
itr=-1, will try to read

  fname.meta or fname.001.001.meta, ...

If itrs is a list of integers of an integer, it will read the corresponding

  fname.000000iter.meta, ...

If itrs is np.NaN, it will read all iterations for which files are found. If itrs is np.Inf, it will
read the highest iteration found.

- **machineformat** (int) – endianness (‘b’ or ‘l’, default ‘b’)

- **rec** (list of int or None) – list of records to read (default all) useful for pickups and
multi-field diagnostics files

- **fill_value** (float) – fill value for missing (blank) tiles (default 0)

- **astype** (data type) – data type to return (default: double precision) None: keep data
type/precision of file

- **region** (tuple of int) – (x0,x1,y0,y1) read only this region (default (0,nx,0,ny))

- **lev** (list of int or tuple of lists of int) – list of levels to read, or, for multiple
dimensions (excluding x,y), tuple(!) of lists (see examples below)

- **usememmap** (bool) – if True, use a memory map for reading data (default False) recom-
mended when using lev, or region with global files to save memory and, possibly, time

Returns

- **a** (array_like) – numpy array of the data read

- **its** (list of int) – list of iteration numbers read (only if returnmeta=True)

- **meta** (dict) – dictionary of metadata (only if returnmeta=True)

Examples

```python
>>> XC = rdmds('XC')
>>> XC = rdmds('res_*/XC')
>>> T = rdmds('T.0000002880')
>>> T = rdmds('T',2880)
>>> T2 = rdmds('T',[2880,5760])
>>> T,its = rdmds('T',numpy.Inf)
>>> VVEL = rdmds('pickup',2880,rec=range(50,100))
>>> a5 = rdmds('diags',2880,rec=0,lev=[5])
>>> a = rdmds('diags',2880,rec=0,lev=[[0],[0,1,5,6,7]])
>>> from numpy import r_
>>> a = rdmds('diags',2880,rec=0,lev=[r_[0,1,5,6,7]])  # same as previous
>>> a = rdmds('diags',2880,rec=0)[0, [0,1,5,6,7], ...]  # same, but less efficient
>>> a = rdmds('diags',2880)[0, 0, [0,1,5,6,7], ...]  # even less efficient
```

MITgcmutils.mds.readmeta(f)

read meta file and extract tile/timestep-specific parameters
MITgcmutils.mds.scanforfiles(fname)
    return list of iteration numbers for which metafiles with base fname exist

MITgcmutils.mds.strip_comments(text)
    strips C and C++ style comments from text

MITgcmutils.mds.wrmds(fbase, arr, itr=None, dataprec='float32', ndims=None, nrecords=None, times=None, fields=None, simulation=None, machineformat='b', deltat=None, dimlist=None)
    Write an array to an mds meta/data file set.
    If it is given, the files will be named fbase.0000000itr.data and fbase.0000000itr.meta, otherwise just fbase.data and fbase.meta.

    Parameters
    • fbase (string) – Name of file to write, without the ‘.data’ or ‘.meta’ suffixes, and without the iteration number if it is give
    • arr (array_like) – Numpy array to write
    • itr (int or None) – If given, this iteration number will be appended to the file name
    • dataprec (string) – precision of resulting file (‘float32’ or ‘float64’)
    • ndims (int) – number of non-record dimensions; extra (leading) dimensions will be folded into 1 record dimension
    • nrecords (int) – number of records; will fold as many leading dimensions as necessary (has to match shape!)
    • times (float or list of floats) – times to write into meta file. Either a single float or a list of two for a time interval
    • fields (list of strings) – list of fields
    • simulation (string) – string describing the simulation
    • machineformat (string) – ‘b’ or ‘l’ for big or little endian
    • deltat (float) – time step; provide in place of either times or itr to have one computed from the other
    • dimlist (tuple) – dimensions as will be stored in file (only useful when passing meta data from an existing file to wrmds as keyword args)

11.1.2 mnc

class MITgcmutils.mnc.MNC(fpatt, layout=None, multitime=False)
    A file object for MNC (tiled NetCDF) data.
    Should behave mostly like scipy.io.netcdf.netcdf_file in ‘r’ mode.

    Parameters
    • fpatt (string) – glob pattern for tile files
    • layout (string) – which global layout to use:
        'model' use layout implied by Nx, Ny
        'exch2' use exch2 global layout
        'faces' variables are lists of exch2 faces
        default is to use exch2 layout if present, model otherwise
Example

```python
>>> nc = mnc_files('mnc_*/state.0000000000.t*.nc')
>>> temp = nc.variables['Temp'][:]
>>> salt = nv.variables['S'][:]
>>> nc.close()
```

temp and salt are now assembled (global) arrays of shape \((N_t, N_r, N_y, N_x)\)
where \(N_t\) is the number iterations found in the file (in this case probably 1).

Notes

The multitime option is not implemented, i.e., MNC cannot read files split in time.

close()
Close tile files

MITgcmutils.mnc\._mnc_files(fpat\_t, layout=\_None)
A file object for MNC (tiled NetCDF) data.
Should behave mostly like scipy.io.netcdf.netcdf_file in ‘r’ mode.

Parameters

- **fpat\_t** (string) – glob pattern for tile files
- **layout** (string) – which global layout to use:
  - ‘model’ use layout implied by \(N_x, N_y\)
  - ‘exch2’ use exch2 global layout
  - ‘faces’ variables are lists of exch2 faces
  default is to use exch2 layout if present, model otherwise

Example

```python
>>> nc = mnc_files('mnc_*/state.0000000000.t*.nc')
>>> temp = nc.variables['Temp'][:]
>>> salt = nv.variables['S'][:]
>>> nc.close()
```

temp and salt are now assembled (global) arrays of shape \((N_t, N_r, N_y, N_x)\)
where \(N_t\) is the number iterations found in the file (in this case probably 1).

Notes

The multitime option is not implemented, i.e., MNC cannot read files split in time.

MITgcmutils.mnc\._rdmnc(fpat\_t, varnames=None, iters=None, slices=Ellipsis, layout=\_None)
Read one or more variables from an mnc file set.

Parameters

- **fpat\_t** (string) – glob pattern for netcdf files comprising the set
- **varnames** (list of strings, optional) – list of variables to read (default all)
- **iters** (list of int, optional) – list of iterations (not time) to read
• **slices** *(tuple of slice objects)* – tuple of slices to read from each variable (typically given as `numpy.s_[...]*

**Returns**
dictionary of variable arrays

**Return type**
dict of numpy arrays

**Example**

```python
>>> S = rdmnc("mnc_*/state.0000000000.*", ['U', 'V'], slices=numpy.s_[..., 10:-10, 10:-10])
>>> u = S['U']
>>> v = S['V']
```

**Notes**

Can currently read only one file set (i.e., 1 file per tile), not several files split in time.

Consider using `mnc_files` for more control (and similar convenience). The same restriction about multiple files applies, however.

### 11.1.3 diagnostics

`MITgcmutils.diagnostics.readstats(fname)`

```python
locals,totals,itrs = readstats(fname)
```

Read a diagstats text file into record arrays (or dictionaries).

**Parameters**

- `fname` *(string)* – name of diagstats file to read

**Returns**

- **locals** *(record array or dict of arrays)* – local statistics, shape (len(itrs), Nr, 5)
- **totals** *(record array or dict of arrays)* – column integrals, shape (len(itrs), 5)
- **itrs** *(list of int)* – iteration numbers found in the file

**Notes**

- The 5 columns of the resulting arrays are average, std.dev, min, max and total volume.
- There is a record (or dictionary key) for each field found in the file.

### 11.1.4 ptracers

`MITgcmutils.ptracers.iolabel(i)`

Map tracer number (1..3843) to 2-character I/O label:

- 1..99 => 01..99
- 100..619 => 0a..0Z, 1a..9Z
- 620..3843 => aa..ZZ

11.1. MITgcmutils
Parameters  

- \(i\) (int) – ptracer number (1..3843)

Returns  2-character I/O label

Return type  string

MITgcmutils.ptracers.iolabel2num(s)

Map 2-character IO label to tracer number, the inverse of iolabel()

### 11.1.5 jmd95

Density of Sea Water using the Jackett and McDougall 1995 (JAOT 12) polynomial

MITgcmutils.jmd95.bulkmodjmd95(s, theta, p)

Compute bulk modulus

MITgcmutils.jmd95.dens(s, theta, p)

Computes in-situ density of sea water


Parameters

- \(s\) (array_like) – salinity [psu (PSS-78)]
- \(\theta\) (array_like) – potential temperature [degree C (IPTS-68)]; same shape as \(s\)
- \(p\) (array_like) – pressure [dbar]; broadcastable to shape of \(s\)

Returns  \(dens\) – density [kg/m^3]

Return type  array

Example

```python
>>> densjmd95(35.5, 3., 3000.)
1041.83267
```

Notes

AUTHOR: Martin Losch 2002-08-09 (mlosch@mit.edu)

Jackett and McDougall, 1995, JAOT 12(4), pp. 381-388

MITgcmutils.jmd95.densjmd95(s, theta, p)

Computes in-situ density of sea water


Parameters

- \(s\) (array_like) – salinity [psu (PSS-78)]
- \(\theta\) (array_like) – potential temperature [degree C (IPTS-68)]; same shape as \(s\)
- \(p\) (array_like) – pressure [dbar]; broadcastable to shape of \(s\)

Returns  \(dens\) – density [kg/m^3]
Return type  array

Example

```python
>>> densjmd95(35.5, 3., 3000.)
1041.83267
```

Notes

AUTHOR: Martin Losch 2002-08-09 (mlosch@mit.edu)
Jackett and McDougall, 1995, JAOT 12(4), pp. 381-388

11.1.6 mdjwf

Density of Sea Water using McDougall et al. 2003 (JAOT 20) polynomial

MITgcmutils.mdjwf.dens\(s, \theta, p\)

Computes in-situ density of sea water

Density of Sea Water using McDougall et al. 2003 (JAOT 20) polynomial (Gibbs Potential).

Parameters

- \(s\) (array_like) – salinity [psu (PSS-78)]
- \(\theta\) (array_like) – potential temperature [degree C (IPTS-68)]; same shape as \(s\)
- \(p\) (array_like) – pressure [dbar]; broadcastable to shape of \(s\)

Returns \(d\) – density [kg/m^3]

Return type  array

Example

```python
>>> densmdjwf(35., 25., 2000.)
1031.654229
```

Notes

AUTHOR: Martin Losch 2002-08-09 (Martin.Losch@awi.de)
McDougall et al., 2003, JAOT 20(5), pp. 730-741

MITgcmutils.mdjwf.densmdjwf\(s, \theta, p\)

Computes in-situ density of sea water

Density of Sea Water using McDougall et al. 2003 (JAOT 20) polynomial (Gibbs Potential).

Parameters

- \(s\) (array_like) – salinity [psu (PSS-78)]
- \(\theta\) (array_like) – potential temperature [degree C (IPTS-68)]; same shape as \(s\)
- \(p\) (array_like) – pressure [dbar]; broadcastable to shape of \(s\)
Returns dens – density [kg/m^3]

Return type array

Example

```python
>>> densmdjwf(35., 25., 2000.)
1031.654229
```

Notes

AUTHOR: Martin Losch 2002-08-09 (Martin.Losch@awi.de)

McDougall et al., 2003, JAOT 20(5), pp. 730-741

11.1.7 cs

MITgcmutils.cs.pcol(x, y, data, projection=None, vmin=None, vmax=None, **kwargs)
Plots 2D scalar fields on the MITgcm cubed sphere grid with pcolormesh.

Parameters

- **x** *(array_like)* – ‘xg’, that is, x coordinate of the points one half grid cell to the left and bottom, that is vorticity points for tracers, etc.
- **y** *(array_like)* – ‘yg’, that is, y coordinate of same points
- **data** *(array_like)* – scalar field at tracer points
- **projection** *(Basemap instance, optional)* – used to transform if present. Unfortunately, cylindrical and conic maps are limited to the [-180 180] range. projection = ‘sphere’ results in a 3D visualization on the sphere without any specific projection. Good for debugging.

Example

```python
>>> from mpl_toolkits.basemap import Basemap
>>> import MITgcmutils as mit
>>> import matplotlib.pyplot as plt
>>> from sq import
>>> x=mit.rdmds('XG'); y=mit.rdmds('YG'); e=mit.rdmds('Eta',np.Inf)
>>> fig = plt.figure();
>>> mp = Basemap(projection='moll',lon_0 = 0.,
               resolution = 'l', area_thresh = 1000.)
>>> plt.clf()
>>> h = mit.cs.pcol(x,y,sq(e), projection = mp)
>>> mp.fillcontinents(color = 'grey')
>>> mp.drawmapboundary()
>>> mp.drawmeridians(np.arange(0, 360, 30))
>>> mp.drawparallels(np.arange(-90, 90, 30))
>>> plt.show()
```
11.1.8 llc

MITgcmutils.llc.contour(*arguments, **kwargs)
Create a contour plot of a 2-D llc array (with tricontour).

Call signatures:

\[
\begin{align*}
\text{contour}(X, Y, C, N, **\text{kwargs}) \\
\text{contour}(X, Y, C, V, **\text{kwargs})
\end{align*}
\]

Parameters

- \(X\) (array-like) – x coordinates of the grid points
- \(Y\) (array-like) – y coordinates of the grid points
- \(C\) (array-like) – array of color values.
- \(N\) (int) – number of levels
- \(V\) (list of float) – list of levels
- \(\text{kwargs}\) – passed to tricontour.

MITgcmutils.llc.contourf(*arguments, **kwargs)
Create a contourf plot of a 2-D llc array (with tricontour).

Call signatures:

\[
\begin{align*}
\text{contourf}(X, Y, C, N, **\text{kwargs}) \\
\text{contourf}(X, Y, C, V, **\text{kwargs})
\end{align*}
\]

Parameters

- \(X\) (array-like) – x coordinates of the grid points
- \(Y\) (array-like) – y coordinates of the grid points
- \(C\) (array-like) – array of color values.
- \(N\) (int) – number of levels
- \(V\) (list of float) – list of levels
- \(\text{kwargs}\) – passed to tricontour.

MITgcmutils.llc.div(u, v, dxg=None, dyg=None, rac=None, hfw=None, hfs=None)
Compute divergence of vector field (U,V) on llc grid

Call signatures:

\[
\begin{align*}
\text{divergence} = \text{div}(U, V, DXG, DYG, RAC, HFW, HFS) \\
\text{divergence} = \text{div}(U, V) \\
\text{divergence} = \text{div}(U, V, DXG, DYG) \\
\text{divergence} = \text{div}(U, V, DXG, DYG, RAC) \\
\text{divergence} = \text{div}(U, V, DXG, DYG, hfw=HFW, hfs=HFS)
\end{align*}
\]

Parameters
• \textbf{u} (array-like (timelevel, depthlevel, jpoint, ipoint)) – x-component of vector field at u-point
• \textbf{v} (array-like (timelevel, depthlevel, jpoint, ipoint)) – y-component of vector field at v-point
• \textbf{dxg} (array-like (jpoint, ipoint), optional) – grid spacing in x across v-point, defaults to one
• \textbf{dyg} (array-like (jpoint, ipoint), optional) – grid spacing in y across u-point, defaults to one
• \textbf{rac} (array-like (jpoint, ipoint), optional) – grid cell area, defaults to \textbf{dxg} \times \textbf{dyg}
• \textbf{hfw} (array-like (depthlevel, jpoint, ipoint), optional) – hFac at u-point, defaults to one
• \textbf{hfs} (array-like (depthlevel, jpoint, ipoint), optional) – hFac at v-point, defaults to one

\texttt{MITgcmutils.llc.faces(fld)}
convert mds multidimensional data into a list with 6 faces

\texttt{MITgcmutils.llc.faces2mds(ff)}
convert 6 faces to mds 2D data, inverse operation of llc.faces

\texttt{MITgcmutils.llc.flat(fld, **kwargs)}
convert mds data into global 2D field only fields with 2 to 5 dimensions are allowed

\texttt{MITgcmutils.llc.grad(X, dxc=None, dyc=None, hfw=None, hfs=None)}
Compute horizontal gradient of scalar field \textit{X} on llc grid
Call signatures:

\begin{verbatim}
dXdx, dXdy = div(X, DXC, DYC, HFW, HFS)
dXdx, dXdy = div(X)
dXdx, dXdy = div(X, DXC, DYC)
dXdx, dXdy = div(X, hfw=HFW, hfs=HFS)
\end{verbatim}

**Parameters**

• \textit{X} (array-like (timelevel, depthlevel, jpoint, ipoint)) – scalar field at c-point
• \textbf{dxc} (array-like (jpoint, ipoint), optional) – grid spacing in x across u-point, defaults to one
• \textbf{dyc} (array-like (jpoint, ipoint), optional) – grid spacing in y across v-point, defaults to one
• \textbf{hfw} (array-like (depthlevel, jpoint, ipoint), optional) – hFac at u-point, defaults to one
• \textbf{hfs} (array-like (depthlevel, jpoint, ipoint), optional) – hFac at v-point, defaults to one

\texttt{MITgcmutils.llc.mds(fld, center='Atlantic')} convert global ‘flat’ field into mds data; only fields with 2 to 5 dimensions are allowed

\texttt{MITgcmutils.llc.pcol(*arguments, **kwargs)}
Create a pseudo-color plot of a 2-D llc array (with plt.pcolormesh).
Call signatures:
pcol(X, Y, C, **kwargs)

pcol(X, Y, C, m, **kwargs)

Parameters

- **X (array-like)** – x coordinates of the grid point corners (G-points)
- **Y (array-like)** – y coordinates of the grid point corners (G-points)
- **C (array-like)** – array of color values.
- **m (Basemap instance, optional)** – map projection to use. NOTE: currently not all projections work.
- **kwargs** – passed to plt.pcolormesh.

MITgcmutils.llc.uv2c(u, v)

Average vector component (u,v) to center points on llc grid

Call signatures:

uc, vc = uv2c(U, V)

Parameters

- **U (array-like (timelevel, depthlevel, jpoint, ipoint))** – x-component of vector field at u-point
- **V (array-like (timelevel, depthlevel, jpoint, ipoint))** – y-component of vector field at v-point

11.1.9 gluemncbig

This command line script is part of MITgcmutils and provides a convenient method for stitching together NetCDF files into a single file covering the model domain. Be careful though - the resulting files can get very large.


- **-v <vars>** comma-separated list of variable names or glob patterns
- **-2** write a NetCDF version 2 (64-Bit Offset) file allowing for large records
- **--many** many tiles: assemble only along x in memory; less efficient on some filesystems, but opens fewer files simultaneously and uses less memory
- **-q** suppress progress messages
- **--verbose** report variables
- **--help** show this help text

All files must have the same variables.
Each variable (or 1 record of it) must fit in memory.
With --many, only a row of tiles along x must fit in memory.

Examples:
gluemncbig -o ptr.nc mnc_*/ptr_tave.*.nc

gluemncbig -o BIO.nc -v 'BIO_*' mnc_*/ptr_tave.*.nc
12.1 Projects Related to MITgcm

12.1.1 Estimating the Circulation and Climate of the Ocean (ECCO)

ECCO is a community of MITgcm users who create and analyze ocean state estimates. ECCO typically optimizes initial conditions, surface forcing fields, and internal parameters to fit a multi-decadal model solution to various data constraints using MITgcm’s adjoint capabilities. Unlike other data assimilation products, ECCO solutions are dynamically self-consistent, have closed budgets, and can easily be re-run by users.


12.1.2 Gcmfaces: Gridded Earth Variables In Matlab And Octave

The gcmfaces toolbox handles gridded Earth variables as sets of connected arrays. This object-oriented approach allows users to write generic, compact analysis codes that readily become applicable to a wide variety of grids. gcmfaces notably allows for analysis of MITgcm output on any of its familiar grids.


12.1.3 MITprof: In-Situ Ocean Data In Matlab And Octave

The MITprof toolbox handles unevenly distributed in-situ ocean observations. It is notably used, along with gcmfaces, to generate input files for MITgcm’s profiles package (MITgcm/pkg/profiles).

website: https://github.com/gaelforget/MITprof

12.1.4 OceanParcels - Lagrangian Particle Tracker

Parcels provides a set of Python classes and methods to create customizable particle tracking simulations, focusing on tracking of both passive water parcels as well as active plankton, plastic and fish.

website: http://oceanparcels.org/
12.1.5 Southern Ocean State Estimation (SOSE)

SOSE uses the same techniques as ECCO to produce an eddy-permitting state estimate of the Southern Ocean.

website: http://sose.ucsd.edu/

12.1.6 Xgcm: General Circulation Model Postprocessing with xarray

Xgcm is a python package for working with the datasets produced by numerical General Circulation Models (GCMs) and similar gridded datasets that are amenable to finite volume analysis. In these datasets, different variables are located at different positions with respect to a volume or area element (e.g. cell center, cell face, etc.) xgcm solves the problem of how to interpolate and difference these variables from one position to another.

website: http://xgcm.readthedocs.io/en/latest/

12.1.7 Xmitgcm

Xmitgcm is a Python module that loads MITgcm MDS output files as xarray datasets with the associated grid information. These can be easily exported as NetCDF files.


12.2 Highlighted Papers


m

MITgcmutils.cs, 634
MITgcmutils.diagnostics, 631
MITgcmutils.jmd95, 632
MITgcmutils.llc, 635
MITgcmutils.mdjwf, 633
MITgcmutils.mds, 627
MITgcmutils.mnc, 629
MITgcmutils.ptracers, 631