MITgcm User Manual

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

MIT Department of EAPS
77 Massachusetts Ave.
Cambridge, MA 02139-4307

January 21, 2018

Contents

Chapter 1

Overview of MITgcm

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

1.1 Introduction

MITgcm has a number of novel aspects:

- it can be used to study both atmospheric and oceanic phenomena; one hydrodynamical kernel is used to drive forward both atmospheric and oceanic models - see fig 1.1
- it has a non-hydrostatic capability and so can be used to study both small-scale and large scale processes - see fig 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 fig 1.3
- tangent linear and adjoint counterparts are automatically maintained along with the forward model, permitting sensitivity and optimization studies.
- the model is developed to perform efficiently on a wide variety of computational platforms.

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

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

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

Marshall, J., A. Adcroft, C. Hill, L. Perelman, and C. Heisey, (1997)
A finite-volume, incompressible Navier Stokes model for studies of the ocean
```

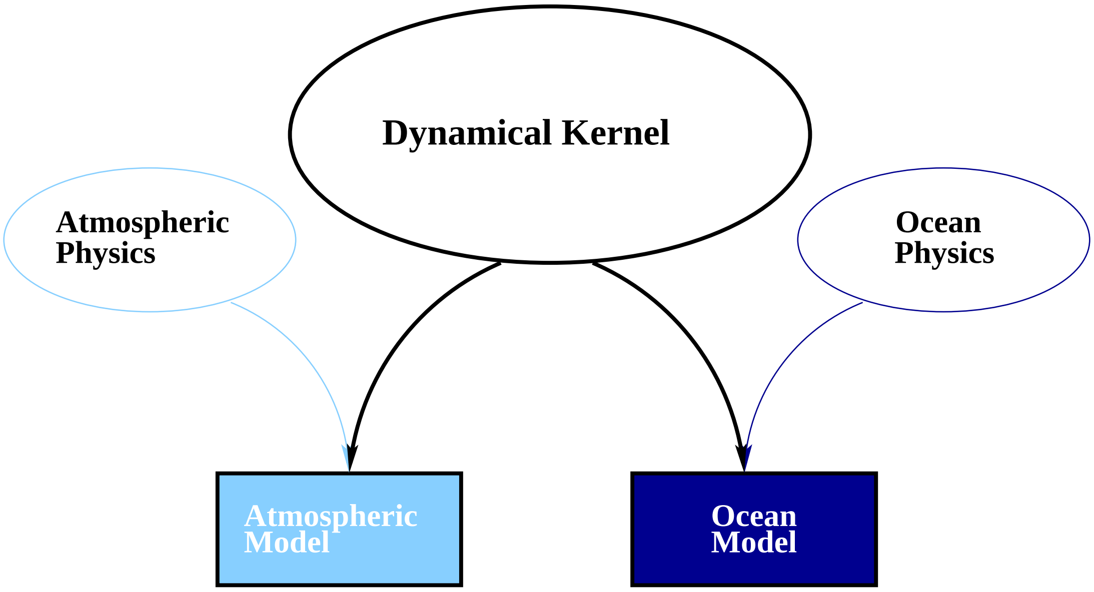


Figure 1.1: MITgcm has a single dynamical kernel that can drive forward either oceanic or atmospheric simulations.

```
on parallel computers,
J. Geophysical Res., 102(C3), 5753-5766.

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

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

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

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

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

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.

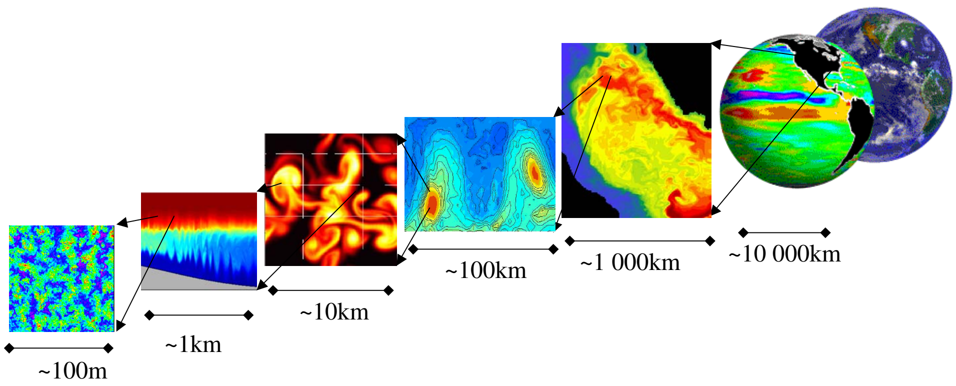


Figure 1.2: MITgcm has non-hydrostatic capabilities, allowing the model to address a wide range of phenomenon - from convection on the left, all the way through to global circulation patterns on the right.

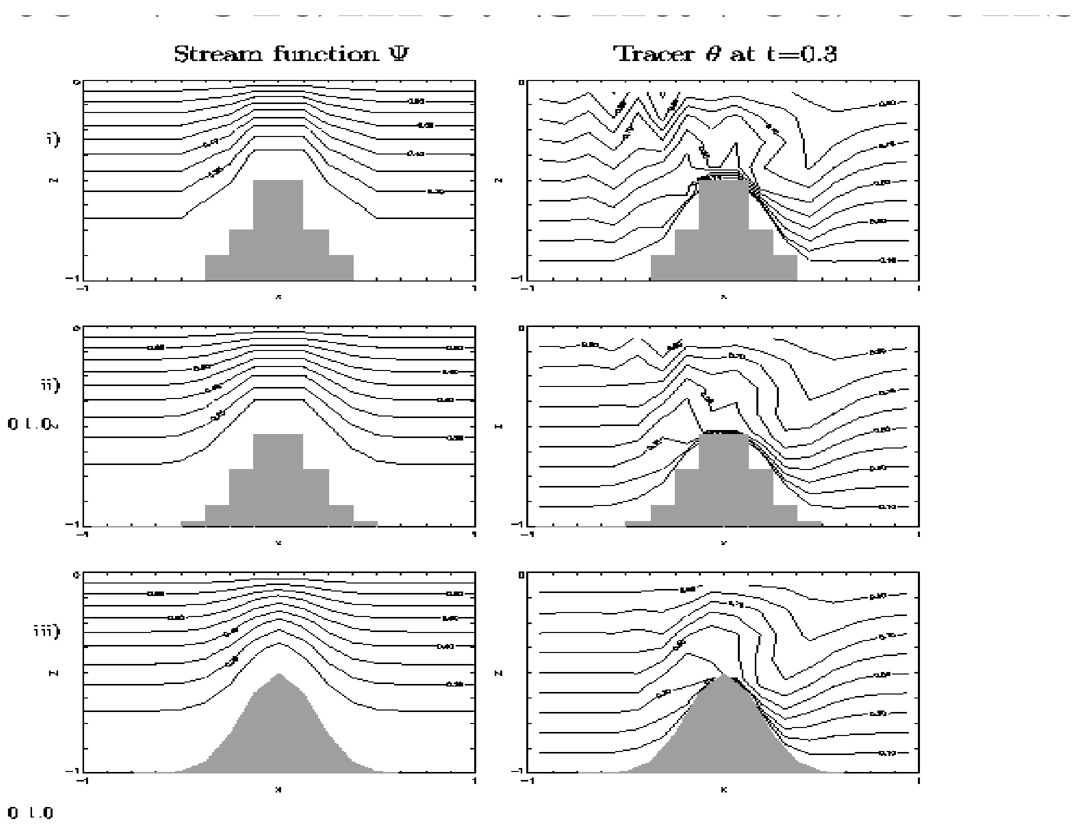


Figure 1.3: Finite volume techniques (bottom panel) are user, permitting a treatment of topography that rivals σ (terrain following) coordinates.

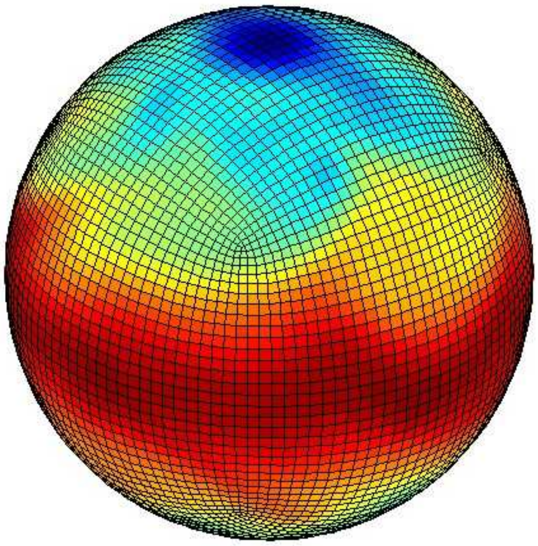


Figure 1.4: Instantaneous plot of the temerature field at 500mb obtained using the atmospheric isomorph of MITgcm

1.2 Illustrations of the model in action

MITgcm has been designed and used to model a wide range of phenomena, from convection on the scale of meters in the ocean to the global pattern of atmospheric winds - see figure 1.2. To give a flavor of the kinds of problems the model has been used to study, we briefly describe some of them here. A more detailed description of the underlying formulation, numerical algorithm and implementation that lie behind these calculations is given later. Indeed many of the illustrative examples shown below can be easily reproduced: simply download the model (the minimum you need is a PC running Linux, together with a FORTRAN 77 compiler) and follow the examples described in detail in the documentation.

1.2.1 Global atmosphere: 'Held-Suarez' benchmark

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

Figure 1.4 shows an instantaneous plot of the $500mb$ 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 designed to test atmospheric hydrodynamical cores - there are no mountains or land-sea contrast.

As described in Adcroft (2001), a 'cubed sphere' is used to discretize the globe permitting a uniform griding and obviated the need to Fourier filter. The 'vector-invariant' form of MITgcm supports any orthogonal curvilinear grid, of which the cubed sphere is just one of many choices.

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

1.2.2 Ocean gyres

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

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

1.2.3 Global ocean circulation

Figure 1.7 (top) 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 $70^{\circ}N$ to $70^{\circ}S$. 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 1.7 (bottom) shows the meridional overturning circulation of the global ocean in Sverdrups.

1.2.4 Convection and mixing over topography

Dense plumes generated by localized cooling on the continental shelf of the ocean may be influenced by rotation when the deformation radius is smaller than the width of the cooling region. Rather than gravity plumes, the mechanism for moving dense fluid down the shelf is then through geostrophic eddies. The simulation shown in the figure 1.8 (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.

1.2.5 Boundary forced internal waves

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

Fig. 1.9 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 nonhydrostatic dynamics.

1.2.6 Parameter sensitivity using the adjoint of MITgcm

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

As one example of application of the MITgcm adjoint, Figure 1.10 maps the gradient $\frac{\partial J}{\partial \mathcal{H}}$where J is the magnitude of the overturning stream-function shown in figure 1.7 at $60^{\circ}N$ and $\mathcal{H}(\lambda, \varphi)$ is the mean, local air-sea heat flux over a 100 year period. We see that J is sensitive to heat fluxes over the Labrador

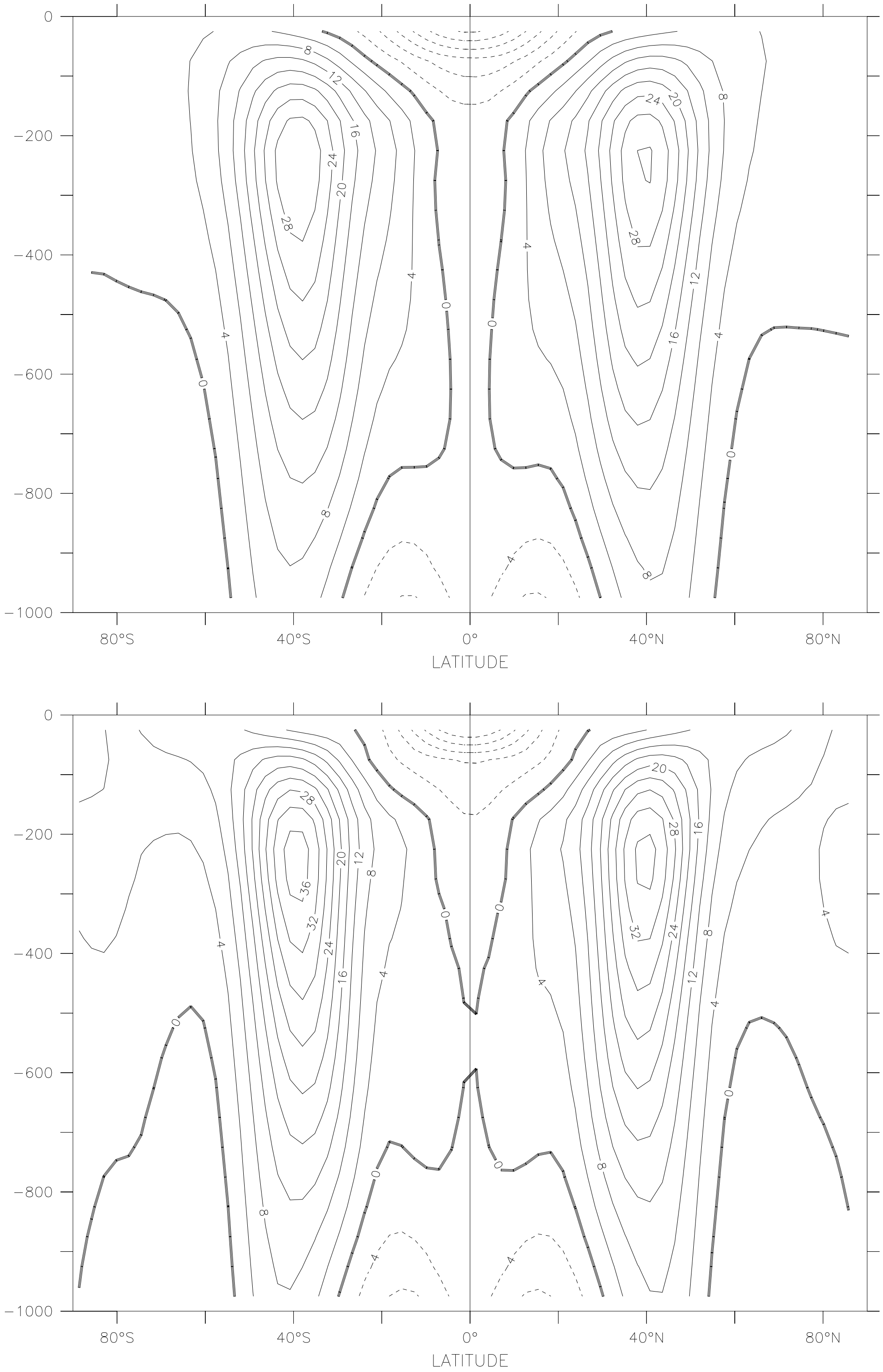


Figure 1.5: Five year mean, zonally averaged zonal flow for latitude-longitude simulation (bottom) and cube-sphere simulation(top) using Held-Suarez forcing. Note the difference in the solutions over the pole - the cubed sphere is superior.

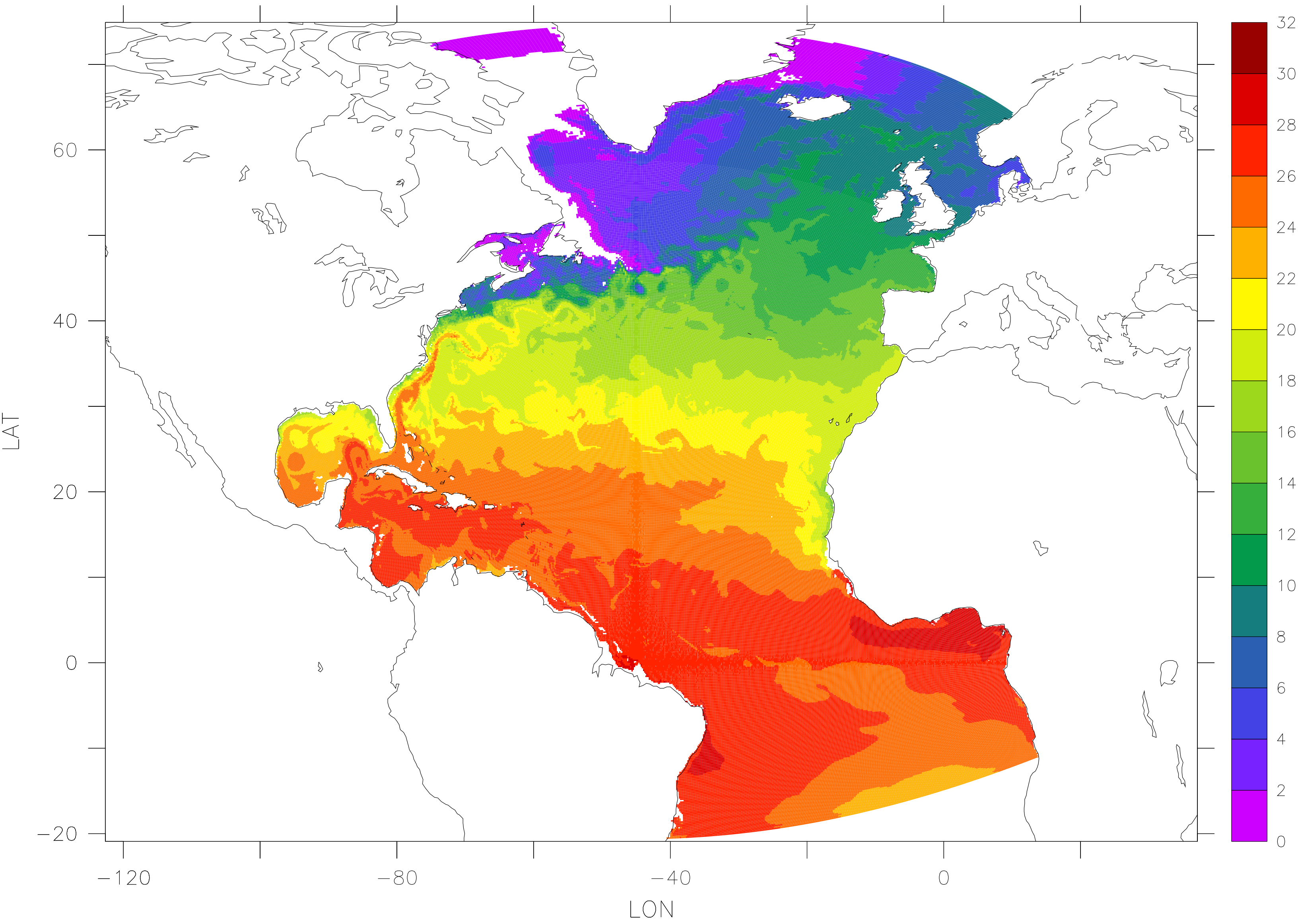


Figure 1.6: Instantaneous temperature map from a $\frac{1}{6}^{\circ}$ simulation of the North Atlantic. The figure shows the temperature in the second layer (37.5m deep).

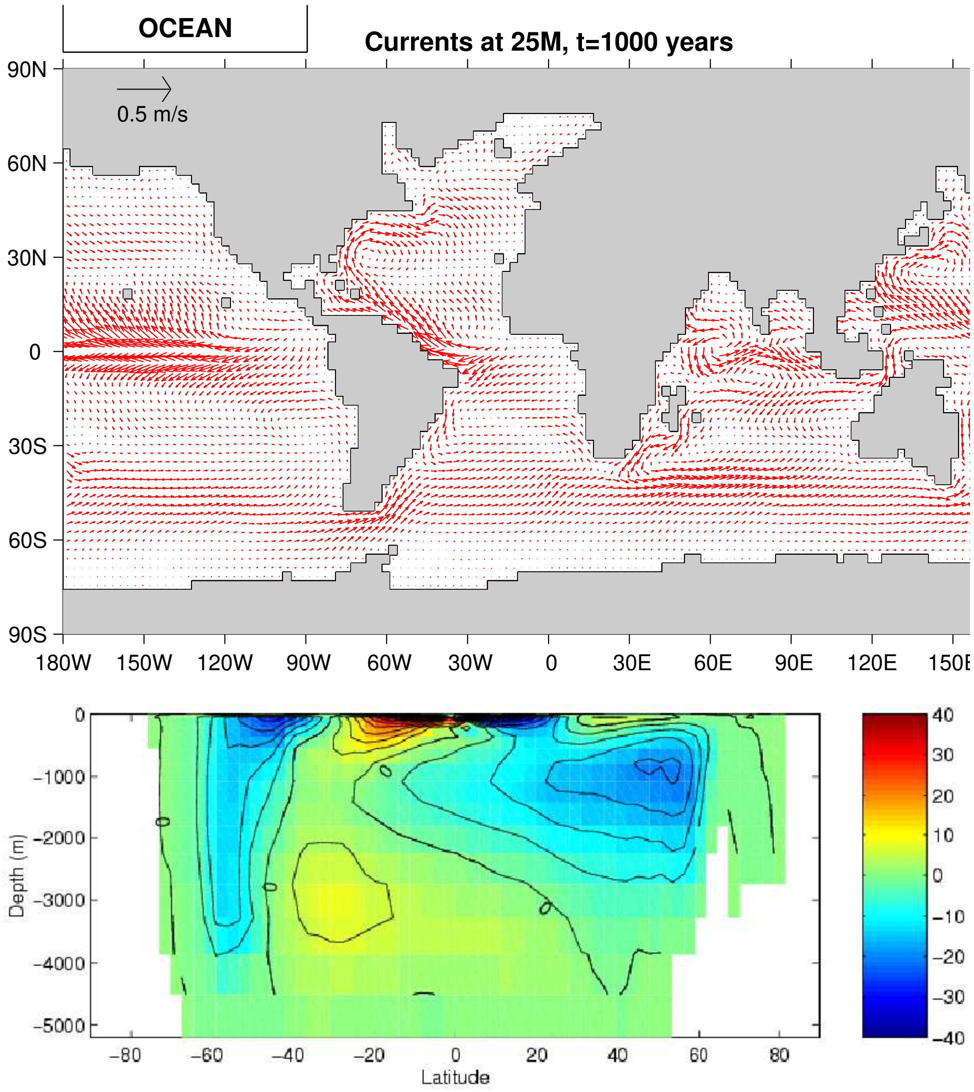


Figure 1.7: Pattern of surface ocean currents (top) and meridional overturning stream function (in Sverdrups) from a global integration of the model at 4° horizontal resolution and with 15 vertical levels.

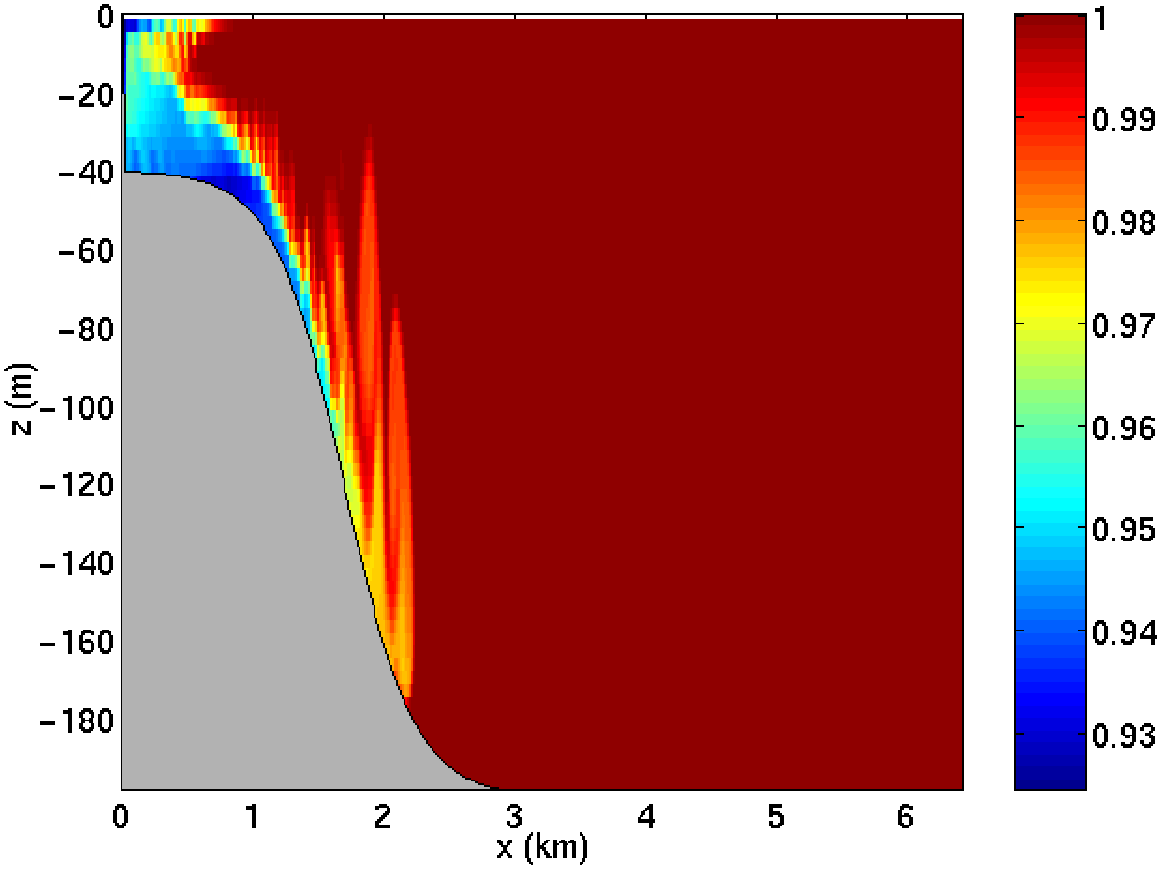


Figure 1.8: MITgcm run in a non-hydrostatic configuration to study convection over a slope.

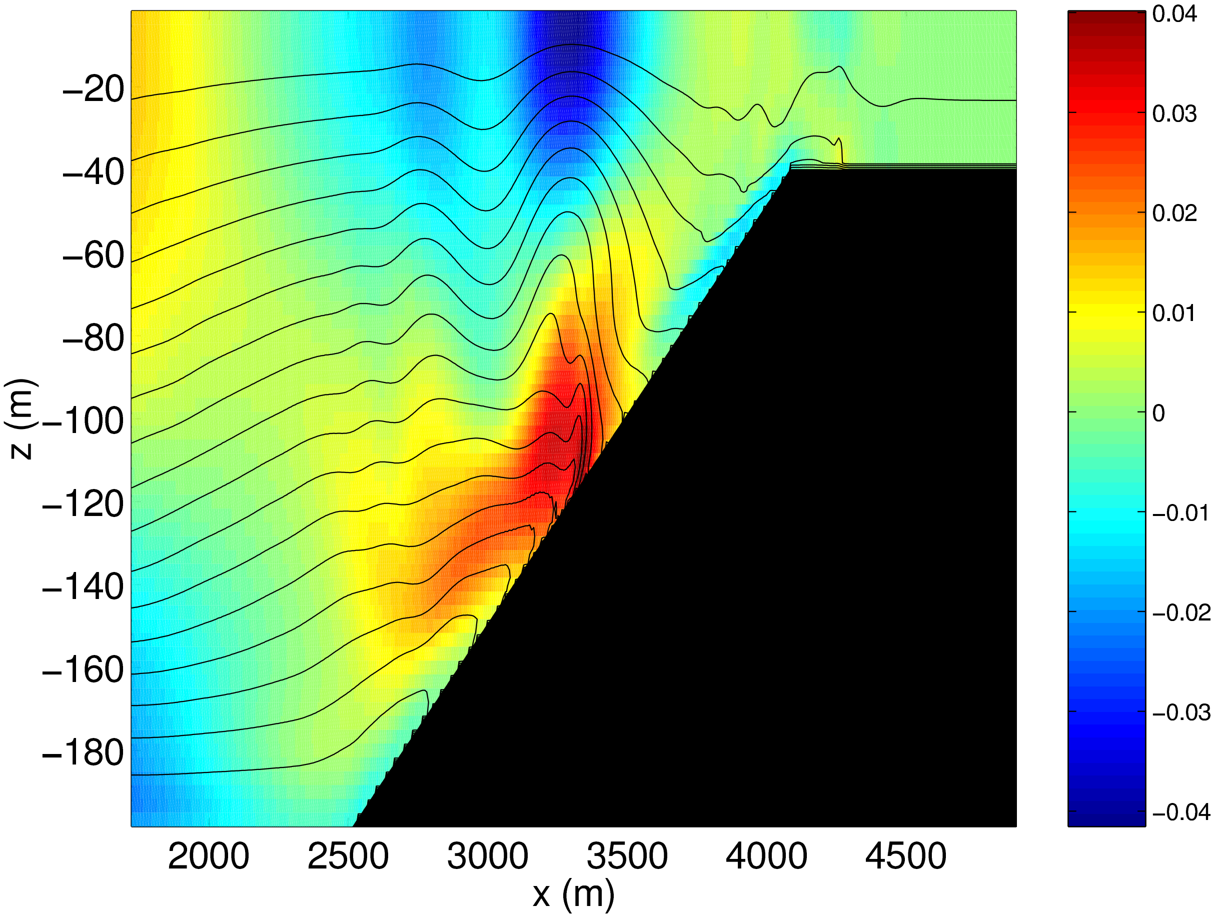


Figure 1.9: Simulation of internal waves forced at an open boundary (on the left) impacting a sloping shelf. The along slope velocity is shown colored, contour lines show density surfaces. The slope is represented with high-fidelity using lopped cells.

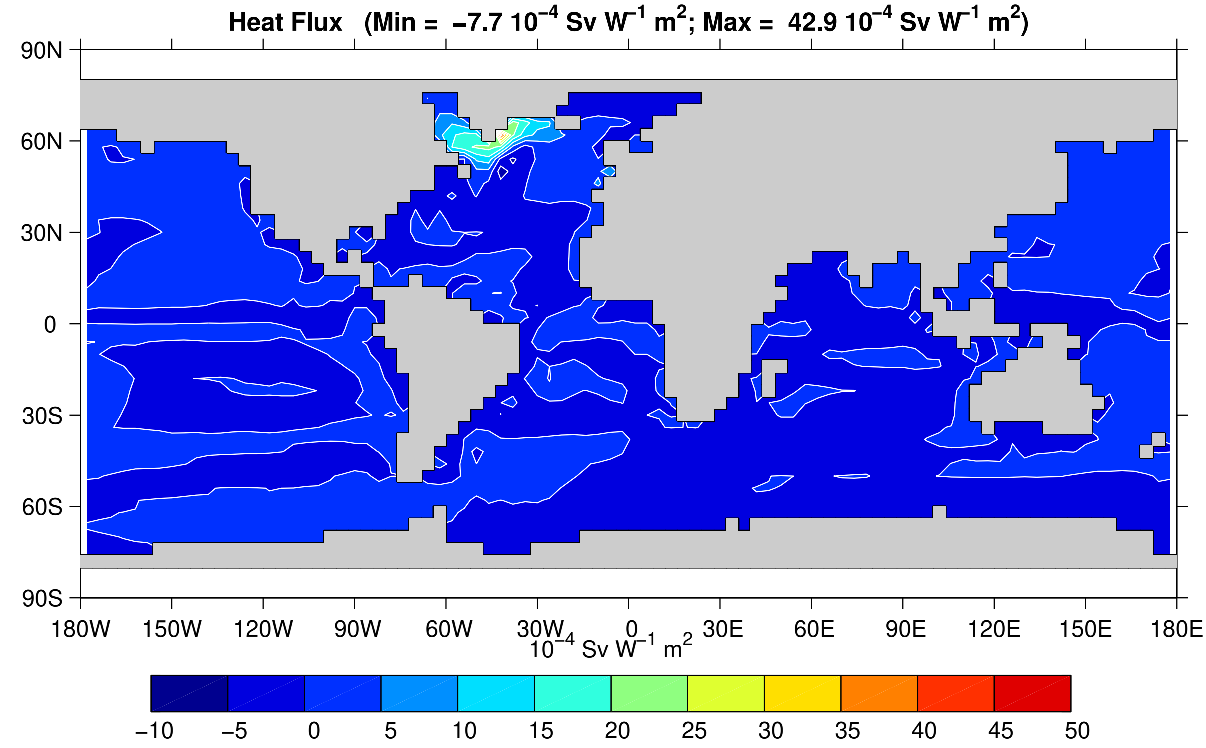


Figure 1.10: Sensitivity of meridional overturning strength to surface heat flux changes. Contours show the magnitude of the response (in Sv $\times 10^{-4}$) that a persistent $+1\mathrm{Wm}^{-2}$ heat flux anomaly at a given gid point would produce.

Sea, one of the important sources of deep water for the thermohaline circulations. This calculation also yields sensitivities to all other model parameters.

1.2.7 Global state estimation of the ocean

An important application of MITgcm is in state estimation of the global ocean circulation. An appropriately defined 'cost function', which measures the departure of the model from observations (both remotely sensed and in-situ) over an interval of time, is minimized by adjusting 'control parameters' such as air-sea fluxes, the wind field, the initial conditions etc. Figure 1.11 shows 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.

1.2.8 Ocean biogeochemical cycles

MITgcm is being used to study global biogeochemical cycles in the ocean. For example one can study the effects of interannual changes in meteorological forcing and upper ocean circulation on the fluxes of carbon dioxide and oxygen between the ocean and atmosphere. Figure 1.12 shows the annual air-sea flux of oxygen and its relation to density outcrops in the southern oceans from a single year of a global, interannually varying simulation. The simulation is run at $1^\circ \times 1^\circ$ resolution telescoping to $\frac{1}{3}^\circ \times \frac{1}{3}^\circ$ in the tropics (not shown).

1.2.9 Simulations of laboratory experiments

Figure 1.13 shows MITgcm being used to simulate a laboratory experiment inquiring into the dynamics of the Antarctic Circumpolar Current (ACC). An initially homogeneous tank of water ($1m$ in diameter) is driven from its free surface by a rotating heated disk. The combined action of mechanical and thermal

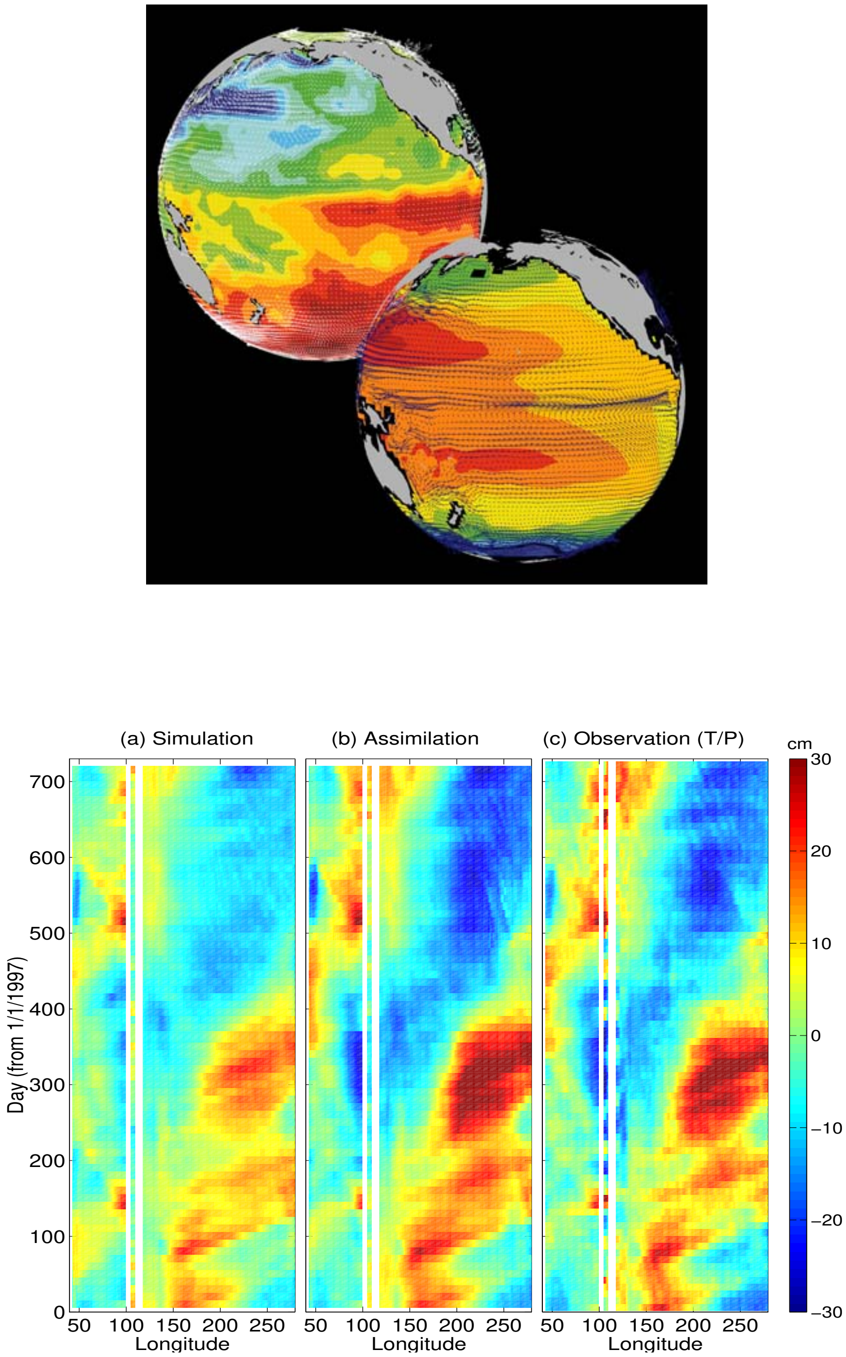


Figure 1.11: Top panel shows circulation patterns from a multi-year, global circulation simulation constrained by Topex altimeter data and WOCE cruise observations. Bottom panel shows the equatorial sea-surface height in unconstrained (left), constrained (middle) simulations and in observations. This output is from a higher resolution, shorter duration experiment with equatorially enhanced grid spacing.

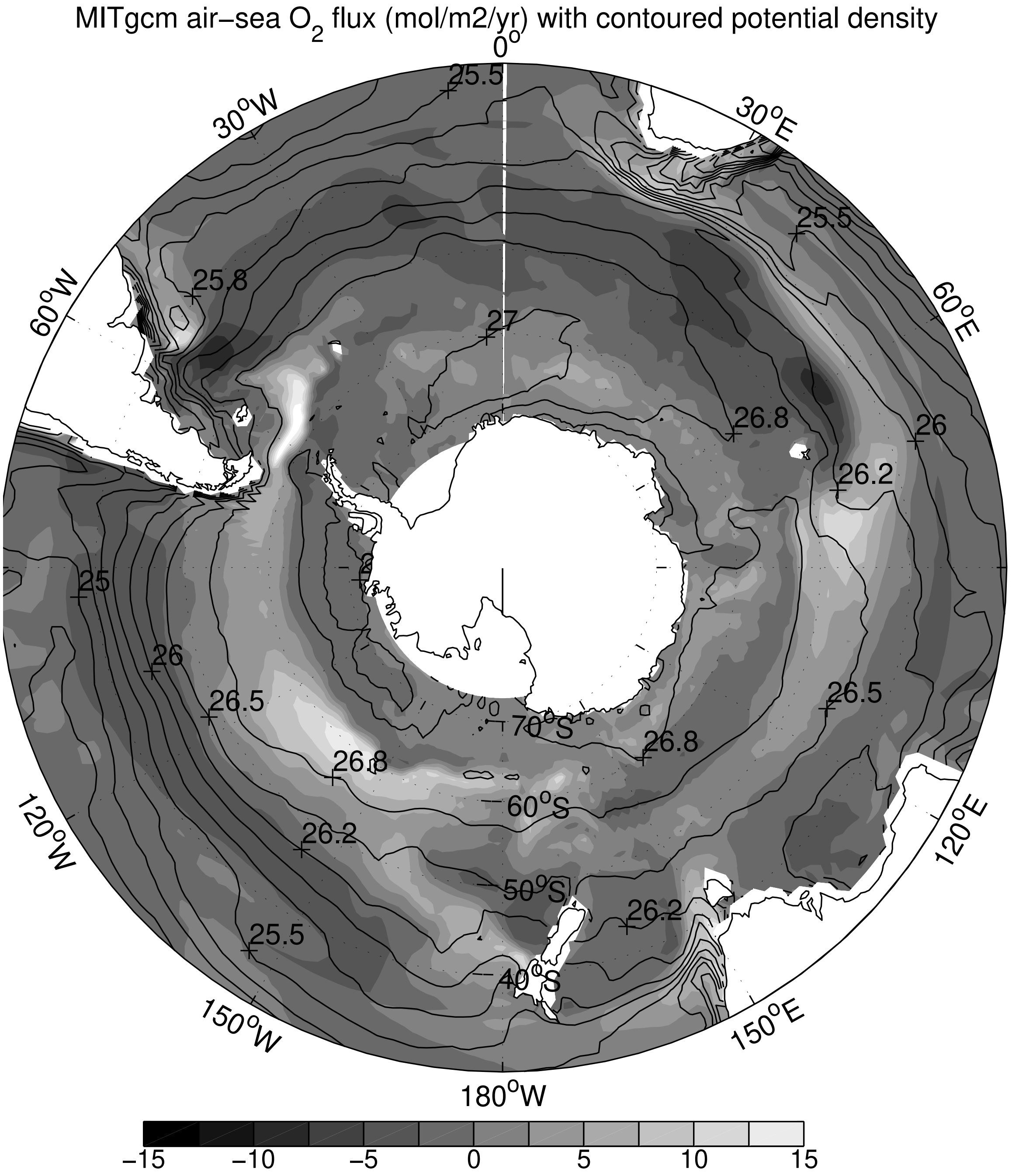


Figure 1.12: Annual air-sea flux of oxygen (shaded) plotted along with potential density outcrops of the surface of the southern ocean from a global $1^\circ \times 1^\circ$ integation with a telescoping grid (to $\frac{1}{3}^\circ$) at the equator.

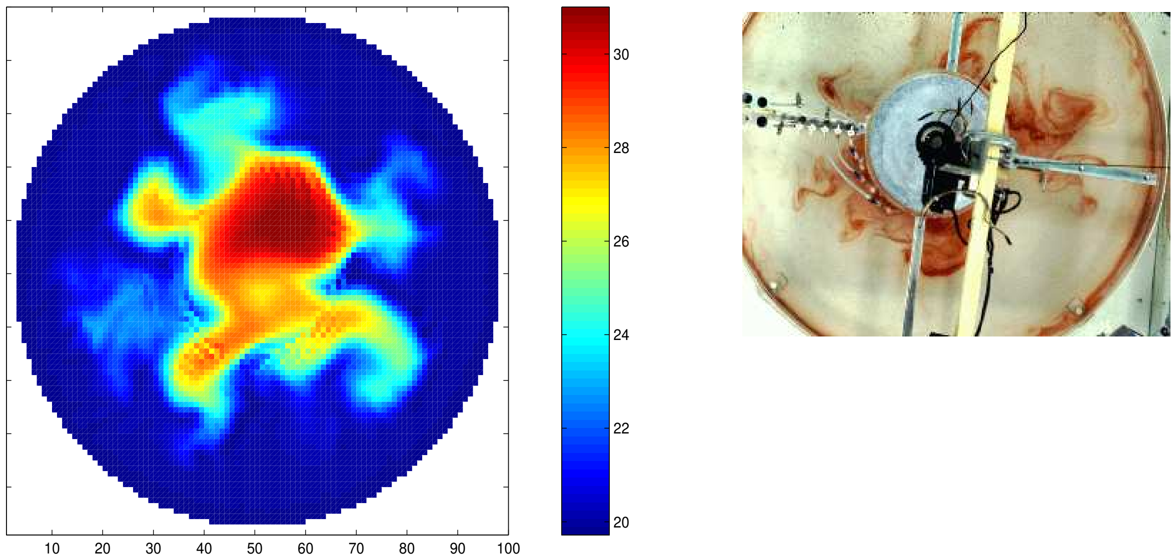


Figure 1.13: A numerical simulation (left) of a 1m diameter laboratory experiment (right) using MITgcm.

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.

1.3 Continuous equations in ‘r’ coordinates

To render atmosphere and ocean models from one dynamical core we exploit ‘isomorphisms’ between equation sets that govern the evolution of the respective fluids - see figure 1.14. 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 1.14) and height, z, if we are modeling the ocean (left hand side of figure 1.14).

The state of the fluid at any time is characterized by the distribution of velocity $\vec{\mathbf{v}}$, active tracers θ and S, a ‘geopotential’ ϕ and density $\rho = \rho(\theta, S, p)$ which may depend on θ, S, and p. The equations that govern the evolution of these fields, obtained by applying the laws of classical mechanics and thermodynamics to a Boussinesq, Navier-Stokes fluid are, written in terms of a generic vertical coordinate, r, so that the appropriate kinematic boundary conditions can be applied isomorphically see figure 1.15.

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

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

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

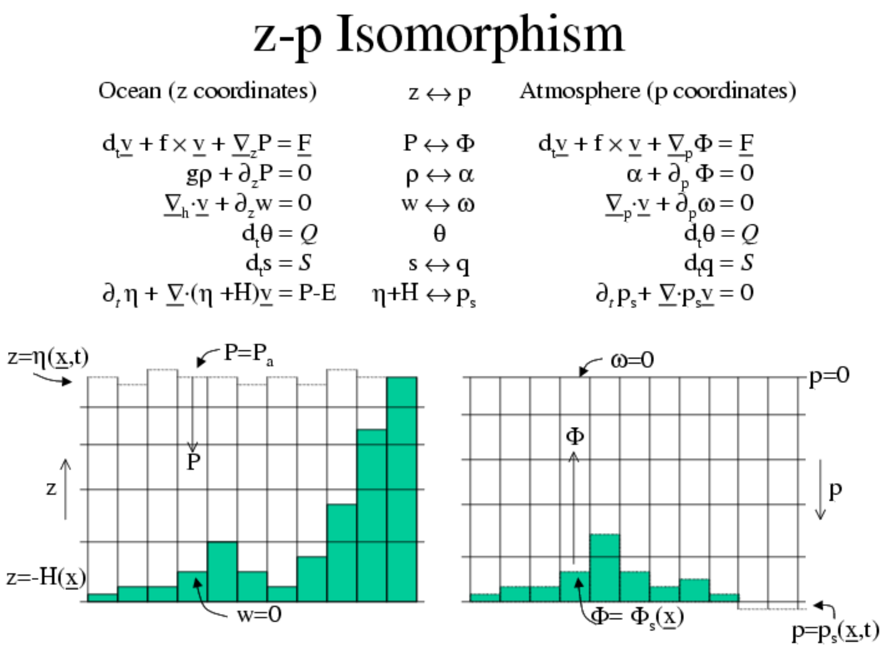


Figure 1.14: Isomorphic equation sets used for atmosphere (right) and ocean (left).

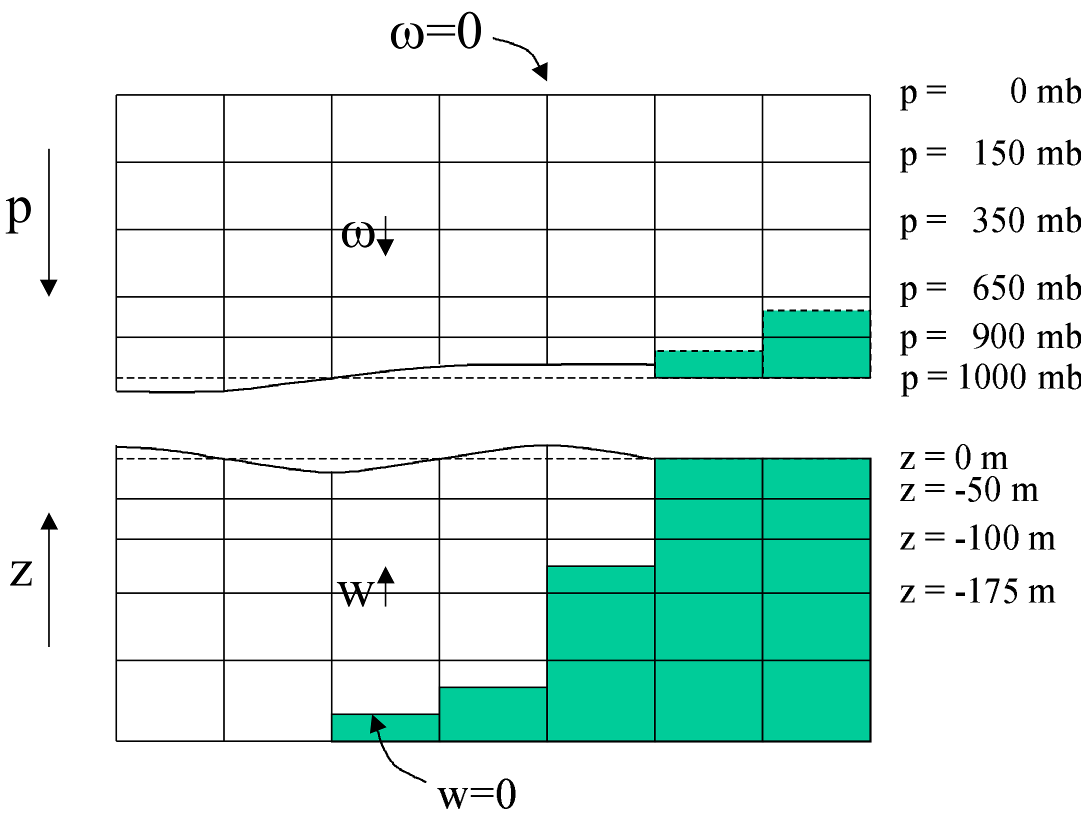


Figure 1.15: Vertical coordinates and kinematic boundary conditions for atmosphere (top) and ocean (bottom).

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

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

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

Here:

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

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

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

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

$$t \text{ is time}$$

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

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

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

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

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

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

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

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

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

The $\mathcal{F}'s$ and $\mathcal{Q}'s$ are provided by 'physics' and forcing packages for atmosphere and ocean. These are described in later chapters.

1.3.1 Kinematic Boundary conditions

1.3.1.1 vertical

at fixed and moving r surfaces we set (see figure 1.15):

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

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

Here

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

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

1.3.1.2 horizontal

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

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

1.3.2 Atmosphere

In the atmosphere, (see figure 1.15), we interpret:

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

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

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

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

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

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

where

$$T \text{ is absolute temperature}$$
$$p \text{ is the pressure}$$

$$z \text{ is the height of the pressure surface}$$
$$g \text{ is the acceleration due to gravity}$$

In the above the ideal gas law, $p = \rho RT$, has been expressed in terms of the Exner function $\Pi(p)$ given by (see Appendix Atmosphere)

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

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

At the top of the atmosphere (which is 'fixed' in our r coordinate):

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

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

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

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

The boundary conditions at top and bottom are given by:

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

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

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

1.3.3 Ocean

In the ocean we interpret:

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

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

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

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

where ρ_c is a fixed reference density of water and g is the acceleration due to gravity.

In the above

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

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

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

Boundary conditions are:

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

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

where η is the elevation of the free surface.

Then equations (1.1-1.6) yield a consistent set of oceanic equations which, for convenience, are written out in z coordinates in Appendix Ocean - see eqs(1.99) to (1.104).

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

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

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

and write eq(1.1) in the form:

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

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

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

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

The $\left(\vec{\mathbf{G}}_{\vec{v}},G_{\dot{r}}\right)$ in eq(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 for a full discussion:

$$\left.\begin{array}{l} G_u=-\vec{\mathbf{v}}.\nabla u \\ -\left\{\underline{\frac{u\dot{r}}{r}}-\frac{uv\tan\varphi}{r}\right\} \\ -\left\{-2\Omega v\sin\varphi+\underline{2\Omega\dot{r}\cos\varphi}\right\} \\ +\mathcal{F}_u \end{array}\right\}\left\{\begin{array}{l} \textit{advection} \\ \textit{metric} \\ \textit{Coriolis} \\ \textit{Forcing/Dissipation} \end{array}\right. \tag{1.29}$$

$$\left.\begin{array}{l} G_v=-\vec{\mathbf{v}}.\nabla v \\ -\left\{\underline{\frac{v\dot{r}}{r}}-\frac{u^2\tan\varphi}{r}\right\} \\ -\left\{-2\Omega u\sin\varphi\right\} \\ +\mathcal{F}_v \end{array}\right\}\left\{\begin{array}{l} \textit{advection} \\ \textit{metric} \\ \textit{Coriolis} \\ \textit{Forcing/Dissipation} \end{array}\right. \tag{1.30}$$

$$\left.\begin{array}{l} G_{\dot{r}}=-\underline{\underline{\vec{\mathbf{v}}.\nabla\dot{r}}} \\ +\left\{\underline{\frac{u^2+v^2}{r}}\right\} \\ +\underline{2\Omega u\cos\varphi} \\ \underline{\underline{\mathcal{F}_{\dot{r}}}} \end{array}\right\}\left\{\begin{array}{l} \textit{advection} \\ \textit{metric} \\ \textit{Coriolis} \\ \textit{Forcing/Dissipation} \end{array}\right. \tag{1.31}$$

In the above 'r' is the distance from the center of the earth and 'φ ' is latitude.

Grad and div operators in spherical coordinates are defined in appendix OPERATORS.

1.3.4.1 Shallow atmosphere approximation

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

1.3.4.2 Hydrostatic and quasi-hydrostatic forms

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

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

In the 'quasi-hydrostatic' equations **(QH)** strict balance between gravity and vertical pressure gradients is not imposed. The $2\Omega u\cos\varphi$ Coriolis term are not neglected and are balanced by a non-hydrostatic contribution to the pressure field: only the terms underlined twice in Eqs. (1.29$\rightarrow$ 1.31) are set to zero and, simultaneously, the shallow atmosphere 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.

[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.

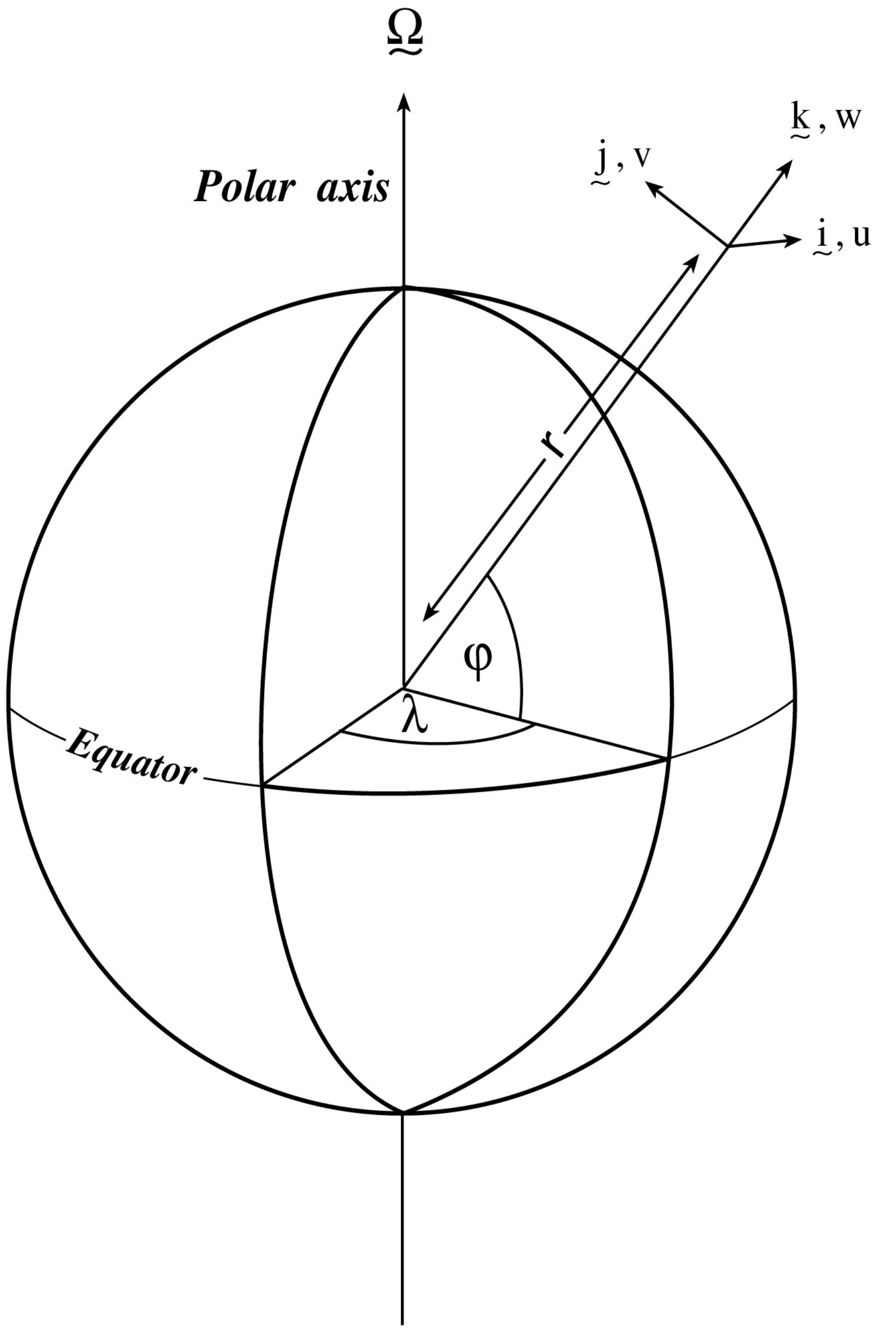


Figure 1.16: Spherical polar coordinates: longitude λ, latitude φ and r the distance from the center.

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. As in **HPE** only a 2-d elliptic problem need be solved.

1.3.4.3 Non-hydrostatic and quasi-nonhydrostatic forms

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

Non-hydrostatic Ocean In the non-hydrostatic ocean model all terms in equations Eqs.(1.29 → 1.31) are retained. A three dimensional elliptic equation must be solved subject to Neumann boundary conditions (see below). It is important to note that use of the full **NH** does not admit any new 'fast' waves in to the system - the incompressible condition eq(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; Marshall et.al. 1997a.

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

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

where p_{hy} is the hydrostatic pressure.

1.3.4.4 Summary of equation sets supported by model

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

Hydrostatic and quasi-hydrostatic The hydrostatic set is written out in $p-$coordinates in appendix Atmosphere - see eq(1.59).

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(1.99) to (1.104).

1.3.5 Solution strategy

The method of solution employed in the **HPE**, **QH** and **NH** models is summarized in Figure 1.17. 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) resulting in a non-hydrostatic algorithm that, in the hydrostatic limit, is as computationally economic as the **HPEs**.

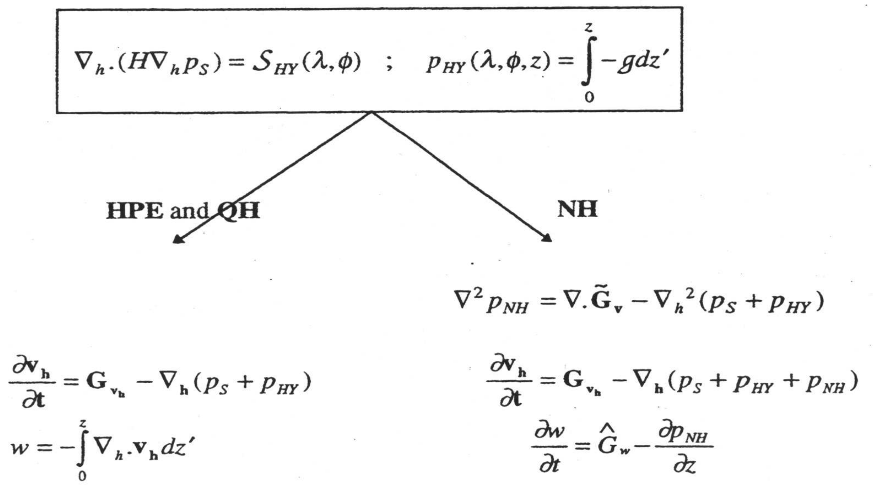


Figure 1.17: Basic solution strategy in MITgcm. **HPE** and **QH** forms diagnose the vertical velocity, in **NH** a prognostic equation for the vertical velocity is integrated.

1.3.6 Finding the pressure field

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

1.3.6.1 Hydrostatic pressure

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

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

and so

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

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

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

1.3.6.2 Surface pressure

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

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

Thus:

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

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

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

where we have incorporated a source term.

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

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

where b_s is the buoyancy at the surface.

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

1.3.6.3 Non-hydrostatic pressure

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

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

For a given rhs this 3-d elliptic equation must be inverted for ϕ_{nh} subject to appropriate choice of boundary conditions. This method is usually called *The Pressure Method* [Harlow and Welch, 1965; Williams, 1969; Potter, 1976]. In the hydrostatic primitive equations case (**HPE**), the 3-d problem does not need to be solved.

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

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

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

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

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

where

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

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

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

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

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

If the flow is 'close' to hydrostatic balance then the 3-d inversion converges rapidly because ϕ_{nh} is then only a small correction to the hydrostatic pressure field (see the discussion in Marshall et al, a,b).

The solution ϕ_{nh} to (1.37) and (1.39) does not vanish at $r = R_{moving}$, and so refines the pressure there.

1.3.7 Forcing/dissipation

1.3.7.1 Forcing

The forcing terms $\mathcal{F}$ on the rhs of the equations are provided by 'physics packages' and forcing packages. These are described later on.

1.3.7.2 Dissipation

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

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

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

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

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

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

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

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

1.3.8 Vector invariant form

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

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

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

1.3.9 Adjoint

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

1.4 Appendix ATMOSPHERE

1.4.1 Hydrostatic Primitive Equations for the Atmosphere in pressure coordinates

The hydrostatic primitive equations (HPEs) in p-coordinates are:

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

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

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

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

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

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

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

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

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

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

Potential temperature is defined:

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

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

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

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

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

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

The heat equation is obtained by noting that

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

and on substituting into (1.50) gives:

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

which is in conservative form.

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

1.4.1.1 Boundary conditions

The upper and lower boundary conditions are :

$$\text{at the top: } p = 0 \qquad , \omega = \frac{Dp}{Dt} = 0 \tag{1.54}$$
$$\text{at the surface: } p = p_s \qquad , \phi = \phi_{topo} = g\ Z_{topo} \tag{1.55}$$

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

1.4.1.2 Splitting the geo-potential

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

$$\theta = \theta_o + \theta' \tag{1.56}$$
$$\alpha = \alpha_o + \alpha' \tag{1.57}$$
$$\phi = \phi_o + \phi' \tag{1.58}$$

The reference state (indicated by subscript "0") 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:

$$\begin{aligned}
\theta_o(p) &= f^n(p) \\
\alpha_o(p) &= \Pi_p \theta_o \\
\phi_o(p) &= \phi_{topo} - \int_{p_0}^{p} \alpha_o dp
\end{aligned}$$

The final form of the HPE's in p coordinates is then:

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

1.5 Appendix OCEAN

1.5.1 Equations of motion for the ocean

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

1.5.1.1 Compressible z-coordinate equations

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

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

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

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

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

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

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

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

1.5.1.2 'Anelastic' z-coordinate equations

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

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

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

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

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

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

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

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

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

$$\frac{1}{\rho_o}\frac{\partial \rho_o}{\partial z} = \frac{-g}{c_s^2} \tag{1.86}$$

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

$$\frac{D\vec{\mathbf{v}}_h}{Dt} + f\hat{\mathbf{k}} \times \vec{\mathbf{v}}_h + \frac{1}{\rho_o}\nabla_z p = \vec{\mathcal{F}} \tag{1.87}$$

$$\epsilon_{nh}\frac{Dw}{Dt} + \frac{g\rho}{\rho_o} + \frac{1}{\rho_o}\frac{\partial p}{\partial z} = \epsilon_{nh}\mathcal{F}_w \tag{1.88}$$

$$\nabla_z \cdot \vec{\mathbf{v}}_h + \frac{1}{\rho_o}\frac{\partial(\rho_o w)}{\partial z} = 0 \tag{1.89}$$

$$\rho = \rho(\theta, S, p_o(z)) \tag{1.90}$$

$$\frac{D\theta}{Dt} = \mathcal{Q}_\theta \tag{1.91}$$

$$\frac{DS}{Dt} = \mathcal{Q}_s \tag{1.92}$$

1.5.1.3 Incompressible z-coordinate equations

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

$$\frac{D\vec{\mathbf{v}}_h}{Dt} + f\hat{\mathbf{k}} \times \vec{\mathbf{v}}_h + \frac{1}{\rho_c}\nabla_z p = \vec{\mathcal{F}} \tag{1.93}$$

$$\epsilon_{nh}\frac{Dw}{Dt} + \frac{g\rho}{\rho_c} + \frac{1}{\rho_c}\frac{\partial p}{\partial z} = \epsilon_{nh}\mathcal{F}_w \tag{1.94}$$

$$\nabla_z \cdot \vec{\mathbf{v}}_h + \frac{\partial w}{\partial z} = 0 \tag{1.95}$$

$$\rho = \rho(\theta, S) \tag{1.96}$$

$$\frac{D\theta}{Dt} = \mathcal{Q}_\theta \tag{1.97}$$

$$\frac{DS}{Dt} = \mathcal{Q}_s \tag{1.98}$$

where ρ_c is a constant reference density of water.

1.5.1.4 Compressible non-divergent equations

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

$$\rho = \rho_o + \rho'$$

We then assert that variations with depth of ρ_o are unimportant while the compressible effects in ρ' are:

$$\rho_o = \rho_c$$

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

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

$$\frac{D\vec{\mathbf{v}}_h}{Dt} + f\hat{\mathbf{k}} \times \vec{\mathbf{v}}_h + \frac{1}{\rho_c}\nabla_z p' = \vec{\mathcal{F}} \tag{1.99}$$

$$\epsilon_{nh}\frac{Dw}{Dt} + \frac{g\rho'}{\rho_c} + \frac{1}{\rho_c}\frac{\partial p'}{\partial z} = \epsilon_{nh}\mathcal{F}_w \tag{1.100}$$

$$\nabla_z \cdot \vec{\mathbf{v}}_h + \frac{\partial w}{\partial z} = 0 \tag{1.101}$$

$$\rho' = \rho(\theta, S, p_o(z)) - \rho_c \tag{1.102}$$

$$\frac{D\theta}{Dt} = \mathcal{Q}_\theta \tag{1.103}$$

$$\frac{DS}{Dt} = \mathcal{Q}_s \tag{1.104}$$

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

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

1.6 Appendix:OPERATORS

1.6.1 Coordinate systems

1.6.1.1 Spherical coordinates

In spherical coordinates, the velocity components in the zonal, meridional and vertical direction respectively, are given by (see Fig.2) :

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

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

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

Here φ is the latitude, λ the longitude, r the radial distance of the particle from the center of the earth, Ω is the angular speed of rotation of the Earth and D/Dt is the total derivative.

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

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

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

Chapter 2

Discretization and Algorithm

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

2.1 Notation

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

The notations we use to describe the discrete formulation of the model are summarized hereafter:
general notation:
$\Delta x, \Delta y, \Delta r$ grid spacing in X,Y,R directions.
A_c, A_w, A_s, A_ζ : horizontal area of a grid cell surrounding θ, u, v, ζ point.
$\mathcal{V}_u, \mathcal{V}_v, \mathcal{V}_w, \mathcal{V}_\theta$: Volume of the grid box surrounding u, v, w, θ point;
i, j, k : current index relative to X,Y,R directions;
basic operator:
δ_i : $\delta_i \Phi = \Phi_{i+1/2} - \Phi_{i-1/2}$
$\overline{}^i$: $\overline{\Phi}^i = (\Phi_{i+1/2} + \Phi_{i-1/2})/2$
δ_x : $\delta_x \Phi = \frac{1}{\Delta x} \delta_i \Phi$

$\overline{\nabla}$ = horizontal gradient operator : $\overline{\nabla} \Phi = \{\delta_x \Phi, \delta_y \Phi\}$
$\overline{\nabla} \cdot$ = horizontal divergence operator : $\overline{\nabla} \cdot \vec{\mathrm{f}} = \frac{1}{\mathcal{A}} \{\delta_i \Delta y \, \mathrm{f}_x + \delta_j \Delta x \, \mathrm{f}_y\}$
$\overline{\nabla}^2$ = horizontal Laplacian operator : $\overline{\nabla}^2 \Phi = \overline{\nabla} \cdot \overline{\nabla} \Phi$

2.2 Time-stepping

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

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

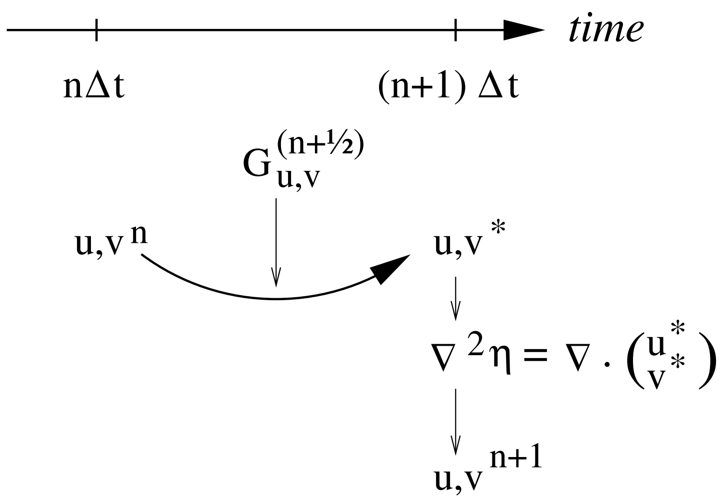


Figure 2.1: A schematic of the evolution in time of the pressure method algorithm. A prediction for the flow variables at time level $n+1$ is made based only on the explicit terms, $G^{(n+^1/_2)}$, and denoted u^*, v^*. Next, a pressure field is found such that u^{n+1}, v^{n+1} will be non-divergent. Conceptually, the $*$ quantities exist at time level $n+1$ but they are intermediate and only temporary.

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

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

2.3 Pressure method with rigid-lid

The horizontal momentum and continuity equations for the ocean (1.99 and 1.101), or for the atmosphere (1.45 and 1.47), can be summarized by:

$$\partial_t u + g\partial_x \eta = G_u \tag{2.1}$$
$$\partial_t v + g\partial_y \eta = G_v \tag{2.2}$$
$$\partial_x u + \partial_y v + \partial_z w = 0 \tag{2.3}$$

where we are adopting the oceanic notation for brevity. All terms in the momentum equations, except for surface pressure gradient, are encapsulated in the G vector. The continuity equation, when integrated

```
FORWARD_STEP
    DYNAMICS
        TIMESTEP                                 u*,v* (2.8,2.9)
    SOLVE_FOR_PRESSURE
        CALC_DIV_GHAT                          Hû*,Hv̂* (2.10)
        CG2D                                   η^{n+1} (2.10)
    MOMENTUM_CORRECTION_STEP
        CALC_GRAD_PHI_SURF                         ∇η^{n+1}
        CORRECTION_STEP               u^{n+1},v^{n+1} (2.11,2.12)
```

Figure 2.2: Calling tree for the pressure method algorithm (FORWARD_STEP)

over the fluid depth, H, and with the rigid-lid/no normal flow boundary conditions applied, becomes:

$$\partial_x H\widehat{u} + \partial_y H\widehat{v} = 0 \tag{2.4}$$

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

$$u^{n+1} + \Delta t g \partial_x \eta^{n+1} = u^n + \Delta t G_u^{(n+1/2)} \tag{2.5}$$
$$v^{n+1} + \Delta t g \partial_y \eta^{n+1} = v^n + \Delta t G_v^{(n+1/2)} \tag{2.6}$$
$$\partial_x H\widehat{u^{n+1}} + \partial_y H\widehat{v^{n+1}} = 0 \tag{2.7}$$

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

Substituting the two momentum equations into the depth integrated continuity equation eliminates u^{n+1} and v^{n+1} yielding an elliptic equation for η^{n+1}. Equations 2.5, 2.6 and 2.7 can then be re-arranged as follows:

$$u^* = u^n + \Delta t G_u^{(n+1/2)} \tag{2.8}$$
$$v^* = v^n + \Delta t G_v^{(n+1/2)} \tag{2.9}$$
$$\partial_x \Delta t g H \partial_x \eta^{n+1} + \partial_y \Delta t g H \partial_y \eta^{n+1} = \partial_x H\widehat{u^*} + \partial_y H\widehat{v^*} \tag{2.10}$$
$$u^{n+1} = u^* - \Delta t g \partial_x \eta^{n+1} \tag{2.11}$$
$$v^{n+1} = v^* - \Delta t g \partial_y \eta^{n+1} \tag{2.12}$$

Equations 2.8 to 2.12, solved sequentially, represent the pressure method algorithm used in the model. The essence of the pressure method lies in the fact that any explicit prediction for the flow would lead to a divergence flow field so a pressure field must be found that keeps the flow non-divergent over each step of the integration. The particular location in time of the pressure field is somewhat ambiguous; in Fig. 2.1 we depicted as co-located with the future flow field (time level $n+1$) but it could equally have been drawn as staggered in time with the flow.

The correspondence to the code is as follows:

- the prognostic phase, equations 2.8 and 2.9, stepping forward u^n and v^n to u^* and v^* is coded in `TIMESTEP()`
- the vertical integration, $H\widehat{u^*}$ and $H\widehat{v^*}$, divergence and inversion of the elliptic operator in equation 2.10 is coded in `SOLVE_FOR_PRESSURE()`
- finally, the new flow field at time level $n+1$ given by equations 2.11 and 2.12 is calculated in `CORRECTION_STEP()`.

The calling tree for these routines is given in Fig. 2.2.

In general, the horizontal momentum time-stepping can contain some terms that are treated implicitly in time, such as the vertical viscosity when using the backward time-stepping scheme (implicitViscosity =.*TRUE*.). The method used to solve those implicit terms is provided in section 2.6, and modifies equations 2.5 and 2.6 to give:

$$u^{n+1} - \Delta t \partial_z A_v \partial_z u^{n+1} + \Delta t g \partial_x \eta^{n+1} = u^n + \Delta t G_u^{(n+1/2)} \quad (2.13)$$

$$v^{n+1} - \Delta t \partial_z A_v \partial_z v^{n+1} + \Delta t g \partial_y \eta^{n+1} = v^n + \Delta t G_v^{(n+1/2)} \quad (2.14)$$

2.4 Pressure method with implicit linear free-surface

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

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

$$\partial_t \eta + \partial_x H \widehat{u} + \partial_y H \widehat{v} = P - E + R \quad (2.15)$$

which differs from the depth integrated continuity equation with rigid-lid (2.4) by the time-dependent term and fresh-water source term.

Equation 2.7 in the rigid-lid pressure method is then replaced by the time discretization of 2.15 which is:

$$\eta^{n+1} + \Delta t \partial_x H \widehat{u^{n+1}} + \Delta t \partial_y H \widehat{v^{n+1}} = \eta^n + \Delta t (P - E) \quad (2.16)$$

where the use of flow at time level $n+1$ makes the method implicit and backward in time. This is the preferred scheme since it still filters the fast unresolved wave motions by damping them. A centered scheme, such as Crank-Nicholson (see section 2.10.1), would alias the energy of the fast modes onto slower modes of motion.

As for the rigid-lid pressure method, equations 2.5, 2.6 and 2.16 can be re-arranged as follows:

$$u^* = u^n + \Delta t G_u^{(n+1/2)} \quad (2.17)$$

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

$$\eta^* = \epsilon_{fs} (\eta^n + \Delta t (P - E)) - \Delta t \left(\partial_x H \widehat{u^*} + \partial_y H \widehat{v^*} \right) \quad (2.19)$$

$$\partial_x g H \partial_x \eta^{n+1} + \partial_y g H \partial_y \eta^{n+1} - \frac{\epsilon_{fs} \eta^{n+1}}{\Delta t^2} = - \frac{\eta^*}{\Delta t^2} \quad (2.20)$$

$$u^{n+1} = u^* - \Delta t g \partial_x \eta^{n+1} \quad (2.21)$$

$$v^{n+1} = v^* - \Delta t g \partial_y \eta^{n+1} \quad (2.22)$$

Equations 2.17 to 2.22, solved sequentially, represent the pressure method algorithm with a backward implicit, linearized free surface. The method is still formerly a pressure method because in the limit of large Δt the rigid-lid method is recovered. However, the implicit treatment of the free-surface allows the flow to be divergent and for the surface pressure/elevation to respond on a finite time-scale (as opposed to instantly). To recover the rigid-lid formulation, we introduced a switch-like parameter, ϵ_{fs} (freesurfFac), which selects between the free-surface and rigid-lid; $\epsilon_{fs} = 1$ allows the free-surface to evolve; $\epsilon_{fs} = 0$ imposes the rigid-lid. The evolution in time and location of variables is exactly as it was for the rigid-lid model so that Fig. 2.1 is still applicable. Similarly, the calling sequence, given in Fig. 2.2, is as for the pressure-method.

2.5 Explicit time-stepping: Adams-Bashforth

In describing the the pressure method above we deferred describing the time discretization of the explicit terms. We have historically used the quasi-second order Adams-Bashforth method for all explicit terms in both the momentum and tracer equations. This is still the default mode of operation but it is now possible to use alternate schemes for tracers (see section 2.17).

FORWARD_STEP			
	THERMODYNAMICS		
		CALC_GT	
		GAD_CALC_RHS	$G_\theta^n = G_\theta(u, \theta^n)$
either		EXTERNAL_FORCING	$G_\theta^n = G_\theta^n + \mathcal{Q}$
		ADAMS_BASHFORTH2	$G_\theta^{(n+1/2)}$ (2.24)
or		EXTERNAL_FORCING	$G_\theta^{(n+1/2)} = G_\theta^{(n+1/2)} + \mathcal{Q}$
		TIMESTEP_TRACER	τ^* (2.23)
		IMPLDIFF	$\tau^{(n+1)}$ (2.27)

Figure 2.3: Calling tree for the Adams-Bashforth time-stepping of temperature with implicit diffusion. (THERMODYNAMICS, ADAMS_BASHFORTH2)

In the previous sections, we summarized an explicit scheme as:

$$\tau^* = \tau^n + \Delta t G_\tau^{(n+1/2)} \tag{2.23}$$

where τ could be any prognostic variable (u, v, θ or S) and τ^* is an explicit estimate of τ^{n+1} and would be exact if not for implicit-in-time terms. The parenthesis about $n+1/2$ indicates that the term is explicit and extrapolated forward in time and for this we use the quasi-second order Adams-Bashforth method:

$$G_\tau^{(n+1/2)} = (3/2 + \epsilon_{AB})G_\tau^n - (1/2 + \epsilon_{AB})G_\tau^{n-1} \tag{2.24}$$

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

A stability analysis for an oscillation equation should be given at this point. AJA needs to f on this...

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

2.6 Implicit time-stepping: backward method

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

$$\tau^{n+1} - \Delta t \partial_r \kappa_v \partial_r \tau^{n+1} = \tau^n + \Delta t G_\tau^{(n+1/2)} \tag{2.25}$$

where $G_\tau^{(n+1/2)}$ is the remaining explicit terms extrapolated using the Adams-Bashforth method as described above. Equation 2.25 can be split split into:

$$\tau^* = \tau^n + \Delta t G_\tau^{(n+1/2)} \tag{2.26}$$

$$\tau^{n+1} = \mathcal{L}_\tau^{-1}(\tau^*) \tag{2.27}$$

where $\mathcal{L}_\tau^{-1}$ is the inverse of the operator

$$\mathcal{L}_\tau = [1 + \Delta t \partial_r \kappa_v \partial_r] \tag{2.28}$$

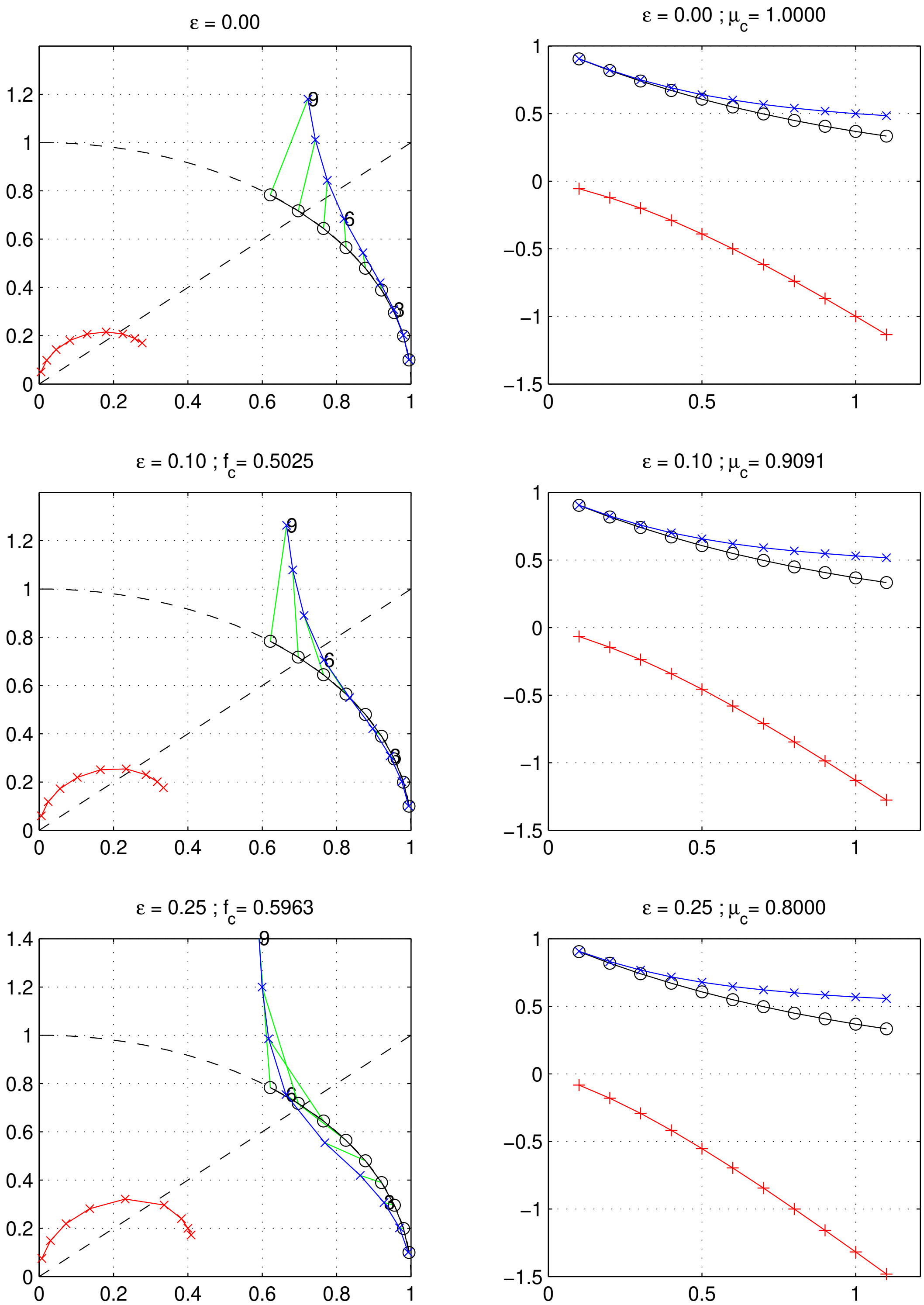


Figure 2.4: Oscillatory and damping response of quasi-second order Adams-Bashforth scheme for different values of the ϵ_{AB} parameter (0., 0.1, 0.25, from top to bottom) The analytical solution (in black), the physical mode (in blue) and the numerical mode (in red) are represented with a CFL step of 0.1. The left column represents the oscillatory response on the complex plane for CFL ranging from 0.1 up to 0.9. The right column represents the damping response amplitude (y-axis) function of the CFL (x-axis).

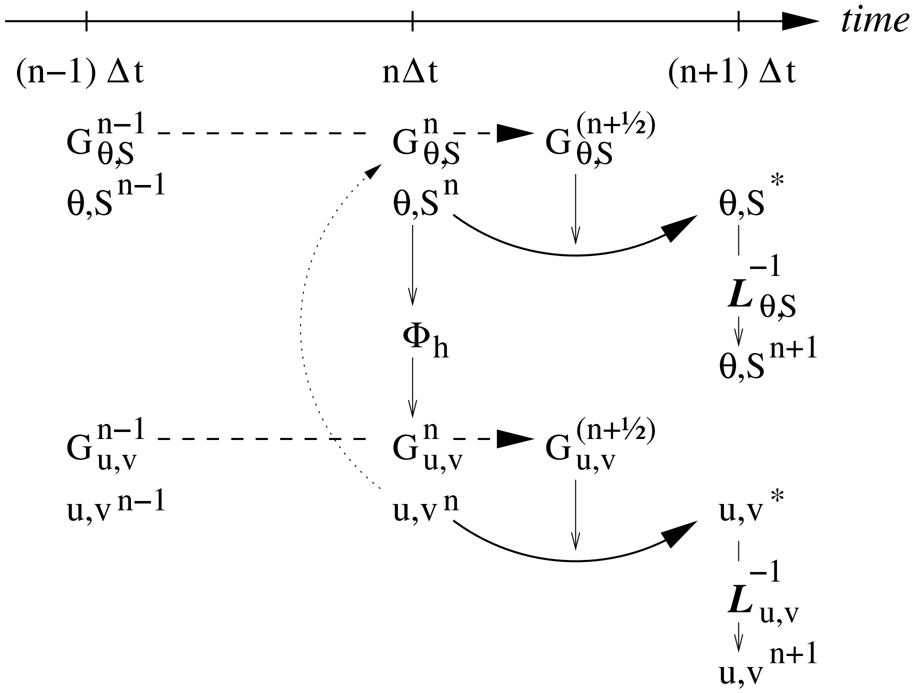


Figure 2.5: A schematic of the explicit Adams-Bashforth and implicit time-stepping phases of the algorithm. All prognostic variables are co-located in time. Explicit tendencies are evaluated at time level n as a function of the state at that time level (dotted arrow). The explicit tendency from the previous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

Equation 2.26 looks exactly as 2.23 while 2.27 involves an operator or matrix inversion. By re-arranging 2.25 in this way we have cast the method as an explicit prediction step and an implicit step allowing the latter to be inserted into the over all algorithm with minimal interference.

Fig. 2.3 shows the calling sequence for stepping forward a tracer variable such as temperature.

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

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

The Adams-Bashforth extrapolation of explicit tendencies fits neatly into the pressure method algorithm when all state variables are co-located in time. Fig. 2.5 illustrates the location of variables in time and the evolution of the algorithm with time. The algorithm can be represented by the sequential solution of

FORWARD_STEP				
		EXTERNAL_FIELDS_LOAD		
		DO_ATMOSPHERIC_PHYS		
		DO_OCEANIC_PHYS		
	THERMODYNAMICS			
		CALC_GT		
			GAD_CALC_RHS	$G_\theta^n = G_\theta(u, \theta^n)$ (2.29)
			EXTERNAL_FORCING	$G_\theta^n = G_\theta^n + \mathcal{Q}$
			ADAMS_BASHFORTH2	$G_\theta^{(n+1/2)}$ (2.30)
		TIMESTEP_TRACER		θ^* (2.31)
		IMPLDIFF		$\theta^{(n+1)}$ (2.32)
	DYNAMICS			
		CALC_PHI_HYD		ϕ_{hyd}^n (2.33)
		MOM_FLUXFORM or MOM_VECINV		$G_{\vec{\mathbf{v}}}^n$ (2.34)
		TIMESTEP		$\vec{\mathbf{v}}^*$ (2.35, 2.36)
		IMPLDIFF		$\vec{\mathbf{v}}^{**}$ (2.37)
	UPDATE_R_STAR or UPDATE_SURF_DR			(NonLin-FS only)
	SOLVE_FOR_PRESSURE			
		CALC_DIV_GHAT		η^* (2.38)
		CG2D		η^{n+1} (2.39)
	MOMENTUM_CORRECTION_STEP			
		CALC_GRAD_PHI_SURF		$\nabla \eta^{n+1}$
		CORRECTION_STEP		u^{n+1},v^{n+1} (2.40)
	TRACERS_CORRECTION_STEP			
		CYCLE_TRACER		θ^{n+1}
		FILTER		Shapiro Filter, Zonal Filter (FFT)
		CONVECTIVE_ADJUSTMENT		

Figure 2.6: Calling tree for the overall synchronous algorithm using Adams-Bashforth time-stepping. The place where the model geometry (**hFac** factors) is updated is added here but is only relevant for the non-linear free-surface algorithm. For completeness, the external forcing, ocean and atmospheric physics have been added, although they are mainly optional

the follow equations:

$$G_{\theta,S}^{n} = G_{\theta,S}(u^n,\theta^n,S^n) \tag{2.29}$$

$$G_{\theta,S}^{(n+1/2)} = (3/2+\epsilon_{AB})G_{\theta,S}^{n} - (1/2+\epsilon_{AB})G_{\theta,S}^{n-1} \tag{2.30}$$

$$(\theta^*,S^*) = (\theta^n,S^n) + \Delta t G_{\theta,S}^{(n+1/2)} \tag{2.31}$$

$$(\theta^{n+1},S^{n+1}) = \mathcal{L}_{\theta,S}^{-1}(\theta^*,S^*) \tag{2.32}$$

$$\phi_{hyd}^{n} = \int b(\theta^n,S^n)dr \tag{2.33}$$

$$\vec{\mathbf{G}}_{\vec{\mathbf{v}}}^{n} = \vec{\mathbf{G}}_{\vec{\mathbf{v}}}(\vec{\mathbf{v}}^n,\phi_{hyd}^{n}) \tag{2.34}$$

$$\vec{\mathbf{G}}_{\vec{\mathbf{v}}}^{(n+1/2)} = (3/2+\epsilon_{AB})\vec{\mathbf{G}}_{\vec{\mathbf{v}}}^{n} - (1/2+\epsilon_{AB})\vec{\mathbf{G}}_{\vec{\mathbf{v}}}^{n-1} \tag{2.35}$$

$$\vec{\mathbf{v}}^{*} = \vec{\mathbf{v}}^{n} + \Delta t\vec{\mathbf{G}}_{\vec{\mathbf{v}}}^{(n+1/2)} \tag{2.36}$$

$$\vec{\mathbf{v}}^{**} = \mathcal{L}_{\vec{\mathbf{v}}}^{-1}(\vec{\mathbf{v}}^{*}) \tag{2.37}$$

$$\eta^* = \epsilon_{fs}\left(\eta^n + \Delta t(P-E)\right) - \Delta t\nabla\cdot H\widehat{\vec{\mathbf{v}}^{**}} \tag{2.38}$$

$$\nabla\cdot gH\nabla\eta^{n+1} - \frac{\epsilon_{fs}\eta^{n+1}}{\Delta t^2} = -\frac{\eta^*}{\Delta t^2} \tag{2.39}$$

$$\vec{\mathbf{v}}^{n+1} = \vec{\mathbf{v}}^{**} - \Delta t g\nabla\eta^{n+1} \tag{2.40}$$

Fig. 2.5 illustrates the location of variables in time and evolution of the algorithm with time. The Adams-Bashforth extrapolation of the tracer tendencies is illustrated by the dashed arrow, the prediction at $n+1$ is indicated by the solid arc. Inversion of the implicit terms, $\mathcal{L}_{\theta,S}^{-1}$, then yields the new tracer fields at $n+1$. All these operations are carried out in subroutine *THERMODYNAMICS* an subsidiaries, which correspond to equations 2.29 to 2.32. Similarly illustrated is the Adams-Bashforth extrapolation of accelerations, stepping forward and solving of implicit viscosity and surface pressure gradient terms, corresponding to equations 2.34 to 2.40. These operations are carried out in subroutines *DYNAMCIS*, *SOLVE_FOR_PRESSURE* and *MOMENTUM_CORRECTION_STEP*. This, then, represents an entire algorithm for stepping forward the model one time-step. The corresponding calling tree is given in 2.6.

2.8 Staggered baroclinic time-stepping

For well stratified problems, internal gravity waves may be the limiting process for determining a stable time-step. In the circumstance, it is more efficient to stagger in time the thermodynamic variables with the flow variables. Fig. 2.7 illustrates the staggering and algorithm. The key difference between this and Fig. 2.5 is that the thermodynamic variables are solved after the dynamics, using the recently updated flow field. This essentially allows the gravity wave terms to leap-frog in time giving second order accuracy and more stability.

The essential change in the staggered algorithm is that the thermodynamics solver is delayed from half a time step, allowing the use of the most recent velocities to compute the advection terms. Once the thermodynamics fields are updated, the hydrostatic pressure is computed to step forward the dynamics. Note that the pressure gradient must also be taken out of the Adams-Bashforth extrapolation. Also, retaining the integer time-levels, n and $n+1$, does not give a user the sense of where variables are located in time. Instead, we re-write the entire algorithm, 2.29 to 2.40, annotating the position in time

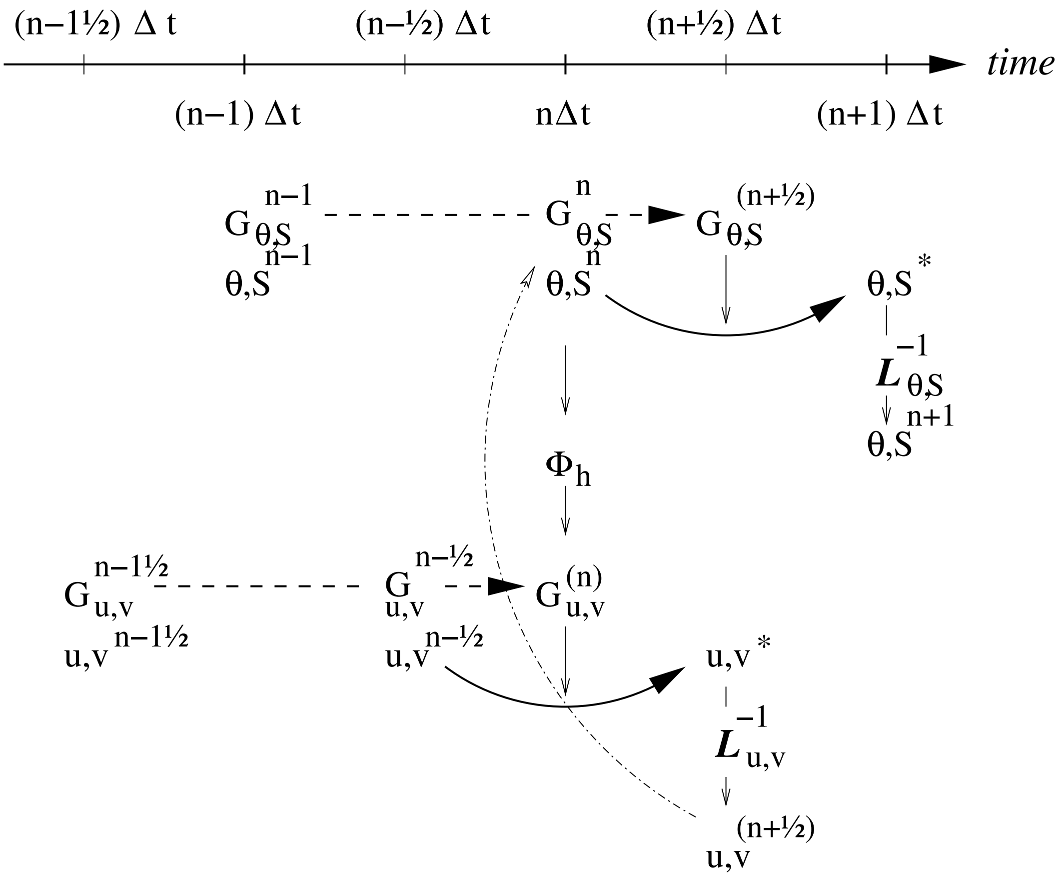


Figure 2.7: A schematic of the explicit Adams-Bashforth and implicit time-stepping phases of the algorithm but with staggering in time of thermodynamic variables with the flow. Explicit momentum tendencies are evaluated at time level $n-1/2$ as a function of the flow field at that time level $n-1/2$. The explicit tendency from the previous time level, $n-3/2$, is used to extrapolate tendencies to n (dashed arrow). The hydrostatic pressure/geo-potential ϕ_{hyd} is evaluated directly at time level n (vertical arrows) and used with the extrapolated tendencies to step forward the flow variables from $n-1/2$ to $n+1/2$ (solid arc-arrow). The implicit-in-time operator $\mathcal{L}_{\mathbf{u},\mathbf{v}}$ (vertical arrows) is then applied to the previous estimation of the the flow field ($*$-variables) and yields to the two velocity components u, v at time level $n+1/2$. These are then used to calculate the advection term (dashed arc-arrow) of the thermo-dynamics tendencies at time step n. The extrapolated thermodynamics tendency, from time level $n-1$ and n to $n+1/2$, allows thermodynamic variables to be stably integrated forward-in-time (solid arc-arrow) up to time level $n+1$.

of variables appropriately:

$$\phi^n_{hyd} = \int b(\theta^n, S^n) dr \tag{2.41}$$
$$\vec{\mathbf{G}}_{\vec{\mathbf{v}}}^{n-1/2} = \vec{\mathbf{G}}_{\vec{\mathbf{v}}}(\vec{\mathbf{v}}^{n-1/2}) \tag{2.42}$$
$$\vec{\mathbf{G}}_{\vec{\mathbf{v}}}^{(n)} = (3/2 + \epsilon_{AB})\vec{\mathbf{G}}_{\vec{\mathbf{v}}}^{n-1/2} - (1/2 + \epsilon_{AB})\vec{\mathbf{G}}_{\vec{\mathbf{v}}}^{n-3/2} \tag{2.43}$$
$$\vec{\mathbf{v}}^{*} = \vec{\mathbf{v}}^{n-1/2} + \Delta t \left(\vec{\mathbf{G}}_{\vec{\mathbf{v}}}^{(n)} - \nabla \phi^n_{hyd} \right) \tag{2.44}$$
$$\vec{\mathbf{v}}^{**} = \mathcal{L}^{-1}_{\vec{\mathbf{v}}}(\vec{\mathbf{v}}^{*}) \tag{2.45}$$
$$\eta^{*} = \epsilon_{fs}\left(\eta^{n-1/2} + \Delta t (P-E)^n \right) - \Delta t \nabla \cdot H \widehat{\vec{\mathbf{v}}^{**}} \tag{2.46}$$
$$\nabla \cdot g H \nabla \eta^{n+1/2} - \frac{\epsilon_{fs}\eta^{n+1/2}}{\Delta t^2} = - \frac{\eta^{*}}{\Delta t^2} \tag{2.47}$$
$$\vec{\mathbf{v}}^{n+1/2} = \vec{\mathbf{v}}^{**} - \Delta t g \nabla \eta^{n+1/2} \tag{2.48}$$
$$G^n_{\theta,S} = G_{\theta,S}(u^{n+1/2}, \theta^n, S^n) \tag{2.49}$$
$$G^{(n+1/2)}_{\theta,S} = (3/2 + \epsilon_{AB}) G^n_{\theta,S} - (1/2 + \epsilon_{AB}) G^{n-1}_{\theta,S} \tag{2.50}$$
$$(\theta^*, S^*) = (\theta^n, S^n) + \Delta t G^{(n+1/2)}_{\theta,S} \tag{2.51}$$
$$(\theta^{n+1}, S^{n+1}) = \mathcal{L}^{-1}_{\theta,S}(\theta^*, S^*) \tag{2.52}$$

The corresponding calling tree is given in 2.8. The staggered algorithm is activated with the run-time flag **staggerTimeStep**=*.TRUE.* in parameter file *data*, namelist *PARM01*.

The only difficulty with this approach is apparent in equation 2.49 and illustrated by the dotted arrow connecting $u, v^{n+1/2}$ with G^n_θ. The flow used to advect tracers around is not naturally located in time. This could be avoided by applying the Adams-Bashforth extrapolation to the tracer field itself and advecting that around but this approach is not yet available. We're not aware of any detrimental effect of this feature. The difficulty lies mainly in interpretation of what time-level variables and terms correspond to.

2.9 Non-hydrostatic formulation

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

The momentum equations are discretized in time as follows:

$$\frac{1}{\Delta t} u^{n+1} + g \partial_x \eta^{n+1} + \partial_x \phi^{n+1}_{nh} = \frac{1}{\Delta t} u^n + G^{(n+1/2)}_u \tag{2.53}$$
$$\frac{1}{\Delta t} v^{n+1} + g \partial_y \eta^{n+1} + \partial_y \phi^{n+1}_{nh} = \frac{1}{\Delta t} v^n + G^{(n+1/2)}_v \tag{2.54}$$
$$\frac{1}{\Delta t} w^{n+1} + \partial_r \phi^{n+1}_{nh} = \frac{1}{\Delta t} w^n + G^{(n+1/2)}_w \tag{2.55}$$

which must satisfy the discrete-in-time depth integrated continuity, equation 2.16 and the local continuity equation

$$\partial_x u^{n+1} + \partial_y v^{n+1} + \partial_r w^{n+1} = 0 \tag{2.56}$$

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

$$u^* = u^n + \Delta t G^{(n+1/2)}_u$$
$$v^* = v^n + \Delta t G^{(n+1/2)}_v$$
$$w^* = w^n + \Delta t G^{(n+1/2)}_w$$

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

$$u^{n+1} = u^{**} - \Delta t \partial_x \phi^{n+1}_{nh} \qquad u^{**} = u^* - \Delta t g \partial_x \eta^{n+1} \tag{2.57}$$
$$v^{n+1} = v^{**} - \Delta t \partial_y \phi^{n+1}_{nh} \qquad v^{**} = v^* - \Delta t g \partial_y \eta^{n+1} \tag{2.58}$$

- FORWARD_STEP
    - EXTERNAL_FIELDS_LOAD
    - DO_ATMOSPHERIC_PHYS
    - DO_OCEANIC_PHYS
  - DYNAMICS
    - CALC_PHI_HYD — ϕ^n_{hyd} (2.41)
    - MOM_FLUXFORM or MOM_VECINV — $G^{n-1/2}_{\vec{\mathbf{v}}}$ (2.42)
    - TIMESTEP — $\vec{\mathbf{v}}^*$ (2.43, 2.44)
    - IMPLDIFF — $\vec{\mathbf{v}}^{**}$ (2.45)
  - UPDATE_R_STAR or UPDATE_SURF_DR — (NonLin-FS only)
  - SOLVE_FOR_PRESSURE
    - CALC_DIV_GHAT — η^* (2.46)
    - CG2D — $\eta^{n+1/2}$ (2.47)
  - MOMENTUM_CORRECTION_STEP
    - CALC_GRAD_PHI_SURF — $\nabla\eta^{n+1/2}$
    - CORRECTION_STEP — $u^{n+1/2}$,$v^{n+1/2}$ (2.48)
  - THERMODYNAMICS
    - CALC_GT
      - GAD_CALC_RHS — $G^n_\theta = G_\theta(u, \theta^n)$ (2.49)
      - EXTERNAL_FORCING — $G^n_\theta = G^n_\theta + \mathcal{Q}$
      - ADAMS_BASHFORTH2 — $G^{(n+1/2)}_\theta$ (2.50)
    - TIMESTEP_TRACER — θ^* (2.51)
    - IMPLDIFF — $\theta^{(n+1)}$ (2.52)
  - TRACERS_CORRECTION_STEP
    - CYCLE_TRACER — θ^{n+1}
    - FILTER — Shapiro Filter, Zonal Filter (FFT)
    - CONVECTIVE_ADJUSTMENT

Figure 2.8: Calling tree for the overall staggered algorithm using Adams-Bashforth time-stepping. The place where the model geometry (**hFac** factors) is updated is added here but is only relevant for the non-linear free-surface algorithm.

Substituting into the depth integrated continuity (equation 2.16) gives

$$\partial_x H \partial_x \left(g\eta^{n+1} + \widehat{\phi}_{nh}^{n+1} \right) + \partial_y H \partial_y \left(g\eta^{n+1} + \widehat{\phi}_{nh}^{n+1} \right) - \frac{\epsilon_{fs}\eta^{n+1}}{\Delta t^2} = -\frac{\eta^*}{\Delta t^2} \tag{2.59}$$

which is approximated by equation 2.20 on the basis that i) ϕ_{nh}^{n+1} is not yet known and ii) $\nabla \widehat{\phi}_{nh} << g\nabla\eta$. If 2.20 is solved accurately then the implication is that $\widehat{\phi}_{nh} \approx 0$ so that the non-hydrostatic pressure field does not drive barotropic motion.

The flow must satisfy non-divergence (equation 2.56) locally, as well as depth integrated, and this constraint is used to form a 3-D elliptic equations for ϕ_{nh}^{n+1}:

$$\partial_{xx}\phi_{nh}^{n+1} + \partial_{yy}\phi_{nh}^{n+1} + \partial_{rr}\phi_{nh}^{n+1} = \left(\partial_x u^{**} + \partial_y v^{**} + \partial_r w^*\right)/\Delta t \tag{2.60}$$

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

$$\begin{aligned}
u^* &= u^n + \Delta t G_u^{(n+1/2)} && (2.61)\\
v^* &= v^n + \Delta t G_v^{(n+1/2)} && (2.62)\\
w^* &= w^n + \Delta t G_w^{(n+1/2)} && (2.63)\\
\eta^* \;=\; \epsilon_{fs}\left(\eta^n + \Delta t(P-E)\right) &- \Delta t \left(\partial_x H \widehat{u^*} + \partial_y H \widehat{v^*} \right) && (2.64)\\
\partial_x g H \partial_x \eta^{n+1} + \partial_y g H \partial_y \eta^{n+1} &- \frac{\epsilon_{fs}\eta^{n+1}}{\Delta t^2} \;=\; -\frac{\eta^*}{\Delta t^2} && (2.65)\\
u^{**} &= u^* - \Delta t g \partial_x \eta^{n+1} && (2.66)\\
v^{**} &= v^* - \Delta t g \partial_y \eta^{n+1} && (2.67)\\
\partial_{xx}\phi_{nh}^{n+1} + \partial_{yy}\phi_{nh}^{n+1} + \partial_{rr}\phi_{nh}^{n+1} &= \left(\partial_x u^{**} + \partial_y v^{**} + \partial_r w^*\right)/\Delta t && (2.68)\\
u^{n+1} &= u^{**} - \Delta t \partial_x \phi_{nh}^{n+1} && (2.69)\\
v^{n+1} &= v^{**} - \Delta t \partial_y \phi_{nh}^{n+1} && (2.70)\\
\partial_r w^{n+1} &= -\partial_x u^{n+1} - \partial_y v^{n+1} && (2.71)
\end{aligned}$$

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

2.10 Variants on the Free Surface

We now describe the various formulations of the free-surface that include non-linear forms, implicit in time using Crank-Nicholson, explicit and [one day] split-explicit. First, we'll reiterate the underlying algorithm but this time using the notation consistent with the more general vertical coordinate r. The elliptic equation for free-surface coordinate (units of r), corresponding to 2.16, and assuming no non-hydrostatic effects ($\epsilon_{nh} = 0$) is:

$$\epsilon_{fs}\eta^{n+1} - \nabla_h \cdot \Delta t^2 (R_o - R_{fixed}) \nabla_h b_s \eta^{n+1} = \eta^* \tag{2.72}$$

where

$$\eta^* = \epsilon_{fs}\, \eta^n - \Delta t \nabla_h \cdot \int_{R_{fixed}}^{R_o} \vec{\mathbf{v}}^* dr \;+\; \epsilon_{fw} \Delta t (P-E)^n \tag{2.73}$$

S/R SOLVE_FOR_PRESSURE (*solve_for_pressure.F*)
u^*: **gU** (*DYNVARS.h*)
v^*: **gV** (*DYNVARS.h*)
η^*: **cg2d_b** (*SOLVE_FOR_PRESSURE.h*)
η^{n+1}: **etaN** (DYNVARS.h)

Once η^{n+1} has been found, substituting into 2.5, 2.6 yields $\vec{\mathbf{v}}^{n+1}$ if the model is hydrostatic ($\epsilon_{nh} = 0$):

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

This is known as the correction step. However, when the model is non-hydrostatic ($\epsilon_{nh} = 1$) we need an additional step and an additional equation for ϕ'_{nh}. This is obtained by substituting 2.53, 2.54 and 2.55 into continuity:

$$\left[\nabla_h^2 + \partial_{rr}\right] {\phi'_{nh}}^{n+1} = \frac{1}{\Delta t}\left(\nabla_h \cdot \vec{\mathbf{v}}^{**} + \partial_r \dot{r}^*\right) \tag{2.74}$$

where

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

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

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

$$\vec{\mathbf{v}}^{n+1} = \vec{\mathbf{v}}^{**} - \epsilon_{nh} \Delta t \nabla_h {\phi'_{nh}}^{n+1} \tag{2.75}$$

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

S/R CORRECTION_STEP (*correction_step.F*)
η^{n+1}: **etaN** (*DYNVARS.h*)
ϕ_{nh}^{n+1}: **phi_nh** (NH_VARS.h)
u^*: **gU** (*DYNVARS.h*)
v^*: **gV** (*DYNVARS.h*)
u^{n+1}: **uVel** (*DYNVARS.h*)
v^{n+1}: **vVel** (*DYNVARS.h*)

Regarding the implementation of the surface pressure solver, all computation are done within the routine *SOLVE_FOR_PRESSURE* and its dependent calls. The standard method to solve the 2D elliptic problem (2.72) uses the conjugate gradient method (routine *CG2D*); the solver matrix and conjugate gradient operator are only function of the discretized domain and are therefore evaluated separately, before the time iteration loop, within *INI_CG2D*. The computation of the RHS η^* is partly done in *CALC_DIV_GHAT* and in *SOLVE_FOR_PRESSURE*.

The same method is applied for the non hydrostatic part, using a conjugate gradient 3D solver (*CG3D*) that is initialized in *INI_CG3D*. The RHS terms of 2D and 3D problems are computed together at the same point in the code.

2.10.1 Crank-Nicolson barotropic time stepping

The full implicit time stepping described previously is unconditionally stable but damps the fast gravity waves, resulting in a loss of potential energy. The modification presented now allows one to combine an implicit part (β, γ) and an explicit part $(1-\beta, 1-\gamma)$ for the surface pressure gradient (β) and for the barotropic flow divergence (γ).

For instance, $\beta = \gamma = 1$ is the previous fully implicit scheme; $\beta = \gamma = 1/2$ is the non damping (energy conserving), unconditionally stable, Crank-Nicolson scheme; $(\beta, \gamma) = (1, 0)$ or $= (0, 1)$ corresponds to the forward - backward scheme that conserves energy but is only stable for small time steps.

In the code, β, γ are defined as parameters, respectively **implicSurfPress**, **implicDiv2DFlow**. They are read from the main parameter file "*data*" (namelist *PARM01*) and are set by default to 1,1.

Equations 2.17 – 2.22 are modified as follows:

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

$$\epsilon_{fs} \frac{\eta^{n+1} - \eta^n}{\Delta t} + \nabla_h \cdot \int_{R_{fixed}}^{R_o} [\gamma \vec{\mathbf{v}}^{n+1} + (1-\gamma)\vec{\mathbf{v}}^n] dr = \epsilon_{fw}(P - E) \tag{2.76}$$

We set

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

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

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

and then to compute (*CORRECTION_STEP*):

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

Notes:

1. The RHS term of equation 2.76 corresponds the contribution of fresh water flux (P-E) to the free-surface variations ($\epsilon_{fw} = 1$, **useRealFreshWater**=*TRUE* in parameter file *data*). In order to remain consistent with the tracer equation, specially in the non-linear free-surface formulation, this term is also affected by the Crank-Nicolson time stepping. The RHS reads: $\epsilon_{fw}(\gamma(P-E)^{n+1/2} + (1-\gamma)(P-E)^{n-1/2})$

2. The stability criteria with Crank-Nicolson time stepping for the pure linear gravity wave problem in cartesian coordinates is:
   - $\beta + \gamma < 1$: unstable
   - $\beta \geq 1/2$ and $\gamma \geq 1/2$: stable
   - $\beta + \gamma \geq 1$: stable if

$$c_{max}^2(\beta - 1/2)(\gamma - 1/2) + 1 \geq 0$$
$$\text{with} \quad c_{max} = 2\Delta t\,\sqrt{gH}\,\sqrt{\frac{1}{\Delta x^2} + \frac{1}{\Delta y^2}}$$

3. A similar mixed forward/backward time-stepping is also available for the non-hydrostatic algorithm, with a fraction β_{nh} ($0 < \beta_{nh} \leq 1$) of the non-hydrostatic pressure gradient being evaluated at time step $n+1$ (backward in time) and the remaining part ($1 - \beta_{nh}$) being evaluated at time step n (forward in time). The run-time parameter **implicitNHPress** corresponding to the implicit fraction β_{nh} of the non-hydrostatic pressure is set by default to the implicit fraction β of surface pressure (**implicSurfPress**), but can also be specified independently (in main parameter file *data*, namelist *PARM01*).

2.10.2 Non-linear free-surface

Recently, new options have been added to the model that concern the free surface formulation.

2.10.2.1 pressure/geo-potential and free surface

For the atmosphere, since $\phi = \phi_{topo} - \int_{p_s}^{p} \alpha dp$, subtracting the reference state defined in section 1.4.1.2 :

$$\phi_o = \phi_{topo} - \int_{p_o}^{p} \alpha_o dp \quad \text{with} \quad \phi_o(p_o) = \phi_{topo}$$

we get:

$$\phi' = \phi - \phi_o = \int_{p}^{p_s} \alpha dp - \int_{p}^{p_o} \alpha_o dp$$

For the ocean, the reference state is simpler since ρ_c does not dependent on z ($b_o = g$) and the surface reference position is uniformly $z = 0$ ($R_o = 0$), and the same subtraction leads to a similar relation. For both fluid, using the isomorphic notations, we can write:

$$\phi' = \int_{r}^{r_{surf}} b\ dr - \int_{r}^{R_o} b_o dr$$

and re-write as:

$$\phi' = \int_{R_o}^{r_{surf}} b\ dr + \int_{r}^{R_o} (b - b_o)dr \tag{2.77}$$

or:

$$\phi' = \int_{R_o}^{r_{surf}} b_o dr + \int_{r}^{r_{surf}} (b - b_o) dr \tag{2.78}$$

In section 1.3.6, following eq.2.77, the pressure/geo-potential ϕ' has been separated into surface (ϕ_s), and hydrostatic anomaly (ϕ'_{hyd}). In this section, the split between ϕ_s and ϕ'_{hyd} is made according to equation 2.78. This slightly different definition reflects the actual implementation in the code and is valid for both linear and non-linear free-surface formulation, in both r-coordinate and r*-coordinate.

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

$$\phi'_{hyd} = \int_{r}^{r_{surf}} (b - b_o) dr \simeq \int_{r}^{R_o} (b - b_o) dr$$

Note that in this case, the two definitions of ϕ_s and ϕ'_{hyd} from equation 2.77 and 2.78 converge toward the same (approximated) expressions: $\phi_s = \int_{R_o}^{r_{surf}} b_o dr$ and $\phi'_{hyd} = \int_{r}^{R_o} b' dr$.
On the contrary, the unapproximated formulation ("non-linear free-surface", see the next section) retains the full expression: $\phi'_{hyd} = \int_{r}^{r_{surf}} (b - b_o) dr$. This is obtained by selecting **nonlinFreeSurf**=4 in parameter file *data*.

Regarding the surface potential:

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

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

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

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

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

In the non-linear formulation, the continuous form of the model equations remains unchanged, except for the 2D continuity equation (2.16) which is now integrated from $R_{fixed}(x, y)$ up to $r_{surf} = R_o + \eta$:

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

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

If the column thickness is evaluated at time step n, and with implicit treatment of the surface potential gradient, equations (2.72 and 2.73) becomes:

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

where

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

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

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

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

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

Finally, another option is a "linearized" formulation where the total column thickness appears only in the integral term of the RHS (2.73) but not directly in the equation (2.72).

Those different options (see Table 2.1) have been tested and show little differences. However, we recommend the use of the most precise method (the 1rst one) since the computation cost involved in the solver matrix update is negligible.

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

Table 2.1: Non-linear free-surface flags

2.10.2.3 Tracer conservation with non-linear free-surface

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

A simplified version of the staggered in time, non-linear free-surface algorithm is detailed hereafter, and can be compared to the equivalent linear free-surface case (eq. 2.42 to 2.52) and can also be easily transposed to the synchronous time-stepping case. Among the simplifications, salinity equation, implicit operator and detailed elliptic equation are omitted. Surface forcing is explicitly written as fluxes of temperature, fresh water and momentum, $Q^{n+1/2}, P^{n+1/2}, F_{\mathbf{v}}^n$ respectively. h^n and dh^n are the column and grid box thickness in r-coordinate.

$$\phi_{hyd}^n = \int b(\theta^n, S^n, r) dr \tag{2.79}$$

$$\vec{\mathbf{G}}_{\vec{\mathbf{v}}}^{n-1/2} = \vec{\mathbf{G}}_{\vec{\mathbf{v}}}(dh^{n-1}, \vec{\mathbf{v}}^{n-1/2}) \;\; ; \;\; \vec{\mathbf{G}}_{\vec{\mathbf{v}}}^{(n)} = \frac{3}{2}\vec{\mathbf{G}}_{\vec{\mathbf{v}}}^{n-1/2} - \frac{1}{2}\vec{\mathbf{G}}_{\vec{\mathbf{v}}}^{n-3/2} \tag{2.80}$$

$$\vec{\mathbf{v}}^* = \vec{\mathbf{v}}^{n-1/2} + \Delta t \frac{dh^{n-1}}{dh^n} \left(\vec{\mathbf{G}}_{\vec{\mathbf{v}}}^{(n)} + F_{\vec{\mathbf{v}}}^n / dh^{n-1} \right) - \Delta t \nabla \phi_{hyd}^n \tag{2.81}$$

$\longrightarrow$ *update model geometry :* $\mathbf{hFac}(dh^n)$

$$\begin{aligned} \eta^{n+1/2} &= \eta^{n-1/2} + \Delta t P^{n+1/2} - \Delta t \nabla \cdot \int \vec{\mathbf{v}}^{n+1/2} dh^n \\ &= \eta^{n-1/2} + \Delta t P^{n+1/2} - \Delta t \nabla \cdot \int \left(\vec{\mathbf{v}}^* - g \Delta t \nabla \eta^{n+1/2} \right) dh^n \end{aligned} \tag{2.82}$$

$$\vec{\mathbf{v}}^{n+1/2} = \vec{\mathbf{v}}^* - g \Delta t \nabla \eta^{n+1/2} \tag{2.83}$$

$$h^{n+1} = h^n + \Delta t P^{n+1/2} - \Delta t \nabla \cdot \int \vec{\mathbf{v}}^{n+1/2} dh^n \tag{2.84}$$

$$G_\theta^n = G_\theta(dh^n, u^{n+1/2}, \theta^n) \;\; ; \;\; G_\theta^{(n+1/2)} = \frac{3}{2} G_\theta^n - \frac{1}{2} G_\theta^{n-1} \tag{2.85}$$

$$\theta^{n+1} = \theta^n + \Delta t \frac{dh^n}{dh^{n+1}} \left(G_\theta^{(n+1/2)} + (P^{n+1/2}(\theta_{\text{rain}} - \theta^n) + Q^{n+1/2})/dh^n \right) \tag{2.86}$$

Two steps have been added to linear free-surface algorithm (eq. 2.42 to 2.52): Firstly, the model "geometry" (here the **hFacC,W,S**) is updated just before entering *SOLVE_FOR_PRESSURE*, using the current dh^n field. Secondly, the vertically integrated continuity equation (eq.2.84) has been added (**exactConserv**=*TRUE*, in parameter file *data*, namelist *PARM01*) just before computing the vertical velocity, in subroutine *INTEGR_CONTINUITY*. Although this equation might appear redundant with eq.2.82, the integrated column thickness h^{n+1} will be different from $\eta^{n+1/2} + H$ in the following cases:

- when Crank-Nicolson time-stepping is used (see section 2.10.1).

- when filters are applied to the flow field, after (2.83) and alter the divergence of the flow.

- when the solver does not iterate until convergence ; for example, because a too large residual target was set (**cg2dTargetResidual**, parameter file *data*, namelist *PARM02*).

In this staggered time-stepping algorithm, the momentum tendencies are computed using dh^{n-1} geometry factors. (eq.2.80) and then rescaled in subroutine *TIMESTEP*, (eq.2.81), similarly to tracer tendencies (see section 2.10.2.3). The tracers are stepped forward later, using the recently updated flow field $\mathbf{v}^{n+1/2}$ and the corresponding model geometry dh^n to compute the tendencies (eq.2.85); Then the tendencies are rescaled by dh^n/dh^{n+1} to derive the new tracers values $(\theta, S)^{n+1}$ (eq.2.86, in subroutine *CALC_GT, CALC_GS*).

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

Regarding the restart procedure, two 2.D fields h^{n-1} and $(h^n - h^{n-1})/\Delta t$ in addition to the standard state variables and tendencies ($\eta^{n-1/2}$, $\mathbf{v}^{n-1/2}$, θ^n, S^n, $\mathbf{G}_{\mathbf{v}}^{n-3/2}$, $G_{\theta,S}^{n-1}$) are stored in a "*pickup*" file. The model restarts reading this "*pickup*" file, then update the model geometry according to h^{n-1}, and compute h^n and the vertical velocity before starting the main calling sequence (eq.2.79 to 2.86, *S/R FORWARD_STEP*).

INTEGR_CONTINUITY (*integr_continuity.F*)
$h^{n+1} - H_o$: **etaH** (*DYNVARS.h*)
$h^n - H_o$: **etaHnm1** (*SURFACE.h*)
$(h^{n+1} - h^n)/\Delta t$: **dEtaHdt** (*SURFACE.h*)

2.10.2.5 Non-linear free-surface and vertical resolution

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

A better alternative to the vanishing level problem has been found and implemented recently, and rely on a different vertical coordinate r^* : The time variation ot the total column thickness becomes part of the r* coordinate motion, as in a σ_z, σ_p model, but the fixed part related to topography is treated as in a height or pressure coordinate model. A complete description is given in *Adcroft and Campin* [2004].

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

2.11 Spatial discretization of the dynamical equations

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

2.11.1 The finite volume method: finite volumes versus finite difference

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

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

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

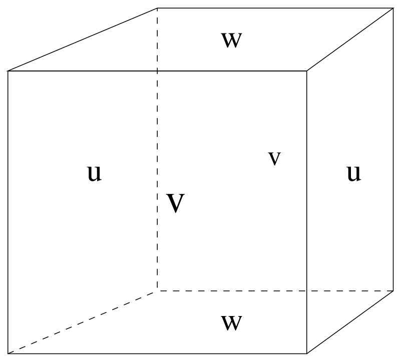


Figure 2.9: Three dimensional staggering of velocity components. This facilitates the natural discretization of the continuity and tracer equations.

can be discretized by integrating over finite sub-domains, i.e. the lengths Δx_i:

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

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

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

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

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

Choosing simple centered second-order interpolation and differencing recovers the same ODE's resulting from finite differencing for the interior of a fluid. Differences arise at boundaries where a boundary is not positioned on a regular or smoothly varying grid. This method is used to represent the topography using lopped cell, see *Adcroft et al.* [1997]. Subtle difference also appear in more than one dimension away from boundaries. This happens because the each direction is discretized independently in the finite difference method while the integrating over finite volume implicitly treats all directions simultaneously. Illustration of this is given in *Adcroft and Campin* [2002].

2.11.2 C grid staggering of variables

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

Fig. 2.9 shows the components of flow (u,v,w) staggered in space such that the zonal component falls on the interface between continuity cells in the zonal direction. Similarly for the meridional and vertical directions. The continuity cell is synonymous with tracer cells (they are one and the same).

2.11.3 Grid initialization and data

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

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

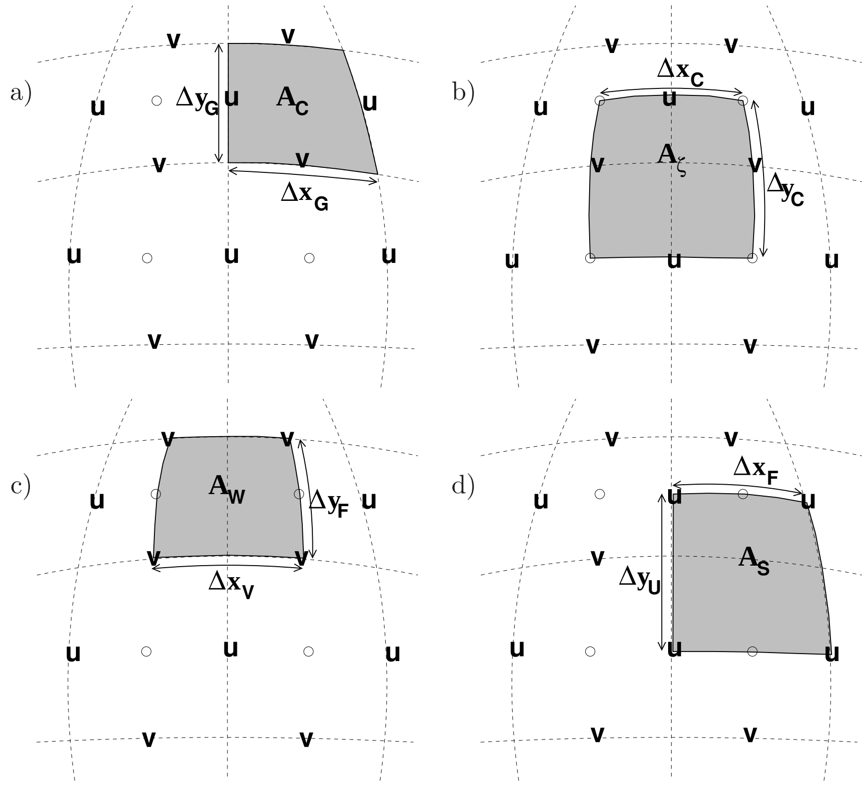


Figure 2.10: Staggering of horizontal grid descriptors (lengths and areas). The grid lines indicate the tracer cell boundaries and are the reference grid for all panels. a) The area of a tracer cell, A_c, is bordered by the lengths Δx_g and Δy_g. b) The area of a vorticity cell, A_ζ, is bordered by the lengths Δx_c and Δy_c. c) The area of a u cell, A_w, is bordered by the lengths Δx_v and Δy_f. d) The area of a v cell, A_s, is bordered by the lengths Δx_f and Δy_u.

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

S/R INI_GRID (*model/src/ini_grid.F*)
S/R INI_MASKS_ETC (*model/src/ini_masks_etc.F*)
grid data: (*model/inc/GRID.h*)

2.11.4 Horizontal grid

The model domain is decomposed into tiles and within each tile a quasi-regular grid is used. A tile is the basic unit of domain decomposition for parallelization but may be used whether parallelized or not; see section 4.2.4 for more details. Although the tiles may be patched together in an unstructured manner (i.e. irregular or non-tessilating pattern), the interior of tiles is a structured grid of quadrilateral cells. The horizontal coordinate system is orthogonal curvilinear meaning we can not necessarily treat the two horizontal directions as separable. Instead, each cell in the horizontal grid is described by the length of it's sides and it's area.

The grid information is quite general and describes any of the available coordinates systems, cartesian, spherical-polar or curvilinear. All that is necessary to distinguish between the coordinate systems is to initialize the grid data (descriptors) appropriately.

In the following, we refer to the orientation of quantities on the computational grid using geographic terminology such as points of the compass. This is purely for convenience but should not be confused with the actual geographic orientation of model quantities. Caution!

Fig. 2.10a shows the tracer cell (synonymous with the continuity cell). The length of the southern edge, Δx_g, western edge, Δy_g and surface area, A_c, presented in the vertical are stored in arrays **DXg**,

DYg and **rAc**. The "g" suffix indicates that the lengths are along the defining grid boundaries. The "c" suffix associates the quantity with the cell centers. The quantities are staggered in space and the indexing is such that **DXg(i,j)** is positioned to the south of **rAc(i,j)** and **DYg(i,j)** positioned to the west.

Fig. 2.10b shows the vorticity cell. The length of the southern edge, Δx_c, western edge, Δy_c and surface area, A_ζ, presented in the vertical are stored in arrays **DXc**, **DYc** and **rAz**. The "z" suffix indicates that the lengths are measured between the cell centers and the "ζ" suffix associates points with the vorticity points. The quantities are staggered in space and the indexing is such that **DXc(i,j)** is positioned to the north of **rAz(i,j)** and **DYc(i,j)** positioned to the east.

Fig. 2.10c shows the "u" or western (w) cell. The length of the southern edge, Δx_v, eastern edge, Δy_f and surface area, A_w, presented in the vertical are stored in arrays **DXv**, **DYf** and **rAw**. The "v" suffix indicates that the length is measured between the v-points, the "f" suffix indicates that the length is measured between the (tracer) cell faces and the "w" suffix associates points with the u-points (w stands for west). The quantities are staggered in space and the indexing is such that **DXv(i,j)** is positioned to the south of **rAw(i,j)** and **DYf(i,j)** positioned to the east.

Fig. 2.10d shows the "v" or southern (s) cell. The length of the northern edge, Δx_f, western edge, Δy_u and surface area, A_s, presented in the vertical are stored in arrays **DXf**, **DYu** and **rAs**. The "u" suffix indicates that the length is measured between the u-points, the "f" suffix indicates that the length is measured between the (tracer) cell faces and the "s" suffix associates points with the v-points (s stands for south). The quantities are staggered in space and the indexing is such that **DXf(i,j)** is positioned to the north of **rAs(i,j)** and **DYu(i,j)** positioned to the west.

S/R INI_CARTESIAN_GRID (*model/src/ini_cartesian_grid.F*)
S/R INI_SPHERICAL_POLAR_GRID (*model/src/ini_spherical_polar_grid.F*)
S/R INI_CURVILINEAR_GRID (*model/src/ini_curvilinear_grid.F*)
A_c, A_ζ, A_w, A_s: **rAc**, **rAz**, **rAw**, **rAs** (*GRID.h*)
Δx_g, Δy_g: **DXg**, **DYg** (*GRID.h*)
Δx_c, Δy_c: **DXc**, **DYc** (*GRID.h*)
Δx_f, Δy_f: **DXf**, **DYf** (*GRID.h*)
Δx_v, Δy_u: **DXv**, **DYu** (*GRID.h*)

2.11.4.1 Reciprocals of horizontal grid descriptors

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

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

S/R INI_MASKS_ETC (*model/src/ini_masks_etc.F*)
A_c^{-1}: **RECIP_Ac** (*GRID.h*)
A_ζ^{-1}: **RECIP_Az** (*GRID.h*)
A_w^{-1}: **RECIP_Aw** (*GRID.h*)
A_s^{-1}: **RECIP_As** (*GRID.h*)
Δx_g^{-1}, Δy_g^{-1}: **RECIP_DXg**, **RECIP_DYg** (*GRID.h*)
Δx_c^{-1}, Δy_c^{-1}: **RECIP_DXc**, **RECIP_DYc** (*GRID.h*)
Δx_f^{-1}, Δy_f^{-1}: **RECIP_DXf**, **RECIP_DYf** (*GRID.h*)
Δx_v^{-1}, Δy_u^{-1}: **RECIP_DXv**, **RECIP_DYu** (*GRID.h*)

2.11.4.2 Cartesian coordinates

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

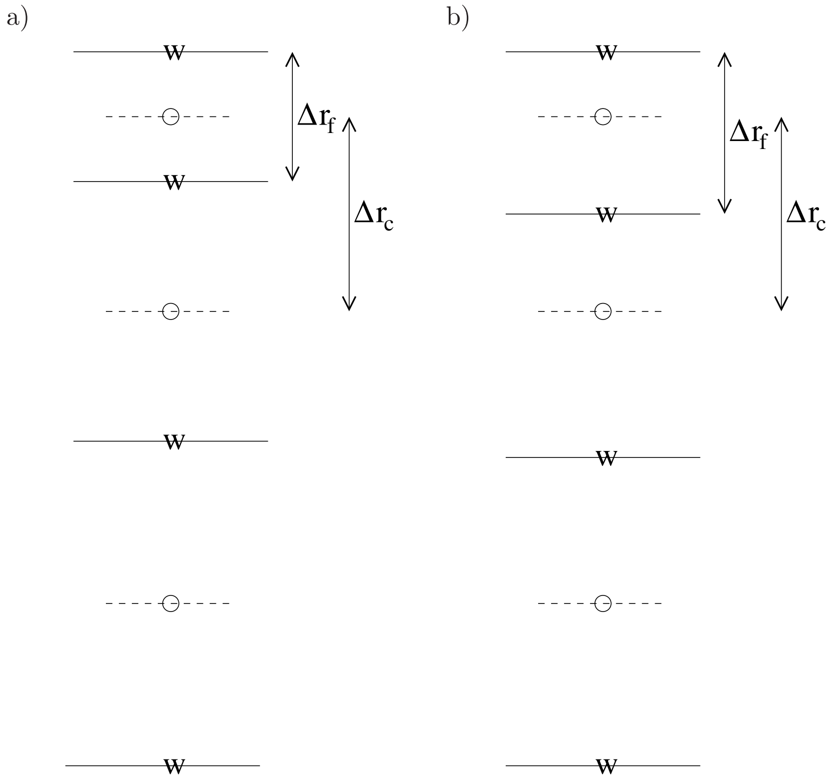


Figure 2.11: Two versions of the vertical grid. a) The cell centered approach where the interface depths are specified and the tracer points centered in between the interfaces. b) The interface centered approach where tracer levels are specified and the w-interfaces are centered in between.

2.11.4.3 Spherical-polar coordinates

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

2.11.4.4 Curvilinear coordinates

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

2.11.5 Vertical grid

As for the horizontal grid, we use the suffixes "c" and "f" to indicates faces and centers. Fig. 2.11a shows the default vertical grid used by the model. Δr_f is the difference in r (vertical coordinate) between the faces (i.e. $\Delta r_f \equiv -\delta_k r$ where the minus sign appears due to the convention that the surface layer has index $k = 1$.). Δr_f: **DRf** Δr_c: **DRc**

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

There are alternative namelist vectors **DELZ** and **DELP** which dictate whether z- or p- coordinates are to be used but we intend to phase this out since they are redundant. Caution!

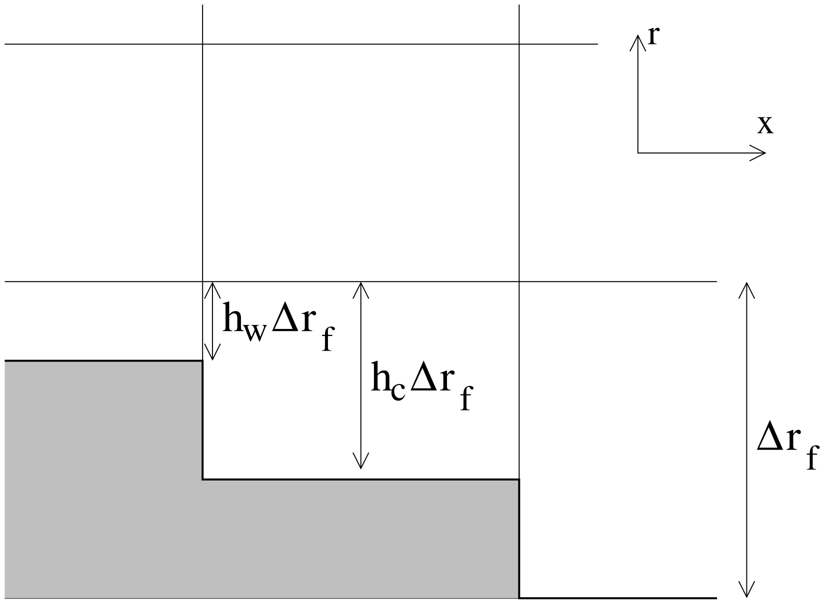


Figure 2.12: A schematic of the x-r plane showing the location of the non-dimensional fractions h_c and h_w. The physical thickness of a tracer cell is given by $h_c(i,j,k)\Delta r_f(k)$ and the physical thickness of the open side is given by $h_w(i,j,k)\Delta r_f(k)$.

The reciprocals Δr_f^{-1} and Δr_c^{-1} are pre-calculated (also in subroutine *INI_VERTICAL_GRID*). All vertical grid descriptors are stored in common blocks in *GRID.h*.

The above grid (Fig. 2.11a) is known as the cell centered approach because the tracer points are at cell centers; the cell centers are mid-way between the cell interfaces. This discretization is selected when the thickness of the levels are provided (**delR**, parameter file *data*, namelist *PARM04*) An alternative, the vertex or interface centered approach, is shown in Fig. 2.11b. Here, the interior interfaces are positioned mid-way between the tracer nodes (no longer cell centers). This approach is formally more accurate for evaluation of hydrostatic pressure and vertical advection but historically the cell centered approach has been used. An alternative form of subroutine *INI_VERTICAL_GRID* is used to select the interface centered approach This form requires to specify $Nr+1$ vertical distances **delRc** (parameter file *data*, namelist *PARM04*, e.g. *verification/ideal_2D_oce/input/data*) corresponding to surface to center, $Nr-1$ center to center, and center to bottom distances.

S/R INI_VERTICAL_GRID (*model/src/ini_vertical_grid.F*) Δr_f: **DRf** (*GRID.h*) Δr_c: **DRc** (*GRID.h*) Δr_f^{-1}: **RECIP_DRf** (*GRID.h*) Δr_c^{-1}: **RECIP_DRc** (*GRID.h*)

2.11.6 Topography: partially filled cells

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

Fig. 2.12 shows a schematic of the x-r plane indicating how the thickness of a level is determined at tracer and u points. The physical thickness of a tracer cell is given by $h_c(i,j,k)\Delta r_f(k)$ and the physical thickness of the open side is given by $h_w(i,j,k)\Delta r_f(k)$. Three 3-D descriptors h_c, h_w and h_s are used to describe the geometry: **hFacC**, **hFacW** and **hFacS** respectively. These are calculated in subroutine *INI_MASKS_ETC* along with there reciprocals **RECIP_hFacC**, **RECIP_hFacW** and **RECIP_hFacS**. h_c: **hFacC** h_w: **hFacW** h_s: **hFacS**

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

S/R INI_MASKS_ETC (*model/src/ini_masks_etc.F*)
h_c: **hFacC** (*GRID.h*)
h_w: **hFacW** (*GRID.h*)
h_s: **hFacS** (*GRID.h*)
h_c^{-1}: **RECIP_hFacC** (*GRID.h*)
h_w^{-1}: **RECIP_hFacW** (*GRID.h*)
h_s^{-1}: **RECIP_hFacS** (*GRID.h*)

2.12 Continuity and horizontal pressure gradient terms

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

$$\partial_t u + \frac{1}{\Delta x_c}\delta_i \left.\frac{\partial \Phi}{\partial r}\right|_s \eta + \frac{\epsilon_{nh}}{\Delta x_c}\delta_i \Phi'_{nh} = G_u - \frac{1}{\Delta x_c}\delta_i \Phi'_h \tag{2.87}$$

$$\partial_t v + \frac{1}{\Delta y_c}\delta_j \left.\frac{\partial \Phi}{\partial r}\right|_s \eta + \frac{\epsilon_{nh}}{\Delta y_c}\delta_j \Phi'_{nh} = G_v - \frac{1}{\Delta y_c}\delta_j \Phi'_h \tag{2.88}$$

$$\epsilon_{nh}\left(\partial_t w + \frac{1}{\Delta r_c}\delta_k \Phi'_{nh}\right) = \epsilon_{nh} G_w + \overline{b}^k - \frac{1}{\Delta r_c}\delta_k \Phi'_h \tag{2.89}$$

$$\delta_i \Delta y_g \Delta r_f h_w u + \delta_j \Delta x_g \Delta r_f h_s v + \delta_k \mathcal{A}_c w = \mathcal{A}_c \delta_k (P-E)_{r=0} \tag{2.90}$$

where the continuity equation has been most naturally discretized by staggering the three components of velocity as shown in Fig. 2.9. The grid lengths Δx_c and Δy_c are the lengths between tracer points (cell centers). The grid lengths Δx_g, Δy_g are the grid lengths between cell corners. Δr_f and Δr_c are the distance (in units of r) between level interfaces (w-level) and level centers (tracer level). The surface area presented in the vertical is denoted $\mathcal{A}_c$. The factors h_w and h_s are non-dimensional fractions (between 0 and 1) that represent the fraction cell depth that is "open" for fluid flow. h_w: **hFacW**

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

$$\mathcal{A}_c \partial_t \eta + \delta_i \sum_k \Delta y_g \Delta r_f h_w u + \delta_j \sum_k \Delta x_g \Delta r_f h_s v = \mathcal{A}_c (P-E)_{r=0} \tag{2.91}$$

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

2.13 Hydrostatic balance

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

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

$$\epsilon_{nh}\partial_t w + g\overline{\rho'}^k + \frac{1}{\Delta z}\delta_k \Phi'_h = \ldots \tag{2.92}$$

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

$$\overline{\theta'}^k + \frac{1}{\Delta\Pi}\delta_k\Phi'_h = 0 \tag{2.93}$$

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

The difference in approach between ocean and atmosphere occurs because of the direct use of the ideal gas equation in forming the potential energy conversion term $\alpha\omega$. The form of these conversion terms is discussed at length in *Adcroft* [2002].

Because of the different representation of hydrostatic balance between ocean and atmosphere there is no elegant way to represent both systems using an arbitrary coordinate.

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

The calculations are made in the subroutine *CALC_PHI_HYD*. Inside this routine, one of other of the atmospheric/oceanic form is selected based on the string variable **buoyancyRelation**.

2.14 Flux-form momentum equations

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

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

$$G_u = G_u^{adv} + G_u^{cor} + G_u^{h-diss} + G_u^{v-diss} + G_u^{metric} + G_u^{nh-metric} \tag{2.94}$$

$$G_v = G_v^{adv} + G_v^{cor} + G_v^{h-diss} + G_v^{v-diss} + G_v^{metric} + G_v^{nh-metric} \tag{2.95}$$

$$G_w = G_w^{adv} + G_w^{cor} + G_w^{h-diss} + G_w^{v-diss} + G_w^{metric} + G_w^{nh-metric} \tag{2.96}$$

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

These terms are calculated in routines called from subroutine *MOM_FLUXFORM* a collected into the global arrays **Gu**, **Gv**, and **Gw**.

S/R MOM_FLUXFORM (*pkg/mom_fluxform/mom_fluxform.F*)
G_u: **Gu** (*DYNVARS.h*)
G_v: **Gv** (*DYNVARS.h*)
G_w: **Gw** (*DYNVARS.h*)

2.14.1 Advection of momentum

The advective operator is second order accurate in space:

$$\mathcal{A}_w \Delta r_f h_w G_u^{adv} = \delta_i \overline{U}^i \overline{u}^i + \delta_j \overline{V}^i \overline{u}^j + \delta_k \overline{W}^i \overline{u}^k \tag{2.97}$$

$$\mathcal{A}_s \Delta r_f h_s G_v^{adv} = \delta_i \overline{U}^j \overline{v}^i + \delta_j \overline{V}^j \overline{v}^j + \delta_k \overline{W}^j \overline{v}^k \tag{2.98}$$

$$\mathcal{A}_c \Delta r_c G_w^{adv} = \delta_i \overline{U}^k \overline{w}^i + \delta_j \overline{V}^k \overline{w}^j + \delta_k \overline{W}^k \overline{w}^k \tag{2.99}$$

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

$$U = \Delta y_g \Delta r_f h_w u \tag{2.100}$$

$$V = \Delta x_g \Delta r_f h_s v \tag{2.101}$$

$$W = \mathcal{A}_c w \tag{2.102}$$

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

S/R MOM_U_ADV_UU (*mom_u_adv_uu.F*) *S/R MOM_U_ADV_VU* (*mom_u_adv_vu.F*) *S/R MOM_U_ADV_WU* (*mom_u_adv_wu.F*) *S/R MOM_U_ADV_UV* (*mom_u_adv_uv.F*) *S/R MOM_U_ADV_VV* (*mom_u_adv_vv.F*) *S/R MOM_U_ADV_WV* (*mom_u_adv_wv.F*) *uu*, *uv*, *vu*, *vv*: **aF** (local to *mom_fluxform.F*)

2.14.2 Coriolis terms

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

$$\mathcal{A}_w \Delta r_f h_w G_u^{Cor} = \overline{f \mathcal{A}_c \Delta r_f h_c \overline{v}^j}^i - \epsilon_{nh} \overline{f' \mathcal{A}_c \Delta r_f h_c \overline{w}^k}^i \tag{2.103}$$

$$\mathcal{A}_s \Delta r_f h_s G_v^{Cor} = -\overline{f \mathcal{A}_c \Delta r_f h_c \overline{u}^i}^j \tag{2.104}$$

$$\mathcal{A}_c \Delta r_c G_w^{Cor} = \epsilon_{nh} \overline{f' \mathcal{A}_c \Delta r_f h_c \overline{u}^i}^k \tag{2.105}$$

where the Coriolis parameters f and f' are defined:

$$f = 2\Omega \sin\varphi \tag{2.106}$$

$$f' = 2\Omega \cos\varphi \tag{2.107}$$

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

$$f = f_o + \beta y \tag{2.108}$$

$$f' = 0 \tag{2.109}$$

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

$$G_u^{Cor} = f_u \overline{v}^{ji} - \epsilon_{nh} f'_u \overline{w}^{ik} \tag{2.110}$$

$$G_v^{Cor} = -f_v \overline{u}^{ij} \tag{2.111}$$

$$G_w^{Cor} = \epsilon_{nh} f'_w \overline{u}^{ik} \tag{2.112}$$

where the subscripts on f and f' indicate evaluation of the Coriolis parameters at the appropriate points in space. The above discretization does *not* conserve anything, especially energy and for historical reasons is the default for the code. A flag controls this discretization: set run-time logical **useEnergyConservingCoriolis** to *true* which otherwise defaults to *false*.

Need to change
code to match t

S/R MOM_CDSCHEME (*mom_cdscheme.F*) *S/R MOM_U_CORIOLIS* (*mom_u_coriolis.F*) *S/R MOM_V_CORIOLIS* (*mom_v_coriolis.F*) G_u^{Cor}, G_v^{Cor}: **cF** (local to *mom_fluxform.F*)

2.14.3 Curvature metric terms

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

$$\mathcal{A}_w \Delta r_f h_w G_u^{metric} = \overline{\frac{\overline{u}^i}{a} \tan\varphi \mathcal{A}_c \Delta r_f h_c \overline{v}^j}^i \tag{2.113}$$

$$\mathcal{A}_s \Delta r_f h_s G_v^{metric} = -\overline{\frac{\overline{u}^i}{a} \tan\varphi \mathcal{A}_c \Delta r_f h_c \overline{u}^i}^j \tag{2.114}$$

$$G_w^{metric} = 0 \tag{2.115}$$

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

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

$$G_u^{metric} = \frac{u\overline{v}^{ij}}{a}\tan\varphi \tag{2.116}$$

$$G_v^{metric} = \frac{\overline{u}^{ij}\overline{u}^{ij}}{a}\tan\varphi \tag{2.117}$$

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

S/R MOM_U_METRIC_SPHERE (*mom_u_metric_sphere.F*)
S/R MOM_V_METRIC_SPHERE (*mom_v_metric_sphere.F*)
G_u^{metric}, G_v^{metric}: **mT** (local to *mom_fluxform.F*)

2.14.4 Non-hydrostatic metric terms

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

$$\mathcal{A}_w \Delta r_f h_w G_u^{metric} = -\overline{\frac{\overline{u}^i\overline{w}^k}{a}\mathcal{A}_c\Delta r_f h_c}^{i} \tag{2.118}$$

$$\mathcal{A}_s \Delta r_f h_s G_v^{metric} = -\overline{\frac{\overline{v}^j\overline{w}^k}{a}\mathcal{A}_c\Delta r_f h_c}^{j} \tag{2.119}$$

$$\mathcal{A}_c \Delta r_c G_w^{metric} = \overline{\frac{\overline{u}^{i\,2}+\overline{v}^{j\,2}}{a}\mathcal{A}_c\Delta r_f h_c}^{k} \tag{2.120}$$

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

$$G_u^{metric} = -\frac{u}{a}\overline{w}^{ik} \tag{2.121}$$

$$G_v^{metric} = -\frac{v}{a}\overline{w}^{jk} \tag{2.122}$$

$$G_w^{metric} = \frac{1}{a}(\overline{u}^{ik\,2}+\overline{v}^{jk\,2}) \tag{2.123}$$

S/R MOM_U_METRIC_NH (*mom_u_metric_nh.F*)
S/R MOM_V_METRIC_NH (*mom_v_metric_nh.F*)
G_u^{metric}, G_v^{metric}: **mT** (local to *mom_fluxform.F*)

2.14.5 Lateral dissipation

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

$$\mathcal{A}_w \Delta r_f h_w G_u^{h-diss} = \delta_i \Delta y_f \Delta r_f h_c \tau_{11} + \delta_j \Delta x_v \Delta r_f h_\zeta \tau_{12} \tag{2.124}$$

$$\mathcal{A}_s \Delta r_f h_s G_v^{h-diss} = \delta_i \Delta y_u \Delta r_f h_\zeta \tau_{21} + \delta_j \Delta x_f \Delta r_f h_c \tau_{22} \tag{2.125}$$

tress defini-

The lateral viscous stresses are discretized:

$$\tau_{11} = A_h c_{11\Delta}(\varphi)\frac{1}{\Delta x_f}\delta_i u - A_4 c_{11\Delta^2}(\varphi)\frac{1}{\Delta x_f}\delta_i \nabla^2 u \quad (2.126)$$

$$\tau_{12} = A_h c_{12\Delta}(\varphi)\frac{1}{\Delta y_u}\delta_j u - A_4 c_{12\Delta^2}(\varphi)\frac{1}{\Delta y_u}\delta_j \nabla^2 u \quad (2.127)$$

$$\tau_{21} = A_h c_{21\Delta}(\varphi)\frac{1}{\Delta x_v}\delta_i v - A_4 c_{21\Delta^2}(\varphi)\frac{1}{\Delta x_v}\delta_i \nabla^2 v \quad (2.128)$$

$$\tau_{22} = A_h c_{22\Delta}(\varphi)\frac{1}{\Delta y_f}\delta_j v - A_4 c_{22\Delta^2}(\varphi)\frac{1}{\Delta y_f}\delta_j \nabla^2 v \quad (2.129)$$

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

Need to tidy u controlling this

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

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

S/R MOM_U_XVISCFLUX (*mom_u_xviscflux.F*)
S/R MOM_U_YVISCFLUX (*mom_u_yviscflux.F*)
S/R MOM_V_XVISCFLUX (*mom_v_xviscflux.F*)
S/R MOM_V_YVISCFLUX (*mom_v_yviscflux.F*)
τ_{11}, τ_{12}, τ_{21}, τ_{22}: **vF**, **v4F** (local to *mom_fluxform.F*)

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

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

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

$$G_u^{side-drag} = \frac{4}{\Delta z_f}\overline{(1-h_\zeta)\frac{\Delta x_v}{\Delta y_u}}^j \left(A_h c_{12\Delta}(\varphi) u - A_4 c_{12\Delta^2}(\varphi)\nabla^2 u\right) \quad (2.130)$$

$$G_v^{side-drag} = \frac{4}{\Delta z_f}\overline{(1-h_\zeta)\frac{\Delta y_u}{\Delta x_v}}^i \left(A_h c_{21\Delta}(\varphi) v - A_4 c_{21\Delta^2}(\varphi)\nabla^2 v\right) \quad (2.131)$$

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

S/R MOM_U_SIDEDRAG (*mom_u_sidedrag.F*)
S/R MOM_V_SIDEDRAG (*mom_v_sidedrag.F*)
$G_u^{side-drag}$, $G_v^{side-drag}$: **vF** (local to *mom_fluxform.F*)

2.14.6 Vertical dissipation

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

$$G_u^{v-diss} = \frac{1}{\Delta r_f h_w} \delta_k \tau_{13} \quad (2.132)$$

$$G_v^{v-diss} = \frac{1}{\Delta r_f h_s} \delta_k \tau_{23} \quad (2.133)$$

$$G_w^{v-diss} = \epsilon_{nh} \frac{1}{\Delta r_f h_d} \delta_k \tau_{33} \quad (2.134)$$

represents the general discrete form of the vertical dissipation terms.

In the interior the vertical stresses are discretized:

$$\tau_{13} = A_v \frac{1}{\Delta r_c} \delta_k u \quad (2.135)$$

$$\tau_{23} = A_v \frac{1}{\Delta r_c} \delta_k v \quad (2.136)$$

$$\tau_{33} = A_v \frac{1}{\Delta r_f} \delta_k w \quad (2.137)$$

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

S/R MOM_U_RVISCLFUX (*mom_u_rviscflux.F*) *S/R MOM_V_RVISCLFUX* (*mom_v_rviscflux.F*) τ_{13}: **urf** (local to *mom_fluxform.F*) τ_{23}: **vrf** (local to *mom_fluxform.F*)

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

$$\tau_{13}^{bottom-drag} = \left(2A_v \frac{1}{\Delta r_c} + r_b + C_d \sqrt{\overline{2KE}^i} \right) u \quad (2.138)$$

$$\tau_{23}^{bottom-drag} = \left(2A_v \frac{1}{\Delta r_c} + r_b + C_d \sqrt{\overline{2KE}^j} \right) v \quad (2.139)$$

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

S/R MOM_U_BOTTOMDRAG (*mom_u_bottomdrag.F*) *S/R MOM_V_BOTTOMDRAG* (*mom_v_bottomdrag.F*) $\tau_{13}^{bottom-drag}/\Delta r_f$, $\tau_{23}^{bottom-drag}/\Delta r_f$: **vf** (local to *mom_fluxform.F*)

2.14.7 Derivation of discrete energy conservation

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

$$KE = \frac{1}{2} \left(\overline{u^2}^i + \overline{v^2}^j + \epsilon_{nh} \overline{w^2}^k \right) \quad (2.140)$$

2.14.8 Mom Diagnostics

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

```
------------------------------------------------------------------------
VISCAHZ | 15 |SZ      MR      |m^2/s          |Harmonic Visc Coefficient (m2/s) (Zeta Pt)
VISCA4Z | 15 |SZ      MR      |m^4/s          |Biharmonic Visc Coefficient (m4/s) (Zeta Pt)
VISCAHD | 15 |SM      MR      |m^2/s          |Harmonic Viscosity Coefficient (m2/s) (Div Pt)
VISCA4D | 15 |SM      MR      |m^4/s          |Biharmonic Viscosity Coefficient (m4/s) (Div Pt)
VAHZMAX | 15 |SZ      MR      |m^2/s          |CFL-MAX Harm Visc Coefficient (m2/s) (Zeta Pt)
VA4ZMAX | 15 |SZ      MR      |m^4/s          |CFL-MAX Biharm Visc Coefficient (m4/s) (Zeta Pt)
VAHDMAX | 15 |SM      MR      |m^2/s          |CFL-MAX Harm Visc Coefficient (m2/s) (Div Pt)
VA4DMAX | 15 |SM      MR      |m^4/s          |CFL-MAX Biharm Visc Coefficient (m4/s) (Div Pt)
VAHZMIN | 15 |SZ      MR      |m^2/s          |RE-MIN Harm Visc Coefficient (m2/s) (Zeta Pt)
VA4ZMIN | 15 |SZ      MR      |m^4/s          |RE-MIN Biharm Visc Coefficient (m4/s) (Zeta Pt)
VAHDMIN | 15 |SM      MR      |m^2/s          |RE-MIN Harm Visc Coefficient (m2/s) (Div Pt)
VA4DMIN | 15 |SM      MR      |m^4/s          |RE-MIN Biharm Visc Coefficient (m4/s) (Div Pt)
VAHZLTH | 15 |SZ      MR      |m^2/s          |Leith Harm Visc Coefficient (m2/s) (Zeta Pt)
VA4ZLTH | 15 |SZ      MR      |m^4/s          |Leith Biharm Visc Coefficient (m4/s) (Zeta Pt)
VAHDLTH | 15 |SM      MR      |m^2/s          |Leith Harm Visc Coefficient (m2/s) (Div Pt)
VA4DLTH | 15 |SM      MR      |m^4/s          |Leith Biharm Visc Coefficient (m4/s) (Div Pt)
VAHZLTHD| 15 |SZ      MR      |m^2/s          |LeithD Harm Visc Coefficient (m2/s) (Zeta Pt)
VA4ZLTHD| 15 |SZ      MR      |m^4/s          |LeithD Biharm Visc Coefficient (m4/s) (Zeta Pt)
VAHDLTHD| 15 |SM      MR      |m^2/s          |LeithD Harm Visc Coefficient (m2/s) (Div Pt)
VA4DLTHD| 15 |SM      MR      |m^4/s          |LeithD Biharm Visc Coefficient (m4/s) (Div Pt)
VAHZSMAG| 15 |SZ      MR      |m^2/s          |Smagorinsky Harm Visc Coefficient (m2/s) (Zeta Pt)
VA4ZSMAG| 15 |SZ      MR      |m^4/s          |Smagorinsky Biharm Visc Coeff. (m4/s) (Zeta Pt)
VAHDSMAG| 15 |SM      MR      |m^2/s          |Smagorinsky Harm Visc Coefficient (m2/s) (Div Pt)
VA4DSMAG| 15 |SM      MR      |m^4/s          |Smagorinsky Biharm Visc Coeff. (m4/s) (Div Pt)
momKE   | 15 |SM      MR      |m^2/s^2        |Kinetic Energy (in momentum Eq.)
momHDiv | 15 |SM      MR      |s^-1           |Horizontal Divergence (in momentum Eq.)
momVort3| 15 |SZ      MR      |s^-1           |3rd component (vertical) of Vorticity
Strain  | 15 |SZ      MR      |s^-1           |Horizontal Strain of Horizontal Velocities
Tension | 15 |SM      MR      |s^-1           |Horizontal Tension of Horizontal Velocities
UBotDrag| 15 |UU   129MR      |m/s^2          |U momentum tendency from Bottom Drag
VBotDrag| 15 |VV   128MR      |m/s^2          |V momentum tendency from Bottom Drag
USidDrag| 15 |UU   131MR      |m/s^2          |U momentum tendency from Side Drag
VSidDrag| 15 |VV   130MR      |m/s^2          |V momentum tendency from Side Drag
Um_Diss | 15 |UU   133MR      |m/s^2          |U momentum tendency from Dissipation
Vm_Diss | 15 |VV   132MR      |m/s^2          |V momentum tendency from Dissipation
Um_Advec| 15 |UU   135MR      |m/s^2          |U momentum tendency from Advection terms
Vm_Advec| 15 |VV   134MR      |m/s^2          |V momentum tendency from Advection terms
Um_Cori | 15 |UU   137MR      |m/s^2          |U momentum tendency from Coriolis term
Vm_Cori | 15 |VV   136MR      |m/s^2          |V momentum tendency from Coriolis term
Um_Ext  | 15 |UU   137MR      |m/s^2          |U momentum tendency from external forcing
Vm_Ext  | 15 |VV   138MR      |m/s^2          |V momentum tendency from external forcing
Um_AdvZ3| 15 |UU   141MR      |m/s^2          |U momentum tendency from Vorticity Advection
Vm_AdvZ3| 15 |VV   140MR      |m/s^2          |V momentum tendency from Vorticity Advection
Um_AdvRe| 15 |UU   143MR      |m/s^2          |U momentum tendency from vertical Advection (Explicit
Vm_AdvRe| 15 |VV   142MR      |m/s^2          |V momentum tendency from vertical Advection (Explicit
ADVx_Um | 15 |UM   145MR      |m^4/s^2        |Zonal      Advective Flux of U momentum
ADVy_Um | 15 |VZ   144MR      |m^4/s^2        |Meridional Advective Flux of U momentum
ADVrE_Um| 15 |WU      LR      |m^4/s^2        |Vertical   Advective Flux of U momentum (Explicit par
ADVx_Vm | 15 |UZ   148MR      |m^4/s^2        |Zonal      Advective Flux of V momentum
ADVy_Vm | 15 |VM   147MR      |m^4/s^2        |Meridional Advective Flux of V momentum
ADVrE_Vm| 15 |WV      LR      |m^4/s^2        |Vertical   Advective Flux of V momentum (Explicit par
VISCx_Um| 15 |UM   151MR      |m^4/s^2        |Zonal      Viscous Flux of U momentum
VISCy_Um| 15 |VZ   150MR      |m^4/s^2        |Meridional Viscous Flux of U momentum
VISrE_Um| 15 |WU      LR      |m^4/s^2        |Vertical   Viscous Flux of U momentum (Explicit part)
VISrI_Um| 15 |WU      LR      |m^4/s^2        |Vertical   Viscous Flux of U momentum (Implicit part)
```

```
VISCx_Vm| 15 |UZ   155MR      |m^4/s^2         |Zonal      Viscous Flux of V momentum
VISCy_Vm| 15 |VM   154MR      |m^4/s^2         |Meridional Viscous Flux of V momentum
VISrE_Vm| 15 |WV      LR      |m^4/s^2         |Vertical   Viscous Flux of V momentum (Explicit part)
VISrI_Vm| 15 |WV      LR      |m^4/s^2         |Vertical   Viscous Flux of V momentum (Implicit part)
```

2.15 Vector invariant momentum equations

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

The non-hydrostatic vector invariant equations read:

$$\partial_t \vec{v} + (2\vec{\Omega} + \vec{\zeta}) \wedge \vec{v} - b\hat{r} + \vec{\nabla} B = \vec{\nabla} \cdot \vec{\tau} \tag{2.141}$$

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

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

$$G_u = G_u^{fv} + G_u^{\zeta_3 v} + G_u^{\zeta_2 w} + G_u^{\partial_x B} + G_u^{\partial_z \tau^x} + G_u^{h-dissip} + G_u^{v-dissip} \tag{2.142}$$

$$G_v = G_v^{fu} + G_v^{\zeta_3 u} + G_v^{\zeta_1 w} + G_v^{\partial_y B} + G_v^{\partial_z \tau^y} + G_v^{h-dissip} + G_v^{v-dissip} \tag{2.143}$$

$$G_w = G_w^{fu} + G_w^{\zeta_1 v} + G_w^{\zeta_2 u} + G_w^{\partial_z B} + G_w^{h-dissip} + G_w^{v-dissip} \tag{2.144}$$

S/R MOM_VECINV (*pkg/mom_vecinv/mom_vecinv.F*)
G_u: **Gu** (*DYNVARS.h*)
G_v: **Gv** (*DYNVARS.h*)
G_w: **Gw** (*DYNVARS.h*)

2.15.1 Relative vorticity

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

Relative vorticity is defined:

$$\zeta_3 = \frac{\Gamma}{A_\zeta} = \frac{1}{\mathcal{A}_\zeta}(\delta_i \Delta y_c v - \delta_j \Delta x_c u) \tag{2.145}$$

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

S/R MOM_VI_CALC_RELVORT3 (*mom_vi_calc_relvort3.F*)
ζ_3: **vort3** (local to *mom_vecinv.F*)

2.15.2 Kinetic energy

The kinetic energy, denoted KE, is defined:

$$KE = \frac{1}{2}(\overline{u^2}^i + \overline{v^2}^j + \epsilon_{nh} \overline{w^2}^k) \tag{2.146}$$

S/R MOM_VI_CALC_KE (*mom_vi_calc_ke.F*)
KE: **KE** (local to *mom_vecinv.F*)

2.15.3 Coriolis terms

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

$$G_u^{fv} = \frac{1}{\Delta x_c} \overline{\frac{f}{h_\zeta}}^j \overline{\overline{\Delta x_g h_s v}^j}^i \quad (2.147)$$

$$G_v^{fu} = -\frac{1}{\Delta y_c} \overline{\frac{f}{h_\zeta}}^i \overline{\overline{\Delta y_g h_w u}^i}^j \quad (2.148)$$

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

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

$$G_u^{\zeta_3 v} = \frac{1}{\Delta x_c} \overline{\frac{\zeta_3}{h_\zeta}}^j \overline{\overline{\Delta x_g h_s v}^j}^i \quad (2.149)$$

$$G_v^{\zeta_3 u} = -\frac{1}{\Delta y_c} \overline{\frac{\zeta_3}{h_\zeta}}^i \overline{\overline{\Delta y_g h_w u}^i}^j \quad (2.150)$$

ζ_3: **vort3**

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

$$G_u^{fv} + G_u^{\zeta_3 v} = \frac{1}{\Delta x_c} \overline{\frac{f+\zeta_3}{h_\zeta}}^j \overline{\overline{\Delta x_g h_s v}^j}^i \quad (2.151)$$

$$G_v^{fu} + G_v^{\zeta_3 u} = -\frac{1}{\Delta y_c} \overline{\frac{f+\zeta_3}{h_\zeta}}^i \overline{\overline{\Delta y_g h_w u}^i}^j \quad (2.152)$$

Run-time contr added for these

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

S/R MOM_VI_CORIOLIS (*mom_vi_coriolis.F*)
S/R MOM_VI_U_CORIOLIS (*mom_vi_u_coriolis.F*)
S/R MOM_VI_V_CORIOLIS (*mom_vi_v_coriolis.F*)
G_u^{fv}, $G_u^{\zeta_3 v}$: **uCf** (local to *mom_vecinv.F*)
G_v^{fu}, $G_v^{\zeta_3 u}$: **vCf** (local to *mom_vecinv.F*)

2.15.4 Shear terms

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

N-H terms ha tried!

$$G_u^{\zeta_2 w} = \frac{1}{\mathcal{A}_w \Delta r_f h_w} \overline{\overline{\mathcal{A}_c w}^i (\delta_k u - \epsilon_{nh} \delta_j w)}^k \quad (2.153)$$

$$G_v^{\zeta_1 w} = \frac{1}{\mathcal{A}_s \Delta r_f h_s} \overline{\overline{\mathcal{A}_c w}^i (\delta_k u - \epsilon_{nh} \delta_j w)}^k \quad (2.154)$$

S/R MOM_VI_U_VERTSHEAR (*mom_vi_u_vertshear.F*)
S/R MOM_VI_V_VERTSHEAR (*mom_vi_v_vertshear.F*)
$G_u^{\zeta_2 w}$: **uCf** (local to *mom_vecinv.F*)
$G_v^{\zeta_1 w}$: **vCf** (local to *mom_vecinv.F*)

2.15.5 Gradient of Bernoulli function

$$G_u^{\partial_x B} = \frac{1}{\Delta x_c} \delta_i (\phi' + KE) \quad (2.155)$$

$$G_v^{\partial_y B} = \frac{1}{\Delta x_y} \delta_j (\phi' + KE) \quad (2.156)$$

S/R MOM_VI_U_GRAD_KE (*mom_vi_u_grad_ke.F*)
S/R MOM_VI_V_GRAD_KE (*mom_vi_v_grad_ke.F*)
$G_u^{\partial_x KE}$: **uCf** (local to *mom_vecinv.F*)
$G_v^{\partial_y KE}$: **vCf** (local to *mom_vecinv.F*)

2.15.6 Horizontal divergence

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

$$D = \frac{1}{\mathcal{A}_c h_c}(\delta_i \Delta y_g h_w u + \delta_j \Delta x_g h_s v) \tag{2.157}$$

S/R MOM_VI_CALC_HDIV (*mom_vi_calc_hdiv.F*)
D: **hDiv** (local to *mom_vecinv.F*)

2.15.7 Horizontal dissipation

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

$$G_u^{h-dissip} = \frac{1}{\Delta x_c}\delta_i(A_D D - A_{D4}D^*) - \frac{1}{\Delta y_u h_w}\delta_j h_\zeta(A_\zeta \zeta - A_{\zeta 4}\zeta^*) \tag{2.158}$$

$$G_v^{h-dissip} = \frac{1}{\Delta x_v h_s}\delta_i h_\zeta(A_\zeta \zeta - A_\zeta \zeta^*) + \frac{1}{\Delta y_c}\delta_j(A_D D - A_{D4}D^*) \tag{2.159}$$

where

$$D^* = \frac{1}{\mathcal{A}_c h_c}(\delta_i \Delta y_g h_w \nabla^2 u + \delta_j \Delta x_g h_s \nabla^2 v) \tag{2.160}$$

$$\zeta^* = \frac{1}{\mathcal{A}_\zeta}(\delta_i \Delta y_c \nabla^2 v - \delta_j \Delta x_c \nabla^2 u) \tag{2.161}$$

S/R MOM_VI_HDISSIP (*mom_vi_hdissip.F*)
$G_u^{h-dissip}$: **uDiss** (local to *mom_vecinv.F*)
$G_v^{h-dissip}$: **vDiss** (local to *mom_vecinv.F*)

2.15.8 Vertical dissipation

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

$$G_u^{v-diss} = \frac{1}{\Delta r_f h_w}\delta_k \tau_{13} \tag{2.162}$$

$$G_v^{v-diss} = \frac{1}{\Delta r_f h_s}\delta_k \tau_{23} \tag{2.163}$$

represents the general discrete form of the vertical dissipation terms.

In the interior the vertical stresses are discretized:

$$\tau_{13} = A_v \frac{1}{\Delta r_c}\delta_k u \tag{2.164}$$

$$\tau_{23} = A_v \frac{1}{\Delta r_c}\delta_k v \tag{2.165}$$

S/R MOM_U_RVISCLFUX (*mom_u_rviscflux.F*)
S/R MOM_V_RVISCLFUX (*mom_v_rviscflux.F*)
τ_{13}: **urf** (local to *mom_vecinv.F*)
τ_{23}: **vrf** (local to *mom_vecinv.F*)

2.16 Tracer equations

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

2.16.1 Time-stepping of tracers: ABII

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

$$\partial_t \tau + G_{adv}^\tau = G_{diff}^\tau + G_{forc}^\tau \tag{2.166}$$

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

$$G_{adv}^\tau = \partial_x u\tau + \partial_y v\tau + \partial_r w\tau - \tau\nabla\cdot\mathbf{v} \tag{2.167}$$

$$G_{diff}^\tau = \nabla\cdot\mathbf{K}\nabla\tau \tag{2.168}$$

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

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

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

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

S/R GAD_CALC_RHS (*pkg/generic_advdiff/gad_calc_rhs.F*)
τ: **tracer** (argument)
$G^{(n)}$: **gTracer** (argument)
F_r: **fVerT** (argument)

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

$$G^{(n+1/2)} = (\frac{3}{2}+\epsilon)G^{(n)} - (\frac{1}{2}+\epsilon)G^{(n-1)} \tag{2.169}$$

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

S/R ADAMS_BASHFORTH2 (*model/src/adams_bashforth2.F*)
$G^{(n+1/2)}$: **gTracer** (argument on exit)
$G^{(n)}$: **gTracer** (argument on entry)
$G^{(n-1)}$: **gTrNm1** (argument)
ϵ: **ABeps** (PARAMS.h)

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

$$\tau^{(n+1)} = \tau^{(n)} + \Delta t G^{(n+1/2)} \tag{2.170}$$

Δt: **deltaTtra**

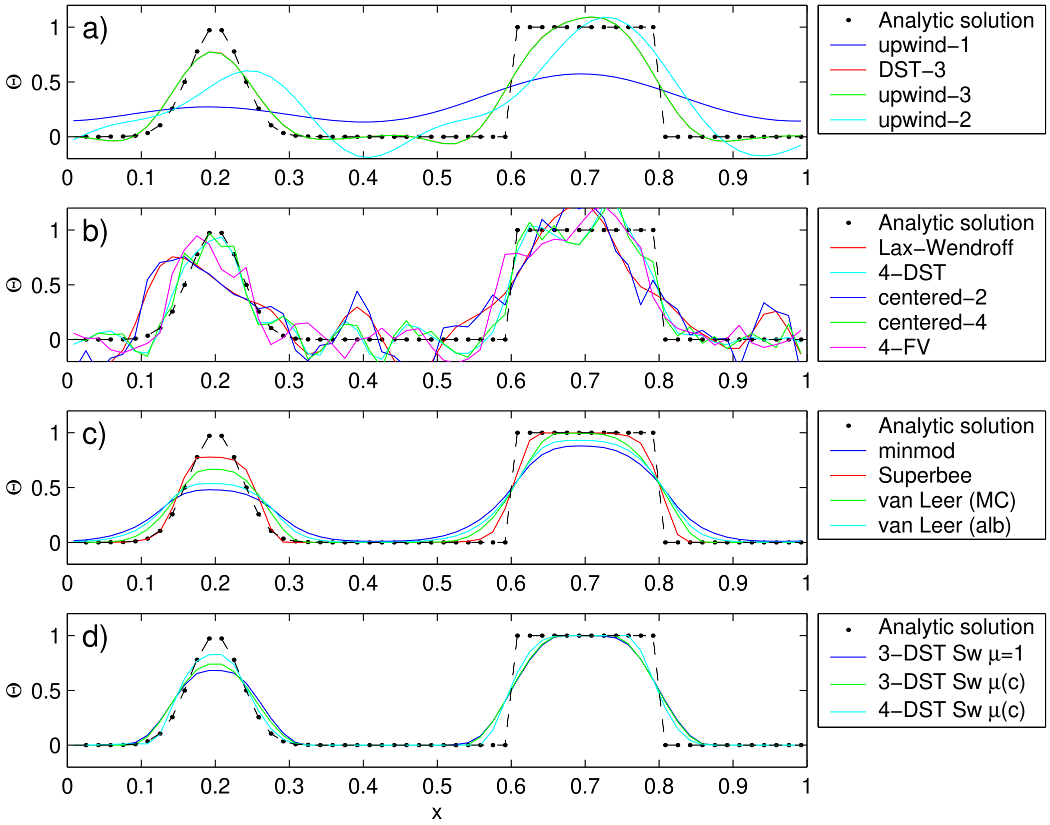


Figure 2.13: Comparison of 1-D advection schemes. Courant number is 0.05 with 60 points and solutions are shown for T=1 (one complete period). a) Shows the upwind biased schemes; first order upwind, DST3, third order upwind and second order upwind. b) Shows the centered schemes; Lax-Wendroff, DST4, centered second order, centered fourth order and finite volume fourth order. c) Shows the second order flux limiters: minmod, Superbee, MC limiter and the van Leer limiter. d) Shows the DST3 method with flux limiters due to Sweby with $\mu = 1$, $\mu = c/(1-c)$ and a fourth order DST method with Sweby limiter, $\mu = c/(1-c)$.

S/R TIMESTEP_TRACER (*model/src/timestep_tracer.F*)
$\tau^{(n+1)}$: **gTracer** (argument on exit)
$\tau^{(n)}$: **tracer** (argument on entry)
$G^{(n+1/2)}$: **gTracer** (argument)
Δt: **deltaTtracer** (PARAMS.h)

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

2.17 Linear advection schemes

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

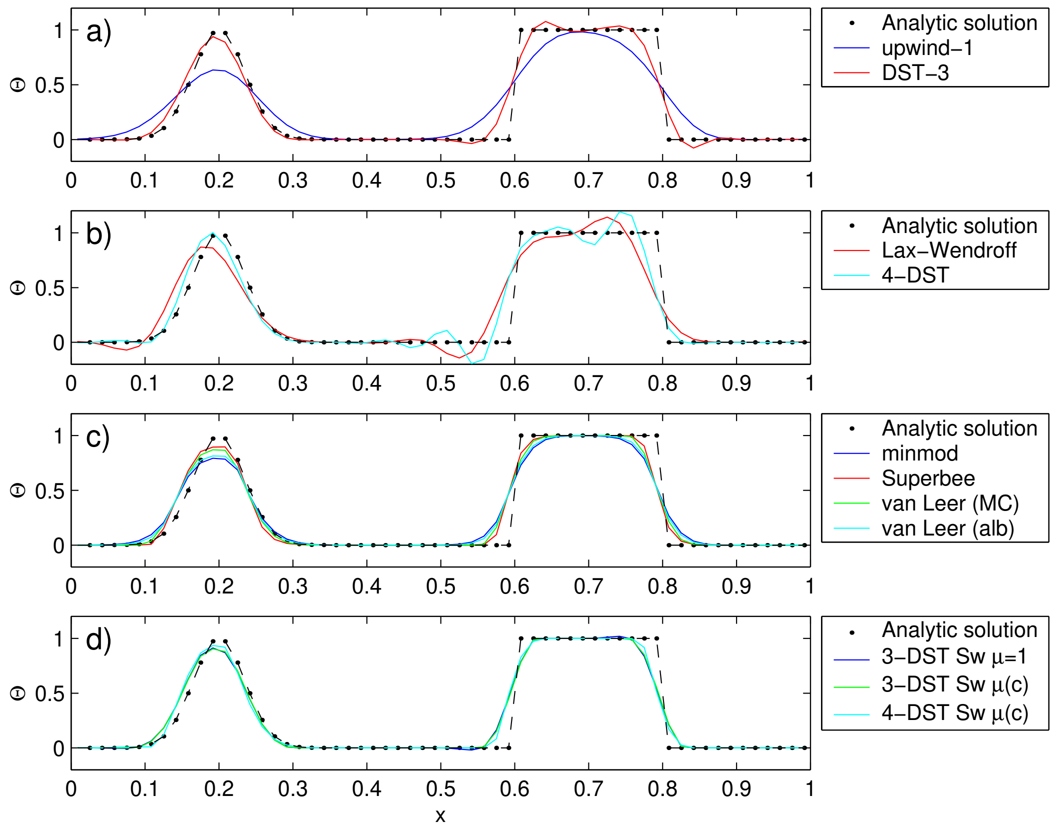


Figure 2.14: Comparison of 1-D advection schemes. Courant number is 0.89 with 60 points and solutions are shown for T=1 (one complete period). a) Shows the upwind biased schemes; first order upwind and DST3. Third order upwind and second order upwind are unstable at this Courant number. b) Shows the centered schemes; Lax-Wendroff, DST4. Centered second order, centered fourth order and finite volume fourth order and unstable at this Courant number. c) Shows the second order flux limiters: minmod, Superbee, MC limiter and the van Leer limiter. d) Shows the DST3 method with flux limiters due to Sweby with $\mu = 1$, $\mu = c/(1-c)$ and a fourth order DST method with Sweby limiter, $\mu = c/(1-c)$.

2.17.1 Centered second order advection-diffusion

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

The advection operator is discretized:

$$\mathcal{A}_c \Delta r_f h_c G^{\tau}_{adv} = \delta_i F_x + \delta_j F_y + \delta_k F_r \tag{2.171}$$

where the area integrated fluxes are given by:

$$F_x = U\overline{\tau}^i \tag{2.172}$$
$$F_y = V\overline{\tau}^j \tag{2.173}$$
$$F_r = W\overline{\tau}^k \tag{2.174}$$

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

$$U = \Delta y_g \Delta r_f h_w u \tag{2.175}$$
$$V = \Delta x_g \Delta r_f h_s v \tag{2.176}$$
$$W = \mathcal{A}_c w \tag{2.177}$$

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

S/R GAD_C2_ADV_X (*gad_c2_adv_x.F*)
F_x: **uT** (argument)
U: **uTrans** (argument)
τ: **tracer** (argument)
S/R GAD_C2_ADV_Y (*gad_c2_adv_y.F*)
F_y: **vT** (argument)
V: **vTrans** (argument)
τ: **tracer** (argument)
S/R GAD_C2_ADV_R (*gad_c2_adv_r.F*)
F_r: **wT** (argument)
W: **rTrans** (argument)
τ: **tracer** (argument)

2.17.2 Third order upwind bias advection

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

The third order upwind fluxes are discretized:

$$F_x = U\overline{\tau - \frac{1}{6}\delta_{ii}\tau}^i + \frac{1}{2}|U|\delta_i \frac{1}{6}\delta_{ii}\tau \tag{2.178}$$
$$F_y = V\overline{\tau - \frac{1}{6}\delta_{ii}\tau}^j + \frac{1}{2}|V|\delta_j \frac{1}{6}\delta_{jj}\tau \tag{2.179}$$
$$F_r = W\overline{\tau - \frac{1}{6}\delta_{ii}\tau}^k + \frac{1}{2}|W|\delta_k \frac{1}{6}\delta_{kk}\tau \tag{2.180}$$

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

S/R GAD_U3_ADV_X (*gad_u3_adv_x.F*)
F_x: **uT** (argument)
U: **uTrans** (argument)
τ: **tracer** (argument)
S/R GAD_U3_ADV_Y (*gad_u3_adv_y.F*)
F_y: **vT** (argument)
V: **vTrans** (argument)
τ: **tracer** (argument)
S/R GAD_U3_ADV_R (*gad_u3_adv_r.F*)
F_r: **wT** (argument)
W: **rTrans** (argument)
τ: **tracer** (argument)

2.17.3 Centered fourth order advection

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

The centered fourth order fluxes are discretized:

$$F_x = U\overline{\tau - \frac{1}{6}\delta_{ii}\tau}^i \tag{2.181}$$

$$F_y = V\overline{\tau - \frac{1}{6}\delta_{ii}\tau}^j \tag{2.182}$$

$$F_r = W\overline{\tau - \frac{1}{6}\delta_{ii}\tau}^k \tag{2.183}$$

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

S/R GAD_C4_ADV_X (*gad_c4_adv_x.F*)
F_x: **uT** (argument)
U: **uTrans** (argument)
τ: **tracer** (argument)
S/R GAD_C4_ADV_Y (*gad_c4_adv_y.F*)
F_y: **vT** (argument)
V: **vTrans** (argument)
τ: **tracer** (argument)
S/R GAD_C4_ADV_R (*gad_c4_adv_r.F*)
F_r: **wT** (argument)
W: **rTrans** (argument)
τ: **tracer** (argument)

2.17.4 First order upwind advection

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

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

$$F_x = U\overline{\tau}^i - \frac{1}{2}|U|\delta_i\tau \tag{2.184}$$

$$F_y = V\overline{\tau}^j - \frac{1}{2}|V|\delta_j\tau \tag{2.185}$$

$$F_r = W\overline{\tau}^k - \frac{1}{2}|W|\delta_k\tau \tag{2.186}$$

If for some reason, the above method is required, then the second order flux limiter scheme described later reduces to the above scheme if the limiter is set to zero.

2.18 Non-linear advection schemes

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

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

2.18.1 Second order flux limiters

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

$$F = F_1 + \psi(r) F_{LW} \tag{2.187}$$

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

$$F_1 = u\overline{\tau}^i - \frac{1}{2}|u|\delta_i\tau \tag{2.188}$$

is the upwind flux,

$$F_{LW} = F_1 + \frac{|u|}{2}(1-c)\delta_i\tau \tag{2.189}$$

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

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

$$r = \frac{\tau_{i-1} - \tau_{i-2}}{\tau_i - \tau_{i-1}} \quad \forall \quad u > 0 \tag{2.190}$$

$$r = \frac{\tau_{i+1} - \tau_i}{\tau_i - \tau_{i-1}} \quad \forall \quad u < 0 \tag{2.191}$$

as it's argument. There are many choices of limiter function but we only provide the Superbee limiter *Roe* [1985]:

$$\psi(r) = \max[0, \min[1, 2r], \min[2, r]] \tag{2.192}$$

S/R GAD_FLUXLIMIT_ADV_X (*gad_fluxlimit_adv_x.F*)
F_x: **uT** (argument)
U: **uTrans** (argument)
τ: **tracer** (argument)
S/R GAD_FLUXLIMIT_ADV_Y (*gad_fluxlimit_adv_y.F*)
F_y: **vT** (argument)
V: **vTrans** (argument)
τ: **tracer** (argument)
S/R GAD_FLUXLIMIT_ADV_R (*gad_fluxlimit_adv_r.F*)
F_r: **wT** (argument)
W: **rTrans** (argument)
τ: **tracer** (argument)

2.18.2 Third order direct space time

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

$$F = u\,(\tau_{i-1} + d_0(\tau_i - \tau_{i-1}) + d_1(\tau_{i-1} - \tau_{i-2})) \quad \forall \quad u > 0 \tag{2.193}$$
$$F = u\,(\tau_i - d_0(\tau_i - \tau_{i-1}) - d_1(\tau_{i+1} - \tau_i)) \quad \forall \quad u < 0 \tag{2.194}$$

where

$$d_1 = \frac{1}{6}(2 - |c|)(1 - |c|) \tag{2.195}$$
$$d_2 = \frac{1}{6}(1 - |c|)(1 + |c|) \tag{2.196}$$

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

The DST3 method described above must be used in a forward-in-time manner and is stable for $0 \leq |c| \leq 1$. Although the scheme appears to be forward-in-time, it is in fact third order in time and the accuracy increases with the Courant number! For low Courant number, DST3 produces very similar results (indistinguishable in Fig. 2.13) to the linear third order method but for large Courant number, where the linear upwind third order method is unstable, the scheme is extremely accurate (Fig. 2.14) with only minor overshoots.

S/R GAD_DST3_ADV_X (*gad_dst3_adv_x.F*)
F_x: **uT** (argument)
U: **uTrans** (argument)
τ: **tracer** (argument)
S/R GAD_DST3_ADV_Y (*gad_dst3_adv_y.F*)
F_y: **vT** (argument)
V: **vTrans** (argument)
τ: **tracer** (argument)
S/R GAD_DST3_ADV_R (*gad_dst3_adv_r.F*)
F_r: **wT** (argument)
W: **rTrans** (argument)
τ: **tracer** (argument)

2.18.3 Third order direct space time with flux limiting

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

$$F = \frac{1}{2}(u + |u|)\left(\tau_{i-1} + \psi(r^+)(\tau_i - \tau_{i-1})\right) + \frac{1}{2}(u - |u|)\left(\tau_{i-1} + \psi(r^-)(\tau_i - \tau_{i-1})\right) \tag{2.197}$$

where

$$r^+ = \frac{\tau_{i-1} - \tau_{i-2}}{\tau_i - \tau_{i-1}} \tag{2.198}$$
$$r^- = \frac{\tau_{i+1} - \tau_i}{\tau_i - \tau_{i-1}} \tag{2.199}$$

and the limiter is the Sweby limiter:

$$\psi(r) = \max[0, \min[\min(1, d_0 + d_1 r], \frac{1-c}{c} r]] \tag{2.200}$$

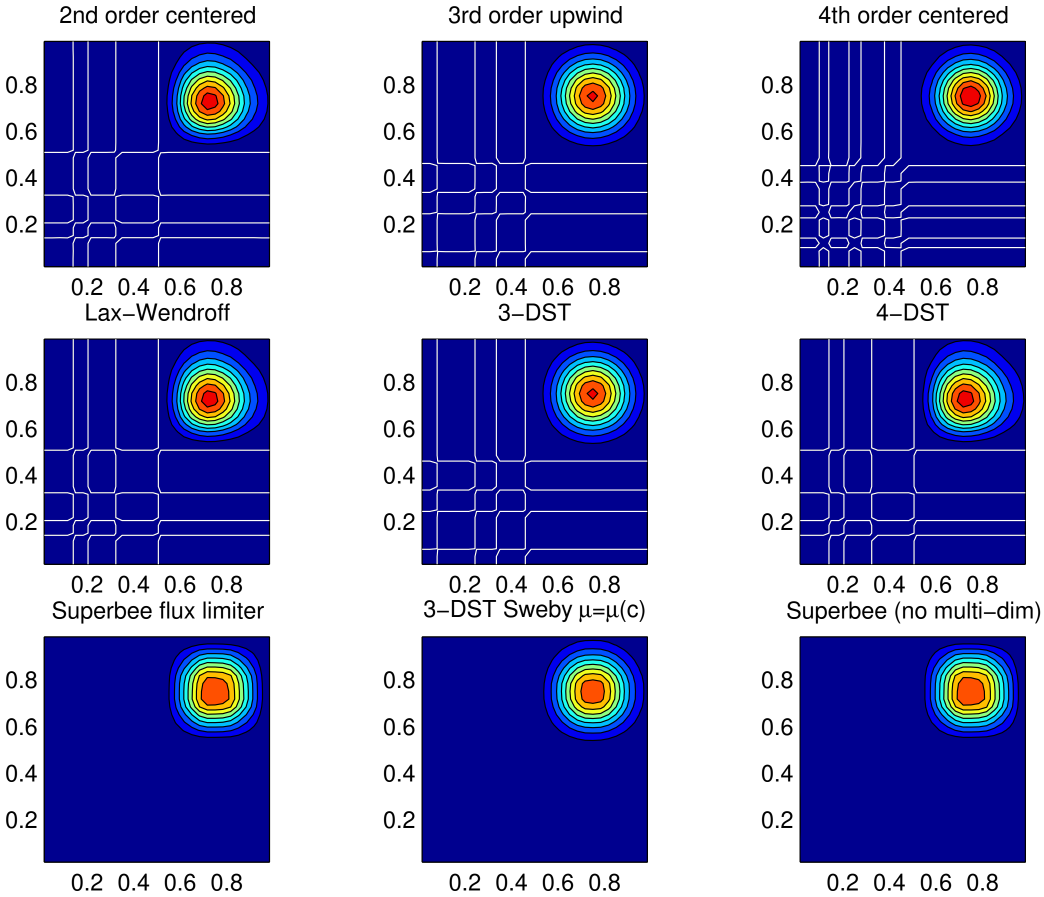


Figure 2.15: Comparison of advection schemes in two dimensions; diagonal advection of a resolved Gaussian feature. Courant number is 0.01 with 30×30 points and solutions are shown for T=1/2. White lines indicate zero crossing (ie. the presence of false minima). The left column shows the second order schemes; top) centered second order with Adams-Bashforth, middle) Lax-Wendroff and bottom) Superbee flux limited. The middle column shows the third order schemes; top) upwind biased third order with Adams-Bashforth, middle) third order direct space-time method and bottom) the same with flux limiting. The top right panel shows the centered fourth order scheme with Adams-Bashforth and right middle panel shows a fourth order variant on the DST method. Bottom right panel shows the Superbee flux limiter (second order) applied independently in each direction (method of lines).

S/R GAD_DST3FL_ADV_X (*gad_dst3_adv_x.F*)
F_x: **uT** (argument)
U: **uTrans** (argument)
τ: **tracer** (argument)
S/R GAD_DST3FL_ADV_Y (*gad_dst3_adv_y.F*)
F_y: **vT** (argument)
V: **vTrans** (argument)
τ: **tracer** (argument)
S/R GAD_DST3FL_ADV_R (*gad_dst3_adv_r.F*)
F_r: **wT** (argument)
W: **rTrans** (argument)
τ: **tracer** (argument)

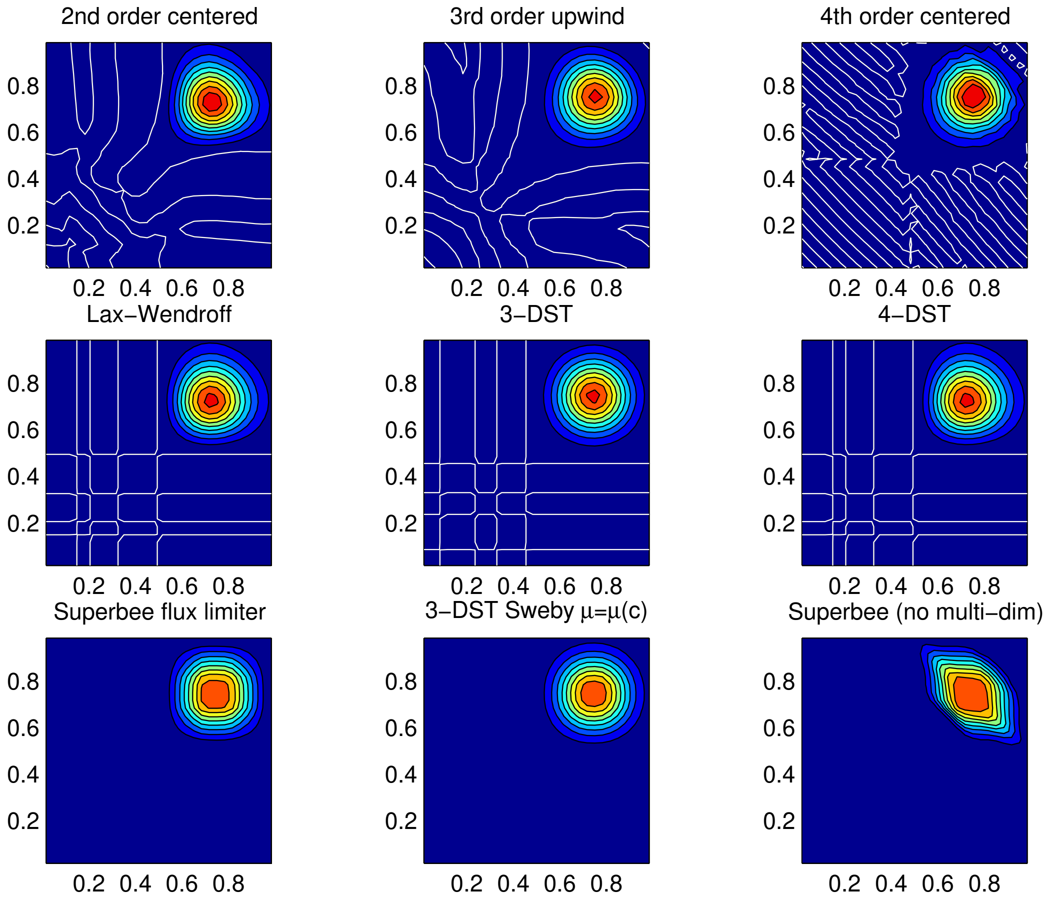


Figure 2.16: Comparison of advection schemes in two dimensions; diagonal advection of a resolved Gaussian feature. Courant number is 0.27 with 30×30 points and solutions are shown for T=1/2. White lines indicate zero crossing (ie. the presence of false minima). The left column shows the second order schemes; top) centered second order with Adams-Bashforth, middle) Lax-Wendroff and bottom) Superbee flux limited. The middle column shows the third order schemes; top) upwind biased third order with Adams-Bashforth, middle) third order direct space-time method and bottom) the same with flux limiting. The top right panel shows the centered fourth order scheme with Adams-Bashforth and right middle panel shows a fourth order variant on the DST method. Bottom right panel shows the Superbee flux limiter (second order) applied independently in each direction (method of lines).

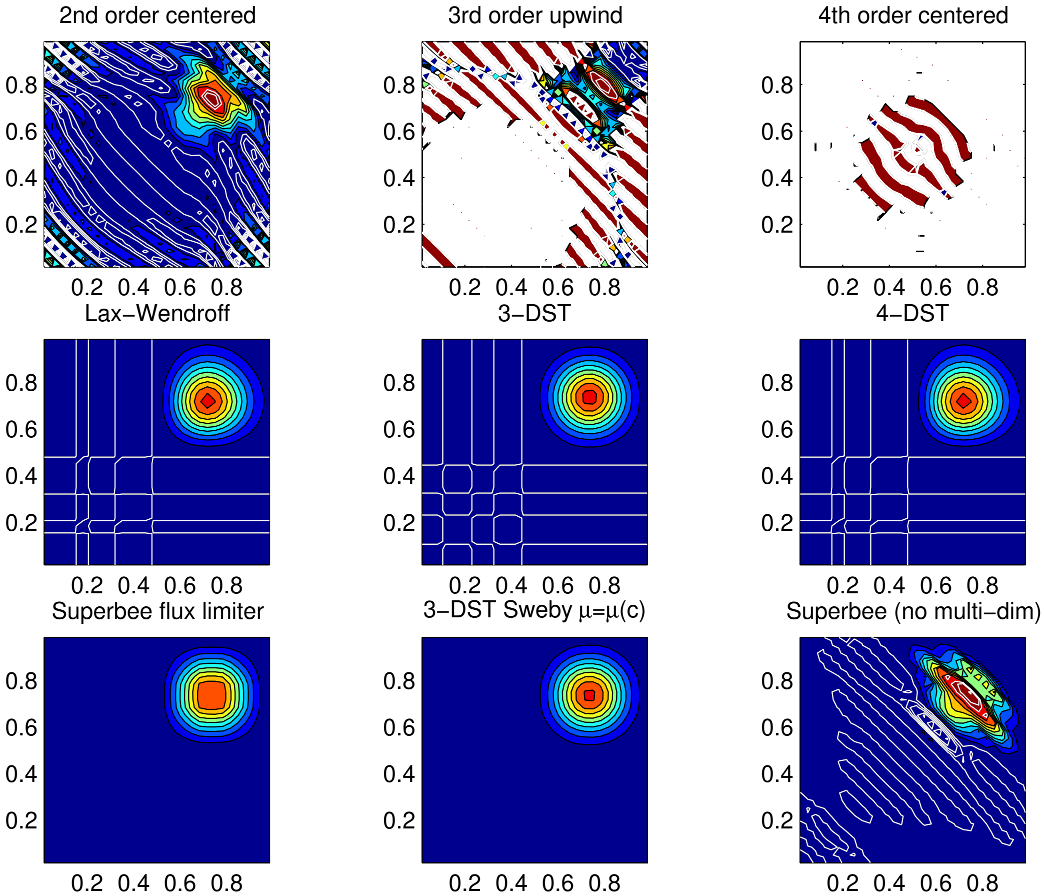


Figure 2.17: Comparison of advection schemes in two dimensions; diagonal advection of a resolved Gaussian feature. Courant number is 0.47 with 30×30 points and solutions are shown for T=1/2. White lines indicate zero crossings and initial maximum values (ie. the presence of false extrema). The left column shows the second order schemes; top) centered second order with Adams-Bashforth, middle) Lax-Wendroff and bottom) Superbee flux limited. The middle column shows the third order schemes; top) upwind biased third order with Adams-Bashforth, middle) third order direct space-time method and bottom) the same with flux limiting. The top right panel shows the centered fourth order scheme with Adams-Bashforth and right middle panel shows a fourth order variant on the DST method. Bottom right panel shows the Superbee flux limiter (second order) applied independently in each direction (method of lines).

2.18.4 Multi-dimensional advection

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

$$\tau^{n+1/3} = \tau^n - \Delta t \left(\frac{1}{\Delta x} \delta_i F^x(\tau^n) + \tau^n \frac{1}{\Delta x} \delta_i u \right) \tag{2.201}$$

$$\tau^{n+2/3} = \tau^{n+1/3} - \Delta t \left(\frac{1}{\Delta y} \delta_j F^y(\tau^{n+1/3}) + \tau^n \frac{1}{\Delta y} \delta_i v \right) \tag{2.202}$$

$$\tau^{n+3/3} = \tau^{n+2/3} - \Delta t \left(\frac{1}{\Delta r} \delta_k F^x(\tau^{n+2/3}) + \tau^n \frac{1}{\Delta r} \delta_i w \right) \tag{2.203}$$

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

$$G^{n+1/2}_{adv} = \frac{1}{\Delta t}(\tau^{n+3/3} - \tau^n) \tag{2.204}$$

So that the over all time-stepping looks likes:

$$\tau^{n+1} = \tau^n + \Delta t \left(G^{n+1/2}_{adv} + G_{diff}(\tau^n) + G^n_{forcing} \right) \tag{2.205}$$

S/R GAD_ADVECTION (*gad_advection.F*)
τ: **Tracer** (argument)
$G^{n+1/2}_{adv}$: **Gtracer** (argument)
F_x, F_y, F_r: **af** (local)
U: **uTrans** (local)
V: **vTrans** (local)
W: **rTrans** (local)

2.19 Comparison of advection schemes

Figs. 2.15, 2.16 and 2.17 show solutions to a simple diagonal advection problem using a selection of schemes for low, moderate and high Courant numbers, respectively. The top row shows the linear schemes, integrated with the Adams-Bashforth method. Theses schemes are clearly unstable for the high Courant number and weakly unstable for the moderate Courant number. The presence of false extrema is very apparent for all Courant numbers. The middle row shows solutions obtained with the unlimited but multi-dimensional schemes. These solutions also exhibit false extrema though the pattern now shows symmetry due to the multi-dimensional scheme. Also, the schemes are stable at high Courant number where the linear schemes weren't. The bottom row (left and middle) shows the limited schemes and most obvious is the absence of false extrema. The accuracy and stability of the unlimited non-linear schemes is retained at high Courant number but at low Courant number the tendency is to loose amplitude in sharp peaks due to diffusion. The one dimensional tests shown in Figs. 2.13 and 2.14 showed this phenomenon.

Finally, the bottom left and right panels use the same advection scheme but the right does not use the multi-dimensional method. At low Courant number this appears to not matter but for moderate Courant number severe distortion of the feature is apparent. Moreover, the stability of the multi-dimensional scheme is determined by the maximum Courant number applied of each dimension while the stability of the method of lines is determined by the sum. Hence, in the high Courant number plot, the scheme is unstable.

With many advection schemes implemented in the code two questions arise: "Which scheme is best?" and "Why don't you just offer the best advection scheme?". Unfortunately, no one advection scheme is

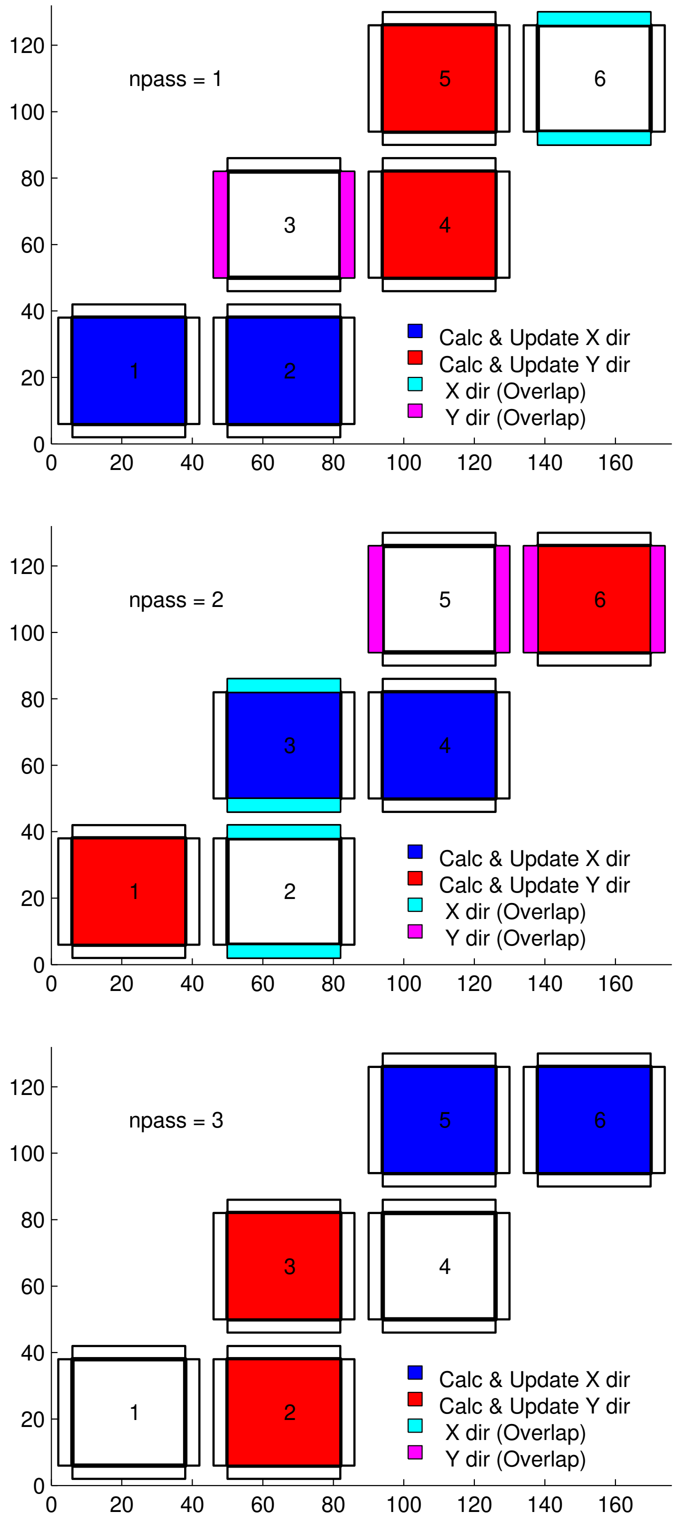


Figure 2.18: Muti-dimensional advection time-stepping with Cubed-Sphere topology

Advection Scheme	code	use A.B.	use Multi-dimension	Stencil (1 dim)	comments
1^{rst}order upwind	1	No	Yes	3 pts	linear/τ, non-linear/v
centered 2^{nd}order	2	Yes	No	3 pts	linear
3^{rd}order upwind	3	Yes	No	5 pts	linear/τ
centered 4^{th}order	4	Yes	No	5 pts	linear
2^{nd}order DST (Lax-Wendroff)	20	No	Yes	3 pts	linear/τ, non-linear/v
3^{rd}order DST	30	No	Yes	5 pts	linear/τ, non-linear/v
2^{nd}order-moment Prather	80	No	Yes		
2^{nd}order Flux Limiters	77	No	Yes	5 pts	non-linear
3^{nd}order DST Flux limiter	33	No	Yes	5 pts	non-linear
2^{nd}order-moment Prather w. limiter	81	No	Yes		
piecewise parabolic w. "null" limiter	40	No	Yes		
piecewise parabolic w. "mono" limiter	41	No	Yes		
piecewise quartic w. "null" limiter	50	No	Yes		
piecewise quartic w. "mono" limiter	51	No	Yes		
piecewise quartic w. "weno" limiter	52	No	Yes		
7^{nd}order one-step method with Monotonicity Preserving Limiter	7	No	Yes		

Table 2.2: Summary of the different advection schemes available in MITgcm. "A.B." stands for Adams-Bashforth and "DST" for direct space time. The code corresponds to the number used to select the corresponding advection scheme in the parameter file (e.g., **tempAdvScheme**=3 in file *data* selects the 3^{rd} order upwind advection scheme for temperature).

"the best" for all particular applications and for new applications it is often a matter of trial to determine which is most suitable. Here are some guidelines but these are not the rule;

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

2.20 Shapiro Filter

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

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

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

The 3 computational filter operators are :

$$\text{S1c}: \qquad [1 - 1/2\frac{\Delta t}{\tau_{shap}}\{(\frac{1}{4}\delta_{ii})^n + (\frac{1}{4}\delta_{jj})^n\}]$$

$$\text{S2c}: \qquad [1 - \frac{\Delta t}{\tau_{shap}}\{\frac{1}{8}(\delta_{ii} + \delta_{jj})\}^n]$$

$$\text{S4c}: \qquad [1 - \frac{\Delta t}{\tau_{shap}}(\frac{1}{4}\delta_{ii})^n][1 - \frac{\Delta t}{\tau_{shap}}(\frac{1}{4}\delta_{jj})^n]$$

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

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

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

2.20.1 SHAP Diagnostics

```
------------------------------------------------------------------------
<-Name->|Levs|<-parsing code->|<--  Units   -->|<- Tile (max=80c)
------------------------------------------------------------------------
SHAP_dT |  5 |SM      MR      |K/s             |Temperature Tendency due to Shapiro Filter
SHAP_dS |  5 |SM      MR      |g/kg/s          |Specific Humidity Tendency due to Shapiro Filter
SHAP_dU |  5 |UU   148MR      |m/s^2           |Zonal Wind Tendency due to Shapiro Filter
SHAP_dV |  5 |VV   147MR      |m/s^2           |Meridional Wind Tendency due to Shapiro Filter
```

2.21 Nonlinear Viscosities for Large Eddy Simulation

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

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

$$\begin{aligned}
\frac{\overline{D}\tilde{\overline{u}}}{\overline{Dt}} - \frac{\tilde{\overline{v}}\sin\theta}{\mathrm{Ro}\sin\theta_0} + \frac{M_{Ro}}{\mathrm{Ro}}\frac{\partial\overline{\pi}}{\partial x} &= -\left(\frac{\overline{D\tilde{u}}}{Dt} - \frac{\overline{D}\tilde{\overline{u}}}{\overline{Dt}}\right) + \frac{\nabla^2\tilde{\overline{u}}}{\mathrm{Re}} \qquad (2.206)\\
\frac{\overline{D}\tilde{\overline{v}}}{\overline{Dt}} + \frac{\tilde{\overline{u}}\sin\theta}{\mathrm{Ro}\sin\theta_0} + \frac{M_{Ro}}{\mathrm{Ro}}\frac{\partial\overline{\pi}}{\partial y} &= -\left(\frac{\overline{D\tilde{v}}}{Dt} - \frac{\overline{D}\tilde{\overline{v}}}{\overline{Dt}}\right) + \frac{\nabla^2\tilde{\overline{v}}}{\mathrm{Re}}\\
\frac{\overline{D}\overline{w}}{\overline{Dt}} + \frac{\frac{\partial\overline{\pi}}{\partial z} - \overline{b}}{\mathrm{Fr}^2\lambda^2} &= -\left(\frac{\overline{Dw}}{Dt} - \frac{\overline{D}\overline{w}}{\overline{Dt}}\right) + \frac{\nabla^2\overline{w}}{\mathrm{Re}}\\
\frac{\overline{D}\,\overline{b}}{\overline{Dt}} + \overline{w} &= -\left(\frac{\overline{Db}}{Dt} - \frac{\overline{D}\,\overline{b}}{\overline{Dt}}\right) + \frac{\nabla^2\overline{b}}{\mathrm{Pr}\,\mathrm{Re}}\\
\mu^2\left(\frac{\partial\tilde{\overline{u}}}{\partial x} + \frac{\partial\tilde{\overline{v}}}{\partial y}\right) + \frac{\partial\overline{w}}{\partial z} &= 0 \qquad (2.207)
\end{aligned}$$

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

The ocean is usually turbulent, and an operational definition of turbulence is that the terms in parentheses (the 'eddy' terms) on the right of (2.206) are of comparable magnitude to the terms on the left-hand side. The terms proportional to the inverse of Re, instead, are many orders of magnitude smaller than all of the other terms in virtually every oceanic application.

2.21.1 Eddy Viscosity

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

$$\left(\overline{\frac{D\tilde{u}}{Dt}} - \frac{\overline{D}\tilde{\overline{u}}}{\overline{Dt}}\right) \approx \frac{\nabla_h^2\tilde{\overline{u}}}{\mathrm{Re}_h} + \frac{\frac{\partial^2\tilde{\overline{u}}}{\partial z^2}}{\mathrm{Re}_v}, \qquad \left(\overline{\frac{D\tilde{v}}{Dt}} - \frac{\overline{D}\tilde{\overline{v}}}{\overline{Dt}}\right) \approx \frac{\nabla_h^2\tilde{\overline{v}}}{\mathrm{Re}_h} + \frac{\frac{\partial^2\tilde{\overline{v}}}{\partial z^2}}{\mathrm{Re}_v}$$

$$\left(\overline{\frac{Dw}{Dt}} - \frac{\overline{D}\overline{w}}{\overline{Dt}}\right) \approx \frac{\nabla_h^2\overline{w}}{\mathrm{Re}_h} + \frac{\frac{\partial^2\overline{w}}{\partial z^2}}{\mathrm{Re}_v}, \qquad \left(\overline{\frac{Db}{Dt}} - \frac{\overline{D}\,\overline{b}}{\overline{Dt}}\right) \approx \frac{\nabla_h^2\overline{b}}{\mathrm{Pr}\,\mathrm{Re}_h} + \frac{\frac{\partial^2\overline{b}}{\partial z^2}}{\mathrm{Pr}\,\mathrm{Re}_v}$$

2.21.1.1 Reynolds-Number Limited Eddy Viscosity

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

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

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

2.21.1.2 Vertical Eddy Viscosities

Vertical eddy viscosities are often chosen in a more subjective way, as model stability is not usually as sensitive to vertical viscosity. Usually the 'observed' value from finescale measurements, etc., is used (*eg.* viscAr$\approx 1\times10^{-4}m^2/s$). However, Smagorinsky *Smagorinsky* [1993] notes that the Smagorinsky parameterization of isotropic turbulence implies a value of the vertical viscosity as well as the horizontal viscosity (see below).

2.21.1.3 Smagorinsky Viscosity

Some *Smagorinsky* [1963, 1993] suggest choosing a viscosity that *depends on the resolved motions*. Thus, the overall viscous operator has a nonlinear dependence on velocity. Smagorinsky chose his form of viscosity by considering Kolmogorov's ideas about the energy spectrum of 3-d isotropic turbulence.

Kolmogorov suppposed that is that energy is injected into the flow at large scales (small k) and is 'cascaded' or transferred conservatively by nonlinear processes to smaller and smaller scales until it is dissipated near the viscous scale. By setting the energy flux through a particular wavenumber k, ϵ, to be a constant in k, there is only one combination of viscosity and energy flux that has the units of length, the Kolmogorov wavelength. It is $L_\epsilon(\nu) \propto \pi\epsilon^{-1/4}\nu^{3/4}$ (the π stems from conversion from wavenumber to wavelength). To ensure that this viscous scale is resolved in a numerical model, the gridscale should be decreased until $L_\epsilon(\nu) > L$ (so-called Direct Numerical Simulation, or DNS). Alternatively, an eddy viscosity can be used and the corresponding Kolmogorov length can be made larger than the gridscale, $L_\epsilon(A_h) \propto \pi\epsilon^{-1/4}A_h^{3/4}$ (for Large Eddy Simulation or LES).

There are two methods of ensuring that the Kolmogorov length is resolved in MITgcm. 1) The user can estimate the flux of energy through spectral space for a given simulation and adjust grid spacing or viscAh to ensure that $L_\epsilon(A_h) > L$. 2) The user may use the approach of Smagorinsky with viscC2Smag, which estimates the energy flux at every grid point, and adjusts the viscosity accordingly.

Smagorinsky formed the energy equation from the momentum equations by dotting them with velocity. There are some complications when using the hydrostatic approximation as described by Smagorinsky *Smagorinsky* [1993]. The positive definite energy dissipation by horizontal viscosity in a hydrostatic flow is νD^2, where D is the deformation rate at the viscous scale. According to Kolmogorov's theory, this should be a good approximation to the energy flux at any wavenumber $\epsilon \approx \nu D^2$. Kolmogorov and Smagorinsky noted that using an eddy viscosity that exceeds the molecular value ν should ensure that the energy flux through viscous scale set by the eddy viscosity is the same as it would have been had we resolved all the way to the true viscous scale. That is, $\epsilon \approx A_{hSmag}\overline{D}^2$. If we use this approximation to estimate the Kolmogorov viscous length, then

$$L_\epsilon(A_{hSmag}) \propto \pi\epsilon^{-1/4}A_{hSmag}^{3/4} \approx \pi(A_{hSmag}\overline{D}^2)^{-1/4}A_{hSmag}^{3/4} = \pi A_{hSmag}^{1/2}\overline{D}^{-1/2} \tag{2.208}$$

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

$$A_{hSmag} = \left(\frac{\mathsf{viscC2Smag}}{\pi}\right)^2 L^2|\overline{D}| \tag{2.209}$$

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

$$|\overline{D}| = \sqrt{\left(\frac{\partial\overline{\tilde{u}}}{\partial x} - \frac{\partial\overline{\tilde{v}}}{\partial y}\right)^2 + \left(\frac{\partial\overline{\tilde{u}}}{\partial y} + \frac{\partial\overline{\tilde{v}}}{\partial x}\right)^2} \tag{2.210}$$

The coefficient viscC2Smag is what an MITgcm user sets, and it replaces the proportionality in the Kolmogorov length with an equality. Others *Griffies and Hallberg* [2000] suggest values of viscC2Smag from 2.2 to 4 for oceanic problems. Smagorinsky *Smagorinsky* [1993] shows that values from 0.2 to 0.9 have been used in atmospheric modeling.

Smagorinsky *Smagorinsky* [1993] shows that a corresponding vertical viscosity should be used:

$$A_{vSmag} = \left(\frac{\mathsf{viscC2Smag}}{\pi}\right)^2 H^2\sqrt{\left(\frac{\partial\overline{\tilde{u}}}{\partial z}\right)^2 + \left(\frac{\partial\overline{\tilde{v}}}{\partial z}\right)^2} \tag{2.211}$$

This vertical viscosity is currently not implemented in MITgcm (although it may be soon).

2.21.1.4 Leith Viscosity

Leith *Leith* [1968, 1996] notes that 2-d turbulence is quite different from 3-d. In two-dimensional turbulence, energy cascades to larger scales, so there is no concern about resolving the scales of energy dissipation. Instead, another quantity, enstrophy, (which is the vertical component of vorticity squared) is conserved in 2-d turbulence, and it cascades to smaller scales where it is dissipated.

Following a similar argument to that above about energy flux, the enstrophy flux is estimated to be equal to the positive-definite gridscale dissipation rate of enstrophy $\eta \approx A_{hLeith}|\nabla\overline{\omega}_3|^2$. By dimensional analysis, the enstrophy-dissipation scale is $L_\eta(A_{hLeith}) \propto \pi A_{hLeith}^{1/2}\eta^{-1/6}$. Thus, the Leith-estimated length scale of enstrophy-dissipation and the resulting eddy viscosity are

$$L_\eta(A_{hLeith}) \propto \pi A_{hLeith}^{1/2}\eta^{-1/6} = \pi A_{hLeith}^{1/3}|\nabla\overline{\omega}_3|^{-1/3} \tag{2.212}$$

$$A_{hLeith} = \left(\frac{\mathsf{viscC2Leith}}{\pi}\right)^3 L^3|\nabla\overline{\omega}_3| \tag{2.213}$$

$$|\nabla\omega_3| \equiv \sqrt{\left[\frac{\partial}{\partial x}\left(\frac{\partial\overline{\tilde{v}}}{\partial x} - \frac{\partial\overline{\tilde{u}}}{\partial y}\right)\right]^2 + \left[\frac{\partial}{\partial y}\left(\frac{\partial\overline{\tilde{v}}}{\partial x} - \frac{\partial\overline{\tilde{u}}}{\partial y}\right)\right]^2} \tag{2.214}$$

2.21.1.5 Modified Leith Viscosity

The argument above for the Leith viscosity parameterization uses concepts from purely 2-dimensional turbulence, where the horizontal flow field is assumed to be divergenceless. However, oceanic flows are only quasi-two dimensional. While the barotropic flow, or the flow within isopycnal layers may behave nearly as two-dimensional turbulence, there is a possibility that these flows will be divergent. In a high-resolution numerical model, these flows may be substantially divergent near the grid scale, and in fact, numerical instabilities exist which are only horizontally divergent and have little vertical vorticity. This causes a difficulty with the Leith viscosity, which can only responds to buildup of vorticity at the grid scale.

MITgcm offers two options for dealing with this problem. 1) The Smagorinsky viscosity can be used instead of Leith, or in conjunction with Leith–a purely divergent flow does cause an increase in Smagorinsky viscosity. 2) The viscC2LeithD parameter can be set. This is a damping specifically targeting purely divergent instabilities near the gridscale. The combined viscosity has the form:

$$A_{hLeith} = L^3\sqrt{\left(\frac{\mathsf{viscC2Leith}}{\pi}\right)^6 |\nabla\overline{\omega}_3|^2 + \left(\frac{\mathsf{viscC2LeithD}}{\pi}\right)^6 |\nabla\nabla\cdot\tilde{\overline{u}}_h|^2} \tag{2.215}$$

$$|\nabla\nabla\cdot\tilde{\overline{u}}_h| \equiv \sqrt{\left[\frac{\partial}{\partial x}\left(\frac{\partial\tilde{\overline{u}}}{\partial x}+\frac{\partial\tilde{\overline{v}}}{\partial y}\right)\right]^2 + \left[\frac{\partial}{\partial y}\left(\frac{\partial\tilde{\overline{u}}}{\partial x}+\frac{\partial\tilde{\overline{v}}}{\partial y}\right)\right]^2} \tag{2.216}$$

Whether there is any physical rationale for this correction is unclear at the moment, but the numerical consequences are good. The divergence in flows with the grid scale larger or comparable to the Rossby radius is typically much smaller than the vorticity, so this adjustment only rarely adjusts the viscosity if viscC2LeithD = viscC2Leith. However, the rare regions where this viscosity acts are often the locations for the largest vales of vertical velocity in the domain. Since the CFL condition on vertical velocity is often what sets the maximum timestep, this viscosity may substantially increase the allowable timestep without severely compromising the verity of the simulation. Tests have shown that in some calculations, a timestep three times larger was allowed when viscC2LeithD = viscC2Leith.

2.21.1.6 Courant–Freidrichs–Lewy Constraint on Viscosity

Whatever viscosities are used in the model, the choice is constrained by gridscale and timestep by the Courant–Freidrichs–Lewy (CFL) constraint on stability:

$$A_h < \frac{L^2}{4\Delta t} \tag{2.217}$$

$$A_4 \le \frac{L^4}{32\Delta t} \tag{2.218}$$

The viscosities may be automatically limited to be no greater than these values in MITgcm by specifying viscAhGridMax<1 and viscA4GridMax<1. Similarly-scaled minimum values of viscosities are provided by viscAhGridMin and viscA4GridMin, which if used, should be set to values $\ll 1$. L is roughly the gridscale (see below).

Following *Griffies and Hallberg* [2000], we note that there is a factor of $\Delta x^2/8$ difference between the harmonic and biharmonic viscosities. Thus, whenever a non-dimensional harmonic coefficient is used in the MITgcm (*eg.* viscAhGridMax<1), the biharmonic equivalent is scaled so that the same non-dimensional value can be used (*eg.* viscA4GridMax<1).

2.21.1.7 Biharmonic Viscosity

Holland [1978] suggested that eddy viscosities ought to be focuses on the dynamics at the grid scale, as larger motions would be 'resolved'. To enhance the scale selectivity of the viscous operator, he suggested a biharmonic eddy viscosity instead of a harmonic (or Laplacian) viscosity:

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

$$\left(\overline{\frac{Dw}{Dt}} - \frac{D\overline{w}}{\overline{\overline{Dt}}}\right) \approx \frac{-\nabla_h^4\overline{w}}{\mathrm{Re}_4} + \frac{\frac{\partial^2\overline{w}}{\partial z^2}}{\mathrm{Re}_v}, \qquad \left(\overline{\frac{Db}{Dt}} - \frac{D\,\overline{b}}{\overline{\overline{Dt}}}\right) \approx \frac{-\nabla_h^4\overline{b}}{\mathrm{Pr}\,\mathrm{Re}_4} + \frac{\frac{\partial^2\overline{b}}{\partial z^2}}{\mathrm{Pr}\,\mathrm{Re}_v}$$

Griffies and Hallberg [2000] propose that if one scales the biharmonic viscosity by stability considerations, then the biharmonic viscous terms will be similarly active to harmonic viscous terms at the gridscale of the model, but much less active on larger scale motions. Similarly, a biharmonic diffusivity can be used for less diffusive flows.

In practice, biharmonic viscosity and diffusivity allow a less viscous, yet numerically stable, simulation than harmonic viscosity and diffusivity. However, there is no physical rationale for such operators being of leading order, and more boundary conditions must be specified than for the harmonic operators. If one considers the approximations of 2.208 and 2.219 to be terms in the Taylor series expansions of the eddy terms as functions of the large-scale gradient, then one can argue that both harmonic and biharmonic terms would occur in the series, and the only question is the choice of coefficients. Using biharmonic viscosity alone implies that one zeros the first non-vanishing term in the Taylor series, which is unsupported by any fluid theory or observation.

Nonetheless, MITgcm supports a plethora of biharmonic viscosities and diffusivities, which are controlled with parameters named similarly to the harmonic viscosities and diffusivities with the substitution $h \rightarrow 4$. MITgcm also supports a biharmonic Leith and Smagorinsky viscosities:

$$A_{4Smag} = \left(\frac{\mathsf{viscC4Smag}}{\pi}\right)^2 \frac{L^4}{8}|D| \tag{2.219}$$

$$A_{4Leith} = \frac{L^5}{8}\sqrt{\left(\frac{\mathsf{viscC4Leith}}{\pi}\right)^6 |\nabla\overline{\omega}_3|^2 + \left(\frac{\mathsf{viscC4LeithD}}{\pi}\right)^6 |\nabla\nabla\cdot\overline{\tilde{\mathbf{u}}}_h|^2} \tag{2.220}$$

However, it should be noted that unlike the harmonic forms, the biharmonic scaling does not easily relate to whether energy-dissipation or enstrophy-dissipation scales are resolved. If similar arguments are used to estimate these scales and scale them to the gridscale, the resulting biharmonic viscosities should be:

$$A_{4Smag} = \left(\frac{\mathsf{viscC4Smag}}{\pi}\right)^5 L^5|\nabla^2\overline{\tilde{\mathbf{u}}}_h| \tag{2.221}$$

$$A_{4Leith} = L^6\sqrt{\left(\frac{\mathsf{viscC4Leith}}{\pi}\right)^{12} |\nabla^2\overline{\omega}_3|^2 + \left(\frac{\mathsf{viscC4LeithD}}{\pi}\right)^{12} |\nabla^2\nabla\cdot\overline{\tilde{\mathbf{u}}}_h|^2} \tag{2.222}$$

Thus, the biharmonic scaling suggested by *Griffies and Hallberg* [2000] implies:

$$|D| \propto L|\nabla^2\overline{\tilde{\mathbf{u}}}_h| \tag{2.223}$$

$$|\nabla\overline{\omega}_3| \propto L|\nabla^2\overline{\omega}_3| \tag{2.224}$$

It is not at all clear that these assumptions ought to hold. Only the *Griffies and Hallberg* [2000] forms are currently implemented in MITgcm.

2.21.1.8 Selection of Length Scale

Above, the length scale of the grid has been denoted L. However, in strongly anisotropic grids, L_x and L_y will be quite different in some locations. In that case, the CFL condition suggests that the minimum of L_x and L_y be used. On the other hand, other viscosities which involve whether a particular wavelength is 'resolved' might be better suited to use the maximum of L_x and L_y. Currently, MITgcm uses useAreaViscLength to select between two options. If false, the geometric mean of L_x^2 and L_y^2 is used for all viscosities, which is closer to the minimum and occurs naturally in the CFL constraint. If useAreaViscLength is true, then the square root of the area of the grid cell is used.

2.21.2 Mercator, Nondimensional Equations

The rotating, incompressible, Boussinesq equations of motion *Gill* [1982] on a sphere can be written in Mercator projection about a latitude θ_0 and geopotential height $z = r - r_0$. The nondimensional form of these equations is:

$$\mathrm{Ro}\frac{D\tilde{u}}{Dt} - \frac{\tilde{v}\sin\theta}{\sin\theta_0} + M_{Ro}\frac{\partial\pi}{\partial x} + \frac{\lambda \mathrm{Fr}^2 M_{Ro}\cos\theta}{\mu\sin\theta_0}w = -\frac{\mathrm{Fr}^2 M_{Ro}\tilde{u}w}{r/H} + \frac{\mathrm{Ro}\hat{\mathbf{x}}\cdot\nabla^2\mathbf{u}}{\mathrm{Re}} \tag{2.225}$$

$$\mathrm{Ro}\frac{D\tilde{v}}{Dt} + \frac{\tilde{u}\sin\theta}{\sin\theta_0} + M_{Ro}\frac{\partial\pi}{\partial y} = -\frac{\mu\mathrm{Ro}\tan\theta(\tilde{u}^2+\tilde{v}^2)}{r/L} - \frac{\mathrm{Fr}^2 M_{Ro}\tilde{v}w}{r/H} + \frac{\mathrm{Ro}\hat{\mathbf{y}}\cdot\nabla^2\mathbf{u}}{\mathrm{Re}} \tag{2.226}$$

$$\begin{aligned}
\mathrm{Fr}^2\lambda^2\frac{Dw}{Dt} - b + \frac{\partial\pi}{\partial z} - \frac{\lambda\cot\theta_0\tilde{u}}{M_{Ro}} &= \frac{\lambda\mu^2(\tilde{u}^2+\tilde{v}^2)}{M_{Ro}(r/L)} + \frac{\mathrm{Fr}^2\lambda^2\hat{\mathbf{z}}\cdot\nabla^2\mathbf{u}}{\mathrm{Re}} && (2.227)\\
\frac{Db}{Dt} + w &= \frac{\nabla^2 b}{\mathrm{Pr}\,\mathrm{Re}}\\
\mu^2\left(\frac{\partial\tilde{u}}{\partial x} + \frac{\partial\tilde{v}}{\partial y}\right) + \frac{\partial w}{\partial z} &= 0 && (2.228)
\end{aligned}$$

Where

$$\mu \equiv \frac{\cos\theta_0}{\cos\theta}, \quad \tilde{u} = \frac{u^*}{V\mu}, \quad \tilde{v} = \frac{v^*}{V\mu} \tag{2.229}$$

$$f_0 \equiv 2\Omega\sin\theta_0, \quad \frac{D}{Dt} \equiv \mu^2\left(\tilde{u}\frac{\partial}{\partial x} + \tilde{v}\frac{\partial}{\partial y}\right) + \frac{\mathrm{Fr}^2 M_{Ro}}{\mathrm{Ro}}w\frac{\partial}{\partial z} \tag{2.230}$$

$$x \equiv \frac{r}{L}\phi\cos\theta_0, \quad y \equiv \frac{r}{L}\int_{\theta_0}^{\theta}\frac{\cos\theta_0\,\mathrm{d}\theta'}{\cos\theta'}, \quad z \equiv \lambda\frac{r-r_0}{L} \tag{2.231}$$

$$t^* = t\frac{L}{V}, \quad b^* = b\frac{Vf_0 M_{Ro}}{\lambda} \tag{2.232}$$

$$\pi^* = \pi V f_0 L M_{Ro}, \quad w^* = wV\frac{\mathrm{Fr}^2\lambda M_{Ro}}{\mathrm{Ro}} \tag{2.233}$$

$$\mathrm{Ro} \equiv \frac{V}{f_0 L}, \quad M_{Ro} \equiv \max[1, \mathrm{Ro}] \tag{2.234}$$

$$\mathrm{Fr} \equiv \frac{V}{N\lambda L}, \quad \mathrm{Re} \equiv \frac{VL}{\nu}, \quad \mathrm{Pr} \equiv \frac{\nu}{\kappa} \tag{2.235}$$

Dimensional variables are denoted by an asterisk where necessary. If we filter over a grid scale typical for ocean models ($1m < L < 100km$, $0.0001 < \lambda < 1$, $0.001m/s < V < 1m/s$, $f_0 < 0.0001s^{-1}$, $0.01s^{-1} < N < 0.0001s^{-1}$), these equations are very well approximated by

$$\begin{aligned}
\mathrm{Ro}\frac{D\tilde{u}}{Dt} - \frac{\tilde{v}\sin\theta}{\sin\theta_0} + M_{Ro}\frac{\partial\pi}{\partial x} &= -\frac{\lambda\mathrm{Fr}^2 M_{Ro}\cos\theta}{\mu\sin\theta_0}w + \frac{\mathrm{Ro}\nabla^2\tilde{u}}{\mathrm{Re}} && (2.236)\\
\mathrm{Ro}\frac{D\tilde{v}}{Dt} + \frac{\tilde{u}\sin\theta}{\sin\theta_0} + M_{Ro}\frac{\partial\pi}{\partial y} &= \frac{\mathrm{Ro}\nabla^2\tilde{v}}{\mathrm{Re}} && (2.237)\\
\mathrm{Fr}^2\lambda^2\frac{Dw}{Dt} - b + \frac{\partial\pi}{\partial z} &= \frac{\lambda\cot\theta_0\tilde{u}}{M_{Ro}} + \frac{\mathrm{Fr}^2\lambda^2\nabla^2 w}{\mathrm{Re}} && (2.238)\\
\frac{Db}{Dt} + w &= \frac{\nabla^2 b}{\mathrm{Pr}\,\mathrm{Re}} && (2.239)\\
\mu^2\left(\frac{\partial\tilde{u}}{\partial x} + \frac{\partial\tilde{v}}{\partial y}\right) + \frac{\partial w}{\partial z} &= 0 && (2.240)\\
\nabla^2 &\approx \left(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} + \frac{\partial^2}{\lambda^2\partial z^2}\right) && (2.241)
\end{aligned}$$

Neglecting the non-frictional terms on the right-hand side is usually called the 'traditional' approximation. It is appropriate, with either large aspect ratio or far from the tropics. This approximation is used here, as it does not affect the form of the eddy stresses which is the main topic. The frictional terms are preserved in this approximate form for later comparison with eddy stresses.

Chapter 3

Getting started with MITgcm

This chapter is divided into two main parts. The first part, which is covered in sections 3.1 through 3.7, contains information about how to run experiments using MITgcm. The second part, covered in sections 3.9 through 3.20, 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 2 and 4.

3.1 Where to find information

There is a web-archived support mailing list for the model that you can email at `MITgcm-support@mitgcm.org` after subscribing to:

`http://mailman.mitgcm.org/mailman/listinfo/mitgcm-support/`

or browse at:

`http://mailman.mitgcm.org/pipermail/mitgcm-support/`

3.2 Obtaining the code

MITgcm can be downloaded from our system by following the instructions below. As a courtesy we ask that you send e-mail to us at MITgcm-support@mitgcm.org to enable us to keep track of who's using the model and in what application. You can download the model two ways:

1. Using CVS software. CVS is a freely available source code management tool. To use CVS you need to have the software installed. Many systems come with CVS pre-installed, otherwise good places to look for the software for a particular platform are cvshome.org and wincvs.org .

2. Using a tar file. This method is simple and does not require any special software. However, this method does not provide easy support for maintenance updates.

3.2.1 Method 1 - Checkout from CVS

If CVS is available on your system, we strongly encourage you to use it. CVS provides an efficient and elegant way of organizing your code and keeping track of your changes. If CVS is not available on your machine, you can also download a tar file.

Before you can use CVS, the following environment variable(s) should be set within your shell. For a csh or tcsh shell, put the following

```
% setenv CVSROOT :pserver:cvsanon@mitgcm.org:/u/gcmpack
```

in your .cshrc or .tcshrc file. For bash or sh shells, put:

```
% export CVSROOT=':pserver:cvsanon@mitgcm.org:/u/gcmpack'
```

in your .profile or .bashrc file.

To get MITgcm through CVS, first register with the MITgcm CVS server using command:

```
% cvs login ( CVS password: cvsanon )
```

You only need to do a "cvs login" once.

To obtain the latest sources type:

```
% cvs co -P MITgcm
```

or to get a specific release type:

```
% cvs co -P -r checkpoint52i_post MITgcm
```

The CVS command "cvs co" is the abreviation of the full-name "cvs checkout" command and using the option "-P" (cvs co -P) will prevent to download unnecessary empty directories.

The MITgcm web site contains further directions concerning the source code and CVS. It also contains a web interface to our CVS archive so that one may easily view the state of files, revisions, and other development milestones:

```
http://mitgcm.org/viewvc/MITgcm/MITgcm/
```

As a convenience, the MITgcm CVS server contains aliases which are named subsets of the codebase. These aliases can be especially helpful when used over slow internet connections or on machines with restricted storage space. Table 3.1 contains a list of CVS aliases

Alias Name	**Information (directories) Contained**
MITgcm_code	Only the source code – none of the verification examples.
MITgcm_verif_basic	Source code plus a small set of the verification examples (global_ocean.90x40x15, aim.5l_cs, hs94.128x64x5, front_relax, and plume_on_slope).
MITgcm_verif_atmos	Source code plus all of the atmospheric examples.
MITgcm_verif_ocean	Source code plus all of the oceanic examples.
MITgcm_verif_all	Source code plus all of the verification examples.

Table 3.1: MITgcm CVS Modules

The checkout process creates a directory called MITgcm. If the directory MITgcm exists this command updates your code based on the repository. Each directory in the source tree contains a directory CVS. This information is required by CVS to keep track of your file versions with respect to the repository. Don't edit the files in CVS! You can also use CVS to download code updates. More extensive information on using CVS for maintaining MITgcm code can be found here . It is important to note that the CVS aliases in Table 3.1 cannot be used in conjunction with the CVS -d DIRNAME option. However, the MITgcm directories they create can be changed to a different name following the check-out:

```
%  cvs co -P MITgcm_verif_basic
%  mv MITgcm MITgcm_verif_basic
```

Note that it is possible to checkout code without "cvs login" and without setting any shell environment variables by specifying the pserver name and password in one line, for example:

```
%  cvs -d :pserver:cvsanon:cvsanon@mitgcm.org:/u/gcmpack co -P MITgcm
```

3.2.1.1 Upgrading from an earlier version

If you already have an earlier version of the code you can "upgrade" your copy instead of downloading the entire repository again. First, "cd" (change directory) to the top of your working copy:

```
% cd MITgcm
```

and then issue the cvs update command such as:

```
% cvs -q update -d -P -r checkpoint52i_post
```

This will update the "tag" to "checkpoint52i_post", add any new directories (-d) and remove any empty directories (-P). The -q option means be quiet which will reduce the number of messages you'll see in the terminal. If you have modified the code prior to upgrading, CVS will try to merge your changes with the upgrades. If there is a conflict between your modifications and the upgrade, it will report that file with a "C" in front, e.g.:

```
C model/src/ini_parms.F
```

If the list of conflicts scrolled off the screen, you can re-issue the cvs update command and it will report the conflicts. Conflicts are indicated in the code by the delimites "<<<<<<<", "=======" and ">>>>>>>". For example,

```
<<<<<<< ini_parms.F
     & bottomDragLinear,myOwnBottomDragCoefficient,
=======
     & bottomDragLinear,bottomDragQuadratic,
>>>>>>> 1.18
```

means that you added "myOwnBottomDragCoefficient" to a namelist at the same time and place that we added "bottomDragQuadratic". You need to resolve this conflict and in this case the line should be changed to:

```
     & bottomDragLinear,bottomDragQuadratic,myOwnBottomDragCoefficient,
```

and the lines with the delimiters (<<<<<<,======,>>>>>>) be deleted. Unless you are making modifications which exactly parallel developments we make, these types of conflicts should be rare.

Upgrading to the current pre-release version We don't make a "release" for every little patch and bug fix in order to keep the frequency of upgrades to a minimum. However, if you have run into a problem for which "we have already fixed in the latest code" and we haven't made a "tag" or "release" since that patch then you'll need to get the latest code:

```
% cvs -q update -d -P -A
```

Unlike, the "check-out" and "update" procedures above, there is no "tag" or release name. The -A tells CVS to upgrade to the very latest version. As a rule, we don't recommend this since you might upgrade while we are in the processes of checking in the code so that you may only have part of a patch. Using this method of updating also means we can't tell what version of the code you are working with. So please be sure you understand what you're doing.

3.2.2 Method 2 - Tar file download

If you do not have CVS on your system, you can download the model as a tar file from the web site at:

```
http://mitgcm.org/download/
```

The tar file still contains CVS information which we urge you not to delete; even if you do not use CVS yourself the information can help us if you should need to send us your copy of the code. If a recent tar file does not exist, then please contact the developers through the MITgcm-support@mitgcm.org mailing list.

3.3 Model and directory structure

The "numerical" model is contained within a execution environment support wrapper. This wrapper is designed to provide a general framework for grid-point models. MITgcmUV is a specific numerical model that uses the framework. Under this structure the model is split into execution environment support code and conventional numerical model code. The execution environment support code is held under the `eesupp` directory. The grid point model code is held under the `model` directory. Code execution actually starts in the `eesupp` routines and not in the `model` routines. For this reason the top-level `MAIN.F` is in the `eesupp/src` directory. In general, end-users should not need to worry about this level. The top-level routine for the numerical part of the code is in `model/src/THE_MODEL_MAIN.F`. Here is a brief description of the directory structure of the model under the root tree (a detailed description is given in section 3: Code structure).

- `doc`: contains brief documentation notes.
- `eesupp`: contains the execution environment source code. Also subdivided into two subdirectories `inc` and `src`.
- `model`: this directory contains the main source code. Also subdivided into two subdirectories `inc` and `src`.
- `pkg`: contains the source code for the packages. Each package corresponds to a subdirectory. For example, `gmredi` contains the code related to the Gent-McWilliams/Redi scheme, `aim` the code relative to the atmospheric intermediate physics. The packages are described in detail in chapter 6.
- `tools`: this directory contains various useful tools. For example, `genmake2` is a script written in csh (C-shell) that should be used to generate your makefile. The directory `adjoint` contains the makefile specific to the Tangent linear and Adjoint Compiler (TAMC) that generates the adjoint code. The latter is described in detail in part 8. This directory also contains the subdirectory build_options, which contains the 'optfiles' with the compiler options for the different compilers and machines that can run MITgcm.
- `utils`: this directory contains various utilities. The subdirectory `knudsen2` contains code and a makefile that compute coefficients of the polynomial approximation to the knudsen formula for an ocean nonlinear equation of state. The `matlab` subdirectory contains matlab scripts for reading model output directly into matlab. `scripts` contains C-shell post-processing scripts for joining processor-based and tiled-based model output. The subdirectory exch2 contains the code needed for the exch2 package to work with different combinations of domain decompositions.
- `verification`: this directory contains the model examples. See section 3.8.
- `jobs`: contains sample job scripts for running MITgcm.
- `lsopt`: Line search code used for optimization.
- `optim`: Interface between MITgcm and line search code.

3.4 Building the code

To compile the code, we use the `make` program. This uses a file (`Makefile`) that allows us to pre-process source files, specify compiler and optimization options and also figures out any file dependencies. We supply a script (`genmake2`), described in section 3.4.2, that automatically creates the `Makefile` for you. You then need to build the dependencies and compile the code.

As an example, assume that you want to build and run experiment `verification/exp2`. The are multiple ways and places to actually do this but here let's build the code in `verification/exp2/build`:

```
% cd verification/exp2/build
```

First, build the `Makefile`:

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

The command line option tells `genmake` to override model source code with any files in the directory `../code/`.

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

Through the MITgcm-support list, the MITgcm developers are willing to provide help writing or modifing "optfiles". And we encourage users to post new "optfiles" (particularly ones for new machines or architectures) to the MITgcm-support@mitgcm.org list.

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

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

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

```
% make depend
```

This modifies the `Makefile` by attaching a (usually, long) list of files upon which other files depend. The purpose of this is to reduce re-compilation if and when you start to modify the code. The `make depend` command also creates links from the model source to this directory. It is important to note that the `make depend` stage will occasionally produce warnings or errors since the dependency parsing tool is unable to find all of the necessary header files (*eg.* `netcdf.inc`). In these circumstances, it is usually OK to ignore the warnings/errors and proceed to the next step.

Next one can compile the code using:

```
% make
```

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

```
% make -j 2
```

where the "2" can be replaced with a number that corresponds to the number of CPUs available.

Now you are ready to run the model. General instructions for doing so are given in section 3.5. Here, we can run the model by first creating links to all the input files:

```
ln -s ../input/* .
```

and then calling the executable with:

```
./mitgcmuv > output.txt
```

where we are re-directing the stream of text output to the file `output.txt`.

3.4.1 Building/compiling the code elsewhere

In the example above (section 3.4) we built the executable in the *input* directory of the experiment for convenience. You can also configure and compile the code in other locations, for example on a scratch disk with out having to copy the entire source tree. The only requirement to do so is you have `genmake2` in your path or you know the absolute path to `genmake2`.

The following sections outline some possible methods of organizing your source and data.

3.4.1.1 Building from the *../code directory*

This is just as simple as building in the *input/* directory:

```
% cd verification/exp2/code
% ../../../tools/genmake2
% make depend
% make
```

However, to run the model the executable (*mitgcmuv*) and input files must be in the same place. If you only have one calculation to make:

```
% cd ../input
% cp ../code/mitgcmuv ./
% ./mitgcmuv > output.txt
```

or if you will be making multiple runs with the same executable:

```
% cd ../
% cp -r input run1
% cp code/mitgcmuv run1
% cd run1
% ./mitgcmuv > output.txt
```

3.4.1.2 Building from a new directory

Since the *input* directory contains input files it is often more useful to keep *input* pristine and build in a new directory within *verification/exp2/*:

```
% cd verification/exp2
% mkdir build
% cd build
% ../../../tools/genmake2 -mods=../code
% make depend
% make
```

This builds the code exactly as before but this time you need to copy either the executable or the input files or both in order to run the model. For example,

```
% cp ../input/* ./
% ./mitgcmuv > output.txt
```

or if you tend to make multiple runs with the same executable then running in a new directory each time might be more appropriate:

```
% cd ../
% mkdir run1
% cp build/mitgcmuv run1/
% cp input/* run1/
% cd run1
% ./mitgcmuv > output.txt
```

3.4.1.3 Building on a scratch disk

Model object files and output data can use up large amounts of disk space so it is often the case that you will be operating on a large scratch disk. Assuming the model source is in */MITgcm* then the following commands will build the model in */scratch/exp2-run1*:

```
% cd /scratch/exp2-run1
% ~/MITgcm/tools/genmake2 -rootdir=~/MITgcm \
  -mods=~/MITgcm/verification/exp2/code
% make depend
% make
```

To run the model here, you'll need the input files:

```
% cp ~/MITgcm/verification/exp2/input/* ./
% ./mitgcmuv > output.txt
```

As before, you could build in one directory and make multiple runs of the one experiment:

```
% cd /scratch/exp2
% mkdir build
% cd build
% ~/MITgcm/tools/genmake2 -rootdir=~/MITgcm \
  -mods=~/MITgcm/verification/exp2/code
% make depend
% make
% cd ../
% cp -r ~/MITgcm/verification/exp2/input run2
% cd run2
% ./mitgcmuv > output.txt
```

3.4.2 Using `genmake2`

To compile the code, first use the program `genmake2` (located in the `tools` directory) to generate a Makefile. `genmake2` is a shell script written to work with all "sh"–compatible shells including bash v1, bash v2, and Bourne. `genmake2` parses information from the following sources:

- a *gemake_local* file if one is found in the current directory
- command-line options
- an "options file" as specified by the command-line option `--optfile=/PATH/FILENAME`
- a *packages.conf* file (if one is found) with the specific list of packages to compile. The search path for file *packages.conf* is, first, the current directory and then each of the "MODS" directories in the given order (see below).

3.4.2.1 Optfiles in `tools/build_options` directory:

The purpose of the optfiles is to provide all the compilation options for particular "platforms" (where "platform" roughly means the combination of the hardware and the compiler) and code configurations. Given the combinations of possible compilers and library dependencies (*eg.* MPI and NetCDF) there may be numerous optfiles available for a single machine. The naming scheme for the majority of the optfiles shipped with the code is

OS_HARDWARE_COMPILER

where

OS is the name of the operating system (generally the lower-case output of the `'uname'` command)

HARDWARE is a string that describes the CPU type and corresponds to output from the `'uname -m'` command:

- **ia32** is for "x86" machines such as i386, i486, i586, i686, and athlon
- **ia64** is for Intel IA64 systems (eg. Itanium, Itanium2)
- **amd64** is AMD x86_64 systems
- **ppc** is for Mac PowerPC systems

COMPILER is the compiler name (generally, the name of the FORTRAN executable)

In many cases, the default optfiles are sufficient and will result in usable Makefiles. However, for some machines or code configurations, new "optfiles" must be written. To create a new optfile, it is generally best to start with one of the defaults and modify it to suit your needs. Like `genmake2`, the optfiles are all written using a simple "sh"–compatible syntax. While nearly all variables used within `genmake2` may be specified in the optfiles, the critical ones that should be defined are:

FC the FORTRAN compiler (executable) to use

DEFINES the command-line DEFINE options passed to the compiler

CPP the C pre-processor to use

NOOPTFLAGS options flags for special files that should not be optimized

For example, the optfile for a typical Red Hat Linux machine ("ia32" architecture) using the GCC (g77) compiler is

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

If you write an optfile for an unrepresented machine or compiler, you are strongly encouraged to submit the optfile to the MITgcm project for inclusion. Please send the file to the

MITgcm-support@mitgcm.org

mailing list.

3.4.2.2 Command-line options:

In addition to the optfiles, `genmake2` supports a number of helpful command-line options. A complete list of these options can be obtained from:

```
% genmake2 -h
```

The most important command-line options are:

`--optfile=/PATH/FILENAME` specifies the optfile that should be used for a particular build.

If no "optfile" is specified (either through the command line or the MITGCM_OPTFILE environment variable), genmake2 will try to make a reasonable guess from the list provided in *tools/build_options*. The method used for making this guess is to first determine the combination of operating system and hardware (eg. "linux_ia32") and then find a working FORTRAN compiler within the user's path. When these three items have been identified, genmake2 will try to find an optfile that has a matching name.

`--mods='DIR1 DIR2 DIR3 ...'` specifies a list of directories containing "modifications". These directories contain files with names that may (or may not) exist in the main MITgcm source tree but will be overridden by any identically-named sources within the "MODS" directories.

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

- "MODS" directories (in the order given)

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

`--pgroups=/PATH/FILENAME` specifies the file where package groups are defined. If not set, the package-groups definition will be read from *pkg/pkg_groups*. It also contains the default list of packages (defined as the group "*default_pkg_list*" which is used when no specific package list (*packages.conf*) is found in current directory or in any "MODS" directory.

`--pdepend=/PATH/FILENAME` specifies the dependency file used for packages.

If not specified, the default dependency file *pkg/pkg_depend* is used. The syntax for this file is parsed on a line-by-line basis where each line containes either a comment ("#") or a simple "PKGNAME1 (+—-)PKGNAME2" pairwise rule where the "+" or "-" symbol specifies a "must be used with" or a "must not be used with" relationship, respectively. If no rule is specified, then it is assumed that the two packages are compatible and will function either with or without each other.

`--adof=/path/to/file` specifies the "adjoint" or automatic differentiation options file to be used. The file is analogous to the "optfile" defined above but it specifies information for the AD build process.

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

`--mpi` This option enables certain MPI features (using CPP `#define`s) within the code and is necessary for MPI builds (see Section 3.4.3).

`--make=/path/to/gmake` Due to the poor handling of soft-links and other bugs common with the `make` versions provided by commercial Unix vendors, GNU `make` (sometimes called `gmake`) should be preferred. This option provides a means for specifying the make executable to be used.

`--bash=/path/to/sh` On some (usually older UNIX) machines, the "bash" shell is unavailable. To run on these systems, `genmake2` can be invoked using an "sh" (that is, a Bourne, POSIX, or compatible) shell. The syntax in these circumstances is:

```
% /bin/sh genmake2 -bash=/bin/sh [...options...]
```

where `/bin/sh` can be replaced with the full path and name of the desired shell.

3.4.3 Building with MPI

Building MITgcm to use MPI libraries can be complicated due to the variety of different MPI implementations available, their dependencies or interactions with different compilers, and their often ad-hoc locations within file systems. For these reasons, its generally a good idea to start by finding and reading the documentation for your machine(s) and, if necessary, seeking help from your local systems administrator.

The steps for building MITgcm with MPI support are:

1. Determine the locations of your MPI-enabled compiler and/or MPI libraries and put them into an options file as described in Section 3.4.2. One can start with one of the examples in:

   ```
   MITgcm/tools/build_options/
   ```

   such as `linux_ia32_g77+mpi_cg01` or `linux_ia64_efc+mpi` and then edit it to suit the machine at hand. You may need help from your user guide or local systems administrator to determine the exact location of the MPI libraries. If libraries are not installed, MPI implementations and related tools are available including:

   - MPICH
   - LAM/MPI
   - MPIexec

2. Build the code with the `genmake2 -mpi` option (see Section 3.4.2) using commands such as:

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

3. Run the code with the appropriate MPI "run" or "exec" program provided with your particular implementation of MPI. Typical MPI packages such as MPICH will use something like:

```
%  mpirun -np 4 -machinefile mf ./mitgcmuv
```

   Sightly more complicated scripts may be needed for many machines since execution of the code may be controlled by both the MPI library and a job scheduling and queueing system such as PBS, LoadLeveller, Condor, or any of a number of similar tools. A few example scripts (those used for our regular verification runs) are available at: http://mitgcm.org/viewvc/MITgcm/MITgcm/tools/example_scripts/ or at: http://mitgcm.org/viewvc/MITgcm/MITgcm_contrib/test_scripts/

An example of the above process on the MITgcm cluster ("cg01") using the GNU g77 compiler and the mpich MPI library is:

```
% cd MITgcm/verification/exp5
% mkdir build
% cd build
% ../../../tools/genmake2 -mpi -mods=../code \
    -of=../../../../tools/build_options/linux_ia32_g77+mpi_cg01
% make depend
% make
% cd ../input
% /usr/local/pkg/mpi/mpi-1.2.4..8a-gm-1.5/g77/bin/mpirun.ch_gm \
    -machinefile mf --gm-kill 5 -v -np 2  ../build/mitgcmuv
```

3.5 Running the model in prognostic mode

If compilation finished succesfully (section 3.4) then an executable called `mitgcmuv` will now exist in the local directory.

To run the model as a single process (*ie.* not in parallel) simply type:

```
% ./mitgcmuv
```

The "./" is a safe-guard to make sure you use the local executable in case you have others that exist in your path (surely odd if you do!). The above command will spew out many lines of text output to your screen. This output contains details such as parameter values as well as diagnostics such as mean Kinetic energy, largest CFL number, etc. It is worth keeping this text output with the binary output so we normally re-direct the `stdout` stream as follows:

```
% ./mitgcmuv > output.txt
```

In the event that the model encounters an error and stops, it is very helpful to include the last few line of this `output.txt` file along with the (`stderr`) error message within any bug reports.

For the example experiments in `verification`, an example of the output is kept in `results/output.txt` for comparison. You can compare your `output.txt` with the corresponding one for that experiment to check that the set-up works.

3.5.1 Output files

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

3.5.1.1 MDSIO output files

The "traditional" output files are generated by the `mdsio` package. At a minimum, the instantaneous "state" of the model is written out, which is made of the following files:

- `U.00000nIter` - zonal component of velocity field (m/s and positive eastward).
- `V.00000nIter` - meridional component of velocity field (m/s and positive northward).
- `W.00000nIter` - vertical component of velocity field (ocean: m/s and positive upward, atmosphere: Pa/s and positive towards increasing pressure i.e. downward).
- `T.00000nIter` - potential temperature (ocean: °C, atmosphere: °K).
- `S.00000nIter` - ocean: salinity (psu), atmosphere: water vapor (g/kg).
- `Eta.00000nIter` - ocean: surface elevation (m), atmosphere: surface pressure anomaly (Pa).

The chain `00000nIter` consists of ten figures that specify the iteration number at which the output is written out. For example, `U.0000000300` is the zonal velocity at iteration 300.

In addition, a "pickup" or "checkpoint" file called:

- `pickup.00000nIter`

is written out. This file represents the state of the model in a condensed form and is used for restarting the integration. If the C-D scheme is used, there is an additional "pickup" file:

- `pickup_cd.00000nIter`

containing the D-grid velocity data and that has to be written out as well in order to restart the integration. Rolling checkpoint files are the same as the pickup files but are named differently. Their name contain the chain `ckptA` or `ckptB` instead of `00000nIter`. They can be used to restart the model but are overwritten every other time they are output to save disk space during long integrations.

3.5.1.2 MNC output files

Unlike the `mdsio` output, the `mnc`–generated output is usually (though not necessarily) placed within a subdirectory with a name such as `mnc_test_$DATE_$SEQ`.

3.5.2 Looking at the output

The "traditional" or mdsio model data are written according to a "meta/data" file format. Each variable is associated with two files with suffix names `.data` and `.meta`. The `.data` file contains the data written in binary form (big_endian by default). The `.meta` file is a "header" file that contains information about the size and the structure of the `.data` file. This way of organizing the output is particularly useful when running multi-processors calculations. The base version of the model includes a few matlab utilities to read output files written in this format. The matlab scripts are located in the directory `utils/matlab` under the root tree. The script `rdmds.m` reads the data. Look at the comments inside the script to see how to use it.

Some examples of reading and visualizing some output in *Matlab*:

```
% matlab
>> H=rdmds('Depth');
>> contourf(H');colorbar;
>> title('Depth of fluid as used by model');

>> eta=rdmds('Eta',10);
>> imagesc(eta');axis ij;colorbar;
>> title('Surface height at iter=10');

>> eta=rdmds('Eta',[0:10:100]);
>> for n=1:11; imagesc(eta(:,:,n)');axis ij;colorbar;pause(.5);end
```

Similar scripts for netCDF output (`rdmnc.m`) are available and they are described in Section 7.2.

The MNC output files are all in the "self-describing" netCDF format and can thus be browsed and/or plotted using tools such as:

- `ncdump` is a utility which is typically included with every netCDF install:

  `http://www.unidata.ucar.edu/packages/netcdf/`

  and it converts the netCDF binaries into formatted ASCII text files.

- `ncview` utility is a very convenient and quick way to plot netCDF data and it runs on most OSes:

  `http://meteora.ucsd.edu/~pierce/ncview_home_page.html`

- MatLAB(c) and other common post-processing environments provide various netCDF interfaces including:

  `http://mexcdf.sourceforge.net/`

  `http://woodshole.er.usgs.gov/staffpages/cdenham/public_html/MexCDF/nc4ml5.html`

3.6 Doing it yourself: customizing the model configuration

When you are ready to run the model in the configuration you want, the easiest thing is to use and adapt the setup of the case studies experiment (described previously) that is the closest to your configuration. Then, the amount of setup will be minimized. In this section, we focus on the setup relative to the "numerical model" part of the code (the setup relative to the "execution environment" part is covered in the parallel implementation section) and on the variables and parameters that you are likely to change.

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

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

3.6.1 Parameters: Computational domain, geometry and time-discretization

dimensions

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

grid

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

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

topography - full and partial cells

The domain bathymetry is read from a file that contains a 2D (x,y) map of depths (in m) for the ocean or pressures (in Pa) for the atmosphere. The file name is represented by the variable **bathyFile**. The file is assumed to contain binary numbers giving the depth (pressure) of the model at each grid cell, ordered with the x coordinate varying fastest. The points are ordered from low coordinate to high coordinate for both axes. The model code applies without modification to enclosed, periodic, and double periodic domains. Periodicity is assumed by default and is suppressed by setting the depths to 0m for the cells at the limits of the computational domain (note: not sure this is the case for the atmosphere). The precision with which to read the binary data is controlled by the integer variable **readBinaryPrec** which can take the value `32` (single precision) or `64` (double precision). See the matlab program *gendata.m* in the *input* directories under *verification* to see how the bathymetry files are generated for the case study experiments.

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

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

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

time-discretization

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

Name	**value**	**Description**	**Reference**
buoyancyRelation		buoyancyRelation = OCEANIC	
fluidIsAir	F	fluid major constituent is Air	
fluidIsWater	T	fuild major constituent is Water	
usingPCoords	F	use p (or p	
usingZCoords	T	use z (or z	
tRef	2.0E+01 at K= top	Reference temperature profile (oC or K)	
sRef	3.0E+01 at K= top	Reference salinity profile (psu)	
viscAh	0.0E+00	Lateral eddy viscosity (m^2/s)	
viscAhMax	1.0E+21	Maximum lateral eddy viscosity (m^2/s)	
viscAhGrid	0.0E+00	Grid dependent lateral eddy viscosity (non-dim.)	
useFullLeith	F	Use Full Form of Leith Viscosity on/off flag	
useStrainTensionVisc	F	Use StrainTension Form of Viscous Operator on/off flag	
useAreaViscLength	F	Use area for visc length instead of geom. mean	
viscC2leith	0.0E+00	Leith harmonic visc. factor (on grad(vort),non-dim.)	
viscC2leithD	0.0E+00	Leith harmonic viscosity factor (on grad(div),non-dim.)	
viscC2smag	0.0E+00	Smagorinsky harmonic viscosity factor (non-dim.)	
viscA4	0.0E+00	Lateral biharmonic viscosity (m^4/s)	
viscA4Max	1.0E+21	Maximum biharmonic viscosity (m^2/s)	
viscA4Grid	0.0E+00	Grid dependent biharmonic viscosity (non-dim.)	
viscC4leith	0.0E+00	Leith biharm viscosity factor (on grad(vort), non-dim.)	
viscC4leithD	0.0E+00	Leith biharm viscosity factor (on grad(div), non-dim.)	
viscC4Smag	0.0E+00	Smagorinsky biharm viscosity factor (non-dim)	
no_slip_sides	T	Viscous BCs: No-slip sides	
sideDragFactor	2.0E+00	side-drag scaling factor (non-dim)	
viscAr	0.0E+00	Vertical eddy viscosity (units of r^2/s)	
no_slip_bottom	T	Viscous BCs: No-slip bottom	
bottomDragLinear	0.0E+00	linear bottom-drag coefficient (m/s)	
bottomDragQuadratic	0.0E+00	quadratic bottom-drag coeff. (1)	
diffKhT	0.0E+00	Laplacian diffusion of heat laterally (m^2/s)	
diffK4T	0.0E+00	Bihaarmonic diffusion of heat laterally (m^4/s)	
diffKhS	0.0E+00	Laplacian diffusion of salt laterally (m^2/s)	
diffK4S	0.0E+00	Bihaarmonic diffusion of salt laterally (m^4/s)	
diffKrNrT	0.0E+00 at K= top	vertical profile of vertical diffusion of Temp (m^2/s)	
diffKrNrS	0.0E+00 at K= top	vertical profile of vertical diffusion of Salt (m^2/s)	
diffKrBL79surf	0.0E+00	Surface diffusion for Bryan and Lewis 1979 (m^2/s)	
diffKrBL79deep	0.0E+00	Deep diffusion for Bryan and Lewis 1979 (m^2/s)	
diffKrBL79scl	2.0E+02	Depth scale for Bryan and Lewis 1979 (m)	
diffKrBL79Ho	-2.0E+03	Turning depth for Bryan and Lewis 1979 (m)	
eosType	LINEAR	Equation of State	
tAlpha	2.0E-04	Linear EOS thermal expansion coefficient (1/oC)	

Name	**value**	**Description**	**Reference**
sBeta	7.4E-04	Linear EOS haline contraction coef (1/psu)	
rhonil	9.998E+02	Reference density (kg/m^3)	
rhoConst	9.998E+02	Reference density (kg/m^3)	
rhoConstFresh	9.998E+02	Reference density (kg/m^3)	
gravity	9.81E+00	Gravitational acceleration (m/s^2)	
gBaro	9.81E+00	Barotropic gravity (m/s^2)	
rotationPeriod	8.6164E+04	Rotation Period (s)	
omega	7.292123516990375E-05	Angular velocity (rad/s)	
f0	1.0E-04	Reference coriolis parameter (1/s)	
beta	9.999999999999999E-12	Beta (1/(m.s))	
freeSurfFac	1.0E+00	Implicit free surface factor	
implicitFreeSurface	T	Implicit free surface on/off flag	
rigidLid	F	Rigid lid on/off flag	
implicSurfPress	1.0E+00	Surface Pressure implicit factor (0-1)	
implicDiv2Dflow	1.0E+00	Barot. Flow Div. implicit factor (0-1)	
exactConserv	F	Exact Volume Conservation on/off flag	
uniformLin_PhiSurf	T	use uniform Bo_surf on/off flag	
nonlinFreeSurf	0	Non-linear Free Surf. options (-1,0,1,2,3)	
hFacInf	2.0E-01	lower threshold for hFac (nonlinFreeSurf only)	
hFacSup	2.0E+00	upper threshold for hFac (nonlinFreeSurf only)	
select_rStar	0	r	
useRealFreshWaterFlux	F	Real Fresh Water Flux on/off flag	
convertFW2Salt	3.5E+01	convert F.W. Flux to Salt Flux (-1=use local S)	
use3Dsolver	F	use 3-D pressure solver on/off flag	
nonHydrostatic	F	Non-Hydrostatic on/off flag	
nh_Am2	1.0E+00	Non-Hydrostatic terms scaling factor	
quasiHydrostatic	F	Quasi-Hydrostatic on/off flag	
momStepping	T	Momentum equation on/off flag	
vectorInvariantMomentum	F	Vector-Invariant Momentum on/off	
momAdvection	T	Momentum advection on/off flag	
momViscosity	T	Momentum viscosity on/off flag	
momImplVertAdv	F	Momentum implicit vert. advection on/off	
implicitViscosity	F	Implicit viscosity on/off flag	
metricTerms	F	metric-Terms on/off flag	
useNHMTerms	F	Non-Hydrostatic Metric-Terms on/off	
useCoriolis	T	Coriolis on/off flag	
useCDscheme	F	CD scheme on/off flag	
useJamartWetPoints	F	Coriolis WetPoints method flag	
useJamartMomAdv	F	V.I. Non-linear terms Jamart flag	

Name	**value**	**Description**	**Reference**
SadournyCoriolis	F	Sadourny Coriolis discr. flag	
upwindVorticity	F	Upwind bias vorticity flag	
useAbsVorticity	F	Work with f	
highOrderVorticity	F	High order interp. of vort. flag	
upwindShear	F	Upwind vertical Shear advection flag	
selectKEscheme	0	Kinetic Energy scheme selector	
momForcing	T	Momentum forcing on/off flag	
momPressureForcing	T	Momentum pressure term on/off flag	
implicitIntGravWave	F	Implicit Internal Gravity Wave flag	
staggerTimeStep	F	Stagger time stepping on/off flag	
multiDimAdvection	T	enable/disable Multi-Dim Advection	
useMultiDimAdvec	F	Multi-Dim Advection is/is-not used	
implicitDiffusion	F	Implicit Diffusion on/off flag	
tempStepping	T	Temperature equation on/off flag	
tempAdvection	T	Temperature advection on/off flag	
tempImplVertAdv	F	Temp. implicit vert. advection on/off	
tempForcing	T	Temperature forcing on/off flag	
saltStepping	T	Salinity equation on/off flag	
saltAdvection	T	Salinity advection on/off flag	
saltImplVertAdv	F	Sali. implicit vert. advection on/off	
saltForcing	T	Salinity forcing on/off flag	
readBinaryPrec	32	Precision used for reading binary files	
writeBinaryPrec	32	Precision used for writing binary files	
globalFiles	F	write ”global” (=not per tile) files	
useSingleCpuIO	F	only master MPI process does I/O	
debugMode	F	Debug Mode on/off flag	
debLevA	1	1rst level of debugging	
debLevB	2	2nd level of debugging	
debugLevel	1	select debugging level	
cg2dMaxIters	150	Upper limit on 2d con. grad iterations	
cg2dChkResFreq	1	2d con. grad convergence test frequency	
cg2dTargetResidual	1.0E-07	2d con. grad target residual	
cg2dTargetResWunit	-1.0E+00	CG2d target residual [W units]	
cg2dPreCondFreq	1	Freq. for updating cg2d preconditioner	
nIter0	0	Run starting timestep number	
nTimeSteps	0	Number of timesteps	
deltatTmom	6.0E+01	Momentum equation timestep (s)	
deltaTfreesurf	6.0E+01	FreeSurface equation timestep (s)	
dTtracerLev	6.0E+01 at K= top	Tracer equation timestep (s)	
deltatTClock	6.0E+01	Model clock timestep (s)	

Name	value	Description	Reference
cAdjFreq	0.0E+00	Convective adjustment interval (s)	
momForcingOutAB	0	=1: take Momentum Forcing out of Adams-Bash.	
tracForcingOutAB	0	=1: take T,S,pTr Forcing out of Adams-Bash.	
momDissip_In_AB	T	put Dissipation Tendency in Adams-Bash.	
doAB_onGtGs	T	apply AB on Tendencies (rather than on T,S)	
abEps	1.0E-02	Adams-Bashforth-2 stabilizing weight	
baseTime	0.0E+00	Model base time (s).	
startTime	0.0E+00	Run start time (s).	
endTime	0.0E+00	Integration ending time (s).	
pChkPtFreq	0.0E+00	Permanent restart/checkpoint file interval (s).	
chkPtFreq	0.0E+00	Rolling restart/checkpoint file interval (s).	
pickup_write_mdsio	T	Model IO flag.	
pickup_read_mdsio	T	Model IO flag.	
pickup_write_immed	F	Model IO flag.	
dumpFreq	0.0E+00	Model state write out interval (s).	
dumpInitAndLast	T	write out Initial and Last iter. model state	
snapshot_mdsio	T	Model IO flag.	
monitorFreq	6.0E+01	Monitor output interval (s).	
monitor_stdio	T	Model IO flag.	
externForcingPeriod	0.0E+00	forcing period (s)	
externForcingCycle	0.0E+00	period of the cyle (s).	
tauThetaClimRelax	0.0E+00	relaxation time scale (s)	
tauSaltClimRelax	0.0E+00	relaxation time scale (s)	
latBandClimRelax	3.703701E+05	max. Lat. where relaxation	
usingCartesianGrid	T	Cartesian coordinates flag (True / False)	
usingSphericalPolarGrid	F	Spherical coordinates flag (True / False)	
usingCylindricalGrid	F	Spherical coordinates flag (True / False)	
Ro_SeaLevel	0.0E+00	r(1) (units of r)	
rkSign	-1.0E+00	index orientation relative to vertical coordinate	
horiVertRatio	1.0E+00	Ratio on units : Horiz - Vertical	
drC	5.0E+03 at K=1	C spacing (units of r)	
drF	1.0E+04 at K= top	W spacing (units of r)	
delX	1.234567E+05 at I= east	U spacing (m - cartesian, degrees - spherical)	
delY	1.234567E+05 at J=1	V spacing (m - cartesian, degrees - spherical)	
ygOrigin	0.0E+00	South edge Y-axis origin (cartesian: m, spherical: deg.)	
xgOrigin	0.0E+00	West edge X-axis origin (cartesian: m, spherical: deg.)	
rSphere	6.37E+06	Radius (ignored - cartesian, m - spherical)	
xcoord	6.172835E+04 at I=1	P-point X coord (m - cartesian, degrees - spherical)	
ycoord	6.172835E+04 at J=1	P-point Y coord (m - cartesian, degrees - spherical)	
rcoord	-5.0E+03 at K=1	P-point R coordinate (units of r)	
rF	0.0E+00 at K=1	W-Interf. R coordinate (units of r)	
dBdrRef	0.0E+00 at K= top	Vertical gradient of reference boyancy $[(m/s/r)^2)]$	

Name	value	Description	Reference
dxF	1.234567E+05 at K= top	dxF(:,1,:,1) (m - cartesian, degrees - spherical)	
dyF	1.234567E+05 at I= east	dyF(:,1,:,1) (m - cartesian, degrees - spherical)	
dxG	1.234567E+05 at I= east	dxG(:,1,:,1) (m - cartesian, degrees - spherical)	
dyG	1.234567E+05 at I= east	dyG(:,1,:,1) (m - cartesian, degrees - spherical)	
dxC	1.234567E+05 at I= east	dxC(:,1,:,1) (m - cartesian, degrees - spherical)	
dyC	1.234567E+05 at I= east	dyC(:,1,:,1) (m - cartesian, degrees - spherical)	
dxV	1.234567E+05 at I= east	dxV(:,1,:,1) (m - cartesian, degrees - spherical)	
dyU	1.234567E+05 at I= east	dyU(:,1,:,1) (m - cartesian, degrees - spherical)	
rA	1.524155677489E+10 at I= east	rA(:,1,:,1) (m - cartesian, degrees - spherical)	
rAw	1.524155677489E+10 at K= top	rAw(:,1,:,1) (m - cartesian, degrees - spherical)	
rAs	1.524155677489E+10 at K= top	rAs(:,1,:,1) (m - cartesian, degrees - spherical)	

Name	Default value	Description	Reference
tempAdvScheme	2	Temp. Horiz.Advection scheme selector	
tempVertAdvScheme	2	Temp. Vert. Advection scheme selector	
tempMultiDimAdvec	F	use Muti-Dim Advec method for Temp	
tempAdamsBashforth	T	use Adams-Bashforth time-stepping for Temp	
saltAdvScheme	2	Salt. Horiz.advection scheme selector	
saltVertAdvScheme	2	Salt. Vert. Advection scheme selector	
saltMultiDimAdvec	F	use Muti-Dim Advec method for Salt	
saltAdamsBashforth	T	use Adams-Bashforth time-stepping for Salt	

3.6.2 Parameters: Equation of state

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

The form of the equation of state is controlled by the character variables **buoyancyRelation** and **eosType**. **buoyancyRelation** is set to `'OCEANIC'` by default and needs to be set to `'ATMOSPHERIC'` for atmosphere simulations. In this case, **eosType** must be set to `'IDEALGAS'`. For the ocean, two forms of the equation of state are available: linear (set **eosType** to `'LINEAR'`) and a polynomial approximation to the full nonlinear equation (set **eosType** to `'POLYNOMIAL'`). In the linear case, you need to specify the thermal and haline expansion coefficients represented by the variables **tAlpha** (in K^{-1}) and **sBeta** (in ppt^{-1}). For the nonlinear case, you need to generate a file of polynomial coefficients called *POLY3.COEFFS*. To do this, use the program *utils/knudsen2/knudsen2.f* under the model tree (a Makefile is available in the same directory and you will need to edit the number and the values of the vertical levels in *knudsen2.f* so that they match those of your configuration).

There there are also higher polynomials for the equation of state:

`'UNESCO'`: The UNESCO equation of state formula of Fofonoff and Millard *Fofonoff and Millard* [1983]. This equation of state assumes in-situ temperature, which is not a model variable; *its use is therefore discouraged, and it is only listed for completeness.*

`'JMD95Z'`: A modified UNESCO formula by Jackett and McDougall *Jackett and McDougall* [1995], which uses the model variable potential temperature as input. The `'Z'` indicates that this equation of state uses a horizontally and temporally constant pressure $p_0 = -g\rho_0 z$.

`'JMD95P'`: A modified UNESCO formula by Jackett and McDougall *Jackett and McDougall* [1995], which uses the model variable potential temperature as input. The `'P'` indicates that this equation of state uses the actual hydrostatic pressure of the last time step. Lagging the pressure in this way requires an additional pickup file for restarts.

`'MDJWF'`: The new, more accurate and less expensive equation of state by McDougall et al. *McDougall et al.* [2003]. It also requires lagging the pressure and therefore an additional pickup file for restarts.

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

3.6.3 Parameters: Momentum equations

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

initialization

The initial horizontal velocity components can be specified from binary files **uVelInitFile** and **vVelInitFile**. These files should contain 3D data ordered in an (x,y,r) fashion with k=1 as the first vertical level (surface level). If no file names are provided, the velocity is initialised to zero. The initial vertical velocity is always derived from the horizontal velocity using the continuity equation, even in the case of non-hydrostatic simulation (see, e.g.: *tutorial_deep_convection/input/data*).

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

forcing

This section only applies to the ocean. You need to generate wind-stress data into two files **zonalWindFile** and **meridWindFile** corresponding to the zonal and meridional components of the wind stress, respectively (if you want the stress to be along the direction of only one of the model horizontal axes, you only need to generate one file). The format of the files is similar to the bathymetry file. The zonal (meridional) stress data are assumed to be in Pa and located at U-points (V-points). As for the bathymetry, the precision with which to read the binary data is controlled by the variable **readBinaryPrec**. See the matlab program *gendata.m* in the *input* directories under *verification* to see how simple analytical wind forcing data are generated for the case study experiments.

There is also the possibility of prescribing time-dependent periodic forcing. To do this, concatenate the successive time records into a single file (for each stress component) ordered in a (x,y,t) fashion and set the following variables: **periodicExternalForcing** to `'.TRUE.'`, **externForcingPeriod** to the period (in s) of which the forcing varies (typically 1 month), and **externForcingCycle** to the repeat time (in s) of the forcing (typically 1 year – note: **externForcingCycle** must be a multiple of **externForcingPeriod**). With these variables set up, the model will interpolate the forcing linearly at each iteration.

dissipation

The lateral eddy viscosity coefficient is specified through the variable **viscAh** (in m^2s^{-1}). The vertical eddy viscosity coefficient is specified through the variable **viscAz** (in m^2s^{-1}) for the ocean and **viscAp** (in Pa^2s^{-1}) for the atmosphere. The vertical diffusive fluxes can be computed implicitly by setting the logical variable **implicitViscosity** to `'.TRUE.'`. In addition, biharmonic mixing can be added as well through the variable **viscA4** (in m^4s^{-1}). On a spherical polar grid, you might also need to set the variable **cosPower** which is set to 0 by default and which represents the power of cosine of latitude to multiply viscosity. Slip or no-slip conditions at lateral and bottom boundaries are specified through the logical variables **no_slip_sides** and **no_slip_bottom**. If set to `'.FALSE.'`, free-slip boundary conditions are applied. If no-slip boundary conditions are applied at the bottom, a bottom drag can be applied as well. Two forms are available: linear (set the variable **bottomDragLinear** in m/s) and quadratic (set the variable **bottomDragQuadratic**, dimensionless).

The Fourier and Shapiro filters are described elsewhere.

C-D scheme

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

calculation of pressure/geopotential

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

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

3.6.4 Parameters: Tracer equations

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

initialization

The initial tracer data can be contained in the binary files **hydrogThetaFile** and **hydrogSaltFile**. These files should contain 3D data ordered in an (x,y,r) fashion with k=1 as the first vertical level. If no file names are provided, the tracers are then initialized with the values of **tRef** and **sRef** mentioned above (in the equation of state section). In this case, the initial tracer data are uniform in x and y for each depth level.

forcing

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

A combination of fluxes data and relaxation terms can be used for driving the tracer equations. For potential temperature, heat flux data (in W/m^2) can be stored in the 2D binary file **surfQfile**. Alternatively or in addition, the forcing can be specified through a relaxation term. The SST data to which the model surface temperatures are restored to are supposed to be stored in the 2D binary file **thetaClimFile**. The corresponding relaxation time scale coefficient is set through the variable **tauThetaClimRelax** (in s). The same procedure applies for salinity with the variable names **EmPmRfile**, **saltClimFile**, and **tauSaltClimRelax** for freshwater flux (in m/s) and surface salinity (in ppt) data files and relaxation time scale coefficient (in s), respectively. Also for salinity, if the CPP key **USE_NATURAL_BCS** is turned on, natural boundary conditions are applied i.e. when computing the surface salinity tendency, the freshwater flux is multiplied by the model surface salinity instead of a constant salinity value.

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

dissipation

Lateral eddy diffusivities for temperature and salinity/specific humidity are specified through the variables **diffKhT** and **diffKhS** (in m^2/s). Vertical eddy diffusivities are specified through the variables **diffKzT** and **diffKzS** (in m^2/s) for the ocean and **diffKpT** and **diffKpS** (in Pa^2/s) for the atmosphere. The vertical diffusive fluxes can be computed implicitly by setting the logical variable **implicitDiffusion** to `'.TRUE.'`. In addition, biharmonic diffusivities can be specified as well through the coefficients **diffK4T** and **diffK4S** (in m^4/s). Note that the cosine power scaling (specified through **cosPower**—see the momentum equations section) is applied to the tracer diffusivities (Laplacian and biharmonic) as well. The Gent and McWilliams parameterization for oceanic tracers is described in the package section. Finally, note that tracers can be also subject to Fourier and Shapiro filtering (see the corresponding section on these filters).

ocean convection

Two options are available to parameterize ocean convection: one is to use the convective adjustment scheme. In this case, you need to set the variable **cadjFreq**, which represents the frequency (in s) with which the adjustment algorithm is called, to a non-zero value (if set to a negative value by the user, the model will set it to the tracer time step). The other option is to parameterize convection with implicit vertical diffusion. To do this, set the logical variable **implicitDiffusion** to `'.TRUE.'` and the real variable **ivdc_kappa** to a value (in m^2/s) you wish the tracer vertical diffusivities to have when mixing tracers vertically due to static instabilities. Note that **cadjFreq** and **ivdc_kappa**can not both have non-zero value.

3.6.5 Parameters: Simulation controls

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

run duration

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

frequency of output

Real variables defining frequencies (in s) with which output files are written on disk need to be set up. **dumpFreq** controls the frequency with which the instantaneous state of the model is saved. **chkPtFreq** and **pchkPtFreq** control the output frequency of rolling and permanent checkpoint files, respectively. See section 1.5.1 Output files for the definition of model state and checkpoint files. In addition, time-averaged fields can be written out by setting the variable **taveFreq** (in s). The precision with which to write the binary data is controlled by the integer variable w**riteBinaryPrec** (set it to `32` or `64`).

3.7 Testing

A script (`testreport`) for automated testing is included in the model within the *verification* directory. While intended mostly for advanced users, the script can be helpful for beginners.

3.7.1 Using `testreport`

On many systems, the program can be run with the command:

```
% cd verification
% ./testreport
```

which will do the following:

1. Locate all "valid" test directories. Here, valid tests are defined to be those directories within the current directory (which is generally *verification*) that contain a subdirectory and file with the names *results/output.txt*.

2. Then within each valid test:

   (a) run `genmake2` to produce a *Makefile*

   (b) build an executable

   (c) run the executable

   (d) compare and the output of the executable with the contents of certain variables within *TEST-NAME/results/output.txt*

   (e) print and, if requested (with the `-addr=EMAIL_ADDRESS` option), send a MIME-encoded email with the testing results

For further details, please see the MITgcm Developers' HOWTO at:

`http://mitgcm.org/public/docs.html`

3.7.2 Automated testing

Automated testing results are produced on a regular basis and they can be viewed at:

`http://mitgcm.org/public/testing.html`

which also includes links to various scripts for batch job submission on a variety of different machines.

3.8 Example experiments

The full MITgcm distribution comes with a set of pre-configured numerical experiments. Some of these example experiments are tests of individual parts of the model code, but many are fully fledged numerical simulations. Full tutorials exist for a few of the examples, and are documented in sections 3.9 - 3.21. The other examples follow the same general structure as the tutorial examples. However, they only include brief instructions in a text file called *README*. The examples are located in subdirectories under the directory `verification`. Each example is briefly described below.

3.8.1 Full list of model examples

1. `tutorial_advection_in_gyre` - Test of various advection schemes in a single-layer double-gyre experiment. This experiment is described in detail in section 3.11.
2. `tutorial_baroclinic_gyre` - Four layer, ocean double gyre. This experiment is described in detail in section 3.10.
3. `tutorial_barotropic_gyre` - Single layer, ocean double gyre (barotropic with free-surface). This experiment is described in detail in section 3.9.
4. `tutorial_cfc_offline` - Offline form of the MITgcm to study advection of a passive tracer and CFCs. This experiment is described in detail in section 3.20.5.
5. `tutorial_deep_convection` - Non-uniformly forced ocean convection in a doubly periodic box. This experiment is described in detail in section 3.15.
6. `tutorial_dic_adoffline` - Offline form of MITgcm dynamics coupled to the dissolved inorganic carbon biogeochemistry model; adjoint set-up.
7. `tutorial_global_oce_biogeo` - Ocean model coupled to the dissolved inorganic carbon biogeochemistry model. This experiment is described in detail in section 3.17.
8. `tutorial_global_oce_in_p` - Global ocean simulation in pressure coordinate (non-Boussinesq ocean model). Described in detail in section 3.13.
9. `tutorial_global_oce_latlon` - 4x4 degree global ocean simulation with steady climatological forcing. This experiment is described in detail in section 3.12.
10. `tutorial_global_oce_optim` - Global ocean state estimation at 4° resolution. This experiment is described in detail in section 3.18.
11. `tutorial_held_suarez_cs` - 3D atmosphere dynamics using Held and Suarez (1994) forcing on cubed sphere grid. This experiment is described in detail in section 3.14.
12. `tutorial_offline` - Offline form of the MITgcm to study advection of a passive tracer. This experiment is described in detail in section 3.20.
13. `tutorial_plume_on_slope` - Gravity Plume on a continental slope. This experiment is described in detail in section 3.16.
14. `tutorial_tracer_adjsens` - Simple passive tracer experiment. Includes derivative calculation. This experiment is described in detail in section 3.19.
    Also contains an additional set-up using Secon Order Moment (SOM) advection scheme (*input_ad.som81/*).
15. `1D_ocean_ice_column` - Oceanic column with seaice on top.
16. `adjustment.128x64x1` - Barotropic adjustment problem on latitude longitude grid with 128x64 grid points (2.8° resolution).
17. `adjustment.cs-32x32x1` - Barotropic adjustment problem on cube sphere grid with 32x32 points per face (roughly 2.8° resolution).
    Also contains a non-linear free-surface adjustment version (*input.nlfs/*).

18. `advect_cs` - Two-dimensional passive advection test on cube sphere grid (32x32 grid points per face, roughly 2.8° resolution)

19. `advect_xy` - Two-dimensional (horizontal plane) passive advection test on Cartesian grid.
Also contains an additional set-up using Adams-Bashforth 3 (*input.ab3_c4/*).

20. `advect_xz` - Two-dimensional (vertical plane) passive advection test on Cartesian grid.
Also contains an additional set-up using non-linear free-surface with divergent barotropic flow and implicit vertical advection (*input.nlfs/*).

21. `aim.5l_Equatorial_Channel` - 5-levels Intermediate Atmospheric physics, 3D Equatorial Channel configuration.

22. `aim.5l_LatLon` - 5-levels Intermediate Atmospheric physics, Global configuration, on latitude longitude grid with 128x64x5 grid points (2.8° resolution).

23. `aim.5l_cs` - 5-levels Intermediate Atmospheric physics, Global configuration on cube sphere grid (32x32 grid points per face, roughly 2.8°).
Also contains an additional set-up with a slab-ocean and thermodynamic sea-ice (*input.thSI/*).

24. `bottom_ctrl_5x5` - Adjoint test using the bottom topography as the control parameter.

25. `cfc_example` - Global ocean with online computation and advection of CFC11 and CFC12.

26. `cheapAML_box` - Example using cheap atmospheric mixed layer (cheapAML) package.

27. `cpl_aim+ocn` - Coupled Ocean - Atmosphere realistic configuration on cubed-sphere cs32 horizontal grid, using Intermediate Atmospheric physics (*pkg/aim_v23*) thermodynamic seaice (*pkg/thsice*) and land packages. on cubed-sphere cs32 in a realistic configuration.

28. `cpl_atm2d+ocn` - Coupled Ocean - Atmosphere realistic configuration using 2-D Atmospheric Model (*pkg/atm2d*).

29. `deep_anelastic` - Convection simulation on a giant planet: relax both the Boussinesq approximation (anelastic) and the thin atmosphere approximation (deep atmosphere).

30. `dome` - Idealized 3D test of a density-driven bottom current.

31. `exp2` - Old version of the global ocean experiment (no GM, no partial-cells).
Also contains an additional set-up with rigid-lid (*input.rigidLid/*).

32. `exp4` - Flow over a Gaussian bump in open-water or channel with open boundaries.
Also contains an additional set-up using non-linear free-surface (*input.nlfs/*).

33. `fizhi-cs-32x32x40` - Global atmospheric simulation with realistic topography, 40 vertical levels, a cubed sphere grid and the full atmospheric physics package.

34. `fizhi-cs-aqualev20` - Global atmospheric simulation on an aqua planet with full atmospheric physics. Run is perpetual march with an analytical SST distribution. This is the configuration for the APE (Aqua Planet Experiment) participation experiment.

35. `fizhi-gridalt-hs` - Global atmospheric simulation Held-Suarez (1994) forcing, with the physical forcing and the dynamical forcing running on different vertical grids.

36. `flt_example` - Example of using float package.

37. `front_relax` - Relaxation of an ocean thermal front (test for Gent/McWilliams scheme). 2D (y-z).
Also contains additional set-ups:

    (a) using the Boundary-Value Problem method (Ferrari et al., 2010) (*input.bvp/*).

    (b) with Mixed-Layer Eddy parameterization (Ferrari & McWilliams, 2007) (*input.mxl/*).

38. `global_ocean.90x40x15` - Global ocean simulation at 4x4 degree resolution. Similar to tutorial_global_oce_latlon, but using z^* coordinates with quasi-non-hydrostatic and non-hydrostatic metric terms. This experiment also illustrate the use of SBO package. Also contains additional set-ups:
    (a) using down-slope package (*pkg/down_slope*) (*input.dwnslp/*)
    (b) an Open-AD adjoint set-up (*code_oad/, input_oad/*).
    (c) four TAF adjoint set-ups (*code_ad/*):
        i. standard experiment (*input_ad/*).
        ii. with bottom drag as a control (*input_ad.bottomdrag/*).
        iii. with kappa GM as a control (*input_ad.kapgm/*).
        iv. with kappa Redi as a control (*input_ad.kapredi/*).
39. `global_ocean.cs32x15` - Global ocean experiment on the cubed sphere grid.
    Also contains additional forward set-ups:
    (a) non-hydrostatic with biharmonic viscosity (*input.viscA4/*)
    (b) using thermodynamic sea ice and bulk force (*input.thsice/*)
    (c) using thermodynamic (*pkg/thsice*) dynamic (*pkg/seaice*) sea-ice and *exf* package (*input.icedyn/*)
    (d) using thermodynamic - dynamic (*pkg/seaice*) sea-ice with *exf* package (*input.seaice/*)

    and few additional adjoint set-ups (*code_ad/*):
    (a) standard experiment without sea-ice (*input_ad/*).
    (b) using thermodynamic - dynamic sea-ice (*input_ad.seaice/*)
    (c) same as above without adjoint sea-ice dynamics (*input_ad.seaice_dynmix/*)
    (d) using thermodynamic sea-ice from *thsice* package (*input_ad.thsice/*)
40. `global_ocean_ebm` - Global ocean experiment on a lat-lon grid coupled to an atmospheric energy balance model. Similar to global_ocean.90x40x15 experiment.
    Also contains an adjoint set-up (*code_ad/, input_ad/*).
41. `global_with_exf` - Global ocean experiment on a lat-lon grid using the *exf* package. Similar to tutorial_global_oce_latlon experiment.
    Also contains a secondary set-up with yearly *exf* fields (*input_ad.yearly/*).
42. `halfpipe_streamice` - Example using package "streamice".
    Also contains adjoint set-ups using TAF (*code_ad/, input_ad/*) and using Open-AD (*code_oad/, input_oad/*).
43. `hs94.128x64x5` - 3D atmosphere dynamics on lat-lon grid, using Held and Suarez '94 forcing.
44. `hs94.1x64x5` - Zonal averaged atmosphere dynamics using Held and Suarez '94 forcing.
    Also contains adjoint set-ups using TAF (*code_ad/, input_ad/*) and using Open-AD (*code_oad/, input_oad/*).
45. `hs94.cs-32x32x5` - 3D atmosphere dynamics using Held and Suarez (1994) forcing on the cubed sphere, similar to tutorial_held_suarez_cs experiment but using linear free-surface and only 5 levels. Also contains an additional set-up with Implicit Internal gravity waves treatment and Adams-Bashforth 3 (*input.impIGW/*).
46. `ideal_2D_oce` - Idealized 2D global ocean simulation on an aqua planet.
47. `internal_wave` - Ocean internal wave forced by open boundary conditions.
    Also contains an additional set-up using *pkg/kl10* (see section 6.4.5, Klymak and Legg, 2010) (*input.kl10/*).
48. `inverted_barometer` - Simple test of ocean response to atmospheric pressure loading.

49. `isomip` - ISOMIP like set-up including ice-shelf cavities (*pkg/shelfice*).
Also contains additional set-ups:
    (a) with "htd" (*input.htd/*) but only Martin knows what "htd" stands for.
    (b) using package *icefront* (*input.icefront*)

    and also adjoint set-ups using TAF (*code_ad/, input_ad/, input_ad.htd/*) or using Open-AD (*code_oad/, input_oad/*).
50. `lab_sea` - Regional Labrador Sea simulation on a lat-lon grid using the sea ice package.
Also contains additional set-ups:
    (a) using the simple "free-drift" assumption for seaice (*input.fd/*)
    (b) using EVP dynamics (instead of LSR solver) and Hibler & Bryan (1987) sea-ice ocean stress (*input.hb87/*)
    (c) using package *salt_plume* (*input.salt_plume/*)

    and also 3 adjoint set-ups (*code_ad/, input_ad/, input_ad.noseaicedyn/, input_ad.noseaice/*).
51. `matrix_example` - Test of experimental method to accelerated convergence towards equilibrium.
52. `MLAdjust` - Simple tests for different viscosity formulations.
Also contains additional set-ups (see: *verification/MLAdjust/README*):
    (a) (*input.A4FlxF/*)
    (b) (*input.AhFlxF/*)
    (c) (*input.AhVrDv/*)
    (d) (*input.AhStTn/*)
53. `natl_box` - Eastern subtropical North Atlantic with KPP scheme; 1 month integration
54. `obcs_ctrl` - Adjoint test using Open-Boundary conditions as control parameters.
55. `offline_exf_seaice` - Seaice on top of oceanic surface layer in an idealized channel. Forcing is computed by bulk-formulae (*pkg/exf*) with temperature relaxation to prescribed SST (offline ocean).
Also contains additional set-ups:
    (a) sea-ice dynamics-only using JFNK solver and *pkg/thsice* advection (*input.dyn_jfnk/*)
    (b) sea-ice dynamics-only using LSR solver and *pkg/seaice* advection (*input.dyn_lsr/*)
    (c) sea-ice thermodynamics-only using *pkg/seaice* (*input.thermo/*)
    (d) sea-ice thermodynamics-only using *pkg/thsice* (*input.thsice/*)

    and also 2 adjoint set-ups (*code_ad/, input_ad/, input_ad.thsice/*).
56. `OpenAD` - Simple Adjoint experiment (used also to test Open-AD compiler)
57. `rotating_tank` - Rotating tank simulation in cylindrical coordinates. This experiment is described in detail in section 3.21.
58. `seaice_itd` - Seaice example using Ice Thickness Distribution (ITD).
Also contains additional set-ups:
    (a) (*input.thermo/*)
    (b) (*input.lipscomb07/*)
59. `seaice_obcs` - Similar to "lab_sea" (*input.salt_plume/*) experiment with only a fraction of the domain and open-boundary conditions derived from "lab_sea" experiment.
Also contains additional set-ups:
    (a) (*input.seaiceSponge/*)

   (b) (*input.tides/*)

60. `short_surf_wave` - Short surface wave adjusment (non-hydrostatic) in homogeneous 2-D vertical section (x-z).

61. `so_box_biogeo` - Open-boundary Southern ocean box around Drake passage, using same model parameters and forcing as experiment "tutorial_global_oce_biogeo" from which initial conditions and OB conditions have been extracted.

62. `solid-body.cs-32x32x1` - Solid body rotation test for cube sphere grid.

63. `tidal_basin_2d` - 2-D vertical section (x-z) with tidal forcing (untested)

64. `vermix` - Simple test in a small domain (3 columns) for ocean vertical mixing schemes. The standard set-up (*input/*) uses KPP scheme [*Large et al.*, 1994].
    Also contains additional set-ups:

    (a) with Double Diffusion scheme from KPP (*input.dd/*)
    (b) with *Gaspar et al.* [1990] (*pkg/ggl90*) scheme (*input.ggl90/*)
    (c) with *Mellor and Yamada* [1982] level 2. (*pkg/my82*) scheme (*input.my82/*)
    (d) with *Paluszkiewicz and Romea* [1997] (*pkg/opps*) scheme (*input.opps/*)
    (e) with *Pacanowski and Philander* [1981] (*pkg/pp81*) scheme (*input.pp81/*)

3.8.2 Directory structure of model examples

Each example directory has the following subdirectories:

- `code`: contains the code particular to the example. At a minimum, this directory includes the following files:

  - `code/packages.conf`: declares the list of packages or package groups to be used. If not included, the default version is located in `pkg/pkg_default`. Package groups are simply convenient collections of commonly used packages which are defined in `pkg/pkg_default`. Some packages may require other packages or may require their absence (that is, they are incompatible) and these package dependencies are listed in `pkg/pkg_depend`.
  - `code/CPP_EEOPTIONS.h`: declares CPP keys relative to the "execution environment" part of the code. The default version is located in `eesupp/inc`.
  - `code/CPP_OPTIONS.h`: declares CPP keys relative to the "numerical model" part of the code. The default version is located in `model/inc`.
  - `code/SIZE.h`: declares size of underlying computational grid. The default version is located in `model/inc`.

  In addition, other include files and subroutines might be present in `code` depending on the particular experiment. See Section 2 for more details.

- `input`: contains the input data files required to run the example. At a minimum, the `input` directory contains the following files:

  - `input/data`: this file, written as a namelist, specifies the main parameters for the experiment.
  - `input/data.pkg`: contains parameters relative to the packages used in the experiment.
  - `input/eedata`: this file contains "execution environment" data. At present, this consists of a specification of the number of threads to use in X and Y under multi-threaded execution.

  In addition, you will also find in this directory the forcing and topography files as well as the files describing the initial state of the experiment. This varies from experiment to experiment. See the verification directories refered to in this chapter for more details.

- `results`: this directory contains the output file `output.txt` produced by the simulation example. This file is useful for comparison with your own output when you run the experiment.
- `build`: this directory is initially empty and is used to compile and load the model, and to generate the executable.
- `run`: this directory is initially empty and is used to run the executable.

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

3.9 Barotropic Ocean Gyre In Cartesian Coordinates

(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 tutorial_barotropic_gyre. The experiment is a numerical rendition of the gyre circulation problem similar to the problems described analytically by Stommel in 1966 *Stommel* [1948] and numerically in Holland et. al *Holland and Lin* [975a].

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

$$f(y) = f_0 + \beta y \tag{3.1}$$

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

The sinusoidal wind-stress variations are defined according to

$$\tau_x(y) = \tau_0 \sin(\pi \frac{y}{L_y}) \tag{3.2}$$

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

Figure 3.1 summarizes the configuration simulated.

3.9.1 Equations Solved

The model is configured in hydrostatic form. The implicit free surface form of the pressure equation described in Marshall et. al *Marshall et al.* [1997b] is employed. A horizontal Laplacian operator ∇_h^2 provides viscous dissipation. The wind-stress momentum input is added to the momentum equation for the "zonal flow", u. Other terms in the model are explicitly switched off for this experiment configuration (see section 3.9.3), yielding an active set of equations solved in this configuration as follows

$$\frac{Du}{Dt} - fv + g\frac{\partial \eta}{\partial x} - A_h\nabla_h^2 u = \frac{\tau_x}{\rho_0 \Delta z} \tag{3.3}$$

$$\frac{Dv}{Dt} + fu + g\frac{\partial \eta}{\partial y} - A_h\nabla_h^2 v = 0 \tag{3.4}$$

$$\frac{\partial \eta}{\partial t} + \nabla_h \cdot \vec{u} = 0 \tag{3.5}$$

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

3.9.2 Discrete Numerical Configuration

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

3.9.2.1 Numerical Stability Criteria

The Laplacian dissipation coefficient, A_h, is set to $400ms^{-1}$. This value is chosen to yield a Munk layer width *Adcroft* [1995],

$$M_w = \pi(\frac{A_h}{\beta})^{\frac{1}{3}} \tag{3.6}$$

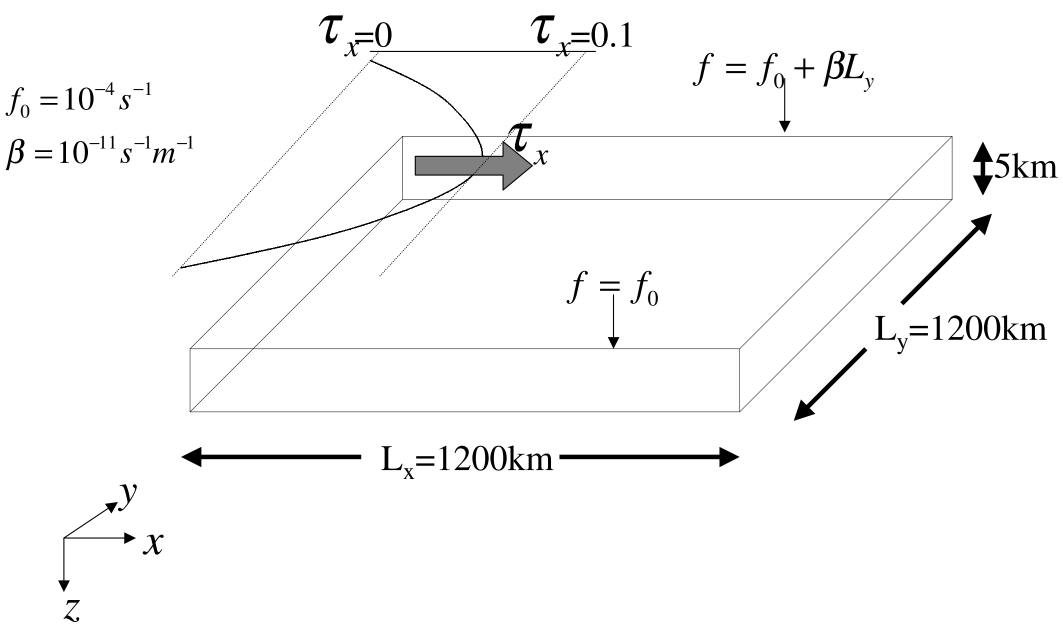


Figure 3.1: Schematic of simulation domain and wind-stress forcing function for barotropic gyre numerical experiment. The domain is enclosed bu solid walls at $x =$ 0,1200km and at $y =$ 0,1200km.

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

The model is stepped forward with a time step $\delta t = 1200$secs. With this time step the stability parameter to the horizontal Laplacian friction *Adcroft* [1995]

$$S_l = 4\frac{A_h \delta t}{\Delta x^2} \tag{3.7}$$

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

The numerical stability for inertial oscillations *Adcroft* [1995]

$$S_i = f^2 \delta t^2 \tag{3.8}$$

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

The advective CFL *Adcroft* [1995] for an extreme maximum horizontal flow speed of $|\vec{u}| = 2ms^{-1}$

$$S_a = \frac{|\vec{u}|\delta t}{\Delta x} \tag{3.9}$$

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

3.9.3 Code Configuration

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

- *input/data*
- *input/data.pkg*
- *input/eedata*,
- *input/windx.sin_y*,
- *input/topog.box*,
- *code/CPP_EEOPTIONS.h*
- *code/CPP_OPTIONS.h*,
- *code/SIZE.h*.

contain the code customizations and parameter settings for this experiments. Below we describe the customizations to these files associated with this experiment.

3.9.3.1 File *input/data*

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

- Line 7,

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

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

- Line 10,

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

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

- Lines 15 and 16

```
rigidLid=.FALSE.,
implicitFreeSurface=.TRUE.,
```

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

- Line 27,

```
startTime=0,
```

this line indicates that the experiment should start from $t = 0$ and implicitly suppresses searching for checkpoint files associated with restarting an numerical integration from a previously saved state.

- Line 29,

```
endTime=12000,
```

this line indicates that the experiment should start finish at $t = 12000s$. A restart file will be written at this time that will enable the simulation to be continued from this point.

- Line 30,

```
deltaTmom=1200,
```

This line sets the momentum equation timestep to $1200s$.

- Line 39,

```
usingCartesianGrid=.TRUE.,
```

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

- Line 41,

```
delX=60*20E3,
```

This line sets the horizontal grid spacing between each x-coordinate line in the discrete grid. The syntax indicates that the discrete grid should be comprise of 60 grid lines each separated by $20 \times 10^3 m$ (20 km).

- Line 42,

```
delY=60*20E3,
```

This line sets the horizontal grid spacing between each y-coordinate line in the discrete grid to $20 \times 10^3 m$ (20 km).

- Line 43,

```
delZ=5000,
```

This line sets the vertical grid spacing between each z-coordinate line in the discrete grid to $5000m$ (5 km).

- Line 46,

  `bathyFile='topog.box'`

  This line specifies the name of the file from which the domain bathymetry is read. This file is a two-dimensional (x, y) map of depths. This file is assumed to contain 64-bit binary numbers giving the depth of the model at each grid cell, ordered with the x coordinate varying fastest. The points are ordered from low coordinate to high coordinate for both axes. The units and orientation of the depths in this file are the same as used in the MITgcm code. In this experiment, a depth of $0m$ indicates a solid wall and a depth of $-5000m$ indicates open ocean. The matlab program *input/gendata.m* shows an example of how to generate a bathymetry file.

- Line 49,

  `zonalWindFile='windx.sin_y'`

  This line specifies the name of the file from which the x-direction surface wind stress is read. This file is also a two-dimensional (x, y) map and is enumerated and formatted in the same manner as the bathymetry file. The matlab program *input/gendata.m* includes example code to generate a valid **zonalWindFile** file.

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

```
 1 # Model parameters
 2 # Continuous equation parameters
 3  &PARM01
 4  tRef=20.,
 5  sRef=10.,
 6  viscAz=1.E-2,
 7  viscAh=4.E2,
 8  diffKhT=4.E2,
 9  diffKzT=1.E-2,
10  beta=1.E-11,
11  tAlpha=2.E-4,
12  sBeta =0.,
13  gravity=9.81,
14  gBaro=9.81,
15  rigidLid=.FALSE.,
16  implicitFreeSurface=.TRUE.,
17  eosType='LINEAR',
18  readBinaryPrec=64,
19  &
20 # Elliptic solver parameters
21  &PARM02
22  cg2dMaxIters=1000,
23  cg2dTargetResidual=1.E-7,
24  &
25 # Time stepping parameters
26  &PARM03
27  startTime=0,
28 #endTime=311040000,
29  endTime=12000.0,
30  deltaTmom=1200.0,
31  deltaTtracer=1200.0,
32  abEps=0.1,
33  pChkptFreq=2592000.0,
34  chkptFreq=120000.0,
35  dumpFreq=2592000.0,
36  &
```

```
# Gridding parameters
 &PARM04
 usingCartesianGrid=.TRUE.,
 usingSphericalPolarGrid=.FALSE.,
 delX=60*20E3,
 delY=60*20E3,
 delZ=5000.,
 &
 &PARM05
 bathyFile='topog.box',
 hydrogThetaFile=,
 hydrogSaltFile=,
 zonalWindFile='windx.sin_y',
 meridWindFile=,
 &
```

3.9.3.2 File *input/data.pkg*

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

3.9.3.3 File *input/eedata*

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

3.9.3.4 File *input/windx.sin_y*

The *input/windx.sin_y* file specifies a two-dimensional (x, y) map of wind stress ,τ_x, values. The units used are Nm^{-2}. Although τ_x is only a function of yn 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.

3.9.3.5 File *input/topog.box*

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

3.9.3.6 File *code/SIZE.h*

Two lines are customized in this file for the current experiment

- Line 39,

  ```
  sNx=60,
  ```

  this line sets the lateral domain extent in grid points for the axis aligned with the x-coordinate.

- Line 40,

  ```
  sNy=60,
  ```

  this line sets the lateral domain extent in grid points for the axis aligned with the y-coordinate.

```
1 C $Header: /u/gcmpack/manual/s_examples/barotropic_gyre/code/SIZE.h.tex,v 1.1.1.1 2001/08/08 16:15:58 adc
2 C $Name:  $
3 C
4 C     /==========================================================\
```

```
C     | SIZE.h Declare size of underlying computational grid.    |
C     |==========================================================|
C     | The design here support a three-dimensional model grid   |
C     | with indices I,J and K. The three-dimensional domain     |
C     | is comprised of nPx*nSx blocks of size sNx along one axis|
C     | nPy*nSy blocks of size sNy along another axis and one    |
C     | block of size Nz along the final axis.                   |
C     | Blocks have overlap regions of size OLx and OLy along the|
C     | dimensions that are subdivided.                          |
C     \==========================================================/
C     Voodoo numbers controlling data layout.
C     sNx - No. X points in sub-grid.
C     sNy - No. Y points in sub-grid.
C     OLx - Overlap extent in X.
C     OLy - Overlat extent in Y.
C     nSx - No. sub-grids in X.
C     nSy - No. sub-grids in Y.
C     nPx - No. of processes to use in X.
C     nPy - No. of processes to use in Y.
C     Nx  - No. points in X for the total domain.
C     Ny  - No. points in Y for the total domain.
C     Nr  - No. points in 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 =  60,
     &           sNy =  60,
     &           OLx =   3,
     &           OLy =   3,
     &           nSx =   1,
     &           nSy =   1,
     &           nPx =   1,
     &           nPy =   1,
     &           Nx  = sNx*nSx*nPx,
     &           Ny  = sNy*nSy*nPy,
     &           Nr  =   1)

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

3.9.3.7 File *code/CPP_OPTIONS.h*

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

3.9.3.8 File *code/CPP_EEOPTIONS.h*

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

3.10 Four Layer Baroclinic Ocean Gyre In Spherical Coordinates

(in directory: *verification/tutorial_baroclinic_gyre/*)

This document describes an example experiment using MITgcm to simulate a baroclinic ocean gyre for four layers in spherical polar coordinates. The files for this experiment can be found in the verification directory under tutorial_baroclinic_gyre.

3.10.1 Overview

This example experiment demonstrates using the MITgcm to simulate a baroclinic, wind-forced, ocean gyre circulation. The experiment is a numerical rendition of the gyre circulation problem similar to the problems described analytically by Stommel in 1966 *Stommel* [1948] and numerically in Holland et. al *Holland and Lin* [975a].

In this experiment the model is configured to represent a mid-latitude enclosed sector of fluid on a sphere, $60^\circ \times 60^\circ$ in lateral extent. The fluid is 2 km deep and is forced by a constant in time zonal wind stress, τ_λ, that varies sinusoidally in the north-south direction. Topologically the simulated domain is a sector on a sphere and the coriolis parameter, f, is defined according to latitude, φ

$$f(\varphi) = 2\Omega \sin(\varphi) \tag{3.10}$$

with the rotation rate, Ω set to $\frac{2\pi}{86400s}$.

The sinusoidal wind-stress variations are defined according to

$$\tau_\lambda(\varphi) = \tau_0 \sin(\pi \frac{\varphi}{L_\varphi}) \tag{3.11}$$

where L_φ is the lateral domain extent (60°) and τ_0 is set to $0.1Nm^{-2}$.

Figure 3.2 summarizes the configuration simulated. In contrast to the example in section 3.9, the current experiment simulates a spherical polar domain. As indicated by the axes in the lower left of the figure the model code works internally in a locally orthogonal coordinate (x, y, z). For this experiment description the local orthogonal model coordinate (x, y, z) is synonymous with the coordinates (λ, φ, r) shown in figure 1.16

The experiment has four levels in the vertical, each of equal thickness, $\Delta z = 500$ m. Initially the fluid is stratified with a reference potential temperature profile, $\theta_{250} = 20^\circ$ C, $\theta_{750} = 10^\circ$ C, $\theta_{1250} = 8^\circ$ C, $\theta_{1750} = 6^\circ$ C. The equation of state used in this experiment is linear

$$\rho = \rho_0(1 - \alpha_\theta \theta^{'}) \tag{3.12}$$

which is implemented in the model as a density anomaly equation

$$\rho^{'} = -\rho_0 \alpha_\theta \theta^{'} \tag{3.13}$$

with $\rho_0 = 999.8\,\mathrm{kg\,m^{-3}}$ and $\alpha_\theta = 2 \times 10^{-4}\,\mathrm{degrees^{-1}}$. Integrated forward in this configuration the model state variable **theta** is equivalent to either in-situ temperature, T, or potential temperature, θ. For consistency with later examples, in which the equation of state is non-linear, we use θ to represent temperature here. This is the quantity that is carried in the model core equations.

3.10.2 Equations solved

For this problem the implicit free surface, **HPE** (see section 1.3.4.2) form of the equations described in Marshall et. al *Marshall et al.* [1997b] are employed. The flow is three-dimensional with just temperature, θ, as an active tracer. The equation of state is linear. A horizontal Laplacian operator ∇_h^2 provides viscous dissipation and provides a diffusive sub-grid scale closure for the temperature equation. A wind-stress

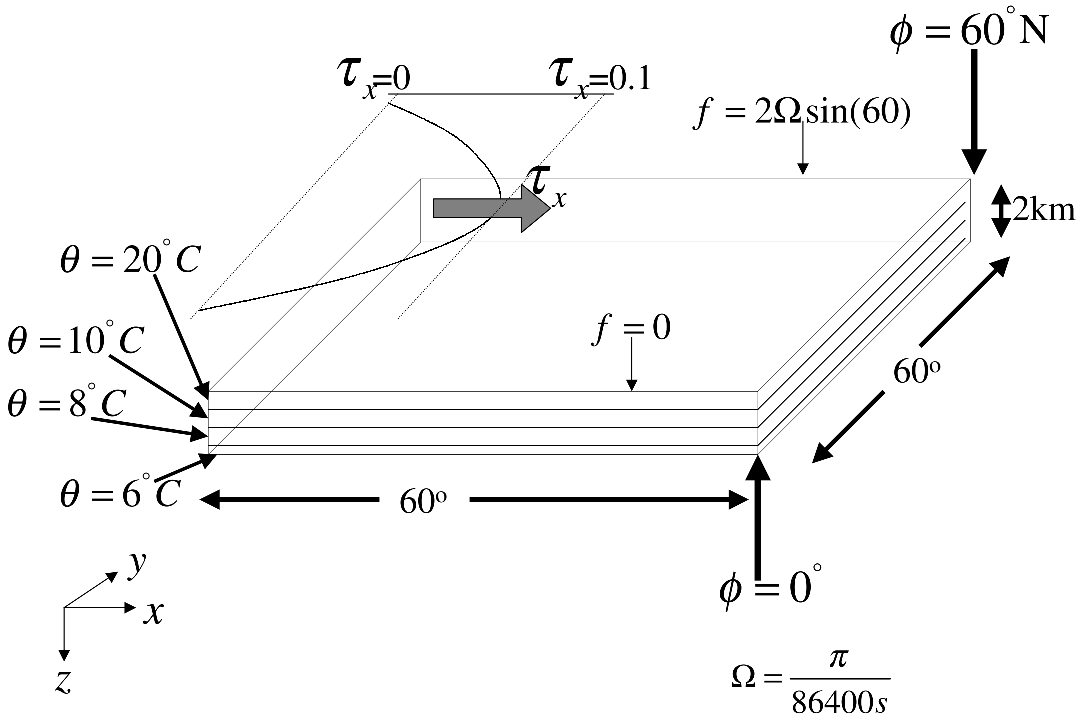


Figure 3.2: Schematic of simulation domain and wind-stress forcing function for the four-layer gyre numerical experiment. The domain is enclosed by solid walls at 0° E, 60° E, 0° N and 60° N. An initial stratification is imposed by setting the potential temperature, θ, in each layer. The vertical spacing, Δz, is constant and equal to 500m.

momentum forcing is added to the momentum equation for the zonal flow, u. Other terms in the model are explicitly switched off for this experiment configuration (see section 3.10.4). This yields an active set of equations solved in this configuration, written in spherical polar coordinates as follows

$$\frac{Du}{Dt} - fv + \frac{1}{\rho}\frac{\partial p'}{\partial \lambda} - A_h \nabla_h^2 u - A_z \frac{\partial^2 u}{\partial z^2} = \mathcal{F}_\lambda \tag{3.14}$$

$$\frac{Dv}{Dt} + fu + \frac{1}{\rho}\frac{\partial p'}{\partial \varphi} - A_h \nabla_h^2 v - A_z \frac{\partial^2 v}{\partial z^2} = 0 \tag{3.15}$$

$$\frac{\partial \eta}{\partial t} + \frac{\partial H\widehat{u}}{\partial \lambda} + \frac{\partial H\widehat{v}}{\partial \varphi} = 0 \tag{3.16}$$

$$\frac{D\theta}{Dt} - K_h \nabla_h^2 \theta - K_z \frac{\partial^2 \theta}{\partial z^2} = 0 \tag{3.17}$$

$$p' = g\rho_0\eta + \int_{-z}^{0} \rho' dz \tag{3.18}$$

$$\rho' = -\alpha_\theta \rho_0 \theta' \tag{3.19}$$

$$\mathcal{F}_\lambda|_s = \frac{\tau_\lambda}{\rho_0 \Delta z_s} \tag{3.20}$$

$$\mathcal{F}_\lambda|_i = 0 \tag{3.21}$$

where u and v are the components of the horizontal flow vector $\vec{u}$ on the sphere ($u = \dot{\lambda}, v = \dot{\varphi}$). The terms $H\widehat{u}$ and $H\widehat{v}$ are the components of the vertical integral term given in equation 1.35 and explained in more detail in section 2.4. However, for the problem presented here, the continuity relation (equation 3.16) differs from the general form given in section 2.4, equation 2.15, because the source terms $\mathcal{P} - \mathcal{E} + \mathcal{R}$ are all 0.

The pressure field, p', is separated into a barotropic part due to variations in sea-surface height, η, and a hydrostatic part due to variations in density, ρ', integrated through the water column.

The suffices s, i indicate surface layer and the interior of the domain. The windstress forcing, $\mathcal{F}_\lambda$, is applied in the surface layer by a source term in the zonal momentum equation. In the ocean interior this term is zero.

In the momentum equations lateral and vertical boundary conditions for the ∇_h^2 and $\frac{\partial^2}{\partial z^2}$ operators are specified when the numerical simulation is run - see section 3.10.4. For temperature the boundary condition is "zero-flux" e.g. $\frac{\partial \theta}{\partial \varphi} = \frac{\partial \theta}{\partial \lambda} = \frac{\partial \theta}{\partial z} = 0$.

3.10.3 Discrete Numerical Configuration

The domain is discretised with a uniform grid spacing in latitude and longitude $\Delta\lambda = \Delta\varphi = 1^\circ$, so that there are sixty grid cells in the zonal and meridional directions. Vertically the model is configured with four layers with constant depth, Δz, of 500 m. The internal, locally orthogonal, model coordinate variables x and y are initialized from the values of λ, φ, $\Delta\lambda$ and $\Delta\varphi$ in radians according to

$$x = r\cos(\varphi)\lambda,\ \Delta x = r\cos(\varphi)\Delta\lambda \tag{3.22}$$

$$y = r\varphi,\ \Delta y = r\Delta\varphi \tag{3.23}$$

The procedure for generating a set of internal grid variables from a spherical polar grid specification is discussed in section 2.11.4.

S/R INI_SPHERICAL_POLAR_GRID (*model/src/ini_spherical_polar_grid.F*)
A_c, A_ζ, A_w, A_s: **rAc**, **rAz**, **rAw**, **rAs** (*GRID.h*)
Δx_g, Δy_g: **DXg**, **DYg** (*GRID.h*)
Δx_c, Δy_c: **DXc**, **DYc** (*GRID.h*)
Δx_f, Δy_f: **DXf**, **DYf** (*GRID.h*)
Δx_v, Δy_u: **DXv**, **DYu** (*GRID.h*)

As described in 2.16, the time evolution of potential temperature, θ, (equation 3.17) is evaluated prognostically. The centered second-order scheme with Adams-Bashforth time stepping described in section 2.16.1 is used to step forward the temperature equation. Prognostic terms in the momentum equations are solved using flux form as described in section 2.14. The pressure forces that drive the fluid motions, ($\frac{\partial p^{'}}{\partial \lambda}$ and $\frac{\partial p^{'}}{\partial \varphi}$), are found by summing pressure due to surface elevation η and the hydrostatic pressure. The hydrostatic part of the pressure is diagnosed explicitly by integrating density. The sea-surface height, η, is diagnosed using an implicit scheme. The pressure field solution method is described in sections 2.4 and 1.3.6.

3.10.3.1 Numerical Stability Criteria

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

$$M_w = \pi (\frac { A_{h} }{ \beta })^{\frac{1}{3}} \tag{3.24}$$

of $\approx$ 100km. This is greater than the model resolution in mid-latitudes $\Delta x = r \cos(\varphi) \Delta \lambda \approx 80$ km at $\varphi = 45^{\circ}$, 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

$$S_l = 4 \frac{A_{h} \delta t}{{\Delta x}^2} \tag{3.25}$$

evaluates to 0.012, which is well below the 0.3 upper limit for stability for this term under ABII time-stepping.

The vertical dissipation coefficient, A_z, is set to $1 \times 10^{-2} \mathrm{m}^2\mathrm{s}^{-1}$. The associated stability limit

$$S_l = 4 \frac{A_{z} \delta t}{{\Delta z}^2} \tag{3.26}$$

evaluates to 4.8×10^{-5} which is again well below the upper limit. The values of A_h and A_z are also used for the horizontal (K_h) and vertical (K_z) diffusion coefficients for temperature respectively.

The numerical stability for inertial oscillations

$$S_i = f^{2} {\delta t}^2 \tag{3.27}$$

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

The advective CFL for a extreme maximum horizontal flow speed of $| \vec{u} | = 2 ms^{-1}$

$$C_a = \frac{| \vec{u} | \delta t}{ \Delta x} \tag{3.28}$$

evaluates to 5×10^{-2}. This is well below the stability limit of 0.5.

The stability parameter for internal gravity waves propagating at 2 m s^{-1}

$$S_c = \frac{c_{g} \delta t}{ \Delta x} \tag{3.29}$$

evaluates to $\approx 5 \times 10^{-2}$. This is well below the linear stability limit of 0.25.

3.10.4 Code Configuration

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

- *input/data*
- *input/data.pkg*
- *input/eedata*,
- *input/windx.sin_y*,
- *input/topog.box*,
- *code/CPP_EEOPTIONS.h*
- *code/CPP_OPTIONS.h*,
- *code/SIZE.h*.

contain the code customisations and parameter settings for this experiment. Below we describe the customisations to these files associated with this experiment.

3.10.4.1 File *input/data*

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

- Line 4,

  ```
  tRef=20.,10.,8.,6.,
  ```

  this line sets the initial and reference values of potential temperature at each model level in units of °C. The entries are ordered from surface to depth. For each depth level the initial and reference profiles will be uniform in x and y. The values specified here are read into the variable `tRef` in the model code, by procedure `INI_PARMS`

S/R INI_THETA(*ini_theta.F*)	`ini_theta.F`

- Line 6,

  ```
  viscAz=1.E-2,
  ```

  this line sets the vertical Laplacian dissipation coefficient to $1 \times 10^{-2}\mathrm{m}^2\mathrm{s}^{-1}$. Boundary conditions for this operator are specified later. The variable `viscAz` is read in the routine `ini_parms.F` and is copied into model general vertical coordinate variable `viscAr` At each time step, the viscous term contribution to the momentum equations is calculated in routine `CALC_DIFFUSIVITY`

  S/R CALC_DIFFUSIVITY(*calc_diffusivity.F*)

- Line 7,

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

  this line sets the horizontal laplacian frictional dissipation coefficient to $1 \times 10^{-2}\mathrm{m}^2\mathrm{s}^{-1}$. Boundary conditions for this operator are specified later. The variable `viscAh` is read in the routine `INI_PARMS` and applied in routine `MOM_FLUXFORM`.

  S/R MOM_FLUXFORM(*mom_fluxform.F*)

- Line 8,

```
no_slip_sides=.FALSE.
```

this line selects a free-slip lateral boundary condition for the horizontal laplacian friction operator e.g. $\frac{\partial u}{\partial y}$=0 along boundaries in y and $\frac{\partial v}{\partial x}$=0 along boundaries in x. The variable no_slip_sides is read in the routine INI_PARMS and the boundary condition is evaluated in routine

S/R MOM_FLUXFORM(*mom_fluxform.F*)	mom_fluxform.F

- Lines 9,

```
no_slip_bottom=.TRUE.
```

this line selects a no-slip boundary condition for bottom boundary condition in the vertical laplacian friction operator e.g. $u = v = 0$ at $z = -H$, where H is the local depth of the domain. The variable no_slip_bottom is read in the routine INI_PARMS and is applied in the routine MOM_FLUXFORM.

S/R MOM_FLUXFORM(*mom_fluxform.F*)	mom_fluxform.F

- Line 10,

```
diffKhT=4.E2,
```

this line sets the horizontal diffusion coefficient for temperature to $400\,\mathrm{m^2s^{-1}}$. The boundary condition on this operator is $\frac{\partial}{\partial x} = \frac{\partial}{\partial y} = 0$ at all boundaries. The variable diffKhT is read in the routine INI_PARMS and used in routine CALC_GT.

S/R CALC_GT(*calc_gt.F*)	calc_gt.F

- Line 11,

```
diffKzT=1.E-2,
```

this line sets the vertical diffusion coefficient for temperature to $10^{-2}\,\mathrm{m^2s^{-1}}$. The boundary condition on this operator is $\frac{\partial}{\partial z} = 0$ on all boundaries. The variable diffKzT is read in the routine INI_PARMS. It is copied into model general vertical coordinate variable diffKrT which is used in routine CALC_DIFFUSIVITY.

S/R CALC_DIFFUSIVITY(*calc_diffusivity.F*)	calc_diffusivity.F

- Line 13,

```
tAlpha=2.E-4,
```

This line sets the thermal expansion coefficient for the fluid to $2 \times 10^{-4}\,\mathrm{degrees}^{-1}$ The variable tAlpha is read in the routine INI_PARMS. The routine FIND_RHO makes use of **tAlpha**.

S/R FIND_RHO(*find_rho.F*)	find_rho.F

- Line 18,

```
eosType='LINEAR'
```

This line selects the linear form of the equation of state. The variable eosType is read in the routine INI_PARMS. The values of **eosType** sets which formula in routine *FIND_RHO* is used to calculate density.

S/R FIND_RHO(*find_rho.F*)	find_rho.F

- Line 40,

```
usingSphericalPolarGrid=.TRUE.,
```

This line requests that the simulation be performed in a spherical polar coordinate system. It affects the interpretation of grid input parameters, for example **delX** and **delY** and causes the grid generation routines to initialize an internal grid based on spherical polar geometry. The variable `usingSphericalPolarGrid` is read in the routine `INI_PARMS`. When set to **.TRUE.** the settings of **delX** and **delY** are taken to be in degrees. These values are used in the routine

S/R INI_SPEHRICAL_POLAR_GRID(*ini_spherical_polar_grid.F*)	`ini_spherical_polar_grid.`

- Line 41,

  `ygOrigin=0.,`

  This line sets the southern boundary of the modeled domain to 0° latitude. This value affects both the generation of the locally orthogonal grid that the model uses internally and affects the initialization of the coriolis force. Note - it is not required to set a longitude boundary, since the absolute longitude does not alter the kernel equation discretisation. The variable `ygOrigin` is read in the routine `INI_PARMS` and is used in routine

S/R INI_SPEHRICAL_POLAR_GRID(*ini_spherical_polar_grid.F*)	`ini_spherical_polar_grid.`

- Line 42,

  `delX=60*1.,`

  This line sets the horizontal grid spacing between each y-coordinate line in the discrete grid to 1° in longitude. The variable `delX` is read in the routine `INI_PARMS` and is used in routine

S/R INI_SPEHRICAL_POLAR_GRID(*ini_spherical_polar_grid.F*)	`ini_spherical_polar_grid.`

- Line 43,

  `delY=60*1.,`

  This line sets the horizontal grid spacing between each y-coordinate line in the discrete grid to 1° in latitude. The variable `delY` is read in the routine `INI_PARMS` and is used in routine

S/R INI_SPEHRICAL_POLAR_GRID(*ini_spherical_polar_grid.F*)	`ini_spherical_polar_grid.`

- Line 44,

  `delZ=500.,500.,500.,500.,`

  This line sets the vertical grid spacing between each z-coordinate line in the discrete grid to 500 m, so that the total model depth is 2 km. The variable `delZ` is read in the routine `INI_PARMS`. It is copied into the internal model coordinate variable `delR` which is used in routine

S/R INI_VERTICAL_GRID(*ini_vertical_grid.F*)	`ini_vertical_grid.F`

- Line 47,

  `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 $0m$ indicates a solid wall and a depth of $-2000m$ indicates open ocean. The matlab program *input/gendata.m* shows an example of how to generate a bathymetry file. The variable `bathyFile` is read in the routine `INI_PARMS`. The bathymetry file is read in the routine

S/R INI_DEPTHS(*ini_depths.F*)	`ini_depths.F`

- Line 50,

  `zonalWindFile='windx.sin_y'`

  This line specifies the name of the file from which the x-direction (zonal) surface wind stress is read. This file is also a two-dimensional (x, y) map and is enumerated and formatted in the same manner as the bathymetry file. The matlab program *input/gendata.m* includes example code to generate a valid **zonalWindFile** file. The variable `zonalWindFile` is read in the routine `INI_PARMS`. The wind-stress file is read in the routine

  S/R EXTERNAL_FIELDS_LOAD(external_fields_load.F) `external_fields_load.F`

other lines in the file *input/data* are standard values.

```
 1 # Model parameters
 2 # Continuous equation parameters
 3  &PARM01
 4  tRef=20.,10.,8.,6.,
 5  sRef=10.,10.,10.,10.,
 6  viscAz=1.E-2,
 7  viscAh=4.E2,
 8  no_slip_sides=.FALSE.,
 9  no_slip_bottom=.TRUE.,
10  diffKhT=4.E2,
11  diffKzT=1.E-2,
12  beta=1.E-11,
13  tAlpha=2.E-4,
14  sBeta =0.,
15  gravity=9.81,
16  rigidLid=.FALSE.,
17  implicitFreeSurface=.TRUE.,
18  eosType='LINEAR',
19  readBinaryPrec=64,
20  &
21 # Elliptic solver parameters
22  &PARM02
23  cg2dMaxIters=1000,
24  cg2dTargetResidual=1.E-13,
25  &
26 # Time stepping parameters
27  &PARM03
28  startTime=0.,
29  endTime=12000.,
30  deltaTmom=1200.0,
31  deltaTtracer=1200.0,
32  abEps=0.1,
33  pChkptFreq=17000.0,
34  chkptFreq=0.0,
35  dumpFreq=2592000.0,
36  &
37 # Gridding parameters
38  &PARM04
39  usingCartesianGrid=.FALSE.,
40  usingSphericalPolarGrid=.TRUE.,
41  ygOrigin=0.,
42  delX=60*1.,
43  delY=60*1.,
44  delZ=500.,500.,500.,500.,
45  &
46  &PARM05
47  bathyFile='topog.box',
```

```
hydrogThetaFile=,
hydrogSaltFile=,
zonalWindFile='windx.sin_y',
meridWindFile=,
&
```

3.10.4.2 File *input/data.pkg*

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

3.10.4.3 File *input/eedata*

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

3.10.4.4 File *input/windx.sin_y*

The *input/windx.sin_y* file specifies a two-dimensional (x, y) map of wind stress ,τ_x, values. The units used are Nm^{-2} (the default for MITgcm). Although τ_x is only a function of latitude, 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 (routine *EXTERNAL_FIELDS_LOAD*. The included matlab program *input/gendata.m* gives a complete code for creating the *input/windx.sin_y* file.

3.10.4.5 File *input/topog.box*

The *input/topog.box* file specifies a two-dimensional (x, y) map of depth values. For this experiment values are either 0 m or -2000 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.

3.10.4.6 File *code/SIZE.h*

Two lines are customized in this file for the current experiment

- Line 39,

  ```
  sNx=60,
  ```

  this line sets the lateral domain extent in grid points for the axis aligned with the x-coordinate.

- Line 40,

  ```
  sNy=60,
  ```

  this line sets the lateral domain extent in grid points for the axis aligned with the y-coordinate.

- Line 49,

  ```
  Nr=4,
  ```

  this line sets the vertical domain extent in grid points.

3.10.4.7 File *code/CPP_OPTIONS.h*

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

3.10.4.8 File *code/CPP_EEOPTIONS.h*

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

3.10.4.9 Other Files

Other files relevant to this experiment are

- *model/src/ini_cori.F*. This file initializes the model coriolis variables **fCorU** and **fCorV**.
- *model/src/ini_spherical_polar_grid.F* This file initializes the model grid discretisation variables **dxF, dyF, dxG, dyG, dxC, dyC**.
- *model/src/ini_parms.F*.

3.10.5 Running The Example

3.10.5.1 Code Download

In order to run the examples you must first download the code distribution. Instructions for downloading the code can be found in section 3.2.

3.10.5.2 Experiment Location

This example experiments is located under the release sub-directory

verification/exp2/

3.10.5.3 Running the Experiment

To run the experiment

1. Set the current directory to *input/*

   ```
   % cd input
   ```

2. Verify that current directory is now correct

   ```
   % pwd
   ```

   You should see a response on the screen ending in
   verification/exp2/input

3. Run the genmake script to create the experiment *Makefile*

   ```
   % ../../../tools/genmake -mods=../code
   ```

4. Create a list of header file dependencies in *Makefile*

   ```
   % make depend
   ```

5. Build the executable file.

   ```
   % make
   ```

6. Run the *mitgcmuv* executable

   ```
   % ./mitgcmuv
   ```

3.11 Ocean Gyre Advection Schemes

(in directory: *verification/tutorial_advection_in_gyre/*)

Author: Oliver Jahn and Chris Hill

This set of examples is based on the barotropic and baroclinic gyre MITgcm configurations, that are described in the tutorial sections 3.9 and 3.10. The examples in this section explain how to introduce a passive tracer into the flow field of the barotropic and baroclinic gyre setups and looks at how the time evolution of the passive tracer depends on the advection or transport scheme that is selected for the tracer.

Passive tracers are useful in many numerical experiments. In some cases tracers are used to track flow pathways, for example in *Dutay et al.* [2002] a passive tracer is used to track pathways of CFC-11 in 13 global ocean models, using a numerical configuration similar to the example described in section 3.20.5). In other cases tracers are used as a way to infer bulk mixing coefficients for a turbulent flow field, for example in *Marshall et al.* [2006] a tracer is used to infer eddy mixing coefficients in the Antarctic Circumpolar Current region. In biogeochemical and ecological simulations large numbers of tracers are used that carry the concentrations of biological nutrients and concentrations of biological species, for example in .... When using tracers for these and other purposes it is useful to have a feel for the role that the advection scheme employed plays in determining properties of the tracer distribution. In particular, in a discrete numerical model tracer advection only approximates the continuum behavior in space and time and different advection schemes introduce diferent approximations so that the resulting tracer distributions vary. In the following text we illustrate how to use the different advection schemes available in MITgcm here, and discuss which properties are well represented by each one. The advection schemes selections also apply to active tracers (e.g. T and S) and the character of the schemes also affect their distributions and behavior.

3.11.1 Advection and tracer transport

In general, the tracer problem we want to solve can be written

$$\frac{\partial C}{\partial t} = -U \cdot \nabla C + S \tag{3.30}$$

where C is the tracer concentration in a model cell, U is the model three-dimensional flow field ($U = (u, v, w)$). In (3.30) S represents source, sink and tendency terms not associated with advective transport. Example of terms in S include (i) air-sea fluxes for a dissolved gas, (ii) biological grazing and growth terms (for a biogeochemical problem) or (iii) convective mixing and other sub-grid parameterizations of mixing. In this section we are primarily concerned with

1. how to introduce the tracer term, C, into an integration

2. the different discretized forms of the $-U \cdot \nabla C$ term that are available

3.11.2 Introducing a tracer into the flow

The MITgcm ptracers package (see section 6.3.3 for a more complete discussion of the ptracers package and section 6.1 for a general introduction to MITgcm packages) provides pre-coded support for a simple passive tracer with an initial distribution at simulation time $t = 0$ of $C_0(x, y, z)$. The steps required to use this capability are

1. **Activating the ptracers package.** This simply requires adding the line `ptracers` to the `packages.conf` file in the *code/* directory for the experiment.

2. **Setting an initial tracer distribution.**

Once the two steps above are complete we can proceed to examine how the tracer we have created is carried by the flow field and what properties of the tracer distribution are preserved under different advection schemes.

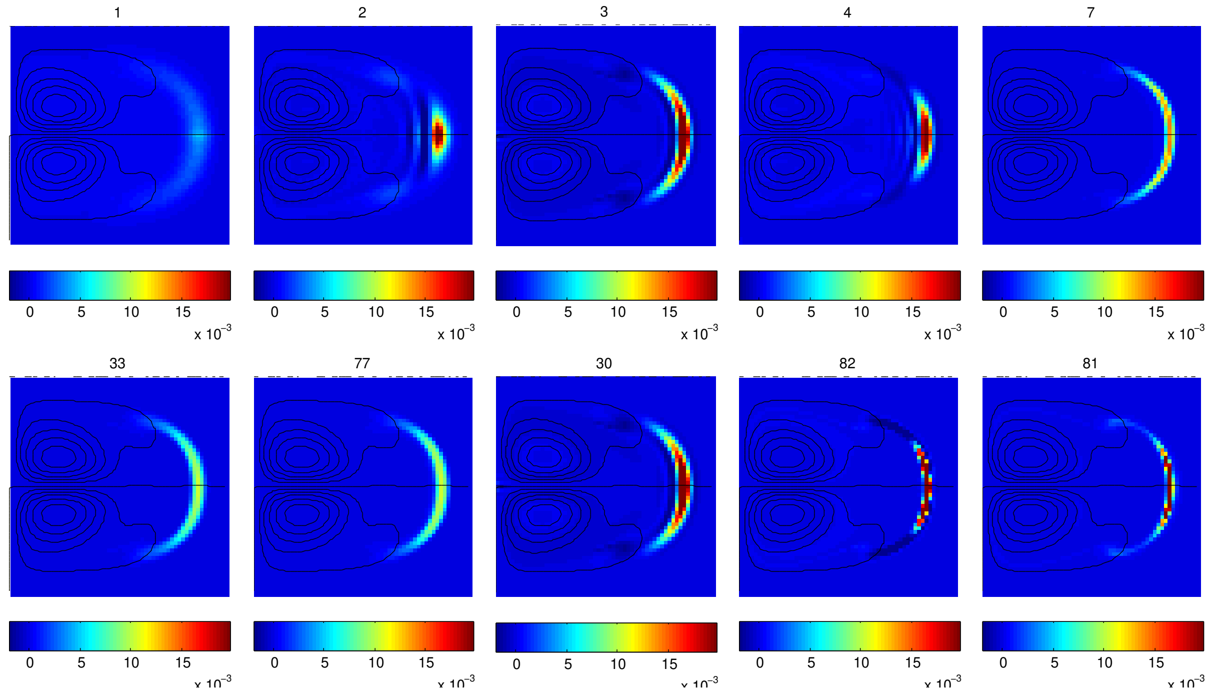


Figure 3.3: Dye evolving in a double gyre with different advection schemes. The figure shows the dye concentration one year after injection into a single grid cell near the left boundary. Stream lines are also shown.

3.11.3 Selecting an advection scheme

- flags in data and data.ptracers
  - overlap width
  - CPP GAD_ALLOW_SOM_ADVECT required for SOM case

3.11.4 Comparison of different advection schemes

1. Conservation
2. Dispersion
3. Diffusion
4. Positive definite

3.11.5 Code and Parameters files for this tutorial

The code and parameters for the experiments can be found in the MITgcm example experiments directory *verification/tutorial_advection_in_gyre/*.

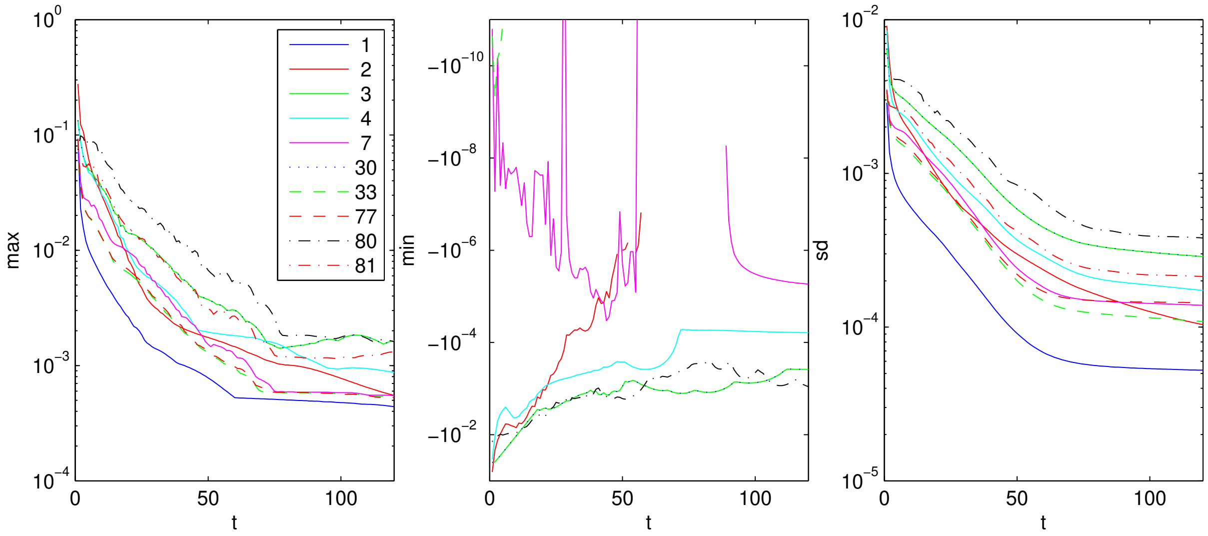


Figure 3.4: Maxima, minima and standard deviation (from left) as a function of time (in months) for the gyre circulation experiment from figure 3.3.

3.12 Global Ocean Simulation at 4° Resolution

(in directory: *verification/tutorial_global_oce_latlon/*)

WARNING: the description of this experiment is not complete. In particular, many parameters are not yet described.

This example experiment demonstrates using the MITgcm to simulate the planetary ocean circulation. The simulation is configured with realistic geography and bathymetry on a $4^\circ \times 4^\circ$ spherical polar grid. The files for this experiment are in the verification directory under tutorial_global_oce_latlon. Fifteen levels are used in the vertical, ranging in thickness from 50 m at the surface to 690 m at depth, giving a maximum model depth of 5200 m. Different time-steps are used to accelerate the convergence to equilibrium [*Bryan*, 1984] so that, at this resolution, the configuration can be integrated forward for thousands of years on a single processor desktop computer.

3.12.1 Overview

The model is forced with climatological wind stress data from *Trenberth et al.* [1990] and NCEP surface flux data from *Kalnay et al.* [1996]. Climatological data [*Levitus and T.P.Boyer*, 1994b] is used to initialize the model hydrography. *Levitus and T.P.Boyer* seasonal climatology data is also used throughout the calculation to provide additional air-sea fluxes. These fluxes are combined with the NCEP climatological estimates of surface heat flux, resulting in a mixed boundary condition of the style described in *Haney* [1971]. Altogether, this yields the following forcing applied in the model surface layer.

$$\mathcal{F}_u = \frac{\tau_x}{\rho_0 \Delta z_s} \tag{3.31}$$

$$\mathcal{F}_v = \frac{\tau_y}{\rho_0 \Delta z_s} \tag{3.32}$$

$$\mathcal{F}_\theta = -\lambda_\theta(\theta - \theta^*) - \frac{1}{C_p \rho_0 \Delta z_s}\mathcal{Q} \tag{3.33}$$

$$\mathcal{F}_s = -\lambda_s(S - S^*) + \frac{S_0}{\Delta z_s}(\mathcal{E} - \mathcal{P} - \mathcal{R}) \tag{3.34}$$

where $\mathcal{F}_u$, $\mathcal{F}_v$, $\mathcal{F}_\theta$, $\mathcal{F}_s$ are the forcing terms in the zonal and meridional momentum and in the potential temperature and salinity equations respectively. The term Δz_s represents the top ocean layer thickness in meters. It is used in conjunction with a reference density, ρ_0 (here set to 999.8 kg m^{-3}), a reference salinity, S_0 (here set to 35 ppt), and a specific heat capacity, C_p (here set to 4000 J °C^{-1} kg^{-1}), to convert input dataset values into time tendencies of potential temperature (with units of °C s^{-1}), salinity (with units ppt s^{-1}) and velocity (with units m s^{-2}). The externally supplied forcing fields used in this experiment are τ_x, τ_y, θ^*, S^*, $\mathcal{Q}$ and $\mathcal{E}-\mathcal{P}-\mathcal{R}$. The wind stress fields (τ_x, τ_y) have units of N m^{-2}. The temperature forcing fields (θ^* and Q) have units of °C and W m^{-2} respectively. The salinity forcing fields (S^* and $\mathcal{E}-\mathcal{P}-\mathcal{R}$) have units of ppt and m s^{-1} respectively. The source files and procedures for ingesting this data into the simulation are described in the experiment configuration discussion in section 3.12.3.

3.12.2 Discrete Numerical Configuration

The model is configured in hydrostatic form. The domain is discretised with a uniform grid spacing in latitude and longitude on the sphere $\Delta\phi = \Delta\lambda = 4°$, so that there are ninety grid cells in the zonal and forty in the meridional direction. The internal model coordinate variables x and y are initialized according to

$$x = r\cos(\phi),\ \Delta x = r\cos(\Delta\phi) \tag{3.35}$$

$$y = r\lambda,\ \Delta y = r\Delta\lambda \tag{3.36}$$

Arctic polar regions are not included in this experiment. Meridionally the model extends from 80°S to 80°N. Vertically the model is configured with fifteen layers with the following thicknesses: $\Delta z_1 = 50$ m, $\Delta z_2 = 70$ m, $\Delta z_3 = 100$ m, $\Delta z_4 = 140$ m, $\Delta z_5 = 190$ m, $\Delta z_6 = 240$ m, $\Delta z_7 = 290$ m, $\Delta z_8 = 340$ m, $\Delta z_9 = 390$ m, $\Delta z_{10} = 440$ m, $\Delta z_{11} = 490$ m, $\Delta z_{12} = 540$ m, $\Delta z_{13} = 590$ m, $\Delta z_{14} = 640$ m, $\Delta z_{15} = 690$ m (here the numeric subscript indicates the model level index number, `k`) to give a total depth, H, of −5200m. The implicit free surface form of the pressure equation described in *Marshall et al.* [1997b] is employed. A Laplacian operator, ∇^2, provides viscous dissipation. Thermal and haline diffusion is also represented by a Laplacian operator.

Wind-stress forcing is added to the momentum equations in (3.37) for both the zonal flow, u and the meridional flow v, according to equations (3.31) and (3.32). Thermodynamic forcing inputs are added to the equations in (3.37) for potential temperature, θ, and salinity, S, according to equations (3.33) and (3.34). This produces a set of equations solved in this configuration as follows:

$$\frac{Du}{Dt} - fv + \frac{1}{\rho}\frac{\partial p^{'}}{\partial x} - \nabla_h \cdot A_h \nabla_h u - \frac{\partial}{\partial z}A_z\frac{\partial u}{\partial z} = \begin{cases} \mathcal{F}_u & \text{(surface)} \\ 0 & \text{(interior)} \end{cases} \tag{3.37}$$

$$\frac{Dv}{Dt} + fu + \frac{1}{\rho}\frac{\partial p^{'}}{\partial y} - \nabla_h \cdot A_h \nabla_h v - \frac{\partial}{\partial z}A_z\frac{\partial v}{\partial z} = \begin{cases} \mathcal{F}_v & \text{(surface)} \\ 0 & \text{(interior)} \end{cases} \tag{3.38}$$

$$\frac{\partial \eta}{\partial t} + \nabla_h \cdot \vec{u} = 0 \tag{3.39}$$

$$\frac{D\theta}{Dt} - \nabla_h \cdot K_h \nabla_h \theta - \frac{\partial}{\partial z}\Gamma(K_z)\frac{\partial \theta}{\partial z} = \begin{cases} \mathcal{F}_\theta & \text{(surface)} \\ 0 & \text{(interior)} \end{cases} \tag{3.40}$$

$$\frac{Ds}{Dt} - \nabla_h \cdot K_h \nabla_h s - \frac{\partial}{\partial z}\Gamma(K_z)\frac{\partial s}{\partial z} = \begin{cases} \mathcal{F}_s & \text{(surface)} \\ 0 & \text{(interior)} \end{cases} \tag{3.41}$$

$$g\rho_0\eta + \int_{-z}^{0} \rho^{'} dz = p^{'} \tag{3.42}$$

where $u = \frac{Dx}{Dt} = r\cos(\phi)\frac{D\lambda}{Dt}$ and $v = \frac{Dy}{Dt} = r\frac{D\phi}{Dt}$ are the zonal and meridional components of the flow vector, $\vec{u}$, on the sphere. As described in MITgcm Numerical Solution Procedure 2, the time evolution of potential temperature, θ, equation is solved prognostically. The total pressure, p, is diagnosed by summing pressure due to surface elevation η and the hydrostatic pressure.

3.12.2.1 Numerical Stability Criteria

The Laplacian dissipation coefficient, A_h, is set to $5 \times 10^5 ms^{-1}$. This value is chosen to yield a Munk layer width [*Adcroft*, 1995],

$$M_w = \pi(\frac{A_h}{\beta})^{\frac{1}{3}} \tag{3.43}$$

of $\approx$ 600km. This is greater than the model resolution in low-latitudes, $\Delta x \approx$ 400km, ensuring that the frictional boundary layer is adequately resolved.

The model is stepped forward with a time step $\Delta t_\theta = 24$ hours for thermodynamic variables and $\Delta t_v =$ 30 minutes for momentum terms. With this time step, the stability parameter to the horizontal Laplacian friction [*Adcroft*, 1995]

$$S_l = 4\frac{A_h \Delta t_v}{\Delta x^2} \tag{3.44}$$

evaluates to 0.6 at a latitude of $\phi = 80°$, which is above the 0.3 upper limit for stability, but the zonal grid spacing Δx is smallest at $\phi = 80°$ where $\Delta x = r\cos(\phi)\Delta\phi \approx$ 77km and the stability criterion is already met 1 grid cell equatorwards (at $\phi = 76°$).
The vertical dissipation coefficient, A_z, is set to $1 \times 10^{-3}\mathrm{m}^2\mathrm{s}^{-1}$. The associated stability limit

$$S_l = 4\frac{A_z \Delta t_v}{\Delta z^2} \tag{3.45}$$

evaluates to 0.0029 for the smallest model level spacing ($\Delta z_1 = 50$m) which is well below the upper stability limit.

The numerical stability for inertial oscillations [*Adcroft*, 1995]

$$S_i = f^2 {\Delta t_v}^2 \tag{3.46}$$

evaluates to 0.07 for $f = 2\omega\sin(80°) = 1.43\times10^{-4}\ \mathrm{s}^{-1}$, which is below the $S_i < 1$ upper limit for stability.

The advective CFL [*Adcroft*, 1995] for a extreme maximum horizontal flow speed of $|\vec{u}| = 2ms^{-1}$

$$S_a = \frac{|\vec{u}|\Delta t_v}{\Delta x} \tag{3.47}$$

evaluates to 5×10^{-2}. This is well below the stability limit of 0.5.

The stability parameter for internal gravity waves propagating with a maximum speed of $c_g = 10\ \mathrm{ms}^{-1}$ [*Adcroft*, 1995]

$$S_c = \frac{c_g \Delta t_v}{\Delta x} \tag{3.48}$$

evaluates to 2.3×10^{-1}. This is close to the linear stability limit of 0.5.

3.12.3 Experiment Configuration

The model configuration for this experiment resides under the directory *tutorial_global_oce_latlon/*. The experiment files

- *input/data*
- *input/data.pkg*

- *input/eedata*,
- *input/trenberth_taux.bin*,
- *input/trenberth_tauy.bin*,
- *input/lev_s.bin*,
- *input/lev_t.bin*,
- *input/lev_sss.bin*,
- *input/lev_sst.bin*,
- *input/bathymetry.bin*,
- *code/SIZE.h*.

contain the code customizations and parameter settings for these experiments. Below we describe the customizations to these files associated with this experiment.

3.12.3.1 Driving Datasets

Figures (3.5-3.10) show the relaxation temperature (θ^*) and salinity (S^*) fields, the wind stress components (τ_x and τ_y), the heat flux (Q) and the net fresh water flux ($\mathcal{E}-\mathcal{P}-\mathcal{R}$) used in equations (3.31-3.34). The figures also indicate the lateral extent and coastline used in the experiment. Figure (— *missing figure* —) shows the depth contours of the model domain.

3.12.3.2 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 7–8

  ```
  tRef= 15*20.,
  sRef= 15*35.,
  ```

  set reference values for potential temperature and salinity at each model level in units of °C and ppt. The entries are ordered from surface to depth. Density is calculated from anomalies at each level evaluated with respect to the reference values set here.

S/R INI_THETA(*ini_theta.F*) *S/R INI_SALT*(*ini_salt.F*)

- Line 9,

  ```
  viscAr=1.E-3,
  ```

  this line sets the vertical Laplacian dissipation coefficient to $1 \times 10^{-3}\mathrm{m}^2\mathrm{s}^{-1}$. Boundary conditions for this operator are specified later.

S/R CALC_DIFFUSIVITY(*calc_diffusivity.F*)

- Line 10,

  ```
  viscAh=5.E5,
  ```

  this line sets the horizontal Laplacian frictional dissipation coefficient to $5 \times 10^{5}\mathrm{m}^2\mathrm{s}^{-1}$. Boundary conditions for this operator are specified later.

- Lines 11 and 13,

```
diffKhT=0.,
diffKhS=0.,
```

set the horizontal diffusion coefficient for temperature and salinity to 0, since package GMREDI is used.

- Lines 12 and 14,

```
diffKrT=3.E-5,
diffKrS=3.E-5,
```

set the vertical diffusion coefficient for temperature and salinity to $3 \times 10^{-5}\,\mathrm{m}^2\mathrm{s}^{-1}$. The boundary condition on this operator is $\frac{\partial}{\partial z} = 0$ at both the upper and lower boundaries.

- Lines 15–17

```
rhonil=1035.,
rhoConstFresh=1000.,
eosType = 'JMD95Z',
```

set the reference densities for sea water and fresh water, and selects the equation of state [*Jackett and McDougall*, 1995]

S/R FIND_RHO (*find_rho.F*)
S/R FIND_ALPHA (*find_alpha.F*)
S/R CALC_PHI_HYD (*calc_phi_hyd.F*)
S/R INI_CG2D (*ini_cg2d.F*)
S/R INI_CG3D (*ini_cg3d.F*)
S/R INI_PARMS (*ini_parms.F*)
S/R SOLVE_FOR_PRESSURE (*solve_for_pressure.F*)

- Lines 18–19,

```
ivdc_kappa=100.,
implicitDiffusion=.TRUE.,
```

specify an "implicit diffusion" scheme with increased vertical diffusivity of 100 m^2/s in case of instable stratification.

- ...

- Line 28,

```
readBinaryPrec=32,
```

Sets format for reading binary input datasets holding model fields to use 32-bit representation for floating-point numbers.

S/R READ_WRITE_FLD (*read_write_fld.F*)
S/R READ_WRITE_REC (*read_write_rec.F*)

- Line 33,

```
cg2dMaxIters=500,
```

Sets maximum number of iterations the two-dimensional, conjugate gradient solver will use, **irrespective of convergence criteria being met**.

S/R CG2D (*cg2d.F*)

- Line 34,

```
cg2dTargetResidual=1.E-13,
```

Sets the tolerance which the two-dimensional, conjugate gradient solver will use to test for convergence in equation 2.20 to 1×10^{-13}. Solver will iterate until tolerance falls below this value or until the maximum number of solver iterations is reached.

S/R CG2D (*cg2d.F*)

- Line 39,

```
nIter0=0,
```

Sets the starting time for the model internal time counter. When set to non-zero this option implicitly requests a checkpoint file be read for initial state. By default the checkpoint file is named according to the integer number of time step value `nIter0`. The internal time counter works in seconds. Alternatively, `startTime` can be set.

- Line 40,

```
nTimeSteps=20,
```

Sets the time step number at which this simulation will terminate. At the end of a simulation a checkpoint file is automatically written so that a numerical experiment can consist of multiple stages. Alternatively `endTime` can be set.

- Line 44,

```
deltaTmom=1800.,
```

Sets the timestep Δt_v used in the momentum equations to 30 mins. See section 2.2.

S/R TIMESTEP(*timestep.F*)

- Line 45,

```
tauCD=321428.,
```

Sets the D-grid to C-grid coupling time scale τ_{CD} used in the momentum equations.

S/R INI_PARMS(*ini_parms.F*) *S/R MOM_FLUXFORM*(*mom_fluxform.F*)

- Lines 46–48,

```
deltaTtracer=86400.,
deltaTClock = 86400.,
deltaTfreesurf= 86400.,
```

Sets the default timestep, Δt_θ, for tracer equations and implicit free surface equations to 24 hours. See section 2.2.

S/R TIMESTEP_TRACER(*timestep_tracer.F*)

- Line 76,

```
bathyFile='bathymetry.bin'
```

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 32-bit binary numbers giving the depth of the model at each grid cell, ordered with the x coordinate varying fastest. The points are ordered from low coordinate to high coordinate for both axes. The units and orientation of the depths in this file are the same as used in the MITgcm code. In this experiment, a depth of $0m$ indicates a solid wall and a depth of $< 0m$ indicates open ocean.

- Lines 79–80,

  ```
  zonalWindFile='trenberth_taux.bin'
  meridWindFile='trenberth_tauy.bin'
  ```

  These lines specify the names of the files from which the x- and y- direction surface wind stress is read. These files are also three-dimensional $(x, y, time)$ maps and are enumerated and formatted in the same manner as the bathymetry file.

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

```
1 # ====================
2 # | Model parameters |
3 # ====================
4 #
5 # Continuous equation parameters
6  &PARM01
7  tRef = 15*20.,
8  sRef = 15*35.,
9  viscAr=1.E-3,
10  viscAh=5.E5,
11  diffKhT=0.,
12  diffKrT=3.E-5,
13  diffKhS=0.,
14  diffKrS=3.E-5,
15  rhonil=1035.,
16  rhoConstFresh=1000.,
17  eosType = 'JMD95Z',
18  ivdc_kappa=100.,
19  implicitDiffusion=.TRUE.,
20  allowFreezing=.TRUE.,
21  exactConserv=.TRUE.,
22  useRealFreshWaterFlux=.TRUE.,
23  useCDscheme=.TRUE.,
24 # turn on looped cells
25  hFacMin=.05,
26  hFacMindr=50.,
27 # set precision of data files
28  readBinaryPrec=32,
29  &
30
31 # Elliptic solver parameters
32  &PARM02
33  cg2dMaxIters=500,
34  cg2dTargetResidual=1.E-13,
35  &
36
37 # Time stepping parameters
38  &PARM03
39  nIter0=      0,
40  nTimeSteps = 20,
41 # 100 years of integration will yield a reasonable flow field
42 # startTime =          0.,
43 # endTime   = 3110400000.,
44  deltaTmom = 1800.,
45  tauCD =     321428.,
46  deltaTtracer= 86400.,
47  deltaTClock = 86400.,
48  deltaTfreesurf= 86400.,
49  abEps = 0.1,
```

```
 pChkptFreq= 1728000.,
 dumpFreq=   864000.,
 taveFreq=   864000.,
 monitorFreq=1.,
# 2 months restoring timescale for temperature
 tauThetaClimRelax=  5184000.,
# 6 months restoring timescale for salinity
 tauSaltClimRelax = 15552000.,
 periodicExternalForcing=.TRUE.,
 externForcingPeriod=2592000.,
 externForcingCycle=31104000.,
 &

# Gridding parameters
 &PARM04
 usingSphericalPolarGrid=.TRUE.,
 delR= 50.,  70., 100., 140., 190.,
      240., 290., 340., 390., 440.,
      490., 540., 590., 640., 690.,
 ygOrigin=-80.,
 dySpacing=4.,
 dxSpacing=4.,
 &

# Input datasets
 &PARM05
 bathyFile=      'bathymetry.bin',
 hydrogThetaFile='lev_t.bin',
 hydrogSaltFile= 'lev_s.bin',
 zonalWindFile=  'trenberth_taux.bin',
 meridWindFile=  'trenberth_tauy.bin',
 thetaClimFile=  'lev_sst.bin',
 saltClimFile=   'lev_sss.bin',
 surfQFile=      'ncep_qnet.bin',
 the_run_name=   'global_oce_latlon',
# fresh water flux is turned on, comment next line to it turn off
# (maybe better with surface salinity restoring)
 EmPmRFile=      'ncep_emp.bin',
 &
```

3.12.3.3 File *input/data.pkg*

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

3.12.3.4 File *input/eedata*

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

3.12.3.5 Files*input/trenberth_taux.bin* and *input/trenberth_tauy.bin*

The *input/trenberth_taux.bin* and *input/trenberth_tauy.bin* files specify a three-dimensional $(x, y, time)$ map of wind stress, (τ_x, τ_y), values [*Trenberth et al.*, 1990]. The units used are Nm^{-2}.

3.12.3.6 File *input/bathymetry.bin*

The *input/bathymetry.bin* file specifies a two-dimensional (x, y) map of depth values. For this experiment values range between 0 and -5200 m, and have been derived from ETOPO5. The file contains a raw binary stream of data that is enumerated in the same way as standard MITgcm two-dimensional, horizontal arrays.

3.12.3.7 File *code/SIZE.h*

Four lines are customized in this file for the current experiment

- Line 40,

  ```
  sNx=45,
  ```

  this line sets the number of grid points of each tile (or sub-domain) along the x-coordinate axis.

- Line 41,

  ```
  sNy=40,
  ```

  this line sets the number of grid points of each tile (or sub-domain) along the y-coordinate axis.

- Lines 46 and 44,

  ```
  nPx=1,
  nSx=2,
  ```

  theses lines set, respectively, the number of processes and the number of tiles per process along the x-coordinate axis. Therefore, the total number of grid points along the x-coordinate axis corresponding to the full domain extent is $Nx = sNx * sNx * nPx = 90$

- Line 50,

  ```
  Nr=15
  ```

  this line sets the vertical domain extent in grid points.

```
 1 C $Header: /u/gcmpack/MITgcm/verification/tutorial_global_oce_latlon/code/SIZE.h,v 1.1 2006/07/14 20:35:2
 2 C $Name:  $
 3
 4 C
 5 C     /==========================================================\
 6 C     | SIZE.h Declare size of underlying computational grid.    |
 7 C     |==========================================================|
 8 C     | The design here support a three-dimensional model grid   |
 9 C     | with indices I,J and K. The three-dimensional domain     |
10 C     | is comprised of nPx*nSx blocks of size sNx along one axis|
11 C     | nPy*nSy blocks of size sNy along another axis and one    |
12 C     | block of size Nz along the final axis.                   |
13 C     | Blocks have overlap regions of size OLx and OLy along the|
14 C     | dimensions that are subdivided.                          |
15 C     \==========================================================/
16 C     Voodoo numbers controlling data layout.
17 C     sNx - No. X points in sub-grid.
18 C     sNy - No. Y points in sub-grid.
19 C     OLx - Overlap extent in X.
20 C     OLy - Overlat extent in Y.
21 C     nSx - No. sub-grids in X.
22 C     nSy - No. sub-grids in Y.
23 C     nPx - No. of processes to use in X.
24 C     nPy - No. of processes to use in Y.
25 C     Nx  - No. points in X for the total domain.
26 C     Ny  - No. points in Y for the total domain.
27 C     Nr  - No. points in Z for full process domain.
28       INTEGER sNx
29       INTEGER sNy
30       INTEGER OLx
```

```
      INTEGER OLy
      INTEGER nSx
      INTEGER nSy
      INTEGER nPx
      INTEGER nPy
      INTEGER Nx
      INTEGER Ny
      INTEGER Nr
      PARAMETER (
     &           sNx =  45,
     &           sNy =  40,
     &           OLx =   2,
     &           OLy =   2,
     &           nSx =   2,
     &           nSy =   1,
     &           nPx =   1,
     &           nPy =   1,
     &           Nx  = sNx*nSx*nPx,
     &           Ny  = sNy*nSy*nPy,
     &           Nr  =  15)

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

```

3.12.3.8 Other Files

3.13 Global Ocean Simulation at 4° Resolution in Pressure Coordinates

(in directory: *verification/tutorial_global_oce_in_p/*)

This example experiment demonstrates using the MITgcm to simulate the planetary ocean circulation in pressure coordinates, that is, without making the Boussinesq approximations. The files for this experiment can be found in the verification directory under tutorial_global_oce_in_p. The simulation is configured as a near copy of tutorial_global_oce_latlon (Section 3.12). with realistic geography and bathymetry on a $4^\circ \times 4^\circ$ spherical polar grid. Fifteen levels are used in the vertical, ranging in thickness from 50.4089 dbar $\approx$ 50 m at the surface to 710.33 dbar $\approx$ 690 m at depth, giving a maximum model depth of 5302.3122 dbar $\approx$ 5200 km. At this resolution, the configuration can be integrated forward for thousands of years on a single processor desktop computer.

3.13.1 Overview

The model is forced with climatological wind stress data from *Trenberth et al.* [1990] and surface flux data from *Jiang et al.* [1999]. Climatological data [*Levitus and T.P.Boyer*, 1994b] is used to initialize the model hydrography. *Levitus and T.P.Boyer* seasonal climatology data is also used throughout the calculation to provide additional air-sea fluxes. These fluxes are combined with the Jiang climatological estimates of surface heat flux, resulting in a mixed boundary condition of the style described in *Haney* [1971]. Altogether, this yields the following forcing applied in the model surface layer.

$$\mathcal{F}_u = g\frac{\tau_x}{\Delta p_s} \tag{3.49}$$

$$\mathcal{F}_v = g\frac{\tau_y}{\Delta p_s} \tag{3.50}$$

$$\mathcal{F}_\theta = -g\lambda_\theta(\theta - \theta^*) - \frac{1}{C_p \Delta p_s}\mathcal{Q} \tag{3.51}$$

$$\mathcal{F}_s = +g\rho_{FW}\frac{S}{\rho\Delta p_s}(\mathcal{E} - \mathcal{P} - \mathcal{R}) \tag{3.52}$$

where $\mathcal{F}_u$, $\mathcal{F}_v$, $\mathcal{F}_\theta$, $\mathcal{F}_s$ are the forcing terms in the zonal and meridional momentum and in the potential temperature and salinity equations respectively. The term Δp_s represents the top ocean layer thickness in Pa. It is used in conjunction with a reference density, ρ_{FW} (here set to 999.8 kg m^{-3}), the surface salinity, S, and a specific heat capacity, C_p (here set to 4000 J $^\circ$C^{-1} kg^{-1}), to convert input dataset values into time tendencies of potential temperature (with units of $^\circ$C s^{-1}), salinity (with units ppt s^{-1}) and velocity (with units m s^{-2}). The externally supplied forcing fields used in this experiment are τ_x, τ_y, θ^*, $\mathcal{Q}$ and $\mathcal{E} - \mathcal{P} - \mathcal{R}$. The wind stress fields (τ_x, τ_y) have units of N m^{-2}. The temperature forcing fields (θ^* and Q) have units of $^\circ$C and W m^{-2} respectively. The salinity forcing fields ($\mathcal{E} - \mathcal{P} - \mathcal{R}$) has units of m s^{-1} respectively. The source files and procedures for ingesting these data into the simulation are described in the experiment configuration discussion in section 3.12.3.

3.13.2 Discrete Numerical Configuration

Due to the pressure coordinate, the model can only be hydrostatic [*de Szoeke and Samelson*, 2002]. The domain is discretized with a uniform grid spacing in latitude and longitude on the sphere $\Delta\phi = \Delta\lambda = 4^\circ$, so that there are ninety grid cells in the zonal and forty in the meridional direction. The internal model coordinate variables x and y are initialized according to

$$x = r\cos(\phi),\ \Delta x = r\cos(\Delta\phi) \tag{3.53}$$

$$y = r\lambda,\ \Delta y = r\Delta\lambda \tag{3.54}$$

Arctic polar regions are not included in this experiment. Meridionally the model extends from 80°S to 80°N. Vertically the model is configured with fifteen layers with the following thicknesses

$$
\begin{aligned}
\Delta p_1 &= 7103300.720021\,\mathrm{Pa}, \\
\Delta p_2 &= 6570548.440790\,\mathrm{Pa}, \\
\Delta p_3 &= 6041670.010249\,\mathrm{Pa}, \\
\Delta p_4 &= 5516436.666057\,\mathrm{Pa}, \\
\Delta p_5 &= 4994602.034410\,\mathrm{Pa}, \\
\Delta p_6 &= 4475903.435290\,\mathrm{Pa}, \\
\Delta p_7 &= 3960063.245801\,\mathrm{Pa}, \\
\Delta p_8 &= 3446790.312651\,\mathrm{Pa}, \\
\Delta p_9 &= 2935781.405664\,\mathrm{Pa}, \\
\Delta p_{10} &= 2426722.705046\,\mathrm{Pa}, \\
\Delta p_{11} &= 1919291.315988\,\mathrm{Pa}, \\
\Delta p_{12} &= 1413156.804970\,\mathrm{Pa}, \\
\Delta p_{13} &= 1008846.750166\,\mathrm{Pa}, \\
\Delta p_{14} &= 705919.025481\,\mathrm{Pa}, \\
\Delta p_{15} &= 504089.693499\,\mathrm{Pa},
\end{aligned}
$$

(here the numeric subscript indicates the model level index number, `k`; note, that the surface layer has the highest index number 15) to give a total depth, H, of -5200m. In pressure, this is $p_b^0 = 53023122.566084\,\mathrm{Pa}$. The implicit free surface form of the pressure equation described in *Marshall et al.* [1997b] with the nonlinear extension by *Campin et al.* [2004] is employed. A Laplacian operator, ∇^2, provides viscous dissipation. Thermal and haline diffusion is also represented by a Laplacian operator.

Wind-stress forcing is added to the momentum equations in (3.55) for both the zonal flow, u and the meridional flow v, according to equations (3.49) and (3.50). Thermodynamic forcing inputs are added to the equations in (3.55) for potential temperature, θ, and salinity, S, according to equations (3.51) and (3.52). This produces a set of equations solved in this configuration as follows:

$$\frac{Du}{Dt} - fv + \frac{1}{\rho}\frac{\partial \Phi'}{\partial x} - \nabla_h \cdot A_h \nabla_h u - (g\rho_0)^2 \frac{\partial}{\partial p} A_r \frac{\partial u}{\partial p} = \begin{cases} \mathcal{F}_u & \text{(surface)} \\ 0 & \text{(interior)} \end{cases} \tag{3.55}$$

$$\frac{Dv}{Dt} + fu + \frac{1}{\rho}\frac{\partial \Phi'}{\partial y} - \nabla_h \cdot A_h \nabla_h v - (g\rho_0)^2 \frac{\partial}{\partial p} A_r \frac{\partial v}{\partial p} = \begin{cases} \mathcal{F}_v & \text{(surface)} \\ 0 & \text{(interior)} \end{cases} \tag{3.56}$$

$$\frac{\partial p_b}{\partial t} + \nabla_h \cdot \vec{u} = 0 \tag{3.57}$$

$$\frac{D\theta}{Dt} - \nabla_h \cdot K_h \nabla_h \theta - (g\rho_0)^2 \frac{\partial}{\partial p} \Gamma(K_r) \frac{\partial \theta}{\partial p} = \begin{cases} \mathcal{F}_\theta & \text{(surface)} \\ 0 & \text{(interior)} \end{cases} \tag{3.58}$$

$$\frac{Ds}{Dt} - \nabla_h \cdot K_h \nabla_h s - (g\rho_0)^2 \frac{\partial}{\partial p} \Gamma(K_r) \frac{\partial S}{\partial p} = \begin{cases} \mathcal{F}_s & \text{(surface)} \\ 0 & \text{(interior)} \end{cases} \tag{3.59}$$

$$\Phi'^{(0)}_{-H} + \alpha_0 p_b + \int_0^p \alpha' dp = \Phi' \tag{3.60}$$

where $u = \frac{Dx}{Dt} = r\cos(\phi)\frac{D\lambda}{Dt}$ and $v = \frac{Dy}{Dt} = r\frac{D\phi}{Dt}$ are the zonal and meridional components of the flow vector, $\vec{u}$, on the sphere. As described in MITgcm Numerical Solution Procedure 2, the time evolution of potential temperature, θ, equation is solved prognostically. The full geopotential height, Φ, is diagnosed by summing the geopotential height anomalies Φ' due to bottom pressure p_b and density variations. The integration of the hydrostatic equation is started at the bottom of the domain. The condition of $p = 0$ at the sea surface requires a time-independent integration constant for the height anomaly due to density variations $\Phi'^{(0)}_{-H}$, which is provided as an input field.

3.13.3 Experiment Configuration

The model configuration for this experiment resides under the directory *tutorial_examples/global_ocean_circulation/*. The experiment files

- *input/data*
- *input/data.pkg*
- *input/eedata*,
- *input/topog.bin*,
- *input/deltageopotjmd95.bin*,
- *input/lev_s.bin*,
- *input/lev_t.bin*,
- *input/trenberth_taux.bin*,
- *input/trenberth_tauy.bin*,
- *input/lev_sst.bin*,
- *input/shi_qnet.bin*,
- *input/shi_empmr.bin*,
- *code/CPP_EEOPTIONS.h*
- *code/CPP_OPTIONS.h*,
- *code/SIZE.h*.

contain the code customizations and parameter settings for these experiments. Below we describe the customizations to these files associated with this experiment.

3.13.3.1 Driving Datasets

Figures (3.5-3.10) show the relaxation temperature (θ^*) and salinity (S^*) fields, the wind stress components (τ_x and τ_y), the heat flux (Q) and the net fresh water flux ($\mathcal{E}-\mathcal{P}-\mathcal{R}$) used in equations 3.49-3.52. The figures also indicate the lateral extent and coastline used in the experiment. Figure (3.11) shows the depth contours of the model domain.

3.13.3.2 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 15,

  ```
  viscAr=1.721611620915750E+05,
  ```

  this line sets the vertical Laplacian dissipation coefficient to $1.72161162091575\times10^{5}\mathrm{Pa}^2\mathrm{s}^{-1}$. Note that, the factor $(g\rho)^2$ needs to be included in this line. Boundary conditions for this operator are specified later. This variable is copied into model general vertical coordinate variable **viscAr**.

S/R CALC_DIFFUSIVITY(*calc_diffusivity.F*)

- Line 9–10,

  ```
  viscAh=3.E5,
  no_slip_sides=.TRUE.
  ```

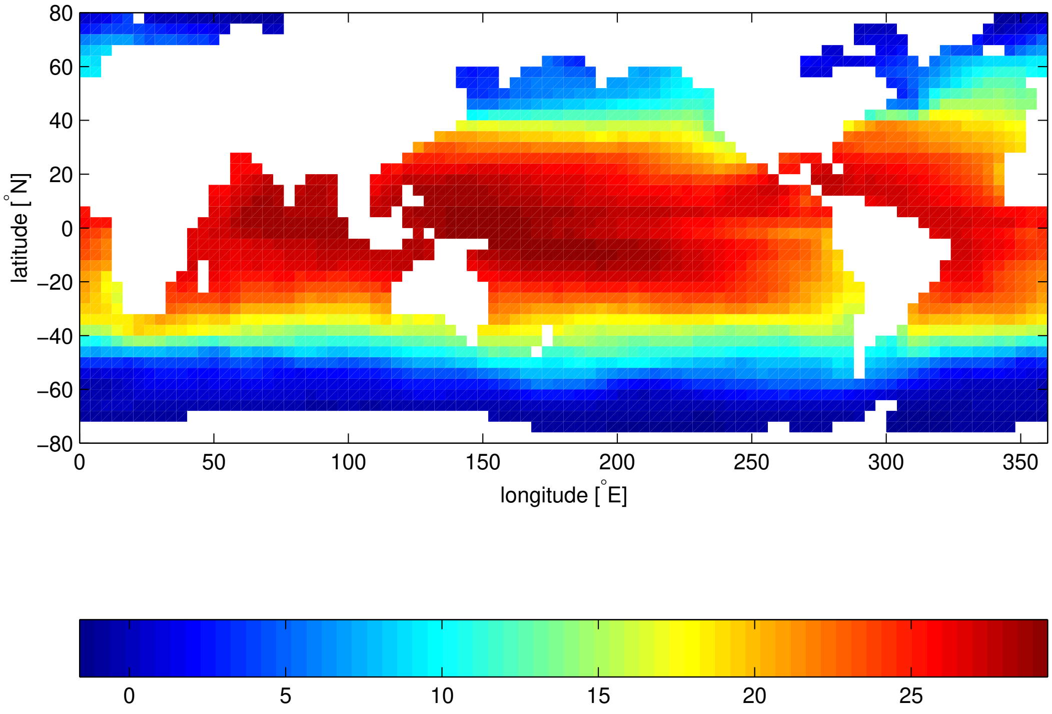


Figure 3.5: Annual mean of relaxation temperature [°C]

these lines set the horizontal Laplacian frictional dissipation coefficient to $3 \times 10^5 \mathrm{m}^2\mathrm{s}^{-1}$ and specify a no-slip boundary condition for this operator, that is, $u = 0$ along boundaries in y and $v = 0$ along boundaries in x.

- Lines 11-13,

```
 viscAr =1.721611620915750e5,
#viscAz =1.67E-3,
 no_slip_bottom=.FALSE.,
```

These lines set the vertical Laplacian frictional dissipation coefficient to $1.721611620915750 \times 10^5\,\mathrm{Pa}^2\mathrm{s}^{-1}$, which corresponds to $1.67 \times 10^{-3}\,\mathrm{m}^2\mathrm{s}^{-1}$ in the commented line, and specify a free slip boundary condition for this operator, that is, $\frac{\partial u}{\partial p} = \frac{\partial v}{\partial p} = 0$ at $p = p_b^0$, where p_b^0 is the local bottom pressure of the domain at rest. Note that, the factor $(g\rho)^2$ needs to be included in this line.

- Line 14,

```
 diffKhT=1.E3,
```

this line sets the horizontal diffusion coefficient for temperature to $1000\,\mathrm{m}^2\mathrm{s}^{-1}$. The boundary condition on this operator is $\frac{\partial}{\partial x} = \frac{\partial}{\partial y} = 0$ on all boundaries.

- Line 15–16,

```
 diffKrT=5.154525811125000e3,
#diffKzT=0.5E-4,
```

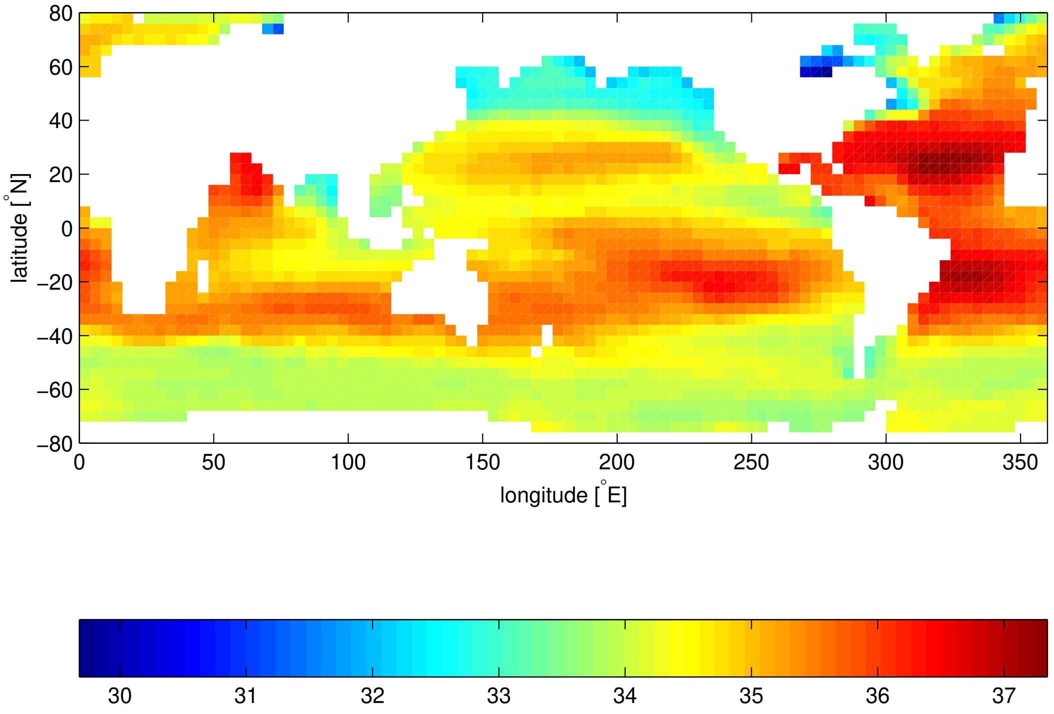


Figure 3.6: Annual mean of relaxation salinity [PSU]

this line sets the vertical diffusion coefficient for temperature to $5.154525811125 \times 10^3\,\mathrm{Pa}^2\mathrm{s}^{-1}$, which corresponds to $5 \times 10^{-4}\,\mathrm{m}^2\mathrm{s}^{-1}$ in the commented line. Note that, the factor $(g\rho)^2$ needs to be included in this line. The boundary condition on this operator is $\frac{\partial}{\partial p} = 0$ at both the upper and lower boundaries.

- Line 17–19,

```
 diffKhS=1.E3,
 diffKrS=5.154525811125000e3,
#diffKzS=0.5E-4,
```

These lines set the same values for the diffusion coefficients for salinity as for temperature.

- Line 20–22,

```
 implicitDiffusion=.TRUE.,
 ivdc_kappa=1.030905162225000E9,
#ivdc_kappa=10.0,
```

Select implicit diffusion as a convection scheme and set coefficient for implicit vertical diffusion to $1.030905162225 \times 10^9\,\mathrm{Pa}^2\mathrm{s}^{-1}$, which corresponds to $10\,\mathrm{m}^2\,\mathrm{s}^{-1}$.

- Line 23-24,

```
 gravity=9.81,
 gravitySign=-1.D0,
```

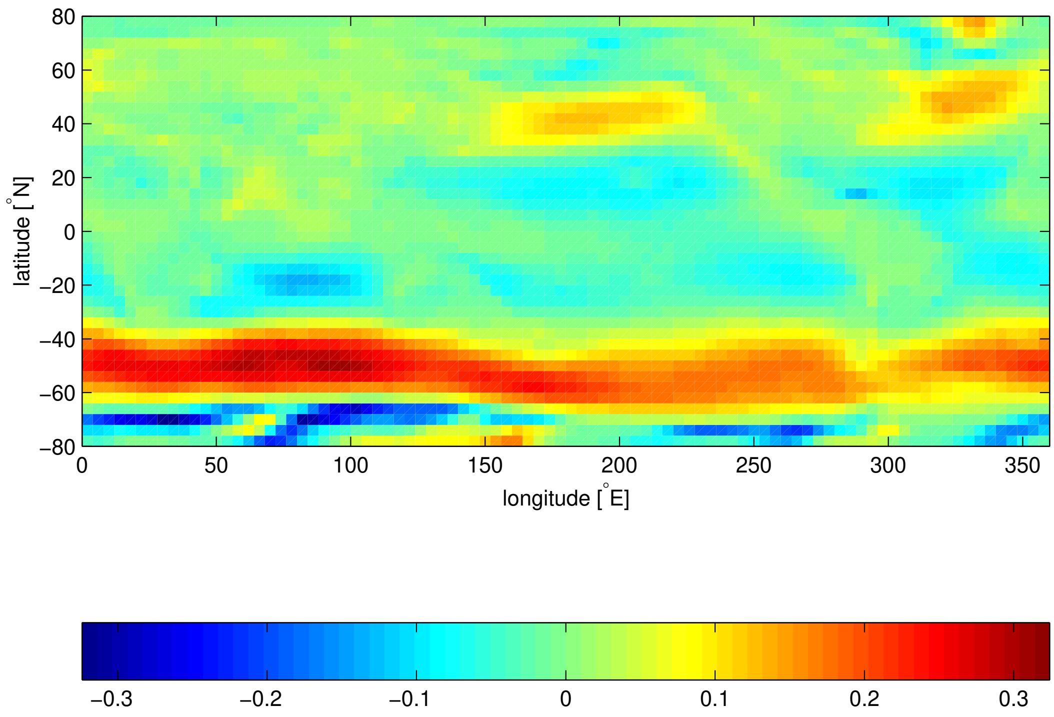


Figure 3.7: Annual mean of zonal wind stress component [$\mathrm{N m\, m^{-2}}$]

These lines set the gravitational acceleration coefficient to $9.81\mathrm{ms}^{-1}$ and define the upward direction relative to the direction of increasing vertical coordinate (in pressure coordinates, up is in the direction of decreasing pressure)

- Line 25,

```
rhoNil=1035.,
```

sets the reference density of sea water to $1035\,\mathrm{kg\,m^{-3}}$.

S/R CALC_PHI_HYD (*calc_phi_hyd.F*)
S/R INI_CG2D (*ini_cg2d.F*)
S/R INI_CG3D (*ini_cg3d.F*)
S/R INI_PARMS (*ini_parms.F*)
S/R SOLVE_FOR_PRESSURE (*solve_for_pressure.F*)

- Line 28

```
eosType='JMD95P',
```

Selects the full equation of state according to *Jackett and McDougall* [1995]. The only other sensible choice is the equation of state by *McDougall et al.* [2003], 'MDJFW'. All other equations of state do not make sense in this configuration.

S/R FIND_RHO (*find_rho.F*)
S/R FIND_ALPHA (*find_alpha.F*)

- Line 28-29,

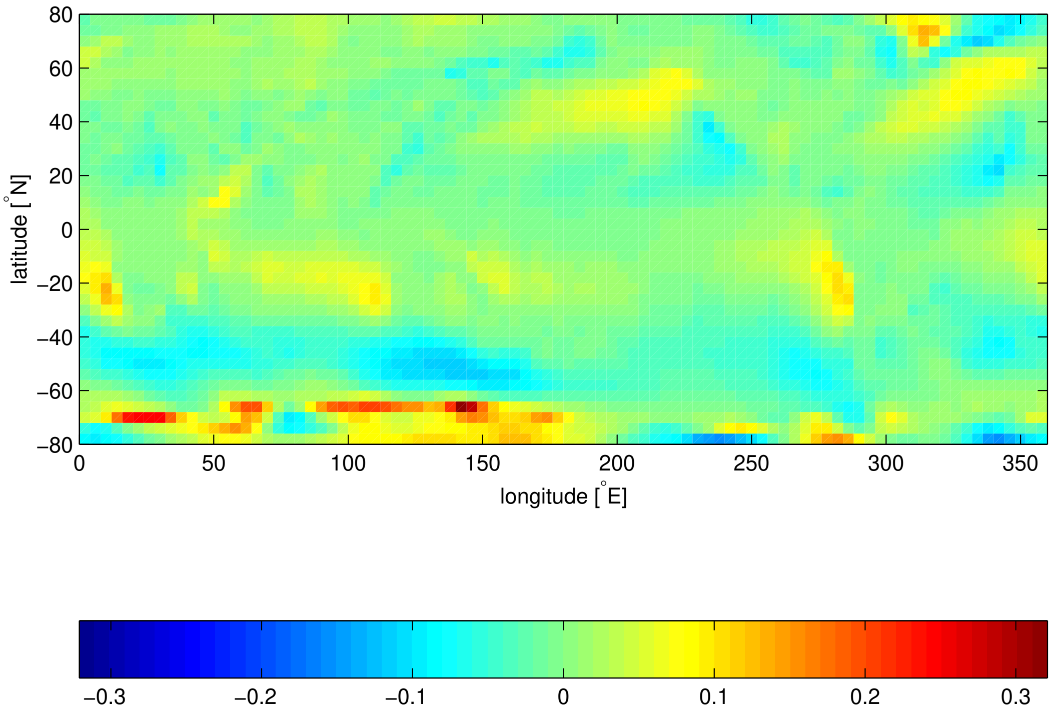


Figure 3.8: Annual mean of meridional wind stress component $[\mathrm{N\,m\,m^{-2}}]$

```
rigidLid=.FALSE.,
implicitFreeSurface=.TRUE.,
```

Selects the barotropic pressure equation to be the implicit free surface formulation.

- Line 30

```
exactConserv=.TRUE.,
```

Select a more accurate conservation of properties at the surface layer by including the horizontal velocity divergence to update the free surface.

- Line 31–33

```
nonlinFreeSurf=3,
hFacInf=0.2,
hFacSup=2.0,
```

Select the nonlinear free surface formulation and set lower and upper limits for the free surface excursions.

- Line 34

```
useRealFreshWaterFlux=.FALSE.,
```

Select virtual salt flux boundary condition for salinity. The freshwater flux at the surface only affect the surface salinity, but has no mass flux associated with it

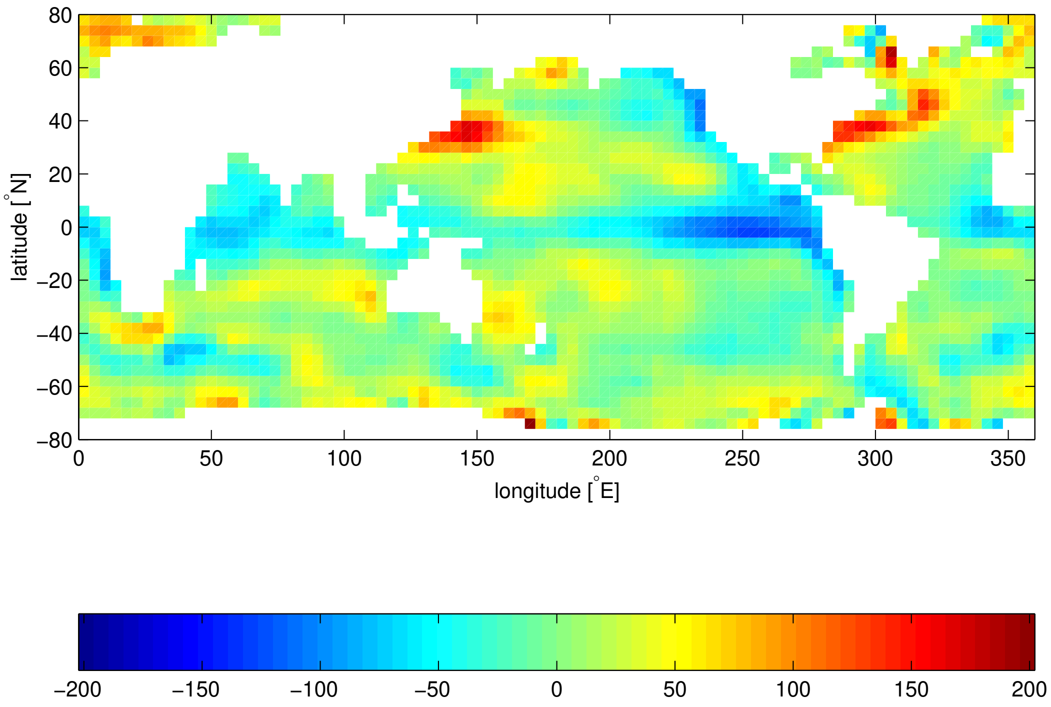


Figure 3.9: Annual mean heat flux [W m^{-2}]

- Line 35–36,

```
readBinaryPrec=64,
writeBinaryPrec=64,
```

Sets format for reading binary input datasets and writing binary output datasets holding model fields to use 64-bit representation for floating-point numbers.

S/R READ_WRITE_FLD (*read_write_fld.F*) *S/R READ_WRITE_REC* (*read_write_rec.F*)

- Line 42,

```
cg2dMaxIters=200,
```

Sets maximum number of iterations the two-dimensional, conjugate gradient solver will use, **irrespective of convergence criteria being met**.

S/R CG2D (*cg2d.F*)

- Line 43,

```
cg2dTargetResidual=1.E-13,
```

Sets the tolerance which the two-dimensional, conjugate gradient solver will use to test for convergence in equation 2.20 to 1×10^{-9}. Solver will iterate until tolerance falls below this value or until the maximum number of solver iterations is reached.

S/R CG2D (*cg2d.F*)

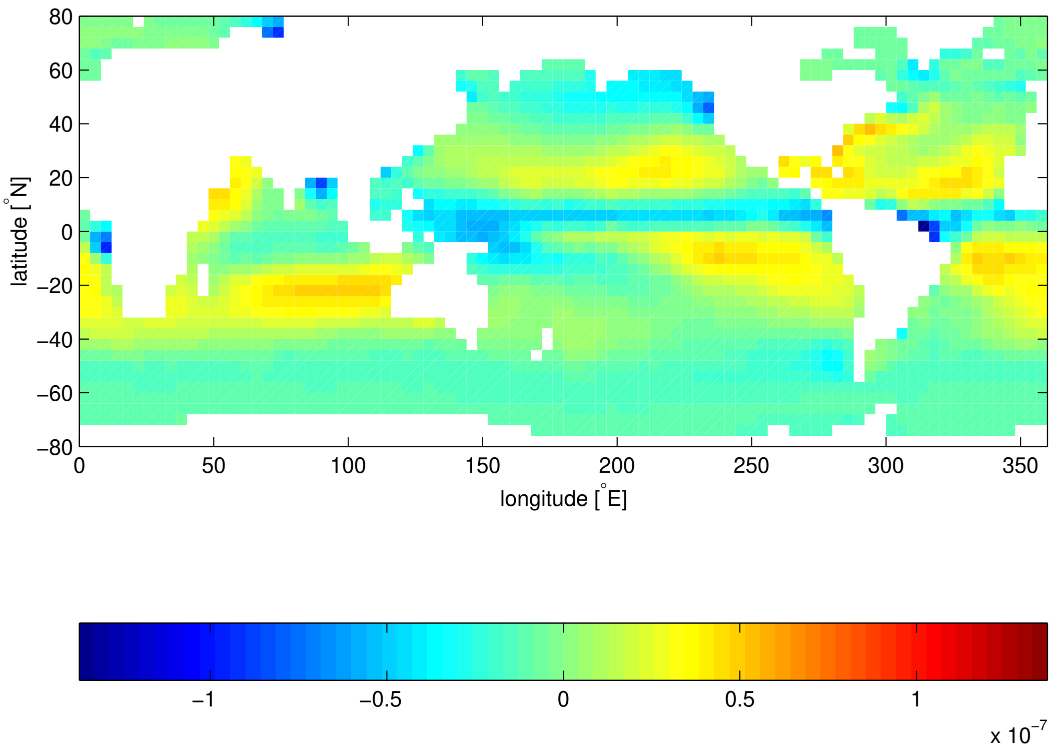


Figure 3.10: Annual mean fresh water flux (Evaporation-Precipitation) [$\mathrm{m\,s^{-1}}$]

- Line 48,

```
startTime=0,
```

Sets the starting time for the model internal time counter. When set to non-zero this option implicitly requests a checkpoint file be read for initial state. By default the checkpoint file is named according to the integer number of time steps in the **startTime** value. The internal time counter works in seconds.

- Line 49–50,

```
 endTime=8640000.,
#endTime=62208000000,
```

Sets the time (in seconds) at which this simulation will terminate. At the end of a simulation a checkpoint file is automatically written so that a numerical experiment can consist of multiple stages. The commented out setting for endTime is for a 2000 year simulation.

- Line 51–53,

```
 deltaTmom       =    1200.0,
 deltaTtracer    = 172800.0,
 deltaTfreesurf  = 172800.0,
```

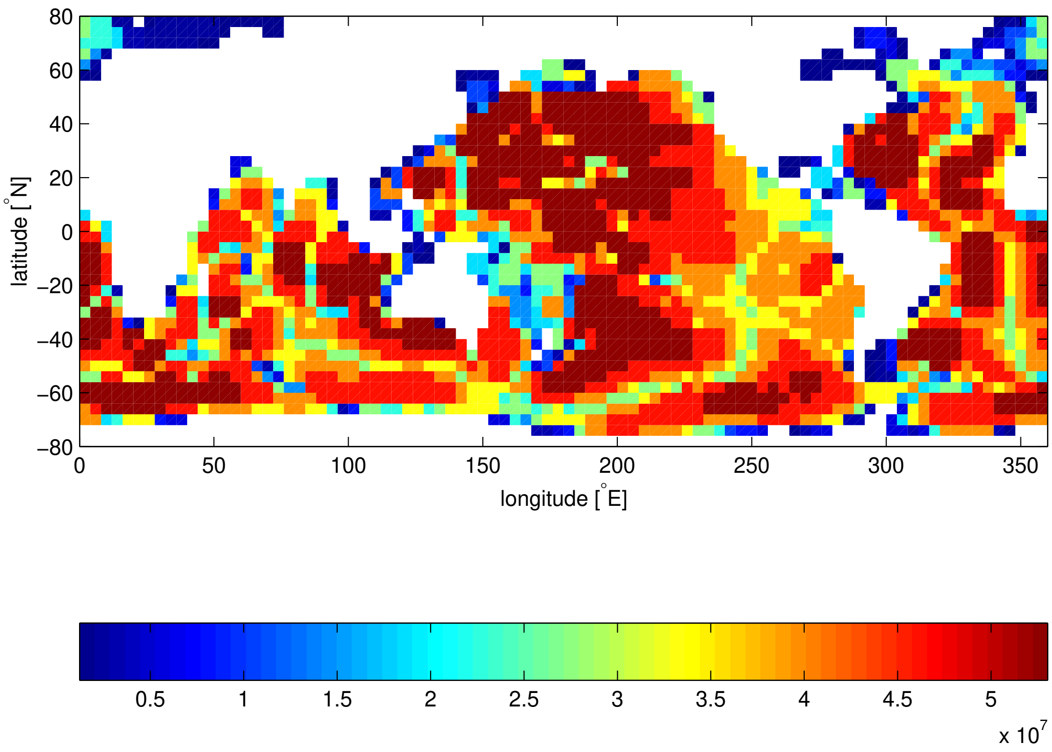


Figure 3.11: Model bathymetry in pressure units [Pa]

Sets the timestep δt_v used in the momentum equations to 20 mins and the timesteps δt_θ in the tracer equations and δt_η in the implicit free surface equation to 48 hours. See section 2.2.

S/R TIMESTEP(*timestep.F*) *S/R INI_PARMS*(*ini_parms.F*) *S/R MOM_FLUXFORM*(*mom_fluxform.F*) *S/R TIMESTEP_TRACER*(*timestep_tracer.F*)

- Line 55,

```
pChkptFreq  =3110400000.,
```

write a pick-up file every 100 years of integration.

- Line 56–58

```
dumpFreq    = 3110400000.,
taveFreq    = 3110400000.,
monitorFreq =   31104000.,
```

write model output and time-averaged model output every 100 years, and monitor statisitics every year.

- Line 59–61

```
periodicExternalForcing=.TRUE.,
externForcingPeriod=2592000.,
externForcingCycle=31104000.,
```

Allow periodic external forcing, set forcing period, during which one set of data is valid, to 1 month and the repeat cycle to 1 year.

S/R EXTERNAL_FORCING_SURF(*external_forcing_surf.F*)

- Line 62

```
tauThetaClimRelax=5184000.0,
```

Set the restoring timescale to 2 months.

S/R EXTERNAL_FORCING_SURF(*external_forcing_surf.F*)

- Line 63

```
abEps=0.1,
```

Adams-Bashford factor (see section 2.5)

- Line 68–69

```
usingCartesianGrid=.FALSE.,
usingSphericalPolarGrid=.TRUE.,
```

Select spherical grid.

- Line 70–71

```
dXspacing=4.,
dYspacing=4.,
```

Set the horizontal grid spacing in degrees spherical distance.

- Line 72

```
Ro_SeaLevel=53023122.566084,
```

specifies the total height (in r-units, i.e., pressure units) of the sea surface at rest. This is a reference value.

- Line 73

```
groundAtK1=.TRUE.,
```

specifies the reversal of the vertical indexing. The vertical index is 1 at the bottom of the doman and maximal (i.e., 15) at the surface.

- Line 74–78

```
delR=7103300.720021, \ldots
```

set the layer thickness in pressure units, starting with the bottom layer.

- Line 84–93,

```
bathyFile='topog.box'
ploadFile='deltageopotjmd95.bin'
hydrogThetaFile='lev_t.bin',
hydrogSaltFile ='lev_s.bin',
zonalWindFile  ='trenberth_taux.bin',
meridWindFile  ='trenberth_tauy.bin',
thetaClimFile  ='lev_sst.bin',
surfQFile      ='shi_qnet.bin',
EmPmRFile      ='shi_empmr.bin',
```

This line specifies the names of the files holding the bathymetry data set, the time-independent geopotential height anomaly at the bottom, initial conditions of temperature and salinity, wind stress forcing fields, sea surface temperature climatology, heat flux, and fresh water flux (evaporation minus precipitation minus run-off) at the surface. See file descriptions in section 3.13.3.

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

```
# ====================
# | Model parameters |
# ====================
#
# Continuous equation parameters
 &PARM01
 tRef=15*20.,
 sRef=15*35.,
 viscAh =3.E5,
 no_slip_sides=.TRUE.,
 viscAr =1.721611620915750e5,
#viscAz =1.67E-3,
 no_slip_bottom=.FALSE.,
 diffKhT=1.E3,
 diffKrT=5.154525811125000e3,
#diffKzT=0.5E-4,
 diffKhS=1.E3,
 diffKrS=5.154525811125000e3,
#diffKzS=0.5E-4,
 implicitDiffusion=.TRUE.,
 ivdc_kappa=1.030905162225000e9,
#ivdc_kappa=10.0,
 gravity=9.81,
 gravitySign=-1.D0,
 rhonil=1035.,
 buoyancyRelation='OCEANICP',
 eosType='JMD95P',
 rigidLid=.FALSE.,
 implicitFreeSurface=.TRUE.,
 exactConserv=.TRUE.,
 nonlinFreeSurf=3,
 hFacInf=0.2,
 hFacSup=2.0,
 useRealFreshWaterFlux=.FALSE.,
 readBinaryPrec=64,
 writeBinaryPrec=64,
 cosPower=0.5,
 &

# Elliptic solver parameters
 &PARM02
 cg2dMaxIters=200,
```

```
 cg2dTargetResidual=1.E-9,
 &

# Time stepping parameters
 &PARM03
 startTime =           0.,
 endTime   =     8640000.,
#endTime   = 62208000000.,
 deltaTmom      =   1200.0,
 deltaTtracer   = 172800.0,
 deltaTfreesurf = 172800.0,
 deltaTClock    = 172800.0,
 pChkptFreq  = 3110400000.,
 dumpFreq    = 3110400000.,
 taveFreq    = 3110400000.,
 monitorFreq =   31104000.,
 periodicExternalForcing=.TRUE.,
 externForcingPeriod=2592000.,
 externForcingCycle=31104000.,
 tauThetaClimRelax=5184000.0,
 abEps=0.1,
 &

# Gridding parameters
 &PARM04
 usingCartesianGrid=.FALSE.,
 usingSphericalPolarGrid=.TRUE.,
 dXspacing=4.,
 dYspacing=4.,
 Ro_SeaLevel=53023122.566084,
 groundAtK1=.TRUE.,
 delR=7103300.720021, 6570548.440790, 6041670.010249,
      5516436.666057, 4994602.034410, 4475903.435290,
      3960063.245801, 3446790.312651, 2935781.405664,
      2426722.705046, 1919291.315988, 1413156.804970,
      1008846.750166,  705919.025481,  504089.693499,
 ygOrigin=-80.,
 &

# Input datasets
 &PARM05
 topoFile       ='topog.bin',
 pLoadFile      ='deltageopotjmd95.bin',
 hydrogThetaFile='lev_t.bin',
 hydrogSaltFile ='lev_s.bin',
 zonalWindFile  ='trenberth_taux.bin',
 meridWindFile  ='trenberth_tauy.bin',
 thetaClimFile  ='lev_sst.bin',
#saltClimFile   ='lev_sss.bin',
 surfQFile      ='shi_qnet.bin',
 EmPmRFile      ='shi_empmr.bin',
 &
```

3.13.3.3 File *input/data.pkg*

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

3.13.3.4 File *input/eedata*

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

3.13.3.5 File *input/topog.bin*

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 (Pa for this experiment). In this experiment, a depth of 0 Pa indicates a land point wall and a depth of > 0 Pa indicates open ocean.

3.13.3.6 File *input/deltageopotjmd95.box*

The file contains 12 identical two dimensional maps (x, y) of geopotential height anomaly at the bottom at rest. The values have been obtained by vertically integrating the hydrostatic equation with the initial density field (from *input/lev_t/s.bin*). This file has to be consitent with the temperature and salinity field at rest and the choice of equation of state!

3.13.3.7 File *input/lev_t/s.bin*

The files *input/lev_t/s.bin* specify the initial conditions for temperature and salinity for every grid point in a three dimensional array (x, y, z). The data are obtain by interpolating monthly mean values [*Levitus and T.P.Boyer*, 1994b] for January onto the model grid. Keep in mind, that the first index corresponds to the bottom layer and highest index to the surface layer.

3.13.3.8 File *input/trenberth_taux/y.bin*

Each of the *input/trenberth_taux/y.bin* files specifies 12 two-dimensional (x, y, t) maps of zonal and meridional wind stress values, τ_x and τ_y, that is monthly mean values from *Trenberth et al.* [1990]. The units used are Nm^{-2}.

3.13.3.9 File *input/lev_sst.bin*

The file *input/lev_sst.bin* contains 12 monthly surface temperature climatologies [*Levitus and T.P.Boyer*, 1994b] in a three dimensional array (x, y, t).

3.13.3.10 File *input/shi_qnet/empmr.bin*

The files *input/shi_qnet/empmr.bin* contain 12 monthly surface fluxes of heat (qnet) and freshwater (empmr) by *Jiang et al.* [1999] in three dimensional arrays (x, y, t). Both fluxes are normalized so that of one year there is no net flux into the ocean. The freshwater flux is actually constant in time.

3.13.3.11 File *code/SIZE.h*

Three lines are customized in this file for the current experiment

- Line 39,

  ```
  sNx=90,
  ```

  this line sets the lateral domain extent in grid points for the axis aligned with the x-coordinate.

- Line 40,

  ```
  sNy=40,
  ```

  this line sets the lateral domain extent in grid points for the axis aligned with the y-coordinate.

- Line 49,

  ```
  Nr=15,
  ```

  this line sets the vertical domain extent in grid points.

```
C $Name:  $
C
C     /==========================================================\
C     | SIZE.h Declare size of underlying computational grid.    |
C     |==========================================================|
C     | The design here support a three-dimensional model grid   |
C     | with indices I,J and K. The three-dimensional domain     |
C     | is comprised of nPx*nSx blocks of size sNx along one axis|
C     | nPy*nSy blocks of size sNy along another axis and one    |
C     | block of size Nz along the final axis.                   |
C     | Blocks have overlap regions of size OLx and OLy along the|
C     | dimensions that are subdivided.                          |
C     \==========================================================/
C     Voodoo numbers controlling data layout.
C     sNx - No. X points in sub-grid.
C     sNy - No. Y points in sub-grid.
C     OLx - Overlap extent in X.
C     OLy - Overlat extent in Y.
C     nSx - No. sub-grids in X.
C     nSy - No. sub-grids in Y.
C     nPx - No. of processes to use in X.
C     nPy - No. of processes to use in Y.
C     Nx  - No. points in X for the total domain.
C     Ny  - No. points in Y for the total domain.
C     Nr  - No. points in Z for full process domain.
      INTEGER sNx
      INTEGER sNy
      INTEGER OLx
      INTEGER OLy
      INTEGER nSx
      INTEGER nSy
      INTEGER nPx
      INTEGER nPy
      INTEGER Nx
      INTEGER Ny
      INTEGER Nr
      PARAMETER (
     &           sNx =  90,
     &           sNy =  40,
     &           OLx =   3,
     &           OLy =   3,
     &           nSx =   1,
     &           nSy =   1,
     &           nPx =   1,
     &           nPy =   1,
     &           Nx  = sNx*nSx*nPx,
     &           Ny  = sNy*nSy*nPy,
     &           Nr  =  15)

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

3.13.3.12 File *code/CPP_OPTIONS.h*

This file uses mostly standard default values except for:

- `#define ATMOSPHERIC_LOADING`

enable pressure loading which is abused to include the initial geopotential height anomaly

- `#define EXACT_CONSERV`
  enable more accurate conservation properties to include the horizontal mass divergence in the free surface

- `#define NONLIN_FRSURF`
  enable the nonlinear free surface

3.13.3.13 File *code/CPP_EEOPTIONS.h*

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

3.14 Held-Suarez atmospheric simulation on cube-sphere grid with 32 square cube faces.

(in directory: *verification/tutorial_held_suarez_cs/*)

This example illustrates the use of the MITgcm as an Atmospheric GCM, using simple *Held and Suarez* [1994] forcing to simulate Atmospheric Dynamics on global scale. The set-up use the rescaled pressure coordinate (p^*)[*Adcroft and Campin*, 2004] in the vertical direction, with 20 equaly-spaced levels, and the conformal cube-sphere grid (C32) [*Adcroft et al.*, 2004a]. The files for this experiment can be found in the verification directory under tutorial_held_suarez_cs.

3.14.1 Overview

This example demonstrates using the MITgcm to simulate the planetary atmospheric circulation, with flat orography and simplified forcing. In particular, only dry air processes are considered and radiation effects are represented by a simple newtownien cooling, Thus this example does not rely on any particular atmospheric physics package. This kind of simplified atmospheric simulation has been widely used in GFD-type experiments and in intercomparison projects of AGCM dynamical cores [*Held and Suarez*, 1994].

The horizontal grid is obtain from the projection of a uniform gridded cube to the sphere. Each of the 6 faces has the same resolution, with 32×32 grid points. The equator line coincide with a grid line and crosses, right in the midle, 4 of the 6 faces, leaving 2 faces for the Northern and Southern polar regions. This curvilinear grid requires the use of the 2nd generation exchange topology (*pkg/exch2*) to connect tile and face edges, but without any limitation on the number of processors.

The use of the p^* coordinate with 20 equally spaced levels ($20 \times 50\,$mb, from $p^* = 1000,$mb to 0 at the top of the atmosphere) follows the choice of *Held and Suarez* [1994]. Note that without topography, the p^* coordinate and the normalized pressure coordinate (σ_p) coincide exactly. No viscosity and zero diffusion are used here, but a 8^th order *Shapiro* [1970] filter is applied to both momentum and potential temperature, to remove selectively grid scale noise. Apart from the horizontal grid, this experiment is made very similar to the grid-point model case used in *Held and Suarez* [1994] study.

At this resolution, the configuration can be integrated forward for many years on a single processor desktop computer.

3.14.2 Forcing

The model is forced by relaxation to a radiative equilibrium temperature from *Held and Suarez* [1994]. A linear frictional drag (Rayleigh damping) is applied in the lower part of the atmosphere and account from surface friction and momentum dissipation in the boundary layer. Altogether, this yields the following forcing [from *Held and Suarez*, 1994] that is applied to the fluid:

$$\vec{\mathcal{F}}_{\mathbf{v}} = -k_{\mathbf{v}}(p)\vec{\mathbf{v}}_h \tag{3.61}$$

$$\mathcal{F}_\theta = -k_\theta(\varphi,p)[\theta - \theta_{eq}(\varphi,p)] \tag{3.62}$$

where $\vec{\mathcal{F}}_{\mathbf{v}}$, $\mathcal{F}_\theta$, are the forcing terms in the zonal and meridional momentum and in the potential temperature equations respectively. The term $k_{\mathbf{v}}$ in equation (3.61) applies a Rayleigh damping that is active within the planetary boundary layer. It is defined so as to decay as pressure decreases according to

$$\begin{aligned} k_{\mathbf{v}} &= k_f \ \max[0,\ (p^*/P_s^0 - \sigma_b)/(1-\sigma_b)] \\ \sigma_b &= 0.7 \ \text{ and } \ k_f = 1/86400 \text{ s}^{-1} \end{aligned}$$

where p^* is the pressure level of the cell center and P_s^0 is the pressure at the base of the atmospheric column, which is constant and uniform here ($= 10^5$Pa), in the absence of topography.

The Equilibrium temperature θ_{eq} and relaxation time scale k_θ are set to:

$$\theta_{eq}(\varphi,p^*) = \max\{200.K(P_s^0/p^*)^\kappa, \quad 315.K - \Delta T_y\ \sin^2(\varphi) - \Delta\theta_z \cos^2(\varphi)\log(p^*/P_s^0)\} \tag{3.63}$$

$$k_\theta(\varphi,p^*) = k_a + (k_s - k_a)\ \cos^4(\varphi)\ \max[0,(p^*/P_s^0 - \sigma_b)/(1-\sigma_b)] \tag{3.64}$$

with:

$$\Delta T_y = 60.K \quad k_a = 1/(40\cdot 86400)\ \mathrm{s}^{-1}$$
$$\Delta\theta_z = 10.K \quad k_s = 1/(4\cdot 86400)\ \mathrm{s}^{-1}$$

Initial conditions correspond to a resting state with horizontally uniform stratified fluid. The initial temperature profile is simply the horizontally average of the radiative equilibrium temperature.

3.14.3 Set-up description

The model is configured in hydrostatic form, using non-boussinesq p^* coordinate. The vertical resolution is uniform, $\Delta p^* = 50.10^2 Pa$, with 20 levels, from $p^* = 10^5 Pa$ to 0 at the top. The domain is discretised using C32 cube-sphere grid [*Adcroft et al.*, 2004a] that cover the whole sphere with a relatively uniform grid-spacing. The resolution at the equator or along the Greenwitch meridian is similar to the 128×64 equaly spaced longitude-latitude grid, but requires 25% less grid points. Grid spacing and grid-point location are not computed by the model but read from files.

The vector-invariant form of the momentum equation (see section 2.15) is used so that no explicit metrics are necessary.

Applying the vector-invariant discretization to the atmospheric equations 1.59, and adding the forcing term (3.61, 3.62) on the right-hand-side, leads to the set of equations that are solved in this configuration:

$$\frac{\partial \vec{\mathbf{v}}_h}{\partial t} + (f+\zeta)\hat{\mathbf{k}}\times\vec{\mathbf{v}}_h + \nabla_p(\textsc{ke}) + \omega\frac{\partial \vec{\mathbf{v}}_h}{\partial p} + \nabla_p\Phi' = -k_{\mathbf{v}}\vec{\mathbf{v}}_h \tag{3.65}$$

$$\frac{\partial \Phi'}{\partial p} + \frac{\partial \Pi}{\partial p}\theta' = 0 \tag{3.66}$$

$$\nabla_p\cdot\vec{\mathbf{v}}_h + \frac{\partial\omega}{\partial p} = 0 \tag{3.67}$$

$$\frac{\partial\theta}{\partial t} + \nabla_p\cdot(\theta\vec{\mathbf{v}}_h) + \frac{\partial(\theta\omega)}{\partial p} = -k_\theta[\theta-\theta_{eq}] \tag{3.68}$$

where $\vec{\mathbf{v}}_h$ and $\omega = \frac{Dp}{Dt}$ are the horizontal velocity vector and the vertical velocity in pressure coordinate, ζ is the relative vorticity and f the Coriolis parameter, $\hat{\mathbf{k}}$ is the vertical unity vector, KE is the kinetic energy, Φ is the geopotential and Π the Exner function ($\Pi = C_p(p/p_c)^\kappa$ with $p_c = 10^5 Pa$). Variables marked with ' corresponds to anomaly from the resting, uniformly stratified state.

As described in MITgcm Numerical Solution Procedure 2, the continuity equation is integrated vertically, to give a prognostic equation for the surface pressure p_s:

$$\frac{\partial p_s}{\partial t} + \nabla_h\cdot\int_0^{p_s}\vec{\mathbf{v}}_h dp = 0 \tag{3.69}$$

The implicit free surface form of the pressure equation described in *Marshall et al.* [1997b] is employed to solve for p_s; Integrating vertically the hydrostatic balance gives the geopotential Φ' and allow to step forward the momentum equation 3.65. The potential temperature, θ, is stepped forward using the new velocity field (*staggered time-step*, section 2.8).

3.14.3.1 Numerical Stability Criteria

The numerical stability for inertial oscillations *Adcroft* [1995]

$$S_i = f^2 \Delta t^2 \tag{3.70}$$

evaluates to $4. \times 10^{-3}$ at the poles, for $f = 2\Omega \sin(\pi/2) = 1.45 \times 10^{-4}$ s^{-1}, which is well below the $S_i < 1$ upper limit for stability.

The advective CFL *Adcroft* [1995] for a extreme maximum horizontal flow speed of $|\vec{u}| = 90.$m/s and the smallest horizontal grid spacing $\Delta x = 1.1 \times 10^5$m :

$$S_a = \frac{|\vec{u}|\Delta t}{\Delta x} \tag{3.71}$$

evaluates to 0.37, which is close to the stability limit of 0.5.

The stability parameter for internal gravity waves propagating with a maximum speed of $c_g = 100$ m/s *Adcroft* [1995]

$$S_c = \frac{c_g \Delta t}{\Delta x} \tag{3.72}$$

evaluates to 4×10^{-1}. This is close to the linear stability limit of 0.5.

3.14.4 Experiment Configuration

The model configuration for this experiment resides under the directory *verification/tutorial_held_suarez_cs*. The experiment files

- *input/data*
- *input/data.pkg*
- *input/eedata*,
- *input/data.shap*,
- *code/packages.conf*,
- *code/CPP_OPTIONS.h*,
- *code/SIZE.h*,
- *code/DIAGNOSTICS_SIZE.h*,
- *code/external_forcing.F*,

contain the code customizations and parameter settings for these experiments. Below we describe the customizations to these files associated with this experiment.

3.14.4.1 File *input/data*

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

- Lines 7,

  ```
  tRef=295.2, 295.5, 295.9, 296.3, 296.7, 297.1, 297.6, 298.1, 298.7, 299.3,
  ```

  ...

  set reference values for potential temperature (in Kelvin units) at each model level. The entries are ordered like model level, from surface up to the top. Density is calculated from anomalies at each level evaluated with respect to the reference values set here.

S/R INI_THETA(*ini_theta.F*)

- Line 10,

```
no_slip_sides=.FALSE.,
```

this line selects a free-slip lateral boundary condition for the horizontal Laplacian friction operator e.g. $\frac{\partial u}{\partial y}$=0 along boundaries in y and $\frac{\partial v}{\partial x}$=0 along boundaries in x.

- Lines 11,

```
no_slip_bottom=.FALSE.,
```

this line selects a free-slip boundary condition at the top, in the vertical Laplacian friction operator e.g. $\frac{\partial u}{\partial p} = \frac{\partial v}{\partial p} = 0$

- Line 12,

```
buoyancyRelation='ATMOSPHERIC',
```

this line sets the type of fluid and the type of vertical coordinate to use, which, in this case, is air with a pressure like coordinate (p or p^*).

- Line 13,

```
eosType='IDEALG',
```

Selects the Ideal gas equation of state.

- Line 15,

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

Selects the way the barotropic equation is solved, using here the implicit free-surface formulation.

S/R SOLVE_FOR_PRESSURE (*solve_for_pressure.F*)

- Line 16,

```
exactConserv=.TRUE.,
```

Explicitly calculate again the surface pressure changes from the divergence of the vertically integrated horizontal flow, after the implicit free surface solver and filters are applied.

S/R INTEGR_CONTINUITY (*integr_continuity.F*)

- Line 17 and Line 18,

```
nonlinFreeSurf=4,
select_rStar=2,
```

Select the Non-Linear free surface formulation, using r^* vertical coordinate (here p^*). Note that, except for the default (= 0), other values of those 2 parameters are only permitted for testing/debuging purpose.

S/R CALC_R_STAR (*calc_r_star.F*)
S/R UPDATE_R_STAR (*update_r_star.F*)

- Line 21

```
uniformLin_PhiSurf=.FALSE.,
```

Select the linear relation between surface geopotential anomaly and surface pressure anomaly to be evaluated from $\frac{\partial \Phi_s}{\partial p_s} = 1/\rho(\theta_{Ref})$ (see section 2.10.2.1). Note that using the default (=TRUE), the constant $1/\rho_0$ is used instead, and is not necessary consistent with other parts of the geopotential that relies on θ_{Ref}.

S/R INI_LINEAR_PHISURF (*ini_linear_phisurf.F*)

- Line 23 and Line 24

```
saltStepping=.FALSE.,
momViscosity=.FALSE.,
```

Do not step forward Water vapour and do not compute viscous terms. This allow to save some computer time.

- Line 25

```
vectorInvariantMomentum=.TRUE.,
```

Select the vector-invariant form to solve the momentum equation.

S/R MOM_VECINV (*mom_vecinv.F*)

- Line 26

```
staggerTimeStep=.TRUE.,
```

Select the staggered time-stepping (rather than syncronous time stepping).

- Line 27 and 28

```
readBinaryPrec=64,
writeBinaryPrec=64,
```

Sets format for reading binary input datasets and writing output fields to use 64-bit representation for floating-point numbers.

S/R READ_WRITE_FLD (*read_write_fld.F*) *S/R READ_WRITE_REC* (*read_write_rec.F*)

- Line 33,

```
cg2dMaxIters=200,
```

Sets maximum number of iterations the two-dimensional, conjugate gradient solver will use, **irrespective of convergence criteria being met**.

S/R CG2D (*cg2d.F*)

- Line 35,

```
cg2dTargetResWunit=1.E-17,
```

Sets the tolerance (in units of ω) which the two-dimensional, conjugate gradient solver will use to test for convergence in equation 2.20 to $1 \times 10^{-17} Pa/s$. Solver will iterate until tolerance falls below this value or until the maximum number of solver iterations is reached.

S/R CG2D (*cg2d.F*)

- Line 40,

```
deltaT=450.,
```

Sets the timestep Δt used in the model to 450 s (= 1/8h).

S/R TIMESTEP(*timestep.F*) *S/R TIMESTEP_TRACER*(*timestep_tracer.F*)

- Line 42,

```
startTime=124416000.,
```

Sets the starting time, in seconds, for the model time counter. A non-zero starting time requires to read the initial state from a pickup file. By default the pickup file is named according to the integer number (*nIter0*) of time steps in the *startTime* value ($nIter0 = startTime/deltaT$).

- Line 44,

```
#nTimeSteps=69120,
```

A commented out setting for the length of the simulation (in number of time-step) that corresponds to 1 year simulation.

- Line 54 and 55,

```
nTimeSteps=16,
monitorFreq=1.,
```

Sets the length of the simulation (in number of time-step) and the frequency (in seconds) for "monitor" output. to 16 iterations and 1 seconds respectively. This choice corresponds to a short simulation test.

- Line 48,

```
pChkptFreq=31104000.,
```

Sets the time interval, in seconds, bewteen 2 consecutive "permanent" pickups ("permanent checkpoint frequency") that are used to restart the simuilation, to 1 year.

- Line 49,

```
chkptFreq=2592000.,
```

Sets the time interval, in seconds, bewteen 2 consecutive "temporary" pickups ("checkpoint frequency") to 1 month. The "temporary" pickup file name is alternatively "ckptA" and "ckptB" ; thoses pickup (as opposed to the permanent ones) are designed to be over-written by the model as the simulation progresses.

- Line 50,

```
dumpFreq=2592000.,
```

Set the frequencies (in seconds) for the snap-shot output to 1 month.

- Line 51,

```
#monitorFreq=43200.,
```

A commented out line setting the frequency (in seconds) for the "monitor" output to 12.h. This frequency fits better the longer simulation of 1 year.

- Line 60,

```
usingCurvilinearGrid=.TRUE.,
```

Set the horizontal type of grid to Curvilinear-Grid.

- Line 61,

  ```
  horizGridFile='grid_cs32',
  ```

  Set the root for the grid file name to "*grid_cs32*". The grid-file names are derived from the root, adding a suffix with the face number (e.g.: *.face001.bin*, *.face002.bin* $\cdots$)

S/R INI_CURVILINEAR_GRID (*ini_curvilinear_grid.F*)

- Lines 63 and XXX,

  ```
  delR=20*50.E2,
  Ro_SeaLevel=1.E5,
  ```

  Those 2 lines define the vertical discretization, in pressure units. The 1^{rst} one sets the increments in pressure units (Pa), to 20 equally thick levels of 50×10^2Pa each. The 2^{nd} one sets the reference pressure at the sea-level, to 10^5Pa. This define the origin (interface $k = 1$) of the vertical pressure axis, with decreasing pressure as the level index k increases.

S/R INI_VERTICAL_GRID (*ini_vertical_grid.F*)

- Line 68,

  ```
  #topoFile='topo.cs.bin'
  ```

  This commented out line would allow to set the file name of a 2-D orography file, in meters units, to '*topo.cs.bin*'.

S/R INI_DEPTH (*ini_depth.F*)

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

```
 1 # ====================
 2 # | Model parameters |
 3 # ====================
 4 #
 5 # Continuous equation parameters
 6  &PARM01
 7  tRef=295.2, 295.5, 295.9, 296.3, 296.7, 297.1, 297.6, 298.1, 298.7, 299.3,
 8       300.0, 300.7, 301.9, 304.1, 308.0, 315.1, 329.5, 362.3, 419.2, 573.8,
 9  sRef=20*0.0,
10  no_slip_sides=.FALSE.,
11  no_slip_bottom=.FALSE.,
12  buoyancyRelation='ATMOSPHERIC',
13  eosType='IDEALG',
14  rotationPeriod=86400.,
15  implicitFreeSurface=.TRUE.,
16  exactConserv=.TRUE.,
17  nonlinFreeSurf=4,
18  select_rStar=2,
19  hFacInf=0.2,
20  hFacSup=2.0,
21  uniformLin_PhiSurf=.FALSE.,
22 #hFacMin=0.2,
23  saltStepping=.FALSE.,
24  momViscosity=.FALSE.,
25  vectorInvariantMomentum=.TRUE.,
26  staggerTimeStep=.TRUE.,
27  readBinaryPrec=64,
```

```
 writeBinaryPrec=64,
 &

# Elliptic solver parameters
 &PARM02
 cg2dMaxIters=200,
#cg2dTargetResidual=1.E-12,
 cg2dTargetResWunit=1.E-17,
 &

# Time stepping parameters
 &PARM03
 deltaT=450.,
#nIter0=276480,
 startTime=124416000.,
#- run for 1 year (192.iterations x 450.s = 1.day, 360*192=69120):
#nTimeSteps=69120,
#forcing_In_AB=.FALSE.,
 tracForcingOutAB=1,
 abEps=0.1,
 pChkptFreq=31104000.,
 chkptFreq=2592000.,
 dumpFreq=2592000.,
#monitorFreq=43200.,
 taveFreq=0.,
#- to run a short test (2.h):
 nTimeSteps=16,
 monitorFreq=1.,
 &

# Gridding parameters
 &PARM04
 usingCurvilinearGrid=.TRUE.,
 horizGridFile='grid_cs32',
 radius_fromHorizGrid=6370.E3,
 delR=20*50.E2,
 &

# Input datasets
 &PARM05
#topoFile='topo.cs.bin',
 &
```

3.14.4.2 File *input/data.pkg*

This file, reproduced completely below, specifies the additional packages that the model uses for the experiment. Note that some packages are used by default (e.g.: *pkg/generic_advdiff*) and some others are selected according to parameter in *input/data* (e.g.: *pkg/mom_vecinv*). The additional packages that are used for this configuration are

- Line 3,

  ```
  useSHAP_FILT=.TRUE.,
  ```

  This line selects the Shapiro filter *Shapiro* [1970] (*pkg/shap_filt*) to be used in this experiment.

- Line 4,

  ```
  useDiagnostics=.TRUE.,
  ```

  This line selects the *pkg/diagnostics* to be used in this experiment.

- Line 5,

```
#useMNC=.TRUE.,
```

This line that would selects the *pkg/mnc* for I/O is commented out.

The entire file *input/data.pkg* is reproduced here below:

```
1 # Packages
2  &PACKAGES
3  useSHAP_FILT=.TRUE.,
4  useDiagnostics=.TRUE.,
5 #useMNC=.TRUE.,
6  &
```

3.14.4.3 File *input/eedata*

This file uses standard default values except line 6:

```
useCubedSphereExchange=.TRUE.,
```

This line selects the cubed-sphere specific exchanges to to connect tiles and faces edges.

3.14.4.4 File *input/data.shap*

This file, reproduced completely below, specifies the parameters that the model uses for the Shapiro filter package (*Shapiro* [1970], section 2.20). The parameters that are significant for this configuration are:

- Line 5,

```
Shap_funct=2,
```

This line selects the shapiro filter function to use, here S2 in this experiment (see section 2.20).

- Lines 6 and 7,

```
nShapT=0,
nShapUV=4,
```

Those lines select the order of the shapiro filter for active tracer (θ and q) and momentum (u, v) respectively. In this case, no filter is applied to active tracers. Regarding the momentum, this sets the integer parameter n to 4, in the equations of section 2.20, which corresponds to a 8th order filter.

- Line 9,

```
nShapUVPhys=4,
```

This line selects the order of the physical space filter (filter function S2g, in section 2.20) that applies to u, v. The difference *nShapUV* - *nShapUVPhys* corresponds to the order of the computational filter (filter function S2c, in section 2.20).

- Lines 12 and 13,

```
#Shap_Trtau=5400.,
#Shap_uvtau=1800.,
```

Those commented lines would have set the time scale of the filter (in seconds), for θ and q and for u and v respectively, to 5400 s (90 min) and 1800 s (30 min) respectively. Without explicitly setting those timescales, the default is used which corresponds to the model time-step.

The entire file *input/data.shap* is reproduced here below:

```
# Shapiro Filter parameters
 &SHAP_PARM01
 shap_filt_uvStar=.FALSE.,
 shap_filt_TrStagg=.TRUE.,
 Shap_funct=2,
 nShapT=0,
 nShapUV=4,
#nShapTrPhys=0,
 nShapUVPhys=4,
#Shap_TrLength=140000.,
#Shap_uvLength=110000.,
#Shap_Trtau=5400.,
#Shap_uvtau=1800.,
#Shap_diagFreq=2592000.,
 &
```

3.14.4.5 File *code/SIZE.h*

Four lines are customized in this file for the current experiment

- Line 39,

  ```
  sNx=32,
  ```

  sets the lateral domain extent in grid points along the x-direction, for 1 face.

- Line 40,

  ```
  sNy=32,
  ```

  sets the lateral domain extent in grid points along the y-direction, for 1 face.

- Line 43,

  ```
  nSx=6,
  ```

  sets the number of tiles in the x-direction, for the model domain decomposition. In this simple case (one processor and 1 tile per face), this number corresponds to the total number of faces.

- Line 49,

  ```
  Nr=20,
  ```

  sets the vertical domain extent in grid points.

3.14.4.6 File *code/packages.conf*

This file, reproduced completely below, specifies the packages that are compiled and made available for this experiment. The additional packages that are used for this configuration are

- Line 4,

  ```
  gfd
  ```

  This line selects the standard set of packages that are used by default.

- Line 5,

  ```
  shap_filt
  ```

This line makes the Shapiro filter package available for this experiment.

- Line 6,

  ```
  exch2
  ```

  This line selects the *exch2* pacakge to be compiled and used in this experiment. Note that, with the present structure of the model, the parameter *useEXCH2* does not exists and therefore, this package is always used when it is compiled.

- Line 7,

  ```
  diagnostics
  ```

  This line selects the *pkg/diagnostics* to be compiled, and makes it available for this experiment.

- Line 8,

  ```
  mnc
  ```

  This line selects the *pkg/mnc* to be compiled, and makes it available for this experiment.

The entire file *code/packages.conf* is reproduced here below:

```
1 # $Header: /u/gcmpack/MITgcm/verification/tutorial_held_suarez_cs/code/packages.conf,v 1.2 2014/08/20 20:
2 # $Name:  $
3
4 gfd
5 shap_filt
6 exch2
7 diagnostics
8 mnc
```

3.14.4.7 File *code/CPP_OPTIONS.h*

This file uses standard default except for Line 40
(*diff CPP_OPTIONS.h ../../../model/inc*):

```
#define NONLIN_FRSURF
```

This line allow to use the non-linear free-surface part of the code, which is required for the p^* coordinate formulation.

3.14.4.8 Other Files

Other files relevant to this experiment are

- *code/external_forcing.F*
- *input/grid_cs32.face00[n].bin*, with $n = 1, 2, 3, 4, 5, 6$

contain the code customisations and binary input files for this experiments. Below we describe the customisations to these files associated with this experiment.

The file *code/external_forcing.F* contains 4 subroutines that calculate the forcing terms (Right-Hand side term) in the momentum equation (3.61, *S/R EXTERNAL_FORCING_U* and *EXTERNAL_FORCING_V*) and in the potential temperature equation (3.62, *S/R EXTERNAL_FORCING_T*). The water-vapour forcing subroutine (*S/R EXTERNAL_FORCING_S*) is left empty for this experiment.

The grid-files *input/grid_cs32.face00[n].bin*, with $n = 1, 2, 3, 4, 5, 6$, are binary files (direct-access, big-endian 64.bits real) that contains all the cubed-sphere grid lengths, areas and grid-point positions, with one file per face. Each file contains 18 2-D arrays (dimension 33×33) that correspond to the model variables: *XC YC DXF DYF RA XG YG DXV DYU RAZ DXC DYC RAW RAS DXG DYG AngleCS AngleSN* (see *GRID.h* file)

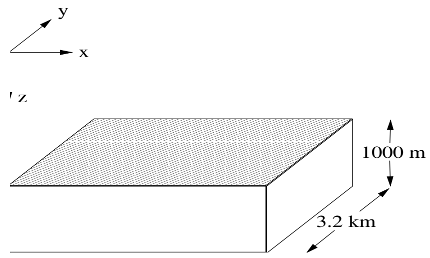


Figure 3.12: Schematic of simulation domain for the surface driven convection experiment. The domain is doubly periodic with an initially uniform temperature of 20 ^oC.

3.15 Surface Driven Convection

(in directory: *verification/tutorial_deep_convection/*)

This experiment, figure 3.12, showcasing MITgcm's non-hydrostatic capability, was designed to explore the temporal and spatial characteristics of convection plumes as they might exist during a period of oceanic deep convection. The files for this experiment can be found in the verification directory under tutorial_deep_convection. It is

- non-hydrostatic
- doubly-periodic with cubic geometry
- has 50 m resolution
- Cartesian
- is on an f-plane
- with a linear equation of state

3.15.1 Overview

The model domain consists of an approximately 3 km square by 1 km deep box of initially unstratified, resting fluid. The domain is doubly periodic.

The experiment has 20 levels in the vertical, each of equal thickness $\Delta z = 50$ m (the horizontal resolution is also 50 m). The fluid is initially unstratified with a uniform reference potential temperature $\theta = 20$ ^{o}C. The equation of state used in this experiment is linear

$$\rho = \rho_0(1 - \alpha_\theta \theta^{'}) \tag{3.73}$$

which is implemented in the model as a density anomaly equation

$$\rho^{'} = -\rho_0 \alpha_\theta \theta^{'} \tag{3.74}$$

with $\rho_0 = 1000\,\mathrm{kg\,m^{-3}}$ and $\alpha_\theta = 2 \times 10^{-4}\,\mathrm{degrees}^{-1}$. Integrated forward in this configuration the model state variable **theta** is equivalent to either in-situ temperature, T, or potential temperature, θ. For consistency with other examples, in which the equation of state is non-linear, we use θ to represent temperature here. This is the quantity that is carried in the model core equations.

As the fluid in the surface layer is cooled (at a mean rate of 800 Wm^2), it becomes convectively unstable and overturns, at first close to the grid-scale, but, as the flow matures, on larger scales (figures 3.13 and 3.14), under the influence of rotation ($f_o = 10^{-4}$ s^{-1}) .

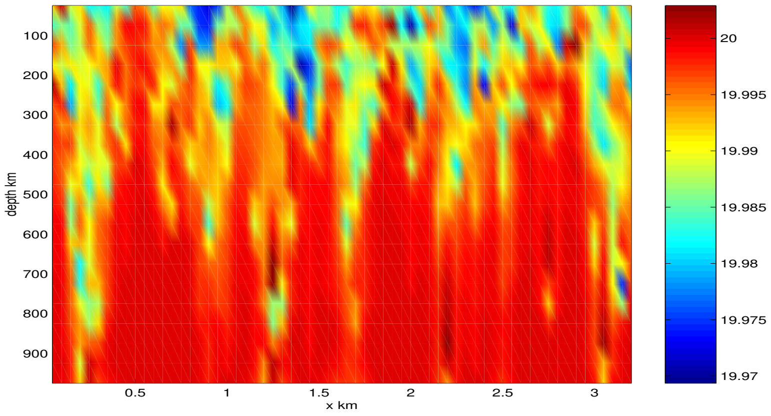


Figure 3.13:

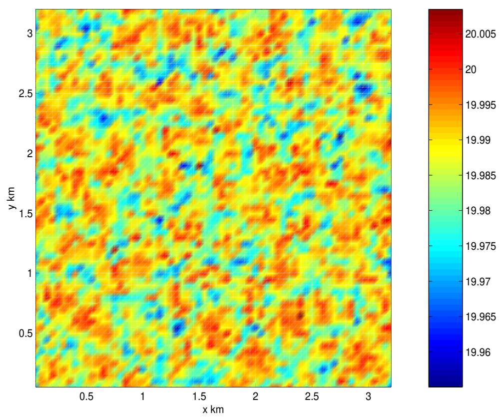


Figure 3.14:

Model parameters are specified in file *input/data*. The grid dimensions are prescribed in *code/SIZE.h*. The forcing (file *input/Qsurf.bin*) is specified in a binary data file generated using the Matlab script *input/gendata.m*.

3.15.2 Equations solved

The model is configured in nonhydrostatic form, that is, all terms in the Navier Stokes equations are retained and the pressure field is found, subject to appropriate bounday condintions, through inversion of a three-dimensional elliptic equation.

The implicit free surface form of the pressure equation described in Marshall et. al *Marshall et al.* [1997b] is employed. A horizontal Laplacian operator ∇_h^2 provides viscous dissipation. The thermodynamic forcing appears as a sink in the potential temperature, θ, equation (3.79). This produces a set of equations solved in this configuration as follows:

$$\frac{Du}{Dt} - fv + \frac{1}{\rho}\frac{\partial p^{'}}{\partial x} - \nabla_h \cdot A_h \nabla_h u - \frac{\partial}{\partial z}A_z\frac{\partial u}{\partial z} = \begin{cases} 0 & \text{(surface)} \\ 0 & \text{(interior)} \end{cases} \tag{3.75}$$

$$\frac{Dv}{Dt} + fu + \frac{1}{\rho}\frac{\partial p^{'}}{\partial y} - \nabla_h \cdot A_h \nabla_h v - \frac{\partial}{\partial z}A_z\frac{\partial v}{\partial z} = \begin{cases} 0 & \text{(surface)} \\ 0 & \text{(interior)} \end{cases} \tag{3.76}$$

$$\frac{Dw}{Dt} + g\frac{\rho^{'}}{\rho} + \frac{1}{\rho}\frac{\partial p^{'}}{\partial z} - \nabla_h \cdot A_h \nabla_h w - \frac{\partial}{\partial z}A_z\frac{\partial w}{\partial z} = \begin{cases} 0 & \text{(surface)} \\ 0 & \text{(interior)} \end{cases} \tag{3.77}$$

$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} + \frac{\partial w}{\partial z} + = 0 \tag{3.78}$$

$$\frac{D\theta}{Dt} - \nabla_h \cdot K_h \nabla_h \theta - \frac{\partial}{\partial z}K_z\frac{\partial \theta}{\partial z} = \begin{cases} \mathcal{F}_\theta & \text{(surface)} \\ 0 & \text{(interior)} \end{cases} \tag{3.79}$$

where $u = \frac{Dx}{Dt}$, $v = \frac{Dy}{Dt}$ and $w = \frac{Dz}{Dt}$ are the components of the flow vector in directions x, y and z. The pressure is diagnosed through inversion (subject to appropriate boundary conditions) of a 3-D elliptic equation derived from the divergence of the momentum equations and continuity (see section 1.3.6).

3.15.3 Discrete numerical configuration

The domain is discretised with a uniform grid spacing in each direction. There are 64 grid cells in directions x and y and 20 vertical levels thus the domain comprises a total of just over 80 000 gridpoints.

3.15.4 Numerical stability criteria and other considerations

For a heat flux of 800 Wm^2 and a rotation rate of 10^{-4} s^{-1} the plume-scale can be expected to be a few hundred meters guiding our choice of grid resolution. This in turn restricts the timestep we can take. It is also desirable to minimise the level of diffusion and viscosity we apply.

For this class of problem it is generally the advective time-scale which restricts the timestep.

For an extreme maximum flow speed of $|\vec{u}| = 1ms^{-1}$, at a resolution of 50 m, the implied maximum timestep for stability, δt_u is

(3.80)

The choice of $\delta t = 10$ s is a safe 20 percent of this maximum.

Interpreted in terms of a mixing-length hypothesis, a magnitude of Laplacian diffusion coefficient $\kappa_h (= \kappa_v) = 0.1$ m^{2}s^{-1} is consistent with an eddy velocity of 2 mm s^{-1} correlated over 50 m.

3.15.5 Experiment configuration

The model configuration for this experiment resides under the directory *verification/convection/*. The experiment files

- *code/CPP_EEOPTIONS.h*
- *code/CPP_OPTIONS.h*,
- *code/SIZE.h*.
- *input/data*
- *input/data.pkg*
- *input/eedata*,
- *input/Qsurf.bin*,

contain the code customisations and parameter settings for this experiment. Below we describe these experiment-specific customisations.

3.15.5.1 File *code/CPP_EEOPTIONS.h*

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

3.15.5.2 File *code/CPP_OPTIONS.h*

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

3.15.5.3 File *code/SIZE.h*

Three lines are customized in this file. These prescribe the domain grid dimensions.

- Line 36,

  ```
  sNx=64,
  ```

  this line sets the lateral domain extent in grid points for the axis aligned with the x-coordinate.

- Line 37,

  ```
  sNy=64,
  ```

  this line sets the lateral domain extent in grid points for the axis aligned with the y-coordinate.

- Line 46,

  ```
  Nr=20,
  ```

  this line sets the vertical domain extent in grid points.

```
 1 C     /==========================================================\
 2 C     | SIZE.h Declare size of underlying computational grid.    |
 3 C     |==========================================================|
 4 C     | The design here support a three-dimensional model grid   |
 5 C     | with indices I,J and K. The three-dimensional domain     |
 6 C     | is comprised of nPx*nSx blocks of size sNx along one axis|
 7 C     | nPy*nSy blocks of size sNy along another axis and one    |
 8 C     | block of size Nz along the final axis.                   |
 9 C     | Blocks have overlap regions of size OLx and OLy along the|
10 C     | dimensions that are subdivided.                          |
11  C     \==========================================================/
```

```
C     Voodoo numbers controlling data layout.
C     sNx - No. X points in sub-grid.
C     sNy - No. Y points in sub-grid.
C     OLx - Overlap extent in X.
C     OLy - Overlat extent in Y.
C     nSx - No. sub-grids in X.
C     nSy - No. sub-grids in Y.
C     nPx - No. of processes to use in X.
C     nPy - No. of processes to use in Y.
C     Nx  - No. points in X for the total domain.
C     Ny  - No. points in Y for the total domain.
C     Nr  - No. points in Z for full process domain.
      INTEGER sNx
      INTEGER sNy
      INTEGER OLx
      INTEGER OLy
      INTEGER nSx
      INTEGER nSy
      INTEGER nPx
     INTEGER nPy
     INTEGER Nx
     INTEGER Ny
     INTEGER Nr
     PARAMETER (
    &          sNx =  64,
    &          sNy =  64,
    &          OLx =   3,
    &          OLy =   3,
    &          nSx =   1,
    &          nSy =   1,
    &          nPx =   1,
    &          nPy =   1,
    &          Nx  = sNx*nSx*nPx,
    &          Ny  = sNy*nSy*nPy,
    &          Nr  =  20)

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

3.15.5.4 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 4,

```
4   tRef=20*20.0,
```

  this line sets the initial and reference values of potential temperature at each model level in units of °C. Here the value is arbitrary since, in this case, the flow evolves independently of the absolute magnitude of the reference temperature. For each depth level the initial and reference profiles will be uniform in x and y. The values specified are read into the variable **tRef** in the model code, by procedure *S/R INI_PARMS (ini_parms.F)* . The temperature field is initialised, by procedure *S/R INI_THETA (ini_theta.F)* .

- Line 5,

```
5   sRef=20*35.0,
```

this line sets the initial and reference values of salinity at each model level in units of ppt. In this case salinity is set to an (arbitrary) uniform value of 35.0 ppt. However since, in this example, density is independent of salinity, an appropriatly defined initial salinity could provide a useful passive tracer. For each depth level the initial and reference profiles will be uniform in x and y. The values specified are read into the variable **sRef** in the model code, by procedure *S/R INI_PARMS (ini_parms.F)* . The salinity field is initialised, by procedure *S/R INI_SALT (ini_salt.F)* .

- Line 6,

```
6   viscAh=0.1,
```

this line sets the horizontal laplacian dissipation coefficient to 0.1 $\mathrm{m^2s^{-1}}$. Boundary conditions for this operator are specified later. The variable **viscAh** is read in the routine *S/R INI_PARMS (ini_params.F)* and applied in routines *S/R CALC_MOM_RHS (calc_mom_rhs.F)* and *S/R CALC_GW (calc_gw.F)* .

- Line 7,

```
7   viscAz=0.1,
```

this line sets the vertical laplacian frictional dissipation coefficient to 0.1 $\mathrm{m^2s^{-1}}$. Boundary conditions for this operator are specified later. The variable **viscAz** is read in the routine *S/R INI_PARMS (ini_parms.F)* and is copied into model general vertical coordinate variable **viscAr** . At each time step, the viscous term contribution to the momentum equations is calculated in routine *S/R CALC_DIFFUSIVITY (calc_diffusivity.F)* .

- Line 8,

```
no_slip_sides=.FALSE.
```

this line selects a free-slip lateral boundary condition for the horizontal laplacian friction operator e.g. $\frac{\partial u}{\partial y}$=0 along boundaries in y and $\frac{\partial v}{\partial x}$=0 along boundaries in x. The variable **no_slip_sides** is read in the routine *S/R INI_PARMS (ini_parms.F)* and the boundary condition is evaluated in routine *S/R CALC_MOM_RHS (calc_mom_rhs.F)* .

- Lines 9,

```
no_slip_bottom=.TRUE.
```

this line selects a no-slip boundary condition for the bottom boundary condition in the vertical laplacian friction operator e.g. $u = v = 0$ at $z = -H$, where H is the local depth of the domain. The variable **no_slip_bottom** is read in the routine *S/R INI_PARMS (ini_parms.F)* and is applied in the routine *S/R CALC_MOM_RHS (calc_mom_rhs.F)* .

- Line 11,

```
diffKhT=0.1,
```

this line sets the horizontal diffusion coefficient for temperature to 0.1 $\mathrm{m^2s^{-1}}$. The boundary condition on this operator is $\frac{\partial}{\partial x} = \frac{\partial}{\partial y} = 0$ at all boundaries. The variable **diffKhT** is read in the routine *S/R INI_PARMS (ini_parms.F)* and used in routine *S/R CALC_GT (calc_gt.F)* .

- Line 12,

```
diffKzT=0.1,
```

this line sets the vertical diffusion coefficient for temperature to 0.1 m^2s^{-1}. The boundary condition on this operator is $\frac{\partial}{\partial z} = 0$ on all boundaries. The variable **diffKzT** is read in the routine *S/R INI_PARMS (ini_parms.F)* . It is copied into model general vertical coordinate variable **diffKrT** which is used in routine *S/R CALC_DIFFUSIVITY (calc_diffusivity.F)* .

- Line 13,

```
diffKhS=0.1,
```

this line sets the horizontal diffusion coefficient for salinity to 0.1 m^2s^{-1}. The boundary condition on this operator is $\frac{\partial}{\partial x} = \frac{\partial}{\partial y} = 0$ on all boundaries. The variable **diffKsT** is read in the routine *S/R INI_PARMS (ini_parms.F)* and used in routine *S/R CALC_GS (calc_gs.F)* .

- Line 14,

```
diffKzS=0.1,
```

this line sets the vertical diffusion coefficient for temperature to 0.1 m^2s^{-1}. The boundary condition on this operator is $\frac{\partial}{\partial z} = 0$ on all boundaries. The variable **diffKzS** is read in the routine *S/R INI_PARMS (ini_parms.F)* . It is copied into model general vertical coordinate variable **diffKrS** which is used in routine *S/R CALC_DIFFUSIVITY (calc_diffusivity.F)* .

- Line 15,

```
f0=1E-4,
```

this line sets the Coriolis parameter to 1×10^{-4} s^{-1}. Since $\beta = 0.0$ this value is then adopted throughout the domain.

- Line 16,

```
beta=0.E-11,
```

this line sets the the variation of Coriolis parameter with latitude to 0.

- Line 17,

```
tAlpha=2.E-4,
```

This line sets the thermal expansion coefficient for the fluid to $2 \times 10^{-4}\,{}^o\,C^{-1}$. The variable **tAlpha** is read in the routine *S/R INI_PARMS (ini_parms.F)* . The routine *S/R FIND_RHO (find_rho.F)* makes use of **tAlpha**.

- Line 18,

```
sBeta=0,
```

This line sets the saline expansion coefficient for the fluid to 0 consistent with salt's passive role in this example.

- Line 23-24,

```
rigidLid=.FALSE.,
implicitFreeSurface=.TRUE.,
```

Selects the barotropic pressure equation to be the implicit free surface formulation.

- Line 25,

```
eosType='LINEAR',
```

Selects the linear form of the equation of state.

- Line 26,

  ```
  nonHydrostatic=.TRUE.,
  ```

  Selects for non-hydrostatic code.

- Line 27,

  ```
  readBinaryPrec=64,
  ```

  Sets format for reading binary input datasets holding model fields to use 64-bit representation for floating-point numbers.

- Line 31,

  ```
  cg2dMaxIters=1000,
  ```

  Inactive - the pressure field in a non-hydrostatic simulation is inverted through a 3D elliptic equation.

- Line 32,

  ```
  cg2dTargetResidual=1.E-9,
  ```

  Inactive - the pressure field in a non-hydrostatic simulation is inverted through a 3D elliptic equation.

- Line 33,

  ```
  cg3dMaxIters=40,
  ```

  This line sets the maximum number of iterations the three-dimensional, conjugate gradient solver will use to 40, **irrespective of the convergence criteria being met**. Used in routine *S/R CG3D (cg3d.F)* .

- Line 34,

  ```
  cg3dTargetResidual=1.E-9,
  ```

  Sets the tolerance which the three-dimensional, conjugate gradient solver will use to test for convergence in equation 2.68 to 1×10^{-9}. The solver will iterate until the tolerance falls below this value or until the maximum number of solver iterations is reached. Used in routine *S/R CG3D (cg3d.F)* .

- Line 38,

  ```
  startTime=0,
  ```

  Sets the starting time for the model internal time counter. When set to non-zero this option implicitly requests a checkpoint file be read for initial state. By default the checkpoint file is named according to the integer number of time steps in the **startTime** value. The internal time counter works in seconds.

- Line 39,

  ```
  nTimeSteps=8640.,
  ```

Sets the number of timesteps at which this simulation will terminate (in this case 8640 timesteps or 1 day or $\delta t = 10$ s). At the end of a simulation a checkpoint file is automatically written so that a numerical experiment can consist of multiple stages.

- Line 40,

  ```
  deltaT=10,
  ```

  Sets the timestep δt to 10 s.

- Line 51,

  ```
  dXspacing=50.0,
  ```

  Sets horizontal (x-direction) grid interval to 50 m.

- Line 52,

  ```
  dYspacing=50.0,
  ```

  Sets horizontal (y-direction) grid interval to 50 m.

- Line 53,

  ```
  delZ=20*50.0,
  ```

  Sets vertical grid spacing to 50 m. Must be consistent with *code/SIZE.h*. Here, 20 corresponds to the number of vertical levels.

- Line 57,

  ```
  surfQfile='Qsurf.bin'
  ```

  This line specifies the name of the file from which the surface heat flux is read. This file is a two-dimensional (x, y) map. It is assumed to contain 64-bit binary numbers giving the value of Q (W m^2) to be applied in each surface grid cell, ordered with the x coordinate varying fastest. The points are ordered from low coordinate to high coordinate for both axes. The matlab program *input/gendata.m* shows how to generate the surface heat flux file used in the example. The variable **Qsurf** is read in the routine *S/R INI_PARMS (ini_parms.F)* and applied in *S/R EXTERNAL_FORCING_SURF (external_forcing_surf.F)* where the flux is converted to a temperature tendency.

```
 1 # Model parameters
 2 # Continuous equation parameters
 3  &PARM01
 4  tRef=20*20.0,
 5  sRef=20*35.0,
 6  viscAh=0.1,
 7  viscAz=0.1,
 8  no_slip_sides=.FALSE.,
 9  no_slip_bottom=.FALSE.,
10   viscA4=0.E12,
11   diffKhT=0.1,
12   diffKzT=0.1,
13   diffKhS=0.1,
14   diffKzS=0.1,
15   f0=1E-4,
16   beta=0.E-11,
17   tAlpha=2.0E-4,
18   sBeta =0.,
19   gravity=9.81,
```

Figure 3.15:

```
 rhoConst=1000.0,
 rhoNil=1000.0,
 heatCapacity_Cp=3900.0,
 rigidLid=.FALSE.,
 implicitFreeSurface=.TRUE.,
 eosType='LINEAR',
 nonHydrostatic=.TRUE.,
 readBinaryPrec=64,
 &
# Elliptic solver parameters
 &PARM02
 cg2dMaxIters=1000,
 cg2dTargetResidual=1.E-9,
 cg3dMaxIters=40,
 cg3dTargetResidual=1.E-9,
 &
# Time stepping parameters
 &PARM03
 nIter0=0,
 nTimeSteps=1440,
 deltaT=60,
 abEps=0.1,
 pChkptFreq=0.0,
 chkptFreq=0.0,
 dumpFreq=600,
 monitorFreq=1.E-5,
 &
# Gridding parameters
 &PARM04
 usingCartesianGrid=.TRUE.,
 usingSphericalPolarGrid=.FALSE.,
 dXspacing=50.0,
  dYspacing=50.0,
  delZ=20*50.0,
  &
 # Input datasets
  &PARM05
  surfQfile='Qsurf.bin',
  &
```

3.15.5.5 File *input/data.pkg*

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

3.15.5.6 File *input/eedata*

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

3.15.5.7 File *input/Qsurf.bin*

The file *input/Qsurf.bin* specifies a two-dimensional (x, y) map of heat flux values where $Q = Q_o \times (0.5 +$ random number between 0 and 1).

In the example $Q_o = 800$ W m^{-2} so that values of Q lie in the range 400 to 1200 W m^{-2} with a mean of Q_o. Although the flux models a loss, because it is directed upwards, according to the model's sign convention, Q is positive.

3.15.6 Running the example

3.15.6.1 Code download

In order to run the examples you must first download the code distribution. Instructions for downloading the code can be found in 3.2.

3.15.6.2 Experiment Location

This example experiments is located under the release sub-directory

verification/convection/

3.15.6.3 Running the Experiment

To run the experiment

1. Set the current directory to *input/*

   ```
   % cd input
   ```

2. Verify that current directory is now correct

   ```
   % pwd
   ```

   You should see a response on the screen ending in
   verification/convection/input

3. Run the genmake script to create the experiment *Makefile*

   ```
   % ../../../tools/genmake -mods=../code
   ```

4. Create a list of header file dependencies in *Makefile*

   ```
   % make depend
   ```

5. Build the executable file.

   ```
   % make
   ```

6. Run the *mitgcmuv* executable

   ```
   % ./mitgcmuv
   ```

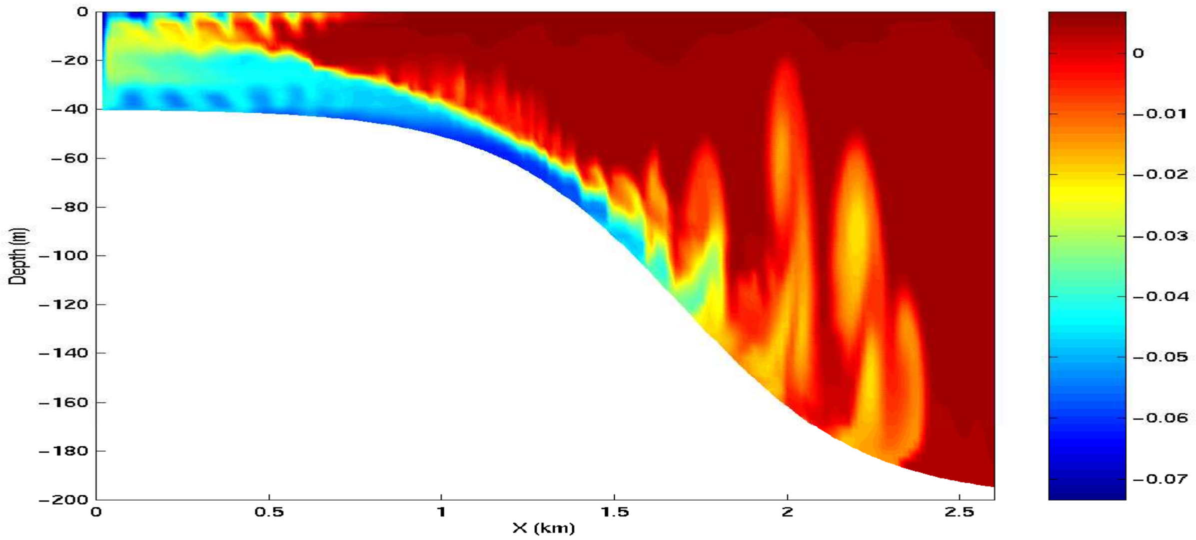


Figure 3.16: Temperature after 23 hours of cooling. The cold dense water is mixed with ambient water as it accelerates down the slope and hence is warmed than the unmixed plume.

3.16 Gravity Plume On a Continental Slope

(in directory: *verification/tutorial_plume_on_slope/*)

An important test of any ocean model is the ability to represent the flow of dense fluid down a slope. One example of such a flow is a non-rotating gravity plume on a continental slope, forced by a limited area of surface cooling above a continental shelf. Because the flow is non-rotating, a two dimensional model can be used in the across slope direction. The experiment is non-hydrostatic and uses open-boundaries to radiate transients at the deep water end. (Dense flow down a slope can also be forced by a dense inflow prescribed on the continental shelf; this configuration is being implemented by the DOME (Dynamics of Overflow Mixing and Entrainment) collaboration to compare solutions in different models). The files for this experiment can be found in the verification directory under tutorial_plume_on_slope.

The fluid is initially unstratified. The surface buoyancy loss B_0 (dimensions of $\mathrm{L}^2\mathrm{T}^{-3}$) over a cross-shelf distance R causes vertical convective mixing and modifies the density of the fluid by an amount

$$\Delta\rho = \frac{B_0 \rho_0 t}{gH} \tag{3.81}$$

where H is the depth of the shelf, g is the acceleration due to gravity, t is time since onset of cooling and ρ_0 is the reference density. Dense fluid slumps under gravity, with a flow speed close to the gravity wave speed:

$$U \sim \sqrt{g'H} \sim \sqrt{\frac{g\Delta\rho H}{\rho_0}} \sim \sqrt{B_0 t} \tag{3.82}$$

A steady state is rapidly established in which the buoyancy flux out of the cooling region is balanced by the surface buoyancy loss. Then

$$U \sim (B_0 R)^{1/3} \; ; \, \Delta\rho \sim \frac{\rho_0}{gH}(B_0 R)^{2/3} \tag{3.83}$$

The Froude number of the flow on the shelf is close to unity (but in practice slightly less than unity, giving subcritical flow). When the flow reaches the slope, it accelerates, so that it may become supercritical (provided the slope angle α is steep enough). In this case, a hydraulic control is established at the shelf break. On the slope, where the Froude number is greater than one, and gradient Richardson number (defined as $Ri \sim g'h^*/U^2$ where h^* is the thickness of the interface between dense and ambient fluid) is reduced below 1/4, Kelvin-Helmholtz instability is possible, and leads to entrainment of ambient fluid

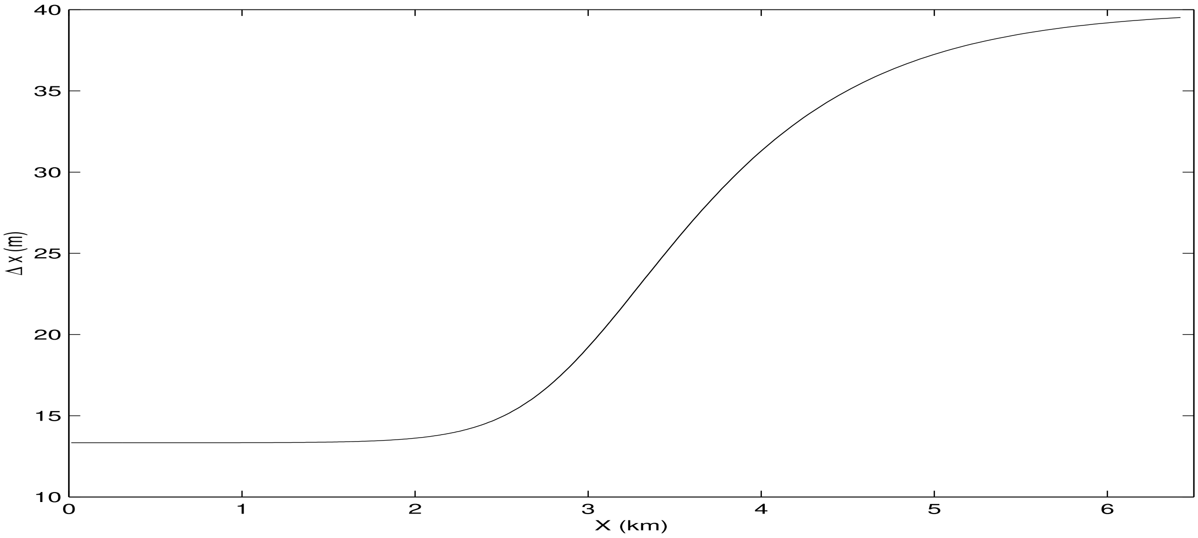


Figure 3.17: Horizontal grid spacing, Δx, in the across-slope direction for the gravity plume experiment.

into the plume, modifying the density, and hence the acceleration down the slope. Kelvin-Helmholtz instability is suppressed at low Reynolds and Peclet numbers given by

$$Re \sim \frac{Uh}{\nu} \sim \frac{(B_0 R)^{1/3} h}{\nu} \; ; \; Pe = RePr \tag{3.84}$$

where h is the depth of the dense fluid on the slope. Hence this experiment is carried out in the high Re, Pe regime. A further constraint is that the convective heat flux must be much greater than the diffusive heat flux (Nusselt number $>> 1$). Then

$$Nu = \frac{Uh^*}{\kappa} >> 1 \tag{3.85}$$

Finally, since we have assumed that the convective mixing on the shelf occurs in a much shorter time than the horizontal equilibration, this implies $H/R << 1$.

Hence to summarize the important nondimensional parameters, and the limits we are considering:

$$\frac{H}{R} << 1 \; ; \; Re >> 1 \; ; \; Pe >> 1 \; ; \; Nu >> 1 \; ; \; \; ; \; Ri < 1/4 \tag{3.86}$$

In addition we are assuming that the slope is steep enough to provide sufficient acceleration to the gravity plume, but nonetheless much less that 1 : 1, since many Kelvin-Helmholtz billows appear on the slope, implying horizontal lengthscale of the slope >> the depth of the dense fluid.

3.16.1 Configuration

The topography, spatial grid, forcing and initial conditions are all specified in binary data files generated using a *Matlab* script called `gendata.m` and detailed in section 3.16.2. Other model parameters are specified in file `data` and `data.obcs` and detailed in section 3.16.4.

3.16.2 Binary input data

The domain is 200 m deep and 6.4 km across. Uniform resolution of $60 \times 3^1/_3$ m is used in the vertical and variable resolution of the form shown in Fig. 3.17 with 320 points is usedin the horizontal. The formula for Δx is:

$$\Delta x(i) = \Delta x_1 + (\Delta x_2 - \Delta x_1)(1 + \tanh\left(\frac{i - i_s}{w}\right))/2$$

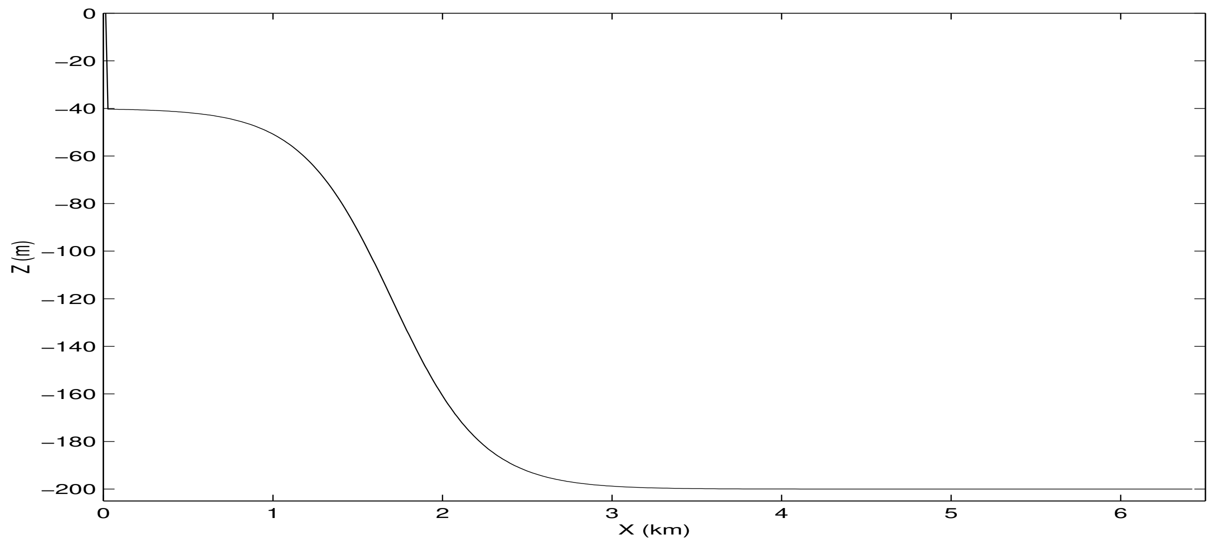


Figure 3.18: Topography, $h(x)$, used for the gravity plume experiment.

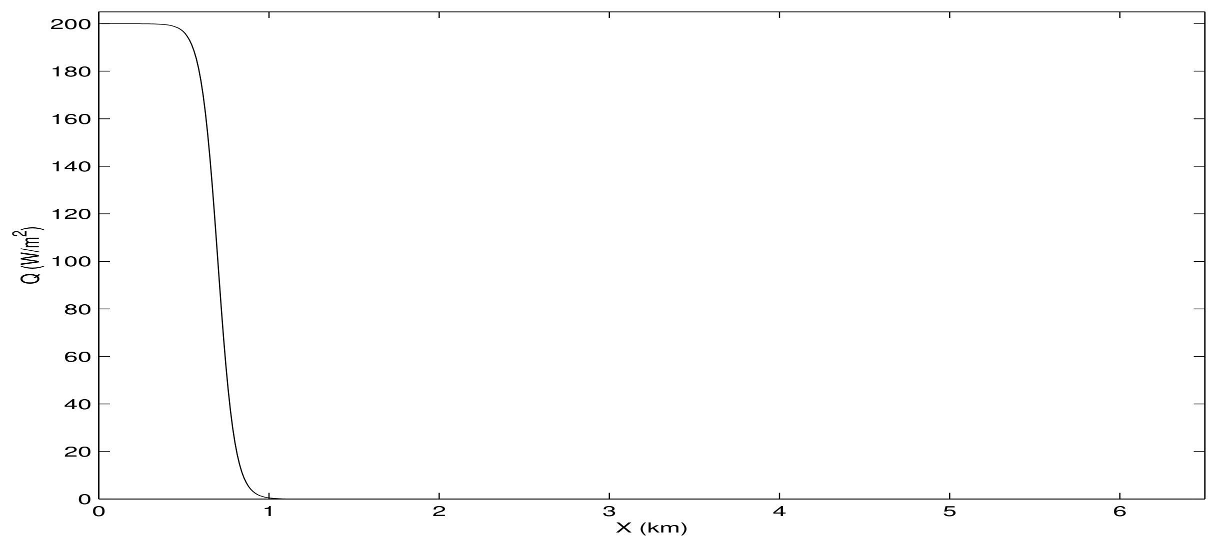


Figure 3.19: Upward surface heat flux, $Q(x)$, used as forcing in the gravity plume experiment.

where

$$\begin{aligned} Nx &= 320 \\ Lx &= 6400 \ \ \text{(m)} \\ \Delta x_1 &= \frac{2}{3}\frac{Lx}{Nx} \ \ \text{(m)} \\ \Delta x_2 &= \frac{Lx/2}{Nx - Lx/(2\Delta x_1)} \ \ \text{(m)} \\ i_s &= Lx/(2\Delta x_1) \\ w &= 40 \end{aligned}$$

Here, Δx_1 is the resolution on the shelf, Δx_2 is the resolution in deep water and Nx is the number of points in the horizontal.

The topography, shown in Fig. 3.18, is given by:

$$H(x) = -H_o + (H_o - h_s)(1 + \tanh\left(\frac{x - x_s}{L_s}\right))/2$$

where

$$\begin{aligned} H_o &= 200 \ \ \text{(m)} \\ h_s &= 40 \ \ \text{(m)} \\ x_s &= 1500 + Lx/2 \ \ \text{(m)} \\ L_s &= \frac{(H_o - h_s)}{2s} \ \ \text{(m)} \\ s &= 0.15 \end{aligned}$$

Here, s is the maximum slope, H_o is the maximum depth, h_s is the shelf depth, x_s is the lateral position of the shelf-break and L_s is the length-scale of the slope.

The forcing is through heat loss over the shelf, shown in Fig. 3.19 and takes the form of a fixed flux with profile:

$$Q(x) = Q_o(1 + \tanh\left(\frac{x - x_q}{L_q}\right))/2$$

where

$$\begin{aligned} Q_o &= 200 \ \ (\text{W m}^{-2}) \\ x_q &= 2500 + Lx/2 \ \ \text{(m)} \\ L_q &= 100 \ \ \text{(m)} \end{aligned}$$

Here, Q_o, is the maximum heat flux, x_q is the position of the cut-off and L_q is the width of the cut-off.

The initial tempeture field is unstratified but with random perturbations, to induce convection early on in the run. The random perturbation are calculated in computational space and because of the variable resolution introduce some spatial correlations but this does not matter for this experiment. The perturbations have range $0 - 0.01$ °K.

3.16.3 Code configuration

The computational domain (number of points) is specified in `code/SIZE.h` and is configured as a single tile of dimensions $320 \times 1 \times 60$. There are no experiment specific source files.

Optional code required to for this experiment are the non-hydrostatic algorithm and open-boundaries:

- Non-hydrostatic terms and algorithm are enabled with **#define ALLOW_NONHYDROSTATIC** in `code/CPP_OPTIONS.h` and activated with **nonHydrostatic=.TRUE.,** in namelist *PARM01* of `input/data`.
- Open boundaries are enabled by adding line **obcs** to package configuration file `code/packages.conf` and activated via **useOBCS=.TRUE,** in namelist *PACKAGES* of `input/data.pkg`.

g	9.81 m s^{-2}	acceleration due to gravity
ρ_o	999.8 kg m^{-3}	reference density
α	2×10^{-4} K^{-1}	expansion coefficient
A_h	1×10^{-2} m^{2}s^{-1}	horizontal viscosity
A_v	1×10^{-3} m^{2}s^{-1}	vertical viscosity
κ_h	0 m^{2}s^{-1}	(explicit) horizontal diffusion
κ_v	0 m^{2}s^{-1}	(explicit) vertical diffusion
Δt	20 s	time step
Δz	$3.3\dot{3}$ m	vertical grid spacing
Δx	$13.\dot{3} - 39.5$ m	horizontal grid spacing

Table 3.2: Model parameters used in the gravity plume experiment.

3.16.4 Model parameters

The model parameters (Table 3.2) are specified in `input/data` and if not assume the default values defined in `model/src/set_defaults.F`. A linear equation of state is used, **eosType='LINEAR'**, but only temperature is active, **sBeta=0.E-4**. For the given heat flux, Q_o, the buoyancy forcing is $B_o = \frac{g\alpha Q}{\rho_o c_p} \sim 10^{-7}$ m^{2}s^{-3}. Using $R = 10^3$ m, the shelf width, then this gives a velocity scale of $U \sim 5 \times 10^{-2}$ m s^{-1} for the initial front but will accelerate by an order of magnitude over the slope. The temperature anomaly will be of order $\Delta\theta \sim 3 \times 10^{-2}$ K. The viscosity is constant and gives a Reynolds number of 100, using $h = 20$ m for the initial front and will be an order magnitude bigger over the slope. There is no explicit diffusion but a non-linear advection scheme is used for temperature which adds enough diffusion so as to keep the model stable. The time-step is set to 20 s and gives Courant number order one when the flow reaches the bottom of the slope.

3.16.5 Build and run the model

Build the model per usual. For example:

```
% cd verification/plume_on_slope
% mkdir build
% cd build
% ../../../tools/genmake -mods=../code -disable=gmredi,kpp,zonal_filt
  ,shap_filt
% make depend
% make
```

When compilation is complete, run the model as usual, for example:

```
% cd ../
% mkdir run
% cp input/* build/mitgcmuv run/
% cd run
% ./mitgcmuv > output.txt
```

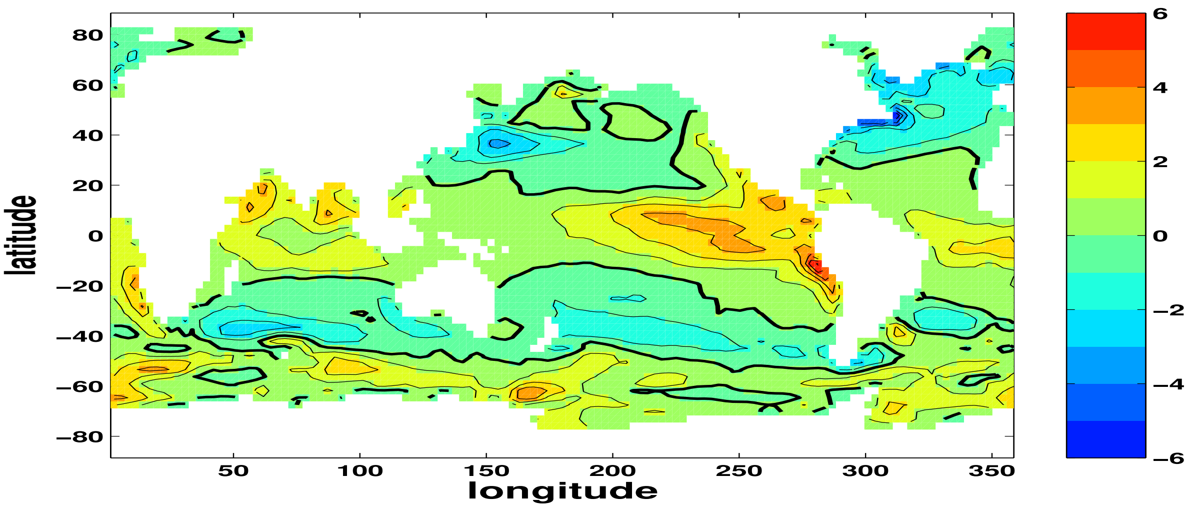


Figure 3.20: Modelled annual mean air-sea CO_2 flux (mol C m^{-2} y^{-1}) for pre-industrial steady-state. Positive indicates flux of CO_2 from ocean to the atmosphere (out-gassing), contour interval is 1 mol C m^{-2} y^{-1}.

3.17 Biogeochemistry Tutorial

(in directory: *verification/tutorial_global_oce_biogeo/*)

3.17.1 Overview

This model overlays the dissolved inorganic carbon biogeochemistry model (the "dic" package) over a 2.8^o global physical model. The physical model has 15 levels, and is forced with a climatological annual cycle of surface wind stresses *Trenberth et al.* [1989], surface heat and freshwater fluxes *Jiang et al.* [1999] with additional relaxation toward climatological sea surface temperature and salinity *Levitus and T.P.Boyer* [1994a,b]. It uses the Gent and and McWilliams *Gent and McWilliams* [1990] eddy parameterization scheme, has an implicit free-surface, implicit vertical diffusion and uses the convective adjustment scheme. The files for this experiment can be found in the verification directory under tutorial_global_oce_biogeo.

The biogeochemical model considers the coupled cycles of carbon, oxygen, phosphorus and alkalinity. A simplified parameterization of biological production is used, limited by the availability of light and phosphate. A fraction of this productivity enters the dissolved organic pool pool, which has an e-folding timescale for remineralization of 6 months [following Yamanaka and Tajika, 1997]. The remaining fraction of this productivity is instantaneously exported as particulate to depth [Yamanaka and Tajika, 1997] where it is remineralized according to the empirical power law relationship determined by Martin et al [1987]. The fate of carbon is linked to that of phosphorus by the Redfield ratio. Carbonate chemistry is explicitly solved *Follows et al.* [ress] and the air-sea exchange of CO_2 is parameterized with a uniform gas transfer coefficient following *Wanninkhof* [1992]. Oxygen is also linked to phosphorus by the Redfield ratio, and oxygen air-sea exchange also follows *Wanninkhof* [1992]. For more details see *Dutkiewicz et al.* [2005].

The example setup described here shows the physical model after 5900 years of spin-up and the biogeochemistry after 2900 years of spin-up. The biogeochemistry is at a pre-industrial steady-state (atmospheric ppmv is kept at 278). Five tracers are resolved: dissolved inorganic carbon (DIC), alkalinity (ALK), phosphate ($PO4$), dissolved organic phosphorus (DOP) and dissolved oxygen ($O2$).

3.17.2 Equations Solved

The physical ocean model velocity and diffusivities are used to redistribute the 5 tracers within the ocean. Additional redistribution comes from chemical and biological sources and sinks. For any tracer A:

$$\frac{\partial A}{\partial t} = -\nabla \cdot (\vec{u^*} A) + \nabla \cdot (\mathbf{K} \nabla A) + S_A$$

where $\vec{u^*}$ is the transformed Eulerian mean circulation (which includes Eulerian and eddy-induced advection), $\mathbf{K}$ is the mixing tensor, and S_A are the sources and sinks due to biological and chemical processes.

The sources and sinks are:

$$\begin{aligned}
S_{DIC} &= F_{CO_2} + V_{CO_2} + r_{C:P} S_{PO_4} + J_{Ca} && (3.87)\\
S_{ALK} &= V_{ALK} - r_{N:P} S_{PO_4} + 2J_{Ca} && (3.88)\\
S_{PO_4} &= -f_{DOP} J_{prod} - \frac{\partial F_P}{\partial z} + \kappa_{remin}[DOP] && (3.89)\\
S_{DOP} &= f_{DOP} J_{prod} - \kappa_{remin}[DOP] && (3.90)\\
S_{O_2} &= \begin{cases} -r_{O:P} S_{PO_4} & \text{if } O_2 > O_{2crit} \\ 0 & \text{if } O_2 < O_{2crit} \end{cases} && (3.91)
\end{aligned}$$

where:

- F_{CO_2} is the flux of CO_2 from the ocean to the atmosphere
- V_{CO_2} is "virtual flux" due to changes in DIC due to the surface freshwater fluxes
- $r_{C:P}$ is Redfield ratio of carbon to phosphorus
- J_{Ca} includes carbon removed from surface due to calcium carbonate formation and subsequent cumulation of the downward flux of $CaCO_3$
- V_{ALK} is "virtual flux" due to changes in alkalinity due to the surface freshwater fluxes
- $r_{N:P}$ Redfield ratio is nitrogen to phosphorus
- f_{DOP} is fraction of productivity that remains suspended in the water column as dissolved organic phosphorus
- J_{prod} is the net community productivity
- $\frac{\partial F_P}{\partial z}$ is the accumulation of remineralized phosphorus with depth
- κ_{remin} is rate with which DOP remineralizes back to PO_4
- F_{O_2} is air-sea flux of oxygen
- $r_{O:P}$ is Redfield ratio of oxygen to phosphorus
- O_{2crit} is a critical level below which oxygen consumption if halted

These terms (for the first four tracers) are described more in *Dutkiewicz et al.* [2005] and by *McKinley et al.* [2004] for the terms relating to oxygen.

3.17.3 Code configuration

The model configuration for this experiment resides in verification/dic_example. The modifications to the code (in *verification/dic_example/code*) are:

- **SIZE.h**: which dictates the size of the model domain (128x64x15).
- **PTRACERS_SIZE.h**: which dictates how many tracers to assign how many tracers will be used (5).

- **GCHEM_OPTIONS.h**: provides some compiler time options for the */pkg/gchem*. In particular this example requires that *DIC_BIOTIC* and *GCHEM_SEPARATE_FORCING* be defined.
- **GMREDI_OPTIONS.h**: assigns the Gent-McWilliam eddy parameterization options.
- **DIAGNOSTICS_SIZE.h**: assigns size information for the diagnostics package.
- **packages.conf**: which dictates which packages will be compiled in this version of the model - among the many that are used for the physical part of the model, this also includes *ptracers*, *gchem*, and *dic* which allow the biogeochemical part of this setup to function.

The input fields needed for this run (in *verification/dic_example/input*) are:

- **data**: specifies the main parameters for the experiment, some parameters that may be useful to know: *nTimeSteps* number timesteps model will run, change to 720 to run for a year *taveFreq* frequency with which time averages are done, change to 31104000 for annual averages.
- **data.diagnostics**: species details of diagnostic pkg output
- **data.gchem**: specifics files and other details needed in the biogeochemistry model run
- **data.gmredi**: species details for the GM parameterization
- **data.mnc**: specifies details for types of output, netcdf or binary
- **data.pkg**: set true or false for various packages to be used
- **data.ptracers**: details of the tracers to be used, including makes, diffusivity information and (if needed) initial files. Of particular importance is the *PTRACERS_numInUse* which states how many tracers are used, and *PTRACERS_Iter0* which states at which timestep the biogeochemistry model tracers were initialized.
- **depth_g77.bin**: bathymetry data file
- **eedata**: This file uses standard default values and does not contain customizations for this experiment.
- **fice.bin**: ice data file, needed for the biogeochemistry
- **lev_monthly_salt.bin**: SSS values which model relaxes toward
- **lev_monthly_temp.bin**: SST values which model relaxes toward
- **pickup.0004248000.data**: variable and tendency values need to restart the physical part of the model
- **pickup_cd.0004248000.data**: variable and tendency values need to restart the cd pkg
- **pickup_ptracers.0004248000.data**: variable and tendency values need to restart the the biogeochemistry part of the model
- **POLY3.COEFFS**: coefficient for the non-linear equation of state
- **shi_empmr_year.bin**: freshwater forcing data file
- **shi_qnet.bin**: heat flux forcing data file
- **sillev1.bin**: silica data file, need for the biogeochemistry
- **tren_speed.bin**: wind speed data file, needed for the biogeochemistry
- **tren_taux.bin**: meridional wind stress data file
- **tren_tauy.bin**: zonal wind stress data file

3.17.4 Running the example

You will first need to download the MITgcm code. Instructions for downloading the code can be found in section 3.2.

1. go to the build directory in verification/dic_example:
   cd verification/dic_example/build

2. create the Makefile:
   ../../../tools/genmake2 -mods=code

3. create all the links:
   make depend

4. compile (the executable will be called mitgcmuv):
   make

5. move the executable to the directory with all the inputs:
   mv mitgcmuv ../input/

6. go to the input directory and run the model:
   cd ../input
   ./mitgcmuv

As the model is set up to run in the verification experiment, it only runs for 4 timestep (2 days) and outputs data at the end of this short run. For a more informative run, you will need to run longer. As set up, this model starts from a pre-spun up state and initializes physical fields and the biogeochemical tracers from the *pickup* files.

Physical data (e.g. S,T, velocities etc) will be output as for any regular ocean run. The biogeochemical output are:

- tracer snap shots: either netcdf, or older-style binary (depending on how *data.mnc* is set up). Look in *data.ptracers* to see which number matches which type of tracer (e.g. ptracer01 is DIC).

- tracer time averages: either netcdf, or older-style binary (depending on how *data.mnc* is set up)

- specific DIC diagnostics: these are averaged over *taveFreq* (set in *data*) and are specific to the dic package, and currently are only available in binary format:
  - **DIC_Biotave**: 3-D biological community productivity (mol P m^{-3} s^{-1})
  - **DIC_Cartave**: 3-D tendencies due to calcium carbonate cycle (mol C m^{-3} s^{-1})
  - **DIC_fluxCO2ave**: 2-D air-sea flux of CO_2 (mol C m^{-2} s^{-1})
  - **DIC_pCO2tave**: 2-D partial pressure of CO_2 in surface layer
  - **DIC_pHtave**: 2-D pH in surface layer
  - **DIC_SurOtave**: 2-D tendency due to air-sea flux of O_2 (mol O m^{-3} s^{-1})
  - **DIC_Surtave**: 2-D surface tendency of DIC due to air-sea flux and virtual flux (mol C m^{-3} s^{-1})

3.18 Global Ocean State Estimation at 4° Resolution

(in directory: *verification/tutorial_global_oce_optim/*)

3.18.1 Overview

This experiment illustrates the optimization capacity of the MITgcm: here, a high level description.

In this tutorial, a very simple case is used to illustrate the optimization capacity of the MITgcm. Using an ocean configuration with realistic geography and bathymetry on a $4 \times 4°$ spherical polar grid, we estimate a time-independent surface heat flux adjustment Q_{netm} that attempts to bring the model climatology into consistency with observations (Levitus dataset, *Levitus and T.P.Boyer* [1994a]). The files for this experiment can be found in the verification directory under tutorial_global_oce_optim.

This adjustment Q_{netm} (a 2D field only function of longitude and latitude) is the control variable of an optimization problem. It is inferred by an iterative procedure using an 'adjoint technique' and a least-squares method (see, for example, *Stammer et al.* [002a] and *Ferreira et al.* [2005]).

The ocean model is run forward in time and the quality of the solution is determined by a cost function, J_1, a measure of the departure of the model climatology from observations:

$$J_1 = \frac{1}{N} \sum_{i=1}^{N} \left[\frac{\overline{T}_i - \overline{T}_i^{lev}}{\sigma_i^T} \right]^2 \tag{3.92}$$

where $\overline{T}_i$ and $\overline{T}_i^{lev}$ are, respectively, the model and observed potential temperature at each grid point i. The differences are weighted by an *a priori* uncertainty σ_i^T on observations (as provided by *Levitus and T.P.Boyer* [1994a]). The error σ_i^T is only a function of depth and varies from 0.5 at the surface to 0.05 K at the bottom of the ocean, mainly reflecting the decreasing temperature variance with depth (Fig. 3.21a). A value of J_1 of order 1 means that the model is, on average, within observational uncertainties.

The cost function also places constraints on the adjustment to insure it is "reasonable", i.e. of order of the uncertainties on the observed surface heat flux:

$$J_2 = \frac{1}{N} \sum_{i=1}^{N} \left[\frac{Q_{\mathrm{netm}}}{\sigma_i^Q} \right]^2 \tag{3.93}$$

where σ_i^Q are the *a priori* errors on the observed heat flux as estimated by Stammer et al. (2002) from 30% of local root-mean-square variability of the NCEP forcing field (Fig 3.21b).

The total cost function is defined as $J = \lambda_1 J_1 + \lambda_2 J_2$ where λ_1 and λ_2 are weights controlling the relative contribution of the two components. The adjoint model then yields the sensitivities $\partial J / \partial Q_{\mathrm{netm}}$ of J relative to the 2D fields Q_{netm}. Using a line-searching algorithm (*Gilbert and Lemaréchal* [1989]), Q_{netm} is adjusted then in the sense to reduce J — the procedure is repeated until convergence.

Fig. 3.22 shows the results of such an optimization. The model is started from rest and from January-mean temperature and salinity initial conditions taken from the Levitus dataset. The experiment is run a year and the averaged temperature over the whole run (i.e. annual mean) is used in the cost function (3.92) to evaluate the model[1]. Only the top 2 levels are used. The first guess Q_{netm} is chosen to be zero. The weights λ_1 and λ_2 are set to 1 and 2, respectively. The total cost function converges after 15 iterations, decreasing from 6.1 to 2.7 (the temperature contribution decreases from 6.1 to 1.8 while the heat flux one increases from 0 to 0.42). The right panels of Fig. (3.22) illustrate the evolution of the temperature error at the surface from iteration 0 to iteration 15. Unsurprisingly, the largest errors at iteration 0 (up to 6°C, top left panels) are found in the Western boundary currents. After optimization, the departure of the model temperature from observations is reduced to 1°C or less almost everywhere except in the Pacific Equatorial Cold Tongue. Comparison of the initial temperature error (top, right) and heat flux adjustment (bottom, left) shows that the system basically increased the heat flux out of the ocean where temperatures were too warm and vice-versa. Obviously, heat flux uncertainties are not the sole responsible for temperature errors and the heat flux adjustment partly compensates the poor representation of narrow currents (Western boundary currents, Equatorial currents) at $4 \times 4°$ resolution.

[1] Because of the daily automatic testing, the experiment as found in the repository is set-up with a very small number of time-steps. To reproduce the results shown here, one needs to set nTimeSteps = 360 and lastinterval=31104000 (both corresponding to a year, see section 3.18.3.2 for further details).

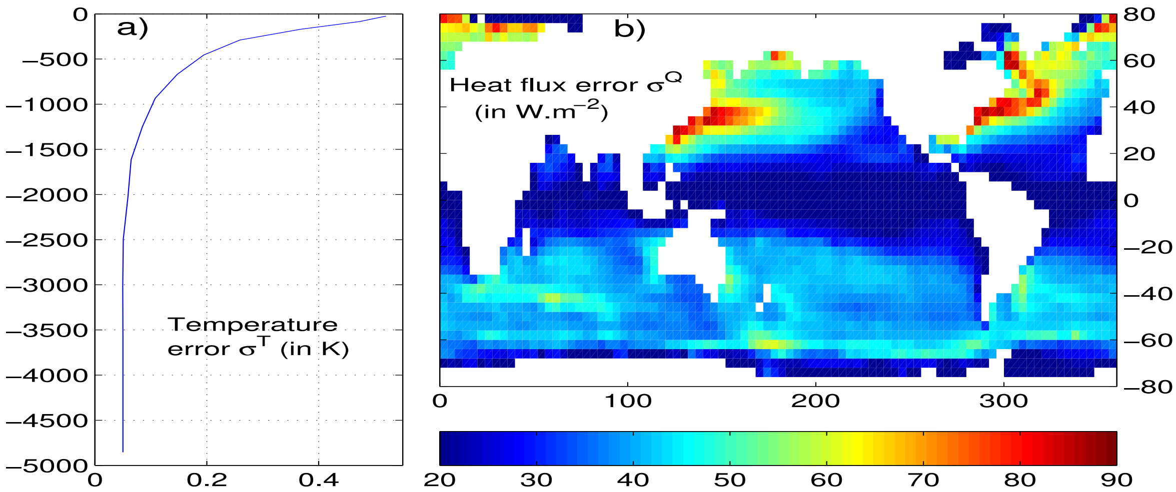


Figure 3.21: *A priori* errors on potential temperature (left, in °C) and surface heat flux (right, in W m^{-2}) used to compute the cost terms J_1 and J_2, respectively.

This is allowed by the large *a priori* error on the heat flux (Fig. 3.21). The Pacific Cold Tongue is a counter example: there, heat fluxes uncertainties are fairly small (about 20 W.m^2), and a large temperature errors remains after optimization.

In the following, section 2 describes in details the implementation of the control variable $Q_{\rm netm}$, the cost function J and the I/O required for the communication between the model and the line-search. Instructions to compile the MITgcm and its adjoint and the line-search algorithm are given in section 3. The method used to run the experiment is described in section 4.

3.18.2 Implementation of the control variable and the cost function

One of the goal of this tutorial is to illustrate how to implement a new control variable. Most of this is fairly generic and is done in the ctrl and cost packages found in the pkg/ directory. The modifications can be tracked by the CPP option ALLOW_HFLUXM_CONTROL or the comment cHFLUXM_CONTROL. The more specific modifications required for the experiment are found in verification/tutorial_global_oce_optim/code_ad. Here follows a brief description of the implementation.

3.18.2.1 The control variable

The adjustment $Q_{\rm netm}$ is activated by setting ALLOW_HFLUXM_CONTROL to "define" in ECCO_OPTIONS.h.

It is first implemented as a "normal" forcing variable. It is defined in FFIELDS.h, initialized to zero in ini_forcing.F, and then used in external_forcing_surf.F. $Q_{\rm netm}$ is made a control variable in the ctrl package by modifying the following subroutines:

- ctrl_init.F where $Q_{\rm netm}$ is defined as the control variable number 24,
- ctrl_pack.F which writes, at the end of each iteration, the sensitivity of the cost function $\partial J/\partial Q_{\rm netm}$ in to a file to be used by the line-search algorithm,
- ctrl_unpack.F which reads, at the start of each iteration, the updated adjustment as provided by the line-search algorithm,
- ctrl_map_forcing.F in which the updated adjustment is added to the first guess $Q_{\rm netm}$.

Note also some minor changes in ctrl.h, ctrl_readparams.F, and ctrl_dummy.h (xx_hfluxm_file, fname_hfluxm, xx_hfluxm_dummy).

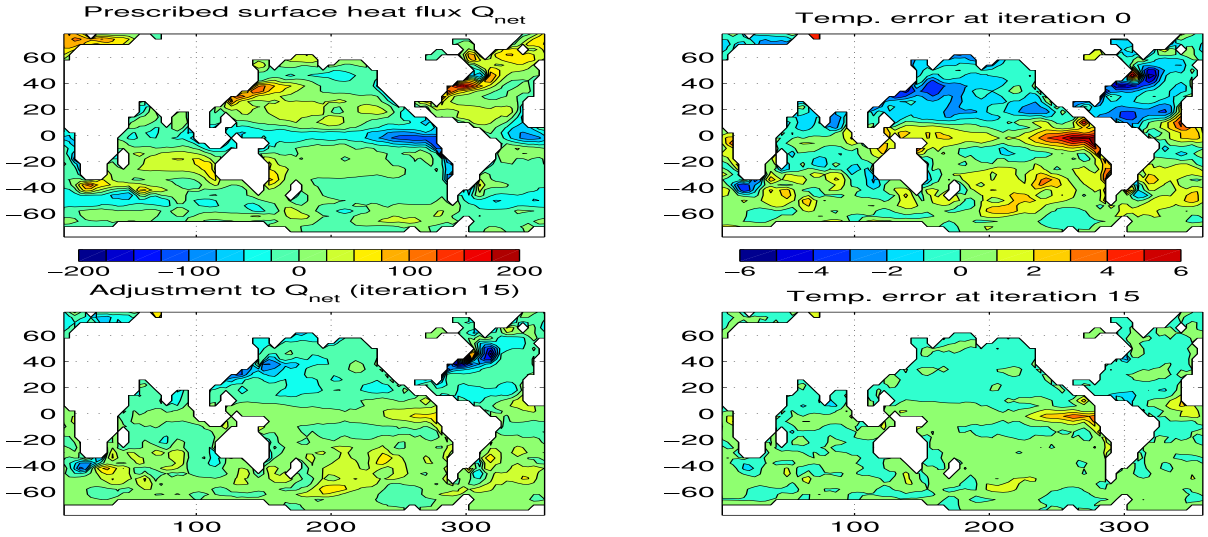


Figure 3.22: Initial annual mean surface heat flux (top right in W.m^{-2}) and adjustment obtained at iteration 15 (bottom right). Averaged difference between model and observed potential temperatures at the surface (in °C) before optimization (iteration 0, top right) and after optimization (iteration 15, bottom right). Contour intervals for heat flux and temperature are 25 W.m^{-2} and 1°C, respectively. A positive flux is out of the ocean.

3.18.2.2 Cost functions

The cost functions are implemented using the *cost* package.

- The temperature cost function J_1 which measures the drift of the mean model temperature from the Levitus climatology is implemented in cost_temp.F. It is activated by ALLOW_COST_TEMP in ECCO_OPTIONS.h. It requires the mean temperature of the model which is obtained by accumulating the temperature in cost_tile.F (called at each time step). The value of the cost function is stored in *objf_temp* and its weight λ_1 in *mult_temp*.
- The heat flux cost function, penalizing the departure of the surface heat flux from observations is implemented in cost_hflux.F, and activated by the key ALLOW_COST_HFLUXM in ECCO_OPTIONS.h. The value of the cost function is stored in *objf_hfluxm* and its weight λ_2 in *mult_hfluxm*.
- The subroutine cost_final.F calls the cost_functions subroutines and make the (weighted) sum of the various contributions.
- The various weights used in the cost functions are read in cost_weights.F. The weight of the cost functions are read in cost_readparams.F from the input file data.cost.

3.18.3 Code Configuration

The model configuration for this experiment resides under the directory *verification/tutorial_global_oce_optim/*. The experiment files in code_ad/ and input_ad/ contain the code customizations and parameter settings. Most of them are identical to those used in the Global Ocean (experiment *tutorial_global_oce_latlon*). Below, we describe some of the customizations required for this experiment.

3.18.3.1 Compilation-time customizations in *code_ad/*

In ECCO_CPPOPTIONS.h:

- define ALLOW_ECCO_OPTIMIZATION
- define ALLOW_COST, ALLOW_COST_TEMP, and ALLOW_COST_HFLUXM

- define ALLOW_HFLUXM_CONTROL

3.18.3.2 Running-time customizations in *input_ad/*

- *data*: note the smaller *cg2dTargetResidual* than in the forward-only experiment,
- *data.optim* specifies the iteration number,
- *data.ctrl* is used, in particular, to specify the name of the sensitivity and adjustment files associated to a control variable,
- *data.cost*: parameters of the cost functions, in particular *lastinterval* specifies the length of time-averaging for the model temperature to be used in the cost function (3.92),
- *data.pkg*: note that the Gradient Check package is turned on by default (useGrdchk=.TRUE.),
- *Err_hflux.bin* and *Err_levitus_15layer.bin* are the files containing the heat flux and potential temperature uncertainties, respectively.

3.18.4 Compiling

The optimization experiment requires two executables: 1) the MITgcm and its adjoint (*mitgcmuv_ad*) and 2) the line-search algorithm (*optim.x*).

3.18.4.1 Compilation of MITgcm and its adjoint: *mitcgmuv_ad*

Before compiling, first note that, in the directory code_ad/, two files must be updated:

- code_ad_diff.list which lists new subroutines to be compiled by the TAF software (cost_temp.F and cost_hflux.F here),
- the adjoint_hfluxm files which provides a list of the control variables and the name of cost function to the TAF software.

Then, in the directory build_ad/, type:

```
% ../../../tools/genmake2 -mods=../code\_ad -adof=../code\_ad/adjoint\_hfluxm
% make depend
% make adall
```

to generate the MITgcm executable *mitgcmuv_ad*.

3.18.4.2 Compilation of the line-search algorithm: *optim.x*

This is done from the directories lsopt/ and optim/ (under MITgcm/). In lsopt/, unzip the blash1 library adapted to your platform, and change the Makefile accordingly. Compile with:

```
% make all
```

(more details in lsopt_doc.txt)

In optim/, the path of the directory where *mitgcm_ad* was compiled must be specified in the Makefile in the variable INCLUDEDIRS. The file name of the control variable (xx_hfluxm_file here) must be added to the name list read by optim_num.F. Then use

```
% make depend
```

and

```
% make
```

to generate the line-search executable *optim.x*.

3.18.5 Running the estimation

Copy the *mitgcmuv_ad* executable to input_ad/ and *optim.x* to the subdirectory input_ad/OPTIM/. Move into input_ad/. The first iteration is somewhat particular and is best done "by hand" while the following iterations can be run automatically (see below). Check that the iteration number is set to 0 in data.optim and run the MITgcm:

```
% ./mitgcmuv_ad
```

The output files adxx_hfluxm.0000000000.* and xx_hfluxm.0000000000.* contain the sensitivity of the cost function to $Q_{\rm netm}$ and the adjustment to $Q_{\rm netm}$ (zero at the first iteration), respectively. Two other files called costhflux_tut_MITgcm.opt0000 and ctrlhflux_tut_MITgcm.opt0000 are also generated. They essentially contain the same information as the adxx_.hfluxm* and xx_hfluxm* files, but in a compressed format. These two files are the only ones involved in the communication between the adjoint model *mitgcmuv_ad* and the line-search algorithm *optim.x*. Only at the first iteration, are they both generated by *mitgcmuv_ad*. Subsenquently, costhflux_tut_MITgcm.opt$n$ is an output of the adjoint model at iteration n and an input of the line-search. The latter returns an updated adjustment in ctrlhflux_tut_MITgcm.opt$n$+1 to be used as an input of the adjoint model at iteration n+1.

At the first iteration, move costhflux_tut_MITgcm.opt0000 and ctrlhflux_tut_MITgcm.opt0000 to OPTIM/, move into this directory and link data.optim and data.ctrl locally:

```
% cd OPTIM/
% ln -s ../data.optim .
% ln -s ../data.ctrl .
```

The target cost function *fmin* needs to be specified too: as a rule of thumb, it should be about 0.95-0.90 times the value of the cost function at the first iteration. This value is only used at the first iteration and does not need to be updated afterward. However, it implicitly specifies the "pace" at which the cost function is going down (if you are lucky and it does indeed diminish!). More in the ECCO section maybe ?

Once this is done, run the line-search algorithm:

```
% ./optim.x
```

which computes the updated adjustment for iteration 1, ctrlhflux_tut_MITgcm.opt0001.

The following iterations can be executed automatically using the shell script *cycsh* found in input_ad/. This script will take care of changing the iteration numbers in the data.optim, launch the adjoint model, clean and store the outputs, move the costhflux* and ctrlhflux* files, and run the line-search algorithm. Edit *cycsh* to specify the prefix of the directories used to store the outputs and the maximum number of iteration.

3.19 Sensitivity of Air-Sea Exchange to Tracer Injection Site

(in directory: *verification/tutorial_tracer_adjsens/*)

MITgcm has been adapted to enable AD using TAMC or TAF. The present description, therefore, is specific to the use of TAMC or TAF as AD tool. The following sections describe the steps which are necessary to generate a tangent linear or adjoint model of MITgcm. We take as an example the sensitivity of carbon sequestration in the ocean. The AD-relevant hooks in the code are sketched in 5.2, 5.4.

3.19.1 Overview of the experiment

We describe an adjoint sensitivity analysis of out-gassing from the ocean into the atmosphere of a carbon-like tracer injected into the ocean interior (see *Hill et al.* [2002]).

3.19.1.1 Passive tracer equation

For this work MITgcm was augmented with a thermodynamically inactive tracer, C. Tracer residing in the ocean model surface layer is out-gassed according to a relaxation time scale, μ. Within the ocean interior, the tracer is passively advected by the ocean model currents. The full equation for the time evolution

$$\frac{\partial C}{\partial t} = -U \cdot \nabla C - \mu C + \Gamma(C) + S \tag{3.94}$$

also includes a source term S. This term represents interior sources of C such as would arise due to direct injection. The velocity term, U, is the sum of the model Eulerian circulation and an eddy-induced velocity, the latter parameterized according to Gent/McWilliams (*Gent and McWilliams* [1990]; *Gent et al.* [1995]). The convection function, Γ, mixes C vertically wherever the fluid is locally statically unstable.

The out-gassing time scale, μ, in eqn. (3.94) is set so that $1/\mu \sim 1$ year for the surface ocean and $\mu = 0$ elsewhere. With this value, eqn. (3.94) is valid as a prognostic equation for small perturbations in oceanic carbon concentrations. This configuration provides a powerful tool for examining the impact of large-scale ocean circulation on CO_2 out-gassing due to interior injections. As source we choose a constant in time injection of $S = 1$ mol/s.

3.19.1.2 Model configuration

The model configuration employed has a constant $4^\circ \times 4^\circ$ resolution horizontal grid and realistic geography and bathymetry. Twenty vertical layers are used with vertical spacing ranging from 50 m near the surface to 815 m at depth. Driven to steady-state by climatological wind-stress, heat and fresh-water forcing the model reproduces well known large-scale features of the ocean general circulation.

3.19.1.3 Out-gassing cost function

To quantify and understand out-gassing due to injections of C in eqn. (3.94), we define a cost function $\mathcal{J}$ that measures the total amount of tracer out-gassed at each timestep:

$$\mathcal{J}(t = T) = \int_{t=0}^{t=T} \int_A \mu C \, dA \, dt \tag{3.95}$$

Equation(3.95) integrates the out-gassing term, μC, from (3.94) over the entire ocean surface area, A, and accumulates it up to time T. Physically, $\mathcal{J}$ can be thought of as representing the amount of CO_2 that our model predicts would be out-gassed following an injection at rate S. The sensitivity of $\mathcal{J}$ to the spatial location of S, $\frac{\partial \mathcal{J}}{\partial S}$, can be used to identify regions from which circulation would cause CO_2 to rapidly out-gas following injection and regions in which CO_2 injections would remain effectively sequestered within the ocean.

3.19.2 Code configuration

The model configuration for this experiment resides under the directory *verification/carbon/*. The code customization routines are in *verification/carbon/code/*:

- *.genmakerc*
- *COST_CPPOPTIONS.h*
- *CPP_EEOPTIONS.h*
- *CPP_OPTIONS.h*
- *CTRL_OPTIONS.h*
- *ECCO_OPTIONS.h*
- *SIZE.h*
- *adcommon.h*
- *tamc.h*

The runtime flag and parameters settings are contained in *verification/carbon/input/*, together with the forcing fields and and restart files:

- *data*
- *data.cost*
- *data.ctrl*
- *data.gmredi*
- *data.grdchk*
- *data.optim*
- *data.pkg*
- *eedata*
- *topog.bin*
- *windx.bin, windy.bin*
- *salt.bin, theta.bin*
- *SSS.bin, SST.bin*
- *pickup**

Finally, the file to generate the adjoint code resides in *adjoint/*:

- *makefile*

Below we describe the customizations of this files which are specific to this experiment.

3.19.2.1 File *.genmakerc*

This file overwrites default settings of *genmake*. In the present example it is used to switch on the following packages which are related to automatic differentiation and are disabled by default:

```
set ENABLE=( autodiff cost ctrl ecco gmredi grdchk kpp )
```

Other packages which are not needed are switched off:

```
set DISABLE=( aim obcs zonal_filt shap_filt cal exf )
```

3.19.2.2 File *COST_CPPOPTIONS.h, CTRL_OPTIONS.h*

These files used to contain package-specific CPP-options (see Section ref:ask-the-author). For technical reasons those options have been grouped together in the file *ECCO_OPTIONS.h*. To retain the modularity, the files have been kept and contain the standard include of the *CPP_OPTIONS.h* file.

3.19.2.3 File *CPP_EEOPTIONS.h*

This file contains 'wrapper'-specific CPP options. It only needs to be changed if the code is to be run in a parallel environment (see Section ref:ask-the-author).

3.19.2.4 File *CPP_OPTIONS.h*

This file contains model-specific CPP options (see Section ref:ask-the-author). Most options are related to the forward model setup. They are identical to the global steady circulation setup of *verification/global_ocean.90x40x15/*. The three options specific to this experiment are

```
#define ALLOW_PASSIVE_TRACER
```

This flag enables the code to carry through the advection/diffusion of a passive tracer along the model integration.

```
#define ALLOW_MIT_ADJOINT_RUN
```

This flag enables the inclusion of some AD-related fields concerning initialization, link between control variables and forward model variables, and the call to the top-level forward/adjoint subroutine *adthe_main_loop* instead of *the_main_loop*.

```
#define ALLOW_GRADIENT_CHECK
```

This flag enables the gradient check package. After computing the unperturbed cost function and its gradient, a series of computations are performed for which
• an element of the control vector is perturbed
• the cost function w.r.t. the perturbed element is computed
• the difference between the perturbed and unperturbed cost function is computed to compute the finite difference gradient
• the finite difference gradient is compared with the adjoint-generated gradient. The gradient check package is further described in Section ???.

3.19.2.5 File *ECCO_OPTIONS.h*

The CPP options of several AD-related packages are grouped in this file:

- Overall ECCO-related execution modus:
  These determine whether a pure forward run, a sensitivity run or an iteration of optimization is performed. These options are not needed in the present context.

- Adjoint support package: *pkg/autodiff/*
  This package contains hand-written adjoint code such as active file handling, flow directives for files which must not be differentiated, and TAMC-specific header files.
  ```
  #define ALLOW_AUTODIFF_TAMC
  ```
  defines TAMC-related features in the code.
  ```
  #define ALLOW_TAMC_CHECKPOINTING
  ```
  enables the checkpointing feature of TAMC (see Section ref:ask-the-author). In the present example a 3-level checkpointing is implemented. The code contains the relevant store directives, common block and tape initializations, storing key computation, and loop index handling. The checkpointing length at each level is defined in file *tamc.h*, cf. below. The out and intermediate loop directivs are contained in the files *checkpoint_lev3_directives.h*, *checkpoint_lev2_directives.h* (package *pkg/autodiff*).
  ```
  #define ALLOW_AUTODIFF_MONITOOR
  ```
  enables the monitoring of intermediate adjoint variables (see Section ref:ask-the-author).
  ```
  #define ALLOW_DIVIDED_ADJOINT
  ```
  enables adjoint dump and restart (see Section ref:ask-the-author).

- Cost function package: *pkg/cost/*
  This package contains all relevant routines for initializing, accumulating and finalizing the cost function (see Section ref:ask-the-author).
  `#define ALLOW_COST`
  enables all general aspects of the cost function handling, in particular the hooks in the forward code for initializing, accumulating and finalizing the cost function.
  `#define ALLOW_COST_TRACER`
  includes the call to the cost function for this particular experiment, eqn. (3.95).

- Control variable package: *pkg/ctrl/*
  This package contains all relevant routines for the handling of the control vector. Each control variable can be enabled/disabled with its own flag:

`#define ALLOW_THETA0_CONTROL`	initial temperature
`#define ALLOW_SALT0_CONTROL`	initial salinity
`#define ALLOW_TR0_CONTROL`	initial passive tracer concentration
`#define ALLOW_TAUU0_CONTROL`	zonal wind stress
`#define ALLOW_TAUV0_CONTROL`	meridional wind stress
`#define ALLOW_SFLUX0_CONTROL`	freshwater flux
`#define ALLOW_HFLUX0_CONTROL`	heat flux
`#define ALLOW_DIFFKR_CONTROL`	diapycnal diffusivity
`#undef ALLOW_KAPPAGM_CONTROL`	isopycnal diffusivity

3.19.2.6 File *SIZE.h*

The file contains the grid point dimensions of the forward model. It is identical to the *verification/exp2/*:

```
sNx = 90
sNy = 40
Nr = 20
```

It corresponds to a single-tile/single-processor setup: `nSx = nSy = 1, nPx = nPy = 1`, with standard overlap dimensioning `OLx = OLy = 3`.

3.19.2.7 File *adcommon.h*

This file contains common blocks of some adjoint variables that are generated by TAMC. The common blocks are used by the adjoint support routine *addummy_in_stepping* which needs to access those variables:

`common /addynvars_r/`	is related to *DYNVARS.h*
`common /addynvars_cd/`	is related to *DYNVARS.h*
`common /addynvars_diffkr/`	is related to *DYNVARS.h*
`common /addynvars_kapgm/`	is related to *DYNVARS.h*
`common /adtr1_r/`	is related to *TR1.h*
`common /adffields/`	is related to *FFIELDS.h*

Note that if the structure of the common block changes in the above header files of the forward code, the structure of the adjoint common blocks will change accordingly. Thus, it has to be made sure that the structure of the adjoint common block in the hand-written file *adcommon.h* complies with the automatically generated adjoint common blocks in *adjoint_model.F*. The header file is enabled via the CPP-option **ALLOW_AUTODIFF_MONITOR**.

3.19.2.8 File *tamc.h*

This routine contains the dimensions for TAMC checkpointing and some indices relevant for storing ky computations.

- `#ifdef ALLOW_TAMC_CHECKPOINTING`
  3-level checkpointing is enabled, i.e. the timestepping is divided into three different levels (see Section ref:ask-the-author). The model state of the outermost (`nchklev_3`) and the intermediate (`nchklev_2`) timestepping loop are stored to file (handled in *the_main_loop*). The innermost loop (`nchklev_1`) avoids I/O by storing all required variables to common blocks. This storing may also be necessary if no checkpointing is chosen (nonlinear functions, if-statements, iterative loops, ...).

In the present example the dimensions are chosen as follows:

```
nchklev_1 = 36
nchklev_2 = 30
nchklev_3 = 60
```

To guarantee that the checkpointing intervals span the entire integration period the following relation must be satisfied:

`nchklev_1*nchklev_2*nchklev_3` $\geq$ `nTimeSteps`

where `nTimeSteps` is either specified in *data* or computed via

`nTimeSteps = (endTime-startTime)/deltaTClock` .

- `#undef ALLOW_TAMC_CHECKPOINTING`
  No checkpointing is enabled. In this case the relevant counter is `nchklev_0`. Similar to above, the following relation has to be satisfied
  `nchklev_0` $\geq$ `nTimeSteps`.

The following parameters may be worth describing:

`isbyte`
`maxpass`

3.19.2.9 File *makefile*

This file contains all relevant parameter flags and lists to run TAMC or TAF. It is assumed that TAMC is available to you, either locally, being installed on your network, or remotely through the 'TAMC Utility'. TAMC is called with the command `tamc` followed by a number of options. They are described in detail in the TAMC manual *Giering* [1999]. Here we briefly discuss the main flags used in the *makefile*. The standard output for TAF is written to file *taf.log*.

```
tamc -input <variable names> -output <variable name> -i4 -r4 ...
     -toplevel <S/R name> -reverse <file names>

 taf -input <variable names> -output <variable name> -i4 -r4 ...
     -toplevel <S/R name> -reverse <file names>
     -flow taf_flow.log -nonew_arg
```

- `-toplevel <S/R name>`
  Name of the toplevel routine, with respect to which the control flow analysis is performed.

- `-input <variable names>`
  List of independent variables u with respect to which the dependent variable J is differentiated.

- `-output <variable name>`
  Dependent variable J which is to be differentiated.

- `-reverse <file names>`
  Adjoint code is generated to compute the sensitivity of an independent variable w.r.t. many dependent variables. In the discussion of Section ??? the generated adjoint top-level routine computes the product of the transposed Jacobian matrix M^T times the gradient vector $\nabla_v J$.
  `<file names>` refers to the list of files *.f* which are to be analyzed by TAMC. This list is generally smaller than the full list of code to be compiled. The files not contained are either above the top-level routine (some initializations), or are deliberately hidden from TAMC, either because hand-written adjoint routines exist, or the routines must not (or don't have to) be differentiated. For each routine which is part of the flow tree of the top-level routine, but deliberately hidden from TAMC (or for each package which contains such routines), a corresponding file *.flow* exists containing flow directives for TAMC.

- `-i4 -r4`

- `-flow taf_flow.log`
  Will cause TAF to produce a flow listing file named *taf_flow.log* in which the set of active and passive variables are identified for each subroutine.

- `-nonew_arg`
  The default in the order of the parameter list of adjoint routines has changed. Before TAF 1.3 the default was compatible with the TAMC-generated list. As of TAF 1.3 the order of adjoint routine parameter lists is no longer copatible with TAMC. To restore compatibility when using TAF 1.3 and higher, this argument is needed. It is currently crucial to use since all hand-written adjoint routines refer to the TAMC default.

3.19.2.10 The input parameter files

File *data*

File *data.cost*

File *data.ctrl*

File *data.gmredi*

File *data.grdchk*

File *data.optim*

File *data.pkg*

File *eedata*

File *topog.bin*
Contains two-dimendional bathymetry information

File *windx.bin, windy.bin, salt.bin, theta.bin, SSS.bin, SST.bin*
These contain the initial values (salinity, temperature, *salt.bin, theta.bin*), surface boundary values (surface wind stresses, (*windx.bin, windy.bin*), and surface restoring fields (*SSS.bin, SST.bin*).

File *pickup**
Contains model state after model spinup.

3.19.3 Compiling the model and its adjoint

The built process of the adjoint model is slightly more complex than that of compiling the forward code. The main reason is that the adjoint code generation requires a specific list of routines that are to be differentiated (as opposed to the automatic generation of a list of files to be compiled by genmake). This list excludes routines that don't have to be or must not be differentiated. For some of the latter routines flow directives may be necessary, a list of which has to be given as well. For this reason, a separate *makefile* is currently maintained in the directory `adjoint/`. This makefile is responsible for the adjoint code generation.

In the following we describe the build process step by step, assuming you are in the directory `bin/`. A summary of steps to follow is given at the end.

Adjoint code generation and compilation – step by step

1. `ln -s ../verification/???/code/.genmakerc .`
   `ln -s ../verification/???/code/*.[Fh] .`
   Link your customized genmake options, header files, and modified code to the compile directory.

2. `../tools/genmake -makefile`
   Generate your Makefile (cf. Section ???).

3. `make depend`
   Dependency analysis for the CPP pre-compiler (cf. Section ???).

4. `cd ../adjoint`
   `make adtaf` or `make adtamc`
   Depending on whether you have TAF or TAMC at your disposal, you'll choose `adtaf` or `adtamc` as your make target for the *makefile* in the directory `adjoint/`. Several things happen at this stage.

   (a) `make adrestore, make ftlrestore`
   The initial template files *adjoint_model.F* and *tangentlinear_model.F* in *pkg/autodiff* which are part of the compiling list created by *genmake* are restored.

   (b) `make depend, make small_f`
   The `bin/` directory is brought up to date, i.e. for recent changes in header or source code *.[Fh]*, corresponding *.f* routines are generated or re-generated. Note that here, only CPP precompiling is performed; no object code *.o* is generated as yet. Precompiling is necessary for TAMC to see the full code.

   (c) `make allcode`
   All Fortran routines `*.f` in `bin/` are concatenated into a single file called *tamc_code.f*.

   (d) `make admodeltaf/admodeltamc`
   Adjoint code is generated by TAMC or TAF. The adjoint code is written to the file *tamc_code_ad.f*. It contains all adjoint routines of the forward routines concatenated in *tamc_code.f*. For a given forward routines `subroutine routinename` the adjoint routine is named `adsubroutine routinename` by default (that default can be changed via the flag `-admark <markname>`). Furthermore, it may contain modified code which incorporates the translation of adjoint store directives into specific Fortran code. For a given forward routines `subroutine routinename` the modified routine is named `mdsubroutine routinename`. TAMC or TAF info is written to file *tamc_code.prot* or *taf.log*, respectively.

   (e) `make adchange`
   The multi-threading capability of MITgcm requires a slight change in the parameter list of some routines that are related to to active file handling. This post-processing invokes the sed script *adjoint_ecco_sed.com* to insert the threading counter **myThId** into the parameter list of those subroutines. The resulting code is written to file *tamc_code_sed_ad.f* and appended to the file *adjoint_model.F*. This concludes the adjoint code generation.

5. `cd ../bin`
   `make`
   The file *adjoint_model.F* now contains the full adjoint code. All routines are now compiled.

N.B.: The targets `make adtaf/adtamc` now comprise a series of targets that in previous versions had to be invoked separarely. This was probably preferable at a more experimental stage, but has now been dropped in favour of a more straightforward build process.

Adjoint code generation and compilation – summary

```
cd bin
ln -s ../verification/my_experiment/code/.genmakerc .
ln -s ../verification/my_experiment/code/*.[Fh] .
../tools/genmake -makefile
make depend
cd ../adjoint
make adtaf <OR: make adtamc>
     contains the targets:
     adrestore small_f allcode admodeltaf/admodeltamc adchange
cd ../bin
make
```

3.20 Offline Experiments

(in directory: *verification/tutorial_offline/*)

This document describes a simple experiment using the offline form of the MITgcm.

3.20.1 Overview

This experiment demonstrates use of the offline form of the MITgcm to study advection of a passive tracer. Time-averaged flow-fields and mixing coefficients, deriving from a prior online run, are re-used leaving only the tracer equation to be integrated.

Figure — *missing figure* — shows a movie of tracer being advected using the offline package of the MITgcm. In the top panel the frames of the movie show the monthly surface evolution of an initially local source of passive tracer. In the lower panel, the frames of the movie show the changing monthly surface evolution where the initial tracer field had a global distribution.

3.20.2 Time-stepping of tracers

see section 2.15 through 2.18 for details of available tracer time-stepping schemes and their characteristics.

3.20.3 Code Configuration

The model configuration for this experiment resides under the directory *verification/tutorial_offline*. The experiment files

- *input/data*
- *input/data.off*
- *input/data.pkg*
- *input/data.ptracers*
- *input/eedata*
- *input/packages.conf*
- *code/PTRACERS_SIZE.h*
- *code/SIZE.h*.

contain the code customisations and parameter settings required to run the example. In addition the following binary data files are required:

- *input/depth_g77.bin*
- *input/tracer1_.bin*
- *input/tracer2_.bin*
- *input/input_off/uVeltave.0000000001-12.data*
- *input/input_off/vVeltave.0000000001-12.data*
- *input/input_off/wVeltave.0000000001-12.data*
- *input/input_off/Convtave.0000000001-12.data*

3.20.3.1 File *input/data*

This file, reproduced completely below, specifies the main parameters for the experiment.

- Line 18, 19

```
nIter0  = 0,
nTimeSteps = 720,
```

nIter0 and nTimesteps control the start time and the length of the run (in timesteps). If nIter0 is non-zero the model will require appropriate pickup files to be present in the run directory. Where nIter0 is zero, as here, the model makes a fresh start. In this case the model has been prescribed to run for 720 timesteps or 1 year.

- Line 20

```
deltaTtracer= 43200.0,
```

deltaTtracer is the tracer timestep in seconds, in this case, 12 hours (43200s = 12 hours). Note that deltatTracer must be specified in *input/data* as well as specifying deltaToffline in *input/data.off*.

- Line 21

```
deltaTClock= 43200.0,
```

When using the MITgcm in offline mode deltaTClock (an internal model counter) should be made equal to the value assigned to deltatTtracer.

- Line 27

```
periodicExternalForcing=.TRUE.,
```

periodicExternalForcing is a flag telling the model whether to cyclically re-use forcing data where there is external forcing (see 'A More Complcated Example', below). Where there is no external forcing, as here, but where there is to be cyclic re-use of the offline flow and mixing fields, periodicExternalForcing must be assigned the value .TRUE.

- Line 28

```
externForcingPeriod=2592000.,
```

externForcingPeriod specifies the period of the external forcing data in seconds. In the absence of external forcing, as in this example, it must be made equal to the value of externForcingPeriod in *input/data.off*, in this case, monthly (2592000s = 1 month).

- Line 29

```
externForcingCycle=31104000.,
```

externForcingCycle specifies the duration of the external forcing data cycle in seconds. In the absence of external forcing, as in this example, it must be made equal to the value of externForcingCycle in *input/data.off*, in this case, the cycle is one year(31104000s = 1 year).

- Line 35

```
usingSphericalPolarGrid=.TRUE.,
```

This line requests that the simulation be performed in a spherical polar coordinate system. It affects the interpretation of grid input parameters and causes the grid generation routines to initialize an internal grid based on spherical polar geometry.

- Line 36

```
delR=  50.,  70., 100., 140., 190.,
      240., 290., 340., 390., 440.,
      490., 540., 590., 640., 690.,
```

This line sets the vertical grid spacing between each z-coordinate line in the discrete grid. Here the total model depth is 5200 m.

- Line 39

```
ygOrigin=-90.,
```

This line sets the southern boundary of the modeled domain to $-90°$ latitude N ($90°$ S). This value affects both the generation of the locally orthogonal grid that the model uses internally and affects the initialization of the coriolis force. Note - it is not required to set a longitude boundary, since the absolute longitude does not alter the kernel equation discretisation.

- Line 40

```
dxSpacing=2.8125,
```

This line sets the horizontal grid spacing between each y-coordinate line in the discrete grid to $2.8125°$ in longitude.

- Line 41

```
dySpacing=2.8125,
```

This line sets the vertical grid spacing between each x-coordinate line in the discrete grid to $2.8125°$ in latitude.

- Line 46

```
bathyFile='depth_g77.bin',
```

This line specifies the name of the file, in this case *depth_g77.bin*, from which the domain bathymetry is read. This file contains a two-dimensional (x, y) map of (assumed 64-bit) binary numbers giving the depth of the model at each grid cell, ordered with the x coordinate varying fastest. The points are ordered from low coordinate to high coordinate for both axes. The units and orientation of the depths in this file are the same as used in the MITgcm code. In this experiment, a depth of $0m$ indicates land.

```
1    # ====================
2    # | Model parameters |
3    # ====================
4    #
5    # Continuous equation parameters
6    &PARM01
7    &
8    #
9    # Elliptic solver parameters
10   &PARM02
11   cg2dMaxIters=1000,
12   cg2dTargetResidual=1.E-13,
13   &
14   #
15
16   # Time stepping parameters
```

```
&PARM03
nIter0  = 0,
nTimeSteps = 720,
deltaTtracer= 43200.0,
deltaTClock = 43200.0,
pChkptFreq=31104000.,
chkptFreq= 31104000.,
dumpFreq=  2592000.,
taveFreq=  311040000.,
monitorFreq= 1.,
periodicExternalForcing=.TRUE.,
externForcingPeriod=2592000.,
externForcingCycle=31104000.,
&
#
# Gridding parameters
&PARM04
usingCartesianGrid=.FALSE.,
usingSphericalPolarGrid=.TRUE.,
delR=  50.,  70., 100., 140., 190.,
      240., 290., 340., 390., 440.,
      490., 540., 590., 640., 690.,
ygOrigin=-90.,
dxSpacing=2.8125,
dySpacing=2.8125,
&
#
# Input datasets
&PARM05
bathyFile=      'depth_g77.bin',
&
```

3.20.3.2 File *input/data.off*

input/data.off provides the MITgcm offline package with package specific parameters. *input/data.off* specifies the location (relative to the run directory) and prefix of files describing the flow field (UvelFile, VvelFile, WvelFile) and the corresponding convective mixing coefficients (ConvFile) which together prescribe the three dimensional, time varying dynamic system within which the offline model will advect the tracer.

- Lines 2 to 5

```
UvelFile= '../input/input_off/uVeltave',
VvelFile= '../input/input_off/vVeltave',
WvelFile= '../input/input_off/wVeltave',
ConvFile= '../input/input_off/Convtave',
```

In the example the offline data is located in the sub-directory *input/input_off*. In this directory are fields describing the velocity and convective mixing histories of a prior forward integration of the MITgcm required for the offline package and identified in *input/data_off*. Based on the values of *deltatToffline*, *offlineForcingPeriod* and *offlineForcingCycle* specified in *input/data.off*, since *offlineForcingCycle* corresponds to 12 forcing periods *offlineForcingPeriod* and since offlineIter0 is zero, there needs to be 12 uVeltave, 12 vVeltave, 12 wVeltave and 12 Convtave files each having a 10 digit sequence identifier between 0000000001 to 0000000012, that is, a total of 48 files.

- Line 9

```
offlineIter0=0,
```

offlineIter0, here specified to be 0 timesteps, corresponds to the timestep at which the tracer model is initialised. Note that offlineIter0 and nIter0 (set in *input/data*) need not be the same.

- Line 10

```
deltaToffline=43200.,
```

deltatToffline sets the timestep associated with the offline model data in seconds, here 12 hours (43200s = 12 hours).

- Line 11

```
offlineForcingPeriod=43200.,
```

offlineForcingPeriod sets the forcing period associated with the offline model data in seconds.

- Line 12

```
offlineForcingCycle=518400.,
```

offlineForcingCycle sets the forcing cycle length associated with the offline model data in seconds. In this example the offline forcing cycle is 6 days, or 12 offline forcing periods. Together deltatToffline, offlineForcingPeriod and offlineForcingCycle determine the value of the 10 digit sequencing tag the model expects files in *input/input_off* to have.

```
1   &OFFLINE_PARM01
2    UvelFile= 'input_off/uVeltave',
3    VvelFile= 'input_off/vVeltave',
4    WvelFile= 'input_off/wVeltave',
5    ConvFile= 'input_off/Convtave',
6   &end

7   &OFFLINE_PARM02
8    offlineIter0=0,
9    deltaToffline=43200.,
10   offlineForcingPeriod=43200.,
11   offlineForcingCycle=518400.,
12  &end
```

3.20.3.3 File *input/data.pkg*

File *input/data.pkg*, reproduced completely below, specifies which MITgcm packages (*/MITgcm/pkg*) are to be used.

- Line 3

```
usePTRACERS=.TRUE.,
```

usePTRACERS is a flag invoking the ptracers package which is responsible for the advection of the tracer within the model.

```
1  # Packages
2   &PACKAGES
3   usePTRACERS=.TRUE.,
4   &
```

3.20.3.4 File *input/data.ptracers*

File *input/data.ptracers*, reproduced completely below, provides the MITgcm ptracers package with package specific parameters, prescribing the nature of the the tracer/tracers as well as the variables associated with their advection.

- Line 2

```
PTRACERS_numInUse=2,
```

PTRACERS_numInUse tells the model how many separate tracers are to be advected, in this case 2. Note: The value of PTRACERS_numInUse must agree with the value specified in *code/PTRACERS_SIZE.h* - see *code/PTRACERS_SIZE.h* below.

- Line 3

```
PTRACERS_Iter0= 0,
```

PTRACERS_Iter0 specifies the iteration at which the tracer is to be introduced. In this case the tracer is initialised at the start of the simulation. i.e. PTRACERS_Iter0 = PTRACERS_nIter0.

- Lines 5 and 10

```
PTRACERS_advScheme(1)=77,
```

PTRACERS_advScheme(n) identifies which advection scheme will be used for tracer n, where n is the number of the tracer up to PTRACERS_numInUse. See section 2.18, 'Comparison of advection schemes', to identify the numerical codes used to specify different advection schemes (e.g. centered 2nd order, 3rd order upwind) as well as details of each.

- Lines 6 and 11

```
PTRACERS_diffKh(1)=1.E3,
```

PTRACERS_diffKh(n) is the horizontal diffusion coefficient for tracer n, where n is the number of the tracer up to PTRACERS_numInUse.

- Lines 7 and 12

```
PTRACERS_diffKr(1)=5.E-5,
```

PTRACERS_diffKr(n) is the vertical diffusion coefficient for tracer n, where n is the number of the tracer up to PTRACERS_numInUse.

- Lines 8 and 13

```
PTRACERS_initialFile(1)='tracer1.bin',
```

PTRACERS_initialFile(n) identifies the initial tracer field to be associated with tracer n, where n is the number of the tracer up to PTRACERS_numInUse. In this example file *input/tracer1.bin* contains localised tracer, *input/tracer2.bin* contains an arbitrary global distribution. Included Matlab script *input/makeinitialtracer.m* provides a template for generating or manipulating initial tracer fields.

```
1   &PTRACERS_PARM01
2   PTRACERS_numInUse=2,
3   PTRACERS_Iter0= 0,
4  # tracer 1
5   PTRACERS_advScheme(1)=77,
6   PTRACERS_diffKh(1)=1.E3,
```

```
 PTRACERS_diffKr(1)=5.E-5,
 PTRACERS_initialFile(1)='tracer1.bin',
# tracer 2
 PTRACERS_advScheme(2)=77,
 PTRACERS_diffKh(2)=1.E3,
 PTRACERS_diffKr(2)=5.E-5,
 PTRACERS_initialFile(2)='tracer2.bin',
 &
```

Note *input/data.ptracers* requires a set of entries for each tracer.

3.20.3.5 File *input/eedata*

This file uses standard default values and does not contain customisations for this experiment.
The following code changes are required to run this exaperiment.

3.20.3.6 File *code/packages.conf*

This file is used to invoke the model components required for a particuylar implementation of the MITgcm. In this case the *code/packages.conf* contains the component names:

```
ptracers
generic_advdiff
mdsio
mom_fluxform
mom_vecinv
timeave
rw
monitor
offline
```

3.20.3.7 File *code/PTRACERS_SIZE.h*

- Line

  ```
  PARAMETER(PTRACERS_num = 2 )
  ```

This line sets the parameters PTRACERS_num (the number of tracers to be integrated) to 2 (in agreement with *input/data.ptracers*).

3.20.3.8 File *code/SIZE.h*

Two lines are customized in this file for the current experiment

- Line 39,

  ```
  sNx=128,
  ```

  this line sets the lateral domain extent in grid points for the axis aligned with the x-coordinate.

- Line 40,

  ```
  sNy=64,
  ```

  this line sets the lateral domain extent in grid points for the axis aligned with the y-coordinate.

- Line 49,

  ```
  Nr=15,
  ```

  this line sets the vertical domain extent in grid points.

3.20.4 Running The Example

3.20.4.1 Code Download

In order to run the examples you must first download the code distribution. Instructions for downloading the code can be found in section 3.2.

3.20.4.2 Experiment Location

This example experiment is located under the release sub-directory

verification/offline/

3.20.4.3 Running the Experiment

To run the experiment

1. Set the current directory to *input/*

   ```
   % cd input
   ```

2. Verify that current directory is now correct

   ```
   % pwd
   ```

   You should see a response on the screen ending in *verification/offline/input*

3. Copy the contents of *input/* including subdirectory *input/input_off/* to a new directory called *run/*

4. Listing directory *run/* you should see:

   ```
   data      data.pkg       depth_g77.bin  input_off    tracer2.bin
   data.off  data.ptracers  eedata         tracer1.bin
   ```

5. Set the current directory to *run/*

6. Run the genmake script to create the experiment *Makefile*

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

7. Create a list of header file dependencies in *Makefile*

   ```
   % make depend
   ```

8. Build the executable file.

   ```
   % make
   ```

9. Run the *mitgcmuv* executable

   ```
   % ./mitgcmuv
   ```

Besides the input files and the files the model generates describing the grid (prefixed Depth, DXC, DXG, hFacC, hFacS and hFacW, you should now have 26 single precision binary files PTRACER01.0000000000-0000000720.001.001.data and PTRACER02.0000000000-0000000720.001.001.data and their 26 corresponding meta files as well as a single pickup file, pickup_ptracers.0000000720.001.001.data and its corresponding meta file pickup_ptracers.0000000720.001.001.meta. To run on simply change nIter0 in file *run/data* to 720...

3.20.5 A more complicated example

(in directory: *verification/tutorial_cfc_offline/*)

The last example demonstrated simple advection of a passive tracer using the offline form of the MITgcm. Now we present a more complicated example in which the model is used to explore contamination of the global ocean through surface exposure to CFC's during the last century. In invoking packages gchem, gmredi and cfc it provides a starting point and template for more complicated offline modeling, involving as it does surface forcing through wind and ice fields, more sophisticated mixing and a time-varying forcing funtion.

The model configuration for this experiment resides under the directory *verification/cfc_offline*. The experiment files

- *input/data*
- *input/data.gchem*
- *input/data.gmredi*
- *input/data.off*
- *input/data.pkg*
- *input/data.ptracers*
- *input/eedata*
- *code/GCHEM_OPTIONS.h*
- *code/GMREDI_OPTIONS.h*
- *input/packages.conf*
- *code/PTRACERS_SIZE.h*
- *code/SIZE.h*.

contain all the code customisations and parameter settings required.
The full list of other files required becomes:

```
cfc1112.atm  data.ptracers
data         depth_g77.bin   pickup.0004269600.data
data.gchem   eedata
data.gmredi  ice.bin         pickup_ptracers.0004269600.data
data.off     data.pkg        tren_speed.bin
```

and

```
input_off/:
Convtave.0004248060.data  GM_Kwz-T.0004248060.data  uVeltave.0004248060.data
Convtave.0004248060.meta  GM_Kwz-T.0004248060.meta  uVeltave.0004248060.meta
Convtave.0004248720.data  GM_Kwz-T.0004248720.data  uVeltave.0004248720.data
Convtave.0004248720.meta  GM_Kwz-T.0004248720.meta  uVeltave.0004248720.meta
GM_Kwx-T.0004248060.data  Stave.0004248060.data     vVeltave.0004248060.data
GM_Kwx-T.0004248060.meta  Stave.0004248060.meta     vVeltave.0004248060.meta
GM_Kwx-T.0004248720.data  Stave.0004248720.data     vVeltave.0004248720.data
GM_Kwx-T.0004248720.meta  Stave.0004248720.meta     vVeltave.0004248720.meta
GM_Kwy-T.0004248060.data  Ttave.0004248060.data     wVeltave.0004248060.data
GM_Kwy-T.0004248060.meta  Ttave.0004248060.meta     wVeltave.0004248060.meta
GM_Kwy-T.0004248720.data  Ttave.0004248720.data     wVeltave.0004248720.data
GM_Kwy-T.0004248720.meta  Ttave.0004248720.meta     wVeltave.0004248720.meta
```

3.20.5.1 File *input/data*

A single line must be added (under PARM01, between lines 6 and 7) in file *input/data* from the previous example

```
&PARM01
implicitDiffusion=.TRUE.,
&
```

When package GMREDI is used, the flag implicitDiffusion must be assigned the value .TRUE. For information about MITgcm package GMREDI see...

In this example the starting timestep nIter0 is set to 4269600 requiring model access to pickup files with the timetag 0004269600. The model will run for 4 timesteps (nTimeSteps = 4). In this case the frequencies with which permanent and rolling checkpoints (pChkptFreq and chkptFreq) have been set is sufficiently long to ensure that only one from the last timestep will be written. This is also true of the values that have been assigned to the frequency with which dumps are written (dumpFreq) and time averaging (taveFreq) is performed however since the model always dumps the state of the model when it stops without error a dump will be written with timetag 0004269604 upon completion.

3.20.5.2 File *input/data.off*

File *input/data.off*, reproduced in full below, specifies the prefixes and locations of additional input files required to run the offline model. Note that *input/input_off* contains only as many offline files as are required to successfully run for 4 timesteps. Where the GMREDI scheme was used in the forward run, as here, package GMREDI must again be invoked when running offline. In this example tracer is specified as having beeen introduced with a non-zero starttime, at timestep 4248000.

```
&OFFLINE_PARM01
 UvelFile= '../input/input_off/uVeltave',
 VvelFile= '../input/input_off/vVeltave',
 WvelFile= '../input/input_off/wVeltave',
 GMwxFile= '../input/input_off/GM_Kwx-T',
 GMwyFile= '../input/input_off/GM_Kwy-T',
 GMwzFile= '../input/input_off/GM_Kwz-T',
 ConvFile= '../input/input_off/Convtave',
&end

&OFFLINE_PARM02
 offlineIter0=4248000,
 deltaToffline=43200.,
 offlineForcingPeriod=2592000.,
 offlineForcingCycle=31104000.,
&end
```

3.20.5.3 File *input/data.pkg*

File *input/data.pkg*, reproduced completely below, specifies which MITgcm packages (*/MITgcm/pkg*) are to be used. It now invokes additional packages *pkg/gmredi* and pkg/gchem.

```
# Packages
&PACKAGES
useGMRedi=.TRUE.,
usePTRACERS=.TRUE.,
useGCHEM=.TRUE.,
&
```

3.20.5.4 File *input/data.ptracers*

File *input/data.ptracers*, reproduced completely below, specifies parameters associated with the CFC11 and CFC12 tracer fields advected in this example.

- Line 3

```
PTRACERS_Iter0= 4248000,
```

In this example the tracers were introduced at iteration 4248000.

- Lines 7 and 14

```
PTRACERS_diffKh(n)=0.E3,
```

Since package GMREDI is being used, regular horizontal diffusion is set to zero.

- Lines 9,10 and 16,17

```
PTRACERS_useGMRedi(n)=.TRUE. ,
PTRACERS_useKPP(n)=.FALSE. ,
```

Setting flag PTRACERS_useGMRedi(n) to .TRUE. identifies that package GMREDI is to be used. Setting flag PTRACERS_useKPP(n) to .FALSE. explicitly turns off KPP mixing.

- Lines 11 and 18

```
PTRACERS_initialFile(n)=' ',
```

Since this is a 'pickup' run the initial tracer files PTRACERS_initialFile(1) and PTRACERS_initialFile(2) aree not needed. The model will obtain the tracer state from pickup_ptracers.0004269600.data

```
1   &PTRACERS_PARM01
2   PTRACERS_numInUse=2,
3   PTRACERS_Iter0= 4248000,
4  #
5  # tracer 1 - CFC11
6   PTRACERS_advScheme(1)=77,
7   PTRACERS_diffKh(1)=0.E3,
8   PTRACERS_diffKr(1)=5.E-5,
9   PTRACERS_useGMRedi(1)=.TRUE. ,
10  PTRACERS_useKPP(1)=.FALSE. ,
11  PTRACERS_initialFile(1)=' ',
12 # tracer 2 - CFC12
13  PTRACERS_advScheme(2)=77,
14  PTRACERS_diffKh(2)=0.E3,
15  PTRACERS_diffKr(2)=5.E-5,
16  PTRACERS_useGMRedi(2)=.TRUE. ,
17  PTRACERS_useKPP(2)=.FALSE. ,
18  PTRACERS_initialFile(2)=' ',
19  &
```

3.20.5.5 File *input/data.gchem*

File *input/data.gchem*, reproduced completely below, names the forcing files needed in package GCHEM.

- Line 3

```
iceFile='ice.bin',
```

File *input/ice.bin* contains 12, monthly surface ice fields.

- Line 3

```
iceFile='tren_speed.bin',
```

File *input/tren_speed.bin* contains 12, monthly surface wind fields.

```
1   &GCHEM_PARM01
2    iceFile='fice.bin',
3    windFile='tren_speed.bin',
4   &
```

Package GCHEM is described in detail in section ??

3.20.5.6 File *input/data.gmredi*

File *input/data.gmredi*, reproduced completely below, provides the parameters required for package GM-REDI.

```
&GM_PARM01
 GM_background_K    = 1.e+3,
 GM_taper_scheme    = 'gkw91',
 GM_maxSlope        = 1.e-2,
 GM_Kmin_horiz      = 100.,
 GM_Scrit           = 4.e-3,
 GM_Sd              = 1.e-3,
&
```

Package GMREDI is described in detail in section ??

3.20.5.7 File *input/cfc1112.atm*

File *input/cfc1112.atm* is a text file containing the CFC source functions over the northern and southern hemispheres annually from 1931 through 1998.

3.20.5.8 File *code/packages.conf*

In this example *code/packages.conf* additionally invokes components gchem, cfc and rmredi:

```
ptracers
generic_advdiff
mdsio
mom_fluxform
mom_vecinv
timeave
rw
monitor
offline
gchem
cfc
gmredi
```

3.20.5.9 File *code/GCHEM_OPTIONS.h*

File *code/GCHEM_OPTIONS.h*, specifies options for package GCHEM. In this case defining the flag ALLOW_CFC to activate the cfc code.

3.20.5.10 File *code/GMREDI_OPTIONS.h*

File *code/GCHEM_OPTIONS.h*, specifies options for package GMREDI.

3.20.5.11 File *code/PTRACERS_SIZE.h*

File *code/PTRACERS_SIZE.h* is unchanged from the simpler example.

3.20.5.12 File *code/SIZE.h*

File *code/SIZE.h* is unchanged from the simpler example.

3.20.5.13 Running the Experiment

The model is run as before and produces the files the model generates describing the grid (prefixed Depth, DXC, DXG, hFacC, hFacS and hFacW) as well as 2, single precision, binary files PTRACER01.0004269600-0004269604.001.001.data and PTRACER02.0004269600-0004269604.001.001.data and their 2 corresponding meta files as well as a single pickup file, pickup_ptracers.ckptA.001.001.data and its corresponding meta file pickup_ptracers.ckptA.001.001.meta from which you could run the model on.

3.21 A Rotating Tank in Cylindrical Coordinates

(in directory: *verification/rotating_tank/*)

3.21.1 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 cyclindrical grid with twenty-nine vertical levels of 0.5cm each. This is a typical laboratory setup for illustration principles of GFD, as well as for a laboratory data assimilation project. The files for this experiment can be found in the verification directory under rotating_tank.

example illustration from GFD lab here

3.21.2 Equations Solved

3.21.3 Discrete Numerical Configuration

The domain is discretised with a uniform cylindrical grid spacing in the horizontal set to $\Delta a = 1$ cm and $\Delta\phi = 3^\circ$, so that there are 120 grid cells in the azimuthal direction and thirty-one grid cells in the radial, representing a tank 62cm in diameter. The bathymetry file sets the depth=0 in the nine lowest radial rows to represent the central of the annulus. Vertically the model is configured with twenty-nine layers of uniform 0.5cm thickness.
something about heat flux

3.21.4 Code Configuration

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

- *input/data*
- *input/data.pkg*
- *input/eedata*,
- *input/bathyPol.bin*,
- *input/thetaPol.bin*,
- *code/CPP_EEOPTIONS.h*
- *code/CPP_OPTIONS.h*,
- *code/SIZE.h*.

contain the code customizations and parameter settings for this experiments. Below we describe the customizations to these files associated with this experiment.

3.21.4.1 File *input/data*

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

- Lines 9-10,

```
viscAh=5.0E-6,
viscAz=5.0E-6,
```

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

- Lines 13-16,

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

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

- Line 17,

```
f0=0.5 ,
```

  this line sets the coriolis term, and represents a tank spinning at about 2.4 rpm.

- Lines 23 and 24

```
rigidLid=.TRUE.,
implicitFreeSurface=.FALSE.,
```

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

- Line 40,

```
nIter=0,
```

  This line indicates that the experiment should start from $t = 0$ and implicitly suppresses searching for checkpoint files associated with restarting an numerical integration from a previously saved state. Instead, the file thetaPol.bin will be loaded to initialized the temperature fields as indicated below, and other variables will be initialized to their defaults.

- Line 43,

```
deltaT=0.1,
```

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

- Line 56,

```
usingCylindricalGrid=.TRUE.,
```

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

- Line 57,

```
dXspacing=3,
```

  This line sets the azimuthal grid spacing between each x-coordinate line in the discrete grid. The syntax indicates that the discrete grid should be comprised of 120 grid lines each separated by 3°.

- Line 58,

```
dYspacing=0.01,
```

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

- Line 59,

```
delZ=29*0.005,
```

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

- Line 64,

```
bathyFile='bathyPol.bin',
```

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

- Line 65,

```
hydrogThetaFile='thetaPol.bin',
```

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

- Lines 66 and 67

```
 tCylIn  = 0
 tCylOut  = 20
```

These line specify the temperatures in degrees Celsius of the interior and exterior walls of the tank – typically taken to be icewater on the inside and room temperature on the outside.

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

```
# ====================
# | 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,
 sBeta =0.,
```

```
 gravity=9.81,
 rhoConst=1000.0,
 rhoNil=1000.0,
# heatCapacity_Cp=3900.0,
 rigidLid=.TRUE.,
 implicitFreeSurface=.FALSE.,
 eosType='LINEAR',
 nonHydrostatic=.TRUE.,
 readBinaryPrec=32,
 &

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

# Time stepping parameters
 &PARM03
 nIter0=0,
 nTimeSteps=20,
# nTimeSteps=36000000,
 deltaT=0.1,
 abEps=0.1,
 pChkptFreq=1.0,
 chkptFreq=1.0,
 dumpFreq=1.0,
 monitorFreq=0.1,
 outputTypesInclusive=.TRUE.,
 &

# Gridding parameters
 &PARM04
 usingCartesianGrid=.FALSE.,
 usingCylindricalGrid=.TRUE.,
 usingCurvilinearGrid=.FALSE.,
 dXspacing=3,
 dYspacing=0.01,
 delZ=29*0.005,
 &

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

3.21.4.2 File *input/data.pkg*

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

3.21.4.3 File *input/eedata*

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

3.21.4.4 File *input/thetaPol.bin*

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

3.21.4.5 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 **-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.

3.21.4.6 File *code/SIZE.h*

Two lines are customized in this file for the current experiment

- Line 39,

  ```
  sNx=120,
  ```

  this line sets the lateral domain extent in grid points for the axis aligned with the x-coordinate.

- Line 40,

  ```
  sNy=31,
  ```

  this line sets the lateral domain extent in grid points for the axis aligned with the y-coordinate.

```
C $Header: /u/gcmpack/manual/s_examples/rotating_tank/code/SIZE.h,v 1.1 2004/07/26 21:09:47 afe Exp $
C $Name:  $
C
C     /==========================================================\
C     | SIZE.h Declare size of underlying computational grid.    |
C     |==========================================================|
C     | The design here support a three-dimensional model grid   |
C     | with indices I,J and K. The three-dimensional domain     |
C     | is comprised of nPx*nSx blocks of size sNx along one axis|
C     | nPy*nSy blocks of size sNy along another axis and one    |
C     | block of size Nz along the final axis.                   |
C     | Blocks have overlap regions of size OLx and OLy along the|
C     | dimensions that are subdivided.                          |
C     \==========================================================/
C     Voodoo numbers controlling data layout.
C     sNx - No. X points in sub-grid.
C     sNy - No. Y points in sub-grid.
C     OLx - Overlap extent in X.
C     OLy - Overlat extent in Y.
C     nSx - No. sub-grids in X.
C     nSy - No. sub-grids in Y.
C     nPx - No. of processes to use in X.
C     nPy - No. of processes to use in Y.
C     Nx  - No. points in X for the total domain.
C     Ny  - No. points in Y for the total domain.
C     Nr  - No. points in Z for full process domain.
      INTEGER sNx
      INTEGER sNy
      INTEGER OLx
      INTEGER OLy
      INTEGER nSx
      INTEGER nSy
      INTEGER nPx
```

```
      INTEGER nPy
      INTEGER Nx
      INTEGER Ny
      INTEGER Nr
      PARAMETER (
     &           sNx =  120,
     &           sNy =  31,
     &           OLx =   3,
     &           OLy =   3,
     &           nSx =   1,
     &           nSy =   1,
     &           nPx =   1,
     &           nPy =   1,
     &           Nx  = sNx*nSx*nPx,
     &           Ny  = sNy*nSy*nPy,
     &           Nr  =  29)

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

3.21.4.7 File *code/CPP_OPTIONS.h*

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

3.21.4.8 File *code/CPP_EEOPTIONS.h*

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

Chapter 4

Software Architecture

This chapter focuses on describing the **WRAPPER** environment within which both the core numerics and the pluggable packages operate. The description presented here is intended to be a detailed exposition and contains significant background material, as well as advanced details on working with the WRAPPER. The tutorial sections of this manual (see sections 3.8 and 3.19) contain more succinct, step-by-step instructions on running basic numerical experiments, of varous types, both sequentially and in parallel. For many projects simply starting from an example code and adapting it to suit a particular situation will be all that is required. The first part of this chapter discusses the MITgcm architecture at an abstract level. In the second part of the chapter we described practical details of the MITgcm implementation and of current tools and operating system features that are employed.

4.1 Overall architectural goals

Broadly, the goals of the software architecture employed in MITgcm are three-fold

- We wish to be able to study a very broad range of interesting and challenging rotating fluids problems.
- We wish the model code to be readily targeted to a wide range of platforms
- On any given platform we would like to be able to achieve performance comparable to an implementation developed and specialized specifically for that platform.

These points are summarized in figure 4.1 which conveys the goals of the MITgcm design. The goals lead to a software architecture which at the high-level can be viewed as consisting of

1. A core set of numerical and support code. This is discussed in detail in section 2.
2. A scheme for supporting optional "pluggable" **packages** (containing for example mixed-layer schemes, biogeochemical schemes, atmospheric physics). These packages are used both to overlay alternate dynamics and to introduce specialized physical content onto the core numerical code. An overview of the **package** scheme is given at the start of part 6.
3. A support framework called **WRAPPER** (Wrappable Application Parallel Programming Environment Resource), within which the core numerics and pluggable packages operate.

This chapter focuses on describing the **WRAPPER** environment under which both the core numerics and the pluggable packages function. The description presented here is intended to be a detailed exposition and contains significant background material, as well as advanced details on working with the WRAPPER. The examples section of this manual (part 3) contains more succinct, step-by-step instructions on running basic numerical experiments both sequentially and in parallel. For many projects simply starting from an example code and adapting it to suit a particular situation will be all that is required.

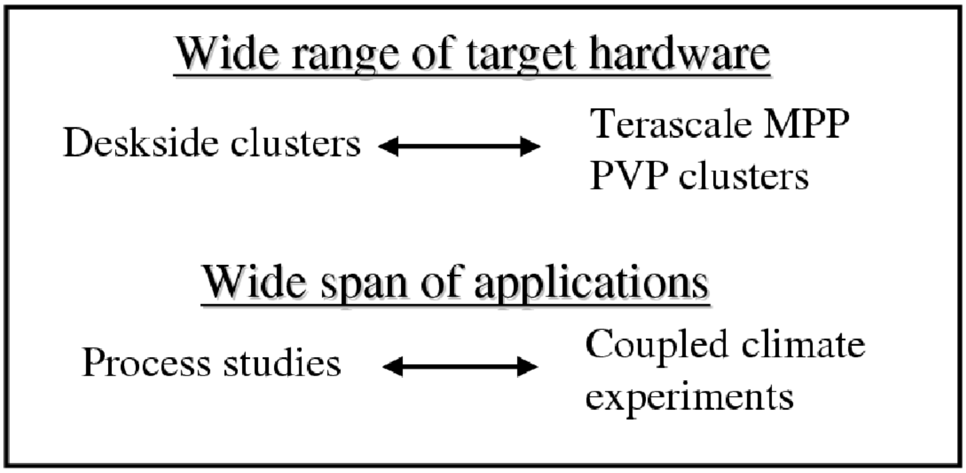


Figure 4.1: The MITgcm architecture is designed to allow simulation of a wide range of physical problems on a wide range of hardware. The computational resource requirements of the applications targeted range from around 10^7 bytes ($\approx$ 10 megabytes) of memory to 10^{11} bytes ($\approx$ 100 gigabytes). Arithmetic operation counts for the applications of interest range from 10^9 floating point operations to more than 10^{17} floating point operations.

4.2 WRAPPER

A significant element of the software architecture utilized in MITgcm is a software superstructure and substructure collectively called the WRAPPER (Wrappable Application Parallel Programming Environment Resource). All numerical and support code in MITgcm is written to "fit" within the WRAPPER infrastructure. Writing code to "fit" within the WRAPPER means that coding has to follow certain, relatively straightforward, rules and conventions (these are discussed further in section 4.3.1).

The approach taken by the WRAPPER is illustrated in figure 4.2 which shows how the WRAPPER serves to insulate code that fits within it from architectural differences between hardware platforms and operating systems. This allows numerical code to be easily retargetted.

4.2.1 Target hardware

The WRAPPER is designed to target as broad as possible a range of computer systems. The original development of the WRAPPER took place on a multi-processor, CRAY Y-MP system. On that system, numerical code performance and scaling under the WRAPPER was in excess of that of an implementation that was tightly bound to the CRAY systems proprietary multi-tasking and micro-tasking approach. Later developments have been carried out on uniprocessor and multi-processor Sun systems with both uniform memory access (UMA) and non-uniform memory access (NUMA) designs. Significant work has also been undertaken on x86 cluster systems, Alpha processor based clustered SMP systems, and on cache-coherent NUMA (CC-NUMA) systems such as Silicon Graphics Altix systems. The MITgcm code, operating within the WRAPPER, is also routinely used on large scale MPP systems (for example, Cray T3E and IBM SP systems). In all cases numerical code, operating within the WRAPPER, performs and scales very competitively with equivalent numerical code that has been modified to contain native optimizations for a particular system *Hoe et al.* [1999].

4.2.2 Supporting hardware neutrality

The different systems listed in section 4.2.1 can be categorized in many different ways. For example, one common distinction is between shared-memory parallel systems (SMP and PVP) and distributed memory parallel systems (for example x86 clusters and large MPP systems). This is one example of a difference between compute platforms that can impact an application. Another common distinction is between vector processing systems with highly specialized CPUs and memory subsystems and commodity

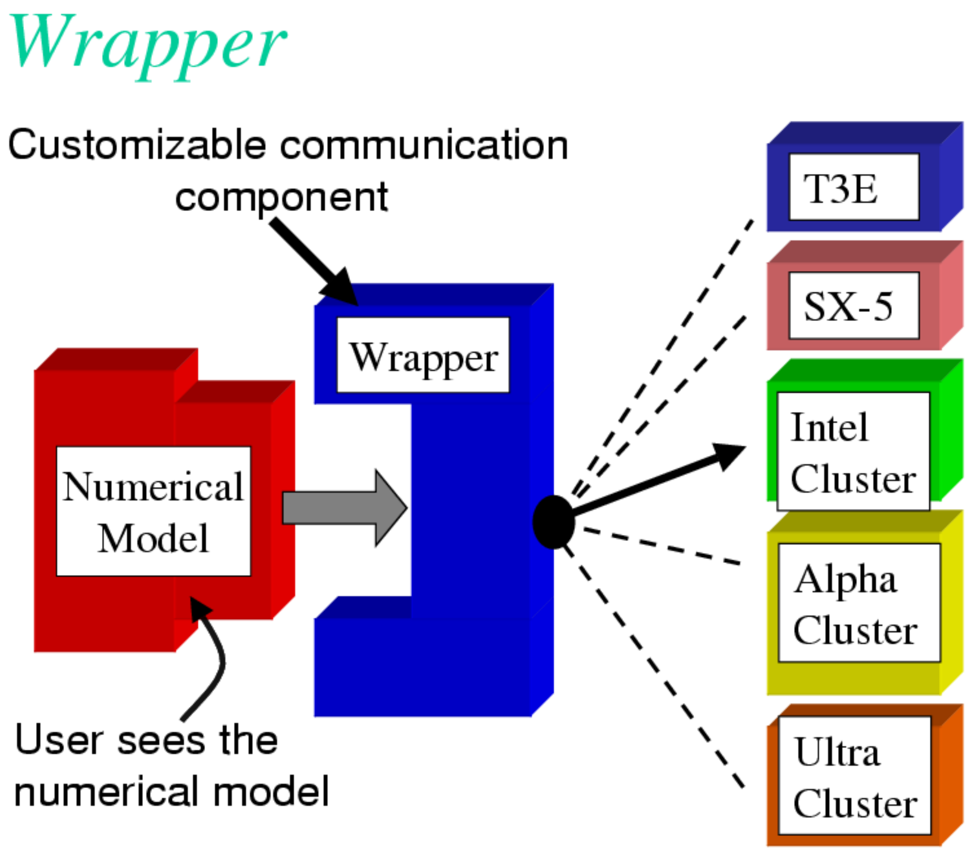


Figure 4.2: Numerical code is written to fit within a software support infrastructure called WRAPPER. The WRAPPER is portable and can be specialized for a wide range of specific target hardware and programming environments, without impacting numerical code that fits within the WRAPPER. Codes that fit within the WRAPPER can generally be made to run as fast on a particular platform as codes specially optimized for that platform.

microprocessor based systems. There are numerous other differences, especially in relation to how parallel execution is supported. To capture the essential differences between different platforms the WRAPPER uses a *machine model*.

4.2.3 WRAPPER machine model

Applications using the WRAPPER are not written to target just one particular machine (for example an IBM SP2) or just one particular family or class of machines (for example Parallel Vector Processor Systems). Instead the WRAPPER provides applications with an abstract *machine model*. The machine model is very general, however, it can easily be specialized to fit, in a computationally efficient manner, any computer architecture currently available to the scientific computing community.

4.2.4 Machine model parallelism

Codes operating under the WRAPPER target an abstract machine that is assumed to consist of one or more logical processors that can compute concurrently. Computational work is divided among the logical processors by allocating "ownership" to each processor of a certain set (or sets) of calculations. Each set of calculations owned by a particular processor is associated with a specific region of the physical space that is being simulated, only one processor will be associated with each such region (domain decomposition).

In a strict sense the logical processors over which work is divided do not need to correspond to physical processors. It is perfectly possible to execute a configuration decomposed for multiple logical processors on a single physical processor. This helps ensure that numerical code that is written to fit within the WRAPPER will parallelize with no additional effort. It is also useful for debugging purposes. Generally, however, the computational domain will be subdivided over multiple logical processors in order to then bind those logical processors to physical processor resources that can compute in parallel.

4.2.4.1 Tiles

Computationally, the data structures (*eg.* arrays, scalar variables, etc.) that hold the simulated state are associated with each region of physical space and are allocated to a particular logical processor. We refer to these data structures as being **owned** by the processor to which their associated region of physical space has been allocated. Individual regions that are allocated to processors are called **tiles**. A processor can own more than one tile. Figure 4.3 shows a physical domain being mapped to a set of logical processors, with each processors owning a single region of the domain (a single tile). Except for periods of communication and coordination, each processor computes autonomously, working only with data from the tile (or tiles) that the processor owns. When multiple tiles are alloted to a single processor, each tile is computed on independently of the other tiles, in a sequential fashion.

4.2.4.2 Tile layout

Tiles consist of an interior region and an overlap region. The overlap region of a tile corresponds to the interior region of an adjacent tile. In figure 4.4 each tile would own the region within the black square and hold duplicate information for overlap regions extending into the tiles to the north, south, east and west. During computational phases a processor will reference data in an overlap region whenever it requires values that lie outside the domain it owns. Periodically processors will make calls to WRAPPER functions to communicate data between tiles, in order to keep the overlap regions up to date (see section 4.2.8). The WRAPPER functions can use a variety of different mechanisms to communicate data between tiles.

4.2.5 Communication mechanisms

Logical processors are assumed to be able to exchange information between tiles and between each other using at least one of two possible mechanisms.

- **Shared memory communication**. Under this mode of communication data transfers are assumed to be possible using direct addressing of regions of memory. In this case a CPU is able to read (and write) directly to regions of memory "owned" by another CPU using simple programming language level assignment operations of the the sort shown in figure 4.5. In this way one CPU (CPU1 in the

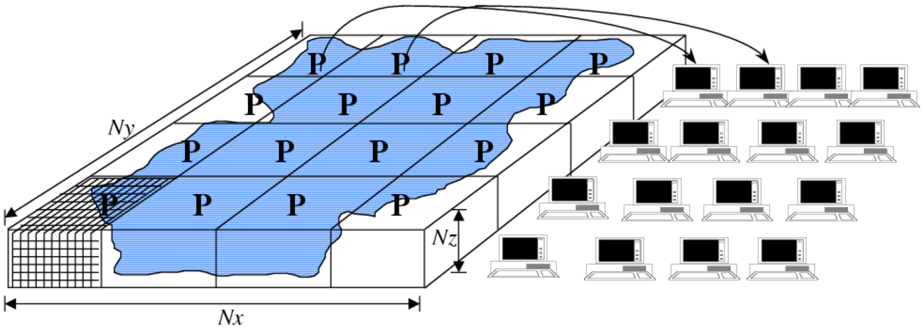


Figure 4.3: The WRAPPER provides support for one and two dimensional decompositions of grid-point domains. The figure shows a hypothetical domain of total size $N_x N_y N_z$. This hypothetical domain is decomposed in two-dimensions along the N_x and N_y directions. The resulting **tiles** are **owned** by different processors. The **owning** processors perform the arithmetic operations associated with a **tile**. Although not illustrated here, a single processor can **own** several **tiles**. Whenever a processor wishes to transfer data between tiles or communicate with other processors it calls a WRAPPER supplied function.

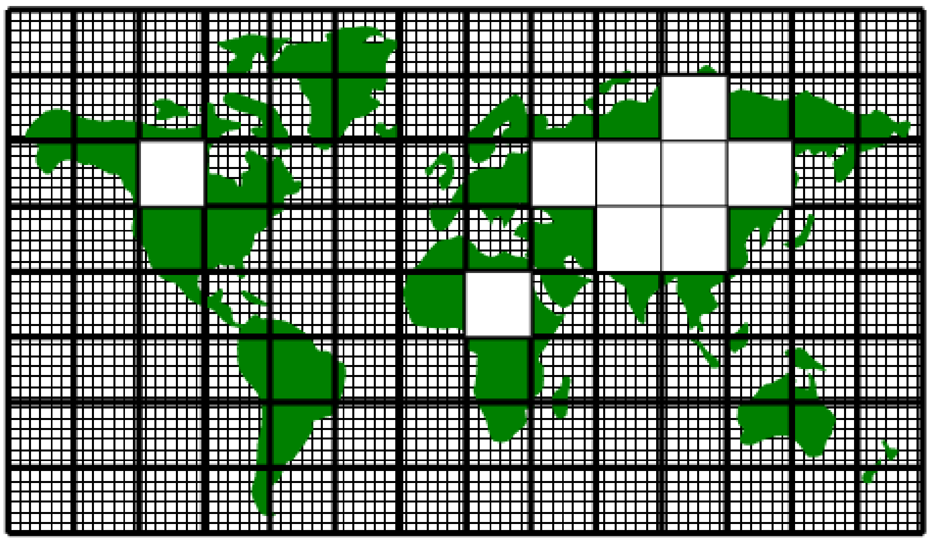


Figure 4.4: A global grid subdivided into tiles. Tiles contain a interior region and an overlap region. Overlap regions are periodically updated from neighboring tiles.

```
         CPU1                   |        CPU2
         ====                   |        ====
                                |
       a(3) = 8                 |        WHILE ( a(3) .NE. 8 )
                                |         WAIT
                                |        END WHILE
                                |
```

Figure 4.5: In the WRAPPER shared memory communication model, simple writes to an array can be made to be visible to other CPUs at the application code level. So that for example, if one CPU (CPU1 in the figure above) writes the value 8 to element 3 of array a, then other CPUs (for example CPU2 in the figure above) will be able to see the value 8 when they read from $a(3)$. This provides a very low latency and high bandwidth communication mechanism.

```
         CPU1                   |        CPU2
         ====                   |        ====
                                |
       a(3) = 8                 |        WHILE ( a(3) .NE. 8 )
       CALL SEND( CPU2,a(3) )   |         CALL RECV( CPU1, a(3) )
                                |        END WHILE
                                |
```

Figure 4.6: In the WRAPPER distributed memory communication model data can not be made directly visible to other CPUs. If one CPU writes the value 8 to element 3 of array a, then at least one of CPU1 and/or CPU2 in the figure above will need to call a bespoke communication library in order for the updated value to be communicated between CPUs.

figure) can communicate information to another CPU (CPU2 in the figure) by assigning a particular value to a particular memory location.

- **Distributed memory communication**. Under this mode of communication there is no mechanism, at the application code level, for directly addressing regions of memory owned and visible to another CPU. Instead a communication library must be used as illustrated in figure 4.6. In this case CPUs must call a function in the API of the communication library to communicate data from a tile that it owns to a tile that another CPU owns. By default the WRAPPER binds to the MPI communication library *Message Passing Interface Forum* [1998] for this style of communication.

The WRAPPER assumes that communication will use one of these two styles of communication. The underlying hardware and operating system support for the style used is not specified and can vary from system to system.

4.2.6 Shared memory communication

Under shared communication independent CPUs are operating on the exact same global address space at the application level. This means that CPU 1 can directly write into global data structures that CPU 2 "owns" using a simple assignment at the application level. This is the model of memory access is supported at the basic system design level in "shared-memory" systems such as PVP systems, SMP systems, and on distributed shared memory systems (*eg.* SGI Origin, SGI Altix, and some AMD Opteron systems). On such systems the WRAPPER will generally use simple read and write statements to access directly application data structures when communicating between CPUs.

In a system where assignments statements, like the one in figure 4.5 map directly to hardware instructions that transport data between CPU and memory banks, this can be a very efficient mechanism for communication. In this case two CPUs, CPU1 and CPU2, can communicate simply be reading and writing to an agreed location and following a few basic rules. The latency of this sort of communication is generally not that much higher than the hardware latency of other memory accesses on the system.

The bandwidth available between CPUs communicating in this way can be close to the bandwidth of the systems main-memory interconnect. This can make this method of communication very efficient provided it is used appropriately.

4.2.6.1 Memory consistency

When using shared memory communication between multiple processors the WRAPPER level shields user applications from certain counter-intuitive system behaviors. In particular, one issue the WRAPPER layer must deal with is a systems memory model. In general the order of reads and writes expressed by the textual order of an application code may not be the ordering of instructions executed by the processor performing the application. The processor performing the application instructions will always operate so that, for the application instructions the processor is executing, any reordering is not apparent. However, in general machines are often designed so that reordering of instructions is not hidden from other second processors. This means that, in general, even on a shared memory system two processors can observe inconsistent memory values.

The issue of memory consistency between multiple processors is discussed at length in many computer science papers. From a practical point of view, in order to deal with this issue, shared memory machines all provide some mechanism to enforce memory consistency when it is needed. The exact mechanism employed will vary between systems. For communication using shared memory, the WRAPPER provides a place to invoke the appropriate mechanism to ensure memory consistency for a particular platform.

4.2.6.2 Cache effects and false sharing

Shared-memory machines often have local to processor memory caches which contain mirrored copies of main memory. Automatic cache-coherence protocols are used to maintain consistency between caches on different processors. These cache-coherence protocols typically enforce consistency between regions of memory with large granularity (typically 128 or 256 byte chunks). The coherency protocols employed can be expensive relative to other memory accesses and so care is taken in the WRAPPER (by padding synchronization structures appropriately) to avoid unnecessary coherence traffic.

4.2.6.3 Operating system support for shared memory.

Applications running under multiple threads within a single process can use shared memory communication. In this case *all* the memory locations in an application are potentially visible to all the compute threads. Multiple threads operating within a single process is the standard mechanism for supporting shared memory that the WRAPPER utilizes. Configuring and launching code to run in multi-threaded mode on specific platforms is discussed in section 4.3.2.1. However, on many systems, potentially very efficient mechanisms for using shared memory communication between multiple processes (in contrast to multiple threads within a single process) also exist. In most cases this works by making a limited region of memory shared between processes. The MMAP and IPC facilities in UNIX systems provide this capability as do vendor specific tools like LAPI and IMC. Extensions exist for the WRAPPER that allow these mechanisms to be used for shared memory communication. However, these mechanisms are not distributed with the default WRAPPER sources, because of their proprietary nature.

4.2.7 Distributed memory communication

Many parallel systems are not constructed in a way where it is possible or practical for an application to use shared memory for communication. For example cluster systems consist of individual computers connected by a fast network. On such systems there is no notion of shared memory at the system level. For this sort of system the WRAPPER provides support for communication based on a bespoke communication library (see figure 4.6). The default communication library used is MPI *Message Passing Interface Forum* [1998]. However, it is relatively straightforward to implement bindings to optimized platform specific communication libraries. For example the work described in *Hoe et al.* [1999] substituted standard MPI communication for a highly optimized library.

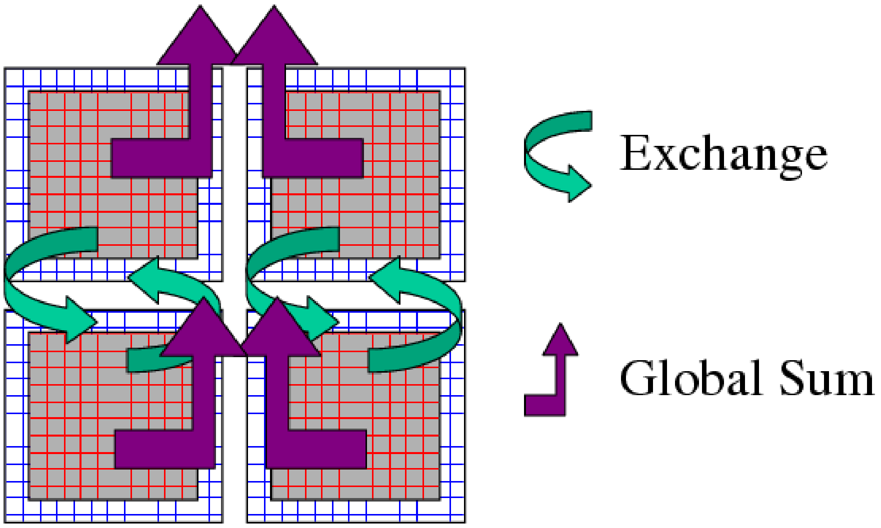


Figure 4.7: Three performance critical parallel primitives are provided by the WRAPPER. These primitives are always used to communicate data between tiles. The figure shows four tiles. The curved arrows indicate exchange primitives which transfer data between the overlap regions at tile edges and interior regions for nearest-neighbor tiles. The straight arrows symbolize global sum operations which connect all tiles. The global sum operation provides both a key arithmetic primitive and can serve as a synchronization primitive. A third barrier primitive is also provided, it behaves much like the global sum primitive.

4.2.8 Communication primitives

Optimized communication support is assumed to be potentially available for a small number of communication operations. It is also assumed that communication performance optimizations can be achieved by optimizing a small number of communication primitives. Three optimizable primitives are provided by the WRAPPER

- **EXCHANGE** This operation is used to transfer data between interior and overlap regions of neighboring tiles. A number of different forms of this operation are supported. These different forms handle
  - Data type differences. Sixty-four bit and thirty-two bit fields may be handled separately.
  - Bindings to different communication methods. Exchange primitives select between using shared memory or distributed memory communication.
  - Transformation operations required when transporting data between different grid regions. Transferring data between faces of a cube-sphere grid, for example, involves a rotation of vector components.
  - Forward and reverse mode computations. Derivative calculations require tangent linear and adjoint forms of the exchange primitives.
- **GLOBAL SUM** The global sum operation is a central arithmetic operation for the pressure inversion phase of the MITgcm algorithm. For certain configurations scaling can be highly sensitive to the performance of the global sum primitive. This operation is a collective operation involving all tiles of the simulated domain. Different forms of the global sum primitive exist for handling
  - Data type differences. Sixty-four bit and thirty-two bit fields may be handled separately.
  - Bindings to different communication methods. Exchange primitives select between using shared memory or distributed memory communication.

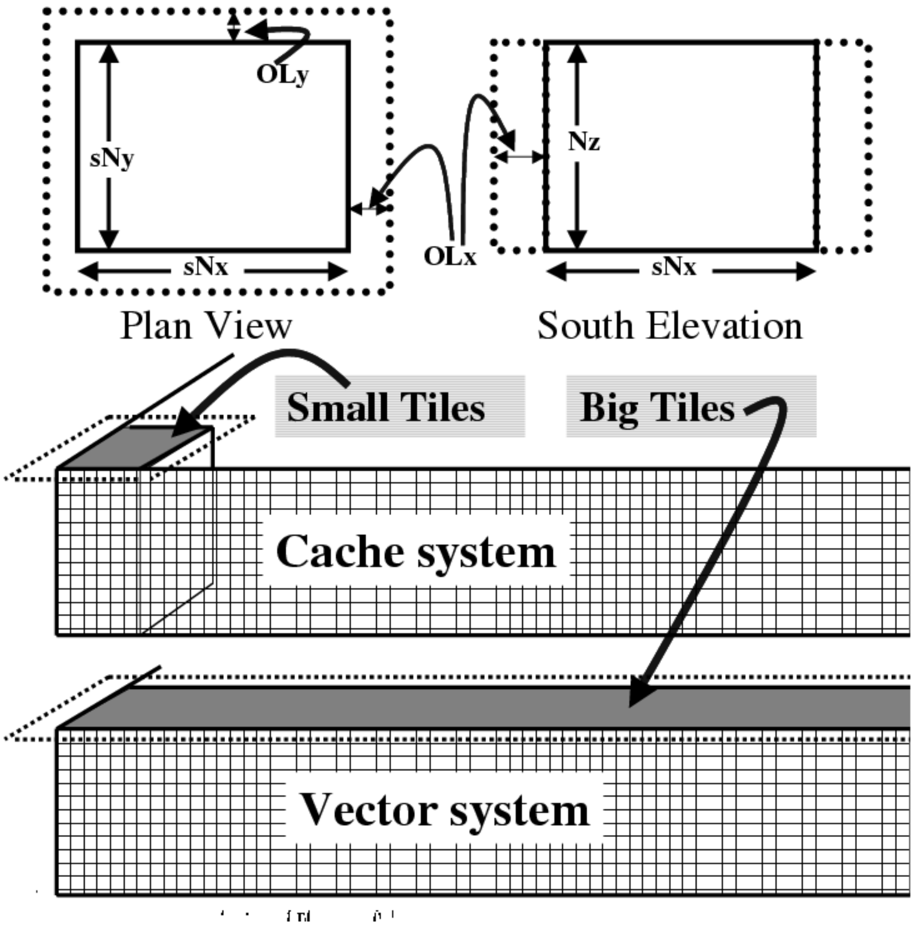


Figure 4.8: The tiling strategy that the WRAPPER supports allows tiles to be shaped to suit the underlying system memory architecture. Compact tiles that lead to greater memory reuse can be used on cache based systems (upper half of figure) with deep memory hierarchies, long tiles with large inner loops can be used to exploit vector systems having highly pipelined memory systems.

  - Forward and reverse mode computations. Derivative calculations require tangent linear and adjoint forms of the exchange primitives.

- **BARRIER** The WRAPPER provides a global synchronization function called barrier. This is used to synchronize computations over all tiles. The **BARRIER** and **GLOBAL SUM** primitives have much in common and in some cases use the same underlying code.

4.2.9 Memory architecture

The WRAPPER machine model is aimed to target efficiently systems with highly pipelined memory architectures and systems with deep memory hierarchies that favor memory reuse. This is achieved by supporting a flexible tiling strategy as shown in figure 4.8. Within a CPU computations are carried out sequentially on each tile in turn. By reshaping tiles according to the target platform it is possible to automatically tune code to improve memory performance. On a vector machine a given domain might be sub-divided into a few long, thin regions. On a commodity microprocessor based system, however, the same region could be simulated use many more smaller sub-domains.

4.2.10 Summary

Following the discussion above, the machine model that the WRAPPER presents to an application has the following characteristics

- The machine consists of one or more logical processors.
- Each processor operates on tiles that it owns.
- A processor may own more than one tile.
- Processors may compute concurrently.
- Exchange of information between tiles is handled by the machine (WRAPPER) not by the application.

Behind the scenes this allows the WRAPPER to adapt the machine model functions to exploit hardware on which

- Processors may be able to communicate very efficiently with each other using shared memory.
- An alternative communication mechanism based on a relatively simple inter-process communication API may be required.
- Shared memory may not necessarily obey sequential consistency, however some mechanism will exist for enforcing memory consistency.
- Memory consistency that is enforced at the hardware level may be expensive. Unnecessary triggering of consistency protocols should be avoided.
- Memory access patterns may need to either repetitive or highly pipelined for optimum hardware performance.

This generic model captures the essential hardware ingredients of almost all successful scientific computer systems designed in the last 50 years.

4.3 Using the WRAPPER

In order to support maximum portability the WRAPPER is implemented primarily in sequential Fortran 77. At a practical level the key steps provided by the WRAPPER are

1. specifying how a domain will be decomposed
2. starting a code in either sequential or parallel modes of operations
3. controlling communication between tiles and between concurrently computing CPUs.

This section describes the details of each of these operations. Section 4.3.1 explains how the way in which a domain is decomposed (or composed) is expressed. Section 4.3.2 describes practical details of running codes in various different parallel modes on contemporary computer systems. Section 4.3.3 explains the internal information that the WRAPPER uses to control how information is communicated between tiles.

4.3.1 Specifying a domain decomposition

At its heart much of the WRAPPER works only in terms of a collection of tiles which are interconnected to each other. This is also true of application code operating within the WRAPPER. Application code is written as a series of compute operations, each of which operates on a single tile. If application code needs to perform operations involving data associated with another tile, it uses a WRAPPER function to obtain that data. The specification of how a global domain is constructed from tiles or alternatively how a global domain is decomposed into tiles is made in the file *SIZE.h*. This file defines the following parameters

Parameters: *sNx, sNy, OLx, OLy, nSx, nSy, nPx, nPy* File: *model/inc/SIZE.h*

Together these parameters define a tiling decomposition of the style shown in figure 4.9. The parameters *sNx* and *sNy* define the size of an individual tile. The parameters *OLx* and *OLy* define the maximum size of the overlap extent. This must be set to the maximum width of the computation stencil that the numerical code finite-difference operations require between overlap region updates. The maximum overlap required by any of the operations in the MITgcm code distributed with this release is three grid points. This is set by the requirements of the ∇^4 dissipation and diffusion operator. Code modifications and enhancements that involve adding wide finite-difference stencils may require increasing *OLx* and *OLy*. Setting *OLx* and *OLy* to a too large value will decrease code performance (because redundant computations will be performed), however it will not cause any other problems.

The parameters *nSx* and *nSy* specify the number of tiles that will be created within a single process. Each of these tiles will have internal dimensions of *sNx* and *sNy*. If, when the code is executed, these tiles are allocated to different threads of a process that are then bound to different physical processors (see the multi-threaded execution discussion in section 4.3.2) then computation will be performed concurrently on each tile. However, it is also possible to run the same decomposition within a process running a single thread on a single processor. In this case the tiles will be computed over sequentially. If the decomposition is run in a single process running multiple threads but attached to a single physical processor, then, in general, the computation for different tiles will be interleaved by system level software. This too is a valid mode of operation.

The parameters *sNx, sNy, OLx, OLy, nSx* and*nSy* are used extensively by numerical code. The settings of *sNx, sNy, OLx* and *OLy* are used to form the loop ranges for many numerical calculations and to provide dimensions for arrays holding numerical state. The *nSx* and*nSy* are used in conjunction with the thread number parameter *myThid*. Much of the numerical code operating within the WRAPPER takes the form

```
DO bj=myByLo(myThid),myByHi(myThid)
 DO bi=myBxLo(myThid),myBxHi(myThid)
    :
    a block of computations ranging
    over 1,sNx +/- OLx and 1,sNy +/- OLy grid points
    :
 ENDDO
ENDDO

communication code to sum a number or maybe update
tile overlap regions

DO bj=myByLo(myThid),myByHi(myThid)
 DO bi=myBxLo(myThid),myBxHi(myThid)
    :
    another block of computations ranging
    over 1,sNx +/- OLx and 1,sNy +/- OLy grid points
    :
 ENDDO
ENDDO
```

The variables *myBxLo(myThid), myBxHi(myThid), myByLo(myThid)* and *myByHi(myThid)* set the bounds of the loops in *bi* and *bj* in this schematic. These variables specify the subset of the tiles in the range 1,*nSx* and 1,*nSy* that the logical processor bound to thread number *myThid* owns. The thread number variable *myThid* ranges from 1 to the total number of threads requested at execution time. For each value of *myThid* the loop scheme above will step sequentially through the tiles owned by that thread. However, different threads will have different ranges of tiles assigned to them, so that separate threads can compute iterations of the *bi*, *bj* loop concurrently. Within a *bi*, *bj* loop computation is performed concurrently over as many processes and threads as there are physical processors available to compute.

An exception to the the use of *bi* and *bj* in loops arises in the exchange routines used when the exch2 package is used with the cubed sphere. In this case *bj* is generally set to 1 and the loop runs from

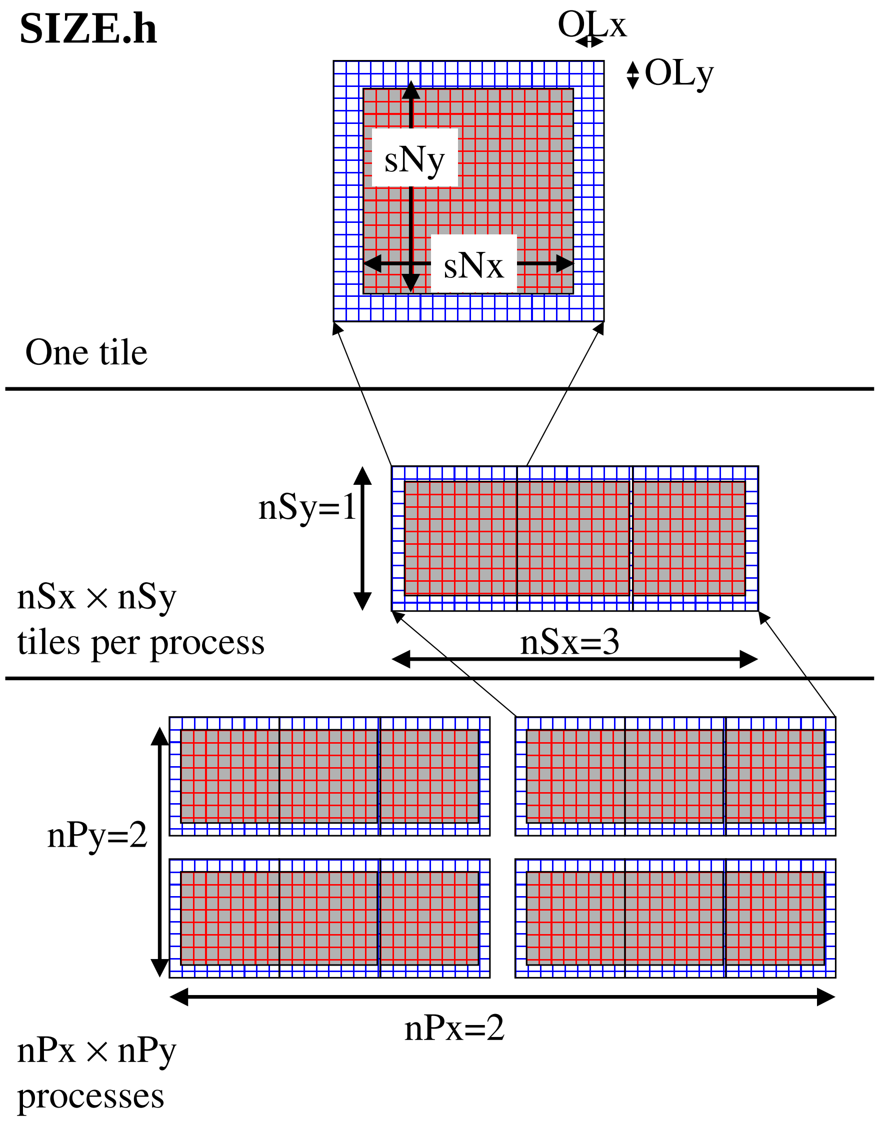


Figure 4.9: The three level domain decomposition hierarchy employed by the WRAPPER. A domain is composed of tiles. Multiple tiles can be allocated to a single process. Multiple processes can exist, each with multiple tiles. Tiles within a process can be spread over multiple compute threads.

```
      REAL*8  cg2d_r(1-OLx:sNx+OLx,1-OLy:sNy+OLy,nSx,nSy)
      REAL*8  err
          :
          :
        other computations
          :
          :
      err = 0.
      DO bj=myByLo(myThid),myByHi(myThid)
       DO bi=myBxLo(myThid),myBxHi(myThid)
        DO J=1,sNy
         DO I=1,sNx
           err            = err            +
     &     cg2d_r(I,J,bi,bj)*cg2d_r(I,J,bi,bj)
         ENDDO
        ENDDO
       ENDDO
      ENDDO

      CALL GLOBAL_SUM_R8( err   , myThid )
      err = SQRT(err)
```

Figure 4.10: Example of numerical code for calculating the l2-norm of a vector within the WRAPPER. Notice that under the WRAPPER arrays such as *cg2d_r* have two extra trailing dimensions. These right most indices are tile indexes. Different threads with a single process operate on different ranges of tile index, as controlled by the settings of *myByLo, myByHi, myBxLo* and *myBxHi.*

1,*bi*. Within the loop *bi* is used to retrieve the tile number, which is then used to reference exchange parameters.

The amount of computation that can be embedded a single loop over *bi* and *bj* varies for different parts of the MITgcm algorithm. Figure 4.10 shows a code extract from the two-dimensional implicit elliptic solver. This portion of the code computes the l2Norm of a vector whose elements are held in the array *cg2d_r* writing the final result to scalar variable *err*. In this case, because the l2norm requires a global reduction, the *bi*,*bj* loop only contains one statement. This computation phase is then followed by a communication phase in which all threads and processes must participate. However, in other areas of the MITgcm code entries subsections of code are within a single *bi*,*bj* loop. For example the evaluation of all the momentum equation prognostic terms (see *S/R DYNAMICS()*) is within a single *bi*,*bj* loop.

The final decomposition parameters are *nPx* and *nPy*. These parameters are used to indicate to the WRAPPER level how many processes (each with $nSx \times nSy$ tiles) will be used for this simulation. This information is needed during initialization and during I/O phases. However, unlike the variables *sNx, sNy, OLx, OLy, nSx* and *nSy* the values of *nPx* and *nPy* are absent from the core numerical and support code.

4.3.1.1 Examples of *SIZE.h* specifications

The following different *SIZE.h* parameter setting illustrate how to interpret the values of *sNx, sNy, OLx, OLy, nSx, nSy, nPx* and *nPy*.

1.
```
      PARAMETER (
     &           sNx =  90,
     &           sNy =  40,
     &           OLx =   3,
     &           OLy =   3,
     &           nSx =   1,
     &           nSy =   1,
```

```
     &           nPx =   1,
     &           nPy =   1)
```

This sets up a single tile with x-dimension of ninety grid points, y-dimension of forty grid points, and x and y overlaps of three grid points each.

2.
```
      PARAMETER (
     &           sNx =  45,
     &           sNy =  20,
     &           OLx =   3,
     &           OLy =   3,
     &           nSx =   1,
     &           nSy =   1,
     &           nPx =   2,
     &           nPy =   2)
```

This sets up tiles with x-dimension of forty-five grid points, y-dimension of twenty grid points, and x and y overlaps of three grid points each. There are four tiles allocated to four separate processes (*nPx=2,nPy=2*) and arranged so that the global domain size is again ninety grid points in x and forty grid points in y. In general the formula for global grid size (held in model variables *Nx* and *Ny*) is

```
                 Nx  = sNx*nSx*nPx
                 Ny  = sNy*nSy*nPy
```

3.
```
      PARAMETER (
     &           sNx =  90,
     &           sNy =  10,
     &           OLx =   3,
     &           OLy =   3,
     &           nSx =   1,
     &           nSy =   2,
     &           nPx =   1,
     &           nPy =   2)
```

This sets up tiles with x-dimension of ninety grid points, y-dimension of ten grid points, and x and y overlaps of three grid points each. There are four tiles allocated to two separate processes (*nPy=2*) each of which has two separate sub-domains *nSy=2*, The global domain size is again ninety grid points in x and forty grid points in y. The two sub-domains in each process will be computed sequentially if they are given to a single thread within a single process. Alternatively if the code is invoked with multiple threads per process the two domains in y may be computed concurrently.

4.
```
      PARAMETER (
     &           sNx =  32,
     &           sNy =  32,
     &           OLx =   3,
     &           OLy =   3,
     &           nSx =   6,
     &           nSy =   1,
     &           nPx =   1,
     &           nPy =   1)
```

This sets up tiles with x-dimension of thirty-two grid points, y-dimension of thirty-two grid points, and x and y overlaps of three grid points each. There are six tiles allocated to six separate logical processors (*nSx=6*). This set of values can be used for a cube sphere calculation. Each tile of size 32×32 represents a face of the cube. Initializing the tile connectivity correctly (see section 4.3.3.3. allows the rotations associated with moving between the six cube faces to be embedded within the tile-tile communication code.

```
MAIN
|
|--EEBOOT               :: WRAPPER initialization
|  |
|  |-- EEBOOT_MINMAL    :: Minimal startup. Just enough to
|  |                       allow basic I/O.
|  |-- EEINTRO_MSG      :: Write startup greeting.
|  |
|  |-- EESET_PARMS      :: Set WRAPPER parameters
|  |
|  |-- EEWRITE_EEENV    :: Print WRAPPER parameter settings
|  |
|  |-- INI_PROCS        :: Associate processes with grid regions.
|  |
|  |-- INI_THREADING_ENVIRONMENT   :: Associate threads with grid regions.
|       |
|       |--INI_COMMUNICATION_PATTERNS :: Initialize between tile
|                                     :: communication data structures
|
|
|--CHECK_THREADS    :: Validate multiple thread start up.
|
|--THE_MODEL_MAIN   :: Numerical code top-level driver routine
```

Figure 4.11: Main stages of the WRAPPER startup procedure. This process proceeds transfer of control to application code, which occurs through the procedure *THE_MODEL_MAIN()*.

4.3.2 Starting the code

When code is started under the WRAPPER, execution begins in a main routine *eesupp/src/main.F* that is owned by the WRAPPER. Control is transferred to the application through a routine called *THE_MODEL_MAIN()* once the WRAPPER has initialized correctly and has created the necessary variables to support subsequent calls to communication routines by the application code. The startup calling sequence followed by the WRAPPER is shown in figure 4.11.

4.3.2.1 Multi-threaded execution

Prior to transferring control to the procedure *THE_MODEL_MAIN()* the WRAPPER may cause several coarse grain threads to be initialized. The routine *THE_MODEL_MAIN()* is called once for each thread and is passed a single stack argument which is the thread number, stored in the variable *myThid*. In addition to specifying a decomposition with multiple tiles per process (see section 4.3.1) configuring and starting a code to run using multiple threads requires the following steps.

Compilation First the code must be compiled with appropriate multi-threading directives active in the file *main.F* and with appropriate compiler flags to request multi-threading support. The header files *MAIN_PDIRECTIVES1.h* and *MAIN_PDIRECTIVES2.h* contain directives compatible with compilers for Sun, Compaq, SGI, Hewlett-Packard SMP systems and CRAY PVP systems. These directives can be activated by using compile time directives *-DTARGET_SUN*, *-DTARGET_DEC*, *-DTARGET_SGI*, *-DTARGET_HP* or *-DTARGET_CRAY_VECTOR* respectively. Compiler options for invoking multi-threaded compilation vary from system to system and from compiler to compiler. The options will be described in the individual compiler documentation. For the Fortran compiler from Sun the following options are needed to correctly compile multi-threaded code

```
-stackvar -explicitpar -vpara -noautopar
```

These options are specific to the Sun compiler. Other compilers will use different syntax that will be described in their documentation. The effect of these options is as follows

1. **-stackvar** Causes all local variables to be allocated in stack storage. This is necessary for local variables to ensure that they are private to their thread. Note, when using this option it may be necessary to override the default limit on stack-size that the operating system assigns to a process. This can normally be done by changing the settings of the command shells *stack-size* limit variable. However, on some systems changing this limit will require privileged administrator access to modify system parameters.

2. **-explicitpar** Requests that multiple threads be spawned in response to explicit directives in the application code. These directives are inserted with syntax appropriate to the particular target platform when, for example, the *-DTARGET_SUN* flag is selected.

3. **-vpara** This causes the compiler to describe the multi-threaded configuration it is creating. This is not required but it can be useful when trouble shooting.

4. **-noautopar** This inhibits any automatic multi-threaded parallelization the compiler may otherwise generate.

An example of valid settings for the *eedata* file for a domain with two subdomains in y and running with two threads is shown below

```
nTx=1,nTy=2
```

This set of values will cause computations to stay within a single thread when moving across the *nSx* sub-domains. In the y-direction, however, sub-domains will be split equally between two threads.

Multi-threading files and parameters The following files and variables are used in setting up multi-threaded execution.

File: *eesupp/inc/MAIN_PDIRECTIVES1.h*
File: *eesupp/inc/MAIN_PDIRECTIVES2.h*
File: *model/src/THE_MODEL_MAIN.F*
File: *eesupp/src/MAIN.F*
File: *tools/genmake2*
File: *eedata*
CPP: *TARGET_SUN*
CPP: *TARGET_DEC*
CPP: *TARGET_HP*
CPP: *TARGET_SGI*
CPP: *TARGET_CRAY_VECTOR*
Parameter: *nTx*
Parameter: *nTy*

4.3.2.2 Multi-process execution

Despite its appealing programming model, multi-threaded execution remains less common than multi-process execution. One major reason for this is that many system libraries are still not "thread-safe". This means that, for example, on some systems it is not safe to call system routines to perform I/O when running in multi-threaded mode (except, perhaps, in a limited set of circumstances). Another reason is that support for multi-threaded programming models varies between systems.

Multi-process execution is more ubiquitous. In order to run code in a multi-process configuration a decomposition specification (see section 4.3.1) is given (in which the at least one of the parameters *nPx* or *nPy* will be greater than one) and then, as for multi-threaded operation, appropriate compile time and run time steps must be taken.

Compilation Multi-process execution under the WRAPPER assumes that the portable, MPI libraries are available for controlling the start-up of multiple processes. The MPI libraries are not required, although they are usually used, for performance critical communication. However, in order to simplify the task of controlling and coordinating the start up of a large number (hundreds and possibly even thousands) of copies of the same program, MPI is used. The calls to the MPI multi-process startup routines must be activated at compile time. Currently MPI libraries are invoked by specifying the appropriate options file with the `-of` flag when running the *genmake2* script, which generates the Makefile for compiling and linking MITgcm. (Previously this was done by setting the *ALLOW_USE_MPI* and *ALWAYS_USE_MPI* flags in the *CPP_EEOPTIONS.h* file.) More detailed information about the use of *genmake2* for specifying local compiler flags is located in section 3.4.2.

Directory: *tools/build_options* File: *tools/genmake2*

Execution The mechanics of starting a program in multi-process mode under MPI is not standardized. Documentation associated with the distribution of MPI installed on a system will describe how to start a program using that distribution. For the open-source MPICH system, the MITgcm program can be started using a command such as

```
mpirun -np 64 -machinefile mf ./mitgcmuv
```

In this example the text *-np 64* specifies the number of processes that will be created. The numeric value *64* must be equal to the product of the processor grid settings of *nPx* and *nPy* in the file *SIZE.h*. The parameter *mf* specifies that a text file called "mf" will be read to get a list of processor names on which the sixty-four processes will execute. The syntax of this file is specified by the MPI distribution.

File: *SIZE.h* Parameter: *nPx* Parameter: *nPy*

Environment variables On most systems multi-threaded execution also requires the setting of a special environment variable. On many machines this variable is called PARALLEL and its values should be set to the number of parallel threads required. Generally the help or manual pages associated with the multi-threaded compiler on a machine will explain how to set the required environment variables.

Runtime input parameters Finally the file *eedata* needs to be configured to indicate the number of threads to be used in the x and y directions. The variables nTx and nTy in this file are used to specify the information required. The product of nTx and nTy must be equal to the number of threads spawned i.e. the setting of the environment variable PARALLEL. The value of nTx must subdivide the number of sub-domains in x (*nSx*) exactly. The value of nTy must subdivide the number of sub-domains in y (*nSy*) exactly. The multiprocess startup of the MITgcm executable *mitgcmuv* is controlled by the routines *EEBOOT_MINIMAL()* and *INI_PROCS()*. The first routine performs basic steps required to make sure each process is started and has a textual output stream associated with it. By default two output files are opened for each process with names **STDOUT.NNNN** and **STDERR.NNNN**. The **NNNNN** part of the name is filled in with the process number so that process number 0 will create output files **STDOUT.0000** and **STDERR.0000**, process number 1 will create output files **STDOUT.0001** and **STDERR.0001**, etc. These files are used for reporting status and configuration information and for reporting error conditions on a process by process basis. The *EEBOOT_MINIMAL()* procedure also sets the variables *myProcId* and *MPI_COMM_MODEL*. These variables are related to processor identification are are used later in the routine *INI_PROCS()* to allocate tiles to processes.

Allocation of processes to tiles is controlled by the routine *INI_PROCS()*. For each process this routine sets the variables *myXGlobalLo* and *myYGlobalLo*. These variables specify, in index space, the coordinates of the southernmost and westernmost corner of the southernmost and westernmost tile owned by this process. The variables *pidW*, *pidE*, *pidS* and *pidN* are also set in this routine. These are used to identify processes holding tiles to the west, east, south and north of a given process. These values are stored in global storage in the header file *EESUPPORT.h* for use by communication routines. The above does not hold when the exch2 package is used. The exch2 sets its own parameters to specify the global indices of tiles and their relationships to each other. See the documentation on the exch2 package (6.2.4) for details.

File: *eesupp/src/eeboot_minimal.F*
File: *eesupp/src/ini_procs.F*
File: *eesupp/inc/EESUPPORT.h*
Parameter: *myProcId*
Parameter: *MPI_COMM_MODEL*
Parameter: *myXGlobalLo*
Parameter: *myYGlobalLo*
Parameter: *pidW*
Parameter: *pidE*
Parameter: *pidS*
Parameter: *pidN*

4.3.3 Controlling communication

The WRAPPER maintains internal information that is used for communication operations and that can be customized for different platforms. This section describes the information that is held and used.

1. **Tile-tile connectivity information** For each tile the WRAPPER sets a flag that sets the tile number to the north, south, east and west of that tile. This number is unique over all tiles in a configuration. Except when using the cubed sphere and the exch2 package, the number is held in the variables *tileNo* (this holds the tiles own number), *tileNoN*, *tileNoS*, *tileNoE* and *tileNoW*. A parameter is also stored with each tile that specifies the type of communication that is used between tiles. This information is held in the variables *tileCommModeN*, *tileCommModeS*, *tileCommModeE* and *tileCommModeW*. This latter set of variables can take one of the following values *COMM_NONE*, *COMM_MSG*, *COMM_PUT* and *COMM_GET*. A value of *COMM_NONE* is used to indicate that a tile has no neighbor to communicate with on a particular face. A value of *COMM_MSG* is used to indicate that some form of distributed memory communication is required to communicate between these tile faces (see section 4.2.7). A value of *COMM_PUT* or *COMM_GET* is used to indicate forms of shared memory communication (see section 4.2.6). The *COMM_PUT* value indicates that a CPU should communicate by writing to data structures owned by another CPU. A *COMM_GET* value indicates that a CPU should communicate by reading from data structures owned by another CPU. These flags affect the behavior of the WRAPPER exchange primitive (see figure 4.7). The routine *ini_communication_patterns()* is responsible for setting the communication mode values for each tile.

   When using the cubed sphere configuration with the exch2 package, the relationships between tiles and their communication methods are set by the exch2 package and stored in different variables. See the exch2 package documentation (6.2.4 for details.

   File: *eesupp/src/ini_communication_patterns.F*
   File: *eesupp/inc/EESUPPORT.h*
   Parameter: *tileNo*
   Parameter: *tileNoE*
   Parameter: *tileNoW*
   Parameter: *tileNoN*
   Parameter: *tileNoS*
   Parameter: *tileCommModeE*
   Parameter: *tileCommModeW*
   Parameter: *tileCommModeN*
   Parameter: *tileCommModeS*

2. **MP directives** The WRAPPER transfers control to numerical application code through the routine *THE_MODEL_MAIN*. This routine is called in a way that allows for it to be invoked by several threads. Support for this is based on either multi-processing (MP) compiler directives or specific calls to multi-threading libraries (*eg.* POSIX threads). Most commercially available Fortran

compilers support the generation of code to spawn multiple threads through some form of compiler directives. Compiler directives are generally more convenient than writing code to explicitly spawning threads. And, on some systems, compiler directives may be the only method available. The WRAPPER is distributed with template MP directives for a number of systems.

These directives are inserted into the code just before and after the transfer of control to numerical algorithm code through the routine *THE_MODEL_MAIN*. Figure 4.12 shows an example of the code that performs this process for a Silicon Graphics system. This code is extracted from the files *main.F* and *MAIN_PDIRECTIVES1.h*. The variable *nThreads* specifies how many instances of the routine *THE_MODEL_MAIN* will be created. The value of *nThreads* is set in the routine *INI_THREADING_ENVIRONMENT*. The value is set equal to the the product of the parameters *nTx* and *nTy* that are read from the file *eedata*. If the value of *nThreads* is inconsistent with the number of threads requested from the operating system (for example by using an environment variable as described in section 4.3.2.1) then usually an error will be reported by the routine *CHECK_THREADS*.

File: *eesupp/src/ini_threading_environment.F*
File: *eesupp/src/check_threads.F*
File: *eesupp/src/main.F*
File: *eesupp/inc/MAIN_PDIRECTIVES1.h*
File: *eedata*
Parameter: *nThreads*
Parameter: *nTx*
Parameter: *nTy*

3. **memsync flags** As discussed in section 4.2.6.1, a low-level system function may be need to force memory consistency on some shared memory systems. The routine *MEMSYNC()* is used for this purpose. This routine should not need modifying and the information below is only provided for completeness. A logical parameter *exchNeedsMemSync* set in the routine *INI_COMMUNICATION_PATTERNS()* controls whether the *MEMSYNC()* primitive is called. In general this routine is only used for multi-threaded execution. The code that goes into the *MEMSYNC()* routine is specific to the compiler and processor used. In some cases, it must be written using a short code snippet of assembly language. For an Ultra Sparc system the following code snippet is used

   ```
   asm("membar #LoadStore|#StoreStore");
   ```

   for an Alpha based system the equivalent code reads

   ```
   asm("mb");
   ```

   while on an x86 system the following code is required

   ```
   asm("lock; addl $0,0(%%esp)": : :"memory")
   ```

4. **Cache line size** As discussed in section 4.2.6.2, milti-threaded codes explicitly avoid penalties associated with excessive coherence traffic on an SMP system. To do this the shared memory data structures used by the *GLOBAL_SUM*, *GLOBAL_MAX* and *BARRIER* routines are padded. The variables that control the padding are set in the header file *EEPARAMS.h*. These variables are called *cacheLineSize*, *lShare1*, *lShare4* and *lShare8*. The default values should not normally need changing.

5. **_BARRIER** This is a CPP macro that is expanded to a call to a routine which synchronizes all the logical processors running under the WRAPPER. Using a macro here preserves flexibility to insert a specialized call in-line into application code. By default this resolves to calling the procedure *BARRIER()*. The default setting for the _BARRIER macro is given in the file *CPP_EEMACROS.h*.

6. **_GSUM** This is a CPP macro that is expanded to a call to a routine which sums up a floating point number over all the logical processors running under the WRAPPER. Using a macro

here provides extra flexibility to insert a specialized call in-line into application code. By default this resolves to calling the procedure *GLOBAL_SUM_R8()* (for 64-bit floating point operands) or *GLOBAL_SUM_R4()* (for 32-bit floating point operands). The default setting for the _GSUM macro is given in the file *CPP_EEMACROS.h*. The _GSUM macro is a performance critical operation, especially for large processor count, small tile size configurations. The custom communication example discussed in section 4.3.3.2 shows how the macro is used to invoke a custom global sum routine for a specific set of hardware.

7. **_EXCH** The _EXCH CPP macro is used to update tile overlap regions. It is qualified by a suffix indicating whether overlap updates are for two-dimensional (_EXCH_XY) or three dimensional (_EXCH_XYZ) physical fields and whether fields are 32-bit floating point (_EXCH_XY_R4, _EXCH_XYZ_R4) or 64-bit floating point (_EXCH_XY_R8, _EXCH_XYZ_R8). The macro mappings are defined in the header file *CPP_EEMACROS.h*. As with _GSUM, the _EXCH operation plays a crucial role in scaling to small tile, large logical and physical processor count configurations. The example in section 4.3.3.2 discusses defining an optimized and specialized form on the _EXCH operation.

   The _EXCH operation is also central to supporting grids such as the cube-sphere grid. In this class of grid a rotation may be required between tiles. Aligning the coordinate requiring rotation with the tile decomposition, allows the coordinate transformation to be embedded within a custom form of the _EXCH primitive. In these cases _EXCH is mapped to exch2 routines, as detailed in the exch2 package documentation 6.2.4.

8. **Reverse Mode** The communication primitives _EXCH and _GSUM both employ hand-written adjoint forms (or reverse mode) forms. These reverse mode forms can be found in the source code directory *pkg/autodiff*. For the global sum primitive the reverse mode form calls are to *GLOBAL_ADSUM_R4* and *GLOBAL_ADSUM_R8*. The reverse mode form of the exchange primitives are found in routines prefixed *ADEXCH*. The exchange routines make calls to the same low-level communication primitives as the forward mode operations. However, the routine argument *simulationMode* is set to the value *REVERSE_SIMULATION*. This signifies to the low-level routines that the adjoint forms of the appropriate communication operation should be performed.

9. **MAX_NO_THREADS** The variable *MAX_NO_THREADS* is used to indicate the maximum number of OS threads that a code will use. This value defaults to thirty-two and is set in the file *EEPARAMS.h*. For single threaded execution it can be reduced to one if required. The value is largely private to the WRAPPER and application code will not normally reference the value, except in the following scenario.

   For certain physical parametrization schemes it is necessary to have a substantial number of work arrays. Where these arrays are allocated in heap storage (for example COMMON blocks) multi-threaded execution will require multiple instances of the COMMON block data. This can be achieved using a Fortran 90 module construct. However, if this mechanism is unavailable then the work arrays can be extended with dimensions using the tile dimensioning scheme of *nSx* and *nSy* (as described in section 4.3.1). However, if the configuration being specified involves many more tiles than OS threads then it can save memory resources to reduce the variable *MAX_NO_THREADS* to be equal to the actual number of threads that will be used and to declare the physical parameterization work arrays with a single *MAX_NO_THREADS* extra dimension. An example of this is given in the verification experiment *aim.5l_cs*. Here the default setting of *MAX_NO_THREADS* is altered to

   ```
   INTEGER MAX_NO_THREADS
   PARAMETER ( MAX_NO_THREADS =    6 )
   ```

   and several work arrays for storing intermediate calculations are created with declarations of the form.

   ```
   common /FORCIN/ sst1(ngp,MAX_NO_THREADS)
   ```

```
C--
C--  Parallel directives for MIPS Pro Fortran compiler
C--
C      Parallel compiler directives for SGI with IRIX
C$PAR  PARALLEL DO
C$PAR&  CHUNK=1,MP_SCHEDTYPE=INTERLEAVE,
C$PAR&  SHARE(nThreads),LOCAL(myThid,I)
C
      DO I=1,nThreads
        myThid = I

C--     Invoke nThreads instances of the numerical model
        CALL THE_MODEL_MAIN(myThid)

      ENDDO
```

Figure 4.12: Prior to transferring control to the procedure *THE_MODEL_MAIN()* the WRAPPER may use MP directives to spawn multiple threads.

> This declaration scheme is not used widely, because most global data is used for permanent not temporary storage of state information. In the case of permanent state information this approach cannot be used because there has to be enough storage allocated for all tiles. However, the technique can sometimes be a useful scheme for reducing memory requirements in complex physical parameterizations.

4.3.3.1 Specializing the Communication Code

The isolation of performance critical communication primitives and the sub-division of the simulation domain into tiles is a powerful tool. Here we show how it can be used to improve application performance and how it can be used to adapt to new griding approaches.

4.3.3.2 JAM example

On some platforms a big performance boost can be obtained by binding the communication routines *_EXCH* and *_GSUM* to specialized native libraries (for example, the shmem library on CRAY T3E systems). The *LETS_MAKE_JAM* CPP flag is used as an illustration of a specialized communication configuration that substitutes for standard, portable forms of *_EXCH* and *_GSUM*. It affects three source files *eeboot.F*, *CPP_EEMACROS.h* and *cg2d.F*. When the flag is defined is has the following effects.

- An extra phase is included at boot time to initialize the custom communications library (see *ini_jam.F*).
- The *_GSUM* and *_EXCH* macro definitions are replaced with calls to custom routines (see *gsum_jam.F* and *exch_jam.F*)
- a highly specialized form of the exchange operator (optimized for overlap regions of width one) is substituted into the elliptic solver routine *cg2d.F*.

Developing specialized code for other libraries follows a similar pattern.

4.3.3.3 Cube sphere communication

Actual *_EXCH* routine code is generated automatically from a series of template files, for example *exch_rx.template*. This is done to allow a large number of variations on the exchange process to be maintained. One set of variations supports the cube sphere grid. Support for a cube sphere grid in MITgcm is based on having each face of the cube as a separate tile or tiles. The exchange routines are then able to absorb much of the detailed rotation and reorientation required when moving around the cube grid. The set of *_EXCH* routines that contain the word cube in their name perform these transformations. They are invoked when the run-time logical parameter *useCubedSphereExchange* is set true. To facilitate the transformations on a staggered C-grid, exchange operations are defined separately for both vector and scalar quantities and for grid-centered and for grid-face and grid-corner quantities. Three sets of exchange routines are defined. Routines with names of the form *exch_rx* are used to exchange cell centered scalar quantities. Routines with names of the form *exch_uv_rx* are used to exchange vector quantities located at the C-grid velocity points. The vector quantities exchanged by the *exch_uv_rx* routines can either be signed (for example velocity components) or un-signed (for example grid-cell separations). Routines with names of the form *exch_z_rx* are used to exchange quantities at the C-grid vorticity point locations.

4.4 MITgcm execution under WRAPPER

Fitting together the WRAPPER elements, package elements and MITgcm core equation elements of the source code produces calling sequence shown in section 4.4.1

4.4.1 Annotated call tree for MITgcm and WRAPPER

WRAPPER layer.

```
MAIN
|
|--EEBOOT               :: WRAPPER initialization
|  |
|  |-- EEBOOT_MINMAL    :: Minimal startup. Just enough to
|  |                       allow basic I/O.
|  |-- EEINTRO_MSG      :: Write startup greeting.
|  |
|  |-- EESET_PARMS      :: Set WRAPPER parameters
|  |
|  |-- EEWRITE_EEENV    :: Print WRAPPER parameter settings
|  |
|  |-- INI_PROCS        :: Associate processes with grid regions.
|  |
|  |-- INI_THREADING_ENVIRONMENT   :: Associate threads with grid regions.
|       |
|       |--INI_COMMUNICATION_PATTERNS :: Initialize between tile
|                                     :: communication data structures
|
|
|--CHECK_THREADS    :: Validate multiple thread start up.
|
|--THE_MODEL_MAIN   :: Numerical code top-level driver routine
```

Core equations plus packages.

```
C
C Invocation from WRAPPER level...
C  :
C  :
C  |
C  |-THE_MODEL_MAIN :: Primary driver for the MITgcm algorithm
C    |              :: Called from WRAPPER level numerical
C    |              :: code invocation routine. On entry
C    |              :: to THE_MODEL_MAIN separate thread and
C    |              :: separate processes will have been established.
C    |              :: Each thread and process will have a unique ID
C    |              :: but as yet it will not be associated with a
C    |              :: specific region in decomposed discrete space.
C    |
C    |-INITIALISE_FIXED :: Set fixed model arrays such as topography,
C    | |                :: grid, solver matrices etc..
C    | |
C    | |-INI_PARMS :: Routine to set kernel model parameters.
C    | |           :: By default kernel parameters are read from file
```

```
C     | |           :: "data" in directory in which code executes.
C     | |
C     | |-MON_INIT :: Initializes monitor package ( see pkg/monitor )
C     | |
C     | |-INI_GRID :: Control grid array (vert. and hori.) initialization.
C     | | |        :: Grid arrays are held and described in GRID.h.
C     | | |
C     | | |-INI_VERTICAL_GRID        :: Initialize vertical grid arrays.
C     | | |
C     | | |-INI_CARTESIAN_GRID       :: Cartesian horiz. grid initialization
C     | | |                          :: (calculate grid from kernel parameters).
C     | | |
C     | | |-INI_SPHERICAL_POLAR_GRID :: Spherical polar horiz. grid
C     | | |                          :: initialization (calculate grid from
C     | | |                          :: kernel parameters).
C     | | |
C     | | |-INI_CURVILINEAR_GRID     :: General orthogonal, structured horiz.
C     | |                            :: grid initializations. ( input from raw
C     | |                            :: grid files, LONC.bin, DXF.bin etc... )
C     | |
C     | |-INI_DEPTHS    :: Read (from "bathyFile") or set bathymetry/orgography.
C     | |
C     | |-INI_MASKS_ETC :: Derive horizontal and vertical cell fractions and
C     | |               :: land masking for solid-fluid boundaries.
C     | |
C     | |-INI_LINEAR_PHSURF :: Set ref. surface Bo_surf
C     | |
C     | |-INI_CORI          :: Set coriolis term. zero, f-plane, beta-plane,
C     | |                   :: sphere options are coded.
C     | |
C     | |-PACAKGES_BOOT      :: Start up the optional package environment.
C     | |                    :: Runtime selection of active packages.
C     | |
C     | |-PACKAGES_READPARMS :: Call active package internal parameter load.
C     | | |
C     | | |-GMREDI_READPARMS    :: GM Package. see pkg/gmredi
C     | | |-KPP_READPARMS       :: KPP Package. see pkg/kpp
C     | | |-SHAP_FILT_READPARMS :: Shapiro filter package. see pkg/shap_filt
C     | | |-OBCS_READPARMS      :: Open bndy package. see pkg/obcs
C     | | |-AIM_READPARMS       :: Intermediate Atmos. pacakage. see pkg/aim
C     | | |-COST_READPARMS      :: Cost function package. see pkg/cost
C     | | |-CTRL_INIT           :: Control vector support package. see pkg/ctrl
C     | | |-OPTIM_READPARMS     :: Optimisation support package. see pkg/ctrl
C     | | |-GRDCHK_READPARMS    :: Gradient check package. see pkg/grdchk
C     | | |-ECCO_READPARMS      :: ECCO Support Package. see pkg/ecco
C     | | |-PTRACERS_READPARMS  :: multiple tracer package, see pkg/ptracers
C     | | |-GCHEM_READPARMS     :: tracer interface package, see pkg/gchem
C     | |
C     | |-PACKAGES_CHECK
C     | | |
C     | | |-KPP_CHECK           :: KPP Package. pkg/kpp
C     | | |-OBCS_CHECK          :: Open bndy Pacakge. pkg/obcs
C     | | |-GMREDI_CHECK        :: GM Package. pkg/gmredi
C     | |
C     | |-PACKAGES_INIT_FIXED
C     | | |-OBCS_INIT_FIXED     :: Open bndy Package. see pkg/obcs
C     | | |-FLT_INIT            :: Floats Package. see pkg/flt
C     | | |-GCHEM_INIT_FIXED    :: tracer interface pachage, see pkg/gchem
C     | |
C     | |-ZONAL_FILT_INIT       :: FFT filter Package. see pkg/zonal_filt
C     | |
C     | |-INI_CG2D              :: 2d con. grad solver initialization.
C     | |
C     | |-INI_CG3D              :: 3d con. grad solver initialization.
C     | |
C     | |-CONFIG_SUMMARY        :: Provide synopsis of kernel setup.
C     |                         :: Includes annotated table of kernel
C     |                         :: parameter settings.
C     |
C     |-CTRL_UNPACK :: Control vector support package. see pkg/ctrl
C     |
```

```
C    |-ADTHE_MAIN_LOOP :: Derivative evaluating form of main time stepping loop
C    !                 :: Auotmatically generated by TAMC/TAF.
C    |
C    |-CTRL_PACK   :: Control vector support package. see pkg/ctrl
C    |
C    |-GRDCHK_MAIN :: Gradient check package. see pkg/grdchk
C    |
C    |-THE_MAIN_LOOP :: Main timestepping loop routine.
C    | |
C    | |-INITIALISE_VARIA :: Set the initial conditions for time evolving
C    | | |                :: variables
C    | | |
C    | | |-INI_LINEAR_PHISURF :: Set ref. surface Bo_surf
C    | | |
C    | | |-INI_CORI     :: Set coriolis term. zero, f-plane, beta-plane,
C    | | |              :: sphere options are coded.
C    | | |
C    | | |-INI_CG2D     :: 2d con. grad solver initialization.
C    | | |-INI_CG3D     :: 3d con. grad solver initialization.
C    | | |-INI_MIXING   :: Initialize diapycnal diffusivity.
C    | | |-INI_DYNVARS  :: Initialize to zero all DYNVARS.h arrays (dynamical
C    | | |              :: fields).
C    | | |
C    | | |-INI_FIELDS   :: Control initializing model fields to non-zero
C    | | | |-INI_VEL    :: Initialize 3D flow field.
C    | | | |-INI_THETA  :: Set model initial temperature field.
C    | | | |-INI_SALT   :: Set model initial salinity field.
C    | | | |-INI_PSURF  :: Set model initial free-surface height/pressure.
C    | | | |-INI_PRESSURE :: Compute model initial hydrostatic pressure
C    | | | |-READ_CHECKPOINT :: Read the checkpoint
C    | | |
C    | | |-THE_CORRECTION_STEP :: Step forward to next time step.
C    | | | |                   :: Here applied to move restart conditions
C    | | | |                   :: (saved in mid timestep) to correct level in
C    | | | |                   :: time (only used for pre-c35).
C    | | | |
C    | | | |-CALC_GRAD_PHI_SURF :: Return DDx and DDy of surface pressure
C    | | | |-CORRECTION_STEP    :: Pressure correction to momentum
C    | | | |-CYCLE_TRACER       :: Move tracers forward in time.
C    | | | |-OBCS_APPLY         :: Open bndy package. see pkg/obcs
C    | | | |-SHAP_FILT_APPLY    :: Shapiro filter package. see pkg/shap_filt
C    | | | |-ZONAL_FILT_APPLY   :: FFT filter package. see pkg/zonal_filt
C    | | | |-CONVECTIVE_ADJUSTMENT :: Control static instability mixing.
C    | | | | |-FIND_RHO  :: Find adjacent densities.
C    | | | | |-CONVECT   :: Mix static instability.
C    | | | | |-TIMEAVE_CUMULATE :: Update convection statistics.
C    | | | |
C    | | | |-CALC_EXACT_ETA        :: Change SSH to flow divergence.
C    | | |
C    | | |-CONVECTIVE_ADJUSTMENT_INI :: Control static instability mixing
C    | | | |                         :: Extra time history interactions.
C    | | | |
C    | | | |-FIND_RHO  :: Find adjacent densities.
C    | | | |-CONVECT   :: Mix static instability.
C    | | | |-TIMEAVE_CUMULATE :: Update convection statistics.
C    | | |
C    | | |-PACKAGES_INIT_VARIABLES :: Does initialization of time evolving
C    | | | |                       :: package data.
C    | | | |
C    | | | |-GMREDI_INIT          :: GM package. ( see pkg/gmredi )
C    | | | |-KPP_INIT             :: KPP package. ( see pkg/kpp )
C    | | | |-KPP_OPEN_DIAGS
C    | | | |-OBCS_INIT_VARIABLES  :: Open bndy. package. ( see pkg/obcs )
C    | | | |-PTRACERS_INIT        :: multi. tracer package,(see pkg/ptracers)
C    | | | |-GCHEM_INIT           :: tracer interface pkg (see pkh/gchem)
C    | | | |-AIM_INIT             :: Interm. atmos package. ( see pkg/aim )
C    | | | |-CTRL_MAP_INI         :: Control vector package.( see pkg/ctrl )
C    | | | |-COST_INIT            :: Cost function package. ( see pkg/cost )
C    | | | |-ECCO_INIT            :: ECCO support package. ( see pkg/ecco )
C    | | | |-INI_FORCING          :: Set model initial forcing fields.
C    | | |   |                    :: Either set in-line or from file as shown.
```

```
C    | | |   |-READ_FLD_XY_RS(zonalWindFile)
C    | | |   |-READ_FLD_XY_RS(meridWindFile)
C    | | |   |-READ_FLD_XY_RS(surfQFile)
C    | | |   |-READ_FLD_XY_RS(EmPmRfile)
C    | | |   |-READ_FLD_XY_RS(thetaClimFile)
C    | | |   |-READ_FLD_XY_RS(saltClimFile)
C    | | |   |-READ_FLD_XY_RS(surfQswFile)
C    | | |
C    | | |-CALC_SURF_DR   :: Calculate the new surface level thickness.
C    | | |-UPDATE_SURF_DR :: Update the surface-level thickness fraction.
C    | | |-UPDATE_CG2D    :: Update 2d conjugate grad. for Free-Surf.
C    | | |-STATE_SUMMARY    :: Summarize model prognostic variables.
C    | | |-TIMEAVE_STATVARS :: Time averaging package ( see pkg/timeave ).
C    | |
C    | |-WRITE_STATE      :: Controlling routine for IO to dump model state.
C    | | |-WRITE_REC_XYZ_RL :: Single file I/O
C    | | |-WRITE_FLD_XYZ_RL :: Multi-file I/O
C    | |
C    | |-MONITOR          :: Monitor state ( see pkg/monitor )
C    | |-CTRL_MAP_FORCING :: Control vector support package. ( see pkg/ctrl )
C====|>|
C====|>| ****************************
C====|>| BEGIN MAIN TIMESTEPPING LOOP
C====|>| ****************************
C====|>|
C/\  | |-FORWARD_STEP     :: Step forward a time-step ( AT LAST !!! )
C/\  | | |
C/\  | | |-DUMMY_IN_STEPPING :: autodiff package ( pkg/autoduff ).
C/\  | | |-CALC_EXACT_ETA :: Change SSH to flow divergence.
C/\  | | |-CALC_SURF_DR   :: Calculate the new surface level thickness.
C/\  | | |-EXF_GETFORCING :: External forcing package. ( pkg/exf )
C/\  | | |-EXTERNAL_FIELDS_LOAD :: Control loading time dep. external data.
C/\  | | | |                    :: Simple interpolation between end-points
C/\  | | | |                    :: for forcing datasets.
C/\  | | | |
C/\  | | | |-EXCH :: Sync forcing. in overlap regions.
C/\  | | |-SEAICE_MODEL   :: Compute sea-ice terms. ( pkg/seaice )
C/\  | | |-FREEZE         :: Limit surface temperature.
C/\  | | |-GCHEM_FIELD_LOAD :: load tracer forcing fields (pkg/gchem)
C/\  | | |
C/\  | | |-THERMODYNAMICS :: theta, salt + tracer equations driver.
C/\  | | | |
C/\  | | | |-INTEGRATE_FOR_W :: Integrate for vertical velocity.
C/\  | | | |-OBCS_APPLY_W    :: Open bndy. package ( see pkg/obcs ).
C/\  | | | |-FIND_RHO        :: Calculates [rho(S,T,z)-RhoConst] of a slice
C/\  | | | |-GRAD_SIGMA      :: Calculate isoneutral gradients
C/\  | | | |-CALC_IVDC       :: Set Implicit Vertical Diffusivity for Convection
C/\  | | | |
C/\  | | | |-OBCS_CALC            :: Open bndy. package ( see pkg/obcs ).
C/\  | | | |-EXTERNAL_FORCING_SURF:: Accumulates appropriately dimensioned
C/\  | | | | |                    :: forcing terms.
C/\  | | | | |-PTRACERS_FORCING_SURF :: Tracer package ( see pkg/ptracers ).
C/\  | | | |
C/\  | | | |-GMREDI_CALC_TENSOR   :: GM package ( see pkg/gmredi ).
C/\  | | | |-GMREDI_CALC_TENSOR_DUMMY :: GM package ( see pkg/gmredi ).
C/\  | | | |-KPP_CALC             :: KPP package ( see pkg/kpp ).
C/\  | | | |-KPP_CALC_DUMMY       :: KPP package ( see pkg/kpp ).
C/\  | | | |-AIM_DO_ATMOS_PHYSICS :: Intermed. atmos package ( see pkg/aim ).
C/\  | | | |-GAD_ADVECTION        :: Generalised advection driver (multi-dim
C/\  | | | |                          advection case) (see pkg/gad).
C/\  | | | |-CALC_COMMON_FACTORS  :: Calculate common data (such as volume flux)
C/\  | | | |-CALC_DIFFUSIVITY     :: Calculate net vertical diffusivity
C/\  | | | | |
C/\  | | | | |-GMREDI_CALC_DIFF   :: GM package ( see pkg/gmredi ).
C/\  | | | | |-KPP_CALC_DIFF      :: KPP package ( see pkg/kpp ).
C/\  | | | |
C/\  | | | |-CALC_GT              :: Calculate the temperature tendency terms
C/\  | | | | |
C/\  | | | | |-GAD_CALC_RHS       :: Generalised advection package
C/\  | | | | | |                  :: ( see pkg/gad )
C/\  | | | | | |-KPP_TRANSPORT_T  :: KPP non-local transport ( see pkg/kpp ).
```

```
C/\  | | | | |
C/\  | | | | |-EXTERNAL_FORCING_T :: Problem specific forcing for temperature.
C/\  | | | | |-ADAMS_BASHFORTH2   :: Extrapolate tendencies forward in time.
C/\  | | | | |-FREESURF_RESCALE_G :: Re-scale Gt for free-surface height.
C/\  | | | |
C/\  | | | |-TIMESTEP_TRACER      :: Step tracer field forward in time
C/\  | | | |
C/\  | | | |-CALC_GS              :: Calculate the salinity tendency terms
C/\  | | | | |
C/\  | | | | |-GAD_CALC_RHS       :: Generalised advection package
C/\  | | | | | |                  :: ( see pkg/gad )
C/\  | | | | | |-KPP_TRANSPORT_S  :: KPP non-local transport ( see pkg/kpp ).
C/\  | | | | |
C/\  | | | | |-EXTERNAL_FORCING_S :: Problem specific forcing for salt.
C/\  | | | | |-ADAMS_BASHFORTH2   :: Extrapolate tendencies forward in time.
C/\  | | | | |-FREESURF_RESCALE_G :: Re-scale Gs for free-surface height.
C/\  | | | |
C/\  | | | |-TIMESTEP_TRACER      :: Step tracer field forward in time
C/\  | | | |
C/\  | | | |-TIMESTEP_TRACER      :: Step tracer field forward in time
C/\  | | | |
C/\  | | | |-PTRACERS_INTEGRATE   :: Integrate other tracer(s) (see pkg/ptracers).
C/\  | | | | |
C/\  | | | | |-GAD_CALC_RHS       :: Generalised advection package
C/\  | | | | | |                  :: ( see pkg/gad )
C/\  | | | | | |-KPP_TRANSPORT_PTR:: KPP non-local transport ( see pkg/kpp ).
C/\  | | | | |
C/\  | | | | |-PTRACERS_FORCING   :: Problem specific forcing for tracer.
C/\  | | | | |-GCHEM_FORCING_INT  :: tracer forcing for gchem pkg (if all
C/\  | | | | |                        tendancy terms calcualted together)
C/\  | | | | |-ADAMS_BASHFORTH2   :: Extrapolate tendencies forward in time.
C/\  | | | | |-FREESURF_RESCALE_G :: Re-scale Gs for free-surface height.
C/\  | | | | |-TIMESTEP_TRACER    :: Step tracer field forward in time
C/\  | | | |
C/\  | | | |-OBCS_APPLY_TS        :: Open bndy. package (see pkg/obcs ).
C/\  | | | |
C/\  | | | |-IMPLDIFF             :: Solve vertical implicit diffusion equation.
C/\  | | | |-OBCS_APPLY_TS        :: Open bndy. package (see pkg/obcs ).
C/\  | | | |
C/\  | | | |-AIM_AIM2DYN_EXCHANGES :: Inetermed. atmos (see pkg/aim).
C/\  | | | |-EXCH                 :: Update overlaps
C/\  | | |
C/\  | | |-DYNAMICS       :: Momentum equations driver.
C/\  | | | |
C/\  | | | |-CALC_GRAD_PHI_SURF :: Calculate the gradient of the surface
C/\  | | | |                      Potential anomaly.
C/\  | | | |-CALC_VISCOSITY     :: Calculate net vertical viscosity
C/\  | | | | |-KPP_CALC_VISC    :: KPP package ( see pkg/kpp ).
C/\  | | | |
C/\  | | | |-CALC_PHI_HYD       :: Integrate the hydrostatic relation.
C/\  | | | |-MOM_FLUXFORM       :: Flux form mom eqn. package ( see
C/\  | | | |                      pkg/mom_fluxform ).
C/\  | | | |-MOM_VECINV         :: Vector invariant form mom eqn. package ( see
C/\  | | | |                      pkg/mom_vecinv   ).
C/\  | | | |-TIMESTEP           :: Step momentum fields forward in time
C/\  | | | |-OBCS_APPLY_UV      :: Open bndy. package (see pkg/obcs ).
C/\  | | | |
C/\  | | | |-IMPLDIFF           :: Solve vertical implicit diffusion equation.
C/\  | | | |-OBCS_APPLY_UV      :: Open bndy. package (see pkg/obcs ).
C/\  | | | |
C/\  | | | |-TIMEAVE_CUMUL_1T   :: Time averaging package ( see pkg/timeave ).
C/\  | | | |-TIMEAVE_CUMUATE    :: Time averaging package ( see pkg/timeave ).
C/\  | | | |-DEBUG_STATS_RL     :: Quick debug package ( see pkg/debug ).
C/\  | | |
C/\  | | |-CALC_GW        :: vert. momentum tendency terms ( NH, QH only ).
C/\  | | |
C/\  | | |-UPDATE_SURF_DR :: Update the surface-level thickness fraction.
C/\  | | |
C/\  | | |-UPDATE_CG2D    :: Update 2d conjugate grad. for Free-Surf.
C/\  | | |
C/\  | | |-SOLVE_FOR_PRESSURE           :: Find surface pressure.
```

```
C/\  | | | |-CALC_DIV_GHAT     :: Form the RHS of the surface pressure eqn.
C/\  | | | |-CG2D              :: Two-dim pre-con. conjugate-gradient.
C/\  | | | |-CG3D              :: Three-dim pre-con. conjugate-gradient solver.
C/\  | | |
C/\  | | |-THE_CORRECTION_STEP          :: Step forward to next time step.
C/\  | | | |
C/\  | | | |-CALC_GRAD_PHI_SURF :: Return DDx and DDy of surface pressure
C/\  | | | |-CORRECTION_STEP    :: Pressure correction to momentum
C/\  | | | |-CYCLE_TRACER       :: Move tracers forward in time.
C/\  | | | |-OBCS_APPLY         :: Open bndy package. see pkg/obcs
C/\  | | | |-SHAP_FILT_APPLY    :: Shapiro filter package. see pkg/shap_filt
C/\  | | | |-ZONAL_FILT_APPLY   :: FFT filter package. see pkg/zonal_filt
C/\  | | | |-CONVECTIVE_ADJUSTMENT :: Control static instability mixing.
C/\  | | | | |-FIND_RHO  :: Find adjacent densities.
C/\  | | | | |-CONVECT   :: Mix static instability.
C/\  | | | | |-TIMEAVE_CUMULATE :: Update convection statistics.
C/\  | | | |
C/\  | | | |-CALC_EXACT_ETA        :: Change SSH to flow divergence.
C/\  | | |
C/\  | | |-DO_FIELDS_BLOCKING_EXCHANGES :: Sync up overlap regions.
C/\  | | | |-EXCH
C/\  | | |
C/\  | | |-GCHEM_FORCING_SEP :: tracer forcing for gchem pkg (if
C/\  | | |                      tracer dependent tendencies calculated
C/\  | | |                      separatly)
C/\  | | |
C/\  | | |-FLT_MAIN         :: Float package ( pkg/flt ).
C/\  | | |
C/\  | | |-MONITOR          :: Monitor package ( pkg/monitor ).
C/\  | | |
C/\  | | |-DO_THE_MODEL_IO  :: Standard diagnostic I/O.
C/\  | | | |-WRITE_STATE    :: Core state I/O
C/\  | | | |-TIMEAVE_STATV_WRITE :: Time averages. see pkg/timeave
C/\  | | | |-AIM_WRITE_DIAGS     :: Intermed. atmos diags. see pkg/aim
C/\  | | | |-GMREDI_DIAGS        :: GM diags. see pkg/gmredi
C/\  | | | |-KPP_DO_DIAGS        :: KPP diags. see pkg/kpp
C/\  | | | |-SBO_CALC            :: SBO diags. see pkg/sbo
C/\  | | | |-SBO_DIAGS           :: SBO diags. see pkg/sbo
C/\  | | | |-SEAICE_DO_DIAGS     :: SEAICE diags. see pkg/seaice
C/\  | | | |-GCHEM_DIAGS         :: gchem diags. see pkg/gchem
C/\  | | |
C/\  | | |-WRITE_CHECKPOINT :: Do I/O for restart files.
C/\  | |
C/\  | |-COST_TILE        :: Cost function package. ( see pkg/cost )
C<===|=|
C<===|=| **************************
C<===|=| END MAIN TIMESTEPPING LOOP
C<===|=| **************************
C<===|=|
C    | |-COST_FINAL       :: Cost function package. ( see pkg/cost )
C    |
C    |-WRITE_CHECKPOINT :: Final state storage, for restart.
C    |
C    |-TIMER_PRINTALL :: Computational timing summary
C    |
C    |-COMM_STATS     :: Summarise inter-proc and inter-thread communication
C                     :: events.
C
```

4.4.2 Measuring and Characterizing Performance

TO BE DONE (CNH)

4.4.3 Estimating Resource Requirements

TO BE DONE (CNH)

4.4.3.1 Atlantic 1/6 degree example

4.4.3.2 Dry Run testing

4.4.3.3 Adjoint Resource Requirements

4.4.3.4 State Estimation Environment Resources

Chapter 5

Automatic Differentiation

Author: Patrick Heimbach

Automatic differentiation (AD), also referred to as algorithmic (or, more loosely, computational) differentiation, involves automatically deriving code to calculate partial derivatives from an existing fully non-linear prognostic code. (see *Griewank* [2000]). A software tool is used that parses and transforms source files according to a set of linguistic and mathematical rules. AD tools are like source-to-source translators in that they parse a program code as input and produce a new program code as output (we restrict our discussion to source-to-source tools, ignoring operator-overloading tools). However, unlike a pure source-to-source translation, the output program represents a new algorithm, such as the evaluation of the Jacobian, the Hessian, or higher derivative operators. In principle, a variety of derived algorithms can be generated automatically in this way.

MITgcm has been adapted for use with the Tangent linear and Adjoint Model Compiler (TAMC) and its successor TAF (Transformation of Algorithms in Fortran), developed by Ralf Giering (*Giering and Kaminski* [1998], *Giering* [1999, 2000]). The first application of the adjoint of MITgcm for sensitivity studies has been published by *Marotzke et al.* [1999]. *Stammer et al.* [1997, 002a] use MITgcm and its adjoint for ocean state estimation studies. In the following we shall refer to TAMC and TAF synonymously, except were explicitly stated otherwise.

As of mid-2007 we are also able to generate fairly efficient adjoint code of the MITgcm using a new, open-source AD tool, called OpenAD (see *Naumann et al.* [2006]; *Utke et al.* [2008]. This enables us for the first time to compare adjoint models generated from different AD tools, providing an additional accuracy check, complementary to finite-difference gradient checks. OpenAD and its application to MITgcm is described in detail in section 5.5.

The AD tool exploits the chain rule for computing the first derivative of a function with respect to a set of input variables. Treating a given forward code as a composition of operations – each line representing a compositional element, the chain rule is rigorously applied to the code, line by line. The resulting tangent linear or adjoint code, then, may be thought of as the composition in forward or reverse order, respectively, of the Jacobian matrices of the forward code's compositional elements.

5.1 Some basic algebra

Let $\mathcal{M}$ be a general nonlinear, model, i.e. a mapping from the m-dimensional space $U \subset I\!R^m$ of input variables $\vec{u} = (u_1, \ldots, u_m)$ (model parameters, initial conditions, boundary conditions such as forcing functions) to the n-dimensional space $V \subset I\!R^n$ of model output variable $\vec{v} = (v_1, \ldots, v_n)$ (model state, model diagnostics, objective function, ...) under consideration,

$$\begin{aligned} \mathcal{M} \, : U \; &\longrightarrow \; V \\ \vec{u} \; &\longmapsto \; \vec{v} \, = \, \mathcal{M}(\vec{u}) \end{aligned} \tag{5.1}$$

The vectors $\vec{u} \in U$ and $v \in V$ may be represented w.r.t. some given basis vectors $\mathrm{span}(U) = \{\vec{e}_i\}_{i=1,\ldots,m}$ and $\mathrm{span}(V) = \{\vec{f}_j\}_{j=1,\ldots,n}$ as

$$\vec{u} \, = \, \sum_{i=1}^{m} u_i \, \vec{e}_i, \qquad \vec{v} \, = \, \sum_{j=1}^{n} v_j \, \vec{f}_j$$

Two routes may be followed to determine the sensitivity of the output variable $\vec{v}$ to its input $\vec{u}$.

5.1.1 Forward or direct sensitivity

Consider a perturbation to the input variables $\delta\vec{u}$ (typically a single component $\delta\vec{u} = \delta u_i \, \vec{e}_i$). Their effect on the output may be obtained via the linear approximation of the model $\mathcal{M}$ in terms of its Jacobian matrix M, evaluated in the point $u^{(0)}$ according to

$$\delta\vec{v} \,=\, M|_{\vec{u}^{(0)}} \, \delta\vec{u} \tag{5.2}$$

with resulting output perturbation $\delta\vec{v}$. In components $M_{ji} \,=\, \partial\mathcal{M}_j/\partial u_i$, it reads

$$\delta v_j \,=\, \sum_i \left. \frac{\partial\mathcal{M}_j}{\partial u_i} \right|_{u^{(0)}} \delta u_i \tag{5.3}$$

Eq. (5.2) is the tangent linear model (TLM). In contrast to the full nonlinear model $\mathcal{M}$, the operator M is just a matrix which can readily be used to find the forward sensitivity of $\vec{v}$ to perturbations in u, but if there are very many input variables ($\gg O(10^6)$ for large-scale oceanographic application), it quickly becomes prohibitive to proceed directly as in (5.2), if the impact of each component $\mathbf{e_i}$ is to be assessed.

5.1.2 Reverse or adjoint sensitivity

Let us consider the special case of a scalar objective function $\mathcal{J}(\vec{v})$ of the model output (e.g. the total meridional heat transport, the total uptake of CO_2 in the Southern Ocean over a time interval, or a measure of some model-to-data misfit)

$$\begin{array}{cccccc} \mathcal{J} \,:\, & U & \longrightarrow & V & \longrightarrow & I\!R \\ & \vec{u} & \longmapsto & \vec{v} = \mathcal{M}(\vec{u}) & \longmapsto & \mathcal{J}(\vec{u}) = \mathcal{J}(\mathcal{M}(\vec{u})) \end{array} \tag{5.4}$$

The perturbation of $\mathcal{J}$ around a fixed point $\mathcal{J}_0$,

$$\mathcal{J} \,=\, \mathcal{J}_0 \,+\, \delta\mathcal{J}$$

can be expressed in both bases of $\vec{u}$ and $\vec{v}$ w.r.t. their corresponding inner product $\langle \;,\; \rangle$

$$\begin{aligned} \mathcal{J} &= \mathcal{J}|_{\vec{u}^{(0)}} \,+\, \left\langle \nabla_u\mathcal{J}^T|_{\vec{u}^{(0)}} \,,\, \delta\vec{u} \right\rangle \,+\, O(\delta\vec{u}^2) \\ &= \mathcal{J}|_{\vec{v}^{(0)}} \,+\, \left\langle \nabla_v\mathcal{J}^T|_{\vec{v}^{(0)}} \,,\, \delta\vec{v} \right\rangle \,+\, O(\delta\vec{v}^2) \end{aligned} \tag{5.5}$$

(note, that the gradient ∇f is a co-vector, therefore its transpose is required in the above inner product). Then, using the representation of $\delta\mathcal{J} = \left\langle \nabla_v\mathcal{J}^T \,,\, \delta\vec{v} \right\rangle$, the definition of an adjoint operator A^* of a given operator A,

$$\langle \, A^*\vec{x} \,,\, \vec{y} \, \rangle = \langle \, \vec{x} \,,\, A\vec{y} \, \rangle$$

which for finite-dimensional vector spaces is just the transpose of A,

$$A^* \,=\, A^T$$

and from eq. (5.2), (5.5), we note that (omitting $|$'s):

$$\delta\mathcal{J} \,=\, \left\langle \nabla_v\mathcal{J}^T \,,\, \delta\vec{v} \right\rangle \,=\, \left\langle \nabla_v\mathcal{J}^T \,,\, M\,\delta\vec{u} \right\rangle \,=\, \left\langle M^T\,\nabla_v\mathcal{J}^T \,,\, \delta\vec{u} \right\rangle \tag{5.6}$$

With the identity (5.5), we then find that the gradient $\nabla_u\mathcal{J}$ can be readily inferred by invoking the adjoint M^* of the tangent linear model M

$$\begin{aligned} \nabla_u\mathcal{J}^T|_{\vec{u}} &= M^T|_{\vec{u}} \cdot \nabla_v\mathcal{J}^T|_{\vec{v}} \\ &= M^T|_{\vec{u}} \cdot \delta\vec{v}^* \\ &= \delta\vec{u}^* \end{aligned} \tag{5.7}$$

Eq. (5.7) is the adjoint model (ADM), in which M^T is the adjoint (here, the transpose) of the tangent linear operator M, $\delta\vec{v}^*$ the adjoint variable of the model state $\vec{v}$, and $\delta\vec{u}^*$ the adjoint variable of the control variable $\vec{u}$.

The reverse nature of the adjoint calculation can be readily seen as follows. Consider a model integration which consists of Λ consecutive operations $\mathcal{M}_\Lambda(\mathcal{M}_{\Lambda-1}(......(\mathcal{M}_\lambda(......(\mathcal{M}_1(\mathcal{M}_0(\vec{u})))))$, where the $\mathcal{M}$'s could be the elementary steps, i.e. single lines in the code of the model, or successive time steps of the model integration, starting at step 0 and moving up to step Λ, with intermediate $\mathcal{M}_\lambda(\vec{u}) = \vec{v}^{(\lambda+1)}$ and final $\mathcal{M}_\Lambda(\vec{u}) = \vec{v}^{(\Lambda+1)} = \vec{v}$. Let $\mathcal{J}$ be a cost function which explicitly depends on the final state $\vec{v}$ only (this restriction is for clarity reasons only). $\mathcal{J}(u)$ may be decomposed according to:

$$\mathcal{J}(\mathcal{M}(\vec{u})) \;=\; \mathcal{J}(\mathcal{M}_\Lambda(\mathcal{M}_{\Lambda-1}(......(\mathcal{M}_\lambda(......(\mathcal{M}_1(\mathcal{M}_0(\vec{u})))))) \tag{5.8}$$

Then, according to the chain rule, the forward calculation reads, in terms of the Jacobi matrices (we've omitted the $|$'s which, nevertheless are important to the aspect of *tangent* linearity; note also that by definition $\langle \nabla_v \mathcal{J}^T \,,\, \delta\vec{v} \rangle = \nabla_v \mathcal{J} \cdot \delta\vec{v}$)

$$\begin{aligned} \nabla_v \mathcal{J}(M(\delta\vec{u})) &= \nabla_v \mathcal{J} \cdot M_\Lambda \cdot ...... \cdot M_\lambda \cdot ...... \cdot M_1 \cdot M_0 \cdot \delta\vec{u} \\ &= \nabla_v \mathcal{J} \cdot \delta\vec{v} \end{aligned} \tag{5.9}$$

whereas in reverse mode we have

$$\boxed{\begin{aligned} M^T(\nabla_v \mathcal{J}^T) &= M_0^T \cdot M_1^T \cdot ...... \cdot M_\lambda^T \cdot ...... \cdot M_\Lambda^T \cdot \nabla_v \mathcal{J}^T \\ &= M_0^T \cdot M_1^T \cdot ...... \cdot \nabla_{v^{(\lambda)}} \mathcal{J}^T \\ &= \nabla_u \mathcal{J}^T \end{aligned}} \tag{5.10}$$

clearly expressing the reverse nature of the calculation. Eq. (5.10) is at the heart of automatic adjoint compilers. If the intermediate steps λ in eqn. (5.8) – (5.10) represent the model state (forward or adjoint) at each intermediate time step as noted above, then correspondingly, $M^T(\delta\vec{v}^{(\lambda)\,*}) = \delta\vec{v}^{(\lambda-1)\,*}$ for the adjoint variables. It thus becomes evident that the adjoint calculation also yields the adjoint of each model state component $\vec{v}^{(\lambda)}$ at each intermediate step λ, namely

$$\boxed{\begin{aligned} \nabla_{v^{(\lambda)}} \mathcal{J}^T|_{\vec{v}^{(\lambda)}} &= M_\lambda^T|_{\vec{v}^{(\lambda)}} \cdot ...... \cdot M_\Lambda^T|_{\vec{v}^{(\lambda)}} \cdot \delta\vec{v}^* \\ &= \delta\vec{v}^{(\lambda)\,*} \end{aligned}} \tag{5.11}$$

in close analogy to eq. (5.7) We note in passing that that the $\delta\vec{v}^{(\lambda)\,*}$ are the Lagrange multipliers of the model equations which determine $\vec{v}^{(\lambda)}$.

In components, eq. (5.7) reads as follows. Let

$$\begin{array}{rclcrclcl} \delta\vec{u} & = & (\delta u_1, \ldots, \delta u_m)^T \,, & \qquad & \delta\vec{u}^* & = & \nabla_u \mathcal{J}^T & = & \left(\frac{\partial \mathcal{J}}{\partial u_1}, \ldots, \frac{\partial \mathcal{J}}{\partial u_m} \right)^T \\ \delta\vec{v} & = & (\delta v_1, \ldots, \delta u_n)^T \,, & & \delta\vec{v}^* & = & \nabla_v \mathcal{J}^T & = & \left(\frac{\partial \mathcal{J}}{\partial v_1}, \ldots, \frac{\partial \mathcal{J}}{\partial v_n} \right)^T \end{array}$$

denote the perturbations in $\vec{u}$ and $\vec{v}$, respectively, and their adjoint variables; further

$$M \;=\; \begin{pmatrix} \frac{\partial \mathcal{M}_1}{\partial u_1} & \ldots & \frac{\partial \mathcal{M}_1}{\partial u_m} \\ \vdots & & \vdots \\ \frac{\partial \mathcal{M}_n}{\partial u_1} & \ldots & \frac{\partial \mathcal{M}_n}{\partial u_m} \end{pmatrix}$$

is the Jacobi matrix of $\mathcal{M}$ (an $n \times m$ matrix) such that $\delta\vec{v} = M \cdot \delta\vec{u}$, or

$$\delta v_j \;=\; \sum_{i=1}^{m} M_{ji}\, \delta u_i \;=\; \sum_{i=1}^{m} \frac{\partial \mathcal{M}_j}{\partial u_i} \delta u_i$$

Then eq. (5.7) takes the form

$$\delta u_i^* \;=\; \sum_{j=1}^{n} M_{ji}\, \delta v_j^* \;=\; \sum_{j=1}^{n} \frac{\partial \mathcal{M}_j}{\partial u_i} \delta v_j^*$$

or

$$\begin{pmatrix} \frac{\partial}{\partial u_1}\mathcal{J}\Big|_{\vec{u}^{(0)}} \\ \vdots \\ \frac{\partial}{\partial u_m}\mathcal{J}\Big|_{\vec{u}^{(0)}} \end{pmatrix} = \begin{pmatrix} \frac{\partial \mathcal{M}_1}{\partial u_1}\Big|_{\vec{u}^{(0)}} & \cdots & \frac{\partial \mathcal{M}_n}{\partial u_1}\Big|_{\vec{u}^{(0)}} \\ \vdots & & \vdots \\ \frac{\partial \mathcal{M}_1}{\partial u_m}\Big|_{\vec{u}^{(0)}} & \cdots & \frac{\partial \mathcal{M}_n}{\partial u_m}\Big|_{\vec{u}^{(0)}} \end{pmatrix} \cdot \begin{pmatrix} \frac{\partial}{\partial v_1}\mathcal{J}\Big|_{\vec{v}} \\ \vdots \\ \frac{\partial}{\partial v_n}\mathcal{J}\Big|_{\vec{v}} \end{pmatrix}$$

Furthermore, the adjoint $\delta v^{(\lambda)\,*}$ of any intermediate state $v^{(\lambda)}$ may be obtained, using the intermediate Jacobian (an $n_{\lambda+1} \times n_\lambda$ matrix)

$$M_\lambda = \begin{pmatrix} \frac{\partial(\mathcal{M}_\lambda)_1}{\partial v_1^{(\lambda)}} & \cdots & \frac{\partial(\mathcal{M}_\lambda)_1}{\partial v_{n_\lambda}^{(\lambda)}} \\ \vdots & & \vdots \\ \frac{\partial(\mathcal{M}_\lambda)_{n_{\lambda+1}}}{\partial v_1^{(\lambda)}} & \cdots & \frac{\partial(\mathcal{M}_\lambda)_{n_{\lambda+1}}}{\partial v_{n_\lambda}^{(\lambda)}} \end{pmatrix}$$

and the shorthand notation for the adjoint variables $\delta v_j^{(\lambda)\,*} = \frac{\partial}{\partial v_j^{(\lambda)}}\mathcal{J}^T$, $j = 1, \ldots, n_\lambda$, for intermediate components, yielding

$$\begin{pmatrix} \delta v_1^{(\lambda)\,*} \\ \vdots \\ \delta v_{n_\lambda}^{(\lambda)\,*} \end{pmatrix} = \begin{pmatrix} \frac{\partial(\mathcal{M}_\lambda)_1}{\partial v_1^{(\lambda)}} & \cdots\cdots & \frac{\partial(\mathcal{M}_\lambda)_{n_{\lambda+1}}}{\partial v_1^{(\lambda)}} \\ \vdots & & \vdots \\ \frac{\partial(\mathcal{M}_\lambda)_1}{\partial v_{n_\lambda}^{(\lambda)}} & \cdots\cdots & \frac{\partial(\mathcal{M}_\lambda)_{n_{\lambda+1}}}{\partial v_{n_\lambda}^{(\lambda)}} \end{pmatrix} \cdot$$

$$\begin{pmatrix} \frac{\partial(\mathcal{M}_{\lambda+1})_1}{\partial v_1^{(\lambda+1)}} & \cdots & \frac{\partial(\mathcal{M}_{\lambda+1})_{n_{\lambda+2}}}{\partial v_1^{(\lambda+1)}} \\ \vdots & & \vdots \\ \vdots & & \vdots \\ \frac{\partial(\mathcal{M}_{\lambda+1})_1}{\partial v_{n_{\lambda+1}}^{(\lambda+1)}} & \cdots & \frac{\partial(\mathcal{M}_{\lambda+1})_{n_{\lambda+2}}}{\partial v_{n_{\lambda+1}}^{(\lambda+1)}} \end{pmatrix} \cdot \ldots \cdot \begin{pmatrix} \delta v_1^* \\ \vdots \\ \delta v_n^* \end{pmatrix} \tag{5.12}$$

Eq. (5.9) and (5.10) are perhaps clearest in showing the advantage of the reverse over the forward mode if the gradient $\nabla_u \mathcal{J}$, i.e. the sensitivity of the cost function $\mathcal{J}$ with respect to *all* input variables u (or the sensitivity of the cost function with respect to *all* intermediate states $\vec{v}^{(\lambda)}$) are sought. In order to be able to solve for each component of the gradient $\partial\mathcal{J}/\partial u_i$ in (5.9) a forward calculation has to be performed for each component separately, i.e. $\delta\vec{u} = \delta u_i \vec{e}_i$ for the i-th forward calculation. Then, (5.9) represents the projection of $\nabla_u \mathcal{J}$ onto the i-th component. The full gradient is retrieved from the m forward calculations. In contrast, eq. (5.10) yields the full gradient $\nabla_u \mathcal{J}$ (and all intermediate gradients $\nabla_{v^{(\lambda)}} \mathcal{J}$) within a single reverse calculation.

Note, that if $\mathcal{J}$ is a vector-valued function of dimension $l > 1$, eq. (5.10) has to be modified according to

$$M^T \left(\nabla_v \mathcal{J}^T \left(\delta\vec{J}\right)\right) = \nabla_u \mathcal{J}^T \cdot \delta\vec{J}$$

where now $\delta\vec{J} \in \mathbb{R}^l$ is a vector of dimension l. In this case l reverse simulations have to be performed for each $\delta J_k,\ k = 1, \ldots, l$. Then, the reverse mode is more efficient as long as $l < n$, otherwise the forward mode is preferable. Strictly, the reverse mode is called adjoint mode only for $l = 1$.

A detailed analysis of the underlying numerical operations shows that the computation of $\nabla_u \mathcal{J}$ in this way requires about 2 to 5 times the computation of the cost function. Alternatively, the gradient vector could be approximated by finite differences, requiring m computations of the perturbed cost function.

To conclude we give two examples of commonly used types of cost functions:

Example 1: $\mathcal{J} = v_j(T)$

The cost function consists of the j-th component of the model state $\vec{v}$ at time T. Then $\nabla_v \mathcal{J}^T = \vec{f}_j$ is just the j-th unit vector. The $\nabla_u \mathcal{J}^T$ is the projection of the adjoint operator onto the j-th component $\mathbf{f_j}$,

$$\nabla_u \mathcal{J}^T = M^T \cdot \nabla_v \mathcal{J}^T = \sum_i M_{ji}^T \vec{e}_i$$

Example 2: $\mathcal{J} = \langle\, \mathcal{H}(\vec{v}) - \vec{d}\,,\, \mathcal{H}(\vec{v}) - \vec{d}\,\rangle$

The cost function represents the quadratic model vs. data misfit. Here, $\vec{d}$ is the data vector and $\mathcal{H}$ represents the operator which maps the model state space onto the data space. Then, $\nabla_v \mathcal{J}$ takes the form

$$\begin{aligned} \nabla_v \mathcal{J}^T &= 2\, H \cdot \left(\mathcal{H}(\vec{v}) - \vec{d} \right) \\ &= 2 \sum_j \left\{ \sum_k \frac{\partial \mathcal{H}_k}{\partial v_j} \left(\mathcal{H}_k(\vec{v}) - d_k \right) \right\} \vec{f_j} \end{aligned}$$

where $H_{kj} = \partial \mathcal{H}_k / \partial v_j$ is the Jacobi matrix of the data projection operator. Thus, the gradient $\nabla_u \mathcal{J}$ is given by the adjoint operator, driven by the model vs. data misfit:

$$\nabla_u \mathcal{J}^T \;=\; 2\, M^T \cdot H \cdot \left(\mathcal{H}(\vec{v}) - \vec{d} \right)$$

5.1.3 Storing vs. recomputation in reverse mode

We note an important aspect of the forward vs. reverse mode calculation. Because of the local character of the derivative (a derivative is defined w.r.t. a point along the trajectory), the intermediate results of the model trajectory $\vec{v}^{(\lambda+1)} = \mathcal{M}_\lambda(v^{(\lambda)})$ may be required to evaluate the intermediate Jacobian $M_\lambda|_{\vec{v}^{(\lambda)}}\, \delta\vec{v}^{(\lambda)}$. This is the case e.g. for nonlinear expressions (momentum advection, nonlinear equation of state), state-dependent conditional statements (parameterization schemes). In the forward mode, the intermediate results are required in the same order as computed by the full forward model $\mathcal{M}$, but in the reverse mode they are required in the reverse order. Thus, in the reverse mode the trajectory of the forward model integration $\mathcal{M}$ has to be stored to be available in the reverse calculation. Alternatively, the complete model state up to the point of evaluation has to be recomputed whenever its value is required.

A method to balance the amount of recomputations vs. storage requirements is called checkpointing (e.g. *Griewank* [1992], *Restrepo et al.* [1998]). It is depicted in 5.1 for a 3-level checkpointing [as an example, we give explicit numbers for a 3-day integration with a 1-hourly timestep in square brackets].

- $lev3$ In a first step, the model trajectory is subdivided into n^{lev3} subsections [n^{lev3}=3 1-day intervals], with the label $lev3$ for this outermost loop. The model is then integrated along the full trajectory, and the model state stored to disk only at every k_i^{lev3}-th timestep [i.e. 3 times, at $i = 0, 1, 2$ corresponding to $k_i^{lev3} = 0, 24, 48$]. In addition, the cost function is computed, if needed.

- $lev2$ In a second step each subsection itself is divided into n^{lev2} subsections [n^{lev2}=4 6-hour intervals per subsection]. The model picks up at the last outermost dumped state $v_{k_n^{lev3}}$ and is integrated forward in time along the last subsection, with the label $lev2$ for this intermediate loop. The model state is now stored to disk at every k_i^{lev2}-th timestep [i.e. 4 times, at $i = 0, 1, 2, 3$ corresponding to $k_i^{lev2} = 48, 54, 60, 66$].

- $lev1$ Finally, the model picks up at the last intermediate dump state $v_{k_n^{lev2}}$ and is integrated forward in time along the last subsection, with the label $lev1$ for this intermediate loop. Within this sub-subsection only, parts of the model state is stored to memory at every timestep [i.e. every hour $i = 0, ..., 5$ corresponding to $k_i^{lev1} = 66, 67, \ldots, 71$]. The final state $v_n = v_{k_n^{lev1}}$ is reached and the model state of all preceding timesteps along the last innermost subsection are available, enabling integration backwards in time along the last subsection. The adjoint can thus be computed along this last subsection k_n^{lev2}.

This procedure is repeated consecutively for each previous subsection $k_{n-1}^{lev2}, \ldots, k_1^{lev2}$ carrying the adjoint computation to the initial time of the subsection k_n^{lev3}. Then, the procedure is repeated for the previous subsection k_{n-1}^{lev3} carrying the adjoint computation to the initial time k_1^{lev3}.

For the full model trajectory of $n^{lev3} \cdot n^{lev2} \cdot n^{lev1}$ timesteps the required storing of the model state was significantly reduced to $n^{lev2} + n^{lev3}$ to disk and roughly n^{lev1} to memory [i.e. for the 3-day integration with a total oof 72 timesteps the model state was stored 7 times to disk and roughly 6 times to memory]. This saving in memory comes at a cost of a required 3 full forward integrations of the model (one for each checkpointing level). The optimal balance of storage vs. recomputation certainly depends on the computing resources available and may be adjusted by adjusting the partitioning among the n^{lev3}, n^{lev2}, n^{lev1}.

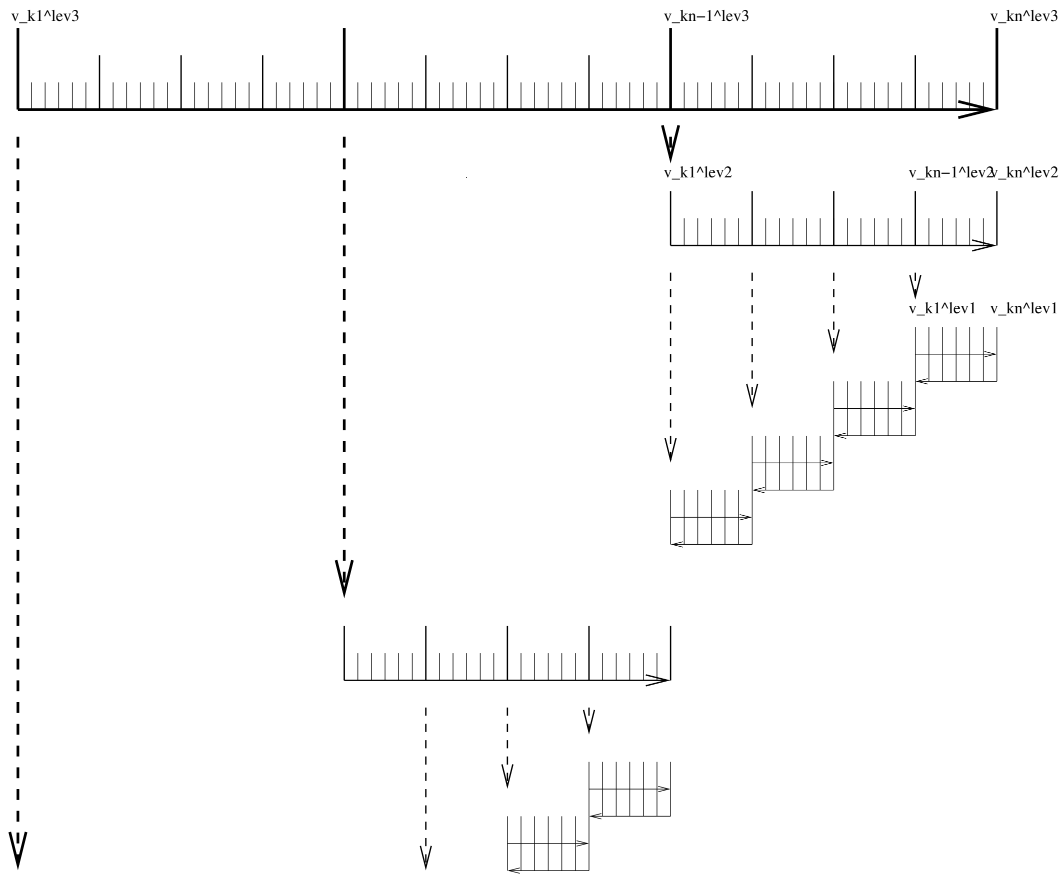


Figure 5.1: Schematic view of intermediate dump and restart for 3-level checkpointing.

5.2 TLM and ADM generation in general

In this section we describe in a general fashion the parts of the code that are relevant for automatic differentiation using the software tool TAF. Modifications to use OpenAD are described in 5.5.

The basic flow is depicted in 5.2. If CPP option `ALLOW_AUTODIFF_TAMC` is defined, the driver routine *the_model_main*, instead of calling *the_main_loop*, invokes the adjoint of this routine, *adthe_main_loop* (case `#define ALLOW_ADJOINT_RUN`), or the tangent linear of this routine *g_the_main_loop* (case `#define ALLOW_TANGENTLINEAR_RUN`), which are the toplevel routines in terms of automatic differentiation. The routines *adthe_main_loop* or *g_the_main_loop* are generated by TAF. It contains both the forward integration of the full model, the cost function calculation, any additional storing that is required for efficient checkpointing, and the reverse integration of the adjoint model.

[DESCRIBE IN A SEPARATE SECTION THE WORKING OF THE TLM]

In Fig. 5.2 the structure of *adthe_main_loop* has been strongly simplified to focus on the essentials; in particular, no checkpointing procedures are shown here. Prior to the call of *adthe_main_loop*, the routine *ctrl_unpack* is invoked to unpack the control vector or initialise the control variables. Following the call of *adthe_main_loop*, the routine *ctrl_pack* is invoked to pack the control vector (cf. Section 5.2.5). If gradient checks are to be performed, the option `ALLOW_GRADIENT_CHECK` is defined. In this case the driver routine *grdchk_main* is called after the gradient has been computed via the adjoint (cf. Section 5.3).

5.2.1 General setup

In order to configure AD-related setups the following packages need to be enabled: The packages are enabled by adding them to your experiment-specific configuration file *packages.conf* (see Section ???).

The following AD-specific CPP option files need to be customized:

autodiff
ctrl
cost
grdchk

- *ECCO_CPPOPTIONS.h*
  This header file collects CPP options for the packages *autodiff, cost, ctrl* as well as AD-unrelated options for the external forcing package *exf.* [1]

- *tamc.h*
  This header configures the splitting of the time stepping loop w.r.t. the 3-level checkpointing (see section ???).

5.2.2 Building the AD code using TAF

The build process of an AD code is very similar to building the forward model. However, depending on which AD code one wishes to generate, and on which AD tool is available (TAF or TAMC), the following `make` targets are available:

Here, the following placeholders are used

- <TOOL>

[1]NOTE: These options are not set in their package-specific headers such as *COST_CPPOPTIONS.h*, but are instead collected in the single header file *ECCO_CPPOPTIONS.h*. The package-specific header files serve as simple placeholders at this point.

```
the_model_main
|
|--- initialise_fixed
|
|--- #ifdef ALLOW_ADJOINT_RUN
|           |
|           |--- ctrl_unpack
|           |
|           |--- adthe_main_loop
|           |    |
|           |    |--- initialise_varia
|           |    |--- ctrl_map_forcing
|           |    |--- do iloop = 1, nTimeSteps
|           |    |       |--- forward_step
|           |    |       |--- cost_tile
|           |    |    end do
|           |    |--- cost_final
|           |    |
|           |    |--- adcost_final
|           |    |--- do iloop = nTimeSteps, 1, -1
|           |    |       |--- adcost_tile
|           |    |       |--- adforward_step
|           |    |    end do
|           |    |--- adctrl_map_forcing
|           |    |--- adinitialise_varia
|           |    o
|           |
|           |--- ctrl_pack
|           |
|--- #else
|           |
|           |--- the_main_loop
|           |
|    #endif
|
|--- #ifdef ALLOW_GRADIENT_CHECK
|           |
|           |--- grdchk_main
|           o
|    #endif
o
```

Figure 5.2:

	AD-target	*output*	*description*
(1)	<MODE><TOOL>only	<MODE>_<TOOL>_output.f	generates code for <MODE> using <TOOL> no `make` dependencies on `.F .h` useful for compiling on remote platforms
(2)	<MODE><TOOL>	<MODE>_<TOOL>_output.f	generates code for <MODE> using <TOOL> includes `make` dependencies on `.F .h` i.e. input for <TOOL> may be re-generated
(3)	<MODE>all	mitgcmuv_<MODE>	generates code for <MODE> using <TOOL> and compiles all code (use of TAF is set as default)

  - `TAF`
  - `TAMC`

- <MODE>
  - `ad` generates the adjoint model (ADM)
  - `ftl` generates the tangent linear model (TLM)
  - `svd` generates both ADM and TLM for singular value decomposition (SVD) type calculations

For example, to generate the adjoint model using TAF after routines (`.F`) or headers (`.h`) have been modified, but without compilation, type `make adtaf`; or, to generate the tangent linear model using TAMC without re-generating the input code, type `make ftltamconly`.

A typical full build process to generate the ADM via TAF would look like follows:

```
% mkdir build
% cd build
% ../../../tools/genmake2 -mods=../code_ad
% make depend
% make adall
```

5.2.3 The AD build process in detail

The make <MODE>all target consists of the following procedures:

1. A header file `AD_CONFIG.h` is generated which contains a CPP option on which code ought to be generated. Depending on the `make` target, the contents is one of the following:
   - `#define ALLOW_ADJOINT_RUN`
   - `#define ALLOW_TANGENTLINEAR_RUN`
   - `#define ALLOW_ECCO_OPTIMIZATION`
2. A single file <MODE>_input_code.f is concatenated consisting of all `.f` files that are part of the list **AD_FILES** and all `.flow` files that are part of the list **AD_FLOW_FILES**.
3. The AD tool is invoked with the <MODE>_<TOOL>_FLAGS. The default AD tool flags in `genmake2` can be overrwritten by an `adjoint_options` file (similar to the platform-specific `build_options`, see Section ???. The AD tool writes the resulting AD code into the file <MODE>_input_code_ad.f
4. A short sed script `adjoint_sed` is applied to <MODE>_input_code_ad.f to reinstate **myThid** into the CALL argument list of active file I/O. The result is written to file <MODE>_<TOOL>_output.f.
5. All routines are compiled and an executable is generated (see Table ???).

5.2.3.1 The list AD_FILES and `.list` files

Not all routines are presented to the AD tool. Routines typically hidden are diagnostics routines which do not influence the cost function, but may create artificial flow dependencies such as I/O of active variables.

`genmake2` generates a list (or variable) **AD_FILES** which contains all routines that are shown to the AD tool. This list is put together from all files with suffix `.list` that `genmake2` finds in its search directories. The list file for the core MITgcm routines is in `model/src/` is called `model_ad_diff.list`. Note that no wrapper routine is shown to TAF. These are either not visible at all to the AD code, or hand-written AD code is available (see next section).

Each package directory contains its package-specific list file `<PKG>_ad_diff.list`. For example, `pkg/ptracers/` contains the file `ptracers_ad_diff.list`. Thus, enabling a package will automatically extend the **AD_FILES** list of `genmake2` to incorporate the package-specific routines. Note that you will need to regenerate the `Makefile` if you enable a package (e.g. by adding it to `packages.conf`) and a `Makefile` already exists.

5.2.3.2 The list AD_FLOW_FILES and `.flow` files

TAMC and TAF can evaluate user-specified directives that start with a specific syntax (`CADJ`, `C$TAF`, `!$TAF`). The main categories of directives are STORE directives and FLOW directives. Here, we are concerned with flow directives, store directives are treated elsewhere.

Flow directives enable the AD tool to evaluate how it should treat routines that are 'hidden' by the user, i.e. routines which are not contained in the **AD_FILES** list (see previous section), but which are called in part of the code that the AD tool does see. The flow directive tell the AD tool

- which subroutine arguments are input/output
- which subroutine arguments are active
- which subroutine arguments are required to compute the cost
- which subroutine arguments are dependent

The syntax for the flow directives can be found in the AD tool manuals.

`genmake2` generates a list (or variable) **AD_FLOW_FILES** which contains all files with suffix`.flow` that it finds in its search directories. The flow directives for the core MITgcm routines of `eesupp/src/` and `model/src/` reside in `pkg/autodiff/`. This directory also contains hand-written adjoint code for the MITgcm WRAPPER (section 4).

Flow directives for package-specific routines are contained in the corresponding package directories in the file `<PKG>_ad.flow`, e.g. ptracers-specific directives are in `ptracers_ad.flow`.

5.2.3.3 Store directives for 3-level checkpointing

The storing that is required at each period of the 3-level checkpointing is controled by three top-level headers.

```
do ilev_3 = 1, nchklev_3
#  include ‘‘checkpoint_lev3.h’’
   do ilev_2 = 1, nchklev_2
#     include ‘‘checkpoint_lev2.h’’
      do ilev_1 = 1, nchklev_1
#        include ‘‘checkpoint_lev1.h’’

...

      end do
   end do
end do
```

All files `checkpoint_lev?.h` are contained in directory `pkg/autodiff/`.

5.2.3.4 Changing the default AD tool flags: ad_options files

5.2.3.5 Hand-written adjoint code

5.2.4 The cost function (dependent variable)

The cost function $\mathcal{J}$ is referred to as the dependent variable. It is a function of the input variables $\vec{u}$ via the composition $\mathcal{J}(\vec{u}) = \mathcal{J}(M(\vec{u}))$. The input are referred to as the independent variables or control variables. All aspects relevant to the treatment of the cost function $\mathcal{J}$ (parameter setting, initialization, accumulation, final evaluation), are controlled by the package *pkg/cost*. The aspects relevant to the treatment of the independent variables are controlled by the package *pkg/ctrl* and will be treated in the next section.

```
the_model_main
|
|-- initialise_fixed
|   |
|   |-- packages_readparms
|       |
|       |-- cost_readparms
|       o
|
|-- the_main_loop
...  |
     |-- initialise_varia
     |   |
     |   |-- packages_init_variables
     |       |
     |       |-- cost_init
     |       o
     |
     |-- do iloop = 1,nTimeSteps
     |       |-- forward_step
     |       |-- cost_tile
     |       |   |
     |       |   |-- cost_tracer
     |   end do
     |
     |-- cost_final
     o
```

Figure 5.3:

5.2.4.1 Enabling the package

packages.conf, *ECCO_CPPOPTIONS.h*

- The package is enabled by adding *cost* to your file *packages.conf* (see Section ???)
-

N.B.: In general the following packages ought to be enabled simultaneously: *autodiff, cost, ctrl*. The basic CPP option to enable the cost function is **ALLOW_COST**. Each specific cost function contribution has its own option. For the present example the option is **ALLOW_COST_TRACER**. All cost-specific options are set in *ECCO_CPPOPTIONS.h* Since the cost function is usually used in conjunction with automatic differentiation, the CPP option **ALLOW_ADJOINT_RUN** (file *CPP_OPTIONS.h*) and **ALLOW_AUTODIFF_TAMC** (file *ECCO_CPPOPTIONS.h*) should be defined.

5.2.4.2 Initialization

The initialization of the *cost* package is readily enabled as soon as the CPP option **ALLOW_COST** is defined.

Parameters: *cost_readparms*

  This S/R reads runtime flags and parameters from file *data.cost*. For the present example the only relevant parameter read is **mult_tracer**. This multiplier enables different cost function contributions to be switched on (= 1.) or off (= 0.) at runtime. For more complex cost functions which involve model vs. data misfits, the corresponding data filenames and data specifications (start date and time, period, ...) are read in this S/R.

- Variables: *cost_init*
  This S/R initializes the different cost function contributions. The contribution for the present example is **objf_tracer** which is defined on each tile (bi,bj).

5.2.4.3 Accumulation

- *cost_tile*, *cost_tracer*

The 'driver' routine *cost_tile* is called at the end of each time step. Within this 'driver' routine, S/R are called for each of the chosen cost function contributions. In the present example (**ALLOW_COST_TRACER**), S/R *cost_tracer* is called. It accumulates **objf_tracer** according to eqn. (ref:ask-the-author).

5.2.4.4 Finalize all contributions

- *cost_final*

At the end of the forward integration S/R *cost_final* is called. It accumulates the total cost function **fc** from each contribution and sums over all tiles:

$$\mathcal{J} \, = \, \text{fc} \, = \, \text{mult\_tracer} \sum_{\text{global sum}} \sum_{bi,\,bj}^{nSx,\,nSy} \text{objf\_tracer}(bi,bj) \, + \, ... \tag{5.13}$$

The total cost function **fc** will be the 'dependent' variable in the argument list for TAF, i.e.

```
taf -output 'fc' ...
```

5.2.5 The control variables (independent variables)

The control variables are a subset of the model input (initial conditions, boundary conditions, model parameters). Here we identify them with the variable $\vec{u}$. All intermediate variables whose derivative w.r.t. control variables do not vanish are called active variables. All subroutines whose derivative w.r.t. the control variables don't vanish are called active routines. Read and write operations from and to file can be viewed as variable assignments. Therefore, files to which active variables are written and from which active variables are read are called active files. All aspects relevant to the treatment of the control variables (parameter setting, initialization, perturbation) are controlled by the package *pkg/ctrl*.

5.2.5.1 genmake and CPP options

- *genmake*, *CPP_OPTIONS.h*, *ECCO_CPPOPTIONS.h*

To enable the directory to be included to the compile list, **ctrl** has to be added to the **enable** list in *.genmakerc* or in *genmake* itself (analogous to *cost* package, cf. previous section). Each control variable is enabled via its own CPP option in *ECCO_CPPOPTIONS.h*.

5.2.5.2 Initialization

- Parameters: *ctrl_readparms*
  This S/R reads runtime flags and parameters from file *data.ctrl*. For the present example the file contains the file names of each control variable that is used. In addition, the number of wet points for each control variable and the net dimension of the space of control variables (counting wet points only) **nvarlength** is determined. Masks for wet points for each tile **(bi, bj)** and vertical layer **k** are generated for the three relevant categories on the C-grid: **nWetCtile** for tracer fields, **nWetWtile** for zonal velocity fields, **nWetStile** for meridional velocity fields.

```
 *************
 the_main_loop
 *************
 |
 |--- initialise_varia
 |    |
 |   ...
 |    |--- packages_init_varia
 |    |    |
 |    |   ...
 |    |    |--- #ifdef ALLOW_ADJOINT_RUN
 |    |    |          call ctrl_map_ini
 |    |    |          call cost_ini
 |    |    |    #endif
 |    |   ...
 |    |    o
 |   ...
 |    o
...
 |--- #ifdef ALLOW_ADJOINT_RUN
 |          call ctrl_map_forcing
 |    #endif
...
 |--- #ifdef ALLOW_TAMC_CHECKPOINTING
            do ilev_3 = 1,nchklev_3
 |            do ilev_2 = 1,nchklev_2
 |              do ilev_1 = 1,nchklev_1
 |                iloop = (ilev_3-1)*nchklev_2*nchklev_1 +
 |                        (ilev_2-1)*nchklev_1           + ilev_1
 |    #else
 |          do iloop = 1, nTimeSteps
 |    #endif
 |    |
 |    |---       call forward_step
 |    |
 |    |--- #ifdef ALLOW_COST
 |    |          call cost_tile
 |    |    #endif
 |    |
 |    |    enddo
 |    o
 |
 |--- #ifdef ALLOW_COST
 |          call cost_final
 |    #endif
 o
```

Figure 5.4:

```
the_model_main
|
|-- initialise_fixed
|   |
|   |-- packages_readparms
|       |
|       |-- ctrl_init                - initialise control
|       o                              package
|
|-- ctrl_unpack                      - unpack control vector
|
|-- adthe_main_loop                  - forward/adjoint run
|   |
|   |-- initialise_variables
|   |   |
|   |   |-- packages_init_variables
|   |       |
|   |       |-- ctrl_map_ini         - link init. state and
|   |       o                          parameters to control
|   |                                  variables
|   |-- ctrl_map_forcing             - link forcing fields to
|  ...                                 control variables
|
|-- ctrl_pack                        - pack control vector
```

Figure 5.5:

- Control variables, control vector, and their gradients: *ctrl_unpack*

  Two important issues related to the handling of the control variables in MITgcm need to be addressed. First, in order to save memory, the control variable arrays are not kept in memory, but rather read from file and added to the initial fields during the model initialization phase. Similarly, the corresponding adjoint fields which represent the gradient of the cost function w.r.t. the control variables are written to file at the end of the adjoint integration. Second, in addition to the files holding the 2-dim. and 3-dim. control variables and the corresponding cost gradients, a 1-dim. control vector and gradient vector are written to file. They contain only the wet points of the control variables and the corresponding gradient. This leads to a significant data compression. Furthermore, an option is available (`ALLOW_NONDIMENSIONAL_CONTROL_IO`) to non-dimensionalise the control and gradient vector, which otherwise would contain different pieces of different magnitudes and units. Finally, the control and gradient vector can be passed to a minimization routine if an update of the control variables is sought as part of a minimization exercise.

  The files holding fields and vectors of the control variables and gradient are generated and initialised in S/R *ctrl_unpack*.

5.2.5.3 Perturbation of the independent variables

The dependency flow for differentiation w.r.t. the controls starts with adding a perturbation onto the input variable, thus defining the independent or control variables for TAF. Three types of controls may be considered:

- *ctrl_map_ini* (initial value sensitivity):

  Consider as an example the initial tracer distribution **tr1** as control variable. After **tr1** has been initialised in *ini_tr1* (dynamical variables such as temperature and salinity are initialised in *ini_fields*), a perturbation anomaly is added to the field in S/R *ctrl_map_ini*

$$\begin{aligned} u &= u_{[0]} + \Delta u \\ \mathbf{tr1}(...) &= \mathbf{tr1_{ini}}(...) + \mathbf{xx\_tr1}(...) \end{aligned} \tag{5.14}$$

  xx_tr1 is a 3-dim. global array holding the perturbation. In the case of a simple sensitivity study this array is identical to zero. However, it's specification is essential in the context of automatic differentiation since TAF treats the corresponding line in the code symbolically when determining the differentiation chain and its origin. Thus, the variable names are part of the argument list when calling TAF:

```
taf -input 'xx_tr1 ...' ...
```

Now, as mentioned above, MITgcm avoids maintaining an array for each control variable by reading the perturbation to a temporary array from file. To ensure the symbolic link to be recognized by TAF, a scalar dummy variable **xx_tr1_dummy** is introduced and an 'active read' routine of the adjoint support package *pkg/autodiff* is invoked. The read-procedure is tagged with the variable **xx_tr1_dummy** enabling TAF to recognize the initialization of the perturbation. The modified call of TAF thus reads

```
taf -input 'xx_tr1_dummy ...' ...
```

and the modified operation to (5.14) in the code takes on the form

```
      call active_read_xyz(
     &      ..., tmpfld3d, ..., xx_tr1_dummy, ... )

      tr1(...) = tr1(...) + tmpfld3d(...)
```

Note, that reading an active variable corresponds to a variable assignment. Its derivative corresponds to a write statement of the adjoint variable, followed by a reset. The 'active file' routines have been designed to support active read and corresponding adjoint active write operations (and vice versa).

- *ctrl_map_forcing* (boundary value sensitivity):
  The handling of boundary values as control variables proceeds exactly analogous to the initial values with the symbolic perturbation taking place in S/R *ctrl_map_forcing*. Note however an important difference: Since the boundary values are time dependent with a new forcing field applied at each time steps, the general problem may be thought of as a new control variable at each time step (or, if the perturbation is averaged over a certain period, at each N timesteps), i.e.

$$u_{\mathrm{forcing}} = \{ u_{\mathrm{forcing}}(t_n) \}_{n=1,\ldots,\mathrm{nTimeSteps}}$$

  In the current example an equilibrium state is considered, and only an initial perturbation to surface forcing is applied with respect to the equilibrium state. A time dependent treatment of the surface forcing is implemented in the ECCO environment, involving the calendar (*cal*) and external forcing (*exf*) packages.

- *ctrl_map_params* (parameter sensitivity):
  This routine is not yet implemented, but would proceed proceed along the same lines as the initial value sensitivity. The mixing parameters **diffkr** and **kapgm** are currently added as controls in *ctrl_map_ini.F*.

5.2.5.4 Output of adjoint variables and gradient

Several ways exist to generate output of adjoint fields.

- *ctrl_map_ini, ctrl_map_forcing*:
  - **xx_...**: the control variable fields
    Before the forward integration, the control variables are read from file **xx_ ...** and added to the model field.
  - **adxx_...**: the adjoint variable fields, i.e. the gradient $\nabla_u \mathcal{J}$ for each control variable
    After the adjoint integration the corresponding adjoint variables are written to **adxx_ ...**.
- *ctrl_unpack, ctrl_pack*:
  - **vector_ctrl**: the control vector
    At the very beginning of the model initialization, the updated compressed control vector is read (or initialised) and distributed to 2-dim. and 3-dim. control variable fields.

  - **vector_grad**: the gradient vector
    At the very end of the adjoint integration, the 2-dim. and 3-dim. adjoint variables are read, compressed to a single vector and written to file.

- *addummy_in_stepping*:

  In addition to writing the gradient at the end of the forward/adjoint integration, many more adjoint variables of the model state at intermediate times can be written using S/R *addummy_in_stepping*. This routine is part of the adjoint support package *pkg/autodiff* (cf.f. below). The procedure is enabled using via the CPP-option **ALLOW_AUTODIFF_MONITOR** (file *ECCO_CPPOPTIONS.h*). To be part of the adjoint code, the corresponding S/R *dummy_in_stepping* has to be called in the forward model (S/R *the_main_loop*) at the appropriate place. The adjoint common blocks are extracted from the adjoint code via the header file *adcommon.h*.

  dummy_in_stepping is essentially empty, the corresponding adjoint routine is hand-written rather than generated automatically. Appropriate flow directives (*dummy_in_stepping.flow*) ensure that TAMC does not automatically generate *addummy_in_stepping* by trying to differentiate *dummy_in_stepping*, but instead refers to the hand-written routine.

  dummy_in_stepping is called in the forward code at the beginning of each timestep, before the call to *dynamics*, thus ensuring that *addummy_in_stepping* is called at the end of each timestep in the adjoint calculation, after the call to *addynamics*.

  addummy_in_stepping includes the header files *adcommon.h*. This header file is also hand-written. It contains the common blocks **/addynvars_r/**, **/addynvars_cd/**, **/addynvars_diffkr/**, **/addynvars_kapgm/**, **/adtr1_r/**, **/adffields/**, which have been extracted from the adjoint code to enable access to the adjoint variables.

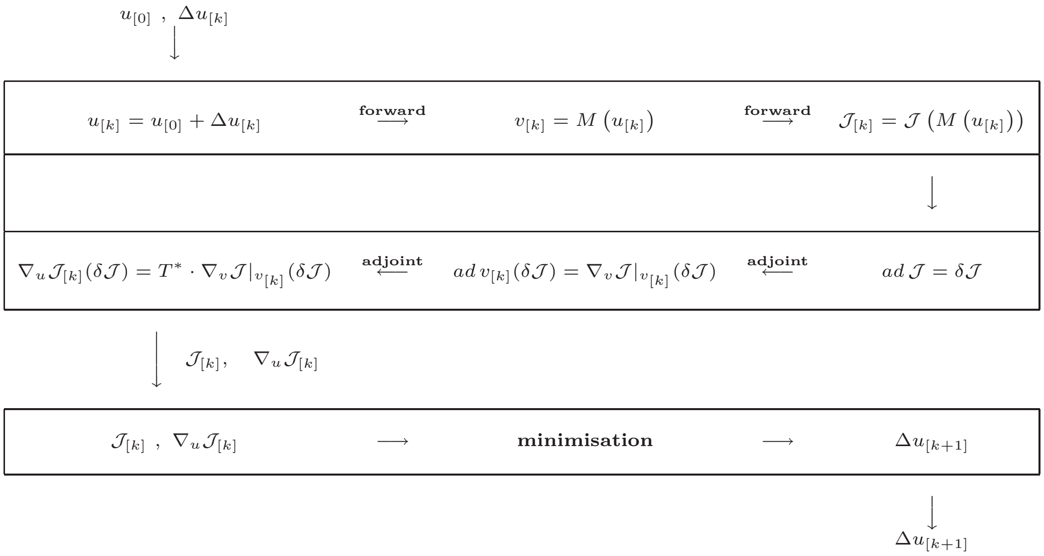
  WARNING: If the structure of the common blocks **/dynvars_r/**, **/dynvars_cd/**, etc., changes similar changes will occur in the adjoint common blocks. Therefore, consistency between the TAMC-generated common blocks and those in *adcommon.h* have to be checked.

5.2.5.5 Control variable handling for optimization applications

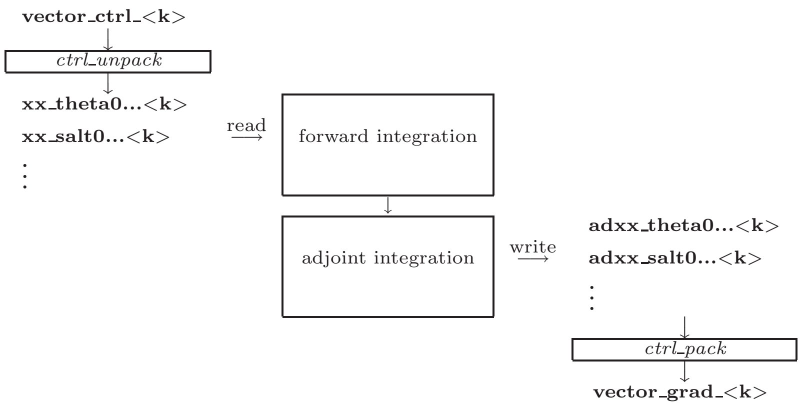
In optimization mode the cost function $\mathcal{J}(u)$ is sought to be minimized with respect to a set of control variables $\delta\mathcal{J} = 0$, in an iterative manner. The gradient $\nabla_u \mathcal{J}|_{u_{[k]}}$ together with the value of the cost function itself $\mathcal{J}(u_{[k]})$ at iteration step k serve as input to a minimization routine (e.g. quasi-Newton method, conjugate gradient, ... *Gilbert and Lemaréchal* [1989]) to compute an update in the control variable for iteration step $k+1$

$$u_{[k+1]} = u_{[0]} + \Delta u_{[k+1]} \quad \text{satisfying} \quad \mathcal{J}\left(u_{[k+1]}\right) < \mathcal{J}\left(u_{[k]}\right)$$

$u_{[k+1]}$ then serves as input for a forward/adjoint run to determine $\mathcal{J}$ and $\nabla_u\mathcal{J}$ at iteration step $k+1$. Tab. ref:ask-the-author sketches the flow between forward/adjoint model and the minimization routine.



The routines *ctrl_unpack* and *ctrl_pack* provide the link between the model and the minimization routine. As described in Section ref:ask-the-author the *unpack* and *pack* routines read and write control and gradient *vectors* which are compressed to contain only wet points, in addition to the full 2-dim. and 3-dim. fields. The corresponding I/O flow looks as follows:



ctrl_unpack reads the updated control vector **vector_ctrl_<k>**. It distributes the different control variables to 2-dim. and 3-dim. files *xx_...<k>*. At the start of the forward integration the control variables are read from *xx_...<k>* and added to the field. Correspondingly, at the end of the adjoint integration the adjoint fields are written to *adxx_...<k>*, again via the active file routines. Finally, *ctrl_pack* collects all adjoint files and writes them to the compressed vector file **vector_grad_<k>**.

5.3 The gradient check package

An indispensable test to validate the gradient computed via the adjoint is a comparison against finite difference gradients. The gradient check package *pkg/grdchk* enables such tests in a straightforward and easy manner. The driver routine *grdchk_main* is called from *the_model_main* after the gradient has been computed via the adjoint model (cf. flow chart ???).

The gradient check proceeds as follows: The $i-$th component of the gradient $(\nabla_u \mathcal{J}^T)_i$ is compared with the following finite-difference gradient:

$$\left(\nabla_u \mathcal{J}^T\right)_i \qquad \text{vs.} \qquad \frac{\partial \mathcal{J}}{\partial u_i} = \frac{\mathcal{J}(u_i + \epsilon) - \mathcal{J}(u_i)}{\epsilon}$$

A gradient check at point u_i may generally considered to be successful if the deviation of the ratio between the adjoint and the finite difference gradient from unity is less than 1 percent,

$$1 - \frac{(\text{grad}\mathcal{J})_i(\text{adjoint})}{(\text{grad}\mathcal{J})_i(\text{finite difference})} < 1\%$$

5.3.1 Code description

5.3.2 Code configuration

The relevant CPP precompile options are set in the following files:

- *.genmakerc*
  option `grdchk` is added to the **enable list** (alternatively, *genmake* may be invoked with the option `-enable=grdchk`).
- *CPP_OPTIONS.h*
  Together with the flag **ALLOW_ADJOINT_RUN**, define the flag **ALLOW_GRADIENT_CHECK**.

The relevant runtime flags are set in the files

- *data.pkg*
  Set **useGrdchk = .TRUE.**
- *data.grdchk*
  - **grdchk_eps**
  - **nbeg**
  - **nstep**
  - **nend**
  - **grdchkvarindex**

```
the_model_main
|
|-- ctrl_unpack
|-- adthe_main_loop           - unperturbed cost function and
|-- ctrl_pack                   adjoint gradient are computed here
|
|-- grdchk_main
    |
    |-- grdchk_init
    |-- do icomp=...          - loop over control vector elements
        |
        |-- grdchk_loc        - determine location of icomp on grid
        |
        |-- grdchk_getxx      - get control vector component from file
        |                       perturb it and write back to file
        |-- grdchk_getadxx    - get gradient component calculated
        |                       via adjoint
        |-- the_main_loop     - forward run and cost evaluation
        |                       with perturbed control vector element
        |-- calculate ratio of adj. vs. finite difference gradient
        |
        |-- grdchk_setxx      - Reset control vector element
        |
        |-- grdchk_print      - print results
```

Figure 5.6:

5.4 Adjoint dump & restart – divided adjoint (DIVA)

Patrick Heimbach & Geoffrey Gebbie, MIT/EAPS, 07-Mar-2003

NOTE:

THIS SECTION IS SUBJECT TO CHANGE. IT REFERS TO TAF-1.4.26.

Previous TAF versions are incomplete and have problems with both TAF options '-pure' and '-mpi'.

The code which is tuned to the DIVA implementation of this TAF version is *checkpoint50* **(MITgcm) and** *ecco_c50_e28* **(ECCO).**

5.4.1 Introduction

Most high performance computing (HPC) centres require the use of batch jobs for code execution. Limits in maximum available CPU time and memory may prevent the adjoint code execution from fitting into any of the available queues. This presents a serious limit for large scale / long time adjoint ocean and climate model integrations. The MITgcm itself enables the split of the total model integration into sub-intervals through standard dump/restart of/from the full model state. For a similar procedure to run in reverse mode, the adjoint model requires, in addition to the model state, the adjoint model state, i.e. all variables with derivative information which are needed in an adjoint restart. This adjoint dump & restart is also termed 'divided adjoint (DIVA).

For this to work in conjunction with automatic differentiation, an AD tool needs to perform the following tasks:

1. identify an adjoint state, i.e. those sensitivities whose accumulation is interrupted by a dump/restart and which influence the outcome of the gradient. Ideally, this state consists of
    - the adjoint of the model state,
    - the adjoint of other intermediate results (such as control variables, cost function contributions, etc.)
    - bookkeeping indices (such as loop indices, etc.)
2. generate code for storing and reading adjoint state variables
3. generate code for bookkeeping , i.e. maintaining a file with index information
4. generate a suitable adjoint loop to propagate adjoint values for dump/restart with a minimum overhad of adjoint intermediate values.

TAF (but not TAMC!) generates adjoint code which performs the above specified tasks. It is closely tied to the adjoint multi-level checkpointing. The adjoint state is dumped (and restarted) at each step of the outermost checkpointing level and adjoint intergration is performed over one outermost checkpointing interval. Prior to the adjoint computations, a full foward sweep is performed to generate the outermost (forward state) tapes and to calculate the cost function. In the current implementation, the forward sweep is immediately followed by the first adjoint leg. Thus, in theory, the following steps are performed (automatically)

- **1st model call:**
  This is the case if file `costfinal` does *not* exist. S/R `mdthe_main_loop` is called.
  1. calculate forward trajectory and dump model state after each outermost checkpointing interval to files `tapelev3`
  2. calculate cost function `fc` and write it to file `costfinal`
- **2nd and all remaining model call:**
  This is the case if file `costfinal` *does* exist. S/R `adthe_main_loop` is called.
  1. (forward run and cost function call is avoided since all values are known)
     – if 1st adjoint leg:
       create index file `divided.ctrl` which contains info on current checkpointing index $ilev3$
     – if not i-th adjoint leg:
       adjoint picks up at $ilev3 = nlev3 - i + 1$ and runs to $nlev3 - i$
  2. perform adjoint leg from $nlev3 - i + 1$ to $nlev3 - i$
  3. dump adjoint state to file `snapshot`
  4. dump index file `divided.ctrl` for next adjoint leg
  5. in the last step the gradient is written.

A few modifications were performed in the forward code, obvious ones such as adding the corresponding TAF-directive at the appropriate place, and less obvious ones (avoid some re-initializations, when in an intermediate adjoint integration interval).

[For TAF-1.4.20 a number of hand-modifications were necessary to compensate for TAF bugs. Since we refer to TAF-1.4.26 onwards, these modifications are not documented here].

5.4.2 Recipe 1: single processor

1. In `ECCO_CPPOPTIONS.h` set:

   ```
   #define ALLOW_DIVIDED_ADJOINT
   #undef  ALLOW_DIVIDED_ADJOINT_MPI
   ```

2. Generate adjoint code. Using the TAF option '`-pure`', two codes are generated:

   - `mdthe_main_loop`:
     Is responsible for the forward trajectory, storing of outermost checkpoint levels to file, computation of cost function, and storing of cost function to file (1st step).
   - `adthe_main_loop`:
     Is responsible for computing one adjoint leg, dump adjoint state to file and write index info to file (2nd and consecutive steps).
     for adjoint code generation, e.g. add '`-pure`' to TAF option list

     ```
     make adtaf
     ```

   - One modification needs to be made to adjoint codes in S/R adecco_the_main_loop:
     There's a remaining issue with the '`-pure`' option. The '`call ad...`' between '`call ad...`' and the read of the `snapshot` file should be called only in the firt adjoint leg between $nlev3$ and $nlev3 - 1$. In the ecco-branch, the following lines should be bracketed by an `if (idivbeg .GE. nchklev_3) then`, thus:

```
...
      xx_psbar_mean_dummy = onetape_xx_psbar_mean_dummy_3h(1)
      xx_tbar_mean_dummy = onetape_xx_tbar_mean_dummy_4h(1)
      xx_sbar_mean_dummy = onetape_xx_sbar_mean_dummy_5h(1)
      call barrier( mythid )
cAdd(
      if (idivbeg .GE. nchklev_3) then
cAdd)

      call adcost_final( mythid )
      call barrier( mythid )
      call adcost_sst( mythid )
      call adcost_ssh( mythid )
      call adcost_hyd( mythid )
      call adcost_averagesfields( mytime,myiter,mythid )
      call barrier( mythid )
cAdd(
      endif
cAdd)

C----------------------------------------------
C read snapshot
C----------------------------------------------
      if (idivbeg .lt. nchklev_3) then
        open(unit=77,file='snapshot',status='old',form='unformatted',
     $iostat=iers)
...
```

For the main code, in all likelihood the block which needs to be bracketed consists of `adcost_final` only.

- Now the code can be copied as usual to `adjoint_model.F` and then be compiled:

```
make adchange
then compile
```

5.4.3 Recipe 2: multi processor (MPI)

1. On the machine where you execute the code (most likely not the machine where you run TAF) find the includes directory for MPI containing `mpif.h`. Either copy `mpif.h` to the machine where you generate the `.f` files before TAF-ing, or add the path to the includes directory to you genmake `platform` setup, TAF needs some MPI parameter settings (essentially `mpi_comm_world` and `mpi_integer`) to incorporate those in the adjoint code.

2. In `ECCO_CPPOPTIONS.h` set

```
#define ALLOW_DIVIDED_ADJOINT
#define ALLOW_DIVIDED_ADJOINT_MPI
```

   This will include the header file `mpif.h` into the top level routine for TAF.

3. Add the TAF option '`-mpi`' to the TAF argument list in the makefile.

4. Follow the same steps as in **Recipe 1** (previous section).

That's it. Good luck & have fun.

5.5 Adjoint code generation using OpenAD

Authors: Jean Utke, Patrick Heimbach and Chris Hill

5.5.1 Introduction

The development of OpenAD was initiated as part of the ACTS (Adjoint Compiler Technology & Standards) project funded by the NSF Information Technology Research (ITR) program. The main goals for OpenAD initially defined for the ACTS project are:

1. develop a flexible, modular, open source tool that can ge'nerate adjoint codes of numerical simulation programs,
2. establish a platform for easy implementation and testing of source transformation algorithms via a language-independent abstract intermediate representation,
3. support for source code written in C and Fortan,
4. generate efficient tangent linear and adjoint for the MIT general circulation model.

OpenAD's homepage is at `http://www-unix.mcs.anl.gov/OpenAD/`. A development WIKI is at `http://wiki.mcs.anl.gov/OpenAD/index.php/Main_Page`. From the WIKI's main page, click on `Handling GCM` for various aspects pertaining to differentiating the MITgcm with OpenAD.

5.5.2 Downloading and installing OpenAD

The OpenAD webpage has a detailed description on how to download and build OpenAD. From its homepage, please click on `Download Test Binaries`. You may either download pre-built binaries for quick trial, or follow the detailed build process described at `http://www-unix.mcs.anl.gov/OpenAD/access.html`

5.5.3 Building MITgcm adjoint with OpenAD

17-January-2008

OpenAD was successfully built on head node of `itrda.acesgrid.org`, for following system:

```
> uname -a
Linux itrda 2.6.22.2-42.fc6 #1 SMP Wed Aug 15 12:34:26 EDT 2007 i686 i686 i386 GNU/Linux

> cat /proc/version
Linux version 2.6.22.2-42.fc6 (brewbuilder@hs20-bc2-4.build.redhat.com)
(gcc version 4.1.2 20070626 (Red Hat 4.1.2-13)) #1 SMP Wed Aug 15 12:34:26 EDT 2007

> module load ifc/9.1.036 icc/9.1.042
```

Head of MITgcm branch (checkpoint59m with some modif.s) was used for building adjoint code. Following routing needed special care (revert to revision 1.1):
`MITgcm_contrib/heimbach/OpenAD/OAD_support/active_module.f90`

Chapter 6

Physical Parameterizations - Packages I

In this chapter and in the following chapter, the MITgcm "packages" are described. While you can carry out many experiments with MITgcm by starting from case studies in section 3.8, 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 3.8 uses a particular combination of packages. Figure 6.1 shows the full set of packages that are available. As shown in the figure packages are classified into different groupings that layer on top of each other. The top layer packages are generally specialized to specific simulation types. In this layer there are packages that deal with biogeochemical processes, ocean interior and boundary layer processes, atmospheric processes, sea-ice, coupled simulations and state estimation. Below this layer are a set of general purpose numerical and computational packages. The general purpose numerical packages provide code for kernel numerical alogorithms that apply to many different simulation types. Similarly, the general purpose computational packages implement non-numerical alogorithms that provide parallelism, I/O and time-keeping functions that are used in many different scenarios.

The following sections describe the packages shown in figure 6.1. Section 6.1 describes the general procedure for using any package in MITgcm. Following that sections 6.2.1-7.4 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 6.1. Organizationally package code is assigned a separate subdirectory in the MITgcm code distribution (within the source code directory `pkg`). The name of this subdirectory is used as the package name in table 6.1.

Table 6.1:

.

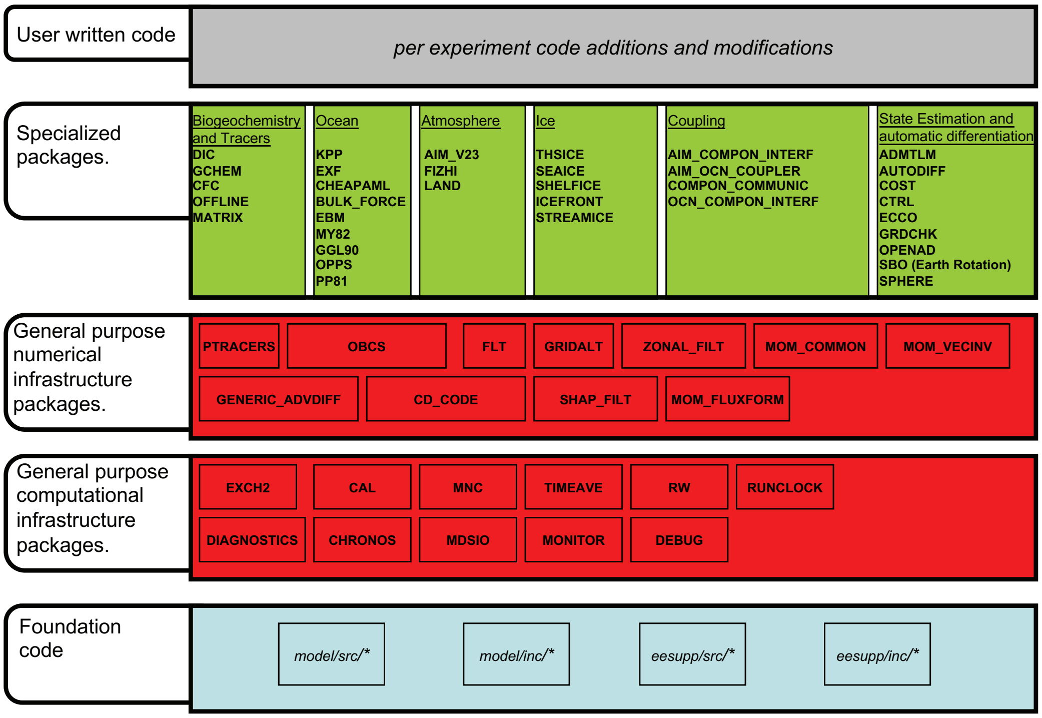


Figure 6.1: Hierarchy of code layers that are assembled to make up an MITgcm simulation. Conceptually (and in terms of code organization) MITgcm consists of several layers. At the base is a layer of core software that provides a basic numerical and computational foundation for MITgcm simulations. This 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.

6.1 Using MITgcm Packages

The set of packages that will be used within a partiucular model can be configured using a combination of both "compile–time" and "run–time" options. Compile–time options are those used to select which packages will be "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.

6.1.1 Package Inclusion/Exclusion

There are numerous ways that one can specify compile–time package inclusion or exclusion and they are all implemented by the `genmake2` program which was previously described in Section 3.4. The options are as follows:

1. Setting the `genamake2` options `--enable PKG` and/or `--disable PKG` specifies inclusion or exclusion. This method is intended as a convenient way to perform a single (perhaps for a quick test) compilation.

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

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

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

5. To help prevent users from creating unusable package groups, the `genmake2` program will parse the contents of the `pkg/pkg_depend` file to determine:

   - whether any two requested packages cannot be simultaneously included (*eg. seaice* and *thsice* are mutually exclusive),
   - whether additional packages must be included in order to satisfy package dependencies (*eg. rw* depends upon functionality within the *mdsio* package), and
   - whether the set of all requested packages is compatible with the dependencies (and producing an error if they aren't).

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

6.1.2 Package Activation

For run–time package control, MITgcm uses flags set through a `data.pkg` file. While some packages (*eg.* `debug`, `mnc`, `exch2`) may have their own usage conventions, most follow a simple flag naming convention of the form:

```
usePackageName=.TRUE.
```

where the `usePackageName` variable can activate or disable the package at runtime. As mentioned previously, packages must be included in order to be activated. Generally, such mistakes will be detected and reported as errors by the code. However, users should still be aware of the dependency.

6.1.3 Package Coding Standards

The following sections describe how to modify and/or create new MITgcm packages.

6.1.3.1 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.

6.1.3.2 File Inclusion Rules

Instead, packages should be viewed only as directories containing "sets of source files" that are built using some simple mechanisms provided by `genmake2`. Conceptually, the build process adds files as they are found and proceeds according to the following rules:

1. `genmake2` locates a "core" or main set of source files (the `-standarddirs` option sets these locations and the default value contains the directories `eesupp` and `model`).
2. `genmake2` then finds additional source files by inspecting the contents of each of the package directories:
   (a) As the new files are found, they are added to a list of source files.
   (b) If there is a file name "collision" (that is, if one of the files in a package has the same name as one of the files previously encountered) then the file within the newer (more recently visited) package will superseed (or "hide") any previous file(s) with the same name.
   (c) Packages are visited (and thus files discovered) *in the order that the packages are enabled* within `genmake2`. Thus, the files in `PackB` may superseed the files in `PackA` if `PackA` is enabled before `PackB`. Thus, package ordering can be significant! For this reason, `genmake2` honors the order in which packages are specified.

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

6.1.3.3 Conditional Compilation and `PACKAGES_CONFIG.h`

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

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

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

An example illustrating the use of these defines is:

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

which is included from the file `calc_diffusivity.F` and shows how both the compile–time `ALLOW_GMREDI` flag and the run–time `useGMRedi` are nested.

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

6.1.3.4 Package Startup or Boot Sequence

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

```
1. S/R PACKAGES_BOOT()
        :
    CALL OPEN_COPY_DATA_FILE( 'data.pkg', 'PACKAGES_BOOT', ... )


2. S/R PACKAGES_READPARMS()
        :
    #ifdef ALLOW_${PKG}
      if ( use${Pkg} )
 &       CALL ${PKG}_READPARMS( retCode )
    #endif

3. S/R PACKAGES_INIT_FIXED()
        :
    #ifdef ALLOW_${PKG}
      if ( use${Pkg} )
 &       CALL ${PKG}_INIT_FIXED( retCode )
    #endif

4. S/R PACKAGES_CHECK()
        :
    #ifdef ALLOW_${PKG}
      if ( use${Pkg} )
 &       CALL ${PKG}_CHECK( retCode )
    #else
      if ( use${Pkg} )
 &       CALL PACKAGES_CHECK_ERROR('${PKG}')
    #endif

5. S/R PACKAGES_INIT_VARIABLES()
        :
    #ifdef ALLOW_${PKG}
      if ( use${Pkg} )
 &       CALL ${PKG}_INIT_VARIA( )
    #endif

 6. S/R DO_THE_MODEL_IO

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

 7. S/R PACKAGES_WRITE_PICKUP()

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

6.1.3.5 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.

6.2 Packages Related to Hydrodynamical Kernel

6.2.1 Generic Advection/Diffusion

The `generic_advdiff` package contains high-level subroutines to solve the advection-diffusion equation of any tracer, either active (potential temperature, salinity or water vapor) or passive (see pkg/ptracers). (see also sections 2.16 to 2.19).

6.2.1.1 Introduction

Package "generic_advdiff" provides a common set of routines for calculating advective/diffusive fluxes for tracers (cell centered quantities on a C-grid).

Many different advection schemes are available: the standard centered second order, centered fourth order and upwind biased third order schemes are known as linear methods and require some stable time-stepping method such as Adams-Bashforth. Alternatives such as flux-limited schemes are stable in the forward sense and are best combined with the multi-dimensional method provided in gad_advection.

6.2.1.2 Key subroutines, parameters and files

There are two high-level routines:

- GAD_CALC_RHS calculates all fluxes at time level "n" and is used for the standard linear schemes. This must be used in conjuction with Adams–Bashforth time stepping. Diffusive and parameterized fluxes are always calculated here.
- GAD_ADVECTION calculates just the advective fluxes using the non-linear schemes and can not be used in conjuction with Adams–Bashforth time stepping.

6.2.1.3 GAD Diagnostics

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

6.2.1.4 Experiments and tutorials that use GAD

- Offline tutorial, in tutorial_offline verification directory, described in section 3.20
- Baroclinic gyre experiment, in tutorial_baroclinic_gyre verification directory, described in section 3.10
- Tracer Sensitivity tutorial, in tutorial_tracer_adjsens verification directory, described in section 3.19

6.2.2 Shapiro Filter

(in directory: `pkg/shap_filt/`)

6.2.2.1 Key subroutines, parameters and files

Implementation of *Shapiro* [1970] filter is described in section 2.20.

Name	Default value	Description	Reference
Shap_funct	2	select Shapiro filter function	
nShapT	4	power of Shapiro filter for Temperat	
nShapS	4	power of Shapiro filter for Salinity	
nShapUV	4	power of Shapiro filter for momentum	
nShapTrPhys	0	power of physical-space filter (Tracer)	
nShapUVPhys	0	power of physical-space filter (Momentum)	
Shap_Trtau	5.4E+03	time scale of Shapiro filter (Tracer)	
Shap_TrLength	0.0E+00	Length scale of Shapiro filter (Tracer)	
Shap_uvtau	1.8E+03	time scale of Shapiro filter (Momentum)	
Shap_uvLength	0.0E+00	Length scale of Shapiro filter (Momentum)	
Shap_noSlip	0.0E+00	No-slip parameter (0=Free-slip ; 1=No-slip)	
Shap_diagFreq	0.0E+00	$Frequency^-1$ for diagnostic output (s)	

6.2.2.2 Experiments and tutorials that use shap filter

- Held Suarez tutorial, in tutorial_held_suarez_cs verification directory, described in section 3.14
- 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`)

6.2.3 FFT Filtering Code

(in directory: `pkg/zonal_filt/`)

6.2.3.1 Key subroutines, parameters and files

6.2.3.2 Experiments and tutorials that use zonal filter

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

6.2.4 exch2: Extended Cubed Sphere Topology

6.2.4.1 Introduction

The `exch2` package extends the original cubed sphere topology configuration to allow more flexible domain decomposition and parallelization. Cube faces (also called subdomains) may be divided into any number of tiles that divide evenly into the grid point dimensions of the subdomain. Furthermore, the tiles can run on separate processors individually or in groups, which provides for manual compile-time load balancing across a relatively arbitrary number of processors.

The exchange parameters are declared in `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 32×32 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 6.2.4.3 *Generating Topology Files for exch2* 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.

6.2.4.2 Invoking exch2

To use exch2 with the cubed sphere, the following conditions must be met:

- The exch2 package is included when `genmake2` is run. The easiest way to do this is to add the line `exch2` to the `packages.conf` file – see Section 3.4 *Building the code* for general details.

- An example of `W2_EXCH2_TOPOLOGY.h` and `w2_e2setup.F` must reside in a directory containing files symbolically linked by the `genmake2` script. The safest place to put these is the directory indicated in the `-mods=DIR` command line modifier (typically `../code`), or the build directory. The default versions of these files reside in `pkg/exch2` and are linked automatically if no other versions exist elsewhere in the build path, but they should be left untouched to avoid breaking configurations other than the one you intend to modify.

- Files containing grid parameters, named `tile00`n`.mitgrid` where n=`(1:6)` (one per subdomain), must be in the working directory when the MITgcm executable is run. These files are provided in the example experiments for cubed sphere configurations with 32×32 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 6.2.4.3 *Generating Topology Files for exch2* and 6.2.4.4 *exch2, SIZE.h, and Multiprocessing*; a more general background on the subject relevant to MITgcm is presented in Section 4.3.1 *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
```

6.2.4.3 Generating Topology Files for exch2

Alternate cubed sphere topologies may be created using the Matlab scripts in `utils/exch2/matlab-topology-generator`. Running the m-file `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 6.4, 6.3, and 6.2 are examples of the generated diagrams. The other m-files in the directory are subroutines called from `driver.m` and should not be run "bare" except for

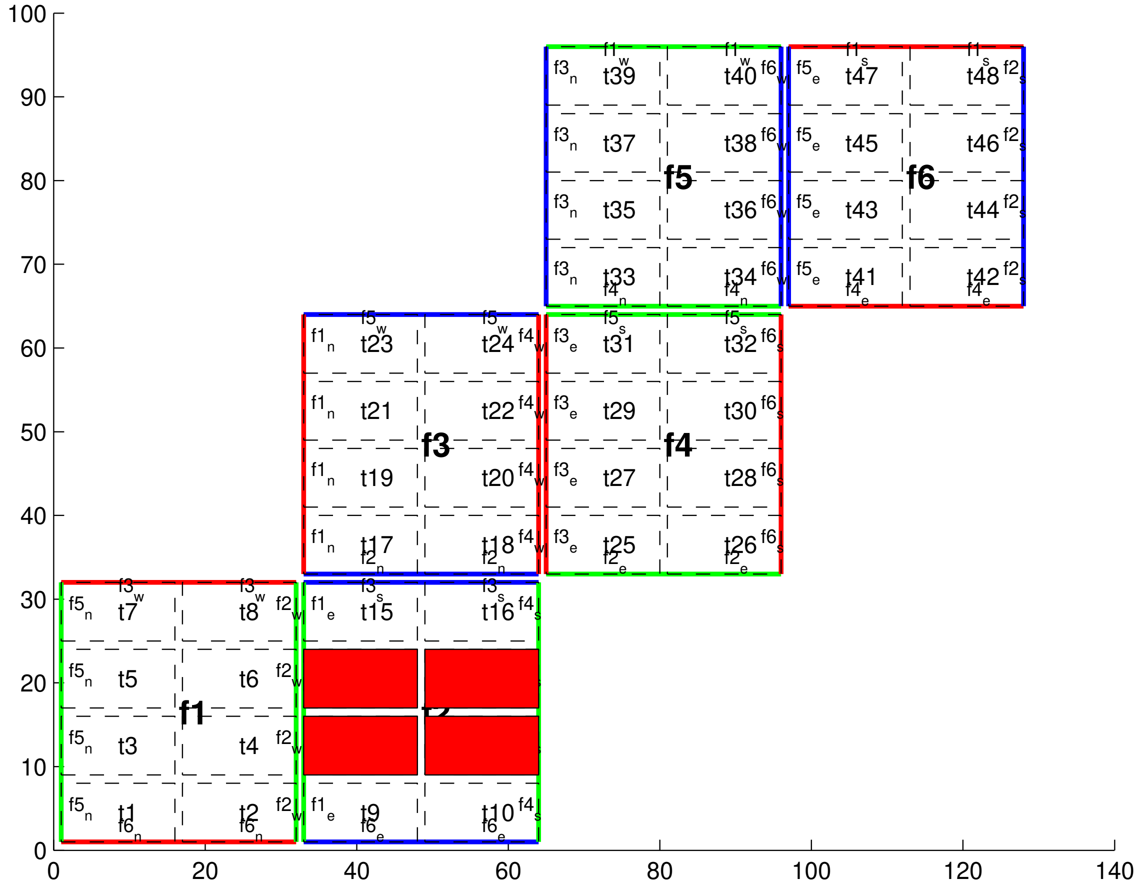


Figure 6.2: Plot of a cubed sphere topology with a 32×192 domain divided into six 32×32 subdomains, each of which is divided into eight tiles of width `tnx=16` and height `tny=8` for a total of forty-eight tiles. The colored borders of the subdomains represent the parameters `nr` (red), `ng` (green), and `nb` (blue). This tiling is used in the example verification/adjustment.cs-32x32x1/ with the option (blanklist.txt) to remove the land-only 4 tiles (11,12,13,14) which are filled in red on the plot.

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 6.2 shows one possible topology for a twenty-four-tile cube, and figure 6.4 shows one for six tiles.

Tiles can be selected from the topology to be omitted from being allocated memory and processors. This tuning is useful in ocean modeling for omitting tiles that fall entirely on land. The tiles omitted are specified in the file `blanklist.txt` by their tile number in the topology, separated by a newline.

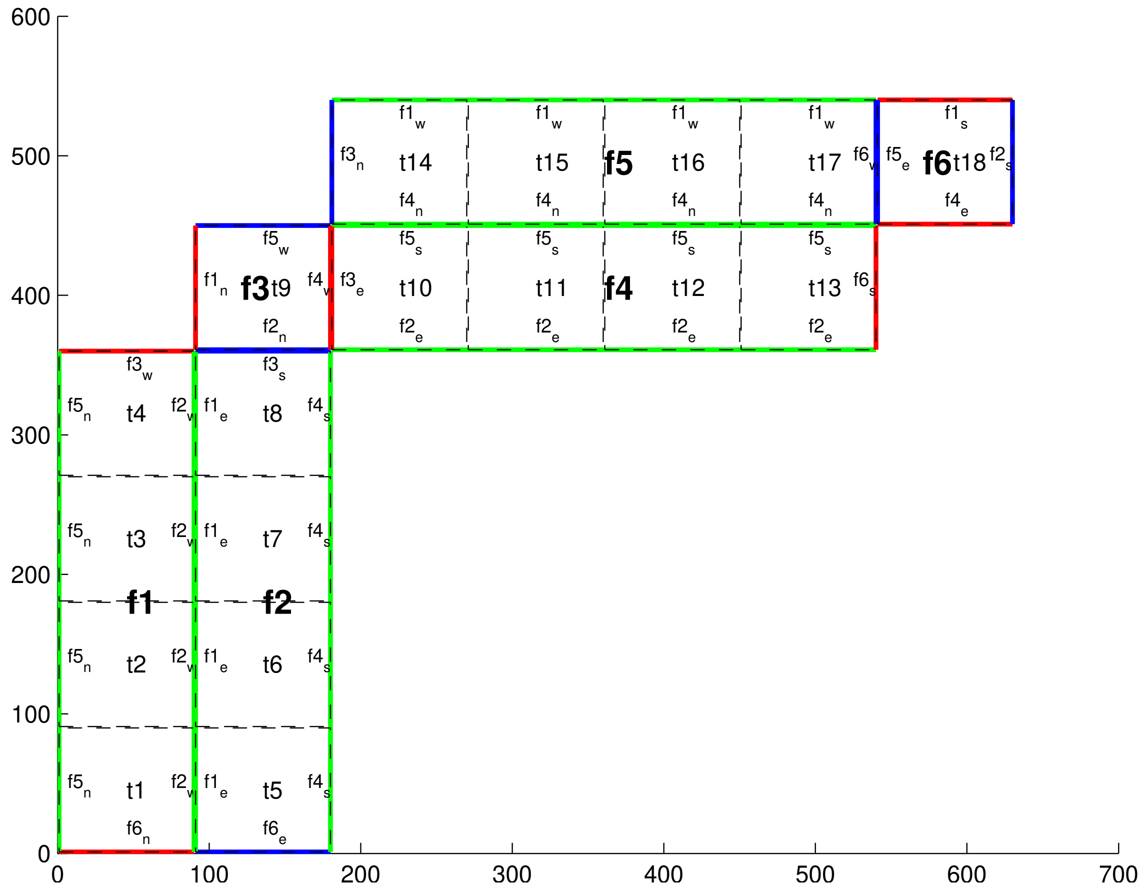


Figure 6.3: Plot of a non-square cubed sphere topology with 6 subdomains of different size (nr=90,ng=360,nb=90), divided into one to four tiles each (`tnx=90, tny=90`), resulting in a total of 18 tiles.

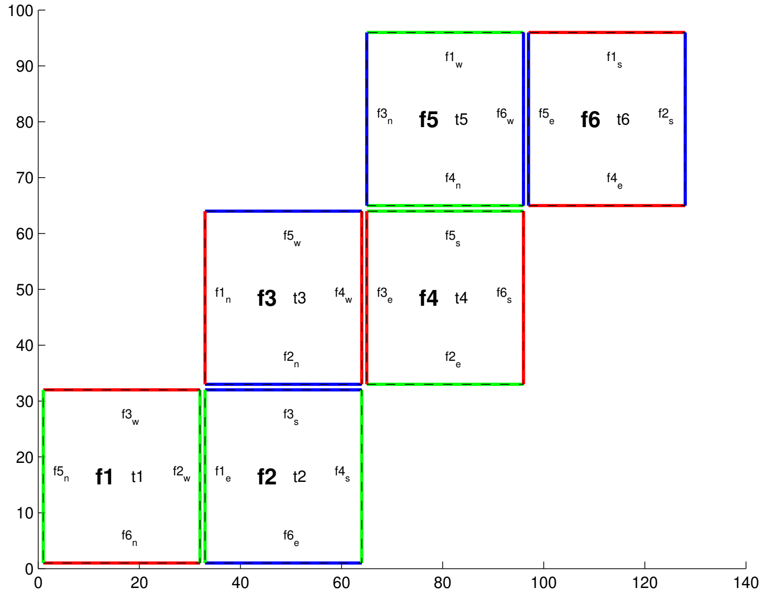


Figure 6.4: Plot of a cubed sphere topology with a 32×192 domain divided into six 32×32 subdomains with one tile each (`tnx=32, tny=32`). This is the default configuration.

6.2.4.4 exch2, SIZE.h, and Multiprocessing

Once the topology configuration files are created, the Fortran `PARAMETER`s in `SIZE.h` must be configured to match. Section 4.3.1 *Specifying a decomposition* provides a general description of domain decomposition within MITgcm and its relation to `SIZE.h`. The current section specifies constraints that the exch2 package imposes and describes how to enable parallel execution with MPI.

As in the general case, the parameters `sNx` and `sNy` define the size of the individual tiles, and so must be assigned the same respective values as `tnx` and `tny` in `driver.m`.

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

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

The number of tiles MITgcm allocates and how they are distributed between processors depends on `nPx` and `nSx`. `nSx` is the number of tiles per processor and `nPx` is the number of processors. The total number of tiles in the topology minus those listed in `blanklist.txt` must equal `nSx*nPx`. Note that in order to obtain maximum usage from a given number of processors in some cases, this restriction might entail sharing a processor with a tile that would otherwise be excluded because it is topographically outside of the domain and therefore in `blanklist.txt`. For example, suppose you have five processors and a domain decomposition of thirty-six tiles that allows you to exclude seven tiles. To evenly distribute the remaining twenty-nine tiles among five processors, you would have to run one "dummy" tile to make an even six tiles per processor. Such dummy tiles are *not* listed in `blanklist.txt`.

The following is an example of `SIZE.h` for the six-tile configuration illustrated in figure 6.4 running on one processor:

```
      PARAMETER (
     &           sNx =  32,
     &           sNy =  32,
     &           OLx =   2,
     &           OLy =   2,
     &           nSx =   6,
     &           nSy =   1,
     &           nPx =   1,
     &           nPy =   1,
     &           Nx  = sNx*nSx*nPx,
     &           Ny  = sNy*nSy*nPy,
     &           Nr  =   5)
```

The following is an example for the forty-eight-tile topology in figure 6.2 running on six processors:

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

6.2.4.5 Key Variables

The descriptions of the variables are divided up into scalars, one-dimensional arrays indexed to the tile number, and two and three-dimensional arrays indexed to tile number and neighboring tile. This division reflects the functionality of these variables: The scalars are common to every part of the topology, the tile-indexed arrays to individual tiles, and the arrays indexed by tile and neighbor to relationships between tiles and their neighbors.

Scalars:
The number of tiles in a particular topology is set with the parameter `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 `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. 6.4) has `exch2_domain_nxt=6` and `exch2_domain_nyt=1`. A topology of forty-eight tiles, eight per subdomain (as in figure 6.2), will have `exch2_domain_nxt=12` and `exch2_domain_nyt=4`. Note that these parameters express the tile layout in order to allow global data files that are tile-layout-neutral. They have no bearing on the internal storage of the arrays. The tiles are stored internally in a range from `bi=(1:NTILES)` in the x axis, and the y axis variable `bj` is assumed to equal `1` throughout the package.

Arrays indexed to tile number:
The following arrays are of length `NTILES` and are indexed to the tile number, which is indicated in the diagrams with the notation tn. The indices are omitted in the descriptions.

The arrays `exch2_tnx` and `exch2_tny` express the x and y dimensions of each tile. At present for each tile `exch2_tnx=sNx` and `exch2_tny=sNy`, as assigned in `SIZE.h` and described in Section 6.2.4.4 *exch2, SIZE.h, and Multiprocessing*. Future releases of MITgcm may allow varying tile sizes.

The arrays `exch2_tbasex` and `exch2_tbasey` determine the tiles' Cartesian origin within a subdomain and locate the edges of different tiles relative to each other. As an example, in the default six-tile topology (Fig. 6.4) each index in these arrays is set to `0` since a tile occupies its entire subdomain. The twenty-four-tile case discussed above will have values of `0` or `16`, depending on the quadrant of the tile within the subdomain. The elements of the arrays `exch2_txglobalo` and `exch2_txglobalo` are similar to `exch2_tbasex` and `exch2_tbasey`, but locate the tile edges within the global address space, similar to that used by global output and input files.

The array `exch2_myFace` contains the number of the subdomain of each tile, in a range `(1:6)` in the case of the standard cube topology and indicated by **f**n in figures 6.4 and 6.2. `exch2_nNeighbours` contains a count of the neighboring tiles each tile has, and sets the bounds for looping over neighboring tiles. `exch2_tProc` holds the process rank of each tile, and is used in interprocess communication.

The arrays `exch2_isWedge`, `exch2_isEedge`, `exch2_isSedge`, and `exch2_isNedge` are set to `1` if the indexed tile lies on the edge of its subdomain, `0` if not. The values are used within the topology generator to determine the orientation of neighboring tiles, and to indicate whether a tile lies on the corner of a subdomain. The latter case requires special exchange and numerical handling for the singularities at the eight corners of the cube.

Arrays Indexed to Tile Number and Neighbor:
The following arrays have vectors of length `MAX_NEIGHBOURS` and `NTILES` and describe the orientations between the the tiles.

The array `exch2_neighbourId(a,T)` holds the tile number `Tn` for each of the tile number `T`'s neighboring tiles `a`. The neighbor tiles are indexed `(1:exch2_nNeighbours(T))` in the order right to left on the north then south edges, and then top to bottom on the east then west edges.

The `exch2_opposingSend_record(a,T)` array holds the index `b` of the element in `exch2_neighbourId(b,Tn)` that holds the tile number `T`, given `Tn=exch2_neighborId(a,T)`. In other words,

```
exch2_neighbourId( exch2_opposingSend_record(a,T),
                   exch2_neighbourId(a,T) ) = T
```

This provides a back-reference from the neighbor tiles.

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

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

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

The arrays exch2_oi, exch2_oj, exch2_oi_f, and exch2_oj_f are indexed to tile number and neighbor and specify the relative offset within the subdomain of the array index of a variable going from a neighboring tile `N` to a local tile `T`. Consider `T=1` in the six-tile topology (Fig. 6.4), where

```
exch2_oi(1,1)=33
exch2_oi(2,1)=0
exch2_oi(3,1)=32
exch2_oi(4,1)=-32
```

The simplest case is `exch2_oi(2,1)`, the southern neighbor, which is `Tn=6`. The axes of `T` and `Tn` have the same orientation and their x axes have the same origin, and so an exchange between the two requires no changes to the x index. For the western neighbor (`Tn=5`), `code_oi(3,1)=32` since the `x=0` vector on `T` corresponds to the `y=32` vector on `Tn`. The eastern edge of `T` shows the reverse case (`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 `exch2_oi` is added to the original x index of `T` multiplied by the transformation factor `exch2_pi(t,N,T)`. Here `exch2_pi(1,1,1)=0` since the x axis of `T` is orthogonal to the x axis of `Tn`. `exch2_pi(2,1,1)=-1` since the x axis of `T` corresponds to the y axis of `Tn`, but the index is reversed. The result is that the index of the northern edge of `T`, which runs `(1:32)`, is transformed to `(-1:-32)`. `exch2_oi(1,1)` is then added to this range to get back `(32:1)` – the index of the y axis of `Tn` relative to `T`. This transformation may seem overly convoluted for the six-tile case, but it is necessary to provide a general solution for various topologies.

Finally, exch2_itlo_c, exch2_ithi_c, exch2_jtlo_c and exch2_jthi_c hold the location and index bounds of the edge segment of the neighbor tile `N`'s subdomain that gets exchanged with the local tile `T`. To take the example of tile `T=2` in the forty-eight-tile topology (Fig. 6.2):

```
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 `N=4`, indicating the western neighbor, which is `Tn=1`. `Tn` resides on the same subdomain as `T`, so the tiles have the same orientation and the same x and y axes. The x axis is orthogonal to the western edge and the tile is 16 points wide, so `exch2_itlo_c` and `exch2_ithi_c` indicate the column beyond `Tn`'s eastern edge, in that tile's halo region. Since the border of the tiles extends through the entire height of the subdomain, the y axis bounds `exch2_jtlo_c` to `exch2_jthi_c` cover the height of `(1:32)`, plus 1 in either direction to cover part of the halo.

For the north edge of the same tile `T=2` where `N=1` and the neighbor tile is `Tn=5`:

```
exch2_itlo_c(1,2)=0
exch2_ithi_c(1,2)=0
exch2_jtlo_c(1,2)=0
exch2_jthi_c(1,2)=17
```

`T`'s northern edge is parallel to the x axis, but since `Tn`'s y axis corresponds to `T`'s x axis, `T`'s northern edge exchanges with `Tn`'s western edge. The western edge of the tiles corresponds to the lower bound of the x axis, so `exch2_itlo_c` and `exch2_ithi_c` are `0`, in the western halo region of `Tn`. The range of `exch2_jtlo_c` and `exch2_jthi_c` correspond to the width of `T`'s northern edge, expanded by one into the halo.

6.2.4.6 Key Routines

Most of the subroutines particular to exch2 handle the exchanges themselves and are of the same format as those described in 4.3.3.3 *Cube sphere communication*. Like the original routines, they are written as templates which the local Makefile converts from `RX` into `RL` and `RS` forms.

The interfaces with the core model subroutines are `EXCH_UV_XY_RX`, `EXCH_UV_XYZ_RX` and `EXCH_XY_RX`. They override the standard exchange routines when `genmake2` is run with `exch2` option. They in turn call the local exch2 subroutines `EXCH2_UV_XY_RX` and `EXCH2_UV_XYZ_RX` for two and three-dimensional vector quantities, and `EXCH2_XY_RX` and `EXCH2_XYZ_RX` for two and three-dimensional scalar quantities. These subroutines set the dimensions of the area to be exchanged, call `EXCH2_RX1_CUBE` for scalars and `EXCH2_RX2_CUBE` for vectors, and then handle the singularities at the cube corners.

The separate scalar and vector forms of `EXCH2_RX1_CUBE` and `EXCH2_RX2_CUBE` reflect that the vector-handling subroutine needs to pass both the u and v components of the physical vectors. This swapping arises from the topological folding discussed above, where the x and y axes get swapped in some cases, and is not an issue with the scalar case. These subroutines call `EXCH2_SEND_RX1` and `EXCH2_SEND_RX2`, which do most of the work using the variables discussed above.

6.2.4.7 Experiments and tutorials that use exch2

- Held Suarez tutorial, in tutorial_held_suarez_cs verification directory, described in section 3.14

6.2.5 Gridalt - Alternate Grid Package

6.2.5.1 Introduction

The gridalt package [*Molod*, 2009] 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 (*Bushell and Martin* [1999]; *Inness et al.* [2001]; *Williamson and Olsen* [1998]; *Bretherton et al.* [1999]). Some of the benefit of increased resolution near the surface is realized by employing the higher resolution for the computation of the forcing due to turbulent and convective processes in the atmosphere.

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

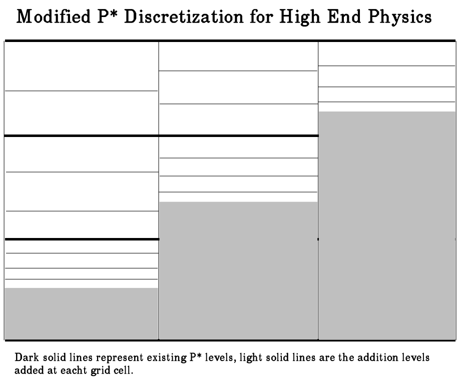


Figure 6.5: Vertical discretization for MITgcm (dark grey lines) and for the atmospheric physics (light grey lines). In this implementation, all MITgcm level interfaces must coincide with atmospheric physics level interfaces.

The algorithm presented here retains the state variables on the high resolution 'physics' grid as well as on the coarser resolution 'dynamics' grid, and ensures that the two estimates of the state 'agree' on the coarse resolution grid. It would have been possible to implement a technique in which the tendencies due to atmospheric physics are computed on the high resolution grid and the state variables are retained at low resolution only. This, however, for the case of the turbulence parameterization, would mean that the turbulent kinetic energy source terms, and all the turbulence terms that are written in terms of gradients of the mean flow, cannot really be computed making use of the fine structure in the vertical.

6.2.5.2 Equations on Both Grids

In addition to computing the physical forcing terms of the momentum, thermodynamic and humidity equations on the modified (higher resolution) grid, the higher resolution structure of the atmosphere (the boundary layer) is retained between physics calculations. This neccessitates a second set of evolution equations for the atmospheric state variables on the modified grid. If the equation for the evolution of U on p^* can be expressed as:

$$\left.\frac{\partial U}{\partial t}\right|_{p^*}^{total} = \left.\frac{\partial U}{\partial t}\right|_{p^*}^{dynamics} + \left.\frac{\partial U}{\partial t}\right|_{p^*}^{physics}$$

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

$$\left.\frac{\partial U}{\partial t}\right|_{p^{*m}}^{total} = \left.\frac{\partial U}{\partial t}\right|_{p^{*m}}^{dynamics} + \left.\frac{\partial U}{\partial t}\right|_{p^{*m}}^{physics} + \gamma(U|_{p^*} - U|_{p^{*m}})$$

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

6.2.5.3 Time stepping Sequence

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

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

6.2.5.4 Interpolation

In order to minimize the correction terms for the state variables on the alternative, higher resolution grid, the vertical interpolation scheme must be constructed so that a dynamics-to-physics interpolation can be exactly reversed with a physics-to-dynamics mapping. The simple scheme employed to achieve this is:

Coarse to fine: For all physics layers l in dynamics layer L, $T_{phys}(l) = \{T_{dyn}(L)\} = T_{dyn}(L)$.
Fine to coarse: For all physics layers l in dynamics layer L, $T_{dyn}(L) = [T_{phys}(l)] = \int T_{phys} dp$.

Where $\{\}$ is defined as the dynamics-to-physics operator and $[]$ is the physics-to-dynamics operator, T stands for any state variable, and the subscripts *phys* and *dyn* stand for variables on the physics and dynamics grids, respectively.

6.2.5.5 Key subroutines, parameters and files

One of the central elements of the gridalt package is the routine which is called from subroutine gridalt_initialise to define the grid to be used for the high end physics calculations. Routine make_phys_grid passes back the parameters which define the grid, ultimately stored in the common block gridalt_mapping.

```
       subroutine make_phys_grid(drF,hfacC,im1,im2,jm1,jm2,Nr,
     . Nsx,Nsy,i1,i2,j1,j2,bi,bj,Nrphys,Lbot,dpphys,numlevphys,nlperdyn)
c***********************************************************************
c Purpose: Define the grid that the will be used to run the high-end
c          atmospheric physics.
c
c Algorithm: Fit additional levels of some (~) known thickness in
c          between existing levels of the grid used for the dynamics
c
c Need:    Information about the dynamics grid vertical spacing
c
c Input:   drF         - delta r (p*) edge-to-edge
c          hfacC       - fraction of grid box above topography
c          im1, im2    - beginning and ending i - dimensions
c          jm1, jm2    - beginning and ending j - dimensions
c          Nr          - number of levels in dynamics grid
c          Nsx,Nsy     - number of processes in x and y direction
c          i1, i2      - beginning and ending i - index to fill
c          j1, j2      - beginning and ending j - index to fill
c          bi, bj      - x-dir and y-dir index of process
c          Nrphys      - number of levels in physics grid
c
c Output:  dpphys      - delta r (p*) edge-to-edge of physics grid
c          numlevphys  - number of levels used in the physics
c          nlperdyn    - physics level number atop each dynamics layer
c
c NOTES: 1) Pressure levs are built up from bottom, using p0, ps and dp:
c              p(i,j,k)=p(i,j,k-1) + dp(k)*ps(i,j)/p0(i,j)
c        2) Output dp's are aligned to fit EXACTLY between existing
c           levels of the dynamics vertical grid
c        3) IMPORTANT! This routine assumes the levels are numbered
c           from the bottom up, ie, level 1 is the surface.
c           IT WILL NOT WORK OTHERWISE!!!
c        4) This routine does NOT work for surface pressures less
c           (ie, above in the atmosphere) than about 350 mb
c***********************************************************************
```

In the case of the grid used to compute the atmospheric physical forcing (fizhi package), the locations of the grid points move in time with the MITgcm p^* coordinate, and subroutine gridalt_update is called during the run to update the locations of the grid points:

```
       subroutine gridalt_update(myThid)
c***********************************************************************
c Purpose: Update the pressure thicknesses of the layers of the
c          alternative vertical grid (used now for atmospheric physics).
c
c Calculate: dpphys    - new delta r (p*) edge-to-edge of physics grid
c                        using dpphys0 (initial value) and rstarfacC
c***********************************************************************
```

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

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

```
C Purpose:
C   To interpolate an arbitrary quantity from the 'dynamics' eta (pstar)
C               grid to the higher resolution physics grid
C Algorithm:
C   Routine works one layer (edge to edge pressure) at a time.
C   Dynamics -> Physics retains the dynamics layer mean value,
C   weights the field either with the profile of the physics grid
C   wind speed (for U and V fields), or uniformly (T and Q)
C
C Input:
C   qdyn..... [im,jm,lmdyn] Arbitrary Quantity on Input Grid
C   pedyn.... [im,jm,lmdyn+1] Pressures at bottom edges of input levels
C   im1,2 ... Limits for Longitude Dimension of Input
C   jm1,2 ... Limits for Latitude  Dimension of Input
C   lmdyn.... Vertical  Dimension of Input
C   Nsx...... Number of processes in x-direction
C   Nsy...... Number of processes in y-direction
C   idim1,2.. Beginning and ending i-values to calculate
C   jdim1,2.. Beginning and ending j-values to calculate
C   bi....... Index of process number in x-direction
C   bj....... Index of process number in x-direction
C   windphy.. [im,jm,lmphy] Magnitude of the wind on the output levels
C   pephy.... [im,jm,lmphy+1] Pressures at bottom edges of output levels
C   lmphy.... Vertical  Dimension of Output
C   nlperdyn. [im,jm,lmdyn] Highest Physics level in each dynamics level
C   flg...... Flag to indicate field type (0 for T or Q, 1 for U or V)
C
C Output:
C   qphy..... [im,jm,lmphy] Quantity at output grid (physics grid)
C
C Notes:
C   1) This algorithm assumes that the output (physics) grid levels
C      fit exactly into the input (dynamics) grid levels
C************************************************************************
```

And similarly, gridalt contains subroutine phys2dyn.

6.2.5.6 Gridalt Diagnostics

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

6.2.5.7 Dos and donts

6.2.5.8 Gridalt Reference

6.2.5.9 Experiments and tutorials that use gridalt

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

6.3 General purpose numerical infrastructure packages

6.3.1 OBCS: Open boundary conditions for regional modeling

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

6.3.1.1 Introduction

The OBCS-package is fundamental to regional ocean modelling with the MITgcm, but there are so many details to be considered in regional ocean modelling that this package cannot accomodate all imaginable and possible options. Therefore, for a regional simulation with very particular details, it is recommended to familiarize oneself not only with the compile- and runtime-options of this package, but also with the code itself. In many cases it will be necessary to adapt the obcs-code (in particular `S/R OBCS_CALC`) to the application in question; in these cases the obcs-package (together with the rbcs-package, section 6.3.2) is a very useful infrastructure for implementing special regional models.

6.3.1.2 OBCS configuration and compiling

As with all MITgcm packages, OBCS can be turned on or off at compile time

- using the `packages.conf` file by adding `obcs` to it,
- or using `genmake2` adding `-enable=obcs` or `-disable=obcs` switches
- *Required packages and CPP options:*
  To alternatives are available for prescribing open boundary values, which differ in the way how OB's are treated in time: A simple time-management (e.g. constant in time, or cyclic with fixed fequency) is provided through S/R `obcs_external_fields_load`. More sophisticated "real-time" (i.e. calendar time) management is available through `obcs_prescribe_read`. The latter case requires packages `cal` and `exf` to be enabled.

(see also Section 3.4).

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 6.3.1.2 summarizes them.

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

Table 6.2:

6.3.1.3 Run-time parameters

Run-time parameters are set in files `data.pkg`, `data.obcs`, and `data.exf` if "real-time" prescription is requested (i.e. package `exf` enabled). vThese 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: (i) switching on/off the package at runtime, (ii) OBCS package flags and parameters, (iii) 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 6.3 summarizes the runtime flags that are set in `data.obcs`, and their default values.

Flag/parameter	default	Description
		basic flags & parameters (OBCS_PARM01)
OB_Jnorth	0	Nx-vector of J-indices (w.r.t. Ny) of Northern OB at each I-position (w.r.t. Nx)
OB_Jsouth	0	Nx-vector of J-indices (w.r.t. Ny) of Southern OB at each I-position (w.r.t. Nx)
OB_Ieast	0	Ny-vector of I-indices (w.r.t. Nx) of Eastern OB at each J-position (w.r.t. Ny)
OB_Iwest	0	Ny-vector of I-indices (w.r.t. Nx) of Western OB at each J-position (w.r.t. Ny)
useOBCSprescribe	`.FALSE.`	
useOBCSsponge	`.FALSE.`	
useOBCSbalance	`.FALSE.`	
OBCS_balanceFacN/S/E/W	1	factor(s) determining the details of the balaning code
useOrlanskiNorth/South/EastWest	`.FALSE.`	turn on Orlanski boundary conditions for individual boundary
useStevensNorth/South/EastWest	`.FALSE.`	turn on Stevens boundary conditions for individual boundary
OB**Xy**File		file name of OB field **X**: **N**(orth), **S**(outh), **E**(ast), **W**(est) **y**: **t**(emperature), **s**(salinity), **u**(-velocity), **v**(-velocity), **w**(-velocity), **eta**(sea surface height) **a**(sea ice area), **h**(sea ice thickness), **sn**(snow thickness), **sl**(sea ice salinity)
		Orlanski parameters (OBCS_PARM02)
cvelTimeScale	2000 sec	averaging period for phase speed
CMAX	0.45 m/s	maximum allowable phase speed-CFL for AB-II
CFIX	0.8 m/s	fixed boundary phase speed
useFixedCEast	`.FALSE.`	
useFixedCWest	`.FALSE.`	
		Sponge-layer parameters (OBCS_PARM03)
spongeThickness	0	sponge layer thickness (in # grid points)
Urelaxobcsinner	0 sec	relaxation time scale at the innermost sponge layer point of a meridional OB
Vrelaxobcsinner	0 sec	relaxation time scale at the innermost sponge layer point of a zonal OB
Urelaxobcsbound	0 sec	relaxation time scale at the outermost sponge layer point of a meridional OB
Vrelaxobcsbound	0 sec	relaxation time scale at the outermost sponge layer point of a zonal OB
		Stevens parameters (OBCS_PARM04)
T/SrelaxStevens	0 sec	relaxation time scale for temperature/salinity
useStevensPhaseVel	`.TRUE.`	
useStevensAdvection	`.TRUE.`	

Table 6.3: pkg OBCS run-time parameters

6.3.1.4 Defining open boundary positions

There are four open boundaries (OBs), a Northern, Southern, Eastern, and Western. All OB locations are specified by their absolute meridional (Northern/Southern) or zonal (Eastern/Western) indices. Thus, for each zonal position $i = 1, \ldots, N_x$ a meridional index j specifies the Northern/Southern OB position, and for each meridional position $j = 1, \ldots, N_y$, a zonal index i specifies the Eastern/Western OB position. For Northern/Southern OB this defines an N_x-dimensional "row" array `OB_Jnorth(Nx)` / `OB_Jsouth(Nx)`, and an N_y-dimenisonal "column" array `OB_Ieast(Ny)` / `OB_Iwest(Ny)`. Positions determined in this way allows Northern/Southern OBs to be at variable j (or y) positions, and Eastern/Western OBs at variable i (or x) positions. Here, indices refer to tracer points on the C-grid. A zero (0) element in `OB_I`..., `OB_J`... means there is no corresponding OB in that column/row. For a Northern/Southern OB, the OB V point is to the South/North. For an Eastern/Western OB, the OB U point is to the West/East. For example,

`OB_Jnorth(3)=34` means that:

	`T(3,34)`	is a an OB point
	`U(3,34)`	is a an OB point
	`V(3,34)`	is a an OB point

`OB_Jsouth(3)=1` means that:

	`T(3,1)`	is a an OB point
	`U(3,1)`	is a an OB point
	`V(3,2)`	is a an OB point

`OB_Ieast(10)=69` means that:

	`T(69,10)`	is a an OB point
	`U(69,10)`	is a an OB point
	`V(69,10)`	is a an OB point
`OB_Iwest(10)=1`	means that:	
	`T(1,10)`	is a an OB point
	`U(2,10)`	is a an OB point
	`V(1,10)`	is a an OB point

For convenience, negative values for `Jnorth`/`Ieast` refer to points relative to the Northern/Eastern edges of the model eg. `OB_Jnorth`$(3) = -1$ means that the point $(3, \texttt{Ny})$ 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 egdes of the domain, the simplest way of specifying the boundary points in `data.obcs` is:

```
  OB_Ieast = 144*-1,
# or OB_Ieast = 144*120,
  OB_Iwest = 144*1,
  OB_Jnorth = 120*-1,
# or OB_Jnorth = 120*144,
  OB_Jsouth = 120*1,
```

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

```
  OB_Jsouth(1:50) = 50*1,
```

Add special comments for case #define NONLIN_FRSURF, see obcs_ini_fixed.F

6.3.1.5 Equations and key routines

OBCS_READPARMS:
Set OB positions through arrays `OB_Jnorth(Nx), OB_Jsouth(Nx), OB_Ieast(Ny), OB_Iwest(Ny)`, and runtime flags (see Table 6.3).

OBCS_CALC:
Top-level routine for filling values to be applied at OB for T, S, U, V, η into corresponding "slice" arrays (x, z), (y, z) for each OB: `OB[N/S/E/W][t/s/u/v]`; e.g. for salinity array at Southern OB, array name is `OBSt`. Values filled are either

- constant vertical T, S profiles as specified in file `data` (`tRef(Nr), sRef(Nr)`) with zero velocities U, V,
- T, S, U, V values determined via Orlanski radiation conditions (see below),
- prescribed time-constant or time-varying fields (see below).
- use prescribed boundary fields to compute Stevens boundary conditions.

ORLANSKI:
Orlanski radiation conditions [*Orlanski*, 1976], examples can be found in `verification/dome` and `verification/tutorial` (3.16).

OBCS_PRESCRIBE_READ:
When `useOBCSprescribe = .TRUE.` the model tries to read temperature, salinity, u- and v-velocities from files specified in the runtime parameters `OB[N/S/E/W][t/s/u/v]File`. These files are the usual IEEE, big-endian files with dimensions of a section along an open boundary:

- For North/South boundary files the dimensions are ($N_x \times N_r \times$time levels), for East/West boundary files the dimensions are ($N_y \times N_r \times$ time levels).
- If a non-linear free surface is used (2.10.2), additional files `OB[N/S/E/W]etaFile` for the sea surface height η with dimension ($N_{x/y} \times$ time levels) may be specified.

- If non-hydrostatic dynamics are used (2.9), additional files `OB[N/S/E/W]wFile` for the vertical velocity w with dimensions ($N_{x/y} \times N_r \times$ time levels) can be specified.

- If `useSEAICE=.TRUE.` then additional files `OB[N/S/E/W][a,h,sl,sn,uice,vice]` for sea ice area, thickness (`HEFF`), seaice salinity, snow and ice velocities ($N_{x/y} \times$ time levels) can be specified.

As in `S/R external_fields_load` or the `exf`-package, the code reads two time levels for each variable, e.g. `OBNu0` and `OBNu1`, and interpolates linearly between these time levels to obtain the value `OBNu` at the current model time (step). When the `exf`-package is used, the time levels are controlled for each boundary separately in the same way as the `exf`-fields in `data.exf`, namelist `EXF_NML_OBCS`. The runtime flags follow the above naming conventions, e.g. for the western boundary the corresponding flags are `OBCWstartdate1/2` and `OBCWperiod`. Sea-ice boundary values are controlled separately with `siobWstartdate1/2` and `siobWperiod`. When the `exf`-package is not used, the time levels are controlled by the runtime flags `externForcingPeriod` and `externForcingCycle` in `data`, see `verification/exp4` for an example.

OBCS_CALC_STEVENS:
(THE IMPLEMENTATION OF THESE BOUNDARY CONDITIONS IS NOT COMPLETE. PASSIVE TRACERS, SEA ICE AND NON-LINEAR FREE SURFACE ARE NOT SUPPORTED PROPERLY.)
The boundary conditions following *Stevens* [1990] require the vertically averaged normal velocity (originally specified as a stream function along the open boundary) $\bar{u}_{ob}$ and the tracer fields χ_{ob} (note: passive tracers are currently not implemented and the code stops when package `ptracers` is used together with this option). Currently, the code vertically averages the normal velocity as specified in `OB[E,W]u` or `OB[N,S]v`. From these prescribed values the code computes the boundary values for the next timestep $n+1$ as follows (as an example, we use the notation for an eastern or western boundary):

- $u^{n+1}(y,z) = \bar{u}_{ob}(y) + (u')^n(y,z)$, where $(u')^n$ is the deviation from the vertically averaged velocity at timestep n on the boundary. $(u')^n$ is computed in the previous time step n from the intermediate velocity u^* prior to the correction step (see section 2.2, e.g., eq. (2.17)). (This velocity is not available at the beginning of the next time step $n+1$, when S/R OBCS_CALC/OBCS_CALC_STEVENS are called, therefore it needs to be saved in S/R DYNAMICS by calling S/R OBCS_SAVE_UV_N and also stored in a separate restart files `pickup_stevens[N/S/E/W].${iteration}.data`)

- If u^{n+1} is directed into the model domain, the boudary value for tracer χ is restored to the prescribed values:
$$\chi^{n+1} = \chi^n + \frac{\Delta t}{\tau_\chi}(\chi_{ob} - \chi^n),$$
where τ_χ is the relaxation time scale `T/SrelaxStevens`. The new χ^{n+1} is then subject to the advection by u^{n+1}.

- If u^{n+1} is directed out of the model domain, the tracer χ^{n+1} on the boundary at timestep $n+1$ is estimated from advection out of the domain with $u^{n+1} + c$, where c is a phase velocity estimated as $\frac{1}{2}\frac{\partial \chi}{\partial t}/\frac{\partial \chi}{\partial x}$. The numerical scheme is (as an example for an eastern boundary):
$$\chi^{n+1}_{i_b,j,k} = \chi^n_{i_b,j,k} + \Delta t(u^{n+1} + c)_{i_b,j,k}\frac{\chi^n_{i_b,j,k} - \chi^n_{i_b-1,j,k}}{\Delta x^C_{i_b,j}},\ \text{if}\ u^{n+1}_{i_b,j,k} > 0,$$
where i_b is the boundary index.
For test purposes, the phase velocity contribution or the entire advection can be turned off by setting the corresponding parameters `useStevensPhaseVel` and `useStevensAdvection` to `.FALSE.`.

See *Stevens* [1990] for details. With this boundary condition specifying the exact net transport across the open boundary is simple, so that balancing the flow with (S/R OBCS_BALANCE_FLOW, see next paragraph) is usually not necessary.

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

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

```
OBCS_balanceFacE = 1.,
OBCS_balanceFacW = -1.,
OBCS_balanceFacN = 2.,
OBCS_balanceFacS = 0.,
```

make the model

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

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

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

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

OBCS_APPLY_*:

OBCS_SPONGE:
The sponge layer code (turned on with `ALLOW_OBCS_SPONGE` and `useOBCSsponge`) adds a relaxation term to the right-hand-side of the momentum and tracer equations. The variables are relaxed towards the boundary values with a relaxation time scale that increases linearly with distance from the boundary

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

where χ is the model variable (U/V/T/S) in the interior, χ_{BC} the boundary value, L the thickness of the sponge layer (runtime parameter `spongeThickness` in number of grid points), $\delta L \in [0, L]$ ($\frac{\delta L}{L} = l \in [0,1]$) the distance from the boundary (also in grid points), and τ_b (runtime parameters `Urelaxobcsbound` and `Vrelaxobcsbound`) and τ_i (runtime parameters `Urelaxobcsinner` and `Vrelaxobcsinner`) the relaxation time scales on the boundary and at the interior termination of the sponge layer. The parameters `Urelaxobcsbound/inner` set the relaxation time scales for the Eastern and Western boundaries, `Vrelaxobcsbound/inner` for the Northern and Southern boundaries.

OB's with nonlinear free surface

6.3.1.6 Flow chart

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

6.3.1.7 OBCS diagnostics

Diagnostics output is available via the diagnostics package (see Section 7.1). Available output fields are summarized in Table 6.3.1.7.

```
------------------------------------------------------
 <-Name->|Levs|grid|<--  Units   -->|<- Tile (max=80c)
------------------------------------------------------
```

Table 6.4:

6.3.1.8 Reference experiments

In the directory `verifcation`, the following experiments use `obcs`:

- `exp4`: box with 4 open boundaries, simulating flow over a Gaussian bump based on *Adcroft et al.* [1997], also tests Stevens-boundary conditions;
- `dome`: based on the project "Dynamics of Overflow Mixing and Entrainment" (`http://www.rsmas.miami.edu/personal/ta` uses Orlanski-BCs;
- `internal_wave`: uses a heavily modified `S/R OBCS_CALC`
- `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 3.16.

6.3.1.9 References

6.3.1.10 Experiments and tutorials that use obcs

- `tutorial_plume_on_slope` (section 3.16)

6.3.2 RBCS Package

6.3.2.1 Introduction

A package which provides the flexibility to relax fields (temperature, salinity, ptracers) in any 3-D location: so could be used as a sponge layer, or as a "source" anywhere in the domain.
For a tracer (T) at every grid point the tendency is modified so that:

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

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

A seperate mask can be used for T,S and ptracers and each of these can be relaxed or not and can have its own timescale τ_T. These are set in data.rbcs (see below).

6.3.2.2 Key subroutines and parameters

The only compile-time parameter you are likely to have to change is in RBCS.h, the number of masks, PARAMETER(maskLEN = 3), see below.

The runtime parameters are set in *data.rbcs*:

Set in RBCS_PARM01:
• **rbcsForcingPeriod**: time interval between forcing fields (in seconds), zero means constant-in-time forcing.
• **rbcsForcingCycle**: repeat cycle of forcing fields (in seconds), zero means non-cyclic forcing.
• **rbcsForcingOffset**: time offset of forcing fields (in seconds, default 0); this is relative to time averages starting at $t = 0$, i.e., the first forcing record/file is placed at 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 (τ_T in equation above). Where mask is 1, relax rate will be 1/tauRelaxT. Default is 1.
• **tauRelaxS**: same for salinity.
• **relaxMaskFile(irbc)**: filename of 3-D file with mask (M_{rbc} in equation above. Need a file for each irbc. 1=temperature, 2=salinity, 3=ptracer01, 4=ptracer02 etc. If the mask numbers end (see maskLEN) are less than the number tracers, then relaxMaskFile(maskLEN) is used for all remaining ptracers.
• **relaxTFile**: name of file where temperatures that need to be relaxed to (T_{rbc} in equation above) are stored. The file must contain 3-D records to match the model domain. If rbcsSingleTimeFiles=F, it must have one record for each forcing period. If T, there must be a separate file for each period and a 10-digit iteration number is appended to the file name (see Table 6.5 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.

	rbcsSingleTimeFiles = T		F
	$c = 0$	$c \neq 0$	$c \neq 0$
model time	**file number**	**file number**	**record**
$t_0 - p/2$	i_0	$i_0 + c/\Delta t_{\rm rbcs}$	c/p
$t_0 + p/2$	$i_0 + p/\Delta t_{\rm rbcs}$	$i_0 + p/\Delta t_{\rm rbcs}$	1
$t_0 + p + p/2$	$i_0 + 2p/\Delta t_{\rm rbcs}$	$i_0 + 2p/\Delta t_{\rm rbcs}$	2
...	...	...	...
$t_0 + c - p/2$	...	$i_0 + c/\Delta t_{\rm rbcs}$	c/p
...	...	...	...

where

$$
\begin{aligned}
p &= \text{rbcsForcingPeriod} \\
c &= \text{rbcsForcingCycle} \\
t_0 &= \text{rbcsForcingOffset} \\
i_0 &= \text{rbcsIter0} \\
\Delta t_{\rm rbcs} &= \text{deltaTrbcs}
\end{aligned}
$$

Table 6.5: Timing of relaxation forcing fields.

6.3.2.3 Timing of relaxation forcing fields

For constant-in-time relaxation, set rbcsForcingPeriod=0. For time-varying relaxation, Table 6.5 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.

6.3.2.4 Example 1: forcing with time averages starting at $t = 0$

Cyclic data in a single file. Set rbcsSingleTimeFiles=F and rbcsForcingOffset=0, and the model will start by interpolating the last and first records of rbcs data, placed at $-p/2$ and $p/2$, resp., as appropriate for fields averaged over the time intervals $[-p, 0]$ and $[0, p]$.

Non-cyclic data, multiple files. Set rbcsForcingCycle=0 and rbcsSingleTimeFiles=T. With rbcsForcingOffset=0, rbcsIter0=0 and deltaTrbcs=rbcsForcingPeriod, the model would then start by interpolating data from files relax*File.0000000000.data and relax*File.0000000001.data, . . . , again placed at $-p/2$ and $p/2$.

6.3.2.5 Example 2: forcing with snapshots starting at $t = 0$

Cyclic data in a single file. Set rbcsSingleTimeFiles=F and rbcsForcingOffset=$-p/2$, and the model will start forcing with the first record at $t = 0$.

Non-cyclic data, multiple files. Set rbcsForcingCycle=0 and rbcsSingleTimeFiles=T. In this case, it is more natural to set rbcsForcingOffset=$+p/2$. With rbcsIter0=0 and deltaTrbcs=rbcsForcingPeriod, the model would then start with data from files relax*File.0000000000.data at $t = 0$. It would then proceed to interpolate between this file and files relax*File.0000000001.data at t = rbcsForcingPeriod.

6.3.2.6 Do's and Don'ts

6.3.2.7 Reference Material

6.3.2.8 Experiments and tutorials that use rbcs

In the directory `verifcation`, the following experiments use `rbcs`:

- `exp4`: box with 4 open boundaries, simulating flow over a Gaussian bump based on *Adcroft et al.* [1997].

6.3.3 PTRACERS Package

6.3.3.1 Introduction

This is a "passive" tracer package. Passive here means that the tracers don't affect the density of the water (as opposed to temperature and salinity) so no not actively affect the physics of the ocean. Tracers are initialized, advected, diffused and various outputs are taken care of in this package. For methods to add additional sources and sinks of tracers use the pkg/gchem (section 6.8.1).

Can use up tp 3843 tracers. But can not use pkg/diagnostics with more than about 90 tracers. Use utils/matlab/ioLb2num.m and num2ioLb.m to find correspondence between tracer number and tracer designation in the code for more than 99 tracers (since tracers only have two digit designations).

6.3.3.2 Equations

6.3.3.3 Key subroutines and parameters

The only code you shoul dhave to modify is: **PTRACERS_SIZE.h** where you need to set in the number of tracers to be used in the experiment: PTRACERS_num.

bf RUN TIME PARAMETERS SET IN **data.ptracers**:
• **PTRACERS_Iter0** which is the integer timestep when the tracer experiment is initialized. If nIter0 = PTRACERS_Iter0 then the tracers are initialized to zero or from initial files. If nIter0 > PTRACERS_Iter0 then tracers (and previous timestep tendency terms) are read in from a the ptracers pickup file. Note that tracers of zeros will be carried around if nIter0 < PTRACERS_Iter0.
• **PTRACERS_numInUse**: number of tracers to be used in the run (needs to be <= PTRACERS_num set in PTRACERS_SIZE.h)
• **PTRACERS_dumpFreq**: defaults to dumpFreq (set in data)
• **PTRACERS_taveFreq**: defaults to taveFreq (set in data)
• **PTRACERS_monitorFreq**: defaults to monitorFreq (set in data)
• **PTRACERS_timeave_mnc**: needs useMNC, timeave_mnc, default to false
• **PTRACERS_snapshot_mnc**: needs useMNC, snapshot_mnc, default to false
• **PTRACERS_monitor_mnc**: needs useMNC, monitor_mnc, default to false
• **PTRACERS_pickup_write_mnc**: needs useMNC, pickup_write_mnc, default to false
• **PTRACERS_pickup_read_mnc**: needs useMNC, pickup_read_mnc, default to false
• **PTRACERS_useRecords**: defaults to false. If true, will write all tracers in a single file, otherwise each tracer in a seperate file.

The following can be set for each tracer (tracer number iTrc):
• **PTRACERS_advScheme(iTrc)** will default to saltAdvScheme (set in data). For other options see Table 2.2.
• **PTRACERS_ImplVertAdv(iTrc)**: implicit vertical advection flag, default to .FALSE.
• **PTRACERS_diffKh(iTrc)**: horizontal Laplacian Diffusivity, dafaults to diffKhS (set in data).
• **PTRACERS_diffK4(iTrc)**: Biharmonic Diffusivity, defaults to diffK4S (set in data).
• **PTRACERS_diffKr(iTrc)**: vertical diffusion, defaults to un-set.
• **PTRACERS_diffKrNr(k,iTrc)**: level specific vertical diffusion, defaults to diffKrNrS. Will be set to PTRACERS_diffKr if this is set.
• **PTRACERS_ref(k,iTrc)**: reference tracer value for each level k, defaults to 0. Currently only used for dilution/concentration of tracers at surface if PTRACERS_EvPrRn(iTrc) is set and convertFW2Salt (set in data) is set to something other than -1 (note default is convertFW2Salt=35).
• **PTRACERS_EvPrRn(iTrc)**: tracer concentration in freshwater. Needed for calculation of dilution/concentration in surface layer due to freshwater addition/evaporation. Defaults to un-set in which case no dilution/concentration occurs.
• **PTRACERS_useGMRedi(iTrc)**: apply GM or not. Defaults to useGMREdi.
• **PTRACERS_useKPP(iTrc)**: apply KPP or not. Defaults to useKPP.
• **PTRACERS_initialFile(iTrc)**: file with initial tracer concentration. Will be used if PTRACERS_Iter0 = nIter0. Default is no name, in which case tracer is initialised as zero. If PTRACERS_Iter0

< nIter0, then tracer concentration will come from pickup_ptracer.
• **PTRACERS_names(iTrc)**: tracer name. Needed for netcdf. Defaults to nothing.
• **PTRACERS_long_names(iTrc)**: optional name in long form of tracer.
• **PTRACERS_units(iTrc)**: optional units of tracer.

6.3.3.4 PTRACERS Diagnostics

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

```
------------------------------------------------------------------------
<-Name->|Levs|<-parsing code->|<--  Units   -->|<- Tile (max=80c)
------------------------------------------------------------------------
TRAC01  | 15 |SM P    MR      |mol C/m         |Mass-Weighted Dissolved Inorganic Carbon
UTRAC01 | 15 |UU   171MR      |mol C/m.m/s     |Zonal Mass-Weighted Transp of Dissolved Inorganic Carbon
VTRAC01 | 15 |VV   170MR      |mol C/m.m/s     |Merid Mass-Weighted Transp of Dissolved Inorganic Carbon
WTRAC01 | 15 |WM      MR      |mol C/m.m/s     |Vert  Mass-Weighted Transp of Dissolved Inorganic Carbon
ADVrTr01| 15 |WM      LR      |mol C/m.m^3/s   |Vertical   Advective Flux of Dissolved Inorganic Carbon
ADVxTr01| 15 |UU   175MR      |mol C/m.m^3/s   |Zonal      Advective Flux of Dissolved Inorganic Carbon
ADVyTr01| 15 |VV   174MR      |mol C/m.m^3/s   |Meridional Advective Flux of Dissolved Inorganic Carbon
DFrETr01| 15 |WM      LR      |mol C/m.m^3/s   |Vertical Diffusive Flux of Dissolved Inorganic Carbon (Explicit part)
DIFxTr01| 15 |UU   178MR      |mol C/m.m^3/s   |Zonal      Diffusive Flux of Dissolved Inorganic Carbon
DIFyTr01| 15 |VV   177MR      |mol C/m.m^3/s   |Meridional Diffusive Flux of Dissolved Inorganic Carbon
DFrITr01| 15 |WM      LR      |mol C/m.m^3/s   |Vertical Diffusive Flux of Dissolved Inorganic Carbon (Implicit part)
TRAC02  | 15 |SM P    MR      |mol eq/         |Mass-Weighted Alkalinity
UTRAC02 | 15 |UU   182MR      |mol eq/.m/s     |Zonal Mass-Weighted Transp of Alkalinity
VTRAC02 | 15 |VV   181MR      |mol eq/.m/s     |Merid Mass-Weighted Transp of Alkalinity
WTRAC02 | 15 |WM      MR      |mol eq/.m/s     |Vert  Mass-Weighted Transp of Alkalinity
ADVrTr02| 15 |WM      LR      |mol eq/.m^3/s   |Vertical   Advective Flux of Alkalinity
ADVxTr02| 15 |UU   186MR      |mol eq/.m^3/s   |Zonal      Advective Flux of Alkalinity
ADVyTr02| 15 |VV   185MR      |mol eq/.m^3/s   |Meridional Advective Flux of Alkalinity
DFrETr02| 15 |WM      LR      |mol eq/.m^3/s   |Vertical Diffusive Flux of Alkalinity (Explicit part)
DIFxTr02| 15 |UU   189MR      |mol eq/.m^3/s   |Zonal      Diffusive Flux of Alkalinity
DIFyTr02| 15 |VV   188MR      |mol eq/.m^3/s   |Meridional Diffusive Flux of Alkalinity
DFrITr02| 15 |WM      LR      |mol eq/.m^3/s   |Vertical Diffusive Flux of Alkalinity (Implicit part)
TRAC03  | 15 |SM P    MR      |mol P/m         |Mass-Weighted Phosphate
UTRAC03 | 15 |UU   193MR      |mol P/m.m/s     |Zonal Mass-Weighted Transp of Phosphate
VTRAC03 | 15 |VV   192MR      |mol P/m.m/s     |Merid Mass-Weighted Transp of Phosphate
WTRAC03 | 15 |WM      MR      |mol P/m.m/s     |Vert  Mass-Weighted Transp of Phosphate
ADVrTr03| 15 |WM      LR      |mol P/m.m^3/s   |Vertical   Advective Flux of Phosphate
ADVxTr03| 15 |UU   197MR      |mol P/m.m^3/s   |Zonal      Advective Flux of Phosphate
ADVyTr03| 15 |VV   196MR      |mol P/m.m^3/s   |Meridional Advective Flux of Phosphate
DFrETr03| 15 |WM      LR      |mol P/m.m^3/s   |Vertical Diffusive Flux of Phosphate (Explicit part)
DIFxTr03| 15 |UU   200MR      |mol P/m.m^3/s   |Zonal      Diffusive Flux of Phosphate
------------------------------------------------------------------------
<-Name->|Levs|<-parsing code->|<--  Units   -->|<- Tile (max=80c)
------------------------------------------------------------------------
DIFyTr03| 15 |VV   199MR      |mol P/m.m^3/s   |Meridional Diffusive Flux of Phosphate
DFrITr03| 15 |WM      LR      |mol P/m.m^3/s   |Vertical Diffusive Flux of Phosphate (Implicit part)
TRAC04  | 15 |SM P    MR      |mol P/m         |Mass-Weighted Dissolved Organic Phosphorus
UTRAC04 | 15 |UU   204MR      |mol P/m.m/s     |Zonal Mass-Weighted Transp of Dissolved Organic Phosphorus
VTRAC04 | 15 |VV   203MR      |mol P/m.m/s     |Merid Mass-Weighted Transp of Dissolved Organic Phosphorus
WTRAC04 | 15 |WM      MR      |mol P/m.m/s     |Vert  Mass-Weighted Transp of Dissolved Organic Phosphorus
ADVrTr04| 15 |WM      LR      |mol P/m.m^3/s   |Vertical   Advective Flux of Dissolved Organic Phosphorus
ADVxTr04| 15 |UU   208MR      |mol P/m.m^3/s   |Zonal      Advective Flux of Dissolved Organic Phosphorus
ADVyTr04| 15 |VV   207MR      |mol P/m.m^3/s   |Meridional Advective Flux of Dissolved Organic Phosphorus
DFrETr04| 15 |WM      LR      |mol P/m.m^3/s   |Vertical Diffusive Flux of Dissolved Organic Phosphorus (Explicit part)
DIFxTr04| 15 |UU   211MR      |mol P/m.m^3/s   |Zonal      Diffusive Flux of Dissolved Organic Phosphorus
DIFyTr04| 15 |VV   210MR      |mol P/m.m^3/s   |Meridional Diffusive Flux of Dissolved Organic Phosphorus
DFrITr04| 15 |WM      LR      |mol P/m.m^3/s   |Vertical Diffusive Flux of Dissolved Organic Phosphorus (Implicit part)
TRAC05  | 15 |SM P    MR      |mol O/m         |Mass-Weighted Dissolved Oxygen
UTRAC05 | 15 |UU   215MR      |mol O/m.m/s     |Zonal Mass-Weighted Transp of Dissolved Oxygen
VTRAC05 | 15 |VV   214MR      |mol O/m.m/s     |Merid Mass-Weighted Transp of Dissolved Oxygen
WTRAC05 | 15 |WM      MR      |mol O/m.m/s     |Vert  Mass-Weighted Transp of Dissolved Oxygen
ADVrTr05| 15 |WM      LR      |mol O/m.m^3/s   |Vertical   Advective Flux of Dissolved Oxygen
ADVxTr05| 15 |UU   219MR      |mol O/m.m^3/s   |Zonal      Advective Flux of Dissolved Oxygen
ADVyTr05| 15 |VV   218MR      |mol O/m.m^3/s   |Meridional Advective Flux of Dissolved Oxygen
```

```
DFrETr05| 15 |WM      LR      |mol O/m.m^3/s   |Vertical Diffusive Flux of Dissolved Oxygen (Explicit part)
DIFxTr05| 15 |UU   222MR      |mol O/m.m^3/s   |Zonal      Diffusive Flux of Dissolved Oxygen
DIFyTr05| 15 |VV   221MR      |mol O/m.m^3/s   |Meridional Diffusive Flux of Dissolved Oxygen
DFrITr05| 15 |WM      LR      |mol O/m.m^3/s   |Vertical Diffusive Flux of Dissolved Oxygen (Implicit part)
```

6.3.3.5 Do's and Don'ts

6.3.3.6 Reference Material

6.4 Ocean Packages

6.4.1 GMREDI: Gent-McWilliams/Redi SGS Eddy Parameterization

There are two parts to the Redi/GM parameterization of geostrophic eddies. The first, the Redi scheme [*Redi*, 1982], 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 [*Gent and McWilliams*, 1990; *Gent et al.*, 1995], 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 *Cox* [1987] in the GFDL ocean circulation model. The original approach failed to distinguish between isopycnals and surfaces of locally referenced potential density (now called neutral surfaces) which are proper isentropes for the ocean. As will be discussed later, it also appears that the Cox implementation is susceptible to a computational mode. Due to this mode, the Cox scheme requires a background lateral diffusion to be present to conserve the integrity of the model fields.

The GM parameterization was then added to the GFDL code in the form of a non-divergent bolus velocity. The method defines two stream-functions expressed in terms of the isoneutral slopes subject to the boundary condition of zero value on upper and lower boundaries. The horizontal bolus velocities are then the vertical derivative of these functions. Here in lies a problem highlighted by *Griffies et al.* [1998]: 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.

6.4.1.1 Redi scheme: Isopycnal diffusion

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

$$\nabla \cdot \kappa_\rho \mathbf{K_{Redi}} \nabla \tau \tag{6.1}$$

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

$$\mathbf{K_{Redi}} = \frac{\mathbf{1}}{\mathbf{1} + |\mathbf{S}|^\mathbf{2}} \left(\begin{array}{ccc} 1 + S_y^2 & -S_x S_y & S_x \\ -S_x S_y & 1 + S_x^2 & S_y \\ S_x & S_y & |S|^2 \end{array} \right) \tag{6.2}$$

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

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

$$\mathbf{K_{Redi}} = \left(\begin{array}{ccc} 1 & 0 & S_x \\ 0 & 1 & S_y \\ S_x & S_y & |S|^2 \end{array} \right) \tag{6.3}$$

6.4.1.2 GM parameterization

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

$$-\nabla \cdot \tau \mathbf{u}^\star \tag{6.4}$$

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

$$\mathbf{u}^\star = \nabla \times \mathbf{F}^\star = \left(\begin{array}{c} -\partial_z F_y^\star \\ +\partial_z F_x^\star \\ \partial_x F_y^\star - \partial_y F_x^\star \end{array} \right), \tag{6.5}$$

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:

$$F_x^\star = -\kappa_{GM} S_y \tag{6.6}$$

$$F_y^\star = \kappa_{GM} S_x \tag{6.7}$$

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

$$\mathbf{u}^\star = \begin{pmatrix} u^\star \\ v^\star \\ w^\star \end{pmatrix} = \begin{pmatrix} -\partial_z(\kappa_{GM} S_x) \\ -\partial_z(\kappa_{GM} S_y) \\ \partial_x(\kappa_{GM} S_x) + \partial_y(\kappa_{GM} S_y) \end{pmatrix} \tag{6.8}$$

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

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

$$\begin{pmatrix} GM\_PsiX \\ GM\_PsiY \end{pmatrix} = \begin{pmatrix} \kappa_{GM} S_x \\ \kappa_{GM} S_y \end{pmatrix} = \begin{pmatrix} F_y^\star \\ -F_x^\star \end{pmatrix}. \tag{6.9}$$

6.4.1.3 Griffies Skew Flux

Griffies [1998] notes that the discretisation of bolus velocities involves multiple layers of differencing and interpolation that potentially lead to noisy fields and computational modes. He pointed out that the bolus flux can be re-written in terms of a non-divergent flux and a skew-flux:

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

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

$$\mathbf{u}^\star \tau = -\kappa_{\mathbf{GM}} \mathbf{K_{GM}} \nabla \tau \tag{6.10}$$

where

$$\mathbf{K_{GM}} = \begin{pmatrix} 0 & 0 & -S_x \\ 0 & 0 & -S_y \\ S_x & S_y & 0 \end{pmatrix} \tag{6.11}$$

is an anti-symmetric tensor.

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

$$\kappa_\rho \mathbf{K_{Redi}} \nabla \tau - \mathbf{u}^\star \tau = (\kappa_\rho \mathbf{K_{Redi}} + \kappa_{\mathbf{GM}} \mathbf{K_{GM}}) \nabla \tau \tag{6.12}$$

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

$$\kappa_\rho \mathbf{K_{Redi}} + \kappa_{\mathbf{GM}} \mathbf{K_{GM}} = \kappa_\rho \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 2S_x & 2S_y & |S|^2 \end{pmatrix} \tag{6.13}$$

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

S/R GMREDI_CALC_TENSOR (*pkg/gmredi/gmredi_calc_tensor.F*)
σ_x: **SlopeX** (argument on entry)
σ_y: **SlopeY** (argument on entry)
σ_z: **SlopeY** (argument)
S_x: **SlopeX** (argument on exit)
S_y: **SlopeY** (argument on exit)

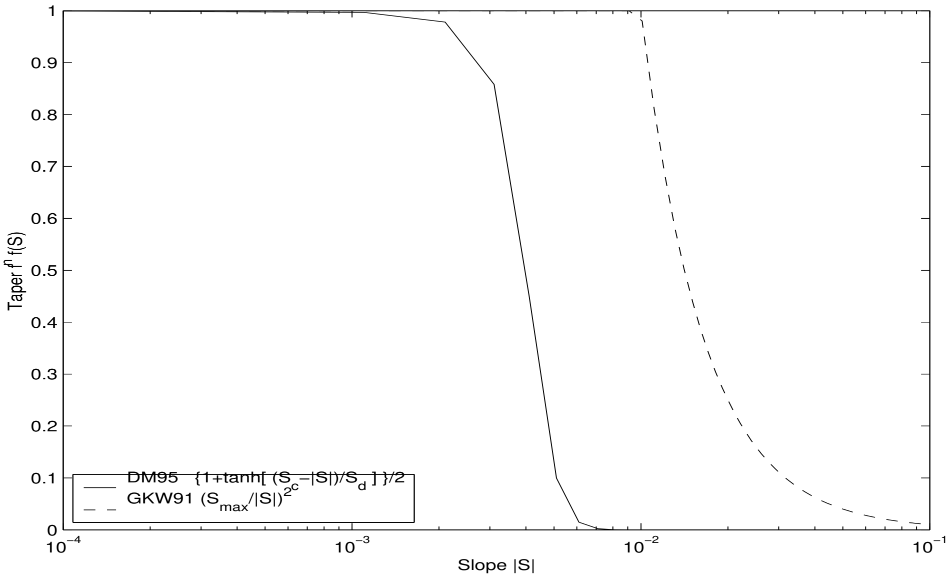


Figure 6.6: Taper functions used in GKW91 and DM95.

6.4.1.4 Variable κ_{GM}

Visbeck et al. [1997] suggest making the eddy coefficient, κ_{GM}, a function of the Eady growth rate, $|f|/\sqrt{Ri}$. The formula involves a non-dimensional constant, α, and a length-scale L:

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

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

$$\frac{1}{Ri} = \frac{(\frac{\partial u}{\partial z})^2}{N^2} = \frac{(\frac{g}{f\rho_o}|\nabla\sigma|)^2}{N^2} = \frac{M^4}{|f|^2 N^2}$$

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

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

6.4.1.5 Tapering and stability

Experience with the GFDL model showed that the GM scheme has to be matched to the convective parameterization. This was originally expressed in connection with the introduction of the KPP boundary layer scheme [*Large et al.*, 1994] but in fact, as subsequent experience with the MIT model has found, is necessary for any convective parameterization.

S/R GMREDI_SLOPE_LIMIT (*pkg/gmredi/gmredi_slope_limit.F*)
σ_x, s_x: **SlopeX** (argument)
σ_y, s_y: **SlopeY** (argument)
σ_z: **dSigmadRReal** (argument)
z^*_σ: **dRdSigmaLtd** (argument)

Slope clipping

Deep convection sites and the mixed layer are indicated by homogenized, unstable or nearly unstable stratification. The slopes in such regions can be either infinite, very large with a sign reversal or simply

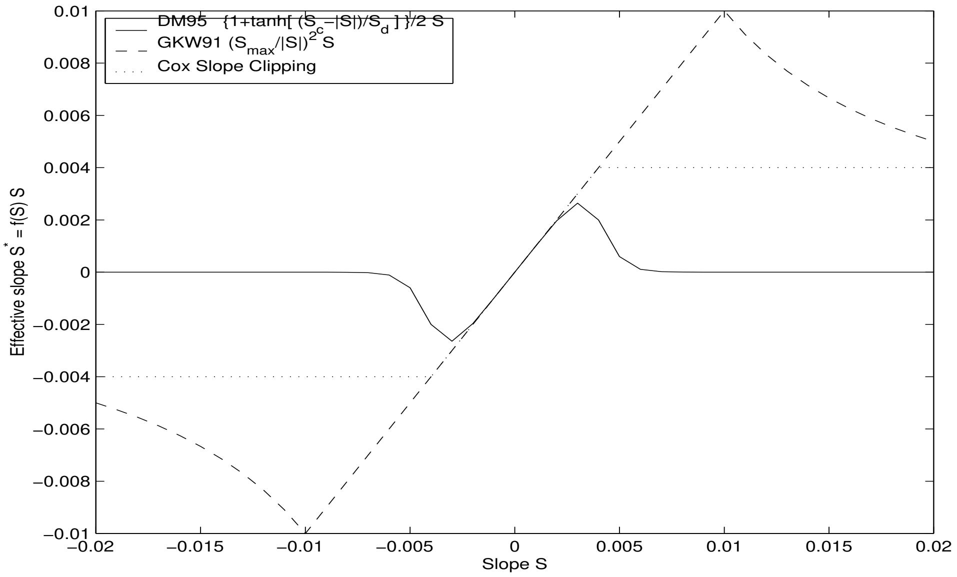


Figure 6.7: Effective slope as a function of “true” slope using Cox slope clipping, GKW91 limiting and DM95 limiting.

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 *Cox* [1987] who implemented “slope clipping” in the isopycnal mixing tensor. Here, the slope magnitude is simply restricted by an upper limit:

$$|\nabla\sigma| = \sqrt{\sigma_x^2 + \sigma_y^2} \tag{6.14}$$

$$S_{lim} = -\frac{|\nabla\sigma|}{S_{max}} \quad \text{where } S_{max} \text{ is a parameter} \tag{6.15}$$

$$\sigma_z^\star = \min(\sigma_z, S_{lim}) \tag{6.16}$$

$$[s_x, s_y] = -\frac{[\sigma_x, \sigma_y]}{\sigma_z^\star} \tag{6.17}$$

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:

$$[s_x, s_y] = -\frac{[\sigma_x, \sigma_y]}{\sigma_z} \tag{6.18}$$

while in the limited regions ($\sigma_z > S_{lim}$) the slopes become:

$$[s_x, s_y] = \frac{[\sigma_x, \sigma_y]}{|\nabla\sigma|/S_{max}} \tag{6.19}$$

so that the slope magnitude is limited $\sqrt{s_x^2 + s_y^2} = S_{max}$.

The slope clipping scheme is activated in the model by setting **GM_taper_scheme = 'clipping'** in *data.gmredi*.

Even using slope clipping, it is normally the case that the vertical diffusion term (with coefficient $\kappa_\rho \mathbf{K}_{33} = \kappa_\rho S_{max}^2$) is large and must be time-stepped using an implicit procedure (see section on discretisation and code later). Fig. 6.8 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.

Figure missing.

Figure 6.8: Mixed layer depth using GM parameterization with a) Cox slope clipping and for comparison b) using horizontal constant diffusion.

Tapering: Gerdes, Koberle and Willebrand, Clim. Dyn. 1991

The tapering scheme used in *Gerdes et al.* [1991] 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:

$$f_1(S) = \min\left[1, \left(\frac{S_{max}}{|S|}\right)^2\right] \tag{6.20}$$

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 *Danabasoglu and McWilliams* [1995] followed a similar procedure but used a different tapering function, $f_1(S)$:

$$f_1(S) = \frac{1}{2}\left(1 + \tanh\left[\frac{S_c - |S|}{S_d}\right]\right) \tag{6.21}$$

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

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

Tapering: Large, Danabasoglu and Doney, JPO 1997

The tapering used in *Large et al.* [1997] 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:

$$f_2(z) = \frac{1}{2}\left(1 + \sin(\pi\frac{z}{D} - \frac{\pi}{2})\right) \tag{6.22}$$

where $D = L_\rho|S|$ is a depth-scale and $L_\rho = c/f$ with $c = 2$ m s^{-1}. This tapering with height was introduced to fix some spurious interaction with the mixed-layer KPP parameterization.

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

6.4.1.6 Package Reference

```
------------------------------------------------------------------------
<-Name->|Levs|<-parsing code->|<--  Units   -->|<- Tile (max=80c)
------------------------------------------------------------------------
GM_VisbK|  1 |SM P     M1     |m^2/s           |Mixing coefficient from Visbeck etal parameterization
GM_Kux  | 15 |UU P 177MR      |m^2/s           |K_11 element (U.point, X.dir) of GM-Redi tensor
GM_Kvy  | 15 |VV P 176MR      |m^2/s           |K_22 element (V.point, Y.dir) of GM-Redi tensor
GM_Kuz  | 15 |UU   179MR      |m^2/s           |K_13 element (U.point, Z.dir) of GM-Redi tensor
```

```
GM_Kvz  | 15 |VV   178MR      |m^2/s           |K_23 element (V.point, Z.dir) of GM-Redi tensor
GM_Kwx  | 15 |UM   181LR      |m^2/s           |K_31 element (W.point, X.dir) of GM-Redi tensor
GM_Kwy  | 15 |VM   180LR      |m^2/s           |K_32 element (W.point, Y.dir) of GM-Redi tensor
GM_Kwz  | 15 |WM P    LR      |m^2/s           |K_33 element (W.point, Z.dir) of GM-Redi tensor
GM_PsiX | 15 |UU   184LR      |m^2/s           |GM Bolus transport stream-function : X component
GM_PsiY | 15 |VV   183LR      |m^2/s           |GM Bolus transport stream-function : Y component
GM_KuzTz| 15 |UU   186MR      |degC.m^3/s      |Redi Off-diagonal Tempetature flux: X component
GM_KvzTz| 15 |VV   185MR      |degC.m^3/s      |Redi Off-diagonal Tempetature flux: Y component
```

6.4.1.7 Experiments and tutorials that use gmredi

- Global Ocean tutorial, in tutorial_global_oce_latlon verification directory, described in section 3.12
- Front Relax experiment, in front_relax verification directory.
- Ideal 2D Ocean experiment, in ideal_2D_oce verification directory.

6.4.2 KPP: Nonlocal K-Profile Parameterization for Vertical Mixing

Authors: Dimitris Menemenlis and Patrick Heimbach

6.4.2.1 Introduction

The nonlocal K-Profile Parameterization (KPP) scheme of *Large et al.* [1994] unifies the treatment of a variety of unresolved processes involved in vertical mixing. To consider it as one mixing scheme is, in the view of the authors, somewhat misleading since it consists of several entities to deal with distinct mixing processes in the ocean's surface boundary layer, and the interior:

1. mixing in the interior is goverened by shear instability (modeled as function of the local gradient Richardson number), internal wave activity (assumed constant), and double-diffusion (not implemented here).

2. a boundary layer depth h or `hbl` is determined at each grid point, based on a critical value of turbulent processes parameterized by a bulk Richardson number;

3. mixing is strongly enhanced in the boundary layer under the stabilizing or destabilizing influence of surface forcing (buoyancy and momentum) enabling boundary layer properties to penetrate well into the thermocline; mixing is represented through a polynomial profile whose coefficients are determined subject to several contraints;

4. the boundary-layer profile is made to agree with similarity theory of turbulence and is matched, in the asymptotic sense (function and derivative agree at the boundary), to the interior thus fixing the polynomial coefficients; matching allows for some fraction of the boundary layer mixing to affect the interior, and vice versa;

5. a "non-local" term $\hat{\gamma}$ or `ghat` which is independent of the vertical property gradient further enhances mixing where the water column is unstable

The scheme has been extensively compared to observations (see e.g. *Large et al.* [1997]) and is now coomon 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 (6.4.2.2), the settings and choices of runtime parameters (6.4.2.3), more detailed description of equations to which these parameters relate (6.4.2.4), and key subroutines where they are used (6.4.2.5), and diagnostics output of KPP-derived diffusivities, viscosities and boundary-layer/mixed-layer depths (6.4.2.6).

6.4.2.2 KPP configuration and compiling

As with all MITgcm packages, KPP can be turned on or off at compile time

- using the `packages.conf` file by adding `kpp` to it,
- or using `genmake2` adding `-enable=kpp` or `-disable=kpp` switches
- *Required packages and CPP options:*
  No additional packages are required, but the MITgcm kernel flag enabling the penetration of shortwave radiation below the surface layer needs to be set in `CPP_OPTIONS.h` as follows:
  `#define SHORTWAVE_HEATING`

(see Section 3.4).

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 6.4.2.2 summarizes them.

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

Table 6.6:

6.4.2.3 Run-time parameters

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

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

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

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

Package flags and parameters
Table 6.4.2.3 summarizes the runtime flags that are set in `data.pkg`, and their default values.

6.4.2.4 Equations and key routines

We restrict ourselves to writing out only the essential equations that relate to main processes and parameters mentioned above. We closely follow the notation of *Large et al.* [1994].

KPP_CALC: Top-level routine.

KPP_MIX: Intermediate-level routine

BLMIX: Mixing in the boundary layer

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

$$\overline{wx}(d) = -K_x \left(\frac{\partial X}{\partial z} - \gamma_x \right) \tag{6.23}$$

- *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 $G(\sigma)$

  $$K_x(\sigma) = h\, w_x(\sigma)\, G(\sigma) \tag{6.24}$$

  with dimensionless vertical coordinate $\sigma = d/h$. For details of $w_x(\sigma)$ and $G(\sigma)$ we refer to *Large et al.* [1994].

Flag/parameter	**default**	**Description**
	I/O related parameters	
kpp_freq	`deltaTClock`	Recomputation frequency for KPP fields
kpp_dumpFreq	`dumpFreq`	Dump frequency of KPP field snapshots
kpp_taveFreq	`taveFreq`	Averaging and dump frequency of KPP fields
KPPmixingMaps	`.FALSE.`	include KPP diagnostic maps in STDOUT
KPPwriteState	`.FALSE.`	write KPP state to file
KPP_ghatUseTotalDiffus	`.FALSE.`	if `.T.` compute non-local term using total vertical diffusivity if `.F.` use KPP vertical diffusivity
	Genral 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 $1/s^2$	maximum dB/dz in mixed layer hMix
concs	98.96	
concv	1.8	
	Boundary layer parameters (S/R `bldepth`)	
Ricr	0.3	critical bulk Richardson number
cekman	0.7	coefficient for Ekman depth
cmonob	1.0	coefficient for Monin-Obukhov depth
concv	1.8	ratio of interior to entrainment depth buoyancy frequency
hbf	1.0	fraction of depth to which absorbed solar radiation contributes to surface buoyancy forcing
Vtc		non-dim. coeff. for velocity scale of turbulant velocity shear (= function of concv,concs,epsilon,vonk,Ricr)
	Boundary layer mixing parameters (S/R `blmix`)	
cstar	10.	proportionality coefficient for nonlocal transport
cg		non-dimensional coefficient for counter-gradient term (= function of cstar,vonk,concs,epsilon)
	Interior mixing parameters (S/R `Ri_iwmix`)	
Riinfty	0.7	gradient Richardson number limit for shear instability
BVDQcon	-0.2E-4 $1/s^2$	Brunt-Väisälä squared
difm0	0.005 m^2/s	viscosity max. due to shear instability
difs0	0.005 m^2/s	tracer diffusivity max. due to shear instability
dift0	0.005 m^2/s	heat diffusivity max. due to shear instability
difmcon	0.1	viscosity due to convective instability
difscon	0.1	tracer diffusivity due to convective instability
diftcon	0.1	heat diffusivity due to convective instability
	Double-diffusive mixing parameters (S/R `ddmix`)	
Rrho0	not used	limit for double diffusive density ratio
dsfmax	not used	maximum diffusivity in case of salt fingering

Table 6.7:

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

$$
\begin{aligned}
\gamma_x &= 0 \qquad\qquad \zeta \geq 0 \\
\left.\begin{aligned}
\gamma_m &= 0 \\
\gamma_s &= C_s \frac{\overline{ws_0}}{w_s(\sigma)h} \\
\gamma_\theta &= C_s \frac{\overline{w\theta_0}+\overline{w\theta_R}}{w_s(\sigma)h}
\end{aligned}\right\} &\quad \zeta < 0
\end{aligned}
\tag{6.25}
$$

In practice, the routine peforms the following tasks:

1. compute velocity scales at hbl
2. find the interior viscosities and derivatives at hbl
3. compute turbulent velocity scales on the interfaces

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

RI_IWMIX: Mixing in the interior
Compute interior viscosity and diffusivity coefficients due to

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

TO BE CONTINUED.

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

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

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

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

TO BE CONTINUED.

KPP_CALC_DIFF_T/_S, KPP_CALC_VISC:
Add contribution to net diffusivity/viscosity from KPP diffusivity/viscosity.
TO BE CONTINUED.

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

Implicit time integration
TO BE CONTINUED.

Penetration of shortwave radiation
TO BE CONTINUED.

6.4.2.5 Flow chart

```
C     !CALLING SEQUENCE:
c ...
c  kpp_calc (TOP LEVEL ROUTINE)
c  |
c  |-- statekpp: o compute all EOS/density-related arrays
c  |             o uses S/R FIND_ALPHA, FIND_BETA, FIND_RHO
c  |
c  |-- kppmix
c  |   |--- ri_iwmix (compute interior mixing coefficients due to constant
```

```
c  |   |              internal wave activity, static instability,
c  |   |              and local shear instability).
c  |   |
c  |   |--- bldepth (diagnose boundary layer depth)
c  |   |
c  |   |--- blmix (compute boundary layer diffusivities)
c  |   |
c  |   |--- enhance (enhance diffusivity at interface kbl - 1)
c  |   o
c  |
c  |-- swfrac
c  o
```

6.4.2.6 KPP diagnostics

Diagnostics output is available via the diagnostics package (see Section 7.1). Available output fields are summarized here:

```
------------------------------------------------------
 <-Name->|Levs|grid|<--  Units   -->|<- Tile (max=80c)
------------------------------------------------------
 KPPviscA| 23 |SM  |m^2/s           |KPP vertical eddy viscosity coefficient
 KPPdiffS| 23 |SM  |m^2/s           |Vertical diffusion coefficient for salt & tracers
 KPPdiffT| 23 |SM  |m^2/s           |Vertical diffusion coefficient for heat
 KPPghat | 23 |SM  |s/m^2           |Nonlocal transport coefficient
 KPPhbl  |  1 |SM  |m               |KPP boundary layer depth, bulk Ri criterion
 KPPmld  |  1 |SM  |m               |Mixed layer depth, dT=.8degC density criterion
 KPPfrac |  1 |SM  |                |Short-wave flux fraction penetrating mixing layer
```

6.4.2.7 Reference experiments

lab_sea:
natl_box:

6.4.2.8 References

6.4.2.9 Experiments and tutorials that use kpp

- Labrador Sea experiment, in lab_sea verification directory

6.4.3 GGL90: a TKE vertical mixing scheme

(in directory: `pkg/ggl90/`)

6.4.3.1 Key subroutines, parameters and files

see *Gaspar et al.* [1990]

6.4.3.2 Experiments and tutorials that use zonal filter

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

6.4.4 OPPS: Ocean Penetrative Plume Scheme

(in directory: `pkg/opps/`)

6.4.4.1 Key subroutines, parameters and files

see *Paluszkiewicz and Romea* [1997]

6.4.4.2 Experiments and tutorials that use zonal filter

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

6.4.5 KL10: Vertical Mixing Due to Breaking Internal Waves

(in directory: `pkg/kl10/`)

Authors: Jody M. Klymak

6.4.5.1 Introduction

The *Klymak and Legg* [2010, KL10] parameterization for breaking internal waves is meant to represent mixing in the ocean "interior" due to convective instability. Many mixing schemes in the presence of unstable stratification simply turn on an arbitrarily large diffusivity and viscosity in the overturning region. This assumes the fluid completely mixes, which is probably not a terrible assumption, but it also makes estimating the turbulence dissipation rate in the overturning region meaningless.

The KL10 scheme overcomes this limitation by estimating the viscosity and diffusivity from a combination of the Ozmidov relation and the Osborn relation, assuming a turbulent Prandtl number of one. The Ozmidov relation says that outer scale of turbulence in an overturn will scale with the strength of the turbulence ϵ, and the stratification N, as

$$L_O^2 \approx \epsilon N^{-3}. \tag{6.26}$$

The Osborn relation relates the strength of the dissipation to the vertical diffusivity as

$$K_v = \Gamma \epsilon N^{-2}, \tag{6.27}$$

where $\Gamma \approx 0.2$ is the mixing ratio of buoyancy flux to thermal dissipation due to the turbulence. Combining the two gives us

$$K_v \approx \Gamma L_O^2 N. \tag{6.28}$$

The ocean turbulence community often approximates the Ozmidov scale by the root-mean-square of the Thorpe displacement, δ_z, in an overturn [*Thorpe*, 1977]. 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 [*Wesson and Gregg*, 1994; *Seim and Gregg*, 1994; *Moum*, 1996].

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

$$K_v(z) \approx \Gamma \delta_z^2 \, N_s(z). \tag{6.29}$$

where $N_s(z)$ is the local sorted stratification. This saves complexity in the code and adds a slight inaccuracy, but we don't believe is biased.

We assume a turbulent Prandtl number of 1, so $A_v = K_v$.

We also calculate and output a turbulent dissipation from this scheme. We do not simply evaluate the overturns for ϵ using (6.26). Instead we compute the vertical shear terms that the viscosity is acting on:

$$\epsilon_v = A_v \left(\left(\frac{\partial u}{\partial z} \right)^2 + \left(\frac{\partial v}{\partial z} \right)^2 \right). \tag{6.30}$$

There are straightforward caveats to this approach, covered in *Klymak and Legg* [2010].

- If your resolution is too low to resolve the breaking internal waves, you won't have any turbulence.
- If the model resolution is too high, the estimates of ϵ_v will start to be exaggerated, particularly if the run in non-hydrostatic. That is because there will be significant shear at small scales that represents the turbulence being parameterized in the scheme. At very high resolutions direct numerical simulation or more sophisticated large-eddy schemes should be used.
- We find that grid cells of approximately 10 to 1 aspect ratio are a good rule of thumb for achieving good results are usual oceanic scales. For a site like the Hawaiian Ridge, and Luzon Strait, this means 10-m vertical resolusion and approximately 100-m horizontal. The 10-m resolution can be relaxed if the stratification drops, and we often WKB-stretch the grid spacing with depth.

- The dissipation estimate is useful for pinpoiting the location of turbulence, but again, is grid size dependent to some extent, and should be treated with a grain of salt. It will also not include any numerical dissipation such as you may find with higher order advection schemes.

6.4.5.2 KL10 configuration and compiling

As with all MITgcm packages, KL10 can be turned on or off at compile time

- using the `packages.conf` file by adding `kl10` to it,
- or using `genmake2` adding `-enable=kl10` or `-disable=kl10` switches
- *Required packages and CPP options:*
  No additional packages are required.

(see Section 3.4).

KL10 has no compile-time options (`KL10_OPTIONS.h` is empty).

6.4.5.3 Run-time parameters

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

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

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

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

Package flags and parameters
Table 6.4.5.3 summarizes the runtime flags that are set in `data.kl10`, and their default values.

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

Table 6.8: KL10 runtime parameters

6.4.5.4 Equations and key routines

KL10_CALC: Top-level routine. Calculates viscosity and diffusivity on the grid cell centers. Note that the runtime parameters `viscAz` and `diffKzT` act as minimum viscosity and diffusivities. So if there are no overturns (or they are weak) then these will be returned.

KL10_CALC_VISC: Calculates viscosity on the W and S grid faces for U and V respectively.

KL10_CALC_DIFF: Calculates the added diffusion from KL10.

6.4.5.5 KL10 diagnostics

Diagnostics output is available via the diagnostics package (see Section 7.1). Available output fields are summarized here:

```
-------------------------------------------------------
 <-Name->|Levs|grid|<--  Units   -->|<- Tile (max=80c)
-------------------------------------------------------
 KLviscAr| Nr |SM  |m^2/s           |KL10 vertical eddy viscosity coefficient
 KLdiffKr| Nr |SM  |m^2/s           |Vertical diffusion coefficient for salt, temperature, & tracers
 KLeps  | Nr |SM  |m^3/s^3           | Turbulence dissipation estimate.
```

subsubsectionReference experiments
internal_wavekl10:

6.4.5.6 References

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

6.4.5.7 Experiments and tutorials that use KL10

- Modified Internal Wave experiment, in internal_wave verification directory

6.4.6 BULK_FORCE: Bulk Formula Package

author: Stephanie Dutkiewicz

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

The current setup requires that there is the thermodynamic-seaice package (*pkg/thsice*, also refered below as seaice) is also used. It would be useful though to have it also setup to run with some very simple parametrization of the sea ice.

The heat and fresh water fluxes are calculated in *bulkf_forcing.F* called from *forward_step.F*. These fluxes are used over open water, fluxes over seaice are recalculated in the sea-ice package. Before the call to *bulkf_forcing.F* we call *bulkf_fields_load.F* to find the current atmospheric conditions. The only other changes to the model code come from the initializing and writing diagnostics of these fluxes.

subroutine BULKF_FIELDS_LOAD

Here we find the atmospheric data needed for the bulk formula calculations. These are read in at periodic intervals and values are interpolated to the current time. The data file names come from **data.blk**. The values that can be read in are: air temperature, air humidity, precipitation, down solar radiation, down long wave radiation, zonal and meridional wind speeds, total wind speed, net heat flux, net freshwater forcing, cloud cover, snow fall, zonal and meridional wind stresses, and SST and SSS used for relaxation terms. Not all these files are necessary or used. For instance cloud cover and snow fall are not used in the current bulk formula calculation. If total wind speed is not supplied, wind speed is calculate from the zonal and meridional components. If wind stresses are not read in, then the stresses are calculated from the wind speed. Net heat flux and net freshwater can be read in and used over open ocean instead of the bulk formula calculations (but over seaice the bulkf formula is always used). This is "hardwired" into *bulkf_forcing* and the "ch" in the variable names suggests that this is "cheating". SST and SSS need to be read in if there is any relaxation used.

subroutine BULKF_FORCING

In *bulkf_forcing.F*, we calculate heat and fresh water fluxes (and wind stress, if necessary) for each grid cell. First we determine if the grid cell is open water or seaice and this information is carried by **iceornot**. There is a provision here for a different designation if there is snow cover (but currently this does not make any difference). We then call *bulkf_formula_lanl.F* which provides values for: up long wave radiation, latent and sensible heat fluxes, the derivative of these three with respect to surface temperature, wind stress, evaporation. Net long wave radiation is calculated from the combination of the down long wave read in and the up long wave calculated.

We then find the albedo of the surface - with a call to *sfc_albedo* if there is sea-ice (see the seaice package for information on the subroutine). If the grid cell is open ocean the albedo is set as 0.1. Note that this is a parameter that can be used to tune the results. The net short wave radiation is then the down shortwave radiation minus the amount reflected.

If the wind stress needed to be calculated in *bulkf_formula_lanl.F*, it was calculated to grid cell center points, so in *bulkf_forcing.F* we regrid to **u** and **v** points. We let the model know if it has read in stresses or calculated stresses by the switch **readwindstress** which is can be set in data.blk, and defaults to **.TRUE.**.

We then calculate **Qnet** and **EmPmR** that will be used as the fluxes over the open ocean. There is a provision for using runoff. If we are "cheating" and using observed fluxes over the open ocean, then there is a provision here to use read in **Qnet** and **EmPmR**.

The final call is to calculate averages of the terms found in this subroutine.

subroutine BULKF_FORMULA_LANL

This is the main program of the package where the heat fluxes and freshwater fluxes over ice and open water are calculated. Note that this subroutine is also called from the seaice package during the iterations to find the ice surface temperature.

Latent heat (L) used in this subroutine depends on the state of the surface: vaporization for open water, fusion and vaporization for ice surfaces. Air temperature is converted from Celsius to Kelvin. If there is no wind speed (u_s) given, then the wind speed is calculated from the zonal and meridional components.

We calculate the virtual temperature:

$$T_o = T_{air}(1 + \gamma q_{air})$$

where T_{air} is the air temperature at h_T, q_{air} is humidity at h_q and γ is a constant.

The saturated vapor pressure is calculate (QQ ref):

$$q_{sat} = \frac{a}{p_o} e^{L(b - \frac{c}{T_{srf}})}$$

where a, b, c are constants, T_{srf} is surface temperature and p_o is the surface pressure.

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

$$\Delta T = T_{air} - T_{srf} + \alpha h_T$$

where α is adiabatic lapse rate and h_T is the height where the air temperature was taken; and the difference between the air humidity and the saturated humidity

$$\Delta q = q_{air} - q_{sat}.$$

We then calculate the turbulent exchange coefficients following Bryan et al (1996) and the numerical scheme of Hunke and Lipscombe (1998). We estimate initial values for the exchange coefficients, c_u, c_T and c_q as

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

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

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

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

$$\psi_m = \psi_s = -5\Upsilon$$

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

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

where

$$\Upsilon = \frac{\kappa g z_{ref}}{u^{*2}}\left(\frac{T^*}{T_o} + \frac{q^*}{1/\gamma + q_a}\right)$$

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

The coefficients are updated through 5 iterations as:

$$\begin{aligned} c_u &= \frac{\hat{c_u}}{1 + \hat{c_u}(\lambda - \psi_m)/\kappa} \\ c_T &= \frac{\hat{c_T}}{1 + \hat{c_T}(\lambda - \psi_s)/\kappa} \\ c_q &= c_T' \end{aligned} \tag{6.31}$$

where $\lambda = ln(h_T/z_{ref})$.

We can then find the bulk formula heat fluxes:

Sensible heat flux:

$$Q_s = \rho_{air} c_{p_{air}} u_s c_u c_T \Delta T$$

Latent heat flux:

$$Q_l = \rho_{air} L u_s c_u c_q \Delta q$$

Up long wave radiation

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

where ϵ is emissivity (which can be different for open ocean, ice and snow), σ is Stefan-Boltzman constant.

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

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

And total derivative $\frac{dQ_o}{dT} = \frac{dQ_s}{dT} + \frac{dQ_l}{dT} + \frac{dQ_{lw}^{up}}{dT}$.

If we do not read in the wind stress, it is calculated here.

Initializing subroutines

bulkf_init.F: Set bulkf variables to zero.
bulkf_readparms.F: Reads **data.blk**

Diagnostic subroutines

bulkf_ave.F: Keeps track of means of the bulkf variables
bulkf_diags.F: Finds averages and writes out diagnostics

Common Blocks

BULKF.h: BULKF Variables, data file names, and logicals **readwindstress** and **readsurface**
BULKF_DIAGS.h: matrices for diagnostics: averages of fields from *bulkf_diags.F*
BULKF_ICE_CONSTANTS.h: all the parameters need 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).

<u>**References**</u>

Bryan F.O., B.G Kauffman, W.G. Large, P.R. Gent, 1996: The NCAR CSM flux coupler. Technical note TN-425+STR, NCAR.

Hunke, E.C and W.H. Lipscomb, circa 2001: CICE: the Los Alamos Sea Ice Model Documentation and Software User's Manual. LACC-98-16v.2.
(note: this documentation is no longer available as CICE has progressed to a very different version 3)

6.4.6.1 Experiments and tutorials that use bulk_force

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

6.4.7 EXF: The external forcing package

Authors: Patrick Heimbach and Dimitris Menemenlis

6.4.7.1 Introduction

The external forcing package, in conjunction with the calendar package (cal), enables the handling of real-time (or "model-time") forcing fields of differing temporal forcing patterns. It comprises climatological restoring and relaxation. Bulk formulae are implemented to convert atmospheric fields to surface fluxes. An interpolation routine provides on-the-fly interpolation of forcing fields an arbitrary grid onto the model grid.

CPP options enable or disable different aspects of the package (Section 6.4.7.2). Runtime options, flags, filenames and field-related dates/times are set in `data.exf` (Section 6.4.7.3). A description of key subroutines is given in Section 6.4.7.6. Input fields, units and sign conventions are summarized in Section 6.4.7.5, and available diagnostics output is listed in Section 6.4.7.7.

6.4.7.2 EXF configuration, compiling & running

Compile-time options

As with all MITgcm packages, EXF can be turned on or off at compile time

- using the `packages.conf` file by adding `exf` to it,
- or using `genmake2` adding `-enable=exf` or `-disable=exf` switches
- *required packages and CPP options*:
  EXF requires the calendar package `cal` to be enabled; no additional CPP options are required.

(see Section 3.4).

Parts of the EXF code can be enabled or disabled at compile time via CPP preprocessor flags. These options are set in either `EXF_OPTIONS.h` or in `ECCO_CPPOPTIONS.h`. Table 6.9 summarizes these options.

6.4.7.3 Run-time parameters

Run-time parameters are set in files `data.pkg` and `data.exf` which is read in `exf_readparms.F`. Run-time parameters may be broken into 3 categories: (i) switching on/off the package at runtime, (ii) general flags and parameters, and (iii) attributes for each forcing and climatological field.

Enabling the package

A package is switched on/off at runtime by setting (e.g. for EXF) `useEXF = .TRUE.` in `data.pkg`.

General flags and parameters

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

Table 6.9:

Flag/parameter	default	Description
useExfCheckRange	.TRUE.	check range of input fields and stop if out of range
useExfYearlyFields	.FALSE.	append current year postfix of form _YYYY on filename
twoDigitYear	.FALSE.	instead of appending _YYYY append YY
repeatPeriod	0.0	>0 : cycle through all input fields at the same period (in seconds) $=0$: use period assigned to each field
exf_offset_atemp	0.0	set to 273.16 to convert from deg. Kelvin (assumed input) to Celsius
windstressmax	2.0	max. allowed wind stress N/m^2
exf_albedo	0.1	surface albedo used to compute downward vs. net radiative fluxes
climtempfreeze	-1.9	???
ocean_emissivity		longwave ocean-surface emissivity
ice_emissivity		longwave seaice emissivity
snow_emissivity		longwave snow emissivity
exf_iceCd	1.63E-3	drag coefficient over sea-ice
exf_iceCe	1.63E-3	evaporation transfer coeff. over sea-ice
exf_iceCh	1.63E-3	sensible heat transfer coeff. over sea-ice
exf_scal_BulkCdn	1.	overall scaling of neutral drag coeff.
useStabilityFct_overIce	.FALSE.	compute turbulent transfer coeff. over sea-ice
readStressOnAgrid	.FALSE.	read wind-streess located on model-grid, A-grid point
readStressOnCgrid	.FALSE.	read wind-streess located on model-grid, C-grid point
useRelativeWind	.FALSE.	subtract [U/V]VEL or [U/VICE from U/V]WIND before computing [U/V]STRESS
zref	10.	reference height
hu	10.	height of mean wind
ht	2.	height of mean temperature and rel. humidity
umin	0.5	minimum absolute wind speed for computing Cd
atmrho	1.2	mean atmospheric density [kg/m$\hat{3}$]
atmcp	1005.	mean atmospheric specific heat [J/kg/K]
cdrag_[n]	???	n = 1,2,3; parameters for drag coeff. function
cstanton_[n]	???	n = 1,2; parameters for Stanton number function
cdalton	???	parameter for Dalton number function
flamb	2500000.	latent heat of evaporation [J/kg]
flami	334000.	latent heat of melting of pure ice [J/kg]
zolmin	-100.	minimum stability parameter
cvapor_fac	640380.	
cvapor_exp	5107.4	
cvapor_fac_ice	11637800.	
cvapor_fac_ice	5897.8	
humid_fac	0.606	parameter for virtual temperature calculation
gamma_blk	0.010	adiabatic lapse rate
saltsat	0.980	reduction of saturation vapor pressure over salt-water
psim_fac	5.	
exf_monFreq	monitorFreq	output frequency [s]
exf_iprec	32	precision of input fields (32-bit or 64-bit)
exf_yftype	'RL'	precision of arrays ('RL' vs. 'RS')

Table 6.10:

Field attributes

All EXF fields are listed in Section 6.4.7.5. Each field has a number of attributes which can be customized. They are summarized in Table 6.11. To obtain an attribute for a specific field, e.g. `uwind` prepend the field name to the listed attribute, e.g. for attribute `period` this yields `uwindperiod`:

$$\begin{array}{ccccc} & \texttt{field} & \& & \texttt{attribute} & \longrightarrow & \texttt{parameter} \\ \text{e.g.} & \text{uwind} & \& & \text{period} & \longrightarrow & \text{uwindperiod} \end{array}$$

Example configuration

The following block is taken from the `data.exf` file of the verification experiment `global_with_exf/`. It defines attributes for the heat flux variable `hflux`:

```
hfluxfile       = 'ncep_qnet.bin',
hfluxstartdate1 = 19920101,
hfluxstartdate2 = 000000,
hfluxperiod     = 2592000.0,
hflux_lon0      = 2
```

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

Table 6.11:Note one exception for the default of `atempconst` = celsius2K = 273.16

```
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.

6.4.7.4 EXF bulk formulae

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

6.4.7.5 EXF input fields and units

The following list is taken from the header file `EXF_FIELDS.h`. It comprises all EXF input fields.

Output fields which EXF provides to the MITgcm are fields **fu**, **fv**, **Qnet**, **Qsw**, **EmPmR**, and **pload**. They are defined in `FFIELDS.h`.

```
c----------------------------------------------------------------------
c             |
c     field   :: Description
c             |
c----------------------------------------------------------------------
c     ustress   :: Zonal surface wind stress in N/m^2
c             |  > 0 for increase in uVel, which is west to
c             |      east for cartesian and spherical polar grids
c             |  Typical range: -0.5 < ustress < 0.5
c             |  Southwest C-grid U point
c             |  Input field
c----------------------------------------------------------------------
c     vstress   :: Meridional surface wind stress in N/m^2
c             |  > 0 for increase in vVel, which is south to
c             |      north for cartesian and spherical polar grids
c             |  Typical range: -0.5 < vstress < 0.5
c             |  Southwest C-grid V point
```

```
c              |  Input field
c---------------------------------------------------------------------
c     hs       :: sensible heat flux into ocean in W/m^2
c              |  > 0 for increase in theta (ocean warming)
c---------------------------------------------------------------------
c     hl       :: latent   heat flux into ocean in W/m^2
c              |  > 0 for increase in theta (ocean warming)
c---------------------------------------------------------------------
c     hflux    :: Net upward surface heat flux in W/m^2
c              |  (including shortwave)
c              |  hflux = latent + sensible + lwflux + swflux
c              |  > 0 for decrease in theta (ocean cooling)
c              |  Typical range: -250 < hflux < 600
c              |  Southwest C-grid tracer point
c              |  Input field
c---------------------------------------------------------------------
c     sflux    :: Net upward freshwater flux in m/s
c              |  sflux = evap - precip - runoff
c              |  > 0 for increase in salt (ocean salinity)
c              |  Typical range: -1e-7 < sflux < 1e-7
c              |  Southwest C-grid tracer point
c              |  Input field
c---------------------------------------------------------------------
c     swflux   :: Net upward shortwave radiation in W/m^2
c              |  swflux = - ( swdown - ice and snow absorption - reflected )
c              |  > 0 for decrease in theta (ocean cooling)
c              |  Typical range: -350 < swflux < 0
c              |  Southwest C-grid tracer point
c              |  Input field
c---------------------------------------------------------------------
c     uwind    :: Surface (10-m) zonal wind velocity in m/s
c              |  > 0 for increase in uVel, which is west to
c              |      east for cartesian and spherical polar grids
c              |  Typical range: -10 < uwind < 10
c              |  Southwest C-grid U point
c              |  Input or input/output field
c---------------------------------------------------------------------
c     vwind    :: Surface (10-m) meridional wind velocity in m/s
c              |  > 0 for increase in vVel, which is south to
c              |      north for cartesian and spherical polar grids
c              |  Typical range: -10 < vwind < 10
c              |  Southwest C-grid V point
c              |  Input or input/output field
c---------------------------------------------------------------------
c     wspeed   :: Surface (10-m) wind speed in m/s
c              |  >= 0 sqrt(u^2+v^2)
c              |  Typical range: 0 < wspeed < 10
c              |  Input or input/output field
c---------------------------------------------------------------------
c     atemp    :: Surface (2-m) air temperature in deg K
c              |  Typical range: 200 < atemp < 300
c              |  Southwest C-grid tracer point
c              |  Input or input/output field
c---------------------------------------------------------------------
c     aqh      :: Surface (2m) specific humidity in kg/kg
c              |  Typical range: 0 < aqh < 0.02
c              |  Southwest C-grid tracer point
c              |  Input or input/output field
c---------------------------------------------------------------------
c     lwflux   :: Net upward longwave radiation in W/m^2
c              |  lwflux = - ( lwdown - ice and snow absorption - emitted )
c              |  > 0 for decrease in theta (ocean cooling)
c              |  Typical range: -20 < lwflux < 170
c              |  Southwest C-grid tracer point
c              |  Input field
c---------------------------------------------------------------------
c     evap     :: Evaporation in m/s
c              |  > 0 for increase in salt (ocean salinity)
c              |  Typical range: 0 < evap < 2.5e-7
c              |  Southwest C-grid tracer point
c              |  Input, input/output, or output field
```

```
c----------------------------------------------------------------------
c     precip    :: Precipitation in m/s
c               |  > 0 for decrease in salt (ocean salinity)
c               |  Typical range: 0 < precip < 5e-7
c               |  Southwest C-grid tracer point
c               |  Input or input/output field
c----------------------------------------------------------------------
c    snowprecip :: snow in m/s
c               |  > 0 for decrease in salt (ocean salinity)
c               |  Typical range: 0 < precip < 5e-7
c               |  Input or input/output field
c----------------------------------------------------------------------
c     runoff    :: River and glacier runoff in m/s
c               |  > 0 for decrease in salt (ocean salinity)
c               |  Typical range: 0 < runoff < ????
c               |  Southwest C-grid tracer point
c               |  Input or input/output field
c               |  !!! WATCH OUT: Default exf_inscal_runoff !!!
c               |  !!! in exf_readparms.F is not 1.0        !!!
c----------------------------------------------------------------------
c     swdown    :: Downward shortwave radiation in W/m^2
c               |  > 0 for increase in theta (ocean warming)
c               |  Typical range: 0 < swdown < 450
c               |  Southwest C-grid tracer point
c               |  Input/output field
c----------------------------------------------------------------------
c     lwdown    :: Downward longwave radiation in W/m^2
c               |  > 0 for increase in theta (ocean warming)
c               |  Typical range: 50 < lwdown < 450
c               |  Southwest C-grid tracer point
c               |  Input/output field
c----------------------------------------------------------------------
c     apressure :: Atmospheric pressure field in N/m^2
c               |  > 0 for ????
c               |  Typical range: ???? < apressure < ????
c               |  Southwest C-grid tracer point
c               |  Input field
c----------------------------------------------------------------------
```

6.4.7.6 Key subroutines

Top-level routine: `exf_getforcing.F`

```
C     !CALLING SEQUENCE:
c ...
c  exf_getforcing (TOP LEVEL ROUTINE)
c  |
c  |-- exf_getclim (get climatological fields used e.g. for relax.)
c  |   |--- exf_set_climsst  (relax. to 2-D SST field)
c  |   |--- exf_set_climsss  (relax. to 2-D SSS field)
c  |   o
c  |
c  |-- exf_getffields <- this one does almost everything
c  |   |   1. reads in fields, either flux or atmos. state,
c  |   |      depending on CPP options (for each variable two fields
c  |   |      consecutive in time are read in and interpolated onto
c  |   |      current time step).
c  |   |   2. If forcing is atmos. state and control is atmos. state,
c  |   |      then the control variable anomalies are read here via ctrl_get_gen
c  |   |      (atemp, aqh, precip, swflux, swdown, uwind, vwind).
c  |   |      If forcing and control are fluxes, then
c  |   |      controls are added later.
c  |   o
c  |
c  |-- exf_radiation
c  |   |    Compute net or downwelling radiative fluxes via
c  |   |    Stefan-Boltzmann law in case only one is known.
c  |   o
```

```
c  |-- exf_wind
c  |   |   Computes wind speed and stresses, if required.
c  |   o
c  |
c  |-- exf_bulkformulae
c  |   |   Compute air-sea buoyancy fluxes from
c  |   |   atmospheric state following Large and Pond, JPO, 1981/82
c  |   o
c  |
c  |-- < hflux is sum of sensible, latent, longwave rad. >
c  |-- < sflux is sum of evap. minus precip. minus runoff  >
c  |
c  |-- exf_getsurfacefluxes
c  |   If forcing and control is flux, then the
c  |   control vector anomalies are read here via ctrl_get_gen
c  |   (hflux, sflux, ustress, vstress)
c  |
c  |-- < update tile edges here >
c  |
c  |-- exf_check_range
c  |   |   Check whether read fields are within assumed range
c  |   |   (may capture mismatches in units)
c  |   o
c  |
c  |-- < add shortwave to hflux for diagnostics >
c  |
c  |-- exf_diagnostics_fill
c  |   |   Do EXF-related diagnostics output here.
c  |   o
c  |
c  |-- exf_mapfields
c  |   |   Forcing fields from exf package are mapped onto
c  |   |   mitgcm forcing arrays.
c  |   |   Mapping enables a runtime rescaling of fields
c  |   o
C  o
```

Radiation calculation: `exf_radiation.F`
Wind speed and stress calculation: `exf_wind.F`
Bulk formula: `exf_bulkformulae.F`
Generic I/O: `exf_set_gen.F`
Interpolation: `exf_interp.F`
Header routines

6.4.7.7 EXF diagnostics

Diagnostics output is available via the diagnostics package (see Section 7.1). Available output fields are summarized in Table 6.12.

6.4.7.8 Experiments and tutorials that use exf

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

6.4.7.9 References

```
---------+----+----+----------------+-----------------
 <-Name->|Levs|grid|<--  Units   -->|<- Tile (max=80c)
---------+----+----+----------------+-----------------
 EXFhs   |  1 | SM | W/m^2          | Sensible heat flux into ocean, >0 increases theta
 EXFhl   |  1 | SM | W/m^2          | Latent heat flux into ocean, >0 increases theta
 EXFlwnet|  1 | SM | W/m^2          | Net upward longwave radiation, >0 decreases theta
 EXFswnet|  1 | SM | W/m^2          | Net upward shortwave radiation, >0 decreases theta
 EXFlwdn |  1 | SM | W/m^2          | Downward longwave radiation, >0 increases theta
 EXFswdn |  1 | SM | W/m^2          | Downward shortwave radiation, >0 increases theta
 EXFqnet |  1 | SM | W/m^2          | Net upward heat flux (turb+rad), >0 decreases theta
 EXFtaux |  1 | SU | N/m^2          | zonal surface wind stress, >0 increases uVel
 EXFtauy |  1 | SV | N/m^2          | meridional surface wind stress, >0 increases vVel
 EXFuwind|  1 | SM | m/s            | zonal 10-m wind speed, >0 increases uVel
 EXFvwind|  1 | SM | m/s            | meridional 10-m wind speed, >0 increases uVel
 EXFwspee|  1 | SM | m/s            | 10-m wind speed modulus ( >= 0 )
 EXFatemp|  1 | SM | degK           | surface (2-m) air temperature
 EXFaqh  |  1 | SM | kg/kg          | surface (2-m) specific humidity
 EXFevap |  1 | SM | m/s            | evaporation, > 0 increases salinity
 EXFpreci|  1 | SM | m/s            | evaporation, > 0 decreases salinity
 EXFsnow |  1 | SM | m/s            | snow precipitation, > 0 decreases salinity
 EXFempmr|  1 | SM | m/s            | net upward freshwater flux, > 0 increases salinity
 EXFpress|  1 | SM | N/m^2          | atmospheric pressure field
```

Table 6.12:

6.4.8 CAL: The calendar package

Authors: Christian Eckert and Patrick Heimbach

This calendar tool was originally intended to enable the use of absolute dates (Gregorian Calendar dates) in MITgcm. There is, however, a fair number of routines that can be used independently of the main MITgcm executable. After some minor modifications the whole package can be used either as a stand-alone calendar or in connection with any dynamical model that needs calendar dates. Some straightforward extensions are still pending e.g. the availability of the Julian Calendar, to be able to resolve fractions of a second, and to have a time- step that is longer than one day.

6.4.8.1 Basic assumptions for the calendar tool

It is assumed that the SMALLEST TIME INTERVAL to be resolved is ONE SECOND.

Further assumptions are that there is an INTEGER NUMBER OF MODEL STEPS EACH DAY, and that AT LEAST ONE STEP EACH DAY is made.

Not each individual routine depends on these assumptions; there are only a few places where they enter.

6.4.8.2 Format of calendar dates

In this calendar tool a complete date specification is defined as the following integer array:

```
c           integer date(4)
c
c           ( yyyymmdd, hhmmss, leap_year, dayofweek )
c
c             date(1) = yyyymmdd    <-- Year-Month-Day
c             date(2) =   hhmmss    <-- Hours-Minutes-Seconds
c             date(3) = leap_year   <-- Leap Year/No Leap Year
c             date(4) = dayofweek   <-- Day of the Week
c
c             leap_year is either equal to 1 (normal year)
c                                 or equal to 2 (leap year)
c
c             dayofweek has a range of 1 to 7.
```

In case the Gregorian Calendar is used, the first day of the week is Friday, since day of the Gregorian Calendar was Friday, 15 Oct. 1582. As a date array this date would be specified as

```
c               refdate(1) = 15821015
c               refdate(2) =        0
c               refdate(3) =        1
c               refdate(4) =        1
```

6.4.8.3 Calendar dates and time intervals

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

```
c         integer datediff(4)
c
c           datediff(1) = # Days
c           datediff(2) = hhmmss
c           datediff(3) =      0
c           datediff(4) =     -1
```

Such time intervals can be added to or can be subtracted from calendar dates. Time intervals can be added to and be subtracted from each other.

6.4.8.4 Using the calendar together with MITgcm

Each routine has as an argument the thread number that it is belonging to, even if this number is not used in the routine itself.

In order to include the calendar tool into the MITgcm setup the MITgcm subroutine "initialise.F" or the routine "initilise_fixed.F", depending on the MITgcm release, has to be modified in the following way:

```
c         #ifdef ALLOW_CALENDAR
c         C--   Initialise the calendar package.
c         #ifdef USE_CAL_NENDITER
c               CALL cal_Init(
c              I              startTime,
c              I              endTime,
c              I              deltaTclock,
c              I              nIter0,
c              I              nEndIter,
c              I              nTimeSteps,
c              I              myThid
c              &            )
c         #else
c               CALL cal_Init(
c              I              startTime,
c              I              endTime,
c              I              deltaTclock,
c              I              nIter0,
c              I              nTimeSteps,
c              I              myThid
c              &            )
c         #endif
c               _BARRIER
c         #endif
```

It is useful to have the CPP flag ALLOW_CALENDAR in order to switch from the usual MITgcm setup to the one that includes the calendar tool. The CPP flag USE_CAL_NENDITER has been introduced in order to enable the use of the calendar for MITgcm releases earlier than checkpoint 25 which do not have the global variable *nEndIter*.

6.4.8.5 The individual calendars

Simple model calendar:

This calendar can be used by defining

```
c                 TheCalendar='model'
```

in the calendar's data file "data.cal".

In this case a year is assumed to have 360 days. The model year is divided into 12 months with 30 days each.

Gregorian Calendar:

This calendar can be used by defining

```
c                 TheCalendar='gregorian'
```

in the calendar's data file "data.cal".

6.4.8.6 Short routine description

```
c      o  cal_Init          - Initialise the calendar. This is the interface
c                             to MITgcm.
c
c      o  cal_Set           - Sets the calendar according to the user
c                             specifications.
c
c      o  cal_GetDate       - Given the model's current timestep or the
c                             model's current time return the corresponding
c                             calendar date.
c
c      o  cal_FullDate      - Complete a date specification (leap year and
c                             day of the week).
c
c      o  cal_IsLeap        - Determine whether a given year is a leap year.
c
c      o  cal_TimePassed    - Determine the time passed between two dates.
c
c      o  cal_AddTime       - Add a time interval either to a time interval
c                             or to a date.
c
c      o  cal_TimeInterval  - Given a time interval return the corresponding
c                             date array.
c
c      o  cal_SubDates      - Determine the time interval between two dates
c                             or between two time intervals.
c
c      o  cal_ConvDate      - Decompose a date array or a time interval
c                             array into its components.
c
c      o  cal_CopyDate      - Copy a date array or a time interval array to
c                             another array.
c
c      o  cal_CompDates     - Compare two calendar dates or time intervals.
c
c      o  cal_ToSeconds     - Given a time interval array return the number
c                             of seconds.
c
c      o  cal_WeekDay       - Return the weekday as a string given the
c                             calendar date.
c
c      o  cal_NumInts       - Return the number of time intervals between two
c                             given dates.
c
c      o  cal_StepsPerDay   - Given an iteration number or the current
c                             integration time return the number of time
c                             steps to integrate in the current calendar day.
c
c      o  cal_DaysPerMonth  - Given an iteration number or the current
c                             integration time return the number of days
```

```
c                            to integrate in this calendar month.
c
c      o  cal_MonthsPerYear - Given an iteration number or the current
c                             integration time return the number of months
c                             to integrate in the current calendar year.
c
c      o  cal_StepsForDay   - Given the integration day return the number
c                             of steps to be integrated, the first step,
c                             and the last step in the day specified. The
c                             first and the last step refer to the total
c                             number of steps (1, ... , cal_IntSteps).
c
c      o  cal_DaysForMonth  - Given the integration month return the number
c                             of days to be integrated, the first day,
c                             and the last day in the month specified. The
c                             first and the last day refer to the total
c                             number of steps (1, ... , cal_IntDays).
c
c      o  cal_MonthsForYear - Given the integration year return the number
c                             of months to be integrated, the first month,
c                             and the last month in the year specified. The
c                             first and the last step refer to the total
c                             number of steps (1, ... , cal_IntMonths).
c
c      o  cal_Intsteps      - Return the number of calendar years that are
c                             affected by the current integration.
c
c      o  cal_IntDays       - Return the number of calendar days that are
c                             affected by the current integration.
c
c      o  cal_IntMonths     - Return the number of calendar months that are
c                             affected by the current integration.
c
c      o  cal_IntYears      - Return the number of calendar years that are
c                             affected by the current integration.
c
c      o  cal_nStepDay      - Return the number of time steps that can be
c                             performed during one calendar day.
c
c      o  cal_CheckDate     - Do some simple checks on a date array or on a
c                             time interval array.
c
c      o  cal_PrintError    - Print error messages according to the flags
c                             raised by the calendar routines.
c
c      o  cal_PrintDate     - Print a date array in some format suitable for
c                             MITgcm's protocol output.
c
c      o  cal_TimeStamp     - Given the time and the iteration number return
c                             the date and print all the above numbers.
c
c      o  cal_Summary       - List all the setttings of the calendar tool.
```

6.4.8.7 Experiments and tutorials that use cal

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

6.5 Atmosphere Packages

6.5.1 Atmospheric Intermediate Physics: AIM

Note: The folowing document below describes the aim_v23 package that is based on the version v23 of the SPEEDY code (*Molteni* [2003]).

6.5.1.1 Key subroutines, parameters and files

6.5.1.2 AIM Diagnostics

```
------------------------------------------------------------------------
<-Name->|Levs|<-parsing code->|<--  Units   -->|<- Tile (max=80c)
------------------------------------------------------------------------
DIABT   |  5 |SM      ML      |K/s             |Pot. Temp.  Tendency (Mass-Weighted) from Diabatic Processes
DIABQ   |  5 |SM      ML      |g/kg/s          |Spec.Humid. Tendency (Mass-Weighted) from Diabatic Processes
RADSW   |  5 |SM      ML      |K/s             |Temperature Tendency due to Shortwave Radiation (TT_RSW)
RADLW   |  5 |SM      ML      |K/s             |Temperature Tendency due to Longwave  Radiation (TT_RLW)
DTCONV  |  5 |SM      MR      |K/s             |Temperature Tendency due to Convection (TT_CNV)
TURBT   |  5 |SM      ML      |K/s             |Temperature Tendency due to Turbulence in PBL (TT_PBL)
DTLS    |  5 |SM      ML      |K/s             |Temperature Tendency due to Large-scale condens. (TT_LSC)
DQCONV  |  5 |SM      MR      |g/kg/s          |Spec. Humidity Tendency due to Convection (QT_CNV)
TURBQ   |  5 |SM      ML      |g/kg/s          |Spec. Humidity Tendency due to Turbulence in PBL (QT_PBL)
DQLS    |  5 |SM      ML      |g/kg/s          |Spec. Humidity Tendency due to Large-Scale Condens. (QT_LSC)
TSR     |  1 |SM P    U1      |W/m^2           |Top-of-atm. net Shortwave Radiation (+=dw)
OLR     |  1 |SM P    U1      |W/m^2           |Outgoing Longwave  Radiation (+=up)
RADSWG  |  1 |SM P    L1      |W/m^2           |Net Shortwave Radiation at the Ground (+=dw)
RADLWG  |  1 |SM      L1      |W/m^2           |Net Longwave  Radiation at the Ground (+=up)
HFLUX   |  1 |SM      L1      |W/m^2           |Sensible Heat Flux (+=up)
EVAP    |  1 |SM      L1      |g/m^2/s         |Surface Evaporation (g/m2/s)
PRECON  |  1 |SM P    L1      |g/m^2/s         |Convective  Precipitation (g/m2/s)
PRECLS  |  1 |SM      M1      |g/m^2/s         |Large Scale Precipitation (g/m2/s)
CLDFRC  |  1 |SM P    M1      |0-1             |Total Cloud Fraction (0-1)
CLDPRS  |  1 |SM PC167M1      |0-1             |Cloud Top Pressure (normalized)
CLDMAS  |  5 |SM P    LL      |kg/m^2/s        |Cloud-base Mass Flux  (kg/m^2/s)
DRAG    |  5 |SM P    LL      |kg/m^2/s        |Surface Drag Coefficient (kg/m^2/s)
WINDS   |  1 |SM P    L1      |m/s             |Surface Wind Speed  (m/s)
TS      |  1 |SM      L1      |K               |near Surface Air Temperature  (K)
QS      |  1 |SM P    L1      |g/kg            |near Surface Specific Humidity  (g/kg)
ENPREC  |  1 |SM      M1      |W/m^2           |Energy flux associated with precip. (snow, rain Temp)
ALBVISDF|  1 |SM P    L1      |0-1             |Surface Albedo (Visible band) (0-1)
DWNLWG  |  1 |SM P    L1      |W/m^2           |Downward Component of Longwave Flux at the Ground (+=dw)
SWCLR   |  5 |SM      ML      |K/s             |Clear Sky Temp. Tendency due to Shortwave Radiation
LWCLR   |  5 |SM      ML      |K/s             |Clear Sky Temp. Tendency due to Longwave  Radiation
TSRCLR  |  1 |SM P    U1      |W/m^2           |Clear Sky Top-of-atm. net Shortwave Radiation (+=dw)
OLRCLR  |  1 |SM P    U1      |W/m^2           |Clear Sky Outgoing Longwave  Radiation  (+=up)
SWGCLR  |  1 |SM P    L1      |W/m^2           |Clear Sky Net Shortwave Radiation at the Ground (+=dw)
LWGCLR  |  1 |SM      L1      |W/m^2           |Clear Sky Net Longwave  Radiation at the Ground (+=up)
UFLUX   |  1 |UM   184L1      |N/m^2           |Zonal Wind Surface Stress  (N/m^2)
VFLUX   |  1 |VM   183L1      |N/m^2           |Meridional Wind Surface Stress  (N/m^2)
DTSIMPL |  1 |SM P    L1      |K               |Surf. Temp Change after 1 implicit time step
```

6.5.1.3 Experiments and tutorials that use aim

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

6.5.2 Land package

6.5.2.1 Introduction

This package provides a simple land model based on Rong Zhang [e-mail:roz@gfdl.noaa.gov] 2 layers model (see documentation below).

It is primarily implemented for AIM (_v23) atmospheric physics but could be adapted to work with a different atmospheric physics. Two subroutines (*aim_aim2land.F aim_land2aim.F* in *pkg/aim_v23*) are used as interface with AIM physics.

Number of layers is a parameter (*land_nLev* in *LAND_SIZE.h*) and can be changed.

Note on Land Model
date: June 1999
author: Rong Zhang

6.5.2.2 Equations and Key Parameters

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

Let T_{g1}, T_{g2} be the temperature of each layer, $W_{1,} W_2$ be the soil moisture of each layer. The field capacity $f_{1,}$ f_2 are the maximum water amount in each layer, so W_i is the ratio of available water to field capacity. $f_i = \gamma z_i, \gamma = 0.24$ is the field capapcity per meter soil, so $f_1 = 0.024m$, $f_2 = 0.96m$.

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

$$z_1 C_1 \frac{dT_{g1}}{dt} = F - \lambda \frac{T_{g1} - T_{g2}}{(z_1 + z_2)/2} \tag{6.32}$$

$$z_2 C_2 \frac{dT_{g2}}{dt} = \lambda \frac{T_{g1} - T_{g2}}{(z_1 + z_2)/2} \tag{6.33}$$

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

$$C_1 = C_w W_1 \gamma + C_s \tag{6.34}$$

$$C_2 = C_w W_2 \gamma + C_s \tag{6.35}$$

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

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

$$\frac{dW_1}{dt} = \frac{P - E - R}{f_1} + \frac{W_2 - W_1}{\tau} \tag{6.36}$$

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

$$\frac{dW_2}{dt} = \frac{f_1}{f_2} \frac{W_1 - W_2}{\tau} \tag{6.37}$$

In the code, $R = 0$ gives better result, W_1, W_2 are set to be within [0, 1]. If W_1 is greater than 1, then let $\delta W_1 = W_1 - 1, W_1 = 1$ and $W_2 = W_2 + p \delta W_1 \frac{f_1}{f_2}$, i.e. the runoff of top layer is put into second layer. $p = 0.5$ is the fraction of top layer runoff that is put into second layer.

The time step is 1 hour, it takes several years to reach equalibrium offline.

6.5.2.3 Land diagnostics

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

6.5.2.4 References

Hansen J. et al. Efficient three-dimensional global models for climate studies: models I and II. *Monthly Weather Review*, vol.111, no.4, pp. 609-62, 1983

6.5.2.5 Experiments and tutorials that use land

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

6.5.3 Fizhi: High-end Atmospheric Physics

6.5.3.1 Introduction

The fizhi (high-end atmospheric physics) package includes a collection of state-of-the-art physical parameterizations for atmospheric radiation, cumulus convection, atmospheric boundary layer turbulence, and land surface processes. The collection of atmospheric physics parameterizations were originally used together as part of the GEOS-3 (Goddard Earth Observing System-3) GCM developed at the NASA/Goddard Global Modelling and Assimilation Office (GMAO).

6.5.3.2 Equations

Moist Convective Processes:

Sub-grid and Large-scale Convection Sub-grid scale cumulus convection is parameterized using the Relaxed Arakawa Schubert (RAS) scheme of *Moorthi and Suarez* [1992], 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, η, normalized by the cloud base mass flux, is a linear function of height, expressed as:

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

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

$$\frac{\partial z}{\partial P^\kappa} = -\frac{c_p}{g}\theta$$

The entrainment parameter, λ, characterizes a particular subensemble based on its detrainment level, and is obtained by assuming that the level of detrainment is the level of neutral buoyancy, ie., the level at which the moist static energy of the cloud, h_c, is equal to the saturation moist static energy of the environment, h^*. Following *Moorthi and Suarez* [1992], λ may be written as

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

where the subscript B refers to cloud base, and the subscript D refers to the detrainment level.

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

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

where γ is $\frac{L}{c_p}\frac{\partial q^*}{\partial T}$ obtained from the Claussius Clapeyron equation, and the subscript c refers to the value inside the cloud.

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

$$m_B = \frac{-\left.\frac{dA}{dt}\right|_{ls}}{K}$$

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

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

$$\left.\frac{\partial\theta}{\partial t}\right|_c = \alpha\frac{m_B}{c_p P^\kappa}\eta\frac{\partial s}{\partial p}$$

and

$$\left.\frac{\partial q}{\partial t}\right|_c = \alpha\frac{m_B}{L}\eta(\frac{\partial h}{\partial p} - \frac{\partial s}{\partial p})$$

where $\theta = \frac{T}{P^\kappa}$, $P = (p/p_0)$, and α is the relaxation parameter.

As an approximation to a full interaction between the different allowable subensembles, many clouds are simulated frequently, each modifying the large scale environment some fraction α of the total adjustment. The parameterization thereby "relaxes" the large scale environment towards equillibrium.

In addition to the RAS cumulus convection scheme, the fizhi package employs a Kessler-type scheme for the re-evaporation of falling rain (*Sud and Molod* [1988]), which correspondingly adjusts the temperature assuming h is conserved. RAS in its current formulation assumes that all cloud water is deposited into the detrainment level as rain. All of the rain is available for re-evaporation, which begins in the level below detrainment. The scheme accounts for some microphysics such as the rainfall intensity, the drop size distribution, as well as the temperature, pressure and relative humidity of the surrounding air. The fraction of the moisture deficit in any model layer into which the rain may re-evaporate is controlled by a free parameter, which allows for a relatively efficient re-evaporation of liquid precipitate and larger rainout for frozen precipitation.

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

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

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

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

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

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

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

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

$$F_{LS} = \min\left[\left(\frac{RH - RH_c}{1 - RH_c}\right)^2, 1.0\right]$$

where

$$
\begin{aligned}
RH_c &= 1 - s(1-s)(2 - \sqrt{3} + 2\sqrt{3}\,s)r \\
s &= p/p_{surf} \\
r &= \left(\frac{1.0 - RH_{min}}{\alpha}\right) \\
RH_{min} &= 0.75 \\
\alpha &= 0.573285.
\end{aligned}
$$

These cloud fractions are suppressed, however, in regions where the convective sub-cloud layer is conditionally unstable. The functional form of RH_c is shown in Figure (6.9).

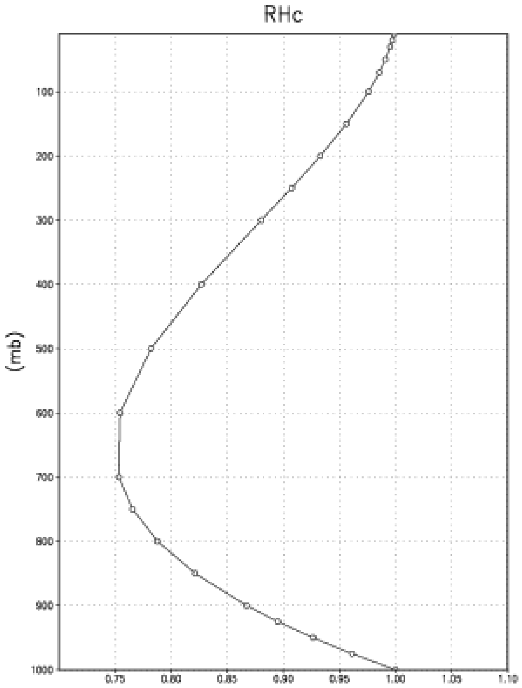


Figure 6.9: Critical Relative Humidity for Clouds.

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

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

Finally, cloud fractions are time-averaged between calls to the radiation packages.

Radiation:

The parameterization of radiative heating in the fizhi package includes effects from both shortwave and longwave processes. Radiative fluxes are calculated at each model edge-level in both up and down directions. The heating rates/cooling rates are then obtained from the vertical divergence of the net radiative fluxes.

The net flux is

$$F = F^{\uparrow} - F^{\downarrow}$$

where F is the net flux, $F^{\uparrow}$ is the upward flux and $F^{\downarrow}$ is the downward flux.

The heating rate due to the divergence of the radiative flux is given by

$$\frac{\partial \rho c_p T}{\partial t} = -\frac{\partial F}{\partial z}$$

or

$$\frac{\partial T}{\partial t} = \frac{g}{c_p \pi} \frac{\partial F}{\partial \sigma}$$

where g is the accelation due to gravity and c_p is the heat capacity of air at constant pressure.

The time tendency for Longwave Radiation is updated every 3 hours. The time tendency for Shortwave Radiation is updated once every three hours assuming a normalized incident solar radiation, and subsequently modified at every model time step by the true incident radiation. The solar constant value used in the package is equal to 1365 W/m^2 and a CO_2 mixing ratio of 330 ppm. For the ozone mixing ratio, monthly mean zonally averaged climatological values specified as a function of latitude and height (*Rosenfield et al.* [1987]) are linearly interpolated to the current time.

Shortwave Radiation The shortwave radiation package used in the package computes solar radiative heating due to the absoption by water vapor, ozone, carbon dioxide, oxygen, clouds, and aerosols and due to the scattering by clouds, aerosols, and gases. The shortwave radiative processes are described by *Chou* [1990, 1992]. 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 *Lacis and Hansen* [1974].

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 6.13, 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 6.14. 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.

UV and Visible Spectral Regions

Region	Band	Wavelength (micron)
UV-C	1.	.175 - .225
	2.	.225 - .245
		.260 - .280
	3.	.245 - .260
UV-B	4.	.280 - .295
	5.	.295 - .310
	6.	.310 - .320
UV-A	7.	.320 - .400
PAR	8.	.400 - .700

Table 6.13: UV and Visible Spectral Regions used in shortwave radiation package.

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

To simplify calculations in a cloudy atmosphere, clouds are grouped into low ($p > 700$ mb), middle (700 mb $\geq p > 400$ mb), and high ($p < 400$ mb) cloud regions. Within each of the three regions, clouds are assumed maximally overlapped, and the cloud cover of the group is the maximum cloud cover of all the layers in the group. The optical thickness of a given layer is then scaled for both the direct (as a

Infrared Spectral Regions

Band	Wavenumber(cm^{-1})	Wavelength (micron)
1	1000-4400	2.27-10.0
2	4400-8200	1.22-2.27
3	8200-14300	0.70-1.22

Table 6.14: Infrared Spectral Regions used in shortwave radiation package.

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 *Chou and M.J.Suarez* [1994]. 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 6.15.

IR Spectral Bands

Band	Spectral Range (cm^{-1})	Absorber	Method
1	0-340	H_2O line	T
2	340-540	H_2O line	T
3a	540-620	H_2O line	K
3b	620-720	H_2O continuum	S
3b	720-800	CO_2	T
4	800-980	H_2O line	K
		H_2O continuum	S
5	980-1100	H_2O line	K
		H_2O continuum	S
		O_3	T
6	1100-1380	H_2O line	K
		H_2O continuum	S
7	1380-1900	H_2O line	T
8	1900-3000	H_2O line	K

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

Table 6.15: IR Spectral Bands, Absorbers, and Parameterization Method (from *Chou and M.J.Suarez* [1994])

The longwave radiation package accurately computes cooling rates for the middle and lower atmosphere from 0.01 mb to the surface. Errors are < 0.4 C day^{-1} in cooling rates and $< 1\%$ in fluxes. From Chou and Suarez, it is estimated that the total effect of neglecting all minor absorption bands and the effects of minor infrared absorbers such as nitrous oxide (N_2O), methane (CH_4), and the chlorofluorocarbons (CFCs), is an underestimate of ≈ 5 W/m^2 in the downward flux at the surface and an overestimate of ≈ 3 W/m^2 in the upward flux at the top of the atmosphere.

Similar to the procedure used in the shortwave radiation package, clouds are grouped into three regions catagorized as low/middle/high. The net clear line-of-site probability (P) between any two levels, p_1 and p_2 $(p_2 > p_1)$, assuming randomly overlapped cloud groups, is simply the product of the probabilities within each group:

$$P_{net} = P_{low} \times P_{mid} \times P_{hi}.$$

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

$$P_{group} = 1 - F_{max},$$

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

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

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

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

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

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

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 6.5.3.2. The optical thickness for the longwave radiative feedback is assumed to be 75 % of these values.

The entire Moist Convective Processes Module is called with a frequency of 10 minutes. The cloud fraction values are time-averaged over the period between Radiation calls (every 3 hours). Therefore, in a time-averaged sense, both convective and large-scale cloudiness can exist in a given grid-box.

Turbulence :

Turbulence is parameterized in the fizhi package to account for its contribution to the vertical exchange of heat, moisture, and momentum. The turbulence scheme is invoked every 30 minutes, and employs a backward-implicit iterative time scheme with an internal time step of 5 minutes. The tendencies of atmospheric state variables due to turbulent diffusion are calculated using the diffusion equations:

$$\begin{aligned}
\frac{\partial u}{\partial t}_{turb} &= \frac{\partial}{\partial z}(-\overline{u^{\prime}w^{\prime}}) = \frac{\partial}{\partial z}(K_m \frac{\partial u}{\partial z}) \\
\frac{\partial v}{\partial t}_{turb} &= \frac{\partial}{\partial z}(-\overline{v^{\prime}w^{\prime}}) = \frac{\partial}{\partial z}(K_m \frac{\partial v}{\partial z}) \\
\frac{\partial T}{\partial t} = P^{\kappa}\frac{\partial \theta}{\partial t}_{turb} &= P^{\kappa}\frac{\partial}{\partial z}(-\overline{w^{\prime}\theta^{\prime}}) = P^{\kappa}\frac{\partial}{\partial z}(K_h \frac{\partial \theta_v}{\partial z}) \\
\frac{\partial q}{\partial t}_{turb} &= \frac{\partial}{\partial z}(-\overline{w^{\prime}q^{\prime}}) = \frac{\partial}{\partial z}(K_h \frac{\partial q}{\partial z})
\end{aligned}$$

Within the atmosphere, the time evolution of second turbulent moments is explicitly modeled by representing the third moments in terms of the first and second moments. This approach is known as a second-order closure modeling. To simplify and streamline the computation of the second moments, the level 2.5 assumption of Mellor and Yamada (1974) and *Yamada* [1977] is employed, in which only the turbulent kinetic energy (TKE),

$$\frac{1}{2}q^2 = \overline{u'^2} + \overline{v'^2} + \overline{w'^2},$$

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

$$\frac{d}{dt}(\frac{1}{2}q^2) - \frac{\partial}{\partial z}(\frac{5}{3}\lambda_1 q \frac{\partial}{\partial z}(\frac{1}{2}q^2)) = -\overline{u'w'}\frac{\partial U}{\partial z} - \overline{v'w'}\frac{\partial V}{\partial z} + \frac{g}{\Theta_0}\overline{w'\theta_v{}'} - \frac{q^3}{\Lambda_1}$$

where q is the turbulent velocity, u', v', w' and θ' are the fluctuating parts of the velocity components and potential temperature, U and V are the mean velocity components, ${\Theta_0}^{-1}$ is the coefficient of thermal expansion, and λ_1 and Λ_1 are constant multiples of the master length scale, ℓ, which is designed to be a characteristic measure of the vertical structure of the turbulent layers.

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

In the level 2.5 approach, the vertical fluxes of the scalars θ_v and q and the wind components u and v are expressed in terms of the diffusion coefficients K_h and K_m, respectively. In the statisically realizable level 2.5 turbulence scheme of *Helfand and Labraga* [1988], these diffusion coefficients are expressed as

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

and

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

where the subscript e refers to the value under conditions of local equillibrium (obtained from the Level 2.0 Model), ℓ is the master length scale related to the vertical structure of the atmosphere, and S_M and S_H are functions of G_H and G_M, the dimensionless buoyancy and wind shear parameters, respectively. Both G_H and G_M, and their equilibrium values G_{H_e} and G_{M_e}, are functions of the Richardson number:

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

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

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

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

and

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

where $u_* = C_u W_s$ is the surface friction velocity, C_D is termed the surface drag coefficient, C_H the heat transfer coefficient, and W_s is the magnitude of the surface layer wind.

C_u is the dimensionless exchange coefficient for momentum from the surface layer similarity functions:

$$C_u = \frac{u_*}{W_s} = \frac{k}{\psi_m}$$

where k is the Von Karman constant and ψ_m is the surface layer non-dimensional wind shear given by

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

Here ζ is the non-dimensional stability parameter, and ϕ_m is the similarity function of ζ which expresses the stability dependance of the momentum gradient. The functional form of ϕ_m is specified differently for stable and unstable layers.

C_t is the dimensionless exchange coefficient for heat and moisture from the surface layer similarity functions:

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

where ψ_h is the surface layer non-dimensional temperature gradient given by

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

Here ϕ_h is the similarity function of ζ, which expresses the stability dependance of the temperature and moisture gradients, and is specified differently for stable and unstable layers according to *Helfand and Schubert* [1995].

ψ_g is the non-dimensional temperature or moisture gradient in the viscous sublayer, which is the mosstly laminar region between the surface and the tops of the roughness elements, in which temperature and moisture gradients can be quite large. Based on *Yaglom and Kader* [1974]:

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

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

The surface roughness length over oceans is 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 *Kondo* [1975] for weak winds, and the piecewise linear relation of *Large and Pond* [1981] for moderate to large winds. Roughness lengths over land are specified from the climatology of *Dorman and Sellers* [1989].

For an unstable surface layer, the stability functions, chosen to interpolate between the condition of small values of β and the convective limit, are the KEYPS function (*Panofsky* [1973]) for momentum, and its generalization for heat and moisture:

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

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

For a stable surface layer, the stability functions are the observationally based functions of *Clarke* [1970], slightly modified for the momemtum flux:

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

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

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

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

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

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

where R_{sw} is the net surface downward shortwave radiative flux and R_{lw} is the net surface upward longwave radiative flux.

H is the upward sensible heat flux, given by:

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

where ρ = the atmospheric density at the surface, c_p is the specific heat of air at constant pressure, and θ represents the potential temperature of the surface and of the lowest σ-level, respectively.

The upward latent heat flux, LE, is given by

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

where β is the fraction of the potential evapotranspiration actually evaporated, L is the latent heat of evaporation, and $q_{surface}$ and q_{NLAY} are the specific humidity of the surface and of the lowest σ-level, respectively.

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

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

where C_{ti} is the thermal conductivity of ice, H_i is the ice thickness, assumed to be 3 m where sea ice is present, T_i is 273 degrees Kelvin, and T_g is the surface temperature of the ice.

C_g is the total heat capacity of the ground, obtained by solving a heat diffusion equation for the penetration of the diurnal cycle into the ground (*Blackadar* [1977]), and is given by:

$$C_g = \sqrt{\frac{\lambda C_s}{2\omega}} = \sqrt{(0.386 + 0.536W + 0.15W^2)2 \times 10^{-3} \frac{86400}{2\pi}} \ .$$

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

Land Surface Processes:

Surface Type The fizhi package surface Types are designated using the Koster-Suarez (*Koster and Suarez* [1991, 1992]) 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 6.16. The surface types and the percent of the grid cell occupied by any surface type were derived from the surface classification of *Defries and Townshend* [1994], and information about the location of permanent ice was obtained from the classifications of *Dorman and Sellers* [1989]. The surface type map for a 1° grid is shown in Figure 6.10. 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.

Surface Roughness The surface roughness length over oceans is computed iteratively with the wind stress by the surface layer parameterization (*Helfand and Schubert* [1995]). It employs an interpolation between the functions of *Large and Pond* [1981] for high winds and of *Kondo* [1975] for weak winds.

Surface Type Designation

Type	Vegetation Designation
1	Broadleaf Evergreen Trees
2	Broadleaf Deciduous Trees
3	Needleleaf Trees
4	Ground Cover
5	Broadleaf Shrubs
6	Dwarf Trees (Tundra)
7	Bare Soil
8	Desert (Bright)
9	Glacier
10	Desert (Dark)
100	Ocean

Table 6.16: Surface type designations.

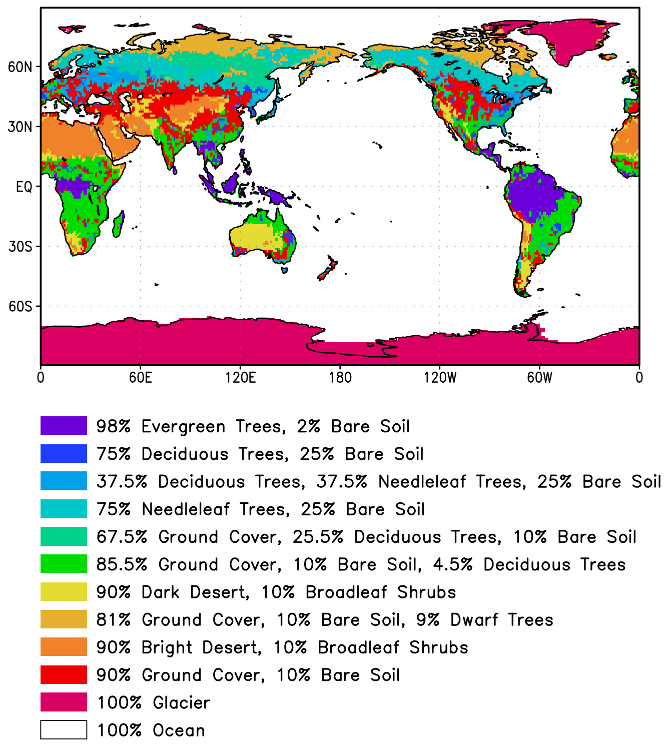


Figure 6.10: Surface Type Combinations.

Albedo The surface albedo computation, described in *Koster and Suarez* [1991], employs the "two stream" approximation used in Sellers' (1987) Simple Biosphere (SiB) Model which distinguishes between the direct and diffuse albedos in the visible and in the near infra-red spectral ranges. The albedos are functions of the observed leaf area index (a description of the relative orientation of the leaves to the sun), the greenness fraction, the vegetation type, and the solar zenith angle. Modifications are made to account for the presence of snow, and its depth relative to the height of the vegetation elements.

Gravity Wave Drag The fizhi package employs the gravity wave drag scheme of *Zhou et al.* [1995]). This scheme is a modified version of Vernekar et al. (1992), which was based on Alpert et al. (1988) and Helfand et al. (1987). In this version, the gravity wave stress at the surface is based on that derived by Pierrehumbert (1986) and is given by:

$$|\vec{\tau}_{sfc}| = \frac{\rho U^3}{N \ell^*} \left(\frac{F_r^2}{1+F_r^2} \right) \ , \tag{6.38}$$

where $F_r = Nh/U$ is the Froude number, N is the *Brunt - Väisälä* frequency, U is the surface wind speed, h is the standard deviation of the sub-grid scale orography, and ℓ^* is the wavelength of the monochromatic gravity wave in the direction of the low-level wind. A modification introduced by Zhou et al. allows for the momentum flux to escape through the top of the model, although this effect is small for the current 70-level model. The subgrid scale standard deviation is defined by h, and is not allowed to exceed 400 m.

The effects of using this scheme within a GCM are shown in *Takacs and Suarez* [1996]. Experiments using the gravity wave drag parameterization yielded significant and beneficial impacts on both the time-mean flow and the transient statistics of the a GCM climatology, and have eliminated most of the worst dynamically driven biases in the a GCM simulation. An examination of the angular momentum budget during climate runs indicates that the resulting gravity wave torque is similar to the data-driven torque produced by a data assimilation which was performed without gravity wave drag. It was shown that the inclusion of gravity wave drag results in large changes in both the mean flow and in eddy fluxes. The result is a more accurate simulation of surface stress (through a reduction in the surface wind strength), of mountain torque (through a redistribution of mean sea-level pressure), and of momentum convergence (through a reduction in the flux of westerly momentum by transient flow eddies).

Boundary Conditions and other Input Data:

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

Boundary condition data sets are available at the model's resolutions for either climatological or yearly varying conditions. Any frequency of boundary condition data can be used in the fizhi package; however, the current selection of data is summarized in Table 6.17. The time mean values are interpolated during each model timestep to the current time.

Fizhi Input Datasets

Variable	Frequency	Years
Sea Ice Extent	monthly	1979-current, climatology
Sea Ice Extent	weekly	1982-current, climatology
Sea Surface Temperature	monthly	1979-current, climatology
Sea Surface Temperature	weekly	1982-current, climatology
Zonally Averaged Upper-Level Moisture	monthly	climatology
Zonally Averaged Ozone Concentration	monthly	climatology

Table 6.17: Boundary conditions and other input data used in the fizhi package. Also noted are the current years and frequencies available.

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.

6.5.3.3 Fizhi Diagnostics

Fizhi Diagnostic Menu:

NAME	UNITS	LEVELS	DESCRIPTION
UFLUX	$Newton/m^2$	1	Surface U-Wind Stress on the atmosphere
VFLUX	$Newton/m^2$	1	Surface V-Wind Stress on the atmosphere
HFLUX	$Watts/m^2$	1	Surface Flux of Sensible Heat
EFLUX	$Watts/m^2$	1	Surface Flux of Latent Heat
QICE	$Watts/m^2$	1	Heat Conduction through Sea-Ice
RADLWG	$Watts/m^2$	1	Net upward LW flux at the ground
RADSWG	$Watts/m^2$	1	Net downward SW flux at the ground
RI	$dimensionless$	Nrphys	Richardson Number
CT	$dimensionless$	1	Surface Drag coefficient for T and Q
CU	$dimensionless$	1	Surface Drag coefficient for U and V
ET	m^2/sec	Nrphys	Diffusivity coefficient for T and Q
EU	m^2/sec	Nrphys	Diffusivity coefficient for U and V
TURBU	$m/sec/day$	Nrphys	U-Momentum Changes due to Turbulence
TURBV	$m/sec/day$	Nrphys	V-Momentum Changes due to Turbulence
TURBT	deg/day	Nrphys	Temperature Changes due to Turbulence
TURBQ	$g/kg/day$	Nrphys	Specific Humidity Changes due to Turbulence
MOISTT	deg/day	Nrphys	Temperature Changes due to Moist Processes
MOISTQ	$g/kg/day$	Nrphys	Specific Humidity Changes due to Moist Processes
RADLW	deg/day	Nrphys	Net Longwave heating rate for each level
RADSW	deg/day	Nrphys	Net Shortwave heating rate for each level
PREACC	mm/day	1	Total Precipitation
PRECON	mm/day	1	Convective Precipitation
TUFLUX	$Newton/m^2$	Nrphys	Turbulent Flux of U-Momentum
TVFLUX	$Newton/m^2$	Nrphys	Turbulent Flux of V-Momentum
TTFLUX	$Watts/m^2$	Nrphys	Turbulent Flux of Sensible Heat

NAME	UNITS	LEVELS	DESCRIPTION
TQFLUX	$Watts/m^2$	Nrphys	Turbulent Flux of Latent Heat
CN	$dimensionless$	1	Neutral Drag Coefficient
WINDS	m/sec	1	Surface Wind Speed
DTSRF	deg	1	Air/Surface virtual temperature difference
TG	deg	1	Ground temperature
TS	deg	1	Surface air temperature (Adiabatic from lowest model layer)
DTG	deg	1	Ground temperature adjustment
QG	g/kg	1	Ground specific humidity
QS	g/kg	1	Saturation surface specific humidity
TGRLW	deg	1	Instantaneous ground temperature used as input to the Longwave radiation subroutine
ST4	$Watts/m^2$	1	Upward Longwave flux at the ground (σT^4)
OLR	$Watts/m^2$	1	Net upward Longwave flux at the top of the model
OLRCLR	$Watts/m^2$	1	Net upward clearsky Longwave flux at the top of the model
LWGCLR	$Watts/m^2$	1	Net upward clearsky Longwave flux at the ground
LWCLR	deg/day	Nrphys	Net clearsky Longwave heating rate for each level
TLW	deg	Nrphys	Instantaneous temperature used as input to the Longwave radiation subroutine
SHLW	g/g	Nrphys	Instantaneous specific humidity used as input to the Longwave radiation subroutine
OZLW	g/g	Nrphys	Instantaneous ozone used as input to the Longwave radiation subroutine
CLMOLW	$0-1$	Nrphys	Maximum overlap cloud fraction used in the Longwave radiation subroutine
CLDTOT	$0-1$	Nrphys	Total cloud fraction used in the Longwave and Shortwave radiation subroutines
LWGDOWN	$Watts/m^2$	1	Downwelling Longwave radiation at the ground
GWDT	deg/day	Nrphys	Temperature tendency due to Gravity Wave Drag
RADSWT	$Watts/m^2$	1	Incident Shortwave radiation at the top of the atmosphere
TAUCLD	$per100mb$	Nrphys	Counted Cloud Optical Depth (non-dimensional) per 100 mb
TAUCLDC	$Number$	Nrphys	Cloud Optical Depth Counter

NAME	UNITS	LEVELS	DESCRIPTION
CLDLOW	$0-1$	Nrphys	Low-Level (1000-700 hPa) Cloud Fraction (0-1)
EVAP	mm/day	1	Surface evaporation
DPDT	hPa/day	1	Surface Pressure tendency
UAVE	m/sec	Nrphys	Average U-Wind
VAVE	m/sec	Nrphys	Average V-Wind
TAVE	deg	Nrphys	Average Temperature
QAVE	g/kg	Nrphys	Average Specific Humidity
OMEGA	hPa/day	Nrphys	Vertical Velocity
DUDT	$m/sec/day$	Nrphys	Total U-Wind tendency
DVDT	$m/sec/day$	Nrphys	Total V-Wind tendency
DTDT	deg/day	Nrphys	Total Temperature tendency
DQDT	$g/kg/day$	Nrphys	Total Specific Humidity tendency
VORT	$10^{-4}/sec$	Nrphys	Relative Vorticity
DTLS	deg/day	Nrphys	Temperature tendency due to Stratiform Cloud Formation
DQLS	$g/kg/day$	Nrphys	Specific Humidity tendency due to Stratiform Cloud Formation
USTAR	m/sec	1	Surface USTAR wind
Z0	m	1	Surface roughness
FRQTRB	$0-1$	Nrphys-1	Frequency of Turbulence
PBL	mb	1	Planetary Boundary Layer depth
SWCLR	deg/day	Nrphys	Net clearsky Shortwave heating rate for each level
OSR	$Watts/m^2$	1	Net downward Shortwave flux at the top of the model
OSRCLR	$Watts/m^2$	1	Net downward clearsky Shortwave flux at the top of the model
CLDMAS	kg/m^2	Nrphys	Convective cloud mass flux
UAVE	m/sec	Nrphys	Time-averaged $u-Wind$

NAME	UNITS	LEVELS	DESCRIPTION
VAVE	m/sec	Nrphys	Time-averaged $v - Wind$
TAVE	deg	Nrphys	Time-averaged $Temperature$
QAVE	g/g	Nrphys	Time-averaged $Specific\ Humidity$
RFT	deg/day	Nrphys	Temperature tendency due Rayleigh Friction
PS	mb	1	Surface Pressure
QQAVE	$(m/sec)^2$	Nrphys	Time-averaged $TurbulentKineticEnergy$
SWGCLR	$Watts/m^2$	1	Net downward clearsky Shortwave flux at the ground
PAVE	mb	1	Time-averaged Surface Pressure
DIABU	$m/sec/day$	Nrphys	Total Diabatic forcing on $u - Wind$
DIABV	$m/sec/day$	Nrphys	Total Diabatic forcing on $v - Wind$
DIABT	deg/day	Nrphys	Total Diabatic forcing on $Temperature$
DIABQ	$g/kg/day$	Nrphys	Total Diabatic forcing on $Specific\ Humidity$
RFU	$m/sec/day$	Nrphys	U-Wind tendency due to Rayleigh Friction
RFV	$m/sec/day$	Nrphys	V-Wind tendency due to Rayleigh Friction
GWDU	$m/sec/day$	Nrphys	U-Wind tendency due to Gravity Wave Drag
GWDU	$m/sec/day$	Nrphys	V-Wind tendency due to Gravity Wave Drag
GWDUS	N/m^2	1	U-Wind Gravity Wave Drag Stress at Surface
GWDVS	N/m^2	1	V-Wind Gravity Wave Drag Stress at Surface
GWDUT	N/m^2	1	U-Wind Gravity Wave Drag Stress at Top
GWDVT	N/m^2	1	V-Wind Gravity Wave Drag Stress at Top
LZRAD	mg/kg	Nrphys	Estimated Cloud Liquid Water used in Radiation

NAME	UNITS	LEVELS	DESCRIPTION
SLP	mb	1	Time-averaged Sea-level Pressure
CLDFRC	$0-1$	1	Total Cloud Fraction
TPW	gm/cm^2	1	Precipitable water
U2M	m/sec	1	U-Wind at 2 meters
V2M	m/sec	1	V-Wind at 2 meters
T2M	deg	1	Temperature at 2 meters
Q2M	g/kg	1	Specific Humidity at 2 meters
U10M	m/sec	1	U-Wind at 10 meters
V10M	m/sec	1	V-Wind at 10 meters
T10M	deg	1	Temperature at 10 meters
Q10M	g/kg	1	Specific Humidity at 10 meters
DTRAIN	kg/m^2	Nrphys	Detrainment Cloud Mass Flux
QFILL	$g/kg/day$	Nrphys	Filling of negative specific humidity

NAME	UNITS	LEVELS	DESCRIPTION
DTCONV	deg/sec	Nr	Temp Change due to Convection
DQCONV	$g/kg/sec$	Nr	Specific Humidity Change due to Convection
RELHUM	$percent$	Nr	Relative Humidity
PRECLS	$g/m^2/sec$	1	Large Scale Precipitation
ENPREC	J/g	1	Energy of Precipitation (snow, rain Temp)

Fizhi Diagnostic Description:

In this section we list and describe the diagnostic quantities available within the GCM. The diagnostics are listed in the order that they appear in the Diagnostic Menu, Section 6.5.3.3. In all cases, each diagnostic as currently archived on the output datasets is time-averaged over its diagnostic output frequency:

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

where $TTOT = \frac{\textbf{NQDIAG}}{\Delta t}$, **NQDIAG** is the output frequency of the diagnostic, and Δt is the timestep over which the diagnostic is updated.

UFLUX Surface Zonal Wind Stress on the Atmosphere ($Newton/m^2$)

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

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

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.

VFLUX Surface Meridional Wind Stress on the Atmosphere ($Newton/m^2$)

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

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

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.

HFLUX Surface Flux of Sensible Heat ($Watts/m^2$)

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:

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

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 θ is the potential temperature at the surface and at the bottom model level.

EFLUX Surface Flux of Latent Heat ($Watts/m^2$)

The turbulent flux of latent heat from the surface to the atmosphere is a function of the gradient of moisture, the potential evapotranspiration fraction and the eddy exchange coefficient:

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

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

QICE Heat Conduction Through Sea Ice ($Watts/m^2$)

Over sea ice there is an additional source of energy at the surface due to the heat conduction from the relatively warm ocean through the sea ice. The heat conduction through sea ice represents an additional energy source term for the ground temperature equation.

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

where C_{ti} is the thermal conductivity of ice, H_i is the ice thickness, assumed to be 3 m where sea ice is present, T_i is 273 degrees Kelvin, and T_g is the temperature of the sea ice.

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

RADLWG Net upward Longwave Flux at the surface ($Watts/m^2$)

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

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

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

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

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

RI Richardson Number (*dimensionless*)
The non-dimensional stability indicator is the ratio of the buoyancy to the shear:

$$\mathbf{RI} = \frac{\frac{g}{\theta_v}\frac{\partial\theta_v}{\partial z}}{(\frac{\partial u}{\partial z})^2 + (\frac{\partial v}{\partial z})^2} = \frac{c_p\frac{\partial\theta_v}{\partial z}\frac{\partial P^\kappa}{\partial z}}{(\frac{\partial u}{\partial z})^2 + (\frac{\partial v}{\partial z})^2}$$

where we used the hydrostatic equation:

$$\frac{\partial\Phi}{\partial P^\kappa} = c_p\theta_v$$

Negative values indicate unstable buoyancy **AND** shear, small positive values (< 0.4) indicate dominantly unstable shear, and large positive values indicate dominantly stable stratification.

CT Surface Exchange Coefficient for Temperature and Moisture (*dimensionless*)
The surface exchange coefficient is obtained from the similarity functions for the stability dependant flux profile relationships:

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

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

$$\psi_h = \int_{\zeta_0}^{\zeta}\frac{\phi_h}{\zeta}d\zeta \qquad and \qquad \psi_g = \frac{0.55(Pr^{2/3} - 0.2)}{\nu^{1/2}}(h_0u_* - h_{0_{ref}}u_{*_{ref}})^{1/2}$$

and: $h_0 = 30z_0$ with a maximum value over land of 0.01
ϕ_h is the similarity function of ζ, which expresses the stability dependance of the temperature and moisture gradients, specified differently for stable and unstable layers according to *Helfand and Schubert* [1995]. k is the Von Karman constant, ζ is the non-dimensional stability parameter, Pr is the Prandtl number for air, ν is the molecular viscosity, z_0 is the surface roughness length, u_* is the surface stress

velocity (see diagnostic number 67), and the subscript ref refers to a reference value.

<u>CU</u> Surface Exchange Coefficient for Momentum (*dimensionless*)
The surface exchange coefficient is obtained from the similarity functions for the stability dependant flux profile relationships:

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

where ψ_m is the surface layer non-dimensional wind shear:

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

ϕ_m is the similarity function of ζ, which expresses the stability dependance of the temperature and moisture gradients, specified differently for stable and unstable layers according to *Helfand and Schubert* [1995]. k is the Von Karman constant, ζ is the non-dimensional stability parameter, u_* is the surface stress velocity (see diagnostic number 67), and W_s is the magnitude of the surface layer wind.

<u>ET</u> 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 *Helfand and Labraga* [1988] adaptation of this closure, K_h takes the form:

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

where q is the turbulent velocity, or $\sqrt{2 * turbulent\ \ kinetic\ \ energy}$, q_e is the turbulence velocity derived from the more simple level 2.0 model, which describes equilibrium turbulence, ℓ is the master length scale related to the layer depth, S_H is a function of G_H and G_M, the dimensionless buoyancy and wind shear parameters, respectively, or a function of G_{H_e} and G_{M_e}, the equilibrium dimensionless buoyancy and wind shear parameters. Both G_H and G_M, and their equilibrium values G_{H_e} and G_{M_e}, are functions of the Richardson number.
For the detailed equations and derivations of the modified level 2.5 closure scheme, see *Helfand and Labraga* [1988].
In the surface layer, **ET** is the exchange coefficient for heat and moisture, in units of m/sec, given by:

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

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

<u>EU</u> 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 *Helfand and Labraga* [1988] adaptation of this closure, K_m takes the form:

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

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

For the detailed equations and derivations of the modified level 2.5 closure scheme, see *Helfand and Labraga* [1988].
In the surface layer, **EU** is the exchange coefficient for momentum, in units of m/sec, given by:

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

where C_u is the dimensionless exchange coefficient for momentum from the surface layer similarity functions (see diagnostic number 10), u_* is the surface friction velocity (see diagnostic number 67), C_D is the surface drag coefficient, and W_s is the magnitude of the surface layer wind.

TURBU Zonal U-Momentum changes due to Turbulence $(m/sec/day)$
The tendency of U-Momentum due to turbulence is written:

$$\mathbf{TURBU} = \frac{\partial u}{\partial t}_{turb} = \frac{\partial}{\partial z}(-\overline{u'w'}) = \frac{\partial}{\partial z}(K_m \frac{\partial u}{\partial z})$$

The Helfand and Labraga level 2.5 scheme models the turbulent flux of u-momentum in terms of K_m, and the equation has the form of a diffusion equation.
TURBV Meridional V-Momentum changes due to Turbulence $(m/sec/day)$
The tendency of V-Momentum due to turbulence is written:

$$\mathbf{TURBV} = \frac{\partial v}{\partial t}_{turb} = \frac{\partial}{\partial z}(-\overline{v'w'}) = \frac{\partial}{\partial z}(K_m \frac{\partial v}{\partial z})$$

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

TURBT Temperature changes due to Turbulence (deg/day)
The tendency of temperature due to turbulence is written:

$$\mathbf{TURBT} = \frac{\partial T}{\partial t} = P^{\kappa}\frac{\partial \theta}{\partial t}_{turb} = P^{\kappa}\frac{\partial}{\partial z}(-\overline{w'\theta'}) = P^{\kappa}\frac{\partial}{\partial z}(K_h \frac{\partial \theta_v}{\partial z})$$

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

TURBQ Specific Humidity changes due to Turbulence $(g/kg/day)$
The tendency of specific humidity due to turbulence is written:

$$\mathbf{TURBQ} = \frac{\partial q}{\partial t}_{turb} = \frac{\partial}{\partial z}(-\overline{w'q'}) = \frac{\partial}{\partial z}(K_h \frac{\partial q}{\partial z})$$

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

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

$$\mathbf{MOISTT} = \left.\frac{\partial T}{\partial t}\right|_c + \left.\frac{\partial T}{\partial t}\right|_{ls}$$

where:

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

and

$$\Gamma_s = g\eta\frac{\partial s}{\partial p}$$

The subscript c refers to convective processes, while the subscript ls refers to large scale precipitation processes, or supersaturation rain. The summation refers to contributions from each cloud type called by RAS. The dry static energy is given as s, the convective cloud base mass flux is given as m_B, and the cloud entrainment is given as η, which are explicitly defined in Section 6.5.3.2, the description of the convective parameterization. The fractional adjustment, or relaxation parameter, for each cloud type is given as α, while R is the rain re-evaporation adjustment.

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

$$\mathbf{MOISTQ} = \left.\frac{\partial q}{\partial t}\right|_c + \left.\frac{\partial q}{\partial t}\right|_{ls}$$

where:

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

and

$$\Gamma_s = g\eta\frac{\partial s}{\partial p} \quad and \quad \Gamma_h = g\eta\frac{\partial h}{\partial p}$$

The subscript c refers to convective processes, while the subscript ls refers to large scale precipitation processes, or supersaturation rain. The summation refers to contributions from each cloud type called by RAS. The dry static energy is given as s, the moist static energy is given as h, the convective cloud base mass flux is given as m_B, and the cloud entrainment is given as η, which are explicitly defined in Section 6.5.3.2, the description of the convective parameterization. The fractional adjustment, or relaxation parameter, for each cloud type is given as α, while R is the rain re-evaporation adjustment.

RADLW Heating Rate due to Longwave Radiation (deg/day)
The net longwave heating rate is calculated as the vertical divergence of the net terrestrial radiative fluxes. Both the clear-sky and cloudy-sky longwave fluxes are computed within the longwave routine. The subroutine calculates the clear-sky flux, $F_{LW}^{clearsky}$, first. For a given cloud fraction, the clear line-of-sight probability $C(p,p')$ is computed from the current level pressure p to the model top pressure, $p' = p_{top}$, and the model surface pressure, $p' = p_{surf}$, for the upward and downward radiative fluxes. (see Section 6.5.3.2). The cloudy-sky flux is then obtained as:

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

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

$$\frac{\partial \rho c_p T}{\partial t} = -\frac{\partial}{\partial z} F_{LW}^{NET},$$

or

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

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

$$F_{LW}^{NET} = F_{LW}^{\uparrow} - F_{LW}^{\downarrow}$$

RADSW Heating Rate due to Shortwave Radiation (deg/day)
The net Shortwave heating rate is calculated as the vertical divergence of the net solar radiative fluxes. The clear-sky and cloudy-sky shortwave fluxes are calculated separately. For the clear-sky case, the shortwave fluxes and heating rates are computed with both CLMO (maximum overlap cloud fraction) and CLRO (random overlap cloud fraction) set to zero (see Section 6.5.3.2). The shortwave routine is then called a second time, for the cloudy-sky case, with the true time-averaged cloud fractions CLMO and CLRO being used. In all cases, a normalized incident shortwave flux is used as input at the top of the atmosphere.
The heating rate due to Shortwave Radiation under cloudy skies is defined as:

$$\frac{\partial \rho c_p T}{\partial t} = -\frac{\partial}{\partial z} F(cloudy)_{SW}^{NET} \cdot \mathrm{RADSWT},$$

or

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

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

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

PREACC Total (Large-scale + Convective) Accumulated Precipition (mm/day)
For a change in specific humidity due to moist processes, Δq_{moist}, the vertical integral or total precipitable amount is given by:

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

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

PRECON Convective Precipition (mm/day)
For a change in specific humidity due to sub-grid scale cumulus convective processes, Δq_{cum}, the vertical integral or total precipitable amount is given by:

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

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

TUFLUX Turbulent Flux of U-Momentum $(Newton/m^2)$
The turbulent flux of u-momentum is calculated for *diagnostic purposes only* from the eddy coefficient for momentum:

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

where ρ is the air density, and K_m is the eddy coefficient.

TVFLUX Turbulent Flux of V-Momentum $(Newton/m^2)$
The turbulent flux of v-momentum is calculated for *diagnostic purposes only* from the eddy coefficient for momentum:

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

where ρ is the air density, and K_m is the eddy coefficient.

TTFLUX Turbulent Flux of Sensible Heat $(Watts/m^2)$
The turbulent flux of sensible heat is calculated for *diagnostic purposes only* from the eddy coefficient for heat and moisture:

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

where ρ is the air density, and K_h is the eddy coefficient.

TQFLUX Turbulent Flux of Latent Heat $(Watts/m^2)$
The turbulent flux of latent heat is calculated for *diagnostic purposes only* from the eddy coefficient for heat and moisture:

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

where ρ is the air density, and K_h is the eddy coefficient.

CN Neutral Drag Coefficient (*dimensionless*)
The drag coefficient for momentum obtained by assuming a neutrally stable surface layer:

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

where k is the Von Karman constant, h is the height of the surface layer, and z_0 is the surface roughness. NOTE: CN is not available through model version 5.3, but is available in subsequent versions.

WINDS Surface Wind Speed (*meter/sec*)
The surface wind speed is calculated for the last internal turbulence time step:

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

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

DTSRF Air/Surface Virtual Temperature Difference (*deg K*)
The air/surface virtual temperature difference measures the stability of the surface layer:

$$\mathbf{DTSRF} = (\theta_{v Nrphys+1} - \theta v_{Nrphys}) P^{\kappa}_{surf}$$

where

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

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

TG Ground Temperature (*deg K*)
The ground temperature equation is solved as part of the turbulence package using a backward implicit time differencing scheme:

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

where R_{sw} is the net surface downward shortwave radiative flux, R_{lw} is the net surface upward longwave radiative flux, Q_{ice} is the heat conduction through sea ice, H is the upward sensible heat flux, LE is the upward latent heat flux, and C_g is the total heat capacity of the ground. C_g is obtained by solving a heat diffusion equation for the penetration of the diurnal cycle into the ground (*Blackadar* [1977]), and is given by:

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

Here, the thermal conductivity, λ, is equal to $2x10^{-3} \ \frac{ly}{sec} \frac{cm}{K}$, the angular velocity of the earth, ω, is written as 86400 sec/day divided by 2π $radians/day$, and the expression for C_s, the heat capacity per unit volume at the surface, is a function of the ground wetness, W.

TS Surface Temperature (*deg K*)
The surface temperature estimate is made by assuming that the model's lowest layer is well-mixed, and therefore that θ is constant in that layer. The surface temperature is therefore:

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

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

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

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

QG Ground Specific Humidity (g/kg)
The ground specific humidity is obtained by interpolating between the specific humidity at the lowest model level and the specific humidity of a saturated ground. The interpolation is performed using the potential evapotranspiration function:

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

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

QS Saturation Surface Specific Humidity (g/kg)
The surface saturation specific humidity is the saturation specific humidity at the ground temprature and surface pressure:

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

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

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

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

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

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

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

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

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

where top indicates the top of the first model layer. In the GCM, $p_{top} = 0.0$ mb.

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

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

where top indicates the top of the first model layer. In the GCM, $p_{top} = 0.0$ mb.

LWGCLR Net upward clearsky Longwave flux at the surface $(Watts/m^2)$

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

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

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

The net longwave heating rate is calculated as the vertical divergence of the net terrestrial radiative fluxes. Both the clear-sky and cloudy-sky longwave fluxes are computed within the longwave routine. The subroutine calculates the clear-sky flux, $F^{clearsky}_{LW}$, first. For a given cloud fraction, the clear line-of-sight probability $C(p,p')$ is computed from the current level pressure p to the model top pressure, $p' = p_{top}$, and the model surface pressure, $p' = p_{surf}$, for the upward and downward radiative fluxes. (see Section 6.5.3.2). The cloudy-sky flux is then obtained as:

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

Thus, **LWCLR** is defined as the net longwave heating rate due to the vertical divergence of the clear-sky longwave radiative flux:

$$\frac{\partial \rho c_p T}{\partial t}_{clearsky} = -\frac{\partial}{\partial z} F(clearsky)^{NET}_{LW},$$

or

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

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

$$F(clearsky)^{Net}_{LW} = F(clearsky)^{\uparrow}_{LW} - F(clearsky)^{\downarrow}_{LW}$$

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

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

where T is the model temperature at the current time step n.

SHLW Instantaneous specific humidity used as input to the Longwave radiation subroutine (kg/kg)

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

where q is the model specific humidity at the current time step n.

OZLW Instantaneous ozone used as input to the Longwave radiation subroutine (kg/kg)

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

where OZ is the interpolated ozone data set from the climatological monthly mean zonally averaged ozone data set.

CLMOLW Maximum Overlap cloud fraction used in LW Radiation $(0-1)$
CLMOLW is the time-averaged maximum overlap cloud fraction that has been filled by the Relaxed Arakawa/Schubert Convection scheme and will be used in the Longwave Radiation algorithm. These are convective clouds whose radiative characteristics are assumed to be correlated in the vertical. For a complete description of cloud/radiative interactions, see Section 6.5.3.2.

$$\mathbf{CLMOLW} = CLMO_{RAS,LW}(\lambda, \phi, level)$$

CLDTOT Total cloud fraction used in LW and SW Radiation $(0-1)$
CLDTOT is the time-averaged total cloud fraction that has been filled by the Relaxed Arakawa/Schubert and Large-scale Convection schemes and will be used in the Longwave and Shortwave Radiation packages. For a complete description of cloud/radiative interactions, see Section 6.5.3.2.

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

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

CLMOSW Maximum Overlap cloud fraction used in SW Radiation $(0-1)$
CLMOSW is the time-averaged maximum overlap cloud fraction that has been filled by the Relaxed Arakawa/Schubert Convection scheme and will be used in the Shortwave Radiation algorithm. These are convective clouds whose radiative characteristics are assumed to be correlated in the vertical. For a complete description of cloud/radiative interactions, see Section 6.5.3.2.

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

CLROSW Random Overlap cloud fraction used in SW Radiation $(0-1)$
CLROSW is the time-averaged random overlap cloud fraction that has been filled by the Relaxed Arakawa/Schubert and Large-scale Convection schemes and will be used in the Shortwave Radiation algorithm. These are convective and large-scale clouds whose radiative characteristics are not assumed to be correlated in the vertical. For a complete description of cloud/radiative interactions, see Section 6.5.3.2.

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

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

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

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

EVAP Surface Evaporation (mm/day)
The surface evaporation is a function of the gradient of moisture, the potential evapotranspiration fraction and the eddy exchange coefficient:

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

where $\rho =$ the atmospheric density at the surface, β is the fraction of the potential evapotranspiration actually evaporated ($\beta = 1$ over oceans), K_h is the turbulent eddy exchange coefficient for heat and moisture at the surface in m/sec and $qsurface$ and q_{Nrphys} are the specific humidity at the surface (see diagnostic number 34) and at the bottom model level, respectively.

DUDT Total Zonal U-Wind Tendency $(m/sec/day)$
DUDT is the total time-tendency of the Zonal U-Wind due to Hydrodynamic, Diabatic, and Analysis forcing.

$$\mathbf{DUDT} = \frac{\partial u}{\partial t}_{Dynamics} + \frac{\partial u}{\partial t}_{Moist} + \frac{\partial u}{\partial t}_{Turbulence} + \frac{\partial u}{\partial t}_{Analysis}$$

DVDT Total Zonal V-Wind Tendency $(m/sec/day)$
DVDT is the total time-tendency of the Meridional V-Wind due to Hydrodynamic, Diabatic, and Analysis forcing.

$$\mathbf{DVDT} = \frac{\partial v}{\partial t}_{Dynamics} + \frac{\partial v}{\partial t}_{Moist} + \frac{\partial v}{\partial t}_{Turbulence} + \frac{\partial v}{\partial t}_{Analysis}$$

DTDT Total Temperature Tendency (deg/day)

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

$$\begin{aligned}\mathbf{DTDT} &= \frac{\partial T}{\partial t}_{Dynamics} + \frac{\partial T}{\partial t}_{MoistProcesses} + \frac{\partial T}{\partial t}_{ShortwaveRadiation} \\ &+ \frac{\partial T}{\partial t}_{LongwaveRadiation} + \frac{\partial T}{\partial t}_{Turbulence} + \frac{\partial T}{\partial t}_{Analysis}\end{aligned}$$

DQDT Total Specific Humidity Tendency $(g/kg/day)$
DQDT is the total time-tendency of Specific Humidity due to Hydrodynamic, Diabatic, and Analysis forcing.

$$\mathbf{DQDT} = \frac{\partial q}{\partial t}_{Dynamics} + \frac{\partial q}{\partial t}_{MoistProcesses} + \frac{\partial q}{\partial t}_{Turbulence} + \frac{\partial q}{\partial t}_{Analysis}$$

USTAR Surface-Stress Velocity (m/sec)
The surface stress velocity, or the friction velocity, is the wind speed at the surface layer top impeded by the surface drag:

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

C_u is the non-dimensional surface drag coefficient (see diagnostic number 10), and W_s is the surface wind speed (see diagnostic number 28).
Z0 Surface Roughness Length (m)
Over the land surface, the surface roughness length is interpolated to the local time from the monthly mean data of *Dorman and Sellers* [1989]. Over the ocean, the roughness length is a function of the surface-stress velocity, u_*.

$$\mathbf{Z0} = c_1 u_*^3 + c_2 u_*^2 + c_3 u_* + c_4 + c_5 u_*$$

where the constants are chosen to interpolate between the reciprocal relation of *Kondo* [1975] for weak winds, and the piecewise linear relation of *Large and Pond* [1981] for moderate to large winds.

FRQTRB Frequency of Turbulence $(0-1)$
The fraction of time when turbulence is present is defined as the fraction of time when the turbulent kinetic energy exceeds some minimum value, defined here to be 0.005 m^2/sec^2. When this criterion is met, a counter is incremented. The fraction over the averaging interval is reported.

PBL Planetary Boundary Layer Depth (mb)
The depth of the PBL is defined by the turbulence parameterization to be the depth at which the turbulent kinetic energy reduces to ten percent of its surface value.

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

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

SWCLR Clear sky Heating Rate due to Shortwave Radiation (deg/day)
The net Shortwave heating rate is calculated as the vertical divergence of the net solar radiative fluxes. The clear-sky and cloudy-sky shortwave fluxes are calculated separately. For the clear-sky case, the shortwave fluxes and heating rates are computed with both CLMO (maximum overlap cloud fraction) and CLRO (random overlap cloud fraction) set to zero (see Section 6.5.3.2). The shortwave routine is then called a second time, for the cloudy-sky case, with the true time-averaged cloud fractions CLMO and CLRO being used. In all cases, a normalized incident shortwave flux is used as input at the top of the atmosphere.
The heating rate due to Shortwave Radiation under clear skies is defined as:

$$\frac{\partial \rho c_p T}{\partial t} = -\frac{\partial}{\partial z} F(clear)_{SW}^{NET} \cdot \mathrm{RADSWT},$$

or

$$\textbf{SWCLR} = \frac{g}{c_p}\frac{\partial}{\partial p}F(clear)^{NET}_{SW} \cdot \text{RADSWT}.$$

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

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

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

$$\textbf{OSR} = F^{NET}_{SW,top}$$

where top indicates the top of the first model layer used in the shortwave radiation routine. In the GCM, $p_{SW_{top}} = 0$ mb.

OSRCLR Net upward clearsky Shortwave flux at the top of the model ($Watts/m^2$)

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

where top indicates the top of the first model layer used in the shortwave radiation routine. In the GCM, $p_{SW_{top}} = 0$ mb.

CLDMAS Convective Cloud Mass Flux (kg/m^2)
The amount of cloud mass moved per RAS timestep from all convective clouds is written:

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

where η is the entrainment, normalized by the cloud base mass flux, and m_B is the cloud base mass flux. m_B and η are defined explicitly in Section 6.5.3.2, 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.

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

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.

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

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.

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

QAVE Time-Averaged Specific Humidity (g/kg)

The diagnostic **QAVE** is simply the time-averaged Specific Humidity over the **NQAVE** output frequency. This is contrasted to the instantaneous Specific Humidity which is archived on the Prognostic Output data stream.

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

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

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

QQAVE Time-Averaged Turbulent Kinetic Energy $(m/sec)^2$
The diagnostic **QQAVE** is simply the time-averaged prognostic Turbulent Kinetic Energy produced by the GCM Turbulence parameterization over the **NQQAVE** output frequency. This is contrasted to the instantaneous Turbulent Kinetic Energy which is archived on the Prognostic Output data stream.

$$\mathbf{QQAVE} = qq(\lambda, \phi, level, t)$$

Note, **QQAVE** is computed and stored at the "mass-point" locations on the staggered C-grid.

SWGCLR Net downward clearsky Shortwave flux at the surface $(Watts/m^2)$

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

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

DIABU Total Diabatic Zonal U-Wind Tendency $(m/sec/day)$
DIABU is the total time-tendency of the Zonal U-Wind due to Diabatic processes and the Analysis forcing.

$$\mathbf{DIABU} = \frac{\partial u}{\partial t}_{Moist} + \frac{\partial u}{\partial t}_{Turbulence} + \frac{\partial u}{\partial t}_{Analysis}$$

DIABV Total Diabatic Meridional V-Wind Tendency $(m/sec/day)$
DIABV is the total time-tendency of the Meridional V-Wind due to Diabatic processes and the Analysis forcing.

$$\mathbf{DIABV} = \frac{\partial v}{\partial t}_{Moist} + \frac{\partial v}{\partial t}_{Turbulence} + \frac{\partial v}{\partial t}_{Analysis}$$

DIABT Total Diabatic Temperature Tendency (deg/day)
DIABT is the total time-tendency of Temperature due to Diabatic processes and the Analysis forcing.

$$\begin{aligned} \mathbf{DIABT} &= \frac{\partial T}{\partial t}_{MoistProcesses} + \frac{\partial T}{\partial t}_{ShortwaveRadiation} \\ &+ \frac{\partial T}{\partial t}_{LongwaveRadiation} + \frac{\partial T}{\partial t}_{Turbulence} + \frac{\partial T}{\partial t}_{Analysis} \end{aligned}$$

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

$$\begin{aligned}\frac{\partial T}{\partial t}_{Diabatic} &= \frac{\partial T}{\partial t}_{MoistProcesses} + \frac{\partial T}{\partial t}_{ShortwaveRadiation} \\ &+ \frac{\partial T}{\partial t}_{LongwaveRadiation} + \frac{\partial T}{\partial t}_{Turbulence}\end{aligned}$$

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

$$\frac{\partial T}{\partial t}_{Diabatic} = \frac{p^\kappa}{\pi}\frac{\partial \pi\theta}{\partial t}_{Diabatic}$$

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

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

DIABQ Total Diabatic Specific Humidity Tendency $(g/kg/day)$
DIABQ is the total time-tendency of Specific Humidity due to Diabatic processes and the Analysis forcing.

$$\mathbf{DIABQ} = \frac{\partial q}{\partial t}_{MoistProcesses} + \frac{\partial q}{\partial t}_{Turbulence} + \frac{\partial q}{\partial t}_{Analysis}$$

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

$$\frac{\partial q}{\partial t}_{Diabatic} = \frac{\partial q}{\partial t}_{MoistProcesses} + \frac{\partial q}{\partial t}_{Turbulence}$$

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

$$\frac{\partial q}{\partial t}_{Diabatic} = \frac{1}{\pi}\frac{\partial \pi q}{\partial t}_{Diabatic}$$

Thus, **DIABQ** may be written as

$$\mathbf{DIABQ} = \frac{1}{\pi}\left(\frac{\partial \pi q}{\partial t}_{Diabatic} + \frac{\partial \pi q}{\partial t}_{Analysis}\right)$$

VINTUQ Vertically Integrated Moisture Flux $(m/sec \cdot g/kg)$
The vertically integrated moisture flux due to the zonal u-wind is obtained by integrating uq over the depth of the atmosphere at each model timestep, and dividing by the total mass of the column.

$$\mathbf{VINTUQ} = \frac{\int_{surf}^{top} uq\rho dz}{\int_{surf}^{top} \rho dz}$$

Using $\rho\delta z = -\frac{\delta p}{g} = -\frac{1}{g}\delta p$, we have

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

VINTVQ Vertically Integrated Moisture Flux $(m/sec \cdot g/kg)$
The vertically integrated moisture flux due to the meridional v-wind is obtained by integrating vq over the depth of the atmosphere at each model timestep, and dividing by the total mass of the column.

$$\mathbf{VINTVQ} = \frac{\int_{surf}^{top} vq\rho dz}{\int_{surf}^{top} \rho dz}$$

Using $\rho\delta z = -\frac{\delta p}{g} = -\frac{1}{g}\delta p$, we have

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

VINTUT Vertically Integrated Heat Flux $(m/sec \cdot deg)$
The vertically integrated heat flux due to the zonal u-wind is obtained by integrating uT over the depth of the atmosphere at each model timestep, and dividing by the total mass of the column.

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

Or,

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

VINTVT Vertically Integrated Heat Flux $(m/sec \cdot deg)$
The vertically integrated heat flux due to the meridional v-wind is obtained by integrating vT over the depth of the atmosphere at each model timestep, and dividing by the total mass of the column.

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

Using $\rho\delta z = -\frac{\delta p}{g}$, we have

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

CLDFRC Total 2-Dimensional Cloud Fracton $(0-1)$
If we define the time-averaged random and maximum overlapped cloudiness as CLRO and CLMO respectively, then the probability of clear sky associated with random overlapped clouds at any level is (1-CLRO) while the probability of clear sky associated with maximum overlapped clouds at any level is (1-CLMO). The total clear sky probability is given by (1-CLRO)*(1-CLMO), thus the total cloud fraction at each level may be obtained by 1-(1-CLRO)*(1-CLMO).

At any given level, we may define the clear line-of-site probability by appropriately accounting for the maximum and random overlap cloudiness. The clear line-of-site probability is defined to be equal to the product of the clear line-of-site probabilities associated with random and maximum overlap cloudiness. The clear line-of-site probability $C(p,p')$ associated with maximum overlap clouds, from the current pressure p to the model top pressure, $p' = p_{top}$, or the model surface pressure, $p' = p_{surf}$, is simply 1.0 minus the largest maximum overlap cloud value along the line-of-site, ie.

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

Thus, even in the time-averaged sense it is assumed that the maximum overlap clouds are correlated in the vertical. The clear line-of-site probability associated with random overlap clouds is defined to be the product of the clear sky probabilities at each level along the line-of-site, ie.

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

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

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

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

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

For a complete description of cloud/radiative interactions, see Section 6.5.3.2.
QINT Total Precipitable Water (gm/cm^2)
The Total Precipitable Water is defined as the vertical integral of the specific humidity, given by:

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

where we have used the hydrostatic relation $\rho \delta z = -\frac{\delta p}{g}$.

U2M Zonal U-Wind at 2 Meter Depth (m/sec)
The u-wind at the 2-meter depth is determined from the similarity theory:

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

where $\psi_m(2m)$ is the non-dimensional wind shear at two meters, and the subscript sl refers to the height of the top of the surface layer. If the roughness height is above two meters, **U2M** is undefined.

V2M Meridional V-Wind at 2 Meter Depth (m/sec)
The v-wind at the 2-meter depth is a determined from the similarity theory:

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

where $\psi_m(2m)$ is the non-dimensional wind shear at two meters, and the subscript sl refers to the height of the top of the surface layer. If the roughness height is above two meters, **V2M** is undefined.

T2M Temperature at 2 Meter Depth ($deg\ K$)
The temperature at the 2-meter depth is a determined from the similarity theory:

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

where:

$$\theta_* = -\frac{(\overline{w'\theta'})}{u_*}$$

where $\psi_h(2m)$ is the non-dimensional temperature gradient at two meters, ψ_g is the non-dimensional temperature gradient in the viscous sublayer, and the subscript sl refers to the height of the top of the surface layer. If the roughness height is above two meters, **T2M** is undefined.

Q2M Specific Humidity at 2 Meter Depth (g/kg)
The specific humidity at the 2-meter depth is determined from the similarity theory:

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

where:

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

where $\psi_h(2m)$ is the non-dimensional temperature gradient at two meters, ψ_g is the non-dimensional temperature gradient in the viscous sublayer, and the subscript sl refers to the height of the top of the surface layer. If the roughness height is above two meters, **Q2M** is undefined.

U10M Zonal U-Wind at 10 Meter Depth (m/sec)
The u-wind at the 10-meter depth is an interpolation between the surface wind and the model lowest level wind using the ratio of the non-dimensional wind shear at the two levels:

$$\mathbf{U10M} = \frac{u_*}{k}\psi_{m_{10m}}\frac{u_{sl}}{W_s} = \frac{\psi_{m_{10m}}}{\psi_{m_{sl}}}u_{sl}$$

where $\psi_m(10m)$ is the non-dimensional wind shear at ten meters, and the subscript *sl* refers to the height of the top of the surface layer.

V10M Meridional V-Wind at 10 Meter Depth (m/sec)
The v-wind at the 10-meter depth is an interpolation between the surface wind and the model lowest level wind using the ratio of the non-dimensional wind shear at the two levels:

$$\mathbf{V10M} = \frac{u_*}{k}\psi_{m_{10m}}\frac{v_{sl}}{W_s} = \frac{\psi_{m_{10m}}}{\psi_{m_{sl}}}v_{sl}$$

where $\psi_m(10m)$ is the non-dimensional wind shear at ten meters, and the subscript *sl* refers to the height of the top of the surface layer.

T10M Temperature at 10 Meter Depth $(deg\ K)$
The temperature at the 10-meter depth is an interpolation between the surface potential temperature and the model lowest level potential temperature using the ratio of the non-dimensional temperature gradient at the two levels:

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

where:

$$\theta_* = -\frac{(\overline{w'\theta'})}{u_*}$$

where $\psi_h(10m)$ is the non-dimensional temperature gradient at two meters, ψ_g is the non-dimensional temperature gradient in the viscous sublayer, and the subscript *sl* refers to the height of the top of the surface layer.

Q10M Specific Humidity at 10 Meter Depth (g/kg)
The specific humidity at the 10-meter depth is an interpolation between the surface specific humidity and the model lowest level specific humidity using the ratio of the non-dimensional temperature gradient at the two levels:

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

where:

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

where $\psi_h(10m)$ is the non-dimensional temperature gradient at two meters, ψ_g is the non-dimensional temperature gradient in the viscous sublayer, and the subscript *sl* refers to the height of the top of the surface layer.

DTRAIN Cloud Detrainment Mass Flux (kg/m^2)
The amount of cloud mass moved per RAS timestep at the cloud detrainment level is written:

$$\mathbf{DTRAIN} = \eta_{r_D} m_B$$

where r_D is the detrainment level, m_B is the cloud base mass flux, and η is the entrainment, defined in Section 6.5.3.2.

QFILL Filling of negative Specific Humidity $(g/kg/day)$
Due to computational errors associated with the numerical scheme used for the advection of moisture, negative values of specific humidity may be generated. The specific humidity is checked for negative values after every dynamics timestep. If negative values have been produced, a filling algorithm is invoked which redistributes moisture from below. Diagnostic **QFILL** is equal to the net filling needed to eliminate negative specific humidity, scaled to a per-day rate:

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

where

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

6.5.3.4 Key subroutines, parameters and files

6.5.3.5 Dos and donts

6.5.3.6 Fizhi Reference

6.5.3.7 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.

6.6 Sea Ice Packages

6.6.1 THSICE: The Thermodynamic Sea Ice Package

Important note: This document has been written by Stephanie Dutkiewicz and describes an earlier implementation of the sea-ice package. This needs to be updated to reflect the recent changes (JMC).
This thermodynamic ice model is based on the 3-layer model by Winton (2000). and the energy-conserving LANL CICE model (Bitz and Lipscomb, 1999). The model considers two equally thick ice layers; the upper layer has a variable specific heat resulting from brine pockets, the lower layer has a fixed heat capacity. A zero heat capacity snow layer lies above the ice. Heat fluxes at the top and bottom surfaces are used to calculate the change in ice and snow layer thickness. Grid cells of the ocean model are either fully covered in ice or are open water. There is a provision to parametrize ice fraction (and leads) in this package. Modifications are discussed in small font following the subroutine descriptions.

6.6.1.1 Key parameters and Routines

The ice model is called from *thermodynamics.F*, subroutine *ice_forcing.F* is called in place of *external_forcing_surf.F*.

subroutine ICE_FORCING
In *ice_forcing.F*, we calculate the freezing potential of the ocean model surface layer of water:

$$\mathbf{frzmlt} = (T_f - SST)\frac{c_{sw}\rho_{sw}\Delta z}{\Delta t}$$

where c_{sw} is seawater heat capacity, ρ_{sw} is the seawater density, Δz is the ocean model upper layer thickness and Δt is the model (tracer) timestep. The freezing temperature, $T_f = \mu S$ is a function of the salinity.

1) Provided there is no ice present and **frzmlt** is less than 0, the surface tendencies of wind, heat and freshwater are calculated as usual (ie. as in *external_forcing_surf.F*).

2) If there is ice present in the grid cell we call the main ice model routine *ice_therm.F* (see below). Output from this routine gives net heat and freshwater flux affecting the top of the ocean.

Subroutine *ice_forcing.F* uses these values to find the sea surface tendencies in grid cells. When there is ice present, the surface stress tendencies are set to zero; the ice model is purely thermodynamic and the effect of ice motion on the sea-surface is not examined.

Relaxation of surface T and S is only allowed equatorward of **relaxlat** (see **DATA.ICE below**), and no relaxation is allowed under the ice at any latitude.

(Note that there is provision for allowing grid cells to have both open water and seaice; if **compact** is between 0 and 1)

subroutine ICE_FREEZE

This routine is called from *thermodynamics.F* after the new temperature calculation, *calc_gt.F*, but before *calc_gs.F*. In *ice_freeze.F*, any ocean upper layer grid cell with no ice cover, but with temperature below freezing, $T_f = \mu S$ has ice initialized. We calculate **frzmlt** from all the grid cells in the water column that have a temperature less than freezing. In this routine, any water below the surface that is below freezing is set to T_f. A call to *ice_start.F* is made if **frzmlt** > 0, and salinity tendancy is updated for brine release.

(There is a provision for fractional ice: In the case where the grid cell has less ice coverage than **icemaskmax** we allow *ice_start.F* to be called).

subroutine ICE_START
The energy available from freezing the sea surface is brought into this routine as **esurp**. The enthalpy of the 2 layers of any new ice is calculated as:

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

where c_f is specific heat of liquid fresh water, c_i is the specific heat of fresh ice, L_i is latent heat of freezing, ρ_i is density of ice and T_{mlt} is melting temperature of ice with salinity of 1. The height of a new layer of ice is

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

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

The surface skin temperature T_s and ice temperatures T_1, T_2 and the sea surface temperature are set at T_f.

(There is provision for fractional ice: new ice is formed over open water; the first freezing in the cell must have a height of **himin0**; this determines the ice fraction **compact**. If there is already ice in the grid cell, the new ice must have the same height and the new ice fraction is

$$i_f = (1 - \hat{i_f})\frac{h_{inew}}{h_i}$$

where $\hat{i_f}$ is ice fraction from previous timestep and h_i is current ice height. Snow is redistributed over the new ice fraction. The ice fraction is not allowed to become larger than **iceMaskmax** and if the ice height is above **hihig** then freezing energy comes from the full grid cell, ice growth does not occur under orginal ice due to freezing water.

subroutine ICE_THERM

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

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

We follow the procedure of Winton (1999) – see equations 3 to 21 – to calculate the surface and internal ice temperatures. The surface temperature is found from the balance of the flux at the surface F_s, the shortwave heat flux absorbed by the ice, **fswint**, and the upward conduction of heat through the snow and/or ice, F_u. We linearize F_s about the surface temperature, $\hat{T_s}$, at the previous timestep (where $\hat{}$ indicates the value at the previous timestep):

$$F_s(T_s) = F_s(\hat{T_s}) + \frac{\partial F_s(\hat{T_s})}{\partial T_s}(T_s - \hat{T_s})$$

where,

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

and

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

F_s and $\frac{dF_s}{dT}$ are currently calculated from the **BULKF** package described separately, but could also be provided by an atmospheric model. The surface albedo is calculated from the ice height and/or surface temperature (see below, *srf_albedo.F*) and the shortwave flux absorbed in the ice is

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

where κ_i is bulk extinction coefficient.

The conductive flux to the surface is

$$F_u = K_{1/2}(T_1 - T_s)$$

where $K_{1/2}$ is the effective conductive coupling of the snow-ice layer between the surface and the mid-point of the upper layer of ice $K_{1/2} = \frac{4K_iK_s}{K_sh_i + 4K_ih_s}$. 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. For the upper layer (1) with brine pockets and the lower fresh layer (2):

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

where c_f is specific heat of liquid fresh water, c_i is the specific heat of fresh ice, and L_i is latent heat of melting fresh ice.

From the new ice temperatures, we can calculate the energy flux at the surface available for melting (if T_s=0) and the energy at the ocean-ice interface for either melting or freezing.

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

where F_b is the heat flux at the ice bottom due to the sea surface temperature variations from freezing. If T_{sst} is above freezing, $F_b = c_{sw}\rho_{sw}\gamma(T_{sst} - T_f)u^*$, γ is the heat transfer coefficient and $u^* = QQ$ is frictional velocity between ice and water. If T_{sst} is below freezing, $F_b = (T_f - T_{sst})c_f\rho_f\Delta z/\Delta t$ and set T_{sst} to T_f. We also include the energy from lower layers that drop below freezing, and set those layers to T_f.

If $E_{top} > 0$ we melt snow from the surface, if all the snow is melted and there is energy left, we melt the ice. If the ice is all gone and there is still energy left, we apply the left over energy to heating the ocean model upper layer (See Winton, 1999, equations 27-29). Similarly if $E_{bot} > 0$ we melt ice from the bottom. If all the ice is melted, the snow is melted (with energy from the ocean model upper layer if necessary). If $E_{bot} < 0$ we grow ice at the bottom

$$\Delta h_i = \frac{-E_{bot}}{(q_{bot}\rho_i)}$$

where $q_{bot} = -c_i T_f + L_i$ is the enthalpy of the new ice, The enthalpy of the second ice layer, q_2 needs to be modified:

$$q_2 = \frac{\hat{h_i}/2\hat{q_2} + \Delta h_i q_{bot}}{\hat{h_i}/2 + \Delta h_i}$$

If there is a ice layer and the overlying air temperature is below 0^oC then any precipitation, P joins the snow layer:

$$\Delta h_s = -P\frac{\rho_f}{\rho_s}\Delta t,$$

ρ_f and ρ_s are the fresh water and snow densities. Any evaporation, similarly, removes snow or ice from the surface. We also calculate the snow age here, in case it is needed for the surface albedo calculation (see *srf_albedo.F* below).

For practical reasons we limit the ice growth to **hilim** and snow is limited to **hslim**. We converts any ice and/or snow above these limits back to water, maintaining the salt balance. Note however, that heat is not conserved in this conversion; sea surface temperatures below the ice are not recalculated.

If the snow/ice interface is below the waterline, snow is converted to ice (see Winton, 1999, equations 35 and 36). The subroutine *new_layers_winton.F*, described below, repartitions the ice into equal thickness layers while conserving energy.

The subroutine *ice_therm.F* now calculates the heat and fresh water fluxes affecting the ocean model surface layer. The heat flux:

$$q_{net} = \mathbf{fswocn} - F_b - \frac{\mathbf{esurp}}{\Delta t}$$

is composed of the shortwave flux that has passed through the ice layer and is absorbed by the water, **fswocn**= QQ, the ocean flux to the ice F_b, and the surplus energy left over from the melting, **esurp**. The fresh water flux is determined from the amount of fresh water and salt in the ice/snow system before and after the timestep.

(There is a provision for fractional ice: If ice height is above **hihig** then all energy from freezing at sea surface is used only in the open water aparts of the cell (ie. F_b will only have the conduction term). The melt energy is partitioned by **frac_energy** between melting ice height and ice extent. However, once ice height drops below **himon0** then all energy melts ice extent.

subroutine SFC_ALBEDO

The routine *ice_therm.F* calls this routine to determine the surface albedo. There are two calculations provided here:
1) from LANL CICE model

$$\alpha = f_s\alpha_s + (1 - f_s)(\alpha_{i_{min}} + (\alpha_{i_{max}} - \alpha_{i_{min}})(1 - e^{-h_i/h_\alpha}))$$

where f_s is 1 if there is snow, 0 if not; the snow albedo, α_s has two values depending on whether $T_s < 0$ or not; $\alpha_{i_{min}}$ and $\alpha_{i_{max}}$ are ice albedos for thin melting ice, and thick bare ice respectively, and h_α is a scale height.
2) From GISS model (Hansen et al 1983)

$$\alpha = \alpha_i e^{-h_s/h_a} + \alpha_s(1 - e^{-h_s/h_a})$$

where α_i is a constant albedo for bare ice, h_a is a scale height and α_s is a variable snow albedo.

$$\alpha_s = \alpha_1 + \alpha_2 e^{-\lambda_a a_s}$$

where α_1 is a constant, α_2 depends on T_s, a_s is the snow age, and λ_a is a scale frequency. The snow age is calculated in *ice_therm.F* and is given in equation 41 in Hansen et al (1983).

subroutine NEW_LAYERS_WINTON

The subroutine *new_layers_winton.F* repartitions the ice into equal thickness layers while conserving energy. We pass to this subroutine, the ice layer enthalpies after melting/growth and the new height of the ice layers. The ending layer height should be half the sum of the new ice heights from *ice_therm.F*. The enthalpies of the ice layers are adjusted accordingly to maintain total energy in the ice model. If layer 2 height is greater than layer 1 height then layer 2 gives ice to layer 1 and:

$$q_1 = f_1\hat{q}_1 + (1 - f1)\hat{q}_2$$

where f_1 is the fraction of the new to old upper layer heights. T_1 will therefore also have changed. Similarly for when ice layer height 2 is less than layer 1 height, except here we need to to be careful that the new T_2 does not fall below the melting temperature.

Initializing subroutines

ice_init.F: Set ice variables to zero, or reads in pickup information from **pickup.ic** (which was written out in *checkpoint.F*)
ice_readparms.F: Reads **data.ice**

Diagnostic subroutines

ice_ave.F: Keeps track of means of the ice variables
ice_diags.F: Finds averages and writes out diagnostics

Common Blocks

ICE.h: Ice Varibles, also **relaxlat** and **startIceModel**
ICE_DIAGS.h: matrices for diagnostics: averages of fields from *ice_diags.F*
BULKF_ICE_CONSTANTS.h (in **BULKF** package): all the parameters need by the ice model

Input file DATA.ICE

Here we need to set **StartIceModel**: which is 1 if the model starts from no ice; and 0 if there is a pickup file with the ice matrices (**pickup.ic**) which is read in *ice_init.F* and written out in *checkpoint.F*. The parameter **relaxlat** defines the latitude poleward of which there is no relaxing of surface T or S to observations. This avoids the relaxation forcing the ice model at these high latitudes.

(Note: **hicemin** is set to 0 here. If the provision for allowing grid cells to have both open water and seaice is ever implemented, this would be greater than 0)

6.6.1.2 Important Notes

1) heat fluxes have different signs in the ocean and ice models.
2) StartIceModel must be changed in **data.ice**: 1 (if starting from no ice), 0 (if using pickup.ic file).

6.6.1.3 THSICE Diagnostics

```
------------------------------------------------------------------------
<-Name->|Levs|<-parsing code->|<--  Units   -->|<- Tile (max=80c)
------------------------------------------------------------------------
SI_Fract|  1 |SM P    M1      |0-1             |Sea-Ice fraction  [0-1]
SI_Thick|  1 |SM PC197M1      |m               |Sea-Ice thickness (area weighted average)
SI_SnowH|  1 |SM PC197M1      |m               |Snow thickness over Sea-Ice (area weighted)
SI_Tsrf |  1 |SM  C197M1      |degC            |Surface Temperature over Sea-Ice (area weighted)
SI_Tice1|  1 |SM  C197M1      |degC            |Sea-Ice Temperature, 1srt layer (area weighted)
SI_Tice2|  1 |SM  C197M1      |degC            |Sea-Ice Temperature, 2nd  layer (area weighted)
SI_Qice1|  1 |SM  C198M1      |J/kg            |Sea-Ice enthalpy, 1srt layer (mass weighted)
SI_Qice2|  1 |SM  C198M1      |J/kg            |Sea-Ice enthalpy, 2nd  layer (mass weighted)
SIalbedo|  1 |SM PC197M1      |0-1             |Sea-Ice Albedo [0-1] (area weighted average)
SIsnwAge|  1 |SM P    M1      |s               |snow age over Sea-Ice
SIsnwPrc|  1 |SM  C197M1      |kg/m^2/s        |snow precip. (+=dw) over Sea-Ice (area weighted)
SIflxAtm|  1 |SM      M1      |W/m^2           |net heat flux from the Atmosphere (+=dw)
SIfrwAtm|  1 |SM      M1      |kg/m^2/s        |fresh-water flux to the Atmosphere (+=up)
SIflx2oc|  1 |SM      M1      |W/m^2           |heat flux out of the ocean (+=up)
SIfrw2oc|  1 |SM      M1      |m/s             |fresh-water flux out of the ocean (+=up)
SIsaltFx|  1 |SM      M1      |psu.kg/m^2      |salt flux out of the ocean (+=up)
SItOcMxL|  1 |SM      M1      |degC            |ocean mixed layer temperature
SIsOcMxL|  1 |SM P    M1      |psu             |ocean mixed layer salinity
```

References

Bitz, C.M. and W.H. Lipscombe, 1999: An Energy-Conserving Thermodynamic Model of Sea Ice. *Journal of Geophysical Research*, 104, 15,669 – 15,677.

Hansen, J., G. Russell, D. Rind, P. Stone, A. Lacis, S. Lebedeff, R. Ruedy and L.Travis, 1983: Efficient Three-Dimensional Global Models for Climate Studies: Models I and II. *Monthly Weather Review*, 111, 609 – 662.

Hunke, E.C and W.H. Lipscomb, circa 2001: CICE: the Los Alamos Sea Ice Model Documentation and Software User's Manual. LACC-98-16v.2.
(note: this documentation is no longer available as CICE has progressed to a very different version 3)

Winton, M, 2000: A reformulated Three-layer Sea Ice Model. *Journal of Atmospheric and Ocean Technology*, 17, 525 – 531.

6.6.1.4 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.

6.6.2 SEAICE Package

Authors: Martin Losch, Dimitris Menemenlis, An Nguyen, Jean-Michel Campin, Patrick Heimbach, Chris Hill and Jinlun Zhang

6.6.2.1 Introduction

Package "seaice" provides a dynamic and thermodynamic interactive sea-ice model.

CPP options enable or disable different aspects of the package (Section 6.6.2.2). Run-Time options, flags, filenames and field-related dates/times are set in `data.seaice` (Section 6.6.2.3). A description of key subroutines is given in Section 6.6.2.5. Input fields, units and sign conventions are summarized in Section 6.6.2.3, and available diagnostics output is listed in Section 6.6.2.6.

6.6.2.2 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 3.4).

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 6.6.2.2 summarizes the most important ones. For more options see the default `pkg/seaice/SEAICE_OPTIONS.h`.

CPP option	Description
`SEAICE_DEBUG`	Enhance STDOUT for debugging
`SEAICE_ALLOW_DYNAMICS`	sea-ice dynamics code
`SEAICE_CGRID`	LSR solver on C-grid (rather than original B-grid)
`SEAICE_ALLOW_EVP`	enable use of EVP rheology solver
`SEAICE_ALLOW_JFNK`	enable use of JFNK rheology solver
`SEAICE_EXTERNAL_FLUXES`	use EXF-computed fluxes as starting point
`SEAICE_ZETA_SMOOTHREG`	use differentialable regularization for viscosities
`SEAICE_VARIABLE_FREEZING_POINT`	enable linear dependence of the freezing point on salinity (by default undefined)
`ALLOW_SEAICE_FLOODING`	enable snow to ice conversion for submerged sea-ice
`SEAICE_VARIABLE_SALINITY`	enable sea-ice with variable salinity (by default undefined)
`SEAICE_SITRACER`	enable sea-ice tracer package (by default undefined)
`SEAICE_BICE_STRESS`	B-grid only for backward compatiblity: turn on ice-stress on ocean
`EXPLICIT_SSH_SLOPE`	B-grid only for backward compatiblity: use ETAN for tilt computations rather than geostrophic velocities

Table 6.18: Some of the most relevant CPP preprocessor flags in the `seaice`-package.

6.6.2.3 Run-time parameters

Run-time parameters (see Table 6.19) 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 6.19 lists most run-time parameters.

Table 6.19: Run-time parameters and default values

Name	Default value	Description	Reference
SEAICEwriteState	T	write sea ice state to file	
SEAICEuseDYNAMICS	T	use dynamics	
SEAICEuseJFNK	F	use the JFNK-solver	
SEAICEuseTEM	F	use truncated ellipse method	
SEAICEuseStrImpCpl	F	use strength implicit coupling in LSR/JFNK	
SEAICEuseMetricTerms	T	use metric terms in dynamics	
SEAICEuseEVPpickup	T	use EVP pickups	
SEAICEuseFluxForm	F	use flux form for 2nd central difference advection scheme	
SEAICErestoreUnderIce	F	enable restoring to climatology under ice	
useHB87stressCoupling	F	turn on ice-ocean stress coupling following *Hibler and Bryan* [1987]	
usePW79thermodynamics	T	flag to turn off zero-layer-thermodynamics for testing	
SEAICEadvHeff/Area/Snow/Salt	T	flag to turn off advection of scalar state variables	
SEAICEuseFlooding	T	use flood-freeze algorithm	
SEAICE_no_slip	F	switch between free-slip and no-slip boundary conditions	
SEAICE_deltaTtherm	dTracerLev(1)	thermodynamic timestep	
SEAICE_deltaTdyn	dTracerLev(1)	dynamic timestep	
SEAICE_deltaTevp	0	EVP sub-cycling time step, values > 0 turn on EVP	
SEAICEuseEVPstar	F	use modified EVP* instead of EVP	
SEAICEuseEVPrev	F	use yet another variation on EVP*	
SEAICEnEVPstarSteps	UNSET	number of modified EVP* iteration	
SEAICE_evpAlpha	UNSET	EVP* parameter	
SEAICE_evpBeta	UNSET	EVP* parameter	
SEAICEaEVPcoeff	UNSET	aEVP parameter	
SEAICEaEVPcStar	4	aEVP parameter	[*Kimmritz et al.*, 2016]
SEAICEaEVPalphaMin	5	aEVP parameter	[*Kimmritz et al.*, 2016]
SEAICE_elasticParm	$\frac{1}{3}$	EVP paramter E_0	
SEAICE_evpTauRelax	$\Delta t_{EVP} E_0$	relaxation time scale T for EVP waves	
SEAICEnewtonIterMax	10	maximum number of JFNK-Newton iterations	
SEAICEkrylovIterMax	10	maximum number of JFNK-Krylov iterations	
SEAICE_JFNK_lsIter	(off)	start line search after "lsIter" Newton iterations	
JFNKgamma_nonlin	1.0E-05	non-linear tolerance parameter for JFNK solver	
JFNKgamma_lin_min/max	0.10/0.99	tolerance parameters for linear JFNK solver	
JFNKres_tFac	UNSET	tolerance parameter for FGMRES residual	
SEAICE_JFNKepsilon	1.0E-06	step size for the FD-Jacobian-times-vector	
SEAICE_dumpFreq	dumpFreq	dump frequency	
SEAICE_taveFreq	taveFreq	time-averaging frequency	
SEAICE_dump_mdsio	T	write snap-shot using MDSIO	
SEAICE_tave_mdsio	T	write TimeAverage using MDSIO	
SEAICE_dump_mnc	F	write snap-shot using MNC	
SEAICE_tave_mnc	F	write TimeAverage using MNC	
SEAICE_initialHEFF	0.00000E+00	initial sea-ice thickness	
SEAICE_drag	2.00000E-03	air-ice drag coefficient	
OCEAN_drag	1.00000E-03	air-ocean drag coefficient	
SEAICE_waterDrag	5.50000E+00	water-ice drag	
SEAICE_dryIceAlb	7.50000E-01	winter albedo	
SEAICE_wetIceAlb	6.60000E-01	summer albedo	
SEAICE_drySnowAlb	8.40000E-01	dry snow albedo	
SEAICE_wetSnowAlb	7.00000E-01	wet snow albedo	
SEAICE_waterAlbedo	1.00000E-01	water albedo	
SEAICE_strength	2.75000E+04	sea-ice strength P^*	
SEAICE_cStar	20.0000E+00	sea-ice strength paramter C^*	
SEAICE_rhoAir	1.3 (or `exf` value)	density of air (kg/m^3)	
SEAICE_cpAir	1004 (or `exf` value)	specific heat of air (J/kg/K)	
SEAICE_lhEvap	2,500,000 (or `exf` value)	latent heat of evaporation	
SEAICE_lhFusion	334,000 (or `exf` value)	latent heat of fusion	
SEAICE_lhSublim	2,834,000	latent heat of sublimation	
SEAICE_dalton	1.75E-03	sensible heat transfer coefficient	
SEAICE_iceConduct	2.16560E+00	sea-ice conductivity	
SEAICE_snowConduct	3.10000E-01	snow conductivity	
SEAICE_emissivity	5.50000E-08	Stefan-Boltzman	
SEAICE_snowThick	1.50000E-01	cutoff snow thickness	
SEAICE_shortwave	3.00000E-01	penetration shortwave radiation	
SEAICE_freeze	-1.96000E+00	freezing temp. of sea water	
SEAICE_saltFrac	0.0	salinity newly formed ice (fraction of ocean surface salinity)	
SEAICE_frazilFrac	0.0	Fraction of surface level negative heat content anomalies (relative to the local freezing point)	
SEAICEstressFactor	1.00000E+00	scaling factor for ice-ocean stress	
Heff/Area/HsnowFile/Hsalt	UNSET	initial fields for variables HEFF/AREA/HSNOW/HSALT	
LSR_ERROR	1.00000E-04	sets accuracy of LSR solver	
DIFF1	0.0	parameter used in advect.F	
HO	5.00000E-01	demarcation ice thickness (AKA lead closing paramter h_0)	
MAX_HEFF	1.00000E+01	maximum ice thickness	
MIN_ATEMP	-5.00000E+01	minimum air temperature	
MIN_LWDOWN	6.00000E+01	minimum downward longwave	
MAX_TICE	3.00000E+01	maximum ice temperature	
MIN_TICE	-5.00000E+01	minimum ice temperature	
IMAX_TICE	10	iterations for ice heat budget	
SEAICE_EPS	1.00000E-10	reduce derivative singularities	
SEAICE_area_reg	1.00000E-5	minimum concentration to regularize ice thickness	
SEAICE_hice_reg	0.05 m	minimum ice thickness for regularization	
SEAICE_multDim	1	number of ice categories for thermodynamics	
SEAICE_useMultDimSnow	F	use SEAICE_multDim snow categories	

Input fields and units

`HeffFile`: Initial sea ice thickness averaged over grid cell in meters; initializes variable `HEFF`;

`AreaFile`: Initial fractional sea ice cover, range $[0, 1]$; initializes variable `AREA`;

`HsnowFile`: Initial snow thickness on sea ice averaged over grid cell in meters; initializes variable `HSNOW`;

`HsaltFile`: Initial salinity of sea ice averaged over grid cell in g/m^2; initializes variable `HSALT`;

6.6.2.4 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 [*Zhang and Hibler*, 1997] first introduced by *Hibler* [1979, 1980]. 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 [*Losch et al.*, 2010]:

- the code has been rewritten for an Arakawa C-grid, both B- and C-grid variants are available; the C-grid code allows for no-slip and free-slip lateral boundary conditions;
- three different solution methods for solving the nonlinear momentum equations have been adopted: LSOR [*Zhang and Hibler*, 1997], EVP [*Hunke and Dukowicz*, 1997], JFNK [*Lemieux et al.*, 2010; *Losch et al.*, 2014];
- ice-ocean stress can be formulated as in *Hibler and Bryan* [1987] or as in *Campin et al.* [2008];
- ice variables are advected by sophisticated, conservative advection schemes with flux limiting;
- growth and melt parameterizations have been refined and extended in order to allow for more stable automatic differentiation of the code.

The sea ice model is tightly coupled to the ocean compontent of the MITgcm. Heat, fresh water fluxes and surface stresses are computed from the atmospheric state and – by default – modified by the ice model at every time step.

The ice dynamics models that are most widely used for large-scale climate studies are the viscous-plastic (VP) model [*Hibler*, 1979], the cavitating fluid (CF) model [*Flato and Hibler*, 1992], and the elastic-viscous-plastic (EVP) model [*Hunke and Dukowicz*, 1997]. Compared to the VP model, the CF model does not allow ice shear in calculating ice motion, stress, and deformation. EVP models approximate VP by adding an elastic term to the equations for easier adaptation to parallel computers. Because of its higher accuracy in plastic solution and relatively simpler formulation, compared to the EVP model, we decided to use the VP model as the default dynamic component of our ice model. To do this we extended the line successive over relaxation (LSOR) method of *Zhang and Hibler* [1997] for use in a parallel configuration. An EVP model and a free-drift implemtation can be selected with runtime flags.

Compatibility with ice-thermodynamics package `thsice`

Note, that by default the `seaice`-package includes the orginial so-called zero-layer thermodynamics following *Hibler* [1980] with a snow cover as in *Zhang et al.* [1998]. 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 *Semtner*'s [1976] three-layer thermodynamic model that permits heat storage in ice. Recently, the three-layer thermodynamic model has been reformulated by *Winton* [2000]. The reformulation improves model physics by representing the brine content of the upper ice with a variable heat capacity. It also improves model numerics and consumes less computer time and memory.

The Winton sea-ice thermodynamics have been ported to the MIT GCM; they currently reside under `pkg/thsice`. The package `thsice` is described in section 6.6.1; it is fully compatible with the packages `seaice` and `exf`. When turned on together with `seaice`, the zero-layer thermodynamics are replaced by the Winton thermodynamics. In order to use the `seaice`-package with the thermodynamics of `thsice`, compile both packages and turn both package on in `data.pkg`; see an example in `global_ocean.cs32x15/input.icedyn`. Note, that once `thsice` is turned on, the variables and diagnostics associated to the default thermodynamics are meaningless, and the diagnostics of `thsice` have to be used instead.

Surface forcing

The sea ice model requires the following input fields: 10-m winds, 2-m air temperature and specific humidity, downward longwave and shortwave radiations, precipitation, evaporation, and river and glacier runoff. The sea ice model also requires surface temperature from the ocean model and the top level horizontal velocity. Output fields are surface wind stress, evaporation minus precipitation minus runoff, net surface heat flux, and net shortwave flux. The sea-ice model is global: in ice-free regions bulk formulae are used to estimate oceanic forcing from the atmospheric fields.

Dynamics

The momentum equation of the sea-ice model is

$$m\frac{D\vec{\mathbf{u}}}{Dt} = -mf\vec{\mathbf{k}} \times \vec{\mathbf{u}} + \vec{\tau}_{air} + \vec{\tau}_{ocean} - m\nabla\phi(0) + \vec{\mathbf{F}}, \tag{6.39}$$

where $m = m_i + m_s$ is the ice and snow mass per unit area; $\vec{\mathbf{u}} = u\vec{\mathbf{i}} + v\vec{\mathbf{j}}$ is the ice velocity vector; $\vec{\mathbf{i}}$, $\vec{\mathbf{j}}$, and $\vec{\mathbf{k}}$ are unit vectors in the x, y, and z directions, respectively; f is the Coriolis parameter; $\vec{\tau}_{air}$ and $\vec{\tau}_{ocean}$ are the wind-ice and ocean-ice stresses, respectively; g is the gravity accelation; $\nabla\phi(0)$ is the gradient (or tilt) of the sea surface height; $\phi(0) = g\eta + p_a/\rho_0 + mg/\rho_0$ is the sea surface height potential in response to ocean dynamics ($g\eta$), to atmospheric pressure loading (p_a/ρ_0, where ρ_0 is a reference density) and a term due to snow and ice loading [*Campin et al.*, 2008]; and $\vec{\mathbf{F}} = \nabla \cdot \sigma$ is the divergence of the internal ice stress tensor σ_{ij}. Advection of sea-ice momentum is neglected. The wind and ice-ocean stress terms are given by

$$\begin{aligned}\vec{\tau}_{air} =& \rho_{air} C_{air} |\vec{\mathbf{U}}_{air} - \vec{\mathbf{u}}| R_{air} (\vec{\mathbf{U}}_{air} - \vec{\mathbf{u}}),\\ \vec{\tau}_{ocean} =& \rho_{ocean} C_{ocean} |\vec{\mathbf{U}}_{ocean} - \vec{\mathbf{u}}| R_{ocean} (\vec{\mathbf{U}}_{ocean} - \vec{\mathbf{u}}),\end{aligned}$$

where $\vec{\mathbf{U}}_{air/ocean}$ are the surface winds of the atmosphere and surface currents of the ocean, respectively; $C_{air/ocean}$ are air and ocean drag coefficients; $\rho_{air/ocean}$ are reference densities; and $R_{air/ocean}$ are rotation matrices that act on the wind/current vectors.

Viscous-Plastic (VP) Rheology

For an isotropic system the stress tensor σ_{ij} $(i, j = 1, 2)$ can be related to the ice strain rate and strength by a nonlinear viscous-plastic (VP) constitutive law [*Hibler*, 1979; *Zhang and Hibler*, 1997]:

$$\sigma_{ij} = 2\eta(\dot{\epsilon}_{ij}, P)\dot{\epsilon}_{ij} + [\zeta(\dot{\epsilon}_{ij}, P) - \eta(\dot{\epsilon}_{ij}, P)]\,\dot{\epsilon}_{kk}\delta_{ij} - \frac{P}{2}\delta_{ij}. \tag{6.40}$$

The ice strain rate is given by

$$\dot{\epsilon}_{ij} = \frac{1}{2}\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right).$$

The maximum ice pressure $P_{\max}$, a measure of ice strength, depends on both thickness h and compactness (concentration) c:

$$P_{\max} = P^* c\, h\, \exp\{-C^* \cdot (1 - c)\}, \tag{6.41}$$

with the constants P^* (run-time parameter `SEAICE_strength`) and $C^* = 20$. The nonlinear bulk and shear viscosities η and ζ are functions of ice strain rate invariants and ice strength such that the principal components of the stress lie on an elliptical yield curve with the ratio of major to minor axis e equal to 2; they are given by:

$$\begin{aligned}\zeta =& \min\left(\frac{P_{\max}}{2\max(\Delta, \Delta_{\min})}, \zeta_{\max}\right)\\ \eta =& \frac{\zeta}{e^2}\end{aligned}$$

with the abbreviation

$$\Delta = \left[\left(\dot{\epsilon}_{11}^2 + \dot{\epsilon}_{22}^2\right)(1 + e^{-2}) + 4e^{-2}\dot{\epsilon}_{12}^2 + 2\dot{\epsilon}_{11}\dot{\epsilon}_{22}(1 - e^{-2})\right]^{\frac{1}{2}}.$$

The bulk viscosities are bounded above by imposing both a minimum $\Delta_{\min}$ (for numerical reasons, run-time parameter SEAICE_EPS with a default value of $10^{-10}\,\mathrm{s}^{-1}$) and a maximum $\zeta_{\max} = P_{\max}/\Delta^*$, where $\Delta^* = (5\times 10^{12}/2\times 10^4)\,\mathrm{s}^{-1}$. (There is also the option of bounding ζ from below by setting run-time parameter SEAICE_zetaMin > 0, but this is generally not recommended). For stress tensor computation the replacement pressure $P = 2\,\Delta\zeta$ [*Hibler and Ip*, 1995] is used so that the stress state always lies on the elliptic yield curve by definition.

Defining the CPP-flag SEAICE_ZETA_SMOOTHREG in SEAICE_OPTIONS.h before compiling replaces the method for bounding ζ by a smooth (differentiable) expression:

$$\begin{aligned}\zeta &= \zeta_{\max}\tanh\left(\frac{P}{2\,\min(\Delta,\Delta_{\min})\,\zeta_{\max}}\right)\\ &= \frac{P}{2\Delta^*}\tanh\left(\frac{\Delta^*}{\min(\Delta,\Delta_{\min})}\right)\end{aligned} \tag{6.42}$$

where $\Delta_{\min} = 10^{-20}\,\mathrm{s}^{-1}$ is chosen to avoid divisions by zero.

LSR and JFNK solver

In the matrix notation, the discretized momentum equations can be written as

$$\mathbf{A}(\vec{\mathbf{x}})\,\vec{\mathbf{x}} = \vec{\mathbf{b}}(\vec{\mathbf{x}}). \tag{6.43}$$

The solution vector $\vec{\mathbf{x}}$ consists of the two velocity components u and v that contain the velocity variables at all grid points and at one time level. The standard (and default) method for solving Eq. (6.43) in the sea ice component of the MITgcm, as in many sea ice models, is an iterative Picard solver: in the k-th iteration a linearized form $\mathbf{A}(\vec{\mathbf{x}}^{k-1})\,\vec{\mathbf{x}}^k = \vec{\mathbf{b}}(\vec{\mathbf{x}}^{k-1})$ is solved (in the case of the MITgcm it is a Line Successive (over) Relaxation (LSR) algorithm [*Zhang and Hibler*, 1997]). Picard solvers converge slowly, but generally the iteration is terminated after only a few non-linear steps [*Zhang and Hibler*, 1997; *Lemieux and Tremblay*, 2009] 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 NPSEUDOTIMESTEPS (default is 2).

In order to overcome the poor convergence of the Picard-solver, *Lemieux et al.* [2010] introduced a Jacobian-free Newton-Krylov solver for the sea ice momentum equations. This solver is also implemented in the MITgcm [*Losch et al.*, 2014]. The Newton method transforms minimizing the residual $\vec{\mathbf{F}}(\vec{\mathbf{x}}) = \mathbf{A}(\vec{\mathbf{x}})\,\vec{\mathbf{x}} - \vec{\mathbf{b}}(\vec{\mathbf{x}})$ to finding the roots of a multivariate Taylor expansion of the residual $\vec{\mathbf{F}}$ around the previous $(k-1)$ estimate $\vec{\mathbf{x}}^{k-1}$:

$$\vec{\mathbf{F}}(\vec{\mathbf{x}}^{k-1}+\delta\vec{\mathbf{x}}^k) = \vec{\mathbf{F}}(\vec{\mathbf{x}}^{k-1}) + \vec{\mathbf{F}}'(\vec{\mathbf{x}}^{k-1})\,\delta\vec{\mathbf{x}}^k \tag{6.44}$$

with the Jacobian $\mathbf{J}\equiv\vec{\mathbf{F}}'$. The root $\vec{\mathbf{F}}(\vec{\mathbf{x}}^{k-1}+\delta\vec{\mathbf{x}}^k) = 0$ is found by solving

$$\mathbf{J}(\vec{\mathbf{x}}^{k-1})\,\delta\vec{\mathbf{x}}^k = -\vec{\mathbf{F}}(\vec{\mathbf{x}}^{k-1}) \tag{6.45}$$

for $\delta\vec{\mathbf{x}}^k$. The next (k-th) estimate is given by $\vec{\mathbf{x}}^k = \vec{\mathbf{x}}^{k-1} + a\,\delta\vec{\mathbf{x}}^k$. In order to avoid overshoots the factor a is iteratively reduced in a line search ($a = 1, \frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \ldots$) until $\|\vec{\mathbf{F}}(\vec{\mathbf{x}}^k)\| < \|\vec{\mathbf{F}}(\vec{\mathbf{x}}^{k-1})\|$, where $\|\cdot\| = \int\cdot\,dx^2$ is the L_2-norm. In practice, the line search is stopped at $a = \frac{1}{8}$. The line search starts after SEAICE_JFNK_lsIter non-linear Newton iterations (off by default).

Forming the Jacobian $\mathbf{J}$ explicitly is often avoided as "too error prone and time consuming" [*Knoll and Keyes*, 2004]. Instead, Krylov methods only require the action of $\mathbf{J}$ on an arbitrary vector $\vec{\mathbf{w}}$ and hence allow a matrix free algorithm for solving Eq. (6.45) [*Knoll and Keyes*, 2004]. The action of $\mathbf{J}$ can be approximated by a first-order Taylor series expansion:

$$\mathbf{J}(\vec{\mathbf{x}}^{k-1})\,\vec{\mathbf{w}} \approx \frac{\vec{\mathbf{F}}(\vec{\mathbf{x}}^{k-1}+\epsilon\vec{\mathbf{w}}) - \vec{\mathbf{F}}(\vec{\mathbf{x}}^{k-1})}{\epsilon} \tag{6.46}$$

or computed exactly with the help of automatic differentiation (AD) tools. SEAICE_JFNKepsilon sets the step size ϵ.

We use the Flexible Generalized Minimum RESidual method [FGMRES, *Saad*, 1993] with right-hand side preconditioning to solve Eq. (6.45) iteratively starting from a first guess of $\delta\vec{\mathbf{x}}^k_0 = 0$. For the preconditioning matrix $\mathbf{P}$ we choose a simplified form of the system matrix $\mathbf{A}(\vec{\mathbf{x}}^{k-1})$ [*Lemieux et al.*, 2010] where $\vec{\mathbf{x}}^{k-1}$ is the estimate of the previous Newton step $k-1$. The transformed equation (6.45) becomes

$$\mathbf{J}(\vec{\mathbf{x}}^{k-1})\,\mathbf{P}^{-1}\delta\vec{\mathbf{z}} = -\vec{\mathbf{F}}(\vec{\mathbf{x}}^{k-1}), \quad \text{with} \quad \delta\vec{\mathbf{z}} = \mathbf{P}\delta\vec{\mathbf{x}}^{k}. \tag{6.47}$$

The Krylov method iteratively improves the approximate solution to (6.47) in subspace ($\vec{\mathbf{r}}_0$, $\mathbf{JP}^{-1}\vec{\mathbf{r}}_0$, $(\mathbf{JP}^{-1})^2\vec{\mathbf{r}}_0$, …, $(\mathbf{JP}^{-1})^m\vec{\mathbf{r}}_0$) with increasing m; $\vec{\mathbf{r}}_0 = -\vec{\mathbf{F}}(\vec{\mathbf{x}}^{k-1}) - \mathbf{J}(\vec{\mathbf{x}}^{k-1})\,\delta\vec{\mathbf{x}}^{k}_0$ is the initial residual of (6.45); $\vec{\mathbf{r}}_0 = -\vec{\mathbf{F}}(\vec{\mathbf{x}}^{k-1})$ with the first guess $\delta\vec{\mathbf{x}}^{k}_0 = 0$. We allow a Krylov-subspace of dimension $m = 50$ and we do not use restarts. The preconditioning operation involves applying $\mathbf{P}^{-1}$ to the basis vectors $\vec{\mathbf{v}}_0, \vec{\mathbf{v}}_1, \vec{\mathbf{v}}_2, \ldots, \vec{\mathbf{v}}_m$ of the Krylov subspace. This operation is approximated by solving the linear system $\mathbf{P}\,\vec{\mathbf{w}} = \vec{\mathbf{v}}_i$. Because $\mathbf{P} \approx \mathbf{A}(\vec{\mathbf{x}}^{k-1})$, we can use the LSR-algorithm [*Zhang and Hibler*, 1997] 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 *Lemieux et al.* [2010]; *Losch et al.* [2014].

To use the JFNK-solver set `SEAICEuseJFNK = .TRUE.` in the namelist file `data.seaice`; `SEAICE_ALLOW_JFNK` needs to be defined in `SEAICE_OPTIONS.h` and we recommend using a smooth regularization of ζ by defining `SEAICE_ZETA_SMOOTHREG` (see above) for better convergence. The non-linear Newton iteration is terminated when the L_2-norm of the residual is reduced by $\gamma_{\rm nl}$ (runtime parameter `JFNKgamma_nonlin = 1.e-4` will already lead to expensive simulations) with respect to the initial norm: $\|\vec{\mathbf{F}}(\vec{\mathbf{x}}^{k})\| < \gamma_{\rm nl}\|\vec{\mathbf{F}}(\vec{\mathbf{x}}^{0})\|$. Within a non-linear iteration, the linear FGMRES solver is terminated when the residual is smaller than $\gamma_k\|\vec{\mathbf{F}}(\vec{\mathbf{x}}^{k-1})\|$ where γ_k is determined by

$$\gamma_k = \begin{cases} \gamma_0 & \text{for } \|\vec{\mathbf{F}}(\vec{\mathbf{x}}^{k-1})\| \geq r, \\ \max\left(\gamma_{\min}, \frac{\|\vec{\mathbf{F}}(\vec{\mathbf{x}}^{k-1})\|}{\|\vec{\mathbf{F}}(\vec{\mathbf{x}}^{k-2})\|}\right) & \text{for } \|\vec{\mathbf{F}}(\vec{\mathbf{x}}^{k-1})\| < r, \end{cases} \tag{6.48}$$

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 [e.g., *Knoll and Keyes*, 2004]. Typical parameter choices are $\gamma_0 =$ `JFNKgamma_lin_max` $= 0.99$, $\gamma_{\min} =$ `JFNKgamma_lin_min` $= 0.1$, and $r =$ `JFNKres_tFac` $\times \|\vec{\mathbf{F}}(\vec{\mathbf{x}}^{0})\|$ with `JFNKres_tFac` $= \frac{1}{2}$. 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" [*Hutchings et al.*, 2004] 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

Hunke and Dukowicz [1997]'s introduced an elastic contribution to the strain rate in order to regularize Eq. 6.40 in such a way that the resulting elastic-viscous-plastic (EVP) and VP models are identical at steady state,

$$\frac{1}{E}\frac{\partial\sigma_{ij}}{\partial t} + \frac{1}{2\eta}\sigma_{ij} + \frac{\eta-\zeta}{4\zeta\eta}\sigma_{kk}\delta_{ij} + \frac{P}{4\zeta}\delta_{ij} = \dot{\epsilon}_{ij}. \tag{6.49}$$

The EVP-model uses an explicit time stepping scheme with a short timestep. According to the recommendation of *Hunke and Dukowicz* [1997], the EVP-model should be stepped forward in time 120 times (`SEAICE_deltaTevp` $=$ `SEAICIE_deltaTdyn`$/120$) within the physical ocean model time step (although this parameter is under debate), to allow for elastic waves to disappear. Because the scheme does not require a matrix inversion it is fast in spite of the small internal timestep and simple to implement on parallel computers [*Hunke and Dukowicz*, 1997]. For completeness, we repeat the equations for the components of the stress tensor $\sigma_1 = \sigma_{11} + \sigma_{22}$, $\sigma_2 = \sigma_{11} - \sigma_{22}$, and σ_{12}. Introducing the divergence $D_D = \dot{\epsilon}_{11} + \dot{\epsilon}_{22}$, and the horizontal tension and shearing strain rates, $D_T = \dot{\epsilon}_{11} - \dot{\epsilon}_{22}$ and $D_S = 2\dot{\epsilon}_{12}$, respectively, and using the above abbreviations, the equations 6.49 can be written as:

$$\frac{\partial\sigma_1}{\partial t} + \frac{\sigma_1}{2T} + \frac{P}{2T} = \frac{P}{2T\Delta}D_D \tag{6.50}$$

$$\frac{\partial\sigma_2}{\partial t} + \frac{\sigma_2 e^2}{2T} = \frac{P}{2T\Delta}D_T \tag{6.51}$$

$$\frac{\partial\sigma_{12}}{\partial t} + \frac{\sigma_{12} e^2}{2T} = \frac{P}{4T\Delta}D_S \tag{6.52}$$

Here, the elastic parameter E is redefined in terms of a damping timescale T for elastic waves

$$E = \frac{\zeta}{T}.$$

$T = E_0\Delta t$ with the tunable parameter $E_0 < 1$ and the external (long) timestep Δt. $E_0 = \frac{1}{3}$ is the default value in the code and close to what *Hunke and Dukowicz* [1997] and *Hunke* [2001] 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. *Hunke and Dukowicz* [1997] recommend order(120) time steps for the EVP solver within one model time step Δt (`deltaTmom`). One can also choose order(120) time steps within the forcing time scale, but then we recommend adjusting the damping time scale T accordingly, by setting either `SEAICE_elasticParm` (E_0), so that $E_0\Delta t =$ forcing time scale, or directly `SEAICE_evpTauRelax` (T) to the forcing time scale.

More stable variants of Elastic-Viscous-Plastic Dynamics: EVP* , mEVP, and aEVP

The genuine EVP schemes appears to give noisy solutions [*Hunke*, 2001; *Lemieux et al.*, 2012; *Bouillon et al.*, 2013]. This has lead to a modified EVP or EVP* [*Lemieux et al.*, 2012; *Bouillon et al.*, 2013; *Kimmritz et al.*, 2015]; here, we refer to these variants by modified EVP (mEVP) and adaptive EVP (aEVP) [*Kimmritz et al.*, 2016]. The main idea is to modify the "natural" time-discretization of the momentum equations:

$$m\frac{D\vec{u}}{Dt} \approx m\frac{u^{p+1}-u^n}{\Delta t} + \beta^*\frac{u^{p+1}-u^p}{\Delta t_{\mathrm{EVP}}} \tag{6.53}$$

where n is the previous time step index, and p is the previous sub-cycling index. The extra "intertial" term $m\,(u^{p+1}-u^n)/\Delta t)$ allows the definition of a residual $|u^{p+1}-u^p|$ that, as $u^{p+1} \rightarrow u^{n+1}$, converges to 0. In this way EVP can be re-interpreted as a pure iterative solver where the sub-cycling has no association with time-relation (through Δt_{EVP}) [*Bouillon et al.*, 2013; *Kimmritz et al.*, 2015]. Using the terminology of *Kimmritz et al.* [2015], the evolution equations of stress σ_{ij} and momentum $\vec{u}$ can be written as:

$$\sigma_{ij}^{p+1} = \sigma_{ij}^{p} + \frac{1}{\alpha}\Big(\sigma_{ij}(\vec{u}^p) - \sigma_{ij}^p\Big), \tag{6.54}$$

$$\vec{u}^{p+1} = \vec{u}^p + \frac{1}{\beta}\Big(\frac{\Delta t}{m}\nabla\cdot\sigma^{p+1} + \frac{\Delta t}{m}\vec{R}^p + \vec{u}_n - \vec{u}^p\Big). \tag{6.55}$$

$\vec{R}$ contains all terms in the momentum equations except for the rheology terms and the time derivative; α and β are free parameters (`SEAICE_evpAlpha`, `SEAICE_evpBeta`) that replace the time stepping parameters `SEAICE_deltaTevp` (ΔT_{EVP}), `SEAICE_elasticParm` (E_0), or `SEAICE_evpTauRelax` (T). α and β determine the speed of convergence and the stability. Usually, it makes sense to use $\alpha = \beta$, and `SEAICEnEVPstarSteps` $\gg (\alpha, \beta)$ [*Kimmritz et al.*, 2015]. Currently, there is no termination criterion and the number of mEVP iterations is fixed to `SEAICEnEVPstarSteps`.

In order to use mEVP in the MITgcm, set `SEAICEuseEVPstar = .TRUE.,` in `data.seaice`. If `SEAICEuseEVPrev =.TRUE.,` the actual form of equations (6.54) and (6.55) is used with fewer implicit terms and the factor of e^2 dropped in the stress equations (6.51) and (6.52). Although this modifies the original EVP-equations, it turns out to improve convergence [*Bouillon et al.*, 2013].

Another variant is the aEVP scheme [*Kimmritz et al.*, 2016], where the value of α is set dynamically based on the stability criterion

$$\alpha = \beta = \max\left(\tilde{c}\pi\sqrt{c\frac{\zeta}{A_c}\frac{\Delta t}{\max(m, 10^{-4}\,\mathrm{kg})}}, \alpha_{\min}\right) \tag{6.56}$$

with the grid cell area A_c and the ice and snow mass m. This choice sacrifices speed of convergence for stability with the result that aEVP converges quickly to VP where α can be small and more slowly in areas where the equations are stiff. In practice, aEVP leads to an overall better convergence than mEVP [*Kimmritz et al.*, 2016]. To use aEVP in the MITgcm set `SEAICEaEVPcoeff` $= \tilde{c}$; this also sets the default values of `SEAICEaEVPcStar` ($c = 4$) and `SEAICEaEVPalphaMin` ($\alpha_{\min} = 5$). Good convergence has been obtained with setting these values [*Kimmritz et al.*, 2016]: `SEAICEaEVPcoeff = 0.5, SEAICEnEVPstarSteps = 500, SEAICEuseEVPstar = .TRUE., SEAICEuseEVPrev = .TRUE.`

Note, that probably because of the C-grid staggering of velocities and stresses, mEVP may not converge as successfully as in *Kimmritz et al.* [2015], and that convergence at very high resolution (order 5 km) has not been studied yet.

Truncated ellipse method (TEM) for yield curve

In the so-called truncated ellipse method the shear viscosity η is capped to suppress any tensile stress [*Hibler and Schulson*, 1997; *Geiger et al.*, 1998]:

$$\eta = \min\left(\frac{\zeta}{e^2}, \frac{\frac{P}{2} - \zeta(\dot{\epsilon}_{11} + \dot{\epsilon}_{22})}{\sqrt{\max(\Delta_{\min}^2, (\dot{\epsilon}_{11} - \dot{\epsilon}_{22})^2 + 4\dot{\epsilon}_{12}^2)}}\right). \tag{6.57}$$

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 $\vec{\tau}_{ocean}$ in Eq. 6.39. This stess is applied directly to the surface layer of the ocean model. An alternative ocean stress formulation is given by *Hibler and Bryan* [1987]. Rather than applying $\vec{\tau}_{ocean}$ directly, the stress is derived from integrating over the ice thickness to the bottom of the oceanic surface layer. In the resulting equation for the *combined* ocean-ice momentum, the interfacial stress cancels and the total stress appears as the sum of windstress and divergence of internal ice stresses: $\delta(z)(\vec{\tau}_{air} + \vec{\mathbf{F}})/\rho_0$, [see also Eq. 2 of *Hibler and Bryan*, 1987]. The disadvantage of this formulation is that now the velocity in the surface layer of the ocean that is used to advect tracers, is really an average over the ocean surface velocity and the ice velocity leading to an inconsistency as the ice temperature and salinity are different from the oceanic variables. To turn on the stress formulation of *Hibler and Bryan* [1987], set `useHB87StressCoupling=.TRUE.` in `data.seaice`.

Finite-volume discretization of the stress tensor divergence

On an Arakawa C grid, ice thickness and concentration and thus ice strength P and bulk and shear viscosities ζ and η are naturally defined a C-points in the center of the grid cell. Discretization requires only averaging of ζ and η to vorticity or Z-points (or ζ-points, but here we use Z in order avoid confusion with the bulk viscosity) at the bottom left corner of the cell to give $\overline{\zeta}^Z$ and $\overline{\eta}^Z$. In the following, the superscripts indicate location at Z or C points, distance across the cell (F), along the cell edge (G), between u-points (U), v-points (V), and C-points (C). The control volumes of the u- and v-equations in the grid cell at indices (i,j) are $A_{i,j}^w$ and $A_{i,j}^s$, respectively. With these definitions (which follow the model code documentation except that ζ-points have been renamed to Z-points), the strain rates are discretized as:

$$
\begin{aligned}
\dot{\epsilon}_{11} &= \partial_1 u_1 + k_2 u_2 \qquad (6.58)\\
=> (\epsilon_{11})^C_{i,j} &= \frac{u_{i+1,j} - u_{i,j}}{\Delta x^F_{i,j}} + k^C_{2,i,j} \frac{v_{i,j+1} + v_{i,j}}{2}\\
\dot{\epsilon}_{22} &= \partial_2 u_2 + k_1 u_1 \qquad (6.59)\\
=> (\epsilon_{22})^C_{i,j} &= \frac{v_{i,j+1} - v_{i,j}}{\Delta y^F_{i,j}} + k^C_{1,i,j} \frac{u_{i+1,j} + u_{i,j}}{2}\\
\dot{\epsilon}_{12} = \dot{\epsilon}_{21} &= \frac{1}{2}\left(\partial_1 u_2 + \partial_2 u_1 - k_1 u_2 - k_2 u_1\right) \qquad (6.60)\\
=> (\epsilon_{12})^Z_{i,j} &= \frac{1}{2}\left(\frac{v_{i,j} - v_{i-1,j}}{\Delta x^V_{i,j}} + \frac{u_{i,j} - u_{i,j-1}}{\Delta y^U_{i,j}}\right.\\
&\qquad \left. - k^Z_{1,i,j} \frac{v_{i,j} + v_{i-1,j}}{2} - k^Z_{2,i,j} \frac{u_{i,j} + u_{i,j-1}}{2}\right),
\end{aligned}
$$

so that the diagonal terms of the strain rate tensor are naturally defined at C-points and the symmetric off-diagonal term at Z-points. No-slip boundary conditions ($u_{i,j-1} + u_{i,j} = 0$ and $v_{i-1,j} + v_{i,j} = 0$ across boundaries) are implemented via “ghost-points”; for free slip boundary conditions $(\epsilon_{12})^Z = 0$ on boundaries.

For a spherical polar grid, the coefficients of the metric terms are $k_1 = 0$ and $k_2 = -\tan\phi/a$, with the spherical radius a and the latitude ϕ; $\Delta x_1 = \Delta x = a\cos\phi\Delta\lambda$, and $\Delta x_2 = \Delta y = a\Delta\phi$. For a general orthogonal curvilinear grid, k_1 and k_2 can be approximated by finite differences of the cell widths:

$$
\begin{aligned}
k^C_{1,i,j} &= \frac{1}{\Delta y^F_{i,j}} \frac{\Delta y^G_{i+1,j} - \Delta y^G_{i,j}}{\Delta x^F_{i,j}} \qquad (6.61)\\
k^C_{2,i,j} &= \frac{1}{\Delta x^F_{i,j}} \frac{\Delta x^G_{i,j+1} - \Delta x^G_{i,j}}{\Delta y^F_{i,j}} \qquad (6.62)\\
k^Z_{1,i,j} &= \frac{1}{\Delta y^U_{i,j}} \frac{\Delta y^C_{i,j} - \Delta y^C_{i-1,j}}{\Delta x^V_{i,j}} \qquad (6.63)\\
k^Z_{2,i,j} &= \frac{1}{\Delta x^V_{i,j}} \frac{\Delta x^C_{i,j} - \Delta x^C_{i,j-1}}{\Delta y^U_{i,j}} \qquad (6.64)
\end{aligned}
$$

The stress tensor is given by the constitutive viscous-plastic relation $\sigma_{\alpha\beta} = 2\eta\dot{\epsilon}_{\alpha\beta} + [(\zeta-\eta)\dot{\epsilon}_{\gamma\gamma} - P/2]\delta_{\alpha\beta}$ [*Hibler*, 1979]. The stress tensor divergence $(\nabla\sigma)_\alpha = \partial_\beta\sigma_{\beta\alpha}$, is discretized in finite volumes [see also *Losch et al.*, 2010]. This conveniently avoids dealing with further metric terms, as these are “hidden” in the differential cell widths. For the u-equation ($\alpha = 1$) we have:

$$
\begin{aligned}
(\nabla\sigma)_1 :\quad & \frac{1}{A^w_{i,j}} \int_{\text{cell}} (\partial_1\sigma_{11} + \partial_2\sigma_{21})\, dx_1\, dx_2 \qquad (6.65)\\
=& \frac{1}{A^w_{i,j}} \left\{ \int_{x_2}^{x_2+\Delta x_2} \sigma_{11} dx_2 \Big|_{x_1}^{x_1+\Delta x_1} + \int_{x_1}^{x_1+\Delta x_1} \sigma_{21} dx_1 \Big|_{x_2}^{x_2+\Delta x_2} \right\}\\
\approx& \frac{1}{A^w_{i,j}} \left\{ \Delta x_2 \sigma_{11} \Big|_{x_1}^{x_1+\Delta x_1} + \Delta x_1 \sigma_{21} \Big|_{x_2}^{x_2+\Delta x_2} \right\}\\
=& \frac{1}{A^w_{i,j}} \left\{ (\Delta x_2\sigma_{11})^C_{i,j} - (\Delta x_2\sigma_{11})^C_{i-1,j} \right.\\
& \qquad \left. + (\Delta x_1\sigma_{21})^Z_{i,j+1} - (\Delta x_1\sigma_{21})^Z_{i,j} \right\}
\end{aligned}
$$

with

$$
\begin{aligned}
(\Delta x_2\sigma_{11})^C_{i,j} = {} & \Delta y^F_{i,j}(\zeta+\eta)^C_{i,j}\frac{u_{i+1,j}-u_{i,j}}{\Delta x^F_{i,j}} \\
& + \Delta y^F_{i,j}(\zeta+\eta)^C_{i,j}k^C_{2,i,j}\frac{v_{i,j+1}+v_{i,j}}{2} \\
& + \Delta y^F_{i,j}(\zeta-\eta)^C_{i,j}\frac{v_{i,j+1}-v_{i,j}}{\Delta y^F_{i,j}} \\
& + \Delta y^F_{i,j}(\zeta-\eta)^C_{i,j}k^C_{1,i,j}\frac{u_{i+1,j}+u_{i,j}}{2} \\
& - \Delta y^F_{i,j}\frac{P}{2}
\end{aligned} \tag{6.66}
$$

$$
\begin{aligned}
(\Delta x_1\sigma_{21})^Z_{i,j} = {} & \Delta x^V_{i,j}\overline{\eta}^Z_{i,j}\frac{u_{i,j}-u_{i,j-1}}{\Delta y^U_{i,j}} \\
& + \Delta x^V_{i,j}\overline{\eta}^Z_{i,j}\frac{v_{i,j}-v_{i-1,j}}{\Delta x^V_{i,j}} \\
& - \Delta x^V_{i,j}\overline{\eta}^Z_{i,j}k^Z_{2,i,j}\frac{u_{i,j}+u_{i,j-1}}{2} \\
& - \Delta x^V_{i,j}\overline{\eta}^Z_{i,j}k^Z_{1,i,j}\frac{v_{i,j}+v_{i-1,j}}{2}
\end{aligned} \tag{6.67}
$$

Similarly, we have for the v-equation ($\alpha = 2$):

$$
\begin{aligned}
(\nabla\sigma)_2 : \quad & \frac{1}{A^s_{i,j}}\int_{\text{cell}}(\partial_1\sigma_{12}+\partial_2\sigma_{22})\,dx_1\,dx_2 \\
= & \frac{1}{A^s_{i,j}}\left\{\int_{x_2}^{x_2+\Delta x_2}\sigma_{12}dx_2\Big|_{x_1}^{x_1+\Delta x_1} + \int_{x_1}^{x_1+\Delta x_1}\sigma_{22}dx_1\Big|_{x_2}^{x_2+\Delta x_2}\right\} \\
\approx & \frac{1}{A^s_{i,j}}\left\{\Delta x_2\sigma_{12}\Big|_{x_1}^{x_1+\Delta x_1} + \Delta x_1\sigma_{22}\Big|_{x_2}^{x_2+\Delta x_2}\right\} \\
= & \frac{1}{A^s_{i,j}}\Big\{(\Delta x_2\sigma_{12})^Z_{i+1,j} - (\Delta x_2\sigma_{12})^Z_{i,j} \\
& \qquad + (\Delta x_1\sigma_{22})^C_{i,j} - (\Delta x_1\sigma_{22})^C_{i,j-1}\Big\}
\end{aligned} \tag{6.68}
$$

with

$$
\begin{aligned}
(\Delta x_1\sigma_{12})^Z_{i,j} = {} & \Delta y^U_{i,j}\overline{\eta}^Z_{i,j}\frac{u_{i,j}-u_{i,j-1}}{\Delta y^U_{i,j}} \\
& + \Delta y^U_{i,j}\overline{\eta}^Z_{i,j}\frac{v_{i,j}-v_{i-1,j}}{\Delta x^V_{i,j}} \\
& - \Delta y^U_{i,j}\overline{\eta}^Z_{i,j}k^Z_{2,i,j}\frac{u_{i,j}+u_{i,j-1}}{2} \\
& - \Delta y^U_{i,j}\overline{\eta}^Z_{i,j}k^Z_{1,i,j}\frac{v_{i,j}+v_{i-1,j}}{2}
\end{aligned} \tag{6.69}
$$

$$
\begin{aligned}
(\Delta x_2\sigma_{22})^C_{i,j} = {} & \Delta x^F_{i,j}(\zeta-\eta)^C_{i,j}\frac{u_{i+1,j}-u_{i,j}}{\Delta x^F_{i,j}} \\
& + \Delta x^F_{i,j}(\zeta-\eta)^C_{i,j}k^C_{2,i,j}\frac{v_{i,j+1}+v_{i,j}}{2} \\
& + \Delta x^F_{i,j}(\zeta+\eta)^C_{i,j}\frac{v_{i,j+1}-v_{i,j}}{\Delta y^F_{i,j}} \\
& + \Delta x^F_{i,j}(\zeta+\eta)^C_{i,j}k^C_{1,i,j}\frac{u_{i+1,j}+u_{i,j}}{2} \\
& - \Delta x^F_{i,j}\frac{P}{2}
\end{aligned}
$$

Again, no slip boundary conditions are realized via ghost points and $u_{i,j-1}+u_{i,j}=0$ and $v_{i-1,j}+v_{i,j}=0$ across boundaries. For free slip boundary conditions the lateral stress is set to zeros. In analogy to $(\epsilon_{12})^Z=0$ on boundaries, we set $\sigma_{21}^Z=0$, or equivalently $\eta_{i,j}^Z=0$, on boundaries.

Thermodynamics

NOTE: THIS SECTION IS TERRIBLY OUT OF DATE

In its original formulation the sea ice model [*Menemenlis et al.*, 2005] uses simple thermodynamics following the appendix of *Semtner* [1976]. This formulation does not allow storage of heat, that is, the heat capacity of ice is zero. Upward conductive heat flux is parameterized assuming a linear temperature profile and together with a constant ice conductivity. It is expressed as $(K/h)(T_w-T_0)$, where K is the ice conductivity, h the ice thickness, and T_w-T_0 the difference between water and ice surface temperatures. This type of model is often refered to as a "zero-layer" model. The surface heat flux is computed in a similar way to that of *Parkinson and Washington* [1979] and *Manabe et al.* [1979].

The conductive heat flux depends strongly on the ice thickness h. However, the ice thickness in the model represents a mean over a potentially very heterogeneous thickness distribution. In order to parameterize a sub-grid scale distribution for heat flux computations, the mean ice thickness h is split into N thickness categories H_n that are equally distributed between $2h$ and a minimum imposed ice thickness of 5 cm by $H_n=\frac{2n-1}{7}h$ for $n\in[1,N]$. The heat fluxes computed for each thickness category is area-averaged to give the total heat flux [*Hibler*, 1984]. To use this thickness category parameterization set `SEAICE_multDim` to the number of desired categories (7 is a good guess, for anything larger than 7 modify `SEAICE_SIZE.h`) in `data.seaice`; 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 produced [*Castro-Morales et al.*, 2014].

The atmospheric heat flux is balanced by an oceanic heat flux from below. The oceanic flux is proportional to $\rho\, c_p\,(T_w-T_{fr})$ where ρ and c_p are the density and heat capacity of sea water and T_{fr} is the local freezing point temperature that is a function of salinity. This flux is not assumed to instantaneously melt or create ice, but a time scale of three days (run-time parameter `SEAICE_gamma_t`) is used to relax T_w to the freezing point. The parameterization of lateral and vertical growth of sea ice follows that of *Hibler* [1979, 1980]; the so-called lead closing parameter h_0 (run-time parameter `HO`) has a default value of 0.5 meters.

On top of the ice there is a layer of snow that modifies the heat flux and the albedo [*Zhang et al.*, 1998]. Snow modifies the effective conductivity according to

$$\frac{K}{h}\rightarrow\frac{1}{\frac{h_s}{K_s}+\frac{h}{K}},$$

where K_s is the conductivity of snow and h_s the snow thickness. If enough snow accumulates so that its weight submerges the ice and the snow is flooded, a simple mass conserving parameterization of snowice formation (a flood-freeze algorithm following Archimedes' principle) turns snow into ice until the ice surface is back at $z=0$ [*Leppäranta*, 1983]. The flood-freeze algorithm is enabled with the CPP-flag `SEAICE_ALLOW_FLOODING` and turned on with run-time parameter `SEAICEuseFlooding=.true.`.

Advection of thermodynamic variables

Effective ice thickness (ice volume per unit area, $c\cdot h$), concentration c and effective snow thickness $(c\cdot h_s)$ are advected by ice velocities:

$$\frac{\partial X}{\partial t}=-\nabla\cdot(\vec{\mathbf{u}}\,X)+\Gamma_X+D_X \tag{6.70}$$

where Γ_X are the thermodynamic source terms and D_X the diffusive terms for quantities $X=(c\cdot h),c,(c\cdot h_s)$. From the various advection scheme that are available in the MITgcm, we recommend flux-limited schemes [multidimensional 2nd and 3rd-order advection scheme with flux limiter *Roe*, 1985; *Hundsdorfer and Trompert*, 1994] 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), `DIFF1` $=D_X/\Delta x$ (default=0.004).

The MITgcm sea ice model provides the option to use the thermodynamics model of *Winton* [2000], which in turn is based on the 3-layer model of *Semtner* [1976] and which treats brine content by means of enthalpy conservation; the corresponding package `thsice` is described in section 6.6.1. This scheme requires additional state variables, namely the enthalpy of the two ice layers (instead of effective ice salinity), to be advected by ice velocities. The internal sea ice temperature is inferred from ice enthalpy. To avoid unphysical (negative) values for ice thickness and concentration, a positive 2nd-order advection scheme with a SuperBee flux limiter [*Roe*, 1985] should be used to advect all sea-ice-related quantities of the *Winton* [2000] 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.

6.6.2.5 Key subroutines

Top-level routine: `seaice_model.F`

```
C     !CALLING SEQUENCE:
c ...
c  seaice_model (TOP LEVEL ROUTINE)
c  |
c  |-- #ifdef SEAICE_CGRID
c  |     SEAICE_DYNSOLVER
c  |     |
c  |     |-- < compute proxy for geostrophic velocity >
c  |     |
c  |     |-- < set up mass per unit area and Coriolis terms >
c  |     |
c  |     |-- < dynamic masking of areas with no ice >
c  |     |
c  |     |

c  |   #ELSE
c  |     DYNSOLVER
c  |   #ENDIF
c  |
c  |-- if ( useOBCS )
c  |     OBCS_APPLY_UVICE
c  |
c  |-- if ( SEAICEadvHeff .OR. SEAICEadvArea .OR. SEAICEadvSnow .OR. SEAICEadvSalt )
c  |     SEAICE_ADVDIFF
c  |
c  |-- if ( usePW79thermodynamics )
c  |     SEAICE_GROWTH
c  |
c  |-- if ( useOBCS )
c  |     if ( SEAICEadvHeff ) OBCS_APPLY_HEFF
c  |     if ( SEAICEadvArea ) OBCS_APPLY_AREA
c  |     if ( SEAICEadvSALT ) OBCS_APPLY_HSALT
c  |     if ( SEAICEadvSNOW ) OBCS_APPLY_HSNOW
c  |
c  |-- < do various exchanges >
c  |
c  |-- < do additional diagnostics >
c  |
c  o
```

6.6.2.6 SEAICE diagnostics

Diagnostics output is available via the diagnostics package (see Section 7.1). Available output fields are summarized in Table 6.6.2.6.

6.6.2.7 Experiments and tutorials that use seaice

- Labrador Sea experiment in `lab_sea` verification directory.
- `seaice_obcs`, based on `lab_sea`
- `offline_exf_seaice/input.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`

6.6.3 SHELFICE Package

Authors: Martin Losch, Jean-Michel Campin

6.6.3.1 Introduction

Package “shelfice” provides a thermodynamic model for basal melting underneath floating ice shelves.

CPP options enable or disable different aspects of the package (Section 6.6.3.2). Run-Time options, flags, filenames and field-related dates/times are set in `data.shelfice` (Section 6.6.3.3). A description of key subroutines is given in Section 6.6.3.5. Input fields, units and sign conventions are summarized in Section 6.6.3.3, and available diagnostics output is listed in Section 6.6.3.6.

6.6.3.2 SHELFICE configuration, compiling & running

Compile-time options

As with all MITgcm packages, SHELFICE can be turned on or off at compile time

- using the `packages.conf` file by adding `shelfice` to it,
- or using `genmake2` adding `-enable=shelfice` or `-disable=shelfice` switches
- *required packages and CPP options*:
  SHELFICE does not require any additional packages, but it will only work with conventional vertical z-coordinates (pressure coordinates are not implemented, yet). If you use it together with vertical mixing schemes, be aware, that non-local parameterizations have been turned off, e.g. for KPP (6.4.2).

(see Section 3.4).

Parts of the SHELFICE code can be enabled or disabled at compile time via CPP preprocessor flags. These options are set `SHELFICE_OPTIONS.h`. Table 6.6.3.2 summarizes these options.

6.6.3.3 Run-time parameters

Run-time parameters are set in files `data.pkg` (read in `packages_readparms.F`), and `data.shelfice` (read in `shelfice_readparms.F`).

Enabling the package

A package is switched on/off at run-time by setting (e.g. for SHELFICE) `useSHELFICE = .TRUE.` in `data.pkg`.

General flags and parameters

Table 6.22 lists all run-time parameters.

Input fields and units

`SHEFLICEtopoFile`**:** under-ice topography of ice shelves in meters; upwards is positive, that as for the bathymetry files, negative values are required for topography below the sea-level;

`SHEFLICEloadAnomalyFile`**:** pressure load anomaly at the bottom of the ice shelves in pressure units (Pa); this field is absolutely required to avoid large excursions of the free surface during initial adjustment processes; obtained by integrating an approximate density from the surface at $z = 0$ down to the bottom of the last fully dry cell within the ice shelf, see Eq. (6.75); however, the file `SHEFLICEloadAnomalyFile` must not be p_{top}, but $p_{top} - g\sum_{k'=1}^{n-1}\rho_0\Delta z_{k'}$, with $\rho_0 =$ `rhoConst`, so that in the absenses of a ρ^* that is different from ρ_0, the anomaly is zero.

6.6.3.4 Description

In the light of isomorphic equations for pressure and height coordinates, the ice shelf topography on top of the water column has a similar role as (and in the language of *Marshall et al.* [2004] is isomorphic to) the orography and the pressure boundary conditions at the bottom of the fluid for atmospheric and oceanic models in pressure coordinates.

The total pressure p_{tot} in the ocean can be divided into the pressure at the top of the water column p_{top}, the hydrostatic pressure and the non-hydrostatic pressure contribution p_{NH}:

$$p_{tot} = p_{top} + \int_{z}^{\eta-h} g\,\rho\,dz + p_{NH}, \tag{6.71}$$

with the gravitational acceleration g, the density ρ, the vertical coordinate z (positive upwards), and the dynamic sea-surface height η. For the open ocean, $p_{top} = p_a$ (atmospheric pressure) and $h = 0$. Underneath an ice-shelf that is assumed to be floating in isostatic equilibrium, p_{top} at the top of the water column is the atmospheric pressure p_a plus the weight of the ice-shelf. It is this weight of the ice-shelf that has to be provided as a boundary condition at the top of the water column (in run-time parameter `SHELFICEloadAnomalyFile`). The weight is conveniently computed by integrating a density profile ρ^*, that is constant in time and corresponds to the sea-water replaced by ice, from $z = 0$ to a "reference" ice-shelf draft at $z = -h$ [*Beckmann et al.*, 1999], so that

$$p_{top} = p_a + \int_{-h}^{0} g\,\rho^*\,dz. \tag{6.72}$$

Underneath the ice shelf, the "sea-surface height" η is the deviation from the "reference" ice-shelf draft h. During a model integration, η adjusts so that the isostatic equilibrium is maintained for sufficiently slow and large scale motion.

In the MITgcm, the total pressure anomaly p'_{tot} which is used for pressure gradient computations is defined by substracting a purely depth dependent contribution $-g\rho_0 z$ with a constant reference density ρ_0 from p_{tot}. Eq. (6.71) becomes

$$p_{tot} = p_{top} - g\,\rho_0\,(z+h) + g\,\rho_0\,\eta + \int_{z}^{\eta-h} g\,(\rho-\rho_0)\,dz + p_{NH}, \tag{6.73}$$

and after rearranging

$$p'_{tot} = p'_{top} \qquad\qquad + g\,\rho_0\,\eta + \int_{z}^{\eta-h} g\,(\rho-\rho_0)\,dz + p_{NH}, \tag{6.74}$$

with $p'_{tot} = p_{tot} + g\,\rho_0\,z$ and $p'_{top} = p_{top} - g\,\rho_0\,h$. The non-hydrostatic pressure contribution p_{NH} is neglected in the following.

In practice, the ice shelf contribution to p_{top} is computed by integrating Eq. (6.72) from $z = 0$ to the bottom of the last fully dry cell within the ice shelf:

$$p_{top} = g \sum_{k'=1}^{n-1} \rho^*_{k'} \Delta z_{k'} + p_a \tag{6.75}$$

where n is the vertical index of the first (at least partially) "wet" cell and $\Delta z_{k'}$ is the thickness of the k'-th layer (counting downwards). The pressure anomaly for evaluating the pressure gradient is computed in the center of the "wet" cell k as

$$p'_k = p'_{top} + g\rho_n\eta + g\sum_{k'=n}^{k}\left((\rho_{k'}-\rho_0)\Delta z_{k'}\frac{1+H(k'-k)}{2}\right) \tag{6.76}$$

where $H(k'-k) = 1$ for $k' < k$ and 0 otherwise.

Setting `SHELFICEboundaryLayer=.true.` introduces a simple boundary layer that reduces the potential noise problem at the cost of increased vertical mixing. For this purpose the water temperature at the k-th layer abutting ice shelf topography for use in the heat flux parameterizations is computed as a mean temperature $\overline{\theta}_k$ over a boundary layer of the same thickness as the layer thickness Δz_k:

$$\overline{\theta}_k = \theta_k h_k + \theta_{k+1}(1 - h_k) \tag{6.77}$$

where $h_k \in (0, 1]$ is the fractional layer thickness of the k-th layer. The original contributions due to ice shelf-ocean interaction g_θ to the total tendency terms G_θ in the time-stepping equation $\theta^{n+1} = f(\theta^n, \Delta t, G_\theta^n)$ are

$$g_{\theta,k} = \frac{Q}{\rho_0 c_p h_k \Delta z_k} \text{ and } g_{\theta,k+1} = 0 \tag{6.78}$$

for layers k and $k+1$ (c_p is the heat capacity). Averaging these terms over a layer thickness Δz_k (e.g., extending from the ice shelf base down to the dashed line in cell C) and applying the averaged tendency to cell A (in layer k) and to the appropriate fraction of cells C (in layer $k+1$) yields

$$g^*_{\theta,k} = \frac{Q}{\rho_0 c_p \Delta z_k} \tag{6.79}$$

$$g^*_{\theta,k+1} = \frac{Q}{\rho_0 c_p \Delta z_k} \frac{\Delta z_k (1 - h_k)}{\Delta z_{k+1}}. \tag{6.80}$$

Eq. (6.80) describes averaging over the part of the grid cell $k+1$ that is part of the boundary layer with tendency $g^*_{\theta,k}$ and the part with no tendency. Salinity is treated in the same way. The momentum equations are not modified.

Three-Equations-Thermodynamics Freezing and melting form a boundary layer between ice shelf and ocean. Phase transitions at the boundary between saline water and ice imply the following fluxes across the boundary: the freshwater mass flux q (< 0 for melting); the heat flux that consists of the diffusive flux through the ice, the latent heat flux due to melting and freezing and the heat that is carried by the mass flux; and the salinity that is carried by the mass flux, if the ice has a non-zero salinity S_I. Further, the position of the interface between ice and ocean changes because of q, so that, say, in the case of melting the volume of sea water increases. As a consequence salinity and temperature are modified.

The turbulent exchange terms for tracers at the ice-ocean interface are generally expressed as diffusive fluxes. Following *Jenkins et al.* [2001], the boundary conditions for a tracer take into account that this boundary is not a material surface. The implied upward freshwater flux q (in mass units, negative for melting) is included in the boundary conditions for the temperature and salinity equation as an advective flux:

$$\rho K \frac{\partial X}{\partial z}\bigg|_b = (\rho \gamma_X - q)(X_b - X) \tag{6.81}$$

where tracer X stands for either temperature T or salinity S. X_b is the tracer at the interface (taken to be at freezing), X is the tracer at the first interior grid point, ρ is the density of seawater, and γ_X is the turbulent exchange coefficient (in units of an exchange velocity). The left hand side of Eq. (6.81) is shorthand for the (downward) flux of tracer X across the boundary. T_b, S_b and the freshwater flux q are obtained from solving a system of three equations that is derived from the heat and freshwater balance at the ice ocean interface.

In this so-called three-equation-model [e.g., *Hellmer and Olbers*, 1989; *Jenkins et al.*, 2001] the heat balance at the ice-ocean interface is expressed as

$$c_p \rho \gamma_T (T - T_b) + \rho_I c_{p,I} \kappa \frac{(T_S - T_b)}{h} = -Lq \tag{6.82}$$

where ρ is the density of sea-water, $c_p = 3974\,\mathrm{J\,kg^{-1}\,K^{-1}}$ is the specific heat capacity of water and γ_T the turbulent exchange coefficient of temperature. The value of γ_T is discussed in *Holland and Jenkins* [1999]. $L = 334000\,\mathrm{J\,kg^{-1}}$ is the latent heat of fusion. $\rho_I = 920\,\mathrm{kg\,m^{-3}}$, $c_{p,I} = 2000\,\mathrm{J\,kg^{-1}\,K^{-1}}$, and T_S are the density, heat capacity and the surface temperature of the ice shelf; $\kappa = 1.54 \times 10^{-6}\,\mathrm{m^2\,s^{-1}}$ is the heat diffusivity through the ice-shelf and h is the ice-shelf draft. The second term on the right hand side describes the heat flux through the ice shelf. A constant surface temperature $T_S = -20°$ is imposed. T is the temperature of the model cell adjacent to the ice-water interface. The temperature at the interface

T_b is assumed to be the in-situ freezing point temperature of sea-water T_f which is computed from a linear equation of state

$$T_f = (0.0901 - 0.0575\ S_b)^{\circ} - 7.61 \times 10^{-4} \frac{\mathrm{K}}{\mathrm{dBar}}\ p_b \tag{6.83}$$

with the salinity S_b and the pressure p_b (in dBar) in the cell at the ice-water interface. From the salt budget, the salt flux across the shelf ice-ocean interface is equal to the salt flux due to melting and freezing:

$$\rho \gamma_S (S - S_b) = -q\,(S_b - S_I), \tag{6.84}$$

where $\gamma_S = 5.05 \times 10^{-3} \gamma_T$ is the turbulent salinity exchange coefficient, and S and S_b are defined in analogy to temperature as the salinity of the model cell adjacent to the ice-water interface and at the interface, respectively. Note, that the salinity of the ice shelf is generally neglected ($S_I = 0$). Equations (6.82) to (6.84) can be solved for S_b, T_b, and the freshwater flux q due to melting. These values are substituted into expression (6.81) to obtain the boundary conditions for the temperature and salinity equations of the ocean model.

This formulation tends to yield smaller melt rates than the simpler formulation of the ISOMIP protocol because the freshwater flux due to melting decreases the salinity which raises the freezing point temperature and thus leads to less melting at the interface. For a simpler thermodynamics model where S_b is not computed explicitly, for example as in the ISOMIP protocol, equation (6.81) cannot be applied directly. In this case equation (6.84) can be used with Eq. (6.81) to obtain:

$$\rho K \frac{\partial S}{\partial z}\bigg|_b = q\,(S - S_I). \tag{6.85}$$

This formulation can be used for all cases for which equation (6.84) is valid. Further, in this formulation it is obvious that melting ($q < 0$) leads to a reduction of salinity.

The default value of `SHELFICEconserve=.false.` removes the contribution $q(X_b - X)$ from Eq. (6.81), making the boundary conditions for temperature non-conservative.

ISOMIP-Thermodynamics A simpler formulation follows the ISOMIP protocol (http://efdl.cims.nyu.edu/projec The freezing and melting in the boundary layer between ice shelf and ocean is parameterized following *Grosfeld et al.* [1997]. In this formulation Eq. (6.82) reduces to

$$c_p \rho \gamma_T (T - T_b) = -Lq \tag{6.86}$$

and the fresh water flux q is computed from

$$q = -\frac{c_p \rho \gamma_T (T - T_b)}{L}. \tag{6.87}$$

In order to use this formulation, set run-time parameter `useISOMIPTD=.true.` in data.shelfice.

Remark The shelfice package and experiments demonstrating its strenghts and weaknesses are also described in *Losch* [2008]. However, note that unfortunately the description of the thermodynamics in the appendix of *Losch* [2008] is wrong.

6.6.3.5 Key subroutines

Top-level routine: `shelfice_model.F`

```
C     !CALLING SEQUENCE:
C ...
C |-FORWARD_STEP           :: Step forward a time-step ( AT LAST !!! )
C ...
C | |-DO_OCEANIC_PHY       :: Control oceanic physics and parameterization
C ...
C | | |-SHELFICE_THERMODYNAMICS :: main routine for thermodynamics
C                                  with diagnostics
C ...
C | |-THERMODYNAMICS       :: theta, salt + tracer equations driver.
```

```
C ...
C | | |-EXTERNAL_FORCING_T :: Problem specific forcing for temperature.
C | | |-SHELFICE_FORCING_T :: apply heat fluxes from ice shelf model
C ...
C | | |-EXTERNAL_FORCING_S :: Problem specific forcing for salinity.
C | | |-SHELFICE_FORCING_S :: apply fresh water fluxes from ice shelf model
C ...
C | |-DYNAMICS            :: Momentum equations driver.
C ...
C | | |-MOM_FLUXFORM      :: Flux form mom eqn. package ( see
C ...
C | | | |-SHELFICE_U_DRAG  :: apply drag along ice shelf to u-equation
C                             with diagnostics
C ...
C | | |-MOM_VECINV        :: Vector invariant form mom eqn. package ( see
C ...
C | | | |-SHELFICE_V_DRAG  :: apply drag along ice shelf to v-equation
C                             with diagnostics
C ...
C  o
```

6.6.3.6 SHELFICE diagnostics

Diagnostics output is available via the diagnostics package (see Section 7.1). Available output fields are summarized in Table 6.6.3.6.

6.6.3.7 Experiments and tutorials that use shelfice

- ISOMIP, Experiment 1 (http://efdl.cims.nyu.edu/project_oisi/isomip/overview.html) in isomip verification directory.

6.6.4 STREAMICE Package

Authors: Daniel Goldberg

6.6.4.1 Introduction

Package "streamice" provides a dynamic land ice model for MITgcm.

6.6.4.2 Equations Solved

As of now, the model tracks only 3 variables: x-velocity (u), y-velocity (v), and thickness (h). There is also a variable that tracks coverage of fractional cells, discussed...

By default the model solves the Shallow Shelf approximation (SSA) for velocity. The SSA is appropriate for floating ice (ice shelf) or ice flowing over a low-friction bed (e.g. MacAyeal, 1989). The SSA consists of the x-momentum balance:

$$\partial_x(h\nu(4\dot{\varepsilon}_{xx} + 2\dot{\varepsilon}_{yy})) + \partial_y(2h\nu\dot{\varepsilon}_{xy}) - \tau_{bx} = \rho g h s_x \tag{6.88}$$

the y-momentum balance:

$$\partial_x(2h\nu\dot{\varepsilon}_{xy}) + \partial_y(h\nu(4\dot{\varepsilon}_{yy} + 2\dot{\varepsilon}_{xx})) - \tau_{by} = \rho g h s_y. \tag{6.89}$$

From the velocity field, thickness evolves according to the continuity equation:

$$h_t + \nabla \cdot (h\vec{u}) = -\dot{b}, \tag{6.90}$$

Where $\dot{b}$ is a basal mass balance (e.g. melting due to contact with the ocean), positive where there is melting. Surface mass balance is not considered, since `pkg/streamice` is intended for localized areas (tens to hundreds of kilometers) over which the integrated effect of surface mass balance is generally small. Where ice is grounded, surface elevation is given by

$$s = R + h, \tag{6.91}$$

where $R(x, y)$ is the bathymetry, and the basal elevation b is equal to R. If ice is floating, then the assumption of hydrostasy and constant density gives

$$s = (1 - \frac{\rho}{\rho_w}h, \tag{6.92}$$

where ρ_w is a representative ocean density, and $b = -(\rho/\rho_w)h$. Again by hydrostasy, floation is assumed wherever

$$h \leq -\frac{\rho_w}{\rho}R \tag{6.93}$$

is satisfied. Floatation criteria is stored in `float_frac_streamice`, equal to 1 where ice is at floatation.

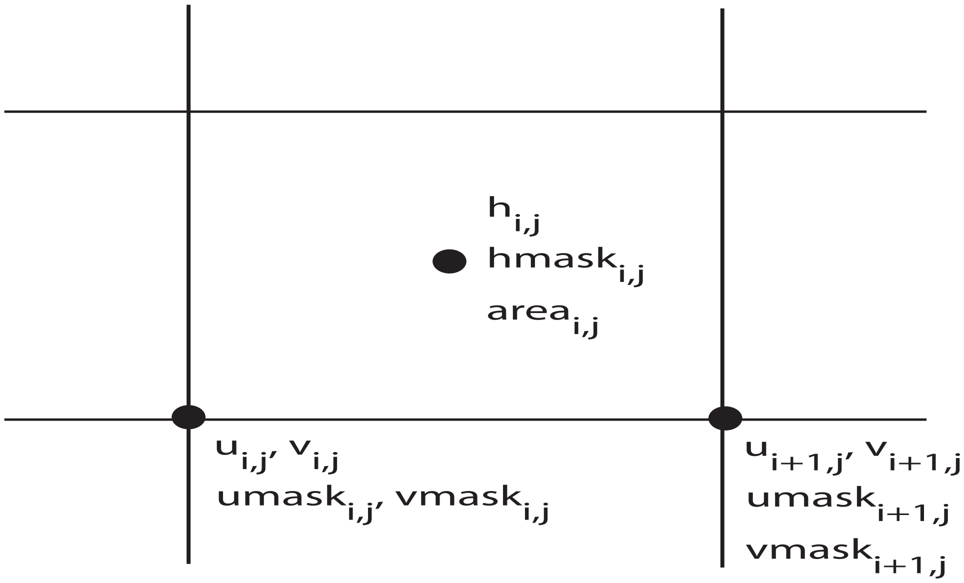
The strain rates ε_{ij} are generalized to the case of orthogonal curvilinear coordinates, to include the "metric" terms that arise when casting the equations of motion on a sphere or projection on to a sphere (see `pkg/SEAICE`, 6.6.2.4.8 of the MITgcm documentation). Thus

$$\begin{aligned}
\dot{\varepsilon}_{xx} =& u_x + k_1 v, \\
\dot{\varepsilon}_{yy} =& v_y + k_1 u, \\
\dot{\varepsilon}_{xy} =& \frac{1}{2}(u_y + v_x) + k_1 u + k_2 v.
\end{aligned}$$

ν has the form arising from Glen's law

$$\nu = \frac{1}{2}A^{-\frac{1}{n}}\left(\dot{\varepsilon}_{xx}^2 + \dot{\varepsilon}_{yy}^2 + \dot{\varepsilon}_{xx}\dot{\varepsilon}_{yy} + \dot{\varepsilon}_{xy}^2 + \dot{\varepsilon}_{min}^2\right)^{\frac{1-n}{2n}}, \tag{6.94}$$

though the form is slightly different if a hybrid formulation is used. Whether τ_b is nonzero depends on whether the floatation condition is satisfied. Currently this is determined simply on an instantaneous



cell-by-cell basis (unless subgrid interpolation is used), as is the surface elevation s, but possibly this should be rethought if the effects of tides are to be considered. $\vec{\tau}_b$ has the form

$$\vec{\tau}_b = C(|\vec{u}|^2 + u_{min}^2)^{\frac{m-1}{2}}\vec{u}. \tag{6.95}$$

Again, the form is slightly different if a hybrid formulation is to be used. The scalar term multiplying $\vec{u}$ is referred to as β below.

The momentum equations are solved together with appropriate boundary conditions, discussed below. In the case of a calving front boundary condition (CFBC), the boundary condition has the following form:

$$(h\nu(4\dot{\varepsilon}_{xx} + 2\dot{\varepsilon}_{yy}))n_x + (2h\nu\dot{\varepsilon}_{xy})n_y = \frac{1}{2}g\left(\rho h^2 - \rho_w b^2\right)n_x \tag{6.96}$$

$$(2h\nu\dot{\varepsilon}_{xy})n_x + (h\nu(4\dot{\varepsilon}_{yy} + 2\dot{\varepsilon}_{xx}))n_y = \frac{1}{2}g\left(\rho h^2 - \rho_w b^2\right)n_y. \tag{6.97}$$

Here $\vec{n}$ is the normal to the boundary, and $R(x, y)$ is the bathymetry.

Hybrid SIA-SSA stress balance The SSA does not take vertical shear stress or strain rates (e.g., σ_{xz}, $\partial u/\partial z$) into account. Although there are other terms in the stress tensor, studies have found that in all but a few cases, vertical shear and longitudinal stresses (represented by the SSA) are sufficient to represent glaciological flow. `streamice` can allow for representation of vertical shear, although the approximation is made that longitudinal stresses are depth-independent. The stress balance is referred to as "hybrid" because it is a joining of the SSA and the Shallow Ice Approximation (SIA), which only accounts only for vertical shear. Such hybrid formulations have been shown to be valid over a larger range of conditions than SSA (*Schoof and Hindmarsh* 2010, *Goldberg* 2011).

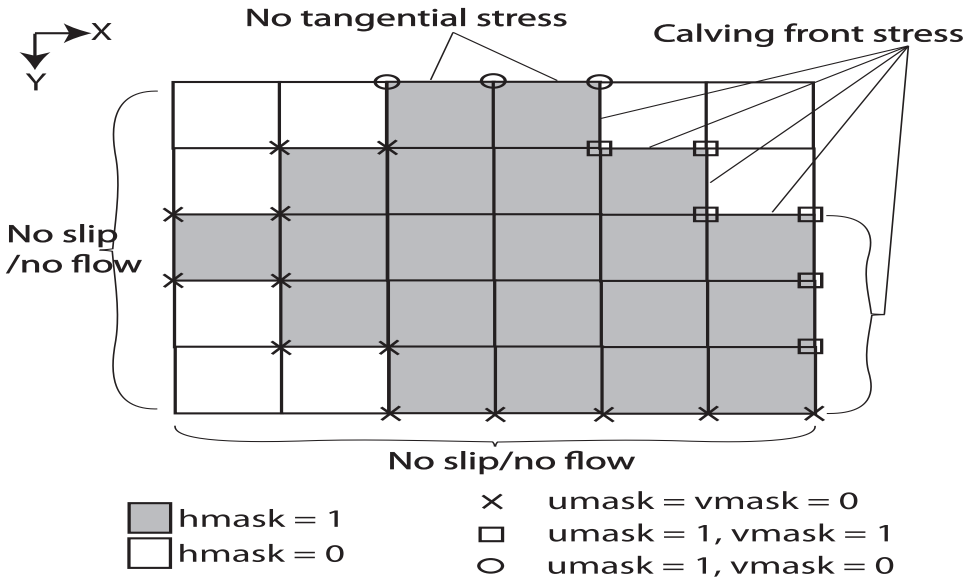
In the hybrid formulation, $\overline{u}$ and $\overline{v}$, the depth-averaged $x-$ and $y-$ velocities, replace u and v in (6.88), (6.89), and (6.90), and gradients such as u_x are replaced by $(\overline{u})_x$. Viscosity becomes

$$\nu = \frac{1}{2}A^{-\frac{1}{n}}\left(\dot{\varepsilon}_{xx}^2 + \dot{\varepsilon}_{yy}^2 + \dot{\varepsilon}_{xx}\dot{\varepsilon}_{yy} + \dot{\varepsilon}_{xy}^2 + \frac{1}{4}u_z^2 + \frac{1}{4}v_z^2 + \dot{\varepsilon}_{min}^2\right)^{\frac{1-n}{2n}}. \tag{6.98}$$

In the formulation for τ_b, u_b, the horizontal velocity at u_b is used instead. The details are given in *Goldberg* (2011).

6.6.4.3 Numerical Scheme

Stress/momentum equations The stress balance is solved for velocities using a very straightforward, structured-grid finite element method. Generally a finite element method is used to deal with irregularly



shaped domains, or to deal with complicated boundary conditions; in this case it is the latter, as explained below. (NOTE: this is not meant as a finite element tutorial, and various mathematical objects are defined nonrigorously. It is simply meant as an overview of the numerical scheme used.) In this method, the numerical velocities u and v (or $\overline{u}$, $\overline{v}$ if a hybrid formulation) have a bilinear shape within each cell. For instance, referring to Fig. 6.6.4.3, at a point (x, y) within cell (i, j), and assuming a rectangular mesh, the $x-$velocity u would be given by

$$u(x,y) = (1-\zeta_1)\times(1-\zeta_2)\times u_{i,j} + \tag{6.99}$$
$$(\zeta_1)\times(1-\zeta_2)\times u_{i+1,j} + \tag{6.100}$$
$$(1-\zeta_1)\times(\zeta_2)\times u_{i,j+1} + \tag{6.101}$$
$$(\zeta_1)\times(\zeta_2)\times u_{i+1,j+1}, \tag{6.102}$$

where

$$\zeta_1 = x - x_i, \quad \zeta_2 = y - y_j \tag{6.103}$$

and *e.g.* $u_{i,j+1}$ is the nodal value of u at the top right corner of the cell. In finite element terms, the functions multiplying a nodal value of u are the *shape functions* ϕ corresponding to that node, e.g. $\phi_{ij} = \zeta_1\zeta_2$ in cell (i, j). Note that h_{ij} is defined at the center of each cell. In the code velocities are `U_STREAMICE` and `V_STREAMICE`, and thickness is `H_STREAMICE`.

If we take a vector-valued function $\Phi = (\phi, \psi)$ which is in the "solution space" of (u, v) (i.e. ϕ and ψ are bilinear in each cell as defined above, and are zero where velocities are imposed), take the dot product with by (6.88,6.89), and integrate by parts over the domain D, we get the weak form of the momentum equation:

$$-\int_D \left(h\nu\phi_x(4\dot{\varepsilon}_{xx}+2\dot{\varepsilon}_{yy}) + h\nu\phi_y\dot{\varepsilon}_{xy} + \beta(u\phi+v\psi) + h\nu\psi_x\dot{\varepsilon}_{xy} + h\nu\psi_y(4\dot{\varepsilon}_{yy}+2\dot{\varepsilon}_{xx})\right)dA =$$
$$\int_D \Phi\vec{\tau}_d - \int_\Gamma \frac{1}{2}g\left(\rho h^2 - \rho_w R^2\right)\Phi\cdot\vec{n}dS, \tag{6.104}$$

where Γ is the part of the domain boundary where a calving front stress condition is imposed, and $\vec{\tau}_d$ is the driving stress $\rho g h \nabla s$. Note that the boundary integral does not involve viscosity, or any velocity-dependent terms. This is the advantage of using a finite-element formulation: viscosity need not be represented at boundaries, and so no complicated one-sided derivative expressions need to be used.

Let (U, V) be the vector of all nodal values $u_{i,j}, v_{i,j}$ that determine (u, v). If we assume that ν and β are independent of (U, V), then for any (ϕ, ψ), the left hand side of (6.104) is a linear functional of (U, V), i.e. a linear operator that returns a scalar. We can choose ϕ and ψ so that they are nonzero only at a single node; and we can evaluate (6.104) for each such ϕ_{ij} and ψ_{ij}, giving a linearly independent set of equations for U and V. The left hand side of (6.104) for each ϕ_{ij} and ψ_{ij} is evaluated in the subroutine `STREAMICE_CG_ACTION()` in the source file `streamice_cg_functions.f`. The right hand side is evaluated in `STREAMICE_DRIVING_STRESS()`. The latter is not a straightforward calculation, since the thickness h and the surface s are not expressed by piecewise bilinear functions, and is discussed below.

`STREAMICE_CG_ACTION()` represents the *action* of a matrix on the vector represented by (U, V) and `STREAMICE_DRIVING_STRESS()` gives the right hand side of a linear system of equations. This information is enough to solve the linear system, and this is done with the method of conjugate gradients, implemented (with simple Jacobi preconditioning) in `STREAMICE_CG_SOLVE()`.

The full nonlinear system is solved in `STREAMICE_VEL_SOLVE()`. This subroutine evaluates driving stress and then calls a loop in which a sequence of linear systems is solved. After each linear solve, the viscosity ν (the array `visc_streamice`) and basal traction β (`tau_beta_eff_streamice`) is updated from the new iterate of u and v. The iteration continues until a desired tolerance is reached or the maximum number of iterations is reached.

Rather than calling `STREAMICE_CG_ACTION()` each iteration of the conjugate gradient, the sparse matrix can be constructed explicitly. This is done if `STREAMICE_CONSTRUCT_MATRIX` is `#define`d in `STREAMICE_OPTIONS.h`. This speeds up the solution because `STREAMICE_CG_ACTION()` contains a number of nested loops related to quadrature points and interactions between nodes of a cell.

The driving stress $\tau_d = \rho g \nabla s$ is not as straightforward to deal with in the weak formulation, as h and s are considered constant in a cell. Furthermore, in the ice shelf the driving stress can be written as

$$\vec{\tau}_d = \frac{1}{2}\rho g\left(1 - \frac{\rho}{\rho_w}\right)\nabla(h^2). \tag{6.105}$$

Among other things, this (along with equations 6.96 and 6.97) implies that for an unconfined shelf (one for which the only boundaries are a calving front and a grounding line), the stress state along the grounding line depends only on the location at depth of the grounding line. The numerical representation of velocities must respect this. Thus the driving stress contribution in the $x-$ momentum equation at node (i, j) is given as follows.

$$\tau_{d,x}(i,j) = \begin{cases} \frac{1}{4}\rho g(\gamma_{ij}h_{ij} + \gamma_{i-1,j}h_{i-1,j})(\gamma_{i-1,j}s_{i-1,j} - \gamma_{ij}s_{ij}) & \text{case I} \\ \frac{1}{4}\frac{rA_{i-1,j}}{\Delta x_{f,i-1,j}} g(\rho h_{i-1,j}^2 - \rho_w b_{i-1,j}^2) & \text{case II} \\ -\frac{1}{4}\frac{rA_{i,j}}{\Delta x_{f,ij}} g(\rho h_{ij}^2 - \rho_w b_{ij}^2) & \text{case III.} \end{cases} \tag{6.106}$$

where

$$\gamma_{ij} = \sqrt{\frac{rA_{ij}}{\Delta x_{f,ij}}}, \tag{6.107}$$

rA_{ij} is the area of cell (i, j) and $\Delta x_{f,ij}$ is the width of the cell at its $y-$midpoint. Cases I, II, and III refer to the situations when (I) both cells (i, j) and $(i-1, j)$ are in the domain, (II) only cell $(i-1, j)$ is in the domain, and (III) only cell (i, j) is in the domain. (The fact that $\tau_{d,x}(i, j)$ is being calculated means that the implied boundary is a calving stress boundary, as explained below.) The above expression gives the contribution from the cells above node (i, j) (in the $y-$direction). A similar expression gives the contribution from cells $(i-1, j-1)$ and $(i, j-1)$, and corresponding expressions approximate driving stress in the $y-$ momentum equation. In the ice shelf, the case I expressions give approximations for (6.105) which are discretely integrable; in grounded ice the expression need not be integrable, so any errors associated with varying grid metrics can be subsumed into the discretization error.

Orthogonal curvilinear coordinates If the grid is not rectangular, but given in orthogonal curvilinear coordinates, the evaluation of (6.104) is not exact, since the gradient of the basis functions are approximated, as we do not have complete knowledge of the coordinate system, only the grid metrics. Still, we consider nodal basis functions that are equal to 1 at a single node, and zero elsewhere in a cell.

A cell (i, j) has separation $\Delta x_{i,j}$ at the bottom and $\Delta x_{i,j+1}$ at the top (referred to as `dxG(i,j)` and `dxG(i,j+1)` in the code). So, for instance, for a cell given by $[\xi_1, \xi_2] \times [\eta_1, \eta_2]$, the ξ-derivative of the basis function centered at (ξ_1, η_1) is approximated by

$$\left(\frac{1-q}{\Delta x_{i,j}}\right) + \left(\frac{q}{\Delta x_{i,j+1}}\right) \tag{6.108}$$

where q is the y-coordinate of the quadrature point being used. (The code currently uses 2-point gauss quadrature to evaluate the integrals in (6.104); with a rectangular grid and cellwise-constant β and ν, this is exact.)

Basis function derivatives and quadrature weights are stored in `Dphi` and `grid_jacq_streamice`. Both are initialized in `STREAMICE_INIT_PHI`, called only once.

Boundary conditions and masks The computational domain (which may be smaller than the array/grid as defined by `SIZE.h` and `GRID.h`) is determined by a number of mask arrays within the `STREAMICE` package. They are

- $hmask$ (`STREAMICE_hmask`): equal to 1 (ice-covered), 0 (open ocean), 2 (partly-covered), or -1 (out of domain)
- $umask$ (`STREAMICE_umask`): equal to 1 (an "active" velocity node), 3 (a Dirichlet node), or 0 (zero velocity)
- $vmask$ (`STREAMICE_vmask`): similar to umask
- $ufacemaskbdry$ (`STREAMICE_ufacemaskbdry`): equal to -1 (interior face), 0 (no-slip), 1 (no-stress), 2 (calving stress front), 3 (Dirichlet), or 4 (flux input boundary); when 4, then `u_flux_bdry_SI` must be initialized, through binary or parameter file
- $vfacemaskbdry$ (`STREAMICE_vfacemaskbdry`): similar to ufacemaskbdry

$hmask$ is defined at cell centers, like h. $umask$ and $vmask$ are defined at cell nodes, like velocities. $ufacemaskbdry$ and $vfacemaskbdry$ are defined at cell faces, like velocities in a C-grid - but unless `STREAMICE_GEOM_FILE_SETUP` is `#define`d in `STREAMICE_OPTIONS.h`, the values are only relevant at the boundaries of the grid.

Essentially $hmask$ controls the domain; it is specified at initialization and does not change unless calving front advance is allowed. $ufacemaskbdry$ and $vfacemaskbdry$ are defined at initialization as well. Masks are either initialized through binary files or through the text file `data.streamice` (see the streamice tutorial)

The values of $umask$ and $vmask$ determine which nodal values of u and v are involved in the solve for velocities. Before each call to `STREAMICE_VEL_SOLVE()`, $umask$ and $vmask$ are re-initialized. Values are only relevant if they border a cell where $hmask = 1$. Furthermore, if a cell face is a "no-slip" face, both $umask$ and $vmask$ are set to zero at the face's nodal endpoints. If the face is a Dirichlet boundary, both nodal endpoints are set to 3. If $ufacemaskbdry = 1$ on the $x-$boundary of a cell, i.e. it is a no-stress boundary, then $vmask$ is set to 1 at both endpoints while $umask$ is set to zero (i.e. no normal flow). If the face is a flux boundary, velocities are set to zero (see ???). For a calving stress front, $umask$ and $vmask$ are 1: the nodes represent active degrees of freedom.

With $umask$ and $vmask$ appropriately initialized, `STREAMICE_VEL_SOLVE` can proceed rather generally. Contributions to (6.104) are only evaluated if $hmask = 1$ in a given cell, and a given nodal basis function is only considered if $umask = 1$ or $vmask = 1$ at that node.

Thickness evolution (6.90) is solved in the subroutine `STREAMICE_ADVECT_THICKNESS`, similarly to the advection routines in MITgcm. Mass fluxes are evaluated, first in the x-direction. This is done in the generic subroutine `STREAMICE_ADV_FLUX_FL_X`. Flux velocity in the $x-$direction at face (i, j) are generated by averaging $u_{i,j}$ and $u_{i,j+1}$. Assuming the flux velocity is positive, if $hmask_{i-2,j}$, $maski - 1, j$ and $hmask_{i,j}$ are equal to 1, then flux thickness, i.e. the interpolation of h at face (i, j), is through a minmod limiter as in the `generic_advdiff` package. If these values are not available, then a zero-order upwind flux is used. The exception is when `STREAMICE_ufacemask(i,j)` is equal to 4; then `u_flux_bdry_SI(i,j)` is used for the flux. Fluxes are then differenced to update h in cells that are active ($hmask = 1$); a similar procedure follows for the $y-$direction.

Ice front advance Flux out of the domain across Dirichlet boundaries is ignored, and flux at no-flow or no-stress boundaries is zero. However, fluxes that cross calving boundaries are stored, and if `STREAMICE_move_front=.true.`, then `STREAMICE_ADV_FRONT` is called after all cells with $hmask = 1$ have their thickness updated. In this subroutine, cells with $hmask = 0$ or $hmask = 2$ with nonzero fluxes entering their boundaries are processed.

The algorithm is based on *Albrecht* (2011). In this scheme, if cell (i, j) fits the criteria, a **reference** thickness $h_{ref}(i, j)$ is found, defined as an average over the thickness of all neighboring cells with $hmask = 1$ that are contributing mass to (i, j). The total mass input over the time step to (i, j) is calculated as $V_{in}(ij)$. This is added to the volume of ice already in the cell, which is nonzero only if $hmask_{ij} = 2$ at the beginning of the time step, and is equal to

$$V_{cell}(i, j) = area_{ij} h_{ij}. \tag{6.109}$$

(If $hmask_{ij}$ is not equal to 1, then h_{ij} has not yet been updated in the current time step.) $area_{ij}$ is stored in the array `area_shelf_streamice`. The total volume, $V_{tot}(i, j)$, is then equal to $V_{cell}(i, j) + V_{in}(i, j)$. Next an effective thickness

$$h_{eff}(i, j) = \frac{V_{ij}}{rA_{ij}} \tag{6.110}$$

is found. If $h_{eff}(ij) < h_{ref}(i, j)$ (but nonzero) then

- $hmask_{ij}$ is set to 2,
- h_{ij} is set to $h_{ref}(ij)$,
- $area_{ij}$ is set to $\frac{V_{tot}(i,j)}{h_{ref}(i,j)}$.

If, on the other hand, $h_{eff}(ij) \geq h_{ref}(i, j)$,

- $hmask_{ij}$ is set to 1,
- h_{ij} is set to $h_{ref}(ij)$,
- $area_{ij}$ is set to rA_{ij},
- Excess flux $q_{ex}(i, j) = V_{tot}(i, j) - h_{eff}(i, j) rA_{ij}$ is found.

If $q_{ex}(i, j)$ is zero, nothing more needs to happen. If it is positive, then

- adjacent cells that are not completely covered ($hmask = 0$ or 1) are searched for.
- if there are no suitable cells, $q_{ex}(i, j)$ is put back into cell (i, j) (h_{ij} is set to $h_{ref}(i, j) + \frac{q_{ex}(i,j)}{rA_{ij}}$).
- if there are suitable cells, q_{ex} is divided **uniformly** among the corresponding faces of the receiving cells, and the algorithm begins again.

The last step is a recursive one; in theory the recursion could continue indefinitely, but in practice at most one recursive call is done. The decision to spread the excess flux uniformly is not the most physically motivated choice, but with appropriate time steps the excess flux value is expected to be small, and is more a matter of mass/volume conservation. This algorithm, while complicated, is necessary for realistic front advance. If newly-covered cells were given full coverage instead of partial coverage and volume were conserved, the front would quickly diffuse.

If `calve_to_mask=.true.`, this sets a limit to how far the front can advance, even if advance is allowed. The front will not advance into cells where the array `calve_mask` is not equal to 1. This mask must be set through a binary input file.

No calving parameterisation is implemented in `STREAMICE`. However, front advancement is a precursor for such a development to be added.

Surface/basal mass balance After updating thickness in fully- and partially-covered cells, surface and basal mass balances are applied to nonempty cells. Currently surface mass balance is a uniform value, set through the parameter `streamice_adot_uniform`. Basal melt rates are stored in the array `BDOT_streamice`, but are multiplied through by `float_frac_streamice` before being applied.

6.7 Packages Related to Coupled Model

6.7.1 Coupling interface for Atmospheric Intermediate code

6.7.1.1 Key subroutines, parameters and files

6.7.1.2 Experiments and tutorials that use aim_compon_interf

- Global coupled ocean atmosphere experiment in cpl_aim+ocean verification directory.

6.7.2 Coupler for mapping between Atmosphere and ocean

6.7.2.1 Key subroutines, parameters and files

6.7.2.2 Experiments and tutorials that use atm_ocn_coupler

- Global coupled ocean atmosphere experiment in cpl_aim+ocean verification directory.

6.7.3 Toolkit for building couplers

6.7.3.1 Key subroutines, parameters and files

6.7.3.2 Experiments and tutorials that use component_communications

- Global coupled ocean atmosphere experiment in cpl_aim+ocean verification directory.

6.8 Biogeochemistry Packages

6.8.1 GCHEM Package

6.8.1.1 Introduction

This package has been developed as interface to the PTRACERS package. The purpose is to provide a structure where various (any) tracer experiments can be added to the code. For instance there are placeholders for routines to read in parameters needed for any tracer experiments, a routine to read in extra fields required for the tracer code, routines for either external forcing or internal interactions between tracers and routines for additional diagnostics relating to the tracers. Note that the gchem package itself is only a means to call the subroutines used by specific biogeochemical experiments, and does not "do" anything on its own.

There are two examples: **cfc** which looks at 2 tracers with a simple external forcing and **dic** with 4,5 or 6 tracers whose tendency terms are related to one another. We will discuss these here only as how they provide examples to use this package.

6.8.1.2 Key subroutines and parameters

FRAMEWORK
The different biogeochemistry frameworks (e.g. cfc of dic) are specified in the packages_conf file. *GCHEM_OPTIONS.h* includes the compiler options to be used in any experiment. An important compiler option is #define GCHEM_SEPARATE_FORCING which determined how and when the tracer forcing is applied (see discussion on Forcing below). See section on dic for some additional flags that can be set for that experiment.

There are further runtime parameters set in *data.gchem* and kept in common block *GCHEM.h*. These runtime options include:
• Parameters to set the timing for periodic forcing files to be loaded are: *gchem_ForcingPeriod*, *gchem_ForcingCycle*. The former is how often to load, the latter is how often to cycle through those fields (eg. period couple be monthly and cycle one year). This is used in *dic* and *cfc*, with gchem_ForcingPeriod=0 meaning no periodic forcing.
• **nsubtime** is the integer number of extra timesteps required by the tracer experiment. This will give a timestep of **deltaTtracer/nsubtime** for the dependencies between tracers. The default is one.
• File names - these are several filenames than can be read in for external fields needed in the tracer forcing - for instance wind speed is needed in both DIC and CFC packages to calculate the air-sea exchange of gases. Not all file names will be used for every tracer experiment.
• **gchem_int_*** are variable names for run-time set integer numbers. (Currently 1 through 5).
• **gchem_rl_*** are variable names for run-time set real numbers. (Currently 1 through 5).
• Note that the old **tIter0** has been replaced by **PTRACERS_Iter0** which is set in data.ptracers instead.

INITIALIZATION
The values set at runtime in data.gchem are read in using *gchem_readparms.F* which is called from packages_readparms.F. This will include any external forcing files that will be needed by the tracer experiment.

There are two routine used to initialize parameters and fields needed by the experiment packages. These are *gchem_init_fixed.F* which is called from *packages_init_fixed.F*, and *gchem_init_vari.F* called from packages_init_variable.F. The first should be used to call a subroutine specific to the tracer experiment which sets fixed parameters, the second should call a subroutine specific to the tracer experiment which sets (or initializes) time fields that will vary with time.

LOADING FIELDS
External forcing fields used by the tracer experiment are read in by a subroutine (specific to the tracer experiment) called from *gchem_fields_load.F*. This latter is called from *forward_step.F*.

FORCING
Tracer fields are advected-and-diffused by the ptracer package. Additional changes (e.g. surface forcing

or interactions between tracers) to these fields are taken care of by the gchem interface. For tracers that are essentially passive (e.g. CFC's) but may have some surface boundary conditions this can easily be done within the regular tracer timestep. In this case *gchem_calc_tendency.F* is called from *forward_step.F*, where the reactive (as opposed to the advective diffusive) tendencies are computed. These tendencies, stored on the 3D field **gchemTendency**, are added to the passive tracer tendencies **gPtr** in *gchem_add_tendency.F*, which is called from *ptracers_forcing.F*. For tracers with more complicated dependencies on each other, and especially tracers which require a smaller timestep than deltaTtracer, it will be easier to use *gchem_forcing_sep.F* which is called from forward_step.F. There is a compiler option set in *GCHEM_OPTIONS.h* that determines which method is used: #define GCHEM_SEPARATE_FORCING does the latter where tracers are forced separately from the advection-diffusion code, and #undef GCHEM_SEPARATE_FORCING includes the forcing in the regular timestepping.

DIAGNOSTICS
This package also also used the passive tracer routine *ptracers_monitor.F* which prints out tracer statistics as often as the model dynamic statistic diagnostics (dynsys) are written (or as prescribed by the runtime flag **PTRACERS_monitorFreq**, set in *data.ptracers*). There is also a placeholder for any tracer experiment specific diagnostics to be calculated and printed to files. This is done in *gchem_diags.F*. For instance the time average CO2 air-sea fluxes, and sea surface pH (among others) are written out by *dic_biotic_diags.F* which is called from *gchem_diags.F*.

6.8.1.3 GCHEM Diagnostics

These diagnostics are particularly for the **dic** package.

```
------------------------------------------------------------------------
<-Name->|Levs|<-parsing code->|<--  Units   -->|<- Tile (max=80c)
------------------------------------------------------------------------
DICBIOA | 15 |SM P    MR      |mol/m3/sec      |Biological Productivity (mol/m3/s)
DICCARB | 15 |SM P    MR      |mol eq/m3/sec   |Carbonate chg-biol prod and remin (mol eq/m3/s)
DICTFLX |  1 |SM P    L1      |mol/m3/sec      |Tendency of DIC due to air-sea exch (mol/m3/s)
DICOFLX |  1 |SM P    L1      |mol/m3/sec      |Tendency of O2 due to air-sea exch (mol/m3/s)
DICCFLX |  1 |SM P    L1      |mol/m2/sec      |Flux of CO2 - air-sea exch (mol/m2/s)
DICPCO2 |  1 |SM P    M1      |atm             |Partial Pressure of CO2 (atm)
DICPHAV |  1 |SM P    M1      |dimensionless   |pH (dimensionless)
```

6.8.1.4 Do's and Don'ts

The pkg ptracer is required with use with this pkg. Also, as usual, the runtime flag **useGCHEM** must be set to **.TRUE.** in **data.pkg**. By itself, gchem pkg will read in **data.gchem** and will write out gchem diagnostics. It requires tracer experiment specific calls to do anything else (for instance the calls to dic and cfc pkgs).

6.8.1.5 Reference Material

6.8.1.6 Experiments and tutorials that use gchem

- Global Ocean biogeochemical tutorial, in tutorial_global_oce_biogeo verification directory, described in section 3.17 uses gchem and dic
- Global Ocean cfc tutorial, in tutorial_cfc_offline verification directory, uses gchem and cfc (and offline) described in 3.20.5
- Global Ocean online cfc example in cfc_example verification directory, uses gchem and cfc

6.8.2 DIC Package

6.8.2.1 Introduction

This is one of the biogeochemical packages handled from the pkg gchem. The main purpose of this package is to consider the cycling of carbon in the ocean. It also looks at the cycling of phosphorous and potentially oxygen and iron. There are four standard tracers DIC, ALK, $PO4$, DOP and also possibly $O2$ and Fe. The air-sea exchange of CO_2 and O_2 are handled as in the OCMIP experiments (reference). The export of biological matter is computed as a function of available light and PO_4 (and Fe). This export is remineralized at depth according to a Martin curve (again, this is the same as in the OCMIP experiments). There is also a representation of the carbonate flux handled as in the OCMIP experiments. The air-sea exchange on CO_2 is affected by temperature, salinity and the pH of the surface waters. The pH is determined following the method of Follows et al. For more details of the equations see section 3.17.

6.8.2.2 Key subroutines and parameters

INITIALIZATION
DIC_ABIOTIC.h contains the common block for the parameters and fields needed to calculate the air-sea flux of CO_2 and O_2. The fixed parameters are set in *dic_abiotic_param* which is called from *gchem_init_fixed.F*. The parameters needed for the biotic part of the calculations are initialized in *dic_biotic_param* and are stored in *DIC_BIOTIC.h*. The first guess of pH is calculated in *dic_surfforcing_init.F*.

LOADING FIELDS
The air-sea exchange of CO_2 and O_2 need wind, atmospheric pressure (although the current version has this hardwired to 1), and sea-ice coverage. The calculation of pH needs silica fields. These fields are read in in *dic_fields_load.F*. These fields are initialized to zero in *dic_ini_forcing.F*. The fields for interpolating are in common block in *DIC_LOAD.h*.

FORCING
The tracers are advected-diffused in *ptracers_integrate.F*. The updated tracers are passed to *dic_biotic_forcing.F* where the effects of the air-sea exchange and biological activity and remineralization are calculated and the tracers are updated for a second time. Below we discuss the subroutines called from *dic_biotic_forcing.F*.

Air-sea exchange of CO_2 is calculated in *dic_surfforcing*. Air-Sea Exchange of CO_2 depends on T,S and pH. The determination of pH is done in *carbon_chem.F*. There are three subroutines in this file: *carbon_coeffs* which determines the coefficients for the carbon chemistry equations; *calc_pco2* which calculates the pH using a Newton-Raphson method; and *calc_pco2_approx* which uses the much more efficient method of Follows et al. The latter is hard-wired into this package, the former is kept here for completeness.

Biological productivity is determined following Dutkiewicz et al. (2005) and is calculated in *bio_export.F* The light in each latitude band is calculate in *insol.F*, unless using one of the flags listed below. The formation of hard tissue (carbonate) is linked to the biological productivity and has an effect on the alkalinity - the flux of carbonate is calculated in *car_flux.F*, unless using the flag listed below for the Friis et al (2006) scheme. The flux of phosphate to depth where it instantly remineralized is calculated in *phos_flux.F*.

The dilution or concentration of carbon and alkalinity by the addition or subtraction of freshwater is important to their surface patterns. These "virtual" fluxes can be calculated by the model in several ways. The older scheme is done following OCMIP protocols (see more in Dutkiewicz et al 2005), in the subroutines *dic_surfforcing.F* and *alk_surfforcing.F*. To use this you need to set in GCHEM_OPTIONS.h:
#define ALLOW_OLD_VIRTUALFLUX
But this can also be done by the ptracers pkg if this is undefined. You will then need to set the concentration of the tracer in rainwater and potentially a reference tracer value in data.ptracer (PTRACERS_EvPrRn, and PTRACERS_ref respectively).

Oxygen air-sea exchange is calculated in *o2_surfforcing.F*.

Iron chemistry (the amount of free iron) is taken care of in *fe_chem.F*.

DIAGNOSTICS

Averages of air-sea exchanges, biological productivity, carbonate activity and pH are calculated. These are initialized to zero in *dic_biotic_init* and are stored in common block in *DIC_BIOTIC.h*.

COMPILE TIME FLAGS
These are set in GCHEM_OPTIONS.h:

DIC_BIOTIC: needs to be set for dic to work properly (should be fixed sometime).
ALLOW_O2: include the tracer oxygen.
ALLOW_FE: include the tracer iron. Note you will need an iron dust file set in data.gchem in this case.
MINFE: limit the iron, assuming precpitation of any excess free iron.
CAR_DISS: use the calcium carbonate scheme of Friis et al 2006.
ALLOW_OLD_VIRTUALFLUX: use the old OCMIP style virtual flux for alklinity adn carbon (rather than doing it through pkg/ptracers).
READ_PAR: read the light (photosynthetically available radiation) from a file set in data.gchem.
USE_QSW: use the numbers from QSW to be the PAR. Note that a file for Qsw must be supplied in data, or Qsw must be supplied by an atmospheric model.
If the above two flags are not set, the model calculates PAR in insol.F as a function of latitude and year day.
USE_QSW_UNDERICE: if using a sea ice model, or if the Qsw variable has the seaice fraction already taken into account, this flag must be set.

AD_SAFE: will use a tanh function instead of a max function - this is better if using the adjoint
DIC_NO_NEG: will include some failsafes in case any of the variables become negative. (This is advicable).
ALLOW_DIC_COST: was used for calculating cost function (but hasn't been updated or maintained, so not sure if it works still)

6.8.2.3 Do's and Don'ts

This package must be run with both ptracers and gchem enabled. It is set up for at least 4 tracers, but there is the provision for oxygen and iron. Note the flags above.

6.8.2.4 Reference Material

Dutkiewicz. S., A. Sokolov, J. Scott and P. Stone, 2005: A Three-Dimensional Ocean-Seaice-Carbon Cycle Model and its Coupling to a Two-Dimensional Atmospheric Model: Uses in Climate Change Studies, Report 122, Joint Program of the Science and Policy of Global Change, M.I.T., Cambridge, MA.

Follows, M., T. Ito and S. Dutkiewicz, 2006: A Compact and Accurate Carbonate Chemistry Solver for Ocean Biogeochemistry Models. *Ocean Modeling*, 12, 290-301.

Friis, K., R. Najjar, M.J. Follows, and S. Dutkiewicz, 2006: Possible overestimation of shallow-depth calcium carbonate dissolution in the ocean, *Global Biogeochemical Cycles*, 20, GB4019, doi:10.1029/2006GB002727.

6.8.2.5 Experiments and tutorials that use dic

- Global Ocean tutorial, in tutorial_global_oce_biogeo verification directory, described in section 3.17

```
---------+----------+----------------+-----------------
 <-Name->|<- grid ->|<--  Units   -->|<- Tile (max=80c)
---------+----------+----------------+-----------------
 sIceLoad|SM      U1|kg/m^2          |sea-ice loading (in Mass of ice+snow / area unit)
---
SEA ICE STATE:
---
 SIarea  |SM      M1|m^2/m^2         |SEAICE fractional ice-covered area [0 to 1]
 SIheff  |SM      M1|m               |SEAICE effective ice thickness
 SIhsnow |SM      M1|m               |SEAICE effective snow thickness
 SIhsalt |SM      M1|g/m^2           |SEAICE effective salinity
 SIuice  |UU      M1|m/s             |SEAICE zonal ice velocity, >0 from West to East
 SIvice  |VV      M1|m/s             |SEAICE merid. ice velocity, >0 from South to North
---
ATMOSPHERIC STATE AS SEEN BY SEA ICE:
---
 SItices |SM  C   M1|K               |Surface Temperature over Sea-Ice (area weighted)
 SIuwind |UM      U1|m/s             |SEAICE zonal 10-m wind speed, >0 increases uVel
 SIvwind |VM      U1|m/s             |SEAICE meridional 10-m wind speed, >0 increases uVel
 SIsnPrcp|SM      U1|kg/m^2/s        |Snow precip. (+=dw) over Sea-Ice (area weighted)
---
FLUXES ACROSS ICE-OCEAN INTERFACE (ATMOS to OCEAN FOR ICE-FREE REGIONS):
---
 SIfu    |UU      U1|N/m^2           |SEAICE zonal surface wind stress, >0 increases uVel
 SIfv    |VV      U1|N/m^2           |SEAICE merid. surface wind stress, >0 increases vVel
 SIqnet  |SM      U1|W/m^2           |Ocean surface heatflux, turb+rad, >0 decreases theta
 SIqsw   |SM      U1|W/m^2           |Ocean surface shortwave radiat., >0 decreases theta
 SIempmr |SM      U1|kg/m^2/s        |Ocean surface freshwater flux, > 0 increases salt
 SIqneto |SM      U1|W/m^2           |Open Ocean Part of SIqnet, turb+rad, >0 decr theta
 SIqneti |SM      U1|W/m^2           |Ice Covered Part of SIqnet, turb+rad, >0 decr theta
---
FLUXES ACROSS ATMOSPHERE-ICE INTERFACE (ATMOS to OCEAN FOR ICE-FREE REGIONS):
---
 SIatmQnt|SM      U1|W/m^2           |Net atmospheric heat flux, >0 decreases theta
 SIatmFW |SM      U1|kg/m^2/s        |Net freshwater flux from atmosphere & land (+=down)
 SIfwSubl|SM      U1|kg/m^2/s        |Freshwater flux of sublimated ice, >0 decreases ice
---
THERMODYNAMIC DIAGNOSTICS:
---
 SIareaPR|SM      M1|m^2/m^2         |SIarea preceeding ridging process
 SIareaPT|SM      M1|m^2/m^2         |SIarea preceeding thermodynamic growth/melt
 SIheffPT|SM      M1|m               |SIheff preceeeding thermodynamic growth/melt
 SIhsnoPT|SM      M1|m               |SIhsnow preceeeding thermodynamic growth/melt
 SIaQbOCN|SM      M1|m/s             |Potential HEFF rate of change by ocean ice flux
 SIaQbATC|SM      M1|m/s             |Potential HEFF rate of change by atm flux over ice
 SIaQbATO|SM      M1|m/s             |Potential HEFF rate of change by open ocn atm flux
 SIdHbOCN|SM      M1|m/s             |HEFF rate of change by ocean ice flux
 SIdSbATC|SM      M1|m/s             |HSNOW rate of change by atm flux over sea ice
 SIdSbOCN|SM      M1|m/s             |HSNOW rate of change by ocean ice flux
 SIdHbATC|SM      M1|m/s             |HEFF rate of change by atm flux over sea ice
 SIdHbATO|SM      M1|m/s             |HEFF rate of change by open ocn atm flux
 SIdHbFLO|SM      M1|m/s             |HEFF rate of change by flooding snow
 SIdAbATO|SM      M1|m^2/m^2/s       |Potential AREA rate of change by open ocn atm flux
 SIdAbATC|SM      M1|m^2/m^2/s       |Potential AREA rate of change by atm flux over ice
 SIdAbOCN|SM      M1|m^2/m^2/s       |Potential AREA rate of change by ocean ice flux
 SIdA    |SM      M1|m^2/m^2/s       |AREA rate of change (net)
---
DYNAMIC/RHEOLOGY DIAGNOSTICS:
---
 SIpress |SM      M1|m^2/s^2         |SEAICE strength (with upper and lower limit)
 SIzeta  |SM      M1|m^2/s           |SEAICE nonlinear bulk viscosity
 SIeta   |SM      M1|m^2/s           |SEAICE nonlinear shear viscosity
 SIsigI  |SM      M1|no units        |SEAICE normalized principle stress, component one
 SIsigII |SM      M1|no units        |SEAICE normalized principle stress, component two
---
ADVECTIVE/DIFFUSIVE FLUXES OF SEA ICE variables:
---
 ADVxHEFF|UU      M1|m.m^2/s         |Zonal      Advective Flux of eff ice thickn
 ADVyHEFF|VV      M1|m.m^2/s         |Meridional Advective Flux of eff ice thickn
 SIuheff |UU      M1|m^2/s           |Zonal      Transport of eff ice thickn (centered)
 SIvheff |VV      M1|m^2/s           |Meridional Transport of eff ice thickn (centered)
 DFxEHEFF|UU      M1|m^2/s           |Zonal      Diffusive Flux of eff ice thickn
 DFyEHEFF|VV      M1|m^2/s           |Meridional Diffusive Flux of eff ice thickn
 ADVxAREA|UU      M1|m^2/m^2.m^2/s   |Zonal      Advective Flux of fract area
 ADVyAREA|VV      M1|m^2/m^2.m^2/s   |Meridional Advective Flux of fract area
```

CPP option	Description
ALLOW_SHELFICE_DEBUG	Include code for enhanged diagnosis
ALLOW_ISOMIP_TD	Include code for simplifed ISOMIP thermodynamics

Table 6.21: Available CPP-flags to be set in SHELFICE_OPTIONS.h

Table 6.22: Run-time parameters and default values

Name	Default value	Description	Reference
useISOMIPTD	F	use simplified ISOMIP thermodynamics instead of default	
SHELFICEconserve	F	use conservative form of temperature boundary conditions	
SHELFICEboundaryLayer	F	use simple boundary layer mixing parameterization	
SHELFICEloadAnomalyFile	UNSET	inital geopotential anomaly	
SHELFICEtopoFile	UNSET	under-ice topography of ice shelves	
SHELFICElatentHeat	334.0E+03	latent heat of fusion (L)	
SHELFICEHeatCapacity_Cp	2000.0E+00	specific heat capacity of ice ($c_{p,I}$)	
rhoShelfIce	917.0E+00	(constant) mean density of ice shelf (ρ_I)	
SHELFICEheatTransCoeff	1.0E-04	transfer coefficient (exchange velocity) for temperature (γ_T)	
SHELFICEsaltTransCoeff	5.05E-03 × SHELFICEheatTransCoeff	transfer coefficient (exchange velocity) for salinity (γ_S)	
SHELFICEkappa	1.54E-06	temperature diffusion coefficient of the ice shelf ($kappa$)	
SHELFICEthetaSurface	-20.0E+00	(constant) surface temperature above the ice shelf (T_S)	
no_slip_shelfice	no_slip_bottom (data)	flag for slip along bottom of ice shelf	
SHELFICEDragLinear	bottomDragLinear (data)	linear drag coefficient at bottom ice shelf	
SHELFICEDragQuadratic	bottomDragQuadratic (data)	quadratic drag coefficient at bottom ice shelf	
SHELFICEwriteState	F	write ice shelf state to file	
SHELFICE_dumpFreq	dumpFreq (data)	dump frequency	
SHELFICE_taveFreq	taveFreq (data)	time-averaging frequency	
SHELFICE_tave_mnc	timeave_mnc (data.mnc)	write snap-shot using MNC	
SHELFICE_dump_mnc	snapshot_mnc (data.mnc)	write TimeAverage using MNC	

```
---------+----+----+----------------+-----------------
 <-Name->|Levs|grid|<--  Units   -->|<- Tile (max=80c)
---------+----+----+----------------+-----------------
 SHIfwFlx|  1 |SM  |kg/m^2/s        |Ice shelf fresh water flux (positive upward)
 SHIhtFlx|  1 |SM  |W/m^2           |Ice shelf heat flux  (positive upward)
 SHIUDrag| 30 |UU  |m/s^2           |U momentum tendency from ice shelf drag
 SHIVDrag| 30 |VV  |m/s^2           |V momentum tendency from ice shelf drag
 SHIForcT|  1 |SM  |W/m^2           |Ice shelf forcing for theta, >0 increases theta
 SHIForcS|  1 |SM  |g/m^2/s         |Ice shelf forcing for salt, >0 increases salt
```

Table 6.23: Available diagnostics of the shelfice-package

Chapter 7

Diagnostics and I/O - Packages II, and Post–Processing Utilities

MITgcm has several packages related to the input and output consumed and produced during a model integration. The packages used are related to the choice of input/output fields and the on-disk format of the model output.

7.1 Diagnostics–A Flexible Infrastructure

7.1.1 Introduction

This section of the documentation describes the Diagnostics package available within the GCM. A large selection of model diagnostics is available for output. In addition to the diagnostic quantities pre-defined in the GCM, there exists the option, in any experiment, to define a new diagnostic quantity and include it as part of the diagnostic output with the addition of a single subroutine call in the routine where the field is computed. As a matter of philosophy, no diagnostic is enabled as default, thus each user must specify the exact diagnostic information required for an experiment. This is accomplished by enabling the specific diagnostic of interest cataloged in the Diagnostic Menu (see Section 7.1.4.2). Instructions for enabling diagnostic output and defining new diagnostic quantities are found in Section 7.1.4 of this document.
The Diagnostic Menu in this section of the manual is a listing of diagnostic quantities available within the main (dynamics) part of the GCM. Additional diagnostic quantities, defined within the different GCM packages, are available and are listed in the diagnostic menu subsection of the manual section associated with each relevant package. Once a diagnostic is enabled, the GCM will continually increment an array specifically allocated for that diagnostic whenever the appropriate quantity is computed. A counter is defined which records how many times each diagnostic quantity has been incremented. Several special diagnostics are included in the menu. Quantities referred to as "Counter Diagnostics", are defined for selected diagnostics which record the frequency at which a diagnostic is incremented separately for each model grid location. Quantities referred to as "User Diagnostics" are included in the menu to facilitate defining new diagnostics for a particular experiment.

7.1.2 Equations

Not relevant.

7.1.3 Key Subroutines and Parameters

There are several utilities within the GCM available to users to enable, disable, clear, write and retrieve model diagnostics, and may be called from any routine. The available utilities and the CALL sequences are listed below.

DIAGNOSTICS_ADDTOLIST (pkg/diagnostics/diagnostics_addtolist.F): This routine is the underlying interface routine for defining a new permanent diagnostic in the main model or in a package. The calling sequence is:

```
      CALL DIAGNOSTICS_ADDTOLIST (
     O     diagNum,
     I     diagName, diagCode, diagUnits, diagTitle, diagMate,
     I     myThid )

     where:
       diagNum   = diagnostic Id number - Output from routine
       diagName  = name of diagnostic to declare
       diagCode  = parser code for this diagnostic
       diagUnits = field units for this diagnostic
       diagTitle = field description for this diagnostic
       diagMate  = diagnostic mate number
       myThid    = my Thread Id number
```

DIAGNOSTICS_FILL (pkg/diagnostics/diagnostics_fill.F): This is the main user interface routine to the diagnostics package. This routine will increment the specified diagnostic quantity with a field sent through the argument list.

```
        CALL DIAGNOSTICS_FILL(
       I             inpFld, diagName,
       I             kLev, nLevs, bibjFlg, bi, bj, myThid )

     where:
        inpFld   = Field to increment diagnostics array
        diagName = diagnostic identificator name (8 characters long)
        kLev     = Integer flag for vertical levels:
                   > 0 (any integer): WHICH single level to increment in qdiag.
                   0,-1 to increment "nLevs" levels in qdiag,
                   0 : fill-in in the same order as the input array
                   -1: fill-in in reverse order.
        nLevs    = indicates Number of levels of the input field array
                   (whether to fill-in all the levels (kLev<1) or just one (kLev>0))
        bibjFlg  = Integer flag to indicate instructions for bi bj loop
                 = 0 indicates that the bi-bj loop must be done here
                 = 1 indicates that the bi-bj loop is done OUTSIDE
                 = 2 indicates that the bi-bj loop is done OUTSIDE
                     AND that we have been sent a local array (with overlap regions)
                     (local array here means that it has no bi-bj dimensions)
                 = 3 indicates that the bi-bj loop is done OUTSIDE
                     AND that we have been sent a local array
                     AND that the array has no overlap region (interior only)
                   NOTE - bibjFlg can be NEGATIVE to indicate not to increment counter
        bi       = X-direction tile number - used for bibjFlg=1-3
        bj       = Y-direction tile number - used for bibjFlg=1-3
        myThid   =  my thread Id number
```

DIAGNOSTICS_SCALE_FILL (pkg/diagnostics/diagnostics_scale_fill.F): This is a possible alternative routine to DIAGNOSTICS_FILL which performs the same functions and has an additional option to scale the field before filling or raise the field to a power before filling.

```
        CALL DIAGNOSTICS_SCALE_FILL(
       I             inpFld, scaleFact, power, diagName,
```

```
     I              kLev, nLevs, bibjFlg, bi, bj, myThid )


   where all the arguments are the same as for DIAGNOSTICS_FILL with
   the addition of:
      scaleFact   = Scaling factor to apply to the input field product
      power       = Integer power to which to raise the input field (after scaling)
```

DIAGNOSTICS_FRACT_FILL (pkg/diagnostics/diagnostics_fract_fill.F): This is a specific alternative routine to DIAGNOSTICS_[SCALE]_FILL for the case of a diagnostics which is associated to a fraction-weight factor (referred to as the diagnostics "counter mate"). This fraction-weight field is expected to vary during the simulation and is provided as argument to DIAGNOSTICS_FRACT_FILL in order to perform fraction-weighted time-average diagnostics. Note that the fraction-weight field has to correspond to the diagnostics counter-mate which has to be filled independently with a call to DIAGNOSTICS_FILL.

```
      CALL DIAGNOSTICS_FRACT_FILL(
     I              inpFld, fractFld, scaleFact, power, diagName,
     I              kLev, nLevs, bibjFlg, bi, bj, myThid )


   where all the arguments are the same as for DIAGNOSTICS_SCALE_FILL with
   the addition of:
      fractFld    = fraction used for weighted average diagnostics
```

DIAGNOSTICS_IS_ON: Function call to inquire whether a diagnostic is active and should be incremented. Useful when there is a computation that must be done locally before a call to DIAGNOSTICS_FILL. The call sequence:

```
      flag = DIAGNOSTICS_IS_ON( diagName, myThid )

   where:
      diagName = diagnostic identificator name (8 characters long)
      myThid   = my thread Id number
```

DIAGNOSTICS_COUNT (pkg/diagnostics/diagnostics_utils.F): This subroutine increments the diagnostics counter only. In general, the diagnostics counter is incremented at the same time as the diagnostics is filled, by calling DIAGNOSTICS_FILL. However, there are few cases where the counter is not incremented during the filling (e.g., when the filling is done level per level but level 1 is skipped) and needs to be done explicitly with a call to subroutine DIAGNOSTICS_COUNT. The call sequence is:

```
      CALL DIAGNOSTICS_COUNT(
     I                         diagName, bi, bj, myThid )

   where:
      diagName  = name of diagnostic to increment the counter
      bi        = X-direction tile number, or 0 if called outside bi,bj loops
      bj        = Y-direction tile number, or 0 if called outside bi,bj loops
      myThid    = my thread Id number
```

The diagnostics are computed at various times and places within the GCM. Because MITgcm may employ a staggered grid, diagnostics may be computed at grid box centers, corners, or edges, and at the middle or edge in the vertical. Some diagnostics are scalars, while others are components of vectors. An internal array is defined which contains information concerning various grid attributes of each diagnostic. The GDIAG array (in common block `diagnostics` in file `DIAGNOSTICS.h`) is internally defined as a character*16 variable, and is equivalenced to a character*1 "parse" array in output in order to extract the grid-attribute information. The GDIAG array is described in Table 7.1.

Table 7.1: Diagnostic Parsing Array

Diagnostic Parsing Array		
Array	Value	Description
parse(1)	→ S	Scalar Diagnostic
	→ U	U-vector component Diagnostic
	→ V	V-vector component Diagnostic
parse(2)	→ U	C-Grid U-Point
	→ V	C-Grid V-Point
	→ M	C-Grid Mass Point
	→ Z	C-Grid Vorticity (Corner) Point
parse(3)	→	Used for Level Integrated output: cumulate levels
	→ r	same but cumulate product by model level thickness
	→ R	same but cumulate product by hFac & level thickness
parse(4)	→ P	Positive Definite Diagnostic
parse(5)	→ C	with Counter array
	→ P	post-processed (not filled up) from other diags
	→ D	Disabled Diagnostic for output
parse(6-8)		retired, formerly: 3-digit mate number
parse(9)	→ U	model-level plus 1/2
	→ M	model-level middle
	→ L	model-level minus 1/2
parse(10)	→ 0	levels = 0
	→ 1	levels = 1
	→ R	levels = Nr
	→ L	levels = MAX(Nr,NrPhys)
	→ M	levels = MAX(Nr,NrPhys) - 1
	→ G	levels = Ground_level Number
	→ I	levels = sea-Ice_level Number
	→ X	free levels option (need to be set explicitly)

As an example, consider a diagnostic whose associated GDIAG parameter is equal to "UUR   MR". From GDIAG we can determine that this diagnostic is a U-vector component located at the C-grid U-point, model mid-level (M) with Nr levels (last R).
In this way, each Diagnostic in the model has its attributes (ie. vector or scalar, C-grid location, etc.) defined internally. The Output routines use this information in order to determine what type of transformations need to be performed. Any interpolations are done at the time of output rather than during each model step. In this way the User has flexibility in determining the type of gridded data which is output.

7.1.4 Usage Notes

7.1.4.1 Using available diagnostics

To use the diagnostics package, other than enabling it in `packages.conf` and turning the `useDiagnostics` flag in `data.pkg` to .TRUE., there are two further steps the user must take to enable the diagnostics package for output of quantities that are already defined in the GCM under an experiment's configuration of packages. A parameter file `data.diagnostics` must be supplied in the run directory, and the file DIAGNOSTICS_SIZE.h must be included in the code directory. The steps for defining a new (permanent or experiment-specific temporary) diagnostic quantity will be outlined later.
The namelist in parameter file `data.diagnostics` will activate a user-defined list of diagnostics quantities to be computed, specify the frequency and type of output, the number of levels, and the name of all the separate output files. A sample `data.diagnostics` namelist file:

```
# Diagnostic Package Choices
#--------------------
#  dumpAtLast (logical): always write output at the end of simulation (default=F)
#  diag_mnc   (logical): write to NetCDF files (default=useMNC)
#--for each output-stream:
#  fileName(n) : prefix of the output file name (max 80c long) for outp.stream n
#  frequency(n):< 0 : write snap-shot output every |frequency| seconds
#               > 0 : write time-average output every frequency seconds
#  timePhase(n)     : write at time = timePhase + multiple of |frequency|
#    averagingFreq  : frequency (in s) for periodic averaging interval
#    averagingPhase : phase     (in s) for periodic averaging interval
#    repeatCycle    : number of averaging intervals in 1 cycle
#  levels(:,n) : list of levels to write to file (Notes: declared as REAL)
#                when this entry is missing, select all common levels of this list
#  fields(:,n) : list of selected diagnostics fields (8.c) in outp.stream n
#                (see "available_diagnostics.log" file for the full list of diags)
#  missing_value(n) : missing value for real-type fields in output file "n"
#  fileFlags(n)     : specific code (8c string) for output file "n"
#--------------------
 &DIAGNOSTICS_LIST
  fields(1:2,1) = 'UVEL    ','VVEL    ',
   levels(1:5,1) = 1.,2.,3.,4.,5.,
   filename(1) = 'diagout1',
  frequency(1) = 86400.,
  fields(1:2,2) = 'THETA   ','SALT    ',
   filename(2) = 'diagout2',
  fileflags(2) = ' P1     ',
  frequency(2) = 3600.,
 &

 &DIAG_STATIS_PARMS
 &
```

In this example, there are two output files that will be generated for each tile and for each output time. The first set of output files has the prefix diagout1, does time averaging every 86400. seconds,

(frequency is 86400.), and will write fields which are multiple-level fields at output levels 1-5. The names of diagnostics quantities are UVEL and VVEL. The second set of output files has the prefix diagout2, does time averaging every 3600. seconds, includes fields with all levels, and the names of diagnostics quantities are THETA and SALT.
The user must assure that enough computer memory is allocated for the diagnostics and the output streams selected for a particular experiment. This is accomplished by modifying the file DIAGNOSTICS_SIZE.h and including it in the experiment code directory. The parameters that should be checked are called numDiags, numLists, numperList, and diagSt_size.
numDiags (and diagSt_size):
All GCM diagnostic quantities are stored in the single diagnostic array QDIAG which is located in the file `pkg/diagnostics/DIAGNOSTICS.h` and has the form:

```
_RL  qdiag(1-Olx,sNx+Olx,1-Olx,sNx+Olx,numDiags,nSx,nSy)
_RL  qSdiag(0:nStats,0:nRegions,diagSt_size,nSx,nSy)
COMMON / DIAG_STORE_R / qdiag, qSdiag
```

The first two-dimensions of qdiag correspond to the horizontal dimension of a given diagnostic, and the third dimension of qdiag is used to identify diagnostic fields and levels combined. In order to minimize the memory requirement of the model for diagnostics, the default GCM executable is compiled with room for only one horizontal diagnostic array, or with numDiags set to Nr. In order for the User to enable more than 1 three-dimensional diagnostic, the size of the diagnostics common must be expanded to accommodate the desired diagnostics. This can be accomplished by manually changing the parameter numDiags in the file `pkg/diagnostics/DIAGNOSTICS_SIZE.h`. numDiags should be set greater than or equal to the sum of all the diagnostics activated for output each multiplied by the number of levels defined for that diagnostic quantity. For the above example, there are 4 multiple level fields, which the diagnostics menu (see below) indicates are defined at the GCM vertical resolution, Nr. The value of numDiags in DIAGNOSTICS_SIZE.h would therefore be equal to 4*Nr, or, say 40 if $Nr = 10$.
numLists and numperList:
The parameter numLists must be set greater than or equal to the number of separate output streams that the user specifies in the namelist file data.diagnostics. The parameter numperList corresponds to the maximum number of diagnostics requested per output streams.

7.1.4.2 Adding new diagnostics to the code

In order to define and include as part of the diagnostic output any field that is desired for a particular experiment, two steps must be taken. The first is to enable the “User Diagnostic” in `data.diagnostics`. This is accomplished by adding one of the “User Diagnostic” field names (UDIAG1 through UDIAG10, for multi-level fields, or SDIAG1 through SDIAG10 for single level fields) to the data.diagnostics namelist in one of the output streams. These fields are listed in the diagnostics menu. The second step is to add a call to DIAGNOSTICS_FILL from the subroutine in which the quantity desired for diagnostic output is computed.
In order to add a new diagnostic to the permanent set of diagnostics that the main model or any package contains as part of its diagnostics menu, the subroutine DIAGNOSTICS_ADDTOLIST should be called during the initialization phase of the main model or package. For the main model, the call should be made from subroutine DIAGNOSTICS_MAIN_INIT, and for a package, the call should probably be made from from inside the particular package’s init_fixed routine. A typical code sequence to set the input arguments to DIAGNOSTICS_ADDTOLIST would look like:

```
      diagName  = 'RHOAnoma'
      diagTitle = 'Density Anomaly (=Rho-rhoConst)'
      diagUnits = 'kg/m^3          '
      diagCode  = 'SMR     MR      '
      CALL DIAGNOSTICS\_ADDTOLIST( diagNum,
     I          diagName, diagCode, diagUnits, diagTitle, 0, myThid )
```

If the new diagnostic quantity is associated with either a vector pair or a diagnostic counter, the diagMate argument must be provided with the proper index corresponding to the “mate”. The output argument from DIAGNOSTICS_ADDTOLIST that is called diagNum here contains a running total of the number of diagnostics defined in the code up to any point during the run. The sequence number for the next two diagnostics defined (the two components of the vector pair, for instance) will be diagNum+1 and diagNum+2. The definition of the first component of the vector pair must fill the "mate" segment of the diagCode as diagnostic number diagNum+2. Since the subroutine increments diagNum, the definition of the second component of the vector fills the "mate" part of diagCode with diagNum. A code sequence for this case would look like:

```
      diagName  = 'UVEL    '
      diagTitle = 'Zonal Component of Velocity (m/s)'
      diagUnits = 'm/s             '
      diagCode  = 'UUR     MR      '
      diagMate  = diagNum + 2
      CALL DIAGNOSTICS_ADDTOLIST( diagNum,
     I   diagName, diagCode, diagUnits, diagTitle, diagMate, myThid )

      diagName  = 'VVEL    '
      diagTitle = 'Meridional Component of Velocity (m/s)'
      diagUnits = 'm/s             '
      diagCode  = 'VVR     MR      '
      diagMate  = diagNum
      CALL DIAGNOSTICS_ADDTOLIST( diagNum,
     I   diagName, diagCode, diagUnits, diagTitle, diagMate, myThid )
```

NAME	UNITS	LEVELS	DESCRIPTION
UVEL	m/sec	Nr	U-Velocity
VVEL	m/sec	Nr	V-Velocity
UVEL_k2	m/sec	1	U-Velocity
VVEL_k2	m/sec	1	V-Velocity
WVEL	m/sec	Nr	Vertical-Velocity
THETASQ	deg^2	Nr	Square of Potential Temperature
SALTSQ	g^2/kg^2	Nr	Square of Salt (or Water Vapor Mixing Ratio)
SALTSQan	g^2/kg^2	Nr	Square of Salt anomaly (=SALT-35)
UVELSQ	m^2/sec^2	Nr	Square of U-Velocity
VVELSQ	m^2/sec^2	Nr	Square of V-Velocity
WVELSQ	m^2/sec^2	Nr	Square of Vertical-Velocity
UV_VEL_C	m^2/sec^2	Nr	Meridional Transport of Zonal Momentum (cell center)
UV_VEL_Z	m^2/sec^2	Nr	Meridional Transport of Zonal Momentum (corner)
WU_VEL	m^2/sec^2	Nr	Vertical Transport of Zonal Momentum (cell center)
WV_VEL	m^2/sec^2	Nr	Vertical Transport of Meridional Momentum (cell center)
UVELMASS	m/sec	Nr	Zonal Mass-Weighted Component of Velocity
VVELMASS	m/sec	Nr	Meridional Mass-Weighted Component of Velocity
WVELMASS	m/sec	Nr	Vertical Mass-Weighted Component of Velocity
UTHMASS	$m - deg/sec$	Nr	Zonal Mass-Weight Transp of Pot Temp
VTHMASS	$m - deg/sec$	Nr	Meridional Mass-Weight Transp of Pot Temp
WTHMASS	$m - deg/sec$	Nr	Vertical Mass-Weight Transp of Pot Temp
ETAN	(hPa, m)	1	Perturbation of Surface (pressure, height)
ETANSQ	(hPa^2, m^2)	1	Square of Perturbation of Surface (pressure, height)
DETADT2	$r - unit^2/s^2$	1	Square of Eta (Surf.P,SSH) Tendency
THETA	$degK$	Nr	Potential Temperature
SST	$degK$	1	Sea Surface Temperature
SALT	g/kg	Nr	Salt (or Water Vapor Mixing Ratio)
SSS	g/kg	1	Sea Surface Salinity
SALTanom	g/kg	Nr	Salt anomaly (=SALT-35)

MITgcm Kernel Diagnostic Menu:

NAME	UNITS	LEVELS	DESCRIPTION
USLTMASS	$m - kg/sec - kg$	Nr	Zonal Mass-Weight Transp of Salt (or W.Vap Mix Rat.)
VSLTMASS	$m - kg/sec - kg$	Nr	Meridional Mass-Weight Transp of Salt (or W.Vap Mix Rat.)
WSLTMASS	$m - kg/sec - kg$	Nr	Vertical Mass-Weight Transp of Salt (or W.Vap Mix Rat.)
UVELTH	$m - deg/sec$	Nr	Zonal Transp of Pot Temp
VVELTH	$m - deg/sec$	Nr	Meridional Transp of Pot Temp
WVELTH	$m - deg/sec$	Nr	Vertical Transp of Pot Temp
UVELSLT	$m - kg/sec - kg$	Nr	Zonal Transp of Salt (or W.Vap Mix Rat.)
VVELSLT	$m - kg/sec - kg$	Nr	Meridional Transp of Salt (or W.Vap Mix Rat.)
WVELSLT	$m - kg/sec - kg$	Nr	Vertical Transp of Salt (or W.Vap Mix Rat.)
RHOAnoma	kg/m^3	Nr	Density Anomaly (=Rho-rhoConst)
RHOANOSQ	kg^2/m^6	Nr	Square of Density Anomaly (=(Rho-rhoConst))
URHOMASS	$kg/m^2/s$	Nr	Zonal Transport of Density
VRHOMASS	$kg/m^2/s$	Nr	Meridional Transport of Density
WRHOMASS	$kg/m^2/s$	Nr	Vertical Transport of Potential Density
PHIHYD	m^2/s^2	Nr	Hydrostatic (ocean) pressure / (atmos) geo-Potential
PHIHYDSQ	m^4/s^4	Nr	Square of Hyd. (ocean) press / (atmos) geoPotential
PHIBOT	m^2/s^2	Nr	ocean bottom pressure / top. atmos geo-Potential
PHIBOTSQ	m^4/s^4	Nr	Square of ocean bottom pressure / top. geo-Potential
DRHODR	$kg/m^3/r - unit$	Nr	Stratification: d.Sigma/dr
VISCA4	m^4/sec	1	Biharmonic Viscosity Coefficient
VISCAH	m^2/sec	1	Harmonic Viscosity Coefficient
TAUX	N/m^2	1	zonal surface wind stress, ¿0 increases uVel
TAUY	N/m^2	1	meridional surf. wind stress, ¿0 increases vVel
TFLUX	W/m^2	1	net surface heat flux, ¿0 increases theta
TRELAX	W/m^2	1	surface temperature relaxation, ¿0 increases theta
TICE	W/m^2	1	heat from melt/freeze of sea-ice, ¿0 increases theta
SFLUX	$g/m^2/s$	1	net surface salt flux, ¿0 increases salt
SRELAX	$g/m^2/s$	1	surface salinity relaxation, ¿0 increases salt
PRESSURE	Pa	Nr	Atmospheric Pressure (Pa)

NAME	UNITS	LEVELS	DESCRIPTION
ADVr_TH	$K.Pa.m^2/s$	Nr	Vertical Advective Flux of Pot.Temperature
ADVx_TH	$K.Pa.m^2/s$	Nr	Zonal Advective Flux of Pot.Temperature
ADVy_TH	$K.Pa.m^2/s$	Nr	Meridional Advective Flux of Pot.Temperature
DFrE_TH	$K.Pa.m^2/s$	Nr	Vertical Diffusive Flux of Pot.Temperature (Explicit part)
DIFx_TH	$K.Pa.m^2/s$	Nr	Zonal Diffusive Flux of Pot.Temperature
DIFy_TH	$K.Pa.m^2/s$	Nr	Meridional Diffusive Flux of Pot.Temperature
DFrI_TH	$K.Pa.m^2/s$	Nr	Vertical Diffusive Flux of Pot.Temperature (Implicit part)
ADVr_SLT	$g/kg.Pa.m^2/s$	Nr	Vertical Advective Flux of Water-Vapor
ADVx_SLT	$g/kg.Pa.m^2/s$	Nr	Zonal Advective Flux of Water-Vapor
ADVy_SLT	$g/kg.Pa.m^2/s$	Nr	Meridional Advective Flux of Water-Vapor
DFrE_SLT	$g/kg.Pa.m^2/s$	Nr	Vertical Diffusive Flux of Water-Vapor (Explicit part)
DIFx_SLT	$g/kg.Pa.m^2/s$	Nr	Zonal Diffusive Flux of Water-Vapor
DIFy_SLT	$g/kg.Pa.m^2/s$	Nr	Meridional Diffusive Flux of Water-Vapor
DFrI_SLT	$g/kg.Pa.m^2/s$	Nr	Vertical Diffusive Flux of Water-Vapor (Implicit part)

NAME	UNITS	LEVELS	DESCRIPTION
SDIAG1		1	User-Defined Surface Diagnostic-1
SDIAG2		1	User-Defined Surface Diagnostic-2
SDIAG3		1	User-Defined Surface Diagnostic-3
SDIAG4		1	User-Defined Surface Diagnostic-4
SDIAG5		1	User-Defined Surface Diagnostic-5
SDIAG6		1	User-Defined Surface Diagnostic-6
SDIAG7		1	User-Defined Surface Diagnostic-7
SDIAG8		1	User-Defined Surface Diagnostic-8
SDIAG9		1	User-Defined Surface Diagnostic-9
SDIAG10		1	User-Defined Surface Diagnostic-1-
SDIAGC		1	User-Defined Counted Surface Diagnostic
SDIAGCC		1	User-Defined Counted Surface Diagnostic Counter
UDIAG1		Nrphys	User-Defined Upper-Air Diagnostic-1
UDIAG2		Nrphys	User-Defined Upper-Air Diagnostic-2
UDIAG3		Nrphys	User-Defined Multi-Level Diagnostic-3
UDIAG4		Nrphys	User-Defined Multi-Level Diagnostic-4
UDIAG5		Nrphys	User-Defined Multi-Level Diagnostic-5
UDIAG6		Nrphys	User-Defined Multi-Level Diagnostic-6
UDIAG7		Nrphys	User-Defined Multi-Level Diagnostic-7
UDIAG8		Nrphys	User-Defined Multi-Level Diagnostic-8
UDIAG9		Nrphys	User-Defined Multi-Level Diagnostic-9
UDIAG10		Nrphys	User-Defined Multi-Level Diagnostic-10

For a list of the diagnostic fields available in the different MITgcm packages, follow the link to the diagnostics menu in the manual section describing the package:

- aim: 6.5.1.2
- exf: 6.4.7.7
- gchem: 6.8.1.3
- generic_advdiff: 6.2.1.3
- gridalt: 6.2.5.6
- gmredi: 6.4.1.6
- fizhi: 6.5.3.3
- kpp: 6.4.2.6
- land: 6.5.2.3
- mom_common: 2.14.8
- obcs: 6.3.1.7
- thsice: 6.6.1.3
- shap_filt: 2.20.1
- ptracers: 6.3.3.4

7.1.5 Dos and Donts

7.1.6 Diagnostics Reference

7.2 NetCDF I/O: MNC

The `mnc` package is a set of convenience routines written to expedite the process of creating, appending, and reading NetCDF files. NetCDF is an increasingly popular self-describing file format *Rew et al.* [1997] intended primarily for scientific data sets. An extensive collection of NetCDF reference papers, user guides, software, FAQs, and other information can be obtained from UCAR's web site at:

`http://www.unidata.ucar.edu/packages/netcdf/`

Since it is a "wrapper" for netCDF, MNC depends upon the Fortran-77 interface included with the standard netCDF v3.x library which is often called `libnetcdf.a`. Please contact your local systems administrators or the MITgcm-support list for help building and installing netCDF for your particular platform.

7.2.1 Using MNC

7.2.1.1 MNC Configuration:

As with all MITgcm packages, MNC can be turned on or off at compile time using the `packages.conf` file or the `genmake2 -enable=mnc` or `-disable=mnc` switches.

While MNC is likely to work "as is", there are a few compile–time constants that may need to be increased for simulations that employ large numbers of tiles within each process. Note that the important quantity is the maximum number of tiles **per process**. Since MPI configurations tend to distribute large numbers of tiles over relatively large numbers of MPI processes, these constants will rarely need to be increased.

If MNC runs out of space within its "lookup" tables during a simulation, then it will provide an error message along with a recommendation of which parameter to increase. The parameters are all located within `pkg/mnc/mnc_common.h` and the ones that may need to be increased are:

Name	Default	Description
`MNC_MAX_ID`	1000	**IDs for various low-level entities**
`MNC_MAX_INFO`	400	**IDs (mostly for object sizes)**
`MNC_CW_MAX_I`	150	**IDs for the "wrapper" layer**

In those rare cases where MNC "out-of-memory" error messages are encountered, it is a good idea to increase the too-small parameter by a factor of **2–10** in order to avoid wasting time on an iterative compile–test sequence.

7.2.1.2 MNC Inputs:

Like most MITgcm packages, all of MNC can be turned on/off at runtime using a single flag in `data.pkg`

Name	T	Default	Description
`useMNC`	L	`.FALSE.`	overall MNC ON/OFF switch

One important MNC–related flag is present in the main `data` namelist file in the `PARM03` section and it is:

Name	T	Default	Description
`outputTypesInclusive`	L	`.FALSE.`	use all available output "types"

which specifies that turning on MNC for a particular type of output should not simultaneously turn off the default output method as it normally does. Usually, this option is only used for debugging purposes since it is inefficient to write output types using both MNC and MDSIO or ASCII output. This option can also be helpful when transitioning from MDSIO to MNC since the output can be readily compared.

For run-time configuration, most of the MNC–related model parameters are contained within a Fortran namelist file called `data.mnc`. The availabe parameters currently include:

Name	T	Default	Description
mnc_use_outdir	L	.FALSE.	create a directory for output
mnc_outdir_str	S	'mnc_'	output directory name
mnc_outdir_date	L	.FALSE.	embed date in the outdir name
mnc_outdir_num	L	.TRUE.	optional
pickup_write_mnc	L	.TRUE.	use MNC to write pickup files
pickup_read_mnc	L	.TRUE.	use MNC to read pickup files
mnc_use_indir	L	.FALSE.	use a directory (path) for input
mnc_indir_str	S	''	input directory (or path) name
snapshot_mnc	L	.TRUE.	write `snapshot` output w/MNC
monitor_mnc	L	.TRUE.	write `monitor` output w/MNC
timeave_mnc	L	.TRUE.	write `timeave` output w/MNC
autodiff_mnc	L	.TRUE.	write `autodiff` output w/MNC
mnc_max_fsize	R	2.1e+09	max allowable file size (¡2GB)
mnc_filefreq	R	-1	frequency of new file creation (seconds)
readgrid_mnc	L	.FALSE.	read grid quantities using MNC
mnc_echo_gvtypes	L	.FALSE.	list pre-defined "types" (debug)

Unlike the older MDSIO method, MNC has the ability to create or use existing output directories. If either `mnc_outdir_date` or `mnc_outdir_num` is true, then MNC will try to create directories on a *PER PROCESS* basis for its output. This means that a single directory will be created for a non-MPI run and multiple directories (one per MPI process) will be created for an MPI run. This approach was chosen since it works safely on both shared global file systems (such as NFS and AFS) and on local (per-compute-node) file systems. And if both `mnc_outdir_date` and `mnc_outdir_num` are false, then the MNC package will assume that the directory specified in `mnc_outdir_str` already exists and will use it. This allows the user to create and specify directories outside of the model.

For input, MNC can use a single global input directory. This is a just convenience that allows MNC to gather all of its input files from a path other than the current working directory. As with MDSIO, the default is to use the current working directory.

The flags `snapshot_mnc`, `monitor_mnc`, `timeave_mnc`, and `autodiff_mnc` allow the user to turn on MNC for particular "types" of output. If a type is selected, then MNC will be used for all output that matches that type. This applies to output from the main model and from all of the optional MITgcm packages. Mostly, the names used here correspond to the names used for the output frequencies in the main `data` namelist file.

The `mnc_max_fsize` parameter is a convenience added to help users work around common file size limitations. On many computer systems, either the opterating system, the file system(s), and/or the netCDF libraries are unable to handle files greater than two or four gigabytes in size. The MNC package is able to work within this limitation by creating new files which grow along the netCDF "unlimited" (usually, time) dimension. The default value for this parameter is just slightly less than 2GB which is safe on virtually all operating systems. Essentially, this feature is a way to intelligently and automatically split files output along the unlimited dimension. On systems that support large file sizes, these splits can be readily concatenated (that is, un-done) using tools such as the netCDF Operators (with `ncrcat`) which is available at:

`http://nco.sourceforge.net/`

Another way users can force the splitting of MNC files along the time dimension is the `mnc_filefreq` option. With it, files that contain variables with a temporal dimension can be split at regular intervals based solely upon the model time (specified in seconds). For some problems, this can be much more convenient than splitting based upon file size.

Additional MNC–related parameters may be contained within each package. Please see the individual packages for descriptions of their use of MNC.

7.2.1.3 MNC Output:

Depending upon the flags used, MNC will produce zero or more directories containing one or more netCDF files as output. These files are either mostly or entirely compliant with the netCDF "CF" convention (v1.0) and any conformance issues will be fixed over time. The patterns used for file names are:

```
BASENAME.tileNum.nc
BASENAME.nIter.faceNum.nc
BASENAME.nIter.tileNum.nc
```

and examples are:

```
grid.t001.nc, grid.t002.nc
state.0000000000.t001.nc, surfDiag.0000036000.t001.nc
input.0000072000.f001.nc
```

where `BASENAME` is the name selected to represent a set of variables written together, `nIter` is the current iteration number as specified in the main `data` namelist input file and written in a zero-filled 10-digit format, `tileNum` is a three-or-more-digit zero-filled and "`t`"–prefixed tile number, `faceNum` is a three-or-more-digit zero-filled and "`f`"–prefixed face number, and `.nc` is the file suffix specified by the current netCDF "CF" conventions.

Some example `BASENAME` values are:

grid contains the variables that describe the various grid constants related to locations, lengths, areas, etc.

state contains the variables output at the snapshot or `dumpFreq` time frequency

pickup.ckptA, pickup.ckptB are the "rolling" checkpoint files

tave contains the time-averaged quantities from the main model

All MNC output is currently done in a "file-per-tile" fashion since most NetCDF v3.x implementions cannot write safely within MPI or multi-threaded environments. This tiling is done in a global fashion and the tile numbers are appended to the base names as described above. Some scripts to manipulate MNC output are available at `MITgcm/utils/matlab/` which includes a spatial "assembly" script called `MITgcm/utils/matlab/mnc_assembly.m`.

More general manipulations can be performed on netCDF files with

```
the NetCDF Operators (''NCO'')
at http://nco.sourceforge.net
```

or with

```
the Climate Data Operators (''CDO'')
at http://www.mpimet.mpg.de/~cdo/
```

Unlike the older MDSIO routines, MNC reads and writes variables on different "grids" depending upon their location on, for instance, an Arakawa C–grid. The following table provides examples:

Name	C–grid location	# in X	# in Y
Temperature	mass	sNx	sNy
Salinity	mass	sNx	sNy
U velocity	U	sNx+1	sNy
V velocity	V	sNx	sNy+1
Vorticity	vorticity	sNx+1	sNy+1

and the intent is two–fold:

1. For some grid topologies it is impossible to output all quantities using only `sNx,sNy` arrays for every tile. Two examples of this failure are the missing corners problem for vorticity values on the cubesphere and the velocity edge values for some open–boundary domains.

2. Writing quantities located on velocity or vorticity points with the above scheme introduces a very small data redundancy. However, any slight inconvenience is easily offset by the ease with which one can, on every individual tile, interpolate these values to mass points without having to perform an "exchange" (or "halo-filling") operation to collect the values from neighboring tiles. This makes the most common post–processing operations much easier to implement.

7.2.2 MNC Troubleshooting

7.2.2.1 Build Troubleshooting:

In order to build MITgcm with MNC enabled, the netCDF v3.x Fortran-77 (not Fortran-90) library must be available. This library is compposed of a single header file (called `netcdf.inc`) and a single library file (usually called `libnetcdf.a`) and it must be built with the same compiler (or a binary-compatible compiler) with compatible compiler options as the one used to build MITgcm.

For more details concerning the netCDF build and install process, please visit the netCDF home page at:

`http://www.unidata.ucar.edu/packages/netcdf/`

which includes an extensive list of known–good netCDF configurations for various platforms

7.2.2.2 Runtime Troubleshooting:

Please be aware of the following:

- As a safety feature, the MNC package does not, by default, allow pre-existing files to be appended to or overwritten. This is in contrast to the older MDSIO package which will, without any warning, overwrite existing files. If MITgcm aborts with an error message about the inability to open or write to a netCDF file, please check **first** whether you are attempting to overwrite files from a previous run.

- The constraints placed upon the "unlimited" (or "record") dimension inherent with NetCDF v3.x make it very inefficient to put variables written at potentially different intervals within the same file. For this reason, MNC output is split into groups of files which attempt to reflect the nature of their content.

- On many systems, netCDF has practical file size limits on the order of 2–4GB (the maximium memory addressable with 32bit pointers or pointer differences) due to a lack of operating system, compiler, and/or library support. The latest revisions of netCDF v3.x have large file support and, on some operating systems, file sizes are only limited by available disk space.

- There is an 80 character limit to the total length of all file names. This limit includes the directory (or path) since paths and file names are internally appended. Generally, file names will not exceed the limit and paths can usually be shortened using, for example, soft links.

- MNC does not (yet) provide a mechanism for reading information from a single "global" file as can be done with the MDSIO package. This is in progress.

7.2.3 MNC Internals

The `mnc` package is a two-level convenience library (or "wrapper") for most of the NetCDF Fortran API. Its purpose is to streamline the user interface to NetCDF by maintaining internal relations (look-up tables) keyed with strings (or names) and entities such as NetCDF files, variables, and attributes.

The two levels of the `mnc` package are:

Upper level

The upper level contains information about two kinds of associations:

grid type is lookup table indexed with a grid type name. Each grid type name is associated with a number of dimensions, the dimension sizes (one of which may be unlimited), and starting and ending index arrays. The intent is to store all the necessary size and shape information for the Fortran arrays containing MITgcm–style "tile" variables (that is, a central region surrounded by a variably-sized "halo" or exchange region as shown in Figures 4.7 and 4.8).

variable type is a lookup table indexed by a variable type name. For each name, the table contains a reference to a grid type for the variable and the names and values of various attributes.

Within the upper level, these associations are not permanently tied to any particular NetCDF file. This allows the information to be re-used over multiple file reads and writes.

Lower level

In the lower (or internal) level, associations are stored for NetCDF files and many of the entities that they contain including dimensions, variables, and global attributes. All associations are on a per-file basis. Thus, each entity is tied to a unique NetCDF file and will be created or destroyed when files are, respectively, opened or closed.

7.2.3.1 MNC Grid–Types and Variable–Types:

As a convenience for users, the MNC package includes numerous routines to aid in the writing of data to NetCDF format. Probably the biggest convenience is the use of pre-defined "grid types" and "variable types". These "types" are simply look-up tables that store dimensions, indicies, attributes, and other information that can all be retrieved using a single character string.

The "grid types" are a way of mapping variables within MITgcm to NetCDF arrays. Within MITgcm, most spatial variables are defined using two– or three–dimensional arrays with "overlap" regions (see Figures 4.7, a possible vertical index, and 4.8) and tile indicies such as the following "U" velocity:

```
_RL  uVel (1-OLx:sNx+OLx,1-OLy:sNy+OLy,Nr,nSx,nSy)
```

as defined in `model/inc/DYNVARS.h`

The grid type is a character string that encodes the presence and types associated with the four possible dimensions. The character string follows the format

`H0_H1_H2__V__T`

where the terms *H0*, *H1*, *H2*, *V*, *T* can be almost any combination of the following:

Horizontal			Vertical	Time
H0: location	**H1**: dimensions	**H2**: halo	**V**: location	**T**: level
–	xy	Hn	–	–
U	x	Hy	i	t
V	y		c	
Cen				
Cor				

A example list of all pre-defined combinations is contained in the file

`pkg/mnc/pre-defined_grids.txt`.

The variable type is an association between a variable type name and the following items:

Item	**Purpose**
grid type	defines the in-memory arrangement
`bi,bj` dimensions	tiling indices, if present

and is used by the `mnc_cw_*_[R|W]` subroutines for reading and writing variables.

7.2.3.2 Using MNC: Examples:

Writing variables to NetCDF files can be accomplished in as few as two function calls. The first function call defines a variable type, associates it with a name (character string), and provides additional information about the indicies for the tile (`bi,bj`) dimensions. The second function call will write the data at, if necessary, the current time level within the model.

Examples of the initialization calls can be found in the file `model/src/ini_mnc_io.F` where these function calls:

```
C     Create MNC definitions for DYNVARS.h variables
      CALL MNC_CW_ADD_VNAME('iter', '-_-_--__-__t', 0,0, myThid)
      CALL MNC_CW_ADD_VATTR_TEXT('iter',1,
     &     'long_name','iteration_count', myThid)

      CALL MNC_CW_ADD_VNAME('model_time', '-_-_--__-__t', 0,0, myThid)
      CALL MNC_CW_ADD_VATTR_TEXT('model_time',1,
     &     'long_name','Model Time', myThid)
      CALL MNC_CW_ADD_VATTR_TEXT('model_time',1,'units','s', myThid)

      CALL MNC_CW_ADD_VNAME('U', 'U_xy_Hn__C__t', 4,5, myThid)
      CALL MNC_CW_ADD_VATTR_TEXT('U',1,'units','m/s', myThid)
      CALL MNC_CW_ADD_VATTR_TEXT('U',1,
     &     'coordinates','XU YU RC iter', myThid)

      CALL MNC_CW_ADD_VNAME('T', 'Cen_xy_Hn__C__t', 4,5, myThid)
      CALL MNC_CW_ADD_VATTR_TEXT('T',1,'units','degC', myThid)
      CALL MNC_CW_ADD_VATTR_TEXT('T',1,'long_name',
     &     'potential_temperature', myThid)
      CALL MNC_CW_ADD_VATTR_TEXT('T',1,
     &     'coordinates','XC YC RC iter', myThid)
```

initialize four VNAMEs and add one or more NetCDF attributes to each.

The four variables defined above are subsequently written at specific time steps within model/src/write_state.F using the function calls:

```
C       Write dynvars using the MNC package
        CALL MNC_CW_SET_UDIM('state', -1, myThid)
        CALL MNC_CW_I_W('I','state',0,0,'iter', myIter, myThid)
        CALL MNC_CW_SET_UDIM('state', 0, myThid)
        CALL MNC_CW_RL_W('D','state',0,0,'model_time',myTime, myThid)
        CALL MNC_CW_RL_W('D','state',0,0,'U', uVel, myThid)
        CALL MNC_CW_RL_W('D','state',0,0,'T', theta, myThid)
```

While it is easiest to write variables within typical 2D and 3D fields where all data is known at a given time, it is also possible to write fields where only a portion (*eg.* a "slab" or "slice") is known at a given instant. An example is provided within pkg/mom_vecinv/mom_vecinv.F where an offset vector is used:

```
       IF (useMNC .AND. snapshot_mnc) THEN
         CALL MNC_CW_RL_W_OFFSET('D','mom_vi',bi,bj, 'fV', uCf,
   &          offsets, myThid)
         CALL MNC_CW_RL_W_OFFSET('D','mom_vi',bi,bj, 'fU', vCf,
   &          offsets, myThid)
       ENDIF
```

to write a 3D field one depth slice at a time.

Each element in the offset vector corresponds (in order) to the dimensions of the "full" (or virtual) array and specifies which are known at the time of the call. A zero within the offset array means that all values along that dimension are available while a positive integer means that only values along that index of the dimension are available. In all cases, the matrix passed is assumed to start (that is, have an in-memory structure) coinciding with the start of the specified slice. Thus, using this offset array mechanism, a slice can be written along any single dimension or combinations of dimensions.

7.3 Fortran Native I/O: MDSIO and RW

7.3.1 MDSIO

7.3.1.1 Introduction

The `mdsio` package contains a group of Fortran routines intended as a general interface for reading and writing direct-access ("binary") Fortran files. The `mdsio` routines are used by the `rw` package.

The `mdsio` package is currently the primary method for MITgcm I/O, but it is not being actively extended or enhanced. Instead, the `mnc` netCDF package (see Section 7.2) is expected to gain all of the current `mdsio` functionality and, eventually, replace it. For the short term, every effort has been made to allow `mnc` and `mdsio` to peacefully co-exist. In may cases, the model can read one format and write to the other. This side-by-side functionality can be used to, for instance, help convert pickup files or other data sets between the two formats.

7.3.1.2 Using MDSIO

The `mdsio` package is geared toward the reading and writing of floating point (Fortran `REAL*4` or `REAL*8`) arrays. It assumes that the in-memory layout of all arrays follows the per-tile MITgcm convention

```
C     Example of a "2D" array
      _RL anArray(1-OLx:sNx+OLx,1-OLy:sNy+OLy,nSx,nSy)

C     Example of a "3D" array
      _RL anArray(1-OLx:sNx+OLx,1-OLy:sNy+OLy,1:Nr,nSx,nSy)
```

where the first two dimensions are spatial or "horizontal" indicies that include a "halo" or exchange region (please see Chapters 4 and 6.2.4 which describe domain decomposition), and the remaining indicies (`Nr`,`nSx`, and `nSx`) are often present but not required.

In order to write output, the `mdsio` package is called with a function such as:

```
      CALL MDSWRITEFIELD(fn,prec,lgf,typ,Nr,arr,irec,myIter,myThid)
```

where:

- `fn` is a `CHARACTER` string containing a file "base" name which will then be used to create file names that contain tile and/or model iteration indicies
- `prec` is an integer that contains one of two globally defined values (`precFloat64` or `precFloat32`)
- `lgf` is a `LOGICAL` that typically contains the globally defined `globalFile` option which specifies the creation of globally (spatially) concatenated files
- `typ` is a `CHARACTER` string that specifies the type of the variable being written (`'RL'` or `'RS'`)
- `Nr` is an integer that specifies the number of vertical levels within the variable being written
- `arr` is the variable (array) to be written
- `irec` is the starting record within the output file that will contain the array
- `myIter,myThid` are integers containing, respectively, the current model iteration count and the unique thread ID for the current context of execution

As one can see from the above (generic) example, enough information is made available (through both the argument list and through common blocks) for the `mdsio` package to perform the following tasks:

1. open either a per-tile file such as:

   `uVel.0000302400.003.001.data`

   or a "global" file such as

   `uVel.0000302400.data`

2. byte-swap (as necessary) the input array and write its contents (minus any halo information) to the binary file – or to the correct location within the binary file if the globalfile option is used, and

3. create an ASCII–text metadata file (same name as the binary but with a `.meta` extension) describing the binary file contents (often, for later use with the MatLAB `rdmds()` utility).

Reading output with `mdsio` is very similar to writing it. A typical function call is

```
CALL MDSREADFIELD(fn,prec,typ,Nr,arr,irec,myThid)
```

where variables are exactly the same as the `MDSWRITEFIELD` example provided above. It is important to note that the `lgf` argument is missing from the `MDSREADFIELD` function. By default, `mdsio` will first try to read from an appropriately named global file and, failing that, will try to read from a per-tile file.

7.3.1.3 Important Considerations

When using `mdsio`, one should be aware of the following package features and limitations:

Byte-swapping is, for the most part, gracefully handled. All files intended for reading/writing by `mdsio` should contain big-endian (sometimes called "network byte order") data. By handling byte-swapping within the model, MITgcm output is more easily ported between different machines, architectures, compilers, etc. Byteswapping can be turned on/off at compile time within `mdsio` using the `_BYTESWAPIO` CPP macro which is usually set within a `genmake2` options file or "`optfile`" which are located in

```
MITgcm/tools/build_options
```

Additionally, some compilers may have byte-swap options that are speedier or more convenient to use.

Types are currently limited to single– or double–precision floating point values. These values can be converted, on-the-fly, from one to the other so that any combination of either single– or double–precision variables can be read from or written to files containing either single– or double–precision data.

Array sizes are limited. The `mdsio` package is very much geared towards the reading/writing of per-tile (that is, domain-decomposed and halo-ed) arrays. Data that cannot be made to "fit" within these assumed sizes can be challenging to read or write with `mdsio`.

Tiling or domain decomposition is automatically handled by `mdsio` for logically rectangular grid topologies (*eg.* lat-lon grids) and "standard" cubesphere topologies. More complicated topologies will probably not be supported. The `mdsio` package can, without any coding changes, read and write to/from files that were run on the same global grid but with different tiling (grid decomposition) schemes. For example, `mdsio` can use and/or create identical input/output files for a "C32" cube when the model is run with either 6, 12, or 24 tiles (corresponding to 1, 2 or 4 tiles per cubesphere face). Currently, this is one of the primary advantages that the `mdsio` package has over `mnc`.

Single-CPU I/O can be specified with the flag

```
useSingleCpuIO = .TRUE.,
```

in the `PARM01` namelist within the main `data` file. Single–CPU I/O mode is appropriate for computers (*eg.* some SGI systems) where it can either speed overall I/O or solve problems where the operating system or file systems cannot correctly handle multiple threads or MPI processes simultaneously writing to the same file.

Meta-data is written by MITgcm on a per-file basis using a second file with a `.meta` extension as described above. MITgcm itself does not read the `*.meta` files, they are there primarly for convenience during post-processing. One should be careful not to delete the meta-data files when using MatLAB post-processing scripts such as `rdmds()` since it relies upon them.

Numerous files can be written by `mdsio` due to its typically per-time-step and per-variable orientation. The creation of both a binary (`*.data`) and ASCII text meta-data (`*.meta`) file for each output type step tends to exacerbate the problem. Some (mostly, older) operating systems do not gracefully handle large numbers (*eg.* many thousands) of files within one directory. So care should be taken to split output into smaller groups using subdirectories.

Overwriting is the **default behavior** for `mdsio`. If a model tries to write to a file name that already exists, the older file **will be deleted**. For this reason, MITgcm users should be careful to move output that that wish to keep into, for instance, subdirectories before performing subsequent runs that may over–lap in time or otherwise produce files with identical names (*eg.* Monte-Carlo simulations).

No "halo" information is written or read by `mdsio`. Along the horizontal dimensions, all variables are written in an `sNx`–by–`sNy` fashion. So, although variables (arrays) may be defined at different locations on Arakawa grids [U (right/left horizontal edges), V (top/bottom horizontal edges), M (mass or cell center), or Z (vorticity or cell corner) points], they are all written using only interior (`1:sNx` and `1:sNy`) values. For quantities defined at U, V, and M points, writing `1:sNx` and `1:sNy` for every tile is sufficient to ensure that all values are written globally for some grids (eg. cubesphere, re-entrant channels, and doubly-periodic rectangular regions). For Z points, failing to write values at the `sNx+1` and `sNy+1` locations means that, for some tile topologies, not all values are written. For instance, with a cubesphere topology at least two corner values are "lost" (fail to be written for any tile) if the `sNx+1` and `sNy+1` values are ignored. To fix this problem, the `mnc` package writes the `sNx+1` and `sNy+1` grid values for the U, V, and Z locations. Also, the `mnc` package is capable of reading and/or writing entire halo regions and more complicated array shapes which can be helpful when debugging–features that do not exist within `mdsio`.

7.3.2 RW Basic binary I/O utilities

The `rw` package provides a very rudimentary binary I/O capability for quickly writing *single record* direct-access Fortran binary files. It is primarily used for writing diagnostic output.

7.3.2.1 Introduction

Package `rw` is an interface to the more general `mdsio` package. The `rw` package can be used to write or read direct-access Fortran binary files for two-dimensional XY and three-dimensional XYZ arrays. The arrays are assumed to have been declared according to the standard MITgcm two-dimensional or three-dimensional floating point array type:

```
C     Example of declaring a standard two dimensional "long"
C     floating point type array (the _RL macro is usually
C     mapped to 64-bit floats in most configurations)
      _RL anArray(1-OLx:sNx+OLx,1-OLy:sNy+OLy,nSx,nSy)
```

Each call to an `rw` read or write routine will read (or write) to the first record of a file. To write direct access Fortran files with multiple records use the package `mdsio` (see section 7.3.1). To write self-describing files that contain embedded information describing the variables being written and the spatial and temporal locations of those variables use the package `mnc` (see section 7.2) which produces netCDF[1] *Rew et al.* [1997] based output.

[1] `http://www.unidata.ucar.edu/packages/netcdf`

7.4 Monitor: Simulation state monitoring toolkit

7.4.1 Introduction

The `monitor` package is primarily intended as a convenient method for calculating and writing the following statistics:

minimum, maximum, mean, and standard deviation

for spatially distributed fields. By default, `monitor` output is sent to the "standard output" channel where it appears as ASCII text containing a `%MON` string such as this example:

```
(PID.TID 0000.0001) %MON time_tsnumber      =                    3
(PID.TID 0000.0001) %MON time_secondsf      =   3.6000000000000E+03
(PID.TID 0000.0001) %MON dynstat_eta_max    =   1.0025466645951E-03
(PID.TID 0000.0001) %MON dynstat_eta_min    =  -1.0008899950901E-03
(PID.TID 0000.0001) %MON dynstat_eta_mean   =   2.1037438449350E-14
(PID.TID 0000.0001) %MON dynstat_eta_sd     =   5.0985228723396E-04
(PID.TID 0000.0001) %MON dynstat_eta_del2   =   3.5216706549525E-07
(PID.TID 0000.0001) %MON dynstat_uvel_max   =   3.7594045977254E-05
(PID.TID 0000.0001) %MON dynstat_uvel_min   =  -2.8264287531564E-05
(PID.TID 0000.0001) %MON dynstat_uvel_mean  =   9.1369201945671E-06
(PID.TID 0000.0001) %MON dynstat_uvel_sd    =   1.6868439193567E-05
(PID.TID 0000.0001) %MON dynstat_uvel_del2  =   8.4315445301916E-08
```

The `monitor` text can be readily parsed by the `testreport` script to determine, somewhat crudely but quickly, how similar the output from two experiments are when run on different platforms or before/after code changes.

The `monitor` output can also be useful for quickly diagnosing practical problems such as CFL limitations, model progress (through iteration counts), and behavior within some packages that use it.

7.4.2 Using Monitor

As with most packages, `monitor` can be turned on or off at compile and/or run times using the `packages.conf` and `data.pkg` files.

The monitor output can be sent to the standard output channel, to an `mnc`–generated file, or to both simultaneously. For `mnc` output, the flag:

```
monitor_mnc=.TRUE.,
```

should be set within the `data.mnc` file. For output to both ASCII and `mnc`, the flag

```
outputTypesInclusive=.TRUE.,
```

should be set within the `PARM03` section of the main `data` file. It should be noted that the `outputTypesInclusive` flag will make **ALL** kinds of output (that is, everything written by `mdsio`, `mnc`, and `monitor`) simultaneously active so it should be used only with caution–and perhaps only for debugging purposes.

7.5 Grid Generation

The horizontal discretizations within MITgcm have been written to work with many different grid types including:

- cartesian coordinates
- spherical polar ("latitude-longitude") coordinates
- general curvilinear orthogonal coordinates

The last of these, especially when combined with the domain decomposition capabilities of MITgcm, allows a great degree of grid flexibility. To date, general curvilinear orthogonal coordinates have been used primarily (in fact, almost exclusively) in conjunction with so-called "cube-sphere" grids. However, it is important to observe that cube-sphere arrangements are only one example of what is possible with domain-decomposed logically rectangular regions each containing curvilinear orthogonal coordinate systems. Much more sophisticated domains can be imagined and constructed.

In order to explore the possibilities of domain-decomposed curvilinear orthogonal coordinate systems, a suite of grid generation software called "SPGrid" (for SPherical Gridding) has been developed. SPGrid is a relatively new facility and papers detailing its algorithms are in preparation. Althogh SPGrid is new and rapidly developing, it has already demonstrated the ability to generate some useful and interesting grids.

This section provides a very brief introduction to SPGrid and shows some early results. For further information, please contact the MITgcm support list at:

MITgcm-support@mitgcm.org

7.5.1 Using SPGrid

The SPGrid software is not a single program. Rather, it is a collection of C++ code and MatLAB scripts that can be used as a framework or library for grid generation and manipulation. Currently, grid creation is accomplished by either directly running matlab scripts or by writing a C++ "driver" program. The matlab scripts are suitable for grids composed of a single "face" (that is, a single logically rectangular region on the surface of a sphere). The C++ driver programs are appropriate for grids composed of multiple connected logically rectangular patches. Each driver is program is written to specify the shape and connectivity of tiles and the preferred grid density (that is, the number of grid cells in each logical direction) and edge locations of the cells where they meet the edges of each face. The driver programs pass this information to the SPGrid library which generates the actual grid and produces the output files that describe it.

Currently, driver programs are available for a few examples including cubes, "lat-lon caps" (cube topologies that have conformal caps at the poles and are exactly lat-lon channels for the remainder of the domain), and some simple "embedded" regions that are meant to be used within typical cubes or traditional lat-lon grids.

To create new grids, one may start with an existing driver program and modify it to describe a domain that has a different arrangement. The number, location, size, and connectivity of grid "faces" (the name used for the logically rectangular regions) can be readily changed. Further, the number of grid cells within faces and the location of the grid cells at the face edges can also be specified.

7.5.1.1 SPGrid Requirements

The following programs and libraries are required to build and/or run the SPGrid suite:

- MatLAB is a run-time requirement since many of the generation algorithms have been written as MatLAB scripts:
  `http://www.mathworks.com`
- the Wild Magic graphics engine (a C++ library) is needed for the main "driver" code:
  `http://geometrictools.com/`

- the NetCDF library is needed for file I/O:
  `http://www.mathworks.com`
- the BOOST Serialization library is used for I/O:
  `http://www.boost.org`
- a typical Linux/Unix build environment including the make utility (preferably Gnu Make) and a C++ compiler (SPGrid was developed with g++ v4.x).

7.5.1.2 Obtaining SPGrid

The latest version can be obtained from:

`http://mitgcm.org/∼edhill/grids/spgrid_releases/`

7.5.1.3 Building SPGrid

The procedure for building is similar to many open source projects:

```
tar -xf spgrid-0.9.4.tar.gz
cd spgrid-0.9.4
export CPPFLAGS="-I/usr/include/netcdf-3"
export LDFLAGS="-L/usr/lib/netcdf-3"
./configure
make
```

where the `CPPFLAGS` and `LDFLAGS` environment variables can be edited to reflect the locations of all the necessary dependencies. SPGrid is known to work on Fedora Core Linux (versions 4 and 5) and is likely to work on most any Linux distribution that provides the needed dependencies.

7.5.1.4 Running SPGrid

Within the `src` sub-directory, various example driver programs exist. These examples describe small, simple domains and can generate the input files (formatted as either binary `*.mitgrid` or netCDF) used by MITgcm.

One such example is called "SpF_test_cube_cap" and it can be run with the following sequence of commands:

```
cd spgrid-0.9.4/src
make SpF_test_cube_cap
mkdir SpF_test_cube_cap.d
( cd SpF_test_cube_cap.d && ln -s ../../scripts/*.m . )
./SpF_test_cube_cap
```

which should create a series of output files:

```
SpF_test_cube_cap.d/grid_*.mitgrid
SpF_test_cube_cap.d/grid_*.nc
SpF_test_cube_cap.d/std_topology.nc
```

where the `grid_*.mitgrid` and `grid_*.nc` files contain the grid information in binary and netCDF formats and the `std_topology.nc` file contains the information describing the connectivity (both edge–edge and corner–corner contacts) between all the faces.

7.5.2 Example Grids

The following grids are various examples created with SPGrid.

7.6 Pre– and Post–Processing Scripts and Utilities

There are numerous tools for pre-processing data, converting model output and analysis written in Matlab, fortran (f77 and f90) and perl. As yet they remain undocumented although many are self-documenting (Matlab routines have "help" written into them).

Here we'll summarize what is available but this is an ever growing resource so this may not cover everything that is out there:

7.6.1 Utilities supplied with the model

We supply some basic scripts with the model to facilitate conversion or reading of data into analysis software.

7.6.1.1 utils/scripts

In the directory *utils/scripts* you will find *joinds* and *joinmds*: these are perl scripts used from joining the multi-part files created by MITgcm. **Use *joinmds* always**. You will only need *joinds* if you are working with output older than two years (prior to c23).

7.6.1.2 utils/matlab

In the directory *utils/matlab* you will find several Matlab scripts (.m or dot-em files). The priniciple script is *rdmds.m* used for reading the multi-part model output files in to matlab. Place the scripts in your matlab path or change the path appropriately, then at the matlab prompt type:

```
>> help rdmds
```

to get help on how to use rdmds.

Another useful script scans the terminal output file for "monitor" information.

Most other scripts are for working in the curvilinear coordinate systems which as yet are unpublished and undocumented.

7.6.1.3 MNC Utils

The MITgcm netCDF integration is relatively new and the tools used to work on netCDF data sets are developing. The following scripts and utilities have been written to help manipulate MNC (netCDF) files:

Tile Assembly: a matlab script `utils/matlab/mnc_assembly.m` is available for spatially "assembling" MNC output. A convenience wrapper script called `utils/matlab/gluemnc.m` is also provided. Please use the matlab help facility for more information.

gmt: As MITgcm evolves to handle more complicated domains and topologies, a suite of matlab tools is being written to more gracefully handle the model files. This suite is called "gmt" which refers to "generalized model topology" pre-/post-processing. Currently, this directory contains a matlab script `utils/matlab/gmt/rdnctiles.m` that is able to read the MNC-created netCDF files for any domain. Additional scripts are being created that will work with these fields on a per-tile basis.

7.6.2 Pre-processing software

There is a suite of pre-processing software for intepolating bathymetry and forcing data, written by Adcroft and Biastoch. At some point, these will be made available for download. If you are in need of such software, contact one of them.

7.7 Potential vorticity Matlab Toolbox

Author: Guillaume Maze

7.7.1 Introduction

This section of the documentation describes a Matlab package that aims to provide useful routines to compute vorticity fields (relative, potential and planetary) and its related components. This is an offline computation. It was developed to be used in mode water studies, so that it comes with other related routines, in particular ones computing surface vertical potential vorticity fluxes.

7.7.2 Equations

7.7.2.1 Potential Vorticity

The package computes the three components of the relative vorticity defined by:

$$\omega = \nabla \times \mathbf{U} = \begin{pmatrix} \omega_x \\ \omega_y \\ \zeta \end{pmatrix} \simeq \begin{pmatrix} -\frac{\partial v}{\partial z} \\ -\frac{\partial u}{\partial z} \\ \frac{\partial v}{\partial x} - \frac{\partial u}{\partial y} \end{pmatrix} \tag{7.1}$$

where we omitted (like all across the package) the vertical velocity component.

The package then computes the potential vorticity as:

$$Q = -\frac{1}{\rho}\omega \cdot \nabla\sigma_\theta \tag{7.2}$$

$$Q = -\frac{1}{\rho}\left(\omega_x \frac{\partial\sigma_\theta}{\partial x} + \omega_y \frac{\partial\sigma_\theta}{\partial y} + (f+\zeta)\frac{\partial\sigma_\theta}{\partial z}\right) \tag{7.3}$$

where ρ is the density, σ_θ is the potential density (both eventually computed by the package) and f is the Coriolis parameter.

The package is also able to compute the simpler planetary vorticity as:

$$splQ = -\frac{f}{\rho}\frac{\sigma_\theta}{\partial z} \tag{7.4}$$

7.7.2.2 Surface vertical potential vorticity fluxes

These quantities are useful in mode water studies because of the impermeability theorem which states that for a given potential density layer (embedding a mode water), the integrated PV only changes through surface input/output.

Vertical PV fluxes due to frictional and diabatic processes are given by:

$$J_z^B = -\frac{f}{h}\left(\frac{\alpha Q_{net}}{C_w} - \rho_0 \beta S_{net}\right) \tag{7.5}$$

$$J_z^F = \frac{1}{\rho\delta_e}\vec{k} \times \tau \cdot \nabla\sigma_m \tag{7.6}$$

These components can be computed with the package. Details on the variables definition and the way these fluxes are derived can be found in section 7.7.5.

We now give some simple explanations about these fluxes and how they can reduce the PV level of an oceanic potential density layer.

Diabatic process Let's take the PV flux due to surface buoyancy forcing from Eq.7.5 and simplify it as:

$$J_z^B \simeq -\frac{\alpha f}{hC_w} Q_{net} \tag{7.7}$$

When the net surface heat flux Q_{net} is upward i.e. negative and cooling the ocean (buoyancy loss), surface density will increase, triggering mixing which reduces the stratification and then the PV.

$$\begin{aligned} Q_{net} &< 0 \text{ (upward, cooling)} \\ J_z^B &> 0 \text{ (upward)} \\ -\rho^{-1}\nabla \cdot J_z^B &< 0 \text{ (PV flux divergence)} \\ PV &\searrow \text{ where } Q_{net} < 0 \end{aligned}$$

Frictional process: "Down-front" wind-stress Now let's take the PV flux due to the "wind-driven buoyancy flux" from Eq.7.6 and simplify it as:

$$\begin{aligned} J_z^F &= \frac{1}{\rho\delta_e}\left(\tau_x \frac{\partial\sigma}{\partial y} - \tau_y \frac{\partial\sigma}{\partial x}\right) \\ J_z^F &\simeq \frac{1}{\rho\delta_e}\tau_x \frac{\partial\sigma}{\partial y} \end{aligned} \tag{7.8}$$

When the wind is blowing from the east above the Gulf Stream (a region of high meridional density gradient), it induces an advection of dense water from the northern side of the GS to the southern side through Ekman currents. Then, it induces a "wind-driven" buoyancy lost and mixing which reduces the stratification and the PV.

$$\begin{aligned} \vec{k} \times \tau \cdot \nabla\sigma &> 0 \text{ ("Down-front" wind)} \\ J_z^F &> 0 \text{ (upward)} \\ -\rho^{-1}\nabla \cdot J_z^F &< 0 \text{ (PV flux divergence)} \\ PV &\searrow \text{ where } \vec{k} \times \tau \cdot \nabla\sigma > 0 \end{aligned}$$

Diabatic versus frictional processes A recent debate in the community arose about the relative role of these processes. Taking the ratio of Eq.7.5 and Eq.7.6 leads to:

$$\begin{aligned} \frac{J_z^F}{J_Z^B} &= \frac{\frac{1}{\rho\delta_e}\vec{k} \times \tau \cdot \nabla\sigma}{-\frac{f}{h}\left(\frac{\alpha Q_{net}}{C_w} - \rho_0 \beta S_{net}\right)} \\ &\simeq \frac{Q_{Ek}/\delta_e}{Q_{net}/h} \end{aligned} \tag{7.9}$$

where appears the lateral heat flux induced by Ekman currents:

$$\begin{aligned} Q_{Ek} &= -\frac{C_w}{\alpha\rho f}\vec{k} \times \tau \cdot \nabla\sigma \\ &= \frac{C_w}{\alpha}\delta_e u_{\vec{E}k} \cdot \nabla\sigma \end{aligned} \tag{7.10}$$

which can be computed with the package. In the aim of comparing both processes, it will be useful to plot surface net and lateral Ekman-induced heat fluxes together with PV fluxes.

7.7.3 Key routines

- `A_compute_potential_density.m`: Compute the potential density field. Requires the potential temperature and salinity (either total or anomalous) and produces one output file with the potential density field (file prefix is `SIGMATHETA`). The routine uses `densjmd95.m` a Matlab counterpart of the MITgcm built-in function to compute the density.

- `B_compute_relative_vorticity.m`: Compute the three components of the relative vorticity defined in Eq. (7.1). Requires the two horizontal velocity components and produces three output files with the three components (files prefix are `OMEGAX`, `OMEGAY` and `ZETA`).

- `C_compute_potential_vorticity.m`: Compute the potential vorticity without the negative ratio by the density. Two options are possible in order to compute either the full component (term into parenthesis in Eq. 7.3) or the planetary component ($f\partial_z\sigma_\theta$ in Eq. 7.4). Requires the relative vorticity components and the potential density, and produces one output file with the potential vorticity (file prefix is `PV` for the full term and `splPV` for the planetary component).

- `D_compute_potential_vorticity.m`: Load the field computed with `C_comp...` and divide it by $-\rho$ to obtain the correct potential vorticity. Require the density field and after loading, overwrite the file with prefix `PV` or `splPV`.

- `compute_density.m`: Compute the density ρ from the potential temperature and the salinity fields.

- `compute_JFz.m`: Compute the surface vertical PV flux due to frictional processes. Requires the wind stress components, density, potential density and Ekman layer depth (all of them, except the wind stress, may be computed with the package), and produces one output file with the PV flux J_z^F (see Eq. 7.6) and with `JFz` as a prefix.

- `compute_JBz.m`: Compute the surface vertical PV flux due to diabatic processes as:

$$J_z^B = -\frac{f}{h}\frac{\alpha Q_{net}}{C_w}$$

  which is a simplified version of the full expression given in Eq. (7.5). Requires the net surface heat flux and the mixed layer depth (of which an estimation can be computed with the package), and produces one output file with the PV flux J_z^B and with `JBz` as a prefix.

- `compute_QEk.m`: Compute the horizontal heat flux due to Ekman currents from the PV flux induced by frictional forces as:

$$Q_{Ek} = -\frac{C_w \delta_e}{\alpha f} J_z^F$$

  Requires the PV flux due to frictional forces and the Ekman layer depth, and produces one output with the heat flux and with `QEk` as a prefix.

- `eg_main_getPV`: A complete example of how to set up a master routine able to compute everything from the package.

7.7.4 Technical details

7.7.4.1 File name

A file name is formed by three parameters which need to be set up as global variables in Matlab before running any routines. They are:

- the prefix, ie the variable name (`netcdf_UVEL` for example). This parameter is specified in the help section of all diagnostic routines.

- `netcdf_domain`: the geographical domain.

- `netcdf_suff`: the netcdf extension (nc or cdf for example).

Then, for example, if the calling Matlab routine had set up:

```
global netcdf_THETA netcdf_SALTanom netcdf_domain netcdf_suff
netcdf_THETA    = 'THETA';
netcdf_SALTanom = 'SALT';
netcdf_domain   = 'north_atlantic';
netcdf_suff     = 'nc';
```

the routine `A_compute_potential_density.m` to compute the potential density field, will look for the files:

```
THETA.north_atlantic.nc
SALT.north_atlantic.nc
```

and the output file will automatically be: `SIGMATHETA.north_atlantic.nc`.

Otherwise indicated, output file prefix cannot be changed.

7.7.4.2 Path to file

All diagnostic routines look for input files in a subdirectory (relative to the Matlab routine directory) called `./netcdf-files` which in turn, is supposed to contain subdirectories for each set of fields. For example, computing the potential density for the timestep 12H00 02/03/2005 will require a subdirectory with the potential temperature and salinity files like:

```
./netcdf-files/200501031200/THETA.north_atlantic.nc
./netcdf-files/200501031200/SALT.north_atlantic.nc
```

The output file `SIGMATHETA.north_atlantic.nc` will be created in `./netcdf-files/200501031200/`. All diagnostic routines take as argument the name of the timestep subdirectory into `./netcdf-files`.

7.7.4.3 Grids

With MITgcm numerical outputs, velocity and tracer fields may not be defined on the same grid. Usually, UVEL and VVEL are defined on a C-grid but when interpolated from a cube-sphere simulation they are defined on a A-grid. When it is needed, routines allow to set up a global variable which define the grid to use.

7.7.5 Notes on the flux form of the PV equation and vertical PV fluxes

7.7.5.1 Flux form of the PV equation

The conservative flux form of the potential vorticity equation is:

$$\frac{\partial \rho Q}{\partial t} + \nabla \cdot \vec{J} \quad = \quad 0 \tag{7.11}$$

where the potential vorticity Q is given by the Eq.7.3.

The generalized flux vector of potential vorticity is:

$$\vec{J} \quad = \quad \rho Q \vec{u} + \vec{N_Q} \tag{7.12}$$

which allows to rewrite Eq.7.11 as:

$$\frac{DQ}{dt} \quad = \quad -\frac{1}{\rho}\nabla \cdot \vec{N_Q} \tag{7.13}$$

where the nonadvective PV flux $\vec{N_Q}$ is given by:

$$\vec{N_Q} \quad = \quad -\frac{\rho_0}{g} B \vec{\omega_a} + \vec{F} \times \nabla \sigma_\theta \tag{7.14}$$

Its first component is linked to the buoyancy forcing[2]:

$$B \quad = \quad -\frac{g}{\rho_o}\frac{D\sigma_\theta}{dt} \tag{7.16}$$

and the second one to the nonconservative body forces per unit mass:

$$\vec{F} \quad = \quad \frac{D\vec{u}}{dt} + 2\Omega \times \vec{u} + \nabla p \tag{7.17}$$

[2] Note that introducing B into Eq.7.14 yields to:

$$\vec{N_Q} \quad = \quad \omega_a \frac{D\sigma_\theta}{dt} + \vec{F} \times \nabla \sigma_\theta \tag{7.15}$$

7.7.5.2 Determining the PV flux at the ocean's surface

In the context of mode water study, we're particularly interested in how the PV may be reduced by surface PV fluxes because a mode water is characterised by a low PV level. Considering the volume limited by two $iso-\sigma_\theta$, PV flux is limited to surface processes and then vertical component of $\vec{N_Q}$. It is supposed that B and $\vec{F}$ will only be nonzero in the mixed layer (of depth h and variable density σ_m) exposed to mechanical forcing by the wind and buoyancy fluxes through the ocean's surface.

Given the assumption of a mechanical forcing confined to a thin surface Ekman layer (of depth δ_e, eventually computed by the package) and of hydrostatic and geostrophic balances, we can write:

$$\vec{u_g} = \frac{1}{\rho f}\vec{k} \times \nabla p \tag{7.18}$$

$$\frac{\partial p_m}{\partial z} = -\sigma_m g \tag{7.19}$$

$$\frac{\partial \sigma_m}{\partial t} + \vec{u}_m \cdot \nabla \sigma_m = -\frac{\rho_0}{g} B \tag{7.20}$$

where:

$$\vec{u}_m = \vec{u}_g + \vec{u}_{Ek} + o(R_o) \tag{7.21}$$

is the full velocity field composed by the geostrophic current $\vec{u}_g$ and the Ekman drift:

$$\vec{u}_{Ek} = -\frac{1}{\rho f}\vec{k} \times \frac{\partial \tau}{\partial z} \tag{7.22}$$

(where τ is the wind stress) and last by other ageostrophic components of $o(R_o)$ which are neglected.

Partitioning the buoyancy forcing as:

$$B = B_g + B_{Ek} \tag{7.23}$$

and using Eq.7.21 and Eq.7.22, the Eq.7.20 becomes:

$$\frac{\partial \sigma_m}{\partial t} + \vec{u}_g \cdot \nabla \sigma_m = -\frac{\rho_0}{g} B_g \tag{7.24}$$

revealing the "wind-driven buoyancy forcing":

$$B_{Ek} = \frac{g}{\rho_0}\frac{1}{\rho f}\left(\vec{k} \times \frac{\partial \tau}{\partial z}\right) \cdot \nabla \sigma_m \tag{7.25}$$

Note that since:

$$\frac{\partial B_g}{\partial z} = \frac{\partial}{\partial z}\left(-\frac{g}{\rho_0}\vec{u_g} \cdot \nabla \sigma_m\right) = -\frac{g}{\rho_0}\frac{\partial \vec{u_g}}{\partial z} \cdot \nabla \sigma_m = 0 \tag{7.26}$$

B_g must be uniform throughout the depth of the mixed layer and then being related to the surface buoyancy flux by integrating Eq.7.23 through the mixed layer:

$$\int_{-h}^{0} B\,dz = hB_g + \textstyle\int_{-h}^{0} B_{Ek}\,dz = \mathcal{B}_{in} \tag{7.27}$$

where $\mathcal{B}_{in}$ is the vertically integrated surface buoyancy (in)flux:

$$\mathcal{B}_{in} = \frac{g}{\rho_o}\left(\frac{\alpha Q_{net}}{C_w} - \rho_0 \beta S_{net}\right) \tag{7.28}$$

with $\alpha \simeq 2.5 \times 10^{-4}\,K^{-1}$ the thermal expansion coefficient (computed by the package otherwise), $C_w = 4187 J.kg^{-1}.K^{-1}$ the specific heat of seawater, $Q_{net}[W.m^{-2}]$ the net heat surface flux (positive downward, warming the ocean), $\beta[PSU^{-1}]$ the saline contraction coefficient, and $S_{net} = S * (E-P)[PSU.m.s^{-1}]$ the net freshwater surface flux with $S[PSU]$ the surface salinity and $(E-P)[m.s^{-1}]$ the fresh water flux.

Introducing the body force in the Ekman layer:

$$F_z = \frac{1}{\rho}\frac{\partial \tau}{\partial z} \tag{7.29}$$

the vertical component of Eq.7.14 is:

$$\begin{aligned}
\vec{N_Q}_z &= -\frac{\rho_0}{g}(B_g + B_{Ek})\omega_z + \frac{1}{\rho}\left(\frac{\partial \tau}{\partial z} \times \nabla\sigma_\theta\right)\cdot\vec{k} \\
&= -\frac{\rho_0}{g}B_g\omega_z - \frac{\rho_0}{g}\left(\frac{g}{\rho_0}\frac{1}{\rho f}\vec{k}\times\frac{\partial \tau}{\partial z}\cdot\nabla\sigma_m\right)\omega_z + \frac{1}{\rho}\left(\frac{\partial \tau}{\partial z}\times\nabla\sigma_\theta\right)\cdot\vec{k} \\
&= -\frac{\rho_0}{g}B_g\omega_z + \left(1 - \frac{\omega_z}{f}\right)\left(\frac{1}{\rho}\frac{\partial \tau}{\partial z}\times\nabla\sigma_\theta\right)\cdot\vec{k}
\end{aligned} \tag{7.30}$$

and given the assumption that $\omega_z \simeq f$, the second term vanishes and we obtain:

$$\vec{N_Q}_z = -\frac{\rho_0}{g}fB_g \tag{7.31}$$

Note that the wind-stress forcing does not appear explicitly here but is implicit in B_g through Eq.7.27: the buoyancy forcing B_g is determined by the difference between the integrated surface buoyancy flux $\mathcal{B}_{in}$ and the integrated "wind-driven buoyancy forcing":

$$\begin{aligned}
B_g &= \frac{1}{h}\left(\mathcal{B}_{in} - \int_{-h}^{0} B_{Ek}dz\right) \\
&= \frac{1}{h}\frac{g}{\rho_0}\left(\frac{\alpha Q_{net}}{C_w} - \rho_0\beta S_{net}\right) - \frac{1}{h}\int_{-h}^{0}\frac{g}{\rho_0}\frac{1}{\rho f}\vec{k}\times\frac{\partial \tau}{\partial z}\cdot\nabla\sigma_m dz \\
&= \frac{1}{h}\frac{g}{\rho_0}\left(\frac{\alpha Q_{net}}{C_w} - \rho_0\beta S_{net}\right) - \frac{g}{\rho_0}\frac{1}{\rho f\delta_e}\vec{k}\times\tau\cdot\nabla\sigma_m
\end{aligned} \tag{7.32}$$

Finally, from Eq.7.14, the vertical surface flux of PV may be written as:

$$\vec{N_Q}_z = J_z^B + J_z^F \tag{7.33}$$

$$J_z^B = -\frac{f}{h}\left(\frac{\alpha Q_{net}}{C_w} - \rho_0\beta S_{net}\right) \tag{7.34}$$

$$J_z^F = \frac{1}{\rho\delta_e}\vec{k}\times\tau\cdot\nabla\sigma_m \tag{7.35}$$

Chapter 8

Ocean State Estimation Packages

This chapter describes packages that have been introduced for ocean state estimation purposes and in relation with automatic differentiation (see Chapter 5)

8.1 ECCO: model-data comparisons using gridded data sets

The functionalities implemented in `pkg/ecco` are: (1) output time-averaged model fields to compare with gridded data sets; (2) compute normalized model-data distances (i.e., cost functions); (3) compute averages and transports (i.e., integrals). The former is achieved as the model runs forwards in time whereas the others occur after time-integration has completed. Following *Forget et al.* [2015] the total cost function is formulated generically as

$$\mathcal{J}(\vec{u}) = \sum_i \alpha_i \left(\vec{d}_i^T R_i^{-1} \vec{d}_i \right) + \sum_j \beta_j \vec{u}^T \vec{u}, \tag{8.1}$$

$$\vec{d}_i = \mathcal{P}(\vec{m}_i - \vec{o}_i), \tag{8.2}$$

$$\vec{m}_i = \mathcal{SDM}(\vec{v}), \tag{8.3}$$

$$\vec{v} = \mathcal{Q}(\vec{u}), \tag{8.4}$$

$$\vec{u} = \mathcal{R}(\vec{u}') \tag{8.5}$$

using symbols defined in table 8.1. Per Eq. (8.3) model counterparts ($\vec{m}_i$) to observational data ($\vec{o}_i$) derive from adjustable model parameters ($\vec{v}$) through model dynamics integration ($\mathcal{M}$), diagnostic calculations ($\mathcal{D}$), and averaging in space and time ($\mathcal{S}$). Alternatively $\mathcal{S}$ stands for subsampling in space and time (section 8.2). Plain model-data misfits ($\vec{m}_i - \vec{o}_i$) can be penalized directly in Eq. (8.1) but penalized misfits ($\vec{d}_i$) more generally derive from $\vec{m}_i - \vec{o}_i$ through the generic $\mathcal{P}$ post-processor (Eq. (8.2)). Eqs. (8.4)-(8.5) pertain to model control parameter adjustment capabilities described in section 8.3.

8.1.1 Generic Cost Function

The parameters available for configuring generic cost function terms in `data.ecco` are given in table 8.2 and examples of possible specifications are available in:

- MITgcm_contrib/verification_other/global_oce_cs32/input/data.ecco
- MITgcm_contrib/verification_other/global_oce_cs32/input_ad.sens/data.ecco
- MITgcm_contrib/gael/verification/global_oce_llc90/input.ecco_v4/data.ecco

The gridded observation file name is specified by `gencost_datafile`. Observational time series may be provided as on big file or split into yearly files finishing in '_1992', '_1993', etc. The corresponding $\vec{m}_i$ physical variable is specified via the `gencost_barfile` root (see table 8.3). A file named as specified by `gencost_barfile` gets created where averaged fields are written progressively as the model steps forward in time. After the final time step this file is re-read by `cost_generic.F` to compute the corresponding cost function term. If `gencost_outputlevel` = 1 and `gencost_name`='foo' then `cost_generic.F` outputs model-data misfit fields (i.e., $\vec{d}_i$) to a file named 'misfit_foo.data' for offline analysis and visualization.

symbol	definition
$\vec{u}$	vector of nondimensional control variables
$\vec{v}$	vector of dimensional control variables
α_i, β_j	misfit and control cost function multipliers (1 by default)
R_i	data error covariance matrix (R_i^{-1} are weights)
$\vec{d}_i$	a set of model-data differences
$\vec{o}_i$	observational data vector
$\vec{m}_i$	model counterpart to $\vec{o}_i$
$\mathcal{P}$	post-processing operator (e.g., a smoother)
$\mathcal{M}$	forward model dynamics operator
$\mathcal{D}$	diagnostic computation operator
$\mathcal{S}$	averaging/subsampling operator
$\mathcal{Q}$	Pre-processing operator
$\mathcal{R}$	Pre-conditioning operator

Table 8.1: Symbol definitions for pkg/ecco and pkg/ctrl generic cost functions.

In the current implementation, model-data error covariance matrices R_i omit non-diagonal terms. Specifying R_i thus boils down to providing uncertainty fields (σ_i such that $R_i = \sigma_i^2$) in a file specified via `gencost_errfile`. By default σ_i is assumed to be time-invariant but a σ_i time series of the same length as the $\vec{o}_i$ time series can be provided using the `variaweight` option (table 8.4). By default cost functions are quadratic but $\vec{d}_i^T R_i^{-1} \vec{d}_i$ can be replaced with $R_i^{-1/2} \vec{d}_i$ using the `nosumsq` option (table 8.4).

In principle, any averaging frequency should be possible, but only 'day', 'month', 'step', and 'const' are implemented for `gencost_avgperiod`. If two different averaging frequencies are needed for a variable used in multiple cost function terms (e.g., daily and monthly) then an extension starting with '_' should be added to `gencost_barfile` (such as '_day' and '_mon'). [1] If two cost function terms use the same variable and frequency, however, then using a common `gencost_barfile` saves disk space.

Climatologies of $\vec{m}_i$ can be formed from the time series of model averages in order to compare with climatologies of $\vec{o}_i$ by activating the 'clim' option via `gencost_preproc` and setting the corresponding `gencost_preproc_i` integer parameter to the number of records (i.e., a # of months, days, or time steps) per climatological cycle. The generic post-processor ($\mathcal{P}$ in Eq. (8.2)) also allows model-data misfits to be, for example, smoothed in space by setting `gencost_posproc` to 'smooth' and specifying the smoother parameters via `gencost_posproc_c` and `gencost_posproc_i` (see table 8.4). Other options associated with the computation of Eq. (8.1) are summarized in table 8.4 and further discussed below. Multiple `gencost_preproc` / `gencost_posproc` options may be specified per cost term.

In general the specification of `gencost_name` is optional, has no impact on the end-result, and only serves to distinguish between cost function terms amongst the model output (STDOUT.0000, STDERR.0000, costfunction000, misfit*.data). Exceptions listed in table 8.6 however activate alternative cost function codes (in place of `cost_generic.F`) described in section 8.1.3. In this section and in table 8.3 (unlike in other parts of the manual) 'zonal' / 'meridional' are to be taken literally and these components are centered (i.e., not at the staggered model velocity points). Preparing gridded velocity data sets for use in cost functions thus boils down to interpolating them to XC / YC.

[1] ecco_check may be missing a test for conflicting names...

parameter	type	function
`gencost_name`	character(*)	Name of cost term
`gencost_barfile`	character(*)	File to receive model counterpart $\vec{m}_i$ (see table 8.3)
`gencost_datafile`	character(*)	File containing observational data $\vec{o}_i$
`gencost_avgperiod`	character(5)	Averaging period for $\vec{o}_i$ and $\vec{m}_i$ (see text)
`gencost_outputlevel`	integer	Greater than 0 will output misfit fields
`gencost_errfile`	character(*)	Uncertainty field name (not used in section 8.1.2)
`gencost_mask`	character(*)	Mask file name root (used only in section 8.1.2)
`mult_gencost`	real	Multiplier α_i (default: 1)
`gencost_preproc`	character(*)	Preprocessor names
`gencost_preproc_c`	character(*)	Preprocessor character arguments
`gencost_preproc_i`	integer(*)	Preprocessor integer arguments
`gencost_preproc_r`	real(*)	Preprocessor real arguments
`gencost_posproc`	character(*)	Post-processor names
`gencost_posproc_c`	character(*)	Post-processor character arguments
`gencost_posproc_i`	integer(*)	Post-processor integer arguments
`gencost_posproc_r`	real(*)	Post-processor real arguments
`gencost_spmin`	real	Data less than this value will be omitted
`gencost_spmax`	real	Data greater than this value will be omitted
`gencost_spzero`	real	Data points equal to this value will be omitted
`gencost_startdate1`	integer	Start date of observations (YYYMMDD)
`gencost_startdate2`	integer	Start date of observations (HHMMSS)
`gencost_is3d`	logical	Needs to be true for 3D fields
`gencost_enddate1`	integer	Not fully implemented (used only in sec. 8.1.3)
`gencost_enddate2`	integer	Not fully implemented (used only in sec. 8.1.3)

Table 8.2: Parameters in `ecco_gencost_nml` namelist in `data.ecco`. All parameters are vectors of length `NGENCOST` (the # of available cost terms) except for `gencost_*proc*` are arrays of size `NGENPPROC`$\times$`NGENCOST`. Notes: `gencost_is3d` is automatically reset to true in all 3D cases in table 8.3; NGENCOST (20) and NGENPPROC (10) can be changed in ecco.h only at compile time.

variable name	description	remarks
`m_eta`	sea surface height	free surface + ice + global steric correction
`m_sst`	sea surface temperature	first level potential temperature
`m_sss`	sea surface salinity	first level salinity
`m_bp`	bottom pressure	phiHydLow
`m_siarea`	sea-ice area	from pkg/seaice
`m_siheff`	sea-ice effective thickness	from pkg/seaice
`m_sihsnow`	snow effective thickness	from pkg/seaice
`m_theta`	potential temperature	three-dimensional
`m_salt`	salinity	three-dimensional
`m_UE`	zonal velocity	three-dimensional
`m_VN`	meridional velocity	three-dimensional
`m_ustress`	zonal wind stress	from pkg/exf
`m_vstress`	meridional wind stress	from pkg/exf
`m_uwind`	zonal wind	from pkg/exf
`m_vwind`	meridional wind	from pkg/exf
`m_atemp`	atmospheric temperature	from pkg/exf
`m_aqh`	atmospheric specific humidity	from pkg/exf
`m_precip`	precipitation	from pkg/exf
`m_swdown`	downward shortwave	from pkg/exf
`m_lwdown`	downward longwave	from pkg/exf
`m_wspeed`	wind speed	from pkg/exf
`m_diffkr`	vertical/diapycnal diffusivity	three-dimensional, constant
`m_kapgm`	GM diffusivity	three-dimensional, constant
`m_kapredi`	isopycnal diffusivity	three-dimensional, constant
`m_geothermalflux`	geothermal heat flux	constant
`m_bottomdrag`	bottom drag	constant

Table 8.3: Implemented `gencost_barfile` options (as of checkpoint 65z) that can be used via `cost_generic.F` (section 8.1.1). An extension starting with '_' can be appended at the end of the variable name to distinguish between separate cost function terms. Note: the 'm_eta' formula depends on the `ATMOSPHERIC_LOADING` and `ALLOW_PSBAR_STERIC` compile time options and 'useRealFreshWaterFlux' run time parameter.

name	description	specs needed via `gencost_preproc_i`, `_r`, or `_c`
`gencost_preproc`		
`clim`	Use climatological misfits	integer: no. of records per climatological cycle
`mean`	Use time mean of misfits	—
`anom`	Use anomalies from time mean	—
`variaweight`	Use time-varying weight W_i	—
`nosumsq`	Use linear misfits	—
`factor`	Multiply $\vec{m}_i$ by a scaling factor	real: the scaling factor
`gencost_posproc`		
`smooth`	Smooth misfits	character: smoothing scale file integer: smoother # of time steps

Table 8.4: `gencost_preproc` and `gencost_posproc` options implemented as of checkpoint 65z. Note: the distinction between `gencost_preproc` and `gencost_posproc` seems unclear and may be revisited in the future.

8.1.2 Generic Integral Function

The functionality described in this section is operated by `cost_gencost_boxmean.F`. It is primarily aimed at obtaining a mechanistic understanding of a chosen physical variable via adjoint sensitivity computations (see Chapter 5) as done for example in *Marotzke et al.* [1999]; *Heimbach et al.* [2011]; *Fukumori et al.* [2015]. Thus the quadratic term in Eq. 8.1 ($\vec{d}_i^T R_i^{-1} \vec{d}_i$) is by default replaced with a d_i scalar[2] that derives from model fields through a generic integral formula (Eq. 8.3). The specification of `gencost_barfile` again selects the physical variable type. Current valid options to use `cost_gencost_boxmean.F` are reported in table 8.5. A suffix starting with `'_'` can again be appended to `gencost_barfile`.

The integral formula is defined by masks provided via binary files which names are specified via `gencost_mask`. There are two cases: (1) if `gencost_mask = 'foo_mask'` and `gencost_barfile` is of the 'm_boxmean*' type then the model will search for horizontal, vertical, and temporal mask files named `foo_maskC`, `foo_maskK`, and `foo_maskT`; (2) if instead `gencost_barfile` is of the 'm_horflux_*' type then the model will search for `foo_maskW`, `foo_maskS`, `foo_maskK`, and `foo_maskT`.

The 'C' mask or the 'W' / 'S' masks are expected to be two-dimensional fields. The 'K' and 'T' masks (both optional; all 1 by default) are expected to be one-dimensional vectors. The 'K' vector length should match Nr. The 'T' vector length should match the # of records that the specification of `gencost_avgperiod` implies but there is no restriction on its values. In case #1 ('m_boxmean*') the 'C' and 'K' masks should consists of +1 and 0 values and a volume average will be computed accordingly. In case #2 ('m_horflux*') the 'W', 'S', and 'K' masks should consists of +1, -1, and 0 values and an integrated horizontal transport (or overturn) will be computed accordingly.

variable name	description	remarks
`m_boxmean_theta`	mean of theta over box	specify box
`m_boxmean_salt`	mean of salt over box	specify box
`m_boxmean_eta`	mean of SSH over box	specify box
`m_horflux_vol`	volume transport through section	specify transect

Table 8.5: Implemented `gencost_barfile` options (as of checkpoint 65z) that can be used via `cost_gencost_boxmean.F` (section 8.1.2).

8.1.3 Custom Cost Functions

This section (very much a work in progress...) pertains to the special cases of `cost_gencost_bpv4.F`, `cost_gencost_seaicev4.F`, `cost_gencost_sshv4.F`, `cost_gencost_sstv4.F`, and `cost_gencost_transp.F`. The cost_gencost_transp.F function can be used to compute a transport of volume, heat, or salt through a specified section (non quadratic cost function). To this end one sets `gencost_name = 'transp*'`, where `*` is an optional suffix starting with `'_'`, and set `gencost_barfile` to one of `m_trVol`, `m_trHeat`, and `m_trSalt`.

8.1.4 Key Routines

TBA... `ecco_readparms.F`, `ecco_check.F`, `ecco_summary.F`, ... `cost_generic.F`, `cost_gencost_boxmean.F`,

`ecco_toolbox.F`, ... `ecco_phys.F`, `cost_gencost_customize.F`, `cost_averagesfields.F`, ...

8.1.5 Compile Options

TBA... ALLOW_GENCOST_CONTRIBUTION, ALLOW_GENCOST3D, ... ALLOW_PSBAR_STERIC,

ALLOW_SHALLOW_ALTIMETRY, ALLOW_HIGHLAT_ALTIMETRY, ... ALLOW_PROFILES_CONTRIBUTION, ... ALLOW_ECCO_OLD_FC_PRINT, ... ECCO_CTRL_DEPRECATED, ... packages required for some

functionalities: smooth, profiles, ctrl

[2] The quadratic option in fact does not yet exist in `cost_gencost_boxmean.F`...

name	description	remarks
`sshv4-mdt`	sea surface height	mean dynamic topography (SSH - geod)
`sshv4-tp`	sea surface height	Along-Track Topex/Jason SLA (level 3)
`sshv4-ers`	sea surface height	Along-Track ERS/Envisat SLA (level 3)
`sshv4-gfo`	sea surface height	Along-Track GFO class SLA (level 3)
`sshv4-lsc`	sea surface height	Large-Scale SLA (from the above)
`sshv4-gmsl`	sea surface height	Global-Mean SLA (from the above)
`bpv4-grace`	bottom pressure	GRACE maps (level 4)
`sstv4-amsre`	sea surface temperature	Along-Swath SST (level 3)
`sstv4-amsre-lsc`	sea surface temperature	Large-Scale SST (from the above)
`si4-cons`	sea ice concentration	needs sea-ice adjoint (level 4)
`si4-deconc`	model sea ice deficiency	proxy penalty (from the above)
`si4-exconc`	model sea ice excess	proxy penalty (from the above)
`transp_trVol`	volume transport	specify section as in section 8.1.2
`transp_trHeat`	heat transport	specify section as in section 8.1.2
`transp_trSalt`	salt transport	specify section as in section 8.1.2

Table 8.6: Pre-defined `gencost_name` special cases (as of checkpoint 65z; section 8.1.3).

8.2 PROFILES: model-data comparisons at observed locations

The purpose of pkg/profiles is to allow sampling of MITgcm runs according to a chosen pathway (after a ship or a drifter, along altimeter tracks, etc.), typically leading to easy model-data comparisons. Given input files that contain positions and dates, pkg/profiles will interpolate the model trajectory at the observed location. In particular, pkg/profiles can be used to do model-data comparison online and formulate a least-squares problem (ECCO application).

pkg/profiles is associated with three CPP keys:
(k1) ALLOW_PROFILES
(k2) ALLOW_PROFILES_GENERICGRID
(k3) ALLOW_PROFILES_CONTRIBUTION
k1 switches the package on. By default, pkg/profiles assumes a regular lat-long grid. For other grids such as the cubed sphere, k2 and pre-processing (see below) are necessary. k3 switches the least-squares application on (pkg/ecco needed). pkg/profiles requires needs pkg/cal and netcdf libraries.

The namelist (data.profiles) is illustrated in table 8.7. This example includes two input netcdf files name (ARGOifremer_r8.nc and XBT_v5.nc are to be provided) and *cost function* multipliers (for least-squares only). The first index is a file number and the second index (in mult* only) is a variable number. By convention, the variable number is an integer ranging 1 to 6: temperature, salinity, zonal velocity, meridional velocity, sea surface height anomaly, and passive tracer.

The input file structure is illustrated in table 8.8. To create such files, one can use the netcdf_ecco_create.m matlab script, which can be checked out of
MITgcm_contrib/gael/profilesMatlabProcessing/
along with a full suite of matlab scripts associated with pkg/profiles. At run time, each file is scanned to determine which variables are included; these will be interpolated. The (final) output file structure is similar but with interpolated model values in prof_T etc., and it contains model mask variables (e.g. prof_Tmask). The very model output consists of one binary (or netcdf) file per processor. The final netcdf output is to be built from those using netcdf_ecco_recompose.m (offline).

When the k2 option is used (e.g. for cubed sphere runs), the input file is to be completed with interpolation grid points and coefficients computed offline using netcdf_ecco_GenericgridMain.m. Typically, you would first provide the standard namelist and files. After detecting that interpolation information is missing, the model will generate special grid files (profilesXCincl1PointOverlap* etc.) and then stop.

You then want to run netcdf_ecco_GenericgridMain.m using the special grid files. *This operation could eventually be inlined.*

```
#
# ******************
# PROFILES cost function
# ******************
&PROFILES_NML
#
profilesfiles(1)= 'ARGOifremer_r8',
mult_profiles(1,1) = 1.,
mult_profiles(1,2) = 1.,
profilesfiles(2)= 'XBT_v5',
mult_profiles(2,1) = 1.,
#
/
```

Table 8.7: pkg/profiles: data.profiles example.

```
netcdf XBT_v5 {
dimensions:
  iPROF = 278026 ;
  iDEPTH = 55 ;
  lTXT = 30 ;
variables:
  double depth(iDEPTH) ;
   depth:units = "meters" ;
  double prof_YYYYMMDD(iPROF) ;
   prof_YYYYMMDD:missing_value = -9999. ;
   prof_YYYYMMDD:long_name = "year (4 digits), month (2 digits), day (2 digits)" ;
  double prof_HHMMSS(iPROF) ;
   prof_HHMMSS:missing_value = -9999. ;
   prof_HHMMSS:long_name = "hour (2 digits), minute (2 digits), seconde (2 digits)" ;
  double prof_lon(iPROF) ;
   prof_lon:units = "(degree E)" ;
   prof_lon:missing_value = -9999. ;
  double prof_lat(iPROF) ;
   prof_lat:units = "(degree N)" ;
   prof_lat:missing_value = -9999. ;
  char prof_descr(iPROF, lTXT) ;
   prof_descr:long_name = "profile description" ;
  double prof_T(iPROF, iDEPTH) ;
   prof_T:long_name = "potential temperature" ;
   prof_T:units = "degree Celsius" ;
   prof_T:missing_value = -9999. ;
  double prof_Tweight(iPROF, iDEPTH) ;
   prof_Tweight:long_name = "weights" ;
   prof_Tweight:units = "(degree Celsius)^2" ;
   prof_Tweight:missing_value = -9999. ;
}
```

Table 8.8: pkg/profiles: input file structure as would be shown by "ncdump -h ARGOifremer_r8.nc".

8.3 CTRL: Model Parameter Adjustment Capability

The parameters available for configuring generic cost terms in `data.ctrl` are given in table 8.9.

parameter	type	function
`xx_gen*_file`	character(*)	Name of control. Prefix from table 8.10 + suffix.
`xx_gen*_weight`	character(*)	Weights in the form of $\sigma_{\vec{u}_j}^{-2}$
`xx_gen*_bounds`	real(5)	Apply bounds
`xx_gen*_preproc`	character(*)	Control preprocessor(s) (see table 8.11)
`xx_gen*_preproc_c`	character(*)	Preprocessor character arguments
`xx_gen*_preproc_i`	integer(*)	Preprocessor integer arguments
`xx_gen*_preproc_r`	real(*)	Preprocessor real arguments
`gen*Precond`	real	Preconditioning factor (= 1 by default)
`mult_gen*`	real	Cost function multiplier β_j (= 1 by default)
`xx_gentim2d_period`	real	Frequency of adjustments (in seconds)
`xx_gentim2d_startdate1`	integer	Adjustment start date
`xx_gentim2d_startdate2`	integer	Default: model start date
`xx_gentim2d_cumsum`	logical	Accumulate control adjustments
`xx_gentim2d_glosum`	logical	Global sum of adjustment (output is still 2D)

Table 8.9: Parameters in `ctrl_nml_genarr` namelist in `data.ctrl`. The `*` can be replaced by `arr2d`, `arr3d`, or `tim2d` for time-invariant two and three dimensional controls and time-varying 2D controls, respectively. Parameters for `genarr2d`, `genarr3d`, and `gentime2d` are arrays of length `maxCtrlArr2D`, `maxCtrlArr3D`, and `maxCtrlTim2D`, respectively, with one entry per term in the cost function.

	name	description
2D, time-invariant controls	`genarr2d`	
	`xx_etan`	initial sea surface height
	`xx_bottomdrag`	bottom drag
	`xx_geothermal`	geothermal heat flux
3D, time-invariant controls	`genarr3d`	
	`xx_theta`	initial potential temperature
	`xx_salt`	initial salinity
	`xx_kapgm`	GM coefficient
	`xx_kapredi`	isopycnal diffusivity
	`xx_diffkr`	diapycnal diffusivity
2D, time-varying controls	`gentim2D`	
	`xx_atemp`	atmospheric temperature
	`xx_aqh`	atmospheric specific humidity
	`xx_swdown`	downward shortwave
	`xx_lwdown`	downward longwave
	`xx_precip`	precipitation
	`xx_uwind`	zonal wind
	`xx_vwind`	meridional wind
	`xx_tauu`	zonal wind stress
	`xx_tauv`	meridional wind stress
	`xx_gen_precip`	globally averaged precipitation?

Table 8.10: Generic control prefixes implemented as of checkpoint 65x.

The control problem is non-dimensional by default, as reflected in the omission of weights in control penalties [($\vec{u}_j^T \vec{u}_j$ in (8.1)]. Non-dimensional controls ($\vec{u}_j$) are scaled to physical units ($\vec{v}_j$) through multiplication by the respective uncertainty fields ($\sigma_{\vec{u}_j}$), as part of the generic preprocessor $\mathcal{Q}$ in (8.4). Besides the scaling of $\vec{u}_j$ to physical units, the preprocessor $\mathcal{Q}$ can include, for example, spatial correlation modeling (using an implementation of Weaver and Coutier, 2001) by setting `xx_gen*_preproc = 'WC01'`. Alternatively, setting `xx_gen*_preproc = 'smooth'` activates the smoothing part of `WC01`, but omits the

name	description	arguments
WC01	Correlation modeling	integer: operator type (default: 1)
smooth	Smoothing without normalization	integer: operator type (default: 1)
docycle	Average period replication	integer: cycle length
replicate	Alias for docycle	(units of xx_gentim2d_period)
rmcycle	Periodic average subtraction	integer: cycle length
variaweight	Use time-varying weight	—
noscaling[a]	Do not scale with xx_gen*_weight	—
documul	Sets xx_gentim2d_cumsum	—
doglomean	Sets xx_gentim2d_glosum	—

Table 8.11: xx_gen????d_preproc options implemented as of checkpoint 65x. Notes: [a]: If noscaling is false, the control adjustment is scaled by one on the square root of the weight before being added to the base control variable; if noscaling is true, the control is multiplied by the weight in the cost function itself.

normalization. Additionally, bounds for the controls can be specified by setting xx_gen*_bounds. In forward mode, adjustments to the i^{th} control are clipped so that they remain between xx_gen*_bounds(i,1) and xx_gen*_bounds(i,4). If xx_gen*_bounds(i,1) $<$ xx_gen*_bounds(i+1,1) for $i = 1, 2, 3$, then the bounds will "emulate a local minimum;"[3] otherwise, the bounds have no effect in adjoint mode.

For the case of time-varying controls, the frequency is specified by xx_gentim2d_period. The generic control package interprets special values of xx_gentim2d_period in the same way as the exf package: a value of -12 implies cycling monthly fields while a value of 0 means that the field is steady. Time varying weights can be provided by specifying the preprocessor variaweight, in which case the xx_gentim2d_weight file must contain as many records as the control parameter time series itself (approximately the run length divided by xx_gentim2d_period).

The parameter mult_gen* sets the multiplier for the corresponding cost function penalty [β_j in (8.1); $\beta_j = 1$ by default). The preconditioner, $\mathcal{R}$, does not directly appear in the estimation problem, but only serves to push the optimization process in a certain direction in control space; this operator is specified by gen*Precond (= 1 by default).

[3] Not sure what this means.

8.4 SMOOTH: Smoothing And Covariance Model

TBA ...

8.5 The line search optimisation algorithm

Author: Patrick Heimbach

8.5.1 General features

The line search algorithm is based on a quasi-Newton variable storage method which was implemented by *Gilbert and Lemaréchal* [1989].

TO BE CONTINUED...

8.5.2 The online vs. offline version

- **Online version**
  Every call to *simul* refers to an execution of the forward and adjoint model. Several iterations of optimization may thus be performed within a single run of the main program (lsopt_top). The following cases may occur:
  - cold start only (no optimization)
  - cold start, followed by one or several iterations of optimization
  - warm start from previous cold start with one or several iterations
  - warm start from previous warm start with one or several iterations
- **Offline version**
  Every call to simul refers to a read procedure which reads the result of a forward and adjoint run Therefore, only one call to simul is allowed, `itmax = 0`, for cold start `itmax = 1`, for warm start Also, at the end, **x(i+1)** needs to be computed and saved to be available for the offline model and adjoint run

In order to achieve minimum difference between the online and offline code **xdiff(i+1)** is stored to file at the end of an (offline) iteration, but recomputed identically at the beginning of the next iteration.

8.5.3 Number of iterations vs. number of simulations

`- itmax:` controls the max. number of iterations
`- nfunc:` controls the max. number of simulations within one iteration

Summary
From one iteration to the next the descent direction changes. Within one iteration more than one forward and adjoint run may be performed. The updated control used as input for these simulations uses the same descent direction, but different step sizes.

Description
From one iteration to the next the descent direction dd changes using the result for the adjoint vector gg of the previous iteration. In lsline the updated control

$$\mathtt{xdiff(i,1)} = \mathtt{xx(i-1)} + \mathtt{tact(i-1,1)} * \mathtt{dd(i-1)}$$

serves as input for a forward and adjoint model run yielding a new `gg(i,1)`. In general, the new solution passes the 1st and 2nd Wolfe tests so `xdiff(i,1)` represents the solution sought:

$$\mathtt{xx(i)} = \mathtt{xdiff(i,1)}$$

If one of the two tests fails, an inter- or extrapolation is invoked to determine a new step size `tact(i-1,2)`. If more than one function call is permitted, the new step size is used together with the "old" descent direction `dd(i-1)` (i.e. dd is not updated using the new `gg(i)`), to compute a new

$$\mathtt{xdiff(i,2)} = \mathtt{xx(i-1)} + \mathtt{tact(i-1,2)} * \mathtt{dd(i-1)}$$

that serves as input in a new forward and adjoint run, yielding gg(i,2). If now, both Wolfe tests are successful, the updated solution is given by

$$\mathtt{xx(i)} = \mathtt{xdiff(i,2)} = \mathtt{xx(i-1)} + \mathtt{tact(i-1,2)} * \mathtt{dd(i-1)}$$

In order to save memory both the fields dd and xdiff have a double usage.

`xdiff`
- in *lsopt_top*: used as `x(i) - x(i-1)` for Hessian update
- in *lsline*: intermediate result for control update `x = x + tact*dd`

`dd`
- in *lsopt_top, lsline*: descent vector, `dd = -gg` and `hessupd`
- in *dgscale*: intermediate result to compute new preconditioner

The parameter file lsopt.par

- **NUPDATE** max. no. of update pairs `(gg(i)-gg(i-1), xx(i)-xx(i-1))` to be stored in `OPWARMD` to estimate Hessian [pair of current iter. is stored in `(2*jmax+2, 2*jmax+3)` jmax must be ¿ 0 to access these entries] Presently `NUPDATE` must be ¿ 0 (i.e. iteration without reference to previous iterations through `OPWARMD` has not been tested)
- **EPSX** relative precision on xx bellow which xx should not be improved
- **EPSG** relative precision on gg below which optimization is considered successful
- **IPRINT** controls verbose (¿=1) or non-verbose output
- **NUMITER** max. number of iterations of optimisation; NUMTER = 0: cold start only, no optimization
- **ITER_NUM** index of new restart file to be created (not necessarily = NUMITER!)
- **NFUNC** max. no. of simulations per iteration (must be ¿ 0); is used if step size `tact` is inter-/extrapolated; in this case, if NFUNC ¿ 1, a new simulation is performed with same gradient but "improved" step size
- **FMIN** first guess cost function value (only used as long as first iteration not completed, i.e. for jmax ¡= 0)

OPWARMI, OPWARMD files Two files retain values of previous iterations which are used in latest iteration to update Hessian:

- **OPWARMI**: contains index settings and scalar variables

`n = nn`	no. of control variables
`fc = ff`	cost value of last iteration
`isize`	no. of bytes per record in OPWARMD
`m = nupdate`	max. no. of updates for Hessian
`jmin, jmax`	pointer indices for OPWARMD file (cf. below)
`gnorm0`	norm of first (cold start) gradient gg
`iabsiter`	total number of iterations with respect to cold start

- **OPWARMD**: contains vectors (control and gradient)

entry	name	description
1	`xx(i)`	control vector of latest iteration
2	`gg(i)`	gradient of latest iteration
3	`xdiff(i),diag`	preconditioning vector; (1,...,1) for cold start
2*jmax+2	`gold=g(i)-g(i-1)`	for last update (jmax)
2*jmax+3	`xdiff=tact*d=xx(i)-xx(i-1)`	for last update (jmax)

Error handling

```
Example 1: jmin = 1, jmax = 3, mupd = 5

  1   2   3   |   4   5     6   7     8   9     empty     empty
|___|___|___| | |___|___| |___|___| |___|___| |___|___| |___|___|
      0       |     1         2         3

Example 2: jmin = 3, jmax = 7, mupd = 5   ---> jmax = 2

  1   2   3   |
|___|___|___| | |___|___| |___|___| |___|___| |___|___| |___|___|
              |     6         7         3         4         5
```

Figure 8.1: Examples of OPWARM file handling

```
  lsopt_top
      |
      |---- check arguments
      |---- CALL INSTORE
      |       |
      |       |---- determine whether OPWARMI available:
      |                * if no:  cold start: create OPWARMI
      |                * if yes: warm start: read from OPWARMI
      |             create or open OPWARMD
      |
      |---- check consistency between OPWARMI and model parameters
      |
      |---- >>> if COLD start: <<<
      |      |  first simulation with f.g. xx_0; output: first ff_0, gg_0
      |      |  set first preconditioner value xdiff_0 to 1
      |      |  store xx(0), gg(0), xdiff(0) to OPWARMD (first 3 entries)
      |      |
      |     >>> else: WARM start: <<<
      |         read xx(i), gg(i) from OPWARMD (first 2 entries)
      |         for first warm start after cold start, i=0
      |
      |
      |
      |---- /// if ITMAX > 0: perform optimization (increment loop index i)
      |      (
      |      )---- save current values of gg(i-1) -> gold(i-1), ff -> fold(i-1)
      |      (---- CALL LSUPDXX
      |      )       |
      |      (       |---- >>> if jmax=0 <<<
      |      )       |      |  first optimization after cold start:
      |      (       |      |  preconditioner estimated via ff_0 - ff_(first guess)
      |      )       |      |  dd(i-1) = -gg(i-1)*preco
      |      (       |      |
      |      )       |     >>> if jmax > 0 <<<
      |      (       |         dd(i-1) = -gg(i-1)
      |      )       |         CALL HESSUPD
      |      (       |           |
      |      )       |           |---- dd(i-1) modified via Hessian approx.
      |      (       |
      |      )       |---- >>> if <dd,gg> >= 0 <<<
      |      (       |         ifail = 4
      |      )       |
      |      (       |---- compute step size: tact(i-1)
      |      )       |---- compute update: xdiff(i) = xx(i-1) + tact(i-1)*dd(i-1)
      |      (
      |      )---- >>> if ifail = 4 <<<
      |      (         goto 1000
      |      )
      |      (---- CALL OPTLINE / LSLINE
      |      )       |
     ...    ...     ...
```

Figure 8.2: Flow chart (part 1 of 3)

```
...    ...
 |      )
 |      (---- CALL OPTLINE / LSLINE
 |      )       |
 |      (       |---- /// loop over simulations
 |      )              (
 |      (              )---- CALL SIMUL
 |      )              (       |
 |      (              )       |----  input: xdiff(i)
 |      )              (       |---- output: ff(i), gg(i)
 |      (              )       |---- >>> if ONLINE <<<
 |      )              (                 runs model and adjoint
 |      (              )             >>> if OFFLINE <<<
 |      )              (                 reads those values from file
 |      (              )
 |      )              (---- 1st Wolfe test:
 |      (              )     ff(i) <= tact*xpara1*<gg(i-1),dd(i-1)>
 |      )              (
 |      (              )---- 2nd Wolfe test:
 |      )              (     <gg(i),dd(i-1)> >= xpara2*<gg(i-1),dd(i-1)>
 |      (              )
 |      )              (---- >>> if 1st and 2nd Wolfe tests ok <<<
 |      (              )      |  320: update xx: xx(i) = xdiff(i)
 |      )              (      |
 |      (              )     >>> else if 1st Wolfe test not ok <<<
 |      )              (      |  500: INTERpolate new tact:
 |      (              )      |  barr*tact < tact < (1-barr)*tact
 |      )              (      |  CALL CUBIC
 |      (              )      |
 |      )              (     >>> else if 2nd Wolfe test not ok <<<
 |      (              )         350: EXTRApolate new tact:
 |      )              (         (1+barmin)*tact < tact < 10*tact
 |      (              )         CALL CUBIC
 |      )              (
 |      (              )---- >>> if new tact > tmax <<<
 |      )              (      |  ifail = 7
 |      (              )      |
 |      )              (---- >>> if new tact < tmin OR tact*dd < machine precision <<<
 |      (              )      |  ifail = 8
 |      )              (      |
 |      (              )---- >>> else <<<
 |      )              (         update xdiff for new simulation
 |      (              )
 |      )             \\\ if nfunc > 1: use inter-/extrapolated tact and xdiff
 |      (                               for new simulation
 |      )                               N.B.: new xx is thus not based on new gg, but
 |      (                                     rather on new step size tact
 |      )
 |      (---- store new values xx(i), gg(i) to OPWARMD (first 2 entries)
 |      )---- >>> if ifail = 7,8,9 <<<
 |      (         goto 1000
 |      )
...    ...
```

Figure 8.3: Flow chart (part 2 of 3)

```
...    ...
 |      )
 |      (---- store new values xx(i), gg(i) to OPWARMD (first 2 entries)
 |      )---- >>> if ifail = 7,8,9 <<<
 |      (         goto 1000
 |      )
 |      (---- compute new pointers jmin, jmax to include latest values
 |      )     gg(i)-gg(i-1), xx(i)-xx(i-1) to Hessian matrix estimate
 |      (---- store gg(i)-gg(i-1), xx(i)-xx(i-1) to OPWARMD
 |      )     (entries 2*jmax+2, 2*jmax+3)
 |      (
 |      )---- CALL DGSCALE
 |      (       |
 |      )       |---- call dostore
 |      (       |       |
 |      )       |       |---- read preconditioner of previous iteration diag(i-1)
 |      (       |             from OPWARMD (3rd entry)
 |      )       |
 |      (       |---- compute new preconditioner diag(i), based upon diag(i-1),
 |      )       |     gg(i)-gg(i-1), xx(i)-xx(i-1)
 |      (       |
 |      )       |---- call dostore
 |      (               |
 |      )               |---- write new preconditioner diag(i) to OPWARMD (3rd entry)
 |      (
 |---- \\\ end of optimization iteration loop
 |
 |
 |
 |---- CALL OUTSTORE
 |       |
 |       |---- store gnorm0, ff(i), current pointers jmin, jmax, iterabs to OPWARMI
 |
 |---- >>> if OFFLINE version <<<
 |         xx(i+1) needs to be computed as input for offline optimization
 |          |
 |          |---- CALL LSUPDXX
 |          |       |
 |          |       |---- compute dd(i), tact(i) -> xdiff(i+1) = x(i) + tact(i)*dd(i)
 |          |
 |          |---- CALL WRITE_CONTROL
 |          |       |
 |          |       |---- write xdiff(i+1) to special file for offline optim.
 |
 |---- print final information
 |
 O
```

Figure 8.4: Flow chart (part 3 of 3)

Chapter 9

Under development

This chapter contains short descriptions of new features that are stil in a development stage.

9.1 Other Time-stepping Options

9.1.1 Adams-Bashforth III

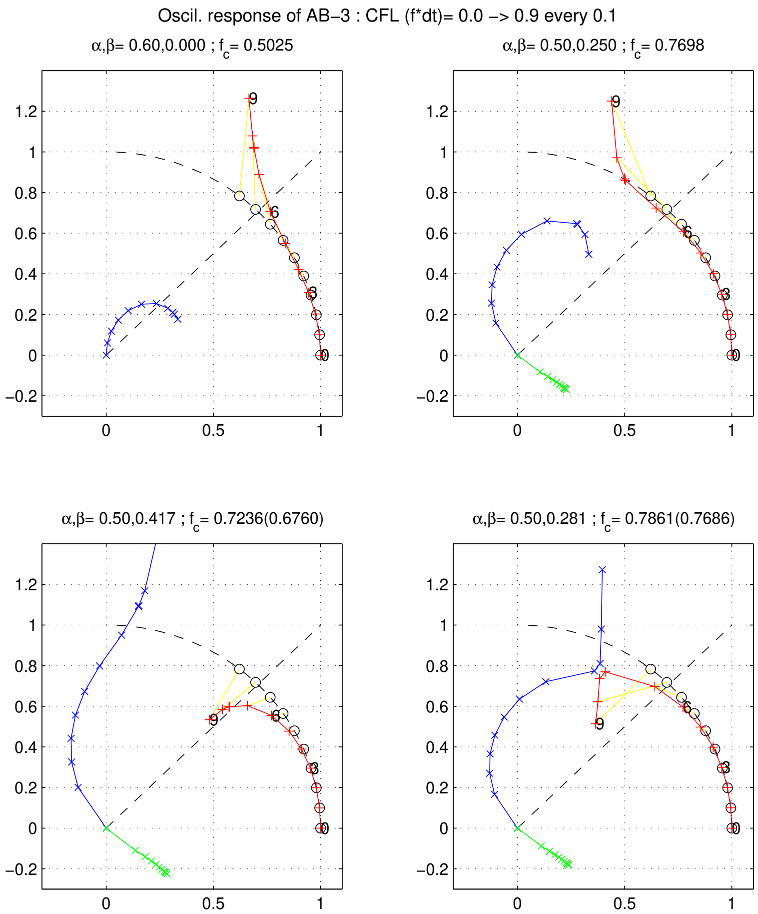


Figure 9.1: Comparison of the oscillatory response of Adams-Bashforth scheme.

The third-order Adams-Bashforth time stepping (AB-3) provides several advantages (see, e.g., *Durran* [1991]) compared to the default quasi-second order Adams-Bashforth (AB-2):

- higher accuracy;
- stable with a longer time-step;
- no additional computation (just requires the storage of one additional time level).

The 3^{rd} order Adams-Bashforth can be used to extrapolate forward in time the tendency (replacing equation 2.24) which writes:

$$G_\tau^{(n+1/2)} = (1 + \alpha_{AB} + \beta_{AB})G_\tau^n - (\alpha_{AB} + 2\beta_{AB})G_\tau^{n-1} + \beta_{AB}G_\tau^{n-2} \tag{9.1}$$

The 3rd order AB is obtained with $(\alpha_{AB}, \beta_{AB}) = (1/2, 5/12)$. Note that selecting $(\alpha_{AB}, \beta_{AB}) = (1/2 + \epsilon_{AB}, 0)$ one recovers the quasi-2nd order AB.

The AB-3 time stepping improves the stability limit for an oscillatory problem like advection or Coriolis. As seen from Fig.9.1, it remains stable up to a CFL of 0.72, compared to only 0.50 with AB-2 and $\epsilon_{AB} = 0.1$. It is interesting to note that the stability limit can be further extended up to a CFL of 0.786 for an oscillatory problem (see fig.9.1) using $(\alpha_{AB}, \beta_{AB}) = (0.5, 0.2811)$ but then the scheme is only 2nd order accurate.

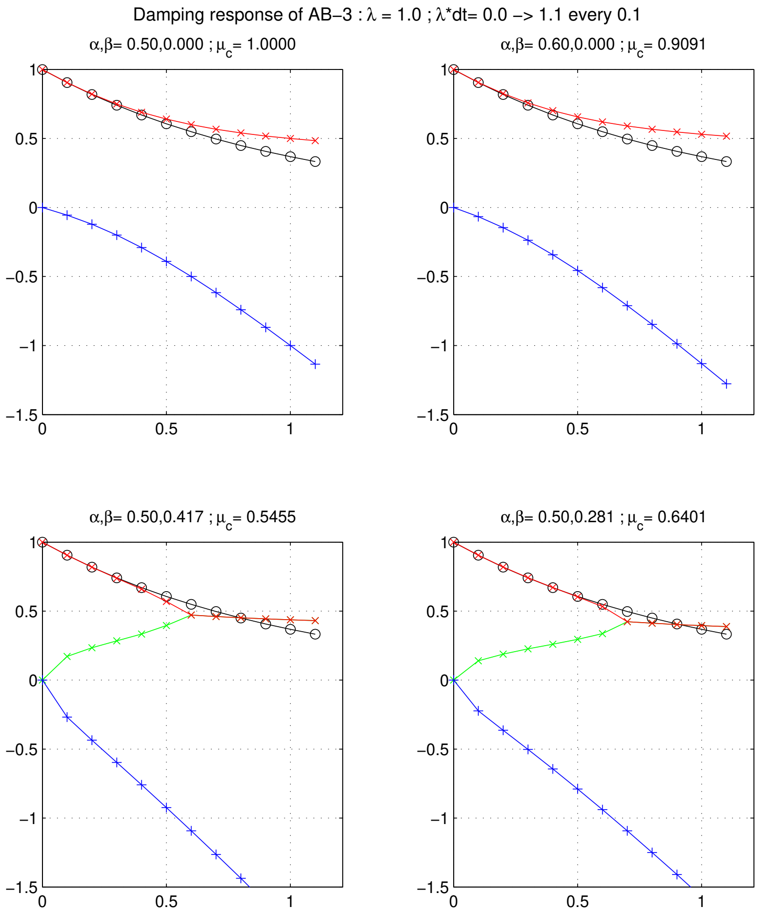


Figure 9.2: Comparison of the damping (diffusion like) response of Adams-Bashforth schemes.

However, the behavior of the AB-3 for a damping problem (like diffusion) is less favorable, since the stability limit is reduced to 0.54 only (and 0.64 with $\beta_{AB} = 0.2811$) compared to 1. (and 0.9 with $\epsilon_{AB} = 0.1$) with the AB-2 (see fig.9.2).

A way to enable the use of a longer time step is to keep the dissipation terms outside the AB extrapolation (setting *momDissip_In_AB=.FALSE.* in main parameter file "`data`", namelist *PARM03*), thus returning to a simple forward time-stepping for dissipation, and to use AB-3 only for advection and Coriolis terms.

The AB-3 time stepping is activated by defining the option *#define ALLOW_ADAMSBASHFORTH_3* in "`CPP_OPTIONS.h`". The parameters α_{AB}, β_{AB} can be set from the main parameter file "`data`" (namelist *PARM03*) and their default value corresponds to the 3rd order Adams-Bashforth. A simple example is provided in "`verification/advect_xy/input.ab3_c4`".

The AB-3 is not yet available for the vertical momentum equation (Non-Hydrostatic) neither for passive tracers.

9.1.2 Time-extrapolation of tracer (rather than tendency)

(to be continued ...)

Chapter 10

Previous Applications of MITgcm

To date, MITgcm has been used for number of purposes. The following papers are an incomplete yet fairly broad sample of MITgcm applications:

- R.S. Gross and I. Fukumori and D. Menemenlis (2003). Atmospheric and oceanic excitation of the Earth's wobbles during 1980-2000. *Journal of Geophysical Research-Solid Earth*, vol.108.
- M.A. Spall and R.S. Pickart (2003). Wind-driven recirculations and exchange in the Labrador and Irminger Seas. *Journal of Physical Oceanography*, vol.33, pp.1829–1845.
- B. Ferron and J. Marotzke (2003). Impact of 4D-variational assimilation of WOCE hydrography on the meridional circulation of the Indian Ocean. *Deep-Sea Research Part II-Topical Studies in Oceanography*, vol.50, pp.2005–2021.
- G.A. McKinley and M.J. Follows and J. Marshall and S.M. Fan (2003). Interannual variability of air-sea O-2 fluxes and the determination of CO2 sinks using atmospheric O-2/N-2. *Geophysical Research Letters*, vol.30.
- B.Y. Huang and P.H. Stone and A.P. Sokolov and I.V. Kamenkovich (2003). The deep-ocean heat uptake in transient climate change. *Journal of Climate*, vol.16, pp.1352–1363.
- G. de Coetlogon and C. Frankignoul (2003). The persistence of winter sea surface temperature in the North Atlantic. *Journal of Climate*, vol.16, pp.1364–1377.
- K.H. Nisancioglu and M.E. Raymo and P.H. Stone (2003). Reorganization of Miocene deep water circulation in response to the shoaling of the Central American Seaway. *Paleoceanography*, vol.18.
- T. Radko and J. Marshall (2003). Equilibration of a Warm Pumped Lens on a beta plane. *Journal of Physical Oceanography*, vol.33, pp.885–899.
- T. Stipa (2002). The dynamics of the N/P ratio and stratification in a large nitrogen-limited estuary as a result of upwelling: a tendency for offshore Nodularia blooms. *Hydrobiologia*, vol.487, pp.219–227.
- D. Stammer and C. Wunsch and R. Giering and C. Eckert and P. Heimbach and J. Marotzke and A. Adcroft and C.N. Hill and J. Marshall (2003). Volume, heat, and freshwater transports of the global ocean circulation 1993-2000, estimated from a general circulation model constrained by World Ocean Circulation Experiment (WOCE) data. *Journal of Geophysical Research-Oceans*, vol.108.
- P. Heimbach and C. Hill and R. Giering (2002). Automatic generation of efficient adjoint code for a parallel Navier-Stokes solver. *Computational Science-ICCS 2002, PT II, Proceedings*, vol.2330, pp.1019–1028.
- D. Stammer and C. Wunsch and R. Giering and C. Eckert and P. Heimbach and J. Marotzke and A. Adcroft and C.N. Hill and J. Marshall (2002). Global ocean circulation during 1992-1997, estimated from ocean observations and a general circulation model. *Journal of Geophysical Research-Oceans*, vol.107.

- M. Solovev and P.H. Stone and P. Malanotte-Rizzoli (2002). Assessment of mesoscale eddy parameterizations for a single-basin coarse-resolution ocean model. *Journal of Geophysical Research-Oceans*, vol.107.
- C. Herbaut and J. Sirven and S. Fevrier (2002). Response of a simplified oceanic general circulation model to idealized NAO-like stochastic forcing. *Journal of Physical Oceanography*, vol.32, pp.3182–3192.
- C. Wunsch (2002). Oceanic age and transient tracers: Analytical and numerical solutions. *Journal of Geophysical Research-Oceans*, vol.107.
- J. Sirven and C. Frankignoul and D. de Coetlogon and V. Taillandier (2002). Spectrum of wind-driven baroclinic fluctuations of the ocean in the midlatitudes. *Journal of Physical Oceanography*, vol.32, pp.2405–2417.
- R. Zhang and M. Follows and J. Marshall (2002). Mechanisms of thermohaline mode switching with application to warm equable climates. *Journal of Climate*, vol.15, pp.2056–2072.
- G. Brostrom (2002). On advection and diffusion of plankton in coarse resolution ocean models. *Journal of Marine Systems*, vol.35, pp.99–110.
- T. Lee and I. Fukumori and D. Menemenlis and Z.F. Xing and L.L. Fu (2002). Effects of the Indonesian Throughflow on the Pacific and Indian oceans. *Journal of Physical Oceanography*, vol.32, pp.1404–1429.
- C. Herbaut and J. Marshall (2002). Mechanisms of buoyancy transport through mixed layers and statistical signatures from isobaric floats. *Journal of Physical Oceanography*, vol.32, pp.545–557.
- J. Marshall and H. Jones and R. Karsten and R. Wardle (2002). Can eddies set ocean stratification?, *Journal of Physical Oceanography*, vol.32, pp.26–38.
- C. Herbaut and J. Sirven and A. Czaja (2001). An idealized model study of the mass and heat transports between the subpolar and subtropical gyres. *Journal of Physical Oceanography*, vol.31, pp.2903–2916.
- R.H. Kase and A. Biastoch and D.B. Stammer (2001). On the Mid-Depth Circulation in the Labrador and Irminger Seas. *Geophysical Research Letters*, vol.28, pp.3433–3436.
- R. Zhang and M.J. Follows and J.P. Grotzinger and J. Marshall (2001). Could the Late Permian deep ocean have been anoxic?. *Paleoceanography*, vol.16, pp.317–329.
- A. Adcroft and J.R. Scott and J. Marotzke (2001). Impact of geothermal heating on the global ocean circulation. *Geophysical Research Letters*, vol.28, pp.1735–1738.
- J. Marshall and D. Jamous and J. Nilsson (2001). Entry, flux, and exit of potential vorticity in ocean circulation. *Journal of Physical Oceanography*, vol.31, pp.777–789.
- A. Mahadevan (2001). An analysis of bomb radiocarbon trends in the Pacific. *Marine Chemistry*, vol.73, pp.273–290.
- G. Brostrom (2000). The role of the annual cycles for the air-sea exchange of CO2. *Marine Chemistry*, vol.72, pp.151–169.
- Y.H. Zhou and H.Q. Wu and N.H. Yu and D.W. Zheng (2000). Excitation of seasonal polar motion by atmospheric and oceanic angular momentums. *Progress in Natural Science*, vol.10, pp.931–936.
- R. Wardle and J. Marshall (2000). Representation of eddies in primitive equation models by a PV flux. *Journal of Physical Oceanography*, vol.30, pp.2481–2503.
- P.S. Polito and O.T. Sato and W.T. Liu (2000). Characterization and validation of the heat storage variability from TOPEX/Poseidon at four oceanographic sites. *Journal of Geophysical Research-Oceans*, vol.105, pp.16911–16921.

- R.M. Ponte and D. Stammer (2000). Global and regional axial ocean angular momentum signals and length-of-day variations (1985-1996). *Journal of Geophysical Research-Oceans*, vol.105, pp.17161–17171.
- J. Wunsch (2000). Oceanic influence on the annual polar motion. *Journal of GEODYNAMICS*, vol.30, pp.389–399.
- D. Menemenlis and M. Chechelnitsky (2000). Error estimates for an ocean general circulation model from altimeter and acoustic tomography data. *Monthly Weather Review*, vol.128, pp.763–778.
- Y.H. Zhou and D.W. Zheng and N.H. Yu and H.Q. Wu (2000). Excitation of annual polar motion by atmosphere and ocean. *Chinese Science Bulletin*, vol.45, pp.139–142.
- J. Marotzke and R. Giering and K.Q. Zhang and D. Stammer and C. Hill and T. Lee (1999). Construction of the adjoint MIT ocean general circulation model and application to Atlantic heat transport sensitivity. *Journal of Geophysical Research-Oceans*, vol.104, pp.29529–29547.
- R.M. Ponte and D. Stammer (1999). Role of ocean currents and bottom pressure variability on seasonal polar motion. *Journal of Geophysical Research-Oceans*, vol.104, pp.23393–23409.
- J. Nastula and R.M. Ponte (1999). Further evidence for oceanic excitation of polar motion. *Geophysical Journal International*, vol.139, pp.123–130.
- K.Q. Zhang and J. Marotzke (1999). The importance of open-boundary estimation for an Indian Ocean GCM-data synthesis. *Journal of Marine Research*, vol.57, pp.305–334.
- J. Marshall and D. Jamous and J. Nilsson (1999). Reconciling thermodynamic and dynamic methods of computation of water-mass transformation rates. *Deep-Sea Research PART I-Oceanographic Research Papers*, vol.46, pp.545–572.
- J. Marshall and F. Schott (1999). Open-ocean convection: Observations, theory, and models. *Reviews of Geophysics*, vol.37, pp.1–64.
- J. Marshall and H. Jones and C. Hill (1998). Efficient ocean modeling using non-hydrostatic algorithms. *Journal of Marine Systems*, vol.18, pp.115–134.
- A.B. Baggeroer and T.G. Birdsall and C. Clark and J.A. Colosi and B.D. Cornuelle and D. Costa and B.D. Dushaw and M. Dzieciuch and A.M.G. Forbes and C. Hill and B.M. Howe and J. Marshall and D. Menemenlis and J.A. Mercer and K. Metzger and W. Munk and R.C. Spindel and D. Stammer and P.F. Worcester and C. Wunsch (1998). Ocean climate change: Comparison of acoustic tomography, satellite altimetry, and modeling. *Science*, vol.281, pp.1327–1332.
- C. Wunsch and D. Stammer (1998). Satellite altimetry, the marine geoid, and the oceanic general circulation. *Annual Review of Earth and Planetary Sciences*, vol.26, pp.219–253.
- A. Shaw and Arvind and K.C. Cho and C. Hill and R.P. Johnson and J. Marshall (1998). A comparison of implicitly parallel multithreaded and data-parallel implementations of an ocean model. *Journal of Parallel and Distributed Computing*, vol.48, pp.1–51.
- T.W.N. Haine and J. Marshall (1998). Gravitational, symmetric, and baroclinic instability of the ocean mixed layer. *Journal of Physical Oceanography*, vol.28, pp.634–658.
- R.M. Ponte and D. Stammer and J. Marshall (1998). Oceanic signals in observed motions of the Earth's pole of rotation. *Nature*, vol.391, pp.476–479.
- D. Menemenlis and C. Wunsch (1997). Linearization of an oceanic general circulation model for data assimilation and climate studies. *Journal of Atmospheric and Oceanic Technology*, vol.14, pp.1420–1443.
- H. Jones and J. Marshall (1997). Restratification after deep convection. *Journal of Physical Oceanography*, vol.27, pp.2276–2287.
- S.R. Jayne and R. Tokmakian (1997). Forcing and sampling of ocean general circulation models: Impact of high-frequency motions. *Journal of Physical Oceanography*, vol.27, pp.1173–1179.

Bibliography

Adcroft, A., Numerical algorithms for use in a dynamical model of the ocean, Ph.D. thesis, Imperial College, London, 1995.

Adcroft, A., and J.-M. Campin, Comparison of finite volume schemes and direct-space-time methods for ocean circulation models, *Ocean Modelling*, in preparation, 2002.

Adcroft, A., and J.-M. Campin, Re-scaled height coordinates for accurate representation of free-surface flows in ocean circulation models, *Ocean Modelling*, *7*, 269–284, 2004. doi:10.1016/j.ocemod.2003.09.003.

Adcroft, A., and D. Marshall, How slippery are piecewise-constant coastlines in numerical ocean models?, *Tellus*, *50*(1), 95–108, 1998.

Adcroft, A., C. Hill, and J. Marshall, Representation of topography by shaved cells in a height coordinate ocean model, *Mon. Wea. Rev.*, *125*, 2293–2315, 1997. Available from: `http://mitgcm.org/pdfs/mwr_1997.pdf`, doi:10.1175/1520-0493%281997%29125¡2293:ROTBSC¿2.0.CO;2.

Adcroft, A., J.-M. Campin, C. Hill, and J. Marshall, Implementation of an atmosphere-ocean general circulation model on the expanded spherical cube, *Mon. Wea. Rev.*, *132*, 2845–2863, 2004a. Available from: `http://mitgcm.org/pdfs/mwr_2004.pdf`, doi:10.1175/MWR2823.1.

Adcroft, A., C. Hill, J.-M. Campin, J. Marshall, and P. Heimbach, Overview of the formulation and numerics of the MITgcm, in *Proceedings of the ECMWF seminar series on Numerical Methods, Recent developments in numerical methods for atmosphere and ocean modelling*, pp. 139–149. ECMWF, 2004b. Available from: `http://mitgcm.org/pdfs/ECMWF2004-Adcroft.pdf`.

Adcroft, A., e. a., Energy conversion in discrete numerical models, *Ocean Modelling*, in preparation, 2002.

Adcroft, A., H. C., and J. Marshall, A new treatment of the coriolis terms in c-grid models at both high and low resolutions, *Mon. Wea. Rev.*, *127*, 1928–1936, 1999. Available from: `http://mitgcm.org/pdfs/mwr_1999.pdf`, doi:10.1175/1520-0493%281999%29127¡1928:ANTOTC¿2.0.CO;2.

Arakawa, A., and V. Lamb, Computational design of the basic dynamical processes of the ucla general circulation model, in *Methods in Computational Physics*, vol. 17, pp. 174–267. Academic Press, 1977.

Beckmann, A., H. H. Hellmer, and R. Timmermann, A numerical model of the Weddell Sea: Large-scale circulation and water mass distribution, *J. Geophys. Res.*, *104*(C10), 23,375–23,392, 1999.

Blackadar, A. K., High resolution models of the planetary boundary layer., in *Advances in Environmental Science and Engineering, Vol 1*, edited by Pfafflin, and Zeigler. Gordon and Breach, Scientific Publishers, 1977.

Bouillon, S., T. Fichefet, V. Legat, and G. Madec, The elastic-viscous-plastic method revisited, *Ocean Modelling*, *71*(0), 2–12, Arctic Ocean, 2013. Available from: `http://dx.doi.org/10.1016/j.ocemod.2013.05.013`, doi:10.1016/j.ocemod.2013.05.013.

Bretherton, C. S., et al., An intercomparison of radiatively driven entrainment and turbulence in a smoke cloud, as simulated by different numerical models., *Q. J. R. Meteorol. Soc.*, *125*, 391–423, 1999.

Bryan, K., Accelerating the convergence to equilibrium of ocean-climate models, *J. Phys. Oceanogr.*, *14*(4), 666–673, 1984.

Bryan, K., S. Manabe, and R. C. Pacanowski, A global ocean-atmosphere climate model. Part II. The oceanic circulation, *Journal of Physical Oceanography*, *5*, 30–46, 1975.

Bushell, A. C., and G. Martin, The impact of vertical resolution upon gcm simulations of marine stratocumulus., *Climate Dyn.*, *15*, 293–318, 1999.

Campin, J.-M., A. Adcroft, C. Hill, and J. Marshall, Conservation of properties in a free-surface model, *Ocean Modelling*, *6*, 221–244, 2004.

Campin, J.-M., J. Marshall, and D. Ferreira, Sea-ice ocean coupling using a rescaled vertical coordinate z^*, *Ocean Modelling*, *24*(1–2), 1–14, 2008. doi:10.1016/j.ocemod.2008.05.005.

Castro-Morales, K., F. Kauker, M. Losch, S. Hendricks, K. Riemann-Campe, and R. Gerdes, Sensitivity of simulated Arctic sea ice to realistic ice thickness distributions and snow parameterizations, *J. Geophys. Res.*, *119*(1), 559–571, 2014. Available from: `http://dx.doi.org/10.1002/2013JC009342`, doi:10.1002/2013JC009342.

Chou, M.-D., Parameterizations for the absorption of solar radiation by o_2 and co_2 with applications to climate studies., *J. Clim.*, *3*, 209–217, 1990.

Chou, M.-D., A solar radiation model for use in climate studies., *J. Atmos. Sci.*, *49*, 762–772, 1992.

Chou, M.-D., and M.J.Suarez, An efficient thermal infrared radiation parameterization for use in general circulation models, NASA Technical Memorandum 104606-Vol 3, National Aeronautics and Space Administration, NASA; Goddard Space Flight Center; Greenbelt (MD), 20771; USA, http://www.gmao.nasa.gov/, 1994.

Chris Hill, Alistair Adcroft, D. J., and J. Marshall, A strategy for terascale climate modeling, in *In Proceedings of the Eighth ECMWF Workshop on the Use of Parallel Processors in Meteorology*, pp. 406–425. World Scientific, 1999.

Clarke, R. H., Observational studies in the atmospheric boundary layer., *Q. J. R. Meteorol. Soc.*, *96*, 91–114, 1970.

Cox, M. D., An isopycnal diffusion in a z-coordinate ocean model, *Ocean modelling*, *74*, 1–5 (Unpublished manuscript), 1987.

Danabasoglu, G., and J. C. McWilliams, Sensitivity of the global ocean circulation to parameterizations of mesoscale tracer transports, *J. Clim.*, *8*(12), 2967–2987, 1995.

de Szoeke, R. A., and R. M. Samelson, The duality between the Boussinesq and Non-Boussinesq hydrostatic equations of motion, *J. Phys. Oceanogr.*, *32*(8), 2194–2203, 2002.

Defries, R. S., and J. R. G. Townshend, Ndvi-derived land cover classification at global scales., *Int'l J. Rem. Sens.*, *15*, 3567–3586, 1994.

Dorman, J. L., and P. J. Sellers, A global climatology of albedo, roughness length and stomatal resistance for atmospheric general circulation models as represented by the simple biosphere model (sib)., *J. Appl. Meteor.*, *28*, 833–855, 1989.

Durran, D. R., The third-order adams-bashforth method: An attractive alternative to leapfrog time differencing, *Mon. Wea. Rev.*, *119*, 702–720, 1991.

Dutay, J.-C., et al., Evaluation of ocean model ventilation with cfc-11: comparison of 13 global ocean models, *Ocean Modelling*, *4*, 89–120, 2002.

Dutkiewicz, S., A. Sokolov, J. Scott, and P. Stone, A three-dimensional ocean-seaice-carbon cycle model and its coupling to a two-dimensional atmospheric model: Uses in climate change studies, report 122, Tech. rep., MIT Joint Program of the Science and Policy of Global Change, Cambridge, MA, http://web.mit.edu/globalchange/www/MITJPSPGC_Rpt122.pdf, 2005.

Ferreira, D., J. Marshall, and P. Heimbach, Estimating eddy stresses by fitting dynamics to observations using a residual-mean ocean circulation model and its adjoint, *J. Phys. Oceanogr.*, *35*, 1891–1910, 2005.

Flato, G. M., and W. D. Hibler, III, Modeling pack ice as a cavitating fluid, *J. Phys. Oceanogr.*, *22*, 626–651, 1992.

Fofonoff, P., and R. Millard, Jr., Algorithms for computation of fundamental properties of seawater, Unesco Technical Papers in Marine Science 44, Unesco, 1983.

Follows, M., T. Ito, and S. Dutkiewicz, A compact and accurate carbonate chemistry solver for ocean biogeochemistry models, *Ocean Modeling*, in press.

Forget, G., J.-M. Campin, P. Heimbach, C. N. Hill, R. M. Ponte, and C. Wunsch, ECCO version 4: an integrated framework for non-linear inverse modeling and global ocean state estimation, *Geoscientific Model Development*, *8*(10), 3071–3104, 2015. Available from: `http://www.geosci-model-dev.net/8/3071/2015/`, doi:10.5194/gmd-8-3071-2015.

Fukumori, I., O. Wang, W. Llovel, I. Fenty, and G. Forget, A near-uniform fluctuation of ocean bottom pressure and sea level across the deep ocean basins of the arctic ocean and the nordic seas, *Progress in Oceanography*, *134*(0), 152 – 172, 2015. Available from: `http://www.sciencedirect.com/science/article/pii/S0079661115000245`, doi:http://dx.doi.org/10.1016/j.pocean.2015.01.013.

Gaspar, P., Y. Grégoris, and J.-M. Lefevre, A simple eddy kinetic energy model for simulations of the oceanic vertical mixing: Tests at station papa and long-term upper ocean study site, *J. Geophys. Res.*, *95 (C9)*, 16,179–16,193, 1990.

Geiger, C. A., W. D. Hibler, III, and S. F. Ackley, Large-scale sea ice drift and deformation: Comparison between models and observations in the western Weddell Sea during 1992, *J. Geophys. Res.*, *103*(C10), 21893–21913, 1998.

Gent, P. R., and J. C. McWilliams, Isopycnal mixing in ocean circulation models, *J. Phys. Oceanogr.*, *20*, 150–155, 1990.

Gent, P. R., J. Willebrand, T. J. McDougall, and J. C. McWilliams, Parameterizing eddy-induced tracer transports in ocean circulation models, *J. Phys. Oceanogr.*, *25*, 463–474, 1995.

Gerdes, R., C. Koberle, and J. Willebrand, The influence of numerical advection schemes on the results of ocean general circulation models, *Clim. Dynamics*, *5*(4), 211–226, 1991. doi:10.1007/BF00210006.

Giering, R., Tangent linear and adjoint model compiler. users manual 1.4 (tamc version 5.2), Report , Massachusetts Institute of Technology, MIT/EAPS; 77 Massachusetts Ave.; Cambridge (MA) 02139; USA, http://puddle.mit.edu/∼ralf/tamc/tamc.html, 1999.

Giering, R., Tangent linear and adjoint biogeochemical models, in *Inverse methods in global biogeochemical cycles*, edited by P. Kasibhatla, M. Heimann, P. Rayner, N. Mahowald, R. Prinn, and D. Hartley, vol. 114 of *Geophysical Monograph*, pp. 33–48. American Geophysical Union, Washington, DC, 2000.

Giering, R., and T. Kaminski, Recipes for adjoint code construction, *ACM Transactions on Mathematical Software*, *24*, 437–474, 1998.

Gilbert, J., and C. Lemaréchal, Some numerical experiments with variable-storage quasi-newton algorithms, *Math. Programming*, *45*, 407–435, 1989.

Gill, A. E., *Atmosphere-Ocean Dynamics*. Academic Press, New York, 662pp., 1982.

Griewank, A., Achieving logarithmic growth of temporal and spatial complexity in reverse Automatic Differentiation, *Optimization Methods and Software*, *1*, 35–54, 1992.

Griewank, A., *Evaluating Derivatives. Principles and Techniques of Algorithmic Differentiation*, vol. 19 of *Frontiers in Applied Mathematics*. SIAM, Philadelphia, 2000.

Griffies, S., M., and W. Hallberg, R., Biharmonic friction with a Smagorinsky-like viscosity for use in large-scale eddy-permitting ocean models, *Mon. Wea. Rev.*, *128*(8), 2935–2946, 2000.

Griffies, S. M., The Gent-McWilliams skew flux, *J. Phys. Oceanogr.*, *28*, 831–841, 1998.

Griffies, S. M., A. Gnanadesikan, R. C. Pacanowski, V. Larichev, J. K. Dukowicz, and R. D. Smith, Isoneutral diffusion in a z-coordinate ocean model, *J. Phys. Oceanogr.*, *28*, 805–830, 1998.

Grosfeld, K., R. Gerdes, and J. Determann, Thermohaline circulation and interaction between ice shelf cavities and the adjacent open water, *J. Geophys. Res.*, *102*(C7), 15,595–15,610, 1997.

Haney, R., Surface thermal boundary conditions for ocean circulation models, *J. Phys. Oceanogr.*, *1*, 241–248, 1971.

Heimbach, P., C. Wunsch, R. M. Ponte, G. Forget, C. Hill, and J. Utke, Timescales and regions of the sensitivity of Atlantic meridional volume and heat transport: Toward observing system design, *Deep Sea Research Part II: Topical Studies in Oceanography*, *58*(17), 1858–1879, 2011.

Held, I., and M. Suarez, A proposal for the intercomparison of the dynamical cores of atmospheric general circulation models, *Bulletin of the American Meteorological Society*, *75(10)*, 1825–1830, 1994.

Helfand, H. M., and J. C. Labraga, Design of a non-singular level 2.5 second-order closure model for the prediction of atmospheric turbulence., *J. Atmos. Sci.*, *45*, 113–132, 1988.

Helfand, H. M., and S. D. Schubert, Climatology of the simulated great plains low-level jet and its contribution to the continental moisture budget of the united states., *J. Clim.*, *8*, 784–806, 1995.

Hellmer, H. H., and D. J. Olbers, A two-dimensional model of the thermohaline circulation under an ice shelf, *Antarct. Sci.*, *1*, 325–336, 1989.

Hibler, III, W. D., A dynamic thermodynamic sea ice model, *J. Phys. Oceanogr.*, *9*, 815–846, 1979.

Hibler, III, W. D., Modeling a variable thickness sea ice cover, *Mon. Wea. Rev.*, *1*, 1943–1973, 1980.

Hibler, III, W. D., The role of sea ice dynamics in modeling co_2 increases, in *Climate processes and climate sensitivity*, edited by J. E. Hansen, and T. Takahashi, vol. 29 of *Geophysical Monograph*, pp. 238–253. AGU, Washington, D.C., 1984.

Hibler, III, W. D., and K. Bryan, A diagnostic ice-ocean model, *J. Phys. Oceanogr.*, *17*(7), 987–1015, 1987.

Hibler, III, W. D., and C. F. Ip, The effect of sea ice rheology on Arcitc buoy drift, in *Ice Mechanics*, edited by J. P. Dempsey, and Y. D. S. Rajapakse, vol. 204 of *AMD*, pp. 255–264. Am. Soc. of Mech. Eng., New York, 1995.

Hibler, III, W. D., and E. M. Schulson, On modeling sea-ice fracture and flow in numerical investigations of climate, *Ann. Glaciol.*, *25*, 26–32, 1997.

Hill, C., and J. Marshall, Application of a parallel navier-stokes model to ocean circulation in parallel computational fluid dynamics, in *Implementations and Results Using Parallel Computers*, edited by N. S. A. Ecer, J. Periaux, and S. Taylor, pp. 545–552. Elsevier Science B.V.: New York, 1995.

Hill, C., M. Follows, V. Bugnion, and J. Marshall, Spatial and temporal impacts of ocean general circulation on carbon sequestration, *Global Biogeochemical Cycles*, submitted, 2002.

Hoe, J. C., C. Hill, and A. Adcroft, A personal supercomputer for climate research, in *In Proceedings of the ACM/IEEE Super-Computing Confrence*, p. 10.1109/SC.1999. IEEE, 1999.

Holland, D. M., and A. Jenkins, Modeling thermodynamic ice-ocean interactions at the base of an ice shelf, *J. Phys. Oceanogr.*, *29*(8), 1787–1800, 1999.

Holland, W., and L. B. Lin, On the origin of mesoscale eddies and their contribution to the general circulation of the ocean. i. a preliminary numerical experiment, *J. Phys. Oceanogr.*, *5*, 642–657, 1975a.

Holland, W. R., The role of mesoscale eddies in the general circulation of the ocean–numerical experiments using a wind-driven quasi-geostrophic model, *Journal of Physical Oceanography*, *8*, 363–392, 1978.

Hundsdorfer, W., and R. A. Trompert, Method of lines and direct discretization: a comparison for linear advection, *Applied Numerical Mathematics*, *13*(6), 469–490, 1994. doi:10.1016/0168-9274(94)90009-4.

Hunke, E. C., Viscous-plastic sea ice dynamics with the EVP model: Linearization issues, *J. Comput. Phys.*, *170*, 18–38, 2001. doi:10.1006/jcph.2001.6710.

Hunke, E. C., and J. K. Dukowicz, An elastic-viscous-plastic model for sea ice dynamics, *J. Phys. Oceanogr.*, *27*, 1849–1867, 1997.

Hutchings, J. K., H. Jasak, and S. W. Laxon, A strength implicit correction scheme for the viscous-plastic sea ice model, *Ocean Modelling*, *7*(1–2), 111–133, 2004. doi:10.1016/S1463-5003(03)00040-4.

Inness, P. M., S. Slingo, R. Woolnough, B. Neale, and V. Pope, Organization of tropical convection in a gcm with verying vertical resolution; implications for the simulation of the madden-julian oscillation., *Climate Dyn.*, *17*, 777–794, 2001.

Jackett, D. R., and T. J. McDougall, Minimal adjustment of hydrographic profiles to achieve static stability, *J. Atmos. Ocean. Technol.*, *12*(4), 381–389, 1995.

Jenkins, A., H. H. Hellmer, and D. M. Holland, The role of meltwater advection in the formulation of conservative boundary conditions at an ice-ocean interface, *J. Phys. Oceanogr.*, *31*, 285–296, 2001.

Jiang, S., P. H. Stone, and P. Malanotte-Rizzoli, An assessment of the Geophysical Fluid Dynamics Laboratory ocean model with coarse resolution: Annual-mean climatology, *J. Geophys. Res.*, *104*(C11), 25,623–25,645, 1999.

Kalnay, E., et al., The NMC/NCAR 40-year reanalysis project, *Bull. Am. Met. Soc.*, *77*, 437–471, 1996.

Kimmritz, M., S. Danilov, and M. Losch, On the convergence of the modified elastic-viscous-plastic method of solving for sea-ice dynamics, *J. Comput. Phys.*, *296*, 90–100, 2015. doi:10.1016/j.jcp.2015.04.051.

Kimmritz, M., S. Danilov, and M. Losch, The adaptive EVP method for solving the sea ice momentum equation, *Ocean Modelling*, 2016. Available from: `http://mitgcm.org/~mlosch/adaptiveEVP_accepted.pdf`.

Klymak, J. M., and S. M. Legg, A simple mixing scheme for models that resolve breaking internal waves, *Ocean Modelling*, *33*, 224–234, 2010. doi:10.1016/j.ocemod.2010.02.005.

Knoll, D., and D. Keyes, Jacobian-free Newton-Krylov methods: a survey of approaches and applications, *J. Comput. Phys.*, *193*, 357–397, 2004. doi:10.1016/j.jcp.2003.08.010.

Kondo, J., Air-sea bulk transfer coefficients in diabatic conditions., *Bound. Layer Meteorol.*, *9*, 91–112, 1975.

Koster, R. D., and M. J. Suarez, A simplified treatment of sib's land surface albedo parameterization., NASA Technical Memorandum 104538, National Aeronautics and Space Administration, NASA; Goddard Space Flight Center; Greenbelt (MD), 20771; USA, http://www.gmao.nasa.gov/, 1991.

Koster, R. D., and M. J. Suarez, Modeling the land surface boundary in climate models as a composite of independent vegetation stands., *J. Geophys. Res.*, *97*, 2697–2715, 1992.

Lacis, A. A., and J. E. Hansen, A parameterization for the absorption of solar radiation in the earth's atmosphere., *J. Atmos. Sci.*, *31*, 118–133, 1974.

Large, W., J. McWilliams, and S. Doney, Oceanic vertical mixing: A review and a model with nonlocal boundary layer parameterization, *Rev. Geophys.*, *32*, 363–403, 1994.

Large, W. G., and S. Pond, Open ocean momentum flux measurements in moderate to strong winds., *J. Phys. Oceanogr.*, *11*, 324–336, 1981.

Large, W. G., G. Danabasoglu, S. C. Doney, and J. C. McWilliams, Sensitivity to surface forcing and boundary layer mixing in a global ocean model: Annual-mean climatology, *J. Phys. Oceanogr.*, *27*(11), 2418–2447, 1997.

Leith, C. E., Large eddy simulation of complex engineering and geophysical flows, *Physics of Fluids*, *10*, 1409–1416, 1968.

Leith, C. E., Stochastic models of chaotic systems, *Physica D.*, *98*, 481–491, 1996.

Lemieux, J.-F., and B. Tremblay, Numerical convergence of viscous-plastic sea ice models, *J. Geophys. Res.*, *114*(C05009), 2009. doi:10.1029/2008JC005017.

Lemieux, J.-F., B. Tremblay, J. Sedláček, P. Tupper, S. Thomas, D. Huard, and J.-P. Auclair, Improving the numerical convergence of viscous-plastic sea ice models with the Jacobian-free Newton-Krylov method, *J. Comput. Phys.*, *229*, 2840–2852, 2010. doi:10.1016/j.jcp.2009.12.011c.

Lemieux, J.-F., D. Knoll, B. Tremblay, D. M. Holland, and M. Losch, A comparison of the Jacobian-free Newton-Krylov method and the EVP model for solving the sea ice momentum equation with a viscous-plastic formulation: a serial algorithm study, *J. Comput. Phys.*, *231*(17), 5926–5944, 2012. doi:10.1016/j.jcp.2012.05.024.

Leppäranta, M., A growth model for black ice, snow ican and snow thickness in subarctic basins, *Nordic Hydrology*, *14*, 59–70, 1983.

Levitus, S., and T.P.Boyer, World Ocean Atlas 1994 Volume 3: Salinity, Tech. rep., NOAA Atlas NESDIS 3, 1994a.

Levitus, S., and T.P.Boyer, World Ocean Atlas 1994 Volume 4: Temperature, Tech. rep., NOAA Atlas NESDIS 4, 1994b.

Losch, M., Modeling ice shelf cavities in a z-coordinate ocean general circulation model, *J. Geophys. Res.*, *113*(C08043), 2008. doi:10.1029/2007JC004368.

Losch, M., D. Menemenlis, J.-M. Campin, P. Heimbach, and C. Hill, On the formulation of sea-ice models. Part 1: Effects of different solver implementations and parameterizations, *Ocean Modelling*, *33*(1–2), 129–144, 2010. doi:10.1016/j.ocemod.2009.12.008.

Losch, M., A. Fuchs, J.-F. Lemieux, and A. Vanselow, A parallel Jacobian-free Newton-Krylov solver for a coupled sea ice-ocean model, *J. Comput. Phys.*, *257*(A), 901–910, 2014. doi:10.1016/j.jcp.2013.09.026.

Manabe, S., K. Bryan, and M. J. Spelman, A global ocean-atmosphere climate model with seasonal variation for future studies of climate sensitivity, *Dyn. Atmos. Oceans*, *3*(393–426), 1979.

Marotzke, J., R. Giering, K. Zhang, D. Stammer, C. Hill, and T. Lee, Construction of the adjoint mit ocean general circulation model and application to atlantic heat transport variability, *J. Geophys. Res.*, *104, C12*, 29,529–29,547, 1999.

Marshall, J., A. Adcroft, C. Hill, L. Perelman, and C. Heisey, A finite-volume, incompressible navier stokes model for studies of the ocean on parallel computers, *J. Geophys. Res.*, *102*(C3), 5753–5766, 1997a. Available from: `http://mitgcm.org/pdfs/96JC02776.pdf`.

Marshall, J., C. Hill, L. Perelman, and A. Adcroft, Hydrostatic, quasi-hydrostatic, and non-hydrostatic ocean modeling, *J. Geophys. Res.*, *102*(C3), 5733–5752, 1997b. Available from: `http://mitgcm.org/pdfs/96JC02775.pdf`.

Marshall, J., H. Jones, and C. Hill, Efficient ocean modeling using non-hydrostatic algorithms, *J. Mar. Sys.*, *18*, 115–134, 1998. Available from: `http://mitgcm.org/pdfs/journal_of_marine_systems_1998.pdf`, doi:10.1016/S0924-7963%2898%2900008-6.

Marshall, J., A. Adcroft, J.-M. Campin, C. Hill, and A. White, Atmosphere-ocean modeling exploiting fluid isomorphisms, *Mon. Wea. Rev.*, *132*, 2882–2894, 2004. Available from: `http://mitgcm.org/pdfs/a_o_iso.pdf`, doi:10.1175/MWR2835.1.

Marshall, J., E. Shuckburgh, H. Jones, and C. Hill, Estimates and implications of surface eddy diffusivity in the southern ocean derived from tracer transport, *J. Phys. Oceanogr.*, *36*, 1806–1821, 2006.

McDougall, T. J., D. R. Jackett, D. G. Wright, and R. Feistel, Accurate and computationally efficient algorithms for potential temperature and density of seawater, *J. Atmos. Ocean. Technol.*, 2003.

McKinley, G., M. J. Follows, and J. C. Marshall, Mechanisms of air-sea co_2 flux variability in the equatorial pacific and the north atlantic, *Global Biogeochemical Cycles*, *18*(doi:10.1029/2003GB002179), 2004.

Mellor, G., and T. Yamada, Development of turbulence closure model for geophysical fluid problems., *Rev. Geophys. Space Phys.*, *20*(4), 851–875, 1982.

Menemenlis, D., et al., NASA supercomputer improves prospects for ocean climate research, *Eos Trans. AGU*, *86*(9), 89, 95–96, 2005.

Message Passing Interface Forum, MPI: A message-passing interface standard (version 2.0), Tech. rep., mpi-forum.org, 1998.

Molod, A., Running GCM physics and dynamics on different grids: algorithm and tests, *Tellus*, *61A*, 381–393, 2009.

Molteni, F., Atmospheric simulations using a GCM with simplified physical parametrization, I: Model climatology and variability in multidecadal experiments, *Clim. Dynamics*, *20*, 175–191, 2003.

Moorthi, S., and M. J. Suarez, Relaxed arakawa schubert: A parameterization of moist convection for general circulation models., *Mon. Wea. Rev.*, *120*, 978–1002, 1992.

Moum, J., Energy-containing scales of turbulence in the ocean thermocline, *J. Geophys. Res.*, *101 (C3)*, 14095–14109, 1996.

Naumann, U., J. Utke, P. Heimbach, C. Hill, D. Ozyurt, C. Wunsch, M. Fagan, N. Tallent, and M. Strout, Adjoint code by source transformation with openad/f, in *European Conference on Computational Fluid Dynamics ECCOMAS CFD 2006*, edited by P. Wesseling, E. O nate and J. Periaux, p. . TU Delft, 2006.

Orlanski, I., A simple boundary condition for unbounded hyperbolic flows, *J. Comput. Phys.*, *21*, 251–269, 1976.

Pacanowski, R., and S. Philander, Parameterization of vertical mixing in numerical models of tropical oceans, *J. Phys. Ocean.*, *11*, 1443–1451, 1981.

Paluszkiewicz, T., and R. Romea, A one-dimensional model for the parameterization of deep convection in the ocean, *Dyn. Atmos. Oceans*, *26*, 95–130, 1997.

Panofsky, H. A., Tower micrometeorology., in *Workshop on Micrometeorology*, edited by D. A. Haugen. American Meteorological Society, 1973.

Parkinson, C. L., and W. M. Washington, A Large-Scale Numerical Model of Sea Ice, *J. Geophys. Res.*, *84*(C1), 311–337, 1979.

Redi, M. H., Oceanic isopycnal mixing by coordinate rotation, *Journal of Physical Oceanography*, *12*, 1154–1158, 1982.

Restrepo, J., G. Leaf, and A. Griewank, Circumventing storage limitations in variational data assimilation studies, *SIAM J. Sci. Comput.*, *19*, 1586–1605, 1998.

Rew, R. K., G. P. Davis, S. Emmerson, and H. Davies, NetCDF User's Guide for C, FORTRAN 77, and FORTRAN 90, an interface for data access, version 3, Report, Unidata Program Center, Boulder, Colorado, http://www.unidata.ucar.edu/packages/netcdf/, 1997.

Roe, P., Some contributions to the modelling of discontinuous flows, in *Large-Scale Computations in Fluid Mechanics*, edited by B. Engquist, S. Osher, and R. Somerville, vol. 22 of *Lectures in Applied Mathematics*, pp. 163–193. American Mathematical Society, Providence, RI, 1985.

Rosenfield, J. E., M. R. Schoeberl, and M. A. Geller, A computation of the stratospheric diabatic circulation using an accurate radiative transfer model, *J. Atmos. Sci.*, *44*, 859–876, 1987.

Saad, Y., A flexible inner-outer preconditioned GMRES method, *SIAM J. Sci. Comput.*, *14*(2), 461–469, 1993.

Seim, H. E., and M. C. Gregg, Detailed observations of a naturally occurring shear instability, *J. Geophys. Res.*, *99 (C5)*, 10049–10073, 1994.

Semtner, Jr., A. J., A model for the thermodynamic growth of sea ice in numerical investigations of climate, *J. Phys. Oceanogr.*, *6*, 379–389, 1976.

Shapiro, R., Smoothing, filtering, and boundary effects, *Reviews of Geophysics and Space Physics*, *8*(2), 359–387, 1970.

Smagorinsky, J., General circulation experiments with the primitive equations I: The basic experiment, *Monthly Weather Review*, *91*(3), 99–164, 1963.

Smagorinsky, J., Large eddy simulation of complex engineering and geophysical flows, in *Evolution of Physical Oceanography*, edited by B. Galperin, and S. A. Orszag, pp. 3–36. Cambridge University Press, 1993.

Stammer, D., C. Wunsch, R. Giering, Q. Zhang, J. Marotzke, J. Marshall, and C. Hill, The global ocean circulation estimated from topex/poseidon altimetry and a general circulation model, Technical Report 49, Center for Global Change Science, Massachusetts Institute of Technology, Cambridge (MA), USA, 1997.

Stammer, D., C. Wunsch, R. Giering, C. Eckert, P. Heimbach, J. Marotzke, A. Adcroft, C. Hill, and J. Marshall, The global ocean circulation and transports during 1992 – 1997, estimated from ocean observations and a general circulation model., *J. Geophys. Res.*, *107(C9)*, 3118, 2002a. doi:10.1029/2001JC000888.

Stevens, D. P., On open boundary conditions for three dimensional primitive equation ocean circulation models, *Geophys. Astrophys. Fl. Dyn.*, *51*, 103–133, 1990.

Stommel, H., The western intensification of wind-driven ocean currents, *Trans. Am. Geophys. Union*, *29*, 206, 1948.

Sud, Y. C., and A. Molod, The roles of dry convection, cloud-radiation feedback processes and the influence of recent improvements in the parameterization of convection in the gla gcm., *Mon. Wea. Rev.*, *116*, 2366–2387, 1988.

Takacs, L. L., and M. Suarez, Dynamical aspects of climate simulations using the geos general circulation model, NASA Technical Memorandum 104606 Volume 10, National Aeronautics and Space Administration, NASA; Goddard Space Flight Center; Greenbelt (MD), 20771; USA, http://www.gmao.nasa.gov/, 1996.

Thorpe, S., Turbulence and mixing in a scottish loch, *Phil. Trans. R. Soc. Lond.*, *286*, 125–181, 1977.

Trenberth, K., J. Olson, and W. Large, A global wind stress climatology based on ecmwf analyses, Tech. Rep. NCAR/TN-338+STR, NCAR, Boulder, CO, 1989.

Trenberth, K. M., J. Olson, and W. G. Large, The mean annual cycle in Global Ocean wind stress, *J. Phys. Oceanogr.*, *20*, 1742–1760, 1990.

Utke, J., U. Naumann, M. Fagan, N. Tallent, M. Strout, P. Heimbach, C. Hill, D. Ozyurt, and C. Wunsch, Openad/f: A modular open source tool for automatic differentiation of fortran codes, *ACM Transactions on Mathematical Software*, *34(4)*, , 2008.

Visbeck, M., J. Marshall, T. Haine, and M. Spall, Specification of eddy transfer coefficients in coarse-resolution ocean circulation models, *J. Phys. Oceanogr.*, *27*(3), 381–402, 1997.

Wajsowicz, R., A consistent formulation of the anisotropic stress tensor for use in models of the large-scale ocean circulation, *J. Comput. Phys.*, *105*(2), 333–338, 1993.

Wanninkhof, R., Relationship between wind speed and gas exchange over the ocean, *J. Geophys. Res.*, *97*, 7373–7382, 1992.

Wesson, J. C., and M. C. Gregg, Mixing at camarinal sill in the strait of gibraltar, *J. Geophys. Res.*, *99 (C5)*, 9847–9878, 1994.

Williamson, D., and J. Olsen, A comparison of semi-langrangian and eulerian polar climate simulations., *Mon. Wea. Rev.*, *126*, 991–1000, 1998.

Winton, M., A reformulated three-layer sea ice model, *J. Atmos. Ocean. Technol.*, *17*, 525–531, 2000.

Yaglom, A. M., and B. A. Kader, Heat and mass transfer between a rough wall and turbulent fluid flow at high reynolds and peclet numbers., *J. Fluid Mech.*, *62*, 601–623, 1974.

Yamada, T., A numerical experiment on pollutant dispersion in a horizontally-homogenious atmospheric boundary layer., *Atmos. Environ.*, *11*, 1015–1024, 1977.

Zhang, J., and W. D. Hibler, III, On an efficient numerical method for modeling sea ice dynamics, *J. Geophys. Res.*, *102*(C4), 8691–8702, 1997.

Zhang, J., W. D. Hibler, III, M. Steele, and D. A. Rothrock, Arctic ice-ocean modeling with and without climate restoring, *J. Phys. Oceanogr.*, *28*, 191–217, 1998.

Zhou, J., Y. Sud, and K.-M. Lau, Impact of orographically induced gravity wave drag in the gla gcm, *Q. J. R. Meteorol. Soc.*, *122*, 903–927, 1995.