
MITgcm Documentation

Release checkpoint66o-98-g1c4ae72

**Alistair Adcroft, Jean-Michel Campin, Ed Doddridge,
Stephanie Dutkiewicz, Constantinos Evangelinos,
David Ferreira, Mick Follows, Gael Forget,
Baylor Fox-Kemper, Patrick Heimbach, Chris Hill, Ed Hill,
Helen Hill, Oliver Jahn, Jody Klymak, Martin Losch,
John Marshall, Guillaume Maze, Matt Mazloff,
Dimitris Menemenlis, Andrea Molod, and Jeff Scott**

Mar 07, 2018

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	5
1.2.2	Ocean gyres	5
1.2.3	Global ocean circulation	7
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	9
1.2.7	Global state estimation of the ocean	11
1.2.8	Ocean biogeochemical cycles	14
1.2.9	Simulations of laboratory experiments	14
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	23
1.3.6	Finding the pressure field	23
1.3.7	Forcing/dissipation	26
1.3.8	Vector invariant form	27
1.3.9	Adjoint	27
1.4	Appendix ATMOSPHERE	27
1.4.1	Hydrostatic Primitive Equations for the Atmosphere in Pressure Coordinates	27
1.5	Appendix OCEAN	30
1.5.1	Equations of Motion for the Ocean	30
1.6	Appendix OPERATORS	34
1.6.1	Coordinate systems	34
2	Discretization and Algorithm	37
2.1	Notation	37
2.2	Time-stepping	38
2.3	Pressure method with rigid-lid	39
2.4	Pressure method with implicit linear free-surface	41
2.5	Explicit time-stepping: Adams-Bashforth	42
2.6	Implicit time-stepping: backward method	42

2.7	Synchronous time-stepping: variables co-located in time	44
2.8	Staggered baroclinic time-stepping	47
2.9	Non-hydrostatic formulation	50
2.10	Variants on the Free Surface	52
2.10.1	Crank-Nicolson barotropic time stepping	53
2.10.2	Non-linear free-surface	54
2.11	Spatial discretization of the dynamical equations	59
2.11.1	The finite volume method: finite volumes versus finite difference	59
2.11.2	C grid staggering of variables	60
2.11.3	Grid initialization and data	60
2.11.4	Horizontal grid	60
2.11.5	Vertical grid	63
2.11.6	Topography: partially filled cells	64
2.12	Continuity and horizontal pressure gradient term	65
2.13	Hydrostatic balance	65
2.14	Flux-form momentum equations	66
2.14.1	Advection of momentum	67
2.14.2	Coriolis terms	67
2.14.3	Curvature metric terms	68
2.14.4	Non-hydrostatic metric terms	69
2.14.5	Lateral dissipation	69
2.14.6	Vertical dissipation	71
2.14.7	Derivation of discrete energy conservation	72
2.14.8	Mom Diagnostics	72
2.15	Vector invariant momentum equations	74
2.15.1	Relative vorticity	75
2.15.2	Kinetic energy	75
2.15.3	Coriolis terms	76
2.15.4	Shear terms	76
2.15.5	Gradient of Bernoulli function	77
2.15.6	Horizontal divergence	77
2.15.7	Horizontal dissipation	77
2.15.8	Vertical dissipation	78
2.16	Tracer equations	78
2.16.1	Time-stepping of tracers: ABII	78
2.17	Linear advection schemes	80
2.17.1	Centered second order advection-diffusion	80
2.17.2	Third order upwind bias advection	81
2.17.3	Centered fourth order advection	82
2.17.4	First order upwind advection	82
2.18	Non-linear advection schemes	83
2.18.1	Second order flux limiters	83
2.18.2	Third order direct space time	84
2.18.3	Third order direct space time with flux limiting	85
2.18.4	Multi-dimensional advection	86
2.19	Comparison of advection schemes	86
2.20	Shapiro Filter	94
2.20.1	SHAP Diagnostics	94
2.21	Nonlinear Viscosities for Large Eddy Simulation	95
2.21.1	Eddy Viscosity	95
2.21.2	Mercator, Nondimensional Equations	100
3	Getting Started with MITgcm	103
3.1	Where to find information	103

3.2	Obtaining the code	103
3.2.1	Method 1	104
3.2.2	Method 2	104
3.3	Updating the code	104
3.4	Model and directory structure	105
3.5	Building the code	106
3.5.1	Building/compiling the code elsewhere	107
3.5.2	Using <code>genmake2</code>	107
3.5.3	Building with MPI	109
3.6	Running the model	110
3.6.1	Running with MPI	110
3.6.2	Output files	111
3.6.3	Looking at the output	112
3.7	Customizing the model configuration	113
3.7.1	Parameters: Computational Domain, Geometry and Time-Discretization	113
3.7.2	Parameters: Equation of State	114
3.7.3	Parameters: Momentum Equations	115
3.7.4	Parameters: Tracer Equations	116
3.7.5	Parameters: Simulation Controls	117
3.7.6	Parameters: Default Values	118
4	MITgcm Tutorial Example Experiments	123
4.1	Barotropic Gyre MITgcm Example	123
4.1.1	Equations Solved	124
4.1.2	Discrete Numerical Configuration	124
4.1.3	Code Configuration	125
4.2	A Rotating Tank in Cylindrical Coordinates	130
4.2.1	Equations Solved	131
4.2.2	Discrete Numerical Configuration	131
4.2.3	Code Configuration	131
5	Contributing to the MITgcm	137
5.1	Bugs and feature requests	137
5.2	Using Git and Github	137
5.2.1	Quickstart Guide	138
5.2.2	Detailed guide for those less familiar with Git and GitHub	138
5.3	Coding style guide	144
5.3.1	Automatic testing with Travis-CI	144
5.4	Contributing to the manual	144
5.4.1	Section headings	144
5.4.2	Internal document references	144
5.4.3	Other embedded links	145
5.4.4	Symbolic Notation	145
5.4.5	Figures	145
5.4.6	Tables	146
5.4.7	Other text blocks	147
5.4.8	Other style conventions	148
5.4.9	Building the manual	148
5.5	Reviewing pull requests	148
6	Software Architecture	151
6.1	Overall architectural goals	151
6.2	WRAPPER	152
6.2.1	Target hardware	152

6.2.2	Supporting hardware neutrality	154
6.2.3	WRAPPER machine model	154
6.2.4	Machine model parallelism	154
6.2.5	Communication mechanisms	156
6.2.6	Communication primitives	157
6.2.7	Memory architecture	159
6.2.8	Summary	159
6.3	Using the WRAPPER	160
6.3.1	Specifying a domain decomposition	161
6.3.2	Starting the code	165
6.3.3	Controlling communication	168
6.4	MITgcm execution under WRAPPER	172
6.4.1	Annotated call tree for MITgcm and WRAPPER	172
6.4.2	Measuring and Characterizing Performance	179
6.4.3	Estimating Resource Requirements	179
7	Automatic Differentiation	181
8	Packages I - Physical Parameterizations	183
8.1	Overview	183
8.1.1	Using MITgcm Packages	183
8.2	Packages Related to Hydrodynamical Kernel	188
8.2.1	Generic Advection/Diffusion	188
8.2.2	Shapiro Filter	189
8.2.3	FFT Filtering Code	190
8.2.4	exch2: Extended Cubed Sphere Topology	190
8.2.5	Gridalt - Alternate Grid Package	197
8.3	General purpose numerical infrastructure packages	201
8.3.1	OBCS: Open boundary conditions for regional modeling	201
8.3.2	RBCS Package	208
8.3.3	PTRACERS Package	210
8.4	Ocean Packages	213
8.4.1	GMREDI: Gent-McWilliams/Redi SGS Eddy Parameterization	213
8.4.2	KPP: Nonlocal K-Profile Parameterization for Vertical Mixing	220
8.4.3	GGL90: a TKE vertical mixing scheme	226
8.4.4	OPPS: Ocean Penetrative Plume Scheme	226
8.4.5	KL10: Vertical Mixing Due to Breaking Internal Waves	226
8.4.6	BULK_FORCE: Bulk Formula Package	229
8.4.7	EXF: The external forcing package	233
8.4.8	CAL: The calendar package	241
8.5	Atmosphere Packages	245
8.5.1	Atmospheric Intermediate Physics: AIM	245
8.5.2	Land package	247
8.5.3	Fizhi: High-end Atmospheric Physics	248
8.6	Sea Ice Packages	286
8.6.1	THSICE: The Thermodynamic Sea Ice Package	286
8.6.2	SEAICE Package	291
9	Packages II - Diagnostics and I/O	309
10	Ocean State Estimation Packages	311
10.1	ECCO: model-data comparisons using gridded data sets	311
10.1.1	Generic Cost Function	312
10.1.2	Generic Integral Function	315
10.1.3	Custom Cost Functions	315

10.1.4	Key Routines	316
10.1.5	Compile Options	316
10.2	PROFILES: model-data comparisons at observed locations	316
10.3	CTRL: Model Parameter Adjustment Capability	318
10.4	SMOOTH: Smoothing And Covariance Model	320
10.5	The line search optimisation algorithm	320
10.5.1	General features	320
10.5.2	The online vs. offline version	320
10.5.3	Number of iterations vs. number of simulations	321
10.6	Test Cases For Estimation Package Capabilities	325
11	Under Development	327
12	Previous Applications of MITgcm	329
	Bibliography	331

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](#)
- it has a non-hydrostatic capability and so can be used to study both small-scale and large scale processes - see [Figure 1.2](#)
- 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.

Key publications reporting on and charting the development of the model are Hill and Marshall (1995), Marshall et al. (1997a), Marshall et al. (1997b), Adcroft and Marshall (1997), Marshall et al. (1998), Adcroft and Marshall (1999), Hill et al. (1999), Marotzke et al. (1999), Adcroft and Campin (2004), Adcroft et al. (2004b), Marshall et al. (2004) (an overview on the model formulation can also be found in Adcroft et al. (2004c)):

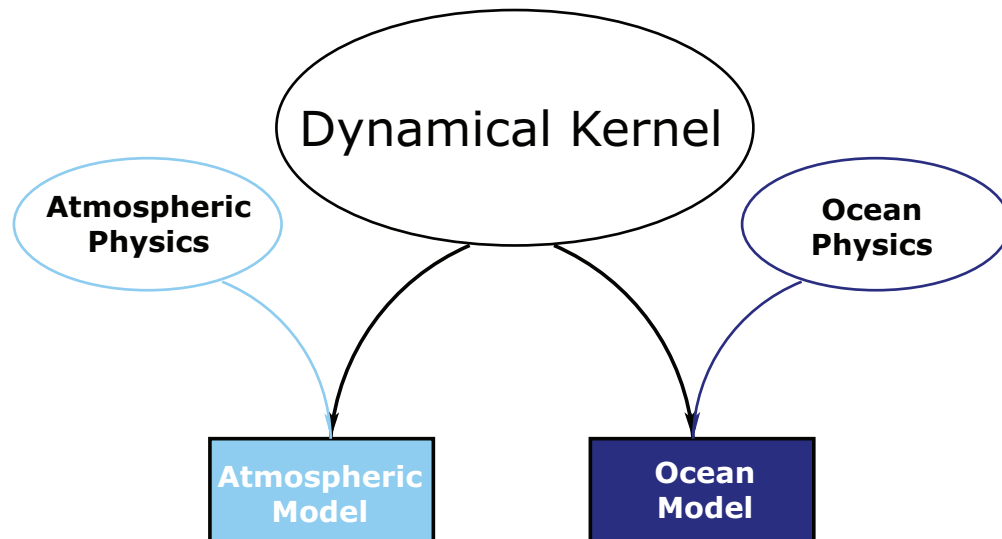


Figure 1.1: MITgcm has a single dynamical kernel that can drive forward either oceanic or atmospheric simulations.

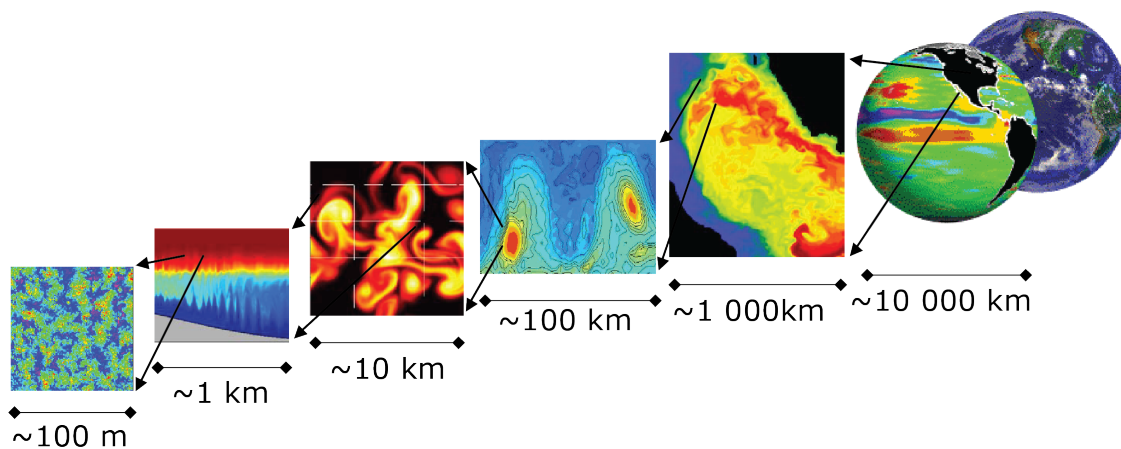


Figure 1.2: MITgcm has non-hydrostatic capabilities, allowing the model to address a wide range of phenomenon - from convection on the left, all the way through to global circulation patterns on the right.

Finite Volume: Shaved Cells

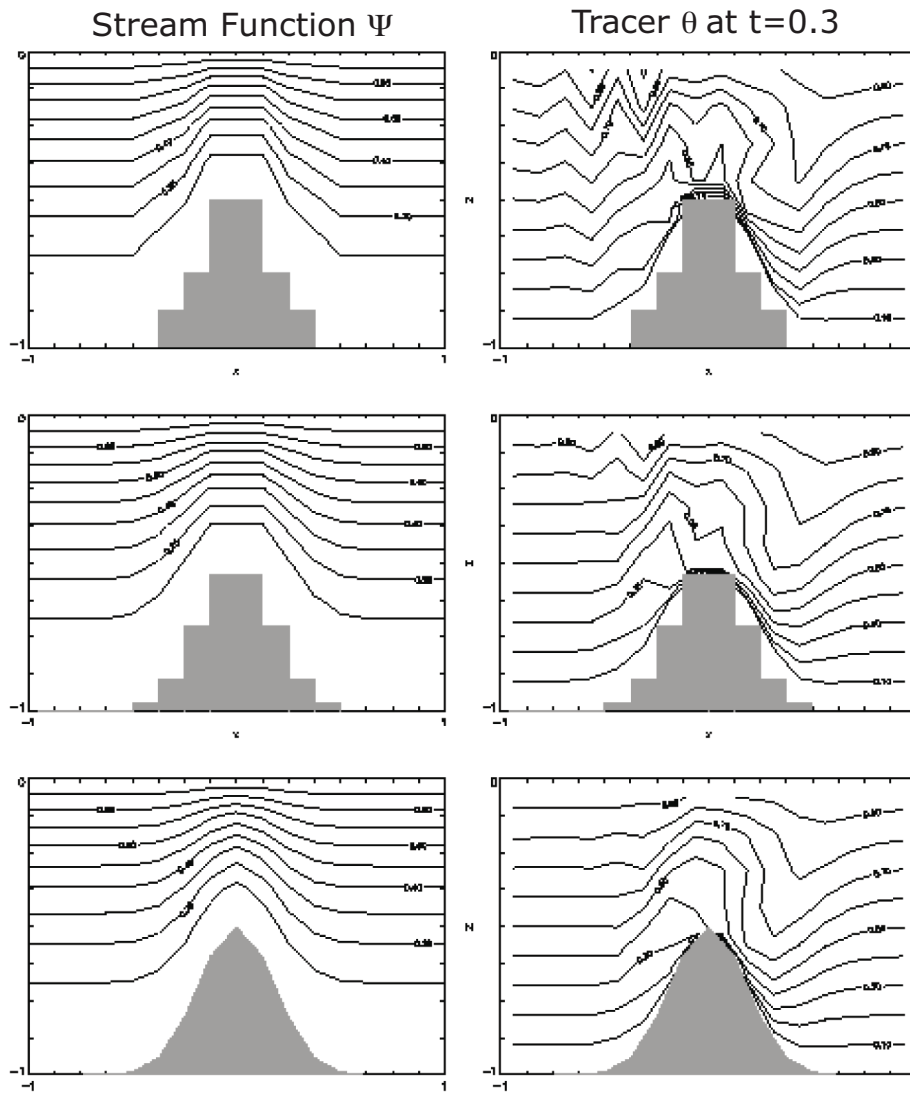


Figure 1.3: Finite volume techniques (bottom panel) are used, permitting a treatment of topography that rivals σ (terrain following) coordinates.

Hill, C. and J. Marshall, (1995) Application of a Parallel Navier-Stokes Model to Ocean Circulation in Parallel Computational Fluid Dynamics, In Proceedings of Parallel Computational Fluid Dynamics: Implementations and Results Using Parallel Computers, 545-552. Elsevier Science B.V.: New York [HM95]

Marshall, J., C. Hill, L. Perelman, and A. Adcroft, (1997a) Hydrostatic, quasi-hydrostatic, and nonhydrostatic ocean modeling, J. Geophysical Res., **102(C3)**, 5733-5752 [MHPA97]

Marshall, J., A. Adcroft, C. Hill, L. Perelman, and C. Heisey, (1997b) A finite-volume, incompressible Navier Stokes model for studies of the ocean on parallel computers, J. Geophysical Res., **102(C3)**, 5753-5766 [MAH+97]

Adcroft, A.J., Hill, C.N. and J. Marshall, (1997) Representation of topography by shaved cells in a height coordinate ocean model, Mon Wea Rev, **125**, 2293-2315 [AHM97]

Marshall, J., Jones, H. and C. Hill, (1998) Efficient ocean modeling using non-hydrostatic algorithms, Journal of Marine Systems, **18**, 115-134 [MJH98]

Adcroft, A., Hill C. and J. Marshall: (1999) A new treatment of the Coriolis terms in C-grid models at both high and low resolutions, Mon. Wea. Rev., **127**, 1928-1936 [AHM99]

Hill, C, Adcroft,A., Jamous,D., and J. Marshall, (1999) A Strategy for Terascale Climate Modeling, In Proceedings of the Eighth ECMWF Workshop on the Use of Parallel Processors in Meteorology, 406-425 World Scientific Publishing Co: UK [HAJM99]

Marotzke, J, Giering,R., Zhang, K.Q., Stammer,D., Hill,C., and T.Lee, (1999) Construction of the adjoint MIT ocean general circulation model and application to Atlantic heat transport variability, J. Geophysical Res., **104(C12)**, 29,529-29,547 [MGZ+99]

A. Adcroft and J.-M. Campin, (2004a) Re-scaled height coordinates for accurate representation of free-surface flows in ocean circulation models, Ocean Modelling, **7**, 269–284 [AC04]

A. Adcroft, J.-M. Campin, C. Hill, and J. Marshall, (2004b) Implementation of an atmosphere-ocean general circulation model on the expanded spherical cube, Mon Wea Rev , **132**, 2845–2863 [ACHM04]

J. Marshall, A. Adcroft, J.-M. Campin, C. Hill, and A. White, (2004) Atmosphere-ocean modeling exploiting fluid isomorphisms, Mon. Wea. Rev., **132**, 2882–2894 [MAC+04]

A. Adcroft, C. Hill, J.-M. Campin, J. Marshall, and P. Heimbach, (2004c) Overview of the formulation and numerics of the MITgcm, In Proceedings of the ECMWF seminar series on Numerical Methods, Recent developments in numerical methods for atmosphere and ocean modelling, 139–149. URL: <http://mitgcm.org/pdfs/ECMWF2004-Adcroft.pdf> [AHCampin+04]

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.

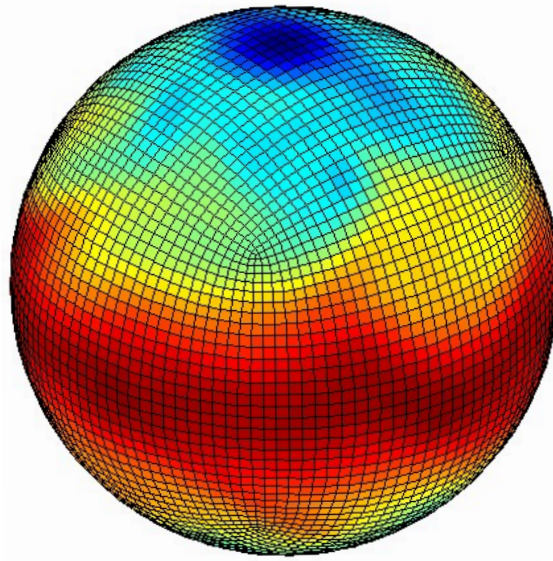


Figure 1.4: Instantaneous plot of the temperature field at 500 mb obtained using the atmospheric isomorph of MITgcm

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 favorably 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.

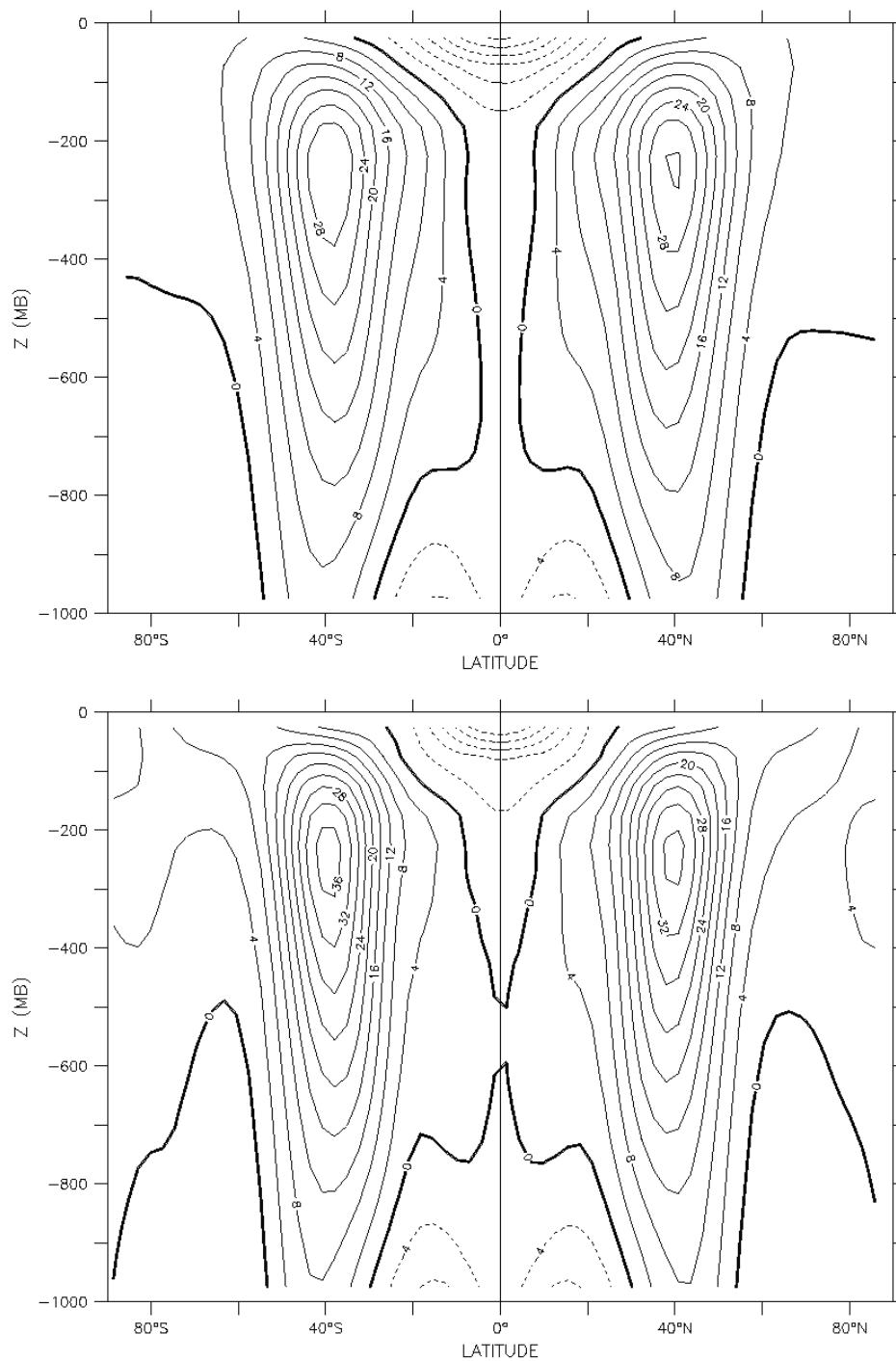


Figure 1.5: Five year mean, zonally averaged zonal flow for cube-sphere simulation (top) and latitude-longitude simulation (bottom) and using Held-Suarez forcing. Note the difference in the solutions over the pole — the cubed sphere is superior.

Figure 1.6 shows the surface temperature and velocity field obtained from MITgcm run at $\frac{1}{6}^\circ$ 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.

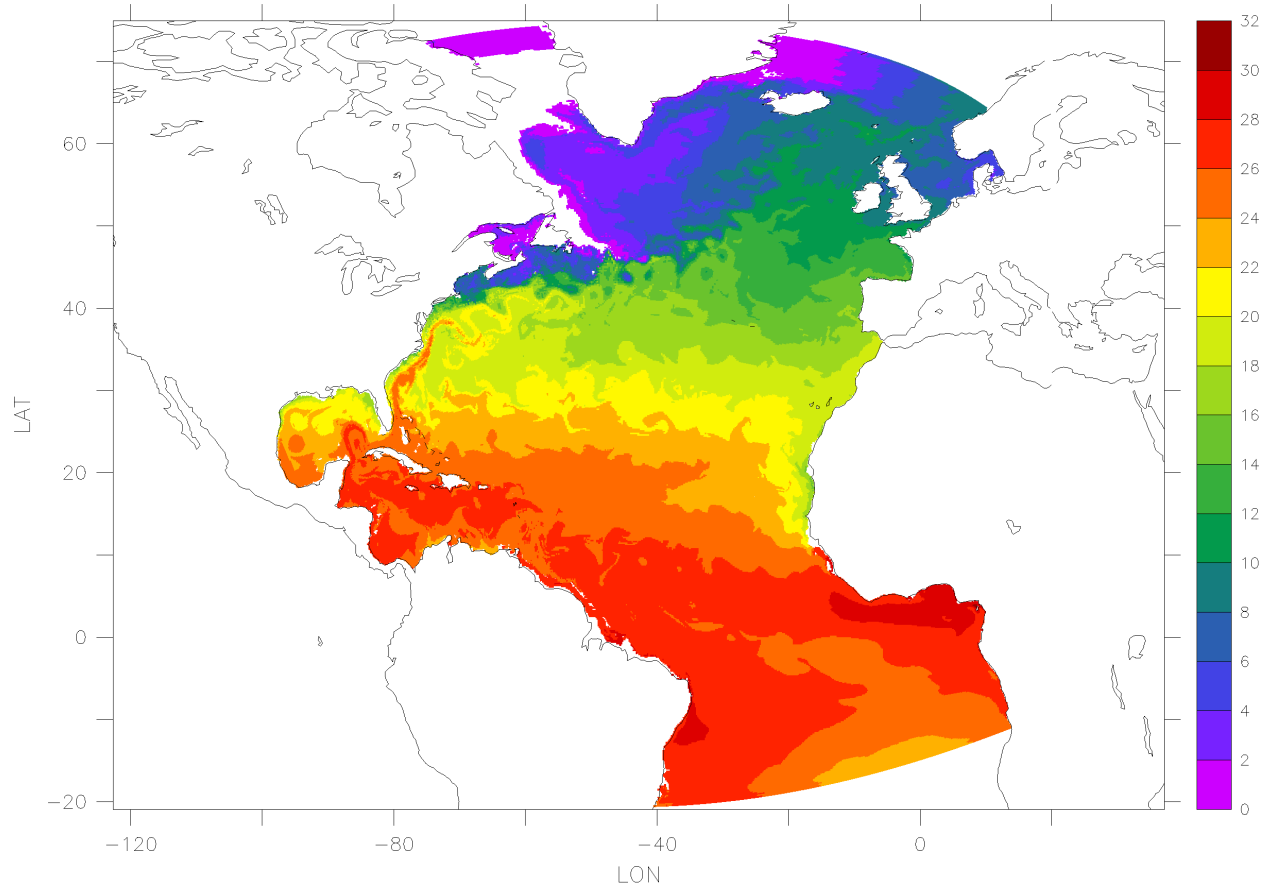


Figure 1.6: Instantaneous temperature map from a $\frac{1}{6}^\circ$ simulation of the North Atlantic. The figure shows the temperature in the second layer (37.5 m deep).

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.

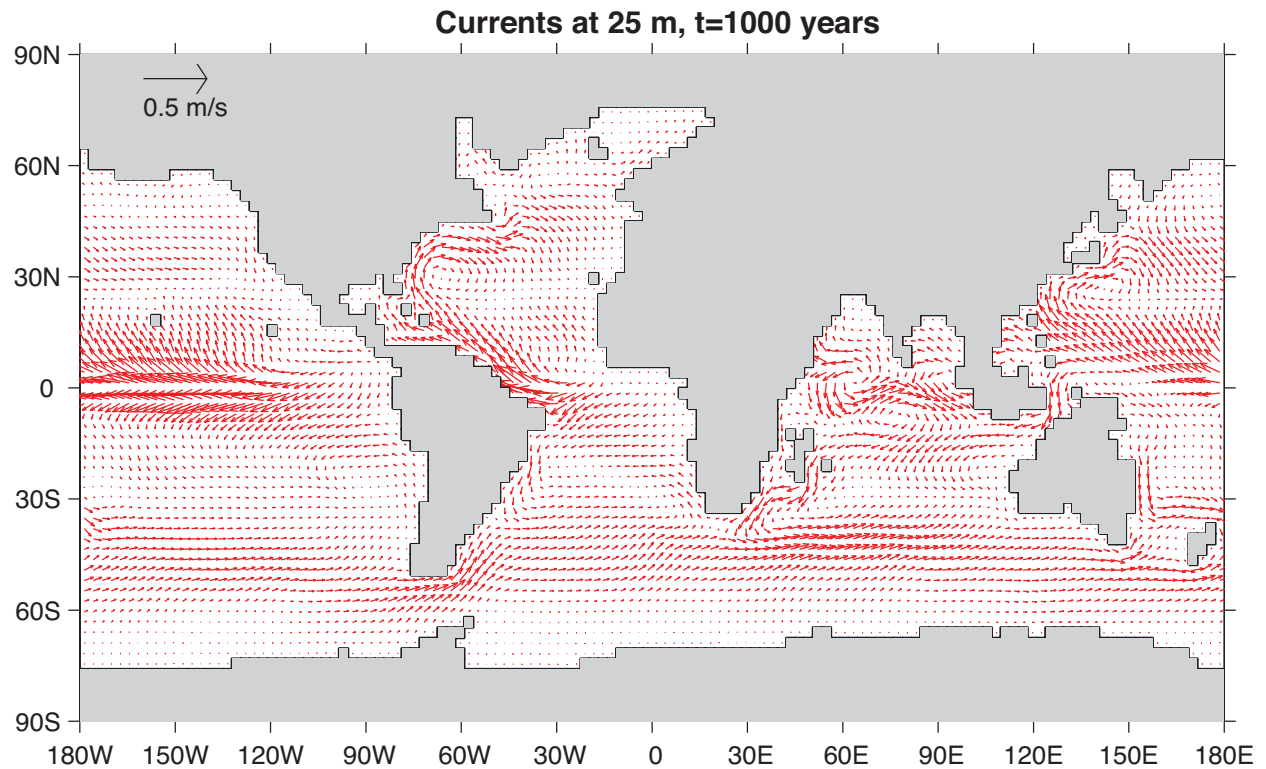


Figure 1.7: Pattern of surface ocean currents from a global integration of the model at 4° horizontal resolution and with 15 vertical levels.

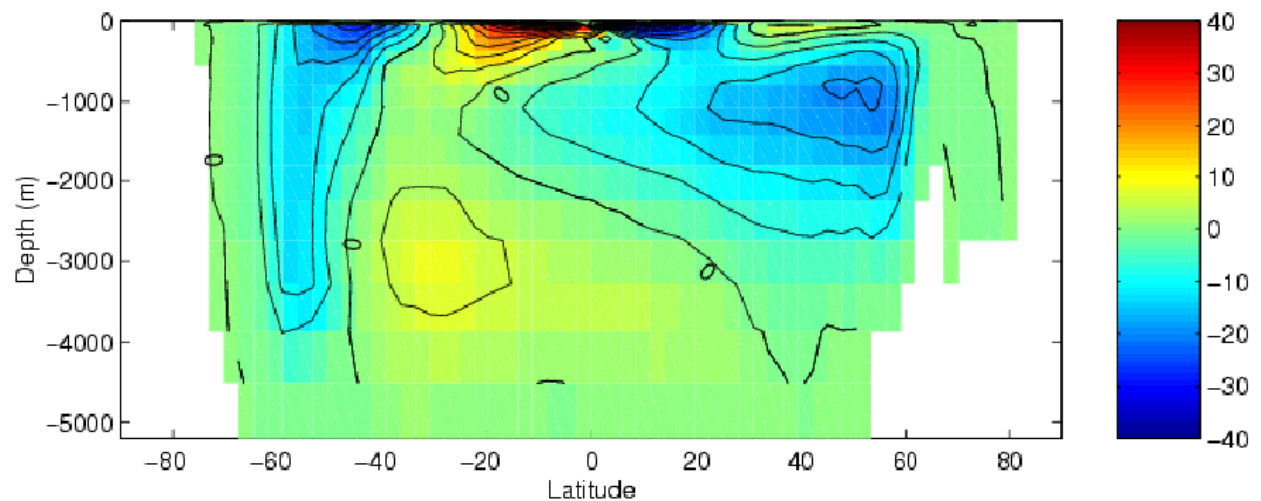


Figure 1.8: Meridional overturning stream function (in Sverdrups) from a global integration of the model at 4° horizontal resolution and with 15 vertical levels.

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.

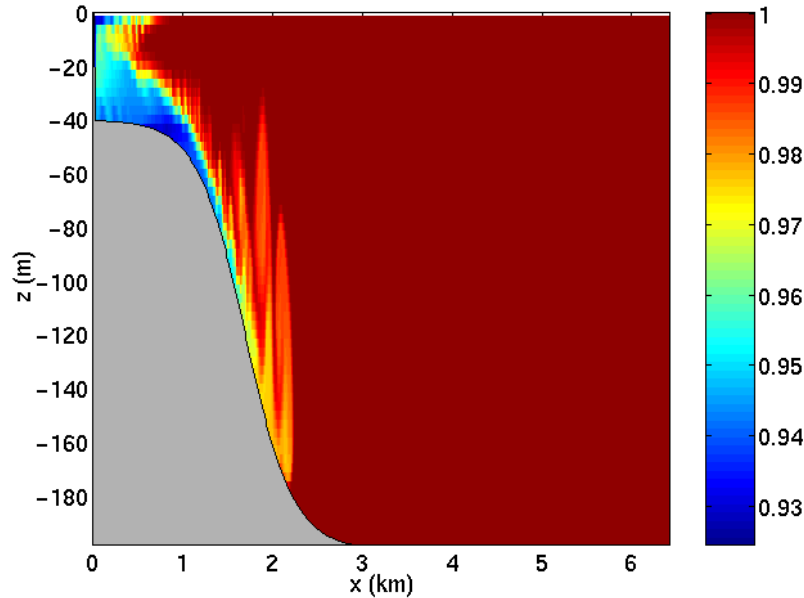


Figure 1.9: MITgcm run in a non-hydrostatic configuration to study convection over a slope.

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.

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.

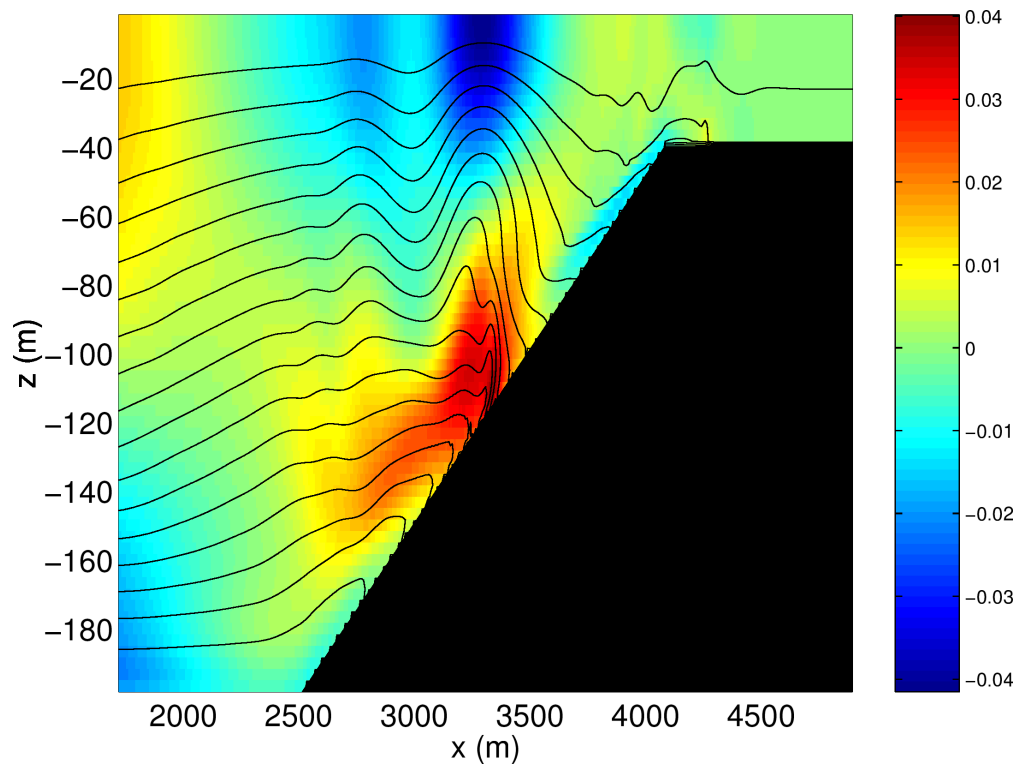


Figure 1.10: Simulation of internal waves forced at an open boundary (on the left) impacting a sloping shelf. The along slope velocity is shown colored, contour lines show density surfaces. The slope is represented with high-fidelity using lopped cells.

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.

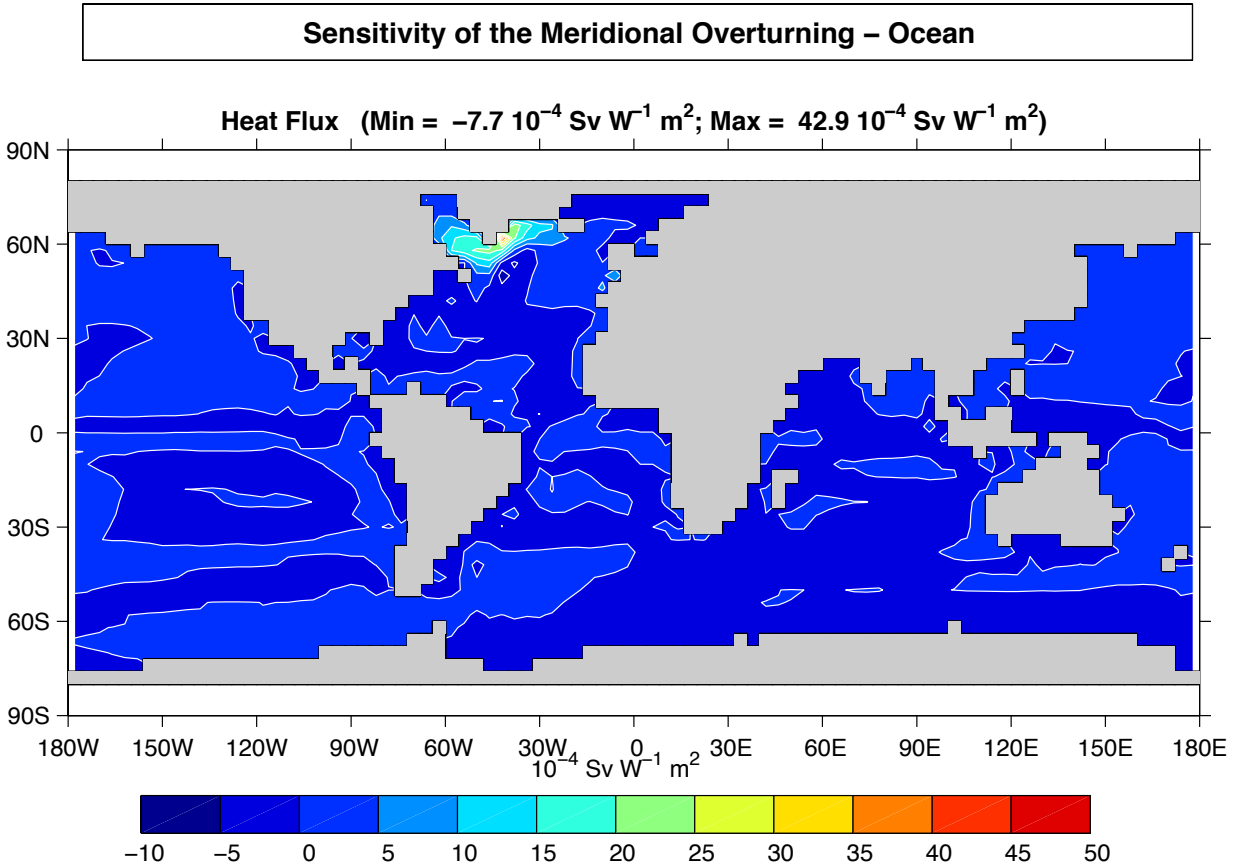


Figure 1.11: Sensitivity of meridional overturning strength to surface heat flux changes. Contours show the magnitude of the response (in $\text{Sv} \times 10^{-4}$) that a persistent $+1 \text{ W m}^{-2}$ heat flux anomaly at a given grid point would produce.

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.

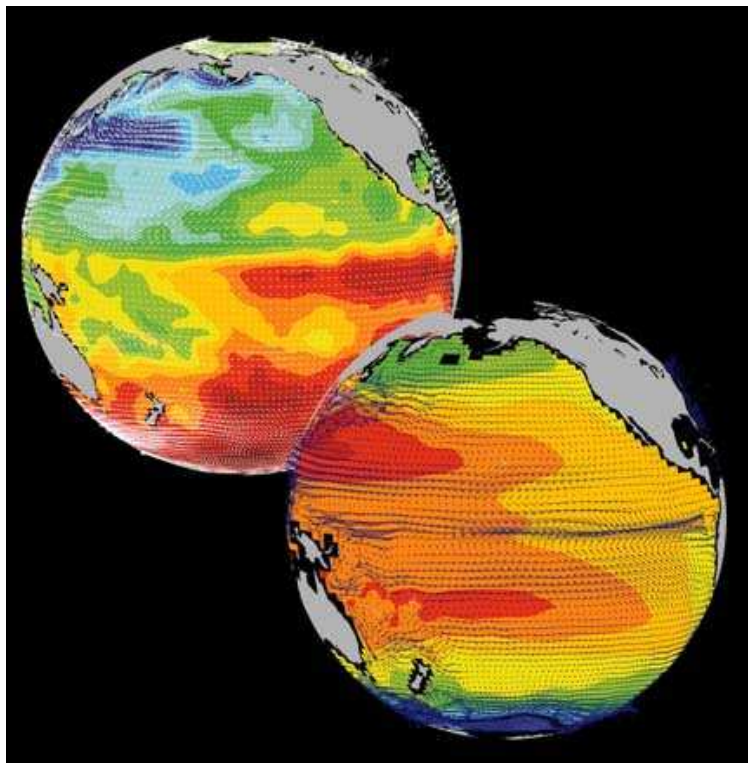


Figure 1.12: Circulation patterns from a multi-year, global circulation simulation constrained by Topex altimeter data and WOCE cruise observations. This output is from a higher resolution, shorter duration experiment with equatorially enhanced grid spacing.

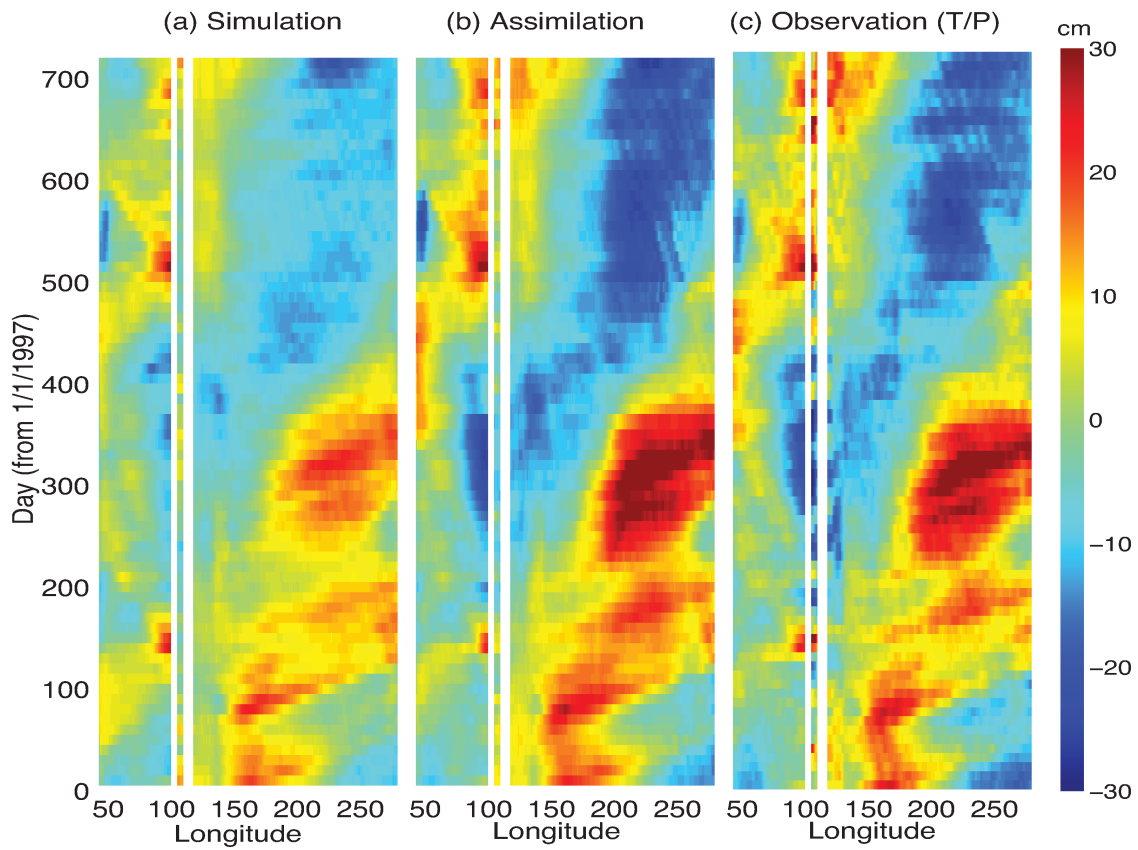


Figure 1.13: Equatorial sea-surface height in unconstrained (left), constrained (middle) simulations and in observations (right).

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).

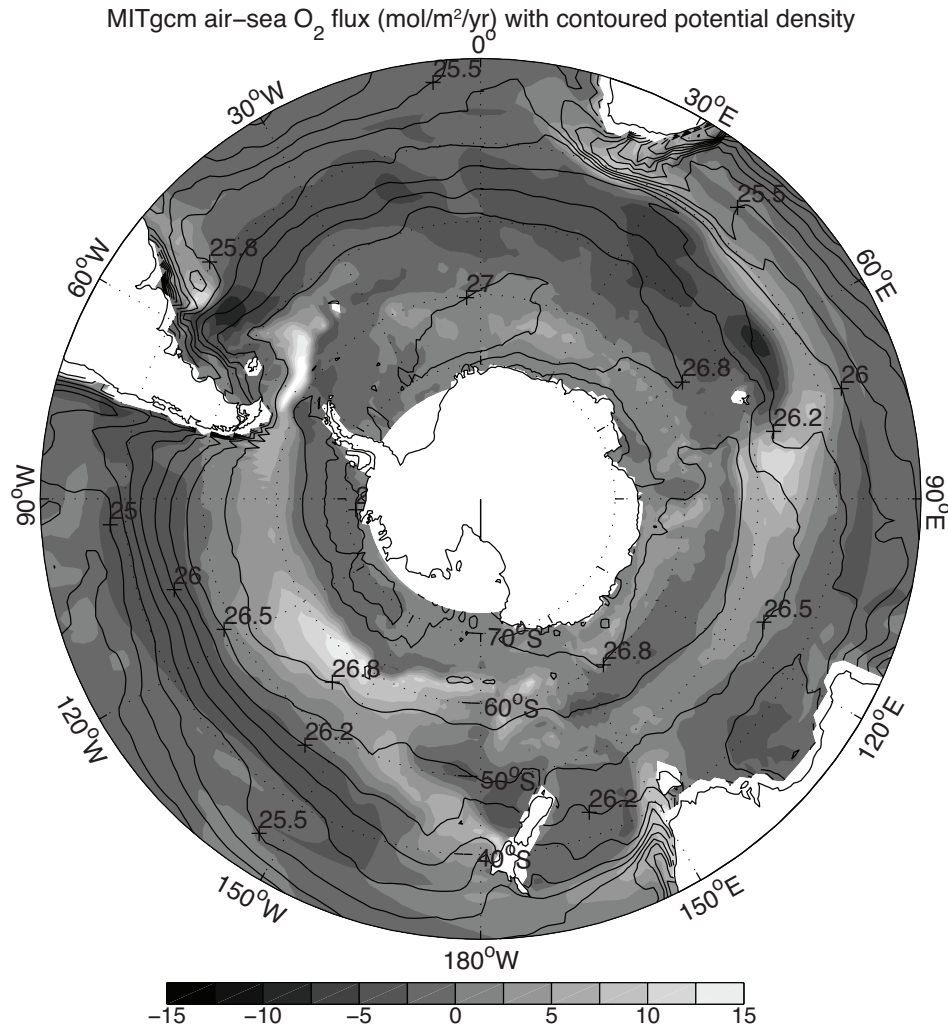


Figure 1.14: Annual air-sea flux of oxygen (shaded) plotted along with potential density outcrops of the surface of the southern ocean from a global $1^\circ \times 1^\circ$ integration with a telescoping grid (to $\frac{1}{3}^\circ$) at the equator.

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.

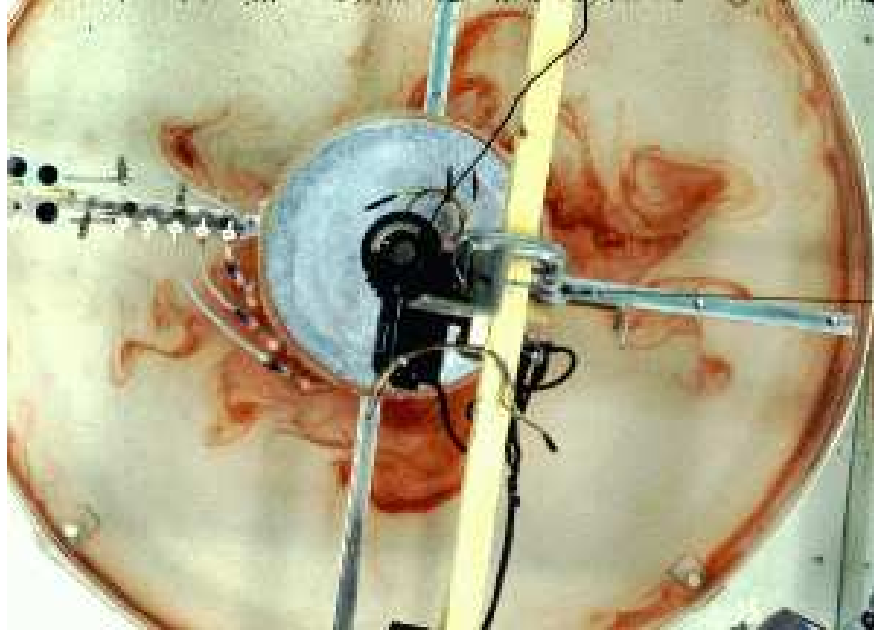


Figure 1.15: A 1 m diameter laboratory experiment simulating the dynamics of the Antarctic Circumpolar Current.

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 θ and S , a ‘geopotential’ ϕ and density $\rho = \rho(\theta, S, p)$ which may depend on θ , 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.

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

$$\frac{D\dot{r}}{Dt} + \hat{k} \cdot \left(2\vec{\Omega} \times \vec{v}\right) + \frac{\partial \phi}{\partial r} + b = \mathcal{F}_{\dot{r}} \text{ vertical momentum} \quad (1.2)$$

$$\nabla_h \cdot \vec{v}_h + \frac{\partial \dot{r}}{\partial r} = 0 \text{ continuity} \quad (1.3)$$

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

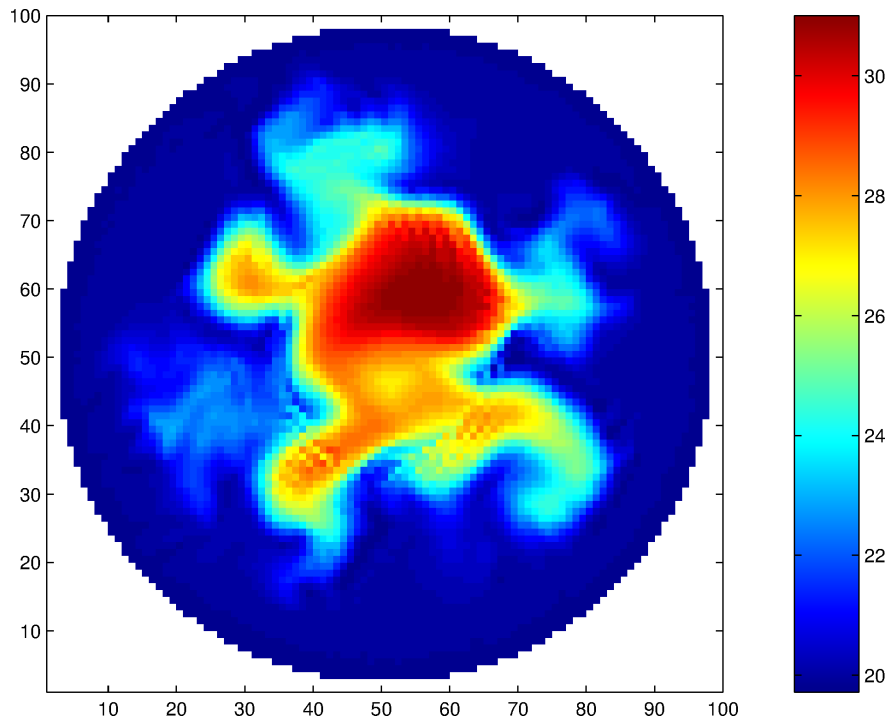


Figure 1.16: A numerical simulation of the laboratory experiment using MITgcm.

z-p Isomorphism

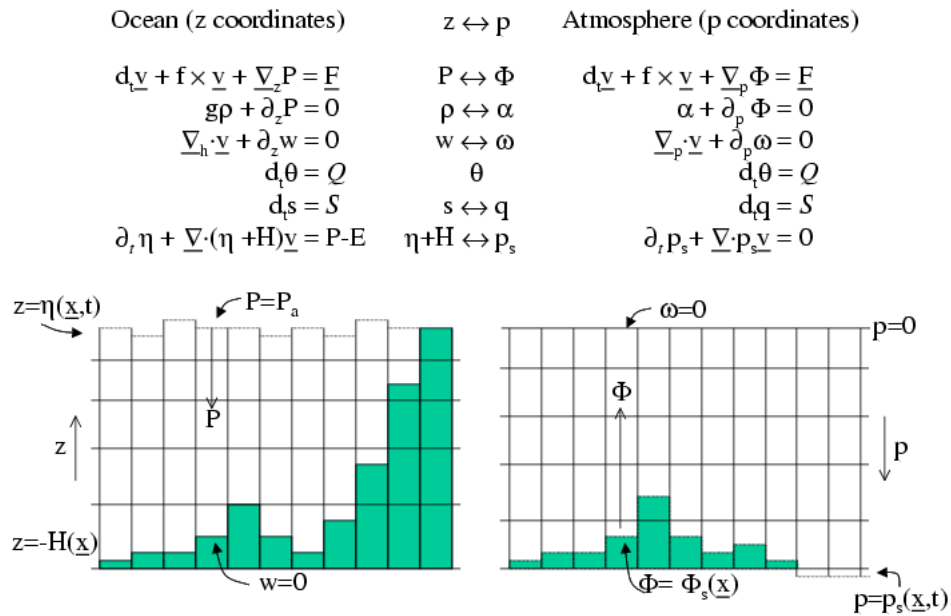


Figure 1.17: Isomorphic equation sets used for atmosphere (right) and ocean (left).

z-p Isomorphism

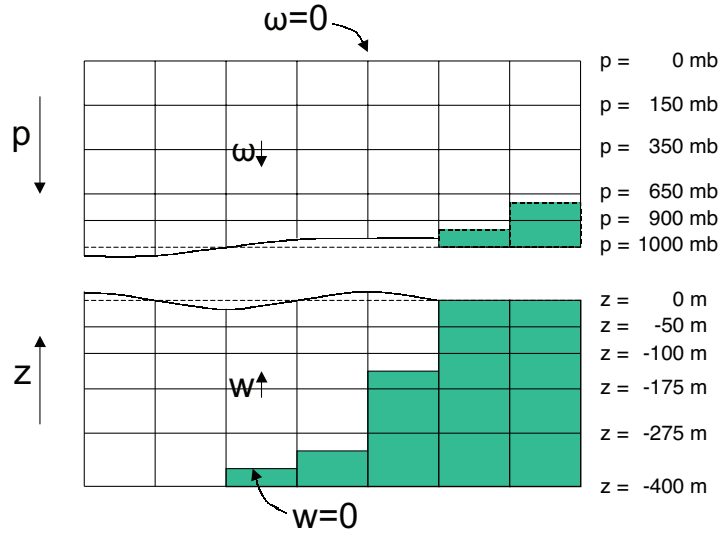


Figure 1.18: Vertical coordinates and kinematic boundary conditions for atmosphere (top) and ocean (bottom).

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

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

Here:

r is the vertical coordinate

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

$$\nabla = \nabla_h + \hat{k} \frac{\partial}{\partial r} \text{ is the 'grad' operator}$$

with ∇_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

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

ϕ is the 'pressure'/'geopotential'

$\vec{\Omega}$ is the Earth's rotation

b is the 'buoyancy'

θ is potential temperature

S is specific humidity in the atmosphere; salinity in the ocean

$\mathcal{F}_{\vec{v}}$ are forcing and dissipation of \vec{v}

\mathcal{Q}_{θ} are forcing and dissipation of θ

\mathcal{Q}_S are forcing and dissipation of S

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](#)):

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

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

Here

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

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 η 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:

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

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

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

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

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

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

where

T is absolute temperature

p is the pressure

z is the height of the pressure surface

g is the acceleration due to gravity

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)

$$\Pi(p) = c_p \left(\frac{p}{p_c}\right)^\kappa \quad (1.16)$$

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):

$$R_{fixed} = p_{top} = 0$$

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

$$R_{moving} = R_o(x, y) = p_o(x, y)$$

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:

$$\omega = 0 \text{ at } r = R_{fixed} \text{ (top of the atmosphere)} \quad (1.17)$$

$$\omega = \frac{Dp_s}{Dt} \text{ at } r = R_{moving} \text{ (bottom of the atmosphere)} \quad (1.18)$$

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:

$$r = z \text{ is the height} \quad (1.19)$$

$$\dot{r} = \frac{Dz}{Dt} = w \text{ is the vertical velocity} \quad (1.20)$$

$$\phi = \frac{p}{\rho_c} \text{ is the pressure} \quad (1.21)$$

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

where ρ_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: $R_{fixed}(x, y) = -H(x, y)$.

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

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

Boundary conditions are:

$$w = 0 \text{ at } r = R_{fixed} \text{ (ocean bottom)} \quad (1.23)$$

$$w = \frac{D\eta}{Dt} \text{ at } r = R_{moving} = \eta \text{ (ocean surface)} \quad (1.24)$$

where η 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 ϕ in to surface, hydrostatic and non-hydrostatic terms:

$$\phi(x, y, r) = \phi_s(x, y) + \phi_{hyd}(x, y, r) + \phi_{nh}(x, y, r) \quad (1.25)$$

and write (1.1) in the form:

$$\frac{\partial \vec{v}_h}{\partial t} + \nabla_h \phi_s + \nabla_h \phi_{hyd} + \epsilon_{nh} \nabla_h \phi_{nh} = \vec{G}_{\vec{v}_h} \quad (1.26)$$

$$\frac{\partial \phi_{hyd}}{\partial r} = -b \quad (1.27)$$

$$\epsilon_{nh} \frac{\partial \dot{r}}{\partial t} + \frac{\partial \phi_{nh}}{\partial r} = G_{\dot{r}} \quad (1.28)$$

Here ϵ_{nh} is a non-hydrostatic parameter.

The $(\vec{G}_{\vec{v}}, G_{\dot{r}})$ in (1.26) and (1.28) represent advective, metric and Coriolis terms in the momentum equations. In spherical coordinates they take the form¹ - see Marshall et al. (1997a) [MHPA97] for a full discussion:

$$\begin{aligned}
 G_u = & -\vec{v} \cdot \nabla u && \text{advection} \\
 & - \left\{ \frac{u\dot{r}}{r} - \frac{uv \tan \varphi}{r} \right\} && \text{metric} \\
 & - \{ -2\Omega v \sin \varphi + \underline{2\Omega \dot{r} \cos \varphi} \} && \text{Coriolis} \\
 & + \mathcal{F}_u && \text{forcing/dissipation}
 \end{aligned} \tag{1.29}$$

$$\begin{aligned}
 G_v = & -\vec{v} \cdot \nabla v && \text{advection} \\
 & - \left\{ \frac{v\dot{r}}{r} - \frac{u^2 \tan \varphi}{r} \right\} && \text{metric} \\
 & - \{ -2\Omega u \sin \varphi \} && \text{Coriolis} \\
 & + \mathcal{F}_v && \text{forcing/dissipation}
 \end{aligned} \tag{1.30}$$

$$\begin{aligned}
 G_{\dot{r}} = & -\underline{\vec{v} \cdot \nabla \dot{r}} && \text{advection} \\
 & - \left\{ \frac{u^2 + v^2}{r} \right\} && \text{metric} \\
 & + \underline{2\Omega u \cos \varphi} && \text{Coriolis} \\
 & + \underline{\mathcal{F}_{\dot{r}}} && \text{forcing/dissipation}
 \end{aligned} \tag{1.31}$$

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

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

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) \rightarrow (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 ϕ_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:

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.

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:

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

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 \hat{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; \hat{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**.

1.3.6 Finding the pressure field

Unlike the prognostic variables u , v , w , θ 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:

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

and so

$$\phi_{hyd}(x, y, r) = \int_r^{R_o} b dr \quad (1.33)$$

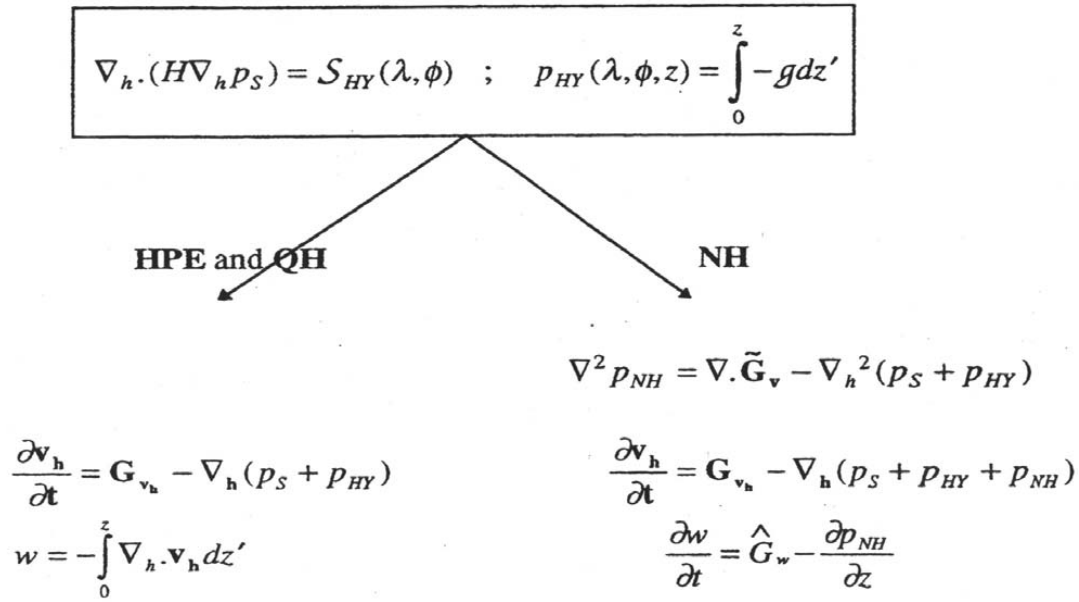


Figure 1.19: Basic solution strategy in MITgcm. **HPE** and **QH** forms diagnose the vertical velocity, in **NH** a prognostic equation for the vertical velocity is integrated.

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

$$\phi_{hyd}(r = R_o) = loading \quad (1.34)$$

1.3.6.2 Surface pressure

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

$$\int_{R_{fixed}}^{R_{moving}} (\nabla_h \cdot \vec{\mathbf{v}}_h + \partial_r \dot{r}) dr = 0$$

Thus:

$$\frac{\partial \eta}{\partial t} + \vec{\mathbf{v}} \cdot \nabla \eta + \int_{R_{fixed}}^{R_{moving}} \nabla_h \cdot \vec{\mathbf{v}}_h dr = 0$$

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

$$\frac{\partial \eta}{\partial t} + \nabla_h \cdot \int_{R_{fixed}}^{R_{moving}} \vec{\mathbf{v}}_h dr = \text{source} \quad (1.35)$$

where we have incorporated a source term.

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

$$\nabla_h \phi_s = \nabla_h (b_s \eta) \quad (1.36)$$

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 ϕ_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:

$$\nabla_3^2 \phi_{nh} = \nabla \cdot \vec{G}_{\vec{v}} - (\nabla_h^2 \phi_s + \nabla^2 \phi_{hyd}) = \nabla \cdot \vec{F} \quad (1.37)$$

For a given rhs this 3-d elliptic equation must be inverted for ϕ_{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:

$$\vec{v} \cdot \hat{n} = 0 \quad (1.38)$$

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 $r = R_{moving}$. 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:

$$\hat{n} \cdot \nabla \phi_{nh} = \hat{n} \cdot \vec{F} \quad (1.39)$$

where

$$\vec{F} = \vec{G}_{\vec{v}} - (\nabla_h \phi_s + \nabla \phi_{hyd})$$

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 δ -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 $\hat{n} \cdot \vec{F} = 0$ and $\hat{n} \cdot \nabla \phi_{nh} = 0$ on the boundary the following self-consistent but simpler homogenized Elliptic problem is obtained:

$$\nabla^2 \phi_{nh} = \nabla \cdot \tilde{\vec{F}}$$

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

$$\hat{n} \cdot \nabla \phi_{nh} = 0 \quad (1.40)$$

If the flow is ‘close’ to hydrostatic balance then the 3-d inversion converges rapidly because ϕ_{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 ϕ_{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:

$$D_V = A_h \nabla_h^2 v + A_v \frac{\partial^2 v}{\partial z^2} + A_4 \nabla_h^4 v \quad (1.41)$$

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.

$$D_{T,S} = \nabla \cdot [\underline{\underline{K}} \nabla (T, S)] + K_4 \nabla_h^4 (T, S) \quad (1.42)$$

where $\underline{\underline{K}}$ 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, $\underline{\underline{K}}$, reduces to a diagonal matrix with constant coefficients:

$$K = \begin{pmatrix} K_h & 0 & 0 \\ 0 & K_h & 0 \\ 0 & 0 & K_v \end{pmatrix} \quad (1.43)$$

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:

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

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:

$$\frac{D\vec{v}_h}{Dt} + f\hat{\mathbf{k}} \times \vec{v}_h + \nabla_p \phi = \vec{F} \quad (1.45)$$

$$\frac{\partial \phi}{\partial p} + \alpha = 0 \quad (1.46)$$

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

$$p\alpha = RT \quad (1.48)$$

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

where $\vec{v}_h = (u, v, 0)$ is the ‘horizontal’ (on pressure surfaces) component of velocity, $\frac{D}{Dt} = \frac{\partial}{\partial t} + \vec{v}_h \cdot \nabla_p + \omega \frac{\partial}{\partial p}$ is the total derivative, $f = 2\Omega \sin \varphi$ is the Coriolis parameter, $\phi = gz$ is the geopotential, $\alpha = 1/\rho$ is the specific volume, $\omega = \frac{Dp}{Dt}$ 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 θ 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:

$$c_p \frac{DT}{Dt} - \alpha \frac{Dp}{Dt} = \mathcal{Q} \quad (1.50)$$

Potential temperature is defined:

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

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:

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

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

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

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

The heat equation is obtained by noting that

$$c_p \frac{DT}{Dt} = \frac{D(\Pi \theta)}{Dt} = \Pi \frac{D\theta}{Dt} + \theta \frac{D\Pi}{Dt} = \Pi \frac{D\theta}{Dt} + \alpha \frac{Dp}{Dt}$$

and on substituting into (1.50) gives:

$$\Pi \frac{D\theta}{Dt} = \mathcal{Q} \quad (1.53)$$

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:

$$\text{at the top: } p = 0, \omega = \frac{Dp}{Dt} = 0 \quad (1.54)$$

$$\text{at the surface: } p = p_s, \phi = \phi_{topo} = g Z_{topo} \quad (1.55)$$

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 (ϕ 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:

$$\theta = \theta_o + \theta' \quad (1.56)$$

$$\alpha = \alpha_o + \alpha' \quad (1.57)$$

$$\phi = \phi_o + \phi' \quad (1.58)$$

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:

$$\frac{D\vec{v}_h}{Dt} + f\hat{\mathbf{k}} \times \vec{v}_h + \nabla_p \phi' = \vec{F} \quad (1.59)$$

$$\frac{\partial \phi'}{\partial p} + \alpha' = 0 \quad (1.60)$$

$$\nabla_p \cdot \vec{v}_h + \frac{\partial \omega}{\partial p} = 0 \quad (1.61)$$

$$\frac{\partial \Pi}{\partial p} \theta' = \alpha' \quad (1.62)$$

$$\frac{D\theta}{Dt} = \frac{Q}{\Pi} \quad (1.63)$$

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:

$$\frac{D\vec{v}_h}{Dt} + f\hat{\mathbf{k}} \times \vec{v}_h + \frac{1}{\rho}\nabla_z p = \vec{F} \quad (1.64)$$

$$\epsilon_{nh}\frac{Dw}{Dt} + g + \frac{1}{\rho}\frac{\partial p}{\partial z} = \epsilon_{nh}\mathcal{F}_w \quad (1.65)$$

$$\frac{1}{\rho}\frac{D\rho}{Dt} + \nabla_z \cdot \vec{v}_h + \frac{\partial w}{\partial z} = 0 \quad (1.66)$$

$$\rho = \rho(\theta, S, p) \quad (1.67)$$

$$\frac{D\theta}{Dt} = \mathcal{Q}_\theta \quad (1.68)$$

$$\frac{DS}{Dt} = \mathcal{Q}_s \quad (1.69)$$

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 ρ 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 ρ . It is therefore necessary to manipulate the system as follows. Differentiating the EOS (equation of state) gives:

$$\frac{D\rho}{Dt} = \left.\frac{\partial\rho}{\partial\theta}\right|_{S,p}\frac{D\theta}{Dt} + \left.\frac{\partial\rho}{\partial S}\right|_{\theta,p}\frac{DS}{Dt} + \left.\frac{\partial\rho}{\partial p}\right|_{\theta,S}\frac{Dp}{Dt} \quad (1.70)$$

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:

$$\frac{1}{\rho c_s^2}\frac{Dp}{Dt} + \nabla_z \cdot \vec{v} + \partial_z w \approx 0 \quad (1.71)$$

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:

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

$$\epsilon_{nh}\frac{Dw}{Dt} + g + \frac{1}{\rho}\frac{\partial p}{\partial z} = \epsilon_{nh}\mathcal{F}_w \quad (1.73)$$

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

$$\rho = \rho(\theta, S, p) \quad (1.75)$$

$$\frac{D\theta}{Dt} = \mathcal{Q}_\theta \quad (1.76)$$

$$\frac{DS}{Dt} = \mathcal{Q}_s \quad (1.77)$$

1.5.1.1 Compressible z-coordinate equations

Here we linearize the acoustic modes by replacing ρ 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 ρ is the gravitational acceleration:

$$\frac{D\vec{v}_h}{Dt} + f\hat{\mathbf{k}} \times \vec{v}_h + \frac{1}{\rho_o} \nabla_z p = \vec{\mathcal{F}} \quad (1.78)$$

$$\epsilon_{nh} \frac{Dw}{Dt} + \frac{g\rho}{\rho_o} + \frac{1}{\rho_o} \frac{\partial p}{\partial z} = \epsilon_{nh} \mathcal{F}_w \quad (1.79)$$

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

$$\rho = \rho(\theta, S, p) \quad (1.81)$$

$$\frac{D\theta}{Dt} = \mathcal{Q}_\theta \quad (1.82)$$

$$\frac{DS}{Dt} = \mathcal{Q}_s \quad (1.83)$$

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 $\frac{Dp'}{Dt} \ll (\frac{Dp_o}{Dt}, \nabla \cdot \vec{v}_h)$ in the continuity equations and $\frac{\partial \rho}{\partial p} \Big|_{\theta, S} \frac{Dp'}{Dt} \ll \frac{\partial \rho}{\partial p} \Big|_{\theta, S} \frac{Dp_o}{Dt}$ in the EOS (1.70). Thus we set $\epsilon_s = 0$, removing the dependency on p' in the continuity equation and EOS. Expanding $\frac{Dp_o(z)}{Dt} = -g\rho_o w$ then leads to the anelastic continuity equation:

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

A slightly different route leads to the quasi-Boussinesq continuity equation where we use the scaling $\frac{\partial \rho'}{\partial t} + \nabla_3 \cdot \rho' \vec{v} \ll \nabla_3 \cdot \rho_o \vec{v}$ yielding:

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

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

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

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

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

$$\epsilon_{nh} \frac{Dw}{Dt} + \frac{g\rho}{\rho_o} + \frac{1}{\rho_o} \frac{\partial p}{\partial z} = \epsilon_{nh} \mathcal{F}_w \quad (1.87)$$

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

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

$$\frac{D\theta}{Dt} = \mathcal{Q}_\theta \quad (1.90)$$

$$\frac{DS}{Dt} = \mathcal{Q}_s \quad (1.91)$$

1.5.1.3 Incompressible z-coordinate equations

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

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

$$\epsilon_{nh} \frac{Dw}{Dt} + \frac{g\rho}{\rho_c} + \frac{1}{\rho_c} \frac{\partial p}{\partial z} = \epsilon_{nh} \mathcal{F}_w \quad (1.93)$$

$$\nabla_z \cdot \vec{v}_h + \frac{\partial w}{\partial z} = 0 \quad (1.94)$$

$$\rho = \rho(\theta, S) \quad (1.95)$$

$$\frac{D\theta}{Dt} = \mathcal{Q}_\theta \quad (1.96)$$

$$\frac{DS}{Dt} = \mathcal{Q}_s \quad (1.97)$$

where ρ_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 ρ_o are unimportant while the compressible effects in ρ' are:

$$\rho_o = \rho_c$$

$$\rho' = \rho(\theta, S, p_o(z)) - \rho_o$$

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

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

$$\epsilon_{nh} \frac{Dw}{Dt} + \frac{g\rho'}{\rho_c} + \frac{1}{\rho_c} \frac{\partial p'}{\partial z} = \epsilon_{nh} \mathcal{F}_w \quad (1.99)$$

$$\nabla_z \cdot \vec{v}_h + \frac{\partial w}{\partial z} = 0 \quad (1.100)$$

$$\rho' = \rho(\theta, S, p_o(z)) - \rho_c \quad (1.101)$$

$$\frac{D\theta}{Dt} = Q_\theta \quad (1.102)$$

$$\frac{DS}{Dt} = Q_s \quad (1.103)$$

Note that the hydrostatic pressure of the resting fluid, including that associated with ρ_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:

$$u = r \cos \varphi \frac{D\lambda}{Dt}$$

$$v = r \frac{D\varphi}{Dt}$$

$$\dot{r} = \frac{Dr}{Dt}$$

(see [Figure 1.20](#)) Here φ is the latitude, λ the longitude, r the radial distance of the particle from the center of the earth, Ω is the angular speed of rotation of the Earth and D/Dt is the total derivative.

The ‘grad’ (∇) and ‘div’ ($\nabla \cdot$) operators are defined by, in spherical coordinates:

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

$$\nabla \cdot v \equiv \frac{1}{r \cos \varphi} \left\{ \frac{\partial u}{\partial \lambda} + \frac{\partial}{\partial \varphi} (v \cos \varphi) \right\} + \frac{1}{r^2} \frac{\partial (r^2 \dot{r})}{\partial r}$$

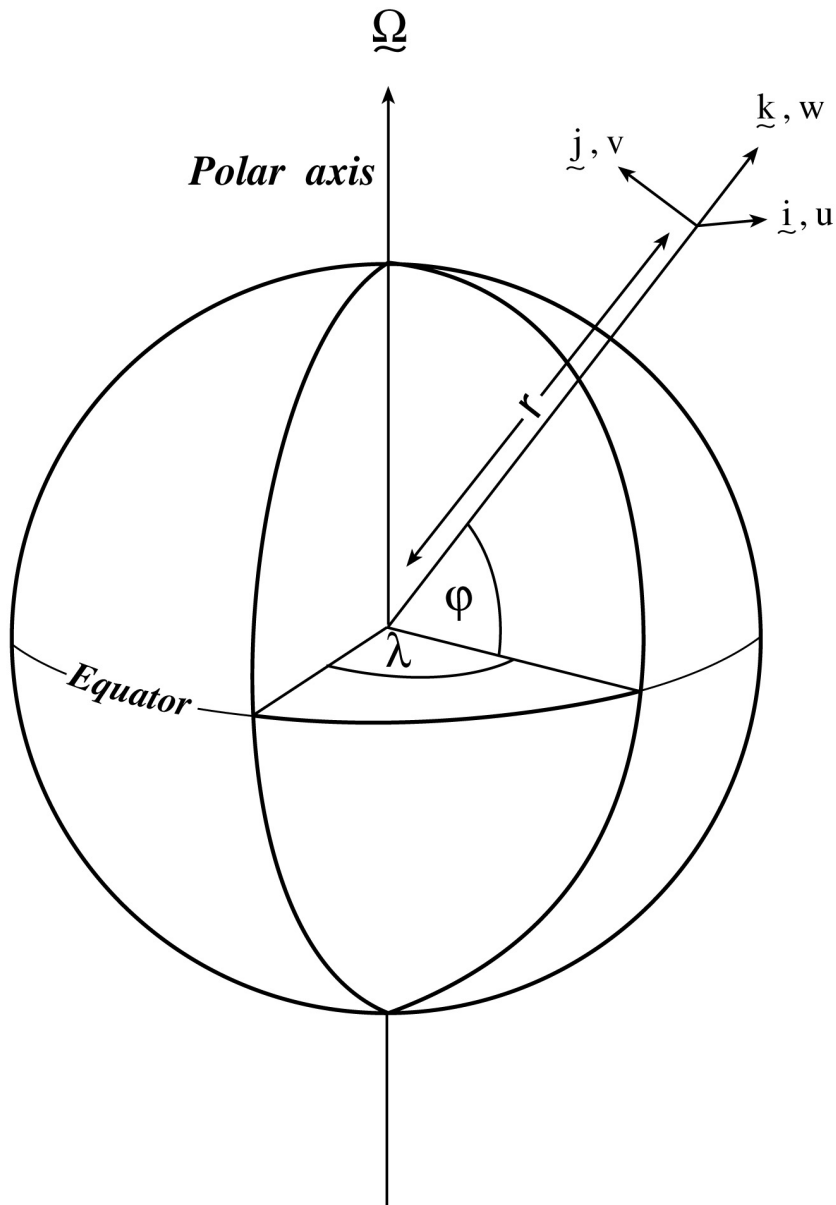


Figure 1.20: Spherical polar coordinates: longitude λ , latitude ϕ and r the distance from the center.

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 \mathbf{v}_h or \vec{v}_h in chapter 1) and a 3D vector is simply written \vec{v} (instead of \vec{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_ζ : horizontal area of a grid cell surrounding θ, u, v, ζ point

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

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

Basic operators:

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

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

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

$$\bar{\nabla} = \text{horizontal gradient operator} : \bar{\nabla} \Phi = \{\delta_x \Phi, \delta_y \Phi\}$$

$$\bar{\nabla} \cdot = \text{horizontal divergence operator} : \bar{\nabla} \cdot \vec{f} = \frac{1}{\Delta} \{\delta_i \Delta y f_x + \delta_j \Delta x f_y\}$$

$$\bar{\nabla}^2 = \text{horizontal Laplacian operator} : \bar{\nabla}^2 \Phi = \bar{\nabla} \cdot \bar{\nabla} \Phi$$

2.2 Time-stepping

The equations of motion integrated by the model involve four prognostic equations for flow, u and v , temperature, θ , and salt/moisture, S , and three diagnostic equations for vertical flow, w , density/buoyancy, ρ/b , and pressure/geo-potential, ϕ_{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:

$$\begin{aligned}\partial_t u + g \partial_x \eta &= G_u \\ \partial_t v + g \partial_y \eta &= G_v \\ \partial_x u + \partial_y v + \partial_z w &= 0\end{aligned}$$

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:

$$\partial_x H \hat{u} + \partial_y H \hat{v} = 0 \quad (2.1)$$

Here, $H \hat{u} = \int_H u dz$ is the depth integral of u , similarly for $H \hat{v}$. 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 discretized in time and space as follows:

$$u^{n+1} + \Delta t g \partial_x \eta^{n+1} = u^n + \Delta t G_u^{(n+1/2)} \quad (2.2)$$

$$v^{n+1} + \Delta t g \partial_y \eta^{n+1} = v^n + \Delta t G_v^{(n+1/2)} \quad (2.3)$$

$$\partial_x H \widehat{u^{n+1}} + \partial_y H \widehat{v^{n+1}} = 0 \quad (2.4)$$

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 η^{n+1} . Equations (2.2), (2.3) and (2.4) can then be re-arranged as follows:

$$u^* = u^n + \Delta t G_u^{(n+1/2)} \quad (2.5)$$

$$v^* = v^n + \Delta t G_v^{(n+1/2)} \quad (2.6)$$

$$\partial_x \Delta t g H \partial_x \eta^{n+1} + \partial_y \Delta t g H \partial_y \eta^{n+1} = \partial_x H \widehat{u^*} + \partial_y H \widehat{v^*} \quad (2.7)$$

$$u^{n+1} = u^* - \Delta t g \partial_x \eta^{n+1} \quad (2.8)$$

$$v^{n+1} = v^* - \Delta t g \partial_y \eta^{n+1} \quad (2.9)$$

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.

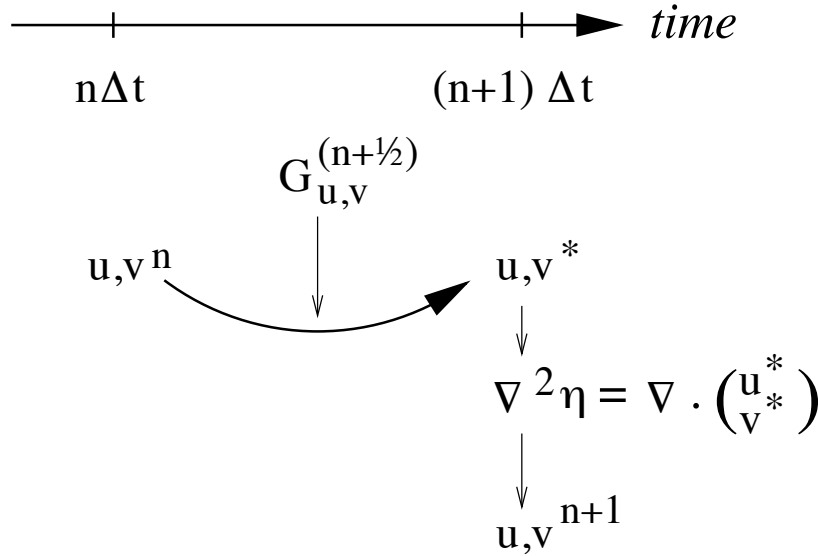


Figure 2.1: A schematic of the evolution in time of the pressure method algorithm. A prediction for the flow variables at time level $n + 1$ is made based only on the explicit terms, $G^{(n+1/2)}$, and denoted u^*, v^* . Next, a pressure field is found such that u^{n+1}, v^{n+1} will be non-divergent.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

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, $\widehat{Hu^*}$ and $\widehat{Hv^*}$, 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

FORWARD_STEP	
DYNAMICS	
TIMESTEP	u^*, v^* (2.5) , (2.6)
SOLVE_FOR_PRESSURE	
CALC_DIV_GHAT	$\widehat{Hu^*}, \widehat{Hv^*}$ (2.7)
CG2D	η^{n+1} (2.7)

MOMENTUM_CORRECTION_STEP
 CALC_GRAD_PHI_SURF
 CORRECTION_STEP

$\nabla\eta^{n+1}$
 u^{n+1}, v^{n+1} (2.8), (2.9)

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 (`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:

$$\begin{aligned} u^{n+1} - \Delta t \partial_z A_v \partial_z u^{n+1} + \Delta t g \partial_x \eta^{n+1} &= u^n + \Delta t G_u^{(n+1/2)} \\ v^{n+1} - \Delta t \partial_z A_v \partial_z v^{n+1} + \Delta t g \partial_y \eta^{n+1} &= v^n + \Delta t G_v^{(n+1/2)} \end{aligned}$$

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:

$$\partial_t \eta + \partial_x H \hat{u} + \partial_y H \hat{v} = P - E + R \quad (2.10)$$

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

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

$$\eta^{n+1} + \Delta t \partial_x H \widehat{u^{n+1}} + \Delta t \partial_y H \widehat{v^{n+1}} = \eta^n + \Delta t (P - E) \quad (2.11)$$

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:

$$u^* = u^n + \Delta t G_u^{(n+1/2)} \quad (2.12)$$

$$v^* = v^n + \Delta t G_v^{(n+1/2)} \quad (2.13)$$

$$\eta^* = \epsilon_{fs} (\eta^n + \Delta t (P - E)) - \Delta t (\partial_x H \widehat{u^*} + \partial_y H \widehat{v^*}) \quad (2.14)$$

$$\partial_x g H \partial_x \eta^{n+1} + \partial_y g H \partial_y \eta^{n+1} - \frac{\epsilon_{fs} \eta^{n+1}}{\Delta t^2} = -\frac{\eta^*}{\Delta t^2} \quad (2.15)$$

$$u^{n+1} = u^* - \Delta t g \partial_x \eta^{n+1} \quad (2.16)$$

$$v^{n+1} = v^* - \Delta t g \partial_y \eta^{n+1} \quad (2.17)$$

Equations (2.12) to (2.17), solved sequentially, represent the pressure method algorithm with a backward implicit, linearized free surface. The method is still formerly a pressure method because in the limit of large Δt 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 introduced a switch-like parameter, ϵ_{fs} (`freesurfFac`), which selects between the free-surface and rigid-lid; $\epsilon_{fs} = 1$ allows the free-surface to evolve; $\epsilon_{fs} = 0$ 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 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 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). In the previous sections, we summarized an explicit scheme as:

$$\tau^* = \tau^n + \Delta t G_\tau^{(n+1/2)} \quad (2.18)$$

where τ could be any prognostic variable (u , v , θ or S) and τ^* is an explicit estimate of τ^{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 and for this we use the quasi-second order Adams-Bashforth method:

$$G_\tau^{(n+1/2)} = (3/2 + \epsilon_{AB})G_\tau^n - (1/2 + \epsilon_{AB})G_\tau^{n-1} \quad (2.19)$$

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 ϵ_{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.

2.6 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:

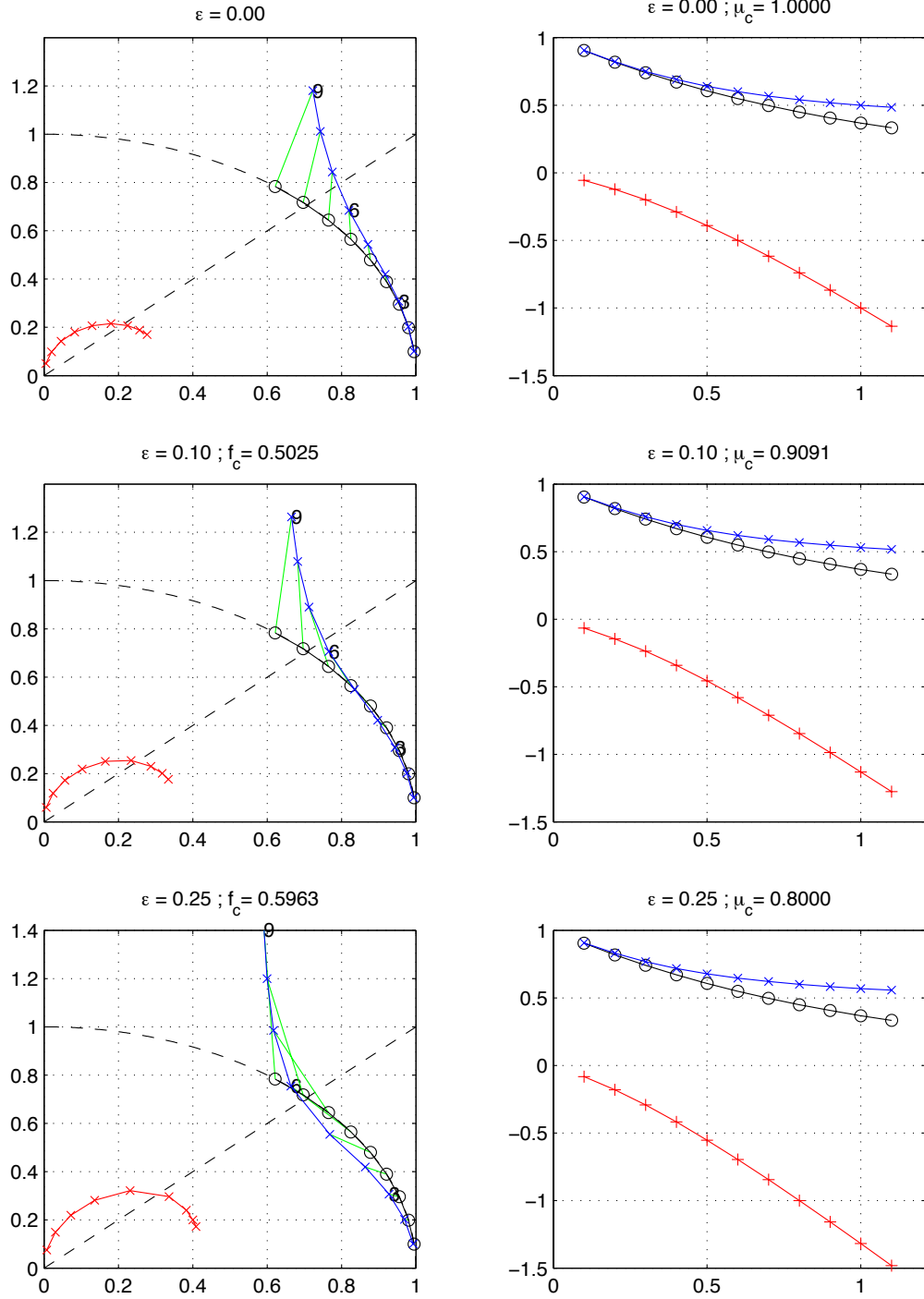


Figure 2.2: Oscillatory and damping response of quasi-second order Adams-Bashforth scheme for different values of the ϵ_{AB} parameter (0.0, 0.1, 0.25, from top to bottom) The analytical solution (in black), the physical mode (in blue) and the numerical mode (in red) are represented with a CFL step of 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).

$$\tau^{n+1} - \Delta t \partial_r \kappa_v \partial_r \tau^{n+1} = \tau^n + \Delta t G_\tau^{(n+1/2)} \quad (2.20)$$

where $G_\tau^{(n+1/2)}$ is the remaining explicit terms extrapolated using the Adams-Bashforth method as described above. Equation (2.20) can be split into:

$$\tau^* = \tau^n + \Delta t G_\tau^{(n+1/2)} \quad (2.21)$$

$$\tau^{n+1} = \mathcal{L}_\tau^{-1}(\tau^*) \quad (2.22)$$

where \mathcal{L}_τ^{-1} is the inverse of the operator

$$\mathcal{L}_\tau = [1 + \Delta t \partial_r \kappa_v \partial_r]$$

Equation (2.21) looks exactly as (2.18) while (2.22) involves an operator or matrix inversion. By re-arranging (2.20) 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 over all 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	
either	
EXTERNAL_FORCING	$G_\theta^n = G_\theta(u, \theta^n)$
ADAMS_BASHFORTH2	$G_\theta^n = G_\theta^n + \mathcal{Q}$
or	
EXTERNAL_FORCING	$G_\theta^{(n+1/2)} = G_\theta^{(n+1/2)} + \mathcal{Q}$
TIMESTEP_TRACER	τ^* (2.18)
IMPLDIFF	$\tau^{(n+1)}$ (2.22)

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 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 follow equations:

$$G_{\theta,S}^n = G_{\theta,S}(u^n, \theta^n, S^n) \quad (2.23)$$

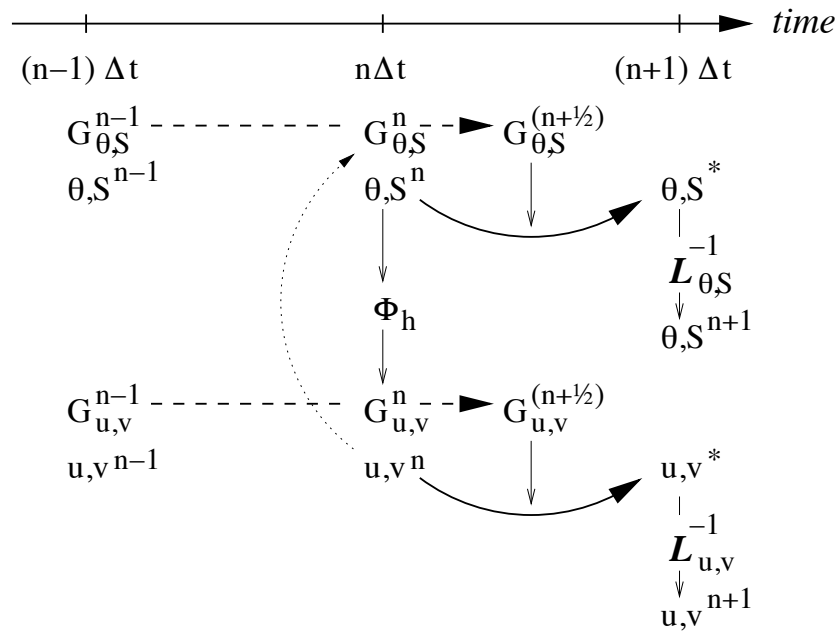


Figure 2.3: A schematic of the explicit Adams-Bashforth and implicit time-stepping phases of the algorithm. All prognostic variables are co-located in time. Explicit tendencies are evaluated at time level n as a function of the state at that time level (dotted arrow). The explicit tendency from the previous 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 $n + 1$.

$$G_{\theta,S}^{(n+1/2)} = (3/2 + \epsilon_{AB})G_{\theta,S}^n - (1/2 + \epsilon_{AB})G_{\theta,S}^{n-1} \quad (2.24)$$

$$(\theta^*, S^*) = (\theta^n, S^n) + \Delta t G_{\theta,S}^{(n+1/2)} \quad (2.25)$$

$$(\theta^{n+1}, S^{n+1}) = \mathcal{L}_{\theta,S}^{-1}(\theta^*, S^*) \quad (2.26)$$

$$\phi_{hyd}^n = \int b(\theta^n, S^n) dr \quad (2.27)$$

$$\vec{G}_{\vec{v}}^n = \vec{G}_{\vec{v}}(\vec{v}^n, \phi_{hyd}^n) \quad (2.28)$$

$$\vec{G}_{\vec{v}}^{(n+1/2)} = (3/2 + \epsilon_{AB})\vec{G}_{\vec{v}}^n - (1/2 + \epsilon_{AB})\vec{G}_{\vec{v}}^{n-1} \quad (2.29)$$

$$\vec{v}^* = \vec{v}^n + \Delta t \vec{G}_{\vec{v}}^{(n+1/2)} \quad (2.30)$$

$$\vec{v}^{**} = \mathcal{L}_{\vec{v}}^{-1}(\vec{v}^*) \quad (2.31)$$

$$\eta^* = \epsilon_{fs}(\eta^n + \Delta t(P - E)) - \Delta t \nabla \cdot H \widehat{\vec{v}^{**}} \quad (2.32)$$

$$\nabla \cdot g H \nabla \eta^{n+1} - \frac{\epsilon_{fs} \eta^{n+1}}{\Delta t^2} = - \frac{\eta^*}{\Delta t^2} \quad (2.33)$$

$$\vec{v}^{n+1} = \vec{v}^{**} - \Delta t g \nabla \eta^{n+1} \quad (2.34)$$

Figure 2.3 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, $\mathcal{L}_{\theta,S}^{-1}$, then yields the new tracer fields at $n + 1$. All these operations are carried out in subroutine `THERMODYNAMICS` and subsidiaries, which correspond to equations (2.23) to (2.26). Similarly illustrated is the Adams-Bashforth extrapolation of accelerations, stepping forward and solving of implicit viscosity and surface pressure gradient terms, corresponding to equations (2.28) to (2.34). 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

FORWARD_STEP	
EXTERNAL_FIELDS_LOAD	
DO_ATMOSPHERIC_PHYS	
DO_OCEANIC_PHYS	
THERMODYNAMICS	
CALC_GT	
GAD_CALC_RHS	$G_\theta^n = G_\theta(u, \theta^n)$ (2.23)
EXTERNAL_FORCING	$G_\theta^n = G_\theta^n + Q$
ADAMS_BASHFORTH2	$G_\theta^{(n+1/2)}$ (2.24)
TIMESTEP_TRACER	θ^* (2.25)
IMPLDIFF	$\theta^{(n+1)}$ (2.26)
DYNAMICS	
CALC_PHI_HYD	ϕ_{hyd}^n (2.27)
MOM_FLUXFORM or MOM_VECINV	$G_{\vec{v}}^n$ (2.28)
TIMESTEP	\vec{v}^* (2.29), (2.30)
IMPLDIFF	\vec{v}^{**} (2.31)
UPDATE_R_STAR or UPDATE_SURF_DR (NonLin-FS only)	
SOLVE_FOR_PRESSURE	
CALC_DIV_GHAT	η^* (2.32)
CG2D	η^{n+1} (2.33)
MOMENTUM_CORRECTION_STEP	
CALC_GRAD_PHI_SURF	$\nabla \eta^{n+1}$
CORRECTION_STEP	u^{n+1}, v^{n+1} (2.34)
TRACERS_CORRECTION_STEP	
CYCLE_TRACER	θ^{n+1}
SHAP_FILT_APPLY_TS or ZONAL_FILT_APPLY_TS	
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 the thermodynamic variables with the flow variables. Figure 2.4 illustrates the staggering and algorithm. The key difference between this and Figure 2.3 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, (2.23) to (2.34), annotating the position in time of variables appropriately:

$$\phi_{hyd}^n = \int b(\theta^n, S^n) dr \quad (2.35)$$

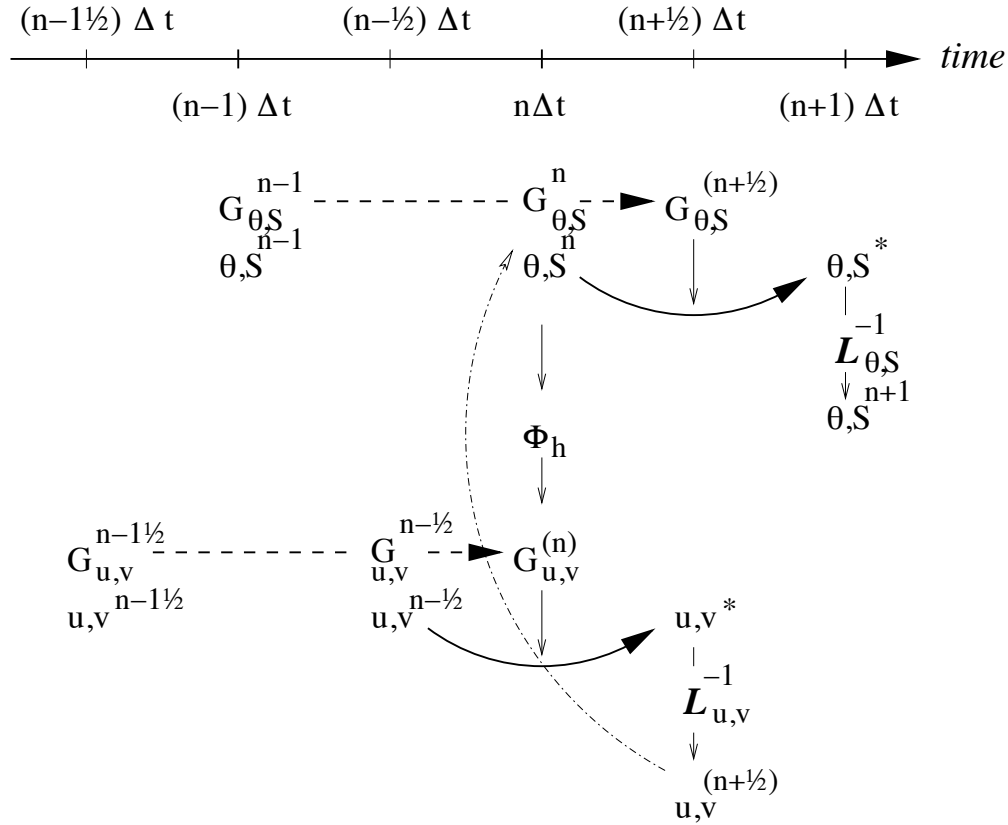


Figure 2.4: A schematic of the explicit Adams-Bashforth and implicit time-stepping phases of the algorithm but with staggering in time of thermodynamic variables with the flow. Explicit momentum tendencies are evaluated at time level $n - 1/2$ as a function of the flow field at that time level $n - 1/2$. 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 ϕ_{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 $\mathcal{L}_{u,v}$ (vertical arrows) is then applied to the previous estimation of the 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 $n + 1$.

$$\vec{G}_{\vec{v}}^{n-1/2} = \vec{G}_{\vec{v}}(\vec{v}^{n-1/2}) \quad (2.36)$$

$$\vec{G}_{\vec{v}}^{(n)} = (3/2 + \epsilon_{AB})\vec{G}_{\vec{v}}^{n-1/2} - (1/2 + \epsilon_{AB})\vec{G}_{\vec{v}}^{n-3/2} \quad (2.37)$$

$$\vec{v}^* = \vec{v}^{n-1/2} + \Delta t \left(\vec{G}_{\vec{v}}^{(n)} - \nabla \phi_{hyd}^n \right) \quad (2.38)$$

$$\vec{v}^{**} = \mathcal{L}_{\vec{v}}^{-1}(\vec{v}^*) \quad (2.39)$$

$$\eta^* = \epsilon_{fs} \left(\eta^{n-1/2} + \Delta t (P - E)^n \right) - \Delta t \nabla \cdot H \widehat{\vec{v}^{**}} \quad (2.40)$$

$$\nabla \cdot g H \nabla \eta^{n+1/2} - \frac{\epsilon_{fs} \eta^{n+1/2}}{\Delta t^2} = - \frac{\eta^*}{\Delta t^2} \quad (2.41)$$

$$\vec{v}^{n+1/2} = \vec{v}^{**} - \Delta t g \nabla \eta^{n+1/2} \quad (2.42)$$

$$G_{\theta,S}^n = G_{\theta,S}(u^{n+1/2}, \theta^n, S^n) \quad (2.43)$$

$$G_{\theta,S}^{(n+1/2)} = (3/2 + \epsilon_{AB})G_{\theta,S}^n - (1/2 + \epsilon_{AB})G_{\theta,S}^{n-1} \quad (2.44)$$

$$(\theta^*, S^*) = (\theta^n, S^n) + \Delta t G_{\theta,S}^{(n+1/2)} \quad (2.45)$$

$$(\theta^{n+1}, S^{n+1}) = \mathcal{L}_{\theta,S}^{-1}(\theta^*, S^*) \quad (2.46)$$

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

ϕ_{hyd}^n (2.35)

MOM_FLUXFORM or MOM_VECINV	$G_{\vec{v}}^{n-1/2}$ (2.36)
TIMESTEP	\vec{v}^* (2.37), (2.38)
IMPLDIFF	\vec{v}^{**} (2.39)
UPDATE_R_STAR or UPDATE_SURF_DR (NonLin-FS only)	
SOLVE_FOR_PRESSURE	
CALC_DIV_GHAT	η^* (2.40)
CG2D	$\eta^{n+1/2}$ (2.41)
MOMENTUM_CORRECTION_STEP	
CALC_GRAD_PHI_SURF	$\nabla \eta^{n+1/2}$
CORRECTION_STEP	$u^{n+1/2}, v^{n+1/2}$ (2.42)
THERMODYNAMICS	
CALC_GT	
GAD_CALC_RHS	$G_{\theta}^n = G_{\theta}(u, \theta^n)$ (2.43)
EXTERNAL_FORCING	$G_{\theta}^n = G_{\theta}^n + Q$
ADAMS_BASHFORTH2	$G_{\theta}^{(n+1/2)}$ (2.44)
TIMESTEP_TRACER	θ^* (2.45)
IMPLDIFF	$\theta^{(n+1)}$ (2.46)
TRACERS_CORRECTION_STEP	
CYCLE_TRACER	θ^{n+1}
SHAP_FILT_APPLY_TS or ZONAL_FILT_APPLY_TS	
CONVECTIVE_ADJUSTMENT	

The only difficulty with this approach is apparent in equation (2.43) and illustrated by the dotted arrow connecting $u, v^{n+1/2}$ with G_{θ}^n . 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:

$$\frac{1}{\Delta t} u^{n+1} + g \partial_x \eta^{n+1} + \partial_x \phi_{nh}^{n+1} = \frac{1}{\Delta t} u^n + G_u^{(n+1/2)} \quad (2.47)$$

$$\frac{1}{\Delta t} v^{n+1} + g \partial_y \eta^{n+1} + \partial_y \phi_{nh}^{n+1} = \frac{1}{\Delta t} v^n + G_v^{(n+1/2)} \quad (2.48)$$

$$\frac{1}{\Delta t} w^{n+1} + \partial_r \phi_{nh}^{n+1} = \frac{1}{\Delta t} w^n + G_w^{(n+1/2)} \quad (2.49)$$

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

$$\partial_x u^{n+1} + \partial_y v^{n+1} + \partial_r w^{n+1} = 0 \quad (2.50)$$

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

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

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

$$\begin{aligned} u^{n+1} &= u^{**} - \Delta t \partial_x \phi_{nh}^{n+1} & u^{**} &= u^* - \Delta t g \partial_x \eta^{n+1} \\ v^{n+1} &= v^{**} - \Delta t \partial_y \phi_{nh}^{n+1} & v^{**} &= v^* - \Delta t g \partial_y \eta^{n+1} \end{aligned}$$

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

$$\partial_x H \partial_x (g \eta^{n+1} + \hat{\phi}_{nh}^{n+1}) + \partial_y H \partial_y (g \eta^{n+1} + \hat{\phi}_{nh}^{n+1}) - \frac{\epsilon_{fs} \eta^{n+1}}{\Delta t^2} = -\frac{\eta^*}{\Delta t^2} \quad (2.51)$$

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

The flow must satisfy non-divergence (equation (2.50)) locally, as well as depth integrated, and this constraint is used to form a 3-D elliptic equations for ϕ_{nh}^{n+1} :

$$\partial_{xx} \phi_{nh}^{n+1} + \partial_{yy} \phi_{nh}^{n+1} + \partial_{rr} \phi_{nh}^{n+1} = \partial_x u^{**} + \partial_y v^{**} + \partial_r w^* \quad (2.52)$$

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

$$u^* = u^n + \Delta t G_u^{(n+1/2)} \quad (2.53)$$

$$v^* = v^n + \Delta t G_v^{(n+1/2)} \quad (2.54)$$

$$w^* = w^n + \Delta t G_w^{(n+1/2)} \quad (2.55)$$

$$\eta^* = \epsilon_{fs} (\eta^n + \Delta t (P - E)) - \Delta t (\partial_x H \hat{u}^* + \partial_y H \hat{v}^*) \quad (2.56)$$

$$\partial_x g H \partial_x \eta^{n+1} + \partial_y g H \partial_y \eta^{n+1} - \frac{\epsilon_{fs} \eta^{n+1}}{\Delta t^2} = -\frac{\eta^*}{\Delta t^2} \quad (2.57)$$

$$u^{**} = u^* - \Delta t g \partial_x \eta^{n+1} \quad (2.58)$$

$$v^{**} = v^* - \Delta t g \partial_y \eta^{n+1} \quad (2.59)$$

$$\partial_{xx} \phi_{nh}^{n+1} + \partial_{yy} \phi_{nh}^{n+1} + \partial_{rr} \phi_{nh}^{n+1} = \partial_x u^{**} + \partial_y v^{**} + \partial_r w^* \quad (2.60)$$

$$u^{n+1} = u^{**} - \Delta t \partial_x \phi_{nh}^{n+1} \quad (2.61)$$

$$v^{n+1} = v^{**} - \Delta t \partial_y \phi_{nh}^{n+1} \quad (2.62)$$

$$\partial_r w^{n+1} = -\partial_x u^{n+1} - \partial_y v^{n+1} \quad (2.63)$$

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 [one day] 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:

$$\epsilon_{fs} \eta^{n+1} - \nabla_h \cdot \Delta t^2 (R_o - R_{fixed}) \nabla_h b_s \eta^{n+1} = \eta^* \quad (2.64)$$

where

$$\eta^* = \epsilon_{fs} \eta^n - \Delta t \nabla_h \cdot \int_{R_{fixed}}^{R_o} \vec{\nabla}^* dr + \epsilon_{fw} \Delta t (P - E)^n \quad (2.65)$$

S/R SOLVE_FOR_PRESSURE

```

u* : gU ( DYNVARS.h )
v* : gV ( DYNVARS.h )
η* : cg2d_b ( SOLVE_FOR_PRESSURE.h )
ηn+1 : etaN ( DYNVARS.h )

```

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

$$\vec{v}^{n+1} = \vec{v}^* - \Delta t \nabla_h b_s \eta^{n+1}$$

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 ϕ'_{nh} . This is obtained by substituting (2.47), (2.48) and (2.49) into continuity:

$$[\nabla_h^2 + \partial_{rr}] \phi'_{nh}{}^{n+1} = \frac{1}{\Delta t} \nabla_h \cdot \vec{v}^{**} + \partial_r \dot{r}^* \quad (2.66)$$

where

$$\vec{v}^{**} = \vec{v}^* - \Delta t \nabla_h b_s \eta^{n+1}$$

Note that η^{n+1} is also used to update the second RHS term $\partial_r \dot{r}^*$ 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:

$$\vec{v}^{n+1} = \vec{v}^{**} - \epsilon_{nh} \Delta t \nabla_h \phi'_{nh}{}^{n+1} \quad (2.67)$$

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

```

etan+1 : etaN ( DYNVARS.h )
phin+1nh : phi_nh ( NH_VARS.h )
u* : gU ( DYNVARS.h )
v* : gV ( DYNVARS.h )
un+1 : uVel ( DYNVARS.h )
vn+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 (2.64) 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 η^* 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 (β, γ) and an explicit part ($1 - \beta, 1 - \gamma$) for the surface pressure gradient (β) and for the barotropic flow divergence (γ). For instance, $\beta = \gamma = 1$ is the previous fully implicit scheme; $\beta = \gamma = 1/2$ is the non damping (energy conserving), unconditionally stable, Crank-Nicolson scheme; $(\beta, \gamma) = (1, 0)$ or $(0, 1)$ corresponds to the forward - backward scheme that conserves energy but is only stable for small time steps. In the code, β, γ 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.

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

$$\frac{\vec{v}^{n+1}}{\Delta t} + \nabla_h b_s [\beta \eta^{n+1} + (1 - \beta) \eta^n] + \epsilon_{nh} \nabla_h \phi'_{nh}{}^{n+1} = \frac{\vec{v}^n}{\Delta t} + \vec{G}_{\vec{v}}^{(n+1/2)} + \nabla_h \phi'_{hyd}{}^{(n+1/2)}$$

$$\epsilon_{fs} \frac{\eta^{n+1} - \eta^n}{\Delta t} + \nabla_h \cdot \int_{R_{fixed}}^{R_o} [\gamma \vec{v}^{n+1} + (1 - \gamma) \vec{v}^n] dr = \epsilon_{fw}(P - E) \quad (2.68)$$

We set

$$\begin{aligned} \vec{v}^* &= \vec{v}^n + \Delta t \vec{G}_{\vec{v}}^{(n+1/2)} + (\beta - 1) \Delta t \nabla_h b_s \eta^n + \Delta t \nabla_h \phi'_{hyd}{}^{(n+1/2)} \\ \eta^* &= \epsilon_{fs} \eta^n + \epsilon_{fw} \Delta t (P - E) - \Delta t \nabla_h \cdot \int_{R_{fixed}}^{R_o} [\gamma \vec{v}^* + (1 - \gamma) \vec{v}^n] dr \end{aligned}$$

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

$$\epsilon_{fs} \eta^{n+1} - \nabla_h \cdot \beta \gamma \Delta t^2 b_s (R_o - R_{fixed}) \nabla_h \eta^{n+1} = \eta^*$$

and then to compute (CORRECTION_STEP):

$$\vec{v}^{n+1} = \vec{v}^* - \beta \Delta t \nabla_h b_s \eta^{n+1}$$

Notes:

1. The RHS term of equation (2.68) corresponds the contribution of fresh water flux (P-E) to the free-surface variations ($\epsilon_{fw} = 1$, `useRealFreshWaterFlux = 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: $\epsilon_{fw}(\gamma(P - E)^{n+1/2} + (1 - \gamma)(P - E)^{n-1/2})$
2. The stability criteria with Crank-Nicolson time stepping for the pure linear gravity wave problem in cartesian coordinates is:
 - $\beta + \gamma < 1$: unstable
 - $\beta \geq 1/2$ and $\gamma \geq 1/2$: stable
 - $\beta + \gamma \geq 1$: stable if $c_{max}^2(\beta - 1/2)(\gamma - 1/2) + 1 \geq 0$ with $c_{max} = 2\Delta t \sqrt{gH} \sqrt{\frac{1}{\Delta x^2} + \frac{1}{\Delta y^2}}$
3. A similar mixed forward/backward time-stepping is also available for the non-hydrostatic algorithm, with a fraction β_{nh} ($0 < \beta_{nh} \leq 1$) of the non-hydrostatic pressure gradient being evaluated at time step $n + 1$ (backward in time) and the remaining part $(1 - \beta_{nh})$ being evaluated at time step n (forward in time). The run-time parameter `implicitNHPress` corresponding to the implicit fraction β_{nh} of the non-hydrostatic pressure is set by default to the implicit fraction β of surface pressure (`implicitSurfPress`), but can also be specified independently (in main parameter file `data`, `namelist PARM01`).

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 $\phi = \phi_{topo} - \int_{p_s}^p \alpha dp$, 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 ρ_c does not depend on z ($b_o = g$) and the surface reference position is uniformly $z = 0$ ($R_o = 0$), 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:

$$\phi' = \int_{R_o}^{r_{surf}} b \, dr + \int_r^{R_o} (b - b_o) \, dr \quad (2.69)$$

or:

$$\phi' = \int_{R_o}^{r_{surf}} b_o \, dr + \int_r^{r_{surf}} (b - b_o) \, dr \quad (2.70)$$

In section [Section 1.3.6](#), following eq. (2.69), the pressure/geo-potential ϕ' has been separated into surface (ϕ_s), and hydrostatic anomaly (ϕ'_{hyd}). In this section, the split between ϕ_s and ϕ'_{hyd} is made according to equation (2.70). 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 ϕ'_{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 ϕ_s and ϕ'_{hyd} from equations (2.69) and (2.70) converge toward the same (approximated) expressions: $\phi_s = \int_{R_o}^{r_{surf}} b_o \, dr$ and $\phi'_{hyd} = \int_r^{R_o} b' \, dr$. On the contrary, the unapproximated formulation (“non-linear free-surface”, see the next section) 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 : $b_s = b_o(R_o) = c_p \kappa (p_o / P_{SL}^o)^{(\kappa-1)} \theta_{ref}(p_o)$, with P_{SL}^o 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 η , 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 $R_{fixed}(x, y)$ up to $r_{surf} = R_o + \eta$:

$$\epsilon_{fs} \partial_t \eta = \dot{r}|_{r=r_{surf}} + \epsilon_{fw}(P - E) = -\nabla_h \cdot \int_{R_{fixed}}^{R_o + \eta} \vec{\nabla} dr + \epsilon_{fw}(P - E)$$

Since η has a direct effect on the horizontal velocity (through $\nabla_h \Phi_{surf}$), 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 (2.64) and (2.65) become:

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

where

$$\eta^* = \epsilon_{fs} \eta^n - \Delta t \nabla_h \cdot \int_{R_{fixed}}^{R_o + \eta^n} \vec{\nabla}^* dr + \epsilon_{fw} \Delta t (P - E)^n$$

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

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

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

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 (2.65) but not directly in the equation (2.64).

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 2.1: Non-linear free-surface flags

parameter	value	description
	-1	linear free-surface, restart from a pickup file
		produced with <code>#undef EXACT_CONSERV</code> code
	0	Linear free-surface
<code>nonlinFreeSurf</code>	4	Non-linear free-surface
	3	same as 4 but neglecting $\int_{R_o}^{R_o + \eta} b' dr$ in Φ'_{hyd}
	2	same as 3 but do not update cg2d solver matrix
	1	same as 2 but treat momentum as in Linear FS
	0	do not use r^* vertical coordinate (= default)
<code>select_rStar</code>	2	use r^* vertical coordinate
	1	same as 2 but without the contribution of the
		slope of the coordinate in $\nabla \Phi$

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 η) 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 θ 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) [CAHMO4]). 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_θ terms (first step) as in the linear free surface formulation (with the “surface correction” turned “on”, see tracer section):

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

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

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

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

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 simplified 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. (2.36) to (2.46)) 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, $Q^{n+1/2}$, $P^{n+1/2}$, $F_{\vec{v}}^n$ respectively. h^n and dh^n are the column and grid box thickness in r-coordinate.

$$\phi_{hyd}^n = \int b(\theta^n, S^n, r) dr \quad (2.71)$$

$$\vec{G}_{\vec{v}}^{n-1/2} = \vec{G}_{\vec{v}}(dh^{n-1}, \vec{v}^{n-1/2}) ; \quad \vec{G}_{\vec{v}}^{(n)} = \frac{3}{2} \vec{G}_{\vec{v}}^{n-1/2} - \frac{1}{2} \vec{G}_{\vec{v}}^{n-3/2} \quad (2.72)$$

$$\vec{v}^* = \vec{v}^{n-1/2} + \Delta t \frac{dh^{n-1}}{dh^n} \left(\vec{G}_{\vec{v}}^{(n)} + F_{\vec{v}}^n / dh^{n-1} \right) - \Delta t \nabla \phi_{hyd}^n \quad (2.73)$$

→ update model geometry : **hFac**(dh^n)

$$\begin{aligned}\eta^{n+1/2} &= \eta^{n-1/2} + \Delta t P^{n+1/2} - \Delta t \nabla \cdot \int \vec{v}^{n+1/2} dh^n \\ &= \eta^{n-1/2} + \Delta t P^{n+1/2} - \Delta t \nabla \cdot \int (\vec{v}^* - g \Delta t \nabla \eta^{n+1/2}) dh^n\end{aligned}\quad (2.74)$$

$$\vec{v}^{n+1/2} = \vec{v}^* - g \Delta t \nabla \eta^{n+1/2} \quad (2.75)$$

$$h^{n+1} = h^n + \Delta t P^{n+1/2} - \Delta t \nabla \cdot \int \vec{v}^{n+1/2} dh^n \quad (2.76)$$

$$G_\theta^n = G_\theta(dh^n, u^{n+1/2}, \theta^n) ; \quad G_\theta^{(n+1/2)} = \frac{3}{2} G_\theta^n - \frac{1}{2} G_\theta^{n-1} \quad (2.77)$$

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

Two steps have been added to linear free-surface algorithm (eq. (2.36) to (2.46)): Firstly, the model “geometry” (here the **hFacC,W,S**) is updated just before entering **SOLVE_FOR_PRESSURE**, using the current dh^n field. Secondly, the vertically integrated continuity equation (2.76) has been added (**exactConserv** = **TRUE**., in parameter file **data**, namelist **PARM01**) just before computing the vertical velocity, in subroutine **INTEGR_CONTINUITY**. Although this equation might appear redundant with (2.74), the integrated column thickness h^{n+1} will be different from $\eta^{n+1/2} + H$ in the following cases:

- when Crank-Nicolson time-stepping is used (see Section 2.10.1).
- when filters are applied to the flow field, after (2.75), and alter the divergence of the flow.
- when the solver does not iterate until convergence; for example, because a too large residual target was set (**cg2dTargetResidual**, parameter file **data**, namelist **PARM02**).

In this staggered time-stepping algorithm, the momentum tendencies are computed using dh^{n-1} geometry factors (2.72) and then rescaled in subroutine **TIMESTEP**, (2.73), similarly to tracer tendencies (see Section 2.10.2.3). The tracers are stepped forward later, using the recently updated flow field $\mathbf{v}^{n+1/2}$ and the corresponding model geometry dh^n to compute the tendencies (2.77); then the tendencies are rescaled by dh^n/dh^{n+1} to derive the new tracers values $(\theta, S)^{n+1}$ ((2.78), in subroutines **CALC_GT**, **CALC_GS**).

Note that the fresh-water input is added in a consistent way in the continuity equation and in the tracer equation, taking into account the fresh-water temperature θ_{rain} .

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 ($\eta^{n-1/2}$, $\mathbf{v}^{n-1/2}$, θ^n , S^n , $\mathbf{G}_v^{n-3/2}$, $G_{\theta,S}^{n-1}$) 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. (2.71) to (2.78), **FORWARD_STEP**).

S/R INTEGR_CONTINUITY

```
h^{n+1} - H_o : etaH ( DYNVARS.h )
h^n - H_o : etaHnm1 ( SURFACE.h )
(h^{n+1} - h^n)/\Delta t : 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 later 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 σ_z, σ_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 spacial 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 Δx_i :

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

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

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

$$F = u\bar{\theta} - \frac{\kappa}{\Delta x_c} \partial_i \theta$$

and this is where truncation errors can enter the solution. The method for obtaining $\bar{\theta}$ 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 2.5 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).

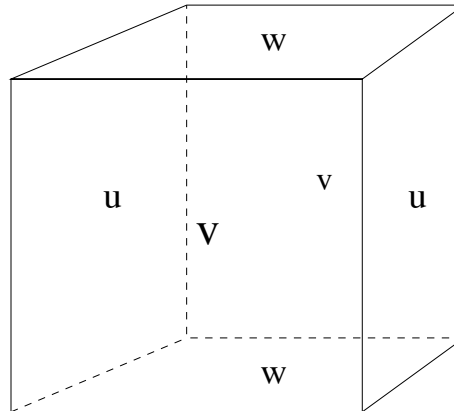


Figure 2.5: Three dimensional staggering of velocity components. This facilitates the natural discretization of the continuity and tracer equations.

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 are 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 its sides and its 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.

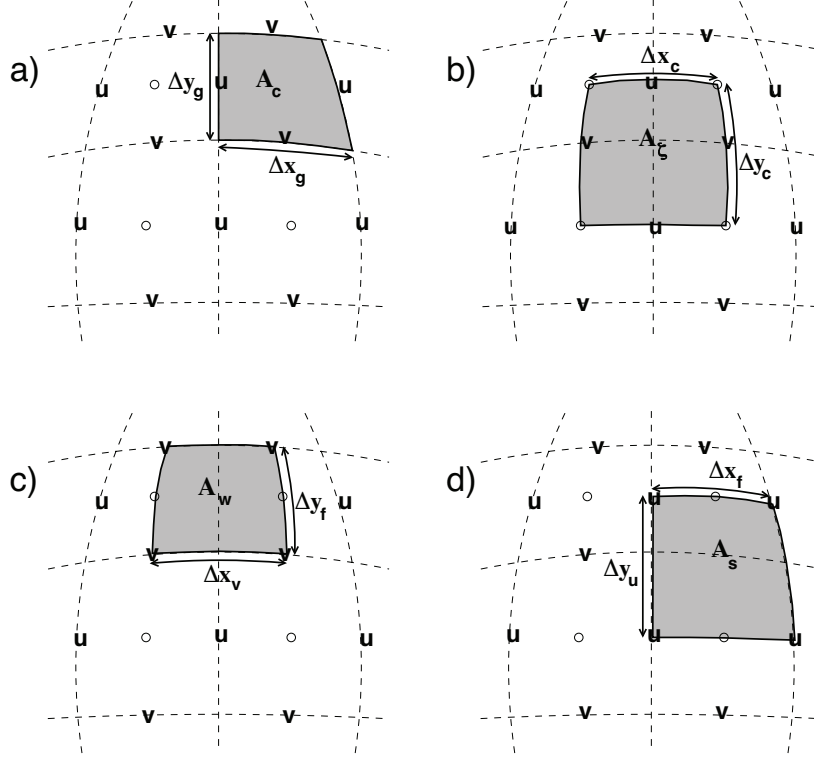


Figure 2.6: Staggering of horizontal grid descriptors (lengths and areas). The grid lines indicate the tracer cell boundaries and are the reference grid for all panels. a) The area of a tracer cell, A_c , is bordered by the lengths Δx_g and Δy_g . b) The area of a vorticity cell, A_ζ , is bordered by the lengths Δx_c and Δy_c . c) The area of a u cell, A_w , is bordered by the lengths Δx_v and Δy_f . d) The area of a v cell, A_s , is bordered by the lengths Δx_f and Δy_u .

Figure 2.6 (a) shows the tracer cell (synonymous with the continuity cell). The length of the southern edge, Δx_g , western edge, Δy_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 2.6 (b) shows the vorticity cell. The length of the southern edge, Δx_c , western edge, Δy_c and surface area, A_ζ , presented in the vertical are stored in arrays **dxC**, **dyC** and **rAz**. The “z” suffix indicates that the lengths are measured between the cell centers and the “ζ” 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 2.6 (c) shows the “u” or western (w) cell. The length of the southern edge, Δx_v , eastern edge, Δy_f and surface area, A_w , 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 2.6 (d) shows the “v” or southern (s) cell. The length of the northern edge, Δx_f , western edge, Δy_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 $\mathbf{dxF(i,j)}$ is positioned to the north of $\mathbf{rAs(i,j)}$ and $\mathbf{dyU(i,j)}$ positioned to the west.

S/R INI_CARTESIAN_GRID , INI_SPHERICAL_POLAR_GRID , INI_CURVILINEAR_GRID

$A_c, A_\zeta, A_w, A_s : \mathbf{rA}, \mathbf{rAz}, \mathbf{rAw}, \mathbf{rAs}$ (GRID.h)

$\Delta x_g, \Delta y_g : \mathbf{dxG}, \mathbf{dyG}$ (GRID.h)

$\Delta x_c, \Delta y_c : \mathbf{dxC}, \mathbf{dyC}$ (GRID.h)

$\Delta x_f, \Delta y_f : \mathbf{dxF}, \mathbf{dyF}$ (GRID.h)

$\Delta x_v, \Delta y_u : \mathbf{dxV}, \mathbf{dyU}$ (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

$A_c^{-1}, A_\zeta^{-1}, A_w^{-1}, A_s^{-1} : \mathbf{recip_rA}, \mathbf{recip_rAz}, \mathbf{recip_rAw}, \mathbf{recip_rAs}$ (GRID.h)

$\Delta x_g^{-1}, \Delta y_g^{-1} : \mathbf{recip_dxG}, \mathbf{recip_dyG}$ (GRID.h)

$\Delta x_c^{-1}, \Delta y_c^{-1} : \mathbf{recip_dxC}, \mathbf{recip_dyC}$ (GRID.h)

$\Delta x_f^{-1}, \Delta y_f^{-1} : \mathbf{recip_dxF}, \mathbf{recip_dyF}$ (GRID.h)

$\Delta x_v^{-1}, \Delta y_u^{-1} : \mathbf{recip_dxV}, \mathbf{recip_dyU}$ (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 can not 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

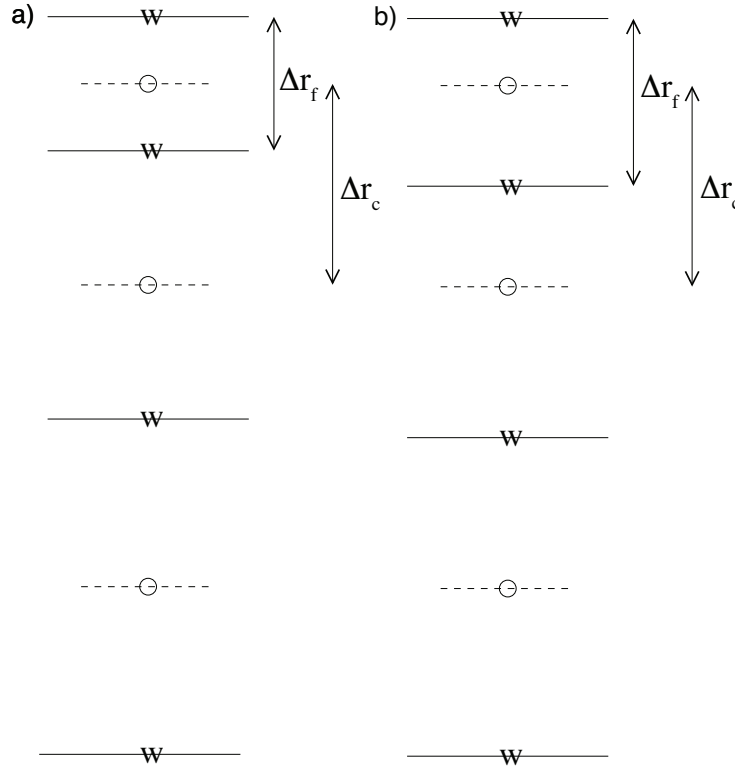


Figure 2.7: Two versions of the vertical grid. a) The cell centered approach where the interface depths are specified and the tracer points centered in between the interfaces. b) The interface centered approach where tracer levels are specified and the w-interfaces are centered in between.

As for the horizontal grid, we use the suffixes “c” and “f” to indicate faces and centers. Figure 2.7 (a) shows the default vertical grid used by the model. Δr_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 Δr_f^{-1} and Δr_c^{-1} are pre-calculated (also in subroutine `INI_VERTICAL_GRID`). All vertical grid descriptors are stored in common blocks in `GRID.h`.

The above grid Figure 2.7 (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 2.7 (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 $Nr + 1$ vertical distances `delRc` (parameter

file data, namelist PARM04, e.g. `ideal_2D_oce/input/data`) corresponding to surface to center, $Nr-1$ 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.

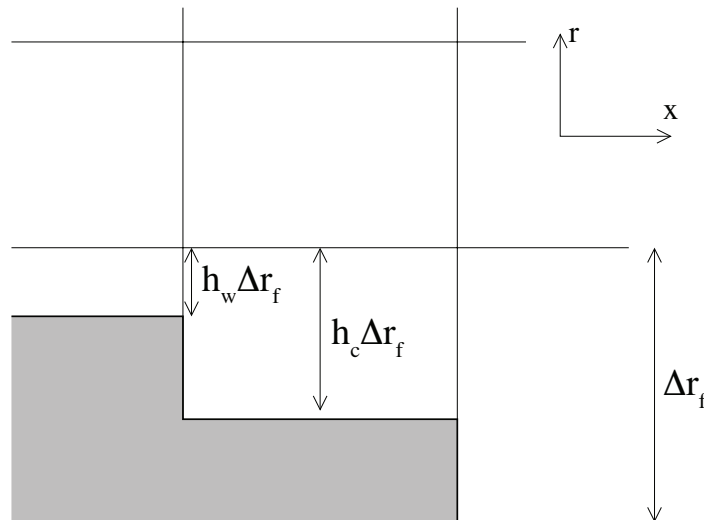


Figure 2.8: A schematic of the x-r plane showing the location of the non-dimensional fractions h_c and h_w . The physical thickness of a tracer cell is given by $h_c(i, j, k)\Delta r_f(k)$ and the physical thickness of the open side is given by $h_w(i, j, k)\Delta r_f(k)$.

Figure 2.8 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, $\min(\Delta z_f)$, but

setting `hFacMin` to some small fraction then the model will only lop thick layers but retain stability based on the thinnest unlopped thickness; $\min(\Delta z_f, hFacMinDr)$.

S/R :filelink:INI_MASKS_ETC

h_c, h_w, h_s : `hFacC, hFacW, hFacS (GRID.h)`

$h_c^{-1}, h_w^{-1}, h_s^{-1}$: `recip_hFacC, recip_hFacW, recip_hFacS (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:

$$\partial_t u + \frac{1}{\Delta x_c} \delta_i \left. \frac{\partial \Phi}{\partial r} \right|_s \eta + \frac{\epsilon_{nh}}{\Delta x_c} \delta_i \Phi'_{nh} = G_u - \frac{1}{\Delta x_c} \delta_i \Phi'_h \quad (2.78)$$

$$\partial_t v + \frac{1}{\Delta y_c} \delta_j \left. \frac{\partial \Phi}{\partial r} \right|_s \eta + \frac{\epsilon_{nh}}{\Delta y_c} \delta_j \Phi'_{nh} = G_v - \frac{1}{\Delta y_c} \delta_j \Phi'_h \quad (2.79)$$

$$\epsilon_{nh} \left(\partial_t w + \frac{1}{\Delta r_c} \delta_k \Phi'_{nh} \right) = \epsilon_{nh} G_w + \bar{b}^k - \frac{1}{\Delta r_c} \delta_k \Phi'_h \quad (2.80)$$

$$\delta_i \Delta y_g \Delta r_f h_w u + \delta_j \Delta x_g \Delta r_f h_s v + \delta_k \mathcal{A}_c w = \mathcal{A}_c \delta_k (P - E)_{r=0} \quad (2.81)$$

where the continuity equation has been most naturally discretized by staggering the three components of velocity as shown in [Figure 2.5](#). The grid lengths Δx_c and Δy_c are the lengths between tracer points (cell centers). The grid lengths Δx_g , Δy_g are the grid lengths between cell corners. Δr_f and Δr_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 \mathcal{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:

$$\mathcal{A}_c \partial_t \eta + \delta_i \sum_k \Delta y_g \Delta r_f h_w u + \delta_j \sum_k \Delta x_g \Delta r_f h_s v = \mathcal{A}_c (P - E)_{r=0} \quad (2.82)$$

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:

$$\epsilon_{nh} \partial_t w + g \bar{\rho}'^k + \frac{1}{\Delta z} \delta_k \Phi'_h = \dots \quad (2.83)$$

In the atmosphere, using p-coordinates, hydrostatic balance is discretized:

$$\bar{\theta}'^k + \frac{1}{\Delta \Pi} \delta_k \Phi'_h = 0 \quad (2.84)$$

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 `CALC_PHI_HYD`. Inside this routine, one of other of the atmospheric/oceanic form is selected based on the string variable `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:

$$G_u = G_u^{adv} + G_u^{cor} + G_u^{h-diss} + G_u^{v-diss} + G_u^{metric} + G_u^{nh-metric} \quad (2.85)$$

$$G_v = G_v^{adv} + G_v^{cor} + G_v^{h-diss} + G_v^{v-diss} + G_v^{metric} + G_v^{nh-metric} \quad (2.86)$$

$$G_w = G_w^{adv} + G_w^{cor} + G_w^{h-diss} + G_w^{v-diss} + G_w^{metric} + G_w^{nh-metric} \quad (2.87)$$

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 `MOM_FLUXFORM` and collected into the global arrays `gU`, `gV`, and `gW`.

S/R MOM_FLUXFORM

G_u : `gU` (`DYNVARS.h`)

G_v : `gV` (`DYNVARS.h`)

G_w : `gW` (`NH_VARS.h`)

2.14.1 Advection of momentum

The advective operator is second order accurate in space:

$$\mathcal{A}_w \Delta r_f h_w G_u^{adv} = \delta_i \bar{U}^i \bar{u}^i + \delta_j \bar{V}^j \bar{u}^j + \delta_k \bar{W}^k \bar{u}^k \quad (2.88)$$

$$\mathcal{A}_s \Delta r_f h_s G_v^{adv} = \delta_i \bar{U}^j \bar{v}^i + \delta_j \bar{V}^j \bar{v}^j + \delta_k \bar{W}^j \bar{v}^k \quad (2.89)$$

$$\mathcal{A}_c \Delta r_c G_w^{adv} = \delta_i \bar{U}^k \bar{w}^i + \delta_j \bar{V}^k \bar{w}^j + \delta_k \bar{W}^k \bar{w}^k \quad (2.90)$$

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:

$$U = \Delta y_g \Delta r_f h_w u \quad (2.91)$$

$$V = \Delta x_g \Delta r_f h_s v \quad (2.92)$$

$$W = \mathcal{A}_c w \quad (2.93)$$

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, wv : 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:

$$\mathcal{A}_w \Delta r_f h_w G_u^{Cor} = \overline{f \mathcal{A}_c \Delta r_f h_c \bar{v}^j}^i - \epsilon_{nh} \overline{f' \mathcal{A}_c \Delta r_f h_c \bar{w}^k}^i \quad (2.94)$$

$$\mathcal{A}_s \Delta r_f h_s G_v^{Cor} = -\overline{f \mathcal{A}_c \Delta r_f h_c \bar{u}^i}^j \quad (2.95)$$

$$\mathcal{A}_c \Delta r_c G_w^{Cor} = \epsilon_{nh} \overline{f' \mathcal{A}_c \Delta r_f h_c \bar{u}^i}^k \quad (2.96)$$

where the Coriolis parameters f and f' are defined:

$$\begin{aligned} f &= 2\Omega \sin \varphi \\ f' &= 2\Omega \cos \varphi \end{aligned}$$

where φ is geographic latitude when using spherical geometry, otherwise the β -plane definition is used:

$$\begin{aligned} f &= f_o + \beta y \\ f' &= 0 \end{aligned}$$

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:

$$G_u^{Cor} = f_u \bar{v}^{ji} - \epsilon_{nh} f'_u \bar{w}^{ik} \quad (2.97)$$

$$G_v^{Cor} = -f_v \bar{u}^{ij} \quad (2.98)$$

$$G_w^{Cor} = \epsilon_{nh} f'_w \bar{u}^{ik} \quad (2.99)$$

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 logical `useEnergyConservingCoriolis` to `.TRUE.` which otherwise defaults to `.FALSE.`

S/R CD_CODE_SCHEME, MOM_U_CORIOLIS, MOM_V_CORIOLIS

G_u^{Cor}, G_v^{Cor} : cF (local to `MOM_FLUXFORM.F`)

2.14.3 Curvature metric terms

The most commonly used coordinate system on the sphere is the geographic system (λ, φ) . 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:

$$\mathcal{A}_w \Delta r_f h_w G_u^{metric} = \frac{\bar{u}^i}{a} \tan \varphi \mathcal{A}_c \Delta r_f h_c \bar{v}^j \quad (2.100)$$

$$\mathcal{A}_s \Delta r_f h_s G_v^{metric} = -\frac{\bar{u}^i}{a} \tan \varphi \mathcal{A}_c \Delta r_f h_c \bar{u}^i \quad (2.101)$$

$$G_w^{metric} = 0 \quad (2.102)$$

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 $\frac{u}{a} \tan \varphi$ can be viewed as a modification of the vertical Coriolis parameter: $f \rightarrow f + \frac{u}{a} \tan \varphi$.

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

$$\begin{aligned} G_u^{metric} &= \frac{u \bar{v}^{ij}}{a} \tan \varphi \\ G_v^{metric} &= \frac{\bar{u}^{ij} \bar{u}^{ij}}{a} \tan \varphi \end{aligned}$$

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

S/R MOM_U_METRIC_SPHERE, MOM_V_METRIC_SPHERE

$G_u^{metric}, G_v^{metric}$: mT (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:

$$\mathcal{A}_w \Delta r_f h_w G_u^{metric} = - \frac{\overline{\bar{u}^i \bar{w}^k}}{a} \mathcal{A}_c \Delta r_f h_c \quad (2.103)$$

$$\mathcal{A}_s \Delta r_f h_s G_v^{metric} = - \frac{\overline{\bar{v}^j \bar{w}^k}}{a} \mathcal{A}_c \Delta r_f h_c \quad (2.104)$$

$$\mathcal{A}_c \Delta r_c G_w^{metric} = \frac{\overline{\bar{u}^{i2} + \bar{v}^{j2}}}{a} \mathcal{A}_c \Delta r_f h_c \quad (2.105)$$

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

$$\begin{aligned} G_u^{metric} &= -\frac{u}{a} \bar{w}^{ik} \\ G_v^{metric} &= -\frac{v}{a} \bar{w}^{jk} \\ G_w^{metric} &= \frac{1}{a} (\bar{u}^{ik2} + \bar{v}^{jk2}) \end{aligned}$$

S/R MOM_U_METRIC_NH, MOM_V_METRIC_NH

$G_u^{metric}, G_v^{metric}$: mT (local to MOM_FLUXFORM.F)

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.

$$A_w \Delta r_f h_w G_u^{h-diss} = \delta_i \Delta y_f \Delta r_f h_c \tau_{11} + \delta_j \Delta x_v \Delta r_f h_c \tau_{12} \quad (2.106)$$

$$A_s \Delta r_f h_s G_v^{h-diss} = \delta_i \Delta y_u \Delta r_f h_c \tau_{21} + \delta_j \Delta x_f \Delta r_f h_c \tau_{22} \quad (2.107)$$

The lateral viscous stresses are discretized:

$$\tau_{11} = A_h c_{11\Delta}(\varphi) \frac{1}{\Delta x_f} \delta_i u - A_4 c_{11\Delta^2}(\varphi) \frac{1}{\Delta x_f} \delta_i \nabla^2 u \quad (2.108)$$

$$\tau_{12} = A_h c_{12\Delta}(\varphi) \frac{1}{\Delta y_u} \delta_j u - A_4 c_{12\Delta^2}(\varphi) \frac{1}{\Delta y_u} \delta_j \nabla^2 u \quad (2.109)$$

$$\tau_{21} = A_h c_{21\Delta}(\varphi) \frac{1}{\Delta x_v} \delta_i v - A_4 c_{21\Delta^2}(\varphi) \frac{1}{\Delta x_v} \delta_i \nabla^2 v \quad (2.110)$$

$$\tau_{22} = A_h c_{22\Delta}(\varphi) \frac{1}{\Delta y_f} \delta_j v - A_4 c_{22\Delta^2}(\varphi) \frac{1}{\Delta y_f} \delta_j \nabla^2 v \quad (2.111)$$

where the non-dimensional factors $c_{lm\Delta^n}(\varphi)$, $\{l, m, n\} \in \{1, 2\}$ 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 $m^4 s^{-1}$.

S/R MOM_U_XVISCFLUX, MOM_U_YVISCFLUX

τ_{11}, τ_{12} : **vF**, **v4F** (local to **MOM_FLUXFORM.F**)

S/R MOM_V_XVISCFLUX, MOM_V_YVISCFLUX

τ_{21}, τ_{22} : **vF**, **v4F** (local to **MOM_FLUXFORM.F**)

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:

$$G_u^{side-drag} = \frac{4}{\Delta z_f} \overline{(1 - h_\zeta) \frac{\Delta x_v}{\Delta y_u}}^j (A_h c_{12\Delta}(\varphi)u - A_4 c_{12\Delta^2}(\varphi)\nabla^2 u) \quad (2.112)$$

$$G_v^{side-drag} = \frac{4}{\Delta z_f} \overline{(1 - h_\zeta) \frac{\Delta y_u}{\Delta x_v}}^i (A_h c_{21\Delta}(\varphi)v - A_4 c_{21\Delta^2}(\varphi)\nabla^2 v) \quad (2.113)$$

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. $1 - h_w < 1 - h_\zeta$

S/R MOM_U_SIDEDRAG, MOM_V_SIDEDRAG

$G_u^{side-drag}, G_v^{side-drag}$: vF (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.

$$G_u^{v-diss} = \frac{1}{\Delta r_f h_w} \delta_k \tau_{13} \quad (2.114)$$

$$G_v^{v-diss} = \frac{1}{\Delta r_f h_s} \delta_k \tau_{23} \quad (2.115)$$

$$G_w^{v-diss} = \epsilon_{nh} \frac{1}{\Delta r_f h_d} \delta_k \tau_{33} \quad (2.116)$$

represents the general discrete form of the vertical dissipation terms.

In the interior the vertical stresses are discretized:

$$\begin{aligned} \tau_{13} &= A_v \frac{1}{\Delta r_c} \delta_k u \\ \tau_{23} &= A_v \frac{1}{\Delta r_c} \delta_k v \\ \tau_{33} &= A_v \frac{1}{\Delta r_f} \delta_k w \end{aligned}$$

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.

S/R MOM_U_RVISCFLUX, MOM_V_RVISCFLUX

τ_{13} : fVrUp, fVrDw (local to MOM_FLUXFORM.F)

τ_{23} : fVrUp, fVrDw (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:

$$\tau_{13}^{bottom-drag} = \left(2A_v \frac{1}{\Delta r_c} + r_b + C_d \sqrt{2KE^i} \right) u \quad (2.117)$$

$$\tau_{23}^{bottom-drag} = \left(2A_v \frac{1}{\Delta r_c} + r_b + C_d \sqrt{2KE^j} \right) v \quad (2.118)$$

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

S/R MOM_U_BOTTOMDRAG, MOM_V_BOTTOMDRAG

$\tau_{13}^{bottom-drag} / \Delta r_f, \tau_{23}^{bottom-drag} / \Delta r_f : \text{vF}$ (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:

$$KE = \frac{1}{2} \left(\overline{u^2}^i + \overline{v^2}^j + \epsilon_{nh} \overline{w^2}^k \right) \quad (2.119)$$

2.14.8 Mom Diagnostics

```
-----
<-Name->|Levs|<-parsing code->|<--  Units    -->|<- Tile (max=80c)
-----
VISCALHZ | 15 |SZ      MR      |m^2/s      |Harmonic Visc Coefficient (m2/s)┐
↪ (Zeta Pt)
VISCAL4Z | 15 |SZ      MR      |m^4/s      |Biharmonic Visc Coefficient (m4/s)┐
↪ (Zeta Pt)
VISCALHD | 15 |SM      MR      |m^2/s      |Harmonic Viscosity Coefficient (m2/s)┐
↪ (Div Pt)
VISCAL4D | 15 |SM      MR      |m^4/s      |Biharmonic Viscosity Coefficient (m4/
↪ s) (Div Pt)
VAHZMAX  | 15 |SZ      MR      |m^2/s      |CFL-MAX Harm Visc Coefficient (m2/s)┐
↪ (Zeta Pt)
VA4ZMAX  | 15 |SZ      MR      |m^4/s      |CFL-MAX Biharm Visc Coefficient (m4/
↪ s) (Zeta Pt)
VAHDMAX  | 15 |SM      MR      |m^2/s      |CFL-MAX Harm Visc Coefficient (m2/s)┐
↪ (Div Pt)
VA4DMAX  | 15 |SM      MR      |m^4/s      |CFL-MAX Biharm Visc Coefficient (m4/
↪ s) (Div Pt)
```

(continues on next page)

(continued from previous page)

VAHZMIN 15 SZ	MR	m ² /s	RE-MIN Harm Visc Coefficient (m2/s)
↪ (Zeta Pt)			
VA4ZMIN 15 SZ	MR	m ⁴ /s	RE-MIN Biharm Visc Coefficient (m4/s)
↪ (Zeta Pt)			
VAHDMIN 15 SM	MR	m ² /s	RE-MIN Harm Visc Coefficient (m2/s)
↪ (Div Pt)			
VA4DMIN 15 SM	MR	m ⁴ /s	RE-MIN Biharm Visc Coefficient (m4/s)
↪ (Div Pt)			
VAHZLTH 15 SZ	MR	m ² /s	Leith Harm Visc Coefficient (m2/s)
↪ (Zeta Pt)			
VA4ZLTH 15 SZ	MR	m ⁴ /s	Leith Biharm Visc Coefficient (m4/s)
↪ (Zeta Pt)			
VAHDLTH 15 SM	MR	m ² /s	Leith Harm Visc Coefficient (m2/s)
↪ (Div Pt)			
VA4DLTH 15 SM	MR	m ⁴ /s	Leith Biharm Visc Coefficient (m4/s)
↪ (Div Pt)			
VAHZLTHD 15 SZ	MR	m ² /s	LeithD Harm Visc Coefficient (m2/s)
↪ (Zeta Pt)			
VA4ZLTHD 15 SZ	MR	m ⁴ /s	LeithD Biharm Visc Coefficient (m4/s)
↪ (Zeta Pt)			
VAHDLTHD 15 SM	MR	m ² /s	LeithD Harm Visc Coefficient (m2/s)
↪ (Div Pt)			
VA4DLTHD 15 SM	MR	m ⁴ /s	LeithD Biharm Visc Coefficient (m4/s)
↪ (Div Pt)			
VAHZSMAG 15 SZ	MR	m ² /s	Smagorinsky Harm Visc Coefficient (m2/
↪ s) (Zeta Pt)			
VA4ZSMAG 15 SZ	MR	m ⁴ /s	Smagorinsky Biharm Visc Coeff. (m4/s)
↪ (Zeta Pt)			
VAHDSMAG 15 SM	MR	m ² /s	Smagorinsky Harm Visc Coefficient (m2/
↪ s) (Div Pt)			
VA4DSMAG 15 SM	MR	m ⁴ /s	Smagorinsky Biharm Visc Coeff. (m4/s)
↪ (Div Pt)			
momKE 15 SM	MR	m ² /s ²	Kinetic Energy (in momentum Eq.)
momHDiv 15 SM	MR	s ⁻¹	Horizontal Divergence (in momentum Eq.
↪)			
momVort3 15 SZ	MR	s ⁻¹	3rd component (vertical) of Vorticity
Strain 15 SZ	MR	s ⁻¹	Horizontal Strain of Horizontal
↪ Velocities			
Tension 15 SM	MR	s ⁻¹	Horizontal Tension of Horizontal
↪ Velocities			
UBotDrag 15 UU	129MR	m/s ²	U momentum tendency from Bottom Drag
VBotDrag 15 VV	128MR	m/s ²	V momentum tendency from Bottom Drag
USidDrag 15 UU	131MR	m/s ²	U momentum tendency from Side Drag
VSidDrag 15 VV	130MR	m/s ²	V momentum tendency from Side Drag
Um_Diss 15 UU	133MR	m/s ²	U momentum tendency from Dissipation
Vm_Diss 15 VV	132MR	m/s ²	V momentum tendency from Dissipation
Um_Advec 15 UU	135MR	m/s ²	U momentum tendency from Advection
↪ terms			
Vm_Advec 15 VV	134MR	m/s ²	V momentum tendency from Advection
↪ terms			
Um_Cori 15 UU	137MR	m/s ²	U momentum tendency from Coriolis term
Vm_Cori 15 VV	136MR	m/s ²	V momentum tendency from Coriolis term
Um_Ext 15 UU	137MR	m/s ²	U momentum tendency from external
↪ forcing			
Vm_Ext 15 VV	138MR	m/s ²	V momentum tendency from external
↪ forcing			
Um_AdvZ3 15 UU	141MR	m/s ²	U momentum tendency from Vorticity
↪ Advection			

(continues on next page)

(continued from previous page)

Vm_AdvZ3	15	VV	140MR	m/s^2	V momentum tendency from Vorticity_
↪Advection					
Um_AdvRe	15	UU	143MR	m/s^2	U momentum tendency from vertical_
↪Advection (Explicit part)					
Vm_AdvRe	15	VV	142MR	m/s^2	V momentum tendency from vertical_
↪Advection (Explicit part)					
ADVx_Um	15	UM	145MR	m^4/s^2	Zonal Advective Flux of U_
↪momentum					
ADVy_Um	15	VZ	144MR	m^4/s^2	Meridional Advective Flux of U_
↪momentum					
ADVrE_Um	15	WU	LR	m^4/s^2	Vertical Advective Flux of U_
↪momentum (Explicit part)					
ADVx_Vm	15	UZ	148MR	m^4/s^2	Zonal Advective Flux of V_
↪momentum					
ADVy_Vm	15	VM	147MR	m^4/s^2	Meridional Advective Flux of V_
↪momentum					
ADVrE_Vm	15	WV	LR	m^4/s^2	Vertical Advective Flux of V_
↪momentum (Explicit part)					
VISCx_Um	15	UM	151MR	m^4/s^2	Zonal Viscous Flux of U momentum
VISCy_Um	15	VZ	150MR	m^4/s^2	Meridional Viscous Flux of U momentum
VISrE_Um	15	WU	LR	m^4/s^2	Vertical Viscous Flux of U momentum_
↪(Explicit part)					
VISrI_Um	15	WU	LR	m^4/s^2	Vertical Viscous Flux of U momentum_
↪(Implicit part)					
VISCx_Vm	15	UZ	155MR	m^4/s^2	Zonal Viscous Flux of V momentum
VISCy_Vm	15	VM	154MR	m^4/s^2	Meridional Viscous Flux of V momentum
VISrE_Vm	15	WV	LR	m^4/s^2	Vertical Viscous Flux of V momentum_
↪(Explicit part)					
VISrI_Vm	15	WV	LR	m^4/s^2	Vertical Viscous Flux of V momentum_
↪(Implicit part)					

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:

$$\partial_t \vec{v} + (2\vec{\Omega} + \vec{\zeta}) \wedge \vec{v} - b\hat{r} + \vec{\nabla} B = \vec{\nabla} \cdot \vec{\tau} \quad (2.120)$$

which describe motions in any orthogonal curvilinear coordinate system. Here, B is the Bernoulli function and $\vec{\zeta} = \vec{\nabla} \wedge \vec{v}$ 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:

$$G_u = G_u^{fv} + G_u^{\zeta_3 v} + G_u^{\zeta_2 w} + G_u^{\partial_x B} + G_u^{\partial_z \tau^x} + G_u^{h-dissip} + G_u^{v-dissip} \quad (2.121)$$

$$G_v = G_v^{fu} + G_v^{\zeta_3 u} + G_v^{\zeta_1 w} + G_v^{\partial_y B} + G_v^{\partial_z \tau^y} + G_v^{h-dissip} + G_v^{v-dissip} \quad (2.122)$$

$$G_w = G_w^{fu} + G_w^{\zeta_1 v} + G_w^{\zeta_2 u} + G_w^{\partial_z B} + G_w^{h-dissip} + G_w^{v-dissip} \quad (2.123)$$

S/R MOM_VECINV

G_u : gU (DYNVARS.h)

G_v : gV (DYNVARS.h)

G_w : gW (NH_VARS.h)

2.15.1 Relative vorticity

The vertical component of relative vorticity is explicitly calculated and use 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:

$$\zeta_3 = \frac{\Gamma}{\mathcal{A}_\zeta} = \frac{1}{\mathcal{A}_\zeta} (\delta_i \Delta y_c v - \delta_j \Delta x_c u) \quad (2.124)$$

where \mathcal{A}_ζ is the area of the vorticity cell presented in the vertical and Γ is the circulation about that cell.

S/R MOM_CALC_RELVORT3

ζ_3 : vort3 (local to MOM_VECINV.F)

2.15.2 Kinetic energy

The kinetic energy, denoted KE , is defined:

$$KE = \frac{1}{2} (\overline{u^2}^i + \overline{v^2}^j + \epsilon_{nh} \overline{w^2}^k) \quad (2.125)$$

S/R MOM_CALC_KE

KE : KE (local to MOM_VECINV.F)

2.15.3 Coriolis terms

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

$$G_u^{fv} = \frac{1}{\Delta x_c} \frac{\overline{f}^j}{h_\zeta} \overline{\Delta x_g h_s v}^i \quad (2.126)$$

$$G_v^{fu} = -\frac{1}{\Delta y_c} \frac{\overline{f}^i}{h_\zeta} \overline{\Delta y_g h_w u}^j \quad (2.127)$$

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

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

$$G_u^{\zeta_3 v} = \frac{1}{\Delta x_c} \frac{\overline{\zeta_3}^j}{h_\zeta} \overline{\Delta x_g h_s v}^i \quad (2.128)$$

$$G_v^{\zeta_3 u} = -\frac{1}{\Delta y_c} \frac{\overline{\zeta_3}^i}{h_\zeta} \overline{\Delta y_g h_w u}^j \quad (2.129)$$

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:

$$G_u^{fv} + G_u^{\zeta_3 v} = \frac{1}{\Delta x_c} \frac{\overline{f + \zeta_3}^j}{h_\zeta} \overline{\Delta x_g h_s v}^i \quad (2.130)$$

$$G_v^{fu} + G_v^{\zeta_3 u} = -\frac{1}{\Delta y_c} \frac{\overline{f + \zeta_3}^i}{h_\zeta} \overline{\Delta y_g h_w u}^j \quad (2.131)$$

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

$G_u^{fv}, G_u^{\zeta_3 v}$: uCf (local to MOM_VECINV.F)

$G_v^{fu}, G_v^{\zeta_3 u}$: vCf (local to MOM_VECINV.F)

2.15.4 Shear terms

The shear terms ($\zeta_2 w$ and $\zeta_1 w$) 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:

$$G_u^{\zeta_2 w} = \frac{1}{\mathcal{A}_w \Delta r_f h_w} \overline{\mathcal{A}_c w}^i (\delta_k u - \epsilon_{nh} \delta_j w)^k \quad (2.132)$$

$$G_v^{\zeta_1 w} = \frac{1}{\mathcal{A}_s \Delta r_f h_s} \overline{\mathcal{A}_c w^i (\delta_k u - \epsilon_{nh} \delta_j w)}^k \quad (2.133)$$

S/R MOM_VI_U_VERTSHEAR, MOM_VI_V_VERTSHEAR

$G_u^{\zeta_2 w}$: uCf (local to MOM_VECINV.F)

$G_v^{\zeta_1 w}$: vCf (local to MOM_VECINV.F)

2.15.5 Gradient of Bernoulli function

$$G_u^{\partial_x B} = \frac{1}{\Delta x_c} \delta_i (\phi' + KE) \quad (2.134)$$

$$G_v^{\partial_y B} = \frac{1}{\Delta x_y} \delta_j (\phi' + KE) \quad (2.135)$$

S/R MOM_VI_U_GRAD_KE, MOM_VI_V_GRAD_KE

$G_u^{\partial_x KE}$: uCf (local to MOM_VECINV.F)

$G_v^{\partial_y KE}$: 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:

$$D = \frac{1}{\mathcal{A}_c h_c} (\delta_i \Delta y_g h_w u + \delta_j \Delta x_g h_s v) \quad (2.136)$$

S/R MOM_CALC_KE

D : hDiv (local to MOM_VECINV.F)

2.15.7 Horizontal dissipation

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

$$G_u^{h-dissip} = \frac{1}{\Delta x_c} \delta_i (A_D D - A_{D4} D^*) - \frac{1}{\Delta y_u h_w} \delta_j h_\zeta (A_\zeta \zeta - A_{\zeta4} \zeta^*) \quad (2.137)$$

$$G_v^{h-dissip} = \frac{1}{\Delta x_v h_s} \delta_i h_\zeta (A_\zeta \zeta - A_{\zeta4} \zeta^*) + \frac{1}{\Delta y_c} \delta_j (A_D D - A_{D4} D^*) \quad (2.138)$$

where

$$D^* = \frac{1}{\mathcal{A}_c h_c} (\delta_i \Delta y_g h_w \nabla^2 u + \delta_j \Delta x_g h_s \nabla^2 v)$$

$$\zeta^* = \frac{1}{\mathcal{A}_\zeta} (\delta_i \Delta y_c \nabla^2 v - \delta_j \Delta x_c \nabla^2 u)$$

S/R MOM_VI_HDISSIP

$G_u^{h-dissip}$: uDissip (local to MOM_VI_HDISSIP.F)

$G_v^{h-dissip}$: vDissip (local to MOM_VI_HDISSIP.F)

2.15.8 Vertical dissipation

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

$$G_u^{v-diss} = \frac{1}{\Delta r_f h_w} \delta_k \tau_{13} \quad (2.139)$$

$$G_v^{v-diss} = \frac{1}{\Delta r_f h_s} \delta_k \tau_{23} \quad (2.140)$$

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

τ_{13}, τ_{23} : vrf (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, τ , the forced advection-diffusion equation reads:

$$\partial_t \tau + G_{adv}^\tau = G_{diff}^\tau + G_{forc}^\tau \quad (2.141)$$

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

$$G_{adv}^\tau = \partial_x u \tau + \partial_y v \tau + \partial_r w \tau - \tau \nabla \cdot \mathbf{v} \quad (2.142)$$

$$G_{diff}^\tau = \nabla \cdot \mathbf{K} \nabla \tau \quad (2.143)$$

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

τ : `tau` (argument)
 $G^{(n)}$: `gTracer` (argument)
 F_r : `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:

$$G^{(n+1/2)} = \left(\frac{3}{2} + \epsilon\right) G^{(n)} - \left(\frac{1}{2} + \epsilon\right) G^{(n-1)} \quad (2.144)$$

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

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

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

$$\tau^{(n+1)} = \tau^{(n)} + \Delta t G^{(n+1/2)} \quad (2.145)$$

S/R TIMESTEP_TRACER

$\tau^{(n+1)}$: gTracer (argument on exit)
 $\tau^{(n)}$: tracer (argument on entry)
 $G^{(n+1/2)}$: gTracer (argument)
 Δt : 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 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 it self. We discuss these first since they are most commonly used in the field and most familiar.

2.17.1 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:

$$\mathcal{A}_c \Delta r_f h_c G_{adv}^T = \delta_i F_x + \delta_j F_y + \delta_k F_r \quad (2.146)$$

where the area integrated fluxes are given by:

$$\begin{aligned} F_x &= U \bar{\tau}^i \\ F_y &= V \bar{\tau}^j \\ F_r &= W \bar{\tau}^k \end{aligned}$$

The quantities U , V and W are volume fluxes. defined as:

$$\begin{aligned} U &= \Delta y_g \Delta r_f h_w u \\ V &= \Delta x_g \Delta r_f h_s v \\ W &= \mathcal{A}_c w \end{aligned}$$

For non-divergent flow, this discretization can be shown to conserve the tracer both locally and globally and to globally conserve tracer variance, τ^2 . The proof is given in Adcroft (1995) [Adc95] and Adcroft et al. (1997) [AHM97] .

S/R GAD_C2_ADV_X

F_x : uT (argument)
 U : uTrans (argument)
 τ : tracer (argument)

S/R GAD_C2_ADV_Y

F_y : **vT** (argument)
 V : **vTrans** (argument)
 τ : **tracer** (argument)

S/R GAD_C2_ADV_R

F_r : **wT** (argument)
 W : **rTrans** (argument)
 τ : **tracer** (argument)

2.17.2 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:

$$\begin{aligned} F_x &= \overline{U\tau - \frac{1}{6}\delta_{ii}\tau}^i + \frac{1}{2}|U|\delta_i\frac{1}{6}\delta_{ii}\tau \\ F_y &= \overline{V\tau - \frac{1}{6}\delta_{ii}\tau}^j + \frac{1}{2}|V|\delta_j\frac{1}{6}\delta_{jj}\tau \\ F_r &= \overline{W\tau - \frac{1}{6}\delta_{ii}\tau}^k + \frac{1}{2}|W|\delta_k\frac{1}{6}\delta_{kk}\tau \end{aligned}$$

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

S/R GAD_U3_ADV_X

F_x : **uT** (argument)
 U : **uTrans** (argument)
 τ : **tracer** (argument)

S/R GAD_U3_ADV_Y

F_y : **vT** (argument)
 V : **vTrans** (argument)
 τ : **tracer** (argument)

S/R GAD_U3_ADV_R

F_r : **wT** (argument)
 W : **rTrans** (argument)

τ : `tracer` (argument)

2.17.3 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:

$$\begin{aligned} F_x &= \overline{U\tau - \frac{1}{6}\delta_{ii}\tau}^i \\ F_y &= \overline{V\tau - \frac{1}{6}\delta_{ii}\tau}^j \\ F_r &= \overline{W\tau - \frac{1}{6}\delta_{ii}\tau}^k \end{aligned}$$

As for the third order scheme, the best discretization near boundaries is under investigation but currently $\delta_i\tau = 0$ on a boundary.

S/R GAD_C4_ADV_X

F_x : `uT` (argument)
 U : `uTrans` (argument)
 τ : `tracer` (argument)

S/R GAD_C4_ADV_Y

F_y : `vT` (argument)
 V : `vTrans` (argument)
 τ : `tracer` (argument)

S/R GAD_C4_ADV_R

F_r : `wT` (argument)
 W : `rTrans` (argument)
 τ : `tracer` (argument)

2.17.4 First order upwind advection

Although the upwind scheme is the underlying scheme for the robust or non-linear methods given in [Section 2.18](#), 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:

$$\begin{aligned} F_x &= U\bar{\tau}^i - \frac{1}{2}|U|\delta_i\tau \\ F_y &= V\bar{\tau}^j - \frac{1}{2}|V|\delta_j\tau \\ F_r &= W\bar{\tau}^k - \frac{1}{2}|W|\delta_k\tau \end{aligned}$$

If for some reason the above method is desired, the second order flux limiter scheme described in [Section 2.18.1](#) reduces to the above scheme if the limiter is set to zero.

2.18 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.

2.18.1 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:

$$F = F_1 + \psi(r)F_{LW} \quad (2.147)$$

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

$$F_1 = u\bar{\tau}^i - \frac{1}{2}|u|\delta_i\tau$$

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

$$\begin{aligned} r &= \frac{\tau_{i-1} - \tau_{i-2}}{\tau_i - \tau_{i-1}} \quad \forall \quad u > 0 \\ r &= \frac{\tau_{i+1} - \tau_i}{\tau_i - \tau_{i-1}} \quad \forall \quad u < 0 \end{aligned}$$

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

$$\psi(r) = \max[0, \min[1, 2r], \min[2, r]]$$

F_x : `uT` (argument)
 U : `uTrans` (argument)
 τ : `tracer` (argument)

S/R GAD_FLUXLIMIT_ADV_Y

F_y : `vT` (argument)
 V : `vTrans` (argument)
 τ : `tracer` (argument)

S/R GAD_FLUXLIMIT_ADV_R

F_r : `wT` (argument)
 W : `rTrans` (argument)
 τ : `tracer` (argument)

2.18.2 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:

$$\begin{aligned}
 F &= u (\tau_{i-1} + d_0(\tau_i - \tau_{i-1}) + d_1(\tau_{i-1} - \tau_{i-2})) \quad \forall \quad u > 0 \\
 F &= u (\tau_i - d_0(\tau_i - \tau_{i-1}) - d_1(\tau_{i+1} - \tau_i)) \quad \forall \quad u < 0
 \end{aligned}
 \tag{2.148}$$

where

$$\begin{aligned}
 d_1 &= \frac{1}{6}(2 - |c|)(1 - |c|) \\
 d_2 &= \frac{1}{6}(1 - |c|)(1 + |c|)
 \end{aligned}$$

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 2.10](#)) 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 2.11](#)) with only minor overshoots.

S/R GAD_DST3_ADV_X

F_x : `uT` (argument)
 U : `uTrans` (argument)
 τ : `tracer` (argument)

S/R GAD_DST3_ADV_Y

F_y : **vT** (argument)
 V : **vTrans** (argument)
 τ : **tracer** (argument)

S/R GAD_DST3_ADV_R

F_r : **wT** (argument)
 W : **rTrans** (argument)
 τ : **tracer** (argument)

2.18.3 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:

$$F = \frac{1}{2}(u + |u|) (\tau_{i-1} + \psi(r^+)(\tau_i - \tau_{i-1})) + \frac{1}{2}(u - |u|) (\tau_{i-1} + \psi(r^-)(\tau_i - \tau_{i-1})) \quad (2.149)$$

where

$$\begin{aligned}
 r^+ &= \frac{\tau_{i-1} - \tau_{i-2}}{\tau_i - \tau_{i-1}} \\
 r^- &= \frac{\tau_{i+1} - \tau_i}{\tau_i - \tau_{i-1}}
 \end{aligned}$$

and the limiter is the Sweby limiter:

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

S/R GAD_DST3FL_ADV_X

F_x : **uT** (argument)
 U : **uTrans** (argument)
 τ : **tracer** (argument)

S/R GAD_DST3FL_ADV_Y

F_y : **vT** (argument)
 V : **vTrans** (argument)
 τ : **tracer** (argument)

S/R GAD_DST3FL_ADV_R

F_r : **wT** (argument)

W : `rTrans` (argument) τ : `tracer` (argument)

2.18.4 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:

$$\begin{aligned}
 \tau^{n+1/3} &= \tau^n - \Delta t \left(\frac{1}{\Delta x} \delta_i F^x(\tau^n) + \tau^n \frac{1}{\Delta x} \delta_i u \right) \\
 \tau^{n+2/3} &= \tau^{n+1/3} - \Delta t \left(\frac{1}{\Delta y} \delta_j F^y(\tau^{n+1/3}) + \tau^{n+1/3} \frac{1}{\Delta y} \delta_j v \right) \\
 \tau^{n+3/3} &= \tau^{n+2/3} - \Delta t \left(\frac{1}{\Delta r} \delta_k F^r(\tau^{n+2/3}) + \tau^{n+2/3} \frac{1}{\Delta r} \delta_k w \right)
 \end{aligned} \tag{2.150}$$

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 likes:

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

S/R GAD_ADECTION

 τ : `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 2.9](#) .

2.19 Comparison of advection schemes

[Table 2.2](#) shows a summary of the different advection schemes available in MITgcm. “A.B.” 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., `tempAdvScheme=3` in file `data` selects the 3rd order upwind advection scheme for temperature).

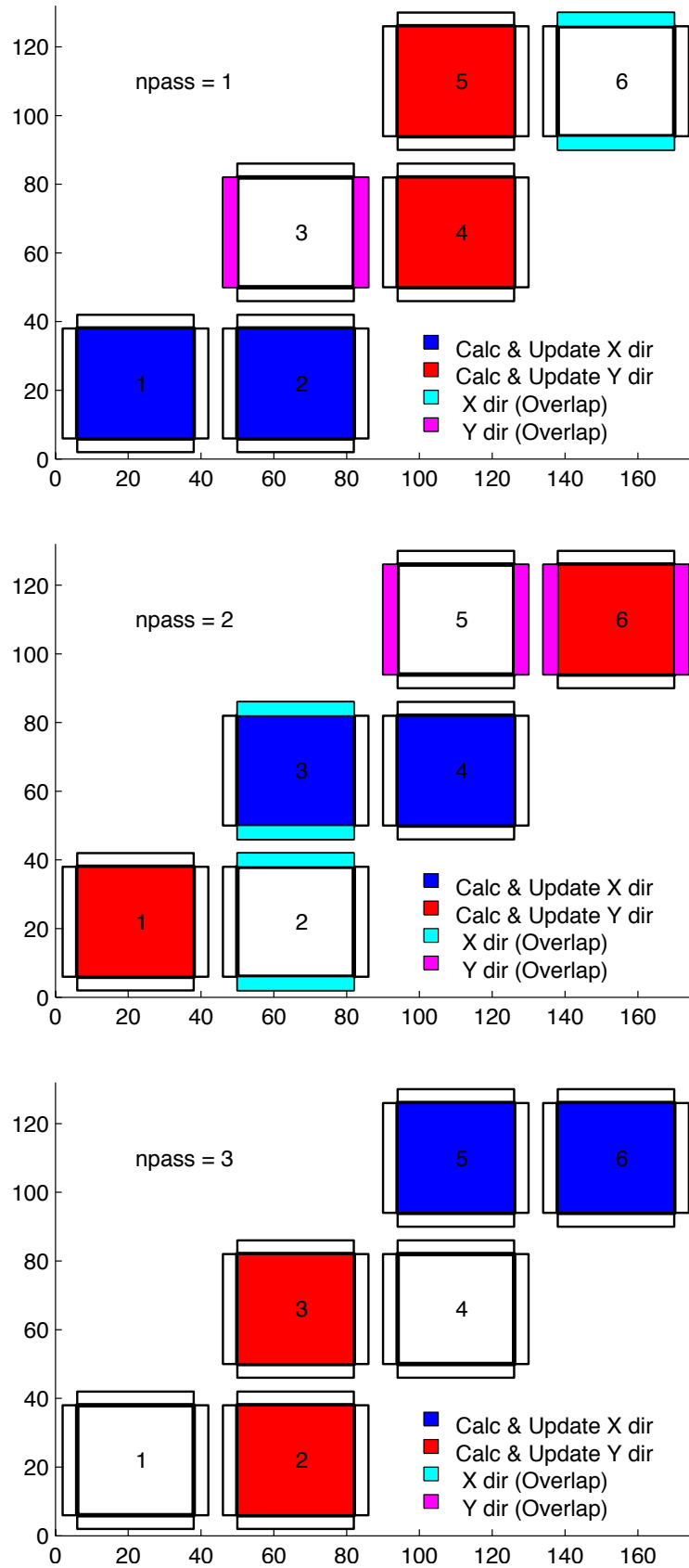


Figure 2.9: Multi-dimensional advection time-stepping with cubed-sphere topology.

Table 2.2: MITgcm Advection Schemes

			use		
		use	multi	stencil	
Advection Scheme	Code	AB?	-dim?	(1-D)	comments
1st order upwind	1	no	yes	3	linear τ , non-linear v
centered 2nd order	2	yes	no	3	linear
3rd order upwind	3	yes	no	5	linear τ
centered 4th order	4	yes	no	5	linear
2nd order DST (Lax-Wendroff)	20	no	yes	3	linear τ , non-linear v
3rd order DST	30	no	yes	5	linear τ , non-linear v
2nd order-moment Prather	80	no	yes		
2nd order flux limiters	77	no	yes	5	non-linear
3rd order DST flux limiter	33	no	yes	5	non-linear
2nd order-moment Prather w/limiter	81	no	yes		
piecewise parabolic w/“null” limiter	40	no	yes		
piecewise parabolic w/“mono” limiter	41	no	yes		
piecewise quartic w/“null” limiter	50	no	yes		
piecewise quartic w/“mono” limiter	51	no	yes		
piecewise quartic w/“weno” limiter	52	no	yes		
7th order one-step method w/	7	no	yes		
monotonicity preserving limiter					

Shown in [Figure 2.10](#) and [Figure 2.11](#) is a 1-D comparison of advection schemes. Here we advect both a smooth hill and a hill with a more abrupt shock. [Figure 2.10](#) shows the result for a weak flow (low Courant number) whereas [Figure 2.11](#) shows the result for a stronger flow (high Courant number).

[Figure 2.12](#), [Figure 2.13](#) and [Figure 2.14](#) 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. These 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 2.10](#) and [Figure 2.11](#) 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 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.

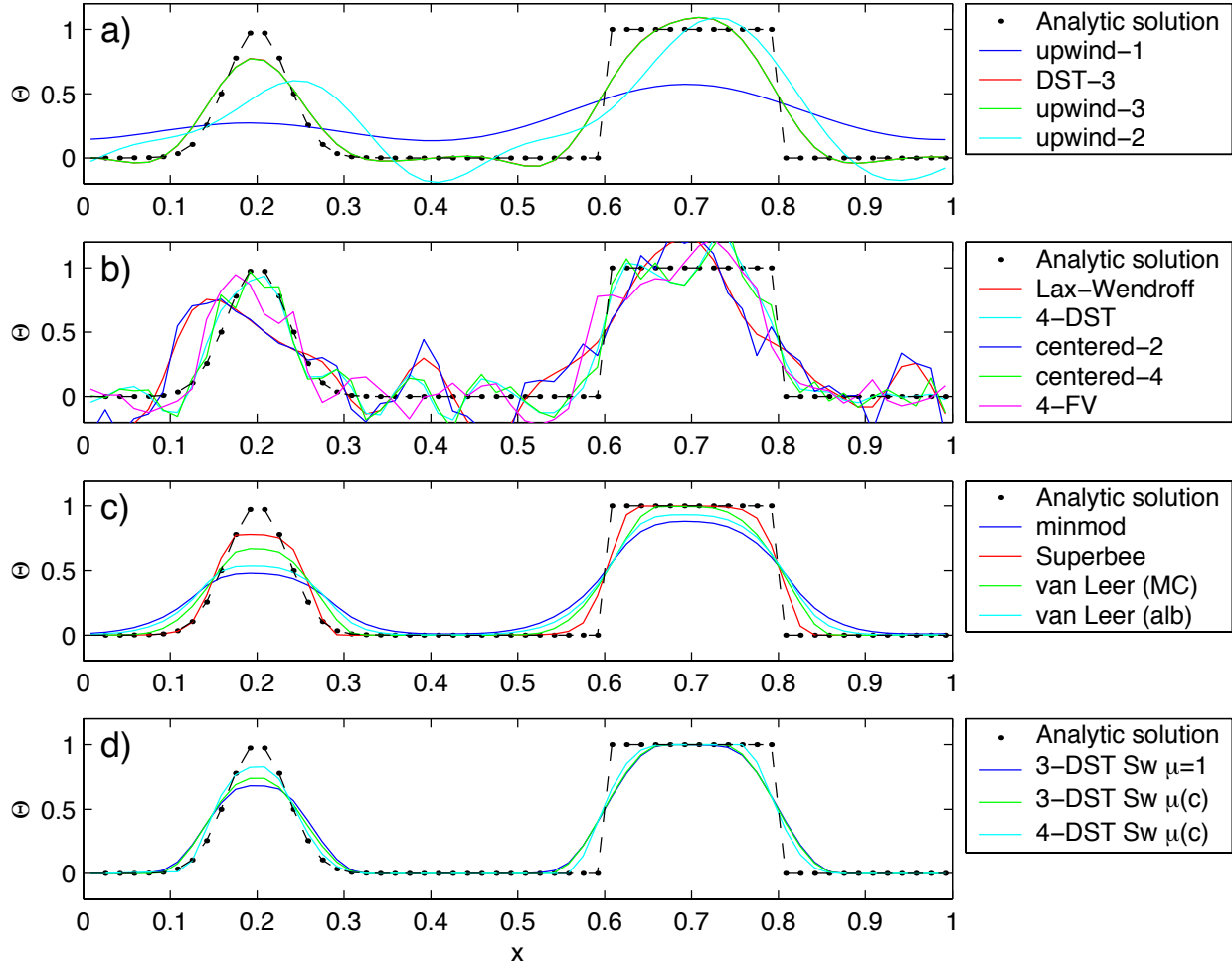


Figure 2.10: Comparison of 1-D advection schemes: Courant number is 0.05 with 60 points and solutions are shown for $T=1$ (one complete period). a) Shows the upwind biased schemes; first order upwind, DST3, third order upwind and second order upwind. b) Shows the centered schemes; Lax-Wendroff, DST4, 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, $\mu = c/(1 - c)$.

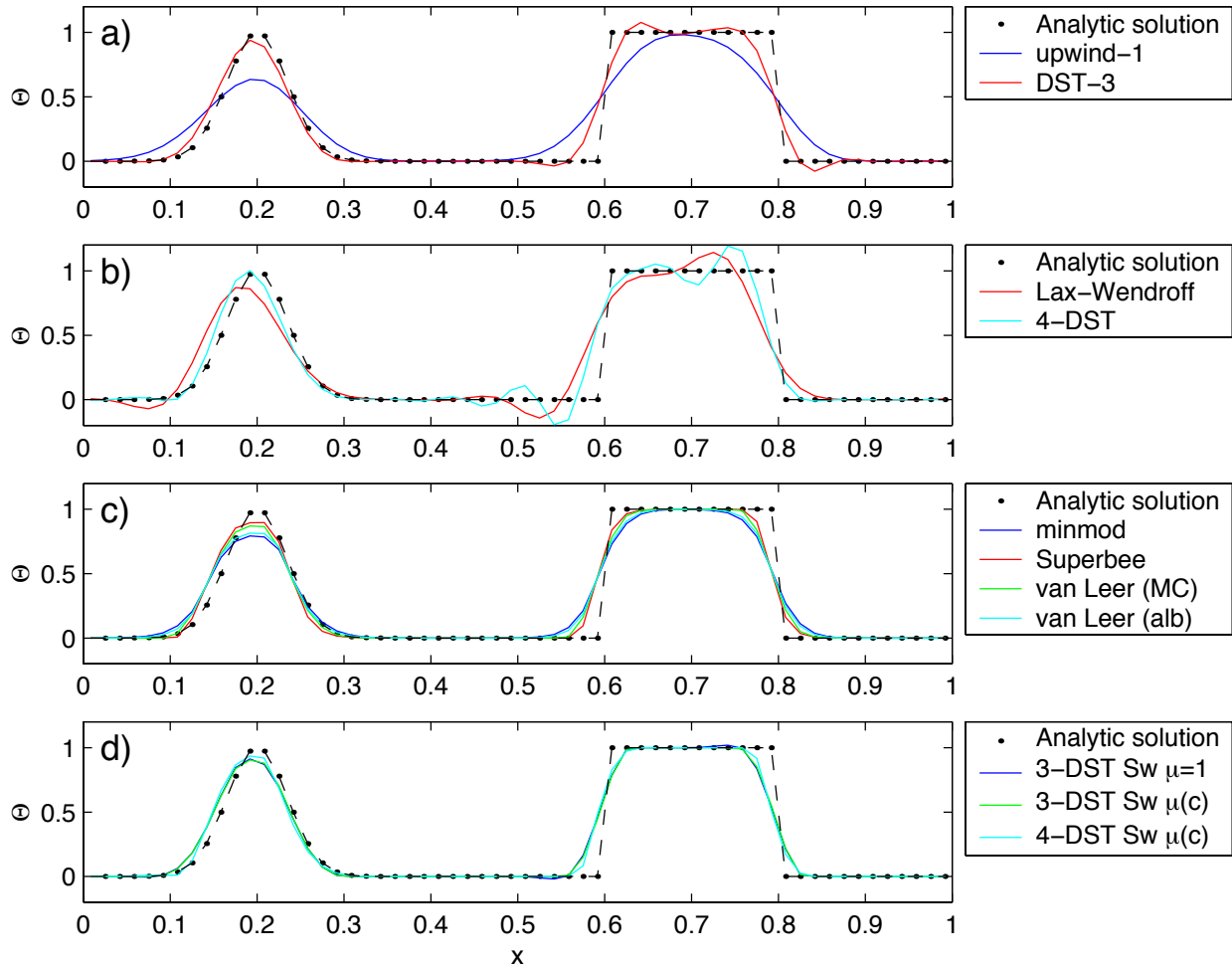


Figure 2.11: Comparison of 1-D advection schemes: Courant number is 0.89 with 60 points and solutions are shown for $T=1$ (one complete period). a) Shows the upwind biased schemes; first order upwind and DST3. Third order upwind and second order upwind are unstable at this Courant number. b) Shows the 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, $\mu = c/(1 - c)$.

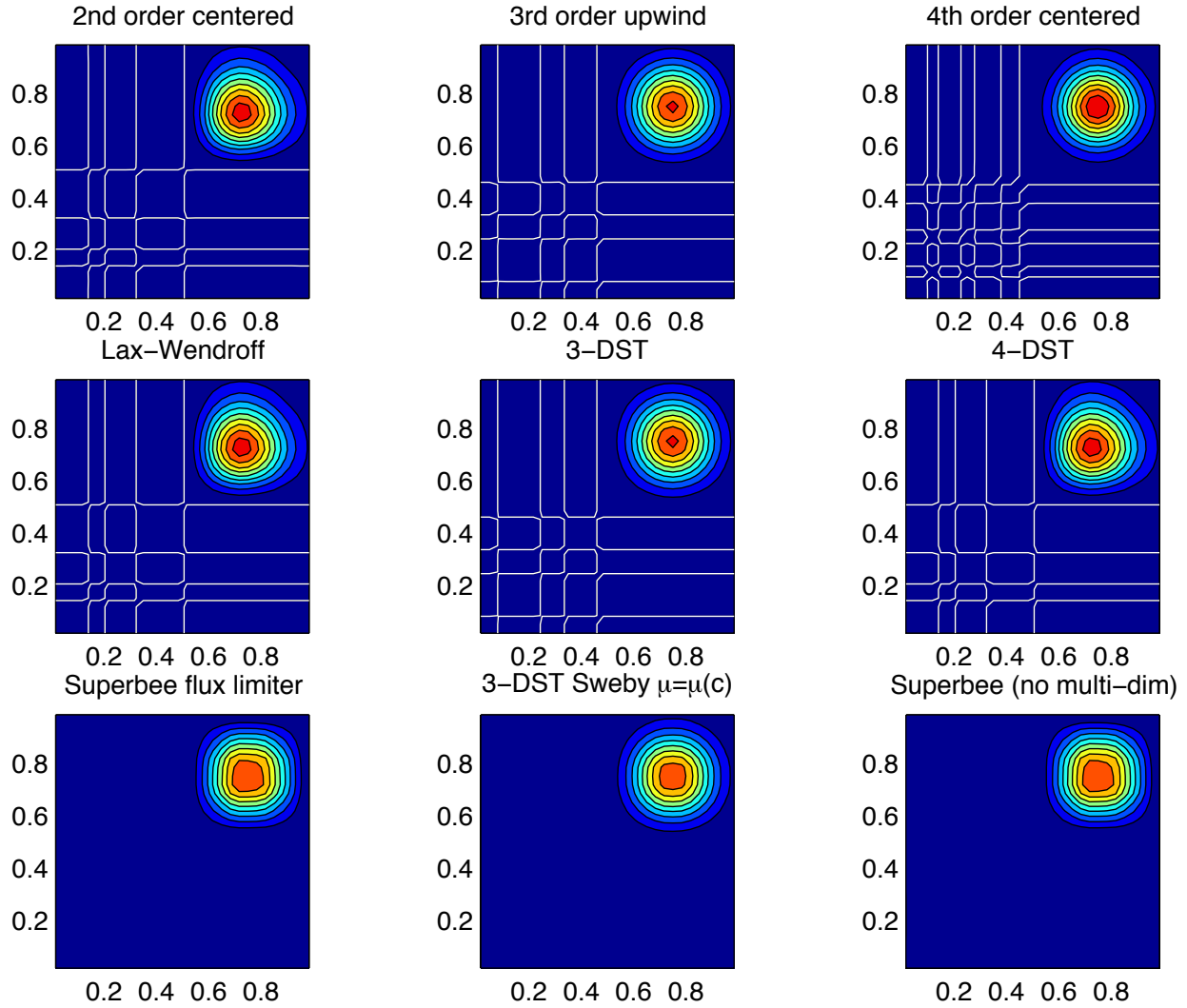


Figure 2.12: Comparison of advection schemes in two dimensions; diagonal advection of a resolved Gaussian feature. Courant number is 0.01 with 30×30 points and solutions are shown for $T=1/2$. White lines indicate zero crossing (ie. the presence of false minima). The left column shows the second order 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).

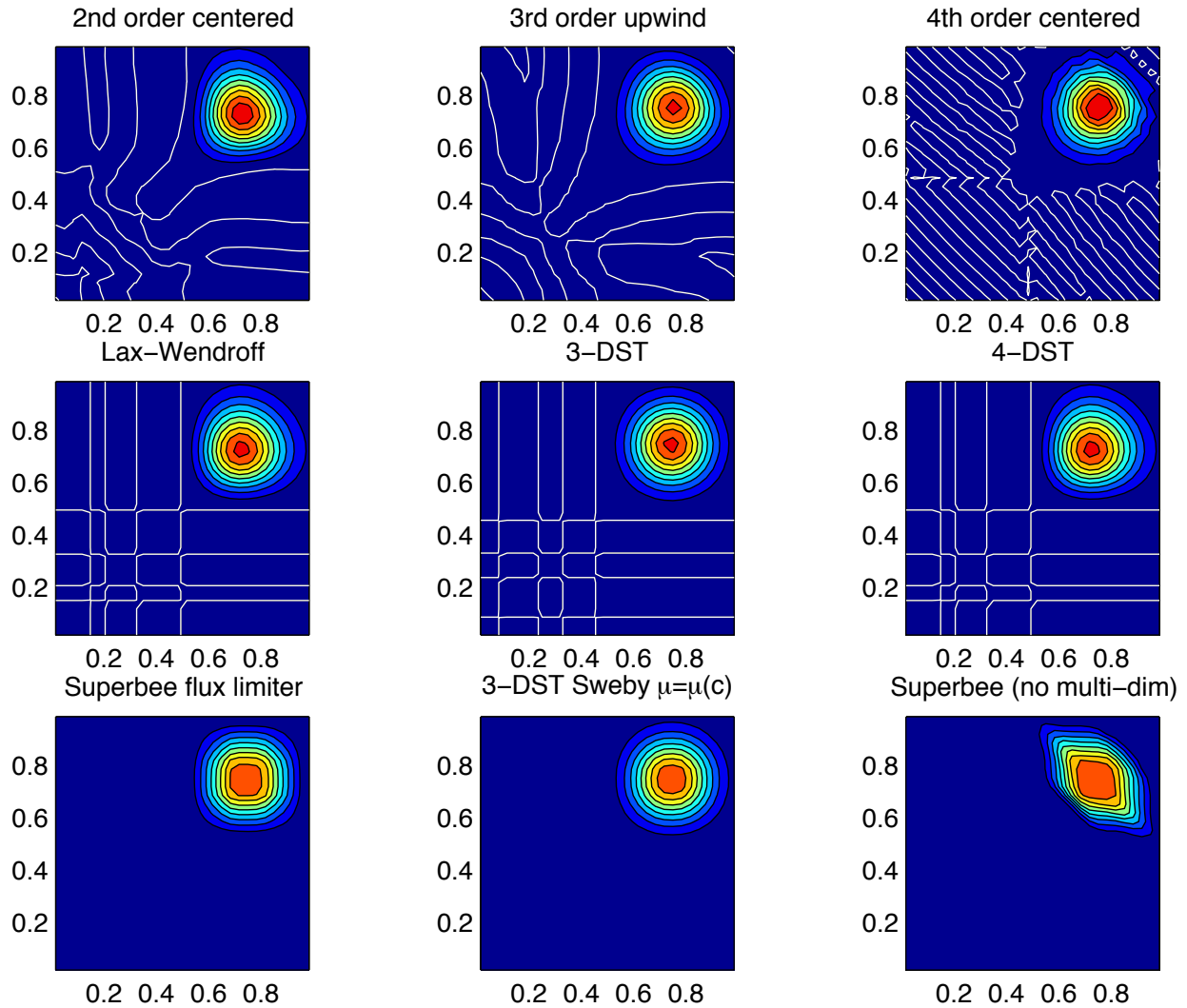


Figure 2.13: Comparison of advection schemes in two dimensions; diagonal advection of a resolved Gaussian feature. Courant number is 0.27 with 30×30 points and solutions are shown for $T=1/2$. White lines indicate zero crossing (ie. the presence of false minima). The left column shows the second order 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).

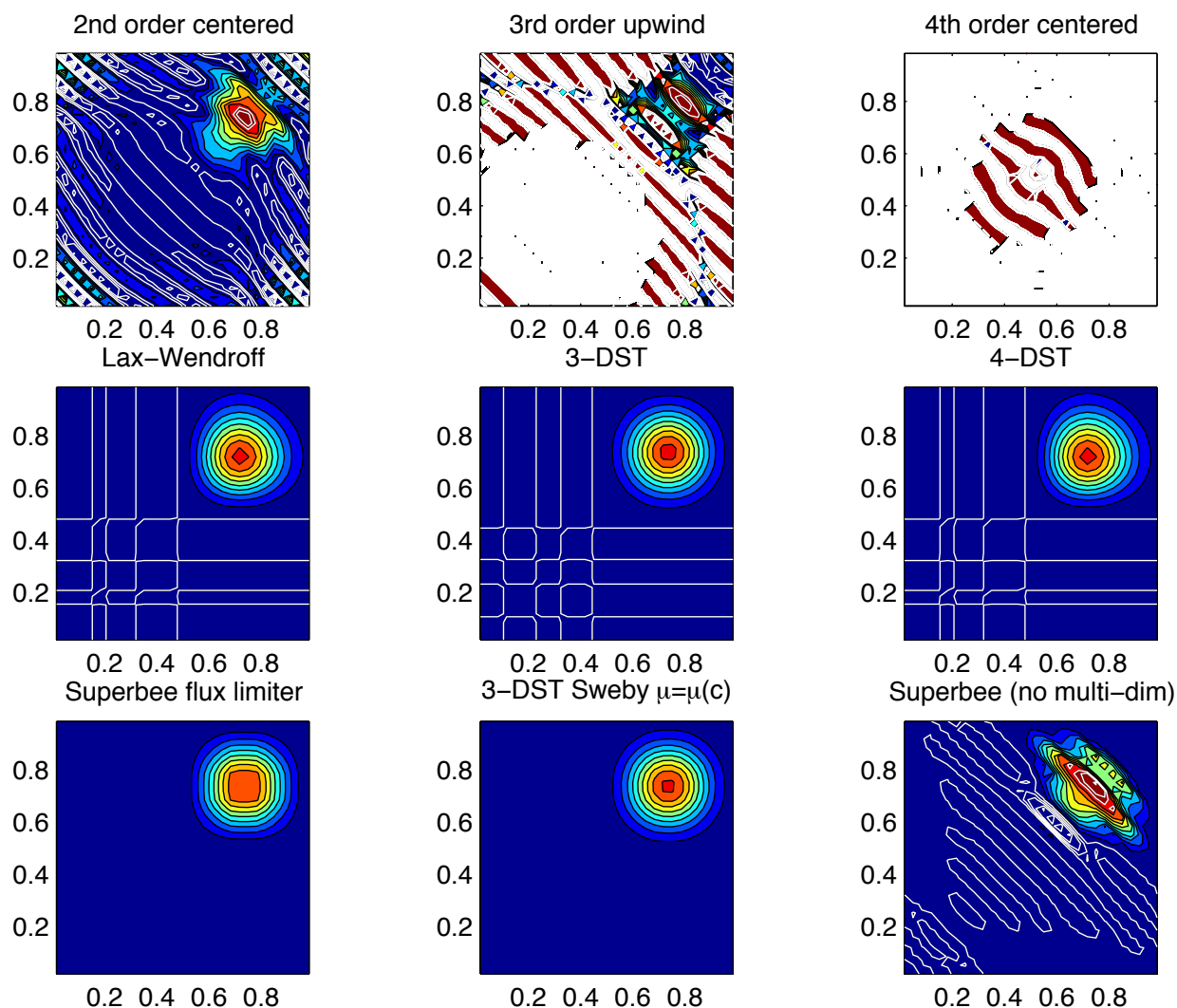


Figure 2.14: Comparison of advection schemes in two dimensions; diagonal advection of a resolved Gaussian feature. Courant number is 0.47 with 30×30 points and solutions are shown for $T=1/2$. White lines indicate zero crossings and initial maximum values (ie. the presence of false extrema). The left 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).

- 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.

2.20 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 τ_{shap} defines the strength of the filter damping.

The three computational filter operators are :

$$\begin{aligned}
 \text{S1c :} \quad & [1 - 1/2 \frac{\Delta t}{\tau_{shap}} \{ (\frac{1}{4} \delta_{ii})^n + (\frac{1}{4} \delta_{jj})^n \}] \\
 \text{S2c :} \quad & [1 - \frac{\Delta t}{\tau_{shap}} \{ \frac{1}{8} (\delta_{ii} + \delta_{jj}) \}^n] \\
 \text{S4c :} \quad & [1 - \frac{\Delta t}{\tau_{shap}} (\frac{1}{4} \delta_{ii})^n] [1 - \frac{\Delta t}{\tau_{shap}} (\frac{1}{4} \delta_{jj})^n]
 \end{aligned}$$

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

$$\text{S2g :} \quad [1 - \frac{\Delta t}{\tau_{shap}} \{ \frac{L_{shap}^2}{8} \bar{\nabla}^2 \}^n]$$

with the Laplacian operator $\bar{\nabla}^2$ and a length scale parameter L_{shap} . The stability of this S2g filter requires $L_{shap} < \text{Min}^{(Global)}(\Delta x, \Delta y)$.

2.20.1 SHAP Diagnostics

<-Name-> Levs parsing code <-Units-> <- Tile (max=80c)						

SHAP_dT		5	SM	MR	K/s	Temperature Tendency due to Shapiro Filter
SHAP_dS		5	SM	MR	g/kg/s	Specific Humidity Tendency due to Shapiro Filter
SHAP_dU		5	UU	148MR	m/s^2	Zonal Wind Tendency due to Shapiro Filter
SHAP_dV		5	VV	147MR	m/s^2	Meridional Wind Tendency due to Shapiro Filter

2.21 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 ($L_\nu \approx 1\text{cm}$). 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 : $u_\alpha \rightarrow \overline{u_\alpha}$ and $L: b \rightarrow \bar{b}$. 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 (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:

$$\frac{\overline{D\tilde{u}}}{Dt} - \frac{\tilde{v} \sin \theta}{\text{Ro} \sin \theta_0} + \frac{M_{Ro}}{\text{Ro}} \frac{\partial \pi}{\partial x} = - \left(\frac{\overline{D\tilde{u}}}{Dt} - \frac{\overline{D\tilde{u}}}{Dt} \right) + \frac{\nabla^2 \tilde{u}}{\text{Re}} \quad (2.151)$$

$$\frac{\overline{D\tilde{v}}}{Dt} - \frac{\tilde{u} \sin \theta}{\text{Ro} \sin \theta_0} + \frac{M_{Ro}}{\text{Ro}} \frac{\partial \pi}{\partial y} = - \left(\frac{\overline{D\tilde{v}}}{Dt} - \frac{\overline{D\tilde{v}}}{Dt} \right) + \frac{\nabla^2 \tilde{v}}{\text{Re}} \quad (2.152)$$

$$\frac{\overline{D\bar{w}}}{Dt} + \frac{\frac{\partial \pi}{\partial z} - \bar{b}}{\text{Fr}^2 \lambda^2} = - \left(\frac{\overline{D\bar{w}}}{Dt} - \frac{\overline{D\bar{w}}}{Dt} \right) + \frac{\nabla^2 \bar{w}}{\text{Re}}$$

$$\frac{\overline{D\bar{b}}}{Dt} + \bar{w} = - \left(\frac{\overline{D\bar{b}}}{Dt} - \frac{\overline{D\bar{b}}}{Dt} \right) + \frac{\nabla^2 \bar{b}}{\text{Pr Re}}$$

$$\mu^2 \left(\frac{\partial \tilde{u}}{\partial x} + \frac{\partial \tilde{v}}{\partial y} \right) + \frac{\partial \bar{w}}{\partial z} = 0 \quad (2.153)$$

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 (2.151) - (2.153) 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.

2.21.1 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 (2.151) - (2.153):

$$\left(\frac{\overline{D\tilde{u}}}{Dt} - \frac{\overline{D\tilde{u}}}{Dt} \right) \approx \frac{\nabla_h^2 \tilde{u}}{\text{Re}_h} + \frac{\partial^2 \tilde{u}}{\partial z^2 \text{Re}_v} \quad (2.154)$$

$$\left(\frac{\overline{D\tilde{v}}}{Dt} - \frac{\overline{D\tilde{v}}}{Dt} \right) \approx \frac{\nabla_h^2 \tilde{v}}{\text{Re}_h} + \frac{\partial^2 \tilde{v}}{\partial z^2 \text{Re}_v} \quad (2.155)$$

$$\left(\frac{Dw}{Dt} - \frac{D\bar{w}}{Dt} \right) \approx \frac{\nabla_h^2 \bar{w}}{\text{Re}_h} + \frac{\partial^2 \bar{w}}{\partial z^2 \text{Re}_v} \quad (2.156)$$

$$\left(\frac{Db}{Dt} - \frac{D\bar{b}}{Dt} \right) \approx \frac{\nabla_h^2 \bar{b}}{\text{Pr Re}_h} + \frac{\partial^2 \bar{b}}{\partial z^2 \text{Pr Re}_v}$$

2.21.1.1 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 $\text{Re}_h < 2$.

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 $\text{Re}_h < 2$. 2) The user may use `viscAhReMax`, which ensures that the viscosity is always chosen so that $\text{Re}_h < \text{viscAhReMax}$. 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.

2.21.1.2 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` $\approx 1 \times 10^{-4} \text{m}^2/\text{s}$). 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).

2.21.1.3 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 , ϵ , 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 π 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 $L_\epsilon(\nu) > L$ (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 `viscAh` to ensure that

$L_\epsilon(A_h) > L$; 2) The user may use the approach of Smagorinsky with `viscC2Smag`, which estimates the energy flux at every grid point, and adjusts the viscosity accordingly.

Smagorinsky formed the energy equation from the momentum equations 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 νD^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 ν 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, $\epsilon \approx A_{hSmag} \bar{D}^2$. If we use this approximation to estimate the Kolmogorov viscous length, then

$$L_\epsilon(A_{hSmag}) \propto \pi \epsilon^{-1/4} A_{hSmag}^{3/4} \approx \pi (A_{hSmag} \bar{D}^2)^{-1/4} A_{hSmag}^{3/4} = \pi A_{hSmag}^{1/2} \bar{D}^{-1/2} \quad (2.157)$$

To make $L_\epsilon(A_{hSmag})$ scale with the gridscale, then

$$A_{hSmag} = \left(\frac{\text{viscC2Smag}}{\pi} \right)^2 L^2 |\bar{D}| \quad (2.158)$$

Where the deformation rate appropriate for hydrostatic flows with shallow-water scaling is

$$|\bar{D}| = \sqrt{\left(\frac{\partial \bar{u}}{\partial x} - \frac{\partial \bar{v}}{\partial y} \right)^2 + \left(\frac{\partial \bar{u}}{\partial y} + \frac{\partial \bar{v}}{\partial x} \right)^2} \quad (2.159)$$

The coefficient `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 `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:

$$A_{vSmag} = \left(\frac{\text{viscC2Smag}}{\pi} \right)^2 H^2 \sqrt{\left(\frac{\partial \bar{u}}{\partial z} \right)^2 + \left(\frac{\partial \bar{v}}{\partial z} \right)^2} \quad (2.160)$$

This vertical viscosity is currently not implemented in MITgcm.

2.21.1.4 Leith Viscosity

Leith (1968, 1996) [Lei68] [Lei96] notes that 2-d turbulence is quite different from 3-d. In two-dimensional 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 $\eta \approx A_{hLeith} |\nabla \bar{\omega}_3|^2$. By dimensional analysis, the enstrophy-dissipation scale is $L_\eta(A_{hLeith}) \propto \pi A_{hLeith}^{1/2} \eta^{-1/6}$. Thus, the Leith-estimated length scale of enstrophy-dissipation and the resulting eddy viscosity are

$$L_\eta(A_{hLeith}) \propto \pi A_{hLeith}^{1/2} \eta^{-1/6} = \pi A_{hLeith}^{1/3} |\nabla \bar{\omega}_3|^{-1/3} \quad (2.161)$$

$$A_{hLeith} = \left(\frac{\text{viscC2Leith}}{\pi} \right)^3 L^3 |\nabla \bar{\omega}_3| \quad (2.162)$$

$$|\nabla \omega_3| \equiv \sqrt{\left[\frac{\partial}{\partial x} \left(\frac{\partial \bar{v}}{\partial x} - \frac{\partial \bar{u}}{\partial y} \right) \right]^2 + \left[\frac{\partial}{\partial y} \left(\frac{\partial \bar{v}}{\partial x} - \frac{\partial \bar{u}}{\partial y} \right) \right]^2} \quad (2.163)$$

2.21.1.5 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:

$$A_{hLeith} = L^3 \sqrt{\left(\frac{\text{viscC2Leith}}{\pi} \right)^6 |\nabla \bar{\omega}_3|^2 + \left(\frac{\text{viscC2LeithD}}{\pi} \right)^6 |\nabla \nabla \cdot \bar{u}_h|^2} \quad (2.164)$$

$$|\nabla \nabla \cdot \bar{u}_h| \equiv \sqrt{\left[\frac{\partial}{\partial x} \left(\frac{\partial \bar{u}}{\partial x} + \frac{\partial \bar{v}}{\partial y} \right) \right]^2 + \left[\frac{\partial}{\partial y} \left(\frac{\partial \bar{u}}{\partial x} + \frac{\partial \bar{v}}{\partial y} \right) \right]^2} \quad (2.165)$$

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 vales 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`.

2.21.1.6 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 `viscAhGridMax` < 1 and `viscA4GridMax` < 1. Similarly-scaled minimum values of viscosities are provided by `viscAhGridMin` and `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. `viscAhGridMax` < 1), the biharmonic equivalent is scaled so that the same non-dimensional value can be used (e.g. `viscA4GridMax` < 1).

2.21.1.7 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:

$$\left(\frac{\overline{D\tilde{u}}}{Dt} - \frac{\overline{D\tilde{u}}}{Dt}\right) \approx \frac{-\nabla_h^4 \tilde{u}}{\text{Re}_4} + \frac{\frac{\partial^2 \tilde{u}}{\partial z^2}}{\text{Re}_v} \quad (2.166)$$

$$\left(\frac{\overline{D\tilde{v}}}{Dt} - \frac{\overline{D\tilde{v}}}{Dt}\right) \approx \frac{-\nabla_h^4 \tilde{v}}{\text{Re}_4} + \frac{\frac{\partial^2 \tilde{v}}{\partial z^2}}{\text{Re}_v}$$

$$\left(\frac{\overline{D\tilde{w}}}{Dt} - \frac{\overline{D\tilde{w}}}{Dt}\right) \approx \frac{-\nabla_h^4 \tilde{w}}{\text{Re}_4} + \frac{\frac{\partial^2 \tilde{w}}{\partial z^2}}{\text{Re}_v}$$

$$\left(\frac{\overline{D\tilde{b}}}{Dt} - \frac{\overline{D\tilde{b}}}{Dt}\right) \approx \frac{-\nabla_h^4 \tilde{b}}{\text{Pr Re}_4} + \frac{\frac{\partial^2 \tilde{b}}{\partial z^2}}{\text{Pr Re}_v}$$

Griffies and Hallberg (2000) [GH00] propose 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 (2.154) - (2.157) and (2.166) - (2.167) 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:

$$A_{4Smag} = \left(\frac{\text{viscC4Smag}}{\pi}\right)^2 \frac{L^4}{8} |D| \quad (2.167)$$

$$A_{4Leith} = \frac{L^5}{8} \sqrt{\left(\frac{\text{viscC4Leith}}{\pi}\right)^6 |\nabla \bar{\omega}_3|^2 + \left(\frac{\text{viscC4LeithD}}{\pi}\right)^6 |\nabla \nabla \cdot \bar{\mathbf{u}}_h|^2} \quad (2.168)$$

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:

$$A_{4Smag} = \left(\frac{\text{viscC4Smag}}{\pi} \right)^5 L^5 |\nabla^2 \bar{\mathbf{u}}_h| \quad (2.169)$$

$$A_{4Leith} = L^6 \sqrt{\left(\frac{\text{viscC4Leith}}{\pi} \right)^{12} |\nabla^2 \bar{\omega}_3|^2 + \left(\frac{\text{viscC4LeithD}}{\pi} \right)^{12} |\nabla^2 \nabla \cdot \bar{\mathbf{u}}_h|^2} \quad (2.170)$$

Thus, the biharmonic scaling suggested by Griffies and Hallberg (2000) [GH00] implies:

$$\begin{aligned} |D| &\propto L |\nabla^2 \bar{\mathbf{u}}_h| \\ |\nabla^2 \bar{\omega}_3| &\propto L |\nabla^2 \bar{\omega}_3| \end{aligned}$$

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.

2.21.1.8 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 geometric 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.

2.21.2 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 θ_0 and geopotential height $z = r - r_0$. The nondimensional form of these equations is:

$$\text{Ro} \frac{D\tilde{u}}{Dt} - \frac{\tilde{v} \sin \theta}{\sin \theta_0} + M_{Ro} \frac{\partial \pi}{\partial x} + \frac{\lambda \text{Fr}^2 M_{Ro} \cos \theta}{\mu \sin \theta_0} w = -\frac{\text{Fr}^2 M_{Ro} \tilde{u} w}{r/H} + \frac{\text{Ro} \hat{\mathbf{x}} \cdot \nabla^2 \mathbf{u}}{\text{Re}} \quad (2.171)$$

$$\text{Ro} \frac{D\tilde{v}}{Dt} + \frac{\tilde{u} \sin \theta}{\sin \theta_0} + M_{Ro} \frac{\partial \pi}{\partial y} = -\frac{\mu \text{Ro} \tan \theta (\tilde{u}^2 + \tilde{v}^2)}{r/L} - \frac{\text{Fr}^2 M_{Ro} \tilde{v} w}{r/H} + \frac{\text{Ro} \hat{\mathbf{y}} \cdot \nabla^2 \mathbf{u}}{\text{Re}} \quad (2.172)$$

$$\text{Fr}^2 \lambda^2 \frac{Dw}{Dt} - b + \frac{\partial \pi}{\partial z} - \frac{\lambda \cot \theta_0 \tilde{u}}{M_{Ro}} = \frac{\lambda \mu^2 (\tilde{u}^2 + \tilde{v}^2)}{M_{Ro}(r/L)} + \frac{\text{Fr}^2 \lambda^2 \hat{\mathbf{z}} \cdot \nabla^2 \mathbf{u}}{\text{Re}} \quad (2.173)$$

$$\frac{Db}{Dt} + w = \frac{\nabla^2 b}{\text{Pr Re}}$$

$$\mu^2 \left(\frac{\partial \tilde{u}}{\partial x} + \frac{\partial \tilde{v}}{\partial y} \right) + \frac{\partial w}{\partial z} = 0 \quad (2.174)$$

Where

$$\begin{aligned}
\mu &\equiv \frac{\cos \theta_0}{\cos \theta}, \quad \tilde{u} = \frac{u^*}{V\mu}, \quad \tilde{v} = \frac{v^*}{V\mu} \\
f_0 &\equiv 2\Omega \sin \theta_0, \quad \frac{D}{Dt} \equiv \mu^2 \left(\tilde{u} \frac{\partial}{\partial x} + \tilde{v} \frac{\partial}{\partial y} \right) + \frac{\text{Fr}^2 M_{Ro}}{Ro} w \frac{\partial}{\partial z} \\
x &\equiv \frac{r}{L} \phi \cos \theta_0, \quad y \equiv \frac{r}{L} \int_{\theta_0}^{\theta} \frac{\cos \theta_0 d\theta'}{\cos \theta'}, \quad z \equiv \lambda \frac{r - r_0}{L} \\
t^* &= t \frac{L}{V}, \quad b^* = b \frac{V f_0 M_{Ro}}{\lambda} \\
\pi^* &= \pi V f_0 L M_{Ro}, \quad w^* = w V \frac{\text{Fr}^2 \lambda M_{Ro}}{Ro} \\
Ro &\equiv \frac{V}{f_0 L}, \quad M_{Ro} \equiv \max[1, Ro] \\
Fr &\equiv \frac{V}{N \lambda L}, \quad Re \equiv \frac{V L}{\nu}, \quad Pr \equiv \frac{\nu}{\kappa}
\end{aligned}$$

Dimensional variables are denoted by an asterisk where necessary. If we filter over a grid scale typical for ocean models:

$$\begin{aligned}
1\text{m} &< L < 100\text{km} \\
0.0001 &< \lambda < 1 \\
0.001\text{m/s} &< V < 1\text{ m/s} \\
f_0 &< 0.0001\text{ s}^{-1} \\
0.01\text{ s}^{-1} &< N < 0.0001\text{ s}^{-1}
\end{aligned}$$

these equations are very well approximated by

$$Ro \frac{D\tilde{u}}{Dt} - \frac{\tilde{v} \sin \theta}{\sin \theta_0} + M_{Ro} \frac{\partial \pi}{\partial x} = - \frac{\lambda \text{Fr}^2 M_{Ro} \cos \theta}{\mu \sin \theta_0} w + \frac{Ro \nabla^2 \tilde{u}}{Re} \quad (2.175)$$

$$Ro \frac{D\tilde{v}}{Dt} + \frac{\tilde{u} \sin \theta}{\sin \theta_0} + M_{Ro} \frac{\partial \pi}{\partial y} = \frac{Ro \nabla^2 \tilde{v}}{Re} \quad (2.176)$$

$$\text{Fr}^2 \lambda^2 \frac{Dw}{Dt} - b + \frac{\partial \pi}{\partial z} = \frac{\lambda \cot \theta_0 \tilde{u}}{M_{Ro}} + \frac{\text{Fr}^2 \lambda^2 \nabla^2 w}{Re} \quad (2.177)$$

$$\frac{Db}{Dt} + w = \frac{\nabla^2 b}{Pr Re} \quad (2.178)$$

$$\mu^2 \left(\frac{\partial \tilde{u}}{\partial x} + \frac{\partial \tilde{v}}{\partial y} \right) + \frac{\partial w}{\partial z} = 0 \quad (2.179)$$

$$\nabla^2 \approx \left(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} + \frac{\partial^2}{\lambda^2 \partial z^2} \right)$$

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.

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 the MITgcm. The second part, covered in [Section 4](#), contains a set of step-by-step tutorials for running specific pre-configured atmospheric and oceanic experiments.

We believe the best way to familiarize yourself with the model is to run the case study examples provided in the MITgcm repository. Information is also provided here on how to customize the code when you are ready to try implementing the configuration you have in mind. The code and algorithm are described more fully in [Section 2](#) and [Section 6](#) and chapters thereafter.

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 checkpointXXX` 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 through 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 100MB. 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 the 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/checkpointXXX.zip
```

3.3 Updating the code

There are several different approaches one can use to obtain updates to the MITgcm; which is best for you depends a bit on how you intend to use the MITgcm and your knowledge of git (and/or willingness to learn). Below we outline three suggested update pathways:

1. Fresh Download of the 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 the 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 in [Section 3.5.1](#), 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 the 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 `chapWrapper` 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 `inc` (includes files) and `src` (source code).
- `eesupp`: contains the execution environment source code. Also subdivided into two subdirectories `inc` and `src`.
- `pkg`: contains the source code for the packages. Each package corresponds to a subdirectory. For example, `gmredi` contains the code related to the Gent-McWilliams/Redi scheme, `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](#).
- `doc`: contains the 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 `build_options` contains ‘optfiles’ with the compiler options for many different compilers and machines that can run MITgcm (see [Section 3.5.2.1](#)). This

directory also contains subdirectories `adjoint` and `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 `matlab` subdirectory contains matlab scripts for reading model output directly into matlab. The subdirectory `python` contains similar routines for python. `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 code

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 3.5.2](#), that automatically creates the `Makefile` for you. You then need to build the dependencies and compile the code.

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, build the `Makefile`:

```
% ../../../../tools/genmake2 -mods ../code
```

The `-mods` command line option tells `genmake2` to override model source code with any files in the directory `../code/`. This and additional `genmake2` command line options are described more fully in [Section 3.5.2.2](#).

On many systems, the `genmake2` program will be able to automatically recognize the hardware, find compilers and other tools within the user's path ("echo \$PATH"), and then choose an appropriate set of options from the files ("optfiles") contained in the `tools/build_options` directory. Under some circumstances, a user may have to create a new optfile in order to specify the exact combination of compiler, compiler flags, libraries, and other options necessary to build a particular configuration of MITgcm. In such cases, it is generally helpful to peruse the existing optfiles and mimic their syntax. See [Section 3.5.2.1](#).

The MITgcm developers are willing to provide help writing or modifying optfiles. And we encourage users to ask for assistance or post new optfiles (particularly ones for new machines or architectures) through the [GitHub issue tracker](#) or email the MITgcm-support@mitgcm.org list.

To specify an optfile to `genmake2`, the command line syntax is:

```
% ../../../../tools/genmake2 -mods ../code -of /path/to/optfile
```

Once a `Makefile` has been generated, we create the dependencies with the command:

```
% make depend
```

This 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 links from the model source to this directory, except for links to those files in the specified `-mods` directory. **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 was 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 to this newly edited file).

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`). 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
```

The `make` command creates an executable called `mitgcmuv`. Additional `make` “targets” are defined within the makefile to aid in the production of adjoint and other versions of MITgcm. On computers with multiple processor cores or shared multi-processor (a.k.a. SMP) systems, the build process can often be sped up appreciably using the command:

```
% 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`
- `make CLEAN` removes pretty much everything, including any executables and output from `genmake2`

Now you are ready to run the model. General instructions for doing so are given in section [Section 3.6](#).

3.5.1 Building/compiling the code elsewhere

In the example above ([Section 3.5](#)) we built the executable in the `build` directory of the experiment. Model object files and output data can use up large amounts of disk space so it is often preferable to operate on a large scratch disk. Here, we show how to configure and compile the code on a scratch disk, without having to copy the entire source tree. The only requirement to do so is you have `genmake2` in your path, or you know the absolute path to `genmake2`.

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
```

Note the use of the command line option `-rootdir` to tell `genmake2` where to find the MITgcm directory tree. In general, one can compile the code in any given directory by following this procedure.

3.5.2 Using `genmake2`

This section describes further details and capabilities of `genmake2` (located in the `tools` directory), the MITgcm tool used to generate a Makefile. `genmake2` is a shell script written to work with all “sh”-compatible shells including `bash v1`, `bash v2`, and `Bourne` (like many unix tools, there is a help option that is invoked thru `genmake -h`). `genmake2` parses information from the following sources:

- a `genmake_local` file if one is found in the current directory
- command-line options
- an “options file” as specified by the command-line option `-of /path/to/filename`
- a `packages.conf` file (if one is found) with the specific list of packages to compile. The search path for file `packages.conf` is first the current directory, and then each of the `-mods` directories in the given order (see [here](#)).

3.5.2.1 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 and 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 terminal `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 is 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 executable)

In many cases, the default optfiles are sufficient and will result in usable Makefiles. However, for some machines or 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 using a simple sh-compatible syntax. While nearly all 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

DEFINES the command-line DEFINE options passed to the compiler

CPP the C pre-processor to use

NOOPTFLAGS options flags for special files that should not be optimized

For example, the optfile for a typical Red Hat Linux machine (amd64 architecture) using the GCC (g77) compiler is

```
FC=g77
DEFINES='-D_BYTESWAP_IO -DWORDLENGTH=4'
CPP='cpp -traditional -P'
NOOPTFLAGS='-O0'
# For IEEE, use the "-ffloat-store" option
if test "x$IEEE" = x ; then
    FFLAGS='-Wimplicit -Wunused -Wuninitialized'
    FOPTIM='-O3 -malign-double -funroll-loops'
else
    FFLAGS='-Wimplicit -Wunused -ffloat-store'
    FOPTIM='-O0 -malign-double'
fi
```

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. Please submit the file through the [GitHub issue tracker](#) or email the MITgcm-support@mitgcm.org list.

3.5.2.2 Command-line options:

In addition to the optfiles, `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/file specifies the optfile that should be used for a particular build.

If no optfile is specified (either through the command line or the `MITGCM_OPTFILE` environment variable), `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 (eg. “linux_amd64”) 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.

-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.

The order of precedence for this “name-hiding” is as follows:

- “mods” directories (in the order given)
- Packages either explicitly specified or provided by default (in the order given)
- Packages included due to package dependencies (in the order that that package dependencies are parsed)
- The “standard dirs” (which may have been specified by the “-standarddirs” option)

-oad generates a makefile for a OpenAD build

-adof /path/to/file 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.

The default file is located in `tools/adjoint_options/adjoint_default` and it defines the “TAF” and “TAMC” compilers. An alternate version is also available at `tools/adjoint_options/adjoint_staf` that selects the newer “STAF” compiler. As with any compilers, it is helpful to have their directories listed in your `$PATH` environment variable.

-mpi enables certain MPI features (using CPP `#define`) within the code and is necessary for MPI builds (see [Section 3.5.3](#)).

-omp enables OPENMP code and compiler flag `OMPFLAG`

-ieee use IEEE numerics (requires support in optfile)

-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.

3.5.3 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, its 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. Determine the locations of your MPI-enabled compiler and/or MPI libraries and put them into an options file as described in [Section 3.5.2.1](#). One can start with one of the examples in [tools/build_options](#) such as `linux_amd64_gfortran` or `linux_amd64_ift+impi` and then edit it to suit the machine at hand. You may need help from your user guide or local systems administrator to determine the exact location of the MPI libraries. If libraries are not installed, MPI implementations and related tools are available including:
 - [Open MPI](#)
 - [MVAPICH2](#)
 - [MPICH](#)
 - [Intel MPI](#)
2. Build the code with the `genmake2 -mpi` option (see [Section 3.5.2.2](#)) using commands such as:

```
% ../../tools/genmake2 -mods=../code -mpi -of=YOUR_OPTFILE
% make depend
% make
```

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 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 experiments in `verification`, an example of the output is kept in `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, 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 Output files

The model produces various output files and, when using `mnc` (i.e., NetCDF), sometimes even directories. Depending upon the I/O package(s) selected at compile time (either `mdsio` or `mnc` or both as determined by `code/packages.conf`) and the run-time flags set (in `input/data.pkg`), the following output may appear. More complete information describing output files and model diagnostics is described in `chap_diagnosticsio`.

3.6.2.1 MDSIO output files

The “traditional” output files are generated by the `mdsio` package (link to `section_mdsio`). The `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 (psu), 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 packages and parameterizations also produce separate pickup files, e.g.,

- `pickup_cd.00000nIter` if the C-D scheme is used (see link to description)
- `pickup_seaice.00000nIter` if the seaice package is turned on (see link to description)
- `pickup_ptracers.00000nIter` if passive tracers are included in the simulation (see link to description)

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.

3.6.2.2 MNC output files

The MNC package (link to section_mnc) is a set of routines written to read, write, and append [NetCDF files](#). Unlike the `mdsio` output, the `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 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.

3.6.3 Looking at the output

3.6.3.1 MATLAB

MDSIO output

The repository includes a few Matlab utilities to read output files written in the `mdsio` format. The Matlab scripts are located in the directory `utils/matlab` under the root tree. The script `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=rdmds('Eta',[0:10:100]);
>> for n=1:11; imagesc(eta(:,:,n));axis ij;colorbar;pause(.5);end
```

NetCDF

Similar scripts for netCDF output (`rdmnc.m`) are available and they are described in Section [\[sec:pkg:mnc\]](#).

3.6.3.2 Python

MDSIO output

The repository includes Python scripts for reading the `mdsio` format under `utils/python`. The following example shows how to load in some data:


```
# python
import mds

Eta = mds.rdmads('Eta', itrs=10)
```

The docstring for `mds.rdmads` contains much more detail about using this function and the options that it takes.

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 `gluemncbig.py` in the `utils/python` folder can do this efficiently from the command line.

The following example shows how to use the [xarray package](#) to read the resulting NetCDF file into python:

```
# python
import xarray as xr

Eta = xr.open_dataset('Eta.nc')
```

3.7 Customizing the model configuration

When you are ready to run the model in the configuration you want, the easiest thing is to use and adapt the setup of the case studies experiment (described in [Section 4](#)) that is the closest to your configuration. Then, the amount of setup will be minimized. In this section, we focus on the setup relative to the “numerical model” part of the code (the setup relative to the “execution environment” part is covered in the software architecture/wrapper section) and on the variables and parameters that you are likely to change.

In what follows, the parameters are grouped into categories related to the computational domain, the equations solved in the model, and the simulation controls.

3.7.1 Parameters: Computational Domain, Geometry and Time-Discretization

Dimensions

The number of points in the x, y, and r directions are represented by the variables `sNx`, `sNy` and `Nr` respectively which are declared and set in the file `SIZE.h`. (Again, this assumes a mono-processor calculation. For multiprocessor calculations see the section on parallel implementation.)

Grid

Three different grids are available: cartesian, spherical polar, and curvilinear (which includes the cubed sphere). The grid is set through the logical variables `usingCartesianGrid`, `usingSphericalPolarGrid`, and `usingCurvilinearGrid`. In the case of spherical and curvilinear grids, the southern boundary is defined through the variable `ygOrigin` which corresponds to the latitude of the southern most cell face (in degrees). The resolution along the x and y directions is controlled by the 1D arrays `delx` and `dely` (in meters in the case of a cartesian grid, in degrees otherwise). The vertical grid spacing is set through the 1D array `delz` for the ocean (in meters) or `delp` for the atmosphere (in Pa). The variable `Ro_SeaLevel` represents the standard position of sea level in “r” coordinate. This is typically set to 0 m for the ocean (default value) and 10^5 Pa for the atmosphere. For the atmosphere, also set the logical variable `groundAtK1` to `.TRUE.` which puts the first level (k=1) at the lower boundary (ground).

For the cartesian grid case, the Coriolis parameter f is set through the variables `f0` and `beta` which correspond to the reference Coriolis parameter (in s^{-1}) and $\frac{\partial f}{\partial y}$ (in $\text{m}^{-1}\text{s}^{-1}$) respectively. If `beta` is set to a nonzero value, `f0` is the value of f at the southern edge of the domain.

Topography - Full and Partial Cells

The domain bathymetry is read from a file that contains a 2D (x,y) map of depths (in m) for the ocean or pressures (in Pa) for the atmosphere. The file name is represented by the variable `bathyFile`. The file is assumed to contain binary numbers giving the depth (pressure) 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 model code applies without modification to enclosed, periodic, and 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 (note: not sure this is the case for the atmosphere). The precision with which to read the binary data is controlled by the integer variable `readBinaryPrec` which can take the value 32 (single precision) or 64 (double precision). See the matlab program `gendata.m` in the `input` directories of `verification` for several tutorial examples (e.g. `gendata.m` in the *barotropic gyre tutorial*) to see how the bathymetry files are generated for the case study experiments.

To use the partial cell capability, the variable `hFacMin` needs to be set to a value between 0 and 1 (it is set to 1 by default) corresponding to the minimum fractional size of the cell. For example if the bottom cell is 500 m thick and `hFacMin` is set to 0.1, the actual thickness of the cell (i.e. used in the code) can cover a range of discrete values 50 m apart from 50 m to 500 m depending on the value of the bottom depth (in `bathyFile`) at this point.

Note that the bottom depths (or pressures) need not coincide with the models levels as deduced from `delz` or `delp`. The model will interpolate the numbers in `bathyFile` so that they match the levels obtained from `delz` or `delp` and `hFacMin`.

(Note: the atmospheric case is a bit more complicated than what is written here. To come soon...)

Time-Discretization

The time steps are set through the real variables `deltaTMom` and `deltaTtracer` (in s) which represent the time step for the momentum and tracer equations, respectively. For synchronous integrations, simply set the two variables to the same value (or you can prescribe one time step only through the variable `deltaT`). The Adams-Bashforth stabilizing parameter is set through the variable `abEps` (dimensionless). The stagger baroclinic time stepping can be activated by setting the logical variable `staggerTimeStep` to `.TRUE..`

3.7.2 Parameters: Equation of State

First, because the model equations are written in terms of perturbations, a reference thermodynamic state needs to be specified. This is done through the 1D arrays `tRef` and `sRef`. `tRef` specifies the reference potential temperature profile (in $^{\circ}\text{C}$ for the ocean and K for the atmosphere) starting from the level $k=1$. Similarly, `sRef` specifies the reference salinity profile (in ppt) for the ocean or the reference specific humidity profile (in g/kg) for the atmosphere.

The form of the equation of state is controlled by the character variables `buoyancyRelation` and `eosType`. `buoyancyRelation` is set to `OCEANIC` by default and needs to be set to `ATMOSPHERIC` for atmosphere simulations. In this case, `eosType` must be set to `IDEALGAS`. For the ocean, two forms of the equation of state are available: linear (set `eosType` to `LINEAR`) and a polynomial approximation to the full nonlinear equation (set `eosType` to `POLYNOMIAL`). In the linear case, you need to specify the thermal and haline expansion coefficients represented by the variables `tAlpha` (in K^{-1}) and `sBeta` (in ppt^{-1}). For the nonlinear case, 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 and 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).

There are also higher polynomials for the equation of state:

- 'UNESCO': The UNESCO equation of state formula 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, and it is only listed for completeness.*
- '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': The new, more accurate and less expensive equation of state by McDougall et al. (1983) [MJWF03]. It also requires lagging the pressure and therefore an additional pickup file for restarts.

For none of these options an reference profile of temperature or salinity is required.

3.7.3 Parameters: Momentum Equations

In this section, we only focus for now on the parameters that you are likely to change, i.e. the ones relative to forcing and dissipation for example. The details relevant to the vector-invariant form of the equations and the various advection schemes are not covered for the moment. We assume that you use the standard form of the momentum equations (i.e. the flux-form) with the default advection scheme. Also, 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`, `momForcing`, `useCoriolis`, `momPressureForcing`, `momStepping` and `metricTerms` and are assumed to be set to `.TRUE.` here. Look at the file `PARAMS.h` for a precise definition of these variables.

Initialization

The initial horizontal velocity components can be specified from binary files `uVelInitFile` and `vVelInitFile`. These files should contain 3D 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, even in the case of non-hydrostatic simulation (see, e.g., [verification/tutorial_deep_convection/input/](#)).

In the case of a restart (from the end of a previous simulation), the velocity field is read from a pickup file (see section on simulation control parameters) and the initial velocity files are ignored.

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 Pa and located at U-points (V-points). As for the bathymetry, the precision with which to read the binary data is controlled by the variable `readBinaryPrec`. 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.

There is also the possibility of prescribing time-dependent periodic forcing. To do this, concatenate the successive time records into a single file (for each stress component) ordered in a (x,y,t) fashion and set the following variables: `periodicExternalForcing` to `.TRUE.`, `externForcingPeriod` to the period (in s) of which the forcing varies (typically 1 month), and `externForcingCycle` to the repeat time (in s) of the forcing (typically 1 year; note `externForcingCycle` must be a multiple of `externForcingPeriod`). With these variables set up, the model will interpolate the forcing linearly at each iteration.

Dissipation

The lateral eddy viscosity coefficient is specified through the variable `viscAh` (in m^2s^{-1}). The vertical eddy viscosity coefficient is specified through the variable `viscAz` (in m^2s^{-1}) for the ocean and `viscAp` (in Pa^2s^{-1}) for the atmosphere. The vertical diffusive fluxes can be computed implicitly by setting the logical variable `implicitViscosity` to `.TRUE.`. In addition, biharmonic mixing can be added as well through the variable `viscA4` (in m^4s^{-1}). On a spherical polar grid, you might also need to set the variable `cosPower` which is set to 0 by default and which represents the power of cosine of latitude to multiply viscosity. 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 m/s) and quadratic (set the variable `bottomDragQuadratic`, dimensionless).

The Fourier and Shapiro filters are described elsewhere.

C-D Scheme

If you run at a sufficiently coarse resolution, you will need the C-D scheme for the computation of the Coriolis terms. The variable `tauCD`, which represents the C-D scheme coupling timescale (in s) needs to be set.

Calculation of Pressure/Geopotential

First, to run a non-hydrostatic ocean simulation, set the logical variable `nonHydrostatic` to `.TRUE.`. The pressure field is then inverted through a 3D elliptic equation. (Note: this capability is not available for the atmosphere yet.) By default, a hydrostatic simulation is assumed and a 2D elliptic equation is used to invert the pressure field. The parameters controlling the behavior of the elliptic solvers are the variables `cg2dMaxIters` and `cg2dTargetResidual` for the 2D case and `cg3dMaxIters` and `cg3dTargetResidual` for the 3D case. You probably won't need to alter the default values (are we sure of this?).

For the calculation of the surface pressure (for the ocean) or surface geopotential (for the atmosphere) you need to set the logical variables `rigidLid` and `implicitFreeSurface` (set one to `.TRUE.` and the other to `.FALSE.` depending on how you want to deal with the ocean upper or atmosphere lower boundary).

3.7.4 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. As for the momentum equations, we only describe for now the parameters that you are likely to change. The logical variables `tempDiffusion`, `tempAdvection`, `tempForcing`, and `tempStepping` allow you to turn on/off terms in the temperature equation (same thing for salinity or specific humidity with variables `saltDiffusion`, `saltAdvection` etc.). These variables are all assumed here to be set to `.TRUE.`. Look at file `PARAMS.h` for a precise definition.

Initialization

The initial tracer data can be contained in the binary files `hydrogThetaFile` and `hydrogSaltFile`. These files should contain 3D 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` mentioned *above*. In this case, the initial tracer data are uniform in x and y for each depth level.

Forcing

This part is more relevant for the ocean, the procedure for the atmosphere not being completely stabilized at the moment.

A combination of fluxes data and relaxation terms can be used for driving the tracer equations. For potential temperature, heat flux data (in W/m^2) can be stored in the 2D binary file `surfQfile`. Alternatively or in addition, the forcing can be specified through a relaxation term. The SST data to which the model surface temperatures are restored to are supposed to be stored in the 2D binary file `thetaClimFile`. The corresponding relaxation time scale coefficient is set through the variable `tauThetaClimRelax` (in s). The same

procedure applies for salinity with the variable names `EmPmRfile`, `saltClimFile`, and `tauSaltClimRelax` for freshwater flux (in m/s) and surface salinity (in ppt) data files and relaxation time scale coefficient (in s), respectively. Also for salinity, if the CPP key `USE_NATURAL_BCS` is turned on, natural boundary conditions are applied, i.e., when computing the surface salinity tendency, the freshwater flux is multiplied by the model surface salinity instead of a constant salinity value.

As for the other input files, the precision with which to read the data is controlled by the variable `readBinaryPrec`. Time-dependent, periodic forcing can be applied as well following the same procedure used for the wind forcing data (see [above](#)).

Dissipation

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 `diffKzT` and `diffKzS` (in m^2/s) for the ocean and `diffKpT` and `diffKpS` (in Pa^2/s) for the atmosphere. The vertical diffusive fluxes can be computed implicitly by setting the logical variable `implicitDiffusion` to `.TRUE.`. In addition, biharmonic diffusivities can be specified as well through the coefficients `diffK4T` and `diffK4S` (in m^4/s). Note that the cosine power scaling (specified through `cosPower`; see [above](#)) is applied to the tracer diffusivities (Laplacian and biharmonic) as well. The Gent and McWilliams parameterization for oceanic tracers is described in the package section. Finally, note that tracers can be also subject to Fourier and Shapiro filtering (see the corresponding section on these filters).

Ocean convection

Two options are available to parameterize ocean convection. To use the first option, a convective adjustment scheme, you need to set the variable `cadjFreq`, which represents the frequency (in s) 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 tracer 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` to a value (in m^2/s) you wish the tracer vertical diffusivities to have when mixing tracers vertically due to static instabilities. Note that `cadjFreq` and `ivdc_kappa` cannot both have non-zero value.

3.7.5 Parameters: Simulation Controls

The model "clock" is defined by the variable `deltaTClock` (in s) which determines the I/O frequencies and is used in tagging output. Typically, you will set it to the tracer time step for accelerated runs (otherwise it is simply set to the default time step `deltaT`). Frequency of checkpointing and dumping of the model state are referenced to this clock (see [below](#)).

Run Duration

The beginning of a simulation is set by specifying a start time (in s) through the real variable `startTime` or by specifying an initial iteration number through the integer variable `nIter0`. If these variables are set to nonzero values, the model will look for a "pickup" file `pickup.0000nIter0` to restart the integration. The end of a simulation is set through the real variable `endTime` (in s). Alternatively, you can specify instead the number of time steps to execute through the integer variable `nTimeSteps`.

Frequency of Output

Real variables defining frequencies (in s) with which output files are written on disk need to be set up. `dumpFreq` controls the frequency with which the instantaneous state of the model is saved. `chkPtFreq` and `pchkPtFreq` control the output frequency of rolling and permanent checkpoint files, respectively. In addition, time-averaged fields can be written out by setting the variable `taveFreq` (in s). The precision with which to write the binary data is controlled by the integer variable `writeBinaryPrec` (set it to 32 or 64).

3.7.6 Parameters: Default Values

The CPP keys relative to the “numerical model” part of the code are all defined and set in the file `CPP_OPTIONS.h` in the directory `model/inc/` or in one of the `code` directories of the case study experiments under `verification/`. The model parameters are defined and declared in the file `PARAMS.h` and their default values are set in the routine `set_defaults.F`. The default values can be modified in the namelist file `data` which needs to be located in the directory where you will run the model. The parameters are initialized in the routine `ini_parms.F`. Look at this routine to see in what part of the namelist the parameters are located. Here is a complete list of the model parameters related to the main model (namelist parameters for the packages are located in the package descriptions), their meaning, and their default values:

Name	Value	Description
<code>buoyancyRelation</code>	OCEANIC	buoyancy relation
<code>fluidIsAir</code>	F	fluid major constituent is air
<code>fluidIsWater</code>	T	fluid major constituent is water
<code>usingPCoords</code>	F	use pressure coordinates
<code>usingZCoords</code>	T	use z-coordinates
<code>tRef</code>	2.0E+01 at k=top	reference temperature profile (°C or K)
<code>sRef</code>	3.0E+01 at k=top	reference salinity profile (psu)
<code>viscAh</code>	0.0E+00	lateral eddy viscosity (m ² /s)
<code>viscAhMax</code>	1.0E+21	maximum lateral eddy viscosity (m ² /s)
<code>viscAhGrid</code>	0.0E+00	grid dependent lateral eddy viscosity (non-dim.)
<code>useFullLeith</code>	F	use full form of Leith viscosity on/off flag
<code>useStrainTensionVisc</code>	F	use StrainTension form of viscous operator on/off flag
<code>useAreaViscLength</code>	F	use area for visc length instead of geom. mean
<code>viscC2leith</code>	0.0E+00	Leith harmonic visc. factor (on grad(vort),non-dim.)
<code>viscC2leithD</code>	0.0E+00	Leith harmonic viscosity factor (on grad(div),non-dim.)
<code>viscC2smag</code>	0.0E+00	Smagorinsky harmonic viscosity factor (non-dim.)
<code>viscA4</code>	0.0E+00	lateral biharmonic viscosity (m ⁴ /s)
<code>viscA4Max</code>	1.0E+21	maximum biharmonic viscosity (m ⁴ /s)
<code>viscA4Grid</code>	0.0E+00	grid dependent biharmonic viscosity (non-dim.)
<code>viscC4leith</code>	0.0E+00	Leith biharmonic viscosity factor (on grad(vort), non-dim.)
<code>viscC4leithD</code>	0.0E+00	Leith biharmonic viscosity factor (on grad(div), non-dim.)
<code>viscC4smag</code>	0.0E+00	Smagorinsky biharmonic viscosity factor (non-dim)
<code>no_slip_sides</code>	T	viscous BCs: no-slip sides
<code>sideDragFactor</code>	2.0E+00	side-drag scaling factor (non-dim)
<code>viscAr</code>	0.0E+00	vertical eddy viscosity (units of r ² /s)
<code>no_slip_bottom</code>	T	viscous BCs: no-slip bottom
<code>bottomDragLinear</code>	0.0E+00	linear bottom-drag coefficient (m/s)
<code>bottomDragQuadratic</code>	0.0E+00	quadratic bottom-drag coeff. (1)
<code>diffKhT</code>	0.0E+00	Laplacian diffusion of heat laterally (m ² /s)
<code>diffK4T</code>	0.0E+00	biharmonic diffusion of heat laterally (m ⁴ /s)
<code>diffKhS</code>	0.0E+00	Laplacian diffusion of salt laterally (m ² /s)
<code>diffK4S</code>	0.0E+00	biharmonic diffusion of salt laterally (m ⁴ /s)
<code>diffKrNrT</code>	0.0E+00 at k=top	vertical profile of vertical diffusion of temp (m ² /s)
<code>diffKrNrS</code>	0.0E+00 at k=top	vertical profile of vertical diffusion of salt (m ² /s)
<code>diffKrBL79surf</code>	0.0E+00	surface diffusion for Bryan and Lewis 1979 (m ² /s)
<code>diffKrBL79deep</code>	0.0E+00	deep diffusion for Bryan and Lewis 1979 (m ² /s)
<code>diffKrBL79scl</code>	2.0E+02	depth scale for Bryan and Lewis 1979 (m)
<code>diffKrBL79Ho</code>	-2.0E+03	turning depth for Bryan and Lewis 1979 (m)
<code>eosType</code>	LINEAR	equation of state

Continued on next page

Table 3.1 – continued from previous page

tAlpha	2.0E-04	linear EOS thermal expansion coefficient (1/°C)
--------	---------	---

Name	Value	Description
sBeta	7.4E-04	linear EOS haline contraction coef (1/psu)
rhonil	9.998E+02	reference density (kg/m ³)
rhoConst	9.998E+02	reference density (kg/m ³)
rhoConstFresh	9.998E+02	reference density (kg/m ³)
gravity	9.81E+00	gravitational acceleration (m/s ²)
gBaro	9.81E+00	barotropic gravity (m/s ²)
rotationPeriod	8.6164E+04	rotation period (s)
omega	2 π /rotationPeriod	angular velocity (rad/s)
f0	1.0E-04	reference coriolis parameter (1/s)
beta	1.0E-11	beta (m ⁻¹ s ⁻¹)
freeSurfFac	1.0E+00	implicit free surface factor
implicitFreeSurface	T	implicit free surface on/off flag
rigidLid	F	rigid lid on/off flag
implicSurfPress	1.0E+00	surface pressure implicit factor (0-1)
implicDiv2Dflow	1.0E+00	barotropic flow div. implicit factor (0-1)
exactConserv	F	exact volume conservation on/off flag
uniformLin_PhiSurf	T	use uniform Bo_surf on/off flag
nonlinFreeSurf	0	non-linear free surf. options (-1,0,1,2,3)
hFacInf	2.0E-01	lower threshold for hFac (nonlinFreeSurf only)
hFacSup	2.0E+00	upper threshold for hFac (nonlinFreeSurf only)
select_rStar	0	r
useRealFreshWaterFlux	F	real freshwater flux on/off flag
convertFW2Salt	3.5E+01	convert FW flux to salt flux (-1=use local S)
use3Dsolver	F	use 3-D pressure solver on/off flag
nonHydrostatic	F	non-hydrostatic on/off flag
nh_Am2	1.0E+00	non-hydrostatic terms scaling factor
quasiHydrostatic	F	quasi-hydrostatic on/off flag
momStepping	T	momentum equation on/off flag
vectorInvariantMomentum	F	vector-invariant momentum on/off
momAdvection	T	momentum advection on/off flag
momViscosity	T	momentum viscosity on/off flag
momImplVertAdv	F	momentum implicit vert. advection on/off
implicitViscosity	F	implicit viscosity on/off flag
metricTerms	F	metric terms on/off flag
useNHMTerms	F	non-hydrostatic metric terms on/off
useCoriolis	T	Coriolis on/off flag
useCDscheme	F	CD scheme on/off flag
useJamartWetPoints	F	Coriolis wetpoints method flag
useJamartMomAdv	F	VI non-linear terms Jamart flag

Name	Value	Description
SadournyCoriolis	F	Sadourny Coriolis discretization flag
upwindVorticity	F	upwind bias vorticity flag

Continued on next page

Table 3.3 – continued from previous page

useAbsVorticity	F	work with f
highOrderVorticity	F	high order interp. of vort. flag
upwindShear	F	upwind vertical shear advection flag
selectKEscheme	0	kinetic energy scheme selector
momForcing	T	momentum forcing on/off flag
momPressureForcing	T	momentum pressure term on/off flag
implicitIntGravWave	F	implicit internal gravity wave flag
staggerTimeStep	F	stagger time stepping on/off flag
multiDimAdvection	T	enable/disable multi-dim advection
useMultiDimAdvec	F	multi-dim advection is/is-not used
implicitDiffusion	F	implicit diffusion on/off flag
tempStepping	T	temperature equation on/off flag
tempAdvection	T	temperature advection on/off flag
tempImplVertAdv	F	temp. implicit vert. advection on/off
tempForcing	T	temperature forcing on/off flag
saltStepping	T	salinity equation on/off flag
saltAdvection	T	salinity advection on/off flag
saltImplVertAdv	F	salinity implicit vert. advection on/off
saltForcing	T	salinity forcing on/off flag
readBinaryPrec	32	precision used for reading binary files
writeBinaryPrec	32	precision used for writing binary files
globalFiles	F	write “global” (=not per tile) files
useSingleCpuIO	F	only master MPI process does I/O
debugMode	F	debug Mode on/off flag
debLevA	1	1st level of debugging
debLevB	2	2nd level of debugging
debugLevel	1	select debugging level
cg2dMaxIters	150	upper limit on 2d con. grad iterations
cg2dChkResFreq	1	2d con. grad convergence test frequency
cg2dTargetResidual	1.0E-07	2d con. grad target residual
cg2dTargetResWunit	-1.0E+00	cg2d target residual [W units]
cg2dPreCondFreq	1	freq. for updating cg2d pre-conditioner
nIter0	0	run starting timestep number
nTimeSteps	0	number of timesteps
deltatTmom	6.0E+01	momentum equation timestep (s)
deltaTfreesurf	6.0E+01	freeSurface equation timestep (s)
dTtracerLev	6.0E+01 at k=top	tracer equation timestep (s)
deltaTClock	6.0E+01	model clock timestep (s)

Name	Value	Description
cAdjFreq	0.0E+00	convective adjustment interval (s)
momForcingOutAB	0	=1: take momentum forcing out of Adams-Bashforth
tracForcingOutAB	0	=1: take T,S,pTr forcing out of Adams-Bashforth
momDissip_In_AB	T	put dissipation tendency in Adams-Bashforth
doAB_onGtGs	T	apply AB on tendencies (rather than on T,S)
abEps	1.0E-02	Adams-Bashforth-2 stabilizing weight
baseTime	0.0E+00	model base time (s)
startTime	0.0E+00	run start time (s)

Continued on next page

Table 3.4 – continued from previous page

endTime	0.0E+00	integration ending time (s)
pChkPtFreq	0.0E+00	permanent restart/checkpoint file interval (s)
chkPtFreq	0.0E+00	rolling restart/checkpoint file interval (s)
pickup_write_mdsio	T	model I/O flag
pickup_read_mdsio	T	model I/O flag
pickup_write_immed	F	model I/O flag
dumpFreq	0.0E+00	model state write out interval (s)
dumpInitAndLast	T	write out initial and last iteration model state
snapshot_mdsio T model I/O flag.		
monitorFreq	6.0E+01	monitor output interval (s)
monitor_stdio T model I/O flag.		
externForcingPeriod	0.0E+00	forcing period (s)
externForcingCycle	0.0E+00	period of the cycle (s)
tauThetaClimRelax	0.0E+00	relaxation time scale (s)
tauSaltClimRelax	0.0E+00	relaxation time scale (s)
latBandClimRelax	3.703701E+05	maximum latitude where relaxation applied
usingCartesianGrid	T	Cartesian coordinates flag (true / false)
usingSphericalPolarGrid	F	spherical coordinates flag (true / false)
usingCylindricalGrid	F	spherical coordinates flag (true / false)
Ro_SeaLevel	0.0E+00	r(1) (units of r)
rkSign	-1.0E+00	index orientation relative to vertical coordinate
horiVertRatio	1.0E+00	ratio on units : horizontal - vertical
drC	5.0E+03 at k=1	center cell separation along Z axis (units of r)
drF	1.0E+04 at k=top	cell face separation along Z axis (units of r)
delX	1.234567E+05 at i=east	U-point spacing (m - cartesian, degrees - spherical)
delY	1.234567E+05 at j=1	V-point spacing (m - cartesian, degrees - spherical)
ygOrigin	0.0E+00	South edge Y-axis origin (cartesian: m, spherical: deg.)
xgOrigin	0.0E+00	West edge X-axis origin (cartesian: m, spherical: deg.)
rSphere	6.37E+06	Radius (ignored - cartesian, m - spherical)
xcoord	6.172835E+04 at i=1	P-point X coord (m - cartesian, degrees - spherical)
ycoord	6.172835E+04 at j=1	P-point Y coord (m - cartesian, degrees - spherical)
rcoord	-5.0E+03 at k=1	P-point r coordinate (units of r)
rF	0.0E+00 at k=1	W-interface r coordinate (units of r)
dBdrRef	0.0E+00 at k=top	vertical gradient of reference buoyancy [(m/s/r) ²]

Name	Value	Description
dxF	1.234567E+05 at k=top	dxF(:,1,1) (m - cartesian, degrees - spherical)
dyF	1.234567E+05 at i=east	dyF(:,1,1) (m - cartesian, degrees - spherical)
dxG	1.234567E+05 at i=east	dxG(:,1,1) (m - cartesian, degrees - spherical)
dyG	1.234567E+05 at i=east	dyG(:,1,1) (m - cartesian, degrees - spherical)
dxC	1.234567E+05 at i=east	dxC(:,1,1) (m - cartesian, degrees - spherical)
dyC	1.234567E+05 at i=east	dyC(:,1,1) (m - cartesian, degrees - spherical)
dxV	1.234567E+05 at i=east	dxV(:,1,1) (m - cartesian, degrees - spherical)
dyU	1.234567E+05 at i=east	dyU(:,1,1) (m - cartesian, degrees - spherical)
rA	1.524155E+10 at i=east	rA(:,1,1) (m - cartesian, degrees - spherical)
rAw	1.524155E+10 at k=top	rAw(:,1,1) (m - cartesian, degrees - spherical)
rAs	1.524155E+10 at k=top	rAs(:,1,1) (m - cartesian, degrees - spherical)

Name	Value	Description
tempAdvScheme	2	temp. horiz. advection scheme selector
tempVertAdvScheme	2	temp. vert. advection scheme selector
tempMultiDimAdvec	F	use multi-dim advection method for temp
tempAdamsBashforth	T	use Adams-Bashforth time-stepping for temp
saltAdvScheme	2	salinity horiz. advection scheme selector
saltVertAdvScheme	2	salinity vert. advection scheme selector
saltMultiDimAdvec	F	use multi-dim advection method for salt
saltAdamsBashforth	T	use Adams-Bashforth time-stepping for salt

MITgcm Tutorial Example Experiments

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 4.2](#). The other examples follow the same general structure as the tutorial examples. However, they only include brief instructions in text file README. The examples are located in subdirectories under the directory [verification](#). Each example is briefly described below.

4.1 Barotropic Gyre MITgcm Example

(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 similar to the problems described analytically by Stommel in 1966 [[Sto48](#)] and numerically in Holland et. al [[HL75](#)].

In this experiment the model is configured to represent a rectangular enclosed box of fluid, 1200×1200 km in lateral extent. The fluid is 5 km deep and is forced by a constant in time zonal wind stress, τ_x , that varies sinusoidally in the ‘north-south’ direction. Topologically the grid is Cartesian and the coriolis parameter f is defined according to a mid-latitude beta-plane equation

$$f(y) = f_0 + \beta y \quad (4.1)$$

where y is the distance along the ‘north-south’ axis of the simulated domain. For this experiment f_0 is set to $10^{-4} s^{-1}$ in (4.1) and $\beta = 10^{-11} s^{-1} m^{-1}$.

The sinusoidal wind-stress variations are defined according to

$$\tau_x(y) = \tau_0 \sin\left(\pi \frac{y}{L_y}\right) \quad (4.2)$$

where L_y is the lateral domain extent (1200~km) and τ_0 is set to $0.1 Nm^{-2}$.

Figure 4.1 summarizes the configuration simulated.

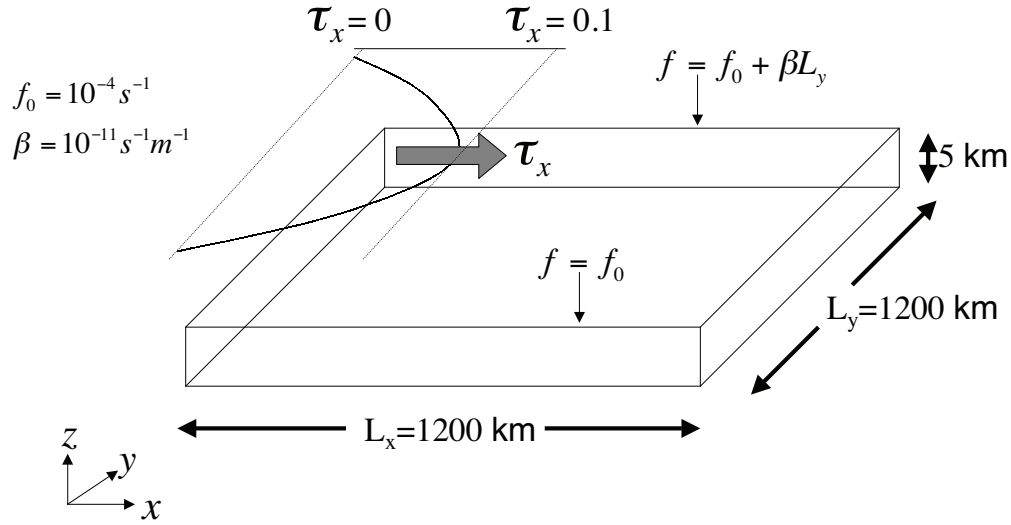


Figure 4.1: Schematic of simulation domain and wind-stress forcing function for barotropic gyre numerical experiment. The domain is enclosed by solid walls at $x = 0, 1200$ km and at $y = 0, 1200$ km.

4.1.1 Equations Solved

The model is configured in hydrostatic form. The implicit free surface form of the pressure equation described in [MHPA97] is employed. A horizontal Laplacian operator ∇_h^2 provides viscous dissipation. The wind-stress momentum input 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 Section 4.1.3), yielding an active set of equations solved in this configuration as follows

$$\begin{aligned} \frac{Du}{Dt} - fv + g\frac{\partial\eta}{\partial x} - A_h\nabla_h^2 u &= \frac{\tau_x}{\rho_0\Delta z} \\ \frac{Dv}{Dt} + fu + g\frac{\partial\eta}{\partial y} - A_h\nabla_h^2 v &= 0 \\ \frac{\partial\eta}{\partial t} + \nabla_h \cdot \vec{u} &= 0 \end{aligned} \tag{4.3}$$

where u and v and the x and y components of the flow vector \vec{u} .

4.1.2 Discrete Numerical Configuration

The domain is discretised with a uniform grid spacing in the horizontal set to $\Delta x = \Delta y = 20$ km, so that there are sixty grid cells in the x and y directions. Vertically the model is configured with a single layer with depth, Δz , of 5000 m.

4.1.2.1 Numerical Stability Criteria

The Laplacian dissipation coefficient, A_h , is set to $400m s^{-1}$. This value is chosen to yield a Munk layer width [Adc95],

$$M_w = \pi \left(\frac{A_h}{\beta} \right)^{\frac{1}{3}} \quad (4.4)$$

of $\approx 100km$. This is greater than the model resolution Δx , ensuring that the frictional boundary layer is well resolved.

The model is stepped forward with a time step $\delta t = 1200$ secs. With this time step the stability parameter to the horizontal Laplacian friction [Adc95]

$$S_l = 4 \frac{A_h \delta t}{\Delta x^2} \quad (4.5)$$

evaluates to 0.012, which is well below the 0.3 upper limit for stability.

The numerical stability for inertial oscillations [Adc95]

$$S_i = f^2 \delta t^2 \quad (4.6)$$

evaluates to 0.0144, which is well below the 0.5 upper limit for stability.

The advective CFL [Adc95] for an extreme maximum horizontal flow speed of $|\vec{u}| = 2m s^{-1}$

$$S_a = \frac{|\vec{u}| \delta t}{\Delta x} \quad (4.7)$$

evaluates to 0.12. This is approaching the stability limit of 0.5 and limits δt to 1200 s.

4.1.3 Code Configuration

The model configuration for this experiment resides under the directory `verification/tutorial_barotropic_gyre/`.

The experiment files

- `input/data`
- `input/data.pkg`
- `input/eedata`
- `input/windx.sin_y`
- `input/topog.box`
- `code/CPP_EEOPTIONS.h`
- `code/CPP_OPTIONS.h`
- `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.

4.1.3.1 File *input/data*

This file, reproduced completely below, specifies the main parameters for the experiment. The parameters that are significant for this configuration are

```
7 viscAh=4.E2,
```

- this line sets the Laplacian friction coefficient to $400m^2s^{-1}$

```
10 beta=1.E-11,
```

- this line sets β (the gradient of the coriolis parameter, f) to $10^{-11}s^{-1}m^{-1}$

```
15 rigidLid=.FALSE.,
```

```
16 implicitFreeSurface=.TRUE.,
```

- these lines suppress the rigid lid formulation of the surface pressure inverter and activate the implicit free surface form of the pressure inverter.

The above was coded like this in rst:

```
.. literalinclude:: ../../../../verification/tutorial_barotropic_gyre/input/data
   :start-at: viscAh=
   :end-at: viscAh=
   :linenos:
   :lineno-match:

- this line sets the Laplacian friction coefficient to :math:`400 m^2s^{-1}`

.. literalinclude:: ../../../../verification/tutorial_barotropic_gyre/input/data
   :start-at: beta=
   :end-at: beta=
   :linenos:
   :lineno-match:

- this line sets :math:`\beta` (the gradient of the coriolis parameter, :math:`f`)
  to :math:`10^{-11} s^{-1}m^{-1}`

.. literalinclude:: ../../../../verification/tutorial_barotropic_gyre/input/data
   :start-at: rigidLid=
   :end-at: implicitFreeSurface=
   :linenos:
   :lineno-match:

- these lines suppress the rigid lid formulation of the surface pressure
  inverter and activate the implicit free surface form of the pressure inverter.
```

...

- Line 27
 - `startTime=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.
- Line 29
 - `endTime=12000,`

- this line indicates that the experiment should start finish at $t = 12000s$. A restart file will be written at this time that will enable the simulation to be continued from this point.
- Line 30
 - `deltaTmom=1200,`
 - This line sets the momentum equation timestep to 1200s.
- Line 39
 - `usingCartesianGrid=.TRUE.,`
 - This line requests that the simulation be performed in a Cartesian coordinate system.
- Line 41
 - `delX=60*20E3,`
 - This line sets the horizontal grid spacing between each x-coordinate line in the discrete grid. The syntax indicates that the discrete grid should be comprise of 60\$ grid lines each separated by 20×10^3m (20 km).
- Line 42
 - `delY=60*20E3,`
 - This line sets the horizontal grid spacing between each y-coordinate line in the discrete grid to 20×10^3m (20 km).
- Line 43
 - `delZ=5000,`
 - This line sets the vertical grid spacing between each z-coordinate line in the discrete grid to 5000m (5 km).
- Line 46
 - `bathyFile='topog.box'`
 - This line specifies the name of the file from which the domain bathymetry is read. This file is a two-dimensional (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. In this experiment, a depth of 0 m indicates a solid wall and a depth of -5000 m indicates open ocean. The matlab program `input/gendata.m` shows an example of how to generate a bathymetry file.
- Line 49
 - `zonalWindFile='windx.sin_y'`
 - This line specifies the name of the file from which the x-direction surface wind stress is read. This file is also a two-dimensional (x, y) map and is enumerated and formatted in the same manner as the bathymetry file. The matlab program `input/gendata.m` includes example code to generate a valid `zonalWindFile` file.

other lines in the file `input/data` are standard values that are described in the MITgcm Getting Started and MITgcm Parameters notes.

Listing 4.1: verification/tutorial_barotropic_gyre/input/data

```

1 # Model parameters
2 # Continuous equation parameters
3 &PARM01
4 tRef=20.,
```

(continues on next page)

(continued from previous page)

```

5  sRef=10.,
6  viscAz=1.E-2,
7  viscAh=4.E2,
8  diffKhT=4.E2,
9  diffKzT=1.E-2,
10 beta=1.E-11,
11 tAlpha=2.E-4,
12 sBeta =0.,
13 gravity=9.81,
14 gBaro=9.81,
15 rigidLid=.FALSE.,
16 implicitFreeSurface=.TRUE.,
17 eosType='LINEAR',
18 readBinaryPrec=64,
19 &
20
21 # Elliptic solver parameters
22 &PARM02
23 cg2dMaxIters=1000,
24 cg2dTargetResidual=1.E-7,
25 &
26
27 # Time stepping parameters
28 &PARM03
29 startTime=0,
30 #endTime=311040000,
31 endTime=12000.0,
32 deltaTmom=1200.0,
33 deltaTtracer=1200.0,
34 abEps=0.1,
35 pChkptFreq=2592000.0,
36 chkptFreq=120000.0,
37 dumpFreq=2592000.0,
38 monitorSelect=2,
39 monitorFreq=1.,
40 &
41
42 # Gridding parameters
43 &PARM04
44 usingCartesianGrid=.TRUE.,
45 usingSphericalPolarGrid=.FALSE.,
46 delX=60*20E3,
47 delY=60*20E3,
48 delZ=5000.,
49 &
50
51 # Input datasets
52 &PARM05
53 bathyFile='topog.box',
54 hydrogThetaFile=,
55 hydrogSaltFile=,
56 zonalWindFile='windx.sin_y',
57 meridWindFile=,
58 &

```


4.1.3.2 File *input/data.pkg*

This file uses standard default values and does not contain customizations for this experiment.

4.1.3.3 File *input/eedata*

This file uses standard default values and does not contain customizations for this experiment.

4.1.3.4 File *input/windx.sin_y*

The *input/windx.sin_y* file specifies a two-dimensional (x, y) map of wind stress, τ_x , values. The units used are Nm^{-2} . Although τ_x is only a function of y in this experiment this file must still define a complete two-dimensional map in order to be compatible with the standard code for loading forcing fields in MITgcm. The included matlab program *input/gendata.m* gives a complete code for creating the *input/windx.sin_y* file.

4.1.3.5 File *input/topog.box*

The *input/topog.box* file specifies a two-dimensional (x, y) map of depth values. For this experiment values are either 0 m or $-delZ$ m, corresponding respectively to a wall or to deep ocean. The file contains a raw binary stream of data that is enumerated in the same way as standard MITgcm two-dimensional, horizontal arrays. The included matlab program *input/gendata.m* gives a complete code for creating the *input/topog.box* file.

4.1.3.6 File *code/SIZE.h*

Two lines are customized in this file for the current experiment

- Line 39
 - $sNx=60$,
 - this line sets the lateral domain extent in grid points for the axis aligned with the x-coordinate.
- Line 40
 - $sNy=60$,
 - this line sets the lateral domain extent in grid points for the axis aligned with the y-coordinate.

Listing 4.2: verification/tutorial_barotropic_gyre/code/SIZE.h

```

1 C
2 C  /=====\
3 C  | SIZE.h Declare size of underlying computational grid. |
4 C  |=====|
5 C  | The design here support a three-dimensional model grid |
6 C  | with indices I,J and K. The three-dimensional domain   |
7 C  | is comprised of nPx*nSx blocks of size sNx along one axis|
8 C  | nPy*nSy blocks of size sNy along another axis and one  |
9 C  | block of size Nz along the final axis.                  |
10 C  | Blocks have overlap regions of size OLx and OLy along the|
11 C  | dimensions that are subdivided.                          |
12 C  \=====/
13 C  Voodoo numbers controlling data layout.
14 C  sNx - No. X points in sub-grid.
15 C  sNy - No. Y points in sub-grid.
```

(continues on next page)

(continued from previous page)

```

16 C      OLx - Overlap extent in X.
17 C      OLy - Overlat extent in Y.
18 C      nSx - No. sub-grids in X.
19 C      nSy - No. sub-grids in Y.
20 C      nPx - No. of processes to use in X.
21 C      nPy - No. of processes to use in Y.
22 C      Nx  - No. points in X for the total domain.
23 C      Ny  - No. points in Y for the total domain.
24 C      Nr  - No. points in R for full process domain.
25 C      INTEGER sNx
26 C      INTEGER sNy
27 C      INTEGER OLx
28 C      INTEGER OLy
29 C      INTEGER nSx
30 C      INTEGER nSy
31 C      INTEGER nPx
32 C      INTEGER nPy
33 C      INTEGER Nx
34 C      INTEGER Ny
35 C      INTEGER Nr
36 C      PARAMETER (
37 &          sNx = 30,
38 &          sNy = 30,
39 &          OLx = 2,
40 &          OLy = 2,
41 &          nSx = 2,
42 &          nSy = 2,
43 &          nPx = 1,
44 &          nPy = 1,
45 &          Nx  = sNx*nSx*nPx,
46 &          Ny  = sNy*nSy*nPy,
47 &          Nr  = 1)
48
49 C      MAX_OLX - Set to the maximum overlap region size of any array
50 C      MAX_OLY that will be exchanged. Controls the sizing of exch
51 C      routine buufers.
52 C      INTEGER MAX_OLX
53 C      INTEGER MAX_OLY
54 C      PARAMETER ( MAX_OLX = OLx,
55 &          MAX_OLY = OLy )
56

```

4.1.3.7 File *code/CPP_OPTIONS.h*

This file uses standard default values and does not contain customizations for this experiment.

4.1.3.8 File *code/CPP_EEOPTIONS.h*

This file uses standard default values and does not contain customizations for this experiment.

4.2 A Rotating Tank in Cylindrical Coordinates

(in directory: [verification/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 $3^\circ \times 1\text{cm}$ cylindrical grid with twenty-nine vertical levels of 0.5cm each. This is a typical laboratory setup for illustration principles of GFD, as well as for a laboratory data assimilation project.

example illustration from GFD lab here

4.2.1 Equations Solved

4.2.2 Discrete Numerical Configuration

The domain is discretised with a uniform cylindrical grid spacing in the horizontal set to $\Delta a = 1^\circ$ and $\Delta \phi = 3^\circ$, so that there are 120 grid cells in the azimuthal direction and thirty-one grid cells in the radial, representing a tank 62cm 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 twenty-nine layers of uniform 0.5cm thickness.

something about heat flux

4.2.3 Code Configuration

The model configuration for this experiment resides under the directory `verification/rotatingi_tank/`. The experiment files

- `input/data`
- `input/data.pkg`
- `input/eedata`
- `input/bathyPol.bin`
- `input/thetaPol.bin`
- `code/CPP/_EEOPTIONS.h`
- `code/CPP/_OPTIONS.h`
- `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.

4.2.3.1 File `input/data`

This file, reproduced completely below, specifies the main parameters for the experiment. The parameters that are significant for this configuration are

- Lines 9-10,
 - `viscAh=5.0E-6,`
 - `viscAz=5.0E-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,
 - `diffKhT=2.5E-6,`

- `diffKzT=2.5E-6,`
- `diffKhS=1.0E-6,`
- `diffKzS=1.0E-6,`

These lines set horizontal and vertical diffusion coefficients for temperature and salinity. Similarly to the friction coefficients, the values are a couple of orders of magnitude less than most configurations.

- Line 17, `f0=0.5`, this line sets the coriolis term, and represents a tank spinning at about 2.4 rpm.
- Lines 23 and 24
 - `rigidLid=.TRUE.,`
 - `implicitFreeSurface=.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 a numerical integration from a previously saved state. Instead, the file `thetaPol.bin` will be loaded to initialize the temperature fields as indicated below, and other variables will be initialized to their defaults.

- Line 43,
 - `deltaT=0.1,`

This line sets the integration timestep to `0.1s`. 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 56,
 - `usingCylindricalGrid=.TRUE.,`

This line requests that the simulation be performed in a cylindrical coordinate system.

- Line 57,
 - `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 58,
 - `dYspacing=0.01,`

This line sets the radial cylindrical grid spacing between each `a`-coordinate line in the discrete grid to `1cm`.

- Line 59,
 - `delZ=29*0.005,`

This line sets the vertical grid spacing between each of 29 `z`-coordinate lines in the discrete grid to `0.005m` (`5mm`).

- Line 64,
 - `bathyFile='bathyPol.bin',`

This line specifies the name of the file from which the domain ‘bathymetry’ (tank depth) is read. This file is a two-dimensional (a, ϕ) 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 ϕ 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 $0m$ indicates an area outside of the tank and a depth of $-0.145m$ indicates the tank itself.

- Line 65,
– `hydrogThetaFile='thetaPol.bin'`,

This line specifies the name of the file from which the initial values of temperature are read. This file is a three-dimensional (x, y, z) map and is enumerated and formatted in the same manner as the bathymetry file.

- Lines 66 and 67
– `tCylIn = 0`
– `tCylOut = 20`

These lines 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 `input/data` are standard values that are described in the MITgcm Getting Started and MITgcm Parameters notes.

Listing 4.3: `verification/rotating_tank/input/data`

```

1  # =====
2  # | Model parameters |
3  # =====
4  #
5  # Continuous equation parameters
6  &PARM01
7  tRef=29*20.0,
8  sRef=29*35.0,
9  viscAh=5.0E-6,
10 viscAz=5.0E-6,
11 no_slip_sides=.FALSE.,
12 no_slip_bottom=.FALSE.,
13 diffKhT=2.5E-6,
14 diffKzT=2.5E-6,
15 diffKhS=1.0E-6,
16 diffKzS=1.0E-6,
17 f0=0.5,
18 eosType='LINEAR',
19 sBeta =0.,
20 gravity=9.81,
21 rhoConst=1000.0,
22 rhoNil=1000.0,
23 #heatCapacity_Cp=3900.0,
24 rigidLid=.TRUE.,
25 implicitFreeSurface=.FALSE.,
26 nonHydrostatic=.TRUE.,
27 readBinaryPrec=32,
28 &
29
30 # Elliptic solver parameters
31 &PARM02
32 cg2dMaxIters=1000,

```

(continues on next page)

(continued from previous page)

```
33 cg2dTargetResidual=1.E-7,
34 cg3dMaxIters=10,
35 cg3dTargetResidual=1.E-9,
36 &
37
38 # Time stepping parameters
39 &PARM03
40 nIter0=0,
41 nTimeSteps=20,
42 #nTimeSteps=36000000,
43 deltaT=0.1,
44 abEps=0.1,
45 pChkptFreq=2.0,
46 #chkptFreq=2.0,
47 dumpFreq=2.0,
48 monitorSelect=2,
49 monitorFreq=0.1,
50 &
51
52 # Gridding parameters
53 &PARM04
54 usingCylindricalGrid=.TRUE.,
55 dXspacing=3.,
56 dYspacing=0.01,
57 delZ=29*0.005,
58 ygOrigin=0.07,
59 &
60
61 # Input datasets
62 &PARM05
63 hydrogThetaFile='thetaPolR.bin',
64 bathyFile='bathyPolR.bin',
65 tCylIn = 0.,
66 tCylOut = 20.,
67 &
```

4.2.3.2 File *input/data.pkg*

This file uses standard default values and does not contain customizations for this experiment.

4.2.3.3 File *input/eedata*

This file uses standard default values and does not contain customizations for this experiment.

4.2.3.4 File *input/thetaPol.bin*

The {it input/thetaPol.bin} file specifies a three-dimensional (x,y,z) map of initial values of θ in degrees Celsius. This particular experiment is set to random values x around 20C to provide initial perturbations.

4.2.3.5 File *input/bathyPol.bin*

The {it input/bathyPol.bin} file specifies a two-dimensional (x,y) map of depth values. For this experiment values are either $0m$ 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 two-dimensional, horizontal arrays.

4.2.3.6 File *code/SIZE.h*

Two lines are customized in this file for the current experiment

- Line 39, - *sNx=120*,

this line sets the lateral domain extent in grid points for the axis aligned with the x-coordinate.

- Line 40, - *sNy=31*,

this line sets the lateral domain extent in grid points for the axis aligned with the y-coordinate.

Listing 4.4: *verification/rotating_tank/code/SIZE.h*

```

1 C
2 C      /=====\
3 C      | SIZE.h Declare size of underlying computational grid. |
4 C      |=====|
5 C      | The design here support a three-dimensional model grid |
6 C      | with indices I,J and K. The three-dimensional domain   |
7 C      | is comprised of nPx*nSx blocks of size sNx along one axis|
8 C      | nPy*nSy blocks of size sNy along another axis and one  |
9 C      | block of size Nz along the final axis.                  |
10 C      | Blocks have overlap regions of size OLx and OLy along the|
11 C      | dimensions that are subdivided.                          |
12 C      \=====/
13 C      Voodoo numbers controlling data layout.
14 C      sNx - No. X points in sub-grid.
15 C      sNy - No. Y points in sub-grid.
16 C      OLx - Overlap extent in X.
17 C      OLy - Overlat extent in Y.
18 C      nSx - No. sub-grids in X.
19 C      nSy - No. sub-grids in Y.
20 C      nPx - No. of processes to use in X.
21 C      nPy - No. of processes to use in Y.
22 C      Nx  - No. points in X for the total domain.
23 C      Ny  - No. points in Y for the total domain.
24 C      Nr  - No. points in Z for full process domain.
25 C      INTEGER sNx
26 C      INTEGER sNy
27 C      INTEGER OLx
28 C      INTEGER OLy
29 C      INTEGER nSx
30 C      INTEGER nSy
31 C      INTEGER nPx
32 C      INTEGER nPy
33 C      INTEGER Nx
34 C      INTEGER Ny
35 C      INTEGER Nr
36 C      PARAMETER (
37 &          sNx = 30,
38 &          sNy = 23,
39 &          OLx = 3,
40 &          OLy = 3,
41 &          nSx = 4,
42 &          nSy = 1,
```

(continues on next page)

(continued from previous page)

```
43      &          nPx = 1,
44      &          nPy = 1,
45      &          Nx = sNx*nSx*nPx,
46      &          Ny = sNy*nSy*nPy,
47      &          Nr = 29)
48
49 C      MAX_OLX - Set to the maximum overlap region size of any array
50 C      MAX_OLY that will be exchanged. Controls the sizing of exch
51 C      routine buufers.
52      INTEGER MAX_OLX
53      INTEGER MAX_OLY
54      PARAMETER ( MAX_OLX = OLx,
55      &          MAX_OLY = OLy )
56
```

4.2.3.7 File *code/CPP_OPTIONS.h*

This file uses standard default values and does not contain customizations for this experiment.

4.2.3.8 File *code/CPP_EEOPTIONS.h*

This file uses standard default values and does not contain customizations for this experiment.

Contributing to the MITgcm

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 5.4](#).

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/user_name/MITgcm.git
```

3. Move into your local clone directory (cd MITgcm) 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, here we call the new feature branch newfeature) and make edits on this branch:

```
% git fetch upstream  
% git checkout -b newfeature 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 newfeature
```

7. On GitHub, go to your fork and hit the pull request (PR) button, 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. A response may accept the changes, 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 and Oliver Jahn.

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 a development branch (here, newfeature), not the master branch.

```
% git checkout master  
% git pull upstream master  
% git push origin master  
% git checkout newfeature  
% 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://akrobat.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 sourcecode “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.

A detailed explanation of steps for contributing MITgcm repository edits:

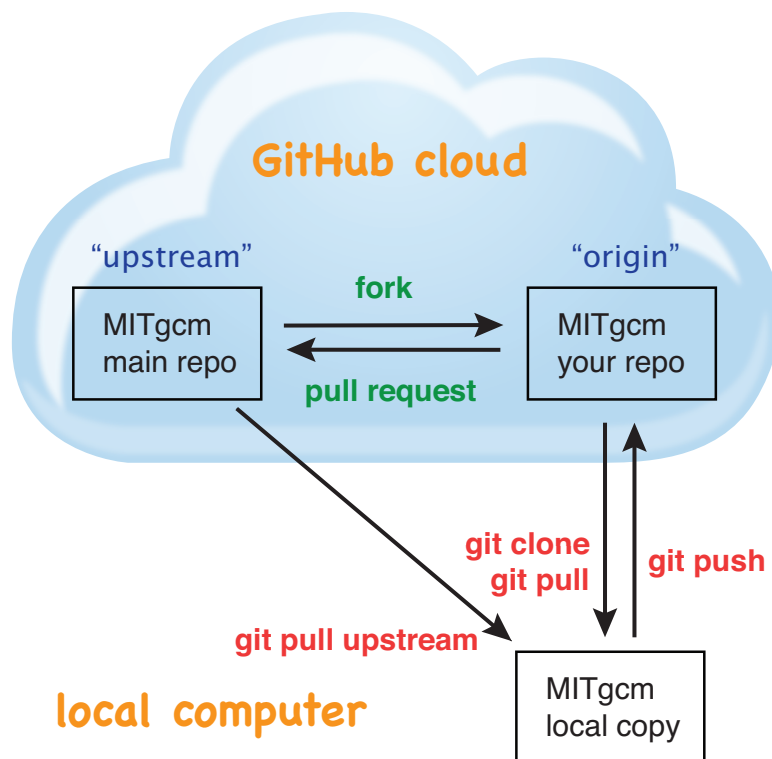


Figure 5.1: A conceptual map of the GitHub setup. Git terminal commands are shown in red, GitHub commands are shown in green.

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:

```
% git clone https://github.com/your_github_user_name/MITgcm.git
```

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:

```
% cd MITgcm
```

We need to set up a remote that points to the main repository:

```
% 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.

```
% git fetch upstream
% git checkout -b newfeature 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, which we name `newfeature`, from the `upstream/master` branch, it also switches you onto this newly created branch. Naming the branch something more descriptive than ‘newfeature’ 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, then 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.
- if you make any change to the code, however small, i.e., flavor ii or iii above, we expect you to add your changes to the top of [doc/tag-index](#) (starting at line 4), which is a running history of all development of the MITgcm. Again, be concise, describing your changes in one or several lines of text. We will not accept code changes without this edit.

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.

```
% git push -u origin newfeature
```

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; in fact, it is very likely [doc/tag-index](#) will have been updated in the main repo if even a week has elapsed. 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 (exception would be to [doc/tag-index](#), which the maintainers can easily resolve). 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:

```
% git checkout master
% git pull upstream master
% git push origin master
% git checkout newfeature
% 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/your_github_user_name/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), and flag up any issues. 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). But much more likely than the latter, 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 instead of outright rejection. 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.

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 and Oliver Jahn.

5.3 Coding style guide

Detailed instructions or link to be added.

5.3.1 Automatic testing with Travis-CI

The MITgcm uses the continuous integration service Travis-CI to test code before it is accepted into the repository. When you submit a pull request your contributions will be automatically tested. However, it is a good idea to test before submitting a pull request, so that you have time to fix any issues that are identified. To do this, you will need to activate Travis-CI for your fork of the repository.

Detailed instructions or link to be added.

5.4 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.

5.4.1 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.

5.4.2 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, and using the syntax `:ref:`some_text_to_clickon <LABELNAME>`` for the link.

5.4.3 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 a 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/filename``. 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/filename>`` (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/filename.

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``.

5.4.4 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 5.4.2](#)). 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.

Discuss conversion of .tex files.

5.4.5 Figures

The syntax to insert a figure is as follows:

```

.. figure:: pathname/filename.*
   :width: 80%
   :align: center
   :alt: text description of figure here
   :name: myfigure

```

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 (see [here](#) for more info on compatible formats).

`: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:`myfigure``.

Note the indentation and line spacing employed above.

5.4.6 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. |
+-----+-----+-----+

```

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 span rows.	<ul style="list-style-type: none"> • Cells • contain • blocks.
body row 4		

Simple Table Example:

```

=====
Inputs      Output
-----
A           B      A or B

```

(continues on next page)

(continued from previous page)

```

=====
False  False  False
True   False  True
False  True   True
True   True   True
=====

```

Inputs		Output
A	B	A or B
False	False	False
True	False	True
False	True	True
True	True	True

Note that the spacing of your tables in your `.rst` file(s) will not match the generated output; rather, when you build the final output, the `rst` builder (Sphinx) will determine how wide the columns need to be and space them appropriately.

5.4.7 Other text blocks

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) A splashier 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](#).

5.4.8 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.

Alternatively, latex `:math:` directives (see [above](#)) may also be used to display units, using the `\text{}` syntax to display non-italic characters.

- double quotes for inline literal computer command, variables, syntax etc.
- discuss how to break up sections into smaller files
- discuss | lines

5.4.9 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 modules installed (use `pip install MODULE` in the terminal):

- sphinx
- sphinxcontrib-bibtex
- sphinx_rtd_theme

Then, run `make html` in the `docs` directory.

5.5 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.

To test pull requests locally you should download the pull request branch. You can do this either by cloning the branch from the pull request:

```
git clone -b BRANCHNAME https://github.com/USERNAME/MITgcm.git
```

where *USERNAME* is replaced by the username of the person proposing the pull request, and *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. Move directories in to your local repository and then

```
git remote add USERNAME https://github.com/USERNAME/MITgcm.git
```

where *USERNAME* is replaced by the user name of the person who has made the pull request. Then download the branch from the pull request

```
git fetch USERNAME
```

and switch to the desired branch

```
git checkout --track USERNAME/foo
```

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.

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.

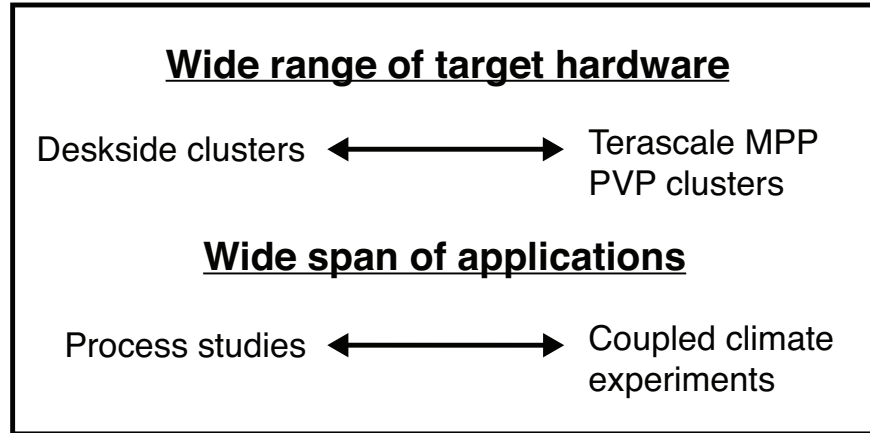


Figure 6.1: The MITgcm architecture is designed to allow simulation of a wide range of physical problems on a wide range of hardware. The computational resource requirements of the applications targeted range from around 10^7 bytes (≈ 10 megabytes) of memory to 10^{11} bytes (≈ 100 gigabytes). Arithmetic operation counts for the applications of interest range from 10^9 floating point operations to more than 10^{17} floating point operations.

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.

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

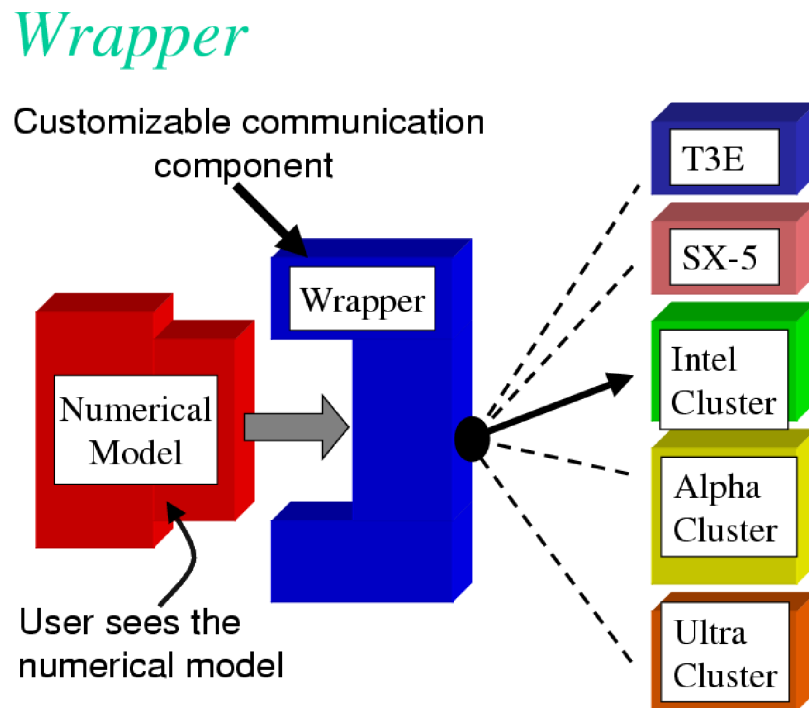


Figure 6.2: Numerical code is written to fit within a software support infrastructure called WRAPPER. The WRAPPER is portable and can be specialized for a wide range of specific target hardware and programming environments, without impacting numerical code that fits within the WRAPPER. Codes that fit within the WRAPPER can generally be made to run as fast on a particular platform as codes specially optimized for that platform.

competitively with equivalent numerical code that has been modified to contain native optimizations for a particular system (see Hoe et al. 1999) [HHA99].

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.

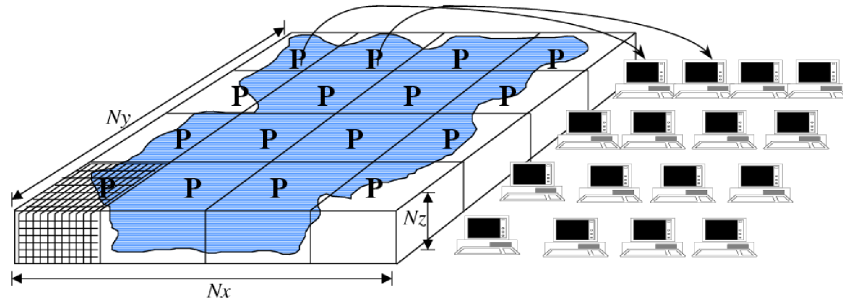


Figure 6.3: The WRAPPER provides support for one and two dimensional decompositions of grid-point domains. The figure shows a hypothetical domain of total size $N_x N_y N_z$. This hypothetical domain is decomposed in two-dimensions along the N_x and N_y directions. The resulting tiles are owned by different 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.

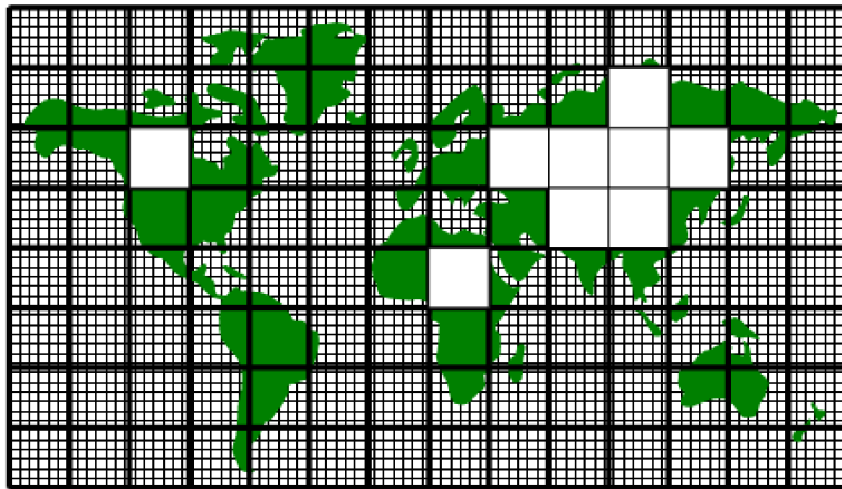


Figure 6.4: A global grid subdivided into tiles. Tiles contain a interior region and an overlap region. Overlap regions are periodically updated from neighboring tiles.

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.

CPU1		CPU2
====		====
a(3) = 8		WHILE (a(3) .NE. 8)
		WAIT
		END WHILE

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).

CPU1		CPU2
====		====
a(3) = 8		WHILE (a(3) .NE. 8)
CALL SEND(CPU2, a(3))		CALL RECV(CPU1, a(3))
		END WHILE

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.

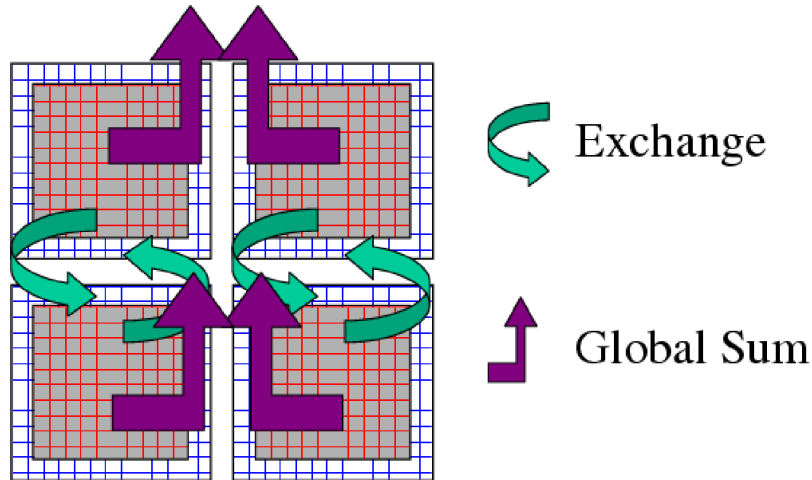


Figure 6.5: Three performance critical parallel primitives are provided by the WRAPPER. These primitives are always used to communicate data between tiles. The figure shows four tiles. The curved arrows indicate exchange primitives which transfer data between the overlap regions at tile edges and interior 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.

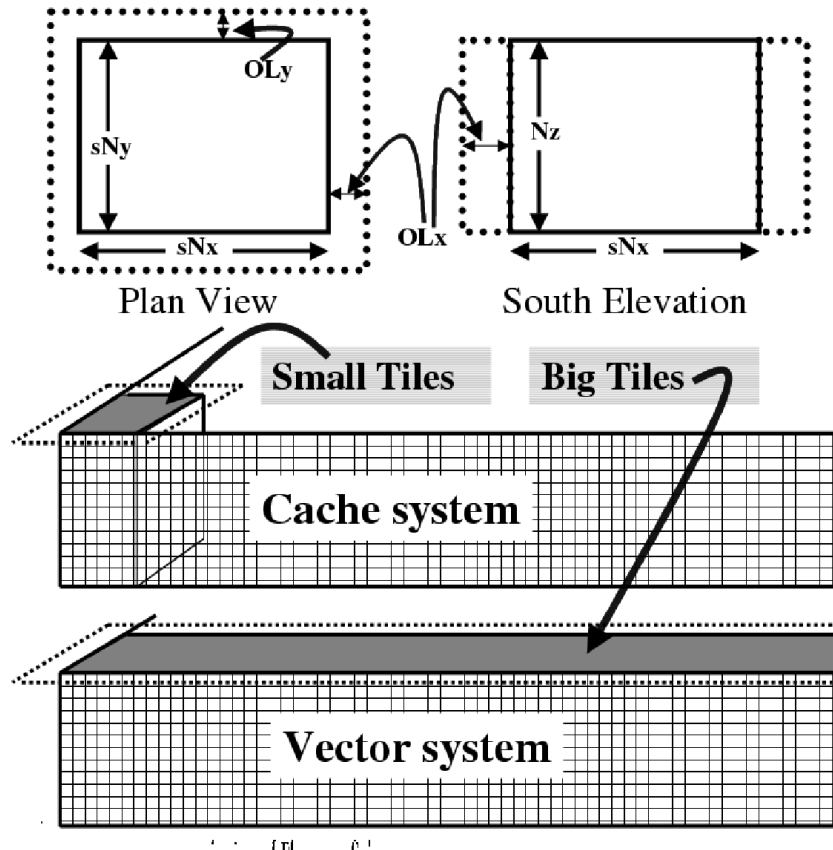


Figure 6.6: The tiling strategy that the WRAPPER supports allows tiles to be shaped to suit the underlying system memory architecture. Compact tiles that lead to greater memory reuse can be used on cache based systems (upper half of figure) with deep memory hierarchies, whereas long tiles with large inner loops can be used to exploit vector systems having highly pipelined memory systems.

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.

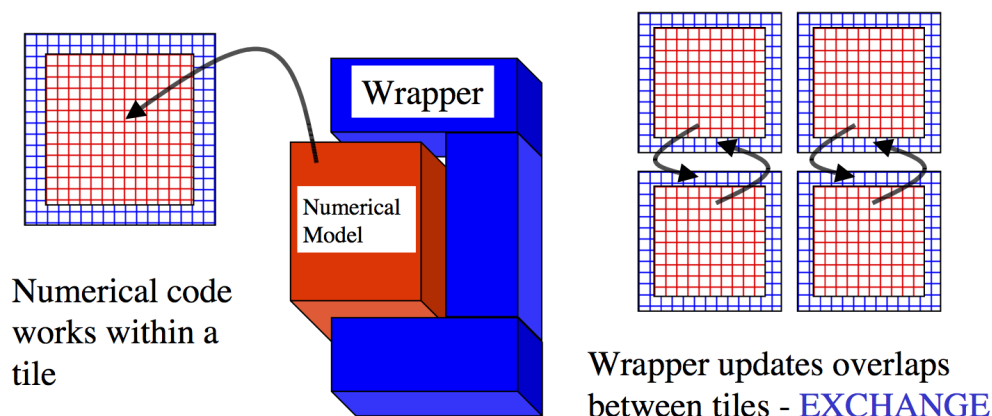


Figure 6.7: Summary of the WRAPPER machine model.

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`, `sNy`

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 `sNy` 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 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)
```

(continues on next page)

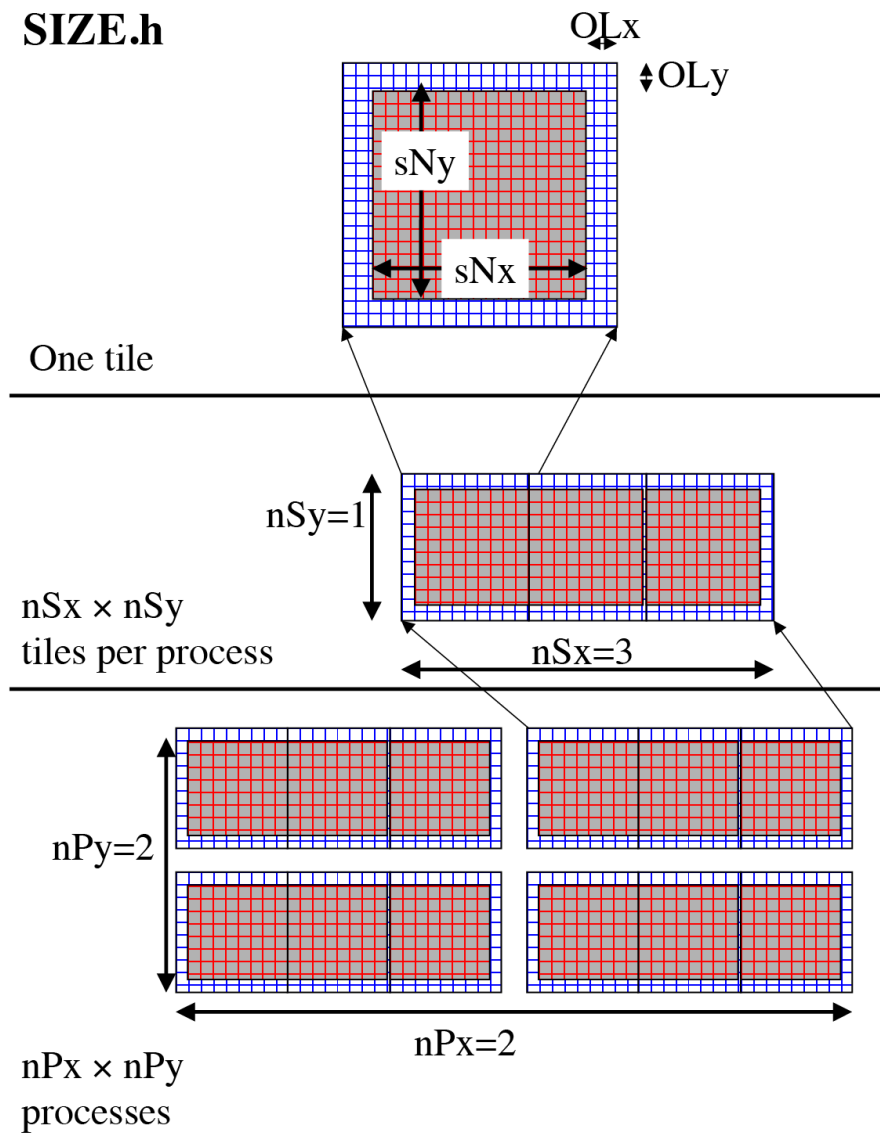


Figure 6.8: The three level domain decomposition hierarchy employed by the WRAPPER. A domain is composed of tiles. Multiple tiles can be allocated to a single process. Multiple processes can exist, each with multiple tiles. Tiles within a process can be spread over multiple compute threads.

(continued from previous page)

```

:
  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, `nSy` 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 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 two-dimensional implicit elliptic solver:

```

REAL*8  cg2d_r(1-OLx:sNx+OLx,1-OLy:sNy+OLy,nSx,nSy)
REAL*8  err
:
:
  other computations
:
:
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
CALL GLOBAL_SUM_R8( err , myThid )
err = SQRT(err)

```

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. For example the evaluation of all the momentum equation prognostic terms (see *dynamics.F*) is 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

```
Nx = sNx*nSx*nPx
Ny = sNy*nSy*nPy
```

```
3.  PARAMETER (
&          sNx = 90,
&          sNy = 10,
&          OLx = 3,
&          OLy = 3,
&          nSx = 1,
&          nSy = 2,
&          nPx = 1,
&          nPy = 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 (`nPy=2`) each of which has two separate sub-domains `nSy=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 (
&      sNx = 32,
&      sNy = 32,
&      OLx = 3,
&      OLy = 3,
&      nSx = 6,
&      nSy = 1,
&      nPx = 1,
&      nPy = 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 ($nSx=6$). This set of values can be used for a cube sphere calculation. Each tile of size 32×32 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_MINMAL  :: Minimal startup. Just enough to
| |                   allow basic I/O.
| |-- EEINTRO_MSG    :: Write startup greeting.
| |
| |-- EESET_PARMS    :: Set WRAPPER parameters
| |
| |-- EEWRITE_EEENV  :: 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

```

The steps above preceeds 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

```
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 `ALLOW_USE_MPI` and `ALWAYS_USE_MPI` flags in the `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

```
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 `SIZE.h`. The option `-machinefile mf` specifies that a text file called `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 `NNNN` 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--  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`.

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 `exch-NeedsMemSync` 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 `eeboot.F`, `CPP_EEMACROS.h` and `cg2d.F`. When the flag is defined it 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_MINMAL  :: 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
C Invocation from WRAPPER level...
C :
C :
C |
C |--THE_MODEL_MAIN :: Primary driver for the MITgcm algorithm
C |                 :: Called from WRAPPER level numerical
C |                 :: code invocation routine. On entry

```

(continues on next page)

(continued from previous page)

```

C      |           :: to THE_MODEL_MAIN separate thread and
C      |           :: 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
C      |           :: specific region in decomposed discrete space.
C      |
C      | -INITIALISE_FIXED :: Set fixed model arrays such as topography,
C      | |                 :: grid, solver matrices etc..
C      | |
C      | | -INI_PARMS :: Routine to set kernel model parameters.
C      | |           :: By default kernel parameters are read from file
C      | |           :: "data" in directory in which code executes.
C      | |
C      | | -MON_INIT :: Initializes monitor package ( see pkg/monitor )
C      | |
C      | | -INI_GRID :: Control grid array (vert. and hori.) initialization.
C      | | |         :: Grid arrays are held and described in GRID.h.
C      | | |
C      | | | -INI_VERTICAL_GRID      :: Initialize vertical grid arrays.
C      | | |
C      | | | -INI_CARTESIAN_GRID     :: Cartesian horiz. grid initialization
C      | | | |                     :: (calculate grid from kernel parameters).
C      | | |
C      | | | -INI_SPHERICAL_POLAR_GRID :: Spherical polar horiz. grid
C      | | | |                     :: initialization (calculate grid from
C      | | | |                     :: kernel parameters).
C      | | |
C      | | | -INI_CURVILINEAR_GRID  :: General orthogonal, structured horiz.
C      | | | |                     :: grid initializations. ( input from raw
C      | | | |                     :: grid files, LONC.bin, DXF.bin etc... )
C      | | |
C      | | | -INI_DEPTHS           :: Read (from "bathyFile") or set bathymetry/orgography.
C      | | |
C      | | | -INI_MASKS_ETC :: Derive horizontal and vertical cell fractions and
C      | | | |             :: land masking for solid-fluid boundaries.
C      | | |
C      | | | -INI_LINEAR_PHSURF :: Set ref. surface Bo_surf
C      | | |
C      | | | -INI_CORI             :: Set coriolis term. zero, f-plane, beta-plane,
C      | | | |                     :: sphere options are coded.
C      | | |
C      | | | -PACAKGES_BOOT        :: Start up the optional package environment.
C      | | | |                     :: Runtime selection of active packages.
C      | | |
C      | | | -PACKAGES_READPARMS :: Call active package internal parameter load.
C      | | | |
C      | | | | -GMREDI_READPARMS    :: GM Package. see pkg/gmredi
C      | | | | -KPP_READPARMS      :: KPP Package. see pkg/kpp
C      | | | | -SHAP_FILT_READPARMS :: Shapiro filter package. see pkg/shap_filt
C      | | | | -OBCS_READPARMS     :: Open bndy package. see pkg/obcs
C      | | | | -AIM_READPARMS      :: Intermediate Atmos. pacakage. see pkg/aim
C      | | | | -COST_READPARMS     :: Cost function package. see pkg/cost
C      | | | | -CTRL_INIT          :: Control vector support package. see pkg/ctrl
C      | | | | -OPTIM_READPARMS    :: Optimisation support package. see pkg/ctrl
C      | | | | -GRDCHK_READPARMS   :: Gradient check package. see pkg/grdchk
C      | | | | -ECCO_READPARMS     :: ECCO Support Package. see pkg/ecco
C      | | | | -PTRACERS_READPARMS :: multiple tracer package, see pkg/ptracers

```

(continues on next page)

(continued from previous page)

```

C | | |-GCHEM_READPARMS      :: tracer interface package, see pkg/gchem
C | |
C | | |-PACKAGES_CHECK
C | | |
C | | |-KPP_CHECK            :: KPP Package. pkg/kpp
C | | |-OBCS_CHECK           :: Open bndy Pacakge. pkg/obcs
C | | |-GMREDI_CHECK         :: GM Package. pkg/gmredi
C | |
C | | |-PACKAGES_INIT_FIXED
C | | | |-OBCS_INIT_FIXED    :: Open bndy Package. see pkg/obcs
C | | | |-FLT_INIT           :: Floats Package. see pkg/flt
C | | | |-GCHEM_INIT_FIXED   :: tracer interface package, see pkg/gchem
C | |
C | | |-ZONAL_FILT_INIT      :: FFT filter Package. see pkg/zonal_filt
C | |
C | | |-INI_CG2D              :: 2d con. grad solver initialization.
C | |
C | | |-INI_CG3D              :: 3d con. grad solver initialization.
C | |
C | | |-CONFIG_SUMMARY       :: Provide synopsis of kernel setup.
C | |                       :: Includes annotated table of kernel
C | |                       :: parameter settings.
C |
C | |-CTRL_UNPACK :: Control vector support package. see pkg/ctrl
C |
C | |-ADTHE_MAIN_LOOP :: Derivative evaluating form of main time stepping loop
C | !               :: Auotmatically generated by TAMC/TAF.
C |
C | |-CTRL_PACK      :: Control vector support package. see pkg/ctrl
C |
C | |-GRDCHK_MAIN    :: Gradient check package. see pkg/grdchk
C |
C | |-THE_MAIN_LOOP  :: Main timestepping loop routine.
C | |
C | | |-INITIALISE_VARIA :: Set the initial conditions for time evolving
C | | |                   :: variables
C | | |
C | | | |-INI_LINEAR_PHISURF :: Set ref. surface Bo_surf
C | | |
C | | | |-INI_CORI           :: Set coriolis term. zero, f-plane, beta-plane,
C | | |                   :: sphere options are coded.
C | | |
C | | | |-INI_CG2D           :: 2d con. grad solver initialization.
C | | | |-INI_CG3D           :: 3d con. grad solver initialization.
C | | | |-INI_MIXING         :: Initialize diapycnal diffusivity.
C | | | |-INI_DYNVARS        :: Initialize to zero all DYNVARS.h arrays (dynamical
C | | |                   :: fields).
C | | |
C | | | |-INI_FIELDS        :: Control initializing model fields to non-zero
C | | | | |-INI_VEL          :: Initialize 3D flow field.
C | | | | |-INI_THETA        :: Set model initial temperature field.
C | | | | |-INI_SALT          :: Set model initial salinity field.
C | | | | |-INI_PSURF        :: Set model initial free-surface height/pressure.
C | | | | |-INI_PRESSURE     :: Compute model initial hydrostatic pressure
C | | | | |-READ_CHECKPOINT  :: Read the checkpoint
C | | |
C | | |
C | | | |-THE_CORRECTION_STEP :: Step forward to next time step.

```

(continues on next page)

(continued from previous page)

```

C      | | | | :: Here applied to move restart conditions
C      | | | | :: (saved in mid timestep) to correct level in
C      | | | | :: time (only used for pre-c35).
C      | | | |
C      | | | |-CALC_GRAD_PHI_SURF :: Return DDx and DDy of surface pressure
C      | | | |-CORRECTION_STEP   :: Pressure correction to momentum
C      | | | |-CYCLE_TRACER      :: Move tracers forward in time.
C      | | | |-OB_CS_APPLY       :: Open bndy package. see pkg/obcs
C      | | | |-SHAP_FILT_APPLY    :: Shapiro filter package. see pkg/shap_filt
C      | | | |-ZONAL_FILT_APPLY   :: FFT filter package. see pkg/zonal_filt
C      | | | |-CONVECTIVE_ADJUSTMENT :: Control static instability mixing.
C      | | | | |-FIND_RHO        :: Find adjacent densities.
C      | | | | |-CONVECT        :: Mix static instability.
C      | | | | |-TIMEAVE_CUMULATE :: Update convection statistics.
C      | | | |
C      | | | |-CALC_EXACT_ETA      :: Change SSH to flow divergence.
C      | | | |
C      | | | |-CONVECTIVE_ADJUSTMENT_INI :: Control static instability mixing
C      | | | | :: Extra time history interactions.
C      | | | |
C      | | | | |-FIND_RHO        :: Find adjacent densities.
C      | | | | |-CONVECT        :: Mix static instability.
C      | | | | |-TIMEAVE_CUMULATE :: Update convection statistics.
C      | | | |
C      | | | |-PACKAGES_INIT_VARIABLES :: Does initialization of time evolving
C      | | | | :: package data.
C      | | | |
C      | | | | |-GMREDI_INIT      :: GM package. ( see pkg/gmredi )
C      | | | | |-KPP_INIT        :: KPP package. ( see pkg/kpp )
C      | | | | |-KPP_OPEN_DIAGS
C      | | | | |-OB_CS_INIT_VARIABLES :: Open bndy. package. ( see pkg/obcs )
C      | | | | |-PTRACERS_INIT    :: multi. tracer package, (see pkg/ptracers)
C      | | | | |-GCHEM_INIT      :: tracer interface pkg (see pkg/gchem)
C      | | | | |-AIM_INIT        :: Inter. atmos package. ( see pkg/aim )
C      | | | | |-CTRL_MAP_INI    :: Control vector package. ( see pkg/ctrl )
C      | | | | |-COST_INIT       :: Cost function package. ( see pkg/cost )
C      | | | | |-ECCO_INIT       :: ECCO support package. ( see pkg/ecco )
C      | | | | |-INI_FORCING     :: Set model initial forcing fields.
C      | | | | | :: Either set in-line or from file as shown.
C      | | | | |-READ_FLD_XY_RS(zonalWindFile)
C      | | | | |-READ_FLD_XY_RS(meridWindFile)
C      | | | | |-READ_FLD_XY_RS(surfQFile)
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      | | | |
C      | | | |-CALC_SURF_DR      :: Calculate the new surface level thickness.
C      | | | |-UPDATE_SURF_DR    :: Update the surface-level thickness fraction.
C      | | | |-UPDATE_CG2D      :: Update 2d conjugate grad. for Free-Surf.
C      | | | |-STATE_SUMMARY    :: Summarize model prognostic variables.
C      | | | |-TIMEAVE_STATVARS :: Time averaging package ( see pkg/timeave ).
C      | | | |
C      | | | |-WRITE_STATE      :: Controlling routine for IO to dump model state.
C      | | | | |-WRITE_REC_XYZ_RL :: Single file I/O
C      | | | | |-WRITE_FLD_XYZ_RL :: Multi-file I/O
C      | | | |

```

(continues on next page)

(continued from previous page)

```

C      | |-MONITOR           :: Monitor state ( see pkg/monitor )
C      | |-CTRL_MAP_FORCING :: Control vector support package. ( see pkg/ctrl )
C====|>|
C====|>| *****
C====|>| BEGIN MAIN TIMESTEPPING LOOP
C====|>| *****
C====|>|
C/\    | |-FORWARD_STEP      :: Step forward a time-step ( AT LAST !!! )
C/\    | | |
C/\    | | |-DUMMY_IN_STEPPING :: autodiff package ( pkg/autoduff ).
C/\    | | |-CALC_EXACT_ETA :: Change SSH to flow divergence.
C/\    | | |-CALC_SURF_DR    :: Calculate the new surface level thickness.
C/\    | | |-EXF_GETFORCING :: External forcing package. ( pkg/exf )
C/\    | | |-EXTERNAL_FIELDS_LOAD :: Control loading time dep. external data.
C/\    | | | |
C/\    | | | | :: Simple interpolation between end-points
C/\    | | | | :: for forcing datasets.
C/\    | | | |
C/\    | | | |-EXCH :: Sync forcing. in overlap regions.
C/\    | | | |-SEAICE_MODEL :: Compute sea-ice terms. ( pkg/seaice )
C/\    | | | |-FREEZE      :: Limit surface temperature.
C/\    | | | |-GCHEM_FIELD_LOAD :: load tracer forcing fields (pkg/gchem)
C/\    | | | |
C/\    | | | |-THERMODYNAMICS :: theta, salt + tracer equations driver.
C/\    | | | |
C/\    | | | |-INTEGRATE_FOR_W :: Integrate for vertical velocity.
C/\    | | | |-OB_CS_APPLY_W    :: Open bndy. package ( see pkg/obcs ).
C/\    | | | |-FIND_RHO        :: Calculates [rho(S,T,z)-RhoConst] of a slice
C/\    | | | |-GRAD_SIGMA      :: Calculate isoneutral gradients
C/\    | | | |-CALC_IVDC       :: Set Implicit Vertical Diffusivity for Convection
C/\    | | | |
C/\    | | | |-OB_CS_CALC      :: Open bndy. package ( see pkg/obcs ).
C/\    | | | |-EXTERNAL_FORCING_SURF :: Accumulates appropriately dimensioned
C/\    | | | | :: forcing terms.
C/\    | | | |-PTRACERS_FORCING_SURF :: Tracer package ( see pkg/ptracers ).
C/\    | | | |
C/\    | | | |-GMREDI_CALC_TENSOR :: GM package ( see pkg/gmredi ).
C/\    | | | |-GMREDI_CALC_TENSOR_DUMMY :: GM package ( see pkg/gmredi ).
C/\    | | | |-KPP_CALC        :: KPP package ( see pkg/kpp ).
C/\    | | | |-KPP_CALC_DUMMY  :: KPP package ( see pkg/kpp ).
C/\    | | | |-AIM_DO_ATMOS_PHYSICS :: Intermed. atmos package ( see pkg/aim ).
C/\    | | | |-GAD_ADVECTION   :: Generalised advection driver (multi-dim
C/\    | | | | advection case) (see pkg/gad).
C/\    | | | |-CALC_COMMON_FACTORS :: Calculate common data (such as volume flux)
C/\    | | | |-CALC_DIFFUSIVITY :: Calculate net vertical diffusivity
C/\    | | | |
C/\    | | | |-GMREDI_CALC_DIFF :: GM package ( see pkg/gmredi ).
C/\    | | | |-KPP_CALC_DIFF    :: KPP package ( see pkg/kpp ).
C/\    | | | |
C/\    | | | |-CALC_GT          :: Calculate the temperature tendency terms
C/\    | | | |
C/\    | | | |-GAD_CALC_RHS     :: Generalised advection package
C/\    | | | | :: ( see pkg/gad )
C/\    | | | |-KPP_TRANSPORT_T :: KPP non-local transport ( see pkg/kpp ).
C/\    | | | |
C/\    | | | |-EXTERNAL_FORCING_T :: Problem specific forcing for temperature.
C/\    | | | |-ADAMS_BASHFORTH2 :: Extrapolate tendencies forward in time.
C/\    | | | |-FREESURF_RESCALE_G :: Re-scale Gt for free-surface height.

```

(continues on next page)

(continued from previous page)

C/\					
C/\				-TIMESTEP_TRACER	:: Step tracer field forward in time
C/\					
C/\				-CALC_GS	:: Calculate the salinity tendency terms
C/\					
C/\				-GAD_CALC_RHS	:: Generalised advection package
C/\					:: (see pkg/gad)
C/\				-KPP_TRANSPORT_S	:: KPP non-local transport (see pkg/kpp).
C/\					
C/\				-EXTERNAL_FORCING_S	:: Problem specific forcing for salt.
C/\				-ADAMS_BASHFORTH2	:: Extrapolate tendencies forward in time.
C/\				-FREESURF_RESCALE_G	:: Re-scale Gs for free-surface height.
C/\					
C/\				-TIMESTEP_TRACER	:: Step tracer field forward in time
C/\					
C/\				-TIMESTEP_TRACER	:: Step tracer field forward in time
C/\					
C/\				-PTRACERS_INTEGRATE	:: Integrate other tracer(s) (see pkg/ptracers).
C/\					
C/\				-GAD_CALC_RHS	:: Generalised advection package
C/\					:: (see pkg/gad)
C/\				-KPP_TRANSPORT_PTR	:: KPP non-local transport (see pkg/kpp).
C/\					
C/\				-PTRACERS_FORCING	:: Problem specific forcing for tracer.
C/\				-GCHEM_FORCING_INT	:: tracer forcing for gchem pkg (if all
C/\					tendency terms calculated together)
C/\				-ADAMS_BASHFORTH2	:: Extrapolate tendencies forward in time.
C/\				-FREESURF_RESCALE_G	:: Re-scale Gs for free-surface height.
C/\				-TIMESTEP_TRACER	:: Step tracer field forward in time
C/\					
C/\				-OB_CS_APPLY_TS	:: Open bndy. package (see pkg/obcs).
C/\					
C/\				-IMPLDIFF	:: Solve vertical implicit diffusion equation.
C/\				-OB_CS_APPLY_TS	:: Open bndy. package (see pkg/obcs).
C/\					
C/\				-AIM_AIM2DYN_EXCHANGES	:: Inetermed. atmos (see pkg/aim).
C/\				-EXCH	:: Update overlaps
C/\					
C/\				-DYNAMICS	:: Momentum equations driver.
C/\					
C/\				-CALC_GRAD_PHI_SURF	:: Calculate the gradient of the surface
C/\					Potential anomaly.
C/\				-CALC_VISCOSITY	:: Calculate net vertical viscosity
C/\				-KPP_CALC_VISC	:: KPP package (see pkg/kpp).
C/\					
C/\				-CALC_PHI_HYD	:: Integrate the hydrostatic relation.
C/\				-MOM_FLUXFORM	:: Flux form mom eqn. package (see
C/\					pkg/mom_fluxform).
C/\				-MOM_VECINV	:: Vector invariant form mom eqn. package (see
C/\					pkg/mom_vecinv).
C/\				-TIMESTEP	:: Step momentum fields forward in time
C/\				-OB_CS_APPLY_UV	:: Open bndy. package (see pkg/obcs).
C/\					
C/\				-IMPLDIFF	:: Solve vertical implicit diffusion equation.
C/\				-OB_CS_APPLY_UV	:: Open bndy. package (see pkg/obcs).
C/\					
C/\				-TIMEAVE_CUMUL_1T	:: Time averaging package (see pkg/timeave).

(continues on next page)

(continued from previous page)

```

C/\ | | | |-TIMEAVE_CUMUATE      :: Time averaging package ( see pkg/timeave ).
C/\ | | | |-DEBUG_STATS_RL      :: Quick debug package ( see pkg/debug ).
C/\ | | |
C/\ | | | |-CALC_GW              :: vert. momentum tendency terms ( NH, QH only ).
C/\ | | |
C/\ | | | |-UPDATE_SURF_DR      :: Update the surface-level thickness fraction.
C/\ | | |
C/\ | | | |-UPDATE_CG2D         :: Update 2d conjugate grad. for Free-Surf.
C/\ | | |
C/\ | | | |-SOLVE_FOR_PRESSURE   :: Find surface pressure.
C/\ | | | | |-CALC_DIV_GHAT      :: Form the RHS of the surface pressure eqn.
C/\ | | | | |-CG2D              :: Two-dim pre-con. conjugate-gradient.
C/\ | | | | |-CG3D              :: Three-dim pre-con. conjugate-gradient solver.
C/\ | | |
C/\ | | | |-THE_CORRECTION_STEP  :: Step forward to next time step.
C/\ | | | |
C/\ | | | | |-CALC_GRAD_PHI_SURF :: Return DDx and DDy of surface pressure
C/\ | | | | |-CORRECTION_STEP    :: Pressure correction to momentum
C/\ | | | | |-CYCLE_TRACER       :: Move tracers forward in time.
C/\ | | | | |-OBCS_APPLY         :: Open bndy package. see pkg/oecs
C/\ | | | | |-SHAP_FILT_APPLY    :: Shapiro filter package. see pkg/shap_filt
C/\ | | | | |-ZONAL_FILT_APPLY   :: FFT filter package. see pkg/zonal_filt
C/\ | | | | |-CONVECTIVE_ADJUSTMENT :: Control static instability mixing.
C/\ | | | | | |-FIND_RHO        :: Find adjacent densities.
C/\ | | | | | |-CONVECT        :: Mix static instability.
C/\ | | | | | |-TIMEAVE_CUMULATE :: Update convection statistics.
C/\ | | | |
C/\ | | | | |-CALC_EXACT_ETA     :: Change SSH to flow divergence.
C/\ | | | |
C/\ | | | | |-DO_FIELDS_BLOCKING_EXCHANGES :: Sync up overlap regions.
C/\ | | | | | |-EXCH
C/\ | | | |
C/\ | | | | |-GCHEM_FORCING_SEP :: tracer forcing for gchem pkg (if
C/\ | | | | |                                     tracer dependent tendencies calculated
C/\ | | | | |                                     separatly)
C/\ | | | |
C/\ | | | | |-FLT_MAIN          :: Float package ( pkg/flt ).
C/\ | | | |
C/\ | | | | |-MONITOR           :: Monitor package ( pkg/monitor ).
C/\ | | | |
C/\ | | | | |-DO_THE_MODEL_IO   :: Standard diagnostic I/O.
C/\ | | | | | |-WRITE_STATE      :: Core state I/O
C/\ | | | | | |-TIMEAVE_STATV_WRITE :: Time averages. see pkg/timeave
C/\ | | | | | |-AIM_WRITE_DIAGS  :: Intermed. atmos diags. see pkg/aim
C/\ | | | | | |-GMREDI_DIAGS    :: GM diags. see pkg/gmredi
C/\ | | | | | |-KPP_DO_DIAGS    :: KPP diags. see pkg/kpp
C/\ | | | | | |-SBO_CALC        :: SBO diags. see pkg/sbo
C/\ | | | | | |-SBO_DIAGS       :: SBO diags. see pkg/sbo
C/\ | | | | | |-SEAICE_DO_DIAGS :: SEAICE diags. see pkg/seaice
C/\ | | | | | |-GCHEM_DIAGS     :: gchem diags. see pkg/gchem
C/\ | | | |
C/\ | | | | |-WRITE_CHECKPOINT :: Do I/O for restart files.
C/\ | | |
C/\ | | | |-COST_TILE           :: Cost function package. ( see pkg/cost )
C<===|=|
C<===|=| *****
C<===|=| END MAIN TIMESTEPPING LOOP

```

(continues on next page)

(continued from previous page)

```

C<===|=| *****
C<===|=|
C  | |-COST_FINAL      :: Cost function package. ( see pkg/cost )
C  |
C  | |-WRITE_CHECKPOINT :: Final state storage, for restart.
C  |
C  | |-TIMER_PRINTALL  :: Computational timing summary
C  |
C  | |-COMM_STATS      :: Summarise inter-proc and inter-thread communication
C                        :: events.
C

```

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

CHAPTER 7

Automatic Differentiation

Packages I - Physical Parameterizations

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 [ref{sec:modelExamples}](#), 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 [ref{sec:modelExamples}](#) 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 [ref{sec:pkg:using}](#) describes the general procedure for using any package in MITgcm. Following that sections [ref{sec:pkg:gad}](#)-[ref{sec:pkg:monitor}](#) 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 table [ref{tab:package_summary_tab}](#). Organizationally package code is assigned a separate subdirectory in the MITgcm code distribution (within the source code directory `texttt{pkg}`). The name of this subdirectory is used as the package name in table [ref{tab:package_summary_tab}](#).

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

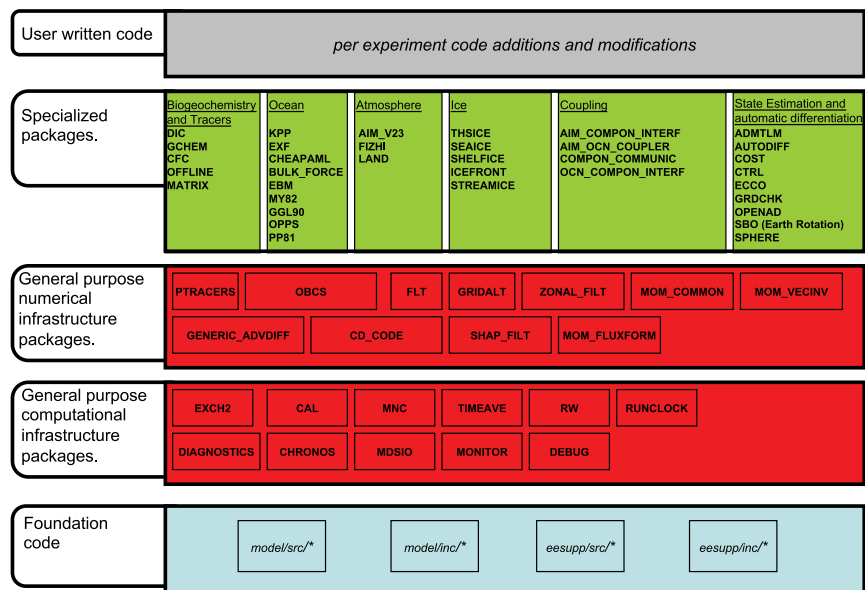


Figure 8.1: Hierarchy of code layers that are assembled to make up an MITgcm simulation. Conceptually (and in terms of code organization) MITgcm consists of several layers. At the base is a layer of core software that provides a basic numerical and computational foundation for MITgcm simulations. This 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.

“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.

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 [sec:buildingCode]. 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:
 - (a) As the new files are found, they are added to a list of source files.
 - (b) 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 supersede (or “hide”) any previous file(s) with the same name.
 - (c) Packages are visited (and thus files discovered) *in the order that the packages are enabled* within `genmake2`. Thus, the files in `PackB` may supersede 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:

```
#ifdef 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)

(continued from previous 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} )
&     CALL ${PKG}_INIT_VARIA( )
   #endif

```

(continues on next page)

(continued from previous page)

```

6. S/R DO_THE_MODEL_IO

    #ifdef ALLOW_${PKG}
        if ( use${Pkg} )
&        CALL ${PKG}_OUTPUT ( )
    #endif

7. S/R PACKAGES_WRITE_PICKUP ( )

    #ifdef ALLOW_${PKG}
        if ( use${Pkg} )
&        CALL ${PKG}_WRITE_PICKUP ( )
    #endif

```

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 *PARAM_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/ptracers*). (see also sections [sec:tracer:sub:equations] to [sec:tracer:sub:advection_schemes]).

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.

8.2.1.3 GAD Diagnostics

<-Name->	Levs	<-parsing code->	<-- Units -->	<- Tile (max=80c)		

ADVr_TH	15	WM LR	degC.m^3/s	Vertical	Advective Flux of Pot.	
↪Temperature						
ADVx_TH	15	UU 087MR	degC.m^3/s	Zonal	Advective Flux of Pot.	
↪Temperature						
ADVy_TH	15	VV 086MR	degC.m^3/s	Meridional	Advective Flux of Pot.	
↪Temperature						
DFrE_TH	15	WM LR	degC.m^3/s	Vertical	Diffusive Flux of Pot.	
↪Temperature (Explicit part)						
DIFx_TH	15	UU 090MR	degC.m^3/s	Zonal	Diffusive Flux of Pot.	
↪Temperature						
DIFy_TH	15	VV 089MR	degC.m^3/s	Meridional	Diffusive Flux of Pot.	
↪Temperature						
DFrI_TH	15	WM LR	degC.m^3/s	Vertical	Diffusive Flux of Pot.	
↪Temperature (Implicit part)						
ADVr_SLT	15	WM LR	psu.m^3/s	Vertical	Advective Flux of Salinity	
ADVx_SLT	15	UU 094MR	psu.m^3/s	Zonal	Advective Flux of Salinity	
ADVy_SLT	15	VV 093MR	psu.m^3/s	Meridional	Advective Flux of Salinity	
DFrE_SLT	15	WM LR	psu.m^3/s	Vertical	Diffusive Flux of Salinity	┐
↪ (Explicit part)						
DIFx_SLT	15	UU 097MR	psu.m^3/s	Zonal	Diffusive Flux of Salinity	
DIFy_SLT	15	VV 096MR	psu.m^3/s	Meridional	Diffusive Flux of Salinity	
DFrI_SLT	15	WM LR	psu.m^3/s	Vertical	Diffusive Flux of Salinity	┐
↪ (Implicit part)						

8.2.1.4 Experiments and tutorials that use GAD

- Offline tutorial, in tutorial_offline verification directory, described in section [sec:eg-offline]
- Baroclinic gyre experiment, in tutorial_baroclinic_gyre verification directory, described in section [sec:eg-fourlayer]
- Tracer Sensitivity tutorial, in tutorial_tracer_adjsens verification directory, described in section [sec:eg-simple-tracer-adjoint]

8.2.2 Shapiro Filter

(in directory: pkg/shap_filt/)

8.2.2.1 Key subroutines, parameters and files

Implementation of filter is described in section [sec:shapiro-filter].

8.2.2.2 Experiments and tutorials that use shap filter

- Held Suarez tutorial, in tutorial_held_suarez_cs verification directory, described in section [sec:eg-hs]

- 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)

8.2.3 FFT Filtering Code

(in directory: pkg/zonal_filt/)

8.2.3.1 Key subroutines, parameters and files

8.2.3.2 Experiments and tutorials that use zonal filter

- Held Suarez verification experiment (hs94.128x64x5)
- AIM verification experiment (aim.5l_LatLon)

8.2.4 `exch2`: Extended Cubed Sphere Topology

8.2.4.1 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 `\pkgexch2\W2_EXCH2_TOPOLOGY.h` and assigned in `pkg/exch2/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 [ref{sec:topogen}](#) 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.

8.2.4.2 Invoking `exch2`

To use `exch2` with the cubed sphere, the following conditions must be met:

- The `exch2` package is included when `genmake2` is run. The easiest way to do this is to add the line `code{exch2}` to the `packages.conf` file – see Section [ref{sec:buildingCode}](#) section [title{Building the code}](#) 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 `genmake2` 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 `tile00n.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 32×32 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 [ref{sec:topogen} sectiontitle{Generating Topology Files for exch2}](#) and [ref{sec:exch2mpi} sectiontitle{exch2, SIZE.h, and Multiprocessing}](#); a more general background on the subject relevant to MITgcm is presented in Section [ref{sec:specifying_a_decomposition} sectiontitle{Specifying a decomposition}](#).

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`

8.2.4.3 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 `utils-exch2-matlab-topology-generator_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 – figures [ref{fig:6tile}](#), [ref{fig:18tile}](#), and [ref{fig:48tile}](#) 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 `nr`, `nb`, `ng`, `tnx` and `tny`, 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 `tnx` and `tny` determine the width and height of the tiles into which the subdomains are decomposed, and must evenly divide the integer assigned to `nr`, `nb` and `ng`. 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.

8.2.4.4 `exch2`, `SIZE.h`, and Multiprocessing

Once the topology configuration files are created, each Fortran `PARAMETER` in `SIZE.h` must be configured to match. Section [ref{sec:specifying_a_decomposition} sectiontitle{Specifying a decomposition}](#) provides a general description of domain decomposition within MITgcm and its relation to file `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 `varlink{sNx}{sNx}` and `varlink{sNy}{sNy}` define the size of the individual tiles, and so must be assigned the same respective values as `code{tnx}` and `code{tny}` in file `driver.m`.

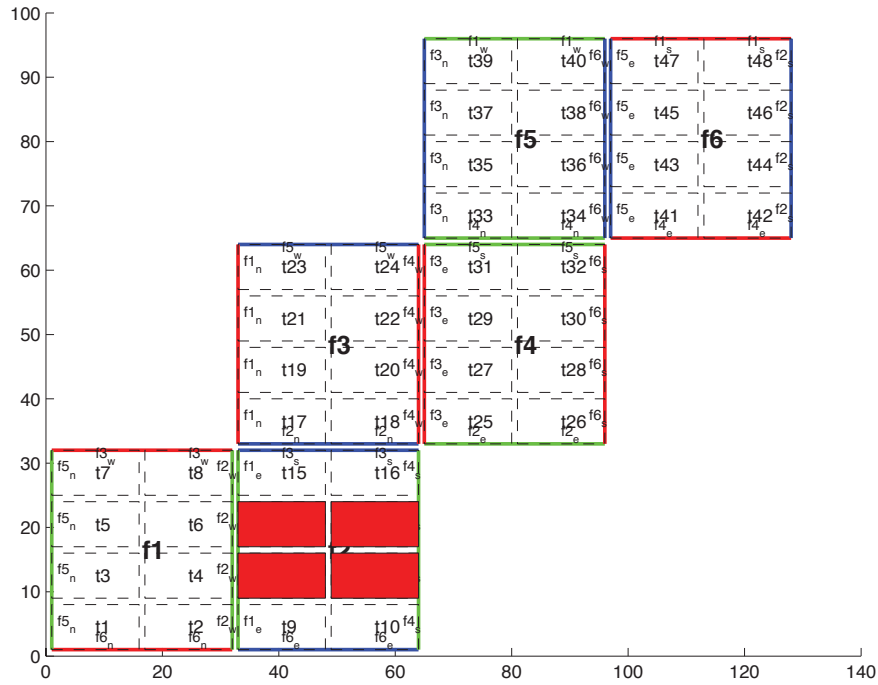


Figure 8.2: Plot of a cubed sphere topology with a 32×192 domain divided into six 32×32 subdomains, each of which is divided into eight tiles of width $t_{nx}=16$ and height $t_{ny}=8$ for a total of forty-eight tiles. The colored borders of the subdomains represent the parameters nr (red), ng (green), 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.

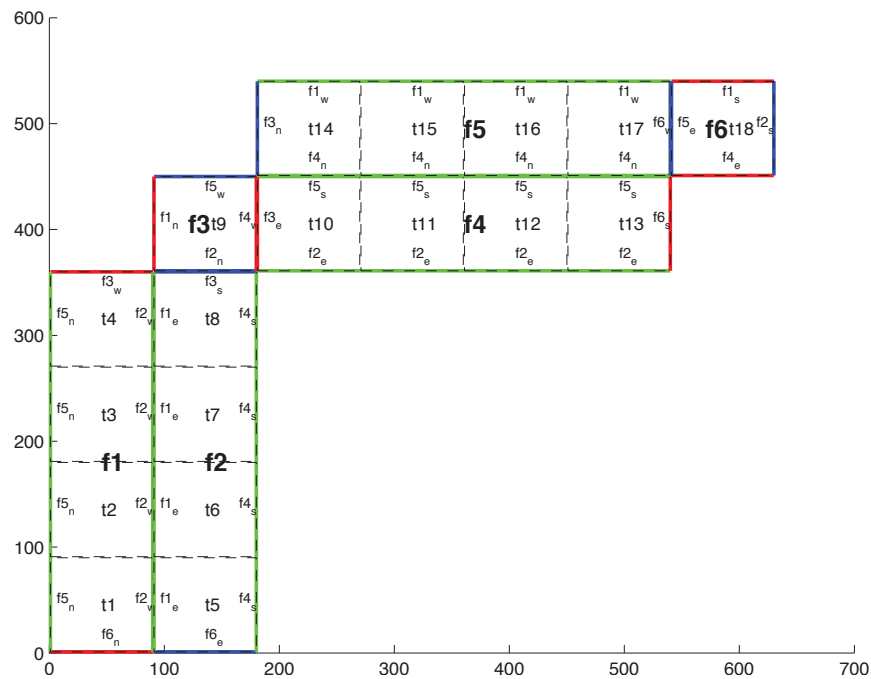


Figure 8.3: Plot of a non-square cubed sphere topology with 6 subdomains of different size ($nr=90, ng=360, nb=90$), divided into one to four tiles each ($t_{nx}=90, t_{ny}=90$), resulting in a total of 18 tiles.

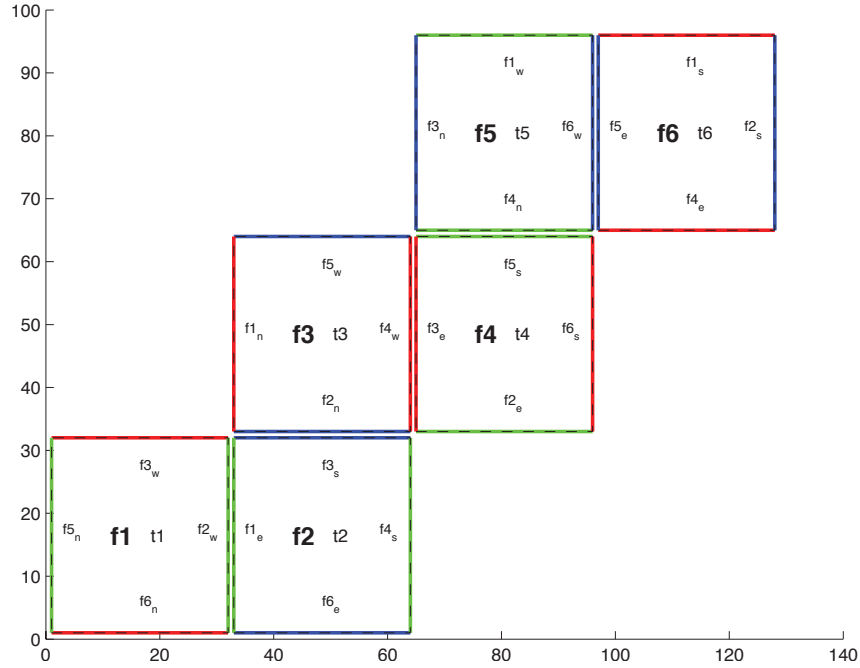


Figure 8.4: Plot of a cubed sphere topology with a 32×192 domain divided into six 32×32 subdomains with one tile each ($\text{tnx}=32$, $\text{tny}=32$). This is the default configuration.

The halo width parameters $\text{varlink}\{\text{OLx}\}\{\text{OLx}\}$ and $\text{varlink}\{\text{OLy}\}\{\text{OLy}\}$ have no special bearing on exch2 and may be assigned as in the general case. The same holds for $\text{varlink}\{\text{Nr}\}\{\text{Nr}\}$, the number of vertical levels in the model.

The parameters $\text{varlink}\{\text{nSx}\}\{\text{nSx}\}$, $\text{varlink}\{\text{nSy}\}\{\text{nSy}\}$, $\text{varlink}\{\text{nPx}\}\{\text{nPx}\}$, and $\text{varlink}\{\text{nPy}\}\{\text{nPy}\}$ 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 $\text{code}\{\text{varlink}\{\text{nSy}\}\{\text{nSy}\}=1\}$ in all cases. Since the tiles as configured by exch2 cannot be split up across processors without regenerating the topology, $\text{code}\{\text{varlink}\{\text{nPy}\}\{\text{nPy}\}=1\}$ as well.

The number of tiles MITgcm allocates and how they are distributed between processors depends on $\text{varlink}\{\text{nPx}\}\{\text{nPx}\}$ and $\text{varlink}\{\text{nSx}\}\{\text{nSx}\}$. $\text{varlink}\{\text{nSx}\}\{\text{nSx}\}$ is the number of tiles per processor and $\text{varlink}\{\text{nPx}\}\{\text{nPx}\}$ is the number of processors. The total number of tiles in the topology minus those listed in $\text{file}\{\text{blanklist.txt}\}$ must equal $\text{code}\{\text{nSx}*\text{nPx}\}$. 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 $\text{file}\{\text{blanklist.txt}\}$. 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 $\text{emph}\{\text{not}\}$ listed in $\text{file}\{\text{blanklist.txt}\}$.

The following is an example of $\text{file}\{\text{SIZE.h}\}$ for the six-tile configuration illustrated in figure [ref\{fig:6tile\}](#) running on one processor:

```
PARAMETER (
&      sNx = 32,
&      sNy = 32,
&      OLx = 2,
&      OLy = 2,
&      nSx = 6,
&      nSy = 1,
&      nPx = 1,
```

(continues on next page)

(continued from previous page)

```

&      nPy = 1,
&      Nx  = sNx*sSx*sPx,
&      Ny  = sNy*sSy*sPy,
&      Nr  = 5)

```

The following is an example for the forty-eight-tile topology in figure ref{fig:48tile} running on six processors:

```

PARAMETER (
&      sNx = 16,
&      sNy = 8,
&      OLx = 2,
&      OLy = 2,
&      nSx = 8,
&      nSy = 1,
&      nPx = 6,
&      nPy = 1,
&      Nx  = sNx*sSx*sPx,
&      Ny  = sNy*sSy*sPy,
&      Nr  = 5)

```

8.2.4.5 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 code{NTILES}, and the maximum number of neighbors of any tiles by code{MAX_NEIGHBOURS}. 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 file{driver.m}.

The scalar parameters `varlink{exch2_domain_nxt}{exch2_domain_nxt}` and `varlink{exch2_domain_nyt}{exch2_domain_nyt}` express the number of tiles in the `x` and `y` global indices. For example, the default setup of six tiles (Fig. ref{fig:6tile}) has code{exch2_domain_nxt=6} and code{exch2_domain_nyt=1}. A topology of forty-eight tiles, eight per subdomain (as in figure ref{fig:48tile}), will have code{exch2_domain_nxt=12} and code{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 code{varlink{bi}{bi}=(1:NTILES)} in the `x` axis, and the `y` axis variable `varlink{bj}{bj}` is assumed to equal code{1} throughout the package.

Arrays indexed to tile number:

The following arrays are of length code{NTILES} and are indexed to the tile number, which is indicated in the diagrams with the notation `textsf{t}n`. The indices are omitted in the descriptions.

The arrays `varlink{exch2_tnx}{exch2_tnx}` and `varlink{exch2_tny}{exch2_tny}` express the `x` and `y` dimensions of each tile. At present for each tile `texttt{exch2_tnx=sNx}` and `texttt{exch2_tny=sNy}`, as assigned in file{SIZE.h} and described in Section ref{sec:exch2mpi} sectiontitle{exch2, SIZE.h, and Multiprocessing}. Future releases of MITgcm may allow varying tile sizes.

The arrays `varlink{exch2_tbasex}{exch2_tbasex}` and `varlink{exch2_tbasey}{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 (Fig. ref{fig:6tile}) each index in these arrays is set to code{0} since a tile occupies its entire subdomain. The twenty-four-tile case discussed above will have values of code{0} or code{16}, depending on

the quadrant of the tile within the subdomain. The elements of the arrays `varlink{exch2_txglobalo}{exch2_txglobalo}` and `varlink{exch2_txglobalo}{exch2_txglobalo}` are similar to `varlink{exch2_tbasex}{exch2_tbasex}` and `varlink{exch2_tbasey}{exch2_tbasey}`, but locate the tile edges within the global address space, similar to that used by global output and input files.

The array `varlink{exch2_myFace}{exch2_myFace}` contains the number of the subdomain of each tile, in a range `code{(1:6)}` in the case of the standard cube topology and indicated by `textbf{texts{f}}{f}` in figures `ref{fig:6tile}` and `ref{fig:48tile}`. `varlink{exch2_nNeighbours}{exch2_nNeighbours}` contains a count of the neighboring tiles each tile has, and sets the bounds for looping over neighboring tiles. `varlink{exch2_tProc}{exch2_tProc}` holds the process rank of each tile, and is used in interprocess communication.

The arrays `varlink{exch2_isWedge}{exch2_isWedge}`, `varlink{exch2_isEdge}{exch2_isEdge}`, `varlink{exch2_isSedge}{exch2_isSedge}`, and `varlink{exch2_isNedge}{exch2_isNedge}` are set to `code{1}` if the indexed tile lies on the edge of its subdomain, `code{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 `code{MAX_NEIGHBOURS}` and `code{NTILES}` and describe the orientations between the tiles.

The array `code{exch2_neighbourId(a,T)}` holds the tile number `code{Tn}` for each of the tile number `code{T}`'s neighboring tiles `code{a}`. The neighbor tiles are indexed `code{(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 `code{exch2_opposingSend_record(a,T)}` array holds the index `code{b}` of the element in `texttt{exch2_neighbourId(b,Tn)}` that holds the tile number `code{T}`, given `code{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 `varlink{exch2_pi}{exch2_pi}` and `varlink{exch2_pj}{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 `code{exch2_pi(t,N,T)}` and `code{exch2_pj(t,N,T)}` are the neighbor ID `code{N}` and the tile number `code{T}` as explained above, plus a vector of length `code{2}` containing transformation factors `code{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, `code{exch2_pi(1,N,T)}` holds the mapping of the x axis index of tile `code{T}` to the x axis of tile `code{T}`'s neighbor `code{N}`, and `code{exch2_pi(2,N,T)}` holds the mapping of `code{T}`'s x index to the neighbor `code{N}`'s y index.

One of the two elements of `code{exch2_pi}` or `code{exch2_pj}` for a given tile `code{T}` and neighbor `code{N}` will be `code{0}`, reflecting the fact that the two axes are orthogonal. The other element will be `code{1}` or `code{-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 `code{(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 `code{(0,1)}`. Those with the opposite index direction will have `code{(0,-1)}` in order to reverse the ordering.

The arrays `varlink{exch2_oi}{exch2_oi}`, `varlink{exch2_oj}{exch2_oj}`, `varlink{exch2_oi_f}{exch2_oi_f}`, and `varlink{exch2_oj_f}{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 code{N} to a local tile code{T}. Consider code{T=1} in the six-tile topology (Fig. ref{fig:6tile}), where

```
exch2_oi(1,1)=33
exch2_oi(2,1)=0
exch2_oi(3,1)=32
exch2_oi(4,1)=-32
```

The simplest case is code{exch2_oi(2,1)}, the southern neighbor, which is code{Tn=6}. The axes of code{T} and code{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 (code{Tn=5}), code{exch2_oi(3,1)=32} since the code{x=0} vector on code{T} corresponds to the code{y=32} vector on code{Tn}. The eastern edge of code{T} shows the reverse case (code{exch2_oi(4,1)=-32}), where code{x=32} on code{T} exchanges with code{x=0} on code{Tn=2}.

The most interesting case, where code{exch2_oi(1,1)=33} and code{Tn=3}, involves a reversal of indices. As in every case, the offset code{exch2_oi} is added to the original x index of code{T} multiplied by the transformation factor code{exch2_pi(t,N,T)}. Here code{exch2_pi(1,1,1)=0} since the x axis of code{T} is orthogonal to the x axis of code{Tn}. code{exch2_pi(2,1,1)=-1} since the x axis of code{T} corresponds to the y axis of code{Tn}, but the index is reversed. The result is that the index of the northern edge of code{T}, which runs code{(1:32)}, is transformed to code{(-1:-32)}. code{exch2_oi(1,1)} is then added to this range to get back code{(32:1)} – the index of the y axis of code{Tn} relative to code{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, varlink{exch2_itlo_c}{exch2_itlo_c}, varlink{exch2_ithi_c}{exch2_ithi_c}, varlink{exch2_jtlo_c}{exch2_jtlo_c} and varlink{exch2_jthi_c}{exch2_jthi_c} hold the location and index bounds of the edge segment of the neighbor tile code{N}'s subdomain that gets exchanged with the local tile code{T}. To take the example of tile code{T=2} in the forty-eight-tile topology (Fig. ref{fig:48tile}):

```
exch2_itlo_c(4,2)=17
exch2_ithi_c(4,2)=17
exch2_jtlo_c(4,2)=0
exch2_jthi_c(4,2)=33
```

Here code{N=4}, indicating the western neighbor, which is code{Tn=1}. code{Tn} resides on the same subdomain as code{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 code{exch2_itlo_c} and code{exch2_ithi_c} indicate the column beyond code{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 code{exch2_jtlo_c} to code{exch2_jthi_c} cover the height of code{(1:32)}, plus 1 in either direction to cover part of the halo.

For the north edge of the same tile code{T=2} where code{N=1} and the neighbor tile is code{Tn=5}:

```
exch2_itlo_c(1,2)=0
exch2_ithi_c(1,2)=0
exch2_jtlo_c(1,2)=0
exch2_jthi_c(1,2)=17
```

code{T}'s northern edge is parallel to the x axis, but since code{Tn}'s y axis corresponds to code{T}'s x axis, code{T}'s northern edge exchanges with code{Tn}'s western edge. The western edge of the tiles corresponds to the lower bound of the x axis, so code{exch2_itlo_c} and code{exch2_ithi_c} are code{0}, in the western halo region of code{Tn}. The range of code{exch2_jtlo_c} and code{exch2_jthi_c} correspond to the width of code{T}'s northern edge, expanded by one into the halo.

8.2.4.6 Key Routines

Most of the subroutines particular to `exch2` handle the exchanges themselves and are of the same format as those described in `ref{sec:cube_sphere_communication}` section `title{Cube sphere communication}`. Like the original routines, they are written as templates which the local Makefile converts from `code{RX}` into `code{RL}` and `code{RS}` forms.

The interfaces with the core model subroutines are `code{EXCH_UV_XY_RX}`, `code{EXCH_UV_XYZ_RX}` and `code{EXCH_XY_RX}`. They override the standard exchange routines when `code{genmake2}` is run with `code{exch2}` option. They in turn call the local `exch2` subroutines `code{EXCH2_UV_XY_RX}` and `code{EXCH2_UV_XYZ_RX}` for two and three-dimensional vector quantities, and `code{EXCH2_XY_RX}` and `code{EXCH2_XYZ_RX}` for two and three-dimensional scalar quantities. These subroutines set the dimensions of the area to be exchanged, call `code{EXCH2_RX1_CUBE}` for scalars and `code{EXCH2_RX2_CUBE}` for vectors, and then handle the singularities at the cube corners.

The separate scalar and vector forms of `code{EXCH2_RX1_CUBE}` and `code{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 `code{EXCH2_SEND_RX1}` and `code{EXCH2_SEND_RX2}`, which do most of the work using the variables discussed above.

8.2.4.7 Experiments and tutorials that use `exch2`

- Held Suarez tutorial, in `tutorial_held_suarez_cs` verification directory, described in section `ref{sec:eg-hs}`

8.2.5 Gridalt - Alternate Grid Package

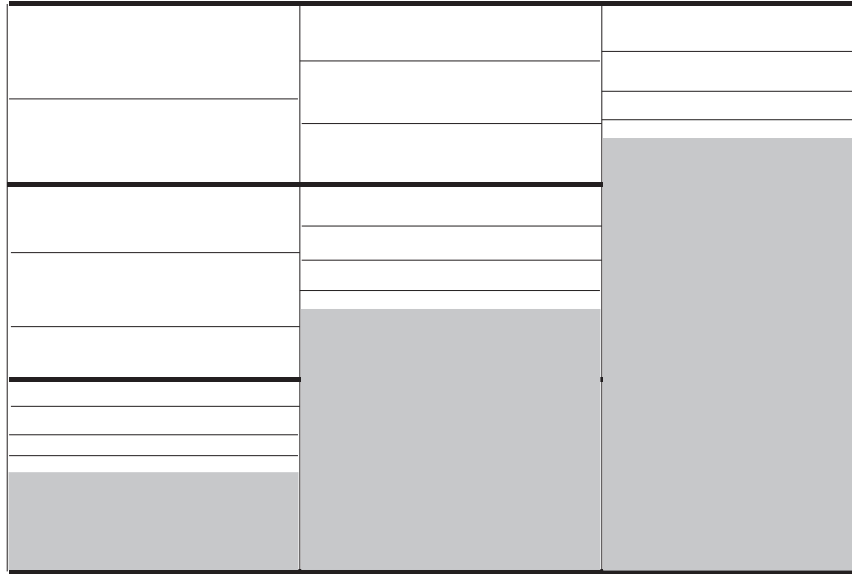
8.2.5.1 Introduction

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 independent 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:

Modified P* Discretization for High End Physics



Dark solid lines represent existing P* levels, light solid lines are the addition levels added at each grid cell.

Figure 8.5: Vertical discretization for MITgcm (dark grey lines) and for the atmospheric physics (light grey lines). In this implementation, all MITgcm level interfaces must coincide with atmospheric physics level interfaces.

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.

8.2.5.2 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:

$$\left. \frac{\partial U}{\partial t} \right|_{p^*}^{total} = \left. \frac{\partial U}{\partial t} \right|_{p^*}^{dynamics} + \left. \frac{\partial U}{\partial t} \right|_{p^*}^{physics}$$

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:

$$\left. \frac{\partial U}{\partial t} \right|_{p^{*m}}^{total} = \left. \frac{\partial U}{\partial t} \right|_{p^{*m}}^{dynamics} + \left. \frac{\partial U}{\partial t} \right|_{p^{*m}}^{physics} + \gamma(U|_{p^*} - U|_{p^{*m}})$$

where p^{*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 γ . In the present implementation, $\gamma = 1$, requiring an immediate agreement between the two ‘states’.

8.2.5.3 Time stepping Sequence

If we write T_{phys} as the temperature (or any other state variable) on the high resolution physics grid, and T_{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: $T_{phys}^{n+1**}(l) = T_{phys}^n(l) + \delta T_{phys}$.
3. Interpolate the physics tendency to the dynamics grid, and advance the dynamics state by physics and dynamics tendencies: $T_{dyn}^{n+1}(L) = T_{dyn}^n(L) + \delta T_{dyn}(L) + [\delta T_{phys}(l)](L)$.
4. Interpolate the dynamics tendency to the physics grid, and update the physics grid due to dynamics tendencies: $T_{phys}^{n+1*}(l) = T_{phys}^{n+1**}(l) + \delta T_{dyn}(L)(l)$.
5. Apply correction term to physics state to account for divergence from dynamics state: $T_{phys}^{n+1}(l) = T_{phys}^{n+1*}(l) + \gamma\{T_{dyn}(L) - [T_{phys}(l)](L)\}(l)$. Where $\gamma = 1$ here.

8.2.5.4 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 , $T_{phys}(l) = \{T_{dyn}(L)\} = T_{dyn}(L)$.

Fine to coarse: For all physics layers l in dynamics layer L , $T_{dyn}(L) = [T_{phys}(l)] = \int T_{phys} dp$.

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 *phys* and *dyn* stand for variables on the physics and dynamics grids, respectively.

8.2.5.5 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.

```

      subroutine make_phys_grid(drF,hfacC,im1,im2,jm1,jm2,Nr,
      . Nsx,Nsy,i1,i2,j1,j2,bi,bj,Nrphys,Lbot,dpphys,numlevphys,nlperdyn)
C*****
C Purpose: Define the grid that the will be used to run the high-end
C           atmospheric physics.
C
C Algorithm: Fit additional levels of some (~) known thickness in
C            between existing levels of the grid used for the dynamics
C
C Need:      Information about the dynamics grid vertical spacing
C
C Input:     drF          - delta r (p*) edge-to-edge
C            hfacC        - fraction of grid box above topography

```

(continues on next page)

(continued from previous page)

```

c      im1, im2      - beginning and ending i - dimensions
c      jm1, jm2      - beginning and ending j - dimensions
c      Nr            - number of levels in dynamics grid
c      Nsx,Nsy       - number of processes in x and y direction
c      il, i2        - beginning and ending i - index to fill
c      jl, j2        - beginning and ending j - index to fill
c      bi, bj        - x-dir and y-dir index of process
c      Nrphys        - number of levels in physics grid
c
c Output:  dpphys      - delta r (p*) edge-to-edge of physics grid
c          numlevphys  - number of levels used in the physics
c          nlperdyn    - physics level number atop each dynamics layer
c
c NOTES: 1) Pressure levs are built up from bottom, using p0, ps and dp:
c          p(i,j,k)=p(i,j,k-1) + dp(k)*ps(i,j)/p0(i,j)
c          2) Output dp's are aligned to fit EXACTLY between existing
c             levels of the dynamics vertical grid
c          3) IMPORTANT! This routine assumes the levels are numbered
c             from the bottom up, ie, level 1 is the surface.
c             IT WILL NOT WORK OTHERWISE!!!
c          4) This routine does NOT work for surface pressures less
c             (ie, above in the atmosphere) than about 350 mb
c*****

```

In the case of the grid used to compute the atmospheric physical forcing (fizhi package), 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:

```

      subroutine gridalt_update(myThid)
c*****
c Purpose: Update the pressure thicknesses of the layers of the
c          alternative vertical grid (used now for atmospheric physics).
c
c Calculate: dpphys      - new delta r (p*) edge-to-edge of physics grid
c                  using dpphys0 (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.

```

      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 (pstar)
c   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
c   wind speed (for U and V fields), or uniformly (T and Q)
c
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

```

(continues on next page)

(continued from previous page)

```

C  im1,2 ... Limits for Longitude Dimension of Input
C  jm1,2 ... Limits for Latitude Dimension of Input
C  lmdyn.... Vertical Dimension of Input
C  Nsx..... Number of processes in x-direction
C  Nsy..... Number of processes in y-direction
C  idim1,2.. Beginning and ending i-values to calculate
C  jdim1,2.. Beginning and ending j-values to calculate
C  bi..... Index of process number in x-direction
C  bj..... Index of process number in x-direction
C  windphy.. [im,jm,lmphy] Magnitude of the wind on the output levels
C  pephy.... [im,jm,lmphy+1] Pressures at bottom edges of output levels
C  lmphy.... Vertical Dimension of Output
C  nlperdyn. [im,jm,lmdyn] Highest Physics level in each dynamics level
C  flg..... Flag to indicate field type (0 for T or Q, 1 for U or V)
C
C Output:
C  qphy..... [im,jm,lmphy] Quantity at output grid (physics grid)
C
C Notes:
C  1) This algorithm assumes that the output (physics) grid levels
C     fit exactly into the input (dynamics) grid levels
C *****

```

And similarly, gridalt contains subroutine phys2dyn.

8.2.5.6 Gridalt Diagnostics

```

-----
<-Name->|Levs|<-parsing code->|<--  Units  -->|<- Tile (max=80c)
-----
DPPHYS  | 20 | SM      ML      |Pascal      |Pressure Thickness of Layers on Fizhi_
->Grid

```

8.2.5.7 Dos and donts

8.2.5.8 Gridalt Reference

8.2.5.9 Experiments and tutorials that use gridalt

- Fizhi experiment, in 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 is fundamental to regional ocean modelling with the MITgcm, but there are so many details to be considered in regional ocean modelling that this package cannot accomodate 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- and runtime-options of this package, but also with the code itself. In many cases it will be necessary to adapt the obcs-code (in particular code{S/R OBCS_CALC}) to the application in question; in these cases the obcs-package (together with the rbcS-package, section ref{sec:pkg:rbcS}) 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:
 - * A simple time-management (e.g. constant in time, or cyclic with fixed fequency) is provided through `S/R obcs_external_fields_load`.
 - * More sophisticated 'real-time' (i.e. calendar time) management is available through `obcs_prescribe_read`.
 - The latter case requires packages `cal` and `exf` to be enabled.

(see also Section ref{sec:buildingCode}).

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 8.1: OBCS CPP options

CPP option	Description
<code>ALLOW_OBCS_NORTH</code>	enable Northern OB
<code>ALLOW_OBCS_SOUTH</code>	enable Southern OB
<code>ALLOW_OBCS_EAST</code>	enable Eastern OB
<code>ALLOW_OBCS_WEST</code>	enable Western OB
<code>ALLOW_OBCS_PRESCRIBE</code>	enable code for prescribing OB's
<code>ALLOW_OBCS_SPONGE</code>	enable sponge layer code
<code>ALLOW_OBCS_BALANCE</code>	enable code for balancing transports through OB's
<code>ALLOW_ORLANSKI</code>	enable Orlanski radiation conditions at OB's
<code>ALLOW_OBCS_STEVENS</code>	enable Stevens (1990) boundary conditions at OB's
	(currently only implemented for eastern and
	western boundaries and NOT for ptracers)

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. package :code:`exf` enabled). These parameter files are read

in `S/R packages_readparms.F`, `obcs_readparms.F`, and `exf_readparms.F`, respectively. Run-time parameters may be broken into 3 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: pkg OBCS run-time parameters

Flag/parameter	default	Description
<i>basic flags & parameters (OBCS_PARM01)</i>		
OB_Jnorth	0	Nx-vector of J-indices (w.r.t. Ny) of Northern OB at each I-position
OB_Jsouth	0	Nx-vector of J-indices (w.r.t. Ny) of Southern OB at each I-position
OB_Ieast	0	Ny-vector of I-indices (w.r.t. Nx) of Eastern OB at each J-position
OB_Iwest	0	Ny-vector of I-indices (w.r.t. Nx) of Western OB at each J-position
useOBCSprescribe	.FALSE.	
useOBCSsponge	.FALSE.	
useOBCSbalance	code{.FALSE.}	
OBCS_balanceFacN/S/E/W	1	factor(s) determining the details of the balancing code
useOrlanskiNorth/South/EastWest	.FALSE.	turn on Orlanski boundary conditions for individual boundary
useStevensNorth/South/EastWest	.FALSE.	turn on Stevens boundary conditions for individual boundary
OBXyFile		file name of OB field
		X : N(orth) S (outh) E (ast) W (est)
		y : t(emperature) s (alinity) u (-velocity) v (-velocity)
		w (-velocity) eta (sea surface height)
		a (sea ice area) h (sea ice thickness) sn (snow thickness) sl (sea ice
<i>Orlanski parameters (OBCS_PARM02)</i>		
cvelTimeScale	2000 sec	averaging period for phase speed
CMAX	0.45 m/s	maximum allowable phase speed-CFL for AB-II
CFIX	0.8 m/s	fixed boundary phase speed
useFixedCEast	.FALSE.	
useFixedCWest	.FALSE.	
<i>Sponge-layer parameters (OBCS_PARM03)</i>		
spongeThickness	0	sponge layer thickness (in grid points)
Urelaxobcsinner	0 sec	relaxation time scale at the innermost sponge layer point of a merid
Vrelaxobcsinner	0 sec	relaxation time scale at the innermost sponge layer point of a zonal
Urelaxobcsbound	0 sec	relaxation time scale at the outermost sponge layer point of a merid
Vrelaxobcsbound	0 sec	relaxation time scale at the outermost sponge layer point of a zonal
<i>Stevens parameters (OBCS_PARM04)</i>		

Continued on

Table 8.2 – continued from previous page

T/SrelaxStevens	0 sec	relaxation time scale for temperature/salinity
useStevensPhaseVel	code{.TRUE.}	
useStevensAdvection	code{.TRUE.}	

8.3.1.4 Defining open boundary positions

There are four open boundaries (OBs), a 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 $i = 1, \dots, N_x$ a meridional index j specifies the Northern/Southern OB position, and for each meridional position $j = 1, \dots, N_y$, 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. 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 an OB point `U(3,34)` is a an OB point `V(3,34)` is a an OB point `OB_Jsouth(3)=1` means that: `T(3,1)` is a an OB point `U(3,1)` is a an OB point `V(3,2)` is a an OB point `OB_Ieast(10)=69` means that: `T(69,10)` is a an OB point `U(69,10)` is a an OB point `V(69,10)` is a an OB point `OB_Iwest(10)=1` means that: `T(1,10)` is a an OB point `U(2,10)` is a an OB point `V(1,10)` is a an OB point

For convenience, negative values for `Jnorth/Ieast` refer to points relative to the Northern/Eastern edges of the model eg. `OB_Jnorth(3) = -1` means that the point `(3, Ny)` is a northern 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 in is:

```
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,
```

If only the first 50 grid points of the southern boundary are boundary points:

```
OB_Jsouth(1:50) = 50*1,
```

8.3.1.5 Equations and key routines

OB_CS_READPARMS:

Set OB positions through arrays `OB_Jnorth(Nx)`, `OB_Jsouth(Nx)`, `OB_Ieast(Ny)`, `OB_Iwest(Ny)`, and runtime flags (see Table [tab:pkg:obcs:runtime:sub:flags]).

OB_CS_CALC:

Top-level routine for filling values to be applied at OB for T, S, U, V, η into corresponding “slice” arrays (x, z) , (y, z) for each OB: `OB[N/S/E/W][t/s/u/v]`; e.g. for salinity array at Southern OB, array name is `OBSt`. 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).
- use prescribed boundary fields to compute Stevens boundary conditions.

ORLANSKI:

Orlanski radiation conditions [Orl76], examples can be found in `verification/dome` and `verification/tutorial_plume_on_slope`

(ref{sec:eg-gravityplume}).

OBCS_PRESCRIBE_READ:

When `useOBCSprescribe = .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 $(N_x \times N_r \times \text{time levels})$, for East/West boundary files the dimensions are $(N_y \times N_r \times \text{time levels})$.
- If a non-linear free surface is used (ref{sec:nonlinear-freesurface}), additional files `OB[N/S/E/W]etaFile` for the sea surface height η with dimension $(N_{x/y} \times \text{time levels})$ may be specified.
- If non-hydrostatic dynamics are used (ref{sec:non-hydrostatic}), additional files `OB[N/S/E/W]wFile` for the vertical velocity w with dimensions $(N_{x/y} \times N_r \times \text{time levels})$ 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 $(N_{x/y} \times \text{time levels})$ can be specified.

As in `S/R external_fields_load` or the `exf`-package, 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 the `exf`-package is used, the time levels are controlled for each boundary separately in the same way as the `exf`-fields in `data.exf`, namelist `EXF_NML_OBCS`. The runtime flags follow the above naming conventions, e.g. for the western boundary the corresponding flags are `OBCWstartdate1/2` and `OBCWperiod`. Sea-ice boundary values are controlled separately with `siobWstartdate1/2` and `siobWperiod`. When the `exf`-package is not used, the time levels are controlled by the runtime flags `externForcingPeriod` and `externForcingCycle` in `data`, see `verification/exp4` for an example.

OBCS_CALC_STEVENS:

(THE IMPLEMENTATION OF THESE BOUNDARY CONDITIONS IS NOT COMPLETE. PASSIVE TRACERS, SEA ICE AND NON-LINEAR FREE SURFACE ARE NOT SUPPORTED PROPERLY.)

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 χ_{ob} (note: passive tracers are currently not implemented and the code stops when package code{ptracers} is used together with this option). Currently, the code vertically averages the normal velocity as specified in code{OB[E,W]u} or code{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 [sec:time:sub:stepping], e.g., eq.([eq:ustar-backward-free-surface])). (This velocity is not available at the beginning of the next time step $n+1$, when S/R OBCS_CALC/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 boundary value for tracer χ is restored to the prescribed values:

$$\chi^{n+1} = \chi^n + \frac{\Delta t}{\tau_\chi} (\chi_{ob} - \chi^n),$$

where τ_χ is the relaxation time scale `T/SrelaxStevens`. The new χ^{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 χ^{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):

$$\chi_{i_b,j,k}^{n+1} = \chi_{i_b,j,k}^n + \Delta t (u^{n+1} + c)_{i_b,j,k} \frac{\chi_{i_b,j,k}^n - \chi_{i_b-1,j,k}^n}{\Delta x_{i_b,j}^C}, \text{ if } u_{i_b,j,k}^{n+1} > 0,$$

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.

OBCS_BALANCE_FLOW:

When turned on (`ALLOW_OBCS_BALANCE` defined in `OBCS_OPTIONS.h` and `useOBCSbalance=.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/S/E/W`. 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 subtracting 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:

$$u(y, z) - \int_{\text{western boundary}} u \, dy \, dz \approx OBNu(j, k) - \sum_{j,k} OBNu(j, k) h_w(i_b, j, k) \Delta y_G(i_b, j) \Delta z(k).$$

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, say, 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 ≥ 0` is the preferred method.

OBCS_APPLY_*:**OBCS_SPONGE:**

The sponge layer code (turned on with `ALLOW_OBCS_SPONGE` and `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

$$G_{\chi}^{(\text{sponge})} = -\frac{\chi - [(L - \delta L)\chi_{BC} + \delta L\chi]/L}{[(L - \delta L)\tau_b + \delta L\tau_i]/L} = -\frac{\chi - [(1 - l)\chi_{BC} + l\chi]}{[(1 - l)\tau_b + l\tau_i]}$$

where χ is the model variable (U/V/T/S) in the interior, χ_{BC} the boundary value, L the thickness of the sponge layer (runtime parameter `spongeThickness` in number of grid points), $\delta L \in [0, L]$ ($\frac{\delta L}{L} = l \in [0, 1]$) the distance from the boundary (also in grid points), and τ_b (runtime parameters `Urelaxobcsbound` and `Vrelaxobcsbound`) and τ_i (runtime parameters `Urelaxobcsinner` and `Vrelaxobcsinner`) the relaxation time scales on the boundary and at the interior termination of the sponge layer. The parameters `Urelaxobcsbound/inner` set the relaxation time scales for the Eastern and Western boundaries, `:code:`Vrelaxobcsbound/inner` for the Northern and Southern boundaries.

OB's with nonlinear free surface**8.3.1.6 Flow chart**

```
C      !CALLING SEQUENCE:
C ...
```

8.3.1.7 OBCS diagnostics

Diagnostics output is available via the diagnostics package (see Section [sec:pkg:diagnostics]). Available output fields are summarized in Table [tab:pkg:obcs:diagnostics].

[tab:pkg:obcs:diagnostics]

```
-----
<-Name->|Levs|grid|<--  Units    -->|<- Tile (max=80c)
-----
```

8.3.1.8 Reference experiments

In the directory `verification`, the following experiments use `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” (<http://www.rsmas.miami.edu/personal/tamay/DOME/dome.html>), uses Orlanski-BCs;
- `internal_wave`: uses a heavily modified `S/R~OBCS_CALC`
- `:code:seaice_obcs`: simple example who to use the sea-ice related code, based on `lab_sea`;
- `tutorial_plume_on_slope`: uses Orlanski-BCs, see also section [sec:eg-gravityplume].

8.3.1.9 References

8.3.1.10 Experiments and tutorials that use obcs

- `tutorial_plume_on_slope` (`section-ref{sec:eg-gravityplume}`)

8.3.2 RBCS Package

8.3.2.1 Introduction

A package which provides the flexibility to relax fields (temperature, salinity, ptracers) in any 3-D location: so could be used as a sponge layer, or as a “source” anywhere in the domain.

For a tracer (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) with values between 0 and 1. Where M_{rbc} is 1, relaxing timescale is $1/\tau_T$. Where it is 0 there is no relaxing. The value relaxed to is a 3-D (potentially varying in time) field given by T_{rbc} .

A separate mask can be used for T, S and ptracers and each of these can be relaxed or not and can have its own timescale τ_T . These are set in `data.rbc`s (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.h`, the number of masks, `PARAMETER(maskLEN = 3)`, see below.

The runtime parameters are set in `data.rbc`s:

Set in `RBCS_PARM01`: - **rbcForcingPeriod**: time interval between forcing fields (in seconds), zero means constant-in-time forcing. - **rbcForcingCycle**: repeat cycle of forcing fields (in seconds), zero means non-cyclic forcing. - **rbcForcingOffset**: time offset of forcing fields (in seconds, default 0); this is relative to time averages starting at $t = 0$, i.e., the first forcing record/file is placed at `rbcForcingOffset + rbcForcingPeriod/2`; see below for examples. - **rbcSingleTimeFiles**: true or false (default false), if true, forcing fields are given 1 file per `rbcForcingPeriod`. - **deltaTrbc**: time step used to compute the iteration numbers for `rbcSingleTimeFiles=T`. - **rbcIter0**: shift in iteration numbers used to label files if `rbcSingleTimeFiles=T` (default 0, see below for examples). - **useRBCtemp**: true or false (default false) - **useRBCsalt**: true or false (default false) - **useRBCptracers**: true or false (default false), must be using ptracers to set true - **tauRelaxT**: timescale in seconds of relaxing in temperature (τ_T in equation above). Where mask is 1, relax rate will be $1/\text{tauRelaxT}$. Default is 1. - **tauRelaxS**: same for salinity. - **relaxMaskFile(irbc)**: filename of 3-D file with mask (M_{rbc} in equation above. Need a file for each irbc. 1=temperature, 2=salinity, 3=ptracer01, 4=ptracer02 etc. If the mask numbers end (see `maskLEN`) are less than the number tracers, then `relaxMaskFile(maskLEN)` is used for all remaining ptracers. - **relaxTFile**: name of file where temperatures that need to be relaxed to (T_{rbc} in equation above) are stored. The file must contain 3-D records to match the model domain. If `rbcSingleTimeFiles=F`, it must have one record for each forcing period. If T, there must be a separate file for each period and a 10-digit iteration number is appended to the file name (see Table [tab:pkg:rbc:timing] and examples below). - **relaxSFile**: same for salinity.

Set in `RBCS_PARM02` for each of the ptracers (`iTrc`): - **useRBCptrnum(iTrc)**: true or false (default is false). - **tauRelaxPTR(iTrc)**: relax timescale. - **relaxPtracerFile(iTrc)**: file with relax fields.

8.3.2.3 Timing of relaxation forcing fields

For constant-in-time relaxation, set `rbcsForcingPeriod=0`. For time-varying relaxation, Table [tab:pkg:rbcs:timing] illustrates the relation between model time and forcing fields (either records in one big file or, for `rbcsSingleTimeFiles=T`, individual files labeled with an iteration number). With `rbcsSingleTimeFiles=T`, this is the same as in the offline package, except that the forcing offset is in seconds.

Table 8.3: Timing of RBCS relaxation fields

	rbcsSingleTimeFiles = T		F
	$c = 0$	$c \neq 0$	$c \neq 0$
model time	file number	file number	record
$t_0 - p/2$	i_0	$i_0 + c/\Delta t_{\text{rbcs}}$	c/p
$t_0 + p/2$	$i_0 + p/\Delta t_{\text{rbcs}}$	$i_0 + p/\Delta t_{\text{rbcs}}$	1
$t_0 + p + p/2$	$i_0 + 2p/\Delta t_{\text{rbcs}}$	$i_0 + 2p/\Delta t_{\text{rbcs}}$	2
...
$t_0 + c - p/2$...	$i_0 + c/\Delta t_{\text{rbcs}}$	c/p
...

where

$p = \text{rbcsForcingPeriod}$

$c = \text{rbcsForcingCycle}$

$t_0 = \text{rbcsForcingOffset}$

$i_0 = \text{rbcsIter0}$

$\Delta t_{\text{rbcs}} = \text{deltaTrbcs}$

8.3.2.4 Example 1: forcing with time averages starting at $t = 0$

Cyclic data in a single file

Set `rbcsSingleTimeFiles=F` and `rbcsForcingOffset=0`, and the model will start by interpolating the last and first records of `rbcs` data, placed at $-p/2$ and $p/2$, resp., as appropriate for fields averaged over the time intervals $[-p, 0]$ and $[0, p]$.

Non-cyclic data, multiple files

Set `rbcsForcingCycle=0` and `rbcsSingleTimeFiles=T`. 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 $p/2$.

8.3.2.5 Example 2: forcing with snapshots starting at $t = 0$

Cyclic data in a single file

Set `rbcsSingleTimeFiles=F` and `rbcsForcingOffset=-p/2`, and the model will start forcing with the first record at $t = 0$.

Non-cyclic data, multiple files

Set `rbcsForcingCycle=0` and `rbcsSingleTimeFiles=T`. In this case, it is more natural to set `rbcsForcingOffset=+p/2`. With `rbcsIter0=0` and `deltaTrbcs=rbcsForcingPeriod`, the model would then start with data from files `relax*File.0000000000.data` at $t = 0$. It would then proceed to interpolate between this file and files `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 `rbcs`:

- `exp4` : box with 4 open boundaries, simulating flow over a Gaussian bump based on [\[AHM97\]](#)

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 `pkg/gchem` (section [\[sec:pkg:gchem\]](#)).

Can use up to 3843 tracers. But can not use `pkg/diagnostics` 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`:

- **PTRACERS_iter0** which is the integer timestep when the tracer experiment is initialized. If `nIter0 = PTRACERS_iter0` then the tracers are initialized to zero or from initial files. If `nIter0 > 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 `nIter0 < 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 [tab:advectionShemes:sub:summary].
- **PTRACERS_ImplVertAdv(iTrc)**: implicit vertical advection flag, default 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 these will only work for 90 or less tracers (some problems with the numbering/designation over this number)

<-Name-> Levs		<-parsing code-> <--		Units	--> <- Tile (max=80c)

TRAC01	15	SM P	MR	mol C/m	Mass-Weighted Dissolved Inorganic_C
↪Carbon					
UTRAC01	15	UU	171MR	mol C/m.m/s	Zonal Mass-Weighted Transp of_C
↪Dissolved Inorganic Carbon					
VTRAC01	15	VV	170MR	mol C/m.m/s	Merid Mass-Weighted Transp of_C
↪Dissolved Inorganic Carbon					
WTRAC01	15	WM	MR	mol C/m.m/s	Vert Mass-Weighted Transp of_C
↪Dissolved Inorganic Carbon					
ADVrTr01	15	WM	LR	mol C/m.m^3/s	Vertical Advective Flux of_C
↪Dissolved Inorganic Carbon					
(continues on next page)					

(continues on next page)

(continued from previous page)

ADVxTr01	15	UU	175MR	mol C/m.m^3/s	Zonal	Advective Flux of	↪
↪Dissolved Inorganic Carbon							
ADVyTr01	15	VV	174MR	mol C/m.m^3/s	Meridional	Advective Flux of	↪
↪Dissolved Inorganic Carbon							
DfErTr01	15	WM	LR	mol C/m.m^3/s	Vertical	Diffusive Flux of Dissolved	↪
↪Inorganic Carbon (Explicit part)							
DIFxTr01	15	UU	178MR	mol C/m.m^3/s	Zonal	Diffusive Flux of	↪
↪Dissolved Inorganic Carbon							
DIFyTr01	15	VV	177MR	mol C/m.m^3/s	Meridional	Diffusive Flux of	↪
↪Dissolved Inorganic Carbon							
DfIrTr01	15	WM	LR	mol C/m.m^3/s	Vertical	Diffusive Flux of Dissolved	↪
↪Inorganic Carbon (Implicit part)							
TRAC02	15	SM P	MR	mol eq/	Mass-Weighted	Alkalinity	
UTRAC02	15	UU	182MR	mol eq/.m/s	Zonal	Mass-Weighted Transp of	↪
↪Alkalinity							
VTRAC02	15	VV	181MR	mol eq/.m/s	Merid	Mass-Weighted Transp of	↪
↪Alkalinity							
WTRAC02	15	WM	MR	mol eq/.m/s	Vert	Mass-Weighted Transp of	↪
↪Alkalinity							
ADVrTr02	15	WM	LR	mol eq/.m^3/s	Vertical	Advective Flux of	↪
↪Alkalinity							
ADVxTr02	15	UU	186MR	mol eq/.m^3/s	Zonal	Advective Flux of	↪
↪Alkalinity							
ADVyTr02	15	VV	185MR	mol eq/.m^3/s	Meridional	Advective Flux of	↪
↪Alkalinity							
DfErTr02	15	WM	LR	mol eq/.m^3/s	Vertical	Diffusive Flux of Alkalinity	↪
↪(Explicit part)							
DIFxTr02	15	UU	189MR	mol eq/.m^3/s	Zonal	Diffusive Flux of	↪
↪Alkalinity							
DIFyTr02	15	VV	188MR	mol eq/.m^3/s	Meridional	Diffusive Flux of	↪
↪Alkalinity							
DfIrTr02	15	WM	LR	mol eq/.m^3/s	Vertical	Diffusive Flux of Alkalinity	↪
↪(Implicit part)							
TRAC03	15	SM P	MR	mol P/m	Mass-Weighted	Phosphate	
UTRAC03	15	UU	193MR	mol P/m.m/s	Zonal	Mass-Weighted Transp of	↪
↪Phosphate							
VTRAC03	15	VV	192MR	mol P/m.m/s	Merid	Mass-Weighted Transp of	↪
↪Phosphate							
WTRAC03	15	WM	MR	mol P/m.m/s	Vert	Mass-Weighted Transp of	↪
↪Phosphate							
ADVrTr03	15	WM	LR	mol P/m.m^3/s	Vertical	Advective Flux of Phosphate	
ADVxTr03	15	UU	197MR	mol P/m.m^3/s	Zonal	Advective Flux of Phosphate	
ADVyTr03	15	VV	196MR	mol P/m.m^3/s	Meridional	Advective Flux of Phosphate	
DfErTr03	15	WM	LR	mol P/m.m^3/s	Vertical	Diffusive Flux of Phosphate	↪
↪(Explicit part)							
DIFxTr03	15	UU	200MR	mol P/m.m^3/s	Zonal	Diffusive Flux of Phosphate	

<-Name-> Levs <-parsing code-> <-- Units --> <- Tile (max=80c)							

DIFyTr03	15	VV	199MR	mol P/m.m^3/s	Meridional	Diffusive Flux of Phosphate	
DfIrTr03	15	WM	LR	mol P/m.m^3/s	Vertical	Diffusive Flux of Phosphate	↪
↪(Implicit part)							
TRAC04	15	SM P	MR	mol P/m	Mass-Weighted	Dissolved Organic	↪
↪Phosphorus							
UTRAC04	15	UU	204MR	mol P/m.m/s	Zonal	Mass-Weighted Transp of	↪
↪Dissolved Organic Phosphorus							
VTRAC04	15	VV	203MR	mol P/m.m/s	Merid	Mass-Weighted Transp of	↪
↪Dissolved Organic Phosphorus							

(continues on next page)

(continued from previous page)

WTRAC04	15	WM	MR	mol P/m.m/s	Vert	Mass-Weighted Transp of	↵
↵Dissolved Organic Phosphorus							
ADVrTr04	15	WM	LR	mol P/m.m^3/s	Vertical	Advective Flux of	↵
↵Dissolved Organic Phosphorus							
ADVxTr04	15	UU	208MR	mol P/m.m^3/s	Zonal	Advective Flux of	↵
↵Dissolved Organic Phosphorus							
ADVyTr04	15	VV	207MR	mol P/m.m^3/s	Meridional	Advective Flux of	↵
↵Dissolved Organic Phosphorus							
DfErTr04	15	WM	LR	mol P/m.m^3/s	Vertical	Diffusive Flux of Dissolved	↵
↵Organic Phosphorus (Explicit part)							
DIFxTr04	15	UU	211MR	mol P/m.m^3/s	Zonal	Diffusive Flux of	↵
↵Dissolved Organic Phosphorus							
DIFyTr04	15	VV	210MR	mol P/m.m^3/s	Meridional	Diffusive Flux of	↵
↵Dissolved Organic Phosphorus							
DfErTr04	15	WM	LR	mol P/m.m^3/s	Vertical	Diffusive Flux of Dissolved	↵
↵Organic Phosphorus (Implicit part)							
TRAC05	15	SM P	MR	mol O/m	Mass-Weighted	Dissolved Oxygen	
UTRAC05	15	UU	215MR	mol O/m.m/s	Zonal	Mass-Weighted Transp of	↵
↵Dissolved Oxygen							
VTRAC05	15	VV	214MR	mol O/m.m/s	Merid	Mass-Weighted Transp of	↵
↵Dissolved Oxygen							
WTRAC05	15	WM	MR	mol O/m.m/s	Vert	Mass-Weighted Transp of	↵
↵Dissolved Oxygen							
ADVrTr05	15	WM	LR	mol O/m.m^3/s	Vertical	Advective Flux of	↵
↵Dissolved Oxygen							
ADVxTr05	15	UU	219MR	mol O/m.m^3/s	Zonal	Advective Flux of	↵
↵Dissolved Oxygen							
ADVyTr05	15	VV	218MR	mol O/m.m^3/s	Meridional	Advective Flux of	↵
↵Dissolved Oxygen							
DfErTr05	15	WM	LR	mol O/m.m^3/s	Vertical	Diffusive Flux of Dissolved	↵
↵Oxygen (Explicit part)							
DIFxTr05	15	UU	222MR	mol O/m.m^3/s	Zonal	Diffusive Flux of	↵
↵Dissolved Oxygen							
DIFyTr05	15	VV	221MR	mol O/m.m^3/s	Meridional	Diffusive Flux of	↵
↵Dissolved Oxygen							
DfErTr05	15	WM	LR	mol O/m.m^3/s	Vertical	Diffusive Flux of Dissolved	↵
↵Oxygen (Implicit part)							

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/GM parameterization 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 implementation 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 τ) of the form:

$$\nabla \cdot \kappa_\rho \mathbf{K}_{\text{Redi}} \nabla \tau$$

where κ_ρ is the along isopycnal diffusivity and \mathbf{K}_{Redi} is a rank 2 tensor that projects the gradient of τ onto the isopycnal surface. The unapproximated projection tensor is:

$$\mathbf{K}_{\text{Redi}} = \frac{1}{1 + |\mathbf{S}|^2} \begin{pmatrix} 1 + S_y^2 & -S_x S_y & S_x \\ -S_x S_y & 1 + S_x^2 & S_y \\ S_x & S_y & |S|^2 \end{pmatrix}$$

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, $|S| \ll 1$. The Redi projection tensor then becomes:

$$\mathbf{K}_{\text{Redi}} = \begin{pmatrix} 1 & 0 & S_x \\ 0 & 1 & S_y \\ S_x & S_y & |S|^2 \end{pmatrix}$$

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, \mathbf{u}^* . The divergence of this advective flux is added to the tracer tendency equation (on the rhs):

$$-\nabla \cdot \tau \mathbf{u}^*$$

The bolus velocity \mathbf{u}^* is defined as the rotational of a streamfunction $\mathbf{F}^* = (F_x^*, F_y^*, 0)$:

$$\mathbf{u}^* = \nabla \times \mathbf{F}^* = \begin{pmatrix} -\partial_z F_y^* \\ +\partial_z F_x^* \\ \partial_x F_y^* - \partial_y F_x^* \end{pmatrix},$$

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 :

$$\begin{aligned} F_x^* &= -\kappa_{GM} S_y \\ F_y^* &= \kappa_{GM} S_x \end{aligned}$$

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:

$$\mathbf{u}^* = \begin{pmatrix} u^* \\ v^* \\ w^* \end{pmatrix} = \begin{pmatrix} -\partial_z(\kappa_{GM} S_x) \\ -\partial_z(\kappa_{GM} S_y) \\ \partial_x(\kappa_{GM} S_x) + \partial_y(\kappa_{GM} S_y) \end{pmatrix}$$

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:

$$\begin{pmatrix} GM_PsiX \\ GM_PsiY \end{pmatrix} = \begin{pmatrix} \kappa_{GM} S_x \\ \kappa_{GM} S_y \end{pmatrix} = \begin{pmatrix} F_y^* \\ -F_x^* \end{pmatrix}.$$

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:

$$\begin{aligned} \mathbf{u}^* \tau &= \begin{pmatrix} -\partial_z(\kappa_{GM} S_x) \tau \\ -\partial_z(\kappa_{GM} S_y) \tau \\ (\partial_x \kappa_{GM} S_x + \partial_y \kappa_{GM} S_y) \tau \end{pmatrix} \\ &= \begin{pmatrix} -\partial_z(\kappa_{GM} S_x \tau) \\ -\partial_z(\kappa_{GM} S_y \tau) \\ \partial_x(\kappa_{GM} S_x \tau) + \partial_y(\kappa_{GM} S_y \tau) \end{pmatrix} + \begin{pmatrix} \kappa_{GM} S_x \partial_z \tau \\ \kappa_{GM} S_y \partial_z \tau \\ -\kappa_{GM} S_x \partial_x \tau - \kappa_{GM} S_y \partial_y \tau \end{pmatrix} \end{aligned}$$

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:

$$\mathbf{u}^* \tau = -\kappa_{GM} \mathbf{K}_{GM} \nabla \tau$$

where

$$\mathbf{K}_{GM} = \begin{pmatrix} 0 & 0 & -S_x \\ 0 & 0 & -S_y \\ S_x & S_y & 0 \end{pmatrix}$$

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 isoneutral mixing scheme:

$$\kappa_\rho \mathbf{K}_{Redi} \nabla \tau - \mathbf{u}^* \tau = (\kappa_\rho \mathbf{K}_{Redi} + \kappa_{GM} \mathbf{K}_{GM}) \nabla \tau$$

In the instance that $\kappa_{GM} = \kappa_\rho$ then

$$\kappa_\rho \mathbf{K}_{Redi} + \kappa_{GM} \mathbf{K}_{GM} = \kappa_\rho \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 2S_x & 2S_y & |S|^2 \end{pmatrix}$$

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*)

σ_x : **SlopeX** (argument on entry)

σ_y : **SlopeY** (argument on entry)

σ_z : **SlopeY** (argument)

S_x : **SlopeX** (argument on exit)

S_y : **SlopeY** (argument on exit)

8.4.1.4 Variable κ_{GM}

[VMHS97] suggest making the eddy coefficient, κ_{GM} , a function of the Eady growth rate, $|f|/\sqrt{Ri}$. The formula involves a non-dimensional constant, α , and a length-scale L :

$$\kappa_{GM} = \alpha L^2 \frac{\overline{|f|}}{\sqrt{Ri}}^z$$

where the Eady growth rate has been depth averaged (indicated by the over-line). A local Richardson number is defined $Ri = N^2/(\partial u/\partial z)^2$ which, when combined with thermal wind gives:

$$\frac{1}{Ri} = \frac{(\frac{\partial u}{\partial z})^2}{N^2} = \frac{(\frac{g}{f\rho_o}|\nabla\sigma|)^2}{N^2} = \frac{M^4}{|f|^2 N^2}$$

where M^2 is defined $M^2 = \frac{g}{\rho_o}|\nabla\sigma|$. Substituting into the formula for κ_{GM} gives:

$$\kappa_{GM} = \alpha L^2 \frac{\overline{M^2}^z}{N} = \alpha L^2 \frac{\overline{M^2}}{N^2} N^z = \alpha L^2 |S| N^z$$

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*)

σ_x, s_x : **SlopeX** (argument)

σ_y, s_y : **SlopeY** (argument)

σ_z : **dSigmadRReal** (argument)

z_σ^* : **dRdSigmaLtd** (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

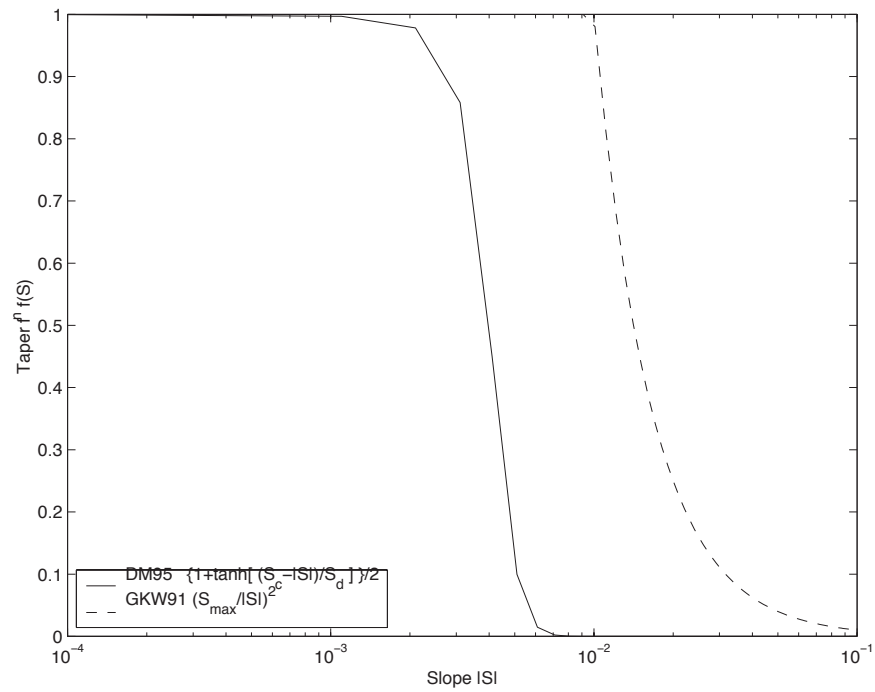


Figure 8.6: Taper functions used in GKW91 and DM95.

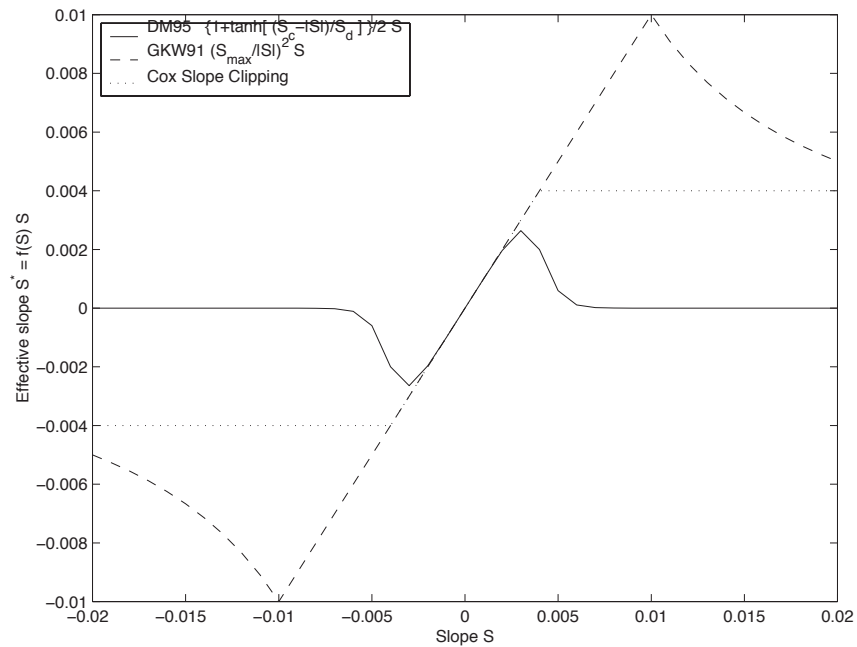


Figure 8.7: Effective slope as a function of 'true' slope using Cox slope clipping, GKW91 limiting and DM95 limiting.

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:

$$\begin{aligned} |\nabla\sigma| &= \sqrt{\sigma_x^2 + \sigma_y^2} \\ S_{lim} &= -\frac{|\nabla\sigma|}{S_{max}} \quad \text{where } S_{max} \text{ is a parameter} \\ \sigma_z^* &= \min(\sigma_z, S_{lim}) \\ [s_x, s_y] &= -\frac{[\sigma_x, \sigma_y]}{\sigma_z^*} \end{aligned}$$

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:

$$[s_x, s_y] = -\frac{[\sigma_x, \sigma_y]}{\sigma_z}$$

while in the limited regions ($\sigma_z > S_{lim}$) the slopes become:

$$[s_x, s_y] = \frac{[\sigma_x, \sigma_y]}{|\nabla\sigma|/S_{max}}$$

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 $\kappa_\rho \mathbf{K}_{33} = \kappa_\rho S_{max}^2$) 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, it 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*.

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:

$$f_2(z) = \frac{1}{2} \left(1 + \sin\left(\pi \frac{z}{D} - \frac{\pi}{2}\right) \right)$$

where $D = L_\rho |S|$ is a depth-scale and $L_\rho = c/f$ with $c = 2 \text{ m s}^{-1}$. 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 **GM_taper_scheme = 'ldd97'** in *data.gmredi*.

8.4.1.10 Package Reference

-----<Name> Levs < parsing code> <-- Units --> <- Tile (max=80c)-----					
GM_VisbK	1	SM P M1	m ² /s		Mixing coefficient from Visbeck et al. ↵
↵		parameterization			
GM_Kux	15	UU P 177MR	m ² /s		K_11 element (U.point, X.dir) of GM- ↵
↵		Redi tensor			
GM_Kvy	15	VV P 176MR	m ² /s		K_22 element (V.point, Y.dir) of GM- ↵
↵		Redi tensor			
GM_Kuz	15	UU 179MR	m ² /s		K_13 element (U.point, Z.dir) of GM- ↵
↵		Redi tensor			
GM_Kvz	15	VV 178MR	m ² /s		K_23 element (V.point, Z.dir) of GM- ↵
↵		Redi tensor			
GM_Kwx	15	UM 181LR	m ² /s		K_31 element (W.point, X.dir) of GM- ↵
↵		Redi tensor			
GM_Kwy	15	VM 180LR	m ² /s		K_32 element (W.point, Y.dir) of GM- ↵
↵		Redi tensor			
GM_Kwz	15	WM P LR	m ² /s		K_33 element (W.point, Z.dir) of GM- ↵
↵		Redi tensor			
GM_PsiX	15	UU 184LR	m ² /s		GM Bolus transport stream-function : ↵
↵		X component			
GM_PsiY	15	VV 183LR	m ² /s		GM Bolus transport stream-function : ↵
↵		Y component			
GM_KuzTz	15	UU 186MR	degC.m ³ /s		Redi Off-diagonal Temperature flux: X ↵
↵		component			
GM_KvzTz	15	VV 185MR	degC.m ³ /s		Redi Off-diagonal Temperature flux: Y ↵
↵		component			

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 [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 `hbl` 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 constraints;
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 `ghat` 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. [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 ([sec:pkg:kpp:comp]), the settings and choices of runtime parameters ([sec:pkg:kpp:runtime]), more detailed description of equations to which these parameters relate ([sec:pkg:kpp:equations]), and key subroutines where they are used ([sec:pkg:kpp:flowchart]), and diagnostics output of KPP-derived diffusivities, viscosities and boundary-layer/mixed-layer depths ([sec:pkg:kpp:diagnostics]).

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 [sec:buildingCode]).

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 8.4 summarizes them.

Table 8.4: CPP flags for KPP

CPP option	Description
<code>_KPP_RL</code>	
<code>FRUGAL_KPP</code>	
<code>KPP_SMOOTH_SHSQ</code>	
<code>KPP_SMOOTH_DVSQ</code>	
<code>KPP_SMOOTH_DENS</code>	
<code>KPP_SMOOTH_VISC</code>	
<code>KPP_SMOOTH_DIFF</code>	
<code>KPP_ESTIMATE_UREF</code>	
<code>INCLUDE_DIAGNOSTICS_INTERFACE_CODE</code>	
<code>KPP_GHAT</code>	
<code>EXCLUDE_KPP_SHEAR_MIX</code>	

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:

<code>implicitViscosity = .TRUE.</code>	enable implicit vertical viscosity
<code>implicitDiffusion = .TRUE.</code>	enable implicit vertical diffusion

Package flags and parameters

Table 8.5 summarizes the runtime flags that are set in `data.pkg`, and their default values.

Table 8.5: Runtime flags for KPP

Flag/parameter	default	Description
<i>I/O related parameters</i>		
<code>kpp_freq</code>	<code>deltaTClock</code>	Recomputation frequency for KPP fields
<code>kpp_dumpFreq</code>	<code>dumpFreq</code>	Dump frequency of KPP field snapshots
<code>kpp_taveFreq</code>	<code>taveFreq</code>	Averaging and dump frequency of KPP fields
<code>KPPmixingMaps</code>	<code>.FALSE.</code>	include KPP diagnostic maps in STDOUT

Continued on next page

Table 8.5 – continued from previous page

Flag/parameter	default	Description
KPPwriteState	.FALSE.	write KPP state to file
KPP_ghatUseTotalDiffus	.FALSE.	if .T. compute non-local term using
		total vertical diffusivity
		if .F. use KPP vertical diffusivity
<i>General KPP parameters</i>		
minKPPhbl	delRc (1)	Minimum boundary layer depth
epsilon	0.1	nondimensional extent of the surface layer
vonk	0.4	von Karman constant
dB_dz	5.2E-5 s ⁻²	maximum dB/dz in mixed layer hMix
concs	98.96	
concv	1.8	
<i>Boundary layer parameters (S/R bldepth)</i>		
Ricr	0.3	critical bulk Richardson number
cekman	0.7	coefficient for Ekman depth
cmonob	1.0	coefficient for Monin-Obukhov depth
concv	1.8	ratio of interior to entrainment depth buoyancy frequency
hbf	1.0	fraction of depth to which absorbed solar radiation contributes to surface buoyancy forcing
Vtc		non-dim. coeff. for velocity scale of turbulent velocity shear (= function of concv,concs,epsilon,vonk,Ricr)
<i>Boundary layer mixing parameters (S/R blmix)</i>		
cstar	10.	proportionality coefficient for non-local transport
cg		non-dimensional coefficient for counter-gradient term (= function of cstar,vonk,concs,epsilon)
<i>Interior mixing parameters (S/R Ri_iwmix)</i>		
Riinfy	0.7	gradient Richardson number limit for shear instability
BVDQcon	-0.2E-4 s ⁻²	Brunt-Väisälä squared
difm0	0.005 m ² s ⁻¹	viscosity max. due to shear instability
difs0	0.005 m ² /s	tracer diffusivity max. due to shear instability
dift0	0.005 m ² /s	heat diffusivity max. due to shear instability
difmcon	0.1	viscosity due to convective instability
difscon	0.1	tracer diffusivity due to convective instability
diftcon	0.1	heat diffusivity due to convective instability

Continued on next page

Table 8.5 – continued from previous page

Flag/parameter	default	Description
Rrho0	not used	limit for double diffusive density ratio
dsfmax	not used	maximum diffusivity in case of salt fingering

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 [LMD94].

KPP_CALC:

Top-level routine.

KPP_MIX:

Intermediate-level routine

BLMIX: Mixing in the boundary layer

The vertical fluxes $\overline{w\bar{x}}$ 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 γ_x that enhances the gradient-flux mixing coefficient K_x

$$\overline{w\bar{x}}(d) = -K_x \left(\frac{\partial X}{\partial z} - \gamma_x \right)$$

- *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 .

- *Nonlocal mixing term* The nonlocal transport term γ 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:

$$\left. \begin{aligned} \gamma_x &= 0 \\ \gamma_m &= 0 \\ \gamma_s &= C_s \frac{\overline{ws_0}}{w_s(\sigma)h} \\ \gamma_\theta &= C_s \frac{\overline{w\theta_0 + w\theta_R}}{w_s(\sigma)h} \end{aligned} \right\} \zeta < 0$$

$$\zeta \geq 0$$

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/unstable 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 |
C |-- statekpp: o compute all EOS/density-related arrays
C |               o uses S/R FIND_ALPHA, FIND_BETA, FIND_RHO
C |
C |-- kppmix
C |   |-- ri_iwmix (compute interior mixing coefficients due to constant
C |   |               internal wave activity, static instability,
C |   |               and local shear instability).
C |   |
C |   |-- bldepth (diagnose boundary layer depth)
C |   |
C |   |-- blmix (compute boundary layer diffusivities)
C |   |
C |   |-- enhance (enhance diffusivity at interface kbl - 1)
C |   o
C |
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:

```

-----
<-Name->|Levs|grid|<--  Units    -->|<- Tile (max=80c)
-----
KPPviscA| 23 |SM |m^2/s          |KPP vertical eddy viscosity coefficient
KPPdiffS| 23 |SM |m^2/s          |Vertical diffusion coefficient for salt & tracers
KPPdiffT| 23 |SM |m^2/s          |Vertical diffusion coefficient for heat
KPPghat | 23 |SM |s/m^2          |Nonlocal transport coefficient
KPPhbl  | 1  |SM |m              |KPP boundary layer depth, bulk Ri criterion
KPPmld  | 1  |SM |m              |Mixed layer depth, dT=.8degC density criterion
KPPfrac | 1  |SM |              |Short-wave flux fraction penetrating mixing layer

```

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 ϵ , and the stratification N , as

$$L_O^2 \approx \epsilon N^{-3}. \quad (8.1)$$

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, δ_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 [WG94][SG94][Mou96].

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 ϵ using ([eq:pkg:kl10:Lo]). 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 ϵ_v will start to be exaggerated, particularly if the run is 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.

(see Section [sec:buildingCode]).

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:

<code>implicitViscosity = .TRUE.</code>	enable implicit vertical viscosity
<code>implicitDiffusion = .TRUE.</code>	enable implicit vertical diffusion

Package flags and parameters

Table 8.6 summarizes the runtime flags that are set in `data.kl10`, and their default values.

Table 8.6: KL10 runtime parameters.

Flag/parameter	default	Description
<code>KLviscMax</code>	$300 \text{ m}^2 \text{ s}^{-1}$	Maximum viscosity the scheme will ever give (useful for stability)
<code>KLdumpFreq</code>	<code>dumpFreq</code>	Dump frequency of KL10 field snapshots
<code>KLtaveFreq</code>	<code>taveFreq</code>	Averaging and dump frequency of KL10 fields
<code>KLwriteState</code>	<code>.FALSE.</code>	write KL10 state to file

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	m ² /s	KL10 vertical eddy viscosity coefficient
KLdiffKr	Nr	SM	m ² /s	Vertical diffusion coefficient for salt, ↪
temperature, & tracers				
KLeps		Nr	SM	m ³ /s ³
				Turbulence dissipation estimate.

8.4.5.6 References

Klymak and Legg, 2010, *Oc. Modell.*

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/thseice*, 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 bulkf 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 **Qnet** and **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 **Qnet** and **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 seaice 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 γ is a constant.

The saturated vapor pressure is calculate (QQ ref):

$$q_{sat} = \frac{a}{p_o} e^{L(b - \frac{c}{T_{srf}})}$$

where a, b, c are constants, T_{srf} 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:

$$\Delta T = T_{air} - T_{srf} + \alpha h_T$$

where α 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

$$\frac{\kappa}{\ln(z_{ref}/z_{rou})}$$

where κ 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:

$$\begin{aligned} u^* &= c_u u_s \\ T^* &= c_T \Delta T \\ q^* &= c_q \Delta q \end{aligned}$$

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 ($\Upsilon < 0$):

$$\begin{aligned} \psi_m &= 2\ln(0.5(1 + \chi)) + \ln(0.5(1 + \chi^2)) - 2 \tan^{-1} \chi + \pi/2 \\ \psi_s &= 2\ln(0.5(1 + \chi^2)) \end{aligned}$$

where

$$\Upsilon = \frac{\kappa g z_{ref}}{u^{*2}} \left(\frac{T^*}{T_o} + \frac{q^*}{1/\gamma + q_a} \right)$$

and $\chi = (1 - 16\Upsilon)^{1/2}$.

The coefficients are updated through 5 iterations as:

$$\begin{aligned} c_u &= \frac{\hat{c}_u}{1 + \hat{c}_u(\lambda - \psi_m)/\kappa} \\ c_T &= \frac{\hat{c}_T}{1 + \hat{c}_T(\lambda - \psi_s)/\kappa} \\ c_q &= \hat{c}_T' \end{aligned}$$

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

$$Q_{lw}^{up} = \epsilon \sigma T_{surf}^4$$

where ϵ is emissivity (which can be different for open ocean, ice and snow), σ is Stefan-Boltzman constant.

We calculate the derivatives of the three above functions with respect to surface temperature

$$\begin{aligned}\frac{dQ_s}{dT} &= \rho_{air} c_{p_{air}} u_s c_u c_T \\ \frac{dQ_l}{dT} &= \frac{\rho_{air} L^2 u_s c_u c_q c}{T_{surf}^2} \\ \frac{dQ_{lw}^{up}}{dT} &= 4\epsilon\sigma T_{surf}^3\end{aligned}$$

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

Bryan F.O., B.G Kauffman, W.G. Large, P.R. Gent, 1996: The NCAR CSM flux coupler. Technical note TN-425+STR, NCAR.

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 [sec:pkg:exf:config]). Runtime options, flags, filenames and field-related dates/times are set in `data.exf` (Section [sec:pkg:exf:runtime]). A description of key subroutines is given in Section [sec:pkg:exf:subroutines]. Input fields, units and sign conventions are summarized in Section [sec:pkg:exf:fields:sub:units], and available diagnostics output is listed in Section [sec:pkg:exf:diagnostics].

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 requires the calendar package `cal` to be enabled; no additional CPP options are required.

(see Section [sec:buildingCode]).

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 8.7 summarizes these options.

Table 8.7: EXF CPP options

CPP option	Description
EXF_VERBOSE	verbose mode (recommended only for testing)
ALLOW_ATM_TEMP	compute heat/freshwater fluxes from atmos. state input
ALLOW_ATM_WIND	compute wind stress from wind speed input
ALLOW_BULKFORMULAE	is used if ALLOW_ATM_TEMP or ALLOW_ATM_WIND is enabled
EXF_READ_EVAP	read evaporation instead of computing it
ALLOW_RUNOFF	read time-constant river/glacier run-off field
ALLOW_DOWNWARD_RADIATION	compute net from downward or downward from net radiation
USE_EXF_INTERPOLATION	enable on-the-fly bilinear or bicubic interpolation of input fields
<i>used in conjunction with relaxation to prescribed (climatological) fields</i>	
ALLOW_CLIMST_RELAXATION	relaxation to 2-D SST climatology
ALLOW_CLIMSSS_RELAXATION	relaxation to 2-D SSS climatology
<i>these are set outside of EXF in CPP_OPTIONS.h</i>	
SHORTWAVE_HEATING	enable shortwave radiation
ATMOSPHERIC_LOADING	enable surface pressure forcing

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 8.8: EXF runtime options

Flag/parameter	default	Description
useExfCheckRange	.TRUE.	check range of input fields and stop if out of range
useExfYearlyFields	.FALSE.	append current year postfix of form <code>_YYYY</code> on filename
twoDigitYear	.FALSE.	instead of appending <code>_YYYY</code> append <code>YY</code>
repeatPeriod	0.0	> 0: cycle through all input fields at the same period (in seconds) = 0: use period assigned to each field
exf_offset_atemp	0.0	set to 273.16 to convert from deg. Kelvin (assumed input) to Celsius
windstressmax	2.0	max. allowed wind stress N m^{-2}
exf_albedo	0.1	surface albedo used to compute downward vs. net radiative fluxes
climtempfreeze	-1.9	???
ocean_emissivity		longwave ocean-surface emissivity
ice_emissivity		longwave seaice emissivity
snow_emissivity		longwave snow emissivity
exf_iceCd	1.63E-3	drag coefficient over sea-ice
exf_iceCe	1.63E-3	evaporation transfer coeff. over sea-ice
exf_iceCh	1.63E-3	sensible heat transfer coeff. over sea-ice
exf_scal_BulkCdn	1.0	overall scaling of neutral drag coeff.

Continued on next page

Table 8.8 – continued from previous page

Flag/parameter	default	Description
useStabilityFct_overIce	.FALSE.	compute turbulent transfer coeff. over sea-ice
readStressOnAgrid	.FALSE.	read wind-streess located on model-grid, A-grid point
readStressOnCgrid	.FALSE.	read wind-streess located on model-grid, C-grid point
useRelativeWind	.FALSE.	subtract [U/V]VEL or [U/V]ICE from U/V]WIND before computing [U/V]STRESS
zref	10.0	reference height
hu	10.0	height of mean wind
ht	2.0	height of mean temperature and rel. humidity
umin	0.5	minimum absolute wind speed for computing Cd
atmrho	1.2	mean atmospheric density [kg/m^3]
atmcp	1005.0	mean atmospheric specific heat [J/kg/K]
cdrag_[n]	???	n = 1,2,3; parameters for drag coeff. function
cstanton_[n]	???	n = 1,2; parameters for Stanton number function
cdalton	???	parameter for Dalton number function
flamb	2500000.0	latent heat of evaporation [J/kg]
flami	334000.0	latent heat of melting of pure ice [J/kg]
zolmin	-100.0	minimum stability parameter
cvapor_fac	640380.0	
cvapor_exp	5107.4	
cvapor_fac_ice	11637800.0	
cvapor_fac_ice	5897.8	
humid_fac	0.606	parameter for virtual temperature calculation
gamma_blk	0.010	adiabatic lapse rate
saltsat	0.980	reduction of saturation vapor pressure over salt-water
psim_fac	5.0	
exf_monFreq	monitorFreq	output frequency [s]
exf_iprec	32	precision of input fields (32-bit or 64-bit)
exf_yftype	'RL'	precision of arrays ('RL' vs. 'RS')

Field attributes

All EXF fields are listed in Section [sec:pkg:exf:fields:sub:units]. Each field has a number of attributes which can be customized. They are summarized in Table [tab:pkg:exf:runtime:sub:attributes]. 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 8.9: EXF runtime attributes Note there is one exception for the default of `atempconst = celsius2K = 273.16`

attribute	Default	Description
<i>field</i> file	‘ ‘	filename; if left empty no file will be read; <code>const</code> will be used instead
<i>field</i> const	0.0	constant that will be used if no file is read
<i>field</i> startdate1	0.0	format: YYYYMMDD; start year (YYYY), month (MM), day (YY) of field to determine record number
<i>field</i> startdate2	0.0	format: HHMMSS; start hour (HH), minute (MM), second(SS) of field to determine record number
<i>field</i> period	0.0	interval in seconds between two records
<i>exf_inscal_field</i>		optional rescaling of input fields to comply with EXF units
<i>exf_outscal_field</i>		optional rescaling of EXF fields when mapped onto MITgcm fields
<i>used in conjunction with</i> EXF_USE_INTERPOLATION		
<i>field</i> _lon0	xgOrigin+delX/ 2	starting longitude of input
<i>field</i> _lon_inc	delX	increment in longitude of input
<i>field</i> _lat0	ygOrigin+delY/ 2	starting latitude of input
<i>field</i> _lat_inc	delY	increment in latitude of input
<i>field</i> _nlon	Nx	number of grid points in longitude of input
<i>field</i> _nlat	Ny	number of grid points in longitude of input

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`:

```
hfluxfile      = 'ncep_qnet.bin',
hfluxstartdate1 = 19920101,
hfluxstartdate2 = 000000,
hfluxperiod    = 2592000.0,
hflux_lon0     = 2
hflux_lon_inc  = 4
hflux_lat0     = -78
hflux_lat_inc  = 39*4
hflux_nlon     = 90
hflux_nlat     = 40
```

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`.

c		
c	field	:: Description
c		

c	ustress	:: Zonal surface wind stress in N/m^2
c		> 0 for increase in uVel, which is west to
c		east for cartesian and spherical polar grids
c		Typical range: -0.5 < ustress < 0.5
c		Southwest C-grid U point
c		Input field

c	vstress	:: Meridional surface wind stress in N/m^2
c		> 0 for increase in vVel, which is south to
c		north for cartesian and spherical polar grids
c		Typical range: -0.5 < vstress < 0.5
c		Southwest C-grid V point
c		Input field

c	hs	:: sensible heat flux into ocean in W/m^2
c		> 0 for increase in theta (ocean warming)

c	hl	:: latent heat flux into ocean in W/m^2
c		> 0 for increase in theta (ocean warming)

c	hflux	:: Net upward surface heat flux in W/m^2
c		(including shortwave)
c		hflux = latent + sensible + lwflux + swflux
c		> 0 for decrease in theta (ocean cooling)
c		Typical range: -250 < hflux < 600
c		Southwest C-grid tracer point
c		Input field

c	sflux	:: Net upward freshwater flux in m/s
c		sflux = evap - precip - runoff
c		> 0 for increase in salt (ocean salinity)
c		Typical range: -1e-7 < sflux < 1e-7
c		Southwest C-grid tracer point
c		Input field

c	swflux	:: Net upward shortwave radiation in W/m^2
c		swflux = - (swdown - ice and snow absorption - reflected)
c		> 0 for decrease in theta (ocean cooling)
c		Typical range: -350 < swflux < 0
c		Southwest C-grid tracer point
c		Input field

c	uwind	:: Surface (10-m) zonal wind velocity in m/s
c		> 0 for increase in uVel, which is west to
c		east for cartesian and spherical polar grids
c		Typical range: -10 < uwind < 10

(continues on next page)

(continued from previous page)

c		Southwest C-grid U point
c		Input or input/output field

c	vwind	:: Surface (10-m) meridional wind velocity in m/s
c		> 0 for increase in vVel, which is south to
c		north for cartesian and spherical polar grids
c		Typical range: -10 < vwind < 10
c		Southwest C-grid V point
c		Input or input/output field

c	wspeed	:: Surface (10-m) wind speed in m/s
c		>= 0 sqrt(u^2+v^2)
c		Typical range: 0 < wspeed < 10
c		Input or input/output field

c	atemp	:: Surface (2-m) air temperature in deg K
c		Typical range: 200 < atemp < 300
c		Southwest C-grid tracer point
c		Input or input/output field

c	aqh	:: Surface (2m) specific humidity in kg/kg
c		Typical range: 0 < aqh < 0.02
c		Southwest C-grid tracer point
c		Input or input/output field

c	lwflux	:: Net upward longwave radiation in W/m^2
c		lwflux = - (lwdown - ice and snow absorption - emitted)
c		> 0 for decrease in theta (ocean cooling)
c		Typical range: -20 < lwflux < 170
c		Southwest C-grid tracer point
c		Input field

c	evap	:: Evaporation in m/s
c		> 0 for increase in salt (ocean salinity)
c		Typical range: 0 < evap < 2.5e-7
c		Southwest C-grid tracer point
c		Input, input/output, or output field

c	precip	:: Precipitation in m/s
c		> 0 for decrease in salt (ocean salinity)
c		Typical range: 0 < precip < 5e-7
c		Southwest C-grid tracer point
c		Input or input/output field

c	snowprecip	:: snow in m/s
c		> 0 for decrease in salt (ocean salinity)
c		Typical range: 0 < precip < 5e-7
c		Input or input/output field

c	runoff	:: River and glacier runoff in m/s
c		> 0 for decrease in salt (ocean salinity)
c		Typical range: 0 < runoff < ????
c		Southwest C-grid tracer point
c		Input or input/output field
c		!!! WATCH OUT: Default exf_inscal_runoff !!!
c		!!! in exf_readparms.F is not 1.0 !!!

(continues on next page)

(continued from previous page)

```

c      swdown      :: Downward shortwave radiation in W/m^2
c                  | > 0 for increase in theta (ocean warming)
c                  | Typical range: 0 < swdown < 450
c                  | Southwest C-grid tracer point
c                  | Input/output field
c-----
c      lwdown      :: Downward longwave radiation in W/m^2
c                  | > 0 for increase in theta (ocean warming)
c                  | Typical range: 50 < lwdown < 450
c                  | Southwest C-grid tracer point
c                  | Input/output field
c-----
c      apressure   :: Atmospheric pressure field in N/m^2
c                  | > 0 for ???
c                  | Typical range: ??? < apressure < ???
c                  | Southwest C-grid tracer point
c                  | Input field
c-----

```

8.4.7.6 Key subroutines

Top-level routine: `exf_getforcing.F`

```

C      !CALLING SEQUENCE:
C      ...
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      |
C      |-- exf_getffields <- this one does almost everything
C      |   | 1. reads in fields, either flux or atmos. state,
C      |   |     depending on CPP options (for each variable two fields
C      |   |     consecutive in time are read in and interpolated onto
C      |   |     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
C      |   |     (atemp, agh, precip, swflux, swdown, uwind, vwind).
C      |   |     If forcing and control are fluxes, then
C      |   |     controls are added later.
C      |   o
C      |
C      |-- exf_radiation
C      |   | Compute net or downwelling radiative fluxes via
C      |   | Stefan-Boltzmann law in case only one is known.
C      |   o
C      |-- exf_wind
C      |   | Computes wind speed and stresses, if required.
C      |   o
C      |
C      |-- exf_bulkformulae
C      |   | Compute air-sea buoyancy fluxes from
C      |   | atmospheric state following Large and Pond, JPO, 1981/82

```

(continues on next page)

(continued from previous page)

```

c | o
c |
c |-- < hflux is sum of sensible, latent, longwave rad. >
c |-- < sflux is sum of evap. minus precip. minus runoff >
c |
c |-- exf_getsurfacefluxes
c |   If forcing and control is flux, then the
c |   control vector anomalies are read here via ctrl_get_gen
c |   (hflux, sflux, ustress, vstress)
c |
c |-- < update tile edges here >
c |
c |-- exf_check_range
c |   |   Check whether read fields are within assumed range
c |   |   (may capture mismatches in units)
c |   o
c |
c |-- < add shortwave to hflux for diagnostics >
c |
c |-- exf_diagnostics_fill
c |   |   Do EXF-related diagnostics output here.
c |   o
c |
c |-- exf_mapfields
c |   |   Forcing fields from exf package are mapped onto
c |   |   mitgcm forcing arrays.
c |   |   Mapping enables a runtime rescaling of fields
c |   o
C o

```

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 [sec:pkg:diagnostics]). Available output fields are summarized below.

```

-----+-----+-----+-----+-----+
<-Name->|Levs|grid|<--  Units  -->|<- Tile (max=80c)
-----+-----+-----+-----+-----+
EXFhs   | 1 | SM | W/m^2          | Sensible heat flux into ocean, >0 increases_
↪theta
EXFhl   | 1 | SM | W/m^2          | Latent heat flux into ocean, >0 increases theta
EXFlwnet| 1 | SM | W/m^2          | Net upward longwave radiation, >0 decreases_
↪theta
EXFswnet| 1 | SM | W/m^2          | Net upward shortwave radiation, >0 decreases_
↪theta

```

(continues on next page)

(continued from previous page)

EXFlwdn		1		SM		W/m^2		Downward longwave radiation, >0 increases theta
EXFswdn		1		SM		W/m^2		Downward shortwave radiation, >0 increases theta
EXFqnet		1		SM		W/m^2		Net upward heat flux (turb+rad), >0 decreases_
↪theta								
EXFtaux		1		SU		N/m^2		zonal surface wind stress, >0 increases uVel
EXFtauy		1		SV		N/m^2		meridional surface wind stress, >0 increases_
↪vVel								
EXFwind		1		SM		m/s		zonal 10-m wind speed, >0 increases uVel
EXFvwind		1		SM		m/s		meridional 10-m wind speed, >0 increases uVel
EXFwspeed		1		SM		m/s		10-m wind speed modulus (>= 0)
EXFatemp		1		SM		degK		surface (2-m) air temperature
EXFaqh		1		SM		kg/kg		surface (2-m) specific humidity
EXFevap		1		SM		m/s		evaporation, > 0 increases salinity
EXFpreci		1		SM		m/s		evaporation, > 0 decreases salinity
EXFsnow		1		SM		m/s		snow precipitation, > 0 decreases salinity
EXFempmr		1		SM		m/s		net upward freshwater flux, > 0 increases_
↪salinity								
EXFpress		1		SM		N/m^2		atmospheric pressure field

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
c      ( yyyyymmdd, hhmmss, leap_year, dayofweek )
c
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
c      leap_year is either equal to 1 (normal year)
c                          or equal to 2 (leap year)
c
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
c      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
c      datediff(2) = hhmmss
c      datediff(3) =      0
c      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 “initilise_fixed.F”, depending on the MITgcm release, has to be modified in the following way:

```
c      #ifdef ALLOW_CALENDAR
c      C--      Initialise the calendar package.
c      #ifdef USE_CAL_NENDITER
c      CALL cal_Init(
c          I          startTime,
c          I          endTime,
c          I          deltaTclock,
c          I          nIter0,
c          I          nEndIter,
c          I          nTimeSteps,
c          I          myThid
```

(continues on next page)

(continued from previous page)

```

c      &      )
c      #else
c          CALL cal_Init(
c              I      startTime,
c              I      endTime,
c              I      deltaTclock,
c              I      nIter0,
c              I      nTimeSteps,
c              I      myThid
c          &      )
c      #endif
c      _BARRIER
c  #endif

```

It is useful to have the CPP flag `ALLOW_CALENDAR` in order to switch from the usual MITgcm setup to the one that includes the calendar tool. The CPP flag `USE_CAL_NENDITER` 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

c	o	cal_Init	- Initialise the calendar. This is the interface to MITgcm.
c			
c	o	cal_Set	- Sets the calendar according to the user specifications.
c			
c	o	cal_GetDate	- Given the model's current timestep or the model's current time return the corresponding calendar date.
c			
c	o	cal_FullDate	- Complete a date specification (leap year and day of the week).
c			
c	o	cal_IsLeap	- Determine whether a given year is a leap year.
c			

(continues on next page)

(continued from previous page)

c	o	cal_TimePassed	- Determine the time passed between two dates.
c			
c	o	cal_AddTime	- Add a time interval either to a time interval or to a date.
c			
c	o	cal_TimeInterval	- Given a time interval return the corresponding date array.
c			
c	o	cal_SubDates	- Determine the time interval between two dates or between two time intervals.
c			
c	o	cal_ConvDate	- Decompose a date array or a time interval array into its components.
c			
c	o	cal_CopyDate	- Copy a date array or a time interval array to another array.
c			
c	o	cal_CompDates	- Compare two calendar dates or time intervals.
c			
c	o	cal_ToSeconds	- Given a time interval array return the number of seconds.
c			
c	o	cal_WeekDay	- Return the weekday as a string given the calendar date.
c			
c	o	cal_NumInts	- Return the number of time intervals between two given dates.
c			
c	o	cal_StepsPerDay	- Given an iteration number or the current integration time return the number of time steps to integrate in the current calendar day.
c			
c	o	cal_DaysPerMonth	- Given an iteration number or the current integration time return the number of days to integrate in this calendar month.
c			
c	o	cal_MonthsPerYear	- Given an iteration number or the current integration time return the number of months to integrate in the current calendar year.
c			
c	o	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).
c			
c	o	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).
c			
c	o	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 step refer to the total number of steps (1, ... , cal_IntMonths).
c			

(continues on next page)

(continued from previous page)

```

c
c      o  cal_Intsteps      - Return the number of calendar years that are
c                          affected by the current integration.
c
c      o  cal_IntDays      - Return the number of calendar days that are
c                          affected by the current integration.
c
c      o  cal_IntMonths    - Return the number of calendar months that are
c                          affected by the current integration.
c
c      o  cal_IntYears      - Return the number of calendar years that are
c                          affected by the current integration.
c
c      o  cal_nStepDay     - Return the number of time steps that can be
c                          performed during one calendar day.
c
c      o  cal_CheckDate    - Do some simple checks on a date array or on a
c                          time interval array.
c
c      o  cal_PrintError   - Print error messages according to the flags
c                          raised by the calendar routines.
c
c      o  cal_PrintDate    - Print a date array in some format suitable for
c                          MITgcm's protocol output.
c
c      o  cal_TimeStamp    - Given the time and the iteration number return
c                          the date and print all the above numbers.
c
c      o  cal_Summary      - List all the setttings of the calendar tool.

```

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 folowing 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)
-----
DIABT   |  5 |SM      ML      |K/s          |Pot. Temp.  Tendency (Mass-Weighted)
--from Diabatic Processes

```

(continues on next page)

(continued from previous page)

DIABQ	5	SM	ML	g/kg/s	Spec.Humid. Tendency (Mass-Weighted)	↵
↵from Diabatic Processes						
RADSW	5	SM	ML	K/s	Temperature Tendency due to Shortwave	↵
↵Radiation (TT_RSW)						
RADLW	5	SM	ML	K/s	Temperature Tendency due to Longwave	↵
↵Radiation (TT_RLW)						
DTCONV	5	SM	MR	K/s	Temperature Tendency due to	↵
↵Convection (TT_CNV)						
TURBT	5	SM	ML	K/s	Temperature Tendency due to	↵
↵Turbulence in PBL (TT_PBL)						
DTLS	5	SM	ML	K/s	Temperature Tendency due to Large-	
↵scale condens. (TT_LSC)						
DQCONV	5	SM	MR	g/kg/s	Spec. Humidity Tendency due to	↵
↵Convection (QT_CNV)						
TURBQ	5	SM	ML	g/kg/s	Spec. Humidity Tendency due to	↵
↵Turbulence in PBL (QT_PBL)						
DQLS	5	SM	ML	g/kg/s	Spec. Humidity Tendency due to Large-	
↵Scale Condens. (QT_LSC)						
TSR	1	SM P	U1	W/m^2	Top-of-atm. net Shortwave Radiation	↵
↵(+=dw)						
OLR	1	SM P	U1	W/m^2	Outgoing Longwave Radiation (+=up)	
RADSWG	1	SM P	L1	W/m^2	Net Shortwave Radiation at the Ground	↵
↵(+=dw)						
RADLWG	1	SM	L1	W/m^2	Net Longwave Radiation at the Ground	↵
↵(+=up)						
HFLUX	1	SM	L1	W/m^2	Sensible Heat Flux (+=up)	
EVAP	1	SM	L1	g/m^2/s	Surface Evaporation (g/m2/s)	
PRECON	1	SM P	L1	g/m^2/s	Convective Precipitation (g/m2/s)	
PRECLS	1	SM	M1	g/m^2/s	Large Scale Precipitation (g/m2/s)	
CLDFRC	1	SM P	M1	0-1	Total Cloud Fraction (0-1)	
CLDPRS	1	SM PC167M1		0-1	Cloud Top Pressure (normalized)	
CLDMAS	5	SM P	LL	kg/m^2/s	Cloud-base Mass Flux (kg/m^2/s)	
DRAG	5	SM P	LL	kg/m^2/s	Surface Drag Coefficient (kg/m^2/s)	
WINDS	1	SM P	L1	m/s	Surface Wind Speed (m/s)	
TS	1	SM	L1	K	near Surface Air Temperature (K)	
QS	1	SM P	L1	g/kg	near Surface Specific Humidity (g/kg)	
ENPREC	1	SM	M1	W/m^2	Energy flux associated with precip.	↵
↵(snow, rain Temp)						
ALBVISDF	1	SM P	L1	0-1	Surface Albedo (Visible band) (0-1)	
DWNLWG	1	SM P	L1	W/m^2	Downward Component of Longwave Flux	↵
↵at the Ground (+=dw)						
SWCLR	5	SM	ML	K/s	Clear Sky Temp. Tendency due to	↵
↵Shortwave Radiation						
LWCLR	5	SM	ML	K/s	Clear Sky Temp. Tendency due to	↵
↵Longwave Radiation						
TSRCLR	1	SM P	U1	W/m^2	Clear Sky Top-of-atm. net Shortwave	↵
↵Radiation (+=dw)						
OLRCLR	1	SM P	U1	W/m^2	Clear Sky Outgoing Longwave	↵
↵Radiation (+=up)						
SWGCLR	1	SM P	L1	W/m^2	Clear Sky Net Shortwave Radiation at	↵
↵the Ground (+=dw)						
LWGCLR	1	SM	L1	W/m^2	Clear Sky Net Longwave Radiation at	↵
↵the Ground (+=up)						
UFLUX	1	UM	184L1	N/m^2	Zonal Wind Surface Stress (N/m^2)	
VFLUX	1	VM	183L1	N/m^2	Meridional Wind Surface Stress (N/m^	
↵2)						
DTSIMPL	1	SM P	L1	K	Surf. Temp Change after 1 implicit	↵
↵time step						

(continues on next page)

8.5.1.3 Experiments and tutorials that use aim

- Global atmosphere experiment in aim.5l_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_i is the ratio of available water to field capacity. $f_i = \gamma z_i, \gamma = 0.24$ is the field capacity 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, λ is the thermal conductivity, $\lambda = 0.42 W m^{-1} K^{-1}$.

$$C_1 = C_w W_1 \gamma + C_s$$

$$C_2 = C_w W_2 \gamma + C_s$$

C_w, C_s are the heat capacity of water and dry soil respectively. $C_w = 4.2 \times 10^6 J m^{-3} K^{-1}, C_s = 1.13 \times 10^6 J m^{-3} K^{-1}$.

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.

$$\frac{dW_2}{dt} = \frac{f_1}{f_2} \frac{W_1 - W_2}{\tau}$$

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 $W_2 = W_2 + p\delta W_1 \frac{f_1}{f_2}$, 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

<-Name->	Levs	<-parsing code->	<-- Units -->	<- Tile (max=80c)	

GrdSurfT	1	SM Lg	degC	Surface Temperature over land	
GrdTemp	2	SM MG	degC	Ground Temperature at each level	
GrdEnth	2	SM MG	J/m3	Ground Enthalpy at each level	
GrdWater	2	SM P MG	0-1	Ground Water (vs Field Capacity) _	
↪Fraction at each level					
LdSnowH	1	SM P Lg	m	Snow Thickness over land	
LdSnwAge	1	SM P Lg	s	Snow Age over land	
RUNOFF	1	SM L1	m/s	Run-Off per surface unit	
EnRunOff	1	SM L1	W/m^2	Energy flux associated with run-Off	
landHFlx	1	SM Lg	W/m^2	net surface downward Heat flux over _	
↪land					
landPmE	1	SM Lg	kg/m^2/s	Precipitation minus Evaporation over _	
↪land					
ldEnFxPr	1	SM Lg	W/m^2	Energy flux (over land) associated _	
↪with Precip (snow,rain)					

8.5.2.4 References

Hansen J. et al. Efficient three-dimensional global models for climate studies: models I and II. *Monthly Weather Review*, vol.111, no.4, pp. 609-62, 1983

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, η , normalized by the cloud base mass flux, is a linear function of height, expressed as:

$$\frac{\partial \eta(z)}{\partial z} = \lambda \quad \text{or} \quad \frac{\partial \eta(P^\kappa)}{\partial P^\kappa} = -\frac{c_p}{g} \theta \lambda$$

where we have used the hydrostatic equation written in the form:

$$\frac{\partial z}{\partial P^\kappa} = -\frac{c_p}{g} \theta$$

The entrainment parameter, λ , characterizes a particular subensemble based on its detraining level, and is obtained by assuming that the level of detraining is the level of neutral buoyancy, ie., 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], λ may be written as

$$\lambda = \frac{h_B - h_D^*}{\frac{c_p}{g} \int_{P_D}^{P_B} \theta (h_D^* - h) dP^\kappa},$$

where the subscript B refers to cloud base, and the subscript D refers to the detraining 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:

$$A = \int_{P_D}^{P_B} \frac{\eta}{1 + \gamma} \left[\frac{h_c - h^*}{P^\kappa} \right] dP^\kappa$$

where γ is $\frac{L}{c_p} \frac{\partial q^*}{\partial T}$ obtained from the Claussius 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 :

$$m_B = \frac{-\frac{dA}{dt}|_{ls}}{K}$$

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 *insitu* 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):

$$\left. \frac{\partial \theta}{\partial t} \right|_c = \alpha \frac{m_B}{c_p P^\kappa} \eta \frac{\partial s}{\partial p}$$

and

$$\left. \frac{\partial q}{\partial t} \right|_c = \alpha \frac{m_B}{L} \eta \left(\frac{\partial h}{\partial p} - \frac{\partial s}{\partial p} \right)$$

where $\theta = \frac{T}{P^\kappa}$, $P = (p/p_0)$, and α 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 α 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:

$$F_{RAS}^n = \min \left[F_{RAS} + \left(1 - \frac{\Delta t_{RAS}}{\tau} \right) F_{RAS}^{n-1}, 1.0 \right]$$

where F_{RAS} is the instantaneous cloud fraction and F_{RAS}^{n-1} is the cloud fraction from the previous RAS timestep. The memory coefficient is computed using a RAS cloud timescale, τ , 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:

$$F_{LS} = \min \left[\left(\frac{RH - RH_c}{1 - RH_c} \right)^2, 1.0 \right]$$

where

$$RH_c = 1 - s(1-s)(2-s) \frac{r}{s} = \frac{p}{p_{\text{surf}}} \frac{r}{s} = \left(\frac{p}{p_{\text{surf}}} \right) RH_{\text{min}} = 0.75 = 0.573285.$$

These cloud fractions are suppressed, however, in regions where the convective sub-cloud layer is conditionally unstable. The functional form of RH_c is shown in Figure 8.8

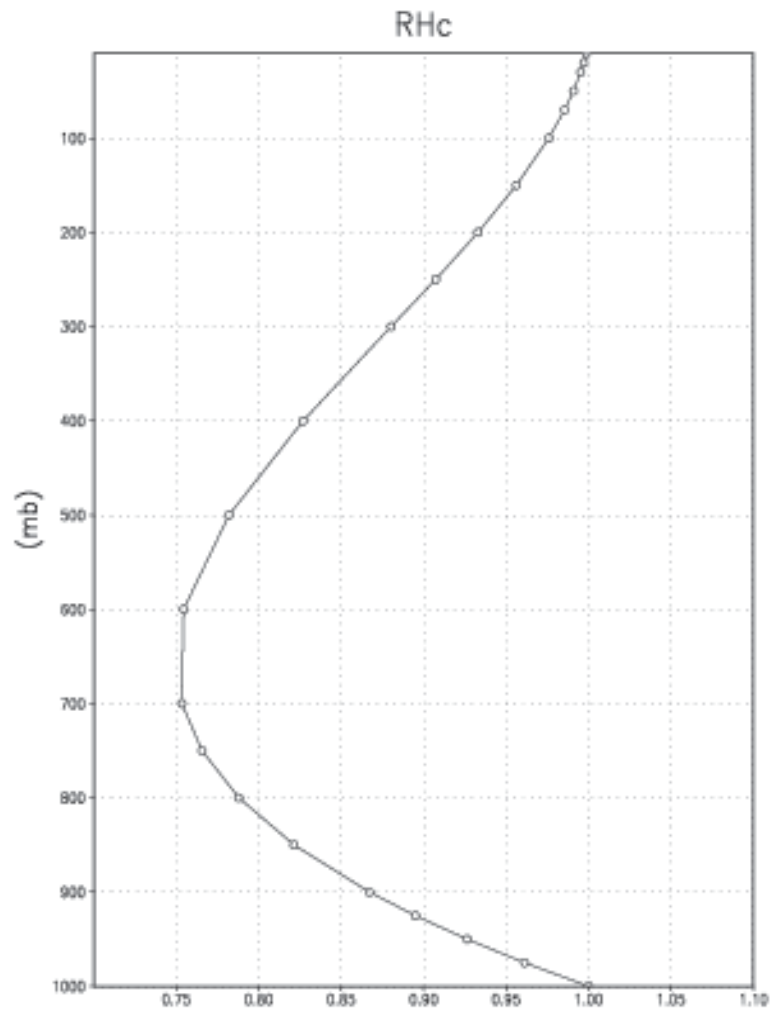


Figure 8.8: Critical Relative Humidity for Clouds.

The total cloud fraction in a grid box is determined by the larger of the two cloud fractions:

$$F_{CLD} = \max[F_{RAS}, F_{LS}].$$

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 processes. 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

$$\frac{\partial \rho c_p T}{\partial t} = - \frac{\partial F}{\partial z}$$

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/m^2 and a CO_2 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 8.10, 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 8.11. 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 8.10: UV and Visible Spectral Regions used in shortwave radiation package.

UV and Visible Spectral Regions		
Region	Band	Wavelength (micron)
UV-C	1.	.175 - .225
	2.	.225 - .245
		.260 - .280
	3.	.245 - .260
UV-B	4.	.280 - .295
	5.	.295 - .310
	6.	.310 - .320
UV-A	7.	.320 - .400
PAR	8.	.400 - .700

Table 8.11: Infrared Spectral Regions used in shortwave radiation package.

Infrared Spectral Regions		
Band	Wavenumber (cm ⁻¹)	Wavelength (micron)
1	1000-4400	2.27-10.0
2	4400-8200	1.22-2.27
3	8200-14300	0.70-1.22

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 ($p > 700$ mb), middle ($700 \text{ mb} \geq p > 400$ mb), and high ($p < 400$ 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 . 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 8.12.

Table 8.12: IR Spectral Bands, Absorbers, and Parameterization Method
(from [CS94])

IR Spectral Bands			
Band	Spectral Range (cm ⁻¹)	Absorber	Method
1	0-340	H ₂ O line	T
2	340-540	H ₂ O line	T
3a	540-620	H ₂ O line	K
3b	620-720	H ₂ O continuum	S
3b	720-800	CO ₂	T
4	800-980	H ₂ O line	K
		H ₂ O continuum	S
		H ₂ O line	K
5	980-1100	H ₂ O continuum	S
		O ₃	T
6	1100-1380	H ₂ O line	K
		H ₂ O continuum	S
7	1380-1900	H ₂ O line	T
8	1900-3000	H ₂ O line	K
K: <i>k</i> -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 $< 0.4 \text{ K day}^{-1}$ 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 (N₂O), methane (CH₄), and the chlorofluorocarbons (CFCs), is an underestimate of $\approx 5 \text{ W/m}^2$ in the downward flux at the surface and an overestimate of $\approx 3 \text{ W/m}^2$ 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, p_1 and p_2 ($p_2 > p_1$), 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, ℓ , 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

$$0.0002 \leq \tau_{ice}(mb^{-1}) \leq 0.002 \quad \text{for } 0 \leq \ell \leq 2 \quad \text{mg/kg},$$

$$0.02 \leq \tau_{h_2o}(mb^{-1}) \leq 0.2 \quad \text{for } 0 \leq \ell \leq 10 \quad \text{mg/kg}.$$

The partitioning, α , between ice particles and water droplets is achieved through a linear scaling in temperature:

$$0 \leq \alpha \leq 1 \quad \text{for } 233.15 \leq T \leq 253.15.$$

The resulting optical depth associated with large-scale cloudiness is given as

$$\tau_{LS} = \alpha \tau_{h_2o} + (1 - \alpha) \tau_{ice}.$$

The optical thickness associated with sub-grid scale convective clouds produced by RAS is given as

$$\tau_{RAS} = 0.16 \quad mb^{-1}.$$

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:

$$\tau = \left(\frac{F_{RAS} \tau_{RAS} + F_{LS} \tau_{LS}}{F_{RAS} + F_{LS}} \right) \Delta p,$$

where F_{RAS} and F_{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:

$$\begin{aligned} \frac{\partial u}{\partial t}_{turb} &= \frac{\partial}{\partial z}(-\overline{u'w'}) = \frac{\partial}{\partial z}(K_m \frac{\partial u}{\partial z}) \\ \frac{\partial v}{\partial t}_{turb} &= \frac{\partial}{\partial z}(-\overline{v'w'}) = \frac{\partial}{\partial z}(K_m \frac{\partial v}{\partial z}) \\ \frac{\partial T}{\partial t} &= P^\kappa \frac{\partial \theta}{\partial t}_{turb} = P^\kappa \frac{\partial}{\partial z}(-\overline{w'\theta'}) = P^\kappa \frac{\partial}{\partial z}(K_h \frac{\partial \theta_v}{\partial z}) \\ \frac{\partial q}{\partial t}_{turb} &= \frac{\partial}{\partial z}(-\overline{w'q'}) = \frac{\partial}{\partial z}(K_h \frac{\partial q}{\partial z}) \end{aligned}$$

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),

$$\frac{1}{2} q^2 = \overline{u'^2} + \overline{v'^2} + \overline{w'^2},$$

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:

$$\frac{d}{dt}\left(\frac{1}{2}q^2\right) - \frac{\partial}{\partial z}\left(\frac{5}{3}\lambda_1 q \frac{\partial}{\partial z}\left(\frac{1}{2}q^2\right)\right) = -\overline{u'w'}\frac{\partial U}{\partial z} - \overline{v'w'}\frac{\partial V}{\partial z} + \frac{g}{\Theta_0}\overline{w'\theta'_v} - \frac{q^3}{\Lambda_1}$$

where q is the turbulent velocity, u' , v' , w' and θ' are the fluctuating parts of the velocity components and potential temperature, U and V are the mean velocity components, Θ_0^{-1} is the coefficient of thermal expansion, and λ_1 and Λ_1 are constant multiples of the master length scale, ℓ , 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 bouyancy, 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 θ_v 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

$$K_h = \begin{cases} q \ell S_H(G_M, G_H) & \text{for decaying turbulence} \\ \frac{q^2}{q_e} \ell S_H(G_{Me}, G_{He}) & \text{for growing turbulence} \end{cases}$$

and

$$K_m = \begin{cases} q \ell S_M(G_M, G_H) & \text{for decaying turbulence} \\ \frac{q^2}{q_e} \ell S_M(G_{Me}, G_{He}) & \text{for growing turbulence} \end{cases}$$

where the subscript e refers to the value under conditions of local equilibrium (obtained from the Level 2.0 Model), ℓ 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_{He} and G_{Me} , are functions of the Richardson number:

$$\text{RI} = \frac{\frac{g}{\theta_v} \frac{\partial \theta_v}{\partial z}}{(\frac{\partial u}{\partial z})^2 + (\frac{\partial v}{\partial z})^2} = \frac{c_p \frac{\partial \theta_v}{\partial z} \frac{\partial P^\kappa}{\partial z}}{(\frac{\partial u}{\partial z})^2 + (\frac{\partial v}{\partial z})^2}.$$

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:

$$K_m(\text{surface}) = C_u \times u_* = C_D W_s$$

and

$$K_h(\text{surface}) = C_t \times u_* = C_H W_s$$

where $u_* = C_u W_s$ 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 ψ_m is the surface layer non-dimensional wind shear given by

$$\psi_m = \int_{\zeta_0}^{\zeta} \frac{\phi_m}{\zeta} d\zeta.$$

Here ζ is the non-dimensional stability parameter, and ϕ_m is the similarity function of ζ which expresses the stability dependance of the momentum gradient. The functional form of ϕ_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:

$$C_t = -\frac{(\overline{w'\theta'})}{u_*\Delta\theta} = -\frac{(\overline{w'q'})}{u_*\Delta q} = \frac{k}{(\psi_h + \psi_g)}$$

where ψ_h is the surface layer non-dimensional temperature gradient given by

$$\psi_h = \int_{\zeta_0}^{\zeta} \frac{\phi_h}{\zeta} d\zeta.$$

Here ϕ_h is the similarity function of ζ , which expresses the stability dependance of the temperature and moisture gradients, and is specified differently for stable and unstable layers according to [HS95].

ψ_g is the non-dimensional temperature or moisture gradient in the viscous sublayer, which is the mostly 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]:

$$\psi_g = \frac{0.55(Pr^{2/3} - 0.2)}{\nu^{1/2}} (h_0 u_* - h_{0_{ref}} u_{*_{ref}})^{1/2}$$

where Pr is the Prandtl number for air, ν is the molecular viscosity, z_0 is the surface roughness length, and the subscript *ref* refers to a reference value. $h_0 = 30z_0$ 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 β and the convective limit, are the KEYPS function [Pan73] for momentum, and its generalization for heat and moisture:

$$\phi_m^4 - 18\zeta\phi_m^3 = 1 \quad ; \quad \phi_h^2 - 18\zeta\phi_h^3 = 1 \quad .$$

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:

$$H = P^{\kappa} \rho c_p C_H W_s (\theta_{surface} - \theta_{NLAY}) \quad \text{where : } C_H = C_u C_t$$

where ρ = the atmospheric density at the surface, c_p is the specific heat of air at constant pressure, and θ represents the potential temperature of the surface and of the lowest σ -level, respectively.

The upward latent heat flux, LE , is given by

$$LE = \rho \beta L C_H W_s (q_{surface} - q_{NLAY}) \quad \text{where : } C_H = C_u C_t$$

where β is the fraction of the potential evapotranspiration actually evaporated, L is the latent heat of evaporation, and $q_{surface}$ and q_{NLAY} are the specific humidity of the surface and of the lowest σ -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 (), and is given by:

$$C_g = \sqrt{\frac{\lambda C_s}{2\omega}} = \sqrt{(0.386 + 0.536W + 0.15W^2) 2 \times 10^{-3} \frac{86400}{2\pi}}.$$

Here, the thermal conductivity, λ , is equal to $2 \times 10^{-3} \frac{ly}{sec} \frac{cm}{K}$, the angular velocity of the earth, ω , is written as 86400 *sec/day* divided by 2π *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 8.13. 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 8.9. 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 $\geq 60\%$ 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 8.13: Surface Type Designation

Type	Vegetation Designation
1	Broadleaf Evergreen Trees
2	Broadleaf Deciduous Trees
3	Needleleaf Trees
4	Ground Cover
5	Broadleaf Shrubs
6	Dwarf Trees (Tundra)
7	Bare Soil
8	Desert (Bright)
9	Glacier
10	Desert (Dark)
100	Ocean

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 infra-red 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.

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:

$$|\vec{\tau}_{sfc}| = \frac{\rho U^3}{N\ell^*} \left(\frac{F_r^2}{1 + F_r^2} \right),$$

where $F_r = Nh/U$ 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 ℓ^* 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.

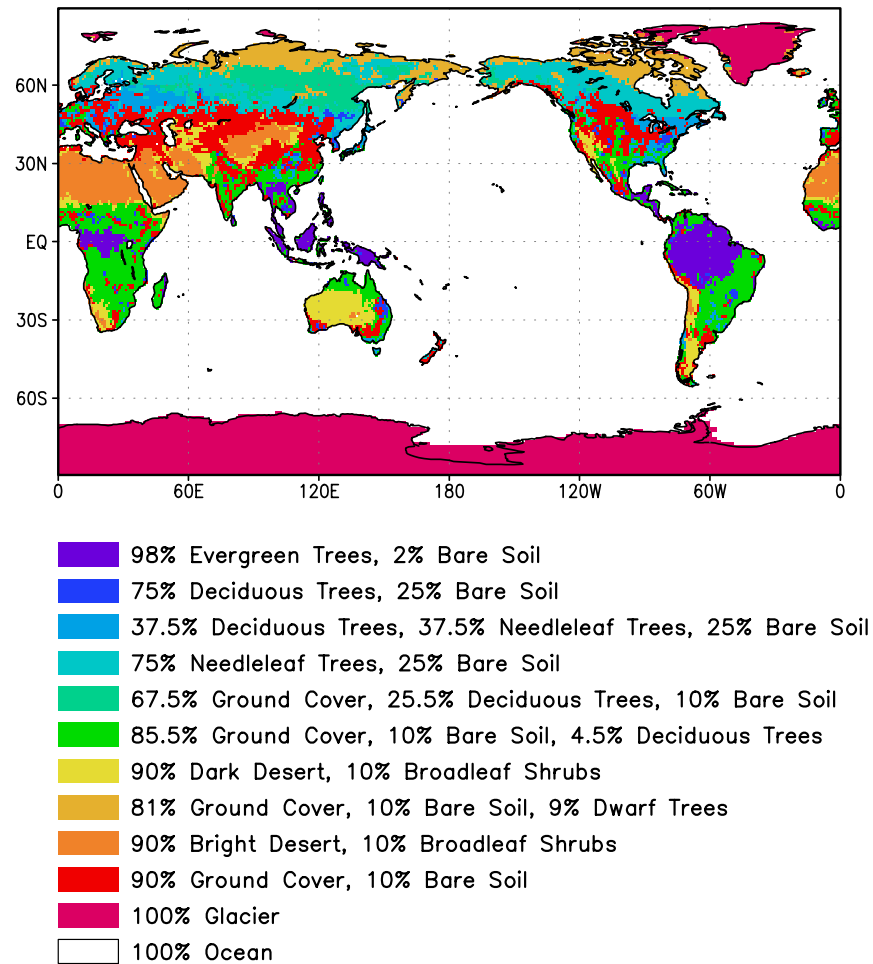


Figure 8.9: Surface type combinations

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 8.14. The time mean values are interpolated during each model timestep to the current time.

Table 8.14: Boundary conditions and other input data used in the fizhi package. Also noted are the current years and frequencies available.

Fizhi Input Datasets		
Sea Ice Extent	monthly	1979-current, climatology
Sea Ice Extent	weekly	1982-current, climatology
Sea Surface Temperature	monthly	1979-current, climatology
Sea Surface Temperature	weekly	1982-current, climatology
Zonally Averaged Upper-Level Moisture	monthly	climatology
Zonally Averaged Ozone Concentration	monthly	climatology

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]

NAME	UNITS	LEVELS	DESCRIPTION
UFLUX	N m^{-2}	1	Surface U-Wind Stress on the atmosphere
VFLUX	N m^{-2}	1	Surface V-Wind Stress on the atmosphere
HFLUX	W m^{-2}	1	Surface Flux of Sensible Heat
EFLUX	W m^{-2}	1	Surface Flux of Latent Heat
QICE	W m^{-2}	1	Heat Conduction through Sea-Ice
RADLWG	W m^{-2}	1	Net upward LW flux at the ground
RADSWG	W m^{-2}	1	Net downward SW flux at the ground
RI	dimensionless	Nrphys	Richardson Number
CT	dimensionless	1	Surface Drag coefficient for T and Q
CU	dimensionless	1	Surface Drag coefficient for U and V
ET	$\text{m}^2 \text{s}^{-1}$	Nrphys	Diffusivity coefficient for T and Q
EU	$\text{m}^2 \text{s}^{-1}$	Nrphys	Diffusivity coefficient for U and V
TURBU	$\text{m s}^{-1} \text{day}^{-1}$	Nrphys	U-Momentum Changes due to Turbulence
TURBV	$\text{m s}^{-1} \text{day}^{-1}$	Nrphys	V-Momentum Changes due to Turbulence
TURBT	deg day^{-1}	Nrphys	Temperature Changes due to Turbulence
TURBQ	g/kg/day	Nrphys	Specific Humidity Changes due to Turbulence
MOISTT	deg day^{-1}	Nrphys	Temperature Changes due to Moist Processes
MOISTQ	g/kg/day	Nrphys	Specific Humidity Changes due to Moist Processes
RADLW	deg day^{-1}	Nrphys	Net Longwave heating rate for each level
RADSW	deg day^{-1}	Nrphys	Net Shortwave heating rate for each level
PREACC	mm/day	1	Total Precipitation
PRECON	mm/day	1	Convective Precipitation
TUFLUX	N m^{-2}	Nrphys	Turbulent Flux of U-Momentum
TVFLUX	N m^{-2}	Nrphys	Turbulent Flux of V-Momentum
TTFLUX	W m^{-2}	Nrphys	Turbulent Flux of Sensible Heat

NAME	UNITS	LEV-ELS	DESCRIPTION
TQFLUX	W m^{-2}	Nrphys	Turbulent Flux of Latent Heat
CN	dimensionless	1	Neutral Drag Coefficient
WINDS	m s^{-1}	1	Surface Wind Speed
DTSRF	deg	1	Air/Surface virtual temperature difference
TG	deg	1	Ground temperature
TS	deg	1	Surface air temperature (Adiabatic from lowest model layer)
DTG	deg	1	Ground temperature adjustment
QG	g kg^{-1}	1	Ground specific humidity
QS	g kg^{-1}	1	Saturation surface specific humidity
TGRLW	deg	1	Instantaneous ground temperature used as input to the Longwave radiation subroutine
ST4	W m^{-2}	1	Upward Longwave flux at the ground (σT^4)
OLR	W m^{-2}	1	Net upward Longwave flux at the top of the model
OLRCLR	W m^{-2}	1	Net upward clearsky Longwave flux at the top of the model
LWGCLR	W m^{-2}	1	Net upward clearsky Longwave flux at the ground
LWCLR	deg day^{-1}	Nrphys	Net clearsky Longwave heating rate for each level
TLW	deg	Nrphys	Instantaneous temperature used as input to the Longwave radiation subroutine
SHLW	g g^{-1}	Nrphys	Instantaneous specific humidity used as input to the Longwave radiation subroutine
OZLW	g g^{-1}	Nrphys	Instantaneous ozone used as input to the Longwave radiation subroutine
CLMOLW	0 – 1	Nrphys	Maximum overlap cloud fraction used in the Longwave radiation subroutine
CLDTOT	0 – 1	Nrphys	Total cloud fraction used in the Longwave and Shortwave radiation subroutines
LWG-DOWN	W m^{-2}	1	Downwelling Longwave radiation at the ground
GWDT	deg day^{-1}	Nrphys	Temperature tendency due to Gravity Wave Drag
RADSWT	W m^{-2}	1	Incident Shortwave radiation at the top of the atmosphere
TAUCLD	per 100 mb	Nrphys	Counted Cloud Optical Depth (non-dimensional) per 100 mb
TAU-CLDC	Number	Nrphys	Cloud Optical Depth Counter

NAME	UNITS	LEVELS	Description
CLDLOW	0-1	Nrphys	Low-Level (1000-700 hPa) Cloud Fraction (0-1)
EVAP	mm/day	1	Surface evaporation
DPDT	hPa/day	1	Surface Pressure time-tendency
UAVE	m/sec	Nrphys	Average U-Wind
VAVE	m/sec	Nrphys	Average V-Wind
TAVE	deg	Nrphys	Average Temperature
QAVE	g/kg	Nrphys	Average Specific Humidity
OMEGA	hPa/day	Nrphys	Vertical Velocity
DUDT	m/sec/day	Nrphys	Total U-Wind tendency
DVDT	m/sec/day	Nrphys	Total V-Wind tendency
DTDT	deg/day	Nrphys	Total Temperature tendency
DQDT	g/kg/day	Nrphys	Total Specific Humidity tendency
VORT	10^{-4} /sec	Nrphys	Relative Vorticity
DTLS	deg/day	Nrphys	Temperature tendency due to Stratiform Cloud Formation
DQLS	g/kg/day	Nrphys	Specific Humidity tendency due to Stratiform Cloud Formation
USTAR	m/sec	1	Surface USTAR wind
Z0	m	1	Surface roughness
FRQTRB	0-1	Nrphys-1	Frequency of Turbulence
PBL	mb	1	Planetary Boundary Layer depth
SWCLR	deg/day	Nrphys	Net clearsky Shortwave heating rate for each level
OSR	$W m^{-2}$	1	Net downward Shortwave flux at the top of the model
OSRCLR	$W m^{-2}$	1	Net downward clearsky Shortwave flux at the top of the model
CLDMAS	kg / m ²	Nrphys	Convective cloud mass flux
UAVE	m/sec	Nrphys	Time-averaged <i>u</i> -Wind

NAME	UNITS	LEVELS	DESCRIPTION
VAVE	m/sec	Nrphys	Time-averaged <i>v</i> -Wind
TAVE	deg	Nrphys	Time-averaged Temperature
QAVE	g/g	Nrphys	Time-averaged Specific Humidity
RFT	deg/day	Nrphys	Temperature tendency due Rayleigh Friction
PS	mb	1	Surface Pressure
QQAVE	(m/sec) ²	Nrphys	Time-averaged Turbulent Kinetic Energy
SWGCLR	$W m^{-2}$	1	Net downward clearsky Shortwave flux at the ground
PAVE	mb	1	Time-averaged Surface Pressure
DIABU	m/sec/day	Nrphys	Total Diabatic forcing on <i>u</i> -Wind
DIABV	m/sec/day	Nrphys	Total Diabatic forcing on <i>v</i> -Wind
DIABT	deg/day	Nrphys	Total Diabatic forcing on Temperature
DIABQ	g/kg/day	Nrphys	Total Diabatic forcing on Specific Humidity
RFU	m/sec/day	Nrphys	U-Wind tendency due to Rayleigh Friction
RFV	m/sec/day	Nrphys	V-Wind tendency due to Rayleigh Friction
GWDU	m/sec/day	Nrphys	U-Wind tendency due to Gravity Wave Drag
GWDV	m/sec/day	Nrphys	V-Wind tendency due to Gravity Wave Drag
GWDUS	$N m^{-2}$	1	U-Wind Gravity Wave Drag Stress at Surface
GWDVS	$N m^{-2}$	1	V-Wind Gravity Wave Drag Stress at Surface
GWDUT	$N m^{-2}$	1	U-Wind Gravity Wave Drag Stress at Top
GWDVT	$N m^{-2}$	1	V-Wind Gravity Wave Drag Stress at Top
LZRAD	mg/kg	Nrphys	Estimated Cloud Liquid Water used in Radiation

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/m ² /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:

$$\text{DIAGNOSTIC} = \frac{1}{TTOT} \sum_{t=1}^{t=TTOT} \text{diag}(t)$$

where $TTOT = \frac{NQDIAG}{\Delta t}$, $NQDIAG$ is the output frequency of the diagnostic, and Δt is the timestep over which the diagnostic is updated.

Surface Zonal Wind Stress on the Atmosphere (*Newton/m²*)

The zonal wind stress is the turbulent flux of zonal momentum from the surface.

$$\text{UFLUX} = -\rho C_D W_s u \quad \text{where : } C_D = C_u^2$$

where ρ = 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 (*Newton/m²*)

The meridional wind stress is the turbulent flux of meridional momentum from the surface.

$$\text{VFLUX} = -\rho C_D W_s v \quad \text{where : } C_D = C_u^2$$

where ρ = 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 (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:

$$\text{HFLUX} = P^{\kappa} \rho c_p C_H W_s (\theta_{\text{surface}} - \theta_{\text{Nrphys}}) \quad \text{where : } C_H = C_u C_t$$

where ρ = 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 θ is the potential temperature at the surface and at the bottom model level.

Surface Flux of Latent Heat (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:

$$\text{EFLUX} = \rho \beta L C_H W_s (q_{\text{surface}} - q_{\text{Nrphys}}) \quad \text{where : } C_H = C_u C_t$$

where ρ = the atmospheric density at the surface, β 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_{surface} and q_{Nrphys} are the specific humidity at the surface and at the bottom model level, respectively.

Heat Conduction Through Sea Ice (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)

$$\begin{aligned} \text{RADLWG} &= F_{\text{LW}, \text{Nrphys}+1}^{\text{Net}} \\ &= F_{\text{LW}, \text{Nrphys}+1}^{\uparrow} - F_{\text{LW}, \text{Nrphys}+1}^{\downarrow} \end{aligned}$$

where Nrphys+1 indicates the lowest model edge-level, or $p = p_{\text{surf}}$. F_{LW}^{\uparrow} is the upward Longwave flux and $F_{\text{LW}}^{\downarrow}$ is the downward Longwave flux.

Net downward shortwave Flux at the surface ($Watts/m^2$)

$$\begin{aligned} \text{RADSWG} &= F_{SW, Nrphys+1}^{Net} \\ &= F_{SW, Nrphys+1}^{\downarrow} - F_{SW, Nrphys+1}^{\uparrow} \end{aligned}$$

where Nrphys+1 indicates the lowest model edge-level, or $p = p_{surf}$. F_{SW}^{\downarrow} is the downward Shortwave flux and F_{SW}^{\uparrow} is the upward Shortwave flux.

Richardson number (*dimensionless*)

The non-dimensional stability indicator is the ratio of the buoyancy to the shear:

$$\text{RI} = \frac{\frac{g}{\theta_v} \frac{\partial \theta_v}{\partial z}}{\left(\frac{\partial u}{\partial z}\right)^2 + \left(\frac{\partial v}{\partial z}\right)^2} = \frac{c_p \frac{\partial \theta_v}{\partial z} \frac{\partial P^\kappa}{\partial z}}{\left(\frac{\partial u}{\partial z}\right)^2 + \left(\frac{\partial v}{\partial z}\right)^2}$$

where we used the hydrostatic equation:

$$\frac{\partial \Phi}{\partial P^\kappa} = c_p \theta_v$$

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:

$$\text{CT} = -\frac{(\overline{w'\theta'})}{u_* \Delta \theta} = -\frac{(\overline{w'q'})}{u_* \Delta q} = \frac{k}{(\psi_h + \psi_g)}$$

where ψ_h is the surface layer non-dimensional temperature change and ψ_g is the viscous sublayer non-dimensional temperature or moisture change:

$$\psi_h = \int_{\zeta_0}^{\zeta} \frac{\phi_h}{\zeta} d\zeta \quad \text{and} \quad \psi_g = \frac{0.55(Pr^{2/3} - 0.2)}{\nu^{1/2}} (h_0 u_* - h_{0ref} u_{*ref})^{1/2}$$

and: $h_0 = 30z_0$ with a maximum value over land of 0.01

ϕ_h is the similarity function of ζ , 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, ζ is the non-dimensional stability parameter, Pr is the Prandtl number for air, ν 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:

$$\text{CU} = \frac{u_*}{W_s} = \frac{k}{\psi_m}$$

where ψ_m is the surface layer non-dimensional wind shear:

$$\psi_m = \int_{\zeta_0}^{\zeta} \frac{\phi_m}{\zeta} d\zeta$$

ϕ_m is the similarity function of ζ , 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, ζ 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 (m²/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:

$$\mathbf{ET} = K_h = -\frac{(\overline{w'\theta'_v})}{\frac{\partial\theta_v}{\partial z}} = \begin{cases} q \ell S_H(G_M, G_H) & \text{for decaying turbulence} \\ \frac{q^2}{q_e} \ell S_H(G_{Me}, G_{He}) & \text{for growing turbulence} \end{cases}$$

where q is the turbulent velocity, or $\sqrt{2 * \text{turbulent kinetic energy}}$, q_e is the turbulence velocity derived from the more simple level 2.0 model, which describes equilibrium turbulence, ℓ 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_{He} and G_{Me} , the equilibrium dimensionless buoyancy and wind shear parameters. Both G_H and G_M , and their equilibrium values G_{He} and G_{Me} , 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, \mathbf{ET} is the exchange coefficient for heat and moisture, in units of m/sec , given by:

$$\mathbf{ET}_{\text{Nrphys}} = C_t * u_* = C_H W_s$$

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 (m²/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:

$$\mathbf{EU} = K_m = -\frac{(\overline{u'w'})}{\frac{\partial U}{\partial z}} = \begin{cases} q \ell S_M(G_M, G_H) & \text{for decaying turbulence} \\ \frac{q^2}{q_e} \ell S_M(G_{Me}, G_{He}) & \text{for growing turbulence} \end{cases}$$

where q is the turbulent velocity, or $\sqrt{2 * \text{turbulent kinetic energy}}$, q_e is the turbulence velocity derived from the more simple level 2.0 model, which describes equilibrium turbulence, ℓ 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_{He} and G_{Me} , the equilibrium dimensionless buoyancy and wind shear parameters. Both G_H and G_M , and their equilibrium values G_{He} and G_{Me} , 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, \mathbf{EU} is the exchange coefficient for momentum, in units of m/sec , given by:

$$\mathbf{EU}_{\text{Nrphys}} = C_u * u_* = C_D W_s$$

where C_u is the dimensionless exchange coefficient for momentum from the surface layer similarity functions (see diagnostic number 10), u_* 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:

$$\text{TURBU} = \frac{\partial u}{\partial t}_{\text{turb}} = \frac{\partial}{\partial z}(-\overline{u'w'}) = \frac{\partial}{\partial z}(K_m \frac{\partial u}{\partial z})$$

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:

$$\text{TURBV} = \frac{\partial v}{\partial t}_{\text{turb}} = \frac{\partial}{\partial z}(-\overline{v'w'}) = \frac{\partial}{\partial z}(K_m \frac{\partial v}{\partial z})$$

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:

$$\text{TURBT} = \frac{\partial T}{\partial t} = P^\kappa \frac{\partial \theta}{\partial t}_{\text{turb}} = P^\kappa \frac{\partial}{\partial z}(-\overline{w'\theta'}) = P^\kappa \frac{\partial}{\partial z}(K_h \frac{\partial \theta_v}{\partial z})$$

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:

$$\text{TURBQ} = \frac{\partial q}{\partial t}_{\text{turb}} = \frac{\partial}{\partial z}(-\overline{w'q'}) = \frac{\partial}{\partial z}(K_h \frac{\partial q}{\partial z})$$

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:

$$\left. \frac{\partial T}{\partial t} \right|_c = R \sum_i \left(\alpha \frac{m_B}{c_p} \Gamma_s \right)_i \quad \text{and} \quad \left. \frac{\partial T}{\partial t} \right|_{ls} = \frac{L}{c_p} (q^* - q)$$

and

$$\Gamma_s = g\eta \frac{\partial s}{\partial p}$$

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 convective cloud base mass flux is given as m_B , and the cloud entrainment is given as η , 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 α , 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:

$$\left. \frac{\partial q}{\partial t} \right|_c = R \sum_i \left(\alpha \frac{m_B}{L} (\Gamma_h - \Gamma_s) \right)_i \quad \text{and} \quad \left. \frac{\partial q}{\partial t} \right|_{ls} = (q^* - q)$$

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 η , 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 α , 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, $F_{LW}^{clearsky}$, 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, $p' = p_{top}$, and the model surface pressure, $p' = p_{surf}$, for the upward and downward radiative fluxes. (see [Section \[sec:fizhi:radcloud\]](#)). The cloudy-sky flux is then obtained as:

$$F_{LW} = C(p, p') \cdot F_{LW}^{clearsky},$$

Finally, the net longwave heating rate is calculated as the vertical divergence of the net terrestrial radiative fluxes:

$$\frac{\partial \rho c_p T}{\partial t} = - \frac{\partial}{\partial z} F_{LW}^{NET},$$

or

$$\text{RADLW} = \frac{g}{c_p \pi} \frac{\partial}{\partial \sigma} F_{LW}^{NET}.$$

where g is the accelation due to gravity, c_p is the heat capacity of air at constant pressure, and

$$F_{LW}^{NET} = F_{LW}^{\uparrow} - F_{LW}^{\downarrow}$$

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

$$\text{RADSW} = \frac{g}{c_p \pi} \frac{\partial}{\partial \sigma} F(\text{cloudy})_{SW}^{NET} \cdot \text{RADSWT}.$$

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, Δq_{moist} , the vertical integral or total precipitable amount is given by:

$$\text{PREACC} = \int_{surf}^{top} \rho \Delta q_{moist} dz = - \int_{surf}^{top} \Delta q_{moist} \frac{dp}{g} = \frac{1}{g} \int_0^1 \Delta q_{moist} dp$$

A precipitation rate is defined as the vertically integrated moisture adjustment per Moist Processes time step, scaled to mm/day .

PRECON - Convective Precipitation (mm/day)

For a change in specific humidity due to sub-grid scale cumulus convective processes, Δq_{cum} , the vertical integral or total precipitable amount is given by:

$$\text{PRECON} = \int_{surf}^{top} \rho \Delta q_{cum} dz = - \int_{surf}^{top} \Delta q_{cum} \frac{dp}{g} = \frac{1}{g} \int_0^1 \Delta q_{cum} dp$$

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:

$$\text{TUFLUX} = \rho \overline{u'w'} = \rho \left(-K_m \frac{\partial U}{\partial z} \right)$$

where ρ 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:

$$\mathbf{TVFLUX} = \rho(\overline{v'w'}) = \rho(-K_m \frac{\partial V}{\partial z})$$

where ρ 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:

$$\mathbf{TTFLUX} = c_p \rho P^\kappa (\overline{w'\theta'}) = c_p \rho P^\kappa (-K_h \frac{\partial \theta_v}{\partial z})$$

where ρ 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:

$$\mathbf{TQFLUX} = L \rho (\overline{w'q'}) = L \rho (-K_h \frac{\partial q}{\partial z})$$

where ρ 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:

$$\mathbf{CN} = \frac{k}{\ln(\frac{h}{z_0})}$$

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:

$$\mathbf{WINDS} = \sqrt{u_{Nrphys}^2 + v_{Nrphys}^2}$$

where the subscript *Nrphys* refers to the lowest model level.

The air/surface virtual temperature difference measures the stability of the surface layer:

$$\mathbf{DTSRF} = (\theta_{vNrphys+1} - \theta_{vNrphys}) P_{surf}^\kappa$$

where

$$\theta_{vNrphys+1} = \frac{T_g}{P_{surf}^\kappa} (1 + .609 q_{Nrphys+1}) \quad \text{and} \quad q_{Nrphys+1} = q_{Nrphys} + \beta(q^*(T_g, P_s) - q_{Nrphys})$$

β 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 $Nrphys$ refers to the lowest model level and level $Nrphys + 1$ 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:

$$C_g = \sqrt{\frac{\lambda C_s}{2\omega}} = \sqrt{(0.386 + 0.536W + 0.15W^2)2 \times 10^{-3} \frac{86400}{2\pi}}.$$

Here, the thermal conductivity, λ , is equal to $2 \times 10^{-3} \frac{\text{ly cm}}{\text{sec K}}$, the angular velocity of the earth, ω , is written as 86400 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 θ is constant in that layer. The surface temperature is therefore:

$$\text{TS} = \theta_{Nrphys} P_{surf}^\kappa$$

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:

$$\text{DTG} = T_g^n - T_g^{n-1}$$

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:

$$\text{QG} = q_{Nrphys+1} = q_{Nrphys} + \beta(q^*(T_g, P_s) - q_{Nrphys})$$

where β 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 σ is the Stefan-Boltzmann constant and T is the temperature.

OLR - Net upward Longwave flux at $p = p_{top}$ (Watts/m^2)

$$OLR = F_{LW,top}^{NET}$$

where top indicates the top of the first model layer. In the GCM, $p_{top} = 0.0$ mb.

OLRCLR - Net upward clearsky Longwave flux at $p = p_{top}$ (Watts/m^2)

$$OLRCLR = F(clearsky)_{LW,top}^{NET}$$

where top indicates the top of the first model layer. In the GCM, $p_{top} = 0.0$ mb.

LWGCLR - Net upward clearsky Longwave flux at the surface (Watts/m^2)

$$\begin{aligned} LWGCLR &= F(clearsky)_{LW,Nrphys+1}^{Net} \\ &= F(clearsky)_{LW,Nrphys+1}^{\uparrow} - F(clearsky)_{LW,Nrphys+1}^{\downarrow} \end{aligned}$$

where $Nrphys+1$ indicates the lowest model edge-level, or $p = p_{surf}$. $F(clearsky)_{LW}^{\uparrow}$ is the upward clearsky Longwave flux and the $F(clearsky)_{LW}^{\downarrow}$ 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, $F_{LW}^{clearsky}$, 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, $p' = p_{top}$, and the model surface pressure, $p' = p_{surf}$, for the upward and downward radiative fluxes. (see Section [sec:fizhi:radcloud]). The cloudy-sky flux is then obtained as:

$$F_{LW} = C(p, p') \cdot F_{LW}^{clearsky},$$

Thus, **LWCLR** is defined as the net longwave heating rate due to the vertical divergence of the clear-sky longwave radiative flux:

$$\frac{\partial \rho c_p T}{\partial t}_{clearsky} = -\frac{\partial}{\partial z} F(clearsky)_{LW}^{NET},$$

or

$$\mathbf{LWCLR} = \frac{g}{c_p \pi} \frac{\partial}{\partial \sigma} F(clearsky)_{LW}^{NET}.$$

where g is the accelation due to gravity, c_p is the heat capacity of air at constant pressure, and

$$F(clearsky)_{LW}^{Net} = F(clearsky)_{LW}^{\uparrow} - F(clearsky)_{LW}^{\downarrow}$$

TLW - Instantaneous temperature used as input to the Longwave radiation subroutine (deg)

$$\mathbf{TLW} = T(\lambda, \phi, level, n)$$

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)

$$\mathbf{SHLW} = q(\lambda, \phi, level, n)$$

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)

$$\mathbf{OZLW} = \text{OZ}(\lambda, \phi, level, n)$$

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].

$$\mathbf{CLMOLW} = CLMO_{RAS, LW}(\lambda, \phi, level)$$

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].

$$\text{CLDTOT} = F_{RAS} + F_{LS}$$

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].

$$\text{CLMOSW} = \text{CLMO}_{RAS,SW}(\lambda, \phi, level)$$

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].

$$\text{CLROSW} = \text{CLRO}_{RAS, LargeScale, SW}(\lambda, \phi, level)$$

RADSWT - Incident Shortwave radiation at the top of the atmosphere (Watts/m^2)

$$\text{RADSWT} = \frac{S_0}{R_a^2} \cdot \cos\phi_z$$

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:

$$\text{EVAP} = \rho\beta K_h(q_{surface} - q_{Nrphys})$$

where ρ = the atmospheric density at the surface, β 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_{Nrphys} 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.

$$\mathbf{DUDT} = \frac{\partial u}{\partial t}_{Dynamics} + \frac{\partial u}{\partial t}_{Moist} + \frac{\partial u}{\partial t}_{Turbulence} + \frac{\partial u}{\partial t}_{Analysis}$$

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.

$$\mathbf{DVDT} = \frac{\partial v}{\partial t}_{Dynamics} + \frac{\partial v}{\partial t}_{Moist} + \frac{\partial v}{\partial t}_{Turbulence} + \frac{\partial v}{\partial t}_{Analysis}$$

DTDT - Total Temperature Tendency (deg/day)

DTDT is the total time-tendency of Temperature due to Hydrodynamic, Diabatic, and Analysis forcing.

$$\begin{aligned} \mathbf{DTDT} = & \frac{\partial T}{\partial t}_{Dynamics} + \frac{\partial T}{\partial t}_{MoistProcesses} + \frac{\partial T}{\partial t}_{ShortwaveRadiation} \\ & + \frac{\partial T}{\partial t}_{LongwaveRadiation} + \frac{\partial T}{\partial t}_{Turbulence} + \frac{\partial T}{\partial t}_{Analysis} \end{aligned}$$

DQDT - Total Specific Humidity Tendency (g/kg/day)

DQDT is the total time-tendency of Specific Humidity due to Hydrodynamic, Diabatic, and Analysis forcing.

$$\mathbf{DQDT} = \frac{\partial q}{\partial t}_{Dynamics} + \frac{\partial q}{\partial t}_{MoistProcesses} + \frac{\partial q}{\partial t}_{Turbulence} + \frac{\partial q}{\partial t}_{Analysis}$$

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:

$$\mathbf{USTAR} = C_u W_s \quad \text{where : } C_u = \frac{k}{\psi_m}$$

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 . Over the ocean, the roughness length is a function of the surface-stress velocity, u_* .

$$\mathbf{Z0} = c_1 u_*^3 + c_2 u_*^2 + c_3 u_* + c_4 + c_5 u_*$$

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:

$$\frac{\partial \rho c_p T}{\partial t} = - \frac{\partial}{\partial z} F(\text{clear})_{SW}^{NET} \cdot \text{RADSWT},$$

or

$$\text{SWCLR} = \frac{g}{c_p} \frac{\partial}{\partial p} F(\text{clear})_{SW}^{NET} \cdot \text{RADSWT}.$$

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

$$F(\text{clear})_{SW}^{Net} = F(\text{clear})_{SW}^{\uparrow} - F(\text{clear})_{SW}^{\downarrow}$$

OSR - Net upward Shortwave flux at the top of the model (Watts/m^2)

$$\text{OSR} = F_{SW,top}^{NET}$$

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)

$$\text{OSRCLR} = F(\text{clearsky})_{SW,top}^{NET}$$

where top indicates the top of the first model layer used in the shortwave radiation routine. In the GCM, $p_{SW,top} = 0$ mb.

CLDMAS - Convective Cloud Mass Flux (kg/m²)

The amount of cloud mass moved per RAS timestep from all convective clouds is written:

$$\text{CLDMAS} = \eta m_B$$

where η is the entrainment, normalized by the cloud base mass flux, and m_B is the cloud base mass flux. m_B and η 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 **UAVE** is simply the time-averaged Zonal U-Wind over the **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, **UAVE** is computed and stored on the staggered C-grid.

VAVE - Time-Averaged Meridional V-Wind (m/sec)

The diagnostic **VAVE** is simply the time-averaged Meridional V-Wind over the **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, **VAVE** is computed and stored on the staggered C-grid.

TAVE - Time-Averaged Temperature (Kelvin)

The diagnostic **TAVE** is simply the time-averaged Temperature over the **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 **QAVE** is simply the time-averaged Specific Humidity over the **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 - P_{TOP} (mb)

The diagnostic **PAVE** is simply the time-averaged Surface Pressure - P_{TOP} over the **NPAVE** output frequency. This is contrasted to the instantaneous Surface Pressure - P_{TOP} which is archived on the Prognostic Output data stream.

$$\begin{aligned} \text{PAVE} &= \pi(\lambda, \phi, level, t) \\ &= p_s(\lambda, \phi, level, t) - p_T \end{aligned}$$

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.

$$\mathbf{QQAVE} = qq(\lambda, \phi, level, t)$$

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)

$$\begin{aligned} \mathbf{SWGCLR} &= F(clearsky)_{SW, Nrphys+1}^{Net} \\ &= F(clearsky)_{SW, Nrphys+1}^{\downarrow} - F(clearsky)_{SW, Nrphys+1}^{\uparrow} \end{aligned}$$

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.

$$\mathbf{DIABU} = \frac{\partial u}{\partial t}_{Moist} + \frac{\partial u}{\partial t}_{Turbulence} + \frac{\partial u}{\partial t}_{Analysis}$$

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.

$$\mathbf{DIABV} = \frac{\partial v}{\partial t}_{Moist} + \frac{\partial v}{\partial t}_{Turbulence} + \frac{\partial v}{\partial t}_{Analysis}$$

DIABT Total Diabatic Temperature Tendency (deg/day)

DIABT is the total time-tendency of Temperature due to Diabatic processes and the Analysis forcing.

$$\begin{aligned} \mathbf{DIABT} &= \frac{\partial T}{\partial t}_{MoistProcesses} + \frac{\partial T}{\partial t}_{ShortwaveRadiation} \\ &+ \frac{\partial T}{\partial t}_{LongwaveRadiation} + \frac{\partial T}{\partial t}_{Turbulence} + \frac{\partial T}{\partial t}_{Analysis} \end{aligned}$$

If we define the time-tendency of Temperature due to Diabatic processes as

$$\begin{aligned} \frac{\partial T}{\partial t}_{Diabatic} &= \frac{\partial T}{\partial t}_{MoistProcesses} + \frac{\partial T}{\partial t}_{ShortwaveRadiation} \\ &+ \frac{\partial T}{\partial t}_{LongwaveRadiation} + \frac{\partial T}{\partial t}_{Turbulence} \end{aligned}$$

then, since there are no surface pressure changes due to Diabatic processes, we may write

$$\frac{\partial T}{\partial t}_{Diabatic} = \frac{p^\kappa}{\pi} \frac{\partial \pi \theta}{\partial t}_{Diabatic}$$

where $\theta = T/p^\kappa$. Thus, **DIABT** may be written as

$$\mathbf{DIABT} = \frac{p^\kappa}{\pi} \left(\frac{\partial \pi \theta}{\partial t}_{Diabatic} + \frac{\partial \pi \theta}{\partial t}_{Analysis} \right)$$

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.

$$\mathbf{DIABQ} = \frac{\partial q}{\partial t}_{MoistProcesses} + \frac{\partial q}{\partial t}_{Turbulence} + \frac{\partial q}{\partial t}_{Analysis}$$

If we define the time-tendency of Specific Humidity due to Diabatic processes as

$$\frac{\partial q}{\partial t}_{Diabatic} = \frac{\partial q}{\partial t}_{MoistProcesses} + \frac{\partial q}{\partial t}_{Turbulence}$$

then, since there are no surface pressure changes due to Diabatic processes, we may write

$$\frac{\partial q}{\partial t}_{Diabatic} = \frac{1}{\pi} \frac{\partial \pi q}{\partial t}_{Diabatic}$$

Thus, **DIABQ** may be written as

$$\mathbf{DIABQ} = \frac{1}{\pi} \left(\frac{\partial \pi q}{\partial t}_{Diabatic} + \frac{\partial \pi q}{\partial t}_{Analysis} \right)$$

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.

$$\mathbf{VINTUQ} = \frac{\int_{surf}^{top} uq \rho dz}{\int_{surf}^{top} \rho dz}$$

Using $\rho \delta z = -\frac{\delta p}{g} = -\frac{1}{g} \delta p$, we have

$$\mathbf{VINTUQ} = \int_0^1 uq dp$$

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.

$$\mathbf{VINTVQ} = \frac{\int_{surf}^{top} vq \rho dz}{\int_{surf}^{top} \rho dz}$$

Using $\rho \delta z = -\frac{\delta p}{g} = -\frac{1}{g} \delta p$, we have

$$\mathbf{VINTVQ} = \int_0^1 vq dp$$

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.

$$\text{VINTUT} = \frac{\int_{surf}^{top} uT \rho dz}{\int_{surf}^{top} \rho dz}$$

Or,

$$\text{VINTUT} = \int_0^1 uT dp$$

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.

$$\text{VINTVT} = \frac{\int_{surf}^{top} vT \rho dz}{\int_{surf}^{top} \rho dz}$$

Using $\rho \delta z = -\frac{\delta p}{g}$, we have

$$\text{VINTVT} = \int_0^1 vT dp$$

CLDFRC - Total 2-Dimensional Cloud Fracton (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.

$$1 - \text{MAX}_p^{p'} (\text{CLMO}_p)$$

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.

$$\prod_p^{p'} (1 - \text{CLRO}_p)$$

The total cloud fraction at a given level associated with a line- of-site calculation is given by

$$1 - \left(1 - \text{MAX}_p^{p'} [\text{CLMO}_p] \right) \prod_p^{p'} (1 - \text{CLRO}_p)$$

The 2-dimensional net cloud fraction as seen from the top of the atmosphere is given by

$$\text{CLDFRC} = 1 - \left(1 - \text{MAX}_{l=l_1}^{\text{Nrphys}} [\text{CLMO}_l]\right) \prod_{l=l_1}^{\text{Nrphys}} (1 - \text{CLRO}_l)$$

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:

$$\begin{aligned} \text{QINT} &= \int_{\text{surf}}^{\text{top}} \rho q dz \\ &= \frac{\pi}{g} \int_0^1 q dp \end{aligned}$$

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:

$$\text{U2M} = \frac{u_*}{k} \psi_{m_{2m}} \frac{u_{sl}}{W_s} = \frac{\psi_{m_{2m}}}{\psi_{m_{sl}}} u_{sl}$$

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:

$$\text{V2M} = \frac{u_*}{k} \psi_{m_{2m}} \frac{v_{sl}}{W_s} = \frac{\psi_{m_{2m}}}{\psi_{m_{sl}}} v_{sl}$$

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 a determined from the similarity theory:

$$\text{T2M} = P^\kappa \left(\frac{\theta_*}{k} (\psi_{h_{2m}} + \psi_g) + \theta_{\text{surf}} \right) = P^\kappa \left(\theta_{\text{surf}} + \frac{\psi_{h_{2m}} + \psi_g}{\psi_{h_{sl}} + \psi_g} (\theta_{sl} - \theta_{\text{surf}}) \right)$$

where:

$$\theta_* = -\frac{(\overline{w'\theta'})}{u_*}$$

where $\psi_h(2m)$ is the non-dimensional temperature gradient at two meters, ψ_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, **T2M** 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:

$$\mathbf{Q2M} = P^\kappa \frac{k(\psi_{h_{2m}} + \psi_g) + q_{surf}}{q_*} = P^\kappa (q_{surf} + \frac{\psi_{h_{2m}} + \psi_g}{\psi_{h_{sl}} + \psi_g} (q_{sl} - q_{surf}))$$

where:

$$q_* = - \frac{(\overline{w'\theta'})}{u_*}$$

where $\psi_h(2m)$ is the non-dimensional temperature gradient at two meters, ψ_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, **Q2M** 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:

$$\mathbf{U10M} = \frac{u_*}{k} \psi_{m_{10m}} \frac{u_{sl}}{W_s} = \frac{\psi_{m_{10m}}}{\psi_{m_{sl}}} u_{sl}$$

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:

$$\mathbf{V10M} = \frac{u_*}{k} \psi_{m_{10m}} \frac{v_{sl}}{W_s} = \frac{\psi_{m_{10m}}}{\psi_{m_{sl}}} v_{sl}$$

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:

$$\mathbf{T10M} = P^\kappa \left(\frac{\theta_*}{k} (\psi_{h_{10m}} + \psi_g) + \theta_{surf} \right) = P^\kappa \left(\theta_{surf} + \frac{\psi_{h_{10m}} + \psi_g}{\psi_{h_{sl}} + \psi_g} (\theta_{sl} - \theta_{surf}) \right)$$

where:

$$\theta_* = - \frac{(\overline{w'\theta'})}{u_*}$$

where $\psi_h(10m)$ is the non-dimensional temperature gradient at two meters, ψ_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:

$$\mathbf{Q10M} = P^\kappa \left(\frac{q_*}{k} (\psi_{h_{10m}} + \psi_g) + q_{surf} \right) = P^\kappa \left(q_{surf} + \frac{\psi_{h_{10m}} + \psi_g}{\psi_{h_{sl}} + \psi_g} (q_{sl} - q_{surf}) \right)$$

where:

$$q_* = - \frac{(\overline{w'q'})}{u_*}$$

where $\psi_h(10m)$ is the non-dimensional temperature gradient at two meters, ψ_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:

$$\mathbf{DTRAIN} = \eta_{r_D} m_B$$

where r_D is the detrainment level, m_B is the cloud base mass flux, and η 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:

$$\mathbf{QFILL} = q_{final}^{n+1} - q_{initial}^{n+1}$$

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 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:

$$\mathbf{frzmlt} = (T_f - SST) \frac{c_{sw} \rho_{sw} \Delta z}{\Delta t}$$

where c_{sw} is seawater heat capacity, ρ_{sw} is the seawater density, Δz is the ocean model upper layer thickness and Δt 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 **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:

$$\begin{aligned} q_1 &= -c_i * T_f + L_i \\ q_2 &= -c_f T_{mlt} + c_i (T_{mlt} - T_f) + L_i \left(1 - \frac{T_{mlt}}{T_f}\right) \end{aligned}$$

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, ρ_i is density of ice and T_{mlt} is melting temperature of ice with salinity of 1. The height of a new layer of ice is

$$h_{inew} = \frac{\text{esurp} \Delta t}{q i_{0av}}$$

where $q i_{0av} = -\frac{\rho_i}{2} (q_1 + q_2)$.

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

$$i_f = (1 - \hat{i}_f) \frac{h_{inew}}{h_i}$$

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 **hihig** then freezing energy comes from the full grid cell, ice growth does not occur under original 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, **fswint**, and the upward conduction of heat through the snow and/or ice, F_u . We linearize F_s about the surface temperature, \hat{T}_s , at the previous timestep (where $\hat{}$ indicates the value at the previous timestep):

$$F_s(T_s) = F_s(\hat{T}_s) + \frac{\partial F_s(\hat{T}_s)}{\partial T_s} (T_s - \hat{T}_s)$$

where,

$$F_s = F_{sensible} + F_{latent} + F_{longwave}^{down} + F_{longwave}^{up} + (1 - \alpha) F_{shortwave}$$

and

$$\frac{dF_s}{dT} = \frac{dF_{sensible}}{dT} + \frac{dF_{latent}}{dT} + \frac{dF_{longwave}^{up}}{dT}.$$

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, $surf_albedo.F$) and the shortwave flux absorbed in the ice is

$$fswint = (1 - e^{\kappa_i h_i})(1 - \alpha)F_{shortwave}$$

where κ_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 :math: 'K_{1/2} = \frac{4 K_i K_s}{K_s h_i + 4 K_i h_s} . :math: 'K_i and K_s are constant thermal conductivities of seaice 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 then 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 seaice with temperature T . For the upper layer (1) with brine pockets and the lower fresh layer (2):

$$\begin{aligned} q_1 &= -c_f T_f + c_i(T_f - T) + L_i(1 - \frac{T_f}{T}) \\ q_2 &= -c_i T + L_i \end{aligned}$$

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.

$$\begin{aligned} E_{top} &= (F_s - K_{1/2}(T_s - T_1))\Delta t \\ E_{bot} &= (\frac{4K_i(T_2 - T_f)}{h_i} - F_b)\Delta t \end{aligned}$$

where F_b is the heat flux at the ice bottom due to the sea surface temperature variations from freezing. If T_{sst} is above freezing, $F_b = c_{sw}\rho_{sw}\gamma(T_{sst} - T_f)u^*$, γ is the heat transfer coefficient and $u^* = \sqrt{Q}$ is frictional velocity between ice and water. If T_{sst} is below freezing, $F_b = (T_f - T_{sst})c_f\rho_f\Delta z/\Delta t$ and set T_{sst} to T_f . We also include the energy from lower layers that drop below freezing, and set those layers to T_f .

If $E_{top} > 0$ 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 $E_{bot} > 0$ 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 $E_{bot} < 0$ we grow ice at the bottom

$$\Delta h_i = \frac{-E_{bot}}{(q_{bot}\rho_i)}$$

where $q_{bot} = -c_i T_f + L_i$ is the enthalpy of the new ice, The enthalpy of the second ice layer, q_2 needs to be modified:

$$q_2 = \frac{\hat{h}_i/2\hat{q}_2 + \Delta h_i q_{bot}}{\hat{h}_i/2 + \Delta h_i}$$

If there is a ice layer and the overlying air temperature is below 0°C then any precipitation, P joins the snow layer:

$$\Delta h_s = -P\frac{\rho_f}{\rho_s}\Delta t,$$

ρ_f and ρ_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 *srf_albedo.F* below).

For practical reasons we limit the ice growth to **hilim** and snow is limited to **hslim**. We convert 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 *new_layers_winton.F*, described below, repartitions the ice into equal thickness layers while conserving energy.

The subroutine *ice_therm.F* now calculates the heat and fresh water fluxes affecting the ocean model surface layer. The heat flux:

$$q_{net} = \text{fswocn} - F_b - \frac{\text{esurp}}{\Delta t}$$

is composed of the shortwave flux that has passed through the ice layer and is absorbed by the water, **fswocn** = Q_Q , the ocean flux to the ice F_b , and the surplus energy left over from the melting, **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 **hihig** then all energy from freezing at sea surface is used only in the open water parts of the cell (ie. F_b will only have the conduction term). The melt energy is partitioned by **frac_energy** between melting ice height and ice extent. However, once ice height drops below **himon0** then all energy melts ice extent.)

subroutine SFC_ALBEDO

The routine *ice_therm.F* calls this routine to determine the surface albedo. There are two calculations provided here:

1. from LANL CICE model

$$\alpha = f_s \alpha_s + (1 - f_s)(\alpha_{i_{min}} + (\alpha_{i_{max}} - \alpha_{i_{min}})(1 - e^{-h_i/h_\alpha}))$$

where f_s is 1 if there is snow, 0 if not; the snow albedo, α_s has two values depending on whether $T_s < 0$ or not; $\alpha_{i_{min}}$ and $\alpha_{i_{max}}$ are ice albedos for thin melting ice, and thick bare ice respectively, and h_α is a scale height.

2. From GISS model (Hansen et al 1983)

$$\alpha = \alpha_i e^{-h_s/h_a} + \alpha_s (1 - e^{-h_s/h_a})$$

where α_i is a constant albedo for bare ice, h_a is a scale height and α_s is a variable snow albedo.

$$\alpha_s = \alpha_1 + \alpha_2 e^{-\lambda_a a_s}$$

where α_1 is a constant, α_2 depends on T_s , a_s is the snow age, and λ_a is a scale frequency. The snow age is calculated in *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:

$$q_1 = f_1 \hat{q}_1 + (1 - f_1) \hat{q}_2$$

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 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, 1srt layer (area_			
↪weighted)									
SI_Tice2	1	SM	C197M1	degC		Sea-Ice Temperature, 2nd layer (area_			
↪weighted)									

(continues on next page)

(continued from previous page)

SI_Qice1	1	SM	C198M1	J/kg	Sea-Ice enthalpy, 1srt 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)
SIfIxAtm	1	SM	M1	W/m^2	net heat flux from the Atmosphere_↵(+dw)
SIfwAtm	1	SM	M1	kg/m^2/s	fresh-water flux to the Atmosphere_↵(+up)
SIfIx2oc	1	SM	M1	W/m^2	heat flux out of the ocean (+up)
SIfw2oc	1	SM	M1	m/s	fresh-water flux out of the ocean_↵(+up)
SIsaltFx	1	SM	M1	psu.kg/m^2	salt flux out of the ocean (+up)
SItOcMxL	1	SM	M1	degC	ocean mixed layer temperature
SIsOcMxL	1	SM	P M1	psu	ocean mixed layer salinity

8.6.1.4 References

Bitz, C.M. and W.H. Lipscombe, 1999: An Energy-Conserving Thermodynamic Model of Sea Ice. *Journal of Geophysical Research*, 104, 15,669 – 15,677.

Hansen, J., G. Russell, D. Rind, P. Stone, A. Lacis, S. Lebedeff, R. Ruedy and L.Travis, 1983: Efficient Three-Dimensional Global Models for Climate Studies: Models I and II. *Monthly Weather Review*, 111, 609 – 662.

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)

Winton, M, 2000: A reformulated Three-layer Sea Ice Model. *Journal of Atmospheric and Ocean Technology*, 17, 525 – 531.

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 and Jinlun Zhang

8.6.2.1 Introduction

Package “seaiice” 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.seaiice (Section 8.6.2.3). A description of key subroutines is given in Section 8.6.2.5. Input fields, units and sign conventions are summarized in Section *Input fields and units*, and available diagnostics output is listed in Section 8.6.2.6.

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

- 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 `exf` to be enabled; no additional CPP options are required.

(see Section [Section 3.5](#)).

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 8.15](#) summarizes the most important ones. For more options see the default `pkg/seaice/SEAIce_OPTIONS.h`.

Table 8.15: Some of the most relevant CPP preprocessor flags in the `seaice`-package.

CPP option	Description
<code>SEAIce_DEBUG</code>	Enhance STDOUT for debugging
<code>SEAIce_ALLOW_DYNAMICS</code>	sea-ice dynamics code
<code>SEAIce_CGRID</code>	LSR solver on C-grid (rather than original B-grid)
<code>SEAIce_ALLOW_EVP</code>	enable use of EVP rheology solver
<code>SEAIce_ALLOW_JFNK</code>	enable use of JFNK rheology solver
<code>SEAIce_ALLOW_KRYLOV</code>	enable use of Krylov rheology solver
<code>SEAIce_LSR_ZEBRA</code>	use a coloring method for LSR solver
<code>SEAIce_EXTERNAL_FLUXES</code>	use EXF-computed fluxes as starting point
<code>SEAIce_ZETA_SMOOTHREG</code>	use differentiable regularization for viscosities
<code>SEAIce_ALLOW_BOTTOMDRAG</code>	enable grounding parameterisation for improved fastice in shallow seas
<code>SEAIce_ITD</code>	run with dynamical sea Ice Thickness Distribution (ITD)
<code>SEAIce_VARIABLE_FREEZING_POINT</code>	enable linear dependence of the freezing point on salinity (by default undefined)
<code>ALLOW_SEAIce_FLOODING</code>	enable snow to ice conversion for submerged sea-ice
<code>SEAIce_VARIABLE_SALINITY</code>	enable sea-ice with variable salinity (by default undefined)
<code>SEAIce_SITRACER</code>	enable sea-ice tracer package (by default undefined)
<code>SEAIce_BICE_STRESS</code>	B-grid only for backward compatibility: turn on ice-stress on ocean
<code>EXPLICIT_SSH_SLOPE</code>	B-grid only for backward compatibility: use ETAN for tilt computations rather than geostrophic velocities

8.6.2.3 Run-time parameters

Run-time parameters (see [Table 8.16](#)) are set in files `data.pkg` (read in `packages_readparms.F`), and `data.seaice` (read in `seaice_readparms.F`).

Enabling the package

A package is switched on/off at run-time by setting (e.g. for SEAIce `useSEAIce = .TRUE.` in `data.pkg`).

General flags and parameters

Table 8.16 lists most run-time parameters.

Table 8.16: Run-time parameters and default values

Name	Default value	Description
SEAICEwriteState	T	write sea ice state to file
SEAICEuseDYNAMICS	T	use dynamics
SEAICEuseJFNK	F	use the JFNK-solver
SEAICEuseTEM	F	use truncated ellipse method
SEAICEuseStrImpCpl	F	use strength implicit coupling in LSR/JFNK
SEAICEuseMetricTerms	T	use metric terms in dynamics
SEAICEuseEVPpickup	T	use EVP pickups
SEAICEuseFluxForm	F	use flux form for 2nd central difference advection scheme
SEAICErestoreUnderIce	F	enable restoring to climatology under ice
useHB87stressCoupling	F	turn on ice-ocean stress coupling following
usePW79thermodynamics	T	flag to turn off zero-layer-thermodynamics for testing
SEAICEadvHeff/Area/Snow/Salt	T	flag to turn off advection of scalar state variables
SEAICEuseFlooding	T	use flood-freeze algorithm
SEAICE_no_slip	F	switch between free-slip and no-slip boundary conditions
SEAICE_deltaTtherm	dTracerLev(1)	thermodynamic timestep
SEAICE_deltaTdyn	dTracerLev(1)	dynamic timestep
SEAICE_deltaTevp	0	EVP sub-cycling time step, values > 0 turn on EVP
SEAICEuseEVPstar	F	use modified EVP* instead of EVP
SEAICEuseEVPprev	F	use yet another variation on EVP*
SEAICEenEVPstarSteps	UNSET	number of modified EVP* iteration
SEAICE_evpAlpha	UNSET	EVP* parameter
SEAICE_evpBeta	UNSET	EVP* parameter
SEAICEaEVPcoeff	UNSET	aEVP parameter
SEAICEaEVPcStar	4	aEVP parameter [KDL16]
SEAICEaEVPalphaMin	5	aEVP parameter [KDL16]
SEAICE_elasticParm	$\frac{1}{3}$	EVP parameter E_0
SEAICE_evpTauRelax	Δt_{EVP}	relaxation time scale T for EVP waves
SEAICEnonLinIterMax	10	maximum number of JFNK-Newton iterations (non-linear)
SEAICElinearIterMax	10	maximum number of JFNK-Krylov iterations (linear)
SEAICE_JFNK_lsIter	(off)	start line search after “lsIter” Newton iterations
SEAICEnonLinTol	1.0E-05	non-linear tolerance parameter for JFNK solver
JFNKgamma_lin_min/max	0.10/0.99	tolerance parameters for linear JFNK solver
JFNKres_tFac	UNSET	tolerance parameter for FGMRES residual
SEAICE_JFNKepsilon	1.0E-06	step size for the FD-Jacobian-times-vector
SEAICE_dumpFreq	dumpFreq	dump frequency
SEAICE_taveFreq	taveFreq	time-averaging frequency
SEAICE_dump_mdsio	T	write snap-shot using MDSIO
SEAICE_tave_mdsio	T	write TimeAverage using MDSIO
SEAICE_dump_mnc	F	write snap-shot using MNC
SEAICE_tave_mnc	F	write TimeAverage using MNC
SEAICE_initialHEFF	0.00000E+00	initial sea-ice thickness
SEAICE_drag	2.00000E-03	air-ice drag coefficient
OCEAN_drag	1.00000E-03	air-ocean drag coefficient
SEAICE_waterDrag	5.50000E+00	water-ice drag
SEAICE_dryIceAlb	7.50000E-01	winter albedo

Table 8.16 – continued from previous page

SEAICE_wetIceAlb	6.60000E-01	summer albedo
SEAICE_drySnowAlb	8.40000E-01	dry snow albedo
SEAICE_wetSnowAlb	7.00000E-01	wet snow albedo
SEAICE_waterAlbedo	1.00000E-01	water albedo
SEAICE_strength	2.75000E+04	sea-ice strength P^*
SEAICE_cStar	20.0000E+00	sea-ice strength paramter C^*
SEAICE_rhoAir	1.3 (or <code>exf</code> value)	density of air (kg/m^3)
SEAICE_cpAir	1004 (or <code>exf</code> value)	specific heat of air (J/kg/K)
SEAICE_lhEvap	2,500,000 (or <code>exf</code> value)	latent heat of evaporation
SEAICE_lhFusion	334,000 (or <code>exf</code> value)	latent heat of fusion
SEAICE_lhSublim	2,834,000	latent heat of sublimation
SEAICE_dalton	1.75E-03	sensible heat transfer coefficient
SEAICE_iceConduct	2.16560E+00	sea-ice conductivity
SEAICE_snowConduct	3.10000E-01	snow conductivity
SEAICE_emissivity	5.50000E-08	Stefan-Boltzman
SEAICE_snowThick	1.50000E-01	cutoff snow thickness
SEAICE_shortwave	3.00000E-01	penetration shortwave radiation
SEAICE_freeze	-1.96000E+00	freezing temp. of sea water
SEAICE_saltFrac	0.0	salinity newly formed ice (fraction of ocean surface salinity)
SEAICE_frazilFrac	0.0	Fraction of surface level negative heat content anomalies (relative to
SEAICEstressFactor	1.00000E+00	scaling factor for ice-ocean stress
Heff/Area/HsnowFile/Hsalt	UNSET	initial fields for variables HEFF/AREA/HSNOW/HSALT
LSR_ERROR	1.00000E-04	sets accuracy of LSR solver
DIFF1	0.0	parameter used in <code>advect.F</code>
HO	5.00000E-01	demarcation ice thickness (AKA lead closing paramter h_0)
MAX_HEFF	1.00000E+01	maximum ice thickness
MIN_ATEMP	-5.00000E+01	minimum air temperature
MIN_LWDOWN	6.00000E+01	minimum downward longwave
MAX_TICE	3.00000E+01	maximum ice temperature
MIN_TICE	-5.00000E+01	minimum ice temperature
IMAX_TICE	10	iterations for ice heat budget
SEAICE_EPS	1.00000E-10	reduce derivative singularities
SEAICE_area_reg	1.00000E-5	minimum concentration to regularize ice thickness
SEAICE_hice_reg	0.05 m	minimum ice thickness for regularization
SEAICE_multDim	1	number of ice categories for thermodynamics
SEAICE_useMultDimSnow	F	use SEAICE_multDim snow categories

Input fields and units

- *HeffFile*: Initial sea ice thickness averaged over grid cell in meters; initializes variable *HEFF*;
- *AreaFile*: Initial fractional sea ice cover, range $[0, 1]$; initializes variable *AREA*;
- *HsnowFile*: Initial snow thickness on sea ice averaged over grid cell in meters; initializes variable *HSNOW*;
- *HsaltFile*: Initial salinity of sea ice averaged over grid cell in g/m^2 ; initializes variable *HSALT*;

8.6.2.4 Description

The MITgcm sea ice model (MITgcm/sim) is based on a variant of the viscous-plastic (VP) dynamic-thermodynamic sea ice model [ZH97] first introduced by [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 [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 [ZH97], EVP [HD97], JFNK [LTSedlacek+10][LFLV14];
- ice-ocean stress can be formulated as in [HB87] or as in [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 [Hib79], the cavitating fluid (CF) model [FWDH92], and the elastic-viscous-plastic (EVP) model [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 [ZH97] for use in a parallel configuration. An EVP model and a free-drift implementation can be selected with runtime flags.

Compatibility with ice-thermodynamics `thsice` package

Note, that by default the `seaice`-package includes the original so-called zero-layer thermodynamics following with a snow cover as in . 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 's [] three-layer thermodynamic model that permits heat storage in ice. Recently, the three-layer thermodynamic model has been reformulated by . The reformulation improves model physics by representing the brine content of the upper ice with a variable heat capacity. It also improves model numerics and consumes less computer time and memory.

The Winton sea-ice thermodynamics have been ported to the MIT GCM; they currently reside under `pkg/thsice`. The package `thsice` is described in section 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 the `seaice`-package with the thermodynamics of `thsice`, compile both packages and turn both package on in `data.pkg`; see an example in `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` have to 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 are used to estimate oceanic forcing from the atmospheric fields.

Dynamics

The momentum equation of the sea-ice model is

$$m \frac{D\mathbf{u}}{Dt} = -mf\mathbf{k} \times \mathbf{u} + \tau_{air} + \tau_{ocean} - m\nabla\phi(0) + \mathbf{F}, \quad (8.2)$$

where $m = m_i + m_s$ is the ice and snow mass per unit area; $\mathbf{u} = u\mathbf{i} + v\mathbf{j}$ is the ice velocity vector; \mathbf{i} , \mathbf{j} , and \mathbf{k} are unit vectors in the x , y , and z directions, respectively; f is the Coriolis parameter; τ_{air} and τ_{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/ρ_0 , where ρ_0 is a reference density) and a term due to snow and ice loading; and $\mathbf{F} = \nabla \cdot \sigma$ is the divergence of the internal ice stress tensor σ_{ij} . Advection of sea-ice momentum is neglected. The wind and ice-ocean stress terms are given by

$$\begin{aligned} \tau_{air} &= \rho_{air} C_{air} |\mathbf{U}_{air} - \mathbf{u}| R_{air} (\mathbf{U}_{air} - \mathbf{u}), \\ \tau_{ocean} &= \rho_{ocean} C_{ocean} |\mathbf{U}_{ocean} - \mathbf{u}| R_{ocean} (\mathbf{U}_{ocean} - \mathbf{u}), \end{aligned}$$

where $\mathbf{U}_{air/ocean}$ are the surface winds of the atmosphere and surface currents of the ocean, respectively; $C_{air/ocean}$ are air and ocean drag coefficients; $\rho_{air/ocean}$ are reference densities; and $R_{air/ocean}$ are rotation matrices that act on the wind/current vectors.

Viscous-Plastic (VP) Rheology

For an isotropic system the stress tensor σ_{ij} ($i, j = 1, 2$) can be related to the ice strain rate and strength by a nonlinear viscous-plastic (VP) constitutive law :

$$\sigma_{ij} = 2\eta(\dot{\epsilon}_{ij}, P)\dot{\epsilon}_{ij} + [\zeta(\dot{\epsilon}_{ij}, P) - \eta(\dot{\epsilon}_{ij}, P)]\dot{\epsilon}_{kk}\delta_{ij} - \frac{P}{2}\delta_{ij}. \quad (8.3)$$

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_{\max} , a measure of ice strength, depends on both thickness h and compactness (concentration) c :

$$P_{\max} = P^* c h \exp\{-C^* \cdot (1 - c)\}, \quad (8.4)$$

with the constants P^* (run-time parameter SEAICE_strength) and $C^* = 20$ (run-time parameter SEAICE_cStar). The nonlinear bulk and shear viscosities η and ζ 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 equal to 2; they are given by:

$$\zeta = \min \left(\frac{P_{\max}}{2 \max(\Delta, \Delta_{\min})}, \zeta_{\max} \right)$$

$$\eta = \frac{\zeta}{e^2}$$

with the abbreviation

$$\Delta = [(\dot{\epsilon}_{11}^2 + \dot{\epsilon}_{22}^2)(1 + e^{-2}) + 4e^{-2}\dot{\epsilon}_{12}^2 + 2\dot{\epsilon}_{11}\dot{\epsilon}_{22}(1 - e^{-2})]^{\frac{1}{2}}.$$

The bulk viscosities are bounded above by imposing both a minimum Δ_{\min} (for numerical reasons, run-time parameter `SEAICE_EPS` with a default value of 10^{-10} s^{-1}) and a maximum $\zeta_{\max} = P_{\max}/\Delta^*$, where $\Delta^* = (5 \times 10^{12}/2 \times 10^4) \text{ s}^{-1}$. (There is also the option of bounding ζ 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 elliptic yield curve by definition.

Defining the CPP-flag `SEAICE_ZETA_SMOOTHREG` in `SEAICE_OPTIONS.h` before compiling replaces the method for bounding ζ by a smooth (differentiable) expression:

$$\begin{aligned} \zeta &= \zeta_{\max} \tanh \left(\frac{P}{2 \min(\Delta, \Delta_{\min}) \zeta_{\max}} \right) \\ &= \frac{P}{2\Delta^*} \tanh \left(\frac{\Delta^*}{\min(\Delta, \Delta_{\min})} \right) \end{aligned} \tag{8.5}$$

where $\Delta_{\min} = 10^{-20} \text{ s}^{-1}$ is chosen to avoid divisions by zero.

LSR and JFNK solver

In the matrix notation, the discretized momentum equations can be written as

$$\mathbf{A}(\mathbf{x}) \mathbf{x} = \mathbf{b}(\mathbf{x}). \tag{8.6}$$

The solution vector \mathbf{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. (8.6) in the sea ice component of the MITgcm, as in many sea ice models, is an iterative Picard solver: in the k -th iteration a linearized form $\mathbf{A}(\mathbf{x}^{k-1}) \mathbf{x}^k = \mathbf{b}(\mathbf{x}^{k-1})$ is solved (in the case of the MITgcm it is a Line Successive (over) Relaxation (LSR) algorithm). Picard solvers converge slowly, but generally the iteration is terminated after only a few non-linear steps and the calculation continues with the next time level. This method is the default method in the MITgcm. The number of non-linear iteration steps or pseudo-time steps can be controlled by the runtime parameter `SEAICEnonLinIterMax` (default is 2).

In order to overcome the poor convergence of the Picard-solver, introduced a Jacobian-free Newton-Krylov solver for the sea ice momentum equations. This solver is also implemented in the MITgcm. The Newton method transforms minimizing the residual $\mathbf{F}(\mathbf{x}) = \mathbf{A}(\mathbf{x}) \mathbf{x} - \mathbf{b}(\mathbf{x})$ to finding the roots of a multivariate Taylor expansion of the residual \mathbf{F} around the previous $(k-1)$ estimate \mathbf{x}^{k-1} :

$$\mathbf{F}(\mathbf{x}^{k-1} + \delta \mathbf{x}^k) = \mathbf{F}(\mathbf{x}^{k-1}) + \mathbf{F}'(\mathbf{x}^{k-1}) \delta \mathbf{x}^k \tag{8.7}$$

with the Jacobian $\mathbf{J} \equiv \mathbf{F}'$. The root $\mathbf{F}(\mathbf{x}^{k-1} + \delta\mathbf{x}^k) = 0$ is found by solving

$$\mathbf{J}(\mathbf{x}^{k-1}) \delta\mathbf{x}^k = -\mathbf{F}(\mathbf{x}^{k-1}) \quad (8.8)$$

for $\delta\mathbf{x}^k$. The next (k -th) estimate is given by $\mathbf{x}^k = \mathbf{x}^{k-1} + a \delta\mathbf{x}^k$. In order to avoid overshoots the factor a is iteratively reduced in a line search ($a = 1, \frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \dots$) until $\|\mathbf{F}(\mathbf{x}^k)\| < \|\mathbf{F}(\mathbf{x}^{k-1})\|$, where $\|\cdot\| = \int \cdot dx^2$ is the L_2 -norm. In practice, the line search is stopped at $a = \frac{1}{8}$. The line search starts after `SEAICE_JFNK_lsIter` non-linear Newton iterations (off by default).

Forming the Jacobian \mathbf{J} explicitly is often avoided as “too error prone and time consuming”. Instead, Krylov methods only require the action of \mathbf{J} on an arbitrary vector \mathbf{w} and hence allow a matrix free algorithm for solving Eq. (8.8). The action of \mathbf{J} can be approximated by a first-order Taylor series expansion:

$$\mathbf{J}(\mathbf{x}^{k-1}) \mathbf{w} \approx \frac{\mathbf{F}(\mathbf{x}^{k-1} + \epsilon\mathbf{w}) - \mathbf{F}(\mathbf{x}^{k-1})}{\epsilon} \quad (8.9)$$

or computed exactly with the help of automatic differentiation (AD) tools. `SEAICE_JFNKepsilon` sets the step size ϵ .

We use the Flexible Generalized Minimum RESidual method with right-hand side preconditioning to solve Eq. (8.8) iteratively starting from a first guess of $\delta\mathbf{x}_0^k = 0$. For the preconditioning matrix \mathbf{P} we choose a simplified form of the system matrix $\mathbf{A}(\mathbf{x}^{k-1})$ where \mathbf{x}^{k-1} is the estimate of the previous Newton step $k-1$. The transformed equation (8.8) becomes

$$\mathbf{J}(\mathbf{x}^{k-1}) \mathbf{P}^{-1} \delta\mathbf{z} = -\mathbf{F}(\mathbf{x}^{k-1}), \quad \text{with} \quad \delta\mathbf{z} = \mathbf{P} \delta\mathbf{x}^k. \quad (8.10)$$

The Krylov method iteratively improves the approximate solution to Eq. (8.10) in subspace $(\mathbf{r}_0, \mathbf{J}\mathbf{P}^{-1}\mathbf{r}_0, (\mathbf{J}\mathbf{P}^{-1})^2\mathbf{r}_0, \dots, (\mathbf{J}\mathbf{P}^{-1})^m\mathbf{r}_0)$ with increasing m ; $\mathbf{r}_0 = -\mathbf{F}(\mathbf{x}^{k-1}) - \mathbf{J}(\mathbf{x}^{k-1}) \delta\mathbf{x}_0^k$ is the initial residual of Eq. (8.8); $\mathbf{r}_0 = -\mathbf{F}(\mathbf{x}^{k-1})$ with the first guess $\delta\mathbf{x}_0^k = 0$. We allow a Krylov-subspace of dimension $m = 50$ and we do not use restarts. The preconditioning operation involves applying \mathbf{P}^{-1} to the basis vectors $\mathbf{v}_0, \mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_m$ of the Krylov subspace. This operation is approximated by solving the linear system $\mathbf{P} \mathbf{w} = \mathbf{v}_i$. Because $\mathbf{P} \approx \mathbf{A}(\mathbf{x}^{k-1})$, we can use the LSR-algorithm already implemented in the Picard solver. Each preconditioning operation uses a fixed number of 10 LSR-iterations avoiding any termination criterion. More details and results can be found in .

To use the JFNK-solver set `SEAICEuseJFNK = .TRUE.`, in the namelist file `data.seaice`; `SEAICE_ALLOW_JFNK` needs to be defined in `SEAICE_OPTIONS.h` and we recommend using a smooth regularization of ζ by defining `SEAICE_ZETA_SMOOTHREG` (see above) for better convergence. The non-linear Newton iteration is terminated when the L_2 -norm of the residual is reduced by γ_{nl} (runtime parameter `SEAICENonLinTol = 1.E-4`, will already lead to expensive simulations) with respect to the initial norm: $\|\mathbf{F}(\mathbf{x}^k)\| < \gamma_{nl} \|\mathbf{F}(\mathbf{x}^0)\|$. Within a non-linear iteration, the linear FGMRES solver is terminated when the residual is smaller than $\gamma_k \|\mathbf{F}(\mathbf{x}^{k-1})\|$ where γ_k is determined by

$$\gamma_k = \begin{cases} \gamma_0 & \text{for } \|\mathbf{F}(\mathbf{x}^{k-1})\| \geq r, \\ \max\left(\gamma_{\min}, \frac{\|\mathbf{F}(\mathbf{x}^{k-1})\|}{\|\mathbf{F}(\mathbf{x}^{k-2})\|}\right) & \text{for } \|\mathbf{F}(\mathbf{x}^{k-1})\| < r, \end{cases} \quad (8.11)$$

so that the linear tolerance parameter γ_k decreases with the non-linear Newton step as the non-linear solution is approached. This inexact Newton method is generally more robust and computationally more efficient than exact methods. Typical parameter choices are $\gamma_0 = \text{JFNKgamma_lin_max} = 0.99$, $\gamma_{\min} = \text{JFNKgamma_lin_min} = 0.1$, and $r = \text{JFNKres_tFac} \times \|\mathbf{F}(\mathbf{x}^0)\|$ with `JFNKres_tFac` = 0.5. We recommend a maximum number of non-linear iterations `SEAICENewtonIterMax` = 100 and a maximum number of Krylov iterations `SEAICEkrylovIterMax` = 50, because the Krylov subspace has a fixed dimension of 50.

Setting `SEAICEuseStrImpCpl = .TRUE.`, turns on “strength implicit coupling” [HJL04] in the LSR-solver and in the LSR-preconditioner for the JFNK-solver. In this mode, the different contributions of the stress divergence terms are re-ordered in order to increase the diagonal dominance of the system matrix. Unfortunately, the convergence rate of the LSR solver is increased only slightly, while the JFNK-convergence appears to be unaffected.

Elastic-Viscous-Plastic (EVP) Dynamics

[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,

$$\frac{1}{E} \frac{\partial \sigma_{ij}}{\partial t} + \frac{1}{2\eta} \sigma_{ij} + \frac{\eta - \zeta}{4\zeta\eta} \sigma_{kk} \delta_{ij} + \frac{P}{4\zeta} \delta_{ij} = \dot{\epsilon}_{ij}. \quad (8.12)$$

The EVP-model uses an explicit time stepping scheme with a short timestep. According to the recommendation of [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 σ_{12} . Introducing the divergence $D_D = \dot{\epsilon}_{11} + \dot{\epsilon}_{22}$, and the horizontal tension and shearing strain rates, $D_T = \dot{\epsilon}_{11} - \dot{\epsilon}_{22}$ and $D_S = 2\dot{\epsilon}_{12}$, respectively, and using the above abbreviations, the equations (8.12) can be written as:

$$\frac{\partial \sigma_1}{\partial t} + \frac{\sigma_1}{2T} + \frac{P}{2T} = \frac{P}{2T\Delta} D_D \quad (8.13)$$

$$\frac{\partial \sigma_2}{\partial t} + \frac{\sigma_2 e^2}{2T} = \frac{P}{2T\Delta} D_T \quad (8.14)$$

$$\frac{\partial \sigma_{12}}{\partial t} + \frac{\sigma_{12} e^2}{2T} = \frac{P}{4T\Delta} D_S \quad (8.15)$$

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) timestep Δt . $E_0 = \frac{1}{3}$ is the default value in the code and close to what is recommended.

To use the EVP solver, make sure that both `SEAICE_CGRID` and `SEAICE_ALLOW_EVP` are defined in `SEAICE_OPTIONS.h` (default). The solver is turned on by setting the sub-cycling time step `SEAICE_deltaTevp` to a value larger than zero. The choice of this time step is under debate. [HD97] recommend order(120) time steps for the EVP solver within one model time step Δt (`deltaTmom`). One can also choose order(120) time steps within the forcing time scale, but then we recommend adjusting the damping time scale T accordingly, by setting either `SEAICE_elasticPlarm` (E_0), so that $E_0 \Delta t = \text{forcing time scale}$, or directly `SEAICE_evpTauRelax` (T) to the forcing time scale. (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 schemes appears to give noisy solutions [Hum01][LKT+12][BFLM13]. This has lead to a modified EVP or EVP* [LKT+12][BFLM13][KDL15]; here, we refer to these variants by modified EVP (mEVP) and adaptive EVP (aEVP) [KDL16]. The main idea is to modify the “natural” time-discretization of the momentum equations:

$$m \frac{D\mathbf{u}}{Dt} \approx m \frac{\mathbf{u}^{p+1} - \mathbf{u}^n}{\Delta t} + \beta^* \frac{\mathbf{u}^{p+1} - \mathbf{u}^p}{\Delta t_{\text{EVP}}} \quad (8.16)$$

where n is the previous time step index, and p is the previous sub-cycling index. The extra “inertial” term $m(\mathbf{u}^{p+1} - \mathbf{u}^n)/\Delta t$ allows the definition of a residual $|\mathbf{u}^{p+1} - \mathbf{u}^p|$ that, as $\mathbf{u}^{p+1} \rightarrow \mathbf{u}^{n+1}$, 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 Δt_{EVP}). Using the terminology of , the evolution equations of stress σ_{ij} and momentum \mathbf{u} can be written as:

$$\sigma_{ij}^{p+1} = \sigma_{ij}^p + \frac{1}{\alpha} \left(\sigma_{ij}(\mathbf{u}^p) - \sigma_{ij}^p \right), \quad (8.17)$$

$$\mathbf{u}^{p+1} = \mathbf{u}^p + \frac{1}{\beta} \left(\frac{\Delta t}{m} \nabla \cdot \sigma^{p+1} + \frac{\Delta t}{m} \mathbf{R}^p + \mathbf{u}_n - \mathbf{u}^p \right). \quad (8.18)$$

\mathbf{R} contains all terms in the momentum equations except for the rheology terms and the time derivative; α and β are free parameters (SEAICE_evAlpha, SEAICE_evBeta) that replace the time stepping parameters SEAICE_deltaTevp (Δt_{EVP}), SEAICE_elasticParm (E_0), or SEAICE_evptauRelax (T). α and β determine the speed of convergence and the stability. Usually, it makes sense to use $\alpha = \beta$, and SEAICE_nEVPstarSteps $\gg (\alpha, \beta)$ [KDL15]. Currently, there is no termination criterion and the number of mEVP iterations is fixed to SEAICE_nEVPstarSteps.

In order to use mEVP in the MITgcm, set SEAICEuseEVPstar = .TRUE., in data.seaice. If SEAICEuseEVPprev = .TRUE., the actual form of equations (8.17) and (8.18) is used with fewer implicit terms and the factor of e^2 dropped in the stress equations (8.14) and (8.15). Although this modifies the original EVP-equations, it turns out to improve convergence [BFLM13].

Another variant is the aEVP scheme [KDL16], where the value of α is set dynamically based on the stability criterion

$$\alpha = \beta = \max \left(\tilde{c} \pi \sqrt{c \frac{\zeta}{A_c} \frac{\Delta t}{\max(m, 10^{-4} \text{ kg})}}, \alpha_{\min} \right) \quad (8.19)$$

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 α can be small and more slowly in areas where the equations are stiff. In practice, aEVP leads to an overall better convergence than mEVP [KDL16]. To use aEVP in the MITgcm set SEAICEaEVPcoeff = \tilde{c} ; this also sets the default values of SEAICEaEVPcStar ($c = 4$) and SEAICEaEVPalphaMin ($\alpha_{\min} = 5$). Good convergence has been obtained with setting these values [KDL16]: SEAICEaEVPcoeff = 0.5, SEAICE_nEVPstarSteps = 500, SEAICEuseEVPstar = .TRUE., SEAICEuseEVPprev = .TRUE.

Note, that probably because of the C-grid staggering of velocities and stresses, mEVP may not converge as successfully as in [KDL15], and that convergence at very high resolution (order 5km) has not been studied yet.

Truncated ellipse method (TEM) for yield curve

In the so-called truncated ellipse method the shear viscosity η is capped to suppress any tensile stress:

$$\eta = \min \left(\frac{\zeta}{e^2}, \frac{\frac{P}{2} - \zeta(\dot{\epsilon}_{11} + \dot{\epsilon}_{22})}{\sqrt{\max(\Delta_{\min}^2, (\dot{\epsilon}_{11} - \dot{\epsilon}_{22})^2 + 4\dot{\epsilon}_{12}^2)}} \right). \quad (8.20)$$

To enable this method, set `#define SEAIce_ALLOW_TEM` in `SEAIce_OPTIONS.h` and turn it on with `SEAIceUseTEM` in `data.seaice`.

Ice-Ocean stress

Moving sea ice exerts a stress on the ocean which is the opposite of the stress τ_{ocean} in Eq. (8.2). This stress is applied directly to the surface layer of the ocean model. An alternative ocean stress formulation is given by [HB87]. Rather than applying τ_{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 windstress and divergence of internal ice stresses: $\delta(z)(\tau_{air} + \mathbf{F})/\rho_0$, see also Eq. 2 of [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 [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 ζ and η are naturally defined at C-points in the center of the grid cell. Discretization requires only averaging of ζ and η to vorticity or Z-points (or ζ -points, but here we use Z in order to avoid confusion with the bulk viscosity) at the bottom left corner of the cell to give $\bar{\zeta}^Z$ and $\bar{\eta}^Z$. 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 $A_{i,j}^w$ and $A_{i,j}^s$, respectively. With these definitions (which follow the model code documentation except that ζ -points have been renamed to Z-points), the strain rates are discretized as:

$$\begin{aligned} \dot{\epsilon}_{11} &= \partial_1 u_1 + k_2 u_2 \\ \Rightarrow (\epsilon_{11})_{i,j}^C &= \frac{u_{i+1,j} - u_{i,j}}{\Delta x_{i,j}^F} + k_{2,i,j}^C \frac{v_{i,j+1} + v_{i,j}}{2} \\ \dot{\epsilon}_{22} &= \partial_2 u_2 + k_1 u_1 \\ \Rightarrow (\epsilon_{22})_{i,j}^C &= \frac{v_{i,j+1} - v_{i,j}}{\Delta y_{i,j}^F} + k_{1,i,j}^C \frac{u_{i+1,j} + u_{i,j}}{2} \\ \dot{\epsilon}_{12} &= \dot{\epsilon}_{21} = \frac{1}{2} \left(\partial_1 u_2 + \partial_2 u_1 - k_1 u_2 - k_2 u_1 \right) \\ \Rightarrow (\epsilon_{12})_{i,j}^Z &= \frac{1}{2} \left(\frac{v_{i,j} - v_{i-1,j}}{\Delta x_{i,j}^V} + \frac{u_{i,j} - u_{i,j-1}}{\Delta y_{i,j}^U} \right. \\ &\quad \left. - k_{1,i,j}^Z \frac{v_{i,j} + v_{i-1,j}}{2} - k_{2,i,j}^Z \frac{u_{i,j} + u_{i,j-1}}{2} \right), \end{aligned}$$

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 $(\epsilon_{12})^Z = 0$ 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 ϕ ; $\Delta x_1 = \Delta x = a \cos \phi \Delta \lambda$, 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:

$$\begin{aligned} k_{1,i,j}^C &= \frac{1}{\Delta y_{i,j}^F} \frac{\Delta y_{i+1,j}^G - \Delta y_{i,j}^G}{\Delta x_{i,j}^F} \\ k_{2,i,j}^C &= \frac{1}{\Delta x_{i,j}^F} \frac{\Delta x_{i,j+1}^G - \Delta x_{i,j}^G}{\Delta y_{i,j}^F} \\ k_{1,i,j}^Z &= \frac{1}{\Delta y_{i,j}^U} \frac{\Delta y_{i,j}^C - \Delta y_{i-1,j}^C}{\Delta x_{i,j}^V} \\ k_{2,i,j}^Z &= \frac{1}{\Delta x_{i,j}^V} \frac{\Delta x_{i,j}^C - \Delta x_{i,j-1}^C}{\Delta y_{i,j}^U} \end{aligned}$$

The stress tensor is given by the constitutive viscous-plastic relation $\sigma_{\alpha\beta} = 2\eta\dot{\epsilon}_{\alpha\beta} + [(\zeta - \eta)\dot{\epsilon}_{\gamma\gamma} - P/2]\delta_{\alpha\beta}$. The stress tensor divergence $(\nabla\sigma)_\alpha = \partial_\beta\sigma_{\beta\alpha}$, 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:

$$\begin{aligned} (\nabla\sigma)_1 &: \frac{1}{A_{i,j}^w} \int_{\text{cell}} (\partial_1\sigma_{11} + \partial_2\sigma_{21}) dx_1 dx_2 \\ &= \frac{1}{A_{i,j}^w} \left\{ \int_{x_2}^{x_2+\Delta x_2} \sigma_{11} dx_2 \Big|_{x_1}^{x_1+\Delta x_1} + \int_{x_1}^{x_1+\Delta x_1} \sigma_{21} dx_1 \Big|_{x_2}^{x_2+\Delta x_2} \right\} \\ &\approx \frac{1}{A_{i,j}^w} \left\{ \Delta x_2 \sigma_{11} \Big|_{x_1}^{x_1+\Delta x_1} + \Delta x_1 \sigma_{21} \Big|_{x_2}^{x_2+\Delta x_2} \right\} \\ &= \frac{1}{A_{i,j}^w} \left\{ (\Delta x_2 \sigma_{11})_{i,j}^C - (\Delta x_2 \sigma_{11})_{i-1,j}^C \right. \\ &\quad \left. + (\Delta x_1 \sigma_{21})_{i,j+1}^Z - (\Delta x_1 \sigma_{21})_{i,j}^Z \right\} \end{aligned}$$

with

$$\begin{aligned} (\Delta x_2 \sigma_{11})_{i,j}^C &= \Delta y_{i,j}^F (\zeta + \eta)_{i,j}^C \frac{u_{i+1,j} - u_{i,j}}{\Delta x_{i,j}^F} \\ &\quad + \Delta y_{i,j}^F (\zeta + \eta)_{i,j}^C k_{2,i,j}^C \frac{v_{i,j+1} + v_{i,j}}{2} \\ &\quad + \Delta y_{i,j}^F (\zeta - \eta)_{i,j}^C \frac{v_{i,j+1} - v_{i,j}}{\Delta y_{i,j}^F} \\ &\quad + \Delta y_{i,j}^F (\zeta - \eta)_{i,j}^C k_{1,i,j}^C \frac{u_{i+1,j} + u_{i,j}}{2} \\ &\quad - \Delta y_{i,j}^F \frac{P}{2} \\ (\Delta x_1 \sigma_{21})_{i,j}^Z &= \Delta x_{i,j}^V \bar{\eta}_{i,j}^Z \frac{u_{i,j} - u_{i,j-1}}{\Delta y_{i,j}^U} \\ &\quad + \Delta x_{i,j}^V \bar{\eta}_{i,j}^Z \frac{v_{i,j} - v_{i-1,j}}{\Delta x_{i,j}^V} \\ &\quad - \Delta x_{i,j}^V \bar{\eta}_{i,j}^Z k_{2,i,j}^Z \frac{u_{i,j} + u_{i,j-1}}{2} \\ &\quad - \Delta x_{i,j}^V \bar{\eta}_{i,j}^Z k_{1,i,j}^Z \frac{v_{i,j} + v_{i-1,j}}{2} \end{aligned}$$

Similarly, we have for the v -equation ($\alpha = 2$):

$$\begin{aligned}
 (\nabla \sigma)_2 &: \frac{1}{A_{i,j}^s} \int_{\text{cell}} (\partial_1 \sigma_{12} + \partial_2 \sigma_{22}) dx_1 dx_2 \\
 &= \frac{1}{A_{i,j}^s} \left\{ \int_{x_2}^{x_2+\Delta x_2} \sigma_{12} dx_2 \Big|_{x_1}^{x_1+\Delta x_1} + \int_{x_1}^{x_1+\Delta x_1} \sigma_{22} dx_1 \Big|_{x_2}^{x_2+\Delta x_2} \right\} \\
 &\approx \frac{1}{A_{i,j}^s} \left\{ \Delta x_2 \sigma_{12} \Big|_{x_1}^{x_1+\Delta x_1} + \Delta x_1 \sigma_{22} \Big|_{x_2}^{x_2+\Delta x_2} \right\} \\
 &= \frac{1}{A_{i,j}^s} \left\{ (\Delta x_2 \sigma_{12})_{i+1,j}^Z - (\Delta x_2 \sigma_{12})_{i,j}^Z \right. \\
 &\quad \left. + (\Delta x_1 \sigma_{22})_{i,j}^C - (\Delta x_1 \sigma_{22})_{i,j-1}^C \right\}
 \end{aligned}$$

with

$$\begin{aligned}
 (\Delta x_1 \sigma_{12})_{i,j}^Z &= \Delta y_{i,j}^U \bar{\eta}_{i,j}^Z \frac{u_{i,j} - u_{i,j-1}}{\Delta y_{i,j}^U} \\
 &\quad + \Delta y_{i,j}^U \bar{\eta}_{i,j}^Z \frac{v_{i,j} - v_{i-1,j}}{\Delta x_{i,j}^V} \\
 &\quad - \Delta y_{i,j}^U \bar{\eta}_{i,j}^Z k_{2,i,j}^Z \frac{u_{i,j} + u_{i,j-1}}{2} \\
 &\quad - \Delta y_{i,j}^U \bar{\eta}_{i,j}^Z k_{1,i,j}^Z \frac{v_{i,j} + v_{i-1,j}}{2} \\
 (\Delta x_2 \sigma_{22})_{i,j}^C &= \Delta x_{i,j}^F (\zeta - \eta)_{i,j}^C \frac{u_{i+1,j} - u_{i,j}}{\Delta x_{i,j}^F} \\
 &\quad + \Delta x_{i,j}^F (\zeta - \eta)_{i,j}^C k_{2,i,j}^C \frac{v_{i,j+1} + v_{i,j}}{2} \\
 &\quad + \Delta x_{i,j}^F (\zeta + \eta)_{i,j}^C \frac{v_{i,j+1} - v_{i,j}}{\Delta y_{i,j}^F} \\
 &\quad + \Delta x_{i,j}^F (\zeta + \eta)_{i,j}^C k_{1,i,j}^C \frac{u_{i+1,j} + u_{i,j}}{2} \\
 &\quad - \Delta x_{i,j}^F \frac{P}{2}
 \end{aligned}$$

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 $(\epsilon_{12})^Z = 0$ on boundaries, we set $\sigma_{21}^Z = 0$, or equivalently $\eta_{i,j}^Z = 0$, on boundaries.

Thermodynamics

***NOTE: THIS SECTION IS TERRIBLY OUT OF DATE**

In its original formulation the sea ice model uses simple thermodynamics following the appendix of [Sem76]. This formulation does not allow storage of heat, that is, the heat capacity of ice is zero. Upward conductive heat flux is parameterized assuming a linear temperature profile and together with a constant ice conductivity. It is expressed as $(K/h)(T_w - T_0)$, where K is the ice conductivity, h the ice thickness, and $T_w - T_0$ the difference between water and ice surface temperatures. This type of model is often referred to as a “zero-layer” model. The surface heat flux is computed in a similar way to that of and .

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 $H_n = \frac{2n-1}{7} h$ for $n \in [1, N]$. The heat fluxes computed for each thickness category is area-averaged to give the total heat flux [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. In order to include the same distribution for snow, set `SEAICE_useMultDimSnow = .TRUE.`; only then, the parameterization of always having a fraction of thin ice is efficient and generally thicker ice is produce [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 ρ 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 [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 [ZWDHSR98]. Snow modifies the effective conductivity according to

$$\frac{K}{h} \rightarrow \frac{1}{\frac{h_s}{K_s} + \frac{h}{K}},$$

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 $z = 0$ [Lepparanta83]. The flood-freeze algorithm is enabled with the CPP-flag `SEAICE_ALLOW_FLOODING` and turned on with run-time parameter `SEAICEuseFlooding=.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:

$$\frac{\partial X}{\partial t} = -\nabla \cdot (\mathbf{u} X) + \Gamma_X + D_X \quad (8.21)$$

where Γ_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 scheme that are available in the 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-timeflags: `SEAICEadvScheme` `` (default=2, is the historic 2nd-order, centered difference scheme), ```DIFF = D_X / \Delta x` (default=0.004).

The MITgcm sea ice model provides the option to use the thermodynamics model of [Win00], which in turn is based on the 3-layer model of [Sem76] and 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 ice thickness and concentration, a positive 2nd-order advection scheme with a SuperBee flux limiter [Roe85] should be used to advect all sea-ice-related quantities of the [Win00] thermodynamic model (runtime flag `thSIceAdvScheme=77` and `thSIce_diffK=D_X=0` in `data.ice`, defaults are 0). Because of the non-linearity 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.

8.6.2.5 Key subroutines

Top-level routine: `seaice_model.F`

```

C      !CALLING SEQUENCE:
C      ...
C      seaice_model (TOP LEVEL ROUTINE)
C      |
C      |-- #ifdef SEAICE_CGRID
C      |      SEAICE_DYNSOLVER
C      |      |
C      |      |-- < compute proxy for geostrophic velocity >
C      |      |
C      |      |-- < set up mass per unit area and Coriolis terms >
C      |      |
C      |      |-- < dynamic masking of areas with no ice >
C      |      |
C      |      |
C      |      #ELSE
C      |      DYNsolver
C      |      #ENDIF
C      |
C      |-- if ( useOBCS )
C      |      OBCS_APPLY_UVICE
C      |
C      |-- if ( SEAICEadvHeff .OR. SEAICEadvArea .OR. SEAICEadvSnow .OR. SEAICEadvSalt )
C      |      SEAICE_ADVDIFF
C      |
C      |      SEAICE_REG_RIDGE
C      |
C      |-- if ( usePW79thermodynamics )
C      |      SEAICE_GROWTH
C      |
C      |-- if ( useOBCS )
C      |      if ( SEAICEadvHeff ) OBCS_APPLY_HEFF
C      |      if ( SEAICEadvArea ) OBCS_APPLY_AREA
C      |      if ( SEAICEadvSALT ) OBCS_APPLY_HSALT
C      |      if ( SEAICEadvSNOW ) OBCS_APPLY_HSNOW
C      |
C      |-- < do various exchanges >
C      |
C      |-- < do additional diagnostics >
C      |
C      o

```

8.6.2.6 SEAICE diagnostics

Diagnostics output is available via the diagnostics package (see Section [sec:pkg:diagnostics]). Available output fields are summarized in the following table:

-----+-----+-----+-----			
<-Name->	<- grid ->	<-- Units -->	<- Tile (max=80c)
-----+-----+-----+-----			
sIceLoad SM	U1 kg/m^2	sea-ice loading (in Mass of ice+snow / area_	
↪unit)			

(continues on next page)

(continued from previous page)

SEA ICE STATE:

```

---
SIarea  |SM      M1|m^2/m^2      |SEAICE fractional ice-covered area [0 to 1]
SIheff  |SM      M1|m          |SEAICE effective ice thickness
SIhsnow |SM      M1|m          |SEAICE effective snow thickness
SIhsalt  |SM      M1|g/m^2        |SEAICE effective salinity
SIuice  |UU      M1|m/s          |SEAICE zonal ice velocity, >0 from West to East
SIvice  |VV      M1|m/s          |SEAICE merid. ice velocity, >0 from South to
->North

```

ATMOSPHERIC STATE AS SEEN BY SEA ICE:

```

---
SIitices |SM  C  M1|K          |Surface Temperature over Sea-Ice (area weighted)
SIuwind  |UM      U1|m/s      |SEAICE zonal 10-m wind speed, >0 increases uVel
SIvwind  |VM      U1|m/s      |SEAICE meridional 10-m wind speed, >0 increases
->uVel
SIsnPrpc |SM      U1|kg/m^2/s      |Snow precip. (+=dw) over Sea-Ice (area weighted)

```

FLUXES ACROSS ICE-OCEAN INTERFACE (ATMOS to OCEAN FOR ICE-FREE REGIONS):

```

---
Sifu     |UU      U1|N/m^2      |SEAICE zonal surface wind stress, >0 increases
->uVel
Sifv     |VV      U1|N/m^2      |SEAICE merid. surface wind stress, >0 increases
->vVel
SIqnet   |SM      U1|W/m^2      |Ocean surface heatflux, turb+rad, >0 decreases
->theta
SIqsw    |SM      U1|W/m^2      |Ocean surface shortwave radiat., >0 decreases
->theta
SIempmr  |SM      U1|kg/m^2/s      |Ocean surface freshwater flux, > 0 increases
->salt
SIqneto  |SM      U1|W/m^2      |Open Ocean Part of SIqnet, turb+rad, >0 decr
->theta
SIqneti  |SM      U1|W/m^2      |Ice Covered Part of SIqnet, turb+rad, >0 decr
->theta

```

FLUXES ACROSS ATMOSPHERE-ICE INTERFACE (ATMOS to OCEAN FOR ICE-FREE REGIONS):

```

---
SIatmQnt |SM      U1|W/m^2      |Net atmospheric heat flux, >0 decreases theta
SIatmFW  |SM      U1|kg/m^2/s      |Net freshwater flux from atmosphere & land
->(+ =down)
SIfwSubl |SM      U1|kg/m^2/s      |Freshwater flux of sublimated ice, >0 decreases
->ice

```

THERMODYNAMIC DIAGNOSTICS:

```

---
SIareaPR |SM      M1|m^2/m^2      |SIarea preceeding ridging process
SIareaPT |SM      M1|m^2/m^2      |SIarea preceeding thermodynamic growth/melt
SIheffPT |SM      M1|m          |SIheff preceeding thermodynamic growth/melt
SIhsnoPT |SM      M1|m          |SIhsnow preceeding thermodynamic growth/melt
SIaQbOCN |SM      M1|m/s          |Potential HEFF rate of change by ocean ice flux
SIaQbATC |SM      M1|m/s          |Potential HEFF rate of change by atm flux over
->ice
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
SIaSbATC |SM      M1|m/s          |HSNOW rate of change by atm flux over sea ice
SIaSbOCN |SM      M1|m/s          |HSNOW rate of change by ocean ice flux

```

(continues on next page)

(continued from previous page)

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_
→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 m^2/s^2	SEAICE strength (with upper and lower limit)
SIZeta SM	M1 m^2/s	SEAICE nonlinear bulk viscosity
SIeta SM	M1 m^2/s	SEAICE nonlinear shear viscosity
SIsigI SM	M1 no units	SEAICE normalized principle stress, component_
→one		
SIsigII SM	M1 no units	SEAICE normalized principle stress, component_
→two		

ADVECTIVE/DIFFUSIVE FLUXES OF SEA ICE variables:		

ADVxHEFF UU	M1 m.m^2/s	Zonal Advective Flux of eff ice thickn
ADVyHEFF VV	M1 m.m^2/s	Meridional Advective Flux of eff ice thickn
SIuheff 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.m^2/s	Zonal Advective Flux of fract area
ADVyAREA VV	M1 m^2/m^2.m^2/s	Meridional Advective Flux of fract area
DFxEAREA UU	M1 m^2/m^2.m^2/s	Zonal Diffusive Flux of fract area
DFyEAREA VV	M1 m^2/m^2.m^2/s	Meridional Diffusive Flux of fract area
ADVxSNOW UU	M1 m.m^2/s	Zonal Advective Flux of eff snow thickn
ADVySNOW VV	M1 m.m^2/s	Meridional Advective Flux of eff snow thickn
DFxESNOW UU	M1 m.m^2/s	Zonal Diffusive Flux of eff snow thickn
DFyESNOW VV	M1 m.m^2/s	Meridional Diffusive Flux of eff snow thickn
ADVxSSLT UU	M1 psu.m^2/s	Zonal Advective Flux of seaice salinity
ADVySSLT VV	M1 psu.m^2/s	Meridional Advective Flux of seaice salinity
DFxESSLT UU	M1 psu.m^2/s	Zonal Diffusive Flux of seaice salinity
DFyESSLT VV	M1 psu.m^2/s	Meridional Diffusive Flux of seaice salinity

8.6.2.7 Experiments and tutorials that use seaice

- Labrador Sea experiment in lab_sea verification directory. }
- seaice_obcs, based on lab_sea
- offline_exf_seaice/input.seaicedt, based on lab_sea
- global_ocean.cs32x15/input.icedyn and global_ocean.cs32x15/input.seaice, global cubed-sphere-experiment with combinations of seaice and thsice

CHAPTER 9

Packages II - Diagnostics and I/O

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

$$\mathcal{J}(\vec{u}) = \sum_i \alpha_i \left(\vec{d}_i^T R_i^{-1} \vec{d}_i \right) + \sum_j \beta_j \vec{u}^T \vec{u} \quad (10.1)$$

$$\vec{d}_i = \mathcal{P}(\vec{m}_i - \vec{o}_i) \quad (10.2)$$

$$\vec{m}_i = \mathcal{SDM}(\vec{v}) \quad (10.3)$$

$$\vec{v} = \mathcal{Q}(\vec{u}) \quad (10.4)$$

$$\vec{u} = \mathcal{R}(\vec{u}') \quad (10.5)$$

using symbols defined in Table 10.1. Per Equation (10.3) model counterparts (\vec{m}_i) to observational data (\vec{o}_i) derive from adjustable model parameters (\vec{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 ($\vec{m}_i - \vec{o}_i$) can be penalized directly in Eq. (10.1) but penalized misfits (\vec{d}_i) more generally derive from $\vec{m}_i - \vec{o}_i$ through the generic \mathcal{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.

symbol	definition
\vec{u}	vector of nondimensional control variables
\vec{v}	vector of dimensional control variables
α_i, β_j	misfit and control cost function multipliers (1 by default)
R_i	data error covariance matrix (R_i^{-1} are weights)
\vec{d}_i	a set of model-data differences
\vec{o}_i	observational data vector
\vec{m}_i	model counterpart to \vec{o}_i
\mathcal{P}	post-processing operator (e.g., a smoother)
\mathcal{M}	forward model dynamics operator
\mathcal{D}	diagnostic computation operator
\mathcal{S}	averaging/subsampling operator
\mathcal{Q}	Pre-processing operator
\mathcal{R}	Pre-conditioning operator

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., \vec{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 (σ_i such that $R_i = \sigma_i^2$) in a file specified via `gencost_errfile`. By default σ_i is assumed to be time-invariant but a σ_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 $\vec{d}_i^T R_i^{-1} \vec{d}_i$ can be replaced with $R_i^{-1/2} \vec{d}_i$ 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

¹ ecco_check may be missing a test for conflicting names...

common `gencost_barfile` saves disk space.

Climatologies of \bar{m}_i can be formed from the time series of model averages in order to compare with climatologies of \bar{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 (\mathcal{P} 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). 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.

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_proc*` are arrays of size `NGENPPROC×NGENCOST` (10 × 20 by default; can be changed in `ecco.h` at compile time). In addition, the `gencost_is3d` internal parameter is reset to true on the fly in all 3D cases in Table 10.3.

parameter	type	function
<code>gencost_name</code>	character(*)	Name of cost term
<code>gencost_barfile</code>	character(*)	File to receive model counterpart \bar{m}_i (See Table 10.3)
<code>gencost_datafile</code>	character(*)	File containing observational data \bar{o}_i
<code>gencost_avgperiod</code>	character(5)	Averaging period for \bar{o}_i and \bar{m}_i (see text)
<code>gencost_outputlevel</code>	integer	Greater than 0 will output misfit fields
<code>gencost_errfile</code>	character(*)	Uncertainty field name (not used in Section 10.1.2)
<code>gencost_mask</code>	character(*)	Mask file name root (used only in Section 10.1.2)
<code>mult_gencost</code>	real	Multiplier α_i (default: 1)
<code>gencost_preproc</code>	character(*)	Preprocessor names
<code>gencost_preproc_c</code>	character(*)	Preprocessor character arguments
<code>gencost_preproc_i</code>	integer(*)	Preprocessor integer arguments
<code>gencost_preproc_r</code>	real(*)	Preprocessor real arguments
<code>gencost_posproc</code>	character(*)	Post-processor names
<code>gencost_posproc_c</code>	character(*)	Post-processor character arguments
<code>gencost_posproc_i</code>	integer(*)	Post-processor integer arguments
<code>gencost_posproc_r</code>	real(*)	Post-processor real arguments
<code>gencost_spmn</code>	real	Data less than this value will be omitted
<code>gencost_spmx</code>	real	Data greater than this value will be omitted
<code>gencost_spzero</code>	real	Data points equal to this value will be omitted
<code>gencost_startdate1</code>	integer	Start date of observations (YYYYMMDD)
<code>gencost_startdate2</code>	integer	Start date of observations (HHMMSS)
<code>gencost_is3d</code>	logical	Needs to be true for 3D fields
<code>gencost_enddate1</code>	integer	Not fully implemented (used only in Section 10.1.3)
<code>gencost_enddate2</code>	integer	Not fully implemented (used only in Section 10.1.3)

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.

variable name	description	remarks
<code>m_eta</code>	sea surface height	free surface + ice + global steric correction
<code>m_sst</code>	sea surface temperature	first level potential temperature
<code>m_sss</code>	sea surface salinity	first level salinity
<code>m_bp</code>	bottom pressure	<code>phiHydLow</code>
<code>m_siarea</code>	sea-ice area	from <code>pkg/seaice</code>
<code>m_siheff</code>	sea-ice effective thickness	from <code>pkg/seaice</code>
<code>m_sihsnow</code>	snow effective thickness	from <code>pkg/seaice</code>
<code>m_theta</code>	potential temperature	three-dimensional
<code>m_salt</code>	salinity	three-dimensional
<code>m_UE</code>	zonal velocity	three-dimensional
<code>m_VN</code>	meridional velocity	three-dimensional
<code>m_ustress</code>	zonal wind stress	from <code>pkg/exf</code>
<code>m_vstress</code>	meridional wind stress	from <code>pkg/exf</code>
<code>m_uwind</code>	zonal wind	from <code>pkg/exf</code>
<code>m_vwind</code>	meridional wind	from <code>pkg/exf</code>
<code>m_atemp</code>	atmospheric temperature	from <code>pkg/exf</code>
<code>m_aqh</code>	atmospheric specific humidity	from <code>pkg/exf</code>
<code>m_precip</code>	precipitation	from <code>pkg/exf</code>
<code>m_sdown</code>	downward shortwave	from <code>pkg/exf</code>
<code>m_lwdown</code>	downward longwave	from <code>pkg/exf</code>
<code>m_wspeed</code>	wind speed	from <code>pkg/exf</code>
<code>m_diffkr</code>	vertical/diapycnal diffusivity	three-dimensional, constant
<code>m_kapgm</code>	GM diffusivity	three-dimensional, constant
<code>m_kapredi</code>	isopycnal diffusivity	three-dimensional, constant
<code>m_geothermalflux</code>	geothermal heat flux	constant
<code>m_bottomdrag</code>	bottom drag	constant

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.

name	description	<code>gencost_preproc_i, _r, or _c</code>
<code>gencost_preproc</code>		
<code>clim</code>	Use climatological misfits	integer: no. of records per climatological cycle
<code>mean</code>	Use time mean of misfits	—
<code>anom</code>	Use anomalies from time mean	—
<code>variaweight</code>	Use time-varying weight W_i	—
<code>nosumsq</code>	Use linear misfits	—
<code>factor</code>	Multiply \vec{m}_i by a scaling factor	real: the scaling factor
<code>gencost_posproc</code>		
<code>smooth</code>	Smooth misfits	character: smoothing scale file
		integer: smoother # of time steps

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 [MGZ+99][HWP+11][FWL+15]. 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 scalar² 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 65z) that can be used via `cost_gencost_boxmean.F` (Section 10.1.2).

variable name	description	remarks
<code>m_boxmean_theta</code>	mean of theta over box	specify box
<code>m_boxmean_salt</code>	mean of salt over box	specify box
<code>m_boxmean_eta</code>	mean of SSH over box	specify box
<code>m_horflux_vol</code>	volume transport through section	specify transect

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`, and `cost_gencost_transp.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`.

² The quadratic option in fact does not yet exist in `cost_gencost_boxmean.F...`

Table 10.6: Pre-defined gencost_name special cases (as of checkpoint 65z; Section 10.1.3).

name	description	remarks
sshv4-mdt	sea surface height	mean dynamic topography (SSH - geod)
sshv4-tp	sea surface height	Along-Track Topex/Jason SLA (level 3)
sshv4-ers	sea surface height	Along-Track ERS/Envisat SLA (level 3)
sshv4-gfo	sea surface height	Along-Track GFO class SLA (level 3)
sshv4-lsc	sea surface height	Large-Scale SLA (from the above)
sshv4-gmsl	sea surface height	Global-Mean SLA (from the above)
bpv4-grace	bottom pressure	GRACE maps (level 4)
sstv4-amsre	sea surface temperature	Along-Swath SST (level 3)
sstv4-amsre-lsc	sea surface temperature	Large-Scale SST (from the above)
si4-cons	sea ice concentration	needs sea-ice adjoint (level 4)
si4-deconc	model sea ice deficiency	proxy penalty (from the above)
si4-exconc	model sea ice excess	proxy penalty (from the above)
transp_trVol	volume transport	specify masks (Section 10.1.2)
transp_trHeat	heat transport	specify masks (Section 10.1.2)
transp_trSalt	salt transport	specify masks (Section 10.1.2)

10.1.4 Key Routines

TBA... ecco_readparms.F, ecco_check.F, ecco_summary.F, ... cost_generic.F, cost_gencost_boxmean.F, ecco_toolbox.F, ... ecco_phys.F, cost_gencost_customize.F, cost_averagesfields.F,...

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, ... ECCO_CTRL_DEPRECATED, ... 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 a 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 (`profilesXCincllPointOverlap*` 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`

```
#
# \*****\*
# 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_YYYYMMDD(iPROF) ;
prof_YYYYMMDD:missing_value = -9999. ;
prof_YYYYMMDD: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) ;
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" ;
```

(continues on next page)

(continued from previous page)

```
prof_Tweight:missing_value = -9999. ;
}
```

10.3 CTRL: Model Parameter Adjustment Capability

Author: Gael Forget

The parameters available for configuring generic cost terms in `data.ctrl` are given in [Table 10.7](#).

Table 10.7: Parameters in `ctrl_nml_genarr` namelist in `data.ctrl`. The `*` can be replaced by `arr2d`, `arr3d`, or `tim2d` for time-invariant two and three dimensional controls and time-varying 2D controls, respectively. Parameters for `genarr2d`, `genarr3d`, and `gentime2d` are arrays of length `maxCtrlArr2D`, `maxCtrlArr3D`, and `maxCtrlTim2D`, respectively, with one entry per term in the cost function.

parameter	type	function
<code>xx_gen*_file</code>	character(*)	Control Name: prefix from Table 10.8 + suffix.
<code>xx_gen*_weight</code>	character(*)	Weights in the form of $\sigma_{\vec{u}_j}^{-2}$
<code>xx_gen*_bounds</code>	real(5)	Apply bounds
<code>xx_gen*_preproc</code>	character(*)	Control preprocessor(s) (see Table 10.9)
<code>xx_gen*_preproc_c</code>	character(*)	Preprocessor character arguments
<code>xx_gen*_preproc_i</code>	integer(*)	Preprocessor integer arguments
<code>xx_gen*_preproc_r</code>	real(*)	Preprocessor real arguments
<code>gen*Precond</code>	real	Preconditioning factor (= 1 by default)
<code>mult_gen*</code>	real	Cost function multiplier β_j (= 1 by default)
<code>xx_gentim2d_period</code>	real	Frequency of adjustments (in seconds)
<code>xx_gentim2d_startdate1</code>	integer	Adjustment start date
<code>xx_gentim2d_startdate2</code>	integer	Default: model start date
<code>xx_gentim2d_cumsum</code>	logical	Accumulate control adjustments
<code>xx_gentim2d_glosum</code>	logical	Global sum of adjustment (output is still 2D)

Table 10.8: Generic control prefixes implemented as of checkpoint 65z.

	name	description
2D, time-invariant controls	genarr2d	
	xx_etan	initial sea surface height
	xx_bottomdrag	bottom drag
	xx_geothermal	geothermal heat flux
3D, time-invariant controls	genarr3d	
	xx_theta	initial potential temperature
	xx_salt	initial salinity
	xx_kapgm	GM coefficient
	xx_kapredi	isopycnal diffusivity
	xx_diffkr	diapycnal diffusivity
2D, time-varying controls	gentim2D	
	xx_atemp	atmospheric temperature
	xx_aqh	atmospheric specific humidity
	xx_swdown	downward shortwave
	xx_lwdown	downward longwave
	xx_precip	precipitation
	xx_uwind	zonal wind
	xx_vwind	meridional wind
	xx_tauu	zonal wind stress
	xx_tauv	meridional wind stress
	xx_gen_precip	globally averaged precipitation?

Table 10.9: `xx_gen????d_preproc` options implemented as of checkpoint 65z. Notes: “a”: 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.

name	description	arguments
WC01	Correlation modeling	integer: operator type (default: 1)
smooth	Smoothing without normalization	integer: operator type (default: 1)
docycle	Average period replication	integer: cycle length
replicate	Alias for docycle	(units of <code>xx_gentim2d_period</code>)
rmcycle	Periodic average subtraction	integer: cycle length
variaweight	Use time-varying weight	—
<code>noscaling:math: ^{1/a}</code>	Do not scale with <code>xx_gen*_weight</code>	—
documul	Sets <code>xx_gentim2d_cumsum</code>	—
doglomean	Sets <code>xx_gentim2d_glosum</code>	—

The control problem is non-dimensional by default, as reflected in the omission of weights in control penalties [$(\tilde{u}_j^T \tilde{u}_j$ in (10.1)]. Non-dimensional controls (\tilde{u}_j) are scaled to physical units (\vec{v}_j) through multiplication by the respective uncertainty fields ($\sigma_{\tilde{u}_j}$), as part of the generic preprocessor \mathcal{Q} in (10.4). Besides the scaling of \tilde{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 `xx_gen*_preproc = 'WC01'`. Alternatively, setting `xx_gen*_preproc = 'smooth'` activates the smoothing part of WC01, but omits the normalization. Additionally, bounds for the controls can be specified by setting `xx_gen*_bounds`. In forward mode, adjustments to the i^{th} control are clipped so that they remain between `xx_gen*_bounds(i, 1)` and `xx_gen*_bounds(i, 4)`. If `xx_gen*_bounds(i, 1) < 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 `xx_gentim2d_period`. The generic control package interprets special values of `xx_gentim2d_period` in the same way as the `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 `variaweight`, in which case the `xx_gentim2d_weight` file must contain as many records as the control parameter time series itself (approximately the run length divided by `xx_gentim2d_period`).

The parameter `mult_gen*` sets the multiplier for the corresponding cost function penalty [β_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 `gen*Precond` (`= 1` by default).

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 [GL89].

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 (`ls_opt_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, $\mathbf{x}(\mathbf{i}+1)$ 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

$$xdiff(i, 1) = xx(i - 1) + tact(i - 1, 1) * dd(i - 1)$$

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

$$xdiff(i, 2) = xx(i - 1) + tact(i - 1, 2) * dd(i - 1)$$

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

$$xx(i) = xdiff(i, 2) = xx(i - 1) + tact(i - 1, 2) * dd(i - 1)$$

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 Isopt.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 below 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 $jmax \leq 0$)

10.5.3.4 OPWARM, OPWARMD files

Two files retain values of previous iterations which are used in latest iteration to update Hessian:

- **OPWARM**: contains index settings and scalar variables

n = nn	no. of control variables
fc = ff	cost value of last iteration
isize	no. of bytes per record in OPWARMD
m = nupdate	max. no. of updates for Hessian
jmin, jmax	pointer indices for OPWARMD file (cf. below)
gnorm0	norm of first (cold start) gradient gg
iabsiter	total number of iterations with respect to cold start

- **OPWARMD**: contains vectors (control and gradient)

entry	name	description
1	$xx(i)$	control vector of latest iteration
2	$gg(i)$	gradient of latest iteration
3	$xdiff(i), diag$	preconditioning vector; (1,...,1) for cold start
$2*jmax+2$	$gold=g(i)-g(i-1)$	for last update ($jmax$)
$2*jmax+3$	$xdiff=tact*d=xx(i)-xx(i-1)$	for last update ($jmax$)

Example 1: $jmin = 1$, $jmax = 3$, $mupd = 5$

```

  1   2   3   |   4   5   6   7   8   9   empty   empty
|___|___|___| | |___|___| |___|___| |___|___| |___|___|
      0       |       1       2       3

```

Example 2: $jmin = 3$, $jmax = 7$, $mupd = 5$ ----> $jmax = 2$

```

  1   2   3   |
|___|___|___| | |___|___| |___|___| |___|___| |___|___|
      |       |       6       7       3       4       5

```

10.5.3.5 Error handling

```

lsopt_top
|
|---- check arguments
|---- CALL INSTORE
|
|    |---- determine whether OPWARM I available:
|    * if no: cold start: create OPWARM I
|    * if yes: warm start: read from OPWARM I
|    create or open OPWARM D
|
|---- check consistency between OPWARM I 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 OPWARM D (first 3 entries)
|    |
|    >>> else: WARM start: <<<
|        read xx(i), gg(i) from OPWARM D (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 <<<
|    (    |    ifail = 4
|    )    |
|    (    |---- compute step size: tact(i-1)
|    )    |---- compute update: xdiff(i) = xx(i-1) + tact(i-1)*dd(i-1)
|    (
|    )---- >>> if ifail = 4 <<<
|    (    goto 1000
|    )
|    (---- CALL OPTLINE / LSLINE
|    )    |
...    ...    ...

```

```

...    ...
|    )
|    (---- CALL OPTLINE / LSLINE

```

(continues on next page)

(continued from previous page)

```

|      )
|      (      |---- /// loop over simulations
|      )
|      (      )---- CALL SIMUL
|      )      (      |
|      (      )      |---- input: xdiff(i)
|      )      (      |---- output: ff(i), gg(i)
|      (      )      |---- >>> if ONLINE <<<
|      )      (      runs model and adjoint
|      (      )      >>> if OFFLINE <<<
|      )      (      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
|      )      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)
|      )---- >>> if ifail = 7,8,9 <<<
|      (      goto 1000
|      )
...    ...

```

```

...    ...
|      )
|      (---- store new values xx(i), gg(i) to OPWARMD (first 2 entries)
|      )---- >>> if ifail = 7,8,9 <<<
|      (      goto 1000

```

(continues on next page)

(continued from previous page)

```

|      )
|      (---- 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 dstore
|      (      |      |
|      )      |      |---- read preconditioner of previous iteration diag(i-1)
|      (      |      |      from OPWARMD (3rd entry)
|      )      |
|      (      |---- compute new preconditioner diag(i), based upon diag(i-1),
|      )      |      gg(i)-gg(i-1), xx(i)-xx(i-1)
|      (      |
|      )      |---- call dstore
|      (      |
|      )      |---- write new preconditioner diag(i) to OPWARMD (3rd entry)
|      (
|---- \\ 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.
|
|---- print final information
|
0

```

10.6 Test Cases For Estimation Package Capabilities

First, download the model as explained in *Getting Started with MITgcm* via the MITgcm git server

```
% git clone https://github.com/user_name/MITgcm.git
```

Then, download the setup from the *MITgcm_contrib/* area by logging into the cvs server

```
% setenv CVSROOT ':pserver:cvsanon@mitgcm.org:/u/gcmlpack'
% cvs login
%      ( enter the CVS password: "cvsanon" )
```

and following the directions provided [here](#) for `global_oce_cs32` or [here](#) for `global_oce_llc90`. 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 this site](#). 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_llc90/` tests with fewer than 12 cores (96 for adjoint tests) to avoid potential memory overloads. `global_oce_llc90/` (595M) uses the same LLC90 grid as the production *ECCO version 4* setup does [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
%
% #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
```

CHAPTER 11

Under Development

CHAPTER 12

Previous Applications of MITgcm

Bibliography

- [Adc95] A. Adcroft. *Numerical Algorithms for use in a Dynamical Model of the Ocean*. PhD thesis, Imperial College, London, 1995.
- [AC04] A. Adcroft and J.-M. Campin. Re-scaled height coordinates for accurate representation of free-surface flows in ocean circulation models. *Ocean Modelling*, 7:269–284, 2004. doi:10.1016/j.ocemod.2003.09.003.
- [ACHM04] A. Adcroft, J.-M. Campin, C. Hill, and J. Marshall. Implementation of an atmosphere-ocean general circulation model on the expanded spherical cube. *Mon.~Wea.~Rev.*, 132:2845–2863, 2004. URL: http://mitgcm.org/pdfs/mwr_2004.pdf, doi:10.1175/MWR2823.1.
- [AHCampin+04] A. Adcroft, C. Hill, J.-M. Campin,, J. Marshall, and P. Heimbach. Overview of the formulation and numerics of the MITgcm. In *Proceedings of the ECMWF seminar series on Numerical Methods, Recent developments in numerical methods for atmosphere and ocean modelling*, 139–149. ECMWF, 2004. URL: <http://mitgcm.org/pdfs/ECMWF2004-Adcroft.pdf>.
- [AHM99] A., Adcroft, C. Hill, and J. Marshall. A new treatment of the coriolis terms in c-grid models at both high and low resolutions. *Mon.~Wea.~Rev.*, 127:1928–1936, 1999. URL: http://mitgcm.org/pdfs/mwr_1999.pdf, doi:10.1175/1520-0493%281999%29127<1928:ANTOTC>2.0.CO;2.
- [AHM97] A.J. Adcroft, C.N. Hill, and J. Marshall. Representation of topography by shaved cells in a height coordinate ocean model. *Mon.~Wea.~Rev.*, 125:2293–2315, 1997. URL: http://mitgcm.org/pdfs/mwr_1997.pdf, doi:10.1175/1520-0493%281997%29125<2293:ROTBSC>2.0.CO;2.
- [AM98] A.J. Adcroft and D. Marshall. How slippery are piecewise-constant coastlines in numerical ocean models? *Tellus*, 50(1):95–108, 1998.
- [AL77] A. Arakawa and V. Lamb. Computational design of the basic dynamical processes of the ucla general circulation model. *Meth. Comput. Phys.*, 17:174–267, 1977.
- [BFLM13] S. Bouillon, T. Fichefet, V. Legat, and G. Madec. The elastic-viscous-plastic method revisited. *Ocean Modelling*, 71(0):2–12, 2013. Arctic Ocean. URL: <http://dx.doi.org/10.1016/j.ocemod.2013.05.013>, doi:10.1016/j.ocemod.2013.05.013.
- [BMP75] K. Bryan, S. Manabe, and R.C. Pacanowski. A global ocean-atmosphere climate model. part ii. the oceanic circulation. *J.~Phys.~Oceanogr.*, 5:30–46, 1975.
- [CAHM04] J.-M. Campin, A. Adcroft, C. Hill, and J. Marshall. Conservation of properties in a free-surface model. *Ocean Modelling*, 6:221–244, 2004.
- [CMF08] J.-M. Campin, J. Marshall, and D. Ferreira. Sea-ice ocean coupling using a rescaled vertical coordinate z^* . *Ocean Modelling*, 24(1–2):1–14, 2008. doi:10.1016/j.ocemod.2008.05.005.

- [CMKL+14] K. Castro-Morales, F. Kauker, M. Losch, S. Hendricks, K. Riemann-Campe, and R. Gerdes. Sensitivity of simulated Arctic sea ice to realistic ice thickness distributions and snow parameterizations. *J.~Geophys.~Res.*, 119(1):559–571, 2014. URL: <http://dx.doi.org/10.1002/2013JC009342>, doi:10.1002/2013JC009342.
- [Cho90] M-D. Chou. Parameterizations for the absorption of solar radiation by α_2 and κ_2 with applications to climate studies. *J.~Clim.*, 3:209–217, 1990.
- [Cho92] M-D. Chou. A solar radiation model for use in climate studies. *J.~Atmos.~Sci.*, 49:762–772, 1992.
- [CS94] M-D. Chou and M.J. Suarez. An efficient thermal infrared radiation parameterization for use in general circulation models. NASA Technical Memorandum 104606-Vol 3, National Aeronautics and Space Administration, NASA; Goddard Space Flight Center; Greenbelt (MD), 20771; USA, 1994. <http://www.gmao.nasa.gov/>.
- [Cla70] R.H. Clarke. Observational studies in the atmospheric boundary layer. *Q.~J.~R.~Meteorol.~Soc.*, 96:91–114, 1970.
- [Cox87] M.D. Cox. An isopycnal diffusion in a z-coordinate ocean model. *Ocean modelling*, 74:1–5 (Unpublished manuscript), 1987.
- [DT94] R.S. Defries and J.R.G. Townshend. Ndzi-derived land cover classification at global scales. *Int’l J. Rem. Sens.*, 15:3567–3586, 1994.
- [DS89] J.L. Dorman and P.J. Sellers. A global climatology of albedo, roughness length and stomatal resistance for atmospheric general circulation models as represented by the simple biosphere model (sib). *J.~Appl.~Meteor.*, 28:833–855, 1989.
- [FWDH92] G.M. Flato and W.D. Hibler. Modeling pack ice as a cavitating fluid. *J.~Phys.~Oceanogr.*, 22:626–651, 1992.
- [FRM83] P. Fofonoff and Jr. R. Millard. Algorithms for computation of fundamental properties of seawater. UNESCO Technical Papers in Marine Science 44, UNESCO, Paris, 1983.
- [FCH+15] G. Forget, J.-M. Campin, P. Heimbach, C. N. Hill, R. M. Ponte, and C. Wunsch. ECCO version 4: an integrated framework for non-linear inverse modeling and global ocean state estimation. *Geoscientific Model Development*, 8(10):3071–3104, 2015. URL: <http://www.geosci-model-dev.net/8/3071/2015/>, doi:10.5194/gmd-8-3071-2015.
- [FWL+15] Ichiro Fukumori, Ou Wang, William Llovel, Ian Fenty, and Gael Forget. A near-uniform fluctuation of ocean bottom pressure and sea level across the deep ocean basins of the arctic ocean and the nordic seas. *Progress in Oceanography*, 134(0):152 – 172, 2015. URL: <http://www.sciencedirect.com/science/article/pii/S0079661115000245>, doi:<http://dx.doi.org/10.1016/j.pcean.2015.01.013>.
- [GGL90] P. Gaspar, Y. Grégoris, and J.-M. Lefevre. A simple eddy kinetic energy model for simulations of the oceanic vertical mixing: tests at station papa and long-term upper ocean study site. *J.~Geophys.~Res.*, 95 (C9):16,179–16,193, 1990.
- [GM90] P.R. Gent and J.C. McWilliams. Isopycnal mixing in ocean circulation models. *J.~Phys.~Oceanogr.*, 20:150–155, 1990.
- [GWMM95] P.R. Gent, J. Willebrand, T.J. McDougall, and J.C. McWilliams. Parameterizing eddy-induced tracer transports in ocean circulation models. *J.~Phys.~Oceanogr.*, 25:463–474, 1995.
- [GKW91] R. Gerdes, C. Koberle, and J. Willebrand. The influence of numerical advection schemes on the results of ocean general circulation models. *Clim.~Dynamics*, 5(4):211–226, 1991. doi:10.1007/BF00210006.
- [GL89] J.C. Gilbert and C. Lemaréchal. Some numerical experiments with variable-storage quasi-newton algorithms. *Math. Programming*, 45:407–435, 1989.
- [Gil82] A.E. Gill. *Atmosphere-Ocean Dynamics*. Academic Press, New York, 1982.
- [Gri98] S.M. Griffies. The Gent-McWilliams skew flux. *J.~Phys.~Oceanogr.*, 28:831–841, 1998.

- [GGP+98] S.M. Griffies, A. Gnanadesikan, R.C. Pacanowski, V. Larichev, J.K. Dukowicz, and R.D. Smith. Isoneutral diffusion in a z-coordinate ocean model. *J.~Phys.~Oceanogr.*, 28:805–830, 1998.
- [GH00] S.M. Griffies and R.W. Hallberg. Biharmonic friction with a smagorinsky-like viscosity for use in large-scale eddy-permitting ocean models. *Mon.~Wea.~Rev.*, 128(8):2935–2946, 2000.
- [HW65] F.H. Harlow and J.E. Welch. Numerical calculation of time-dependent viscous incompressible flow of fluid with free surface. *Physics of Fluids*, 8:2182–2189, 1965.
- [HWP+11] Patrick Heimbach, Carl Wunsch, Rui M Ponte, Gael Forget, Chris Hill, and Jean Utke. Timescales and regions of the sensitivity of Atlantic meridional volume and heat transport: toward observing system design. *Deep Sea Research Part II: Topical Studies in Oceanography*, 58(17):1858–1879, 2011.
- [HS94] I.M. Held and M.J. Suarez. A proposal for the intercomparison of the dynamical cores of atmospheric general circulation models. *Bulletin of the American Meteorological Society*, 75(10):1825–1830, 1994.
- [HL88] H.M. Helfand and J.C. Labraga. Design of a non-singular level 2.5 second-order closure model for the prediction of atmospheric turbulence. *J.~Atmos.~Sci.*, 45:113–132, 1988.
- [HS95] H.M. Helfand and S.D. Schubert. Climatology of the simulated great plains low-level jet and its contribution to the continental moisture budget of the united states. *J.~Clim.*, 8:784–806, 1995.
- [Hib79] W.D. Hibler, III. A dynamic thermodynamic sea ice model. *J.~Phys.~Oceanogr.*, 9:815–846, 1979.
- [Hib80] W.D. Hibler, III. Modeling a variable thickness sea ice cover. *Mon.~Wea.~Rev.*, 1:1943–1973, 1980.
- [Hib84] W.D. Hibler, III. The role of sea ice dynamics in modeling co₂ increases. In J. E. Hansen and T. Takahashi, editors, *Climate processes and climate sensitivity*, volume 29 of Geophysical Monograph, pages 238–253. AGU, Washington, D.C., 1984.
- [HB87] W.D. Hibler, III and K. Bryan. A diagnostic ice-ocean model. *J.~Phys.~Oceanogr.*, 17(7):987–1015, 1987.
- [HAJM99] C. Hill, A. Adcroft, D. Jamous, and J. Marshall. A strategy for terascale climate modeling. In *Proceedings of the Eighth ECMWF Workshop on the Use of Parallel Processors in Meteorology*, 406–425. World Scientific, 1999.
- [HM95] C. Hill and J. Marshall. Application of a parallel navier-stokes model to ocean circulation in parallel computational fluid dynamics. In N. Satofuka A. Ecer, J. Periaux and S. Taylor, editors, *Implementations and Results Using Parallel Computers*, pages 545–552. Elsevier Science B.V.: New York, 1995.
- [HHA99] J. C. Hoe, C. Hill, and A. Adcroft. A personal supercomputer for climate research. In *SC’99: Proceedings of the 1999 ACM/IEEE Conference on Supercomputing*, 59. IEEE, 1999. doi:10.1109/SC.1999.10009.
- [Hol78] W.R. Holland. The role of mesoscale eddies in the general circulation of the ocean-numerical experiments using a wind-driven quasi-geostrophic model. *J.~Phys.~Oceanogr.*, 8:363–392, 1978.
- [HL75] W.R. Holland and L.B. Lin. On the origin of mesoscale eddies and their contribution to the general circulation of the ocean. i. a preliminary numerical experiment. *J.~Phys.~Oceanogr.*, 5:642–657, 1975.
- [Hun01] E.C. Hunke. Viscous-plastic sea ice dynamics with the EVP model: linearization issues. *J.~Comput.~Phys.*, 170:18–38, 2001. doi:10.1006/jcph.2001.6710.
- [HD97] E.C. Hunke and J.K. Dukowicz. An elastic-viscous-plastic model for sea ice dynamics. *J.~Phys.~Oceanogr.*, 27:1849–1867, 1997.
- [HJL04] J.K. Hutchings, H. Jasak, and S.W. Laxon. A strength implicit correction scheme for the viscous-plastic sea ice model. *Ocean Modelling*, 7(1–2):111–133, 2004. doi:10.1016/S1463-5003(03)00040-4.
- [JM95] D. R. Jackett and T. J. McDougall. Minimal adjustment of hydrographic profiles to achieve static stability. *J.~Atmos.~Ocean.~Technol.*, 12(4):381–389, 1995.
- [KDL15] M. Kimmritz, S. Danilov, and M. Losch. On the convergence of the modified elastic-viscous-plastic method of solving for sea-ice dynamics. *J.~Comput.~Phys.*, 296:90–100, 2015. doi:10.1016/j.jcp.2015.04.051.

- [KDL16] M. Kimmritz, S. Danilov, and M. Losch. The adaptive EVP method for solving the sea ice momentum equation. *Ocean Modelling*, 101:59–67, 2016. doi:10.1016/j.ocemod.2016.03.004.
- [KL10] J.M. Klymak and S.M. Legg. A simple mixing scheme for models that resolve breaking internal waves. *Ocean Modelling*, 33:224–234, 2010. doi:10.1016/j.ocemod.2010.02.005.
- [Kon75] J. Kondo. Air-sea bulk transfer coefficients in diabatic conditions. *Bound.-Layer-Meteorol.*, 9:91–112, 1975.
- [KS91] R.D. Koster and M.J. Suarez. A simplified treatment of sib’s land surface albedo parameterization. NASA Technical Memorandum 104538, National Aeronautics and Space Administration, NASA; Goddard Space Flight Center; Greenbelt (MD), 20771; USA, 1991. <http://www.gmao.nasa.gov/>.
- [KS92] R.D. Koster and M.J. Suarez. Modeling the land surface boundary in climate models as a composite of independent vegetation stands. *J.-Geophys.-Res.*, 97:2697–2715, 1992.
- [LH74] A.A. Lacis and J.E. Hansen. A parameterization for the absorption of solar radiation in the earth’s atmosphere. *J.-Atmos.-Sci.*, 31:118–133, 1974.
- [LDDM97] W.G. Large, G. Danabasoglu, S.C. Doney, and J.C. McWilliams. Sensitivity to surface forcing and boundary layer mixing in a global ocean model: annual-mean climatology. *J.-Phys.-Oceanogr.*, 27(11):2418–2447, 1997.
- [LMD94] W.G. Large, J.C. McWilliams, and S.C. Doney. Oceanic vertical mixing: a review and a model with nonlocal boundary layer parameterization. *Rev.-Geophys.*, 32:363–403, 1994.
- [LP81] W.G. Large and S. Pond. Open ocean momentum flux measurements in moderate to strong winds. *J.-Phys.-Oceanogr.*, 11:324–336, 1981.
- [Lei68] C.E. Leith. Large eddy simulation of complex engineering and geophysical flows. *Physics of Fluids*, 10:1409–1416, 1968.
- [Lei96] C.E. Leith. Stochastic models of chaotic systems. *Physica D.*, 98:481–491, 1996.
- [LKT+12] J.-F. Lemieux, D. Knoll, B. Tremblay, D.M. Holland, and M. Losch. A comparison of the Jacobian-free Newton-Krylov method and the EVP model for solving the sea ice momentum equation with a viscous-plastic formulation: a serial algorithm study. *J.-Comput.-Phys.*, 231(17):5926–5944, 2012. doi:10.1016/j.jcp.2012.05.024.
- [LTSedlacek+10] J.-F. Lemieux, B. Tremblay, J. Sedláček, P. Tupper, S. Thomas, D. Huard, and J.-P. Auclair. Improving the numerical convergence of viscous-plastic sea ice models with the Jacobian-free Newton-Krylov method. *J.-Comput.-Phys.*, 229:2840–2852, 2010. doi:10.1016/j.jcp.2009.12.011c.
- [Lepparanta83] M. Leppäranta. A growth model for black ice, snow ican and snow thickness in subarctic basins. *Nordic Hydrology*, 14:59–70, 1983.
- [LFLV14] M. Losch, A. Fuchs, J.-F. Lemieux, and A. Vanselow. A parallel Jacobian-free Newton-Krylov solver for a coupled sea ice-ocean model. *J.-Comput.-Phys.*, 257(A):901–910, 2014. doi:10.1016/j.jcp.2013.09.026.
- [LMC+10] M. Losch, D. Menemenlis, J.-M. Campin, P. Heimbach, and C. Hill. On the formulation of sea-ice models. Part 1: effects of different solver implementations and parameterizations. *Ocean Modelling*, 33(1–2):129–144, 2010. doi:10.1016/j.ocemod.2009.12.008.
- [MGZ+99] J. Marotzke, R. Giering, K.Q. Zhang, D. Stammer, C. Hill, and T. Lee. Construction of the adjoint mit ocean general circulation model and application to atlantic heat transport variability. *J.-Geophys.-Res.*, 104, C12:29,529–29,547, 1999.
- [MAC+04] J. Marshall, A. Adcroft, J.-M. Campin, C. Hill, and A. White. Atmosphere-ocean modeling exploiting fluid isomorphisms. *Mon.-Wea.-Rev.*, 132:2882–2894, 2004. URL: http://mitgcm.org/pdfs/a_o_iso.pdf, doi:10.1175/MWR2835.1.
- [MAH+97] J. Marshall, A. Adcroft, C. Hill, L. Perelman, and C. Heisey. A finite-volume, incompressible navier stokes model for studies of the ocean on parallel computers. *J.-Geophys.-Res.*, 102(C3):5753–5766, 1997. URL: <http://mitgcm.org/pdfs/96JC02776.pdf>.

- [MHPA97] J. Marshall, C. Hill, L. Perelman, and A. Adcroft. Hydrostatic, quasi-hydrostatic, and nonhydrostatic ocean modeling. *J.~Geophys.~Res.*, 102(C3):5733–5752, 1997. URL: <http://mitgcm.org/pdfs/96JC02775.pdf>.
- [MJH98] J. Marshall, H. Jones, and C. Hill. Efficient ocean modeling using non-hydrostatic algorithms. *J.~Mar.~Sys.*, 18:115–134, 1998. URL: http://mitgcm.org/pdfs/journal_of_marine_systems_1998.pdf, doi:10.1016/S0924-7963%2898%2900008-6.
- [MJWF03] T. J. McDougall, D. R. Jackett, D. G. Wright, and R. Feistel. Accurate and computationally efficient algorithms for potential temperature and density of seawater. *J.~Atmos.~Ocean.~Technol.*, 5:730–741, 2003.
- [Mol09] A. Molod. Running GCM physics and dynamics on different grids: algorithm and tests. *Tellus*, 61A:381–393, 2009.
- [MS92] S. Moorthi and M.J. Suarez. Relaxed arakawa schubert: a parameterization of moist convection for general circulation models. *Mon.~Wea.~Rev.*, 120:978–1002, 1992.
- [Mou96] J.N. Moum. Energy-containing scales of turbulence in the ocean thermocline. *J.~Geophys.~Res.*, 101(C3):14095–14109, 1996.
- [Orl76] I. Orlanski. A simple boundary condition for unbounded hyperbolic flows. *J.~Comput.~Phys.*, 21:251–269, 1976.
- [PR97] T. Paluszkiwicz and R.D. Romea. A one-dimensional model for the parameterization of deep convection in the ocean. *Dyn.~Atmos.~Oceans*, 26:95–130, 1997.
- [Pan73] H.A. Panofsky. Tower micrometeorology. In D. A. Haugen, editor, *Workshop on Micrometeorology*. American Meteorological Society, 1973.
- [Pot73] D. Potter. *Computational Physics*. John Wiley, New York, 1973.
- [Red82] M.H. Redi. Oceanic Isopycnal Mixing by Coordinate Rotation. *J.~Phys.~Oceanogr.*, 12(10):1154–1158, oct 1982. doi:10.1175/1520-0485(1982)012<1154:OIMBCR>2.0.CO;2.
- [Roe85] P.L. Roe. Some contributions to the modelling of discontinuous flows. In B.E. Engquist, S. Osher, and R.C.J. Somerville, editors, *Large-Scale Computations in Fluid Mechanics*, volume 22 of *Lectures in Applied Mathematics*, pages 163–193. American Mathematical Society, Providence, RI, 1985.
- [RSG87] J.E. Rosenfield, M.R. Schoeberl, and M.A. Geller. A computation of the stratospheric diabatic circulation using an accurate radiative transfer model. *J.~Atmos.~Sci.*, 44:859–876, 1987.
- [SG94] H.E. Seim and M.C. Gregg. Detailed observations of a naturally occurring shear instability. *J.~Geophys.~Res.*, 99(C5):10049–10073, 1994.
- [Sem76] A.J. Semtner, Jr. A model for the thermodynamic growth of sea ice in numerical investigations of climate. *J.~Phys.~Oceanogr.*, 6:379–389, 1976.
- [Sha70] R. Shapiro. Smoothing, filtering, and boundary effects. *Rev.~Geophys.~Space~Phys.*, 8(2):359–387, 1970.
- [Sma63] J. Smagorinsky. General circulation experiments with the primitive equations i: the basic experiment. *Mon.~Wea.~Rev.*, 91(3):99–164, 1963.
- [Sma93] J. Smagorinsky. Large eddy simulation of complex engineering and geophysical flows. In B. Galperin and S.A. Orszag, editors, *Evolution of Physical Oceanography*, pages 3–36. Cambridge University Press, 1993.
- [Ste90] D.P. Stevens. On open boundary conditions for three dimensional primitive equation ocean circulation models. *Geophys. Astrophys. Fl. Dyn.*, 51:103–133, 1990.
- [Sto48] H. Stommel. The western intensification of wind-driven ocean currents. *Trans. Am. Geophys. Union*, 29:206, 1948.
- [SM88] Y.C. Sud and A. Molod. The roles of dry convection, cloud-radiation feedback processes and the influence of recent improvements in the parameterization of convection in the gla gcm. *Mon.~Wea.~Rev.*, 116:2366–2387, 1988.

- [TS96] L.L. Takacs and M.J. Suarez. Dynamical aspects of climate simulations using the geos general circulation model. NASA Technical Memorandum 104606 Volume 10, National Aeronautics and Space Administration, NASA; Goddard Space Flight Center; Greenbelt (MD), 20771; USA, 1996. <http://www.gmao.nasa.gov/>.
- [Tho77] S.A. Thorpe. Turbulence and mixing in a scottish loch. *Phil.~Trans.~R.~Soc.~Lond.*, 286:125–181, 1977.
- [VMHS97] M. Visbeck, J. Marshall, T. Haine, and M. Spall. Specification of eddy transfer coefficients in coarse-resolution ocean circulation models. *J.~Phys.~Oceanogr.*, 27(3):381–402, 1997.
- [Waj93] R. Wajsowicz. A consistent formulation of the anisotropic stress tensor for use in models of the large-scale ocean circulation. *J.~Comput.~Phys.*, 105(2):333–338, 1993.
- [WG94] J.C. Wesson and M.C. Gregg. Mixing at camarinal sill in the strait of gibraltar. *Q.~J.~R.~Meteorol.~Soc.*, 99 (C5):9847–9878, 1994.
- [WB95] A.A. White and R.A. Bromley. Dynamically consistent, quasi-hydrostatic equations for global models with a complete representation of the coriolis force. *J.~Geophys.~Res.*, 121:399–418, 1995.
- [Wil69] G.P. Williams. Numerical integration of the three-dimensional navier stokes equations for incompressible flow. *J.~Fluid Mech.*, 37:727–750, 1969.
- [Win00] M. Winton. A reformulated three-layer sea ice model. *J.~Atmos.~Ocean.~Technol.*, 17:525–531, 2000.
- [YK74] A.M. Yaglom and B.A. Kader. Heat and mass transfer between a rough wall and turbulent fluid flow at high reynolds and peclet numbers. *J.~Fluid Mech.*, 62:601–623, 1974.
- [Yam77] T. Yamada. A numerical experiment on pollutant dispersion in a horizontally-homogenous atmospheric boundary layer. *Atmos. Environ.*, 11:1015–1024, 1977.
- [ZH97] J. Zhang and W.D. Hibler, III. On an efficient numerical method for modeling sea ice dynamics. *J.~Geophys.~Res.*, 102(C4):8691–8702, 1997.
- [ZWDHSR98] J. Zhang, III W.D. Hibler, M. Steele, and D.A. Rothrock. Arctic ice-ocean modeling with and without climate restoring. *J.~Phys.~Oceanogr.*, 28:191–217, 1998.
- [ZSL95] J. Zhou, Y.C. Sud, and K.-M. Lau. Impact of orographically induced gravity wave drag in the gla gcm. *Q.~J.~R.~Meteorol.~Soc.*, 122:903–927, 1995.