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
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.

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

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

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


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 gridding and obviated the need to Fourier filter. The ‘vector-invariant’ form of MITgcm supports any orthogonal curvilinear grid, of which the cubed sphere is just one of many choices.

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

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.

1.2. Illustrations of the model in action
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.](image_url)

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.

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

Figure 1.11: Sensitivity of meridional overturning strength to surface heat flux changes. Contours show the magnitude of the response (in $\text{Sv} \times 10^{-4}$) that a persistent +1 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 $\frac{1}{3}^\circ \times \frac{1}{3}^\circ$ 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 $\mathbf{v}$, active tracers $\theta$ and $S$, a ‘geopotential’ $\phi$ and density $\rho = \rho(\theta, S, p)$ which may depend on $\theta$, $S$, and $p$. The equations that govern the evolution of these fields, obtained by applying the laws of classical mechanics and thermodynamics to a Boussinesq, Navier-Stokes fluid are, written in terms of a generic vertical coordinate, $r$, so that the appropriate kinematic boundary conditions can be applied isomorphically see Figure 1.18.

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

\[
\frac{D\hat{r}}{Dt} + \hat{k} \cdot \left( \mathbf{2\Omega} \times \mathbf{v} \right) + \frac{\partial \phi}{\partial r} + b = \mathcal{F}_r \quad \text{vertical momentum} \tag{1.2}
\]

\[
\nabla_h \cdot \mathbf{v}_h + \frac{\partial \phi}{\partial r} = 0 \quad \text{continuity} \tag{1.3}
\]

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

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

\[
\frac{DS}{Dt} = Q_S \quad \text{humidity/salinity} \tag{1.6}
\]
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.
z-p Isomorphism

Ocean (z coordinates) \( z \leftrightarrow p \) Atmosphere (p coordinates) \( P \leftrightarrow \Phi \)

\[
\begin{align*}
\frac{d\varphi}{dt} + f \times \varphi + \nabla \cdot P &= F \\
\frac{d\varphi}{dt} + f \times \varphi + \nabla \cdot \varphi &= F \\
\frac{d\varphi}{dt} + f \times \varphi + \nabla \cdot P &= F \\
\frac{d\varphi}{dt} + f \times \varphi + \nabla \cdot \varphi &= F \\
\end{align*}
\]

\[
\begin{align*}
\frac{d\varphi}{dt} + f \times \varphi + \nabla \cdot P &= F \\
\frac{d\varphi}{dt} + f \times \varphi + \nabla \cdot \varphi &= F \\
\frac{d\varphi}{dt} + f \times \varphi + \nabla \cdot P &= F \\
\frac{d\varphi}{dt} + f \times \varphi + \nabla \cdot \varphi &= F \\
\end{align*}
\]

\[
\begin{align*}
\frac{d\varphi}{dt} + f \times \varphi + \nabla \cdot P &= F \\
\frac{d\varphi}{dt} + f \times \varphi + \nabla \cdot \varphi &= F \\
\frac{d\varphi}{dt} + f \times \varphi + \nabla \cdot P &= F \\
\frac{d\varphi}{dt} + f \times \varphi + \nabla \cdot \varphi &= F \\
\end{align*}
\]

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

z-p Isomorphism

z = \eta(x,t) \quad \Phi = \Phi(x)

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

1.3. Continuous equations in ‘r’ coordinates
Here:

\[ r \] is the vertical coordinate

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

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

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

\[ t \] is time

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

\[ \phi \] is the 'pressure'/geopotential'

\[ \mathbf{\Omega} \] is the Earth’s rotation

\[ b \] is the ‘buoyancy’

\[ \theta \] is potential temperature

\[ S \] is specific humidity in the atmosphere; salinity in the ocean

\( F_{\mathbf{v}} \) are forcing and dissipation of \( \mathbf{v} \)

\( Q_\theta \) are forcing and dissipation of \( \theta \)

\( Q_S \) are forcing and dissipation of \( S \)

The \( F'\)s and \( Q'\)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):

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

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

Here

\[ R_{\text{moving}} = R_o + \eta \]

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

**Horizontal**

\[ \mathbf{v} \cdot \mathbf{n} = 0 \] (1.9)

where \( \mathbf{n} \) is the normal to a solid boundary.
Atmosphere

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

\[ r = p \] is the pressure \hspace{1cm} (1.10)

\[ \dot{r} = \frac{Dp}{Dt} = \omega \] is the vertical velocity in p coordinates \hspace{1cm} (1.11)

\[ \phi = g z \] is the geopotential height \hspace{1cm} (1.12)

\[ b = \frac{\partial \Pi}{\partial p} \theta \] is the buoyancy \hspace{1cm} (1.13)

\[ \theta = T \left( \frac{p_c}{p} \right)^\kappa \] is potential temperature \hspace{1cm} (1.14)

\[ S = q \] is the specific humidity \hspace{1cm} (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)

\[ \Pi(p) = c_p (\frac{p}{p_c})^\kappa \] \hspace{1cm} (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:

\[ \omega = 0 \] at \( r = R_{fixed} \) (top of the atmosphere) \hspace{1cm} (1.17)

\[ \omega = \frac{Dp_s}{Dt} \] at \( r = R_{moving} \) (bottom of the atmosphere) \hspace{1cm} (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:

\[ r = z \] is the height \hspace{1cm} (1.19)

\[ \dot{r} = \frac{Dz}{Dt} = w \] is the vertical velocity \hspace{1cm} (1.20)

\[ \phi = \frac{p}{\rho_c} \] is the pressure \hspace{1cm} (1.21)

\[ b(\theta, S, r) = \frac{g}{\rho_c} (\rho(\theta, S, r) - \rho_c) \] is the buoyancy \hspace{1cm} (1.22)

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

In the above:

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

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:

\[ w = 0 \text{ at } r = R_{fixed} \text{ (ocean bottom)} \] \hspace{1cm} (1.23)

\[ w = \frac{D\eta}{Dt} \text{ at } r = R_{moving} = \eta \text{ (ocean surface)} \] \hspace{1cm} (1.24)

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

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

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

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

\[ \phi(x, y, r) = \phi_s(x, y) + \phi_{hyd}(x, y, r) + \phi_{nh}(x, y, r) \] \hspace{1cm} (1.25)

and write (1.1) in the form:

\[ \frac{\partial \nu_h}{\partial t} + \nabla_h \phi_s + \nabla_h \phi_{hyd} + \epsilon_{nh} \nabla_h \phi_{nh} = \bar{G}_h \] \hspace{1cm} (1.26)

\[ \frac{\partial \phi_{hyd}}{\partial r} = -b \] \hspace{1cm} (1.27)

\[ \epsilon_{nh} \frac{\partial \phi_{nh}}{\partial t} + \frac{\partial \phi_{nh}}{\partial r} = G_i \] \hspace{1cm} (1.28)
Here $\epsilon_{nh}$ is a non-hydrostatic parameter.

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

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

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

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

\[
+ \mathcal{F}_u \quad \text{forcing/dissipation}
\]

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

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

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

\[
+ \mathcal{F}_v \quad \text{forcing/dissipation}
\]

\[
G_r = - \bar{v} \nabla r \quad \text{advection}
\]

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

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

\[
+ \mathcal{F}_r \quad \text{forcing/dissipation}
\]

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

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

**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) → (1.31) are neglected and ‘$r$’ is replaced by ‘$a$’, the mean radius of the earth. Once the pressure is found at one level - e.g. by inverting a 2-d Elliptic equation for $\phi$, 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 $2\Omega \cos \varphi$ Coriolis term are not neglected and are balanced by a non-hydrostatic contribution to the pressure field: only the terms underlined twice in Eqs. (1.29) → (1.31) are set to zero and, simultaneously, the shallow atmosphere

\(^1\) 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) → (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:

$$\dot{r} = \frac{Dp}{Dt} = \frac{1}{g} \frac{D\varphi}{Dt}$$ \hspace{1cm} (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 \( u, v, w, \theta \) and \( S \), the pressure field must be obtained diagnostically. We proceed, as before, by dividing the total (pressure/geo) potential in to three parts, a surface part, \( \phi_s(x, y) \), a hydrostatic part \( \phi_{hyd}(x, y, r) \) and a non-hydrostatic part \( \phi_{nh}(x, y, r) \), as in (1.25), and writing the momentum equation as in (1.26).

Hydrostatic pressure

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

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

and so

\[
\phi_{hyd}(x, y, r) = \int_r^{R_o} bdr \tag{1.33}
\]

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 \( r = R_{\text{fixed}} \) to \( r = R_{\text{moving}} \)

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

Thus:

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

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

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

(1.35)

where we have incorporated a source term.

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

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

(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 \( \phi_s \) 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:

$$\nabla^2 \phi_{nh} = \nabla \cdot \vec{G} - \left( \nabla^2 \phi_s + \nabla^2 \phi_{hyd} \right) = \nabla \cdot \vec{F}$$  \hspace{1cm} (1.37)

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

Boundary Conditions

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

$$\vec{v} \cdot \hat{n} = 0$$  \hspace{1cm} (1.38)

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

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

$$\hat{n} \cdot \nabla \phi_{nh} = \hat{n} \cdot \vec{F}$$  \hspace{1cm} (1.39)

where

$$\vec{F} = \vec{G} - \left( \nabla^2 \phi_s + \nabla \phi_{hyd} \right)$$

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

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

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

$$\hat{n} \cdot \nabla \phi_{nh} = 0$$  \hspace{1cm} (1.40)

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

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.

1.3. Continuous equations in ‘r’ coordinates
Dissipation

Momentum

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

\[ D_V = A_h \nabla^2_h v + A_v \frac{\partial^2 v}{\partial z^2} + A_4 \nabla^4_h v \]  (1.41)

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

Tracers

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

\[ D_{T,S} = \nabla \cdot \left[ K \nabla (T, S) \right] + K_4 \nabla^4_h (T, S) \]  (1.42)

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

\[
K = \begin{pmatrix}
K_h & 0 & 0 \\
0 & K_h & 0 \\
0 & 0 & K_v
\end{pmatrix}
\]  (1.43)

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

Vector invariant form

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

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

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

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:

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

(1.45)

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

(1.46)

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

(1.47)

$$
 p\alpha = RT
$$

(1.48)

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

(1.49)

where $\vec{v}_h = (u, v, 0)$ is the ‘horizontal’ (on pressure surfaces) component of velocity, $\frac{D}{Dt} = \frac{\partial}{\partial t} + \vec{v}_h \cdot \nabla_p + \omega \frac{\partial}{\partial p}$ 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 $\theta$ so that it looks more like a generic conservation law. Differentiating (1.48) we get:

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

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

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

(1.50)

Potential temperature is defined:

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

(1.51)

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

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

(1.52)

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

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

where $b = \frac{\partial \Pi}{\partial p} \theta$ is the buoyancy.
The heat equation is obtained by noting that
\[ c_p \frac{DT}{Dt} = \frac{D(\Pi \theta)}{Dt} = \Pi \frac{D\theta}{Dt} + \theta \frac{D \Pi}{Dt} = \Pi \frac{D\theta}{Dt} + \alpha \frac{Dp}{Dt} \]
and on substituting into (1.50) gives:
\[ \Pi \frac{D\theta}{Dt} = \Omega \] (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:

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

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

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

**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:

\[ \theta = \theta_o + \theta' \] (1.56)

\[ \alpha = \alpha_o + \alpha' \] (1.57)

\[ \phi = \phi_o + \phi' \] (1.58)

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

\[ \theta_o(p) = f^n(p) \]

\[ \alpha_o(p) = \Pi_p \theta_o \]

\[ \phi_o(p) = \phi_{topo} - \int_{p_0}^p \alpha_o dp \]

The final form of the HPE’s in \( p \)-coordinates is then:
\[ \frac{D\vec{v}_h}{Dt} + f \hat{k} \times \vec{v}_h + \nabla_p \phi' = \vec{f} \] (1.59)
\[ \frac{\partial \phi'}{\partial p} + \alpha' = 0 \quad (1.60) \]

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

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

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

**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:

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

\[ \epsilon_{nh} \frac{Dw}{Dt} + g + \frac{1}{\rho} \frac{\partial p}{\partial z} = \epsilon_{nh} F_w \quad (1.65) \]

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

\[ \rho = \rho(\theta, S, p) \quad (1.67) \]

\[ \frac{D \theta}{Dt} = Q_\theta \quad (1.68) \]

\[ \frac{DS}{Dt} = Q_s \quad (1.69) \]

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

\[ \frac{D \rho}{Dt} = \frac{\partial \rho}{\partial \theta} \bigg|_{S,p} \frac{D \theta}{Dt} + \frac{\partial \rho}{\partial S} \bigg|_{\theta,p} \frac{DS}{Dt} + \frac{\partial \rho}{\partial p} \bigg|_{\theta,S} \frac{Dp}{Dt} \quad (1.70) \]
Note that $\frac{\partial \rho}{\partial p} = \frac{1}{c_s^2}$ is the reciprocal of the sound speed ($c_s$) squared. Substituting into (1.66) gives:

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

(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:

$$\frac{DV_h}{Dt} + f \hat{k} \times \vec{v}_h + \frac{1}{\rho} \nabla_z p = \vec{F}$$

(1.72)

$$\epsilon_{nh} \frac{Dw}{Dt} + g + \frac{1}{\rho} \frac{Dp}{Dz} = \epsilon_{nh} \mathcal{F}_w$$

(1.73)

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

(1.74)

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

(1.75)

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

(1.76)

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

(1.77)

**Compressible z-coordinate equations**

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

$$\frac{DV_h}{Dt} + f \hat{k} \times \vec{v}_h + \frac{1}{\rho_o} \nabla_z p = \vec{F}$$

(1.78)

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

(1.79)

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

(1.80)

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

(1.81)

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

(1.82)

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

(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. $\frac{\partial \rho}{\partial p} \big|_{\theta, S} = \frac{1}{c_s^2}$, 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 \( \frac{DP}{Dt} \approx -g \rho_o \frac{Dz}{Dt} \approx -g \rho_o w \), but this leads to an inconsistency between continuity and EOS. A better solution is to change the dependency on pressure in the EOS by splitting the pressure into a reference function of height and a perturbation:

\[
\rho = \rho(\theta, S, p_o(z) + \epsilon_s p')
\]

Remembering that the term \( \frac{DP}{Dt} \) in continuity comes from differentiating the EOS, the continuity equation then becomes:

\[
\frac{1}{\rho_o c_s^2} \left( \frac{DP_o}{Dt} + \epsilon_s \frac{DP'}{Dt} \right) + \nabla_z \cdot \vec{v}_h + \frac{\partial w}{\partial z} = 0
\]

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

\[
\nabla_z \cdot \vec{v}_h + \frac{\partial w}{\partial z} - \frac{g}{c_s^2} w = 0
\]  

(1.84)

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

\[
\nabla_z \cdot \vec{v}_h + \frac{1}{\rho_o} \frac{\partial (\rho_o w)}{\partial z} = 0
\]  

(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 \( \rho_o \) is evaluated from prescribed \( \theta_o \) and \( S_o \) profiles, then the EOS dependency on \( p_o \) and the term \( \frac{Dq}{Dt} \) in continuity should be referred to those same profiles. The full set of ‘quasi-Boussinesq’ or ‘anelastic’ equations for the ocean are then:

\[
\frac{D\vec{q}_h}{Dt} + f\hat{k} \times \vec{v}_h + \frac{1}{\rho_o} \nabla_z p = \vec{F}
\]  

(1.86)

\[
\epsilon_{nh} \frac{D\vec{w}}{Dt} + g \rho_o \frac{1}{\rho_o} \frac{\partial p}{\partial z} = \epsilon_{nh} \vec{F}_w
\]  

(1.87)

\[
\nabla_z \cdot \vec{v}_h + \frac{1}{\rho_o} \frac{\partial (\rho_o w)}{\partial z} = 0
\]  

(1.88)

\[
\rho = \rho(\theta, S, p_o(z))
\]  

(1.89)

\[
\frac{D\theta}{Dt} = Q_\theta
\]  

(1.90)

\[
\frac{DS}{Dt} = Q_s
\]  

(1.91)
Incompressible z-coordinate equations

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

\[
\frac{D\mathbf{v}_h}{Dt} + f\mathbf{k} \times \mathbf{v}_h + \frac{1}{\rho_c} \nabla_z p = \mathbf{f}^z \tag{1.92}
\]

\[
\epsilon_{nh} \frac{Dw}{Dt} + \frac{g \rho'}{\rho_c} + \frac{1}{\rho_c} \frac{\partial p'}{\partial z} = \epsilon_{nh} F_w \tag{1.93}
\]

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

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

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

\[
\frac{DS}{Dt} = Q_s \tag{1.97}
\]

where \( \rho_c \) 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:

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

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

\[
\rho_o = \rho_c \nonumber
\]

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

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

\[
\frac{D\mathbf{v}_h}{Dt} + f\mathbf{k} \times \mathbf{v}_h + \frac{1}{\rho_c} \nabla_z p' = \mathbf{f}^z \tag{1.98}
\]

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

\[
\nabla_z \cdot \mathbf{v}_h + \frac{\partial w}{\partial z} = 0 \tag{1.100}
\]
\[ \rho' = \rho(\theta, S, p_o(z)) - \rho_c \]  
\[ \frac{D\theta}{Dt} = Q_\theta \]  
\[ \frac{DS}{Dt} = Q_s \]

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

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

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

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

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

![GitHub setup diagram](image)

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 ‘admonition’ block. The source code to achieve this is:

```plaintext
.. 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 on exit)
   :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.
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, \( \tau_x \), 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

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

where \(L_y\) is the lateral domain extent (1200–km) and \(\tau_0\) is set to 0.1 N m\(^{-2}\).

Figure 4.1 summarizes the configuration simulated.

![Schematic of simulation domain and wind-stress forcing function for barotropic gyre numerical experiment.](image)

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 \(\nabla^2_h\) 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

\[
\begin{align*}
\frac{Du}{Dt} - f v + g \frac{\partial \eta}{\partial x} - A_h \nabla^2_h u = & \frac{\tau_x}{\rho_0 \Delta z} \\
\frac{Dv}{Dt} + f u + g \frac{\partial \eta}{\partial y} - A_h \nabla^2_h v = & 0 \\
\frac{\partial \eta}{\partial t} + \nabla_h \cdot \vec{u} = & 0
\end{align*}
\] (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, \(\Delta z\), of 5000 m.
Numerical Stability Criteria

The Laplacian dissipation coefficient, $A_h$, is set to 400 m$^{-1}$. This value is chosen to yield a Munk layer width \[M_w = \pi \left( \frac{A_h}{\beta} \right)^{\frac{1}{3}}\] of $\approx 100$ km. This is greater than the model resolution $\Delta x$, 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 \[S_l = 4 \frac{A_h \delta t}{\Delta x^2}\] (4.5)
evaluates to 0.012, which is well below the 0.3 upper limit for stability.

The numerical stability for inertial oscillations \[S_i = f^2 \delta t^2\] (4.6)
evaluates to 0.0144, which is well below the 0.5 upper limit for stability.

The advective CFL \[S_a = \frac{|\vec{u}| \delta t}{\Delta x}\] (4.7)
evaluates to 0.12. This is approaching the stability limit of 0.5 and limits $\delta t$ to 1200 s.

Code Configuration

The model configuration for this experiment resides under the directory \texttt{verification/tutorial_barotropic_gyre/}.

The experiment files

- \texttt{input/data}
- \texttt{input/data.pkg}
- \texttt{input/eedata}
- \texttt{input/windx.sin_y}
- \texttt{input/topog.box}
- \texttt{code/CPP_EEOPTIONS.h}
- \texttt{code/CPP_OPTIONS.h}
- \texttt{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 $400 \text{m}^2 \text{s}^{-1}$
- **Line 10**
  - `beta=1.E-11`,
  - this line sets $\beta$ (the gradient of the coriolis parameter, $f$) to $10^{-11} \text{s}^{-1} \text{m}^{-1}$
- **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 $t = 12000 \text{s}$. 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 $20 \times 10^3 \text{m}$ (20 km).
- **Line 42**
  - `delY=60*20E3`,
  - This line sets the horizontal grid spacing between each y-coordinate line in the discrete grid to $20 \times 10^3 \text{m}$ (20 km).
– 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 \texttt{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 \texttt{input/gendata.m} includes example code to generate a valid \texttt{zonalWindFile} file.

other lines in the file \texttt{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

```plaintext
# 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,
```

4.2. Barotropic Gyre MITgcm Example
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, \(\tau_x\), values. The units used are \(Nm^{-2}\). Although \(\tau_x\) 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 \(-\text{del}Z\) m, corresponding respectively to a wall or to deep ocean. The file contains a raw binary stream of data that is enumerated in the same way as standard MITgcm two-dimensional, horizontal arrays. The included matlab program *input/gendata.m* gives a complete code for creating the *input/topog.box* file.

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

```
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 \( nPx \times nSx \) blocks of size \( sNx \) along one axis |
C | \( nPy \times nSy \) 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 - Overlap 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 N\( x \) - No. points in X for the total domain.
C N\( y \) - No. points in Y for the total domain.
C N\( r \) - No. points in R 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 ( 
\( sNx = 30, \)
\( sNy = 30, \)
\( OLx = 2, \)
\( OLy = 2, \)
\( nSx = 2, \)
\( nSy = 2, \)
\( nPx = 1, \)
\( nPy = 1, \)
\( Nx = sNx \times nSx \times nPx, \)
```
\[ N_y = sN_y \times n_S y \times n_P y, \]
\[ N_r = 1 \]

\[
\text{MAX}_\text{OLX} \quad \text{Set to the maximum overlap region size of any array}
\]
\[
\text{MAX}_\text{OLY} \quad \text{that will be exchanged. Controls the sizing of each routine buffers.}
\]
\[
\text{INTEGER} \quad \text{MAX}_\text{OLX}
\]
\[
\text{INTEGER} \quad \text{MAX}_\text{OLY}
\]
\[
\text{PARAMETER} \ ( \text{MAX}_\text{OLX} = \text{OLx}, \text{MAX}_\text{OLY} = \text{OLy} )
\]

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 \(3^\circ \times 1\) cm cylindrical grid with twenty-nine vertical levels of 0.5 cm 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 \(\Delta a = 1^\circ\) cmd: \textit{math}: \text{Delta phi=3}\text{circ}^\circ, \text{so that there are 120 grid cells in the azimuthal direction and thirty-one grid cells in the radial, representing a tank 62 cm in diameter. The bathymetry file sets the depth=0 in the nine lowest radial rows to represent the central of the annulus. Vertically the model is configured with twenty-nine layers of uniform 0.5 cm 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 experiment. 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,
  - $\text{viscAh}=5.0\times10^{-6}$,
  - $\text{viscAz}=5.0\times10^{-6}$,

These lines set the Laplacian friction coefficient in the horizontal and vertical, respectively. Note that they are several orders of magnitude smaller than the other examples due to the small scale of this example.

- Lines 13-16,
  - $\text{diffKhT}=2.5\times10^{-6}$,
  - $\text{diffKzT}=2.5\times10^{-6}$,
  - $\text{diffKhS}=1.0\times10^{-6}$,
  - $\text{diffKzS}=1.0\times10^{-6}$,

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, $\text{f0}=0.5$, this line sets the coriolis term, and represents a tank spinning at about 2.4 rpm.

- Lines 23 and 24
  - $\text{rigidLid}=\text{TRUE.}$,
  - $\text{implicitFreeSurface}=\text{FALSE.}$,

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

- Line 40,
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,
  - $\delta t = 0.1$,

This line sets the integration timestep to $0.1s$. This is an unusually small value among the examples due to the small physical scale of the experiment. Using the ensemble Kalman filter to produce input fields can necessitate even shorter timesteps.

- Line 56,
  - usingCylindricalGrid = .TRUE.,

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

- Line 57,
  - $dX_{\text{spacing}} = 3$,

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,
  - $dY_{\text{spacing}} = 0.01$,

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

- Line 59,
  - $\delta z = 29 \times 0.005$,

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 $(a, \phi)$ map of depths. This file is assumed to contain 64-bit binary numbers giving the depth of the model at each grid cell, ordered with the $\phi$ coordinate varying fastest. The points are ordered from low coordinate to high coordinate for both axes. The units and orientation of the depths in this file are the same as used in the MITgcm code. In this experiment, a depth of $0m$ indicates an area outside of the tank and a depth of $-0.145m$ indicates the tank itself.

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

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

- Lines 66 and 67
  - $t_{\text{CylIn}} = 0$
  - $t_{\text{CylOut}} = 20$

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**

```plaintext
# ===============
# | 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=3600000,
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.,
```

### 4.3. A Rotating Tank in Cylindrical Coordinates

53
dxspacing=3.,
dyspacing=0.01,
delZ=29*0.005,
ygOrigin=0.07,

# Input datasets
sPARM05
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 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 input/bathyPol.bin file specifies a two-dimensional $(x,y)$ map of depth values. For this experiment values are either $0m$ or $-\text{delZ}m$, corresponding respectively to outside or inside of the tank. The file contains a raw binary stream of data that is enumerated in the same way as standard MITgcm two-dimensional, horizontal arrays.

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
The design here supports a three-dimensional model grid with indices I, J and K. The three-dimensional domain is comprised of nPx *nSx blocks of size sNx along one axis, nPy *nSy blocks of size sNy along another axis and one block of size Nz along the final axis. Blocks have overlap regions of size OLx and OLy along the dimensions that are subdivided.

Voodoo numbers controlling data layout.

sNx - No. X points in sub-grid.
sNy - No. Y points in sub-grid.
OLx - Overlap extent in X.
OLy - Overlap extent in Y.
nSx - No. sub-grids in X.
nSy - No. sub-grids in Y.
nPx - No. of processes to use in X.
nPy - No. of processes to use in Y.
Nx - No. points in X for the total domain.
Ny - No. points in Y for the total domain.
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)

max_olx - Set to the maximum overlap region size of any array that will be exchanged. Controls the sizing of exchange routine buffers.

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.
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 algorithms that apply to many different simulation types. Similarly, the general purpose computational packages implement non-numerical algorithms that provide parallelism, I/O and time-keeping functions that are used in many different scenarios.

The following sections describe the packages shown in Figure 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 particular model can be configured using a combination of both “compile–time” and “run–time” options. Compile–time options are those used to select which packages will be
Figure 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 *genmake2* options `-enable PKG` and/or `-disable PKG` specifies inclusion or exclusion. This method is intended as a convenient way to perform a single (perhaps for a quick test) compilation.

2. By creating a text file with the name `packages.conf` in either the local build directory or the `-mods=DIR` directory, one can specify a list of packages (one package per line, with `'#'` as the comment character) to be included. Since the `packages.conf` file can be saved, this is the preferred method for setting and recording (for future reference) the package configuration.

3. For convenience, a list of “standard” package groups is contained in the `pkg/pkg_groups` file. By selecting one of the package group names in the `packages.conf` file, one automatically obtains all packages in that group.

4. By default (that is, if a `packages.conf` file is not found), the *genmake2* program will use the package group default “default_pkg_list” as defined in `pkg/pkg_groups` file.

5. To help prevent users from creating unusable package groups, the *genmake2* program will parse the contents of the `pkg/pkg_depend` file to determine:
   - whether any two requested packages cannot be simultaneously included (*eg.* `seaice` and `thsice` are mutually exclusive),
   - whether additional packages must be included in order to satisfy package dependencies (*eg.* `rw` depends upon functionality within the `mdsio` package), and
   - whether the set of all requested packages is compatible with the dependencies (and producing an error if they aren’t).

Thus, as a result of the dependencies, additional packages may be added to those originally requested.

**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:

```plaintext
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 \texttt{-standarddirs} option sets these locations and the default value contains the directories \texttt{eesupp} and \texttt{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) \textit{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 \texttt{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 \texttt{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,
```
which is included from the file and shows how both the compile-time `ALLOW_GMREDI` flag and the run-time `useGMRedi` are nested.

There are some benefits to using the technique described here. The first is that code snippets or subroutines associated with packages can be placed or called from almost anywhere else within the code. The second benefit is related to memory footprint and performance. Since unused code can be removed, there is no performance penalty due to unnecessary memory allocation, unused function calls, or extra run-time `IF (...)` conditions. The major problems with this approach are the potentially difficult-to-read and difficult-to-debug code caused by an overuse of CPP statements. So while it can be done, developers should exercise some discipline and avoid unnecessarily “smearing” their package implementation details across numerous files.

### Package Startup or Boot Sequence

Calls to package routines within the core code timestepping loop can vary. However, all packages should follow a required “boot” sequence outlined here:

1. S/R `PACKAGES_BOOT()`

   ```
   CALL OPEN_COPY_DATA_FILE( 'data.pkg', 'PACKAGES_BOOT', ... )
   ```

2. S/R `PACKAGES_READPARMS()`

   ```
   #ifdef ALLOW_${PKG}
   if ( use${Pkg} )
   & CALL ${PKG}_READPARMS( retCode )
   #endif
   ```

3. S/R `PACKAGES_INIT_FIXED()`

   ```
   #ifdef ALLOW_${PKG}
   if ( use${Pkg} )
   & CALL ${PKG}_INIT_FIXED( retCode )
   #endif
   ```

4. S/R `PACKAGES_CHECK()`

   ```
   #ifdef ALLOW_${PKG}
   if ( use${Pkg} )
   & CALL ${PKG}_CHECK( retCode )
   #else
   if ( use${Pkg} )
   & CALL PACKAGES_CHECK_ERROR('${PKG}')
   #endif
   ```

5. S/R `PACKAGES_INIT_VARIABLES()`

   ```
   #ifdef ALLOW_${PKG}
   if ( use${Pkg} )
   & CALL ${PKG}_INIT_VARIA( )
   #endif
   ```

6. S/R `DO_THE_MODEL_IO`

---

5.1. Overview
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/tracers). (see also sections [sec:tracer:sub:equations] to [sec:tracer:sub:advection:schemes]).

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 conjunction with Adams–Bashforth time stepping. Diffusive and parameterized fluxes are always calculated here.
• GAD_ADVECTION calculates just the advective fluxes using the non-linear schemes and can not be used in conjunction with Adams–Bashforth time stepping.

### GAD Diagnostics

<table>
<thead>
<tr>
<th>&lt;-Name-&gt;</th>
<th>Levs</th>
<th>&lt;-parsing code-&gt;</th>
<th>&lt;- Units --&gt;</th>
<th>&lt;- Tile (max=80c)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADVr_TH</td>
<td>15</td>
<td>WM LR</td>
<td>degC.m^3/s</td>
<td>Vertical Advective Flux of Pot. Temperature</td>
</tr>
<tr>
<td>ADVx_TH</td>
<td>15</td>
<td>UU 087MR</td>
<td>degC.m^3/s</td>
<td>Zonal Advective Flux of Pot. Temperature</td>
</tr>
<tr>
<td>ADVy_TH</td>
<td>15</td>
<td>VV 086MR</td>
<td>degC.m^3/s</td>
<td>Meridional Advective Flux of Pot. Temperature</td>
</tr>
<tr>
<td>DFrE_TH</td>
<td>15</td>
<td>WM LR</td>
<td>degC.m^3/s</td>
<td>Vertical Diffusive Flux of Pot. Temperature (Explicit part)</td>
</tr>
<tr>
<td>DIFx_TH</td>
<td>15</td>
<td>UU 090MR</td>
<td>degC.m^3/s</td>
<td>Zonal Diffusive Flux of Pot. Temperature</td>
</tr>
<tr>
<td>DIFy_TH</td>
<td>15</td>
<td>VV 089MR</td>
<td>degC.m^3/s</td>
<td>Meridional Diffusive Flux of Pot. Temperature</td>
</tr>
<tr>
<td>DFrI_TH</td>
<td>15</td>
<td>WM LR</td>
<td>degC.m^3/s</td>
<td>Vertical Diffusive Flux of Pot. Temperature (Implicit part)</td>
</tr>
<tr>
<td>ADVr_SLT</td>
<td>15</td>
<td>WM LR</td>
<td>psu.m^3/s</td>
<td>Vertical Advective Flux of Salinity Temperature</td>
</tr>
<tr>
<td>ADVx_SLT</td>
<td>15</td>
<td>UU 094MR</td>
<td>psu.m^3/s</td>
<td>Zonal Advective Flux of Salinity Temperature</td>
</tr>
<tr>
<td>ADVy_SLT</td>
<td>15</td>
<td>VV 093MR</td>
<td>psu.m^3/s</td>
<td>Meridional Advective Flux of Salinity Temperature</td>
</tr>
<tr>
<td>DFrE_SLT</td>
<td>15</td>
<td>WM LR</td>
<td>psu.m^3/s</td>
<td>Vertical Diffusive Flux of Salinity Temperature (Explicit part)</td>
</tr>
<tr>
<td>DIFx_SLT</td>
<td>15</td>
<td>UU 097MR</td>
<td>psu.m^3/s</td>
<td>Zonal Diffusive Flux of Salinity Temperature</td>
</tr>
<tr>
<td>DIFy_SLT</td>
<td>15</td>
<td>VV 096MR</td>
<td>psu.m^3/s</td>
<td>Meridional Diffusive Flux of Salinity Temperature</td>
</tr>
<tr>
<td>DFrI_SLT</td>
<td>15</td>
<td>WM LR</td>
<td>psu.m^3/s</td>
<td>Vertical Diffusive Flux of Salinity Temperature (Implicit part)</td>
</tr>
</tbody>
</table>

### 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]

5.2. Packages Related to Hydrodynamical Kernel
• 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 `pkg/exch2/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 `exch2` to the `packages.conf` file – see Section ref{sec:buildingCode} 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 $32 \times 192$ domain divided into six $32 \times 32$ subdomains, each of which is divided into eight tiles of width $t_{nx}=16$ and height $t_{ny}=8$ for a total of forty-eight tiles. The colored borders of the subdomains represent the parameters $nr$ (red), $ng$ (green), and $nb$ (blue). This tiling is used in the example verification/adjustment.cs-32x32x1/ with the option (blanklist.txt) to remove the land-only 4 tiles (11,12,13,14) which are filled in red on the plot.

Figure 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 ($t_{nx}=90, t_{ny}=90$), resulting in a total of 18 tiles.
Figure 5.4: Plot of a cubed sphere topology with a $32 \times 192$ domain divided into six $32 \times 32$ subdomains with one tile each ($t_{nx}=32, t_{ny}=32$). This is the default configuration.

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

The parameters $\text{nSx}$, $\text{nSy}$, $\text{nPx}$, and $\text{nPy}$ relate to the number of tiles and how they are distributed on processors. When using exch2, the tiles are stored in the $x$ dimension, and so $\text{nSy}=1$ in all cases. Since the tiles as configured by exch2 cannot be split up across processors without regenerating the topology, $\text{nPy}=1$ as well.

The number of tiles MITgcm allocates and how they are distributed between processors depends on $\text{nSx}$ and $\text{nPx}$, where $\text{nSx}$ is the number of tiles per processor and $\text{nPx}$ is the number of processors. The total number of tiles in the topology minus those listed in file{blanklist.txt} must equal $\text{nSx*nPx}$. 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 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,
```

5.2. Packages Related to Hydrodynamical Kernel
The following is an example for the forty-eight-tile topology in figure ref{fig:48tile} running on six processors:

```plaintext
PARAMETER ( 
  sNx = 16, 
  sNy = 8, 
  OLx = 2, 
  OLy = 2, 
  nSx = 8, 
  nSy = 1, 
  nPx = 6, 
  nPy = 1, 
  Nx = sNx*nSx*nPx, 
  Ny = sNy*nSy*nPy, 
  Nr = 5 )
```

**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 `NTILES`, and the maximum number of neighbors of any tiles by `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 `exch2_domain_nxt` and `exch2_domain_nyt` express the number of tiles in the $x$ and $y$ global indices. For example, the default setup of six tiles (Fig. ref{fig:6tile}) has `exch2_domain_nxt=6` and `exch2_domain_nyt=1`. A topology of forty-eight tiles, eight per subdomain (as in figure ref{fig:48tile}), will have `exch2_domain_nxt=12` and `exch2_domain_nyt=4`. Note that these parameters express the tile layout in order to allow global data files that are tile-layout-neutral. They have no bearing on the internal storage of the arrays. The tiles are stored internally in a range from `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 `NTILES` and are indexed to the tile number, which is indicated in the diagrams with the notation `texttt{t}`. The indices are omitted in the descriptions.

The arrays `exch2_tnx` and `exch2_tny` express the $x$ and $y$ dimensions of each tile. At present for each tile `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 `exch2_tbasex` and `exch2_basey` 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}`
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 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,

\begin{verbatim}
exch2_neighbourId( exch2_opposingSend_record(a,T),
exch2_neighbourId(a,T) ) = T
\end{verbatim}

This provides a back-reference from the neighbor tiles.

The arrays varlink{exch2_pi}{exch2_pi} and varlink{exch2_pj}{exch2_pj} specify the transformations of indices in exchanges between the neighboring tiles. These transformations are necessary in exchanges between subdomains because a horizontal dimension in one subdomain may map to other horizontal dimension in an adjacent subdomain, and may also have its indexing reversed. This swapping arises from the “folding” of two-dimensional arrays into a three-dimensional cube.

The dimensions of code{exch2_pi(t,N,T)} and code{exch2_pj(t,N,T)} are the neighbor ID code{N} and the tile number code{T} as explained above, plus a vector of length code{2} containing transformation factors code{t}. The first element of the transformation vector holds the factor to multiply the index in the same dimension, and the second element holds the the same for the orthogonal dimension. To clarify, code{exch2_pi(1,N,T)} holds the mapping of the $x$ axis index of tile code{T} to the $x$ axis of tile code{T}’s neighbor code{N}, and code{exch2_pi(2,N,T)} holds the mapping of code{T}’s $x$ index to the neighbor code{N}’s $y$ index.

One of the two elements of code{exch2_pi} or code{exch2_pj} for a given tile code{T} and neighbor code{N} will be code{0}, reflecting the fact that the two axes are orthogonal. The other element will be code{1} or code{-1}, depending on whether the axes are indexed in the same or opposite directions. For example, the transform vector of the arrays for all tile neighbors on the same subdomain will be code{(1,0)}, since all tiles on the same subdomain are oriented identically. An axis that corresponds to the orthogonal dimension with the same index direction in a particular tile-neighbor orientation will have code{(0,1)}. Those with the opposite index direction will have code{(0,-1)} in order to reverse the ordering.

The arrays varlink{exch2_oi}{exch2_oi}, varlink{exch2_oi_f}{exch2_oi_f}, and varlink{exch2_oi_f}{exch2_oi_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
The simplest case is `exch2_oi(2,1)`, the southern neighbor, which is `Tn=6`. The axes of `T` and `Tn` have the same orientation and their $x$-$y$ axes have the same origin, and so an exchange between the two requires no changes to the $x$-$y$ index. For the western neighbor (`Tn=5`), `exch2_oi(3,1)=32` since the $x=0$ vector on `T` corresponds to the $y=32$ vector on `Tn`. The eastern edge of `T` shows the reverse case (`exch2_oi(4,1)=-32`), where $x=32$ on `T` exchanges with $x=0$ on `Tn=2`.

The most interesting case, where `exch2_oi(1,1)=33` and `Tn=3`, involves a reversal of indices. As in every case, the offset code`exch2_oi` is added to the original $x$-$y$ 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 $x$-$y$ axis of code`T` is orthogonal to the $x$-$y$ axis of code`Tn`. code`exch2_pi(2,1,1)=-1` since the $x$-$y$ axis of code`T` corresponds to the $y$-$x$ 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 $y$-$x$ 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 $x$-$y$ axes. The $x$-$y$ 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 $y$-$x$ 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 $x$-$y$ axis, but since code`Tn`'s $y$-$x$ axis corresponds to code`T`'s $x$-$y$ 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 $x$-$y$ 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-

---

MITgcm Documentation, Release 1.0

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 $x$-$y$ axes have the same origin, and so an exchange between the two requires no changes to the $x$-$y$ 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 $x$-$y$ 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 $x$-$y$ axis of code{T} is orthogonal to the $x$-$y$ axis of code{Tn}. code{exch2_pi(2,1,1)=-1} since the $x$-$y$ axis of code{T} corresponds to the $y$-$x$ 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 $y$-$x$ 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 $x$-$y$ axes. The $x$-$y$ 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 $y$-$x$ 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 $x$-$y$ axis, but since code{Tn}'s $y$-$x$ axis corresponds to code{T}'s $x$-$y$ 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 $x$-$y$ 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 independent of the grid on which the equations of motion are integrated, and the 'tendency' terms can be interpolated to the vertical grid on which the equations of motion are integrated. A modified $p^*$ coordinate, which adjusts to the local terrain and adds additional levels between the lower levels of the existing $p^*$ grid (and perhaps between the levels near the tropopause as well), is implemented. The vertical discretization is different for each grid point, although it consist of the same number of levels. Additional 'sponge' levels aloft are added when needed. The levels of the physics grid are constrained to fit exactly into the existing $p^*$ grid, simplifying the mapping between the two vertical coordinates. This is illustrated as follows:

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
Modified P* Discretization for High End Physics

Dark solid lines represent existing P* levels, light solid lines are the addition levels added at each grid cell.

Figure 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 necessitates a second set of evolution equations for the atmospheric state variables on the modified grid. If the equation for the evolution of \( U \) on \( p^* \) can be expressed as:

\[
\frac{\partial U}{\partial t} \bigg|_{p^*} = \frac{\partial U}{\partial t} \bigg|_{p^*}^{\text{dynamics}} + \frac{\partial U}{\partial t} \bigg|_{p^*}^{\text{physics}}
\]

where the physics forcing terms on \( p^* \) have been mapped from the modified grid, then an additional equation to govern the evolution of \( U \) (for example) on the modified grid is written:

\[
\frac{\partial U}{\partial t} \bigg|_{p^*}^{\text{total}} = \frac{\partial U}{\partial t} \bigg|_{p^*}^{\text{dynamics}} + \frac{\partial U}{\partial t} \bigg|_{p^*}^{\text{physics}} + \gamma (U \big|_{p^*} - U \big|_{p^*}^{m})
\]

where \( p^* \) refers to the modified higher resolution grid, and the dynamics forcing terms have been mapped from \( p^* \) space. The last term on the RHS is a relaxation term, meant to constrain the state variables on the modified vertical grid to ‘track’ the state variables on the \( p^* \) grid on some time scale, governed by \( \gamma \). In the present implementation, \( \gamma = 1 \), requiring an immediate agreement between the two ‘states’.

72 Chapter 5. Physical Parameterizations - Packages I
Time stepping Sequence

If we write $T_{phys}$ as the temperature (or any other state variable) on the high resolution physics grid, and $T_{dyn}$ as the temperature on the coarse vertical resolution dynamics grid, then:

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

```fortran
subroutine make_phys_grid(drF,hfacC,im1,im2,jm1,jm2,Nr,Nsx,Nsy,i1,i2,j1,j2,bi,bj,Nrphys,Lbot,dphys,numlevphys,nlperdyn)
c**********************************************************************
c Purpose: Define the grid that 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
```
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:

```fortran
subroutine gridalt_update(myThid)
```

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:

```fortran
subroutine gridalt_update(myThid)
```

The gridalt package also supplies utility routines which perform the mappings from one grid to the other. These routines are called from the code which computes the fields on the alternative (fizhi) grid.

```fortran
subroutine dyn2phys(qdyn,pedyn,im1,im2,jm1,jm2,lmdyn,Nsx,Nsy,. idim1,idim2,jdim1,jdim2,bi,bj,windphy,pephy,Lbot,lmphy,nlperdyn,. flg,qphy)
```

Chapter 5. Physical Parameterizations - Packages I
And similarly, gridalt contains subroutine phys2dyn.

**Gridalt Diagnostics**

<table>
<thead>
<tr>
<th>&lt;-Name-&gt;</th>
<th>Levs</th>
<th>&lt;-parsing code-&gt;</th>
<th>&lt;- Units -&gt;</th>
<th>&lt;- Tile (max=80c) -&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>DPPHYS</td>
<td>20</td>
<td>SM ML</td>
<td>Pascal</td>
<td>Pressure Thickness of Layers on Fizhi Grid</td>
</tr>
</tbody>
</table>

**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 accommodate all imaginable and possible options. Therefore, for a regional simulation with very particular details, it is recommended to familiarize oneself not only with the compile- 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 frequency) 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.

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>
<thead>
<tr>
<th>CPP option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALLOW_OBCS_NORTH</td>
<td>enable Northern OB</td>
</tr>
<tr>
<td>ALLOW_OBCS_SOUTH</td>
<td>enable Southern OB</td>
</tr>
<tr>
<td>ALLOW_OBCS_EAST</td>
<td>enable Eastern OB</td>
</tr>
<tr>
<td>ALLOW_OBCS_WEST</td>
<td>enable Western OB</td>
</tr>
<tr>
<td>ALLOW_OBCS_PREScribe</td>
<td>enable code for prescribing OB’s</td>
</tr>
<tr>
<td>ALLOW_OBCS_SPONGE</td>
<td>enable sponge layer code</td>
</tr>
<tr>
<td>ALLOW_OBCS_BALANCE</td>
<td>enable code for balancing transports through OB’s</td>
</tr>
<tr>
<td>ALLOW_ORLANSKI</td>
<td>enable Orlanski radiation conditions at OB’s</td>
</tr>
<tr>
<td>ALLOW_OBCS_STEVENS</td>
<td>enable Stevens (1990) boundary conditions at OB’s</td>
</tr>
<tr>
<td></td>
<td>(currently only implemented for eastern and</td>
</tr>
<tr>
<td></td>
<td>western boundaries and NOT for ptracers)</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Flag/parameter</th>
<th>default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>basic flags &amp; parameters (OBCS_PARM01)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OB_Jnorth</td>
<td>0</td>
<td>Nx-vector of J-indices (w.r.t. Ny) of Northern OB at each I-position (w.r.t. Nx)</td>
</tr>
<tr>
<td>OB_Jsouth</td>
<td>0</td>
<td>Nx-vector of J-indices (w.r.t. Ny) of Southern OB at each I-position (w.r.t. Nx)</td>
</tr>
<tr>
<td>OB_Ieast</td>
<td>0</td>
<td>Ny-vector of I-indices (w.r.t. Nx) of Eastern OB at each J-position (w.r.t. Ny)</td>
</tr>
<tr>
<td>OB_Iwest</td>
<td>0</td>
<td>Ny-vector of I-indices (w.r.t. Nx) of Western OB at each J-position (w.r.t. Ny)</td>
</tr>
<tr>
<td>useOBCSsprescribe</td>
<td>.FALSE.</td>
<td>factor(s) determining the details of the balancing code</td>
</tr>
<tr>
<td>useOBCSsponge</td>
<td>.FALSE.</td>
<td>turn on Orlanski boundary conditions for individual boundary</td>
</tr>
<tr>
<td>useOBCSbalance</td>
<td>code{.FALSE.}</td>
<td></td>
</tr>
<tr>
<td>OBCS_balanceFacN/S/E/W</td>
<td>1</td>
<td>factor(s) determining the details of the balacing code</td>
</tr>
<tr>
<td>useOrlanskiNorth/South/EastWest</td>
<td>.FALSE.</td>
<td>turn on Orlanski boundary conditions for individual boundary</td>
</tr>
<tr>
<td>useStevensNorth/South/EastWest</td>
<td>.FALSE.</td>
<td>turn on Stevens boundary conditions for individual boundary</td>
</tr>
<tr>
<td>OB_XyFile</td>
<td></td>
<td>file name of OB field</td>
</tr>
<tr>
<td>X: N(orth) S(outh) E(ast) W(est)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>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 salinity)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Orlanski parameters (OBCS_PARM02)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>cvelTimeScale</td>
<td>2000 sec</td>
<td>averaging period for phase speed</td>
</tr>
<tr>
<td>CMAX</td>
<td>0.45 m/s</td>
<td>maximum allowable phase speed-CFL for AB-II</td>
</tr>
<tr>
<td>CFIX</td>
<td>0.8 m/s</td>
<td>fixed boundary phase speed</td>
</tr>
<tr>
<td>useFixedCEast</td>
<td>.FALSE.</td>
<td></td>
</tr>
<tr>
<td>useFixedCWest</td>
<td>.FALSE.</td>
<td></td>
</tr>
<tr>
<td>Sponge-layer parameters (OBCS_PARM03)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>spongeThickness</td>
<td>0</td>
<td>sponge layer thickness (in grid points)</td>
</tr>
<tr>
<td>Urelaxobsinner</td>
<td>0 sec</td>
<td>relaxation time scale at the innermost sponge layer point of a meridional</td>
</tr>
<tr>
<td>Vrelaxobsinner</td>
<td>0 sec</td>
<td>relaxation time scale at the innermost sponge layer point of a zonal</td>
</tr>
<tr>
<td>Urelaxobsbound</td>
<td>0 sec</td>
<td>relaxation time scale at the outermost sponge layer point of a meridional</td>
</tr>
<tr>
<td>Vrelaxobsbound</td>
<td>0 sec</td>
<td>relaxation time scale at the outermost sponge layer point of a zonal</td>
</tr>
<tr>
<td>Stevens parameters (OBCS_PARM04)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TrelaxStevens</td>
<td>0 sec</td>
<td>relaxation time scale for temperature/salinity</td>
</tr>
<tr>
<td>useStevensPhaseVel</td>
<td>code{.TRUE.}</td>
<td></td>
</tr>
<tr>
<td>useStevensAdvection</td>
<td>code{.TRUE.}</td>
<td></td>
</tr>
</tbody>
</table>

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 $OB_{Jnorth}(N_x) / OB_{Jsouth}(N_x)$, and an $N_y$-dimensional “column” array $OB_{Ieast}(N_y) / OB_{Iwest}(N_y)$. 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 \(\text{OB} \_\text{I}.., \text{OB} \_\text{J}..\) 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, 

\[
\text{OB \_Jnorth}(3) = 34 \quad \Rightarrow \quad T(3, 34) \text{ is a an OB point, } U(3, 34) \text{ is a an OB point,} V(3, 34) \text{ is a an OB point}
\]

\[
\text{OB \_Jsouth}(3) = 1 \quad \Rightarrow \quad T(3, 1) \text{ is a an OB point, } U(3, 1) \text{ is a an OB point,} V(3, 2) \text{ is a an OB point}
\]

\[
\text{OB \_Ieast}(10) = 69 \quad \Rightarrow \quad T(69, 10) \text{ is a an OB point, } U(69, 10) \text{ is a an OB point,} V(69, 10) \text{ is a an OB point}
\]

\[
\text{OB \_Iwest}(10) = 1 \quad \Rightarrow \quad T(1, 10) \text{ is a an OB point, } U(2, 10) \text{ is a an OB point,} V(1, 10) \text{ is a an OB point}
\]

For convenience, negative values for \(J\_\text{north/least}\) refer to points relative to the Northern/Eastern edges of the model eg. \(\text{OB \_Jnorth}(3) = -1\) means that the point \((3, Ny)\) is a northern OB.

**Simple examples:** For a model grid with \(N_x \times N_y = 120 \times 144\) horizontal grid points with four open boundaries along the four edges of the domain, the simplest way of specifying the boundary points in is:

```plaintext
OB \_Ileast = 144*\text{-}1,
# or OB \_Ileast = 144*120,
OB \_Iwest = 144*1,
OB \_Jnorth = 120*\text{-}1,
# or OB \_Jnorth = 120*144,
OB \_Jsouth = 120*1,
```

If only the first 50 grid points of the southern boundary are boundary points:

```plaintext
OB \_Jsouth(1:50) = 50*1,
```

**Equations and key routines**

**OBCS\_READPARMS:**

Set OB positions through arrays \(\text{OB \_Jnorth}(Nx), \text{OB \_Jsouth}(Nx), \text{OB \_Ieast}(Ny), \text{OB \_Iwest}(Ny)\), and runtime flags (see Table [tab:pkg:obcs:runtime:sub:flags]).

**OBCS\_CALC:**

Top-level routine for filling values to be applied at OB for \(T, S, U, V, \eta\) into corresponding “slice” arrays \((x, z), (y, z)\) for each OB: \(\text{OB}[N_x/S/E/W][t/s/u/v]\); e.g. for salinity array at Southern OB, array name is \(\text{OBSt}\). 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 `useOBCS prescribe = .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 \((N_y \times N_r \times \text{time levels})\), for East/West boundary files the dimensions are \((N_x \times N_r \times \text{time levels})\).
- If a non-linear free surface is used (ref{sec:nonlinear-freesurface}), additional files `OB[N/S/E/W]etaFile` for the sea surface height \(\eta\) with dimension \((N_x \times N_y \times \text{time levels})\) may be specified.
- If non-hydrostatic dynamics are used (ref{sec:non-hydrostatic}), additional files `OB[N/S/E/W]wFile` for the vertical velocity \(w\) with dimensions \((N_x \times N_y \times \text{time levels})\) can be specified.
- If `useSEAICE=.TRUE.` then additional files `OB[N/S/E/W] [a,h,sl,sn,uice,vice]` for sea ice area, thickness (HEFF), sea ice salinity, snow and ice velocities \((N_x \times N_y \times \text{time levels})\) can be specified.

As in S/R `external\_fields\_load` or the `exf`-package, the code reads two time levels for each variable, e.g. `OBNu0` and `OBNu1`, and interpolates linearly between these time levels to obtain the value `OBNu` at the current model time (step). When the `exf`-package is used, the time levels are controlled for each boundary separately in the same way as the `exf-fields in data.exf`, namelist `EXF\_NML\_OBCS`. The runtime flags follow the above naming conventions, e.g. for the western boundary the corresponding flags are `OBWstartdate1/2` and `OBWperiod`. 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 `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 \(\chi_{ob}\) (note: passive tracers are currently not implemented and the code stops when package `ptracers` is used together with this option). Currently, the code vertically averages the normal velocity as specified in code\{OB[E,W][u]\} or code\{OB[N,S][v]\}. From these prescribed values the code computes the boundary values for the next timestep \(n+1\) as follows (as an example, we use the notation for an eastern or western boundary):

- \(u^{n+1}(y,z) = \bar{u}_{ob}(y) + (u')^n(y,z)\), where \((u')^n\) is the deviation from the vertically averaged velocity at timestep \(n\) on the boundary. \((u')^n\) is computed in the previous time step \(n\) from the intermediate velocity \(u^*\) prior to the correction step (see section [sec:time:sub:stepping], e.g., eq.\{eq:ustar-backward-free-surface\}). (This velocity is not available at the beginning of the next time step \(n+1\), when S/R OBCS_CALC/OBCS_CALC_STEVENS are called, therefore it needs to be saved in S/R DYNAMICS by calling S/R OBCS_SAVE_UV_N and also stored in a separate restart files `pickup_stevens[N/S/E/W].$(iteration).data`)  
  - If \(u^{n+1}\) is directed into the model domain, the boudary value for tracer \(\chi\) is restored to the prescribed values:
    \[
    \chi^{n+1} = \chi^n + \frac{\Delta t}{\tau_\chi} (\chi_{ob} - \chi^n),
    \]
    where \(\tau_\chi\) is the relaxation time scale \(T/SrelaxStevens\). The new \(\chi^{n+1}\) is then subject to the advection by \(u^{n+1}\).  
  - If \(u^{n+1}\) is directed out of the model domain, the tracer \(\chi^{n+1}\) on the boundary at timestep \(n+1\) is estimated from advection out of the domain with \(u^{n+1} + c\), where \(c\) is a phase velocity estimated as \(\frac{1}{2} \frac{\partial \chi}{\partial T} / \frac{\partial \chi}{\partial \tau}\). The numerical
scheme is (as an example for an eastern boundary):

\[
\chi_{i_b,j,k}^{n+1} = \chi_{i_b,j,k}^n + \Delta t (u_{i_b,j,k}^{n+1} + c) \frac{\chi_{i_b,j,k}^n - \chi_{i_b-1,j,k}^n}{\Delta x_{i_b,j}}, \text{ if } u_{i_b,j,k}^{n+1} > 0,
\]

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

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

### OBCS_BALANCE_FLOW:

When turned on (`ALLOW_OBCS_BALANCE` defined in `OBCS_OPTIONS.h` and `useOBCSbalance=.true.` in `data.obcs/OBCS_PARM01`), this routine balances the net flow across the open boundaries. By default the net flow across the boundaries is computed and all normal velocities on boundaries are adjusted to obtain zero net inflow.

This behavior can be controlled with the runtime flags `OBCS_balanceFacN/S/E/W`. The values of these flags determine how the net inflow is redistributed as small correction velocities between the individual sections. A value `-1` balances an individual boundary, values > 0 determine the relative size of the correction. For example, the values

\[
\text{OBCS_balanceFacE} = 1., \quad \text{OBCS_balanceFacW} = -1., \quad \text{OBCS_balanceFacN} = 2., \quad \text{OBCS_balanceFacS} = 0.,
\]

make the model

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

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

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

This also ensures a net total inflow of zero through all boundaries, but this combination of flags is not useful if you want to simulate, say, a sector of the Southern Ocean with a strong ACC entering through the western and leaving through the eastern boundary, because the value of "-1" for these flags will make sure that the strong inflow is removed. Clearly, global balancing with `OBCS_balanceFacE/W/N/S ≥ 0` is the preferred method.

### OBCS_APPLY_9*

### 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

\[
G_x^{(\text{sponge})} = -\frac{\chi - [(L - \delta L) \chi_{BC} + \delta L \chi]/L}{[(L - \delta L) \tau_b + \delta L \tau_1]/L} = -\frac{\chi - [(1 - l) \chi_{BC} + l \chi]}{[(1 - l) \tau_b + l \tau_1]}
\]
where $\chi$ is the model variable (U/V/T/S) in the interior, $\chi_{BC}$ the boundary value, $L$ the thickness of the sponge layer (runtime parameter spongeThickness in number of grid points), $\delta L \in [0, L]$ ($\delta L = l \in [0, 1]$) the distance from the boundary (also in grid points), and $\tau_o$ (runtime parameters Urelaxobcsbound and Vrelaxobcsbound) and $\tau_i$ (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
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].

<table>
<thead>
<tr>
<th>Name</th>
<th>Levs</th>
<th>grid</th>
<th>Units</th>
<th>Tile (max=80c)</th>
</tr>
</thead>
</table>

**Reference experiments**

In the directory verification, the following experiments use obcs:

- **exp4**: box with 4 open boundaries, simulating flow over a Gaussian bump based on , also tests Stevens-boundary conditions;
- **dome**: based on the project “Dynamics of Overflow Mixing and Entrainment” (http://www.rsmas.miami.edu/personal/tamay/DOME/dome.html), uses Orlanski-BCs;
- **internal_wave**: uses a heavily modified S/R-OBCS\_CALC
- **code:seaice_obcs**: simple example who to use the sea-ice related code, based on lab_sea;
- **tutorial_plume_on_slope**: uses Orlanski-BCs, see also section [sec:eg-gravityplume].

**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:

\[
\frac{dT}{dt} = \frac{dT}{dt} - M_{rbc} \tau_T (T - T_{rbc})
\]

where \( M_{rbc} \) is a 3-D mask (no time dependence) with values between 0 and 1. Where \( M_{rbc} = 1 \), relaxing timescale is \( 1/\tau_T \). Where it is 0 there is no relaxing. The value relaxed to is a 3-D (potentially varying in time) field given by \( T_{rbc} \).

A separate mask can be used for T,S and ptracers and each of these can be relaxed or not and can have its own timescale \( \tau_T \). 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, default 0); this is relative to time averages starting at \( t = 0 \), i.e., the first forcing record/file is placed at rbcsForcingOffset + 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 (\( \tau_T \) 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 \( T_{rbc} \) 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

<table>
<thead>
<tr>
<th>rbcsSingleTimeFiles = T</th>
<th>rbcsSingleTimeFiles = F</th>
</tr>
</thead>
<tbody>
<tr>
<td>model time</td>
<td>c = 0</td>
</tr>
<tr>
<td></td>
<td>c ≠ 0</td>
</tr>
<tr>
<td>file number</td>
<td>F</td>
</tr>
<tr>
<td></td>
<td>c ≠ 0</td>
</tr>
<tr>
<td>record</td>
<td></td>
</tr>
<tr>
<td>t₀ − p/2</td>
<td>i₀</td>
</tr>
<tr>
<td></td>
<td>i₀ + c/Δt₁₀bcs</td>
</tr>
<tr>
<td></td>
<td>c/p</td>
</tr>
<tr>
<td>t₀ + p/2</td>
<td>i₀ + p/Δt₁₀bcs</td>
</tr>
<tr>
<td></td>
<td>i₀ + p/Δt₁₀bcs</td>
</tr>
<tr>
<td></td>
<td>1</td>
</tr>
<tr>
<td>t₀ + p + p/2</td>
<td>i₀ + 2p/Δt₁₀bcs</td>
</tr>
<tr>
<td></td>
<td>i₀ + 2p/Δt₁₀bcs</td>
</tr>
<tr>
<td></td>
<td>2</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>t₀ + c − p/2</td>
<td>i₀ + c/Δt₁₀bcs</td>
</tr>
<tr>
<td></td>
<td>c/p</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td></td>
<td>...</td>
</tr>
</tbody>
</table>

where

\( p = \text{rbcsForcingPeriod} \)

\( c = \text{rbcsForcingCycle} \)

\( t₀ = \text{rbcsForcingOffset} \)

\( i₀ = \text{rbcsIter0} \)

\( \Delta t₁₀bcs = \text{deltaTrbcs} \)

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 \( t = \text{rbcsForcingPeriod} \).
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 not actively affect the physics of the ocean. Tracers are initialized, advected, diffused and various outputs are taken care of in this package. For methods to add additional sources and sinks of tracers use the pkg/gchem (section [sec:pkg:gchem]).

Can use up to 3843 tracers. But cannot 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 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 separate file.

The following can be set for each tracer (tracer number iTrc):
• \texttt{PTRACERS\_advScheme(iTrc)} will default to \texttt{saltAdvScheme} (set in data). For other options see Table \ref{tab:advectionShemes:sub:summary}.

• \texttt{PTRACERS\_ImplVertAdv(iTrc)}: implicit vertical advection flag, default to .FALSE.

• \texttt{PTRACERS\_diffKh(iTrc)}: horizontal Laplacian Diffusivity, defaults to \texttt{diffKhS} (set in data).

• \texttt{PTRACERS\_diffK4(iTrc)}: Biharmonic Diffusivity, defaults to \texttt{diffK4S} (set in data).

• \texttt{PTRACERS\_diffKr(iTrc)}: vertical diffusion, defaults to un-set.

• \texttt{PTRACERS\_diffKrNr(k,iTrc)}: level specific vertical diffusion, defaults to \texttt{diffKrNrS}. Will be set to \texttt{PTRACERS\_diffKr} if this is set.

• \texttt{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 \texttt{PTRACERS\_EvPrRn(iTrc)} is set and \texttt{convertFW2Salt} (set in data) is set to something other than -1 (note default is \texttt{convertFW2Salt}=35).

• \texttt{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.

• \texttt{PTRACERS\_useGMRedi(iTrc)}: apply GM or not. Defaults to \texttt{useGMRedi}.

• \texttt{PTRACERS\_useKPP(iTrc)}: apply KPP or not. Defaults to \texttt{useKPP}.

• \texttt{PTRACERS\_initialFile(iTrc)}: file with initial tracer concentration. Will be used if \texttt{PTRACERS\_Iter0}=nIter0. Default is no name, in which case tracer is initialised as zero. If \texttt{PTRACERS\_Iter0}<nIter0, then tracer concentration will come from \texttt{pickup_ptracer}.

• \texttt{PTRACERS\_names(iTrc)}: tracer name. Needed for netcdf. Defaults to nothing.

• \texttt{PTRACERS\_long_names(iTrc)}: optional name in long form of tracer.

• \texttt{PTRACERS\_units(iTrc)}: optional units of tracer.

\textbf{PTRACERS Diagnostics}

Note that these will only work for 90 or less tracers (some problems with the numbering/designation over this number)

\begin{verbatim}
<table>
<thead>
<tr>
<th>&lt;--Name--&gt;</th>
<th>Levs</th>
<th>&lt;--parsing code-- &gt;=</th>
<th>&lt;-- Units --&gt;</th>
<th>&lt;-- Tile (max=80c)</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRAC01</td>
<td>15</td>
<td>SM P MR</td>
<td>mol C/m</td>
<td>Mass-Weighted Dissolved Inorganic Carbon</td>
</tr>
<tr>
<td>UTRAC01</td>
<td>15</td>
<td>UU 171MR</td>
<td>mol C/m.m/s</td>
<td>Zonal Mass-Weighted Transp of Dissolved Inorganic Carbon</td>
</tr>
<tr>
<td>VTRAC01</td>
<td>15</td>
<td>VV 170MR</td>
<td>mol C/m.m/s</td>
<td>Merid Mass-Weighted Transp of Dissolved Inorganic Carbon</td>
</tr>
<tr>
<td>WTRAC01</td>
<td>15</td>
<td>WM MR</td>
<td>mol C/m.m/s</td>
<td>Vert Mass-Weighted Transp of Dissolved Inorganic Carbon</td>
</tr>
<tr>
<td>ADVrTr01</td>
<td>15</td>
<td>WM LR</td>
<td>mol C/m.m^3/s</td>
<td>Vertical Advection Flux of Dissolved Inorganic Carbon</td>
</tr>
<tr>
<td>ADVxTr01</td>
<td>15</td>
<td>UU 175MR</td>
<td>mol C/m.m^3/s</td>
<td>Zonal Advection Flux of Dissolved Inorganic Carbon</td>
</tr>
<tr>
<td>ADVyTr01</td>
<td>15</td>
<td>VV 174MR</td>
<td>mol C/m.m^3/s</td>
<td>Meridional Advection Flux of Dissolved Inorganic Carbon</td>
</tr>
<tr>
<td>DFrETr01</td>
<td>15</td>
<td>WM LR</td>
<td>mol C/m.m^3/s</td>
<td>Vertical Diffusive Flux of Dissolved Inorganic Carbon (Explicit part)</td>
</tr>
<tr>
<td>DIFxTr01</td>
<td>15</td>
<td>UU 178MR</td>
<td>mol C/m.m^3/s</td>
<td>Zonal Diffusive Flux of Dissolved Inorganic Carbon</td>
</tr>
<tr>
<td>DIFyTr01</td>
<td>15</td>
<td>VV 177MR</td>
<td>mol C/m.m^3/s</td>
<td>Meridional Diffusive Flux of Dissolved Inorganic Carbon</td>
</tr>
</tbody>
</table>
\end{verbatim}

5.3. General purpose numerical infrastructure packages

85
<table>
<thead>
<tr>
<th>Name</th>
<th>Levs</th>
<th>Parsing Code</th>
<th>Units</th>
<th>Tile (max=80c)</th>
</tr>
</thead>
<tbody>
<tr>
<td>\textit{DFrITr01}</td>
<td>15 LM</td>
<td>MR</td>
<td>mol C/m.m^3/s</td>
<td>Vertical Diffusive Flux of Dissolved Inorganic Carbon (Implicit part)</td>
</tr>
<tr>
<td>TRAC02</td>
<td>15 SM P</td>
<td>MR</td>
<td>mol eq/</td>
<td>Mass-Weighted Alkalinity</td>
</tr>
<tr>
<td>UTRAC02</td>
<td>15 UU 182MR</td>
<td></td>
<td>mol eq./m/s</td>
<td>Zonal Mass-Weighted Transp of Alkalinity</td>
</tr>
<tr>
<td>VTRAC02</td>
<td>15 VV 181MR</td>
<td></td>
<td>mol eq./m/s</td>
<td>Merid Mass-Weighted Transp of Alkalinity</td>
</tr>
<tr>
<td>WTRAC02</td>
<td>15 WM MR</td>
<td></td>
<td>mol eq./m/s</td>
<td>Vert Mass-Weighted Transp of Alkalinity</td>
</tr>
<tr>
<td>ADVrTr02</td>
<td>15 WM</td>
<td>LR</td>
<td>mol eq./m^3/s</td>
<td>Vertical Advective Flux of Alkalinity</td>
</tr>
<tr>
<td>\textit{DIFxTr02}</td>
<td>15 UU 189MR</td>
<td></td>
<td>mol eq./m^3/s</td>
<td>Zonal Diffusive Flux of Alkalinity</td>
</tr>
<tr>
<td>\textit{DIFyTr02}</td>
<td>15 VV 188MR</td>
<td></td>
<td>mol eq./m^3/s</td>
<td>Meridional Diffusive Flux of Alkalinity</td>
</tr>
<tr>
<td>\textit{DFrITr02}</td>
<td>15 WM LR</td>
<td>MR</td>
<td>mol eq./m^3/s</td>
<td>Vertical Diffusive Flux of Alkalinity (Implicit part)</td>
</tr>
<tr>
<td>TRAC03</td>
<td>15 SM P</td>
<td>MR</td>
<td>mol P/m</td>
<td>Mass-Weighted Phosphate</td>
</tr>
<tr>
<td>UTRAC03</td>
<td>15 UU 193MR</td>
<td></td>
<td>mol P./m.m/s</td>
<td>Zonal Mass-Weighted Transp of Phosphate</td>
</tr>
<tr>
<td>VTRAC03</td>
<td>15 VV 192MR</td>
<td></td>
<td>mol P./m.m/s</td>
<td>Merid Mass-Weighted Transp of Phosphate</td>
</tr>
<tr>
<td>WTRAC03</td>
<td>15 WM MR</td>
<td></td>
<td>mol P./m.m/s</td>
<td>Vert Mass-Weighted Transp of Phosphate</td>
</tr>
<tr>
<td>ADVrTr03</td>
<td>15 WM</td>
<td>LR</td>
<td>mol P./m.m^3/s</td>
<td>Vertical Advective Flux of Phosphate</td>
</tr>
<tr>
<td>\textit{DIFxTr03}</td>
<td>15 UU 197MR</td>
<td></td>
<td>mol P./m.m^3/s</td>
<td>Zonal Diffusive Flux of Phosphate</td>
</tr>
<tr>
<td>\textit{DIFyTr03}</td>
<td>15 VV 196MR</td>
<td></td>
<td>mol P./m.m^3/s</td>
<td>Meridional Diffusive Flux of Phosphate</td>
</tr>
<tr>
<td>\textit{DFrITr03}</td>
<td>15 WM LR</td>
<td>MR</td>
<td>mol P./m.m^3/s</td>
<td>Vertical Diffusive Flux of Phosphate (Explicit part)</td>
</tr>
<tr>
<td>TRAC04</td>
<td>15 SM P</td>
<td>MR</td>
<td>mol P/m</td>
<td>Mass-Weighted Dissolved Organic Phosphorus</td>
</tr>
<tr>
<td>UTRAC04</td>
<td>15 UU 204MR</td>
<td></td>
<td>mol P./m.m/s</td>
<td>Zonal Mass-Weighted Transp of Dissolved Organic Phosphorus</td>
</tr>
<tr>
<td>VTRAC04</td>
<td>15 VV 203MR</td>
<td></td>
<td>mol P./m.m/s</td>
<td>Merid Mass-Weighted Transp of Dissolved Organic Phosphorus</td>
</tr>
<tr>
<td>WTRAC04</td>
<td>15 WM</td>
<td>MR</td>
<td>mol P./m.m/s</td>
<td>Vert Mass-Weighted Transp of Dissolved Organic Phosphorus</td>
</tr>
<tr>
<td>ADVrTr04</td>
<td>15 WM</td>
<td>LR</td>
<td>mol P./m.m^3/s</td>
<td>Vertical Advective Flux of Dissolved Organic Phosphorus</td>
</tr>
<tr>
<td>\textit{DIFxTr04}</td>
<td>15 UU 200MR</td>
<td></td>
<td>mol P./m.m^3/s</td>
<td>Zonal Diffusive Flux of Dissolved Organic Phosphorus</td>
</tr>
<tr>
<td>\textit{DIFyTr04}</td>
<td>15 VV 199MR</td>
<td></td>
<td>mol P./m.m^3/s</td>
<td>Meridional Advective Flux of Dissolved Organic Phosphorus</td>
</tr>
<tr>
<td>\textit{DFrETr04}</td>
<td>15 WM LR</td>
<td>MR</td>
<td>mol P./m.m^3/s</td>
<td>Vertical Diffusive Flux of Dissolved Organic Phosphorus (Explicit part)</td>
</tr>
</tbody>
</table>
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 scheme is susceptible to a computational mode. Due to this mode, the Cox scheme requires a background lateral diffusion to be present to conserve the integrity of the model fields.

The GM parameterization was then added to the GFDL code in the form of a non-divergent bolus velocity. The method defines two stream-functions expressed in terms of the isoneutral slopes subject to the boundary condition of zero value on upper and lower boundaries. The horizontal bolus velocities are then the vertical derivative of these functions. Here in lies a problem highlighted by [GGP+98]: the bolus velocities involve multiple derivatives on the potential density field, which can consequently give rise to noise. Griffies et al. point out that the GM bolus fluxes can be identically written as a skew flux which involves fewer differential operators. Further, combining the skew flux formulation and
Redi scheme, substantial cancellations take place to the point that the horizontal fluxes are unmodified from the lateral diffusion parameterization.

**Redi scheme: Isopycnal diffusion**

The Redi scheme diffuses tracers along isopycnals and introduces a term in the tendency (rhs) of such a tracer (here $\tau$) of the form:

$$\nabla \cdot \kappa_{\text{Redi}} \nabla \tau$$

where $\kappa_{\text{Redi}}$ is the along isopycnal diffusivity and $K_{\text{Redi}}$ is a rank 2 tensor that projects the gradient of $\tau$ onto the isopycnal surface. The unapproximated projection tensor is:

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

Here, $S_x = -\partial_x \sigma / \partial z \sigma$ and $S_y = -\partial_y \sigma / \partial z \sigma$ are the components of the isoneutral slope.

The first point to note is that a typical slope in the ocean interior is small, say of the order $10^{-4}$. A maximum slope might be of order $10^{-2}$ and only exceeds such in unstratified regions where the slope is ill defined. It is therefore justifiable, and customary, to make the small slope approximation, $|S| \ll 1$. The Redi projection tensor then becomes:

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

**GM parameterization**

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

$$-\nabla \cdot \tau u^*$$

The bolus velocity $u^*$ is defined as the rotational of a streamfunction $F^*=(F_x^*, F_y^*, 0)$:

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

and thus is automatically non-divergent. In the GM parameterization, the streamfunction is specified in terms of the isoneutral slopes $S_x$ and $S_y$:

$$F_x^* = -\kappa_{GM} S_y$$
$$F_y^* = \kappa_{GM} S_x$$

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

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

This is the form of the GM parameterization as applied by Donabasaglu, 1997, in MOM versions 1 and 2.

Note that in the MITgcm, the variables containing the GM bolus streamfunction are:

$$\begin{pmatrix} GM_PsiX \\ GM_PsiY \end{pmatrix} = \begin{pmatrix} \kappa_{GM} S_x \\ \kappa_{GM} S_y \end{pmatrix} = \begin{pmatrix} F_x^* \\ -F_y^* \end{pmatrix}.$$
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:

\[ \mathbf{u}^* \tau = \begin{pmatrix} -\partial_z (\kappa_{GM} S_x \tau) \\ -\partial_z (\kappa_{GM} S_y \tau) \\ \partial_z (\kappa_{GM} S_x \tau) + \partial_y (\kappa_{GM} S_y \tau) \end{pmatrix} \]

\[ + \begin{pmatrix} (\partial_z \kappa_{GM} S_x + \partial_y \kappa_{GM} S_y) \tau \\ \kappa_{GM} S_y \partial_x \tau \\ -\kappa_{GM} S_x \partial_y \tau - \kappa_{GM} S_y \partial_y \tau \end{pmatrix} \]

The first vector is non-divergent and thus has no effect on the tracer field and can be dropped. The remaining flux can be written:

\[ \mathbf{u}^* \tau = -\kappa_{GM} \mathbf{K}_{GM} \nabla \tau \]

where

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

is an anti-symmetric tensor.

This formulation of the GM parameterization involves fewer derivatives than the original and also involves only terms that already appear in the Redi mixing scheme. Indeed, a somewhat fortunate cancellation becomes apparent when we use the GM parameterization in conjunction with the Redi isoneutral mixing scheme:

\[ \kappa_{R} \mathbf{K}_{Redi} \nabla \tau - \mathbf{u}^* \tau = (\kappa_{R} + \kappa_{GM} \mathbf{K}_{GM}) \nabla \tau \]

In the instance that \( \kappa_{GM} = \kappa_{R} \) then

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

which differs from the variable Laplacian diffusion tensor by only two non-zero elements in the \( z \)-row.

Subroutine

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

\( \sigma_z \): SlopeX (argument on entry)

\( \sigma_y \): SlopeY (argument on entry)

\( \sigma_z \): SlopeY (argument)

\( S_x \): SlopeX (argument on exit)

\( S_y \): SlopeY (argument on exit)
Variable $\kappa_{GM}$

[VMH97] suggest making the eddy coefficient, $\kappa_{GM}$, a function of the Eady growth rate, $|f|/\sqrt{R_i}$. The formula involves a non-dimensional constant, $\alpha$, and a length-scale $L$:

$$\kappa_{GM} = \alpha L^2 \frac{|f|}{\sqrt{R_i}}$$

where the Eady growth rate has been depth averaged (indicated by the over-line). A local Richardson number is defined $R_i = N^2/(\partial u/\partial z)^2$ which, when combined with thermal wind gives:

$$\frac{1}{R_i} = \left(\frac{\partial u}{\partial z}\right)^2 = \left(\frac{\partial v}{\partial z}\right)^2 = \frac{M^2}{|f|^2 N^2}$$

where $M^2$ is defined $M^2 = \frac{\partial}{\partial z} |\nabla \sigma|$. Substituting into the formula for $\kappa_{GM}$ gives:

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

Tapering and stability

Experience with the GFDL model showed that the GM scheme has to be matched to the convective parameterization. This was originally expressed in connection with the introduction of the KPP boundary layer scheme [LMD94] but in fact, as subsequent experience with the MIT model has found, is necessary for any convective parameterization.

Subroutine

S/R GMREDI_SLOPE_LIMIT (pkg/gmredi/gmredi_slope_limit.F)

$\sigma_x$, $s_x$: SlopeX (argument)

$\sigma_y$, $s_y$: SlopeY (argument)

$\sigma_z$: dSigmadRReal (argument)

$\sigma^*_z$: 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:

$$|\nabla \sigma| = \sqrt{\sigma_x^2 + \sigma_y^2}$$

$$S_{lim} = \frac{|\nabla \sigma|}{S_{max}}$$

where $S_{max}$ is a parameter

$$\sigma_0^* = \min(\sigma_z, S_{lim})$$

$$[s_x, s_y] = \frac{[\sigma_x, \sigma_y]}{\sigma_z^*}$$
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 \(\kappa_p K_{33} = \kappa_p S_{max}^2\)) 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.


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`. 

92 Chapter 5. Physical Parameterizations - Packages I
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 $c = 2 \text{ m} \cdot \text{s}^{-1}$. This tapering with height was introduced to fix some spurious interaction with the mixed-layer KPP parameterization.

The LDD97 tapering scheme is activated in the model by setting `GM_taper_scheme = 'ldd97'` in `data.gmredi`.

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**

<table>
<thead>
<tr>
<th>Name</th>
<th>Levs</th>
<th>Parsing code</th>
<th>Units</th>
<th>Tile (max=80c)</th>
</tr>
</thead>
<tbody>
<tr>
<td>GM_VisbK</td>
<td>1</td>
<td>SM P M1</td>
<td>m^2/s</td>
<td>Mixing coefficient from Visbeck et al.</td>
</tr>
<tr>
<td>GM_Kux</td>
<td>15</td>
<td>UU P 177MR</td>
<td>m^2/s</td>
<td>K_11 element (U.point, X.dir) of GM-Redi tensor</td>
</tr>
<tr>
<td>GM_Kvy</td>
<td>15</td>
<td>VV P 176MR</td>
<td>m^2/s</td>
<td>K_22 element (V.point, Y.dir) of GM-Redi tensor</td>
</tr>
<tr>
<td>GM_Kuz</td>
<td>15</td>
<td>UU 179MR</td>
<td>m^2/s</td>
<td>K_13 element (U.point, Z.dir) of GM-Redi tensor</td>
</tr>
<tr>
<td>GM_Kvz</td>
<td>15</td>
<td>VV 178MR</td>
<td>m^2/s</td>
<td>K_23 element (V.point, Z.dir) of GM-Redi tensor</td>
</tr>
<tr>
<td>GM_Kwx</td>
<td>15</td>
<td>UM 181LR</td>
<td>m^2/s</td>
<td>K_31 element (U.point, X.dir) of GM-Redi tensor</td>
</tr>
<tr>
<td>GM_Kwy</td>
<td>15</td>
<td>VM 180LR</td>
<td>m^2/s</td>
<td>K_32 element (W.point, Y.dir) of GM-Redi tensor</td>
</tr>
<tr>
<td>GM_Kwz</td>
<td>15</td>
<td>WM P LR</td>
<td>m^2/s</td>
<td>K_33 element (W.point, Z.dir) of GM-Redi tensor</td>
</tr>
<tr>
<td>GM_PsiX</td>
<td>15</td>
<td>UU 184LR</td>
<td>m^2/s</td>
<td>GM Bolus transport stream-function : X component</td>
</tr>
<tr>
<td>GM_PsiY</td>
<td>15</td>
<td>VV 183LR</td>
<td>m^2/s</td>
<td>GM Bolus transport stream-function : Y component</td>
</tr>
<tr>
<td>GM_KuzTz</td>
<td>15</td>
<td>UU 186MR</td>
<td>degC.m^3/s</td>
<td>Redi Off-diagonal Temperature flux: X component</td>
</tr>
<tr>
<td>GM_KvzTz</td>
<td>15</td>
<td>VV 185MR</td>
<td>degC.m^3/s</td>
<td>Redi Off-diagonal Temperature flux: Y component</td>
</tr>
</tbody>
</table>

**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 governed by shear instability (modeled as function of the local gradient Richardson number), internal wave activity (assumed constant), and double-diffusion (not implemented here).
2. a boundary layer depth $h$ or $h_{bl}$ is determined at each grid point, based on a critical value of turbulent processes parameterized by a bulk Richardson number;
3. mixing is strongly enhanced in the boundary layer under the stabilizing or destabilizing influence of surface forcing (buoyancy and momentum) enabling boundary layer properties to penetrate well into the thermocline; mixing is represented through a polynomial profile whose coefficients are determined subject to several 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 $\gamma \text{ or } \hat{\gamma}$ 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

<table>
<thead>
<tr>
<th>CPP option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>_KPP_RL</td>
<td></td>
</tr>
<tr>
<td>FRUGAL_KPP</td>
<td></td>
</tr>
<tr>
<td>KPP_SMOOTH_SHSQ</td>
<td></td>
</tr>
<tr>
<td>KPP_SMOOTH_DVSQ</td>
<td></td>
</tr>
<tr>
<td>KPP_SMOOTH_DENS</td>
<td></td>
</tr>
<tr>
<td>KPP_SMOOTH_VISC</td>
<td></td>
</tr>
<tr>
<td>KPP_SMOOTH_DIFF</td>
<td></td>
</tr>
<tr>
<td>KPP_ESTIMATE_UREF</td>
<td></td>
</tr>
<tr>
<td>INCLUDE_DIAGNOSTICS_INTERFACE_CODE</td>
<td></td>
</tr>
<tr>
<td>KPP_GHAT</td>
<td></td>
</tr>
<tr>
<td>EXCLUDE_KPP_SHEAR_MIX</td>
<td></td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Flag/parameter</th>
<th>default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>I/O related parameters</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>kpp_freq</td>
<td>deltaTClock</td>
<td>Recomputation frequency for KPP fields</td>
</tr>
<tr>
<td>kpp_dumpFreq</td>
<td>dumpFreq</td>
<td>Dump frequency of KPP field snapshots</td>
</tr>
<tr>
<td>kpp_taveFreq</td>
<td>taveFreq</td>
<td>Averaging and dump frequency of KPP fields</td>
</tr>
<tr>
<td>KPPmixingMaps</td>
<td>.FALSE.</td>
<td>include KPP diagnostic maps in STDOUT</td>
</tr>
</tbody>
</table>

Continued on next page
Table 5.5 – continued from previous page

<table>
<thead>
<tr>
<th>Flag/parameter</th>
<th>default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>KPPwriteState</td>
<td>.FALSE.</td>
<td>write KPP state to file</td>
</tr>
<tr>
<td>KPP_ghatUseTotalDiffus</td>
<td>.FALSE.</td>
<td>if .T. compute non-local term using total vertical diffusivity</td>
</tr>
<tr>
<td></td>
<td></td>
<td>if .F. use KPP vertical diffusivity</td>
</tr>
</tbody>
</table>

**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 turbulent velocity shear (= function of concv,concs,epsilon,vonk,Ricr) |

**Boundary layer mixing parameters (S/R blmix)**

| cstar                           | 10.       | proportionality coefficient for non-local 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-Väisälä squared                                                       |
| difm0                           | 0.005 m² s⁻¹| viscosity max. due to shear instability                                      |
| difs0                           | 0.005 m²/s | tracer diffusivity max. due to shear instability                             |
| dift0                           | 0.005 m²/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>
<thead>
<tr>
<th>Flag/parameter</th>
<th>default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rrho0</td>
<td>not used</td>
<td>limit for double diffusive density ratio</td>
</tr>
<tr>
<td>dsfmax</td>
<td>not used</td>
<td>maximum diffusivity in case of salt fingering</td>
</tr>
</tbody>
</table>

### Equations and Key Routines

We restrict ourselves to writing out only the essential equations that relate to main processes and parameters mentioned above. We closely follow the notation of [LMD94].

**KPP_CALC:**

Top-level routine.

**KPP_MIX:**

Intermediate-level routine

**BLMIX: Mixing in the Boundary Layer**

The vertical fluxes $\overline{w\sigma}$ of momentum and tracer properties $X$ is composed of a gradient-flux term (proportional to the vertical property divergence $\partial_z X$), and a “nonlocal” term $\gamma_x$ that enhances the gradient-flux mixing coefficient $K_x$

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

- **Boundary layer mixing profile** It is expressed as the product of the boundary layer depth $h$, a depth-dependent turbulent velocity scale $w_x(\sigma)$ and a non-dimensional shape function $G(\sigma)$

$$
K_x(\sigma) = h \, w_x(\sigma) \, G(\sigma)
$$

with dimensionless vertical coordinate $\sigma = d/h$. For details of `w_x(sigma)` and $G(\sigma)$ we refer to .

- **Nonlocal mixing term** The nonlocal transport term $\gamma$ is nonzero only for tracers in unstable (convective) forcing conditions. Thus, depending on the stability parameter $\zeta = d/L$ (with depth $d$, Monin-Obukhov length scale $L$) it has the following form:

$$
\begin{align*}
\gamma_x &= 0 & \zeta &\geq 0 \\
\gamma_m &= 0 \\
\gamma_s &= C_s \frac{\overline{w\sigma}}{w_x(\sigma)h} & \zeta &< 0 \\
\gamma_\theta &= C_s \frac{\overline{w\theta} + w\bar{\theta}_zh}{w_x(\sigma)h}
\end{align*}
$$

In practice, the routine performs the following tasks:

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

5.4. Ocean Packages
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, $h_{bl}$, is determined as the shallowest depth where the bulk Richardson number is equal to the critical value, $R_{icr}$.

Bulk Richardson numbers are evaluated by computing velocity and buoyancy differences between values at $z_{grid}(k)$ < 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 $\epsilon*z_{grid}(k)$.

When the bulk Richardson number at $k$ exceeds $R_{icr}$, $h_{bl}$ is linearly interpolated between grid levels $z_{grid}(k)$ and $z_{grid}(k-1)$.

The water column and the surface forcing are diagnosed for stable/ustable forcing conditions, and where $h_{bl}$ 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 ($g_{hat}$) 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    |
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)
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>KPPviscA</td>
<td>23</td>
<td>SM</td>
<td>m^2/s</td>
<td>KPP vertical eddy viscosity coefficient</td>
</tr>
<tr>
<td>KPPdiffS</td>
<td>23</td>
<td>SM</td>
<td>m^2/s</td>
<td>Vertical diffusion coefficient for salt &amp; tracers</td>
</tr>
<tr>
<td>KPPdiffT</td>
<td>23</td>
<td>SM</td>
<td>m^2/s</td>
<td>Vertical diffusion coefficient for heat</td>
</tr>
<tr>
<td>KPPghat</td>
<td>23</td>
<td>SM</td>
<td>s/m^2</td>
<td>Nonlocal transport coefficient</td>
</tr>
<tr>
<td>KPPhbl</td>
<td>1</td>
<td>SM</td>
<td>m</td>
<td>KPP boundary layer depth, bulk Ri criterion</td>
</tr>
<tr>
<td>KPPmld</td>
<td>1</td>
<td>SM</td>
<td>m</td>
<td>Mixed layer depth, dT=.8degC density criterion</td>
</tr>
<tr>
<td>KPPfrac</td>
<td>1</td>
<td>SM</td>
<td></td>
<td>Short-wave flux fraction penetrating mixing layer</td>
</tr>
</tbody>
</table>

Reference experiments

lab_sea:
natl_box:

5.4. Ocean Packages
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 $\epsilon$, 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, \( \delta_z \), in an overturn [Tho77]. The Thorpe displacement is the distance one would have to move a water parcel for the water column to be stable, and is readily measured in a measured profile by sorting the profile and tracking how far each parcel moves during the sorting procedure. This method gives an imperfect estimate of the turbulence, but it has been found to agree on average over a large range of overturns [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 \( \epsilon \) 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 \( \epsilon_v \) will start to be exaggerated, particularly if the run in non-hydrostatic. That is because there will be significant shear at small scales that represents the turbulence being parameterized in the scheme. At very high resolutions direct numerical simulation or more sophisticated large-eddy schemes should be used.
- We find that grid cells of approximately 10 to 1 aspect ratio are a good rule of thumb for achieving good results are usual oceanic scales. For a site like the Hawaiian Ridge, and Luzon Strait, this means 10-m vertical resolution and approximately 100-m horizontal. The 10-m resolution can be relaxed if the stratification drops, and we often WKB-stretch the grid spacing with depth.
- The dissipation estimate is useful for 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:

<table>
<thead>
<tr>
<th>Flag/parameter</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>implicitViscosity</td>
<td>.TRUE.</td>
<td>enable implicit vertical viscosity</td>
</tr>
<tr>
<td>implicitDiffusion</td>
<td>.TRUE.</td>
<td>enable implicit vertical diffusion</td>
</tr>
</tbody>
</table>

Package flags and parameters

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

<table>
<thead>
<tr>
<th>Flag/parameter</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>KLviscMax</td>
<td>300 m s$^{-1}$</td>
<td>Maximum viscosity the scheme will ever give (useful for stability)</td>
</tr>
<tr>
<td>KLDumpFreq</td>
<td>dumpFreq</td>
<td>Dump frequency of KL10 field snapshots</td>
</tr>
<tr>
<td>KLtaveFreq</td>
<td>taveFreq</td>
<td>Averaging and dump frequency of KL10 fields</td>
</tr>
<tr>
<td>KLwriteState</td>
<td>.FALSE.</td>
<td>write KL10 state to file</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>&lt;-Name-&gt;</th>
<th>Levs</th>
<th>grid</th>
<th>&lt;- Units --&gt;</th>
<th>&lt;- Tile (max=80c)</th>
</tr>
</thead>
<tbody>
<tr>
<td>KLviscAr</td>
<td>Nr</td>
<td>SM</td>
<td>m^2/s</td>
<td>KL10 vertical eddy viscosity coefficient</td>
</tr>
<tr>
<td>KLdiffKr</td>
<td>Nr</td>
<td>SM</td>
<td>m^2/s</td>
<td>Vertical diffusion coefficient for salt, temperature, &amp; tracers</td>
</tr>
<tr>
<td>KLeps</td>
<td>Nr</td>
<td>SM</td>
<td>m^3/s^3</td>
<td>Turbulence dissipation estimate.</td>
</tr>
</tbody>
</table>

References


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.

5.4. Ocean Packages
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 \(\gamma\) is a constant.

The saturated vapor pressure is calculated (QQ ref):

\[ q_{sat} = \frac{a}{p_o} e^{L(b - \frac{x_{surf}}{T_{surf}})} \]

where \(a, b, c\) are constants, \(T_{surf}\) is surface temperature and \(p_o\) is the surface pressure.

The two values crucial for the bulk formula calculations are the difference between air at sea surface and sea surface temperature:

\[ \Delta T = T_{air} - T_{surf} + \alpha h_T \]

where \(\alpha\) is adiabatic lapse rate and \(h_T\) is the height where the air temperature was taken; and the difference between the air humidity and the saturated humidity

\[ \Delta q = q_{air} - q_{sat}. \]
We then calculate the turbulent exchange coefficients following Bryan et al (1996) and the numerical scheme of Hunke and Lipscombe (1998). We estimate initial values for the exchange coefficients, \( c_u \), \( c_T \) and \( c_q \) as

\[
\frac{\kappa}{\ln(z_{ref}/z_{rou})}
\]

where \( \kappa \) is the Von Karman constant, \( z_{ref} \) is a reference height and \( z_{rou} \) is a roughness length scale which could be a function of type of surface, but is here set as a constant. Turbulent scales are:

\[
\begin{align*}
    u^* &= c_u u_s \\
    T^* &= c_T \Delta T \\
    q^* &= c_q \Delta q
\end{align*}
\]

We find the “integrated flux profile” for momentum and stability if there are stable QQ conditions (\( \Upsilon > 0 \)):

\[
\psi_m = -5 \Upsilon
\]

and for unstable QQ conditions (\( \Upsilon < 0 \)):

\[
\begin{align*}
    \psi_m &= 2 \ln(0.5(1 + \chi)) + \ln(0.5(1 + \chi^2)) - 2 \tan^{-1} \chi + \pi/2 \\
    \psi_s &= 2 \ln(0.5(1 + \chi^2))
\end{align*}
\]

where

\[
\Upsilon = \frac{\kappa g z_{ref}}{u^*} \left( \frac{T^*}{T_o} + \frac{q^*}{1/\gamma + q_a} \right)
\]

and \( \chi = (1 - 16 \Upsilon)^{1/2} \).

The coefficients are updated through 5 iterations as:

\[
\begin{align*}
    c_u &= \frac{c_u}{1 + c_u (\lambda - \psi_m) / \kappa} \\
    c_T &= \frac{c_T}{1 + c_T (\lambda - \psi_s) / \kappa} \\
    c_q &= \frac{c_q}{1 + c_q (\lambda - \psi_s) / \kappa}
\end{align*}
\]

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

\[
Q_{up}^{lw} = \epsilon \sigma T_{surf}^4
\]

where \( \epsilon \) is emissivity (which can be different for open ocean, ice and snow), \( \sigma \) is Stefan-Boltzmann constant.

We calculate the derivatives of the three above functions with respect to surface temperature

\[
\begin{align*}
    \frac{dQ_s}{dT} &= \rho_{air} c_{p,air} u_s c_u c_T \\
    \frac{dQ_l}{dT} &= \rho_{air} L^2 u_s c_u c_q \frac{c_q}{T_{surf}} \\
    \frac{dQ_{up}^{lw}}{dT} &= 4 \epsilon \sigma T_{surf}^3
\end{align*}
\]

5.4. Ocean Packages

105
And total derivative \( \frac{dQ}{dt} = \frac{dQ_s}{dt} + \frac{dQ_l}{dt} + \frac{dQ_{up}}{dt} \).

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**


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:units], 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_CPOPTIONS.h. Table 5.7 summarizes these options.

Table 5.7: EXF CPP options

<table>
<thead>
<tr>
<th>CPP option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXF_VERBOSE</td>
<td>verbose mode (recommended only for testing)</td>
</tr>
<tr>
<td>ALLOW_ATM_TEMP</td>
<td>compute heat/freshwater fluxes from atmos. state input</td>
</tr>
<tr>
<td>ALLOW_ATM_WIND</td>
<td>compute wind stress from wind speed input</td>
</tr>
<tr>
<td>ALLOW_BULKFORMULAE</td>
<td>is used if ALLOW_ATM_TEMP or ALLOW_ATM_WIND is enabled</td>
</tr>
<tr>
<td>EXF_READ_EVAP</td>
<td>read evaporation instead of computing it</td>
</tr>
<tr>
<td>ALLOW_RUNOFF</td>
<td>read time-constant river/glacier run-off field</td>
</tr>
<tr>
<td>ALLOW_DOWNWARD_RADIATION</td>
<td>compute net from downward or downward from net radiation</td>
</tr>
<tr>
<td>USE_EXF_INTERPOLATION</td>
<td>enable on-the-fly bilinear or bicubic interpolation of input fields</td>
</tr>
<tr>
<td>used in conjunction with relaxation to prescribed (climatological) fields</td>
<td></td>
</tr>
<tr>
<td>ALLOW_CLIMSSST_RELAXATION</td>
<td>relaxation to 2-D SST climatology</td>
</tr>
<tr>
<td>ALLOW_CLIMSSS_RELAXATION</td>
<td>relaxation to 2-D SSS climatology</td>
</tr>
<tr>
<td>these are set outside of EXF in CPP_OPTIONS.h</td>
<td></td>
</tr>
<tr>
<td>SHORTWAVE_HEATING</td>
<td>enable shortwave radiation</td>
</tr>
<tr>
<td>ATMOSPHERIC_LOADING</td>
<td>enable surface pressure forcing</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Flag/parameter</th>
<th>default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>useExfCheckRange</td>
<td>.TRUE.</td>
<td>check range of input fields and stop if out of range</td>
</tr>
<tr>
<td>useExfYearlyFields</td>
<td>.FALSE.</td>
<td>append current year postfix of form _YYYY on filename</td>
</tr>
<tr>
<td>twoDigitYear</td>
<td>.FALSE.</td>
<td>instead of appending _YYYY append YY</td>
</tr>
<tr>
<td>repeatPeriod</td>
<td>0.0</td>
<td>0: cycle through all input fields at the same period (in seconds)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>= 0: use period assigned to each field</td>
</tr>
<tr>
<td>exf_offset_atemp</td>
<td>0.0</td>
<td>set to 273.16 to convert from deg. Kelvin (assumed input) to Celsius</td>
</tr>
<tr>
<td>windstressmax</td>
<td>2.0</td>
<td>max. allowed wind stress N m$^{-2}$</td>
</tr>
<tr>
<td>exf_albedo</td>
<td>0.1</td>
<td>surface albedo used to compute downward vs. net radiative fluxes</td>
</tr>
<tr>
<td>climtempfreeze</td>
<td>-1.9</td>
<td>??</td>
</tr>
<tr>
<td>ocean_emissivity</td>
<td></td>
<td>longwave ocean-surface emissivity</td>
</tr>
<tr>
<td>ice_emissivity</td>
<td></td>
<td>longwave seaice emissivity</td>
</tr>
<tr>
<td>snow_emissivity</td>
<td></td>
<td>longwave snow emissivity</td>
</tr>
<tr>
<td>exf_iceCd</td>
<td>1.63E-3</td>
<td>drag coefficient over sea-ice</td>
</tr>
<tr>
<td>exf_iceCe</td>
<td>1.63E-3</td>
<td>evaporation transfer coeff. over sea-ice</td>
</tr>
<tr>
<td>exf_iceCh</td>
<td>1.63E-3</td>
<td>sensible heat transfer coeff. over sea-ice</td>
</tr>
<tr>
<td>exf_scal_BulkCdn</td>
<td>1.</td>
<td>overall scaling of neutral drag coeff.</td>
</tr>
<tr>
<td>useStabilityFct_overIce</td>
<td>.FALSE.</td>
<td>compute turbulent transfer coeff. over sea-ice</td>
</tr>
<tr>
<td>readStressOnAgrid</td>
<td>.FALSE.</td>
<td>read wind-stress located on model-grid, A-grid point</td>
</tr>
<tr>
<td>readStressOnCgrid</td>
<td>.FALSE.</td>
<td>read wind-stress located on model-grid, C-grid point</td>
</tr>
<tr>
<td>useRelativeWind</td>
<td>.FALSE.</td>
<td>subtract [U/V]VEL or [U/VICE from U/V]WIND before computing [U/V]STRESS</td>
</tr>
<tr>
<td>zref</td>
<td>10.</td>
<td>reference height</td>
</tr>
</tbody>
</table>

Continued on next page
### Table 5.8 – continued from previous page

<table>
<thead>
<tr>
<th>Flag/parameter</th>
<th>default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hu</td>
<td>10.</td>
<td>height of mean wind</td>
</tr>
<tr>
<td>ht</td>
<td>2.</td>
<td>height of mean temperature and rel. humidity</td>
</tr>
<tr>
<td>umin</td>
<td>0.5</td>
<td>minimum absolute wind speed for computing Cd</td>
</tr>
<tr>
<td>atmrho</td>
<td>1.2</td>
<td>mean atmospheric density [kg/m^3]</td>
</tr>
<tr>
<td>atmcp</td>
<td>1005.</td>
<td>mean atmospheric specific heat [J/kg/K]</td>
</tr>
<tr>
<td>cdrag_[n]</td>
<td>???</td>
<td>n = 1,2,3; parameters for drag coeff. function</td>
</tr>
<tr>
<td>cstanton_[n]</td>
<td>???</td>
<td>n = 1,2; parameters for Stanton number function</td>
</tr>
<tr>
<td>cdalton</td>
<td>???</td>
<td>parameter for Dalton number function</td>
</tr>
<tr>
<td>flamb</td>
<td>2500000.</td>
<td>latent heat of evaporation [J/kg]</td>
</tr>
<tr>
<td>flami</td>
<td>334000.</td>
<td>latent heat of melting of pure ice [J/kg]</td>
</tr>
<tr>
<td>zolmin</td>
<td>-100.</td>
<td>minimum stability parameter</td>
</tr>
<tr>
<td>cvapor_fac</td>
<td>640380.</td>
<td></td>
</tr>
<tr>
<td>cvapor_exp</td>
<td>5107.4</td>
<td></td>
</tr>
<tr>
<td>cvapor_fac_ice</td>
<td>11637800.</td>
<td></td>
</tr>
<tr>
<td>cvapor_fac_ice</td>
<td>5897.8</td>
<td></td>
</tr>
<tr>
<td>humid_fac</td>
<td>0.606</td>
<td>parameter for virtual temperature calculation</td>
</tr>
<tr>
<td>gamma_blk</td>
<td>0.010</td>
<td>adiabatic lapse rate</td>
</tr>
<tr>
<td>saltsat</td>
<td>0.980</td>
<td>reduction of saturation vapor pressure over salt-water</td>
</tr>
<tr>
<td>psim_fac</td>
<td>5.</td>
<td></td>
</tr>
<tr>
<td>exf_monFreq</td>
<td>monitorFreq</td>
<td>output frequency [s]</td>
</tr>
<tr>
<td>exf_iprec</td>
<td>32</td>
<td>precision of input fields (32-bit or 64-bit)</td>
</tr>
<tr>
<td>exf_yftype</td>
<td>‘RL’</td>
<td>precision of arrays (‘RL’ vs. ‘RS’)</td>
</tr>
</tbody>
</table>

### Field attributes

All EXF fields are listed in Section [sec:pkg:exf:fields:sub:units]. Each field has a number of attributes which can be customized. They are summarized in Table [tab:pkg:exf:runtime:sub:attributes]. To obtain an attribute for a specific
field, e.g. \textit{uwind} prepend the field name to the listed attribute, e.g. for attribute \textit{period} this yields \textit{uwindperiod}:

\[
\text{field} & \quad \text{attribute} & \rightarrow & \quad \text{parameter} \\
\text{e.g.} & \quad \text{uwind} & \quad \text{period} & \rightarrow & \quad \text{uwindperiod}
\]

Table 5.9: EXF runtime attributes Note there is one exception for the default of atempconst = celsius2K = 273.16

<table>
<thead>
<tr>
<th>attribute</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\textit{field} file</td>
<td>...</td>
<td>filename; if left empty no file will be read; \textit{const} will be used instead</td>
</tr>
<tr>
<td>\textit{field} const</td>
<td>0.</td>
<td>constant that will be used if no file is read</td>
</tr>
<tr>
<td>\textit{field} startdate1</td>
<td>0.</td>
<td>format: YYYYMMDD; start year (YYYY), month (MM), day (YY)</td>
</tr>
<tr>
<td>\textit{field} startdate2</td>
<td>0.</td>
<td>format: HHMMSS; start hour (HH), minute (MM), second (SS)</td>
</tr>
<tr>
<td>\textit{field} period</td>
<td>0.</td>
<td>interval in seconds between two records</td>
</tr>
<tr>
<td>\textit{exf_inscal_field}</td>
<td></td>
<td>optional rescaling of input fields to comply with EXF units</td>
</tr>
<tr>
<td>\textit{exf_outscal_field}</td>
<td></td>
<td>optional rescaling of EXF fields when mapped onto MITgcm fields</td>
</tr>
</tbody>
</table>

\textit{used in conjunction with EXF\_USE\_INTERPOLATION}

| \textit{field} _lon0 | xgOrigin+delX/2 | starting longitude of input                                                |
| \textit{field} _lon_inc | delX       | increment in longitude of input                                           |
| \textit{field} _lat0  | ygOrigin+delY/2  | starting latitude of input                                                |
| \textit{field} _lat_inc | delY       | increment in latitude of input                                            |
| \textit{field} _nlon | Nx         | number of grid points in longitude of input                              |
| \textit{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 \textit{global_with_exf/}. It defines attributes for the heat flux variable \textit{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`.

<table>
<thead>
<tr>
<th>field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ustress</td>
<td>Zonal surface wind stress in N/m²</td>
</tr>
<tr>
<td></td>
<td>&gt; 0 for increase in uVel, which is west to east for cartesian and spherical</td>
</tr>
<tr>
<td></td>
<td>polar grids</td>
</tr>
<tr>
<td></td>
<td>Typical range: -0.5 &lt; ustress &lt; 0.5</td>
</tr>
<tr>
<td></td>
<td>Southwest C-grid U point</td>
</tr>
<tr>
<td></td>
<td>Input field</td>
</tr>
<tr>
<td>vstress</td>
<td>Meridional surface wind stress in N/m²</td>
</tr>
<tr>
<td></td>
<td>&gt; 0 for increase in vVel, which is south to north for cartesian and spherical</td>
</tr>
<tr>
<td></td>
<td>polar grids</td>
</tr>
<tr>
<td></td>
<td>Typical range: -0.5 &lt; vstress &lt; 0.5</td>
</tr>
<tr>
<td></td>
<td>Southwest C-grid V point</td>
</tr>
<tr>
<td></td>
<td>Input field</td>
</tr>
<tr>
<td>hs</td>
<td>Sensible heat flux into ocean in W/m²</td>
</tr>
<tr>
<td></td>
<td>&gt; 0 for increase in theta (ocean warming)</td>
</tr>
<tr>
<td>hl</td>
<td>Latent heat flux into ocean in W/m²</td>
</tr>
<tr>
<td></td>
<td>&gt; 0 for increase in theta (ocean warming)</td>
</tr>
<tr>
<td>hflux</td>
<td>Net upward surface heat flux in W/m²</td>
</tr>
<tr>
<td></td>
<td>(including shortwave)</td>
</tr>
<tr>
<td></td>
<td>hflux = latent + sensible + lwflux + swflux</td>
</tr>
<tr>
<td></td>
<td>&gt; 0 for decrease in theta (ocean cooling)</td>
</tr>
<tr>
<td></td>
<td>Typical range: -250 &lt; hflux &lt; 600</td>
</tr>
<tr>
<td></td>
<td>Southwest C-grid tracer point</td>
</tr>
<tr>
<td></td>
<td>Input field</td>
</tr>
<tr>
<td>sflux</td>
<td>Net upward freshwater flux in m/s</td>
</tr>
<tr>
<td></td>
<td>sflux = evap - precip - runoff</td>
</tr>
<tr>
<td></td>
<td>&gt; 0 for increase in salt (ocean salinity)</td>
</tr>
<tr>
<td></td>
<td>Typical range: -1e-7 &lt; sflux &lt; 1e-7</td>
</tr>
<tr>
<td></td>
<td>Southwest C-grid tracer point</td>
</tr>
</tbody>
</table>
c Input field

---

c swflux :: Net upward shortwave radiation in W/m^2
| swflux = - ( swdown - ice and snow absorption - reflected )
| > 0 for decrease in theta (ocean cooling)
| Typical range: -350 < swflux < 0
| Southwest C-grid tracer point
| Input field

---

c uwind :: Surface (10-m) zonal wind velocity in m/s
| > 0 for increase in uVel, which is west to east for cartesian and spherical polar grids
| Typical range: -10 < uwind < 10
| Southwest C-grid U point
| Input or input/output field

---

c vwind :: Surface (10-m) meridional wind velocity in m/s
| > 0 for increase in vVel, which is south to north for cartesian and spherical polar grids
| Typical range: -10 < vwind < 10
| Southwest C-grid V point
| Input or input/output field

---

c wspeed :: Surface (10-m) wind speed in m/s
| >= 0 sqrt(u^2+v^2)
| Typical range: 0 < wspeed < 10
| Input or input/output field

---

c atemp :: Surface (2-m) air temperature in deg K
| Typical range: 200 < atemp < 300
| Southwest C-grid tracer point
| Input or input/output field

---

c aqh :: Surface (2m) specific humidity in kg/kg
| Typical range: 0 < aqh < 0.02
| Southwest C-grid tracer point
| Input or input/output field

---

c lwflux :: Net upward longwave radiation in W/m^2
| lwflux = - ( lwdown - ice and snow absorption - emitted )
| > 0 for decrease in theta (ocean cooling)
| Typical range: -20 < lwflux < 170
| Southwest C-grid tracer point
| Input field

---

c evap :: Evaporation in m/s
| > 0 for increase in salt (ocean salinity)
| Typical range: 0 < evap < 2.5e-7
| Southwest C-grid tracer point
| Input, input/output, or output field

---

c precip :: Precipitation in m/s
| > 0 for decrease in salt (ocean salinity)
| Typical range: 0 < precip < 5e-7
| Southwest C-grid tracer point
| Input or input/output field

---

c snowprecip :: snow in m/s
> 0 for decrease in salt (ocean salinity)
Typical range: 0 < precip < 5e-7
Input or input/output field

runoff :: River and glacier runoff in m/s
> 0 for decrease in salt (ocean salinity)
Typical range: 0 < runoff < ????
Southwest C-grid tracer point
Input or input/output field
!!! WATCH OUT: Default exf_inscal_runoff !!!
!!! in exf_readparms.F is not 1.0 !!!

swdown :: Downward shortwave radiation in W/m^2
> 0 for increase in theta (ocean warming)
Typical range: 0 < swdown < 450
Southwest C-grid tracer point
Input/output field

lwdown :: Downward longwave radiation in W/m^2
> 0 for increase in theta (ocean warming)
Typical range: 50 < lwdown < 450
Southwest C-grid tracer point
Input/output field

apressure :: Atmospheric pressure field in N/m^2
> 0 for ????
Typical range: ??? < apressure < ???
Southwest C-grid tracer point
Input field

Key subroutines

Top-level routine: exf_getforcing.F

!CALLING SEQUENCE:
... exf_getforcing (TOP LEVEL ROUTINE)
|-- exf_getclim (get climatological fields used e.g. for relax.)
|   |-- exf_set_climsst (relax. to 2-D SST field)
|   |-- exf_set_climsss (relax. to 2-D SSS field)
|-- exf_getffields <- this one does almost everything
   1. reads in fields, either flux or atmos. state,
   depending on CPP options (for each variable two fields
   consecutive in time are read in and interpolated onto
   current time step).
   2. If forcing is atmos. state and control is atmos. state,
   then the control variable anomalies are read here via ctrl_get_gen
   (atemp, aqh, precip, swflux, swdown, uwind, vwind).
   If forcing and control are fluxes, then
   controls are added later.
|-- exf_radiation
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/m² | Sensible heat flux into ocean, >0 increases theta |
| EXFh1  | 1  | SM  | W/m² | Latent heat flux into ocean, >0 increases theta |
| EXFlwnet| 1  | SM  | W/m² | Net upward longwave radiation, >0 decreases theta |
| EXFswnet| 1  | SM  | W/m² | Net upward shortwave radiation, >0 decreases theta |
| EXFlwdn| 1  | SM  | W/m² | Downward longwave radiation, >0 increases theta |
| EXFswdn| 1  | SM  | W/m² | Downward shortwave radiation, >0 increases theta |
| EXFqnet| 1  | SM  | W/m² | Net upward heat flux (turb+rad), >0 decreases theta |
| EXFtaux| 1  | SU  | N/m² | zonal surface wind stress, >0 increases uVel |
| EXFtauy| 1  | SV  | N/m² | 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/m² | 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)
```

( `yyyymmdd`, `hhmmss`, `leap_year`, `dayofweek` )

- `date(1) = yyyymmdd` <-- Year-Month-Day
- `date(2) = hhmmss` <-- Hours-Minutes-Seconds
- `date(3) = leap_year` <-- Leap Year/No Leap Year
- `date(4) = dayofweek` <-- Day of the Week

leap_year is either equal to 1 (normal year) or equal to 2 (leap year)

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
refdate(2) = 0
refdate(3) = 1
refdate(4) = 1
```

Calendar dates and time intervals

Subtracting calendar dates yields time intervals. Time intervals have the following format:

```c
integer datediff(4)
```

- `datediff(1) = # Days`
- `datediff(2) = hhmmss`
- `datediff(3) = 0`
- `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 “initialise_fixed.F”, depending on the MITgcm release, has to be modified in the following way:

```c
#define ALLOW_CALENDAR
C-- Initialise the calendar package.
#define USE_CAL_NENDITER
CALL cal_Init(
    I startTime,
    I endTime,
    I deltaTclock,
    I nIter0,
```
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**

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

o cal_TimePassed - Determine the time passed between two dates.

o cal_AddTime - Add a time interval either to a time interval or to a date.

o cal_TimeInterval - Given a time interval return the corresponding date array.

o cal_SubDates - Determine the time interval between two dates or between two time intervals.

o cal_ConvDate - Decompose a date array or a time interval array into its components.

o cal_CopyDate - Copy a date array or a time interval array to another array.

o cal_CompDates - Compare two calendar dates or time intervals.

o cal_ToSeconds - Given a time interval array return the number of seconds.

o cal_WeekDay - Return the weekday as a string given the calendar date.

o cal_NumInts - Return the number of time intervals between two given dates.

o cal_StepsPerDay - Given an iteration number or the current integration time return the number of time steps to integrate in the current calendar day.

o cal_DaysPerMonth - Given an iteration number or the current integration time return the number of days to integrate in this calendar month.

o cal_MonthsPerYear - Given an iteration number or the current integration time return the number of months to integrate in the current calendar year.

o cal_StepsForDay - Given the integration day return the number of steps to be integrated, the first step, and the last step in the day specified. The first and the last step refer to the total number of steps (1, ..., cal_IntSteps).

o cal_DaysForMonth - Given the integration month return the number of days to be integrated, the first day, and the last day in the month specified. The first and the last day refer to the total number of steps (1, ..., cal_IntDays).

o cal_MonthsForYear - Given the integration year return the number of months to be integrated, the first month, and the last month in the year specified. The first and the last month refer to the total number of months (1, ..., cal_IntYears).
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 following 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
```
<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
<th>Short Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DIAIQ</td>
<td>5</td>
<td>SM ML</td>
<td>Spec. Humid. Tendency (Mass-Weighted)</td>
</tr>
<tr>
<td>RADSW</td>
<td>5</td>
<td>SM ML</td>
<td>Temperature Tendency due to Shortwave Radiation</td>
</tr>
<tr>
<td>RADIW</td>
<td>5</td>
<td>SM ML</td>
<td>Temperature Tendency due to Longwave Radiation</td>
</tr>
<tr>
<td>DTCNV</td>
<td>5</td>
<td>SM MR</td>
<td>Temperature Tendency due to Convection</td>
</tr>
<tr>
<td>TURBT</td>
<td>5</td>
<td>SM ML</td>
<td>Temperature Tendency due to Turbulence in PBL</td>
</tr>
<tr>
<td>DTLS</td>
<td>5</td>
<td>SM ML</td>
<td>Temperature Tendency due to Large-scale condens.</td>
</tr>
<tr>
<td>DCQCNV</td>
<td>5</td>
<td>SM MR</td>
<td>Spec. Humidity Tendency due to Convection</td>
</tr>
<tr>
<td>TURBQ</td>
<td>5</td>
<td>SM ML</td>
<td>Spec. Humidity Tendency due to Turbulence in PBL</td>
</tr>
<tr>
<td>DQLS</td>
<td>5</td>
<td>SM ML</td>
<td>Spec. Humidity Tendency due to Large-scale condens.</td>
</tr>
<tr>
<td>TSR</td>
<td>1</td>
<td>SM P U1</td>
<td>Top-of-atm. net Shortwave Radiation (+=dw)</td>
</tr>
<tr>
<td>OLR</td>
<td>1</td>
<td>SM P U1</td>
<td>Outgoing Longwave Radiation (+=up)</td>
</tr>
<tr>
<td>RADSWG</td>
<td>1</td>
<td>SM P L1</td>
<td>Net Shortwave Radiation at the Ground (+=dw)</td>
</tr>
<tr>
<td>RADLWG</td>
<td>1</td>
<td>SM L1</td>
<td>Net Longwave Radiation at the Ground (+=up)</td>
</tr>
<tr>
<td>HFLUX</td>
<td>1</td>
<td>SM L1</td>
<td>Sensible Heat Flux (+=up)</td>
</tr>
<tr>
<td>EVAP</td>
<td>1</td>
<td>SM L1</td>
<td>Surface Evaporation (g/m²/s)</td>
</tr>
<tr>
<td>PRECON</td>
<td>1</td>
<td>SM P L1</td>
<td>Convective Precipitation (g/m²/s)</td>
</tr>
<tr>
<td>PRECLS</td>
<td>1</td>
<td>SM M1</td>
<td>Large Scale Precipitation (g/m²/s)</td>
</tr>
<tr>
<td>CLDFRC</td>
<td>1</td>
<td>SM P M1</td>
<td>Total Cloud Fraction (0-1)</td>
</tr>
<tr>
<td>CLDPRS</td>
<td>1</td>
<td>SM PC167M1</td>
<td>Cloud Top Pressure (normalized)</td>
</tr>
<tr>
<td>CLDMAS</td>
<td>5</td>
<td>SM P LL</td>
<td>Cloud-base Mass Flux (kg/m²/s)</td>
</tr>
<tr>
<td>DRAG</td>
<td>5</td>
<td>SM P LL</td>
<td>Surface Drag Coefficient (kg/m²/s)</td>
</tr>
<tr>
<td>WINDS</td>
<td>1</td>
<td>SM P L1</td>
<td>Surface Wind Speed (m/s)</td>
</tr>
<tr>
<td>TS</td>
<td>1</td>
<td>SM L1</td>
<td>Near Surface Air Temperature (K)</td>
</tr>
<tr>
<td>QS</td>
<td>1</td>
<td>SM P L1</td>
<td>Near Surface Specific Humidity (g/kg)</td>
</tr>
<tr>
<td>ENPREC</td>
<td>1</td>
<td>SM M1</td>
<td>Energy flux associated with precip.</td>
</tr>
<tr>
<td>ALBVISDF</td>
<td>1</td>
<td>SM P L1</td>
<td>Surface Albedo (Visible band) (0-1)</td>
</tr>
<tr>
<td>DNWLG</td>
<td>1</td>
<td>SM P L1</td>
<td>Downward Component of Longwave Flux</td>
</tr>
<tr>
<td>SWCLR</td>
<td>5</td>
<td>SM ML</td>
<td>Clear Sky Temp. Tendency due to Shortwave Radiation</td>
</tr>
<tr>
<td>LWCLR</td>
<td>5</td>
<td>SM ML</td>
<td>Clear Sky Temp. Tendency due to Longwave Radiation</td>
</tr>
<tr>
<td>TSRCLR</td>
<td>1</td>
<td>SM P U1</td>
<td>Clear Sky Top-of-atm. net Shortwave Radiation (+=dw)</td>
</tr>
<tr>
<td>OLRCLR</td>
<td>1</td>
<td>SM P U1</td>
<td>Clear Sky Outgoing Longwave Radiation (+=up)</td>
</tr>
<tr>
<td>SWGCLR</td>
<td>1</td>
<td>SM P L1</td>
<td>Clear Sky Net Shortwave Radiation at the Ground (+=up)</td>
</tr>
<tr>
<td>LGWCLR</td>
<td>1</td>
<td>SM L1</td>
<td>Clear Sky Net Longwave Radiation at the Ground (+=up)</td>
</tr>
<tr>
<td>UFLUX</td>
<td>1</td>
<td>UM 184L1</td>
<td>Zonal Wind Surface Stress (N/m²)</td>
</tr>
<tr>
<td>VFLUX</td>
<td>1</td>
<td>VM 183L1</td>
<td>Meridional Wind Surface Stress (N/m²)</td>
</tr>
<tr>
<td>DTSIMPL</td>
<td>1</td>
<td>SM P L1</td>
<td>Surf. Temp Change after 1 implicit time step</td>
</tr>
</tbody>
</table>
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 $z_1 = 0.1m$, the second layer depth $z_2 = 4m$.

Let $T_{g1}, T_{g2}$ be the temperature of each layer, $W_1, W_2$ be the soil moisture of each layer. The field capacity $f_1, f_2$ are the maximum water amount in each layer, so $W_i$ is the ratio of available water to field capacity. $f_i = \gamma z_i, \gamma = 0.24$ is the field capapcity per meter soil, so $f_1 = 0.024m$, $f_2 = 0.96m$.

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

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

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

here $C_1, C_2$ are the heat capacity of each layer, $\lambda$ is the thermal conductivity, $\lambda = 0.42 \text{Wm}^{-1}\text{K}^{-1}$.

$$C_1 = C_w W_1 \gamma + C_s$$

$$C_2 = C_w W_2 \gamma + C_s$$

$C_w, C_s$ are the heat capacity of water and dry soil respectively. $C_w = 4.2 \times 10^6 \text{Jm}^{-3}\text{K}^{-1}, C_s = 1.13 \times 10^6 \text{Jm}^{-3}\text{K}^{-1}$.

The soil moisture is determined by precipitation $P(m/s)$, surface evaporation $E(m/s)$ and runoff $R(m/s)$.

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

$\tau = 2 \text{ days}$ is the time constant for diffusion of moisture between layers.

$$\frac{dW_2}{dt} = \frac{f_1 W_1 - W_2}{f_2 \tau}$$

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 $W_2 = W_2 + p \delta W_1 f_1$, i.e. the runoff of top layer is put into second layer. $p = 0.5$ is the fraction of top layer runoff that is put into second layer.

The time step is 1 hour, it takes several years to reach equilibrium offline.
Land diagnostics

<table>
<thead>
<tr>
<th>Name</th>
<th>Levs</th>
<th>Parsing code</th>
<th>Units</th>
<th>Tile (max=80c)</th>
</tr>
</thead>
<tbody>
<tr>
<td>GrdSurfT</td>
<td>1</td>
<td>SM Lg</td>
<td>degC</td>
<td>Surface Temperature over land</td>
</tr>
<tr>
<td>GrdTemp</td>
<td>2</td>
<td>SM MG</td>
<td>degC</td>
<td>Ground Temperature at each level</td>
</tr>
<tr>
<td>GrdEnth</td>
<td>2</td>
<td>SM MG</td>
<td>J/m3</td>
<td>Ground Enthalpy at each level</td>
</tr>
<tr>
<td>GrdWater</td>
<td>2</td>
<td>SM P MG</td>
<td>0-1</td>
<td>Ground Water (vs Field Capacity)</td>
</tr>
<tr>
<td>LdSnowH</td>
<td>1</td>
<td>SM P Lg</td>
<td>m</td>
<td>Snow Thickness over land</td>
</tr>
<tr>
<td>LdSnwAge</td>
<td>1</td>
<td>SM P Lg</td>
<td>s</td>
<td>Snow Age over land</td>
</tr>
<tr>
<td>RUNOFF</td>
<td>1</td>
<td>SM L1</td>
<td>m/s</td>
<td>Run-Off per surface unit</td>
</tr>
<tr>
<td>landHFlx</td>
<td>1</td>
<td>SM Lg</td>
<td>W/m^2</td>
<td>net surface downward Heat flux over land</td>
</tr>
<tr>
<td>landPmE</td>
<td>1</td>
<td>SM Lg</td>
<td>kg/m^2/s</td>
<td>Precipitation minus Evaporation over land</td>
</tr>
<tr>
<td>ldEnFxPr</td>
<td>1</td>
<td>SM Lg</td>
<td>W/m^2</td>
<td>Energy flux (over land) associated with Precip (snow, rain)</td>
</tr>
</tbody>
</table>

References


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, \( s = c_p T + gz \), and the moist static energy, \( h = c_p T + gz + Lq \). The conceptual model behind RAS depicts each subensemble as a rising plume cloud, entraining mass from the environment during ascent, and detraining all cloud air at the level of neutral buoyancy. RAS assumes that the normalized cloud mass flux, \( \eta \), normalized by the cloud base mass flux, is a linear function of height, expressed as:

\[
\eta(z) = \lambda \quad \text{or} \quad \eta(P) = -\frac{c_p \theta}{g} P
\]

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

\[
z P = -\frac{c_p \theta}{g}
\]

The entrainment parameter, \( \lambda \), characterizes a particular subensemble based on its detrainment level, and is obtained by assuming that the level of detrainment is the level of neutral buoyancy, i.e., the level at which the moist static energy of the cloud, \( h_c \), is equal to the saturation moist static energy of the environment, \( h^* \). Following [MS92], \( \lambda \) may be written as

\[
\lambda = \frac{h_B - h^*_D}{\frac{c_p}{g} \int_{P_B}^{P_D} \theta(h^*_D - h) dP},
\]

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:

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

where \( \gamma = \frac{L}{c_v} q^* T \) obtained from the Clausius Clapeyron equation, and the subscript \( c \) refers to the value inside the cloud.

To determine the cloud base mass flux, the rate of change of \( A \) in time due to dissipation by the clouds is assumed to approximately balance the rate of change of \( A \) due to the generation by the large scale. This is the quasi-equilibrium assumption, and results in an expression for \( m_B \):

\[
m_B = -\frac{dA}{dt} |_{ls} K
\]

where \( K \) is the cloud kernel, defined as the rate of change of the cloud work function per unit cloud base mass flux, and is currently obtained by analytically differentiating the expression for \( A \) in time. The rate of change of \( A \) due to the generation by the large scale can be written as the difference between the current \( A(t + \Delta t) \) and its equilibrated value after the previous convective time step \( A(t) \), divided by the time step. \( A(t) \) is approximated as some critical \( A_{crit} \), computed by Lord (1982) from \textit{in situ} observations.

The predicted convective mass fluxes are used to solve grid-scale temperature and moisture budget equations to determine the impact of convection on the large scale fields of temperature (through latent heating and compensating subsidence) and moisture (through precipitation and detrainment):

\[
\theta(t)|_c = \alpha \frac{m_B}{c_p P} \eta sp
\]

and

\[
q(t)|_c = \alpha \frac{m_B}{L} \eta(hp - sp)
\]

where: \( \text{math} : \mathcal{\theta} = \frac{T}{P} \); \( \text{math} : \mathcal{P} = \frac{(p/p_0)}{\mathcal{P}} \), and
\( \alpha \) is the relaxation parameter.

As an approximation to a full interaction between the different allowable subensembles, many clouds are simulated frequently, each modifying the large scale environment some fraction \( \alpha \) of the total adjustment. The parameterization thereby “relaxes” the large scale environment towards equilibrium.

In addition to the RAS cumulus convection scheme, the fizhi package employs a Kessler-type scheme for the re-evaporation of falling rain [SM88], which correspondingly adjusts the temperature assuming \( h \) is conserved. RAS in its current formulation assumes that all cloud water is deposited into the detrainment level as rain. All of the rain is available for re-evaporation, which begins in the level below detrainment. The scheme accounts for some microphysics such as the rainfall intensity, the drop size distribution, as well as the temperature, pressure and relative humidity of the surrounding air. The fraction of the moisture deficit in any model layer into which the rain may re-evaporate is controlled by a free parameter, which allows for a relatively efficient re-evaporation of liquid precipitate and larger rainout for frozen precipitation.

Due to the increased vertical resolution near the surface, the lowest model layers are averaged to provide a 50 mb thick sub-cloud layer for RAS. Each time RAS is invoked (every ten simulated minutes), a number of randomly chosen subensembles are checked for the possibility of convection, from just above cloud base to 10 mb.

Supersaturation or large-scale precipitation is initiated in the fizhi package whenever the relative humidity in any grid-box exceeds a critical value, currently 100 %. The large-scale precipitation re-evaporates during descent to partially saturate lower layers in a process identical to the re-evaporation of convective rain.

**Cloud Formation**

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

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

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

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

where \( F_{RAS} \) is the instantaneous cloud fraction and \( F_{RAS}^{n-1} \) is the cloud fraction from the previous RAS timestep. The memory coefficient is computed using a RAS cloud timescale, \( \tau \), equal to 1 hour. RAS cloud fractions are cleared when they fall below 5 %.

Large-scale cloudiness is defined, following Slingo and Ritter (1985), as a function of relative humidity:

\[
F_{LS} = \min \left( \left( \frac{R_H - R_{Hc}}{1 - R_{Hc}} \right)^2, 1.0 \right)
\]

where

\[
R_{Hc} = 1 - s(1-s)(2+2 s)r \quad s = p/p_{surf} \quad r = ( ) RH_{min} \quad s = 0.75 \quad r = 0.573285
\]

These cloud fractions are suppressed, however, in regions where the convective sub-cloud layer is conditionally unstable. The functional form of \( R_{Hc} \) 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

\[ \rho c_p T t = -F z \]

or

\[ T t = \frac{g}{c_p \pi} F \sigma \]

where \( g \) is the acceleration due to gravity and \( c_p \) is the heat capacity of air at constant pressure.

The time tendency for Longwave Radiation is updated every 3 hours. The time tendency for Shortwave Radiation is updated once every three hours assuming a normalized incident solar radiation, and subsequently modified at every model time step by the true incident radiation. The solar constant value used in the package is equal to 1365 W/m² and a CO₂ mixing ratio of 330 ppm. For the ozone mixing ratio, monthly mean zonally averaged climatological values specified as a function of latitude and height [RSG87] are linearly interpolated to the current time.

**Shortwave Radiation**

The shortwave radiation package used in the package computes solar radiative heating due to the absorption by water vapor, ozone, carbon dioxide, oxygen, clouds, and aerosols and due to the scattering by clouds, aerosols, and gases. The shortwave radiative processes are described by [Cho90][Cho92]. This shortwave package uses the Delta-Eddington approximation to compute the bulk scattering properties of a single layer following King and Harshvardhan (JAS, 1986). The transmittance and reflectance of diffuse radiation follow the procedures of Sagan and Pollock (JGR, 1967) and [LH74].

Highly accurate heating rate calculations are obtained through the use of an optimal grouping strategy of spectral bands. By grouping the UV and visible regions as indicated in Table 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.

<table>
<thead>
<tr>
<th>UV and Visible Spectral Regions</th>
<th>Region</th>
<th>Band</th>
<th>Wavelength (micron)</th>
</tr>
</thead>
<tbody>
<tr>
<td>UV-C</td>
<td>1.</td>
<td>1.</td>
<td>.175 - .225</td>
</tr>
<tr>
<td></td>
<td>2.</td>
<td>2.</td>
<td>.225 - .245</td>
</tr>
<tr>
<td></td>
<td>3.</td>
<td>3.</td>
<td>.260 - .280</td>
</tr>
<tr>
<td>UV-B</td>
<td>4.</td>
<td>4.</td>
<td>.280 - .295</td>
</tr>
<tr>
<td></td>
<td>5.</td>
<td>5.</td>
<td>.295 - .310</td>
</tr>
<tr>
<td></td>
<td>6.</td>
<td>6.</td>
<td>.310 - .320</td>
</tr>
<tr>
<td>UV-A</td>
<td>7.</td>
<td>7.</td>
<td>.320 - .400</td>
</tr>
<tr>
<td>PAR</td>
<td>8.</td>
<td>8.</td>
<td>.400 - .700</td>
</tr>
</tbody>
</table>

Table 5.11: Infrared Spectral Regions used in shortwave radiation package.

<table>
<thead>
<tr>
<th>Infrared Spectral Regions</th>
<th>Band</th>
<th>Wavenumber (cm⁻¹)</th>
<th>Wavelength (micron)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>1000-4400</td>
<td>2.27 - 10.0</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>4400-8200</td>
<td>1.22 - 2.27</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>8200-14300</td>
<td>0.70 - 1.22</td>
</tr>
</tbody>
</table>

Within the shortwave radiation package, both ice and liquid cloud particles are allowed to co-exist in any of the model layers. Two sets of cloud parameters are used, one for ice particles and the other for liquid particles. Cloud parameters are defined as the cloud optical thickness and the effective cloud particle size. In the fizhi package, the effective radius for water droplets is given as 10 microns, while 65 microns is used for ice particles. The absorption due to aerosols is currently set to zero.

To simplify calculations in a cloudy atmosphere, clouds are grouped into low ($p > 700$ mb), middle ($700 \text{ mb} \geq p > 400 \text{ mb}$), and high ($p < 400 \text{ mb}$) cloud regions. Within each of the three regions, clouds are assumed maximally overlapped, and the cloud cover of the group is the maximum cloud cover of all the layers in the group. The optical thickness of a given layer is then scaled for both the direct (as a function of the solar zenith angle) and diffuse beam radiation so that the grouped layer reflectance is the same as the original reflectance. The solar flux is computed for each of eight cloud realizations possible within this low/middle/high classification, and appropriately averaged to produce the net solar flux.

**Longwave Radiation**

The longwave radiation package used in the fizhi package is thoroughly described by. 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])

<table>
<thead>
<tr>
<th>IR Spectral Bands</th>
<th>Band</th>
<th>Spectral Range (cm⁻¹)</th>
<th>Absorber</th>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>0-340</td>
<td>H₂O line</td>
<td>T</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>340-540</td>
<td>H₂O line</td>
<td>T</td>
</tr>
<tr>
<td></td>
<td>3a</td>
<td>540-620</td>
<td>H₂O line</td>
<td>K</td>
</tr>
<tr>
<td></td>
<td>3b</td>
<td>620-720</td>
<td>H₂O continuum</td>
<td>S</td>
</tr>
<tr>
<td></td>
<td>3b</td>
<td>720-800</td>
<td>CO₂</td>
<td>T</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>800-980</td>
<td>H₂O line</td>
<td>K</td>
</tr>
<tr>
<td></td>
<td></td>
<td>H₂O continuum</td>
<td>S</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>H₂O line</td>
<td>K</td>
<td></td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>980-1100</td>
<td>H₂O continuum</td>
<td>S</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>1100-1380</td>
<td>O₃</td>
<td>T</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>1380-1900</td>
<td>H₂O line</td>
<td>K</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>1900-3000</td>
<td>H₂O line</td>
<td>K</td>
</tr>
</tbody>
</table>

K: k-distribution method with linear pressure scaling  
T: Table look-up with temperature and pressure scaling  
S: One-parameter temperature scaling

The longwave radiation package accurately computes cooling rates for the middle and lower atmosphere from 0.01 mb to the surface. Errors are < 0.4 C day⁻¹ in cooling rates and < 1% in fluxes. From Chou and Suarez, it is estimated that the total effect of neglecting all minor absorption bands and the effects of minor infrared absorbers such as nitrous oxide (N₂O), methane (CH₄), and the chlorofluorocarbons (CFCs), is an underestimate of ≈ 5 W/m² in the downward flux at the surface and an overestimate of ≈ 3 W/m² in the upward flux at the top of the atmosphere.

Similar to the procedure used in the shortwave radiation package, clouds are grouped into three regions categorized as low/middle/high. The net clear line-of-site probability (P) between any two levels, p₁ and p₂ (p₂ > p₁), assuming randomly overlapped cloud groups, is simply the product of the probabilities within each group:

\[ P_{\text{net}} = P_{\text{low}} \times P_{\text{mid}} \times P_{\text{hi}}. \]

Since all clouds within a group are assumed maximally overlapped, the clear line-of-site probability within a group is given by:

\[ P_{\text{group}} = 1 - F_{\text{max}}, \]

where \( F_{\text{max}} \) is the maximum cloud fraction encountered between \( p_1 \) and \( p_2 \) within that group. For groups and/or levels outside the range of \( p_1 \) and \( p_2 \), a clear line-of-site probability equal to 1 is assigned.

Cloud-Radiation Interaction

The cloud fractions and diagnosed cloud liquid water produced by moist processes within the fizhi package are used in the radiation packages to produce cloud-radiative forcing. The cloud optical thickness associated with large-scale cloudiness is made proportional to the diagnosed large-scale liquid water, \( \ell \), detrained due to super-saturation. Two values are used corresponding to cloud ice particles and water droplets. The range of optical thickness for these clouds is given as

\[ 0.0002 \leq \tau_{\text{ice}}(\text{mb}^{-1}) \leq 0.002 \quad \text{for} \quad 0 \leq \ell \leq 2 \quad \text{mg/kg}, \]
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, \( \alpha \), between ice particles and water droplets is achieved through a linear scaling in temperature:

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

The resulting optical depth associated with large-scale cloudiness is given as

\[ \tau_{LS} = \alpha \tau_{h_2o} + (1 - \alpha)\tau_{ice}. \]

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:

\[ u_t_{turb} = z(-u'w') = z(K_m u z) \]
\[ v_t_{turb} = z(-v'w') = z(K_m v z) \]
\[ T_t = P^e \theta_{turb} = P^e z(-w'\theta') = P^e z(K_h \theta_v z) \]
\[ q_t_{turb} = z(-w'q') = z(K_h q z) \]

Within the atmosphere, the time evolution of second turbulent moments is explicitly modeled by representing the third moments in terms of the first and second moments. This approach is known as a second-order closure modeling. To simplify and streamline the computation of the second moments, the level 2.5 assumption of Mellor and Yamada (1974) and [Yam77] is employed, in which only the turbulent kinetic energy (TKE),

\[ q^2 = u'^2 + v'^2 + w'^2, \]

is solved prognostically and the other second moments are solved diagnostically. The prognostic equation for TKE allows the scheme to simulate some of the transient and diffusive effects in the turbulence. The TKE budget equation is solved numerically using an implicit backward computation of the terms linear in \( q^2 \) and is written:

\[ t(q^2) - z\left(\frac{5}{3} \lambda_1 q z(q^2)\right) = -u'w'Uz - v'w'Vz + \frac{g}{\Theta_0} \frac{w'\theta_v}{T} - \frac{q^3}{A_1} \]
where \(q\) is the turbulent velocity, \(u', v', w', \) and \(\theta'\) are the fluctuating parts of the velocity components and potential temperature, \(U\) and \(V\) are the mean velocity components, \(\Theta_0^{-1}\) is the coefficient of thermal expansion, and \(\Lambda_1\) and \(\Lambda_1\) are constant multiples of the master length scale, \(\ell\), which is designed to be a characteristic measure of the vertical structure of the turbulent layers.

The first term on the left-hand side represents the time rate of change of TKE, and the second term is a representation of the triple correlation, or turbulent transport term. The first three terms on the right-hand side represent the sources of TKE due to shear and buoyancy, and the last term on the right hand side is the dissipation of TKE.

In the level 2.5 approach, the vertical fluxes of the scalars \(\theta\) and \(q\) and the wind components \(u\) and \(v\) are expressed in terms of the diffusion coefficients \(K_h\) and \(K_m\), respectively. In the statistically realizable level 2.5 turbulence scheme of [HL88], these diffusion coefficients are expressed as

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

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 equilibrium (obtained from the Level 2.0 Model), \(\ell\) is the master length scale related to the vertical structure of the atmosphere, and \(S_h\) and \(S_M\) are functions of \(G_H\) and \(G_M\), the dimensionless buoyancy and wind shear parameters, respectively. Both \(G_H\) and \(G_M\), and their equilibrium values \(G_{H_e}\) and \(G_{M_e}\), are functions of the Richardson number:

\[
RI = \frac{\frac{g_\theta \theta'_w}{(u_z)^2 + (v_z)^2}}{c_p \psi m z} = \frac{c_p \theta_w z \rho^\kappa z}{(u_z)^2 + (v_z)^2}.
\]

Negative values indicate unstable buoyancy and shear, small positive values \((< 0.2)\) indicate dominantly unstable shear, and large positive values indicate dominantly stable stratification.

Turbulent eddy diffusion coefficients of momentum, heat and moisture in the surface layer, which corresponds to the lowest GCM level (see — missing table —), are calculated using stability-dependant functions based on Monin-Obukhov theory:

\[
K_m(surface) = C_u \times u_s = C_D W_s
\]

and

\[
K_h(surface) = C_t \times u_s = C_H W_s
\]

where : math : 'u_s = C_u W_s' isthesurfacefrictionvelocity, : 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:

\[
C_u = \frac{u_s}{W_s} = \frac{k}{\psi m}
\]

where \(k\) is the Von Karman constant and \(\psi_m\) is the surface layer non-dimensional wind shear given by

\[
\psi_m = \int_{\zeta_0}^{\zeta} \phi_m d\zeta.
\]

Here : math : '\zeta' isthenondimensionalstabilityparameter, and \(\phi_m\) is the similarity function of \(\zeta\) which expresses the stability dependance of the momentum gradient. The functional form of \(\phi_m\) is specified differently for stable and unstable layers.
$C_t$ is the dimensionless exchange coefficient for heat and moisture from the surface layer similarity functions:

$$C_t = \frac{-\langle w' \theta' \rangle}{u_* \Delta \theta} = \frac{-\langle w' q' \rangle}{u_* \Delta q} = \frac{k}{\psi_h + \psi_g}$$

where $\psi_h$ is the surface layer non-dimensional temperature gradient given by

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

Expressing the stability dependence of the temperature and moisture gradients, and is specified differently for stable and unstable layers according to [HS95].

$\psi_g$ is the non-dimensional temperature or moisture gradient in the viscous sublayer, which is the mostly laminar region between the surface and the tops of the roughness elements, in which temperature and moisture gradients can be quite large. Based on [YK74]:

$$\psi_g = 0.55 \left( \frac{Pr^{2/3} - 0.2}{\nu^{1/2}} \right) \left( h_0 u_* - h_{0,ref} u_{*ref} \right)^{1/2}$$

where $Pr$ is the Prandtl number for air, $\nu$ is the molecular viscosity, $z_0$ is the surface roughness length, and the subscript ref refers to a reference value. $h_0 = 30 z_0$ with a maximum value over land of 0.01.

The surface roughness length over oceans is is a function of the surface-stress velocity,

$$z_0 = c_1 u_*^3 + c_2 u_*^2 + c_3 u_* + c_4 + \frac{c_5}{u_*}$$

where the constants are chosen to interpolate between the reciprocal relation of [Kon75] for weak winds, and the piecewise linear relation of [LP81] for moderate to large winds. Roughness lengths over land are specified from the climatology of [DS89].

For an unstable surface layer, the stability functions, chosen to interpolate between the condition of small values of $\beta$ and the convective limit, are the KEYPS function [Pan73] for momentum, and its generalization for heat and moisture:

$$\phi_m^4 - 18 \zeta \phi_m^3 = 1 ; \quad \phi_h^2 - 18 \zeta \phi_h^3 = 1 .$$

The function for heat and moisture assures non-vanishing heat and moisture fluxes as the wind speed approaches zero.

For a stable surface layer, the stability functions are the observationally based functions of [Cla70], slightly modified for the momentum flux:

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

The moisture flux also depends on a specified evapotranspiration coefficient, set to unity over oceans and dependant on the climatological ground wetness over land.

Once all the diffusion coefficients are calculated, the diffusion equations are solved numerically using an implicit backward operator.

**Atmospheric Boundary Layer**

The depth of the atmospheric boundary layer (ABL) is diagnosed by the parameterization as the level at which the turbulent kinetic energy is reduced to a tenth of its maximum near surface value. The vertical structure of the ABL is explicitly resolved by the lowest few (3-8) model layers.
Surface Energy Budget

The ground temperature equation is solved as part of the turbulence package using a backward implicit time differencing scheme:

\[ C_g T_g + R_{sw} - R_{lw} + Q_{ice} - H - LE \]

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:

\[ H = P \rho c_p C_H W_s (\theta_{surface} - \theta_{LAY}) \]

where \( \rho \) = the atmospheric density at the surface, \( c_p \) is the specific heat of air at constant pressure, and \( \theta \) represents the potential temperature of the surface and of the lowest \( \sigma \)-level, respectively.

The upward latent heat flux, \( LE \), is given by

\[ LE = \rho \beta L C_H W_s (q_{surface} - q_{LAY}) \]

where \( \beta \) is the fraction of the potential evapotranspiration actually evaporated, \( L \) is the latent heat of evaporation, and \( q_{surface} \) and \( q_{LAY} \) are the specific humidity of the surface and of the lowest \( \sigma \)-level, respectively.

The heat conduction through sea ice, \( Q_{ice} \), is given by

\[ Q_{ice} = \frac{C_{ti}}{H_i} (T_i - T_g) \]

where \( C_{ti} \) is the thermal conductivity of ice, \( H_i \) is the ice thickness, assumed to be 3 m where sea ice is present, \( T_1 \) 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:

\[ C_g = \sqrt{\frac{\lambda C_s}{2 \omega}} = \lim_{\omega \to 0} \left( \frac{0.386 + 0.536W + 0.15W^2}{2} \right) \times 10^{-3} \frac{86400}{2\pi} \]

Here, the thermal conductivity, \( \lambda \), is equal to \( 2 \times 10^{-3} \frac{W}{\text{sec cm}^K} \), the angular velocity of the earth, \( \omega \), is written as 86400 \( \text{sec/day} \) divided by \( 2\pi \text{ radians/day} \), and the expression for \( C_s \), the heat capacity per unit volume at the surface, is a function of the ground wetness, \( W \).

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

<table>
<thead>
<tr>
<th>Type</th>
<th>Vegetation Designation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Broadleaf Evergreen Trees</td>
</tr>
<tr>
<td>2</td>
<td>Broadleaf Deciduous Trees</td>
</tr>
<tr>
<td>3</td>
<td>Needleleaf Trees</td>
</tr>
<tr>
<td>4</td>
<td>Ground Cover</td>
</tr>
<tr>
<td>5</td>
<td>Broadleaf Shrubs</td>
</tr>
<tr>
<td>6</td>
<td>Dwarf Trees (Tundra)</td>
</tr>
<tr>
<td>7</td>
<td>Bare Soil</td>
</tr>
<tr>
<td>8</td>
<td>Desert (Bright)</td>
</tr>
<tr>
<td>9</td>
<td>Glacier</td>
</tr>
<tr>
<td>10</td>
<td>Desert (Dark)</td>
</tr>
<tr>
<td>100</td>
<td>Ocean</td>
</tr>
</tbody>
</table>

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:

\[ |\tau_{sf}^*| = \frac{\rho U^3}{N\ell^*} \left( \frac{F_r^2}{1 + F_r^2} \right) , \]

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 \( \ell^* \) is the wavelength of the monochromatic gravity wave in the direction of the low-level wind. A modification introduced by Zhou et al. allows for the momentum flux to escape through the top of the model, although this effect is small for the current 70-level model. The subgrid scale standard deviation is defined by \( h \), and is not allowed to exceed 400 m.

The effects of using this scheme within a GCM are shown in [TS96]. Experiments using the gravity wave drag parameterization yielded significant and beneficial impacts on both the time-mean flow and the transient statistics of the a GCM climatology, and have eliminated most of the worst dynamically driven biases in the a GCM simulation. An examination of the angular momentum budget during climate runs indicates that the resulting gravity wave torque is similar to the data-driven torque produced by a data assimilation which was performed without gravity wave drag. It was shown that the inclusion of gravity wave drag results in large changes in both the mean flow and in eddy fluxes. The result is a more accurate simulation of surface stress (through a reduction in the surface wind strength), of mountain torque (through a redistribution of mean sea-level pressure), and of momentum convergence (through a reduction in the flux of westerly momentum by transient flow eddies).

Boundary Conditions and other Input Data:

Required fields which are not explicitly predicted or diagnosed during model execution must either be prescribed internally or obtained from external data sets. In the fizhi package these fields include: sea surface temperature, sea ice extent, surface geopotential variance, vegetation index, and the radiation-related background levels of: ozone, carbon dioxide, and stratospheric moisture.

Boundary condition data sets are available at the model’s resolutions for either climatological or yearly varying conditions. Any frequency of boundary condition data can be used in the fizhi package; however, the current selection of data is summarized in Table 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.

<table>
<thead>
<tr>
<th>Fizhi Input Datasets</th>
<th>Frequency</th>
<th>Time Period</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sea Ice Extent</td>
<td>monthly</td>
<td>1979-current, climatology</td>
</tr>
<tr>
<td>Sea Ice Extent</td>
<td>weekly</td>
<td>1982-current, climatology</td>
</tr>
<tr>
<td>Sea Surface Temperature</td>
<td>monthly</td>
<td>1979-current, climatology</td>
</tr>
<tr>
<td>Sea Surface Temperature</td>
<td>weekly</td>
<td>1982-current, climatology</td>
</tr>
<tr>
<td>Zonally Averaged Upper-Level Moisture</td>
<td>monthly</td>
<td>climatology</td>
</tr>
<tr>
<td>Zonally Averaged Ozone Concentration</td>
<td>monthly</td>
<td>climatology</td>
</tr>
</tbody>
</table>

### Topography and Topography Variance

Surface geopotential heights are provided from an averaging of the Navy 10 minute by 10 minute dataset supplied by the National Center for Atmospheric Research (NCAR) to the model’s grid resolution. The original topography is first rotated to the proper grid-orientation which is being run, and then averages the data to the model resolution.

The standard deviation of the subgrid-scale topography is computed by interpolating the 10 minute data to the model’s resolution and re-interpolating back to the 10 minute by 10 minute resolution. The sub-grid scale variance is constructed based on this smoothed dataset.

### Upper Level Moisture

The fizhi package uses climatological water vapor data above 100 mb from the Stratospheric Aerosol and Gas Experiment (SAGE) as input into the model’s radiation packages. The SAGE data is archived as monthly zonal means at 5° latitudinal resolution. The data is interpolated to the model’s grid location and current time, and blended with the GCM’s moisture data. Below 300 mb, the model’s moisture data is used. Above 100 mb, the SAGE data is used. Between 100 and 300 mb, a linear interpolation (in pressure) is performed using the data from SAGE and the GCM.

### Fizhi Diagnostics

Fizhi Diagnostic Menu: [sec:pkg:fizhi:diagnostics]
<table>
<thead>
<tr>
<th>NAME</th>
<th>UNITS</th>
<th>LEVELS</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>UFLUX</td>
<td>N m^-2</td>
<td>1</td>
<td>Surface U-Wind Stress on the atmosphere</td>
</tr>
<tr>
<td>VFLUX</td>
<td>N m^-2</td>
<td>1</td>
<td>Surface V-Wind Stress on the atmosphere</td>
</tr>
<tr>
<td>HFLUX</td>
<td>W m^-2</td>
<td>1</td>
<td>Surface Flux of Sensible Heat</td>
</tr>
<tr>
<td>EFLUX</td>
<td>W m^-2</td>
<td>1</td>
<td>Surface Flux of Latent Heat</td>
</tr>
<tr>
<td>QICE</td>
<td>W m^-2</td>
<td>1</td>
<td>Heat Conduction through Sea-Ice</td>
</tr>
<tr>
<td>RADLWG</td>
<td>W m^-2</td>
<td>1</td>
<td>Net upward LW flux at the ground</td>
</tr>
<tr>
<td>RADSWG</td>
<td>W m^-2</td>
<td>1</td>
<td>Net downward SW flux at the ground</td>
</tr>
<tr>
<td>RI</td>
<td>dimensionless</td>
<td>Nrphys</td>
<td>Richardson Number</td>
</tr>
<tr>
<td>CT</td>
<td>dimensionless</td>
<td>1</td>
<td>Surface Drag coefficient for T and Q</td>
</tr>
<tr>
<td>CU</td>
<td>dimensionless</td>
<td>1</td>
<td>Surface Drag coefficient for U and V</td>
</tr>
<tr>
<td>ET</td>
<td>m^2 s^-1</td>
<td>Nrphys</td>
<td>Diffusivity coefficient for T and Q</td>
</tr>
<tr>
<td>EU</td>
<td>m^2 s^-1</td>
<td>Nrphys</td>
<td>Diffusivity coefficient for U and V</td>
</tr>
<tr>
<td>TURBU</td>
<td>m s^-1 day^-1</td>
<td>Nrphys</td>
<td>U-Momentum Changes due to Turbulence</td>
</tr>
<tr>
<td>TURBV</td>
<td>m s^-1 day^-1</td>
<td>Nrphys</td>
<td>V-Momentum Changes due to Turbulence</td>
</tr>
<tr>
<td>TURBT</td>
<td>deg day^-1</td>
<td>Nrphys</td>
<td>Temperature Changes due to Turbulence</td>
</tr>
<tr>
<td>TURBQ</td>
<td>g/kg/day</td>
<td>Nrphys</td>
<td>Specific Humidity Changes due to Turbulence</td>
</tr>
<tr>
<td>MOISTT</td>
<td>deg day^-1</td>
<td>Nrphys</td>
<td>Temperature Changes due to Moist Processes</td>
</tr>
<tr>
<td>MOISTQ</td>
<td>g/kg/day</td>
<td>Nrphys</td>
<td>Specific Humidity Changes due to Moist Processes</td>
</tr>
<tr>
<td>RADLW</td>
<td>deg day^-1</td>
<td>Nrphys</td>
<td>Net Longwave heating rate for each level</td>
</tr>
<tr>
<td>RADSW</td>
<td>deg day^-1</td>
<td>Nrphys</td>
<td>Net Shortwave heating rate for each level</td>
</tr>
<tr>
<td>PREACC</td>
<td>mm/day</td>
<td>1</td>
<td>Total Precipitation</td>
</tr>
<tr>
<td>PRECON</td>
<td>mm/day</td>
<td>1</td>
<td>Convective Precipitation</td>
</tr>
<tr>
<td>TUFLUX</td>
<td>N m^-2</td>
<td>Nrphys</td>
<td>Turbulent Flux of U-Momentum</td>
</tr>
<tr>
<td>TVFLUX</td>
<td>N m^-2</td>
<td>Nrphys</td>
<td>Turbulent Flux of V-Momentum</td>
</tr>
<tr>
<td>TTFLUX</td>
<td>W m^-2</td>
<td>Nrphys</td>
<td>Turbulent Flux of Sensible Heat</td>
</tr>
<tr>
<td>NAME</td>
<td>UNITS</td>
<td>LEV-ELS</td>
<td>DESCRIPTION</td>
</tr>
<tr>
<td>------------</td>
<td>-------------</td>
<td>---------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>TQFLUX</td>
<td>W m⁻²</td>
<td>Nrphys</td>
<td>Turbulent Flux of Latent Heat</td>
</tr>
<tr>
<td>CN</td>
<td>dimension-less</td>
<td>1</td>
<td>Neutral Drag Coefficient</td>
</tr>
<tr>
<td>WINDS</td>
<td>m s⁻¹</td>
<td>1</td>
<td>Surface Wind Speed</td>
</tr>
<tr>
<td>DTSRF</td>
<td>deg</td>
<td>1</td>
<td>Air/Surface virtual temperature difference</td>
</tr>
<tr>
<td>TG</td>
<td>deg</td>
<td>1</td>
<td>Ground temperature</td>
</tr>
<tr>
<td>TS</td>
<td>deg</td>
<td>1</td>
<td>Surface air temperature (Adiabatic from lowest model layer)</td>
</tr>
<tr>
<td>DTG</td>
<td>deg</td>
<td>1</td>
<td>Ground temperature adjustment</td>
</tr>
<tr>
<td>QG</td>
<td>g kg⁻¹</td>
<td>1</td>
<td>Ground specific humidity</td>
</tr>
<tr>
<td>QS</td>
<td>g kg⁻¹</td>
<td>1</td>
<td>Saturation surface specific humidity</td>
</tr>
<tr>
<td>TGRLW</td>
<td>deg</td>
<td>1</td>
<td>Instantaneous ground temperature used as input to the Longwave radiation subroutine</td>
</tr>
<tr>
<td>ST4</td>
<td>W m⁻²</td>
<td>1</td>
<td>Upward Longwave flux at the ground (\sigma T^4)</td>
</tr>
<tr>
<td>OLR</td>
<td>W m⁻²</td>
<td>1</td>
<td>Net upward Longwave flux at the top of the model</td>
</tr>
<tr>
<td>OLRCLR</td>
<td>W m⁻²</td>
<td>1</td>
<td>Net upward clearsky Longwave flux at the top of the model</td>
</tr>
<tr>
<td>LWCLR</td>
<td>W m⁻²</td>
<td>1</td>
<td>Net upward clearsky Longwave flux at the ground</td>
</tr>
<tr>
<td>LWCLR</td>
<td>deg day⁻¹</td>
<td>Nrphys</td>
<td>Net clearsky Longwave heating rate for each level</td>
</tr>
<tr>
<td>TLW</td>
<td>deg</td>
<td>Nrphys</td>
<td>Instantaneous temperature used as input to the Longwave radiation subroutine</td>
</tr>
<tr>
<td>SHLW</td>
<td>g g⁻¹</td>
<td>Nrphys</td>
<td>Instantaneous specific humidity used as input to the Longwave radiation subroutine</td>
</tr>
<tr>
<td>OZLW</td>
<td>g g⁻¹</td>
<td>Nrphys</td>
<td>Instantaneous ozone used as input to the Longwave radiation subroutine</td>
</tr>
<tr>
<td>CLMOLW</td>
<td>0 − 1</td>
<td>Nrphys</td>
<td>Maximum overlap cloud fraction used in the Longwave radiation subroutine</td>
</tr>
<tr>
<td>CLDTOT</td>
<td>0 − 1</td>
<td>Nrphys</td>
<td>Total cloud fraction used in the Longwave and Shortwave radiation subroutines</td>
</tr>
<tr>
<td>LWG-DOWN</td>
<td>W m⁻²</td>
<td>1</td>
<td>Downwelling Longwave radiation at the ground</td>
</tr>
<tr>
<td>GWDT</td>
<td>deg day⁻¹</td>
<td>Nrphys</td>
<td>Temperature tendency due to Gravity Wave Drag</td>
</tr>
<tr>
<td>RADSWT</td>
<td>W m⁻²</td>
<td>1</td>
<td>Incident Shortwave radiation at the top of the atmosphere</td>
</tr>
<tr>
<td>TAUCLD</td>
<td>per 100 mb</td>
<td>Nrphys</td>
<td>Counted Cloud Optical Depth (non-dimensional) per 100 mb</td>
</tr>
<tr>
<td>TAU-CLDC</td>
<td>Number</td>
<td>Nrphys</td>
<td>Cloud Optical Depth Counter</td>
</tr>
</tbody>
</table>
### Physical Parameterizations - Packages I

#### 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 | $10^4[\text{deg}]/\text{sec}$ | 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/kg | Nrphys | Time-averaged Specific Humidity
RFT | deg/day | Nrphys | Temperature tendency due Rayleigh Friction
PS | mb | 1 | Surface Pressure
QKAVE | (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
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<sup>–2</sup> | 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<sup>–2</sup> | 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<sup>2</sup>/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:

\[
\text{DIAGNOSTIC} = \frac{1}{TTOT} \sum_{t=1}^{TTOT} \text{diag}(t)
\]

where \( TTOT = \frac{NQDIAG}{\Delta t} \). \( NQDIAG \) is the output frequency of the diagnostic, and \( \Delta t \) is the timestep over which the diagnostic is updated.

### Surface Zonal Wind Stress on the Atmosphere (Newton/m<sup>2</sup>)

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

\[
\text{UFLUX} = -\rho C_D W_s u \quad \text{where:} \quad C_D = C_u^2
\]

where \( \rho \) = the atmospheric density at the surface, \( C_D \) is the surface drag coefficient, \( C_u \) is the dimensionless surface exchange coefficient for momentum (see diagnostic number 10), \( W_s \) is the magnitude of the surface layer wind, and \( u \) is the zonal wind in the lowest model layer.

### Surface Meridional Wind Stress on the Atmosphere (Newton/m<sup>2</sup>)

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

\[
\text{VFLUX} = -\rho C_D W_s v \quad \text{where:} \quad C_D = C_u^2
\]

where \( \rho \) = 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:

\[
HFLUX = \rho c_p C_H W_s (\theta_{\text{surface}} - \theta_{\text{Nrphys}})
\]

where \( \rho \) = the atmospheric density at the surface, \( c_p \) is the specific heat of air, \( C_H \) is the dimensionless surface heat transfer coefficient, \( W_s \) is the magnitude of the surface layer wind, \( C_u \) is the dimensionless surface exchange coefficient for momentum (see diagnostic number 10), \( C_t \) is the dimensionless surface exchange coefficient for heat and moisture (see diagnostic number 9), and \( \theta \) is the potential temperature at the surface and at the bottom model level.

Surface Flux of Latent Heat (Watts/m²)

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:

\[
EFLUX = \rho \beta L C_H W_s (q_{\text{surface}} - q_{\text{Nrphys}})
\]

where \( \rho \) = the atmospheric density at the surface, \( \beta \) is the fraction of the potential evapotranspiration actually evaporated, \( L \) is the latent heat of evaporation, \( C_H \) is the dimensionless surface heat transfer coefficient, \( W_s \) is the magnitude of the surface layer wind, \( C_u \) is the dimensionless surface exchange coefficient for momentum (see diagnostic number 10), \( C_t \) is the dimensionless surface exchange coefficient for heat and moisture (see diagnostic number 9), and \( q_{\text{surface}} \) and \( q_{\text{Nrphys}} \) are the specific humidity at the surface and at the bottom model level, respectively.

Heat Conduction Through Sea Ice (Watts/m²)

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.

\[
QICE = \frac{C_{ti} H_i}{C_t} (T_i - T_g)
\]

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/m²)

\[
RADLWG = F^{Net}_{LW,Nrphys+1} = F_{LW,Nrphys+1}^{\uparrow} - F_{LW,Nrphys+1}^{\downarrow}
\]

where Nrphys+1 indicates the lowest model edge-level, or \( p = p_{\text{surf}} \). \( F_{LW}^{\uparrow} \) is the upward Longwave flux and \( F_{LW}^{\downarrow} \) is the downward Longwave flux.
Net downward shortwave Flux at the surface \((\text{Watts/m}^2)\)

\[
\text{RADSWG} = F_{SW, Nrphys+1}^{\downarrow} = F_{SW, Nrphys+1}^{\uparrow} = F_{SW, Nrphys+1}^{\downarrow} - F_{SW, Nrphys+1}^{\uparrow}
\]

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

**Richardson number (dimensionless)**

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

\[
RI = \frac{\frac{\rho}{\rho_0} \frac{\theta}{\theta_0} z}{(u z)^2 + (v z)^2} = \frac{c_p \theta_v z P^\kappa}{(u z)^2 + (v z)^2}
\]

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:

\[
\text{CT} = -\frac{(w'\theta')}{u_\ast \Delta \theta} = -\frac{(w'q')}{u_\ast \Delta q} = \frac{k}{(\psi_h + \psi_y)}
\]

where \(\psi_h\) is the surface layer non-dimensional temperature change and \(\psi_y\) is the viscous sublayer non-dimensional temperature or moisture change:

\[
\psi_h = \int_{\zeta_0}^{\zeta} \phi_h \frac{d\zeta}{\zeta} \quad \text{and} \quad \psi_y = 0.55 \left( Pr \frac{2^{3/3} - 0.2}{\nu^{1/2}} \frac{h_0 u_\ast - h_{0,re}}{u_\ast,ref} \right)^{1/2}
\]

and: \(h_0 = 30 \zeta_0\) with a maximum value over land of 0.01

\(\phi_h\) is the similarity function of \(\zeta\), which expresses the stability dependance of the temperature and moisture gradients, specified differently for stable and unstable layers according to \(k\) is the Von Karman constant, \(\zeta\) is the non-dimensional stability parameter, \(Pr\) is the Prandtl number for air, \(\nu\) is the molecular viscosity, \(\zeta_0\) is the surface roughness length, \(u_\ast\) is the surface stress velocity (see diagnostic number 67), and the subscript ref refers to a reference value.

**CU - Surface Exchange Coefficient for Momentum (dimensionless)**

The surface exchange coefficient is obtained from the similarity functions for the stability dependant flux profile relationships:

\[
\text{CU} = \frac{u_\ast}{W_s} = \frac{k}{\psi_m}
\]
where $\psi_m$ is the surface layer non-dimensional wind shear:

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

$\phi_m$ is the similarity function of $\zeta$, which expresses the stability dependence of the temperature and moisture gradients, specified differently for stable and unstable layers according to $\zeta$. $k$ is the Von Karman constant, $\zeta$ is the non-dimensional stability parameter, $u_*$ is the surface stress velocity (see diagnostic number 67), and $W_s$ is the magnitude of the surface layer wind.

**ET - Diffusivity Coefficient for Temperature and Moisture (m^2/sec)**

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

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

where $q$ is the turbulent velocity, or $\sqrt{2 \times \text{turbulent kinetic energy}}$, $q_e$ is the turbulence velocity derived from the more simple level 2.0 model, which describes equilibrium turbulence, $\ell$ is the master length scale related to the layer depth, $S_H$ is a function of $G_H$ and $G_M$, the dimensionless buoyancy and wind shear parameters, respectively, or a function of $G_{H_e}$ and $G_{M_e}$, the equilibrium dimensionless buoyancy and wind shear parameters. Both $G_H$ and $G_M$, and their equilibrium values $G_{H_e}$ and $G_{M_e}$, are functions of the Richardson number.

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

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

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

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

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

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

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

where $q$ is the turbulent velocity, or $\sqrt{2 \times \text{turbulent kinetic energy}}$, $q_e$ is the turbulence velocity derived from the more simple level 2.0 model, which describes equilibrium turbulence, $\ell$ is the master length scale related to the layer depth, $S_M$ is a function of $G_H$ and $G_M$, the dimensionless buoyancy and wind shear parameters, respectively, or a function of $G_{H_e}$ and $G_{M_e}$, the equilibrium dimensionless buoyancy and wind shear parameters. Both $G_H$ and $G_M$, and their equilibrium values $G_{H_e}$ and $G_{M_e}$, are functions of the Richardson number.

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

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

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

---

MITgcm Documentation, Release 1.0

Chapter 5. Physical Parameterizations - Packages I
where $C_u$ is the dimensionless exchange coefficient for momentum from the surface layer similarity functions (see diagnostic number 10), $u_s$ is the surface friction velocity (see diagnostic number 67), $C_D$ is the surface drag coefficient, and $W_s$ is the magnitude of the surface layer wind.

**TURBU - Zonal U-Momentum changes due to Turbulence (m/sec/day)**

The tendency of U-Momentum due to turbulence is written:

$$TURBU = u_t = z(-u'w') = z(K_m u_z)$$

The Helfand and Labraga level 2.5 scheme models the turbulent flux of u-momentum in terms of $K_m$, and the equation has the form of a diffusion equation.

**TURBV - Meridional V-Momentum changes due to Turbulence (m/sec/day)**

The tendency of V-Momentum due to turbulence is written:

$$TURBV = v_t = z(-v'w') = z(K_m v_z)$$

The Helfand and Labraga level 2.5 scheme models the turbulent flux of v-momentum in terms of $K_m$, and the equation has the form of a diffusion equation.

**TURBT - Temperature changes due to Turbulence (deg/day)**

The tendency of temperature due to turbulence is written:

$$TURBT = T = P^c \theta_t = P^c z(-w'\theta') = P^c z(K_h \theta_v z)$$

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:

$$TURBQ = q_t = z(-w'q') = z(K_h q_z)$$

The Helfand and Labraga level 2.5 scheme models the turbulent flux of temperature in terms of $K_h$, and the equation has the form of a diffusion equation.

**MOISTT - Temperature Changes Due to Moist Processes (deg/day)**

$$MOISTT = Tt\big|_c + Tt\big|_{ls}$$
where:

\[ T_t|_c = R \sum_i \left( \frac{m_B}{c_p} \Gamma_s \right)_i \quad \text{and} \quad T_t|_{ls} = \frac{L}{c_p} (q^* - q) \]

\[ \Gamma_s = g \eta sp \]

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 \( \eta \), 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 \( \alpha \), while \( R \) is the rain re-evaporation adjustment.

**MOISTQ - Specific Humidity Changes Due to Moist Processes (g/kg/day)**

\[ \text{MOISTQ} = q_t|_c + q_t|_{ls} \]

where:

\[ q_t|_c = R \sum_i \left( \frac{m_B}{L} (\Gamma_h - \Gamma_s) \right)_i \quad \text{and} \quad q_t|_{ls} = (q^* - q) \]

\[ \Gamma_s = g \eta sp \quad \text{and} \quad \Gamma_h = g \eta hp \]

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 \( \eta \), 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 \( \alpha \), while \( R \) is the rain re-evaporation adjustment.

**RADLW - Heating Rate due to Longwave Radiation (deg/day)**

The net longwave heating rate is calculated as the vertical divergence of the net terrestrial radiative fluxes. Both the clear-sky and cloudy-sky longwave fluxes are computed within the longwave routine. The subroutine calculates the clear-sky flux, \( F_{LW}^{\text{clearsky}} \), first. For a given cloud fraction, the clear line-of-sight probability \( C(p, p') \) is computed from the current level pressure \( p \) to the model top pressure, \( p' = p_{top} \), 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:

\[ F_{LW} = C(p, p') \cdot F_{LW}^{\text{clearsky}} \]

Finally, the net longwave heating rate is calculated as the vertical divergence of the net terrestrial radiative fluxes:

\[ \rho c_p T_t = -z F_{LW}^{\text{NET}} \]

or

\[ \text{RADLW} = \frac{g}{c_p \pi} \sigma F_{LW}^{\text{NET}} \]

where \( g \) is the acceleration due to gravity, \( c_p \) is the heat capacity of air at constant pressure, and

\[ F_{LW}^{\text{NET}} = F_{LW}^{\uparrow} - F_{LW}^{\downarrow} \]
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:

\[
\rho c_p T_t = -z F(\text{cloudy})^{NET}_{SW} \cdot \text{RADSWT},
\]

or

\[
\text{RADSW} = \frac{g}{c_p \pi} \sigma F(\text{cloudy})^{NET}_{SW} \cdot \text{RADSWT},
\]

where \( g \) is the acceleration due to gravity, \( c_p \) is the heat capacity of air at constant pressure, RADSWT is the true incident Shortwave radiation at the top of the atmosphere (See Diagnostic #48), and

\[
F(\text{cloudy})^{NET}_{SW} = F(\text{cloudy})^{\uparrow}_{SW} - F(\text{cloudy})^{\downarrow}_{SW}
\]

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

For a change in specific humidity due to moist processes, \( \Delta q_{\text{moist}} \), the vertical integral or total precipitable amount is given by:

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

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

PRECON - Convective Precipitation (mm/day)

For a change in specific humidity due to sub-grid scale cumulus convective processes, \( \Delta q_{\text{cum}} \), the vertical integral or total precipitable amount is given by:

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

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

TUFLUX - Turbulent Flux of U-Momentum (Newton/m^2)

The turbulent flux of u-momentum is calculated for diagnostic purposes from the eddy coefficient for momentum:

\[
\text{TUFLUX} = \rho \langle \bar{u} \bar{w} \rangle = \rho (-K_m U z)
\]

where \( \rho \) is the air density, and \( K_m \) is the eddy coefficient.
TVFLUX - Turbulent Flux of V-Momentum (Newton/m^2)

The turbulent flux of v-momentum is calculated for diagnostic purposes only from the eddy coefficient for momentum:

$$TVFLUX = \rho \langle \dot{w} \rangle = \rho (-K_m V z)$$

where $\rho$ is the air density, and $K_m$ is the eddy coefficient.

TTFLUX - Turbulent Flux of Sensible Heat (Watts/m^2)

The turbulent flux of sensible heat is calculated for diagnostic purposes only from the eddy coefficient for heat and moisture:

$$TTFLUX = c_p \rho \langle \dot{w} \theta' \rangle = c_p \rho \langle -K_h \theta v_z \rangle$$

where $\rho$ is the air density, and $K_h$ is the eddy coefficient.

TQFLUX - Turbulent Flux of Latent Heat (Watts/m^2)

The turbulent flux of latent heat is calculated for diagnostic purposes only from the eddy coefficient for heat and moisture:

$$TQFLUX = L \rho \langle \dot{w} q' \rangle = L \rho (-K_h q z)$$

where $\rho$ is the air density, and $K_h$ is the eddy coefficient.

CN - Neutral Drag Coefficient (dimensionless)

The drag coefficient for momentum obtained by assuming a neutrally stable surface layer:

$$CN = \frac{k}{\ln \left( \frac{h}{z_0} \right)}$$

where $k$ is the Von Karman constant, $h$ is the height of the surface layer, and $z_0$ is the surface roughness.

NOTE: CN is not available through model version 5.3, but is available in subsequent versions.

WINDS - Surface Wind Speed (meter/sec)

The surface wind speed is calculated for the last internal turbulence time step:

$$WINDS = \sqrt{u_{Nphys}^2 + v_{Nphys}^2}$$

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

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

$$DTSRF = (\theta_{v,Nphys+1} - \theta_{v,Nphys}) P_{surf}$$

where

$$\theta_{v,Nphys+1} = \frac{T_g}{P_{surf}} (1 + .609 q_{Nphys+1})$$

and

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

$\beta$ is the surface potential evapotranspiration coefficient ($\beta = 1$ over oceans), $q^*(T_g, P_s)$ is the saturation specific humidity at the ground temperature and surface pressure, level $Nphys$ refers to the lowest model level and level $Nphys + 1$ 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:

\[ TG = \frac{C_g}{\lambda C_s} \frac{\Omega}{2\omega} = \sqrt{\left(0.386 + 0.536W + 0.15W^2\right)2x10^{-3} \frac{86400}{2\pi}}. \]

Here, the thermal conductivity, \( \lambda \), is equal to \( 2x10^{-3} \frac{\text{ly cm}}{\text{sec K}} \), the angular velocity of the earth, \( \omega \), is written as \( 86400 \frac{\text{sec}}{\text{day}} \) 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 \( \theta \) is constant in that layer. The surface temperature is therefore:

\[ TS = \theta_{N\text{rphys}} P_{surf}^{\text{c}} \]

DTG - Surface Temperature Adjustment (deg K)

The change in surface temperature from one turbulence time step to the next, solved using the Ground Temperature Equation (see diagnostic number 30) is calculated:

\[ DTG = T_g^n - T_g^{n-1} \]

where superscript \( n \) refers to the new, updated time level, and the superscript \( n - 1 \) refers to the value at the previous turbulence time level.

QG - Ground Specific Humidity (g/kg)

The ground specific humidity is obtained by interpolating between the specific humidity at the lowest model level and the specific humidity of a saturated ground. The interpolation is performed using the potential evapotranspiration function:

\[ QG = q_{N\text{rphys}+1} = q_{N\text{rphys}} + \beta(q^*(T_g, P_s) - q_{N\text{rphys}}) \]

where \( \beta \) is the surface potential evapotranspiration coefficient (\( \beta = 1 \) over oceans), and \( q^*(T_g, P_s) \) is the saturation specific humidity at the ground temperature and surface pressure.

QS - Saturation Surface Specific Humidity (g/kg)

The surface saturation specific humidity is the saturation specific humidity at the ground temperature and surface pressure:

\[ QS = q^*(T_g, P_s) \]
TGRLW - Instantaneous ground temperature used as input to the Longwave radiation subroutine (deg)

\[ \text{TGRLW} = T_g(\lambda, \phi, n) \]

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

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

\[ \text{ST4} = \sigma T^4 \]

where \( \sigma \) is the Stefan-Boltzmann constant and \( T \) is the temperature.

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

\[ \text{OLR} = F^{\text{NET}}_{\text{LW,top}} \]

where \( p_{top} \) 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)

\[ \text{OLRCLR} = F^{\text{NET}}_{\text{LW,top}}(\text{clearsky}) \]

where \( p_{top} \) 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)

\[ \text{LWGCLR} = F^{\text{NET}}_{\text{LW,Nrphys+1}}(\text{clearsky}) \]

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

LWCLR - Heating Rate due to Clearsky Longwave Radiation (deg/day)

The net longwave heating rate is calculated as the vertical divergence of the net terrestrial radiative fluxes. Both the clear-sky and cloudy-sky longwave fluxes are computed within the longwave routine. The subroutine calculates the clear-sky flux, \( F^{\text{clearsky}}_{\text{LW}} \), 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, $p' = p_{\text{top}}$, and the model surface pressure, $p' = p_{\text{surf}}$, for the upward and downward radiative fluxes. (see Section [sec:fizhi:radcloud]). The cloudy-sky flux is then obtained as:

$$F_{\text{LW}} = C(p, p') \cdot F_{\text{clearsky}}^{\text{LW}},$$

Thus, $\text{LWCLR}$ is defined as the net longwave heating rate due to the vertical divergence of the clear-sky longwave radiative flux:

$$\rho c_p T_{\text{clearsky}} = -z F(\text{clearsky})_{\text{LW}}^{\text{NET}},$$

or

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

where $g$ is the acceleration due to gravity, $c_p$ is the heat capacity of air at constant pressure, and

$$F(\text{clearsky})_{\text{LW}}^{\text{NET}} = F(\text{clearsky})_{\text{LW}}^{\uparrow} - F(\text{clearsky})_{\text{LW}}^{\downarrow}.$$

### TLW - Instantaneous temperature used as input to the Longwave radiation subroutine (deg)

$$\text{TLW} = T(\lambda, \phi, \text{level}, n)$$

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)

$$\text{SHLW} = q(\lambda, \phi, \text{level}, n)$$

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)

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

where $\text{OZ}$ 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].

$$\text{CLMOLW} = CLMO_{RAS,LW}(\lambda, \phi, \text{level})$$
CLDTOT - Total cloud fraction used in LW and SW Radiation (0-1)

CLDTOT is the time-averaged total cloud fraction that has been filled by the Relaxed Arakawa/Schubert and Large-scale Convection schemes and will be used in the Longwave and Shortwave Radiation packages. For a complete description of cloud/radiative interactions, see Section [sec:fizhi:radcloud].

\[ \text{CLDTOT} = F_{RAS} + F_{LS} \]

where \( F_{RAS} \) is the time-averaged cloud fraction due to sub-grid scale convection, and \( F_{LS} \) is the time-averaged cloud fraction due to precipitating and non-precipitating large-scale moist processes.

CLMOSW - Maximum Overlap cloud fraction used in SW Radiation (0-1)

CLMOSW is the time-averaged maximum overlap cloud fraction that has been filled by the Relaxed Arakawa/Schubert Convection scheme and will be used in the Shortwave Radiation algorithm. These are convective clouds whose radiative characteristics are assumed to be correlated in the vertical. For a complete description of cloud/radiative interactions, see Section [sec:fizhi:radcloud].

\[ \text{CLMOSW} = CLOM_{RAS,SW}(\lambda, \phi, \text{level}) \]

CLROSW - Random Overlap cloud fraction used in SW Radiation (0-1)

CLROSW is the time-averaged random overlap cloud fraction that has been filled by the Relaxed Arakawa/Schubert and Large-scale Convection schemes and will be used in the Shortwave Radiation algorithm. These are convective and large-scale clouds whose radiative characteristics are not assumed to be correlated in the vertical. For a complete description of cloud/radiative interactions, see Section [sec:fizhi:radcloud].

\[ \text{CLROSW} = CLR_{RAS,LargeScale,SW}(\lambda, \phi, \text{level}) \]

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

\[ \text{RADSWT} = \frac{S_0}{R_a^2} \cdot \cos \phi_z \]

where \( S_0 \) is the extra-terrestrial solar constant, \( R_a \) is the earth-sun distance in Astronomical Units, and \( \cos \phi_z \) is the cosine of the zenith angle. It should be noted that RADSWT, as well as OSR and OSRCLR, are calculated at the top of the atmosphere (p=0 mb). However, the OLR and OLRCLR diagnostics are currently calculated at \( p = p_{top} \) (0.0 mb for the GCM).

EVAP - Surface Evaporation (mm/day)

The surface evaporation is a function of the gradient of moisture, the potential evapotranspiration fraction and the eddy exchange coefficient:

\[ \text{EVAP} = \rho \beta K_h (q_{surface} - q_{Nrvphys}) \]

where \( \rho \) is the atmospheric density at the surface, \( \beta \) is the fraction of the potential evapotranspiration actually evaporated (\( \beta = 1 \) over oceans), \( K_h \) is the turbulent eddy exchange coefficient for heat and moisture at the surface in \( m/sec \) and \( q_{surface} \) and \( q_{Nrvphys} \) 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)**

\[ \text{DUDT} = u_{t_Dynamics} + u_{t_Moist} + u_{t_Turbulence} + u_{t_Analysis} \]

**DVDT - Total Zonal V-Wind Tendency (m/sec/day)**

\[ \text{DVDT} = v_{t_Dynamics} + v_{t_Moist} + v_{t_Turbulence} + v_{t_Analysis} \]

**DTDT - Total Temperature Tendency (deg/day)**

\[ \text{DTDT} = Tt_{Dynamics} + Tt_{MoistProcesses} + Tt_{ShortwaveRadiation} + Tt_{LongwaveRadiation} + Tt_{Turbulence} + Tt_{Analysis} \]

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

\[ \text{DQDT} = q_{t_Dynamics} + q_{t_MoistProcesses} + q_{t_Turbulence} + q_{t_Analysis} \]

**USTAR - Surface-Stress Velocity (m/sec)**

The surface stress velocity, or the friction velocity, is the wind speed at the surface layer top impeded by the surface drag:

\[ \text{USTAR} = C_u W_s \quad \text{where :} \quad C_u = \frac{k}{\psi_m} \]

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

**Z0 - Surface Roughness Length (m)**

Over the land surface, the surface roughness length is interpolated to the local time from the monthly mean data of . Over the ocean, the roughness length is a function of the surface-stress velocity, \( u_* \).

\[ Z0 = c_1 u_*^3 + c_2 u_*^2 + c_3 u_* + c_4 + c_5 u_* \]

where the constants are chosen to interpolate between the reciprocal relation of for weak winds, and the piecewise linear relation of for moderate to large winds.

**FRQTRB - Frequency of Turbulence (0-1)**

The fraction of time when turbulence is present is defined as the fraction of time when the turbulent kinetic energy exceeds some minimum value, defined here to be \( 0.005 \text{ m}^2/\text{sec}^2 \). When this criterion is met, a counter is incremented. The fraction over the averaging interval is reported.
PBL - Planetary Boundary Layer Depth (mb)

The depth of the PBL is defined by the turbulence parameterization to be the depth at which the turbulent kinetic energy reduces to ten percent of its surface value.

\[ \text{PBL} = P_{PBL} - P_{surface} \]

where \( P_{PBL} \) is the pressure in mb at which the turbulent kinetic energy reaches one tenth of its surface value, and \( P_s \) is the surface pressure.

SWCLR - Clear sky Heating Rate due to Shortwave Radiation (deg/day)

The net Shortwave heating rate is calculated as the vertical divergence of the net solar radiative fluxes. The clear-sky and cloudy-sky shortwave fluxes are calculated separately. For the clear-sky case, the shortwave fluxes and heating rates are computed with both CLMO (maximum overlap cloud fraction) and CLRO (random overlap cloud fraction) set to zero (see Section [sec:fizhi:radcloud]). The shortwave routine is then called a second time, for the cloudy-sky case, with the true time-averaged cloud fractions CLMO and CLRO being used. In all cases, a normalized incident shortwave flux is used as input at the top of the atmosphere.

The heating rate due to Shortwave Radiation under clear skies is defined as:

\[ \rho c_p T t = -z F(\text{clear})_{SW}^{\text{NET}} \cdot \text{RADSWT} , \]

or

\[ \text{SWCLR} = \frac{g}{c_p} F(\text{clear})_{SW}^{\text{NET}} \cdot \text{RADSWT} . \]

where \( g \) is the accleration due to gravity, \( c_p \) is the heat capacity of air at constant pressure, \( \text{RADSWT} \) is the true incident shortwave radiation at the top of the atmosphere (See Diagnostic #48), and

\[ F(\text{clear})_{SW}^{\text{NET}} = F(\text{clear})_{SW}^{\uparrow} - F(\text{clear})_{SW}^{\downarrow} \]

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

\[ \text{OSR} = F_{SW,\text{top}}^{\text{NET}} \]

where top indicates the top of the first model layer used in the shortwave radiation routine. In the GCM, \( p_{SW,\text{top}} = 0 \) mb.

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

\[ \text{OSRCLR} = F(\text{clearsky})_{SW,\text{top}}^{\text{NET}} \]

where top indicates the top of the first model layer used in the shortwave radiation routine. In the GCM, \( p_{SW,\text{top}} = 0 \) mb.
CLDMAS - Convective Cloud Mass Flux (kg/m^2)

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

\[ \text{CLDMAS} = \eta m_B \]

where \( \eta \) is the entrainment, normalized by the cloud base mass flux, and \( m_B \) is the cloud base mass flux. \( m_B \) and \( \eta \) are defined explicitly in Section 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.

\[ \text{UAVE} = u(\lambda, \phi, level, t) \]

Note, UAVE is computed and stored on the staggered C-grid.

VAVE - Time-Averaged Meridional V-Wind (m/sec)

The diagnostic VAVE is simply the time-averaged Meridional V-Wind over the NVAVE output frequency. This is contrasted to the instantaneous Meridional V-Wind which is archived on the Prognostic Output data stream.

\[ \text{VAVE} = v(\lambda, \phi, level, t) \]

Note, VAVE is computed and stored on the staggered C-grid.

TAVE - Time-Averaged Temperature (Kelvin)

The diagnostic TAVE is simply the time-averaged Temperature over the NTAVE output frequency. This is contrasted to the instantaneous Temperature which is archived on the Prognostic Output data stream.

\[ \text{TAVE} = T(\lambda, \phi, level, t) \]

QAVE - Time-Averaged Specific Humidity (g/kg)

The diagnostic QAVE is simply the time-averaged Specific Humidity over the NQAVE output frequency. This is contrasted to the instantaneous Specific Humidity which is archived on the Prognostic Output data stream.

\[ \text{QAVE} = q(\lambda, \phi, level, t) \]

PAVE - Time-Averaged Surface Pressure - 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.

\[ \text{PAVE} = \pi(\lambda, \phi, level, t) = p_s(\lambda, \phi, level, t) - p_T \]
QQAVE - Time-Averaged Turbulent Kinetic Energy (m/sec)^2

The diagnostic QQAVE is simply the time-averaged prognostic Turbulent Kinetic Energy produced by the GCM Turbulence parameterization over the NQQAVE output frequency. This is contrasted to the instantaneous Turbulent Kinetic Energy which is archived on the Prognostic Output data stream.

$$QQAVE = qq(\lambda, \phi, level, t)$$

Note, QQAVE is computed and stored at the “mass-point” locations on the staggered C-grid.

SWGCLR - Net downward clearsky Shortwave flux at the surface (Watts/m^2^)

$$SWGCLR = F(\text{clearsky})^{\text{Net,SW,}N_{\text{phys}}+1} = F(\text{clearsky})^{\downarrow,SW,N_{\text{phys}}+1} - F(\text{clearsky})^{\uparrow,SW,N_{\text{phys}}+1}$$

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

DIABU - Total Diabatic Zonal U-Wind Tendency (m/sec/day)

DIABU is the total time-tendency of the Zonal U-Wind due to Diabatic processes and the Analysis forcing.

$$DIABU = u_t^{Moist} + u_t^{Turbulence} + u_t^{Analysis}$$

DIABV - Total Diabatic Meridional V-Wind Tendency (m/sec/day)

DIABV is the total time-tendency of the Meridional V-Wind due to Diabatic processes and the Analysis forcing.

$$DIABV = v_t^{Moist} + v_t^{Turbulence} + v_t^{Analysis}$$

DIABT Total Diabatic Temperature Tendency (deg/day)

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

$$DIABT = T_t^{Moist Processes} + T_t^{Shortwave Radiation} + T_t^{Longwave Radiation} + T_t^{Turbulence} + T_t^{Analysis}$$

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

$$T_t^{Diabatic} = T_t^{Moist Processes} + T_t^{Shortwave Radiation} + T_t^{Longwave Radiation} + T_t^{Turbulence}$$

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

$$T_t^{Diabatic} = \frac{p^\kappa}{\pi} \pi \theta T_{Diabatic}$$

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

$$DIABT = \frac{p^\kappa}{\pi} (\pi \theta T_{Diabatic} + \pi \theta T_{Analysis})$$
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.

\[ DIABQ = q_{\text{Moist Processes}} + q_{\text{Turbulence}} + q_{\text{Analysis}} \]

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

\[ q_{\text{Diabatic}} = q_{\text{Moist Processes}} + q_{\text{Turbulence}} \]

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

\[ q_{\text{Diabatic}} = \frac{1}{\pi} q_{\text{Diabatic}} \]

Thus, \( DIABQ \) may be written as

\[ DIABQ = \frac{1}{\pi} (\pi q_{\text{Diabatic}} + \pi q_{\text{Analysis}}) \]

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.

\[ VINTUQ = \frac{\int_{\text{surf}}^{\text{top}} uq \rho \, dz}{\int_{\text{surf}}^{\text{top}} \rho \, dz} \]

Using \( \rho \delta z = -\frac{\delta p}{g} = -\frac{1}{g} \delta p \), we have

\[ VINTUQ = \int_{0}^{1} uq dp \]

VINTVQ - Vertically Integrated Moisture Flux (m/sec g/kg)

The vertically integrated moisture flux due to the meridional v-wind is obtained by integrating \( vq \) over the depth of the atmosphere at each model timestep, and dividing by the total mass of the column.

\[ VINTVQ = \frac{\int_{\text{surf}}^{\text{top}} vq \rho \, dz}{\int_{\text{surf}}^{\text{top}} \rho \, dz} \]

Using \( \rho \delta z = -\frac{\delta p}{g} = -\frac{1}{g} \delta p \), we have

\[ VINTVQ = \int_{0}^{1} vq dp \]

VINTUT - Vertically Integrated Heat Flux (m/sec deg)

The vertically integrated heat flux due to the zonal u-wind is obtained by integrating \( uT \) over the depth of the atmosphere at each model timestep, and dividing by the total mass of the column.

\[ VINTUT = \frac{\int_{\text{surf}}^{\text{top}} uT \rho \, dz}{\int_{\text{surf}}^{\text{top}} \rho \, dz} \]
Or,

\[
VINTUT = \int_0^1 uTdp
\]

**VINTVT - Vertically Integrated Heat Flux (m/sec \cdot \text{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.

\[
\frac{\int_{\text{surf}}^{\text{top}} vT \rho dz}{\int_{\text{surf}}^{\text{top}} \rho dz}
\]

Using \(\rho \Delta z = -\frac{\Delta p}{g}\), we have

\[
VINTVT = \int_0^1 vT dp
\]

**CLDFRC - Total 2-Dimensional Cloud Fraction (0-1)**

If we define the time-averaged random and maximum overlapped cloudiness as CLRO and CLMO respectively, then the probability of clear sky associated with random overlapped clouds at any level is \((1-\text{CLRO})\) while the probability of clear sky associated with maximum overlapped clouds at any level is \((1-\text{CLMO})\). The total clear sky probability is given by \((1-\text{CLRO}) \times (1-\text{CLMO})\), thus the total cloud fraction at each level may be obtained by \(1 - (1-\text{CLRO}) \times (1-\text{CLMO})\).

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, \(p' = p_{\text{top}}\), or the model surface pressure, \(p' = p_{\text{surf}}\), is simply 1.0 minus the largest maximum overlap cloud value along the line-of-site, ie.

\[
1 - \text{MAX}_{p'}^p (\text{CLMO}_p)
\]

Thus, even in the time-averaged sense it is assumed that the maximum overlap clouds are correlated in the vertical. The clear line-of-site probability associated with random overlap clouds is defined to be the product of the clear sky probabilities at each level along the line-of-site, ie.

\[
\prod_{p} (1 - \text{CLRO}_p)
\]

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

\[
1 - \left(1 - \text{MAX}_{p'}^p (\text{CLMO}_p)\right) \prod_{p} (1 - \text{CLRO}_p)
\]

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

\[
\text{CLDFRC} = 1 - \left(1 - \text{MAX}_{l=1}^{\text{Nrphys}} (\text{CLMO}_l)\right) \prod_{l=1}^{\text{Nrphys}} (1 - \text{CLRO}_l)
\]

For a complete description of cloud/radiative interactions, see Section [sec:fizhi:radcloud].
QINT - Total Precipitable Water (gm/cm^2)

The Total Precipitable Water is defined as the vertical integral of the specific humidity, given by:

\[
QINT = \int_{surf}^{top} \rho q \, dz = \frac{\pi}{g} \int_{0}^{1} q \, dp
\]

where we have used the hydrostatic relation \( \rho \Delta z = -\frac{\Delta p}{g} \).

U2M Zonal U-Wind at 2 Meter Depth (m/sec)

The u-wind at the 2-meter depth is determined from the similarity theory:

\[
U2M = \frac{u_s}{k} \psi_{m2m} \frac{u_{sl}}{W_s} = \frac{\psi_{m2m}}{\psi_{m,sl}} u_{sl}
\]

where \( \psi_m(2m) \) is the non-dimensional wind shear at two meters, and the subscript \( sl \) refers to the height of the top of the surface layer. If the roughness height is above two meters, \( U2M \) is undefined.

V2M - Meridional V-Wind at 2 Meter Depth (m/sec)

The v-wind at the 2-meter depth is determined from the similarity theory:

\[
V2M = \frac{u_s}{k} \psi_{m2m} \frac{v_{sl}}{W_s} = \frac{\psi_{m2m}}{\psi_{m,sl}} v_{sl}
\]

where \( \psi_m(2m) \) is the non-dimensional wind shear at two meters, and the subscript \( sl \) refers to the height of the top of the surface layer. If the roughness height is above two meters, \( V2M \) is undefined.

T2M - Temperature at 2 Meter Depth (deg K)

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

\[
T2M = P^\kappa \left( \frac{\theta_s}{k} (\psi_{h2m} + \psi_g) + \theta_{surf} \right) = P^\kappa (\theta_{surf} + \psi_{h2m} \frac{\psi_{h,sl} + \psi_g}{\psi_{h,sl} + \psi_g} (\theta_{sl} - \theta_{surf}))
\]

where:

\[
\theta_s = -\frac{(\vec{w} \theta')}{u_s}
\]

where \( \psi_h(2m) \) is the non-dimensional temperature gradient at two meters, \( \psi_g \) is the non-dimensional temperature gradient in the viscous sublayer, and the subscript \( sl \) refers to the height of the top of the surface layer. If the roughness height is above two meters, \( T2M \) 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:

\[
Q2M = P^\kappa \left( \frac{q_s}{k} (\psi_{h2m} + \psi_g) + q_{surf} \right) = P^\kappa (q_{surf} + \psi_{h2m} \frac{\psi_{h,sl} + \psi_g}{\psi_{h,sl} + \psi_g} (q_{sl} - q_{surf}))
\]
where:

\[ q_s = \frac{-\langle w'q' \rangle}{u_s} \]

where \( \psi_h(2m) \) is the non-dimensional temperature gradient at two meters, \( \psi_g \) is the non-dimensional temperature gradient in the viscous sublayer, and the subscript \( sl \) refers to the height of the top of the surface layer. If the roughness height is above two meters, \( Q2M \) 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:

\[ U10M = \frac{u_s}{k} \frac{\psi_{10m}}{\psi_{sl}} \frac{u_{sl}}{W_s} = \frac{\psi_{10m}}{\psi_{sl}} u_{sl} \]

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:

\[ V10M = \frac{u_s}{k} \frac{\psi_{10m}}{\psi_{sl}} \frac{v_{sl}}{W_s} = \frac{\psi_{10m}}{\psi_{sl}} v_{sl} \]

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:

\[ T10M = P \kappa \left( \frac{\theta_s}{\psi_{10m}} (\psi_{h10m} + \psi_g) + \theta_{surf} \right) = P \kappa \left( \theta_{surf} + \frac{\psi_{10m}}{\psi_{sl}} + \psi_g (\theta_{sl} - \theta_{surf}) \right) \]

where:

\[ \theta_s = \frac{-\langle w'\theta' \rangle}{u_s} \]

where \( \psi_{h}(10m) \) is the non-dimensional temperature gradient at two meters, \( \psi_g \) is the non-dimensional temperature gradient in the viscous sublayer, and the subscript \( sl \) refers to the height of the top of the surface layer.

**Q10M - Specific Humidity at 10 Meter Depth (g/kg)**

The specific humidity at the 10-meter depth is an interpolation between the surface specific humidity and the model lowest level specific humidity using the ratio of the non-dimensional temperature gradient at the two levels:

\[ Q10M = P \kappa \left( \frac{q_s}{\psi_{10m}} (\psi_{h10m} + \psi_g) + q_{surf} \right) = P \kappa \left( q_{surf} + \frac{\psi_{10m}}{\psi_{sl}} + \psi_g (q_{sl} - q_{surf}) \right) \]
where:

\[
q_* = -\frac{(u'q')}{u_s}
\]

where \(\psi_b(10m)\) is the non-dimensional temperature gradient at two meters, \(\psi_g\) is the non-dimensional temperature gradient in the viscous sublayer, and the subscript \(sl\) refers to the height of the top of the surface layer.

**DTRAIN - Cloud Detrainment Mass Flux (kg/m^2)**

The amount of cloud mass moved per RAS timestep at the cloud detrainment level is written:

\[
DTRAIN = \eta_{rD} m_B
\]

where \(r_D\) is the detrainment level, \(m_B\) is the cloud base mass flux, and \(\eta\) is the entrainment, defined in Section 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:

\[
QFILL = q_{n+1}^{\text{final}} - q_{n+1}^{\text{initial}}
\]

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:

$$\text{frzmlt} = (T_f - SST) \frac{c_{sw} \rho_{sw} \Delta z}{\Delta t}$$

where $c_{sw}$ is seawater heat capacity, $\rho_{sw}$ is the seawater density, $\Delta z$ is the ocean model upper layer thickness and $\Delta t$ is the model (tracer) timestep. The freezing temperature, $T_f = \mu S$ is a function of the salinity.

1. Provided there is no ice present and frzmlt is less than 0, the surface tendencies of wind, heat and freshwater are calculated as usual (i.e. 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:

$$q_1 = -c_i \ast T_f + L_i$$

$$q_2 = -c_f T_{mlt} + c_i (T_{mlt} - T_f) + L_i (1 - \frac{T_{mlt}}{T_f})$$
where $c_f$ is specific heat of liquid fresh water, $c_i$ is the specific heat of fresh ice, $L_i$ is latent heat of freezing, $\rho_i$ is density of ice and $T_{mlt}$ is melting temperature of ice with salinity of 1. The height of a new layer of ice is

$$h_{\text{new}} = \frac{\text{esurp}\Delta t}{q_{0\text{av}}}$$

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

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 $h_{\text{imin}}$; this determines the ice fraction compact. If there is already ice in the grid cell, the new ice must have the same height and the new ice fraction is

$$i_f = (1 - \hat{i}_f) \frac{h_{\text{new}}}{h_i}$$

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 $h_{\text{ihig}}$ then freezing energy comes from the full grid cell, ice growth does not occur under original ice due to freezing water.)

**subroutine ICE_THERM**

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

If the current ice height is less than $h_{\text{imin}}$ 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 $h_{\text{imin}}$ 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, $f_{\text{swint}}$, 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 $\cdot$ indicates the value at the previous timestep):

$$F_s(T_s) = F_s(\hat{T}_s) + \frac{\partial F_s(T_s)}{\partial T_s}(T_s - \hat{T}_s)$$

where,

$$F_s = F_{\text{sensible}} + F_{\text{latent}} + F_{\text{down}} + F_{\text{up}} + (1 - \alpha)F_{\text{shortwave}}$$

and

$$\frac{dF_s}{dT} = \frac{dF_{\text{sensible}}}{dT} + \frac{dF_{\text{latent}}}{dT} + \frac{dF_{\text{down}}}{dT} + (1 - \alpha)F_{\text{shortwave}}.$$ 

$F_s$ 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

$$f_{\text{swint}} = (1 - e^{\kappa_i h_i})(1 - \alpha)F_{\text{shortwave}}$$

where $\kappa_i$ is bulk extinction coefficient.

The conductive flux to the surface is

$$F_u = K_{1/2}(T_1 - T_s)$$
where \( K_{1/2} \) is the effective conductive coupling of the snow-ice layer between the surface and the mid-point of the upper layer of ice. \( K_1/2 = \frac{4 K_i K_s}{h_i + 4 K_i h_s} \). \( K_i \) 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 \( T \). 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.

\[
q_1 = -c_f T_f + c_i (T_f - T) + L_i (1 - \frac{T_f}{T})
\]

\[
q_2 = -c_i T + L_i
\]

where \( c_f \) is specific heat of liquid fresh water, \( c_i \) is the specific heat of fresh ice, and \( L_i \) is latent heat of melting fresh ice.

From the new ice temperatures, we can calculate the energy flux at the surface available for melting (if \( T_s = 0 \)) and the energy at the ocean-ice interface for either melting or freezing.

\[
E_{top} = (F_s - K_{1/2} (T_s - T_1)) \Delta t
\]

\[
E_{bot} = \left(\frac{4K_i(T_2 - T_f)}{h_i}\right) - F_b) \Delta t
\]

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, \( F_b = c_{sw} \rho_{sw} \gamma (T_{sst} - T_f) u^* \). \( \gamma \) 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:

\[
q_2 = \frac{\hat{h}_i}{2} + \Delta h_i q_{bot}
\]

If there is a ice layer and the overlying air temperature is below 0°C then any precipitation, \( P \) joins the snow layer:

\[
\Delta h_s = - \frac{P \rho_f}{\rho_s} \Delta t,
\]

\( \rho_f \) and \( \rho_s \) are the fresh water and snow densities. Any evaporation, similarly, removes snow or ice from the surface. We also calculate the snow age here, in case it is needed for the surface albedo calculation (see 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 \textit{ice\_therm.F} now calculates the heat and fresh water fluxes affecting the ocean model surface layer. The heat flux:

\[ q_{\text{net}} = \text{fswocn} - F_b - \frac{\text{esurp}}{\Delta t} \]

is composed of the shortwave flux that has passed through the ice layer and is absorbed by the water, \text{fswocn} = \text{QQ}, the ocean flux to the ice \( F_b \), and the surplus energy left over from the melting, \text{esurp}. The fresh water flux is determined from the amount of fresh water and salt in the ice/snow system before and after the timestep.

(There is a provision for fractional ice: If ice height is above \text{hihig} then all energy from freezing at sea surface is used only in the open water aparts of the cell (ie. \( F_b \) will only have the conduction term). The melt energy is partitioned by \text{frac\_energy} between melting ice height and ice extent. However, once ice height drops below \text{himon0} then all energy melts ice extent.)

\textbf{subroutine SFC\_ALBEDO}

The routine \textit{ice\_therm.F} calls this routine to determine the surface albedo. There are two calculations provided here:

1. from LANL CICE model

\[ \alpha = f_s \alpha_s + (1 - f_s) (\alpha_{i_{\text{min}}} + (\alpha_{i_{\text{max}}} - \alpha_{i_{\text{min}}}) (1 - e^{-h_i/h_a})) \]

where \( f_s \) is 1 if there is snow, 0 if not; the snow albedo, \( \alpha_s \), has two values depending on whether \( T_s \) < 0 or not; \( \alpha_{i_{\text{min}}} \) and \( \alpha_{i_{\text{max}}} \) are ice albedos for thin melting ice, and thick bare ice respectively, and \( h_a \) is a scale height.

2. From GISS model (Hansen et al 1983)

\[ \alpha = \alpha_i e^{-h_i/h_s} + \alpha_s (1 - e^{-h_i/h_s}) \]

where \( \alpha_i \) is a constant albedo for bare ice, \( h_s \) is a scale height and \( \alpha_s \) is a variable snow albedo.

\[ \alpha_s = \alpha_1 + \alpha_2 e^{-\lambda_a a_s} \]

where \( \alpha_1 \) is a constant, \( \alpha_2 \) depends on \( T_s \), \( a_s \) is the snow age, and \( \lambda_a \) is a scale frequency. The snow age is calculated in \textit{ice\_therm.F} and is given in equation 41 in Hansen et al (1983).

\textbf{subroutine NEW\_LAYERS\_WINTON}

The subroutine \textit{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 \textit{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:

\[ q_1 = f_1 \dot{q}_1 + (1 - f_1) \dot{q}_2 \]

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.

\textbf{Initializing subroutines}

\textit{ice\_init.F}: Set ice variables to zero, or reads in pickup information from \texttt{pickup.ic} (which was written out in \textit{checkpoint.F})

\textit{ice\_readparms.F}: Reads \texttt{data.ice}

\textbf{5.6. Sea Ice Packages}
Diagnostic subroutines

`ice_ave.F`: Keeps track of means of the ice variables

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

Common Blocks

`ICE.h`: Ice Variables, also `relaxlat` and `startIceModel`

`ICE_DIAGS.h`: matrices for diagnostics: averages of fields from `ice_diags.F`

`BULKF_ICE_CONSTANTS.h` (in `BULKF` package): all the parameters need by the ice model

Input file `DATA.ICE`

Here we need to set `StartIceModel`: which is 1 if the model starts from no ice; and 0 if there is a pickup file with the ice matrices (`pickup.ic`) which is read in `ice_init.F` and written out in `checkpoint.F`. The parameter `relaxlat` defines the latitude poleward of which there is no relaxing of surface $T$ or $S$ to observations. This avoids the relaxation forcing the ice model at these high latitudes.

(Note: `hicemin` is set to 0 here. If the provision for allowing grid cells to have both open water and seaice is ever implemented, this would be greater than 0)

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**

<table>
<thead>
<tr>
<th>&lt;-Name-&gt;</th>
<th>Levs</th>
<th>&lt;-parsing code-&gt;</th>
<th>&lt;-- Units --&gt;</th>
<th>&lt;- Tile (max=80c)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SI_Fract</td>
<td>1</td>
<td>SM P M1</td>
<td>0-1</td>
<td>Sea-Ice fraction [0-1]</td>
</tr>
<tr>
<td>SI_Thick</td>
<td>1</td>
<td>SM PC197M1</td>
<td>m</td>
<td>Sea-Ice thickness (area weighted average)</td>
</tr>
<tr>
<td>SI_SnowH</td>
<td>1</td>
<td>SM PC197M1</td>
<td>m</td>
<td>Snow thickness over Sea-Ice (area weighted)</td>
</tr>
<tr>
<td>SI_Tsrf</td>
<td>1</td>
<td>SM C197M1</td>
<td>degC</td>
<td>Surface Temperature over Sea-Ice (area weighted)</td>
</tr>
<tr>
<td>SI_Tice1</td>
<td>1</td>
<td>SM C197M1</td>
<td>degC</td>
<td>Sea-Ice Temperature, 1st layer (area weighted)</td>
</tr>
<tr>
<td>SI_Tice2</td>
<td>1</td>
<td>SM C197M1</td>
<td>degC</td>
<td>Sea-Ice Temperature, 2nd layer (area weighted)</td>
</tr>
<tr>
<td>SI_Qice1</td>
<td>1</td>
<td>SM C198M1</td>
<td>J/kg</td>
<td>Sea-Ice enthalpy, 1st layer (mass weighted)</td>
</tr>
<tr>
<td>SI_Qice2</td>
<td>1</td>
<td>SM C198M1</td>
<td>J/kg</td>
<td>Sea-Ice enthalpy, 2nd layer (mass weighted)</td>
</tr>
<tr>
<td>SIalbedo</td>
<td>1</td>
<td>SM PC197M1</td>
<td>0-1</td>
<td>Sea-Ice Albedo [0-1] (area weighted average)</td>
</tr>
<tr>
<td>SIsnwAge</td>
<td>1</td>
<td>SM P M1</td>
<td>s</td>
<td>Snow age over Sea-Ice</td>
</tr>
<tr>
<td>SIsnwPrc</td>
<td>1</td>
<td>SM C197M1</td>
<td>kg/m^2/s</td>
<td>Snow precip. (+=dw) over Sea-Ice (area weighted)</td>
</tr>
</tbody>
</table>
### References


Hunke, E.C and W.H. Lipscomb, circa 2001: CICE: the Los Alamos Sea Ice Model Documentation and Software User’s Manual. LACC-98-16v.2. (note: this documentation is no longer available as CICE has progressed to a very different version 3)


### 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 Jinhun 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 sub-routines is given in Section Section 5.6.2.3. Input fields, units and sign conventions are summarized in Section [sec:pkg:seaice:fields:sub:units], 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 preprocessor flags in the `seaice`-package.

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

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Default value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SEAICEwriteState</td>
<td>T</td>
<td>write sea ice state to file</td>
</tr>
<tr>
<td>SEAICEuseDYNAMICS</td>
<td>T</td>
<td>use dynamics</td>
</tr>
<tr>
<td>SEAICEuseJFNK</td>
<td>F</td>
<td>use the JFNK-solver</td>
</tr>
<tr>
<td>SEAICEuseTEM</td>
<td>F</td>
<td>use truncated ellipse method</td>
</tr>
<tr>
<td>SEAICEuseStrImpCpl</td>
<td>F</td>
<td>use strength implicit coupling in LSR/JFNK</td>
</tr>
</tbody>
</table>

Continued on next page
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Setting</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SEAICE_useMetricTerms</td>
<td>T</td>
<td>use metric terms in dynamics</td>
</tr>
<tr>
<td>SEAICE_useEVPpickup</td>
<td>T</td>
<td>use EVP pickups</td>
</tr>
<tr>
<td>SEAICE_useFluxForm</td>
<td>F</td>
<td>use flux form for 2nd central difference advection scheme</td>
</tr>
<tr>
<td>SEAICE_restoreUnderIce</td>
<td>F</td>
<td>enable restoring to climatology under ice</td>
</tr>
<tr>
<td>useHB87stressCoupling</td>
<td>F</td>
<td>turn on ice-ocean stress coupling following</td>
</tr>
<tr>
<td>usePW79thermodynamics</td>
<td>T</td>
<td>flag to turn off zero-layer-thermodynamics for testing</td>
</tr>
<tr>
<td>SEAICE_eadvHeff/Area/Snow/Salt</td>
<td>T</td>
<td>flag to turn off advection of scalar state variables</td>
</tr>
<tr>
<td>SEAICE_useFlooding</td>
<td>T</td>
<td>use flood-freeze algorithm</td>
</tr>
<tr>
<td>SEAICE_no_slip</td>
<td>F</td>
<td>switch between free-slip and no-slip boundary conditions</td>
</tr>
<tr>
<td>SEAICE_deltaTherm</td>
<td>dTracerLev(1)</td>
<td>thermodynamic timestep</td>
</tr>
<tr>
<td>SEAICE_deltaDyn</td>
<td>dTracerLev(1)</td>
<td>dynamic timestep</td>
</tr>
<tr>
<td>SEAICE_deltaUevp</td>
<td>0</td>
<td>EVP sub-cycling time step, values &gt; 0 turn on EVP</td>
</tr>
<tr>
<td>SEAICE_euseEVPstar</td>
<td>F</td>
<td>use modified EVP* instead of EVP</td>
</tr>
<tr>
<td>SEAICE_euseEVPrev</td>
<td>F</td>
<td>use yet another variation on EVP*</td>
</tr>
<tr>
<td>SEAICE_eEnEvPstarSteps</td>
<td>UNSET</td>
<td>number of modified EVP* iteration</td>
</tr>
<tr>
<td>SEAICE_evpAlpha</td>
<td>UNSET</td>
<td>EVP* parameter</td>
</tr>
<tr>
<td>SEAICE_evpBeta</td>
<td>UNSET</td>
<td>EVP* parameter</td>
</tr>
<tr>
<td>SEAICE_eEvPcoeff</td>
<td>UNSET</td>
<td>aEVP parameter</td>
</tr>
<tr>
<td>SEAICE_eEvPcStar</td>
<td>4</td>
<td>aEVP parameter {KDL16}</td>
</tr>
<tr>
<td>SEAICE_eEvPalphaMin</td>
<td>5</td>
<td>aEVP parameter {KDL16}</td>
</tr>
<tr>
<td>SEAICE_elasticParm</td>
<td>\frac{1}{E_0}</td>
<td>EVP parameter</td>
</tr>
<tr>
<td>SEAICE_evpTauRelax</td>
<td>\Delta t_{EVP}</td>
<td>relaxation time scale \Delta t for EVP waves</td>
</tr>
<tr>
<td>SEAICE_eNonLinIterMax</td>
<td>10</td>
<td>maximum number of JFNK-Newton iterations (non-linear)</td>
</tr>
<tr>
<td>SEAICE_eLinearIterMax</td>
<td>10</td>
<td>maximum number of JFNK-Krylov iterations (linear)</td>
</tr>
<tr>
<td>SEAICE_eJFNK_lnIter</td>
<td>(off)</td>
<td>start line search after “lnIter” Newton iterations</td>
</tr>
<tr>
<td>SEAICE_eNonLinTol</td>
<td>1.0E-05</td>
<td>non-linear tolerance parameter for JFNK solver</td>
</tr>
<tr>
<td>JFNKgamma_lin_min/max</td>
<td>0.10/0.99</td>
<td>tolerance parameters for linear JFNK solver</td>
</tr>
<tr>
<td>JFNKres_tFac</td>
<td>UNSET</td>
<td>tolerance parameter for FGMRES residual</td>
</tr>
<tr>
<td>SEAICE_eJFNKepsilon</td>
<td>1.0E-06</td>
<td>step size for the FD-Jacobian-times-vector</td>
</tr>
<tr>
<td>SEAICE_dumpFreq</td>
<td>dumpFreq</td>
<td>dump frequency</td>
</tr>
<tr>
<td>SEAICE_eSaveFreq</td>
<td>taveFreq</td>
<td>time-averaging frequency</td>
</tr>
<tr>
<td>SEAICE_dump_mdsio</td>
<td>T</td>
<td>write snap-shot using MDSIO</td>
</tr>
<tr>
<td>SEAICE_eSave_mdsio</td>
<td>T</td>
<td>write TimeAverage using MDSIO</td>
</tr>
<tr>
<td>SEAICE_dump_mnc</td>
<td>F</td>
<td>write snap-shot using MNC</td>
</tr>
<tr>
<td>SEAICE_eSave_mnc</td>
<td>F</td>
<td>write TimeAverage using MNC</td>
</tr>
<tr>
<td>SEAICE_eInitialHeff</td>
<td>0.00000E+00</td>
<td>initial sea-ice thickness</td>
</tr>
<tr>
<td>SEAICE_drag</td>
<td>2.00000E-03</td>
<td>air-ice drag coefficient</td>
</tr>
<tr>
<td>OCEAN_drag</td>
<td>1.00000E-03</td>
<td>air-ocean drag coefficient</td>
</tr>
<tr>
<td>SEAICE_eWaterDrag</td>
<td>5.50000E+00</td>
<td>water-ice drag</td>
</tr>
<tr>
<td>SEAICE_eDryIceAlb</td>
<td>7.50000E-01</td>
<td>winter albedo</td>
</tr>
<tr>
<td>SEAICE_eWetIceAlb</td>
<td>6.60000E-01</td>
<td>summer albedo</td>
</tr>
<tr>
<td>SEAICE_eDrySnowAlb</td>
<td>8.40000E-01</td>
<td>dry snow albedo</td>
</tr>
<tr>
<td>SEAICE_eWetSnowAlb</td>
<td>7.00000E-01</td>
<td>wet snow albedo</td>
</tr>
<tr>
<td>SEAICE_eWaterAlbedo</td>
<td>1.00000E-01</td>
<td>water albedo</td>
</tr>
<tr>
<td>SEAICE_eStrength</td>
<td>2.75000E+04</td>
<td>sea-ice strength (P^*)</td>
</tr>
<tr>
<td>SEAICE_eCStar</td>
<td>20.00000E+00</td>
<td>sea-ice strength parameter (C^*)</td>
</tr>
<tr>
<td>SEAICE_eRhoAir</td>
<td>1.3</td>
<td>density of air (kg/m:math:^3)</td>
</tr>
<tr>
<td>SEAICE_eCPAir</td>
<td>1004</td>
<td>specific heat of air (J/kg/K)</td>
</tr>
<tr>
<td>SEAICE_eLHvap</td>
<td>2,500,000 (or val ue)</td>
<td>latent heat of evaporation</td>
</tr>
<tr>
<td>SEAICE_eLHFusion</td>
<td>334,000 (or value)</td>
<td>latent heat of fusion</td>
</tr>
</tbody>
</table>

5.6. Sea Ice Packages  

167
Table 5.16 – continued from previous page

<table>
<thead>
<tr>
<th>Variable</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SEAICE_lhSublim</td>
<td>2,834,000</td>
<td>latent heat of sublimation</td>
</tr>
<tr>
<td>SEAICE_dalton</td>
<td>1.75E-03</td>
<td>sensible heat transfer coefficient</td>
</tr>
<tr>
<td>SEAICE_iceConduct</td>
<td>2.16560E+00</td>
<td>sea-ice conductivity</td>
</tr>
<tr>
<td>SEAICE_snowConduct</td>
<td>3.10000E-01</td>
<td>snow conductivity</td>
</tr>
<tr>
<td>SEAICE_emissivity</td>
<td>5.50000E-08</td>
<td>Stefan-Boltzman</td>
</tr>
<tr>
<td>SEAICE_snowThick</td>
<td>1.96000E+00</td>
<td>cutoff snow thickness</td>
</tr>
<tr>
<td>SEAICE_freeze</td>
<td>-1.96000E+00</td>
<td>freezing temp. of sea water</td>
</tr>
<tr>
<td>SEAICE_saltFrac</td>
<td>0.0</td>
<td>salinity newly formed ice (fraction of ocean surface salinity)</td>
</tr>
<tr>
<td>SEAICE_hice_reg</td>
<td>0.0</td>
<td>scaling factor for ice-ocean stress</td>
</tr>
<tr>
<td>Heff/Area/HsnowFile/Hsalt</td>
<td>UNSET</td>
<td>initial fields for variables HEFF/AREA/HSNOW/HSALT</td>
</tr>
<tr>
<td>LSR_ERROR</td>
<td>1.00000E-04</td>
<td>sets accuracy of LSR solver</td>
</tr>
<tr>
<td>DIFF1</td>
<td>0.0</td>
<td>parameter used in advect.F</td>
</tr>
<tr>
<td>HO</td>
<td>5.00000E-01</td>
<td>demarcation ice thickness (AKA lead closing parameter $h_0$)</td>
</tr>
<tr>
<td>MAX_HEFF</td>
<td>1.00000E+01</td>
<td>maximum ice thickness</td>
</tr>
<tr>
<td>MIN_ATEMP</td>
<td>-5.00000E+01</td>
<td>minimum air temperature</td>
</tr>
<tr>
<td>MIN_LWDOWN</td>
<td>6.00000E+01</td>
<td>minimum downward longwave</td>
</tr>
<tr>
<td>MAX_TICE</td>
<td>3.00000E+01</td>
<td>maximum ice temperature</td>
</tr>
<tr>
<td>MIN_TICE</td>
<td>-5.00000E+01</td>
<td>minimum ice temperature</td>
</tr>
<tr>
<td>MAX_TICE</td>
<td>10</td>
<td>iterations for ice heat budget</td>
</tr>
<tr>
<td>SEAICE_area_reg</td>
<td>1.00000E-10</td>
<td>reduce derivative singularities</td>
</tr>
<tr>
<td>SEAICE_hice_reg</td>
<td>1.00000E-5</td>
<td>minimum concentration to regularize ice thickness</td>
</tr>
<tr>
<td>SEAICE_multDim</td>
<td>1</td>
<td>number of ice categories for thermodynamics</td>
</tr>
<tr>
<td>SEAICE_useMultDimSnow</td>
<td>F</td>
<td>use SEAICE_multDim snow categories</td>
</tr>
</tbody>
</table>

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 component of the MITgcm. Heat, fresh water fluxes and surface
stresses are computed from the atmospheric state and – by default – modified by the ice model at every time step.

The ice dynamics models that are most widely used for large-scale climate studies are the viscous-plastic (VP) model
[Hib79], the cavitating fluid (CF) model [FWDH92], and the elastic-viscous-plastic (EVP) model [HD97]. Compared
to the VP model, the CF model does not allow ice shear in calculating ice motion, stress, and deformation. EVP models
approximate VP by adding an elastic term to the equations for easier adaptation to parallel computers. Because of its
higher accuracy in plastic solution and relatively simpler formulation, compared to the EVP model, we decided to use
the VP model as the default dynamic component of our ice model. To do this we extended the line successive over
relaxation (LSOR) method of [ZH97] for use in a parallel configuration. An EVP model and a free-drift implementation
can be selected with runtime flags.

Compatibility with ice-thermodynamics thsice package

Note, that by default the seaice-package includes the original so-called zero-layer thermodynamics following with
a snow cover as in . The zero-layer thermodynamic model assumes that ice does not store heat and, therefore, tends
to exaggerate the seasonal variability in ice thickness. This exaggeration can be significantly reduced by using ’s []
three-layer thermodynamic model that permits heat storage in ice. Recently, the three-layer thermodynamic model has
been reformulated by . The reformulation improves model physics by representing the brine content of the upper ice
with a variable heat capacity. It also improves model numerics and consumes less computer time and memory.

The Winton sea-ice thermodynamics have been ported to the MIT GCM; they currently reside under pkg/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 thermody-
namics. 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

\[ \dot{\omega}_{\text{ocean}} - m \nabla \phi(0) + \mathbf{F}, (5.1) \]
where \( m = m_i + m_s \) is the ice and snow mass per unit area; \( u = u_i + v_j \) 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; \( \tau \) and \( \tau_{\text{ocean}} \) are the wind-ice and ocean-ice stresses, respectively; \( g \) is the gravity acceleration; \( \nabla \phi(0) \) is the gradient (or tilt) of the sea surface height; \( \phi(0) = g\eta + p_a/p_0 + mg/p_0 \) is the sea surface height potential in response to ocean dynamics \((\eta)\), to atmospheric pressure loading \((p_a/p_0)\), where \( p_0 \) is a reference density) and a term due to snow and ice loading ; and \( \nabla \cdot \sigma \) is the divergence of the internal ice stress tensor \( \sigma_{ij} \). Advection of sea-ice momentum is neglected. The wind and ice-ocean stress terms are given by

\[
\tau_{\text{air}} = \rho_{\text{air}} C_{\text{air}} |U_{\text{air}} - u|R_{\text{air}}(U_{\text{air}} - u),
\]

\[
\tau_{\text{ocean}} = \rho_{\text{ocean}} C_{\text{ocean}} |U_{\text{ocean}} - u|R_{\text{ocean}}(U_{\text{ocean}} - u),
\]

where \( U_{\text{air/ocean}} \) are the surface winds of the atmosphere and surface currents of the ocean, respectively; \( C_{\text{air/ocean}} \) are air and ocean drag coefficients; \( \rho_{\text{air/ocean}} \) are reference densities; and \( R_{\text{air/ocean}} \) are rotation matrices that act on the wind/current vectors.

**Viscous-Plastic (VP) Rheology**

For an isotropic system the stress tensor \( \sigma_{ij} \) \((i, j = 1, 2)\) can be related to the ice strain rate and strength by a nonlinear viscous-plastic (VP) constitutive law:

\[
\sigma_{ij} = 2\eta(\dot{e}_{ij}, P)\dot{e}_{ij} + \left[\zeta(\dot{e}_{ij}, P) - \eta(\dot{e}_{ij}, P)\right]\dot{e}_{kk}\delta_{ij} - \frac{P}{2}\delta_{ij}.
\]

(5.2)

The ice strain rate is given by

\[
\dot{e}_{ij} = \frac{1}{2}\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right).
\]

The maximum ice pressure \( P_{\text{max}} \), a measure of ice strength, depends on both thickness \( h \) and compactness (concentration) \( c \):

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

with the constants \( P^* \) (run-time parameter \( \text{SEAICE_strength} \)) and \( C^* = 20 \). The nonlinear bulk and shear viscosities \( \eta \) and \( \zeta \) are functions of ice strain rate invariants and ice strength such that the principal components of the stress lie on an elliptical yield curve with the ratio of major to minor axis \( e \) equal to 2; they are given by:

\[
\zeta = \min\left(\frac{P_{\text{max}}}{2 \text{max}(\Delta, \Delta_{\text{min}})}, \zeta_{\text{max}}\right)
\]

\[
\eta = \frac{\zeta}{e^2}
\]

with the abbreviation

\[
\Delta = \left[(\epsilon_{11}^2 + \epsilon_{22}^2)(1 + e^{-2}) + 4e^{-2}\epsilon_{12}^2 + 2\epsilon_{11}\epsilon_{22}(1 - e^{-2})\right]^{1/2}.
\]

The bulk viscosities are bounded above by imposing both a minimum \( \Delta_{\text{min}} \) (for numerical reasons, run-time parameter \( \text{SEAICE_EPS} \) with a default value of \( 10^{-10} \) s\(^{-1}\)) and a maximum \( \zeta_{\text{max}} = P_{\text{max}}/\Delta^* \), where \( \Delta^* = (5 \times 10^{12}/2 \times 10^4) \) s\(^{-1}\). (There is also the option of bounding \( \zeta \) from below by setting run-time parameter \( \text{SEAICE_zetaMin} > 0 \), 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 $\zeta$ by a smooth (differentiable) expression:

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

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

**LSR and JFNK solver**

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

$$
A(x) x = b(x). \quad (5.3)
$$

The solution vector $x$ consists of the two velocity components $u$ and $v$ that contain the velocity variables at all grid points and at one time level. The standard (and default) method for solving Eq. (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 $A(x^{k-1}) x^k = b(x^{k-1})$ 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 $F(x) = A(x) x - b(x)$ to finding the roots of a multivariate Taylor expansion of the residual $F$ around the previous $(k-1)$ estimate $x^{k-1}$:

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

with the Jacobian $J \equiv F'$. The root $F(x^{k-1} + \delta x^k) = 0$ is found by solving

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

for $\delta x^k$. The next ($k$-th) estimate is given by $x^k = x^{k-1} + a \delta x^k$. In order to avoid overshoots the factor $a$ is iteratively reduced in a line search ($a = 1, \frac{1}{2}, \frac{1}{3}, \frac{1}{4}, \ldots$) until $\|F(x^k)\| < \|F(x^{k-1})\|$, where $\| \cdot \| = \int f \cdot dx^2$ 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 $J$ explicitly is often avoided as “too error prone and time consuming”. Instead, Krylov methods only require the action of $J$ on an arbitrary vector $w$ and hence allow a matrix free algorithm for solving Eq. (5.5). The action of $J$ can be approximated by a first-order Taylor series expansion:

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

or computed exactly with the help of automatic differentiation (AD) tools. `SEAICE_JFNKepsilon` sets the step size $\epsilon$.

We use the Flexible Generalized Minimum RESidual method with right-hand side preconditioning to solve Eq. (5.5) iteratively starting from a first guess of $\delta x_0^k = 0$. For the preconditioning matrix $P$ we choose a simplified form of
the system matrix $A(x^{k-1})$ where $x^{k-1}$ is the estimate of the previous Newton step $k-1$. The transformed equation (5.5) becomes

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

The Krylov method iteratively improves the approximate solution to Eq. (5.7) in subspace $(r_0, JP^{-1}r_0, (JP^{-1})^2r_0, \ldots, (JP^{-1})^m r_0)$ with increasing $m$; $r_0 = -F(x^{k-1}) - J(x^{k-1})δx_0^k$ is the initial residual of Eq. (5.5); $r_0 = -F(x^{k-1})$ with the first guess $δx^k_0 = 0$. We allow a Krylov-subspace of dimension $m = 50$ and we do not use restarts. The preconditioning operation involves applying $P^{-1}$ to the basis vectors $v_0, v_1, v_2, \ldots, v_m$ of the Krylov subspace. This operation is approximated by solving the linear system $Pw = v_i$. Because $P \approx A(x^{k-1})$, 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: $\|F(x^k)\| < γ_{nl}\|F(x^0)\|$. Within a non-linear iteration, the linear FGMRES solver is terminated when the residual is smaller than $γ_k\|F(x^{k-1})\|$ where $γ_k$ is determined by

$$γ_k = \begin{cases} γ_0 & \text{for } \|F(x^{k-1})\| \geq r, \\ \max\left( γ_{min}, \frac{\|F(x^{k-1})\|}{\|F(x^0)\|} \right) & \text{for } \|F(x^{k-1})\| < r, \end{cases} \tag{5.8}$$

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 $γ_0 = JFNKgamma_lin_max = 0.99, γ_{min} = JFNKgamma_lin_min = 0.1$, and $r = JFNKres_tFac \times \|F(x^0)\|$ 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,

$$\frac{1}{E} \frac{∂σ_{ij}}{∂t} + \frac{1}{2η}σ_{ij} + \frac{η - ζ}{4ζη}σ_{kk}δ_{ij} + \frac{P}{4ζ}δ_{ij} = \dot{ε}_{ij}. \tag{5.9}$$

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 = SEAICE_deltaTdyn/120`) within the physical ocean model time step (although this parameter is under debate), to allow for elastic waves to disappear. Because the scheme does not require a matrix inversion it is fast in spite of the small internal timestep and simple to implement on parallel computers. For completeness, we repeat the equations for the components of the stress tensor $σ_1 = σ_{11} + 2τ_{21}, σ_2 = σ_{12} - σ_{22},$ and $σ_12$. Introducing the divergence $D_D = \dot{ε}_{11} + \dot{ε}_{22},$ and the horizontal tension and shearing strain rates, $D_T = \dot{ε}_{11} - \dot{ε}_{22}$ and $D_S = 2\dot{ε}_{12},$ respectively,
and using the above abbreviations, the equations (5.9) can be written as:

\[
\frac{\partial \sigma_1}{\partial t} + \frac{\sigma_1}{2T} + \frac{P}{2T} = \frac{P}{2T} D_D \\
\frac{\partial \sigma_2}{\partial t} + \frac{\sigma_2}{2T} = \frac{P}{2T} D_T \\
\frac{\partial \sigma_{12}}{\partial t} + \frac{\sigma_{12}}{2T} = \frac{P}{4T} D_S
\]

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 \( \Delta t \). \( E_0 = \frac{1}{3} \) 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 \( \Delta t (\text{deltaTmon}) \). 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 solutions [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:

\[
m \frac{D \mathbf{u}}{D t} \approx m \frac{\mathbf{u}^{p+1} - \mathbf{u}^n}{\Delta t} + \beta \frac{\mathbf{u}^{p+1} - \mathbf{u}^p}{\Delta t_{\text{EVP}}} \tag{5.10}\]

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 \( \mathbf{u}^{p+1} \rightarrow \mathbf{u}^{n+1} \), converges to 0. In this way EVP can be re-interpreted as a pure iterative solver where the sub-cycling has no association with time-resolution (\( \Delta t_{\text{EVP}} \)). Using the terminology of , the evolution equations of stress \( \sigma_{ij} \) and momentum \( \mathbf{u} \) can be written as:

\[
\sigma_{ij}^{p+1} = \sigma_{ij}^p + \frac{1}{\alpha} \left( \sigma_{ij}(\mathbf{u}^p) - \sigma_{ij}^p \right), \\
\mathbf{u}^{p+1} = \mathbf{u}^p + \frac{1}{\beta} \left( \frac{\Delta t}{m} \nabla \cdot \mathbf{u}^{p+1} + \frac{\Delta t}{m} \mathbf{R}^p + \mathbf{u}_n - \mathbf{u}^p \right).
\]

\( \mathbf{R} \) contains all terms in the momentum equations except for the rheology terms and the time derivative; \( \alpha \) and \( \beta \) are free parameters (SEAICE_evpAlpha, SEAICE_evpBeta) that replace the time stepping parameters SEAICE_deltaTevp (\( \Delta t_{\text{EVP}} \)), SEAICE_elasticPlarm (\( E_0 \)), or SEAICE_evpTauRelax (\( T \)). \( \alpha \) and \( \beta \) determine the speed of convergence and the stability. Usually, it makes sense to use \( \alpha = \beta \), and SEAICE允许EVPstarSteps \( \gg (\alpha, \beta) \) [KDL15]. Currently, there is no termination criterion and the number of mEVP iterations is fixed to SEAICE允许EVPstarSteps.

In order to use mEVP in the MITgcm, set SEAICEuseEVPstar = .TRUE., in data.seaice. If SEAICEuseEVPrev = .TRUE., the actual form of equations ([eq:evpstresstensor12]) 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 $\alpha$ is set dynamically based on the stability criterion

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

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 $\alpha$ can be small and more slowly in areas where the equations are stiff. In practice, aEVP leads to an overall better convergence than mEVP [KDL16]. To use aEVP in the MITgcm set $\text{SEAICEaEVPcoeff} = \tilde{c}$; this also sets the default values of $\text{SEAICEaEVPcStar} (c = 4)$ and $\text{SEAICEaEVPalphaMin} (\alpha_{\text{min}} = 5)$. Good convergence has been obtained with setting these values [KDL16]: $\text{SEAICEaEVPcoeff} = 0.5$, $\text{SEAICEeVPstarSteps} = 500$, $\text{SEAICEuseEVPstar} = .\text{TRUE.}$, $\text{SEAICEuseEVPrev} = .\text{TRUE.}$

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 $\eta$ is capped to suppress any tensile stress:

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

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 $\tau_{\text{ocean}}$ in Eq. (5.6.2.3). This stress is applied directly to the surface layer of the ocean model. An alternative ocean stress formulation is given by [HB87]. Rather than applying $\tau_{\text{ocean}}$ 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:

$$\delta(z) (\tau_{\text{air}} + F)/\rho_0,$$

see also Eq. 2 of [HB87]. The disadvantage of this formulation is that now the velocity in the surface layer of the ocean that is used to advect tracers, is really an average over the ocean surface velocity and the ice velocity leading to an inconsistency as the ice temperature and salinity are different from the oceanic variables. To turn on the stress formulation of [HB87], set `useHB87StressCoupling=.TRUE.` in `data.seaice`.

**Finite-volume discretization of the stress tensor divergence**

On an Arakawa C grid, ice thickness and concentration and thus ice strength $P$ and bulk and shear viscosities $\zeta$ and $\eta$ are naturally defined a C-points in the center of the grid cell. Discretization requires only averaging of $\zeta$ and $\eta$ to vorticity or Z-points (or $\zeta$-points, but here we use $Z$ in order avoid confusion with the bulk viscosity) at the bottom left corner of the cell to give $\tilde{\zeta}$ and $\tilde{\eta}$. In the following, the superscripts indicate location at Z or C points, distance across the cell (F), along the cell edge (G), between $u$-points (U), $v$-points (V), and C-points (C). The control volumes of the $u$- and $v$-equations in the grid cell at indices $(i, j)$ are $A^u_{i,j}$ and $A^v_{i,j}$, respectively. With these definitions (which follow
For a spherical polar grid, the coefficients of the metric terms are via "ghost-points"; for free slip boundary conditions (\(\zeta\)-points have been renamed to Z-points), the strain rates are discretized as:

\[
\begin{align*}
\dot{\epsilon}_{11} &= \partial_1 u_1 + k_2 u_2 \\
\Rightarrow (\epsilon_{11})_{i,j}^C &= \frac{u_{i+1,j} - u_{i,j}}{\Delta x_{i,j}^F} + k_{2,i,j} \frac{u_{i+1,j} + u_{i,j}}{2} \\
\dot{\epsilon}_{22} &= \partial_2 u_2 + k_1 u_1 \\
\Rightarrow (\epsilon_{22})_{i,j}^C &= \frac{v_{i,j+1} - v_{i,j}}{\Delta y_{i,j}^F} + k_{1,i,j} \frac{u_{i,j+1} + u_{i,j}}{2} \\
\dot{\epsilon}_{12} &= \dot{\epsilon}_{21} = \frac{1}{2} \left( \partial_1 u_2 + \partial_2 u_1 - k_1 u_2 - k_2 u_1 \right) \\
\Rightarrow (\epsilon_{12})_{i,j}^Z &= \frac{1}{2} \left( \frac{v_{i,j} - v_{i-1,j}}{\Delta x_{i,j}^F} + \frac{u_{i,j} + u_{i,j-1}}{\Delta y_{i,j}^F} \right)
\end{align*}
\]

so that the diagonal terms of the strain rate tensor are naturally defined at C-points and the symmetric off-diagonal term at Z-points. No-slip boundary conditions \((u_{i,j-1} + u_{i,j} = 0)\) and \((v_{i-1,j} + v_{i,j} = 0)\) across boundaries are implemented via "ghost-points"; for free slip boundary conditions \((\epsilon_{12})^Z = 0\) 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 \(\phi\); \(\Delta x_1 = \Delta x = a \cos \phi \Delta \lambda\), and \(\Delta x_2 = \Delta y = a \Delta \phi\). For a general orthogonal curvilinear grid, \(k_1\) and \(k_2\) can be approximated by finite differences of the cell widths:

\[
\begin{align*}
k_{1,i,j}^C &= \frac{1}{\Delta y_{i,j}^F} \left( \frac{\Delta y_{i+1,j}^G - \Delta y_{i,j}^G}{\Delta x_{i,j}^F} \right) \\
k_{2,i,j}^C &= \frac{1}{\Delta y_{i,j}^F} \left( \frac{\Delta x_{i,j+1}^G - \Delta x_{i,j}^G}{\Delta y_{i,j}^F} \right) \\
k_{1,i,j}^Z &= \frac{1}{\Delta x_{i,j}^F} \left( \frac{\Delta y_{i,j}^G - \Delta y_{i,j-1}^G}{\Delta x_{i,j}^F} \right) \\
k_{2,i,j}^Z &= \frac{1}{\Delta y_{i,j}^F} \left( \frac{\Delta x_{i,j}^G - \Delta x_{i,j-1}^G}{\Delta y_{i,j}^F} \right)
\end{align*}
\]

The stress tensor is given by the constitutive viscous-plastic relation \(\sigma_{\alpha\beta} = 2\eta c_{\alpha\beta} + [(\zeta - \eta) \dot{\gamma} - P/2] \delta_{\alpha\beta}\). 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:

\[
(\nabla \sigma)_1 : \frac{1}{A_{i,j}^w} \int_{\text{cell}} (\partial_1 \sigma_{11} + \partial_2 \sigma_{21}) \, dx_1 \, dx_2 \\
= \frac{1}{A_{i,j}^w} \left\{ \int_{x_2}^{x_2 + \Delta x_2} \sigma_{11} \, dx_2 \left|_{x_1 + \Delta x_1}^{x_1 + \Delta x_1} \right. + \int_{x_1}^{x_1 + \Delta x_1} \sigma_{21} \, dx_1 \left|_{x_2 + \Delta x_2}^{x_2 + \Delta x_2} \right. \right\} \\
\approx \frac{1}{A_{i,j}^w} \left\{ \left( \Delta x_2 \sigma_{11} \right) \left|_{x_1}^{x_1 + \Delta x_1} \right. + \Delta x_1 \sigma_{21} \left|_{x_2}^{x_2 + \Delta x_2} \right. \right\} \\
= \frac{1}{A_{i,j}^w} \left\{ \left( \Delta x_2 \sigma_{11} \right)_{i,j}^C - \left( \Delta x_2 \sigma_{11} \right)_{i-1,j}^C \right. \\
+ \left( \Delta x_1 \sigma_{21} \right)_{i,j+1}^Z - \left( \Delta x_1 \sigma_{21} \right)_{i,j}^Z \right\}
\]
Similarly, we have for the \( v \)-equation (\( \alpha = 2 \)):

\[
(\nabla \sigma)_2 : \frac{1}{A_{i,j}} \int_{cell} (\partial_1 \sigma_{12} + \partial_2 \sigma_{22}) \, dx_1 \, dx_2 \\
\approx \frac{1}{A_{i,j}} \left\{ \int_{x_2}^{x_2 + \Delta x_2} \sigma_{12} \, dx_2 \bigg|_{x_1}^{x_1 + \Delta x_1} + \int_{x_1}^{x_1 + \Delta x_1} \sigma_{22} \, dx_1 \bigg|_{x_2}^{x_2 + \Delta x_2} \right\}
\]

\[
\Delta x_{12,c}^2 Z_{i,j} = \frac{\Delta x_{12,c}^2 Z_{i,j}}{\Delta x_{12,c}^2 Z_{i,j-1}}
\]

with

\[
(\Delta x_{12,c}^2 Z_{i,j})_{i,j}^C = \frac{\Delta x_{12,c}^2 Z_{i,j}}{\Delta x_{12,c}^2 Z_{i,j-1}}
\]

\[
\Delta x_{22,c}^F C_{i,j} = \frac{\Delta x_{22,c}^F C_{i,j}}{\Delta x_{22,c}^F C_{i,j-1}}
\]

\[
\Delta x_{12,c}^2 Z_{i,j} = \frac{\Delta x_{12,c}^2 Z_{i,j}}{\Delta x_{12,c}^2 Z_{i,j-1}}
\]
Again, no slip boundary conditions are realized via ghost points and \( u_{i,j-1} + u_{i,j} = 0 \) and \( v_{i-1,j} + v_{i,j} = 0 \) across boundaries. For free slip boundary conditions the lateral stress is set to zeros. In analogy to \((\epsilon_{12})^2 = 0\) on boundaries, we set \( \sigma_{21}^e = 0 \), or equivalently \( \eta_{i,j}^e = 0 \), on boundaries.

**Thermodynamics**

**NOTE: THIS SECTION IS TERRIBLY OUT OF DATE**

In its original formulation the sea ice model uses simple thermodynamics following the appendix of [Sem76]. This formulation does not allow storage of heat, that is, the heat capacity of ice is zero. Upward conductive heat flux is parameterized assuming a linear temperature profile and together with a constant ice conductivity. It is expressed as \( (K/h)(T_w - T_0) \), where \( K \) is the ice conductivity, \( h \) the ice thickness, and \( T_w - T_0 \) the difference between water and ice surface temperatures. This type of model is often referred to as a “zero-layer” model. The surface heat flux is computed in a similar way to that of .

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 \( H_n = \frac{2h}{N} h \) 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 \( \rho \) and \( c_p \) are the density and heat capacity of sea water and \( T_{fr} \) is the local freezing point temperature that is a function of salinity. This flux is not assumed to instantaneously melt or create ice, but a time scale of three days (run-time parameter SEAICE_gamma_t) is used to relax \( T_w \) to the freezing point. The parameterization of lateral and vertical growth of sea ice follows that of [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

\[
\frac{K}{h} \rightarrow \frac{1}{\frac{K_s}{h_s} + \frac{K}{h}}
\]

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:

\[
\frac{\partial X}{\partial t} = -\nabla \cdot (\vec{u} \cdot X) + \Gamma_X + D_X
\]
where $\Gamma_X$ are the thermodynamic source terms and $D_X$ the diffusive terms for quantities $X = (c \cdot h), c, (c \cdot h_s)$.

From the various advection 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  seaice_model (TOP LEVEL ROUTINE)
c |
c  !-- #ifdef SEAICE_CGRID
  |
  |  SEAICE_DYNSOLVER
  | |
  |  |  |-- < compute proxy for geostrophic velocity >
  | |
  |  |  |-- < set up mass per unit area and Coriolis terms >
  | |
  |  |  |-- < dynamic masking of areas with no ice >
  | |
  |  |  #ELSE
  |  |   DYNSOLVER
  |  #ENDIF
  |
  |  |-- if ( useOBCS )
  |  |
  |  |   OBCS_APPLY_UVICE
  |  |
  |  |-- if ( SEAICEadvHeff .OR. SEAICEadvArea .OR. SEAICEadvSnow .OR. SEAICEadvSalt )
  |  |
  |  |   SEAICE_ADVDIFF
  |  |
  |  |   SEAICE_REG_RIDGE
  |
  |  |-- if ( usePW79thermodynamics )
  |
  |  |-- if ( useOBCS )
  |
  |  |   if ( SEAICEadvHeff ) OBCS_APPLY_HEFF
```

178 Chapter 5. Physical Parameterizations - Packages I
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


