MITgcm Documentation

Release 1.0

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

Sep 25, 2017

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 . . . 11
  - 1.2.9 Simulations of laboratory experiments . . . 14
- 1.3 Continuous equations in 'r' coordinates . . . 14
  - 1.3.1 Kinematic Boundary conditions . . . 18
  - 1.3.2 Atmosphere . . . 19
  - 1.3.3 Ocean . . . 20
  - 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 . . . 25
  - 1.3.8 Vector invariant form . . . 26
  - 1.3.9 Adjoint . . . 26
- 1.4 Appendix ATMOSPHERE . . . 27
  - 1.4.1 Hydrostatic Primitive Equations for the Atmosphere in Pressure Coordinates . . . 27
- 1.5 Appendix OCEAN . . . 29
  - 1.5.1 Equations of Motion for the Ocean . . . 29
- 1.6 Appendix OPERATORS . . . 33
  - 1.6.1 Coordinate systems . . . 33

2 Getting Started with MITgcm **35**
- 2.1 Where to find information . . . 35

3 Contributing to the MITgcm **37**
- 3.1 Bugs and feature requests . . . 37
- 3.2 Contributing to the code . . . 37
  - 3.2.1 Quickstart Guide . . . 38
  - 3.2.2 Detailed guide . . . 39

3.2.3 Style guide . . . 39
3.2.4 Automatic testing with Travis-CI . . . 39
3.3 Contributing to the manual . . . 39
3.3.1 Section headings . . . 40
3.3.2 Cross referencing . . . 40
3.3.3 Maths . . . 40
3.3.4 Units . . . 40
3.3.5 Describing subroutine inputs and outputs . . . 41
3.4 Reviewing pull requests . . . 41

4 MITgcm Example Experiments **43**
4.1 Full list of model examples . . . 43
4.2 Barotropic Gyre MITgcm Example . . . 43
4.2.1 Equations Solved . . . 44
4.2.2 Discrete Numerical Configuration . . . 44
4.2.3 Code Configuration . . . 45
4.3 A Rotating Tank in Cylindrical Coordinates . . . 50
4.3.1 Overview . . . 50
4.3.2 Equations Solved . . . 50
4.3.3 Discrete Numerical Configuration . . . 50
4.3.4 Code Configuration . . . 51

5 Physical Parameterizations - Packages I **57**
5.1 Overview . . . 57
5.1.1 Using MITgcm Packages . . . 57
5.2 Packages Related to Hydrodynamical Kernel . . . 62
5.2.1 Generic Advection/Diffusion . . . 62
5.2.2 Shapiro Filter . . . 63
5.2.3 FFT Filtering Code . . . 64
5.2.4 exch2: Extended Cubed Sphere Topology . . . 64
5.2.5 Gridalt - Alternate Grid Package . . . 71
5.3 General purpose numerical infrastructure packages . . . 75
5.3.1 OBCS: Open boundary conditions for regional modeling . . . 75
5.3.2 RBCS Package . . . 82
5.3.3 PTRACERS Package . . . 84
5.4 Ocean Packages . . . 87
5.4.1 GMREDI: Gent-McWilliams/Redi SGS Eddy Parameterization . . . 87
5.4.2 KPP: Nonlocal K-Profile Parameterization for Vertical Mixing . . . 94
5.4.3 GGL90: a TKE vertical mixing scheme . . . 100
5.4.4 OPPS: Ocean Penetrative Plume Scheme . . . 100
5.4.5 KL10: Vertical Mixing Due to Breaking Internal Waves . . . 100
5.4.6 BULK_FORCE: Bulk Formula Package . . . 103
5.4.7 EXF: The external forcing package . . . 106
5.4.8 CAL: The calendar package . . . 115
5.5 Atmosphere Packages . . . 119
5.5.1 Atmospheric Intermediate Physics: AIM . . . 119
5.5.2 Land package . . . 121
5.5.3 Fizhi: High-end Atmospheric Physics . . . 122
5.6 Sea Ice Packages . . . 159
5.6.1 THSICE: The Thermodynamic Sea Ice Package . . . 159
5.6.2 SEAICE Package . . . 165

Bibliography **181**

CHAPTER 1

Overview

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.

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)):

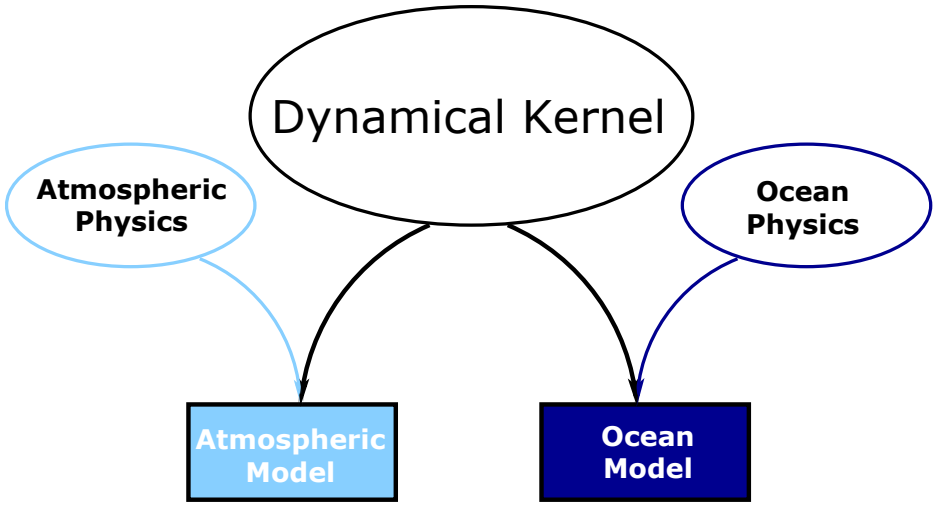

Figure 1.1: MITgcm has a single dynamical kernel that can drive forward either oceanic or atmospheric simulations.

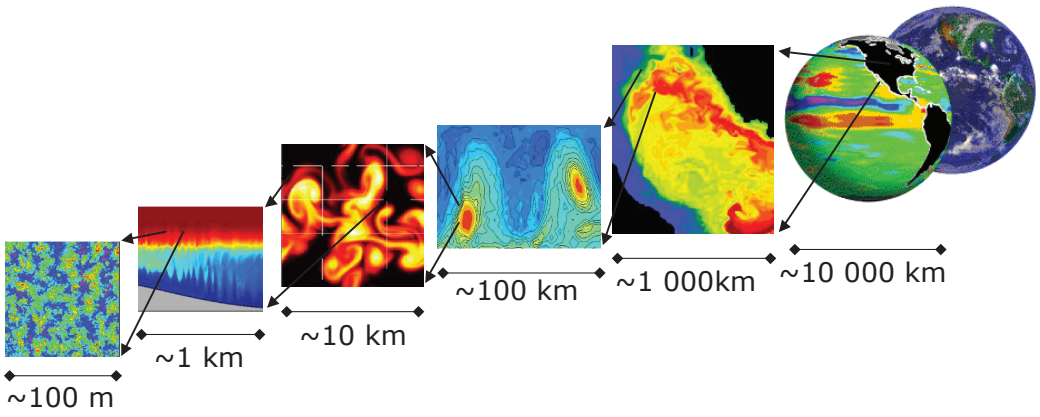

Figure 1.2: MITgcm has non-hydrostatic capabilities, allowing the model to address a wide range of phenomenon - from convection on the left, all the way through to global circulation patterns on the right.

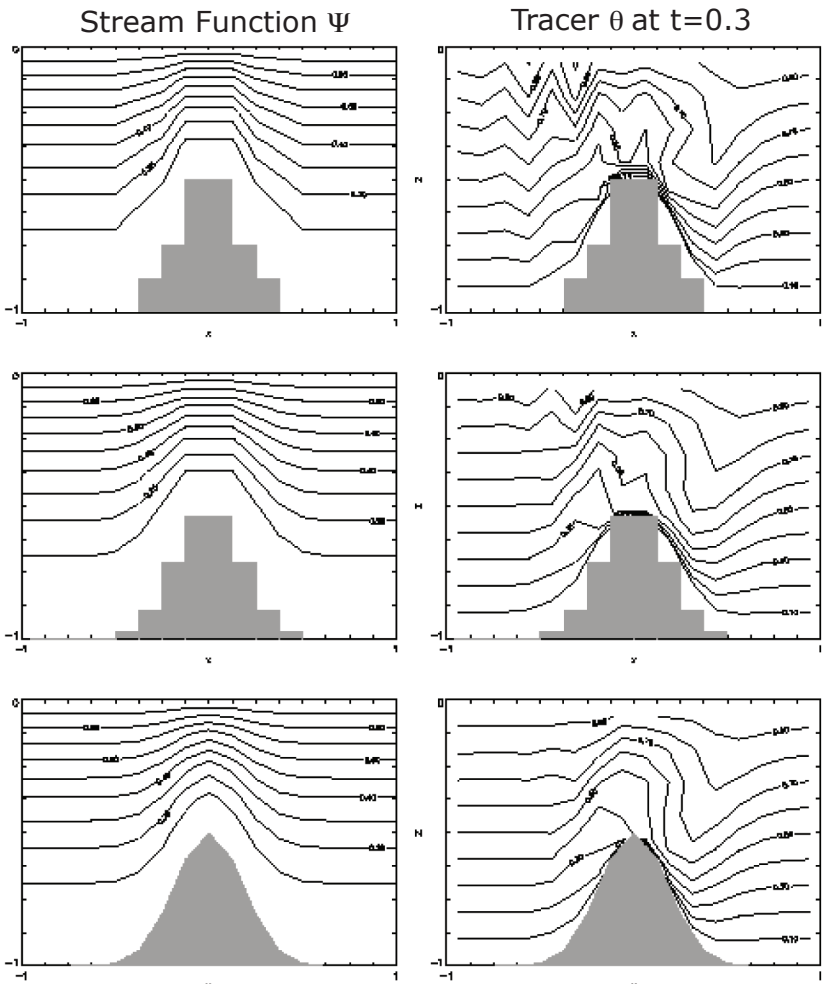

Figure 1.3: Finite volume techniques (bottom panel) are used, permitting a treatment of topography that rivals σ (terrain following) coordinates.

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.

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.

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.

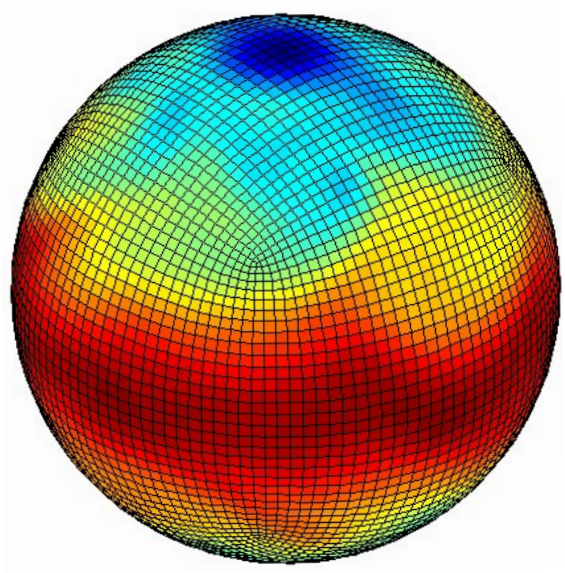

Figure 1.4: Instantaneous plot of the temperature field at 500 mb obtained using the atmospheric isomorph of MITgcm

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

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

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.

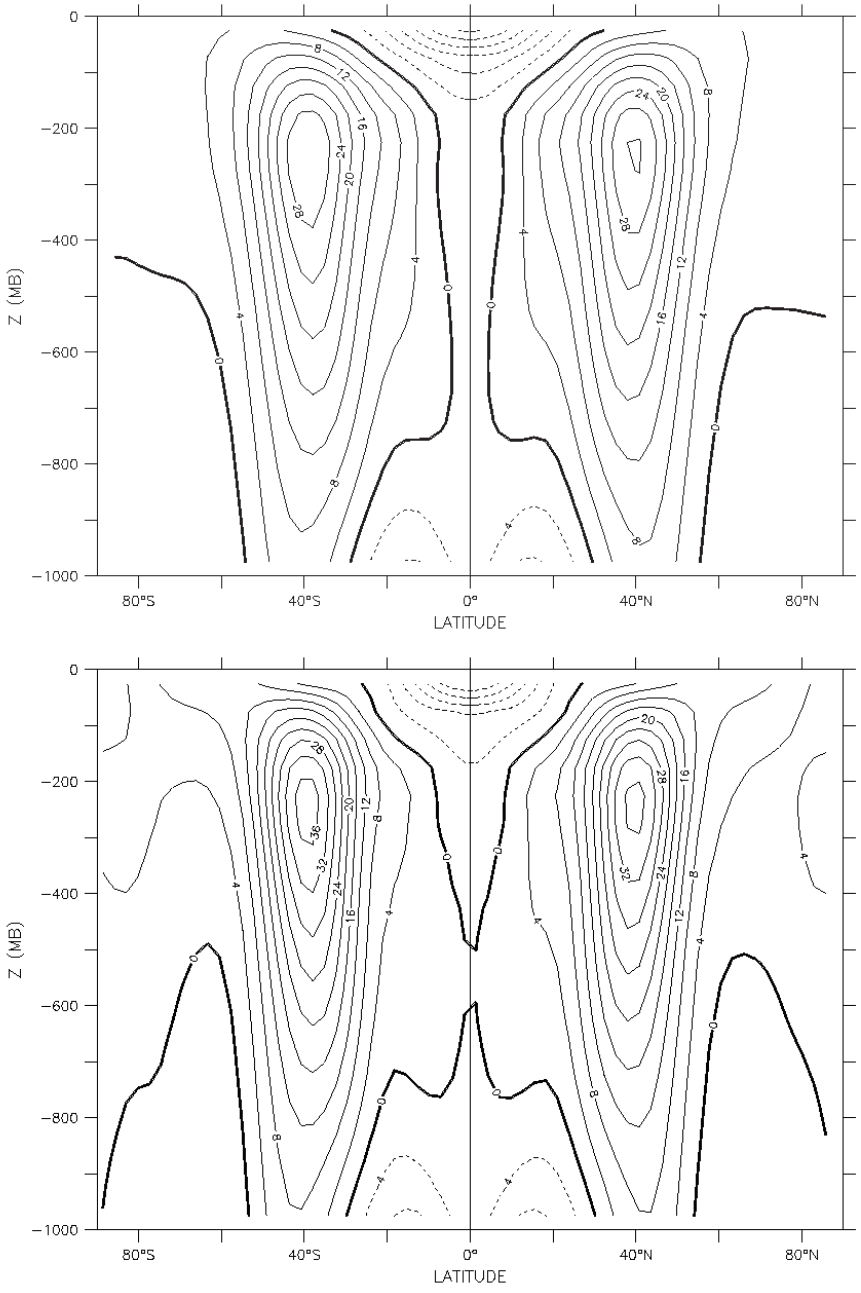

Figure 1.5: Five year mean, zonally averaged zonal flow for cube-sphere simulation (top) and latitude-longitude simulation (bottom) and using Held-Suarez forcing. Note the difference in the solutions over the pole — the cubed sphere is superior.

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.

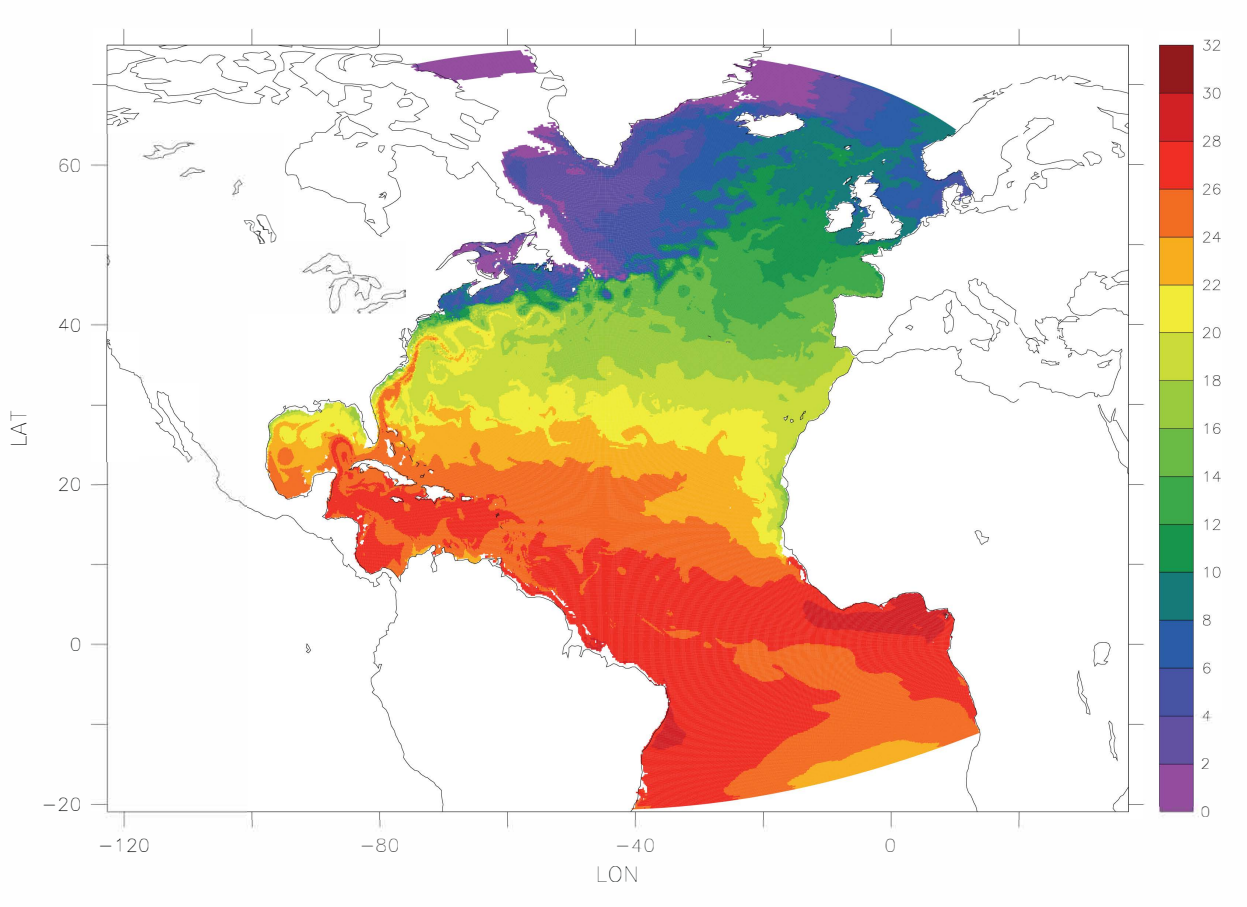

Figure 1.6: Instantaneous temperature map from a $\frac{1}{6}^\circ$ simulation of the North Atlantic. The figure shows the temperature in the second layer (37.5 m deep).

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.

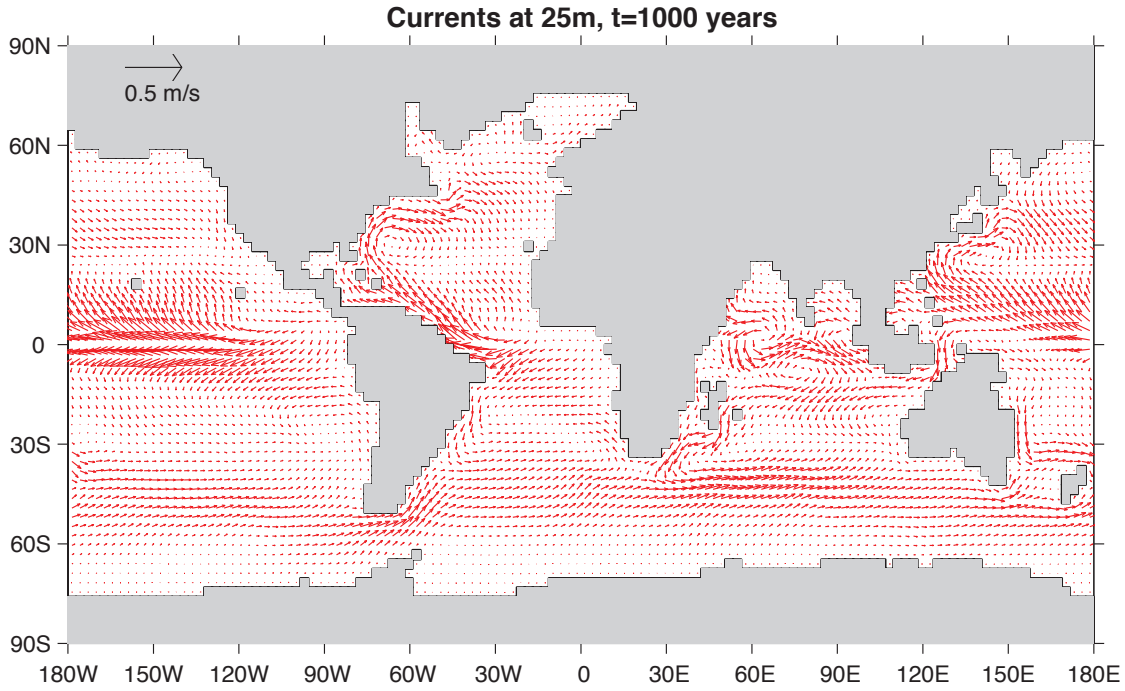

Figure 1.7: Pattern of surface ocean currents from a global integration of the model at 4° horizontal resolution and with 15 vertical levels.

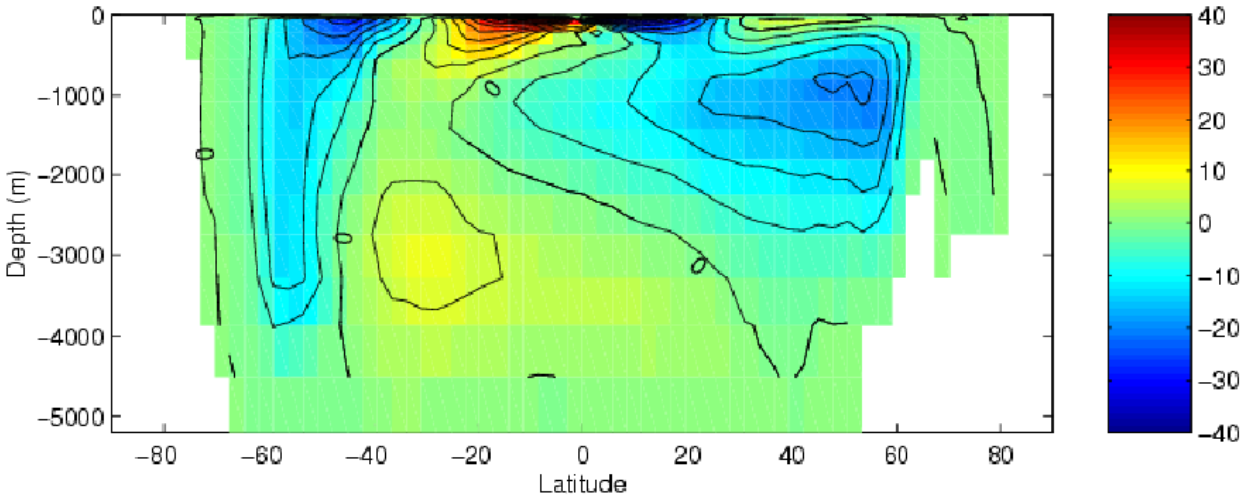

Figure 1.8: Meridional overturning stream function (in Sverdrups) from a global integration of the model at 4° horizontal resolution and with 15 vertical levels.

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.

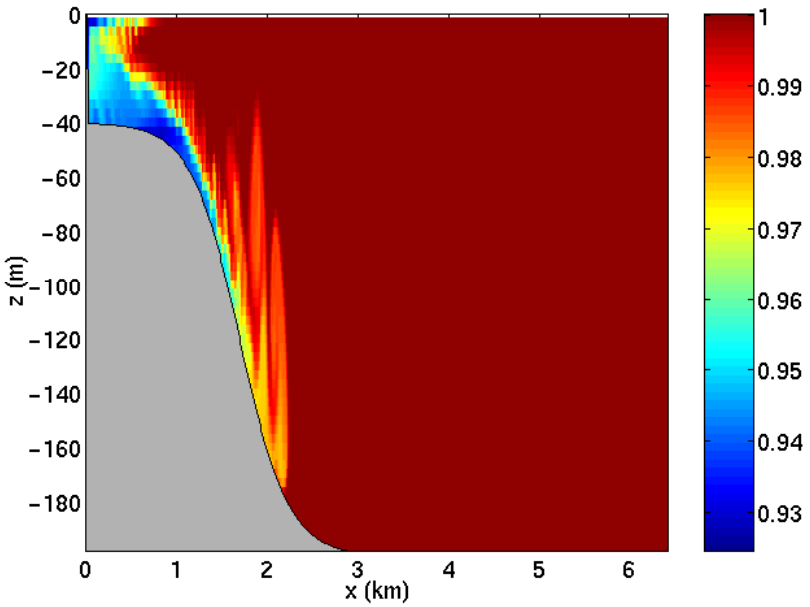

Figure 1.9: MITgcm run in a non-hydrostatic configuration to study convection over a slope.

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.

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.

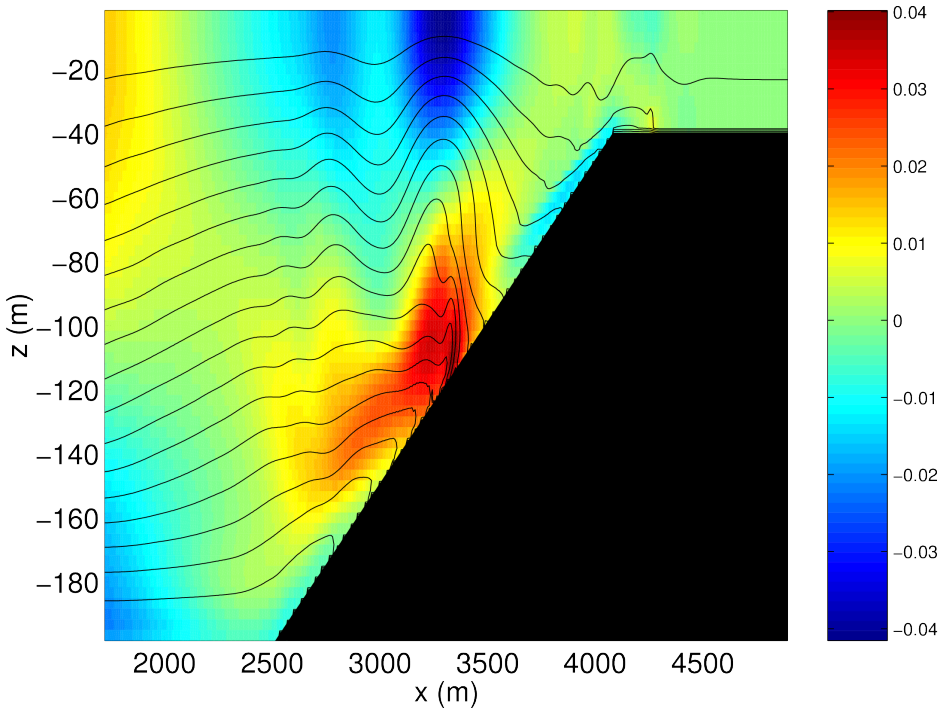

Figure 1.10: Simulation of internal waves forced at an open boundary (on the left) impacting a sloping shelf. The along slope velocity is shown colored, contour lines show density surfaces. The slope is represented with high-fidelity using lopped cells.

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.

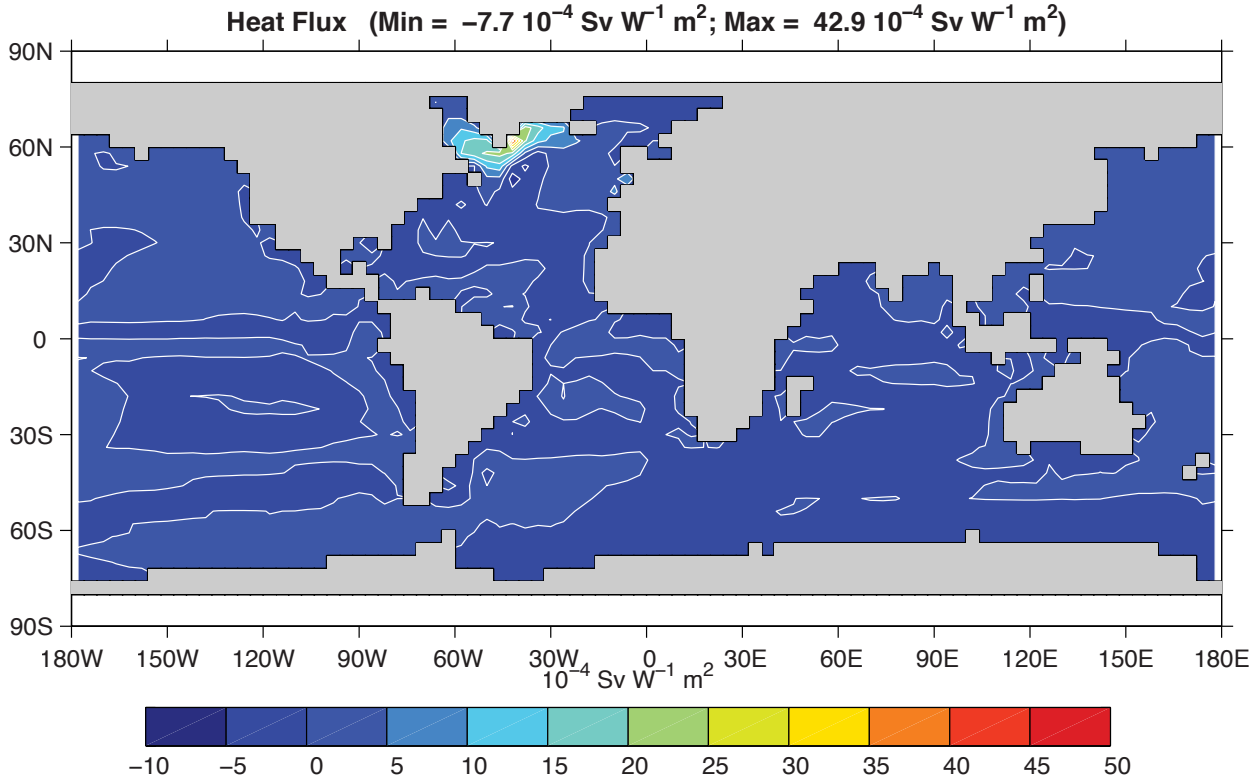

Figure 1.11: Sensitivity of meridional overturning strength to surface heat flux changes. Contours show the magnitude of the response (in Sv x 10^{-4}) that a persistent +1 Wm^{-2} heat flux anomaly at a given grid point would produce.

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.

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

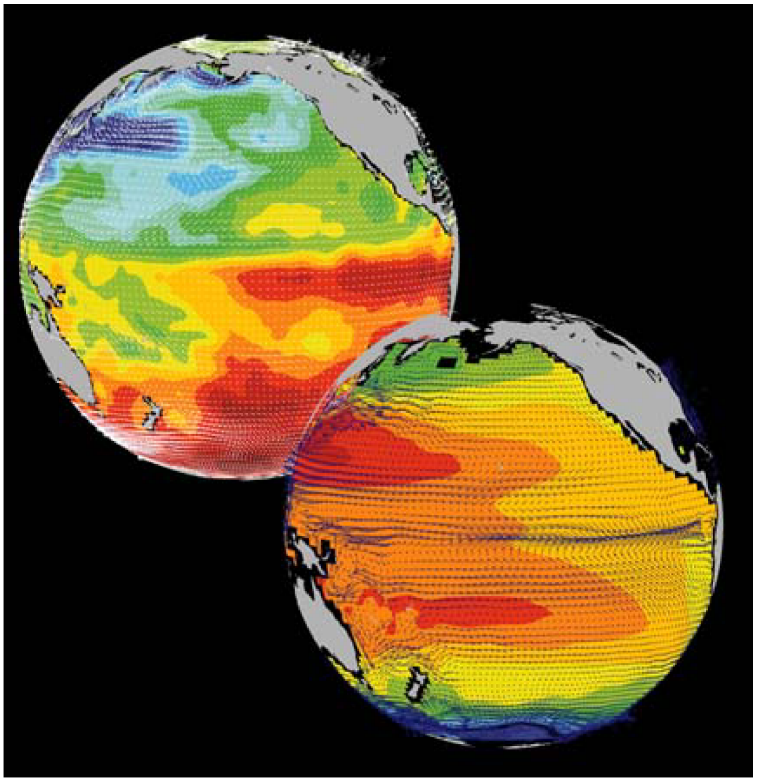

Figure 1.12: Circulation patterns from a multi-year, global circulation simulation constrained by Topex altimeter data and WOCE cruise observations. This output is from a higher resolution, shorter duration experiment with equatorially enhanced grid spacing.

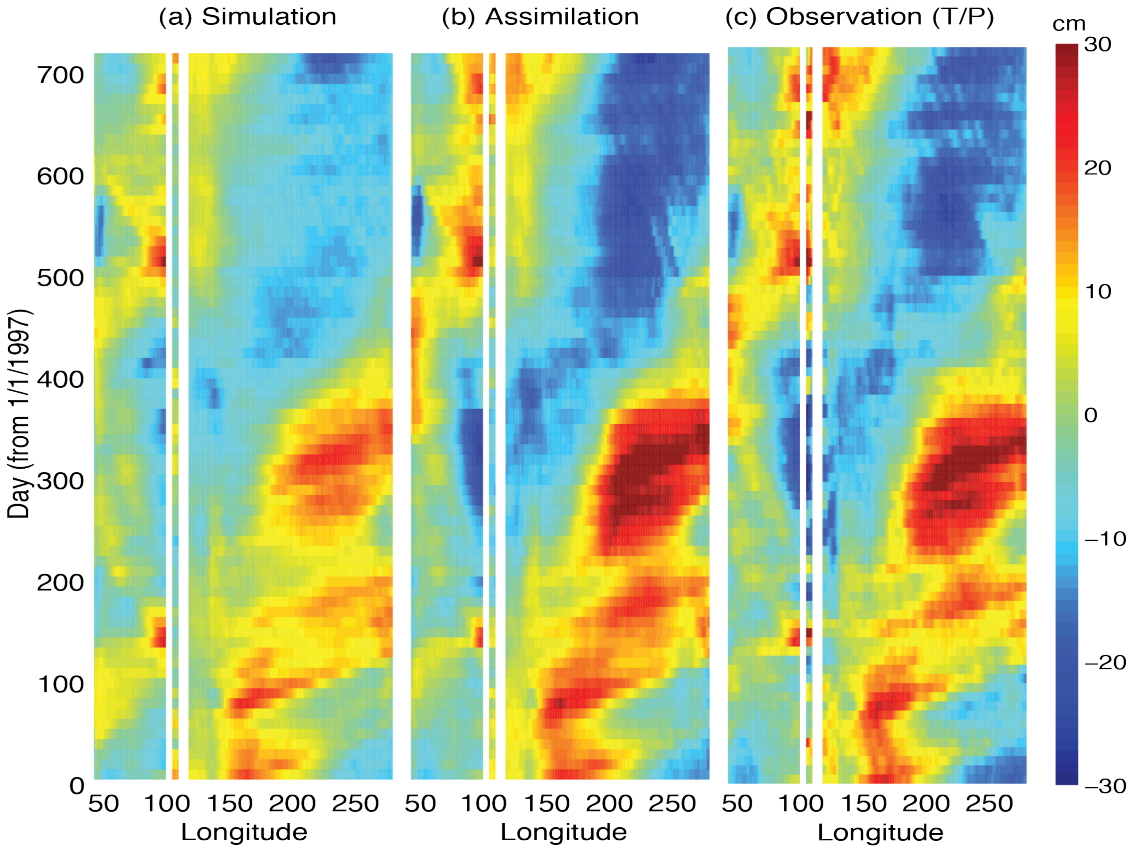

Figure 1.13: Equatorial sea-surface height in unconstrained (left), constrained (middle) simulations and in observations (right).

outcrops in the southern oceans from a single year of a global, interannually varying simulation. The simulation is run at 1°x1° resolution telescoping to $\frac{1}{3}^\circ$ x $\frac{1}{3}^\circ$ in the tropics (not shown).

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.

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 $\vec{\mathbf{v}}$, 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.

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

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

$$\nabla_h \cdot \vec{\mathbf{v}}_h + \frac{\partial \dot{r}}{\partial r} = 0 \text{ continuity} \tag{1.3}$$

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

$$\frac{D\theta}{Dt} = \mathcal{Q}_\theta \text{ potential temperature} \tag{1.5}$$

$$\frac{DS}{Dt} = \mathcal{Q}_S \text{ humidity/salinity} \tag{1.6}$$

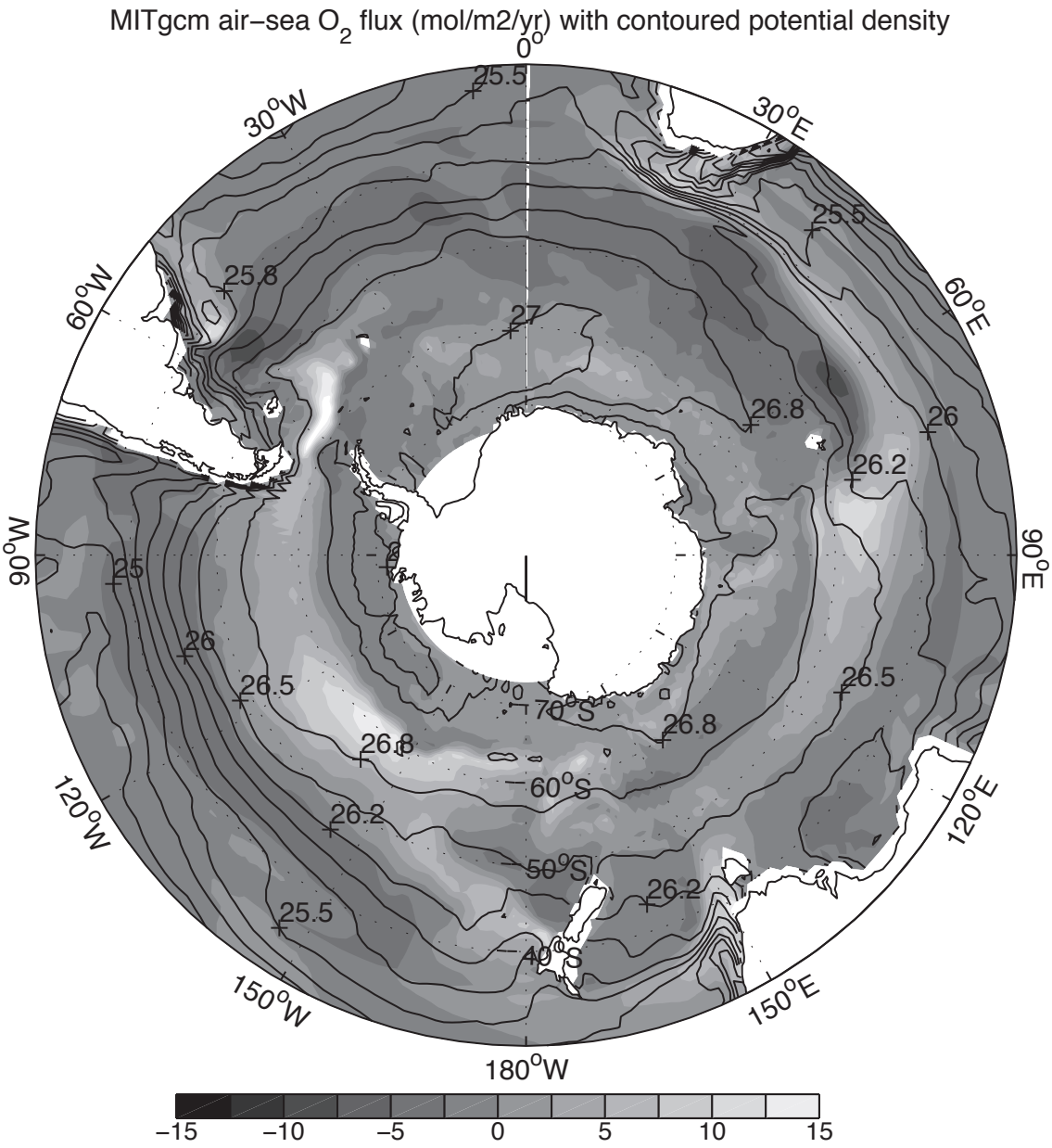

Figure 1.14: Annual air-sea flux of oxygen (shaded) plotted along with potential density outcrops of the surface of the southern ocean from a global 1°x1° integration with a telescoping grid (to $\frac{1}{3}^\circ$) at the equator.

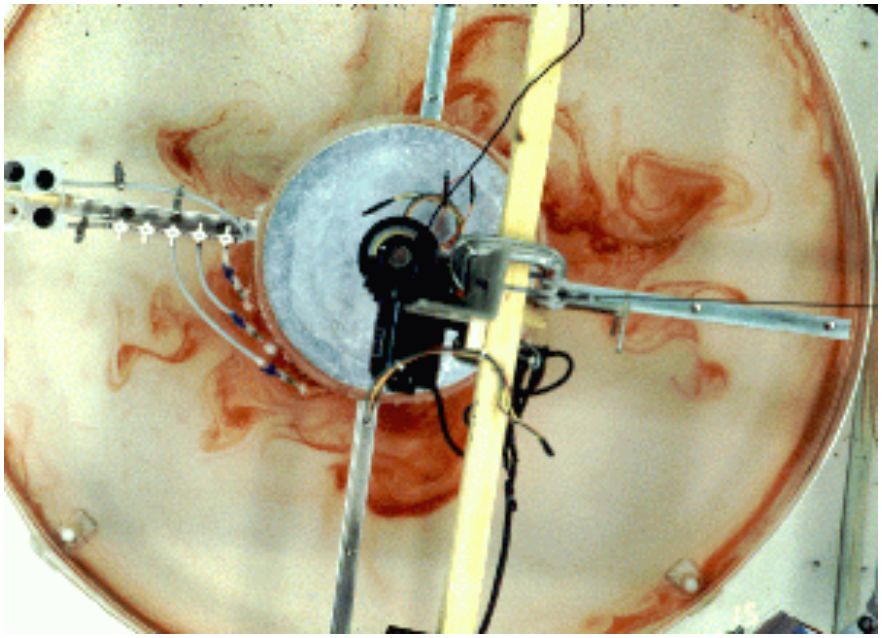

Figure 1.15: A 1 m diameter laboratory experiment simulating the dynamics of the Antarctic Circumpolar Current.

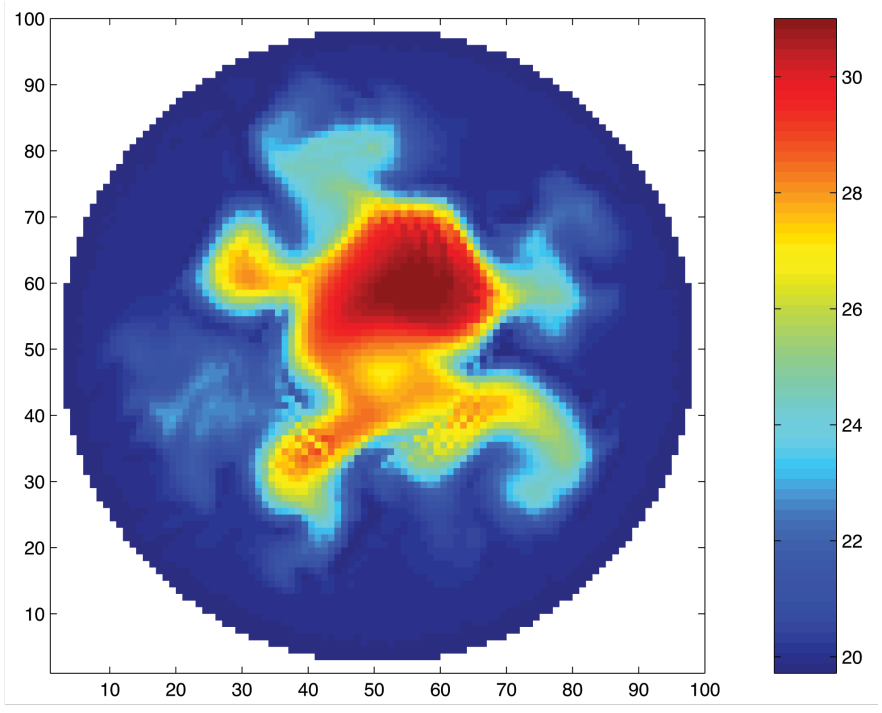

Figure 1.16: A numerical simulation of the laboratory experiment using MITgcm.

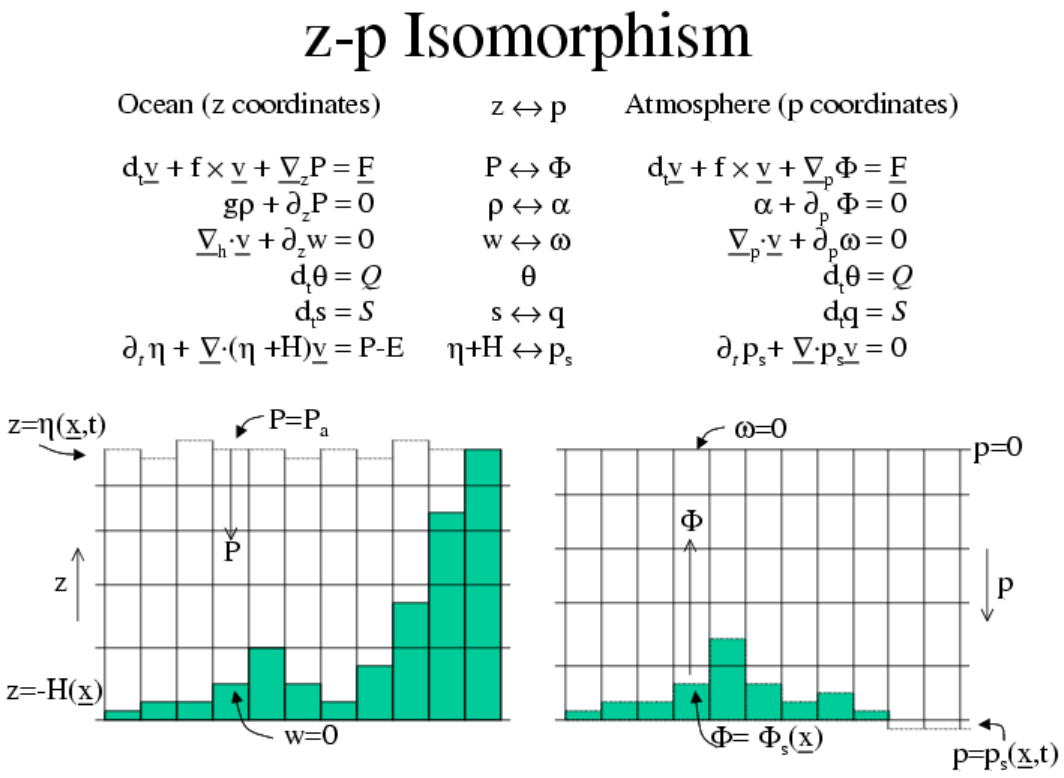

Figure 1.17: Isomorphic equation sets used for atmosphere (right) and ocean (left).

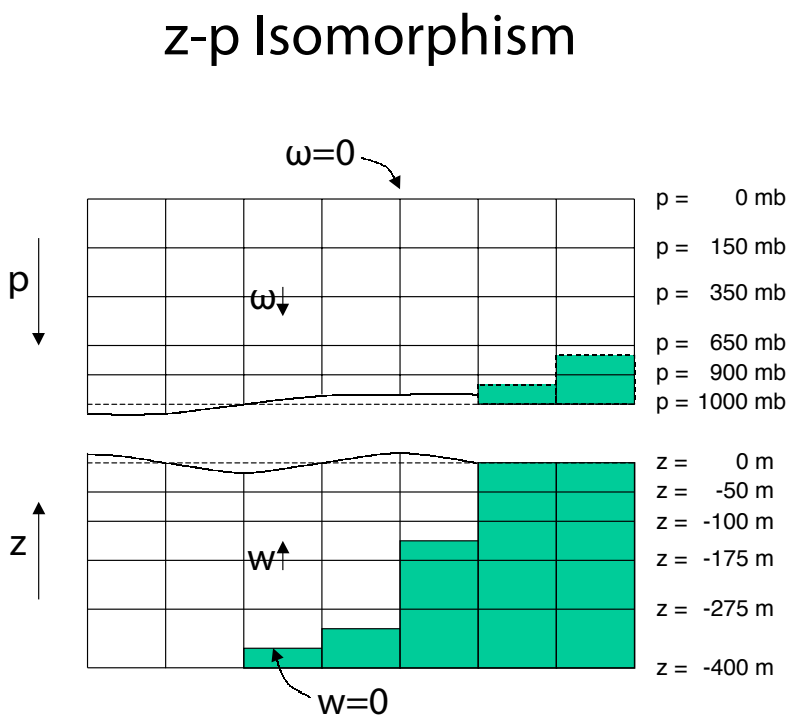

Figure 1.18: Vertical coordinates and kinematic boundary conditions for atmosphere (top) and ocean (bottom).

Here:

$$r \text{ is the vertical coordinate}$$

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

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

with ∇_h operating in the horizontal and $\widehat{k}\frac{\partial}{\partial r}$ operating in the vertical, where $\widehat{k}$ is a unit vector in the vertical

$$t \text{ is time}$$

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

$$\phi \text{ is the 'pressure'/'geopotential'}$$

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

$$b \text{ is the 'buoyancy'}$$

$$\theta \text{ is potential temperature}$$

$$S \text{ is specific humidity in the atmosphere; salinity in the ocean}$$

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

$$\mathcal{Q}_\theta \text{ are forcing and dissipation of } \theta$$

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

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.

Kinematic Boundary conditions

Vertical

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

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

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

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.

Horizontal

$$\vec{\mathbf{v}} \cdot \vec{\mathbf{n}} = 0 \tag{1.9}$$

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

Atmosphere

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

$$r = p \text{ is the pressure} \tag{1.10}$$

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

$$\phi = g\,z \text{ is the geopotential height} \tag{1.12}$$

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

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

$$S = q \text{ is the specific humidity} \tag{1.15}$$

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)

$$\Pi(p) = c_p(\frac{p}{p_c})^{\kappa} \tag{1.16}$$

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:

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

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

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

Ocean

In the ocean we interpret:

$$r = z \text{ is the height} \tag{1.19}$$

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

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

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

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:

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

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

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).

Hydrostatic, Quasi-hydrostatic, Quasi-nonhydrostatic and Non-hydrostatic forms

Let us separate ϕ in to surface, hydrostatic and non-hydrostatic terms:

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

and write (1.1) in the form:

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

$$\frac{\partial \phi_{hyd}}{\partial r} = -b \tag{1.27}$$

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

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

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

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

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

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

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*.

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.

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: only the terms underlined twice in Eqs. (1.29) $\rightarrow$ (1.31) are set to zero and, simultaneously, the shallow atmosphere

[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.

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.

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) $\rightarrow$ (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:

$$\dot{r} = \frac{Dp}{Dt} = \frac{1}{g}\frac{D\phi}{Dt} \tag{1.32}$$

where p_{hy} is the hydrostatic pressure.

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).

Solution strategy

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

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

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).

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

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

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

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

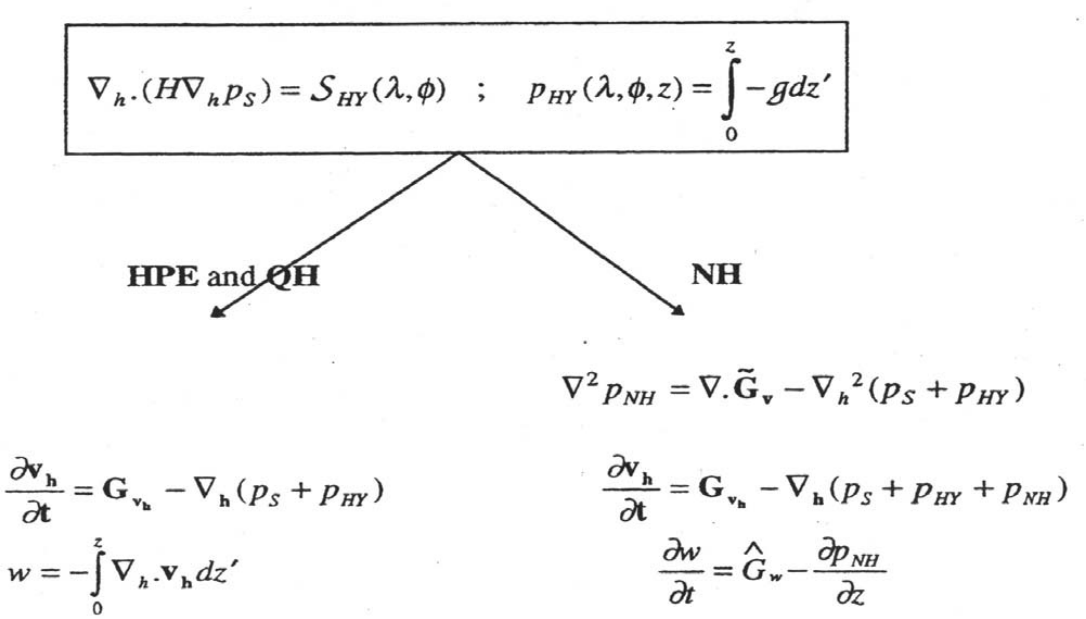

Figure 1.19: Basic solution strategy in MITgcm. **HPE** and **QH** forms diagnose the vertical velocity, in **NH** a prognostic equation for the vertical velocity is integrated.

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:

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

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:

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

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

$$\nabla_h \phi_s = \nabla_h (b_s \eta) \tag{1.36}$$

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.

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:

$$\nabla_3^2 \phi_{nh} = \nabla.\vec{\mathbf{G}}_{\vec{v}} - \left(\nabla_h^2 \phi_s + \nabla^2 \phi_{hyd}\right) = \nabla.\vec{\mathbf{F}} \tag{1.37}$$

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:

$$\vec{\mathbf{v}}.\widehat{n} = 0 \tag{1.38}$$

where $\widehat{n}$ 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:

$$\widehat{n}.\nabla\phi_{nh} = \widehat{n}.\vec{\mathbf{F}} \tag{1.39}$$

where

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

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 $\vec{\mathbf{F}}$. By simultaneously setting $\widehat{n}.\vec{\mathbf{F}} = 0$ and $\widehat{n}.\nabla\phi_{nh} = 0$ on the boundary the following self-consistent but simpler homogenized Elliptic problem is obtained:

$$\nabla^2 \phi_{nh} = \nabla.\widetilde{\vec{\mathbf{F}}}$$

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

$$\widehat{n}.\nabla\phi_{nh} = 0 \tag{1.40}$$

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.

Forcing/dissipation

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.

Dissipation

Momentum

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

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

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.

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

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:

$$K = \begin{pmatrix} K_h & 0 & 0 \\ 0 & K_h & 0 \\ 0 & 0 & K_v \end{pmatrix} \tag{1.43}$$

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

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:

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

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.

Adjoint

Tangent linear and adjoint counterparts of the forward model are described in Chapter 5.

Appendix ATMOSPHERE

Hydrostatic Primitive Equations for the Atmosphere in Pressure Coordinates

The hydrostatic primitive equations (**HPE**'s) in $p-$coordinates are:

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

$$\frac{\partial \phi}{\partial p} + \alpha = 0 \tag{1.46}$$

$$\nabla_p \cdot \vec{\mathbf{v}}_h + \frac{\partial \omega}{\partial p} = 0 \tag{1.47}$$

$$p\alpha = RT \tag{1.48}$$

$$c_v \frac{DT}{Dt} + p\frac{D\alpha}{Dt} = \mathcal{Q} \tag{1.49}$$

where $\vec{\mathbf{v}}_h = (u, v, 0)$ is the 'horizontal' (on pressure surfaces) component of velocity, $\frac{D}{Dt} = \frac{\partial}{\partial t} + \vec{\mathbf{v}}_h \cdot \nabla_p + \omega \frac{\partial}{\partial p}$ 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:

$$c_p \frac{DT}{Dt} - \alpha\frac{Dp}{Dt} = \mathcal{Q} \tag{1.50}$$

Potential temperature is defined:

$$\theta = T(\frac{p_c}{p})^\kappa \tag{1.51}$$

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:

$$\Pi(p) = c_p(\frac{p}{p_c})^\kappa \tag{1.52}$$

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

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

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:

$$\Pi \frac{D\theta}{Dt} = \mathcal{Q} \tag{1.53}$$

which is in conservative form.

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

Boundary conditions

The upper and lower boundary conditions are:

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

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

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 $\omega \neq 0$).

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:

$$\theta = \theta_o + \theta' \tag{1.56}$$

$$\alpha = \alpha_o + \alpha' \tag{1.57}$$

$$\phi = \phi_o + \phi' \tag{1.58}$$

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:

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

$$\frac{\partial \phi'}{\partial p} + \alpha' = 0 \tag{1.60}$$

$$\nabla_p \cdot \vec{\mathbf{v}}_h + \frac{\partial \omega}{\partial p} = 0 \tag{1.61}$$

$$\frac{\partial \Pi}{\partial p}\theta' = \alpha' \tag{1.62}$$

$$\frac{D\theta}{Dt} = \frac{\mathcal{Q}}{\Pi} \tag{1.63}$$

Appendix OCEAN

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:

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

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

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

$$\rho = \rho(\theta, S, p) \tag{1.67}$$

$$\frac{D\theta}{Dt} = \mathcal{Q}_\theta \tag{1.68}$$

$$\frac{DS}{Dt} = \mathcal{Q}_s \tag{1.69}$$

These equations permit acoustics modes, inertia-gravity waves, non-hydrostatic motions, a geostrophic (Rossby) mode and a thermohaline mode. As written, they cannot be integrated forward consistently - if we step 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:

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

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

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

where we have used an approximation sign to indicate that we have assumed adiabatic motion, dropping the $\frac{D\theta}{Dt}$ and $\frac{DS}{Dt}$. Replacing (1.66) with (1.71) yields a system that can be explicitly integrated forward:

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

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

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

$$\rho = \rho(\theta, S, p) \tag{1.75}$$

$$\frac{D\theta}{Dt} = \mathcal{Q}_\theta \tag{1.76}$$

$$\frac{DS}{Dt} = \mathcal{Q}_s \tag{1.77}$$

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:

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

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

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

$$\rho = \rho(\theta, S, p) \tag{1.81}$$

$$\frac{D\theta}{Dt} = \mathcal{Q}_\theta \tag{1.82}$$

$$\frac{DS}{Dt} = \mathcal{Q}_s \tag{1.83}$$

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.

'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:

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

If the time- and space-scales of the motions of interest are longer than those of acoustic modes, then $\frac{Dp'}{Dt} << (\frac{Dp_o}{Dt}, \nabla \cdot \vec{\mathbf{v}}_h)$ in the continuity equations and $\left.\frac{\partial \rho}{\partial p}\right|_{\theta,S} \frac{Dp'}{Dt} << \left.\frac{\partial \rho}{\partial p}\right|_{\theta,S} \frac{Dp_o}{Dt}$ 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:

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

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

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

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:

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

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

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

$$\rho = \rho(\theta, S, p_o(z)) \tag{1.89}$$

$$\frac{D\theta}{Dt} = \mathcal{Q}_\theta \tag{1.90}$$

$$\frac{DS}{Dt} = \mathcal{Q}_s \tag{1.91}$$

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:

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

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

$$\nabla_z \cdot \vec{\mathbf{v}}_h + \frac{\partial w}{\partial z} = 0 \tag{1.94}$$

$$\rho = \rho(\theta, S) \tag{1.95}$$

$$\frac{D\theta}{Dt} = \mathcal{Q}_\theta \tag{1.96}$$

$$\frac{DS}{Dt} = \mathcal{Q}_s \tag{1.97}$$

where ρ_c is a constant reference density of water.

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:

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

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

$$\nabla_z \cdot \vec{\mathbf{v}}_h + \frac{\partial w}{\partial z} = 0 \tag{1.100}$$

$$\rho' = \rho(\theta, S, p_o(z)) - \rho_c \tag{1.101}$$

$$\frac{D\theta}{Dt} = \mathcal{Q}_\theta \tag{1.102}$$

$$\frac{DS}{Dt} = \mathcal{Q}_s \tag{1.103}$$

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).

Appendix OPERATORS

Coordinate systems

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:

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

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

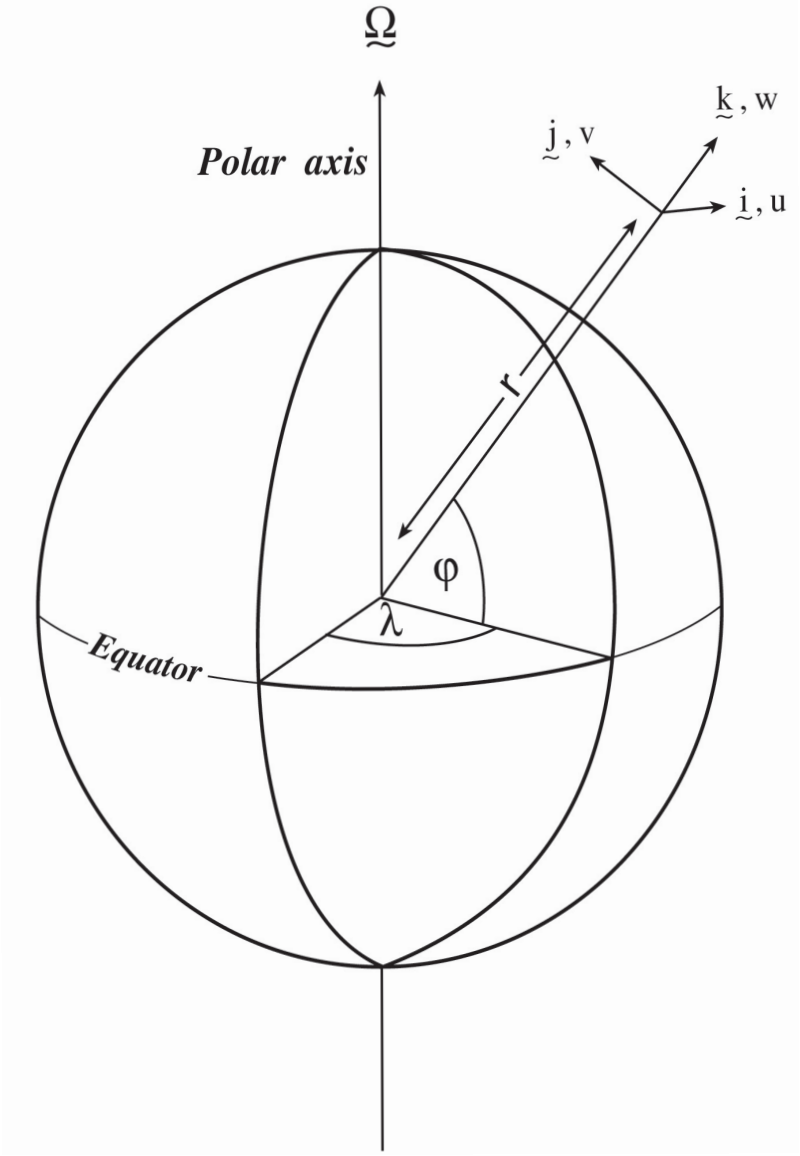

Figure 1.20: Spherical polar coordinates: longitude λ, latitude ϕ and r the distance from the center.

CHAPTER 2

Getting Started with MITgcm

This chapter is divided into two main parts. The first part, which is covered in sections ref{sec:whereToFindInfo} through ref{sec:testing}, contains information about how to run experiments using MITgcm. The second part, covered in sections ref{sec:eg-baro} through ref{sec:eg-offline}, 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 with the base version. Information on how to obtain, compile, and run the code is found here as well as a brief description of the model structure directory and the case study examples. 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 chapters ref{chap:discretization} and ref{chap:sarch}.

Where to find information

CHAPTER 3

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.

Bugs and feature requests

If you think you've found a bug, the best 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 [url to be inserted once we have the proper repo]. 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.

Contributing to the code

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 are expected to conform with the *Style guide*.

Quickstart Guide

You will need a GitHub account, but that's pretty much it!

1: Fork the project and create a local clone (copy)

> You can fork by clicking the button, and create a clone either also by using the button, or in a terminal: `git clone https://github.com/user_name/MITgcm66h.git` (substitute your own user name on github)
>
> move into the new directory: `cd MITgcm66h`
>
> Finally, we need to set up a remote that points to the original project: `git remote add upstream https://github.com/altMITgcm/MITgcm66h.git`
>
> This means that we have two "remotes" of the project, one pointing to your space (origin), and one pointing to the original (upstream). You can read and write into your "origin" version, but not into the "upstream" version.

2: Doing stuff! This usually comes in two flavours; Fixing bugs or adding a feature. Here we will assume we are fixing a bug and branch from the master, but if adding a new feature branching from develop is usually the way it works.

> To fix this "bug" we check out the master branch, and make sure we're up to date. `git checkout master` `git pull upstream master && git push origin master`
>
> Next make a new branch. Naming it something useful helps. `git checkout -b bugfix/contributingHowTo`
>
> Do the work! Be sure to include useful and detailed commit messages. To do this you should:
>
> - edit the relevant file(s)
> - use `git add FILENAME` to stage the file(s) ready for a commit command
> - use `git commit` to commit the files
> - type a succint (<70 character) summary of what the commit did
> - leave a blank line and type a longer description of why the action in this commit was appropriate
> - it is good practice to link with issues using the syntax `#ISSUE_NUMBER` in one or both of the above.

3: Now we push our branch into the origin remote.

```
git push -u origin bugfix/contributingHowTo
```

4: Then create a pull request (PR). In a browser, go to the fork of the project that you made. There is a button for "Compare and Pull" in your recent branches. Click the button! Now you can add a nice and succinct description of what you've done and flag up any issues.

5: Review by the maintainers!

To sum up from https://akrabat.com/the-beginners-guide-to-contributing-to-a-github-project/

The fundamentals are:

1. Fork the project & clone locally.
2. Create an upstream remote and sync your local copy before you branch.
3. Branch for each separate piece of work.
4. Do the work, write good commit messages, and read the guidelines in the manual.
5. Push to your origin repository.

6. Create a new PR in GitHub.
7. Respond to any code review feedback.

Detailed guide

To be completed.

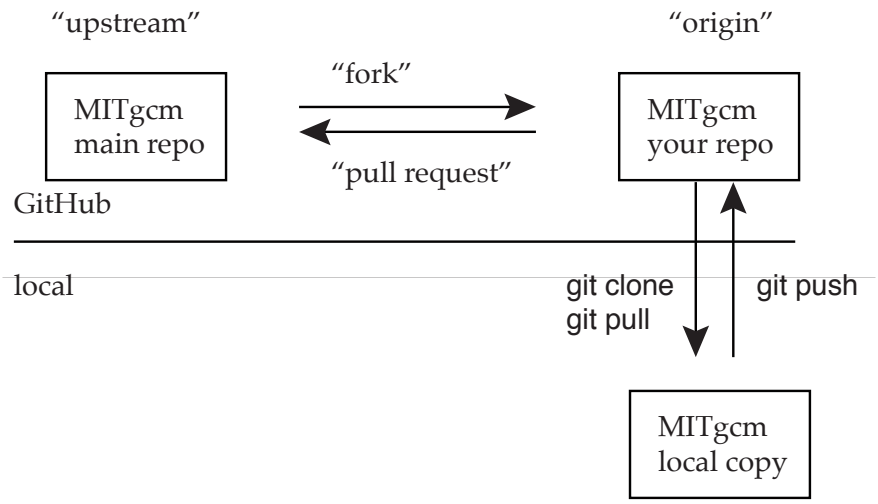

Figure 3.1: A conceptual map of the GitHub setup. Text in serif font are labels or concepts, text in sans serif represent commands.

Style guide

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.

Contributing to the manual

Whether you are correcting typos or describing currently 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)

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.

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.

Cross referencing

Labels go above the section they refer to, with the format `.. _LABELNAME:`. The leading underscore is important.

To reference sections/figures/tables/equations by number use this format for the reference: `:numref:`sec_eg_baro``

To reference sections by name use this format: `:ref:`sec_eg_baro``

Maths

Inline maths is done with `:math:`LATEX_HERE``

Separate equations, which will be typeset on their own lines, are produced with:

```
.. math::
    :label: eqn_label_here

    LATEX_HERE
```

Units

Units should be typeset in normal text, and exponents added with the `:sup:` command.

```
100 N m\ :sup:`--2`
```

If the exponent is negative use two dashes `--` to make the minus sign long enough. The backslash removes the space between the unit and the exponent.

Describing subroutine inputs and outputs

This information should go in an 'adominition' block. The source code to achieve this is:

```
.. admonition:: Subroutine
  :class: note

  S/R GMREDI_CALC_TENSOR (*pkg/gmredi/gmredi_calc_tensor.F*)

  :math:`\sigma_x`: **SlopeX** (argument on entry)

  :math:`\sigma_y`: **SlopeY** (argument on entry)

  :math:`\sigma_z`: **SlopeY** (argument)

  :math:`S_x`: **SlopeX** (argument on exit)

  :math:`S_y`: **SlopeY** (argument on exit)
```

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:

- add the repository of the user proposing the pull request as a remote, `git remote add USERNAME https://github.com/USERNAME/MITgcm66h.git` where USERNAME is replaced by the user name of the person who has made the pull request;
- download a local version of the branch from the pull request, `git fetch USERNAME` followed by `git checkout --track USERNAME/foo`;
- run tests locally; and
- possibly push fixes or changes directly to the pull request.

None of these steps, apart from the final one, require write access to the main repository. This means that anyone can review pull requests. However, unless you are one of the core developers you won't be able to directly push changes. You will instead have to make a comment describing any problems you find.

CHAPTER 4

MITgcm 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 4.2 - `sec_eg_tank`. The other examples follow the same general structure as the tutorial examples. However, they only include brief instructions in a text file called {it README}. The examples are located in subdirectories under the directory texttt{verification}. Each example is briefly described below.

Full list of model examples

Once you have chosen the example you want to run, you are ready to compile the code.

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 files for this experiment can be found in the verification directory *verification/tutorial_barotropic_gyre*. 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 *[HL5a]*.

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

$$f(y) = f_0 + \beta y \tag{4.1}$$

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 (**??**) and $\beta = 10^{-11}s^{-1}m^{-1}$.

The sinusoidal wind-stress variations are defined according to

$$\tau_x(y) = \tau_0 \sin(\pi \frac{y}{L_y}) \tag{4.2}$$

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

Figure 4.1 summarizes the configuration simulated.

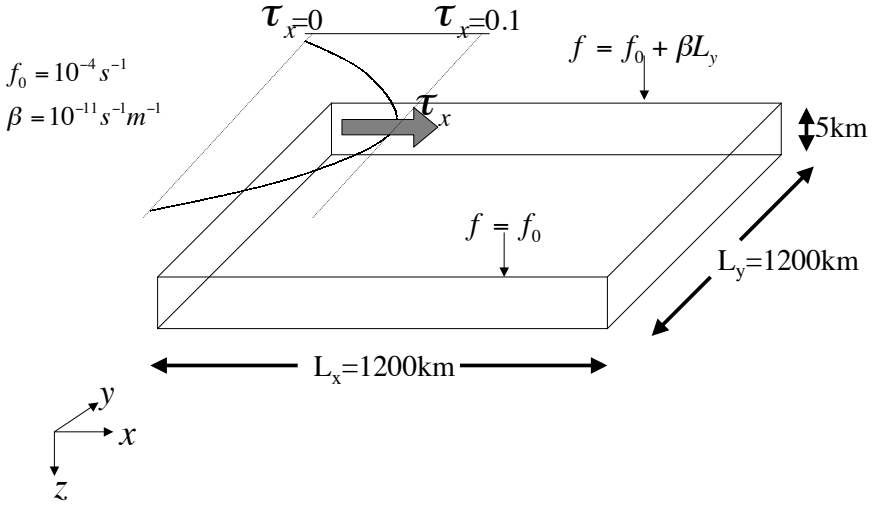

Figure 4.1: Schematic of simulation domain and wind-stress forcing function for barotropic gyre numerical experiment. The domain is enclosed by solid walls at $x = 0$, 1200 km and at $y = 0$, 1200 km.

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 4.2.3), 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 $\vec{u}$.

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.

Numerical Stability Criteria

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

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

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]*

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

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

The numerical stability for inertial oscillations *[Adc95]*

$$S_i = f^2 \delta t^2 \tag{4.6}$$

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 $|\vec{u}| = 2ms^{-1}$

$$S_a = \frac{|\vec{u}|\delta t}{\Delta x} \tag{4.7}$$

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

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.

File *input/data*

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

- Line 7
  - *viscAh=4.E2,*
  - this line sets the Laplacian friction coefficient to $400m^2s^{-1}$
- Line 10
  - *beta=1.E-11,*
  - this line sets β (the gradient of the coriolis parameter, f) to $10^{-11}s^{-1}m^{-1}$
- Lines 15 and 16
  - *rigidLid=.FALSE.,*
  - *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.
- 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 $20 \times 10^3 m$ (20 km).
- Line 42
  - *delY=60*20E3,*
  - This line sets the horizontal grid spacing between each y-coordinate line in the discrete grid to $20 \times 10^3 m$ (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

```
# Model parameters
# Continuous equation parameters
 &PARM01
 tRef=20.,
 sRef=10.,
 viscAz=1.E-2,
 viscAh=4.E2,
 diffKhT=4.E2,
 diffKzT=1.E-2,
 beta=1.E-11,
 tAlpha=2.E-4,
 sBeta =0.,
 gravity=9.81,
 gBaro=9.81,
 rigidLid=.FALSE.,
 implicitFreeSurface=.TRUE.,
 eosType='LINEAR',
 readBinaryPrec=64,
 &

# Elliptic solver parameters
 &PARM02
 cg2dMaxIters=1000,
 cg2dTargetResidual=1.E-7,
 &

# Time stepping parameters
 &PARM03
 startTime=0,
#endTime=311040000,
 endTime=12000.0,
 deltaTmom=1200.0,
```

```
 deltaTtracer=1200.0,
 abEps=0.1,
 pChkptFreq=2592000.0,
 chkptFreq=120000.0,
 dumpFreq=2592000.0,
 monitorSelect=2,
 monitorFreq=1.,
 &

# Gridding parameters
 &PARM04
 usingCartesianGrid=.TRUE.,
 usingSphericalPolarGrid=.FALSE.,
 delX=60*20E3,
 delY=60*20E3,
 delZ=5000.,
 &

# Input datasets
 &PARM05
 bathyFile='topog.box',
 hydrogThetaFile=,
 hydrogSaltFile=,
 zonalWindFile='windx.sin_y',
 meridWindFile=,
 &
```

File *input/data.pkg*

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

File *input/eedata*

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

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.

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 completecode for creating the *input/topog.box* file.

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 $Header$
2  C $Name$
3  C
4  C     /==========================================================\
5  C     | SIZE.h Declare size of underlying computational grid.    |
6  C     |==========================================================|
7  C     | The design here support a three-dimensional model grid   |
8  C     | with indices I,J and K. The three-dimensional domain     |
9  C     | is comprised of nPx*nSx blocks of size sNx along one axis|
10 C     | nPy*nSy blocks of size sNy along another axis and one    |
11 C     | block of size Nz along the final axis.                   |
12 C     | Blocks have overlap regions of size OLx and OLy along the|
13 C     | dimensions that are subdivided.                          |
14 C     \==========================================================/
15 C     Voodoo numbers controlling data layout.
16 C     sNx - No. X points in sub-grid.
17 C     sNy - No. Y points in sub-grid.
18 C     OLx - Overlap extent in X.
19 C     OLy - Overlat extent in Y.
20 C     nSx - No. sub-grids in X.
21 C     nSy - No. sub-grids in Y.
22 C     nPx - No. of processes to use in X.
23 C     nPy - No. of processes to use in Y.
24 C     Nx  - No. points in X for the total domain.
25 C     Ny  - No. points in Y for the total domain.
26 C     Nr  - No. points in R for full process domain.
27       INTEGER sNx
28       INTEGER sNy
29       INTEGER OLx
30       INTEGER OLy
31       INTEGER nSx
32       INTEGER nSy
33       INTEGER nPx
34       INTEGER nPy
35       INTEGER Nx
36       INTEGER Ny
37       INTEGER Nr
38       PARAMETER (
39      &           sNx =  30,
40      &           sNy =  30,
41      &           OLx =   2,
42      &           OLy =   2,
43      &           nSx =   2,
44      &           nSy =   2,
45      &           nPx =   1,
46      &           nPy =   1,
47      &           Nx  = sNx*nSx*nPx,
```

```
      &           Ny   = sNy*nSy*nPy,
      &           Nr   =   1)

C     MAX_OLX  - Set to the maximum overlap region size of any array
C     MAX_OLY    that will be exchanged. Controls the sizing of exch
C                routine buufers.
       INTEGER MAX_OLX
       INTEGER MAX_OLY
       PARAMETER ( MAX_OLX = OLx,
      &            MAX_OLY = OLy )

```

File *code/CPP_OPTIONS.h*

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

File *code/CPP_EEOPTIONS.h*

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

A Rotating Tank in Cylindrical Coordinates

(in directory: `verification/rotating_tank/`)

Overview

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 $3^\circ \times 1\mathrm{cm}$ cyclindrical 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. The files for this experiment can be found in the verification directory under `rotating_tank`.

example illustration from GFD lab here

Equations Solved

Discrete Numerical Configuration

The domain is discretised with a uniform cylindrical grid spacing in the horizontal set to $\Delta a = 1$‘ $cmand$: $math$: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

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.

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 an numerical integration from a previously saved state. Instead, the file thetaPol.bin will be loaded to initialized 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 unsually 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^{circ}.

- 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 $0.005m$ (5~mm).

- 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 phi coordinate varying fastest. The points are ordered from low coordinate to high coordinate for both axes. The units and orientation of the depths in this file are the same as used in the MITgcm code. In this experiment, a depth of $0m$ indicates an area outside of the tank and a depth f $-0.145m$ 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 line specify the temperatures in degrees Celsius of the interior and exterior walls of the tank – typically taken to be icewater on the inside and room temperature on the outside.

Other lines in the file *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*

```
# ====================
# | Model parameters |
# ====================
#
# Continuous equation parameters
 &PARM01
 tRef=29*20.0,
 sRef=29*35.0,
 viscAh=5.0E-6,
 viscAz=5.0E-6,
 no_slip_sides=.FALSE.,
 no_slip_bottom=.FALSE.,
 diffKhT=2.5E-6,
 diffKzT=2.5E-6,
 diffKhS=1.0E-6,
 diffKzS=1.0E-6,
 f0=0.5,
 eosType='LINEAR',
 sBeta =0.,
 gravity=9.81,
 rhoConst=1000.0,
 rhoNil=1000.0,
#heatCapacity_Cp=3900.0,
 rigidLid=.TRUE.,
 implicitFreeSurface=.FALSE.,
 nonHydrostatic=.TRUE.,
 readBinaryPrec=32,
 &

# Elliptic solver parameters
 &PARM02
 cg2dMaxIters=1000,
 cg2dTargetResidual=1.E-7,
 cg3dMaxIters=10,
 cg3dTargetResidual=1.E-9,
 &

# Time stepping parameters
 &PARM03
 nIter0=0,
 nTimeSteps=20,
#nTimeSteps=36000000,
 deltaT=0.1,
 abEps=0.1,
 pChkptFreq=2.0,
#chkptFreq=2.0,
 dumpFreq=2.0,
 monitorSelect=2,
 monitorFreq=0.1,
 &

# Gridding parameters
 &PARM04
 usingCylindricalGrid=.TRUE.,
```

```
 dXspacing=3.,
 dYspacing=0.01,
 delZ=29*0.005,
 ygOrigin=0.07,
 &

# Input datasets
 &PARM05
 hydrogThetaFile='thetaPolR.bin',
 bathyFile='bathyPolR.bin',
 tCylIn  = 0.,
 tCylOut = 20.,
 &
```

File *input/data.pkg*

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

File *input/eedata*

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

File *input/thetaPol.bin*

The {it input/thetaPol.bin} file specifies a three-dimensional (x,y,z) map of initial values of $theta$ in degrees Celsius. This particular experiment is set to random values x around 20C to provide initial perturbations.

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 {bf -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.

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*

```
C $Header$
C $Name$
C
C     /==========================================================\
C     | SIZE.h Declare size of underlying computational grid.    |
C     |==========================================================|
```

```
C     | The design here support a three-dimensional model grid   |
C     | with indices I,J and K. The three-dimensional domain     |
C     | is comprised of nPx*nSx blocks of size sNx along one axis|
C     | nPy*nSy blocks of size sNy along another axis and one    |
C     | block of size Nz along the final axis.                   |
C     | Blocks have overlap regions of size OLx and OLy along the|
C     | dimensions that are subdivided.                          |
C     \==========================================================/
C     Voodoo numbers controlling data layout.
C     sNx - No. X points in sub-grid.
C     sNy - No. Y points in sub-grid.
C     OLx - Overlap extent in X.
C     OLy - Overlat extent in Y.
C     nSx - No. sub-grids in X.
C     nSy - No. sub-grids in Y.
C     nPx - No. of processes to use in X.
C     nPy - No. of processes to use in Y.
C     Nx  - No. points in X for the total domain.
C     Ny  - No. points in Y for the total domain.
C     Nr  - No. points in Z for full process domain.
      INTEGER sNx
      INTEGER sNy
      INTEGER OLx
      INTEGER OLy
      INTEGER nSx
      INTEGER nSy
      INTEGER nPx
      INTEGER nPy
      INTEGER Nx
      INTEGER Ny
      INTEGER Nr
      PARAMETER (
     &           sNx =  30,
     &           sNy =  23,
     &           OLx =   3,
     &           OLy =   3,
     &           nSx =   4,
     &           nSy =   1,
     &           nPx =   1,
     &           nPy =   1,
     &           Nx  = sNx*nSx*nPx,
     &           Ny  = sNy*nSy*nPy,
     &           Nr  =  29)

C     MAX_OLX  - Set to the maximum overlap region size of any array
C     MAX_OLY    that will be exchanged. Controls the sizing of exch
C                routine buufers.
      INTEGER MAX_OLX
      INTEGER MAX_OLY
      PARAMETER ( MAX_OLX = OLx,
     &            MAX_OLY = OLy )

```

File *code/CPP_OPTIONS.h*

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

File *code/CPP_EEOPTIONS.h*

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

CHAPTER 5

Physical Parameterizations - Packages I

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 5.1 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 alogorithms that apply to many different simulation types. Similarly, the general purpose computational packages implement non-numerical alogorithms 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 5.1. 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}.

Overview

Using MITgcm Packages

The set of packages that will be used within a partiucular 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

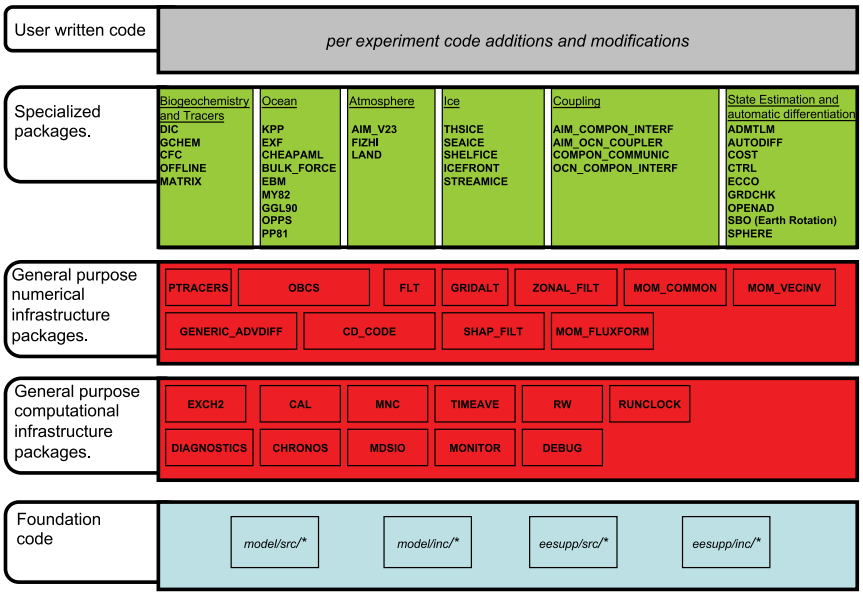

Figure 5.1: Hierarchy of code layers that are assembled to make up an MITgcm simulation. Conceptually (and in terms of code organization) MITgcm consists of several layers. At the base is a layer of core software that provides a basic numerical and computational foundation for MITgcm simulations. This 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.

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 `genamake2` 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.

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.

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 superseed (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 superseed the files in `PackA` if `PackA` is enabled before `PackB`. Thus, package ordering can be significant! For this reason, `genmake2` honors the order in which packages are specified.

These rules were adopted since they provide a relatively simple means for rapidly including (or "hiding") existing files with modified versions.

Conditional Compilation and `PACKAGES_CONFIG.h`

Given that packages are simply groups of files that may be added or removed to form a whole, one may wonder how linking (that is, FORTRAN symbol resolution) is handled. This is the second way that `genmake2` supports the concept of packages. Basically, `genmake2` creates a `Makefile` that, in turn, is able to create a file called `PACKAGES_CONFIG.h` that contains a set of C pre-processor (or "CPP") directives such as:

```
#undef  ALLOW_KPP
#undef  ALLOW_LAND
...
#define ALLOW_GENERIC_ADVDIFF
#define ALLOW_MDSIO
...
```

These CPP symbols are then used throughout the code to conditionally isolate variable definitions, function calls, or any other code that depends upon the presence or absence of any particular package.

An example illustrating the use of these defines is:

```
#ifdef ALLOW_GMREDI
      IF (useGMRedi) CALL GMREDI_CALC_DIFF(
     I        bi,bj,iMin,iMax,jMin,jMax,K,
     I        maskUp,
     O        KappaRT,KappaRS,
```

```
     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 exerecise some discipline and avoid unnecesarily "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

 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${Pkg} * flag. This flag, which is of type logical, **must* be declared in the shared header file *PARAMS.h* in the *PARM_PACKAGES* block. This convention is used to support a single runtime control file *data.pkg* which is read by the startup routine *packages_boot()* and that sets a flag controlling the runtime use of a package. This routine needs to be able to read the flags for packages that were not built at compile time. Therefore when adding a new package, in addition to creating the per-package directory in the *pkg/* subdirectory a developer should add a *use${Pkg} * flag to *PARAMS.h* and a *use${Pkg} * entry to the *packages_boot() PACKAGES* namelist. The only other package specific code that should appear outside the individual package directory are calls to the specific package API.

Packages Related to Hydrodynamical Kernel

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:*e*quations] to [sec:tracer:sub:*a*dvection$_s$chemes]).

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.

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 conjuction 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 conjuction with Adams–Bashforth time stepping.

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)
```

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]

Shapiro Filter

(in directory: pkg/shap_filt/)

Key subroutines, parameters and files

Implementation of filter is described in section [sec:shapiro-filter].

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)

FFT Filtering Code

(in directory: pkg/zonal_filt/)

Key subroutines, parameters and files

Experiments and tutorials that use zonal filter

- Held Suarez verification experiment (hs94.128x64x5)
- AIM verification experiment (aim.5l_LatLon)

exch2: Extended Cubed Sphere Topology

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 $\times$ 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.

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} sectiontitle{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 `tile00$n$.mitgrid` where n=(1:6) (one per subdomain), must be in the working directory when the MITgcm executable is run. These files are provided in the example experiments for cubed sphere configurations with 32 $\times$ 32 cube sides – please contact MITgcm support if you want to generate files for other configurations.

- As always when compiling MITgcm, the file SIZE.h must be placed where genmake2 will find it. In particular for exch2, the domain decomposition specified in SIZE.h must correspond with the particular configuration's topology specified in W2_EXCH2_TOPOLOGY.h and w2_e2setup.F. Domain decomposition issues particular to exch2 are addressed in Section 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

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 5.2 shows one possible topology for a twenty-four-tile cube, and Figure 5.4 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.

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}.

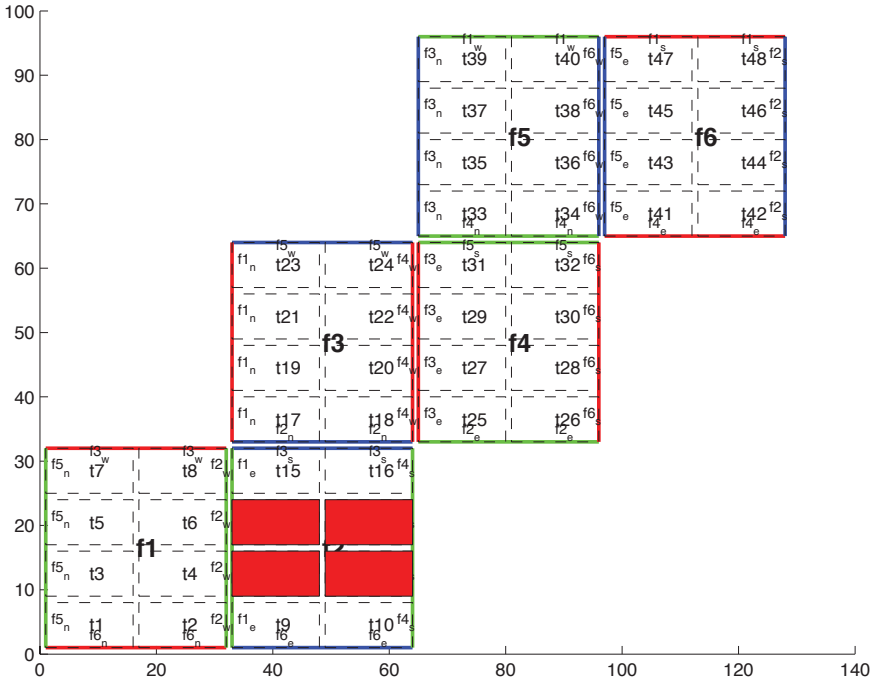

Figure 5.2: Plot of a cubed sphere topology with a 32 × 192 domain divided into six 32 × 32 subdomains, each of which is divided into eight tiles of width `tnx=16` and height `tny=8` for a total of forty-eight tiles. The colored borders of the subdomains represent the parameters `nr` (red), `ng` (green), 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.

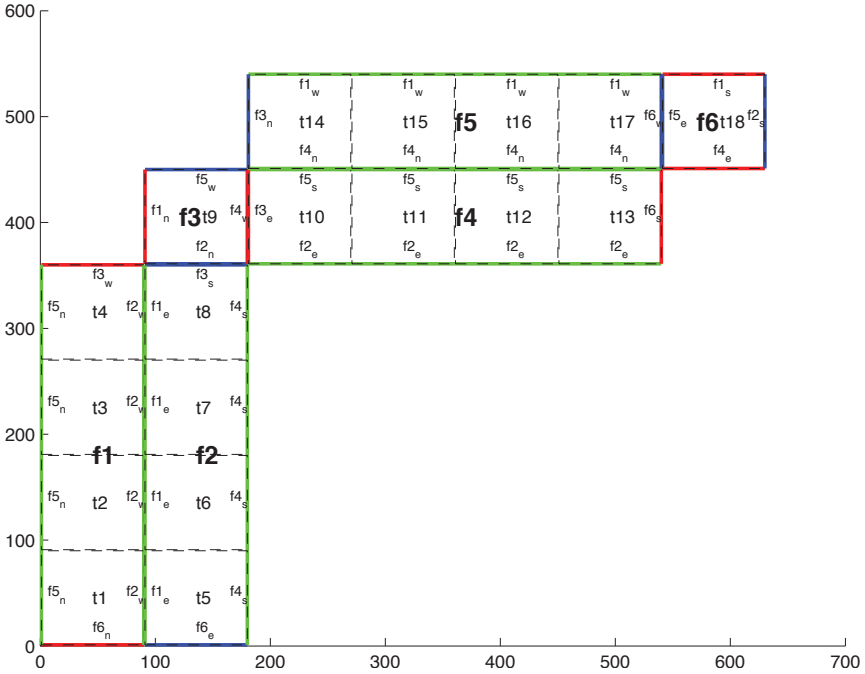

Figure 5.3: Plot of a non-square cubed sphere topology with 6 subdomains of different size (nr=90,ng=360,nb=90), divided into one to four tiles each (`tnx=90, tny=90`), resulting in a total of 18 tiles.

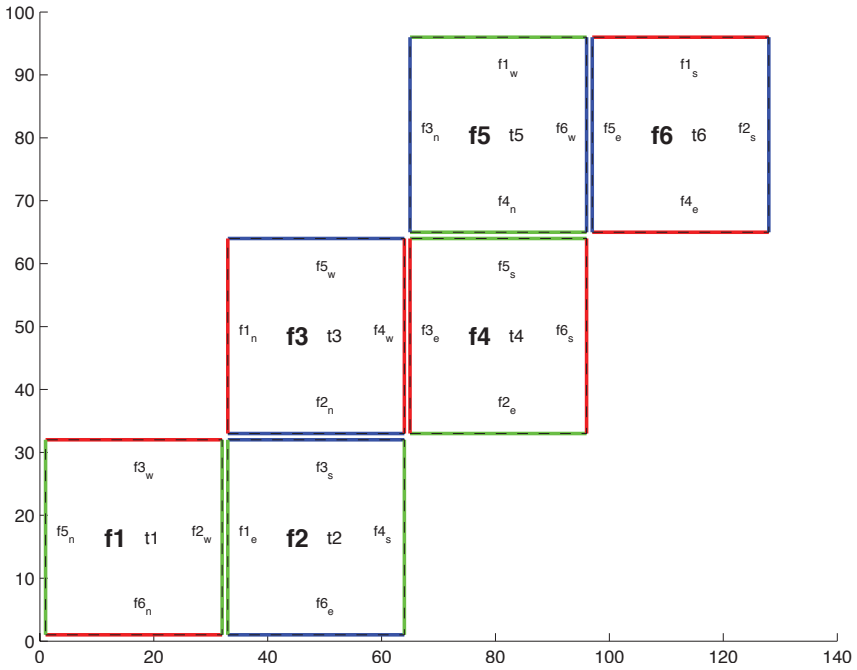

Figure 5.4: Plot of a cubed sphere topology with a 32×192 domain divided into six 32×32 subdomains with one tile each (`tnx=32, tny=32`). This is the default configuration.

The halo width parameters varlink{OLx}{OLx} and varlink{OLy}{OLy} have no special bearing on exch2 and may be assigned as in the general case. The same holds for varlink{Nr}{Nr}, the number of vertical levels in the model.

The parameters varlink{nSx}{nSx}, varlink{nSy}{nSy}, varlink{nPx}{nPx}, and varlink{nPy}{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 code{varlink{nSy}{nSy}=1} in all cases. Since the tiles as configured by exch2 cannot be split up accross processors without regenerating the topology, code{varlink{nPy}{nPy}=1} as well.

The number of tiles MITgcm allocates and how they are distributed between processors depends on varlink{nPx}{nPx} and varlink{nSx}{nSx}. varlink{nSx}{nSx} is the number of tiles per processor and varlink{nPx}{nPx} is the number of processors. The total number of tiles in the topology minus those listed in file{blanklist.txt} must equal code{nSx*nPx}. Note that in order to obtain maximum usage from a given number of processors in some cases, this restriction might entail sharing a processor with a tile that would otherwise be excluded because it is topographically outside of the domain and therefore in file{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 emph{not} listed in file{blanklist.txt}.

The following is an example of file{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,
&            nPy =   1,
```

```
&            Nx   = sNx*nSx*nPx,
&            Ny   = sNy*nSy*nPy,
&            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*nSx*nPx,
&            Ny   = sNy*nSy*nPy,
&            Nr   =   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{textsf{f}}n 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_isEedge}{exch2_isEedge}, 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 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_neighborId(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 horizonal 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{code_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.

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} sectiontitle{Cube sphere communication}. Like the original rou-

tines, 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.

Experiments and tutorials that use exch2

- Held Suarez tutorial, in tutorial_held_suarez_cs verification directory, described in section ref{sec:eg-hs}

Gridalt - Alternate Grid Package

Introduction

The gridalt package *[Mol09]* is designed to allow different components of MITgcm to be run using horizontal and/or vertical grids which are different from the main model grid. The gridalt routines handle the definition of the all the various alternative grid(s) and the mappings between them and the MITgcm grid. The implementation of the gridalt package which allows the high end atmospheric physics (fizhi) to be run on a high resolution and quasi terrain-following vertical grid is documented here. The package has also (with some user modifications) been used for other calculations within the GCM.

The rationale for implementing the atmospheric physics on a high resolution vertical grid involves the fact that the MITgcm p^* (or any pressure-type) coordinate cannot maintain the vertical resolution near the surface as the bottom topography rises above sea level. The vertical length scales near the ground are small and can vary on small time scales, and the vertical grid must be adequate to resolve them. Many studies with both regional and global atmospheric models have demonstrated the improvements in the simulations when the vertical resolution near the surface is increased (). Some of the benefit of increased resolution near the surface is realized by employing the higher resolution for the computation of the forcing due to turbulent and convective processes in the atmosphere.

The parameterizations of atmospheric subgrid scale processes are all essentially one-dimensional in nature, and the computation of the terms in the equations of motion due to these processes can be performed for the air column over one grid point at a time. The vertical grid on which these computations take place can therefore be entirely independant of the grid on which the equations of motion are integrated, and the 'tendency' terms can be interpolated to the vertical grid on which the equations of motion are integrated. A modified p^* coordinate, which adjusts to the local terrain and adds additional levels between the lower levels of the existing p^* grid (and perhaps between the levels near the tropopause as well), is implemented. The vertical discretization is different for each grid point, although it consist of the same number of levels. Additional 'sponge' levels aloft are added when needed. The levels of the physics grid are constrained to fit exactly into the existing p^* grid, simplifying the mapping between the two vertical coordinates. This is illustrated as follows:

The algorithm presented here retains the state variables on the high resolution 'physics' grid as well as on the coarser resolution 'dynamics' grid, and ensures that the two estimates of the state 'agree' on the coarse resolution grid. It

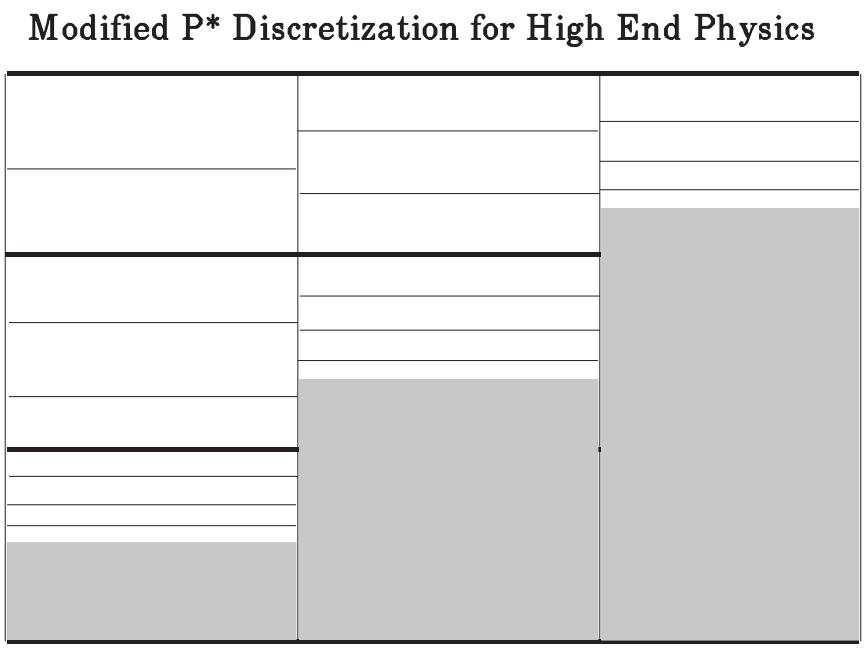

Figure 5.5: Vertical discretization for MITgcm (dark grey lines) and for the atmospheric physics (light grey lines). In this implementation, all MITgcm level interfaces must coincide with atmospheric physics level interfaces.

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.

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 neccessitates 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'.

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: ${T^{n+1}}^{**}{}_{phys}(l) = {T^n}_{phys}(l) + \delta T_{phys}$.
3. Interpolate the physics tendency to the dynamics grid, and advance the dynamics state by physics and dynamics tendencies: ${T^{n+1}}_{dyn}(L) = {T^n}_{dyn}(L) + \delta T_{dyn}(L) + [\delta T_{phys}(l)](L)$.
4. Interpolate the dynamics tendency to the physics grid, and update the physics grid due to dynamics tendencies: ${T^{n+1^*}}_{phys}(l) = {T^{n+1}}^{**}{}_{phys}(l) + \delta T_{dyn}(L)(l)$.
5. Apply correction term to physics state to account for divergence from dynamics state: ${T^{n+1}}_{phys}(l) = {T^{n+1^*}}_{phys}(l) + \gamma\{T_{dyn}(L) - [T_{phys}(l)](L)\}(l)$. Where $\gamma = 1$ here.

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.

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
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          i1, i2      - beginning and ending i - index to fill
```

```
c           j1, 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
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.

Gridalt Diagnostics

```
------------------------------------------------------------------------
<-Name->|Levs|<-parsing code->|<--  Units   -->|<- Tile (max=80c)
------------------------------------------------------------------------
DPPHYS  | 20 |SM      ML      |Pascal          |Pressure Thickness of Layers on Fizhi␣
↪Grid
```

Dos and donts

Gridalt Reference

Experiments and tutorials that use gridalt

- Fizhi experiment, in fizhi-cs-32x32x10 verification directory

General purpose numerical infrastructure packages

OBCS: Open boundary conditions for regional modeling

Authors: Alistair Adcroft, Patrick Heimbach, Samar Katiwala, Martin Losch

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.

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 5.1 summarizes these options.

Table 5.1: OBCS CPP options

CPP option	**Description**
ALLOW_OBCS_NORTH	enable Northern OB
ALLOW_OBCS_SOUTH	enable Southern OB
ALLOW_OBCS_EAST	enable Eastern OB
ALLOW_OBCS_WEST	enable Western OB
ALLOW_OBCS_PRESCRIBE	enable code for prescribing OB's
ALLOW_OBCS_SPONGE	enable sponge layer code
ALLOW_OBCS_BALANCE	enable code for balancing transports through OB's
ALLOW_ORLANSKI	enable Orlanski radiation conditions at OB's
ALLOW_OBCS_STEVENS	enable Stevens (1990) boundary conditions at OB's
	(currently only implemented for eastern and
	western boundaries and NOT for ptracers)

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 5.2 summarizes the runtime flags that are set in `data.obcs`, and their default values.

Table 5.2: pkg OBCS run-time parameters

Flag/parameter	default	Description
basic flags & parameters (OBCS_PARM01)		
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 balaning 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
OB**Xy**File		file name of OB field
		X: **N**(orth) **S**(outh) **E**(ast) **W**(est)
		y: **t**(emperature) **s**(salinity) **u**(-velocity) **v**(-velocity)
		w(-velocity) **eta** (sea surface height)
		a (sea ice area) **h** (sea ice thickness) **sn** (snow thickness) **sl** (sea ice
Orlanski parameters (OBCS_PARM02)		
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.`	
Sponge-layer parameters (OBCS_PARM03)		
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
Stevens parameters (OBCS_PARM04)		
T/SrelaxStevens	0 sec	relaxation time scale for temperature/salinity
useStevensPhaseVel	code{.TRUE.}	
useStevensAdvection	code{.TRUE.}	

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 $i = 1, \ldots, N_x$ a meridional index j specifies the Northern/Southern OB position, and for each meridional position $j = 1, \ldots, N_y$, a zonal index i specifies the Eastern/Western OB position. For Northern/Southern OB this defines an N_x-dimensional "row" array `OB_Jnorth(Nx)` / `OB_Jsouth(Nx)`, and an N_y-dimenisonal "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. $\texttt{OB\_Jnorth}(\texttt{3}) = -\texttt{1}$ means that the point $(\texttt{3}, \texttt{Ny})$ is a northern OB.

Simple examples: For a model grid with :math:‘ N_{x}times N_{y} = 120times144‘ horizontal grid points with four open boundaries along the four egdes 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,
```

Equations and key routines

OBCS_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:*f*lags]).

OBCS_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 $(N_x \times N_r \times$ time levels$)$, for East/West boundary files the dimensions are $(N_y \times N_r \times$ time levels$)$.
- If a non-linear free surface is used (ref{sec:nonlinear-freesurface}), additional files `OB[N/S/E/W]etaFile` for the sea surface height eta with dimension $(N_{x/y} \times$ time levels$)$ may be specified.
- If non-hydrostatic dynamics are used (ref{sec:non-hydrostatic}), additional files `OB[N/S/E/W]wFile` for the vertical velocity w with dimensions $(N_{x/y} \times N_r \times$ time levels$)$ can be specified.
- If `useSEAICE=.TRUE.` then additional files `OB[N/S/E/W][a,h,sl,sn,uice,vice]` for sea ice area, thickness (`HEFF`), seaice salinity, snow and ice velocities $(N_{x/y} \times$ time levels$)$ can be specified.

As in `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:*s*tepping], 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 boudary 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):

$$\chi^{n+1}_{i_b,j,k} = \chi^{n}_{i_b,j,k} + \Delta t (u^{n+1} + c)_{i_b,j,k} \frac{\chi^{n}_{i_b,j,k} - \chi^{n}_{i_b-1,j,k}}{\Delta x^{C}_{i_b,j}}, \text{ if } u^{n+1}_{i_b,j,k} > 0,$$

where i_b is the boundary index. For test purposes, the phase velocity contribution or the entire advection can be turned off by setting the corresponding parameters `useStevensPhaseVel` and `useStevensAdvection` to `.FALSE.`.

See *[Ste90]* for details. With this boundary condition specifying the exact net transport across the open boundary is simple, so that balancing the flow with (S/R~OBCS_BALANCE_FLOW, see next paragraph) is usually not necessary.

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 substracting a uniform velocity to ensure zero net transport through the Western open boundary;
- correct Eastern and Northern normal flow, with the Northern velocity correction two times larger than the Eastern correction, but *not* the Southern normal flow, to ensure that the total inflow through East, Northern, and Southern open boundary is balanced.

The old method of balancing the net flow for all sections individually can be recovered by setting all flags to -1. Then the normal velocities across each of the four boundaries are modified separately, so that the net volume transport across *each* boundary is zero. For example, for the western boundary at $i = i_b$, the modified velocity is:

$$u(y,z) - \int_{\text{western boundary}} u\, dy\, dz \approx OBNu(j,k) - \sum_{j,k} OBNu(j,k) h_w(i_b,j,k) \Delta y_G(i_b,j) \Delta z(k).$$

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, gobal 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

$$G^{(\text{sponge})}_{\chi} = -\frac{\chi - [(L-\delta L)\chi_{BC} + \delta L \chi]/L}{[(L-\delta L)\tau_b + \delta L \tau_i]/L} = -\frac{\chi - [(1-l)\chi_{BC} + l\chi]}{[(1-l)\tau_b + l\tau_i]}$$

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

Flow chart

```
C     !CALLING SEQUENCE:
c ...
```

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)
------------------------------------------------------
```

Reference experiments

In the directory `verifcation`, 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].

References

Experiments and tutorials that use obcs

- `tutorial\_plume\_on\_slope` (section~ref{sec:eg-gravityplume})

RBCS Package

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 seperate 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.rbcs (see below).

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.rbcs`:

Set in RBCS_PARM01: - **rbcsForcingPeriod**: time interval between forcing fields (in seconds), zero means constant-in-time forcing. - **rbcsForcingCycle**: repeat cycle of forcing fields (in seconds), zero means non-cyclic forcing. - **rbcsForcingOffset**: 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 $\mathrm{rbcsForcingOffset} + \mathrm{rbcsForcingPeriod}/2$; see below for examples. - **rbcsSingleTimeFiles**: true or false (default false), if true, forcing fields are given 1 file per rbcsForcingPeriod. - **deltaTrbcs**: time step used to compute the iteration numbers for rbcsSingleTimeFiles=T. - **rbcsIter0**: shift in iteration numbers used to label files if rbcsSingleTimeFiles=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/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 rbcsSingleTimeFiles=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:rbcs: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.

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 5.3: 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 = rbcsForcingPeriod

c = rbcsForcingCycle

t_0 = rbcsForcingOffset

i_0 = rbcsIter0

Δt_{rbcs} = deltaTrbcs

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 $p/2$.

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 $t =$ rbcsForcingPeriod.

Do's and Don'ts

Reference Material

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]*

PTRACERS Package

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 tp 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).

Equations

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 <= 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 seperate 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, dafaults 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.

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␣
↪Carbon
UTRAC01  | 15 |UU   171MR      |mol C/m.m/s     |Zonal Mass-Weighted Transp of␣
↪Dissolved Inorganic Carbon
VTRAC01  | 15 |VV   170MR      |mol C/m.m/s     |Merid Mass-Weighted Transp of␣
↪Dissolved Inorganic Carbon
WTRAC01  | 15 |WM      MR      |mol C/m.m/s     |Vert  Mass-Weighted Transp of␣
↪Dissolved Inorganic Carbon
ADVrTr01| 15 |WM      LR      |mol C/m.m^3/s   |Vertical   Advective Flux of␣
↪Dissolved Inorganic Carbon
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
DFrETr01| 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
```

```
DFrITr01| 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
DFrETr02| 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
DFrITr02| 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
DFrETr03| 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
DFrITr03| 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
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
DFrETr04| 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
DFrITr04| 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
DFrETr05| 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
DFrITr05| 15 |WM      LR      |mol O/m.m^3/s   |Vertical Diffusive Flux of Dissolved␣
↪Oxygen (Implicit part)
```

Do's and Don'ts

Reference Material

Ocean Packages

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.

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:

$$\nabla \cdot \kappa_\rho \mathbf{K_{Redi}} \nabla \tau$$

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:

$$\mathbf{K_{Redi}} = \frac{\mathbf{1}}{\mathbf{1} + |\mathbf{S}|^\mathbf{2}} \begin{pmatrix} 1+S_y^2 & -S_x S_y & S_x \\ -S_x S_y & 1+S_x^2 & S_y \\ S_x & S_y & |S|^2 \end{pmatrix}$$

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, $|S| << 1$. The Redi projection tensor then becomes:

$$\mathbf{K_{Redi}} = \begin{pmatrix} 1 & 0 & S_x \\ 0 & 1 & S_y \\ S_x & S_y & |S|^2 \end{pmatrix}$$

GM parameterization

The GM parameterization aims to represent the "advective" or "transport" effect of geostrophic eddies by means of a "bolus" velocity, $\mathbf{u}^\star$. The divergence of this advective flux is added to the tracer tendency equation (on the rhs):

$$-\nabla \cdot \tau \mathbf{u}^\star$$

The bolus velocity $\mathbf{u}^\star$ is defined as the rotational of a streamfunction $\mathbf{F}^\star$=$(F_x^\star, F_y^\star, 0)$:

$$\mathbf{u}^\star = \nabla \times \mathbf{F}^\star = \begin{pmatrix} -\partial_z F_y^\star \\ +\partial_z F_x^\star \\ \partial_x F_y^\star - \partial_y F_x^\star \end{pmatrix},$$

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:

$$\begin{aligned} F_x^\star &= -\kappa_{GM} S_y \\ F_y^\star &= \kappa_{GM} S_x \end{aligned}$$

with boundary conditions $F_x^\star = F_y^\star = 0$ on upper and lower boundaries. In the end, the bolus transport in the GM parameterization is given by:

$$\mathbf{u}^\star = \begin{pmatrix} u^\star \\ v^\star \\ w^\star \end{pmatrix} = \begin{pmatrix} -\partial_z(\kappa_{GM} S_x) \\ -\partial_z(\kappa_{GM} S_y) \\ \partial_x(\kappa_{GM} S_x) + \partial_y(\kappa_{GM} S_y) \end{pmatrix}$$

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:

$$\begin{pmatrix} GM\_PsiX \\ GM\_PsiY \end{pmatrix} = \begin{pmatrix} \kappa_{GM} S_x \\ \kappa_{GM} S_y \end{pmatrix} = \begin{pmatrix} F_y^\star \\ -F_x^\star \end{pmatrix}.$$

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:

$$\begin{aligned}\mathbf{u}^\star\tau = & \begin{pmatrix} -\partial_z(\kappa_{GM}S_x)\tau \\ -\partial_z(\kappa_{GM}S_y)\tau \\ (\partial_x\kappa_{GM}S_x + \partial_y\kappa_{GM}S_y)\tau \end{pmatrix} \\ = & \begin{pmatrix} -\partial_z(\kappa_{GM}S_x\tau) \\ -\partial_z(\kappa_{GM}S_y\tau) \\ \partial_x(\kappa_{GM}S_x\tau) + \partial_y(\kappa_{GM}S_y\tau) \end{pmatrix} + \begin{pmatrix} \kappa_{GM}S_x\partial_z\tau \\ \kappa_{GM}S_y\partial_z\tau \\ -\kappa_{GM}S_x\partial_x\tau - \kappa_{GM}S_y\partial_y\tau \end{pmatrix}\end{aligned}$$

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:

$$\mathbf{u}^\star\tau = -\kappa_{\mathbf{GM}}\mathbf{K}_{\mathbf{GM}}\nabla\tau$$

where

$$\mathbf{K}_{\mathbf{GM}} = \begin{pmatrix} 0 & 0 & -S_x \\ 0 & 0 & -S_y \\ S_x & S_y & 0 \end{pmatrix}$$

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:

$$\kappa_\rho\mathbf{K}_{\mathbf{Redi}}\nabla\tau - \mathbf{u}^\star\tau = (\kappa_\rho\mathbf{K}_{\mathbf{Redi}} + \kappa_{\mathbf{GM}}\mathbf{K}_{\mathbf{GM}})\nabla\tau$$

In the instance that $\kappa_{GM} = \kappa_\rho$ then

$$\kappa_\rho\mathbf{K}_{\mathbf{Redi}} + \kappa_{\mathbf{GM}}\mathbf{K}_{\mathbf{GM}} = \kappa_\rho \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 2S_x & 2S_y & |S|^2 \end{pmatrix}$$

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)

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:

$$\kappa_{GM} = \alpha L^2 \overline{\frac{|f|}{\sqrt{Ri}}}^z$$

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:

$$\kappa_{GM} = \alpha L^2 \overline{\frac{M^2}{N}}^z = \alpha L^2 \overline{\frac{M^2}{N^2}N}^z = \alpha L^2 \overline{|S|N}^z$$

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)

Slope clipping

Deep convection sites and the mixed layer are indicated by homogenized, unstable or nearly unstable stratification. The slopes in such regions can be either infinite, very large with a sign reversal or simply very large. From a numerical point of view, large slopes lead to large variations in the tensor elements (implying large bolus flow) and can be numerically unstable. This was first recognized by *[Cox87]* who implemented "slope clipping" in the isopycnal mixing tensor. Here, the slope magnitude is simply restricted by an upper limit:

$$\begin{aligned}
|\nabla\sigma| &= \sqrt{\sigma_x^2 + \sigma_y^2} \\
S_{lim} &= -\frac{|\nabla\sigma|}{S_{max}} \quad \text{where } S_{max} \text{ is a parameter} \\
\sigma_z^\star &= \min(\sigma_z, S_{lim}) \\
[s_x, s_y] &= -\frac{[\sigma_x, \sigma_y]}{\sigma_z^\star}
\end{aligned}$$

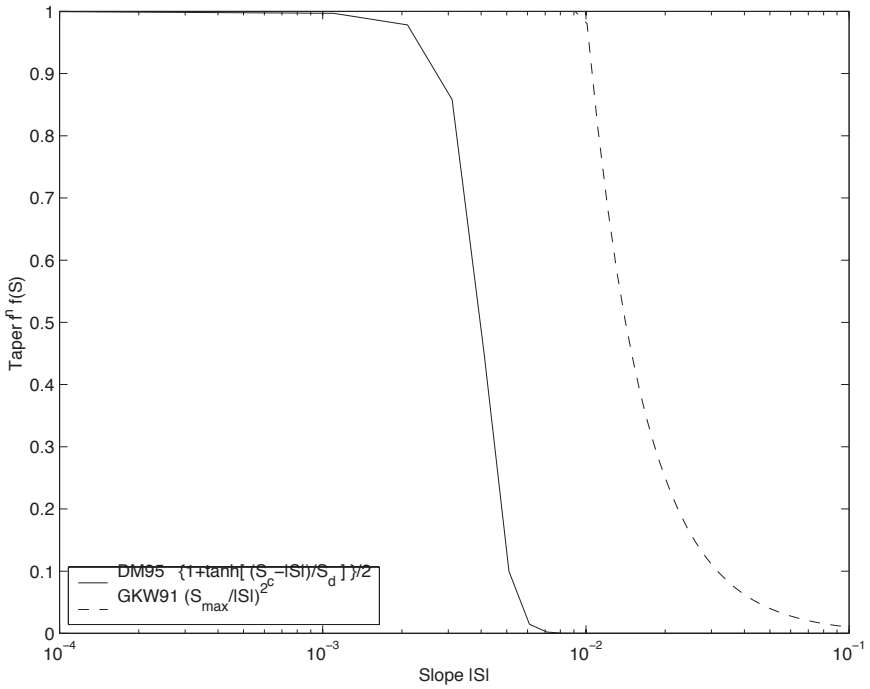

Figure 5.6: Taper functions used in GKW91 and DM95.

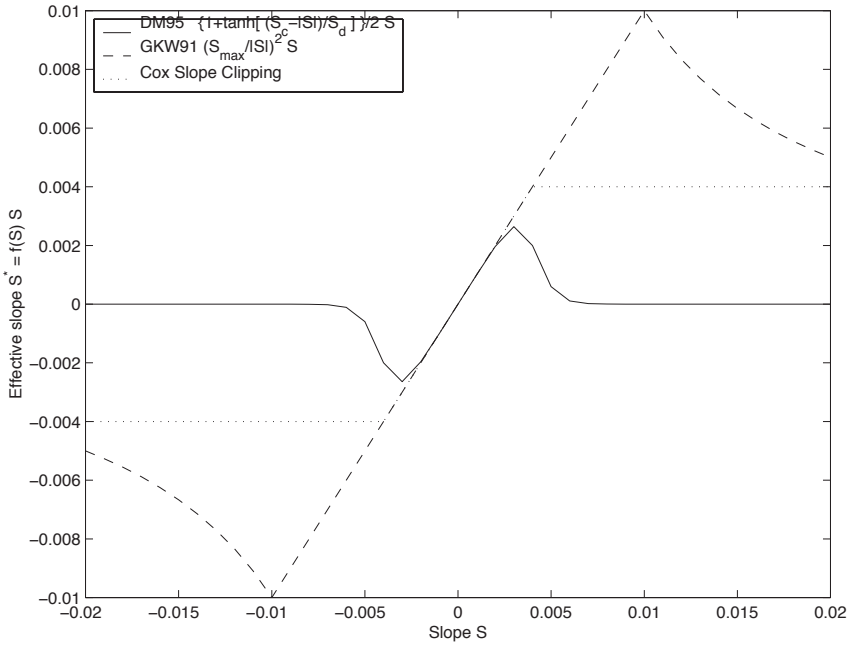

Figure 5.7: Effective slope as a function of 'true' slope using Cox slope clipping, GKW91 limiting and DM95 limiting.

Notice that this algorithm assumes stable stratification through the "min" function. In the case where the fluid is well stratified ($\sigma_z < S_{lim}$) then the slopes evaluate to:

$$[s_x, s_y] = -\frac{[\sigma_x, \sigma_y]}{\sigma_z}$$

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.

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*.

Tapering: Danabasoglu and McWilliams, J. Clim. 1995

The tapering scheme used by followed a similar procedure but used a different tapering function, $f_1(S)$:

$$f_1(S) = \frac{1}{2}\left(1 + \tanh\left[\frac{S_c - |S|}{S_d}\right]\right)$$

where $S_c = 0.004$ is a cut-off slope and $S_d = 0.001$ is a scale over which the slopes are smoothly tapered. Functionally, the operates in the same way as the GKW91 scheme but has a substantially lower cut-off, turning off the GM/Redi SGS parameterization for weaker slopes.

The DM95 tapering scheme is activated in the model by setting **GM_taper_scheme = 'dm95'** in *data.gmredi*.

Tapering: Large, Danabasoglu and Doney, JPO 1997

The tapering used in *[LDDM97]* is based on the DM95 tapering scheme, but also tapers the scheme with an additional function of height, $f_2(z)$, so that the GM/Redi SGS fluxes are reduced near the surface:

$$f_2(z) = \frac{1}{2}\left(1 + \sin(\pi \frac{z}{D} - \frac{\pi}{2})\right)$$

where $D = L_\rho |S|$ is a depth-scale and $L_\rho = c/f$ with $c = 2$ m s:math:^{-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*.

Figure 5.8: **Figure missing** Mixed layer depth using GM parameterization with a) Cox slope clipping and for comparison b) using horizontal constant diffusion.

Package Reference

```
------------------------------------------------------------------------
<-Name->|Levs|<-parsing code->|<--  Units   -->|<- Tile (max=80c)
------------------------------------------------------------------------
GM_VisbK|  1 |SM P    M1      |m^2/s           |Mixing coefficient from Visbeck etal␣
↪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 Tempetature flux: X␣
↪component
GM_KvzTz| 15 |VV   185MR      |degC.m^3/s      |Redi Off-diagonal Tempetature flux: Y␣
↪component
```

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.

KPP: Nonlocal K-Profile Parameterization for Vertical Mixing

Authors: Dimitris Menemenlis and Patrick Heimbach

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 goverened 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]).

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 5.4 summarizes them.

Table 5.4: CPP flags for KPP

CPP option	Description
_KPP_RL	
FRUGAL_KPP	
KPP_SMOOTH_SHSQ	
KPP_SMOOTH_DVSQ	
KPP_SMOOTH_DENS	
KPP_SMOOTH_VISC	
KPP_SMOOTH_DIFF	
KPP_ESTIMATE_UREF	
INCLUDE_DIAGNOSTICS_INTERFACE_CODE	
KPP_GHAT	
EXCLUDE_KPP_SHEAR_MIX	

Run-time parameters

Run-time parameters are set in files `data.pkg` and `data.kpp` which are read in `kpp_readparms.F`. Run-time parameters may be broken into 3 categories: (i) switching on/off the package at runtime, (ii) required MITgcm flags, (iii) package flags and parameters.

Enabling the package

The KPP package is switched on at runtime by setting `useKPP = .TRUE.` in `data.pkg`.

Required MITgcm flags

The following flags/parameters of the MITgcm dynamical kernel need to be set in conjunction with KPP:

`implicitViscosity = .TRUE.`	enable implicit vertical viscosity
`implicitDiffusion = .TRUE.`	enable implicit vertical diffusion

Package flags and parameters

Table 5.5 summarizes the runtime flags that are set in `data.pkg`, and their default values.

Table 5.5: Runtime flags for KPP

Flag/parameter	default	Description
I/O related parameters		
kpp_freq	`deltaTClock`	Recomputation frequency for KPP fields
kpp_dumpFreq	`dumpFreq`	Dump frequency of KPP field snapshots
kpp_taveFreq	`taveFreq`	Averaging and dump frequency of KPP fields
KPPmixingMaps	`.FALSE.`	include KPP diagnostic maps in STDOUT

Continued on next page

Table 5.5 – 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
General KPP parameters		
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	
Boundary layer parameters (S/R bldepth)		
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 turbulant velocity shear (= function of concv,concs,epsilon,vonk,Ricr)
Boundary layer mixing parameters (S/R blmix)		
cstar	10.	proportionality coefficient for nonlocal transport
cg		non-dimensional coefficient for counter-gradient term (= function of cstar,vonk,concs,epsilon)
Interior mixing parameters (S/R Ri_iwmix)		
Riinfty	0.7	gradient Richardson number limit for shear instability
BVDQcon	-0.2E-4 s^{-2}	Brunt-Väisalä 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 5.5 – 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

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 $\overline{wx}$ 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

$$\overline{wx}(d) \;=\; -K_x \left(\frac{\partial X}{\partial z} \;-\; \gamma_x \right)$$

- *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 :math:‘ 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:

$$\gamma_x \;=\; 0 \qquad\qquad \zeta \;\geq\; 0$$

$$\left.\begin{aligned} \gamma_m \;&=\; 0 \\ \gamma_s \;&=\; C_s \tfrac{\overline{ws_0}}{w_s(\sigma)h} \\ \gamma_\theta \;&=\; C_s \tfrac{\overline{w\theta_0}+\overline{w\theta_R}}{w_s(\sigma)h} \end{aligned}\right\} \quad \zeta \;<\; 0$$

In practice, the routine peforms the following tasks:

1. compute velocity scales at hbl
2. find the interior viscosities and derivatives at hbl

3. compute turbulent velocity scales on the interfaces
4. compute the dimensionless shape functions at the interfaces
5. compute boundary layer diffusivities at the interfaces
6. compute nonlocal transport term
7. find diffusivities at kbl-1 grid level

RI_IWMIX: Mixing in the interior

Compute interior viscosity and diffusivity coefficients due to

- shear instability (dependent on a local gradient Richardson number),
- to background internal wave activity, and
- to static instability (local Richardson number < 0).

TO BE CONTINUED.

BLDEPTH: Boundary layer depth calculation:

The oceanic planetary boundary layer depth, `hbl`, is determined as the shallowest depth where the bulk Richardson number is equal to the critical value, `Ricr`.

Bulk Richardson numbers are evaluated by computing velocity and buoyancy differences between values at zgrid(kl) < 0 and surface reference values. In this configuration, the reference values are equal to the values in the surface layer. When using a very fine vertical grid, these values should be computed as the vertical average of velocity and buoyancy from the surface down to epsilon*zgrid(kl).

When the bulk Richardson number at k exceeds Ricr, hbl is linearly interpolated between grid levels zgrid(k) and zgrid(k-1).

The water column and the surface forcing are diagnosed for stable/ustable forcing conditions, and where hbl is relative to grid points (caseA), so that conditional branches can be avoided in later subroutines.

TO BE CONTINUED.

KPP_CALC_DIFF_T/_S, KPP_CALC_VISC:

Add contribution to net diffusivity/viscosity from KPP diffusivity/viscosity.

TO BE CONTINUED.

KPP_TRANSPORT_T/_S/_PTR:

Add non local KPP transport term (ghat) to diffusive temperature/salinity/passive tracer flux. The nonlocal transport term is nonzero only for scalars in unstable (convective) forcing conditions.

TO BE CONTINUED.

Implicit time integration

TO BE CONTINUED.

Penetration of shortwave radiation

TO BE CONTINUED.

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
```

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
```

Reference experiments

lab_sea:

natl_box:

References

Experiments and tutorials that use kpp

- Labrador Sea experiment, in lab_sea verification directory

GGL90: a TKE vertical mixing scheme

(in directory: *pkg/ggl90/*)

Key subroutines, parameters and files

see *[GGL90]*

Experiments and tutorials that use GGL90

- Vertical mixing verification experiment (`vermix/input.ggl90`)

OPPS: Ocean Penetrative Plume Scheme

(in directory: *pkg/opps/*)

Key subroutines, parameters and files

See *[PR97]*

Experiments and tutorials that use OPPS

- Vertical mixing verification experiment (*vermix/input.opps*)

KL10: Vertical Mixing Due to Breaking Internal Waves

(in directory: *pkg/kl10/*)

Authors: Jody M. Klymak

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

$$L_O^2 \approx \epsilon N^{-3}.$$

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 in non-hydrostatic. That is because there will be significant shear at small scales that represents the turbulence being parameterized in the scheme. At very high resolutions direct numerical simulation or more sophisticated large-eddy schemes should be used.
- We find that grid cells of approximately 10 to 1 aspect ratio are a good rule of thumb for achieving good results are usual oceanic scales. For a site like the Hawaiian Ridge, and Luzon Strait, this means 10-m vertical resolusion 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 pinpoiting 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.

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).

Run-time parameters

Run-time parameters are set in files `data.pkg` and `data.kl10` which are read in `kl10_readparms.F`. Run-time parameters may be broken into 3 categories: (i) switching on/off the package at runtime, (ii) required MITgcm flags, (iii) package flags and parameters.

Enabling the package

The KL10 package is switched on at runtime by setting `useKL10 = .TRUE.` in `data.pkg`.

Required MITgcm flags

The following flags/parameters of the MITgcm dynamical kernel need to be set in conjunction with KL10:

`implicitViscosity = .TRUE.`	enable implicit vertical viscosity
`implicitDiffusion = .TRUE.`	enable implicit vertical diffusion

Package flags and parameters

Table 5.6 summarizes the runtime flags that are set in `data.kl10`, and their default values.

Table 5.6: KL10 runtime parameters.

Flag/parameter	default	Description
KLviscMax	300 m^2 s^{-1}	Maximum viscosity the scheme will ever give (useful for stability)
KLdumpFreq	`dumpFreq`	Dump frequency of KL10 field snapshots
KLtaveFreq	`taveFreq`	Averaging and dump frequency of KL10 fields
KLwriteState	`.FALSE.`	write KL10 state to file

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.

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.
```

References

Klymak and Legg, 2010, *Oc. Modell.*.

Experiments and tutorials that use KL10

- Modified Internal Wave experiment, in internal_wave verification directory

BULK_FORCE: Bulk Formula Package

author: Stephanie Dutkiewicz

Instead of forcing the model with heat and fresh water flux data, this package calculates these fluxes using the changing sea surface temperature. We need to read in some atmospheric data: **air temperature, air humidity, down shortwave radiation, down longwave radiation, precipitation, wind speed**. The current setup also reads in **wind stress**, but this can be changed so that the stresses are calculated from the wind speed.

The current setup requires that there is the thermodynamic-seaice package (*pkg/thsice*, also refered 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.

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.

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.

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:

$$\begin{aligned} u^* = &\quad c_u u_s \\ T^* = &\quad c_T \Delta T \\ q^* = &\quad c_q \Delta q \end{aligned}$$

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 ($\Upsilon < 0$):

$$\begin{aligned} \psi_m = &\quad 2ln(0.5(1+\chi)) + ln(0.5(1+\chi^2)) - 2\tan^{-1}\chi + \pi/2 \\ \psi_s = &\quad 2ln(0.5(1+\chi^2)) \end{aligned}$$

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:

$$\begin{aligned} c_u = &\quad \frac{\hat{c_u}}{1 + \hat{c_u}(\lambda - \psi_m)/\kappa} \\ c_T = &\quad \frac{\hat{c_T}}{1 + \hat{c_T}(\lambda - \psi_s)/\kappa} \\ c_q = &\quad c'_T \end{aligned}$$

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

$$Q_{lw}^{up} = \epsilon \sigma T_{srf}^4$$

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

$$\begin{aligned} \frac{dQ_s}{d_T} = &\quad \rho_{air} c_{p_{air}} u_s c_u c_T \\ \frac{dQ_l}{d_T} = &\quad \frac{\rho_{air} L^2 u_s c_u c_q c}{T_{srf}^2} \\ \frac{dQ_{]lw}^{up}}{d_T} = &\quad 4\epsilon\sigma t_{srf}^3 \end{aligned}$$

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.

Initializing subroutines

`bulkf_init.F`: Set bulkf variables to zero.

`bulkf_readparms.F`: Reads **data.blk**

Diagnostic subroutines

`bulkf_ave.F`: Keeps track of means of the bulkf variables

`bulkf_diags.F`: Finds averages and writes out diagnostics

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.

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.

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).

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)

Experiments and tutorials that use bulk_force

- Global ocean experiment in global_ocean.cs32x15 verification directory, input from input.thsice directory.

EXF: The external forcing package

Authors: Patrick Heimbach and Dimitris Menemenlis

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:*u*nits], and available diagnostics output is listed in Section [sec:pkg:exf:diagnostics].

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 5.7 summarizes these options.

Table 5.7: 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
used in conjunction with relaxation to prescribed (climatological) fields	
`ALLOW_CLIMSST_RELAXATION`	relaxation to 2-D SST climatology
`ALLOW_CLIMSSS_RELAXATION`	relaxation to 2-D SSS climatology
these are set outside of EXF in `CPP_OPTIONS.h`	
`SHORTWAVE_HEATING`	enable shortwave radiation
`ATMOSPHERIC_LOADING`	enable surface pressure forcing

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 5.8: 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 `_YYYY` on filename
twoDigitYear	`.FALSE.`	instead of appending `_YYYY` append `YY`
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.	overall scaling of neutral drag coeff.
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/VICE from U/V]WIND before computing [U/V]STRESS
zref	10.	reference height

Continued on next page

Table 5.8 – continued from previous page

Flag/parameter	default	Description
hu	10.	height of mean wind
ht	2.	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.	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.	latent heat of evaporation [J/kg]
flami	334000.	latent heat of melting of pure ice [J/kg]
zolmin	-100.	minimum stability parameter
cvapor_fac	640380.	
cvapor_exp	5107.4	
cvapor_fac_ice	11637800.	
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.	
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:*u*nits]. Each field has a number of attributes which can be customized. They are summarized in Table [tab:pkg:exf:runtime:sub:*a*ttributes]. 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 5.9: EXF runtime attributes Note there is one exception for the default of `atempconst` = celsius2K = 273.16

attribute	Default	Description
field `file`	‘ ‘	filename; if left empty no file will be read; `const` will be used instead
field `const`	0.	constant that will be used if no file is read
field `startdate1`	0.	format: `YYYYMMDD`; start year (YYYY), month (MM), day (YY)
		of field to determine record number
field `startdate2`	0.	format: `HHMMSS`; start hour (HH), minute (MM), second(SS)
		of field to determine record number
field `period`	0.	interval in seconds between two records
`exf_inscal_`*field*		optional rescaling of input fields to comply with EXF units
`exf_outscal_`*field*		optional rescaling of EXF fields when mapped onto MITgcm fields
used in conjunction with `EXF_USE_INTERPOLATION`		
field `_lon0`	`xgOrigin+delX/2`	starting longitude of input
field `_lon_inc`	`delX`	increment in longitude of input
field `_lat0`	`ygOrigin+delY/2`	starting latitude of input
field `_lat_inc`	`delY`	increment in latitude of input
field `_nlon`	`Nx`	number of grid points in longitude of input
field `_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.

EXF bulk formulae

T.B.D. (cross-ref. to parameter list table)

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               |
c     field     :: Description
c               |
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--------------------------------------------------------------------
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--------------------------------------------------------------------
c     hs        :: sensible heat flux into ocean in W/m^2
c               |  > 0 for increase in theta (ocean warming)
c--------------------------------------------------------------------
c     hl        :: latent   heat flux into ocean in W/m^2
c               |  > 0 for increase in theta (ocean warming)
c--------------------------------------------------------------------
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--------------------------------------------------------------------
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-----------------------------------------------------------------------
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-----------------------------------------------------------------------
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
c               |  Southwest C-grid U point
c               |  Input or input/output field
c-----------------------------------------------------------------------
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-----------------------------------------------------------------------
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-----------------------------------------------------------------------
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-----------------------------------------------------------------------
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-----------------------------------------------------------------------
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-----------------------------------------------------------------------
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-----------------------------------------------------------------------
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-----------------------------------------------------------------------
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--------------------------------------------------------------------
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        !!!
c--------------------------------------------------------------------
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--------------------------------------------------------------------
```

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, aqh, 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
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

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
 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
 EXFuwind|  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
 EXFwspee|  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
```

References

Experiments and tutorials that use exf

- Global Ocean experiment, in global_with_exf verification directory
- Labrador Sea experiment, in lab_sea verification directory

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.

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.

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           ( yyyymmdd, hhmmss, leap_year, dayofweek )
c
c             date(1) = yyyymmdd    <-- 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
```

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.

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
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*.

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".

Short routine description

```
c      o  cal_Init          - Initialise the calendar. This is the interface
c                             to MITgcm.
c
c      o  cal_Set           - Sets the calendar according to the user
c                             specifications.
c
c      o  cal_GetDate       - Given the model's current timestep or the
c                             model's current time return the corresponding
c                             calendar date.
c
c      o  cal_FullDate      - Complete a date specification (leap year and
c                             day of the week).
c
```

```
c      o  cal_IsLeap         - Determine whether a given year is a leap year.
c
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
c                              or to a date.
c
c      o  cal_TimeInterval   - Given a time interval return the corresponding
c                              date array.
c
c      o  cal_SubDates       - Determine the time interval between two dates
c                              or between two time intervals.
c
c      o  cal_ConvDate       - Decompose a date array or a time interval
c                              array into its components.
c
c      o  cal_CopyDate       - Copy a date array or a time interval array to
c                              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
c                              of seconds.
c
c      o  cal_WeekDay        - Return the weekday as a string given the
c                              calendar date.
c
c      o  cal_NumInts        - Return the number of time intervals between two
c                              given dates.
c
c      o  cal_StepsPerDay    - Given an iteration number or the current
c                              integration time return the number of time
c                              steps to integrate in the current calendar day.
c
c      o  cal_DaysPerMonth   - Given an iteration number or the current
c                              integration time return the number of days
c                              to integrate in this calendar month.
c
c      o  cal_MonthsPerYear  - Given an iteration number or the current
c                              integration time return the number of months
c                              to integrate in the current calendar year.
c
c      o  cal_StepsForDay    - Given the integration day return the number
c                              of steps to be integrated, the first step,
c                              and the last step in the day specified. The
c                              first and the last step refer to the total
c                              number of steps (1, ... , cal_IntSteps).
c
c      o  cal_DaysForMonth   - Given the integration month return the number
c                              of days to be integrated, the first day,
c                              and the last day in the month specified. The
c                              first and the last day refer to the total
c                              number of steps (1, ... , cal_IntDays).
c
c      o  cal_MonthsForYear  - Given the integration year return the number
c                              of months to be integrated, the first month,
c                              and the last month in the year specified. The
c                              first and the last step refer to the total
```

```
c                            number of steps (1, ... , cal_IntMonths).
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.
```

Experiments and tutorials that use cal

- Global ocean experiment in global_with_exf verification directory.
- Labrador Sea experiment in lab_sea verification directory.

Atmosphere Packages

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 ().

Key subroutines, parameters and files

AIM Diagnostics

```
------------------------------------------------------------------------
<-Name->|Levs|<-parsing code->|<--  Units    -->|<- Tile (max=80c)
------------------------------------------------------------------------
DIABT   |  5 |SM      ML      |K/s              |Pot. Temp.  Tendency (Mass-Weighted)␣
↪from Diabatic Processes
```

```
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
```

Experiments and tutorials that use aim

- Global atmosphere experiment in aim.5l_cs verification directory.

Land package

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

Equations and Key Parameters

This is a simple 2-layer land model. The top layer depth $z1 = 0.1m$, the second layer depth $z2 = 4m$.

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 capapcity per meter soil, so $f_1 = 0.024m$, $f_2 = 0.96m$.

The land temperature is determined by total surface downward heat flux F,

$$z_1 C_1 \frac{dT_{g1}}{dt} = F - \lambda \frac{T_{g1} - T_{g2}}{(z_1 + z_2)/2}$$

$$z_2 C_2 \frac{dT_{g2}}{dt} = \lambda \frac{T_{g1} - T_{g2}}{(z_1 + z_2)/2}$$

here C_1, C_2 are the heat capacity of each layer , λ‘$is the thermal conductivity, : math$:lambda =0.42Wm^{-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 $R(m/s)$.

$$\frac{dW_1}{dt} = \frac{P - E - R}{f_1} + \frac{W_2 - W_1}{\tau}$$

$\tau = 2\ days$ 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 equalibrium offline.

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)
```

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

Experiments and tutorials that use land

- Global atmosphere experiment in aim.5l_cs verification directory.

Fizhi: High-end Atmospheric Physics

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).

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 bouyancy 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, $s = c_pT + gz$, and the moist static energy, $h = c_pT + gz + Lq$. 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:

$$\eta(z)z = \lambda \quad or \quad \eta(P^\kappa)P^\kappa = -\frac{c_p}{g}\theta\lambda$$

where we have used the hydrostatic equation written in the form:

$$zP^\kappa = -\frac{c_p}{g}\theta$$

The entrainment parameter, λ, characterizes a particular subensemble based on its detrainment level, and is obtained by assuming that the level of detrainment is the level of neutral buoyancy, 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 detrainment level.

The convective instability is measured in terms of the cloud work function A, defined as the rate of change of cumulus kinetic energy. The cloud work function is related to the buoyancy, or the difference between the moist static energy in the cloud and in the environment:

$$A = \int_{P_D}^{P_B}\frac{\eta}{1+\gamma}\left[\frac{h_c - h^*}{P^\kappa}\right]dP^\kappa$$

where γ is $\frac{L}{c_p}q^*T$ 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:

$$m_B = \frac{-\frac{dA}{dt}\big|_{ls}}{K}$$

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 equillibrated 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):

$$\theta t|_c = \alpha\frac{m_B}{c_pP^\kappa}\eta sp$$

and

$$qt|_c = \alpha\frac{m_B}{L}\eta(hp - sp)$$

$$where : math : `\theta = \frac{T}{P^\kappa}`, : math : `P = (p/p_0)`, and$$

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 equillibrium.

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 fractons which are used for cloud-radiative interactions are determined diagnostically as part of the cumulus and large-scale parameterizations. Convective cloud fractions produced by RAS are proportional to the detrained liquid water amount given by

$$F_{RAS} = \min\left[\frac{l_{RAS}}{l_c}, 1.0\right]$$

where l_c is an assigned critical value equal to 1.25 g/kg. A memory is associated with convective clouds defined by:

$$F^n_{RAS} = \min\left[F_{RAS} + (1 - \frac{\Delta t_{RAS}}{\tau})F^{n-1}_{RAS}, 1.0\right]$$

where F_{RAS} is the instantanious cloud fraction and F^{n-1}_{RAS} is the cloud fraction from the previous RAS timestep. The memory coefficient is computed using a RAS cloud timescale, 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-+2 s)r s & = & p/p$_{surf}$ r & = & ( ) RH$_{min}$ & = & 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 5.9

The total cloud fraction in a grid box is determined by the larger of the two cloud fractions:

$$F_{CLD} = \max\left[F_{RAS}, F_{LS}\right].$$

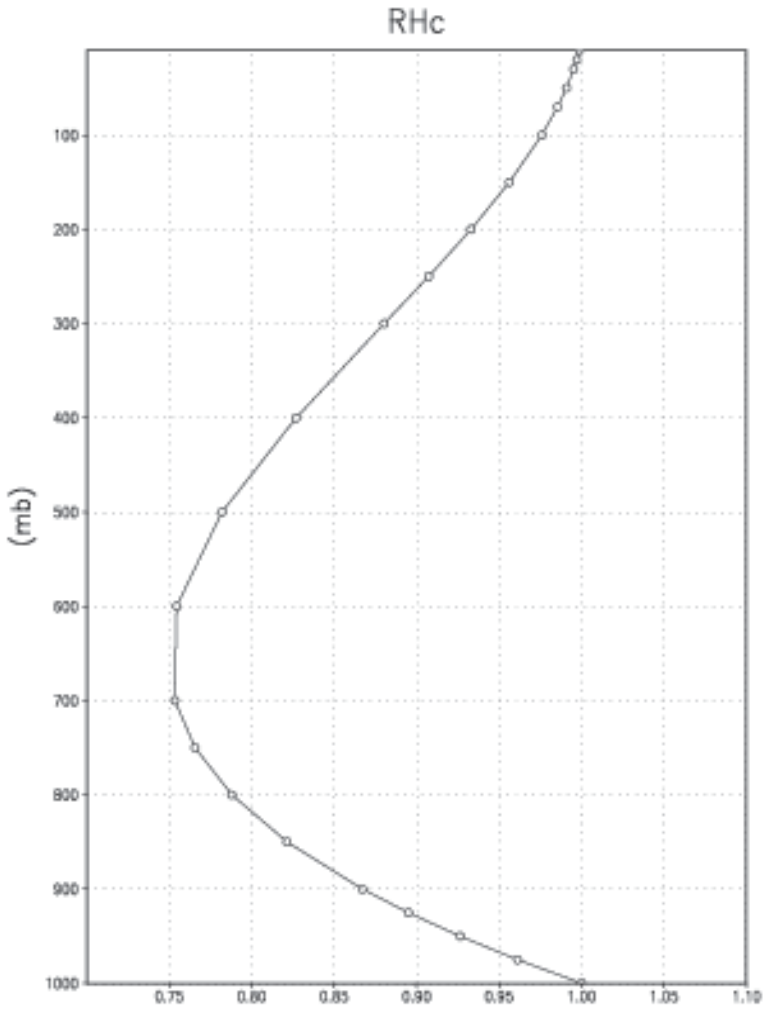

Figure 5.9: Critical Relative Humidity for Clouds.

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

$$\rho c_p Tt = -Fz$$

or

$$Tt = \frac{g}{c_p \pi} F\sigma$$

$$where : math : `g`istheaccelationduetogravityand : math : `c_p`isthe$$

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 absoption 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 5.10, 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 5.11. 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 5.10: 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 5.11: 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 paticles 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 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 5.12.

Table 5.12: IR Spectral Bands, Absorbers, and Parameterization Method (from [CMJSuarez94])

IR Spectral Bands			
Band	Spectral Range (cm^{-1})	Absorber	Method
1	0-340	H_2O line	T
2	340-540	H_2O line	T
3a	540-620	H_2O line	K
3b	620-720	H_2O continuum	S
3b	720-800	CO_2	T
4	800-980	H_2O line	K
		H_2O continuum	S
		H_2O line	K
5	980-1100	H_2O continuum	S
		O_3	T
6	1100-1380	H_2O line	K
		H_2O continuum	S
7	1380-1900	H_2O line	T
8	1900-3000	H_2O line	K
K: k-distribution method with linear pressure scaling			
T: Table look-up with temperature and pressure scaling			
S: One-parameter temperature scaling			

The longwave radiation package accurately computes cooling rates for the middle and lower atmosphere from 0.01 mb to the surface. Errors are < 0.4 C day^{-1} in cooling rates and $< 1\%$ in fluxes. From Chou and Suarez, it is estimated that the total effect of neglecting all minor absorption bands and the effects of minor infrared absorbers such as nitrous oxide (N:math:_2O), methane (CH:math:_4), and the chlorofluorocarbons (CFCs), is an underestimate of ≈ 5 W/m^2 in the downward flux at the surface and an overestimate of ≈ 3 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 catagorized as low/middle/high. The net clear line-of-site probability (P) between any two levels, p_1 and $p_2 \quad (p_2 > p_1)$, 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

$$0.0002 \leq \tau_{ice}(mb^{-1}) \leq 0.002 \quad \text{for} \quad 0 \leq \ell \leq 2 \quad \text{mg/kg},$$

$$0.02 \leq \tau_{h_2o}(mb^{-1}) \leq 0.2 \quad \text{for} \quad 0 \leq \ell \leq 10 \quad \text{mg/kg}.$$

The partitioning, α, between ice particles and water droplets is achieved through a linear scaling in temperature:

$$0 \leq \alpha \leq 1 \quad \text{for} \quad 233.15 \leq T \leq 253.15.$$

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:

$$ut_{turb} = z(-\overline{u'w'}) = z(K_m uz)$$

$$vt_{turb} = z(-\overline{v'w'}) = z(K_m vz)$$

$$Tt = P^\kappa \theta t_{turb} = P^\kappa z(-\overline{w'\theta'}) = P^\kappa z(K_h \theta_v z)$$

$$qt_{turb} = z(-\overline{w'q'}) = z(K_h qz)$$

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),

$$q^2 = \overline{u'^2} + \overline{v'^2} + \overline{w'^2},$$

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:

$$t(q^2) - z(\frac{5}{3}\lambda_1 qz(q^2)) = -\overline{u'w'}Uz - \overline{v'w'}Vz + \frac{g}{\Theta_0}\overline{w'\theta_v'} - \frac{q^3}{\Lambda_1}$$

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, ${\Theta_0}^{-1}$ 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 statisically realizable level 2.5 turbulence scheme of *[HL88]*, these diffusion coefficients are expressed as

$$K_h = \begin{cases} q\,\ell\,S_H(G_M, G_H) & \text{for decaying turbulence} \\ \frac{q^2}{q_e}\,\ell\,S_H(G_{M_e}, G_{H_e}) & \text{for growing turbulence} \end{cases}$$

and

$$K_m = \begin{cases} q\,\ell\,S_M(G_M, G_H) & \text{for decaying turbulence} \\ \frac{q^2}{q_e}\,\ell\,S_M(G_{M_e}, G_{H_e}) & \text{for growing turbulence} \end{cases}$$

where the subscript e refers to the value under conditions of local equillibrium (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_{H_e} and G_{M_e}, are functions of the Richardson number:

$$\mathbf{RI} = \frac{\frac{g}{\theta_v}\theta_v z}{(uz)^2 + (vz)^2} = \frac{c_p \theta_v z P^\kappa z}{(uz)^2 + (vz)^2}.$$

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:

$$K_m(surface) = C_u \times u_* = C_D W_s$$

and

$$K_h(surface) = C_t \times u_* = C_H W_s$$

$$where : math : `u_* = C_u W_s` is the surface friction velocity, : math : `C_D`$$

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

$$\psi_m = \int_{\zeta_0}^{\zeta} \frac{\phi_m}{\zeta} d\zeta.$$

$$Here : math : `\zeta` is the non - dimensional stability parameter, and$$

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

$$\psi_h = \int_{\zeta_0}^{\zeta} \frac{\phi_h}{\zeta} d\zeta.$$

$$Here : math : `\phi_h` is the similarity function of : math : `\zeta`, which$$

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 mosstly 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. $h_0 = 30 z_0$ with a maximum value over land of 0.01

The surface roughness length over oceans is 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:

$$\phi_m{}^4 - 18\zeta\phi_m{}^3 = 1 \qquad ; \qquad \phi_h{}^2 - 18\zeta\phi_h{}^3 = 1 \qquad .$$

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 momemtum flux:

$$\phi_m = \frac{1 + 5\zeta_1}{1 + 0.00794\zeta_1(1 + 5\zeta_1)} \qquad ; \qquad \phi_h = \frac{1 + 5\zeta_1}{1 + 0.00794\zeta(1 + 5\zeta_1)}.$$

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:

$$C_g T_g t = R_{sw} - R_{lw} + Q_{ice} - H - LE$$

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:

$$H = P^{\kappa} \rho c_p C_H W_s (\theta_{surface} - \theta_{NLAY}) \qquad where: \;\; C_H = C_u C_t$$

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

$$LE = \rho \beta L C_H W_s (q_{surface} - q_{NLAY}) \qquad where: \;\; C_H = C_u C_t$$

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 $2\pi \; radians/day$, 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 5.13. 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 5.10. The determination of the land or sea category of surface type was made from NCAR's 10 minute by 10 minute Navy topography dataset, which includes information about the percentage of water-cover at any point. The data were averaged to the model's grid resolutions, and any grid-box whose averaged water percentage was $\geq 60\%$ 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 5.13: 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

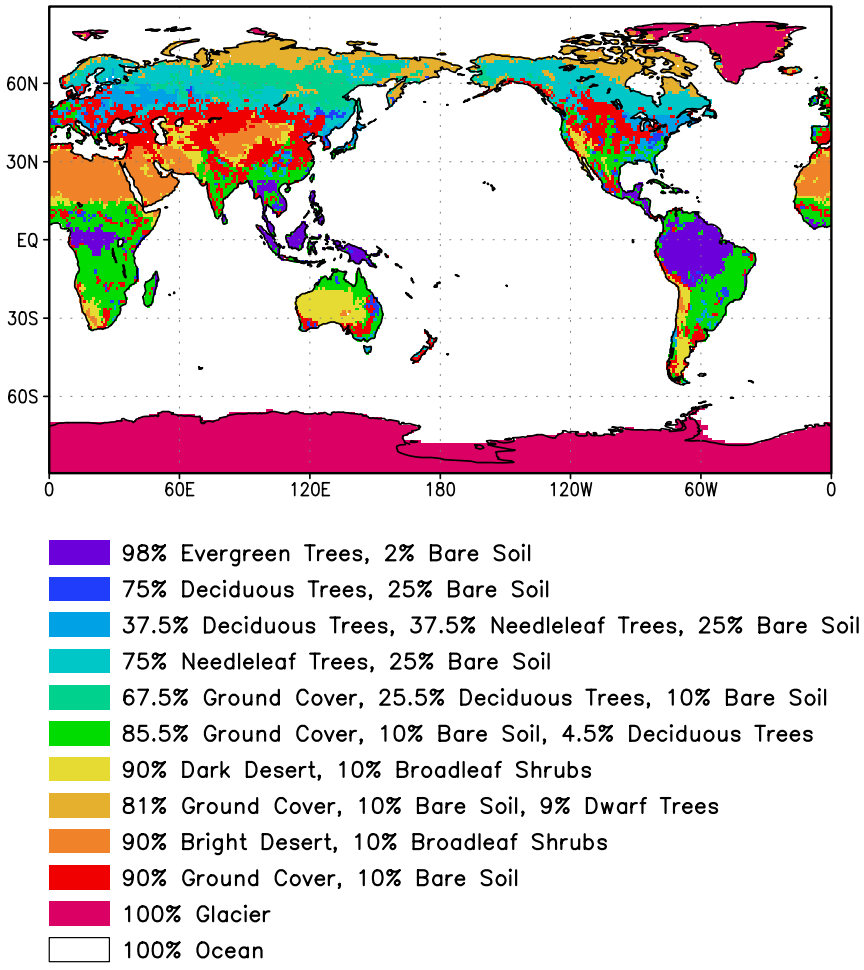

Figure 5.10: Surface type combinations

Surface Roughness

The surface roughness length over oceans is computed iteratively with the wind stress by the surface layer parameterization *[HS95]*. It employs an interpolation between the functions of *[LP81]* for high winds and of *[Kon75]* for weak winds.

Albedo

The surface albedo computation, described in , employs the "two stream" approximation used in Sellers' (1987) Simple Biosphere (SiB) Model which distinguishes between the direct and diffuse albedos in the visible and in the near infrared spectral ranges. The albedos are functions of the observed leaf area index (a description of the relative orientation of the leaves to the sun), the greenness fraction, the vegetation type, and the solar zenith angle. Modifications are made to account for the presence of snow, and its depth relative to the height of the vegetation elements.

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.

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 estent, 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 5.14. The time mean values are interpolated during each model timestep to the current time.

Table 5.14: 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.

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	m^2 s^{-1}	Nrphys	Diffusivity coefficient for T and Q
EU	m^2 s^{-1}	Nrphys	Diffusivity coefficient for U and V
TURBU	m s^{-1} day^{-1}	Nrphys	U-Momentum Changes due to Turbulence
TURBV	m s^{-1} 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	LEVELS	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
LWGDOWN	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
TAUCLDC	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 u-Wind

NAME	UNITS	LEVELS	DESCRIPTION
VAVE	m/sec	Nrphys	Time-averaged v-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 u-Wind
DIABV	m/sec/day	Nrphys	Total Diabatic forcing on v-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
GWDU	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)

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:

$$\mathbf{DIAGNOSTIC} = \frac{1}{TTOT} \sum_{t=1}^{t=TTOT} diag(t)$$

where $TTOT = \frac{\mathbf{NQDIAG}}{\Delta t}$, **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 ($Newton/m^2$)

The zonal wind stress is the turbulent flux of zonal momentum from the surface.

$$\mathbf{UFLUX} = -\rho C_D W_s u \qquad where: \quad C_D = C_u^2$$

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 ($Newton/m^2$)

The meridional wind stress is the turbulent flux of meridional momentum from the surface.

$$\mathbf{VFLUX} = -\rho C_D W_s v \qquad where: \quad C_D = C_u^2$$

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:

$$\mathbf{HFLUX} = P^{\kappa}\rho c_p C_H W_s(\theta_{surface} - \theta_{Nrphys}) \qquad where: \quad C_H = C_u C_t$$

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:

$$\mathbf{EFLUX} = \rho\beta L C_H W_s(q_{surface} - q_{Nrphys}) \qquad where: \quad C_H = C_u C_t$$

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.

$$\mathbf{QICE} = \frac{C_{ti}}{H_i}(T_i - T_g)$$

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 ($Watts/m^2$)

$$\begin{aligned} \mathbf{RADLWG} &= F^{Net}_{LW,Nrphys+1} \\ &= F^{\uparrow}_{LW,Nrphys+1} - F^{\downarrow}_{LW,Nrphys+1} \end{aligned}$$

where Nrphys+1 indicates the lowest model edge-level, or $p = p_{surf}$. $F^{\uparrow}_{LW}$ is the upward Longwave flux and $F^{\downarrow}_{LW}$ is the downward Longwave flux.

Net downard shortwave Flux at the surface ($Watts/m^2$)

$$\begin{aligned} \mathbf{RADSWG} = & \quad F^{Net}_{SW,Nrphys+1} \\ = & \quad F^{\downarrow}_{SW,Nrphys+1} - F^{\uparrow}_{SW,Nrphys+1} \end{aligned}$$

where Nrphys+1 indicates the lowest model edge-level, or $p = p_{surf}$. $F^{\downarrow}_{SW}$ is the downward Shortwave flux and $F^{\uparrow}_{SW}$ is the upward Shortwave flux.

Richardson number ($dimensionless$)

The non-dimensional stability indicator is the ratio of the buoyancy to the shear:

$$\mathbf{RI} = \frac{\frac{g}{\theta_v}\theta_v z}{(uz)^2 + (vz)^2} = \frac{c_p \theta_v z P^\kappa z}{(uz)^2 + (vz)^2}$$

where we used the hydrostatic equation:

$$\Phi P^\kappa = c_p \theta_v$$

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:

$$\mathbf{CT} = -\frac{(\overline{w'\theta'})}{u_* \Delta\theta} = -\frac{(\overline{w'q'})}{u_* \Delta q} = \frac{k}{(\psi_h + \psi_g)}$$

where ψ_h is the surface layer non-dimensional temperature change and ψ_g is the viscous sublayer non-dimensional temperature or moisture change:

$$\psi_h = \int_{\zeta_0}^{\zeta} \frac{\phi_h}{\zeta} d\zeta \qquad and \qquad \psi_g = \frac{0.55(Pr^{2/3} - 0.2)}{\nu^{1/2}} (h_0 u_* - h_{0_{ref}} u_{*_{ref}})^{1/2}$$

and: $h_0 = 30 z_0$ with a maximum value over land of 0.01

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:

$$\mathbf{CU} = \frac{u_*}{W_s} = \frac{k}{\psi_m}$$

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 (m^2/sec)

In the level 2.5 version of the Mellor-Yamada (1974) hierarchy, the turbulent heat or moisture flux for the atmosphere above the surface layer can be expressed as a turbulent diffusion coefficient K_h times the negative of the gradient of potential temperature or moisture. In the *[HL88]* adaptation of this closure, K_h takes the form:

$$\mathbf{ET} = K_h = -\frac{(\overline{w'\theta'_v})}{\theta_v z} = \begin{cases} q\,\ell\,S_H(G_M, G_H) & \text{for decaying turbulence} \\ \frac{q^2}{q_e}\,\ell\,S_H(G_{M_e}, G_{H_e}) & \text{for growing turbulence} \end{cases}$$

where q is the turbulent velocity, or $\sqrt{2 * turbulent\ \ kinetic\ \ energy}$, 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_{H_e} and G_{M_e}, the equilibrium dimensionless buoyancy and wind shear parameters. Both G_H and G_M, and their equilibrium values G_{H_e} and G_{M_e}, are functions of the Richardson number.

For the detailed equations and derivations of the modified level 2.5 closure scheme, see *[HL88]*.

In the surface layer, **ET** is the exchange coefficient for heat and moisture, in units of m/sec, given by:

$$\mathbf{ET_{Nrphys}} = C_t * u_* = C_H W_s$$

where C_t is the dimensionless exchange coefficient for heat and moisture from the surface layer similarity functions (see diagnostic number 9), u_* is the surface friction velocity (see diagnostic number 67), C_H is the heat transfer coefficient, and W_s is the magnitude of the surface layer wind.

EU - Diffusivity Coefficient for Momentum (m^2/sec)

In the level 2.5 version of the Mellor-Yamada (1974) hierarchy, the turbulent heat momentum flux for the atmosphere above the surface layer can be expressed as a turbulent diffusion coefficient K_m times the negative of the gradient of the u-wind. In the *[HL88]* adaptation of this closure, K_m takes the form:

$$\mathbf{EU} = K_m = -\frac{(\overline{u'w'})}{Uz} = \begin{cases} q\,\ell\,S_M(G_M, G_H) & \text{for decaying turbulence} \\ \frac{q^2}{q_e}\,\ell\,S_M(G_{M_e}, G_{H_e}) & \text{for growing turbulence} \end{cases}$$

where q is the turbulent velocity, or $\sqrt{2 * turbulent\ \ kinetic\ \ energy}$, 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_{H_e} and G_{M_e}, the equilibrium dimensionless buoyancy and wind shear parameters. Both G_H and G_M, and their equilibrium values G_{H_e} and G_{M_e}, are functions of the Richardson number.

For the detailed equations and derivations of the modified level 2.5 closure scheme, see *[HL88]*.

In the surface layer, **EU** is the exchange coefficient for momentum, in units of m/sec, given by:

$$\mathbf{EU_{Nrphys}} = C_u * u_* = C_D W_s$$

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:

$$\mathbf{TURBU} = ut_{turb} = z(-\overline{u'w'}) = z(K_m uz)$$

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:

$$\mathbf{TURBV} = vt_{turb} = z(-\overline{v'w'}) = z(K_m vz)$$

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:

$$\mathbf{TURBT} = Tt = P^\kappa \theta t_{turb} = P^\kappa z(-\overline{w'\theta'}) = P^\kappa z(K_h \theta_v z)$$

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:

$$\mathbf{TURBQ} = qt_{turb} = z(-\overline{w'q'}) = z(K_h qz)$$

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)

$$\mathbf{MOISTT} = Tt|_c + Tt|_{ls}$$

where:

$$Tt|_c = R\sum_i \left(\alpha \frac{m_B}{c_p}\Gamma_s\right)_i \quad and \quad Tt|_{ls} = \frac{L}{c_p}(q^* - q)$$

$$and$$

$$\Gamma_s = g\eta sp$$

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 5.5.3.2, 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)

$$\mathbf{MOISTQ} = qt|_c + qt|_{ls}$$

where:

$$qt|_c = R\sum_i \left(\alpha \frac{m_B}{L}(\Gamma_h - \Gamma_s)\right)_i \quad and \quad qt|_{ls} = (q^* - q)$$

and

$$\Gamma_s = g\eta sp \quad and \quad \Gamma_h = g\eta hp$$

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 5.5.3.2, 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:

$$\rho c_p Tt = -zF_{LW}^{NET},$$

or

$$\mathbf{RADLW} = \frac{g}{c_p \pi}\sigma F_{LW}^{NET}.$$

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:

$$\rho c_p Tt = -zF(cloudy)_{SW}^{NET} \cdot \mathrm{RADSWT},$$

or

$$\mathbf{RADSW} = \frac{g}{c_p \pi} \sigma F(cloudy)_{SW}^{NET} \cdot \mathrm{RADSWT}.$$

where g is the accelation 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

$$F(cloudy)_{SW}^{Net} = F(cloudy)_{SW}^{\uparrow} - F(cloudy)_{SW}^{\downarrow}$$

PREACC - Total (Large-scale + Convective) Accumulated Precipition (mm/day)

For a change in specific humidity due to moist processes, Δq_{moist}, the vertical integral or total precipitable amount is given by:

$$\mathbf{PREACC} = \int_{surf}^{top} \rho \Delta q_{moist} dz = -\int_{surf}^{top} \Delta q_{moist} \frac{dp}{g} = \frac{1}{g} \int_0^1 \Delta q_{moist} dp$$

A precipitation rate is defined as the vertically integrated moisture adjustment per Moist Processes time step, scaled to mm/day.

PRECON - Convective Precipition (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:

$$\mathbf{PRECON} = \int_{surf}^{top} \rho \Delta q_{cum} dz = -\int_{surf}^{top} \Delta q_{cum} \frac{dp}{g} = \frac{1}{g} \int_0^1 \Delta q_{cum} dp$$

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 :math:‘diagnostic hspace{.2cm} purposes

hspace{.2cm} only‘ from the eddy coefficient for momentum:

$$\mathbf{TUFLUX} = \rho(\overline{u'w'}) = \rho(-K_m U z)$$

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:

$$\mathbf{TVFLUX} = \rho(\overline{v'w'}) = \rho(-K_m V z)$$

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:

$$\mathbf{TTFLUX} = c_p \rho P^\kappa(\overline{w'\theta'}) = c_p \rho P^\kappa(-K_h \theta_v z)$$

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:

$$\mathbf{TQFLUX} = L\rho(\overline{w'q'}) = L\rho(-K_h q z)$$

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:

$$\mathbf{WINDS} = \sqrt{u^2_{Nrphys} + v^2_{Nrphys}}$$

where the subscript $Nrphys$ refers to the lowest model level.

The air/surface virtual temperature difference measures the stability of the surface layer:

$$\mathbf{DTSRF} = (\theta_{vNrphys+1} - \theta v_{Nrphys}) P^\kappa_{surf}$$

where

$$\theta_{vNrphys+1} = \frac{T_g}{P^\kappa_{surf}}(1 + .609 q_{Nrphys+1}) \qquad and \qquad q_{Nrphys+1} = q_{Nrphys} + \beta(q^*(T_g, P_s) - q_{Nrphys})$$

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:

$$\mathbf{TG}\ is\ obtained\ from:\ C_g T_g t = R_{sw} - R_{lw} + Q_{ice} - H - LE$$

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:

$$C_g = \sqrt{\frac{\lambda C_s}{2\omega}} = \sqrt{(0.386 + 0.536W + 0.15W^2)2x10^{-3}\frac{86400.}{2\pi}}\ .$$

Here, the thermal conductivity, λ, is equal to $2x10^{-3}\ \frac{ly}{sec}\frac{cm}{K}$, the angular velocity of the earth, ω, is written as $86400\ sec/day$ divided by $2\pi\ radians/day$, 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:

$$\mathbf{TS} = \theta_{Nrphys} P_{surf}^{\kappa}$$

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:

$$\mathbf{DTG} = {T_g}^n - {T_g}^{n-1}$$

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:

$$\mathbf{QG} = q_{Nrphys+1} = q_{Nrphys} + \beta(q^*(T_g, P_s) - q_{Nrphys})$$

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 temprature and surface pressure:

$$\mathbf{QS} = q^*(T_g, P_s)$$

TGRLW - Instantaneous ground temperature used as input to the Longwave radiation subroutine (deg)

$$\mathbf{TGRLW} = T_g(\lambda, \phi, n)$$

where T_g is the model ground temperature at the current time step n.

ST4 - Upward Longwave flux at the surface (Watts/m^2)

$$\mathbf{ST4} = \sigma T^4$$

where σ is the Stefan-Boltzmann constant and T is the temperature.

OLR - Net upward Longwave flux at $p = p_{top}$ (Watts/m^2)

$$\mathbf{OLR} = F^{NET}_{LW,top}$$

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)

$$\mathbf{OLRCLR} = F(clearsky)^{NET}_{LW,top}$$

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)

$$\begin{aligned} \mathbf{LWGCLR} = & \quad F(clearsky)^{Net}_{LW,Nrphys+1} \\ = & \quad F(clearsky)^{\uparrow}_{LW,Nrphys+1} - F(clearsky)^{\downarrow}_{LW,Nrphys+1} \end{aligned}$$

where Nrphys+1 indicates the lowest model edge-level, or $p = p_{surf}$. $F(clearsky)^{\uparrow}_{LW}$ is the upward clearsky Longwave flux and the $F(clearsky)^{\downarrow}_{LW}$ 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, $F^{clearsky}_{LW}$, 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:

$$\rho c_p T t_{clearsky} = -zF(clearsky)_{LW}^{NET},$$

or

$$\mathbf{LWCLR} = \frac{g}{c_p \pi} \sigma F(clearsky)_{LW}^{NET}.$$

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)

$$\mathbf{OZLW} = \mathrm{OZ}(\lambda, \phi, level, n)$$

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].

$$\mathbf{CLDTOT} = F_{RAS} + F_{LS}$$

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].

$$\mathbf{CLMOSW} = CLMO_{RAS,SW}(\lambda, \phi, level)$$

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].

$$\mathbf{CLROSW} = CLRO_{RAS,LargeScale,SW}(\lambda, \phi, level)$$

RADSWT - Incident Shortwave radiation at the top of the atmosphere (Watts/m^2)

$$\mathbf{RADSWT} = \frac{S_0}{R_a^2} \cdot cos\phi_z$$

where S_0, is the extra-terrestial solar contant, 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:

$$\mathbf{EVAP} = \rho \beta K_h (q_{surface} - q_{Nrphys})$$

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 $qsurface$ 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.

$$\mathbf{DUDT} = ut_{Dynamics} + ut_{Moist} + ut_{Turbulence} + ut_{Analysis}$$

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.

$$\mathbf{DVDT} = vt_{Dynamics} + vt_{Moist} + vt_{Turbulence} + vt_{Analysis}$$

DTDT - Total Temperature Tendency (deg/day)

DTDT is the total time-tendency of Temperature due to Hydrodynamic, Diabatic, and Analysis forcing.

$$\begin{aligned}\mathbf{DTDT} = \quad & Tt_{Dynamics} + Tt_{MoistProcesses} + Tt_{ShortwaveRadiation} \\ + \quad & Tt_{LongwaveRadiation} + Tt_{Turbulence} + Tt_{Analysis}\end{aligned}$$

DQDT - Total Specific Humidity Tendency (g/kg/day)

DQDT is the total time-tendency of Specific Humidity due to Hydrodynamic, Diabatic, and Analysis forcing.

$$\mathbf{DQDT} = qt_{Dynamics} + qt_{MoistProcesses} + qt_{Turbulence} + qt_{Analysis}$$

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:

$$\mathbf{USTAR} = C_u W_s \qquad where: \quad C_u = \frac{k}{\psi_m}$$

C_u is the non-dimensional surface drag coefficient (see diagnostic number 10), and W_s is the surface wind speed (see diagnostic number 28).

Z0 - Surface Roughness Length (m)

Over the land surface, the surface roughness length is interpolated to the local time from the monthly mean data of . 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 $0.005\ m^2/sec^2$. 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.

$$\mathbf{PBL} = P_{PBL} - P_{surface}$$

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:

$$\rho c_p Tt = -zF(clear)_{SW}^{NET} \cdot \mathrm{RADSWT},$$

or

$$\mathbf{SWCLR} = \frac{g}{c_p} pF(clear)_{SW}^{NET} \cdot \mathrm{RADSWT}.$$

where g is the accelation 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

$$F(clear)_{SW}^{Net} = F(clear)_{SW}^{\uparrow} - F(clear)_{SW}^{\downarrow}$$

OSR - Net upward Shortwave flux at the top of the model (Watts/m^2)

$$\mathbf{OSR} = F_{SW,top}^{NET}$$

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)

$$\mathbf{OSRCLR} = F(clearsky)_{SW,top}^{NET}$$

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/m^2)

The amount of cloud mass moved per RAS timestep from all convective clouds is written:

$$\mathbf{CLDMAS} = \eta m_B$$

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 5.5.3.2, 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.

$$\mathbf{UAVE} = u(\lambda, \phi, level, t)$$

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.

$$\mathbf{VAVE} = v(\lambda, \phi, level, t)$$

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.

$$\mathbf{TAVE} = T(\lambda, \phi, level, t)$$

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.

$$\mathbf{QAVE} = q(\lambda, \phi, level, t)$$

PAVE - Time-Averaged Surface Pressure - PTOP (mb)

The diagnostic **PAVE** is simply the time-averaged Surface Pressure - PTOP over the **NPAVE** output frequency. This is contrasted to the instantaneous Surface Pressure - PTOP which is archived on the Prognostic Output data stream.

$$\begin{aligned} \mathbf{PAVE} = & \quad \pi(\lambda, \phi, level, t) \\ = & \quad p_s(\lambda, \phi, level, t) - p_T \end{aligned}$$

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)

$$\begin{aligned}\mathbf{SWGCLR} = & \quad F(clearsky)^{Net}_{SW,Nrphys+1} \\ = & \quad F(clearsky)^{\downarrow}_{SW,Nrphys+1} - F(clearsky)^{\uparrow}_{SW,Nrphys+1}\end{aligned}$$

where Nrphys+1 indicates the lowest model edge-level, or $p = p_{surf}$. $F(clearsky)SW^{\downarrow}$ is the downward clearsky Shortwave flux and $F(clearsky)^{\uparrow}_{SW}$ 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.

$$\mathbf{DIABU} = ut_{Moist} + ut_{Turbulence} + ut_{Analysis}$$

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.

$$\mathbf{DIABV} = vt_{Moist} + vt_{Turbulence} + vt_{Analysis}$$

DIABT Total Diabatic Temperature Tendency (deg/day)

DIABT is the total time-tendency of Temperature due to Diabatic processes and the Analysis forcing.

$$\begin{aligned}\mathbf{DIABT} = & \quad Tt_{MoistProcesses} + Tt_{ShortwaveRadiation} \\ + & \quad Tt_{LongwaveRadiation} + Tt_{Turbulence} + Tt_{Analysis}\end{aligned}$$

If we define the time-tendency of Temperature due to Diabatic processes as

$$\begin{aligned}Tt_{Diabatic} = & \quad Tt_{MoistProcesses} + Tt_{ShortwaveRadiation} \\ + & \quad Tt_{LongwaveRadiation} + Tt_{Turbulence}\end{aligned}$$

then, since there are no surface pressure changes due to Diabatic processes, we may write

$$Tt_{Diabatic} = \frac{p^{\kappa}}{\pi}\pi\theta t_{Diabatic}$$

where $\theta = T/p^{\kappa}$. Thus, **DIABT** may be written as

$$\mathbf{DIABT} = \frac{p^{\kappa}}{\pi}\left(\pi\theta t_{Diabatic} + \pi\theta t_{Analysis}\right)$$

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.

$$\mathbf{DIABQ} = qt_{MoistProcesses} + qt_{Turbulence} + qt_{Analysis}$$

If we define the time-tendency of Specific Humidity due to Diabatic processes as

$$qt_{Diabatic} = qt_{MoistProcesses} + qt_{Turbulence}$$

then, since there are no surface pressure changes due to Diabatic processes, we may write

$$qt_{Diabatic} = \frac{1}{\pi}\pi qt_{Diabatic}$$

$$Thus, **DIABQ**maybewrittenas$$

$$\mathbf{DIABQ} = \frac{1}{\pi}\left(\pi qt_{Diabatic} + \pi qt_{Analysis}\right)$$

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

$$\mathbf{VINTUQ} = \int_0^1 uqdp$$

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

$$\mathbf{VINTVQ} = \int_0^1 vqdp$$

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.

$$\mathbf{VINTUT} = \frac{\int_{surf}^{top} uT\rho dz}{\int_{surf}^{top} \rho dz}$$

Or,

$$\mathbf{VINTUT} = \int_0^1 uTdp$$

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.

$$\mathbf{VINTVT} = \frac{\int_{surf}^{top} vT\rho dz}{\int_{surf}^{top} \rho dz}$$

Using :math:‘rho delta z = -frac{delta p}{g} ‘, we have

$$\mathbf{VINTVT} = \int_0^1 vTdp$$

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.

$$1 - MAX_p^{p'} \left(CLMO_p\right)$$

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.

$$\prod_p^{p'} \left(1 - CLRO_p\right)$$

The total cloud fraction at a given level associated with a line- of-site calculation is given by

$$1 - \left(1 - MAX_p^{p'} \left[CLMO_p\right]\right) \prod_p^{p'} \left(1 - CLRO_p\right)$$

The 2-dimensional net cloud fraction as seen from the top of the atmosphere is given by

$$\mathbf{CLDFRC} = 1 - \left(1 - MAX_{l=l_1}^{Nrphys} \left[CLMO_l\right]\right) \prod_{l=l_1}^{Nrphys} \left(1 - CLRO_l\right)$$

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:

$$\begin{aligned} \mathbf{QINT} = & \int_{surf}^{top} \rho q dz \\ = & \frac{\pi}{g} \int_0^1 q dp \end{aligned}$$

where we have used the hydrostatic relation :math:‘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:

$$\mathbf{U2M} = \frac{u_*}{k} \psi_{m_{2m}} \frac{u_{sl}}{W_s} = \frac{\psi_{m_{2m}}}{\psi_{m_{sl}}} u_{sl}$$

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, $\mathbf{U2M}$ 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:

$$\mathbf{V2M} = \frac{u_*}{k} \psi_{m_{2m}} \frac{v_{sl}}{W_s} = \frac{\psi_{m_{2m}}}{\psi_{m_{sl}}} v_{sl}$$

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, $\mathbf{V2M}$ is undefined.

T2M - Temperature at 2 Meter Depth (deg K)

The temperature at the 2-meter depth is a determined from the similarity theory:

$$\mathbf{T2M} = P^\kappa (\frac{\theta *}{k} (\psi_{h_{2m}} + \psi_g) + \theta_{surf}) = P^\kappa (\theta_{surf} + \frac{\psi_{h_{2m}} + \psi_g}{\psi_{h_{sl}} + \psi_g} (\theta_{sl} - \theta_{surf}))$$

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, $\mathbf{T2M}$ 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:

$$\mathbf{Q2M} = P^\kappa \frac{(}{q_*} k(\psi_{h_{2m}} + \psi_g) + q_{surf}) = P^\kappa (q_{surf} + \frac{\psi_{h_{2m}} + \psi_g}{\psi_{h_{sl}} + \psi_g} (q_{sl} - q_{surf}))$$

where:

$$q_* = -\frac{(\overline{w'q'})}{u_*}$$

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, $\mathbf{Q2M}$ 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:

$$\mathbf{T10M} = P^\kappa(\frac{\theta*}{k}(\psi_{h_{10m}} + \psi_g) + \theta_{surf}) = P^\kappa(\theta_{surf} + \frac{\psi_{h_{10m}} + \psi_g}{\psi_{h_{sl}} + \psi_g}(\theta_{sl} - \theta_{surf}))$$

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:

$$\mathbf{Q10M} = P^\kappa(\frac{q_*}{k}(\psi_{h_{10m}} + \psi_g) + q_{surf}) = P^\kappa(q_{surf} + \frac{\psi_{h_{10m}} + \psi_g}{\psi_{h_{sl}} + \psi_g}(q_{sl} - q_{surf}))$$

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 5.5.3.2.

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:

$$\mathbf{QFILL} = q^{n+1}_{final} - q^{n+1}_{initial}$$

where

$$q^{n+1} = (\pi q)^{n+1}/\pi^{n+1}$$

Key subroutines, parameters and files

Dos and don'ts

Fizhi Reference

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.

Sea Ice Packages

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.

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 tendancy 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:

$$\begin{aligned} q_1 = & \quad -c_i * T_f + L_i \\ q_2 = & \quad -c_f T_{mlt} + c_i(T_{mlt} - Tf) + L_i(1 - \frac{T_{mlt}}{T_f}) \end{aligned}$$

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

$$h_{inew} = \frac{\mathbf{esurp}\Delta t}{qi_{0av}}$$

where $qi_{0av} = -\frac{\rho_i}{2}(q_1 + q_2)$.

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 orginal ice due to freezing water.)

subroutine ICE_THERM

The main subroutine of this package is *ice_therm.F* where the ice temperatures are calculated and the changes in ice and snow thicknesses are determined. Output provides the net heat and fresh water fluxes that force the top layer of the ocean model.

If the current ice height is less than **himin** then the ice layer is set to zero and the ocean model upper layer temperature is allowed to drop lower than its freezing temperature; and atmospheric fluxes are allowed to effect the grid cell. If the ice height is greater than **himin** we proceed with the ice model calculation.

We follow the procedure of Winton (1999) – see equations 3 to 21 – to calculate the surface and internal ice temperatures. The surface temperature is found from the balance of the flux at the surface F_s, the shortwave heat flux absorbed by the ice, **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,

$$F_s = F_{sensible} + F_{latent} + F^{down}_{longwave} + F^{up}_{longwave} + (1 - \alpha)F_{shortwave}$$

and

$$\frac{dF_s}{dT} = \frac{dF_{sensible}}{dT} + \frac{dF_{latent}}{dT} + \frac{dF^{up}_{longwave}}{dT}.$$

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, *srf_albedo.F*) and the shortwave flux absorbed in the ice is

$$\mathbf{fswint} = (1 - e^{\kappa_i h_i})(1 - \alpha)F_{shortwave}$$

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):

$$\begin{aligned} q_1 = & \quad -c_f T_f + c_i(T_f - T) + L_i(1 - \frac{T_f}{T}) \\ q_2 = & \quad -c_i T + L_i \end{aligned}$$

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.

$$\begin{aligned} E_{top} = & \quad (F_s - K_{1/2}(T_s - T_1))\Delta t \\ E_{bot} = & \quad (\frac{4K_i(T_2 - T_f)}{h_i} - F_b)\Delta t \end{aligned}$$

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 $u^* = QQ$ 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^oC 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 converts any ice and/or snow above these limits back to water, maintaining the salt balance. Note however, that heat is not conserved in this conversion; sea surface temperatures below the ice are not recalculated.

If the snow/ice interface is below the waterline, snow is converted to ice (see Winton, 1999, equations 35 and 36). The subroutine *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:

$$q_{net} = \mathbf{fswocn} - F_b - \frac{\mathbf{esurp}}{\Delta t}$$

is composed of the shortwave flux that has passed through the ice layer and is absorbed by the water, **fswocn**$= QQ$, 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 aparts 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:

$$q_1 = f_1\hat{q_1} + (1 - f1)\hat{q_2}$$

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 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 Varibles, 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)

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).

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)
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)
```

```
SIflxAtm|   1 |SM      M1       |W/m^2            |net heat flux from the Atmosphere␣
↪(+=dw)
SIfrwAtm|   1 |SM      M1       |kg/m^2/s         |fresh-water flux to the Atmosphere␣
↪(+=up)
SIflx2oc|   1 |SM      M1       |W/m^2            |heat flux out of the ocean (+=up)
SIfrw2oc|   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
```

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.

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.

SEAICE Package

Authors: Martin Losch, Dimitris Menemenlis, An Nguyen, Jean-Michel Campin, Patrick Heimbach, Chris Hill and Jinlun Zhang

Introduction

Package "seaice" provides a dynamic and thermodynamic interactive sea-ice model.

CPP options enable or disable different aspects of the package (Section Section 5.6.2.2). Run-Time options, flags, filenames and field-related dates/times are set in data.seaice (Section Section 5.6.2.2). A description of key subroutines is given in Section Section 5.6.2.3. Input fields, units and sign conventions are summarized in Section [sec:pkg:seaice:fields:sub:*u*nits], and available diagnostics output is listed in Section [sec:pkg:seaice:diagnostics].

SEAICE configuration, compiling & running

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 [sec:buildingCode]).

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 5.15 summarizes the most important ones. For more options see the default `pkg/seaice/SEAICE_OPTIONS.h`.

Table 5.15: Some of the most relevant CPP preporocessor flags in the `seaice`-package.

CPP option	Description
`SEAICE_DEBUG`	Enhance STDOUT for debugging
`SEAICE_ALLOW_DYNAMICS`	sea-ice dynamics code
`SEAICE_CGRID`	LSR solver on C-grid (rather than original B-grid)
`SEAICE_ALLOW_EVP`	enable use of EVP rheology solver
`SEAICE_ALLOW_JFNK`	enable use of JFNK rheology solver
`SEAICE_EXTERNAL_FLUXES`	use EXF-computed fluxes as starting point
`SEAICE_ZETA_SMOOTHREG`	use differentialable regularization for viscosities
`SEAICE_VARIABLE_FREEZING_POINT`	enable linear dependence of the freezing point on salinity (by default undefined)
`ALLOW_SEAICE_FLOODING`	enable snow to ice conversion for submerged sea-ice
`SEAICE_VARIABLE_SALINITY`	enable sea-ice with variable salinity (by default undefined)
`SEAICE_SITRACER`	enable sea-ice tracer package (by default undefined)
`SEAICE_BICE_STRESS`	B-grid only for backward compatiblity: turn on ice-stress on ocean
`EXPLICIT_SSH_SLOPE`	B-grid only for backward compatiblity: use ETAN for tilt computations rather than geostrophic velocities

Run-time parameters

Run-time parameters (see Table 5.16) 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 5.16 lists most run-time parameters.

Table 5.16: 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
		Continu

Table 5.16 – continued from previous page

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
SEAICEuseEVPrev	F	use yet another variation on EVP*
SEAICEnEVPstarSteps	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 paramter 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
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 value)	density of air (kg/m:math:^3)
SEAICE_cpAir	1004 (or value)	specific heat of air (J/kg/K)
SEAICE_lhEvap	2,500,000 (or val ue)	latent heat of evaporation
SEAICE_lhFusion	334,000 (or value)	latent heat of fusion

Continu

Table 5.16 – continued from previous page

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 the l
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 advect.F
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*;

Description

[TO BE CONTINUED/MODIFIED]

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 compontent 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 implemtation can be selected with runtime flags.

Compatibility with ice-thermodynamics `thsice` package

Note, that by default the `seaice`-package includes the orginial 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/seaice`. The package `thsice` is described in section [sec:pkg:thsice]; 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

$$_{ocean} - m\nabla\phi(0) + \mathbf{F}, \quad (5.1)$$

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; **i**, **j**, and **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 accelation; $\nabla\phi(0)$ 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 :

$$\sigma_{ij} = 2\eta(\dot{\epsilon}_{ij}, P)\dot{\epsilon}_{ij} + [\zeta(\dot{\epsilon}_{ij}, P) - \eta(\dot{\epsilon}_{ij}, P)]\, \dot{\epsilon}_{kk}\delta_{ij} - \frac{P}{2}\delta_{ij}. \tag{5.2}$$

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:

$$P_{\max} = P^* c\, h\, \exp\{-C^* \cdot (1 - c)\},$$

with the constants P^* (run-time parameter `SEAICE_strength`) and $C^* = 20$. 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:

$$\begin{aligned}\zeta =& \min\left(\frac{P_{\max}}{2\max(\Delta, \Delta_{\min})}, \zeta_{\max}\right)\\ \eta =& \frac{\zeta}{e^2}\end{aligned}$$

with the abbreviation

$$\Delta = \left[\left(\dot{\epsilon}_{11}^2 + \dot{\epsilon}_{22}^2\right)\left(1 + e^{-2}\right) + 4e^{-2}\dot{\epsilon}_{12}^2 + 2\dot{\epsilon}_{11}\dot{\epsilon}_{22}(1 - e^{-2})\right]^{\frac{1}{2}}.$$

The bulk viscosities are bounded above by imposing both a minimum $\Delta_{\min}$ (for numerical reasons, run-time parameter `SEAICE_EPS` with a default value of $10^{-10}\,\mathrm{s}^{-1}$) and a maximum $\zeta_{\max} = P_{\max}/\Delta^*$, where $\Delta^* = (5 \times 10^{12}/2 \times 10^4)\,\mathrm{s}^{-1}$. (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:

$$\begin{aligned}\zeta &= \zeta_{\max} \tanh\left(\frac{P}{2\,\min(\Delta,\Delta_{\min})\,\zeta_{\max}}\right)\\ &= \frac{P}{2\Delta^*}\tanh\left(\frac{\Delta^*}{\min(\Delta,\Delta_{\min})}\right)\end{aligned}$$

where $\Delta_{\min} = 10^{-20}\,\mathrm{s}^{-1}$ is chosen to avoid divisions by zero.

LSR and JFNK solver

In the matrix notation, the discretized momentum equations can be written as

$$\mathbf{A}(\mathbf{x})\,\mathbf{x} = \mathbf{b}(\mathbf{x}). \tag{5.3}$$

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. (5.3) 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 $\mathbf{x}^{k-1}$:

$$\mathbf{F}(\mathbf{x}^{k-1} + \delta\mathbf{x}^k) = \mathbf{F}(\mathbf{x}^{k-1}) + \mathbf{F}'(\mathbf{x}^{k-1})\,\delta\mathbf{x}^k \tag{5.4}$$

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

$$\mathbf{J}(\mathbf{x}^{k-1})\,\delta\mathbf{x}^k = -\mathbf{F}(\mathbf{x}^{k-1}) \tag{5.5}$$

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 ($a = 1, \frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \ldots$) 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. (5.5). The action of $\mathbf{J}$ can be approximated by a first-order Taylor series expansion:

$$\mathbf{J}(\mathbf{x}^{k-1})\,\mathbf{w} \approx \frac{\mathbf{F}(\mathbf{x}^{k-1} + \epsilon\mathbf{w}) - \mathbf{F}(\mathbf{x}^{k-1})}{\epsilon} \tag{5.6}$$

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. (5.5) 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 (5.5) becomes

$$\mathbf{J}(\mathbf{x}^{k-1})\,\mathbf{P}^{-1}\delta\mathbf{z} = -\mathbf{F}(\mathbf{x}^{k-1}), \quad \text{with} \quad \delta\mathbf{z} = \mathbf{P}\delta\mathbf{x}^{k}. \tag{5.7}$$

The Krylov method iteratively improves the approximate solution to Eq. (5.7) in subspace ($\mathbf{r}_0$, $\mathbf{JP}^{-1}\mathbf{r}_0$, $(\mathbf{JP}^{-1})^2\mathbf{r}_0$, ..., $(\mathbf{JP}^{-1})^m\mathbf{r}_0$) 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. (5.5); $\mathbf{r}_0 = -\mathbf{F}(\mathbf{x}^{k-1})$ 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 $\mathbf{v}_0, \mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_m$ 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 `SEAICEuseJNFK = .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: $\|\mathbf{F}(\mathbf{x}^k)\| < \gamma_{\mathrm{nl}}\|\mathbf{F}(\mathbf{x}^0)\|$. 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

$$\gamma_k = \begin{cases} \gamma_0 & \text{for } \|\mathbf{F}(\mathbf{x}^{k-1})\| \geq r, \\ \max\left(\gamma_{\min}, \frac{\|\mathbf{F}(\mathbf{x}^{k-1})\|}{\|\mathbf{F}(\mathbf{x}^{k-2})\|}\right) & \text{for } \|\mathbf{F}(\mathbf{x}^{k-1})\| < r, \end{cases} \tag{5.8}$$

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 γ_0 = `JFNKgamma_lin_max` = 0.99, $\gamma_{\min}$ = `JFNKgamma_lin_min` = 0.1, and r = `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 (5.2) in such a way that the resulting elastic-viscous-plastic (EVP) and VP models are identical at steady state,

$$\frac{1}{E}\frac{\partial\sigma_{ij}}{\partial t} + \frac{1}{2\eta}\sigma_{ij} + \frac{\eta-\zeta}{4\zeta\eta}\sigma_{kk}\delta_{ij} + \frac{P}{4\zeta}\delta_{ij} = \dot{\epsilon}_{ij}. \tag{5.9}$$

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` = `SEAICIE_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 (5.9) can be written as:

$$\frac{\partial \sigma_1}{\partial t} + \frac{\sigma_1}{2T} + \frac{P}{2T} = \frac{P}{2T\Delta} D_D$$
$$\frac{\partial \sigma_2}{\partial t} + \frac{\sigma_2 e^2}{2T} = \frac{P}{2T\Delta} D_T$$
$$\frac{\partial \sigma_{12}}{\partial t} + \frac{\sigma_{12} e^2}{2T} = \frac{P}{4T\Delta} D_S$$

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 and recommend.

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 $E_0 \Delta t =$ 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 solu tions *[Hun01][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:

$$m \frac{D\mathbf{u}}{Dt} \approx m \frac{\mathbf{u}^{p+1} - \mathbf{u}^n}{\Delta t} + \beta^* \frac{\mathbf{u}^{p+1} - \mathbf{u}^p}{\Delta t_{\mathrm{EVP}}} \tag{5.10}$$

where n is the previous time step index, and p is the previous sub-cycling index. The extra "intertial" term $m\,(\mathbf{u}^{p+1} - \mathbf{u}^n)/\Delta t)$ allows the definition of a residual $|\mathbf{u}^{p+1} - \mathbf{u}^p|$ that, as $\mathbf{u}^{p+1} \to \mathbf{u}^{n+1}$, 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:

$$\sigma_{ij}^{p+1} = \sigma_{ij}^p + \frac{1}{\alpha}\Big(\sigma_{ij}(\mathbf{u}^p) - \sigma_{ij}^p\Big),$$
$$\mathbf{u}^{p+1} = \mathbf{u}^p + \frac{1}{\beta}\Big(\frac{\Delta t}{m}\nabla \cdot \sigma^{p+1} + \frac{\Delta t}{m}\mathbf{R}^p + \mathbf{u}_n - \mathbf{u}^p\Big).$$

$\mathbf{R}$ contains all terms in the momentum equations except for the rheology terms and the time derivative; α and β are free parameters (`SEAICE_evpAlpha`, `SEAICE_evpBeta`) 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 `SEAICEnEVPstarSteps` $\gg (\alpha, \beta)$ *[KDL15]*. Currently, there is no termination criterion and the number of mEVP iterations is fixed to `SEAICEnEVPstarSteps`.

In order to use mEVP in the MITgcm, set `SEAICEuseEVPstar = .TRUE.,` in `data.seaice`. If `SEAICEuseEVPrev =.TRUE.,` the actual form of equations ([eq:evpstarsigma]) and ([eq:evpstarmom]) is used with fewer implicit terms and the factor of e^2 dropped in the stress equations ([eq:evpstresstensor2]) and ([eq:evpstresstensor12]). 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

$$\alpha = \beta = \max\left(\tilde{c}\pi\sqrt{c\frac{\zeta}{A_c}\frac{\Delta t}{\max(m, 10^{-4}\,\mathrm{kg})}}, \alpha_{\min}\right)$$

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, SEAICEnEVPstarSteps = 500, SEAICEuseEVPstar = .TRUE., SEAICEuseEVPrev = .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:

$$\eta = \min\left(\frac{\zeta}{e^2}, \frac{\frac{P}{2} - \zeta(\dot{\epsilon}_{11} + \dot{\epsilon}_{22})}{\sqrt{\max(\Delta_{\min}^2, (\dot{\epsilon}_{11} - \dot{\epsilon}_{22})^2 + 4\dot{\epsilon}_{12}^2)}}\right).$$

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. (5.6.2.3). This stess 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 alse 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 a 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 avoid confusion with the bulk viscosity) at the bottom left corner of the cell to give $\overline{\zeta}^Z$ and $\overline{\eta}^Z$. 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:

$$\begin{aligned}
\dot{\epsilon}_{11} &= \partial_1 u_1 + k_2 u_2 \\
\Rightarrow (\epsilon_{11})^C_{i,j} &= \frac{u_{i+1,j} - u_{i,j}}{\Delta x^F_{i,j}} + k^C_{2,i,j} \frac{v_{i,j+1} + v_{i,j}}{2} \\
\dot{\epsilon}_{22} &= \partial_2 u_2 + k_1 u_1 \\
\Rightarrow (\epsilon_{22})^C_{i,j} &= \frac{v_{i,j+1} - v_{i,j}}{\Delta y^F_{i,j}} + k^C_{1,i,j} \frac{u_{i+1,j} + u_{i,j}}{2} \\
\dot{\epsilon}_{12} = \dot{\epsilon}_{21} &= \frac{1}{2}\Big(\partial_1 u_2 + \partial_2 u_1 - k_1 u_2 - k_2 u_1 \Big) \\
\Rightarrow (\epsilon_{12})^Z_{i,j} &= \frac{1}{2}\Bigg(\frac{v_{i,j} - v_{i-1,j}}{\Delta x^V_{i,j}} + \frac{u_{i,j} - u_{i,j-1}}{\Delta y^U_{i,j}} \\
&\qquad - k^Z_{1,i,j} \frac{v_{i,j} + v_{i-1,j}}{2} - k^Z_{2,i,j} \frac{u_{i,j} + u_{i,j-1}}{2} \Bigg),
\end{aligned}$$

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:

$$\begin{aligned}
k^C_{1,i,j} &= \frac{1}{\Delta y^F_{i,j}} \frac{\Delta y^G_{i+1,j} - \Delta y^G_{i,j}}{\Delta x^F_{i,j}} \\
k^C_{2,i,j} &= \frac{1}{\Delta x^F_{i,j}} \frac{\Delta x^G_{i,j+1} - \Delta x^G_{i,j}}{\Delta y^F_{i,j}} \\
k^Z_{1,i,j} &= \frac{1}{\Delta y^U_{i,j}} \frac{\Delta y^C_{i,j} - \Delta y^C_{i-1,j}}{\Delta x^V_{i,j}} \\
k^Z_{2,i,j} &= \frac{1}{\Delta x^V_{i,j}} \frac{\Delta x^C_{i,j} - \Delta x^C_{i,j-1}}{\Delta y^U_{i,j}}
\end{aligned}$$

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:

$$\begin{aligned}
(\nabla\sigma)_1 : \quad & \frac{1}{A^w_{i,j}} \int_{\text{cell}} (\partial_1\sigma_{11} + \partial_2\sigma_{21})\, dx_1\, dx_2 \\
&= \frac{1}{A^w_{i,j}} \Bigg\{ \int_{x_2}^{x_2+\Delta x_2} \sigma_{11} dx_2 \Big|_{x_1}^{x_1+\Delta x_1} + \int_{x_1}^{x_1+\Delta x_1} \sigma_{21} dx_1 \Big|_{x_2}^{x_2+\Delta x_2} \Bigg\} \\
&\approx \frac{1}{A^w_{i,j}} \Bigg\{ \Delta x_2 \sigma_{11} \Big|_{x_1}^{x_1+\Delta x_1} + \Delta x_1 \sigma_{21} \Big|_{x_2}^{x_2+\Delta x_2} \Bigg\} \\
&= \frac{1}{A^w_{i,j}} \Bigg\{ (\Delta x_2\sigma_{11})^C_{i,j} - (\Delta x_2\sigma_{11})^C_{i-1,j} \\
&\qquad + (\Delta x_1\sigma_{21})^Z_{i,j+1} - (\Delta x_1\sigma_{21})^Z_{i,j} \Bigg\}
\end{aligned}$$

with

$$
\begin{aligned}
(\Delta x_2 \sigma_{11})^C_{i,j} = \;& \Delta y^F_{i,j} (\zeta+\eta)^C_{i,j} \frac{u_{i+1,j}-u_{i,j}}{\Delta x^F_{i,j}} \\
& + \Delta y^F_{i,j} (\zeta+\eta)^C_{i,j} k^C_{2,i,j} \frac{v_{i,j+1}+v_{i,j}}{2} \\
& + \Delta y^F_{i,j} (\zeta-\eta)^C_{i,j} \frac{v_{i,j+1}-v_{i,j}}{\Delta y^F_{i,j}} \\
& + \Delta y^F_{i,j} (\zeta-\eta)^C_{i,j} k^C_{1,i,j} \frac{u_{i+1,j}+u_{i,j}}{2} \\
& - \Delta y^F_{i,j} \frac{P}{2} \\
(\Delta x_1 \sigma_{21})^Z_{i,j} = \;& \Delta x^V_{i,j} \overline{\eta}^Z_{i,j} \frac{u_{i,j}-u_{i,j-1}}{\Delta y^U_{i,j}} \\
& + \Delta x^V_{i,j} \overline{\eta}^Z_{i,j} \frac{v_{i,j}-v_{i-1,j}}{\Delta x^V_{i,j}} \\
& - \Delta x^V_{i,j} \overline{\eta}^Z_{i,j} k^Z_{2,i,j} \frac{u_{i,j}+u_{i,j-1}}{2} \\
& - \Delta x^V_{i,j} \overline{\eta}^Z_{i,j} k^Z_{1,i,j} \frac{v_{i,j}+v_{i-1,j}}{2}
\end{aligned}
$$

Similarly, we have for the v-equation ($\alpha = 2$):

$$
\begin{aligned}
(\nabla\sigma)_2 : \quad & \frac{1}{A^s_{i,j}} \int_{\text{cell}} (\partial_1 \sigma_{12} + \partial_2 \sigma_{22})\, dx_1\, dx_2 \\
=& \frac{1}{A^s_{i,j}} \left\{ \int_{x_2}^{x_2+\Delta x_2} \sigma_{12} dx_2 \Big|_{x_1}^{x_1+\Delta x_1} + \int_{x_1}^{x_1+\Delta x_1} \sigma_{22} dx_1 \Big|_{x_2}^{x_2+\Delta x_2} \right\} \\
\approx& \frac{1}{A^s_{i,j}} \left\{ \Delta x_2 \sigma_{12} \Big|_{x_1}^{x_1+\Delta x_1} + \Delta x_1 \sigma_{22} \Big|_{x_2}^{x_2+\Delta x_2} \right\} \\
=& \frac{1}{A^s_{i,j}} \Big\{ (\Delta x_2 \sigma_{12})^Z_{i+1,j} - (\Delta x_2 \sigma_{12})^Z_{i,j} \\
& \qquad + (\Delta x_1 \sigma_{22})^C_{i,j} - (\Delta x_1 \sigma_{22})^C_{i,j-1} \Big\}
\end{aligned}
$$

with

$$
\begin{aligned}
(\Delta x_1 \sigma_{12})^Z_{i,j} = \;& \Delta y^U_{i,j} \overline{\eta}^Z_{i,j} \frac{u_{i,j}-u_{i,j-1}}{\Delta y^U_{i,j}} \\
& + \Delta y^U_{i,j} \overline{\eta}^Z_{i,j} \frac{v_{i,j}-v_{i-1,j}}{\Delta x^V_{i,j}} \\
& - \Delta y^U_{i,j} \overline{\eta}^Z_{i,j} k^Z_{2,i,j} \frac{u_{i,j}+u_{i,j-1}}{2} \\
& - \Delta y^U_{i,j} \overline{\eta}^Z_{i,j} k^Z_{1,i,j} \frac{v_{i,j}+v_{i-1,j}}{2} \\
(\Delta x_2 \sigma_{22})^C_{i,j} = \;& \Delta x^F_{i,j} (\zeta-\eta)^C_{i,j} \frac{u_{i+1,j}-u_{i,j}}{\Delta x^F_{i,j}} \\
& + \Delta x^F_{i,j} (\zeta-\eta)^C_{i,j} k^C_{2,i,j} \frac{v_{i,j+1}+v_{i,j}}{2} \\
& + \Delta x^F_{i,j} (\zeta+\eta)^C_{i,j} \frac{v_{i,j+1}-v_{i,j}}{\Delta y^F_{i,j}} \\
& + \Delta x^F_{i,j} (\zeta+\eta)^C_{i,j} k^C_{1,i,j} \frac{u_{i+1,j}+u_{i,j}}{2} \\
& - \Delta x^F_{i,j} \frac{P}{2}
\end{aligned}
$$

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 $\sigma^Z_{21} = 0$, or equivalently $\eta^Z_{i,j} = 0$, 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 refered 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_FLOODDING` 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:

$$\frac{\partial X}{\partial t} = -\nabla \cdot (\vec{\mathbf{u}}\, X) + \Gamma_X + D_X$$

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 [sec:pkg:thsice]. 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.

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
```

SEAICE diagnostics

Diagnostics output is available via the diagnostics package (see Section [sec:pkg:diagnostics]). Available output fields are summarized in Table [tab:pkg:seaice:diagnostics].

Experiments and tutorials that use seaice

- Labrador Sea experiment in `lab_sea` verification directory. }
- `seaice_obcs`, based on `lab_sea`
- `offline_exf_seaice/input.seaicetd`, 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`

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.

[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.

[CMF08] J.-M. Campin, J. Marshall, and D. Ferreira. Sea-ice ocean coupling using a rescaled vertical coordinate z$^\ast $. *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 o$_2$ and co$_2$ 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.

[CMJSuarez94] 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. Ndvi-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 III W.D. Hibler. Modeling pack ice as a cavitating fluid. *J.~Phys.~Oceanogr.*, 22:626–651, 1992.

[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.

[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.

[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.

[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 co$_2$ 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 John Marshall. A strategy for terascale climate modeling. In *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.

[HL5a] 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, 1975a.

[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.

[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.

[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.

[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. Paluszkiewicz 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.

[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.

[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-homogenious 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.