MOM6 Documentation

Release 0.2a3

Alistair Adcroft Robert Hallberg Stephen Griffies
Matthew Harrison Brandon Reichl Niki Zadeh
John Krasting Nic Hannah

Oct 30, 2020

CONTENTS

Contents:

CHAPTER

ONE

ABOUT THIS DOCUMENTATION

This readthedocs site hosts the nascent MOM6 user documentation. MOM6 documentation is distributed over several formats and locations with each site serving a different purpose.

Here is where to find particular documentation:

Download, compile and run Installation documentation is in the form of user-driven (editable) wiki attached to the MOM6-examples GitHub repository. Goto https://github.com/NOAA-GFDL/MOM6-examples/wiki and look at "Getting Started".

Installation, compilation and running are platform specific operations for which we can only provide templates (as is done in on the wiki) but for which MOM6 developers cannot possibly support since every platform is different. Normally a user needs to know where libraries (such as netcdf and MPI) and compilers are on their system but once these have been established the documented compile process can be adpated to the local system.

User guide This site provides a high-level overview of the model as well as the API reference (documentation of source code).

The user guide is written in reStructuredText (.rst files) that reside in `docs/` of the MOM6 source code. The rst files are processed by sphinx and hosted on readthedocs.

The API reference is generated documentation - we use doxygen for in-code documentation. The Fortran doxygen format is rather cumbersome for writing and we therefore use the C++ .dox files for much of this documentation.

Repository policies Policies governing how to the repositories are organized and operated live at https://github.com/NOAA-GFDL/MOM6-examples/wiki/MOM6-repository-policies.

Developer guide Beyond the API reference above, developer specific wiki pages are attached to the *MOM6 code repository <https://github.com/NOAA-GFDL/MOM6/wiki>*.

CHAPTER

TWO

EQUATIONS

The model equations are the layer-integrated vector-invariant form of the hydrostatic primitive equations (either Boussinesq or non-Boussinesq).

We present the equations starting from the hydrostatic Boussinesq equation in height coordinates and progress through vector-invariant and general-coordinate equations to the final equations used in the A.L.E. algorithm, taken from [1].

2.1 Notation for equations

2.1.1 Symbols for variables

z refers to elevation (or height), increasing upward so that for much of the ocean z is negative.

x and y are the Cartesian horizontal coordinates.

λ and ϕ are the geographic coordinates on a sphere (longitude and latitude respectively).

Horizontal components of velocity are indicated by u and v and vertical component by w .

p is pressure and Φ is geo-potential:

$$\Phi = gz.$$

The thermodynamic state variables are usually salinity, S , and potential temperature, θ or the absolute salinity and conservative temperature, depending on the equation of state. ρ is in-situ density.

2.1.2 Vector notation

The three-dimensional velocity vector is denoted $\boldsymbol{v}$

$$\boldsymbol{v} = \boldsymbol{u} + \widehat{\boldsymbol{k}} w,$$

where $\widehat{\boldsymbol{k}}$ is the unit vector pointed in the upward vertical direction and $\boldsymbol{u} = (u, v, 0)$ is the horizontal component of velocity normal to the vertical.

The gradient operator without a suffix is three dimensional:

$$\boldsymbol{\nabla} = (\boldsymbol{\nabla}_z, \partial_z).$$

but a suffix indicates a lateral gradient along a surface of constant property indicated by the suffix:

$$\boldsymbol{\nabla}_z = \left(\partial_x|_z \, , \partial_y|_z \, , 0\right).$$

2.2 Governing Equations

The Boussinesq hydrostatic equations of motion in height coordinates are

$$D_t \boldsymbol{u} + f\widehat{\boldsymbol{k}} \times \boldsymbol{u} + \frac{\rho}{\rho_o}\boldsymbol{\nabla}_z \Phi + \frac{1}{\rho_o}\boldsymbol{\nabla}_z p = \boldsymbol{\mathcal{F}} \quad \text{momentum} \tag{2.1}$$

$$\rho \frac{\partial \Phi}{\partial z} + \frac{\partial p}{\partial z} = 0 \quad \text{hydrostatic} \tag{2.2}$$

$$\boldsymbol{\nabla}_z \cdot \boldsymbol{u} + \frac{\partial w}{\partial z} = 0 \quad \text{thickness} \tag{2.3}$$

$$D_t \theta = \mathcal{N}^{\gamma}_{\theta} - \frac{\partial J^{(z)}_{\theta}}{\partial z} \quad \text{potential temp} \tag{2.4}$$

$$D_t S = \mathcal{N}^{\gamma}_{S} - \frac{\partial J^{(z)}_{S}}{\partial z} \quad \text{salinity} \tag{2.5}$$

$$\rho = \rho(S, \theta, z) \quad \text{equation of state.} \tag{2.6}$$

where notation is described in *Notation for equations* , $\boldsymbol{\mathcal{F}}$ represents the accelerations due to the divergence of stresses including those provided through boundary interactions.

The prognostic thermodynamic variables are potential temperature, θ , and salinity S , which are related to *in situ* density ρ through the [48] equation of state. In the potential temperature and salinity equations, fluxes due to diabatic, vertically oriented processes are indicated by $J^{(z)}$. The tendency due to the convergence of fluxes oriented along neutral directions is indicated by $\boldsymbol{\mathcal{N}}^{\gamma}$. Our implementation of this *neutral diffusion* parameterization is detailed in Shao et al. (personal comm.)

The total derivative is

$$D_t \equiv \tfrac{\partial}{\partial t} + \boldsymbol{v} \cdot \boldsymbol{\nabla} \tag{2.7}$$

$$= \tfrac{\partial}{\partial t} + \boldsymbol{u} \cdot \boldsymbol{\nabla}_z + w \tfrac{\partial}{\partial z}. \tag{2.8}$$

The non-divergence of flow allows a total derivative to be re-written in flux form:

$$D_t \theta = \tfrac{\partial}{\partial t} + \boldsymbol{\nabla} \cdot (\boldsymbol{v}\theta) \tag{2.9}$$

$$= \tfrac{\partial}{\partial t} + \boldsymbol{\nabla}_z \cdot (\boldsymbol{u}\theta) + \tfrac{\partial (w\theta)}{\partial z}. \tag{2.10}$$

The above equations of motion can thus be written as:

$$D_t \boldsymbol{u} + f\widehat{\boldsymbol{k}} \times \boldsymbol{u} + \frac{\rho}{\rho_o}\boldsymbol{\nabla}_z \Phi + \frac{1}{\rho_o}\boldsymbol{\nabla}_z p = \boldsymbol{\mathcal{F}} \quad \text{momentum} \tag{2.11}$$

$$\rho \frac{\partial \Phi}{\partial z} + \frac{\partial p}{\partial z} = 0 \quad \text{hydrostatic} \tag{2.12}$$

$$\boldsymbol{\nabla}_z \cdot \boldsymbol{u} + \frac{\partial w}{\partial z} = 0 \quad \text{thickness} \tag{2.13}$$

$$\frac{\partial \theta}{\partial t} + \boldsymbol{\nabla}_z \cdot (\boldsymbol{u}\theta) + \frac{\partial (w\theta)}{\partial z} = \mathcal{N}^{\gamma}_{\theta} - \frac{\partial J^{(z)}_{\theta}}{\partial z} \quad \text{potential temp} \tag{2.14}$$

$$\frac{\partial S}{\partial t} + \boldsymbol{\nabla}_z \cdot (\boldsymbol{u}S) + \frac{\partial (wS)}{\partial z} = \mathcal{N}^{\gamma}_{S} - \frac{\partial J^{(z)}_{S}}{\partial z} \quad \text{salinity} \tag{2.15}$$

$$\rho = \rho(S, \theta, z) \quad \text{equation of state.} \tag{2.16}$$

2.2.1 Vector Invariant Equations

MOM6 solves the momentum equations written in vector-invariant form.

A vector identity allows the total derivative of velocity to be written in the vector-invariant form:

$$
\begin{aligned}
D_t \boldsymbol{u} &= \partial_t \boldsymbol{u} + \boldsymbol{v} \cdot \boldsymbol{\nabla} \boldsymbol{u} && (2.17)\\
&= \partial_t \boldsymbol{u} + \boldsymbol{u} \cdot \boldsymbol{\nabla}_z \boldsymbol{u} + w \partial_z \boldsymbol{u} && (2.18)\\
&= \partial_t \boldsymbol{u} + (\boldsymbol{\nabla} \times \boldsymbol{u}) \times \boldsymbol{v} + \boldsymbol{\nabla} \underbrace{\frac{1}{2} |\boldsymbol{u}|^2}_{\equiv K}. && (2.19)
\end{aligned}
$$

The flux-form equations of motion in height coordinates can thus be written succinctly as:

$$
\begin{aligned}
\partial_t \boldsymbol{u} + \left(f \widehat{\boldsymbol{k}} + \boldsymbol{\nabla} \times \boldsymbol{u} \right) \times \boldsymbol{v} + \boldsymbol{\nabla} K + \frac{\rho}{\rho_o} \boldsymbol{\nabla} \Phi + \frac{1}{\rho_o} \boldsymbol{\nabla} p &= \boldsymbol{\mathcal{F}} && \text{momentum} && (2.20)\\
\boldsymbol{\nabla}_z \cdot \boldsymbol{u} + \partial_z w &= 0 && \text{thickness} && (2.21)\\
\partial_t \theta + \boldsymbol{\nabla}_z \cdot (\boldsymbol{u} \theta) + \partial_z (w \theta) &= \boldsymbol{\mathcal{N}}_\theta^\gamma - \tfrac{\partial J_\theta^{(z)}}{\partial z} && \text{potential temp} && (2.22)\\
\partial_t S + \boldsymbol{\nabla}_z \cdot (\boldsymbol{u} S) + \partial_z (w S) &= \boldsymbol{\mathcal{N}}_S^\gamma - \tfrac{\partial J_S^{(z)}}{\partial z} && \text{salinity} && (2.23)\\
\rho &= \rho(S, \theta, z) && \text{equation of state} && (2.24)
\end{aligned}
$$

where the horizontal momentum equations and vertical hydrostatic balance equation have been written as a single three-dimensional equation.

2.3 General coordinate equations

General coordinate equations

Transforming to a vertical coordinate $r(z, x, y, t)$, with $\dot{r} = \frac{\partial r}{\partial t} \ldots$

The Boussinesq hydrostatic equations of motion in general-coordinate r are:

$$
\begin{aligned}
\rho_0 \left(\frac{\partial \mathbf{u}}{\partial t} + (f + \zeta)\, \hat{\mathbf{z}} \times \mathbf{u} + \dot{r}\, \frac{\partial \mathbf{u}}{\partial r} + \nabla_r K \right) &= -\nabla_r p - \rho \nabla_r \Phi + \boldsymbol{\mathcal{F}} && \text{momentum} && (2.25)\\
\rho\, \frac{\partial \Phi}{\partial r} + \frac{\partial p}{\partial r} &= 0 && \text{hydrostatic} && (2.26)\\
\frac{\partial z_r}{\partial t} + \nabla_r \cdot\, (z_r\, \mathbf{u}) + \frac{\partial (z_r\, \dot{r})}{\partial r} &= 0 && \text{thickness} && (2.27)\\
\frac{\partial (\theta\, z_r)}{\partial t} + \nabla_r \cdot (\theta z_r\, \mathbf{u}) + \frac{\partial (\theta\, z_r\, \dot{r})}{\partial r} &= z_r \boldsymbol{\mathcal{N}}_\theta^\gamma - \tfrac{\partial J_\theta^{(z)}}{\partial r} && \text{potential temp} && (2.28)\\
\frac{\partial (S\, z_r)}{\partial t} + \nabla_r \cdot (S\, z_r\, \mathbf{u}) + \frac{\partial (S\, z_r\, \dot{r})}{\partial r} &= z_r \boldsymbol{\mathcal{N}}_S^\gamma - \tfrac{\partial J_S^{(z)}}{\partial r} && \text{salinity} && (2.29)\\
\rho &= \rho\,(S, \theta, -g \rho_0 z(r)) && \text{equation of state.} && (2.30)
\end{aligned}
$$

The time derivatives are now computed with the generalized vertical coordinate fixed rather than the geopotential. We introduced the specific thickness, $z_r = \partial z / \partial r$, which measures the inverse vertical stratification of the vertical coordinate surfaces.

Similar to [6] , MOM6 is discretized in the vertical by integrating between surfaces of r to yield layer equations where the layer thickness is $h = \int z_r dr$ and variables are treated as finite volume averages over each layer:

$$\rho_0 \left(\frac{\partial \mathbf{u}}{\partial t} + \frac{(f+\zeta)}{h} \hat{\mathbf{z}} \times h\,\mathbf{u} + \underbrace{\dot{r}\,\frac{\partial \mathbf{u}}{\partial r}} + \nabla_r K \right) = -\nabla_r p - \rho \nabla_r \Phi + \boldsymbol{\mathcal{F}} \quad \text{momentum} \tag{2.31}$$

$$\rho\,\delta_r \Phi + \delta_r p = 0 \quad \text{hydrostatic} \tag{2.32}$$

$$\frac{\partial h}{\partial t} + \nabla_r \cdot (h\,\mathbf{u}) + \underbrace{\delta_r(z_r \dot{r})} = 0 \quad \text{thickness} \tag{2.33}$$

$$\frac{\partial (\theta\, h)}{\partial t} + \nabla_r \cdot (\theta h\,\mathbf{u}) + \underbrace{\delta_r(\theta\, z_r \dot{r})} = h \boldsymbol{\mathcal{N}}^{\gamma}_{\theta} - \delta_r J^{(z)}_{\theta} \quad \text{potential temp} \tag{2.34}$$

$$\frac{\partial (S\, h)}{\partial t} + \nabla_r \cdot (S\, h\,\mathbf{u}) + \underbrace{\delta_r(S\, z_r \dot{r})} = h \boldsymbol{\mathcal{N}}^{\gamma}_{S} - \delta_r J^{(z)}_{S} \quad \text{salinity} \tag{2.35}$$

$$\rho = \rho\,(S, \theta, -g \rho_0 z(r)) \quad \text{equation of state,} \tag{2.36}$$

where $\delta_r = \mathrm{d}r\,(\partial/\partial r)$ is the discrete vertical difference operator. The pressure gradient accelerations in the momentum equation are written in continuous-in-the-vertical form for brevity; the exact discretization is detailed in [3] . The MOM6 time-stepping algorithm integrates the above layer-averaged equations forward allowing the vertical grid to follow the motion, i.e. $\dot{r} = 0$, so that the underbraced terms are dropped. This approach is generally known as the Lagrangian method but here the Lagrangian method is used only in the vertical direction. After each Lagrangian step, a remap step is applied that generates a new vertical grid of the user's choosing. The ocean state is then mapped from the old to the new grid. The physical state is not meant to change during the remap step, yet truncation errors make remapping imperfect. We employ high-order accurate reconstructions to minimize errors introduced during the remap step ([46] , [47]). The connection between time-stepping and remapping is described in section *ALE Timestep* .

2.4 Specifics

Specifics of the Ocean Model Equations

We here provide more details of the terms appearing in the ocean model equations described in *General coordinate equations* .

2.4.1 Horizontal Momentum Equation

Equation is the horizontal momentum equation written in its vector-invariant advective form with $\mathbf{u} = \hat{\mathbf{x}}\,u + \hat{\mathbf{y}}\,v$ the horizontal velocity, p the hydrostatic pressure, f the Coriolis parameter, and

$$\zeta^{(r)} = \hat{\mathbf{z}} \cdot (\nabla_r \times \mathbf{u})$$

the vertical component of the vorticity using ∇_r for the curl operator. The discretization of the Coriolis term is the enstrophy conserving scheme of [39] . The geopotential coordinate, z , has a value $z = 0$ at a resting ocean surface, $z = \eta(x, y, t)$ at the ocean free surface, and $z = -H(x, y)$ at the ocean bottom. We use the Boussinesq approximation (volume conserving kinematics) with $\rho_0 = 1035$ kg m^{-3} the reference density. Time and horizontal derivatives are computed holding the generalized vertical coordinate fixed rather than the geopotential

$$\frac{\partial}{\partial t} = \left[\frac{\partial}{\partial t}\right]_r \qquad \nabla_r = \hat{\mathbf{x}} \left[\frac{\partial}{\partial x}\right]_r + \hat{\mathbf{y}} \left[\frac{\partial}{\partial y}\right]_r .$$

The transport of seawater crossing surfaces of constant r is measured by the dia-surface velocity component (see section 6.7 of [18])

$$\frac{\partial z}{\partial r}\,\frac{\mathrm{D}r}{\mathrm{D}t} = z_r\,\dot{r},$$

with z_r the specific thickness that is assumed one-signed throughout the ocean, and $\mathrm{D}/\mathrm{D}t$ the material time derivative operator. In the ocean interior where r is aligned with isopycnals, the dia-surface velocity becomes the diapycnal velocity whose value is directly related to irreversible processes such as mixing that act on potential temperature and salinity. In the unstratified mixed layers, $r = z^*$ so that $z_r \dot{r} = (\partial z/\partial z^*)\,\mathrm{D}z^*/\mathrm{D}t$, which is close to the familiar vertical velocity component $\mathrm{D}z/\mathrm{D}t$.

Viscous dissipation (Laplacian and biharmonic friction following [19]) and mechanical boundary forces (winds, bottom stress) contribute to the divergence of the deviatoric (symmetric and trace-free) stress tensor, $\boldsymbol{\mathcal{F}} = \nabla \cdot \tau$. MOM6 and the real ocean have no vertical sidewalls, and MOM6 treats all solid-earth boundaries with bottom stress parameterized as a quadratic drag.

2.4.2 Hydrostatic balance

Equation is the discrete version of the hydrostatic balance. The horizontal pressure gradient force is implemented as a contact force following the method of [3] . These equations differ from [6] who uses the Montgomery potential to calculate pressure gradient accelerations.

2.4.3 Thickness and tracer equations

Volume conservation appears in the form of a prognostic flux-form layer thickness equation, with the non-negative layer thickness given by

$$h = \frac{\partial z}{\partial r}\,\mathrm{d}r,$$

where $\mathrm{d}r$ is the thickness of a layer in r -space (e.g., the density difference between target density classes or the thickness between target depths). The layer thickness increases where horizontal thickness fluxes converge, $\nabla_r \cdot (h_k\,\mathbf{u}) < 0$, and where dia-surface flow converges, $\delta_r(z_r\,\dot{r}) < 0$. The volume flux $h_k\mathbf{u}$ is computed using the quasi-third order PPM scheme ([9]) using a positive-definite limiter rather than the monotonic limiter. This last choice avoids limiting of positive extrema and thus retains third-order accuracy everywhere except near vanishing layers.

Transport in the thickness equation is discretized compatibly with that in the flux-form potential temperature and salinity equations and . Compatibility is required to maintain global and layer integrated conservation properties for volume, heat, and salt. Tracer reconstruction for transport uses PPM with monotonic limiters but using third order interpolation for edge values. This reduces the size of the stencil which helps the computational efficiency of the transport scheme. The flux convergences, $\boldsymbol{\mathcal{N}}_\theta^\gamma$ and $\boldsymbol{\mathcal{N}}_S^\gamma$, provide subgrid scale neutral diffusion for the potential temperature and salinity, whereas $\delta_r J_\theta^{(z)}$ and $\delta_r J_S^{(z)}$ provide subgrid scale vertical diffusion as well as boundary fluxes. In the interior, both subgrid fluxes vanish when their respective tracers are spatially uniform, thus ensuring that the tracer equation reduces to the thickness equation when the tracer is uniform.

Parameterized subgrid scale advection from the submesoscale ([13]) and mesoscale ([17]) parameterizations are combined with the lateral advection of thickness and tracer, thus providing a residual mean advective transport for the scalar fields. Furthermore, we implement subgrid advective terms solely as lateral transports, thus interpreting them as layer bolus transport as appropriate for vertical Lagrangian models rather than a three-dimensional eddy-induced advection as appropriate for vertical Eulerian models (see [32] for details).

2.4.4 Equation of state

The equation of state,, determines *in situ* density as a function of potential temperature, salinity, and pressure. We evaluate the pressure in the equation of state according to $-g\,\rho_0\,z$. Doing so maintains energetic consistency for the Boussinesq fluid according to section 2.4.3 of [44] . We make use of the [48] equation of state so that θ is potential temperature and S is the practical salinity. Although MOM6 has the more updated equation of state from [49] , the required changes for thermodynamic variables were implemented only after the basic model configuration was developed. Time constraints on model development prompted us to retain usage of [48] for OM4.

The freezing point of seawater is approximated as

$$T_f = -0.054S - 7.75 \times 10^{-08} p,$$

where p is in units of Pascals and S is in units of 1×10^{-3} . When the local temperature anywhere in the ocean column falls below the freezing point, the water-equivalent volume of ice is calculated and the fusion heat locally added back to the ocean to raise the liquid seawater temperature back to the freezing point. The frozen water and salt are sent to the sea-ice model.

2.5 ALE

Basics of the Vertical Lagrangian-Remap Method in MOM6

As discussed by [2] , there are two general classes of algorithms that frame how hydrostatic ocean models are formulated. The two classes differ in how they treat the vertical direction. Quasi-Eulerian methods follow the approach traditionally used in geopotential coordinate models, whereby vertical motion is diagnosed via the continuity equation. Quasi-Lagrangian methods are traditionally used by layered isopycnal models, with the Lagrangian approach specifying motion that crosses coordinate surfaces. Indeed, such dia-surface flow can be set to zero using Lagrangian methods for studies of adiabatic dynamics. MOM6 makes use of the vertical Lagrangian remap method, as pioneered for ocean modeling by [6] , which is a limit case of the Arbitrary-Lagrangian-Eulerian method ([24]). Dia-surface transport is implemented via a remapping so that the method can be summarized as the Lagrangian plus remap approach and is essentially a one-dimensional version of the incremental remapping of [10] .

The MOM6 implementation of the vertical Lagrangian-remap method makes use of two general steps. The first evolves the ocean state forward in time according to a vertical Lagrangian limit with $\dot{r} = 0$. Hence, the horizontal momentum, thickness, and tracers are time stepped with the red terms removed in equations, ,, and. All advective transport thus occurs within a layer as defined by constant r -surfaces so that the volume within each layer is fixed. All other terms are retained in their full form, including subgrid scale terms that contribute to the transfer of tracer and momentum into distinct r layers (e.g., dia-surface diffusion of tracer and velocity). Maintaining constant volume within a layer yet allowing for tracers to move between layers engenders no inconsistency between tracer and thickness evolution. The reason is that tracer diffusion, even dia-surface diffusion, does not transfer volume.

The second step in the algorithm comprises the generation of a new vertical grid following a prescription, such as whether the grid should align with isopcynals or constant z^* or a combination. The ocean state is then vertically remapped to the newly generated vertical grid. The remapping step incorporates dia-surface transfer of properties, with such transfer depending on the prescription given for the vertical grid generation. To minimize discretization errors and the associated spurious mixing, the remapping step makes use of the high order accurate methods developed by [46] and [47] .

The underlying algorithm for treatment of the vertical can be related to operator-splitting of the red terms in equations . If we consider, for simplicity, an Euler-forward update for a time-step Δt , the time-stepping for the continuity and temperature equation can be summarized as

$$h^\dagger = h^{(n)} - \Delta t \left[\nabla_r \cdot (h\,\mathbf{u})\right] \quad \text{thickness} \tag{2.37}$$

$$\theta^\dagger\, h^\dagger = \theta^{(n)}\, h^{(n)} - \Delta t \left[\nabla_r \cdot (\theta h\,\mathbf{u}) - h\boldsymbol{\mathcal{N}}_\theta^\gamma + \delta_r J_\theta^{(z)}\right] \quad \text{potential temp} \tag{2.38}$$

$$h^{(n+1)} = h^\dagger - \Delta t\, \delta_r \left(z_r \dot{r}\right) \quad \text{move grid} \tag{2.39}$$

$$\theta^{(n+1)} h^{(n+1)} = \theta^\dagger h^\dagger - \Delta t\, \delta_r \left(z_r \dot{r}\, \theta^\dagger\right) \quad \text{remap temperature.} \tag{2.40}$$

Substituting into recovers a time-discrete form of. The intermediate quantities indicated by † -symbols are the result of the vertical Lagrangian step of the algorithm. What were the red terms in the continuous-in-time equations are used to evolve the the intermediate quantities to the final updated quantities each step. In MOM6, equation is essentially used to define the dia-surface transport $z_r\dot{r}$ by prescribing $h^{(n+1)}$. For example, to recover a z-coordinate model, $h^{(n+1)} = \Delta z$, and $z_r\dot{r}$ becomes the Eulerian vertical velocity, w .

Within the above framework for evolving the ocean state, we make use of a standard split-explicit time stepping method by decomposing the horizontal momentum equation into its fast (depth integrated) and slow (deviation from depth integrated) components. Furthermore, we follow the methods of [23] to ensure that the free surface resulting from time stepping the depth integrated thickness equation (i.e., the free surface equation) is consistent with the sum of the thicknesses that result from time stepping the layer thickness equations for each of the discretized layers; i.e., $\sum_k h = H + \eta$.

CHAPTER

THREE

SPATIAL DISCRETIZATION

The model equations are the layer-integrated vector-invariant form of the hydrostatic primitive equations (either Boussinesq or non-Boussinesq).

We present the equations starting from the hydrostatic Boussinesq equation in height coordinates and progress through vector-invariant and general-coordinate equations to the final equations used in the A.L.E. algorithm.

3.1 Discrete Horizontal and Vertical Grids

Discrete Horizontal and Vertical Grids

3.1.1 Horizontal grids

The placement of model variables on the horizontal C-grid is illustrated here:

Scalars are located at the h -points, velocities are staggered such that u -points and v -points are not co-located, and vorticities are located at q -points. The indexing for points (i, j) in the logically-rectangular domain is such that i increases in the x direction (eastward for spherical polar coordinates), and j increases in the y direction (northward for spherical polar coordinates). A q -point with indices (i, j) lies to the upper right (northeast) of the h -point with the same indices. The index for the vertical dimension k increases with depth, although the vertical coordinate z , measured from the mean surface level $z = 0$, decreases with depth.

When the horizontal grid is generated, it is actually computed on the "supergrid" at twice the nominal resolution of the model. The grid file contains the grid metrics and the areas of this fine grid. The model then decomposes it into the four staggered grids, along with computing the grid metrics as shown here:

The model carries both the metrics as well as their inverses, for instance, IdyT = 1/dyT. There are also the areas and the inverse areas for all four grid locations. areaT and areaBu are the sum of the four areas from the supergrid surrounding each h-point and each q-point, respectively. The velocity faces can be partially blocked and their areas are adjusted accordingly, where dy_Cu and dx_Cv are the blocked distances at u and v points, respectively.

$$\text{areaCu}_{i,j} = dxCu_{i,j} * dy\_Cu_{i,j} \tag{3.1}$$

$$\text{areaCv}_{i,j} = dx\_Cv_{i,j} * dyCv_{i,j} \tag{3.2}$$

$$\text{IareaCu}_{i,j} = 1/\text{areaCu}_{i,j} \tag{3.3}$$

$$\text{IareaCv}_{i,j} = 1/\text{areaCv}_{i,j} \tag{3.4}$$

The horizontal grids can be spherical, tripole, regional, or cubed sphere. The default is for grids to be re-entrant in the x -direction; this needs to be turned off for regional grids.

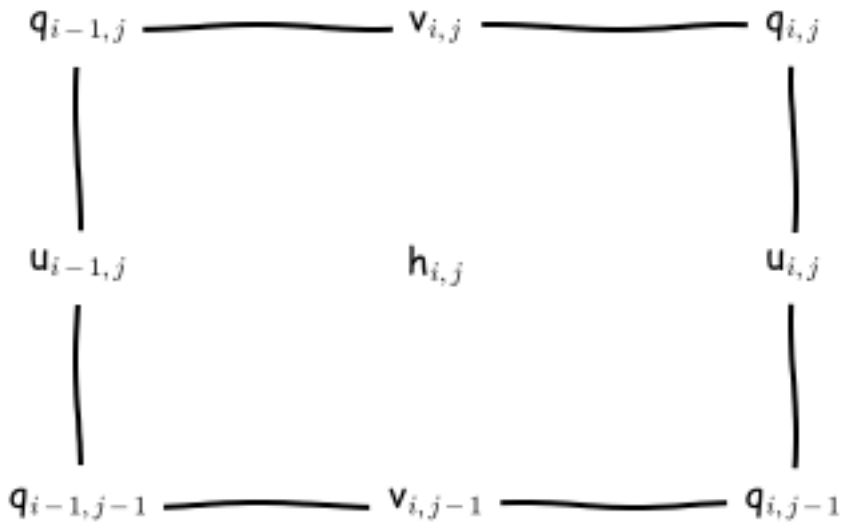

Fig. 1: MOM6 uses an Arakawa C grid staggering of variables with a North-East indexing convention.

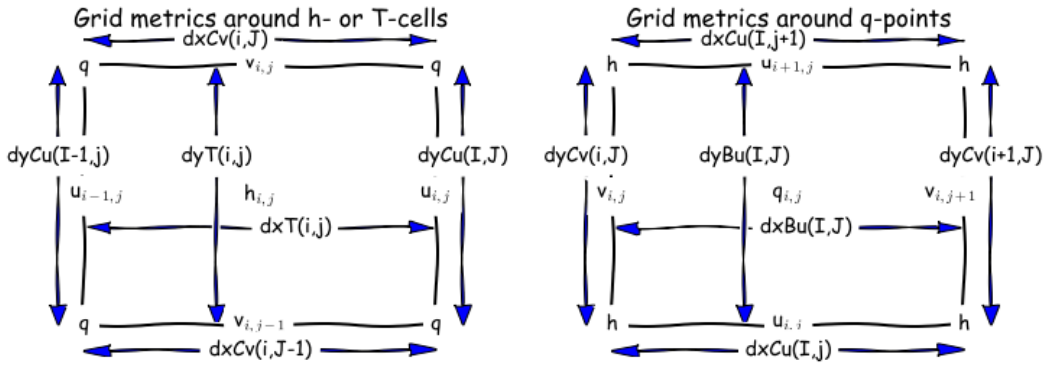

Fig. 2: The grid metrics around both h-points and q-points.

3.1.2 Vertical grids

The placement of model variables in the vertical is illustrated here:

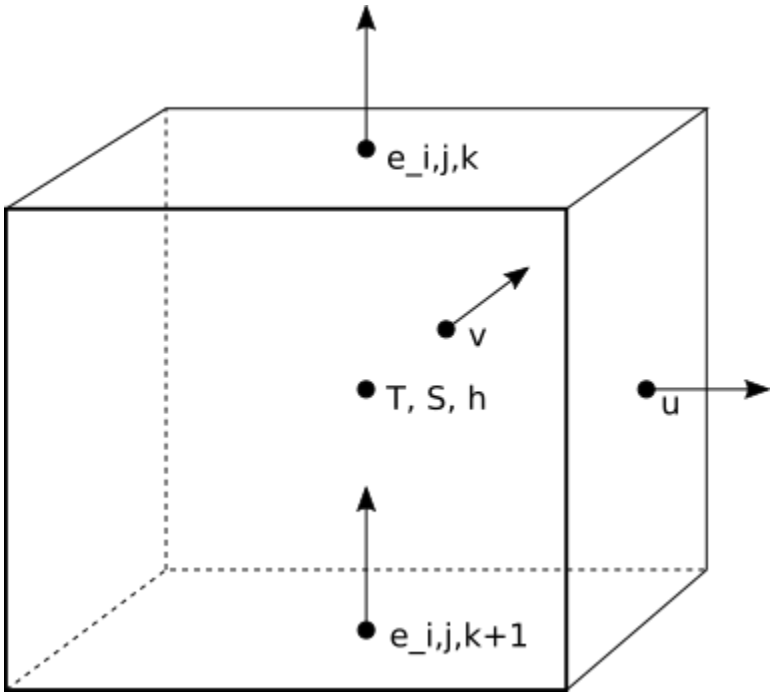

Fig. 3: The MOM6 interfaces are at vertical location e which are separated by the layer thicknesses h.

The vertical coordinate is Lagrangian in that the interfaces between the layers are free to move up and down with time. The interfaces have target depths or target densities, depeneding on the desired vertical coordinate system. They can even have target sigma values for terrain-following coordinates or you can design a hybrid coordinate in which different interfaces have differing behavior. In any case, the interfaces move with the fluid during the dynamic timesteps and then get reset during a remapping operation. See section *ALE Timestep* for details.

3.2 Finite Difference Operators

Finite Difference Operators

3.3 PPM Advection Scheme

PPM Advection Scheme

Following [9] and [8] , we use the Piecewise Parabolic Method (PPM) to represent values within the model cells. Each cell is assumed to have a piecewise parabolic representation, which is uniquely prescribed by conservation and the two edge values. This method has the following features:

- The PPM approach is conservative.
- The (unlimited) order of accuracy is determined by the estimates of the edge values.
- Monotonicity is ensured by adjusting the edge values to flatten the profile.

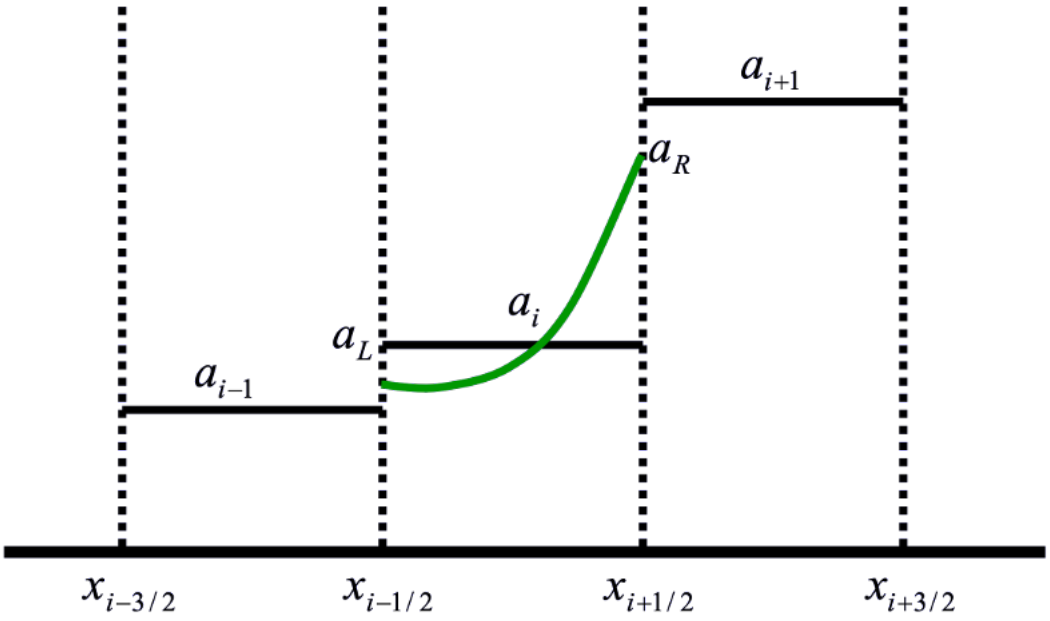

Fig. 4: The parabolic representation of a field within a cell.

An example is shown in this figure:

$$x_i' \equiv \frac{x - x_{i-1/2}}{\Delta x_i}$$

$$\Delta x_i \equiv x_{i+1/2} - x_{i-1/2}$$

$$c \equiv u\Delta t / \Delta x_i$$

$$A_i(x') = a_L + (a_R - a_L)x_i' + a_6 x_i'(1 - x_i')$$

$$a_6 = 6a_i - 3(a_R + a_L)$$

$$\begin{aligned} a_i &= \int_0^1 A_i(x_i')dx_i' = \int_0^1 a_L + (a_R - a_L)x_i' + a_6 x_i'(1 - x_i')dx_i' && (3.5)\\ &= \left[a_L x_i' + \tfrac{1}{2}(a_R - a_L)x_i'^2 + a_6\left(\tfrac{1}{2}x_i'^2 - \tfrac{1}{3}x_i'^3\right)\right]_0^1 && (3.6)\\ &= \tfrac{1}{2}(a_R + a_L) + \tfrac{1}{6}a_6 && (3.7) \end{aligned}$$

$$\begin{aligned} F_{i+1/2} &= \tfrac{1}{\Delta t}\int_{x_{i+1/2}-u\Delta t}^{x_{i+1/2}} A_i^n(x)dx = \tfrac{\Delta x}{\Delta t}\int_{1-c}^1 A_i(x_i')dx_i' && (3.8)\\ &= \tfrac{\Delta x}{\Delta t}\left[a_L x_i' + \tfrac{1}{2}(a_R - a_L)x_i'^2 + a_6\left(\tfrac{1}{2}x_i'^2 - \tfrac{1}{3}x_i'^3\right)\right]_{1-c}^1 && (3.9)\\ &= \tfrac{\Delta x}{\Delta t}\left[a_L c + (a_R - a_L + a_6)\left(c - \tfrac{1}{2}c^2\right) - a_6\left(c - c^2 + \tfrac{1}{3}c^3\right)\right] && (3.10)\\ &= u\left[a_R + \tfrac{1}{2}(a_L - a_R)c + a_6\left(\tfrac{1}{2}c - \tfrac{1}{3}c^2\right)\right] && (3.11) \end{aligned}$$

The choice of a_L and a_R is not unique, but can be done according to [9] (CW84) or [25] (H3) as mentioned in *Tracer Advection* .

3.4 Discrete Coriolis Term

3.4.1 Coriolis Term

In general, the discrete equations are written as simple difference equations based on the Arakawa C-grid as described in section *Horizontal grids* . One of the more interesting exceptions is the Coriolis term. It is computed in the form shown in, or:

$$\frac{(f+\zeta)}{h}\,\hat{\mathbf{z}}\times h\,\mathbf{u}$$

This term needs to be evaluated at u points for the v equation and vice versa, plus we need to keep the thickness, h , positive definite. MOM6 contains a number of options for how to compute this term.

- SADOURNY75_ENERGY Sadourny [39] figured out how to conserve energy or enstrophy but not both. This option is energy conserving. The term in the u equation becomes:

$$\frac{1}{4dx}\left(q_{i,j}(vh_{i+1,j}+vh_{i,j})+q_{i,j-1}(vh_{i+1,j-1}+vh_{i,j-1})\right)$$

where $q = \frac{f+\zeta}{h}$ and h is an area-weighted average of the four thicknesses surrounding the q point, such that it is guaranteed to be positive definite.

There is a variant on this scheme with the CORIOLIS_EN_DIS option. If true, two estimates of the thickness fluxes vh are used to estimate the Coriolis term, and the one that dissipates energy relative to the other one is used.

- SADOURNY75_ENSTRO Also from [39] , this option is enstrophy conserving.

$$\frac{1}{8dx}(q_{i,j}+q_{i,j-1})((vh_{i+1,j}+vh_{i,j})+(vh_{i+1,j-1}+vh_{i,j-1}))$$

- ARAKAWA_LAMB81 From [5] is a scheme which is both energy and enstrophy conserving. Its weaknesses are a large stencil and differing thickness stencils in the numerator and denominator. This scheme and several others (with differing values of a, b, c, d and ep) are implemented as:

$$\frac{1}{dx}(a_{i,j}vh_{i+1,j} \quad +b_{i,j}vh_{i,j}+d_{i,j}vh_{i+1,j-1}+c_{i,j}vh_{i,j-1} \tag{3.12}$$

$$+ep_{i,j}*uh_{i-1,j}-ep_{i+1,j}*uh_{i+1,j}) \tag{3.13}$$

with

$$a_{i,j} = \tfrac{1}{24}(2.0*(q_{i+1,j}+q_{i,j-1})+(q_{i,j}+q_{i+1,j-1})) \tag{3.14}$$

$$b_{i,j} = \tfrac{1}{24}((q_{i,j}+q_{i-1,j-1})+2.0*(q_{i-1,j}+q_{i,j-1})) \tag{3.15}$$

$$c_{i,j} = \tfrac{1}{24}(2.0*(q_{i,j}+q_{i-1,j-1})+(q_{i-1,j}+q_{i,j-1})) \tag{3.16}$$

$$d_{i,j} = \tfrac{1}{24}((q_{i+1,j}+q_{i,j-1})+2.0*(q_{i,j}+q_{i+1,j-1})) \tag{3.17}$$

$$ep_{i,j} = \tfrac{1}{24}((q_{i,j}-q_{i-1,j-1})+(q_{i-1,j}-q_{i,j-1})) \tag{3.18}$$

- ARAKAWA_HSU90 From [4] is a scheme which always conserves energy and conserves enstrophy in the limit of non-divergent flow. This one has a larger stencil than Sadourny's energy scheme, but it's much better behaved in terms of handling vanishing layers than Arakawa and Lamb. This scheme is implemented with:

$$\frac{1}{dx}(a_{i,j}vh_{i+1,j}+b_{i,j}vh_{i,j}+d_{i,j}vh_{i+1,j-1}+c_{i,j}vh_{i,j-1})$$

and

$$a_{i,j} = \frac{1}{12}(q_{i,j} + (q_{i+1,j} + q_{i,j-1})) \tag{3.19}$$
$$b_{i,j} = \frac{1}{12}(q_{i,j} + (q_{i-1,j} + q_{i,j-1})) \tag{3.20}$$
$$c_{i,j} = \frac{1}{12}(q_{i,j} + (q_{i-1,j-1} + q_{i,j-1})) \tag{3.21}$$
$$d_{i,j} = \frac{1}{12}(q_{i,j} + (q_{i+1,j-1} + q_{i,j-1})) \tag{3.22}$$

- ARAKAWA_LAMB_BLEND This is a blending of Arakawa and Lamb, Arakawa and Hsu, and the Sadourny Energy scheme. There are weights CORIOLIS_BLEND_WT_LIN and CORIOLIS_BLEND_F_EFF_MAX to control this scheme. The equation is the same as for Arakawa and Lamb, but the values of a, b, c, d and ep differ when the pure Arakawa and Lamb scheme breaks down due to thickness variations.
- ROBUST_ENSTRO An enstrophy-conserving scheme which is robust to vanishing layers.

Some of these options also support the BOUND_CORIOLIS flag. If true, the Coriolis terms in the u equation are bounded by the four estimates of $\frac{(f+\zeta)}{h}vh$ from the four neighboring v points, and similarly in the v equation. This option would have no effect on the SADOURNY75_ENERGY scheme if it were possible to use centered difference thickness fluxes.

Wall boundary conditions

Two sets of boundary conditions have been coded in the definition of relative vorticity. These are written as:

NOSLIP defined (in spherical coordinates):

$$\text{relvort} = dv/dx \text{ (east \& west), with } v = 0. \tag{3.23}$$
$$\text{relvort} = -\sec(\phi) * d(u\cos(\phi))/dy \text{ (north \& south), with } u = 0. \tag{3.24}$$

Free slip (NOSLIP not defined):

$$\text{relvort} = 0 \text{ (all boundaries)}$$

with ϕ defined as latitude. The free slip boundary condition is much more natural on a C-grid.

3.5 Discrete Pressure Gradient Term

3.5.1 Pressure Gradient Term

Following [3] , the horizontal momentum equation in the general coordinate r can be written as:

$$\frac{\partial \vec{u}}{\partial t} + \nabla_r \Phi + \alpha \nabla_r p = \mathcal{F}$$

where the vector $\mathcal{F}$ represents all the forcing terms other than the pressure gradient. Here, $\vec{u}$ is the horizontal component of the velocity, Φ is the geopotential:

$$\Phi = gz$$

$\alpha = 1/\rho$ is the specific volume and p is the pressure. The gradient operator is a gradient along the coordinate surface r .

MOM6 offers two options, an older one using a Montgomery potential as described in hallberg1997 and [43] . However, it can have the instability described in [22] . The version described here is that in [3] and is the recommended

option (ANALYTIC_FV_PGF = True). The paper describes the Boussinesq form while the code supports that and also a non-Boussinesq form.

In two dimensions (x and p), we can integrate the zonal component of the momentum equation above over a finite volume:

$$-\int dx \int dp \frac{\partial u}{\partial t} = \int dx \int dp \, \frac{\partial \Phi}{\partial x}\Big|_p \tag{3.25}$$

$$= \int_{p_{br}}^{p_{tr}} \Phi dp + \int_{p_{tr}}^{p_{tl}} \Phi dp + \int_{p_{tl}}^{p_{bl}} \Phi dp \quad + \int_{p_{bl}}^{p_{br}} \Phi dp \tag{3.26}$$

We convert to line integrals thanks to the Leibniz rule. See the figure for the location of the line integral ranges:

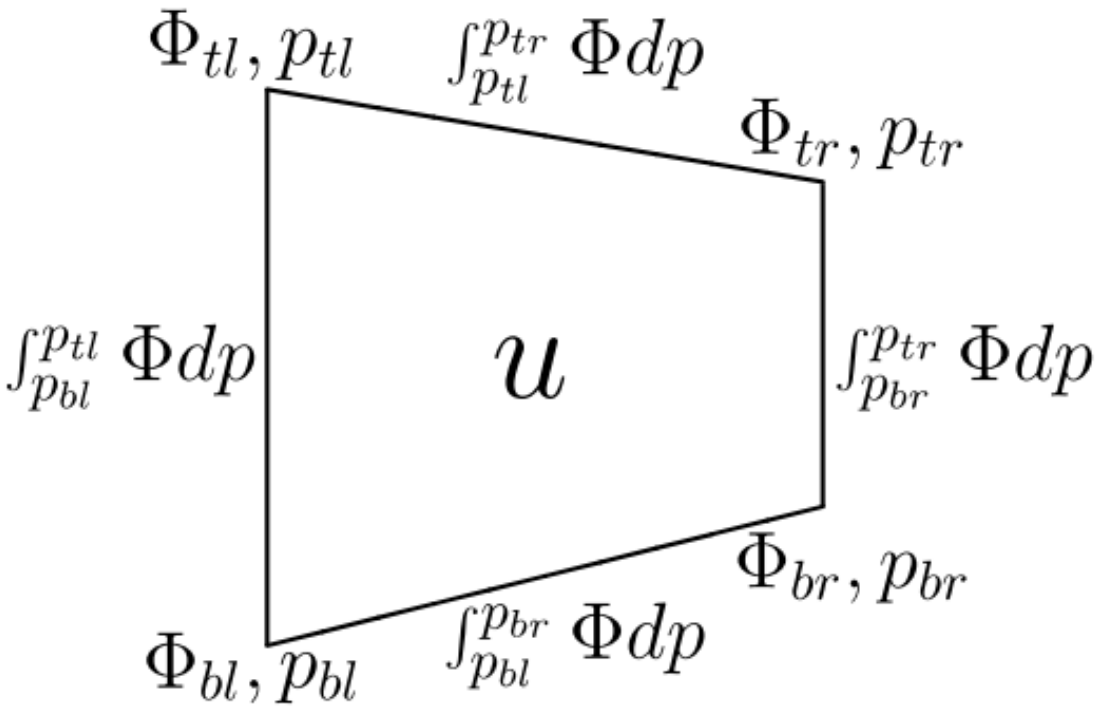

Fig. 5: Schematic of the finite volume used for integrating the u-component of momentum. The thermodynamic variables $$ and s reside on the sides of the depicted volume and are considered uniform for the vertical extent of the volume but with linear variation in the horizontal. The volume is depicted in (x, p) space so p is linear around the volume but $$ can vary arbitrarily along the edges.

The only approximations made are (i) that the potential temperature θ and the salinity s can be represented continuously in the vertical within each layer although discontinuities between layers are allowed and (ii) that θ and s can be represented continuously along each layer. MOM6 has options for piecewise constant (PCM), piecewise linear (PLM), and piecewise parabolic (PPM) in the vertical.

If we use the Wright equation of state ([48]), we can integrate the above integrals analytically. This equation of state can be written as:

$$\alpha(s, \theta, p) = A(s, \theta) + \frac{\lambda(s, \theta)}{P(s, \theta) + p}$$

where A, λ and P are functions only of s and θ . The integral form of hydrostatic balance is:

$$\Phi(p_t) - \Phi(p_b) = \int_{p_t}^{p_b} \alpha(s, \theta, p) dp$$

Assuming piecewise constant values for θ and s and the above equation of state, we get:

$$\begin{aligned} \Phi(p_t) - \Phi(p_b) &= \textstyle\int_{p_t}^{p_b} \alpha(s, \theta, p) dp && (3.27) \\ &= (p_b - p_t)A + \lambda \ln \left| \tfrac{P+p_b}{P+p_t} \right| && (3.28) \\ &= \Delta p \left(A + \tfrac{\lambda}{P+\overline{p}} \tfrac{1}{2\epsilon} \ln \left| \tfrac{1+\epsilon}{1-\epsilon} \right| \right) && (3.29) \end{aligned}$$

which is the exact solution for the continuum only if θ and s are uniform in the interval p_t to p_b . Here, we have introduced the variables:

$$\Delta p = p_b - p_t$$

$$\overline{p} = \frac{1}{2}(p_t + p_b)$$

and

$$\epsilon = \frac{\Delta p}{2(P + \overline{p})}$$

We will show later that $\epsilon \ll 1$. Note the series expansion:

$$\frac{1}{2\epsilon} \ln \left| \frac{1+\epsilon}{1-\epsilon} \right| = \sum_{n=1}^{\infty} \frac{\epsilon^{2n-2}}{2n-1} = 1 + \frac{\epsilon^2}{3} + \frac{\epsilon^4}{5} + \cdots \forall |\epsilon| \leq 1$$

Typical values for the deep ocean with 100 m layer thickness are 6×10^8 Pa and 10^6 Pa, respectively, yielding $\epsilon \sim 8 \times 10^{-4}$ and a corresponding accuracy in the geopotential height calculation of $\frac{\lambda \epsilon^3}{g} \sim 10^{-5}$ m. For this value of ϵ , the series converges with just three terms. In MOM6, we use series rather than the intrinsic log function , since the log is machine dependent and insufficiently accurate. In extreme circumstances, $\Delta p \sim 6 \times 10^7$ Pa (limited by the depth of the ocean) for which $\epsilon \sim 0.04$ with geopotential height errors of order 1 m. In this case, the series converges to machine precision with six terms.

The finite volume acceleration is expression terms of four integrals around the volume, $\int \Phi dp$. The side integrals can be calculated by direct integration of , which gives:

$$\begin{aligned} \int_{p_t}^{p_b} \Phi dp &= \Delta p \left(\Phi_b + \tfrac{1}{2} A \Delta p + \lambda \left(1 - \tfrac{1-\epsilon}{2\epsilon} \ln \left| \tfrac{1+\epsilon}{1-\epsilon} \right| \right) \right) && (3.30) \\ &= \Delta p \left(\Phi_b + \tfrac{1}{2} A \Delta p + \lambda \left(1 - (1-\epsilon) \left(1 + \tfrac{\epsilon^2}{3} + \tfrac{\epsilon^4}{5} + \cdots \right) \right) \right) && (3.31) \\ &= \Delta p \left(\Phi_b + \tfrac{1}{2} A \Delta p + \lambda \left(\epsilon - (1-\epsilon)\epsilon^2 \left(\tfrac{1}{3} + \tfrac{\epsilon^2}{5} + \cdots \right) \right) \right) && (3.32) \end{aligned}$$

where $\Phi, \Delta p, P, A$ and λ are each evaluated on the left or right side of the volume.

The top and bottom integrals in must allow for the effect of varying θ and s on A, λ and P . We evaluate these integrals numerically using sixth-order quadrature; Boole's rule requires evaluating the coefficients in the equation of state at five points, two of which have already been evaluated for the side integrals. For efficiency, we linearly interpolate the coefficients A, P and λ between the end points, which seems to make very little difference to the solution. We also

verified that tenth-order quadrature makes little difference to the solution. The values of the top and bottom integrals are carried upward in a hydrostatic-like integration, obtained as follows:

$$\int_{p_{tl}}^{p_{tr}} \Phi_t dp = (p_{tr} - p_{tl}) \int_0^1 \Phi_t dx \tag{3.33}$$

$$= (p_{tr} - p_{tl}) \int_0^1 \left(\Phi_b + A(x)\Delta p(x) + \lambda(x) \ln \left| \frac{1+\epsilon(x)}{1-\epsilon(x)} \right| \right) dx \tag{3.34}$$

$$= (p_{tr} - p_{tl}) \int_0^1 \Phi_b dx \tag{3.35}$$

$$+ \int_0^1 \Delta p(x) \left(A(x) + \frac{\lambda(x)}{P(x)+\overline{p}(x)} \sum_{n=1}^{\infty} \frac{\epsilon^{2n-2}}{2n-1} \right) dx \tag{3.36}$$

The first integral is either known from the top integral of the layer below or the boundary condition at the ocean bottom. The second integral is evaluated numerically.

All the above definite integrals are specific to the Wright equation of state; the use of a different equation of state requires analytic integration of the appropriate equations. We have found, however, that high-order numerical integration appears to be sufficient. Although the numerical implementation is more general (allowing the use of arbitrary equations of state), it is significantly more expensive and so we advocate the analytic implementation for efficiency.

3.6 Energetic Consistency

Energetic Consistency

3.7 Discrete Open Boundary Conditions

Discrete Open Boundary Conditions

CHAPTER

FOUR

TIME DISCRETIZATION

In MOM6, it is possible to have at least four different timesteps: the barotropic timestep, the baroclinic (momentum dynamics) timestep, the tracer timestep, and the remapping interval. There can also be a forcing timestep on which model coupling can occur.

4.1 Barotropic Momentum Equations

Barotropic Momentum Equations

The barotropic equations are timestepped on a relatively short timestep compared to the rest of the model. Since the timestep constraints for this are known, the barotropic timestep is computed at runtime.

The 2-d linear momentum equations with integrated continuity are:

$$\frac{\partial \eta}{\partial t} + \nabla \cdot ((D+\eta)\vec{u}_{BT} h_k) = P - E$$

$$\frac{\partial \vec{u}_{BT}}{\partial t} = -g\nabla\eta - f\hat{z} \times \vec{u}_{BT} + \vec{F}_{BT}$$

where

$$\vec{u}_{BT} \equiv \frac{1}{D+\eta}\int_{-D}^{\eta} \vec{u} dz$$

and $\vec{F}_{BT}$ is the barotropic momentum forcing from baroclinic processes. Note that explicit mass fluxes such as evaporation and precipitation change the model volume explicitly.

In the mode splitting between baroclinic and barotropic processes, it is important to include the contribution of free surface waves on the internal interface heights on the pressure gradient force, shown here as g_{Eff} :

$$\frac{\partial p}{\partial z} = -\rho g$$

$$g_{Eff} = g + \frac{\partial}{\partial \eta}\left[\frac{1}{D+\eta}\int_{-D}^{\eta} p dz\right]$$

The barotropic momentum equation then becomes:

$$\frac{\partial \vec{u}_{BT}}{\partial t} + f\hat{z} \times \vec{u}_{BT} + \frac{1}{\rho_0}\nabla g_{Eff}\eta = \text{Residual}$$

Without including the internal wave motion in the barotropic equations, one can generate instabilities ([7] , [20]).

4.2 Baroclinic Momentum Equations

Baroclinic Momentum Equations

The baroclinic momentum equations are the stacked shallow water equations:

$$\frac{\partial \vec{u}_k}{\partial t} + (f + \nabla_s \times \vec{u}_k)\hat{z} \times \vec{u}_k = -\frac{\nabla_s p_k}{\rho} - \nabla_s(\phi_k + \frac{1}{2}||\vec{u}_k||^2) + \frac{\nabla \cdot \tilde{\tau}_k}{\rho}$$

$$\frac{\partial h_k}{\partial t} + \nabla_s \cdot (\vec{u} h_k) = 0$$

The timestepping for these equations is a (quasi?) second-order Runge-Kutta step for the inertial oscillations and a forward-backward Euler step for the pressure (gravity) waves. Using the graphical notation from [41] , it looks like:

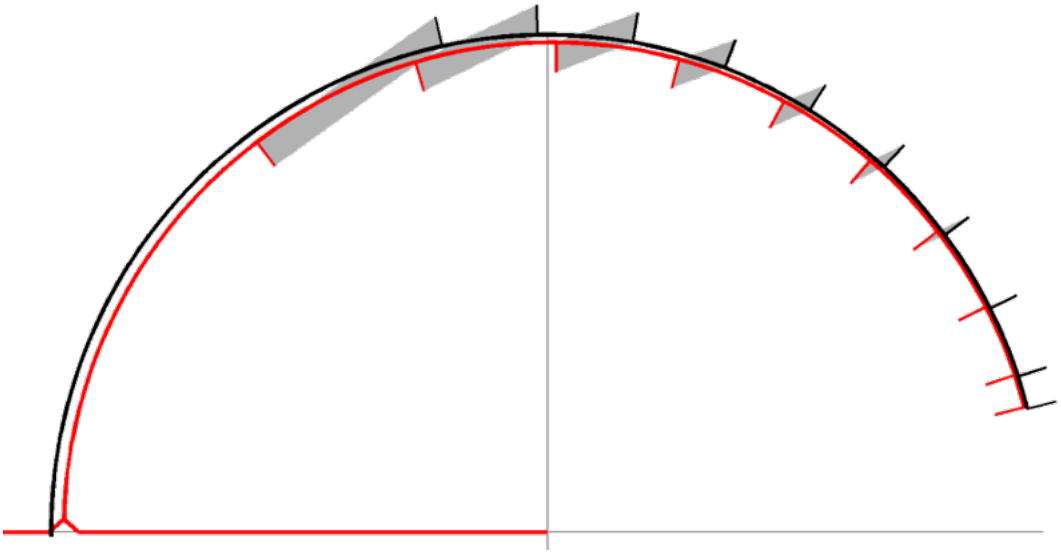

Fig. 1: Graphical notation for timestepping schemes in which the black line represents the ideal solution and the red line shows the actual solution. Phase errors are represented by the grey shapes between the bars normal to the circle.

The timestep used in ROMS looks instead like:

The ROMS timestepping has smaller phase errors, strong damping at high frequency. The MOM6 use as a global climate model has made the phase errors of lower priority. However, the phase errors may become more problematic for future uses of MOM6. While the MOM6 use of the ALE remapping makes an Adams-Bashforth scheme impractical, there may be a better timestepping scheme out there for MOM6. Please let the MOM6 developers know if you would like to work on this problem.

4.3 Barotropic-Baroclinic Coupling

Time-averaged accelerations

The barotropic equations are timestepped with a timestep to resolve the surface gravity waves. With care, the baroclinic timestep only need resolve the inertial oscillations. The barotropic accelerations are averaged over the many barotropic timesteps taken between baroclinic steps. At time n , the baroclinic accelerations are computed. The vertical average of that acceleration is subtracted off and replaced by the time-averaged acceleration from the group of barotropic

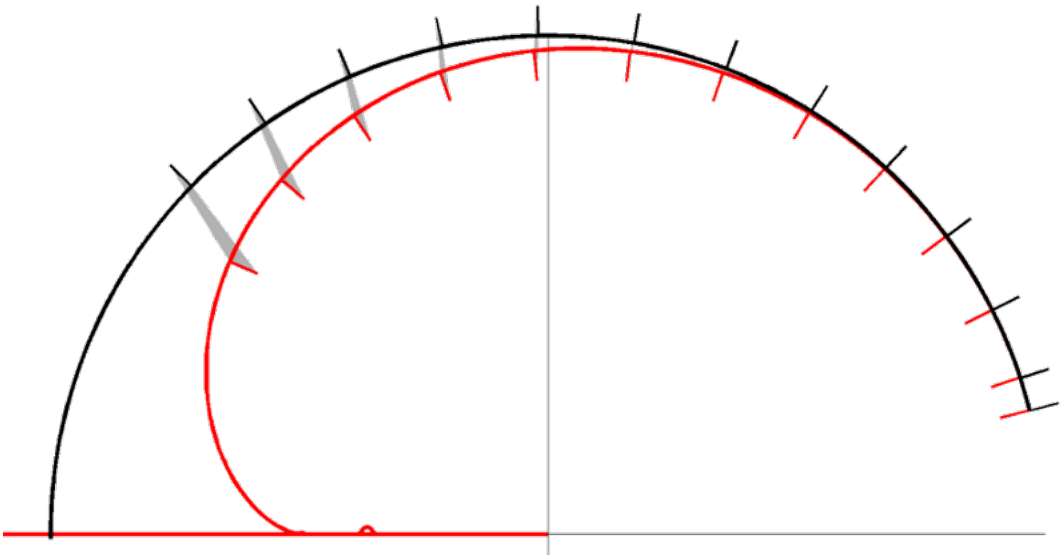

Fig. 2: Graphical notation for the Adams-Bashforth technique used in the ROMS model.

timesteps:

$$\Delta t \frac{\partial \vec{u}}{\partial t} = \Delta t \left(\frac{\partial \vec{u}}{\partial t} - \frac{\partial \vec{u}_{BT}}{\partial t} \right)^n + \Delta t \overline{\frac{\partial \vec{u}_{BT}}{\partial t}}^{\Delta t}$$

Similarly, the velocities used in the tracer equation are a careful blend of the barotropic and baroclinic solutions:

$$\Delta t \frac{\partial \theta}{\partial t} + \Delta t \left(\tilde{\vec{u}} \cdot \nabla \theta + \tilde{w} \frac{\partial \theta}{\partial z} \right)$$

with

$$\tilde{\vec{u}} = \vec{u}_{BC} + \overline{\vec{u}_{BT}}^{\Delta t}$$

$$\frac{\partial \tilde{w}}{\partial z} = -\nabla \cdot \tilde{\vec{u}}$$

4.3.1 Two estimates of the free surface height

The vertically discrete, temporally continuous layer continuity equations are:

$$\frac{\partial h_k}{\partial t} = -\nabla \cdot (\vec{u} h_k) = -\nabla \cdot \mathbf{F}(u_k, h_k)$$

The relationship between the free surface height and the layer thicknesses h_k is:

$$\eta = \sum_{k=1}^{N} h_k - D$$

We get an evolution equation for the free surface height:

$$\frac{\partial \eta}{\partial t} = \sum_{k=1}^{N} \frac{\partial h_k}{\partial t} = -\nabla \cdot \sum_{k=1}^{N} \mathbf{F}(u_k, h_k)$$

If the algorithms for the fluxes in the continuity equations are *linear* in the velocity, the free surface height can be rewritten as:

$$\frac{\partial \eta}{\partial t} \& = -\nabla \cdot \sum_{k=1}^{N} \mathbf{F}(u_k, h_k) = -\nabla \cdot \sum_{k=1}^{N} (\vec{u}_k h_k)$$
$$\& = -\nabla \cdot \left[\sum_{k=1}^{N} h_k \frac{\sum_{k=1}^{N} (\vec{u}_k h_k)}{\sum_{k=1}^{M} k_k} \right] \equiv -\nabla \cdot H\mathbf{U}$$

where

$$\mathbf{U} \equiv \frac{\sum_{k=1}^{N} (\vec{u}_k h_k)}{\sum_{k=1}^{M} k_k}$$

$$H \equiv \sum_{k=1}^{N} h_k$$

However, ALE models like MOM6 require positive-definite, nonlinear continuity solvers (MOM6 uses *PPM Advection Scheme*). We need a different way to reconcile this estimate of free surface height with the one coming from the barotropic equations. After rejecting several other options, MOM6 is now using the scheme from [23] . The barotropic update of η is given by:

$$\frac{\eta^{n+1} - \eta^n}{\Delta t} + \nabla \cdot \left(\overline{UH} \right) = 0$$

Determine the ΔU such that:

$$\sum_{k=1}^{N} \mathbf{F}(\tilde{u}_k, h_k) = \left(\overline{UH} \right)$$

where

$$\tilde{u}_k = u_k + \Delta U$$

The layer timestep equation becomes:

$$h_k^{n+1} = h_k^n - \Delta t \nabla \cdot \mathbf{F}(\tilde{u}_k, h_k)$$

This scheme has these properties:

- Total mass is conserved layer-wise.
- Layer equations retain their flux form.
- Total salt, heat, and other tracers are exactly conserved.
- Free surface heights exactly agree.
- Requires (very few) completely local iterations.
- The velocity corrections are barotropic, and more likely to correspond to the layers whose flow was deficient than in some older schemes.

The solution is unique provided that:

$$\frac{\partial}{\partial \tilde{u}_k} \mathbf{F}(\tilde{u}_k, h_k) > 0$$

This is a reasonable requirement for any discretization of the continuity equation. In the continuous limit, $\mathbf{F}(u, h) = uh$, so one interpretation is:

$$\frac{\partial}{\partial \tilde{u}_k} \mathbf{F}(\tilde{u}_k, h_k) = h_{k,\text{Marginal}}$$

With the PPM continuity scheme:

$$F_{i+1/2} = \frac{1}{\Delta t} \int_{x_{i+1/2}-u\Delta t}^{x_{i+1/2}} h_i^n(x) dx$$

leads to:

$$\frac{\partial F_{i+1/2}}{\partial u_{i+1/2}} = h_i^n(x_{i+1/2} - u_{i+1/2}\Delta t) \equiv h_{k,\text{Marginal}}$$

$h_i(x) > 0$ is already required for positive definiteness.

Newton's method iterations quickly give ΔU :

$$\Delta U^{m+1} = \Delta U^m + \frac{(\overline{UH}) - \sum_{k=1}^{N} F(u_k + \Delta U^m, h_k)}{\sum_{k=1}^{N} h_{k,\text{Marginal}}}$$

How practical is this iterative approach?

The piecewise parabolic method continuity solver uses two steps:

- Set up the positive-definite subgridscale profiles, $h_{PPM}(x)$.

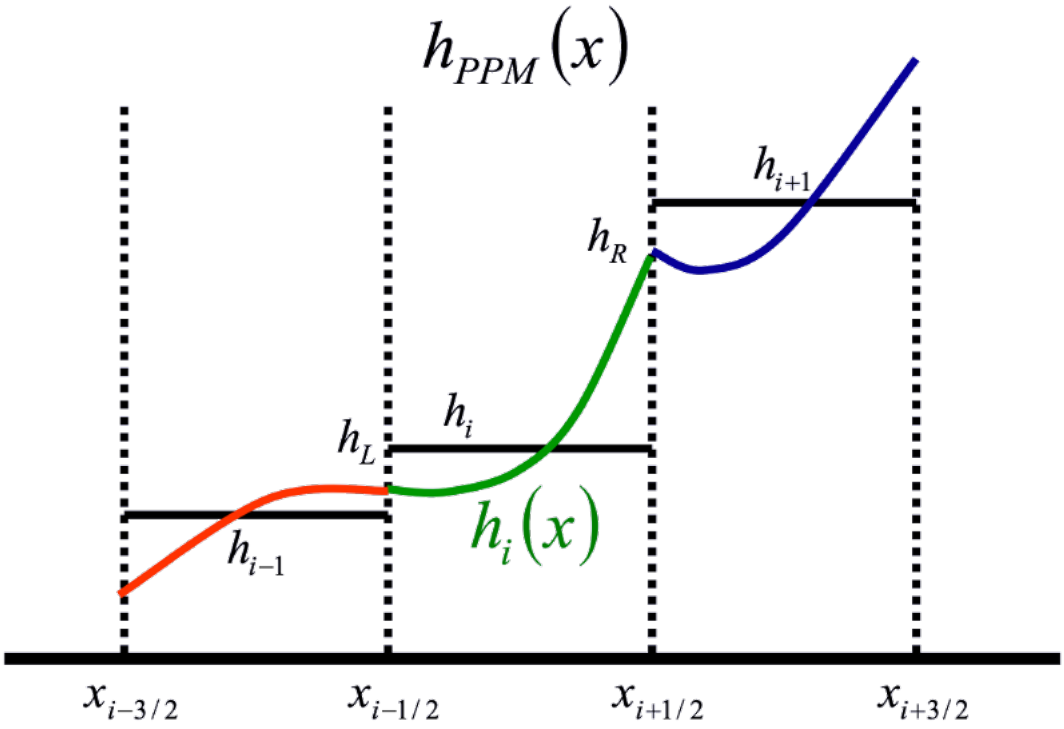

Fig. 3: Piecewise parabolic reconstructions of $h(x)$.

- Integrating the profiles to determine F . Step 1 is typically ~ 3 times as expensive as step 2. $F(u, h)$ is piecewise cubic in u , but often nearly linear. Newton's method iterations converge quickly:

In a global model the sea surface heights converge everywhere to a tolerance of 10^{-6} m within five iterations. These five iterations add ~ 1.6 times more CPU time to the PPM continuity solver and the continuity solver is just 12% of the total model time.

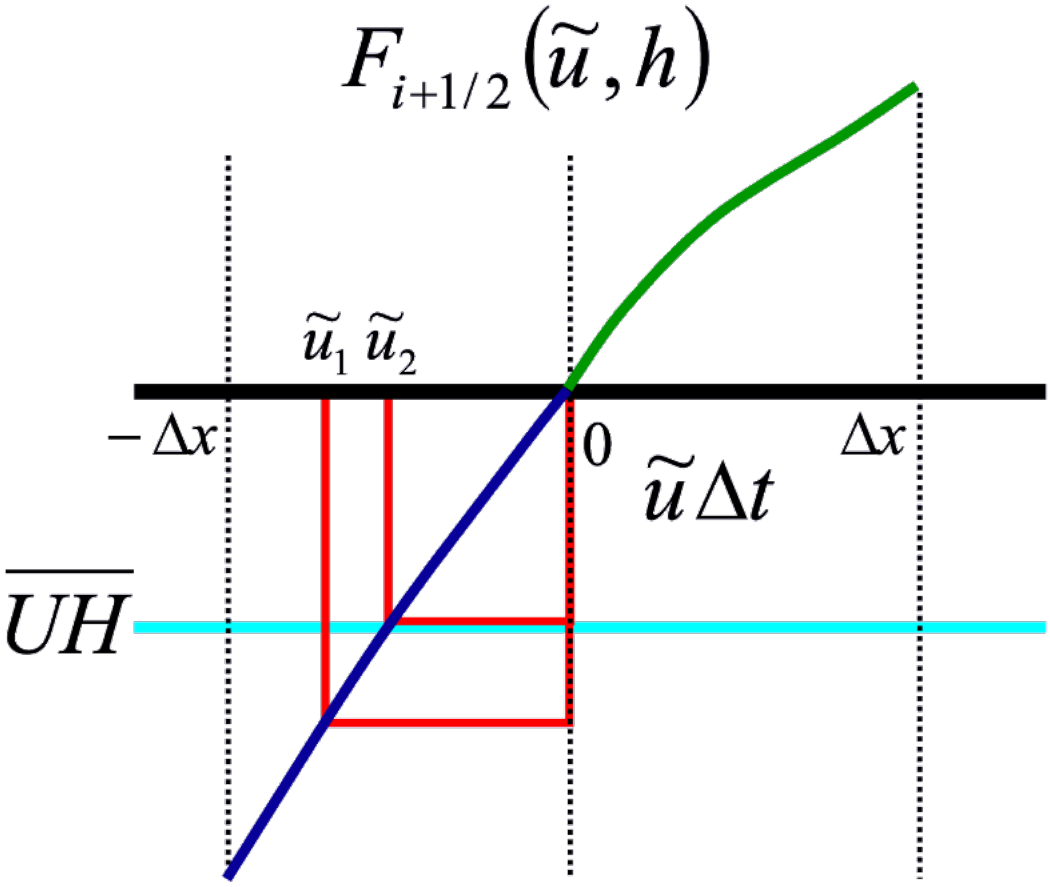

Fig. 4: Newton's method iterations for finding $ U$.

A note on bottom drag

Barotropic accelerations do not lead to barotropic flows after a timestep when vertical viscosity is taken into account.

$$u_k^{n+1} = u_k^n + \Delta t A_k + \Delta t \frac{\tau_{k-1/2} - \tau_{k+1/2}}{h_k}$$

$$\tau_{1/2} = \tau_{Wind}$$
$$\tau_{k+1/2} = \nu_{k+1/2} \frac{u_k^{n+1} - u_{k+1}^{n+1}}{h_{k+1/2}}$$
$$\tau_{N+1/2} = \nu_{N+1/2} \frac{2u_N^{n+1}}{h_{N+1/2}}$$
$$\gamma_k \equiv \frac{1}{\Delta t} \frac{\partial u_k^{n+1}}{\partial \overline{A}}$$

A tridiagonal equation for γ_k results, going from 0 for thin layers near the bottom to 1 far above the bottom.

$$\gamma_1 = 1 + \frac{1}{h_1}\left[-\frac{\nu_{3/2}\Delta t}{h_{3/2}}(\gamma_1 - \gamma_2)\right]$$
$$\gamma_k = 1 + \frac{1}{h_k}\left[\frac{\nu_{k-1/2}\Delta t}{h_{k-1/2}}(\gamma_{k-1} - \gamma_k) - \frac{\nu_{k+1/2}\Delta t}{h_{k+1/2}}(\gamma_k - \gamma_{k+1})\right]$$
$$\gamma_N = 1 + \frac{1}{h_N}\left[\frac{\nu_{N-1/2}\Delta t}{h_{N-1/2}}(\gamma_{N-1} - \gamma_N) - \frac{2\nu_{N+1/2}\Delta t}{h_{N+1/2}}\gamma_N\right]$$

In the continuous limit:

$$\gamma(z) = 1 + \Delta t \frac{d}{dz}\left(\nu \frac{d\gamma}{dz}\right)$$

with boundary conditions:

$$\frac{d\gamma}{dz}(0) = 0$$
$$\gamma(-D) = 0$$

For deep water and contant ν , we get:

$$\gamma(z) = 1 - \exp\left(-\sqrt{\nu \Delta t}(z + D)\right)$$

We want a scheme in which the split model looks exactly like the unsplit model would if we had taken all those short 3D timesteps. Rather than applying a barotropic velocity, we use a barotropic acceleration and allow the continuity solver to determine the tranports consistent with a no-slip bottom boundary layer and perhaps also a no-slip surface boundary layer under an ice shelf.

From above, the barotropic equation is:

$$\frac{\eta^{n+1} - \eta^n}{\Delta t} + \nabla \cdot \left(\overline{UH}\right) = 0$$

We need to determine the $\Delta\overline{A}$ such that:

$$\sum_{k=1}^{N} \mathbf{F}(\tilde{u}_k, h_k) = \left(\overline{UH}\right)$$

where

$$\tilde{u}_k = u_k + \gamma_k \Delta\overline{A} \Delta t$$

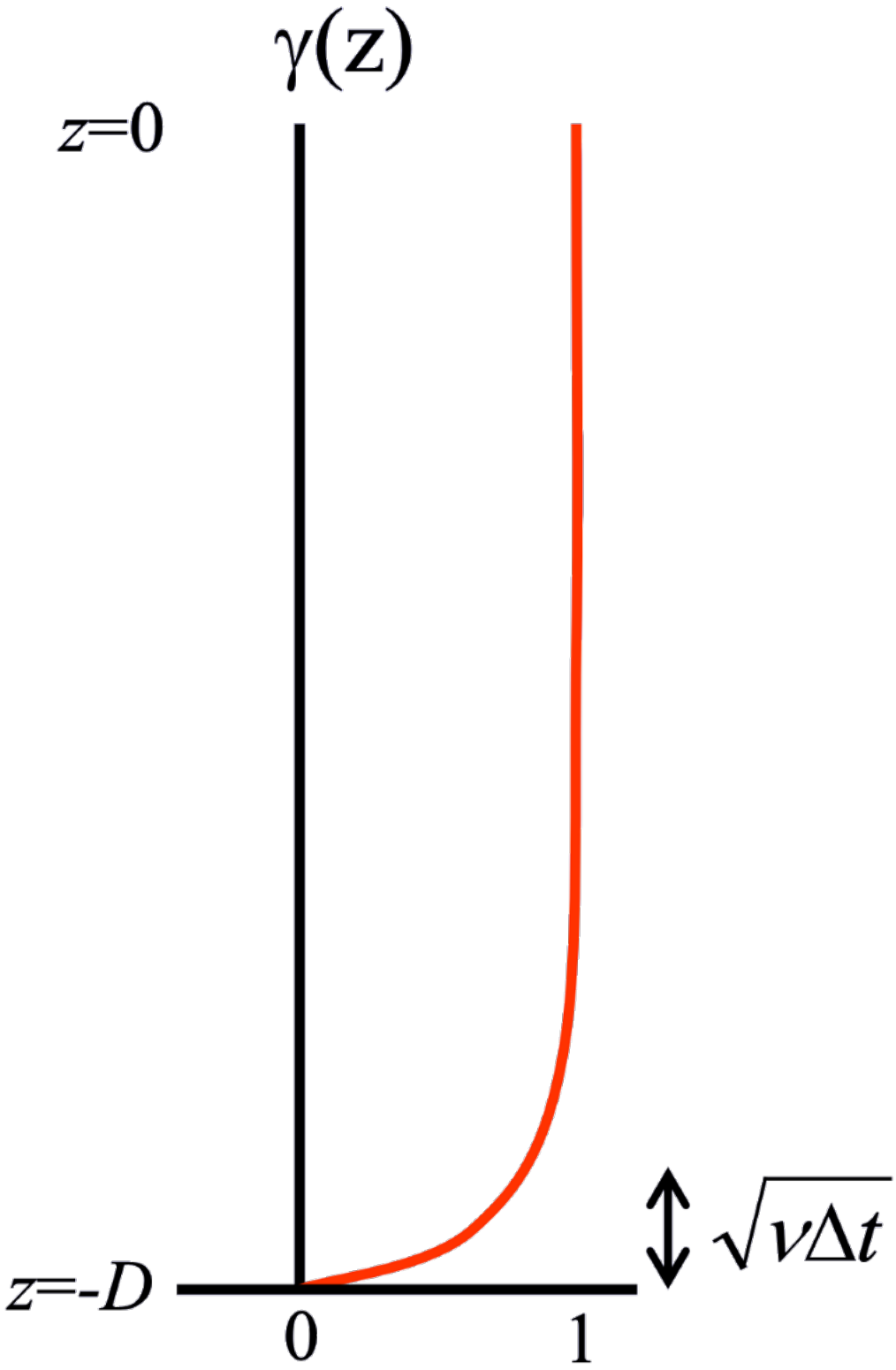

Fig. 5: The continuous solution for barotropic flow plus a no-slip condition at the bottom.

4.3.2 Additional details about the split time stepping

- Transports are used as input and output to the barotropic solver. The continuity solver is inverted to determine velocities.

$$\frac{\partial \eta}{\partial t} = \nabla \cdot \overline{U} + M$$

$$\overline{U}(\overline{u}) = \frac{1}{\Delta t} \int_0^{\overline{u}\Delta t} H(x) dx$$

$$\overline{u}^n = \overline{U}^{-1} \left(\sum_{k=1}^{N} U_k^n \right)$$

$$u_k^{n+1} = \tilde{u}_k^{n+1} + \Delta \overline{u}$$

We need to find $\Delta \overline{u}$ such that:

$$\sum_{k=1}^{N} U_k \left(\tilde{u}_k^{n+1} + \Delta \overline{u} \right) = \overline{U}^{n+1}$$

- Barotropic accelerations are treates as anomalies from the baroclinic state:

$$\frac{\partial \overline{u}}{\partial t} \& = -f \hat{k} \times (\overline{u} - \overline{u}_{Cor}) - \nabla \overline{g} (\eta - \eta_{PF}) -$$

$$\& \frac{c_D \left(||u_{Bot}|| + ||u_{Shelf}|| \right)}{\sum_{k=1}^{N} h_k} (\overline{u} - \overline{u}_{Drag}) + \frac{\sum_{k=1}^{N} h_k \frac{\partial u_k}{\partial t}}{\sum_{k=1}^{N} h_k}$$

- Bottom drag (and under ice surface-drag) is treated implicitly.
- The barotropic continuity solver uses flow-dependent thickness fits which approximate the sum of the layer thicness transports, to accommodate wetting and drying. An image of this is shown here:

4.3.3 Summary of MOM6 split time stepping

- We use an efficient approach for handling fast modes via simplified 2-d equations, while the 3-d baroclinic timestep is determined by baroclinic dynamics.
- The barotropic solver determines free surface height and time-averaged depth-integrated transports.
- By using anomalies, the MOM6 split solver gives identical answers to an equivalent unsplit scheme for short timesteps.
- This scheme has been demonstrated to work with wetting and drying, as well as under ice-shelves.
- The linear barotropic solver allows MOM6 to automatically set a stable barotropic timestep (e.g. to 98% of maximum).

4.4 Tracer Timestep

Overview of Tracer Timestep

The MOM6 code handles advection and lateral diffusion of all tracers. For potential temperature and salinity, it also timesteps the thermodynamics and vertical mixing (column physics). Since evaporation and precipitation are handled as volume changes, the layer thicknesses need to be updated:

$$\frac{\partial h_k}{\partial t} = (P - E)_k$$

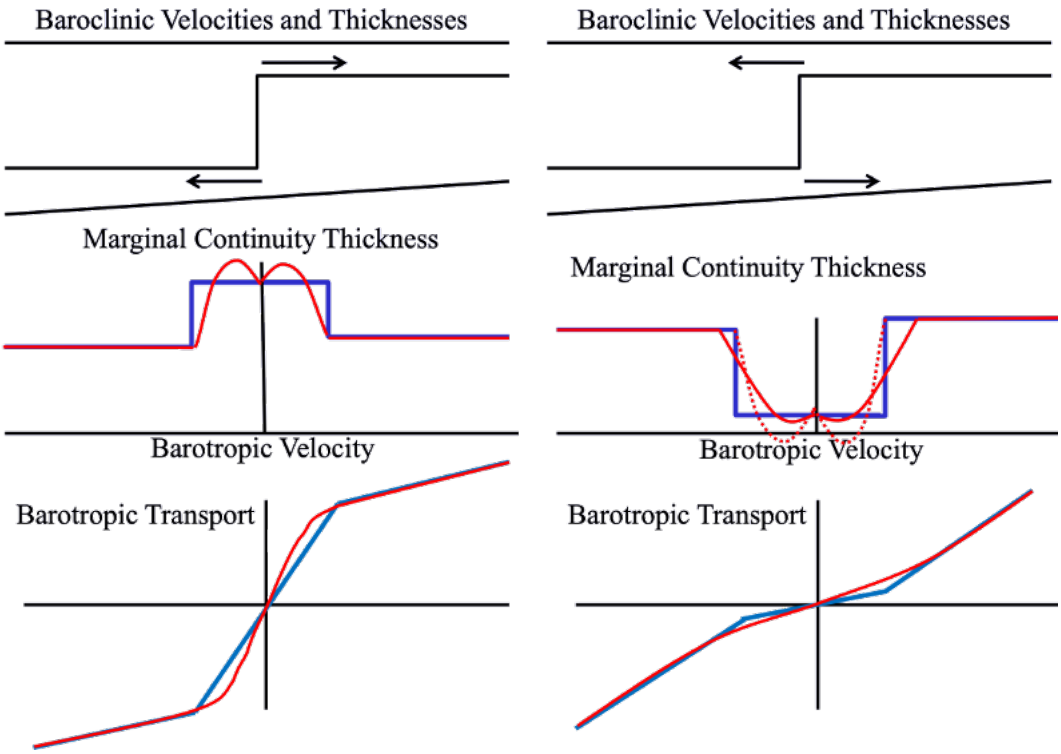

Fig. 6: The barotropic transports depend on the baroclinic flows and thicknesses.

The full tracer equation for tracer θ is:

$$\frac{\partial}{\partial t}(h_k\theta_k) + \nabla_s \cdot (\vec{u}h_k\theta_k) = Q_k^\theta h_k + \frac{1}{h_k}\Delta\left(\kappa\frac{\partial\theta}{\partial z}\right) + \frac{1}{h_k}\nabla_s(h_k K \nabla_s\theta)$$

Here, the advection is on the left hand side of the equation while the right hand side contains thermodynamic processes, vertical diffusion, and horizontal diffusion. There is more than one choice for vertical diffusion; these will be described elsewhere. Also, the lateral diffusion is handled in novel ways so as to avoid introduction of new extrema and to avoid instabilities associated with rotated mixing tensors. The lateral diffusion is described in *Horizontal Diffusion* .

4.5 ALE Timestep

Explanation of ALE remapping

The Arbitrary Lagrangian-Eulerian (ALE) remapping is not a timestep in the traditional sense, but rather an operation performed to bring the vertical coordinate back to the target specification. This remapping can be less frequent than the momentum or thermodynamic timesteps, but must be done before the layer interfaces become entangled with each other.

Assuming the target vertical grid is level z -surfaces, the initial state is shown on the left in the following figure:

Some time later, a wave has perturbed the surfaces which move with the fluid and it has been determined that a remapping operation is needed. The target vertical grid is still level z -surfaces, so this new target grid is shown overlaid on the left as regrid:

The complex part of the operation is remapping the wavy field onto the new grid as shown on the right and again in the final frame after the old deformed coordinate system has been deleted:

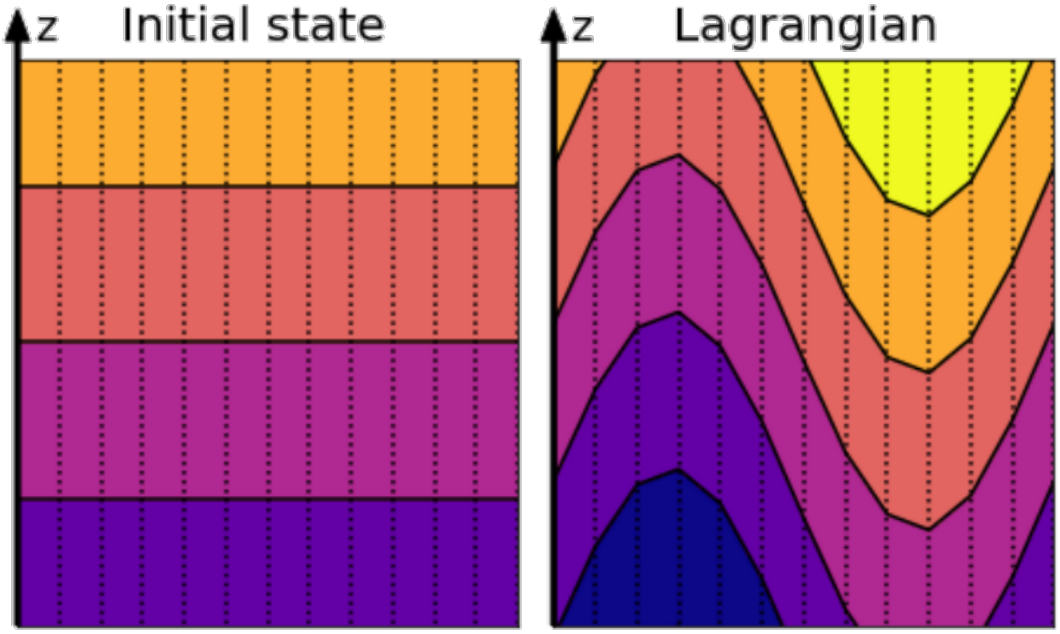

Fig. 7: The initial state with level surface (left) and the perturbed state after a wave has come through (right).

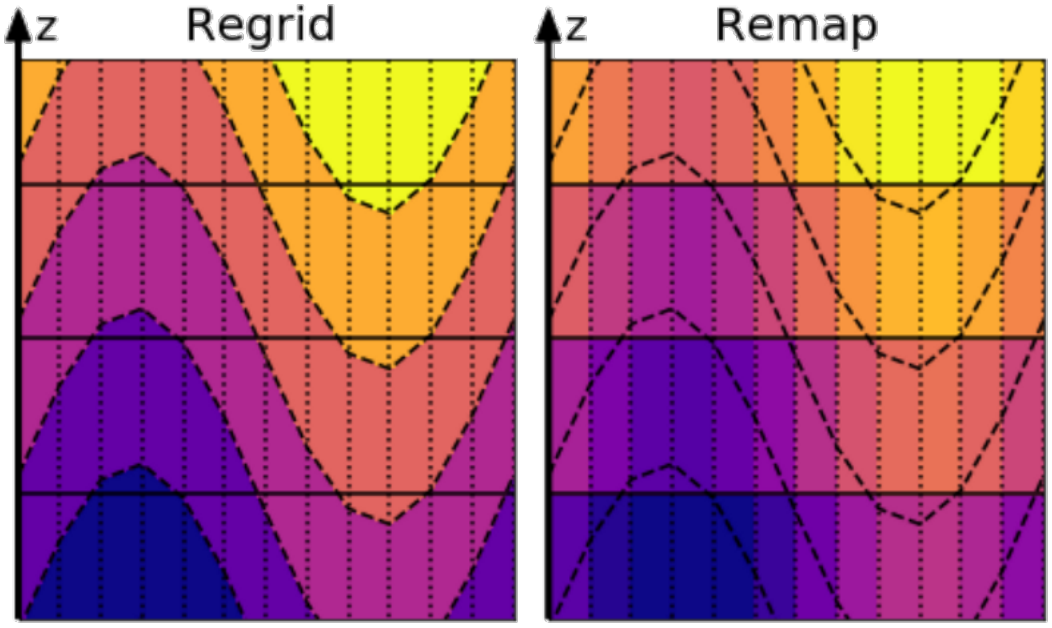

Fig. 8: The regrid operation (left) and the remap operation (right).

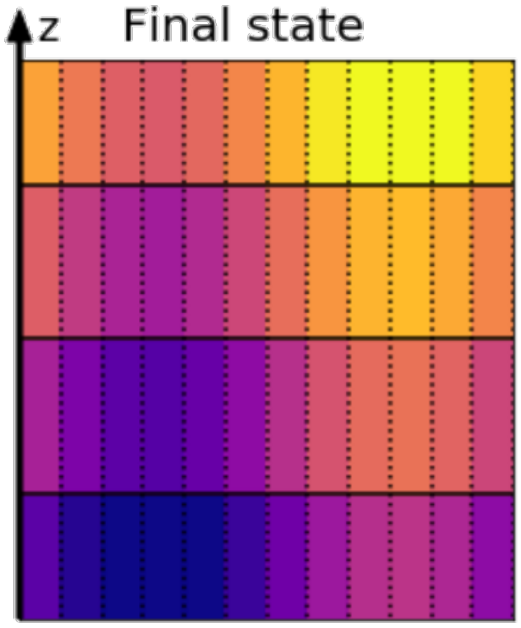

Fig. 9: The final state after remapping.

Mathematically, the new layer thicknesses, h_k , are computed and then populated with the new velocities and tracers:

$$h_k^{\text{new}} = \nabla_k z_{\text{coord}}$$

$$\sum h_k^{\text{new}} = \sum h_k^{\text{old}}$$

$$\vec{u}_k^{\text{new}} = \frac{1}{h_k} \int_{z_{k+\frac{1}{2}}}^{z_{k+\frac{1}{2}}+h_k} \vec{u}^{\text{old}}(z')dz'$$

$$\theta^{\text{new}} = \frac{1}{h_k} \int_{z_{k+\frac{1}{2}}}^{z_{k+\frac{1}{2}}+h_k} \theta^{\text{old}}(z')dz'$$

CHAPTER

FIVE

TRACERS IN MOM6

5.1 Tracer Advection

Tracer transport schemes

MOM6 implements a generalised tracer advection scheme, which is a combination of the modified flux advection scheme [12] with reconstructed tracer distributions. The tracer distributions may be piecewise linear (PLM) or piecewise parabolic (PPM), which may itself use either the [9] (CW84) or [25] (H3) reconstruction.

5.1.1 Flux advection

The modified flux advection scheme preserves the tracer mixing ratio in a cell across directional splitting by accounting for changes in mass changes. Fluxes are applied to alternate directions in turn, restricting the applied flux so as not to evacuate all mass out of a cell. Because of this, we need to know the stencil used during the calculation of the reconstruction. Every iteration of the splitting algorithm, cells at the edge of a processor's data domain are invalidated. When this invalidation region extends below the halo, a group pass is required to refresh the halo. A larger stencil (such as for the CW84 reconstruction) therefore introduces more frequent updates, and may impact performance.

5.1.2 Tracer reconstruction

While MOM6 only carries the mean tracer concentration in a cell, a higher order reconstruction is computed for the purpose of advection. Reconstructions are also modified to ensure that monotonicity is preserved (i.e. spurious minima or maxima cannot be introduced).

The piecewise linear (PLM) reconstruction uses the monotonic modified van Leer scheme [30] . One might think to use the average of the one-sided differences of mean tracer concentration within a cell to calculate the slope of the linear reconstruction, however this method guarantees neither monotonicity, nor positive definiteness. Instead, the method is locally limited to the minimum of this average slope and each of the one-sided slopes, i.e.

$$\Delta\Phi_i = \min\left\{\left|[\Delta\Phi_i]_{\text{avg}}\right|, 2\left(\Phi_i - \Phi_i^{\text{min}}\right), 2\left(\Phi_i^{\text{max}} - \Phi_i\right)\right\}$$

(where Φ_i^{min} is the minimum in the 3-point stencil).

In a PPM scheme (*PPM Advection Scheme*), for a cell with mean tracer concentration Φ_i , the values at the left and right interfaces, $\Phi_{L,i}$ and $\Phi_{R,i}$ must be estimated. First, an interpolation is used to calculate $\Phi_{i-1/2}$ and $\Phi_{i+1/2}$. These values are then modified to preserve monotonicity in each cell, which introduces discontinuities between cell edges (e.g. $\Phi_{R,i}$ and $\Phi_{L,i+1}$).

The reconstruction $\Phi_i(\xi)$ then satisfies three properties:

- total amount of tracer is conserved, $\int_{\xi_{i-1/2}}^{\xi_{i+1/2}} \Phi_i(\xi')\,\mathrm{d}\xi' = \Phi_i$

- left interface value matches, $\Phi(\xi_{i-1/2}) = \Phi_{L,i}$
- right interface value matches, $\Phi(\xi_{i+1/2}) = \Phi_{R,i}$

There are two methods of reconstruction for a piecewise parabolic (PPM) profile. They differ in the estimate of interface values $\Phi_{i+1/2}$ prior to monotonicity limiting. The CW84 scheme makes use of the limited slope $\Delta\Phi_i$ from PLM, above. This has the effect of requiring a larger stencil for each reconstruction. On the other hand, the H3 scheme reduces the requirement of this stencil, by only examining the tracer concentrations in adjacent cells, at the same time reducing order of accuracy of the reconstruction.

5.2 Tracer Transport Equations

Tracer Transport Equations

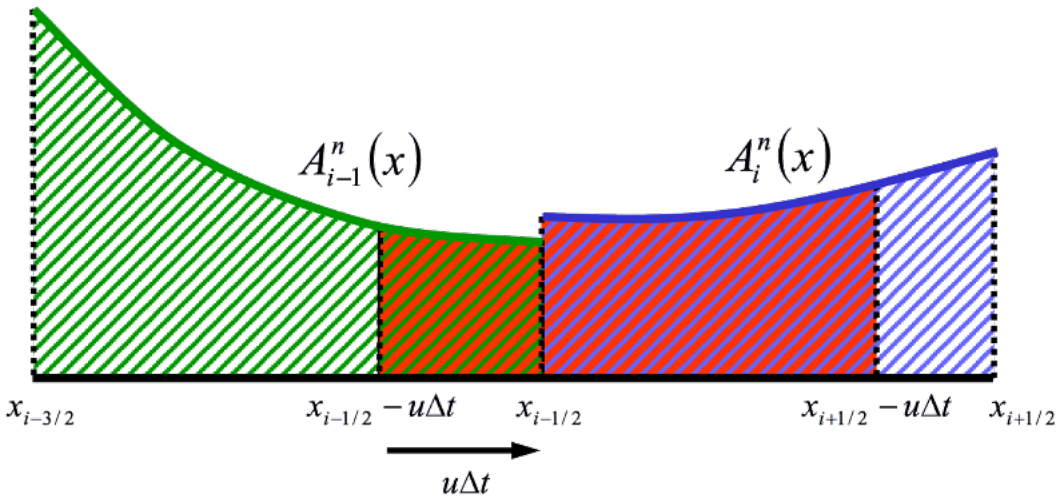

Fig. 1: The 1-D finite volume advection of tracers. The reddish fluid will be in the cell at the end of the timestep.

Given a piecewise polynomial description of the tracer concentration, the new tracer cell concentration is the average of the fluid that will be in the cell after a timestep.

$$\int_{x_{i-1/2}}^{x_{i+1/2}} A_i^{n+1}(x)dx = \int_{x_{i-1/2-u\Delta t}}^{x_{i+1/2-u\Delta t}} A_i^n(x)dx \qquad = \tag{5.1}$$

$$\int_{x_{i-1/2}}^{x_{i+1/2}} A_i^n(x)dx - \int_{x_{i+1/2-u\Delta t}}^{x_{i+1/2}} A_i^n(x)dx \quad + \int_{x_{i-1/2-u\Delta t}}^{x_{i-1/2}} A_i^n(x)dx \tag{5.2}$$

Fluxes are found by analytically integrating the profile over the distance that is swept past the face within a timestep.

$$a_i^n = \frac{1}{\Delta x}\int_{x_{i-1/2}}^{x_{i+1/2}} A_i^n(x)dx$$

$$a_i^{n+1} = a_i^n - \frac{\Delta t}{\Delta x}(F_{i+1/2} - F_{i-1/2})$$

$$F_{i+1/2} = \frac{1}{\Delta t}\int_{x_{i+1/2-u\Delta t}}^{x_{i+1/2}} A_i^n(x)dx$$

$$F_{i-1/2} = \frac{1}{\Delta t}\int_{x_{i-1/2-u\Delta t}}^{x_{i-1/2}} A_i^n(x)dx$$

With piecewise constant profiles, this approach give first order upwind advection. Higher order polynomials (e.g., parabolas) can give higher order accuracy.

5.2.1 Multidimensional Tracer Advection

Using "Easter's Pseudo-compressibility" (easter1993), we start with these basic equations for a tracer ψ :

$$\frac{\partial h}{\partial t} + \vec{\nabla} \cdot (\vec{u}h) = 0 \equiv \frac{\partial h}{\partial t} + \vec{\nabla} \cdot (\vec{U})$$

$$\frac{\partial}{\partial t}(h\psi) + \vec{\nabla} \cdot (\vec{U}\psi) = 0$$

$$\frac{\partial \psi}{\partial t} + \vec{u} \cdot \vec{\nabla}\psi = 0$$

We discretize the first of these equations in space:

$$\frac{\partial h}{\partial t} = \frac{1}{\Delta x}\left(U_{i-\frac{1}{2},j} - U_{i+\frac{1}{2},j}\right) + \frac{1}{\Delta y}\left(V_{i,j-\frac{1}{2}} - V_{i,j+\frac{1}{2}}\right)$$

Using our monotonic one-dimensional flux:

$$F_{i+\frac{1}{2},j}(\psi) = U_{i+\frac{1}{2},j}\psi_{i+\frac{1}{2},j}$$

we come up with an estimate based only on an update in the x direction:

$$\tilde{h}_{i,j}\tilde{\psi}_{i,j} = h^n_{i,j}\psi_{i,j} + \frac{\Delta t}{\Delta x}\left(F_{i-\frac{1}{2},j}(\psi^n) - F_{i+\frac{1}{2},j}(\psi^n)\right)$$

$$\tilde{h}_{i,j} = h^n_{i,j} + \frac{\Delta t}{\Delta x}\left(U_{i-\frac{1}{2},j} - U_{i+\frac{1}{2},j}\right)$$

$$\tilde{\psi}_{i,j} = \frac{\tilde{h}_{i,j}\tilde{\psi}_{i,j}}{\tilde{h}_{i,j}}$$

Next, we update in the y direction:

$$h^{n+1}_{i,j}\psi^{n+1}_{i,j} = \tilde{h}_{i,j}\tilde{\psi}_{i,j} + \frac{\Delta t}{\Delta y}\left(G_{i,j-\frac{1}{2}}(\tilde{\psi}) - G_{i,j+\frac{1}{2}}(\tilde{\psi})\right)$$

$$h^{n+1}_{i,j} = \tilde{h}_{i,j} + \frac{\Delta t}{\Delta y}\left(V_{i,j-\frac{1}{2}} - V_{i,j+\frac{1}{2}}\right)$$

$$\psi^{n+1}_{i,j} = \frac{h^{n+1}_{i,j}\psi^{n+1}_{i,j}}{h^{n+1}_{i,j}}$$

- This method ensures monotonicity. Strang splitting can reduce directional splitting error. See [12] , [11] (section 5.9.4), and [38] .
- Flux-form pseudo-compressibility advection is based on accumulated mass (or volume) fluxes, not velocities.
- Additional pseudo-compressibility passes can be added to accommodate transports exceeding cell masses. Extra passes of tracer advection are used in MOM6 in the small fraction of cells where this is needed.
- Explicit layered dynamics time-steps are limited by Doppler-shifted internal gravity wave speeds or inertial oscillations. Flow speeds in most of the ocean volume are much smaller than the peak internal wave speeds so that the advective time-steps can be longer.
- Advective mass fluxes in MOM6 are often accumulated over multiple dynamic steps. The goal is that as we go to higher resolution, this tracer advection will remain stable at relatively long time-steps, allowing for the inclusion of many biogeochemical tracers without adding an undue burden in computational cost.

5.3 Horizontal Diffusion

Horizontal diffusion of tracers

Lateral mixing due to mesoscale eddies is believed to occur according to this figure:

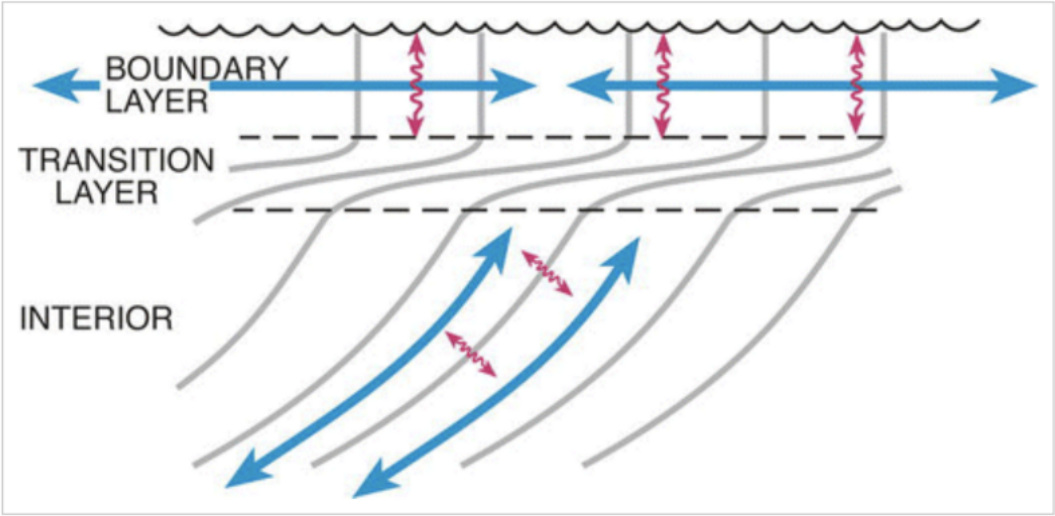

Fig. 2: Horizontal surface boundary layer fluxes and interior epineutral fluxes.

We start by describing an implementation of the mixing in the interior and then introduce a surface mixed layer implementation. A bottom mixed layer implementation is planned for the future.

5.3.1 Epineutral Diffusion

For the interior of the ocean, we would like to have horizontal diffusion with the following properties:

- Suitable for general coordinate models
- Preserves extrema
- Has no need for regularization or tapering (such as needed by rotated mixing tensors)

The algorithm used in MOM6 is described by [40] and will be introduced here. The aim is to allow lateral mixing of tracers within isopycnal layers. It is appropriate for the adiabatic interior of the ocean while a lateral mixing scheme for the surface boundary layer is described below.

Before presenting this scheme, a quick review of polynomial reconstructions is in order. Some choices for the vertical representation of a finite volume quantity are shown here:

Some desired quantities for the polynomial reconstructions to be used are:

- Tracer concentrations represent the cell-averages in vertical discretization.
- Must be monotonic and introduce no new extrema.
- Discontinuous reconstructions are desirable to limit intracell slopes.

The algorithm has three phases: initialization, sorting, and flux calculation.

Fig. 3: Polynomial reconstructions, starting with piecewise constant on the left, piecewise linear in the middle and piecewise parabolic on the right.

Initialization

We begin by generating polynomial reconstructions of the vertical tracer quantities such as shown by the blue lines here:

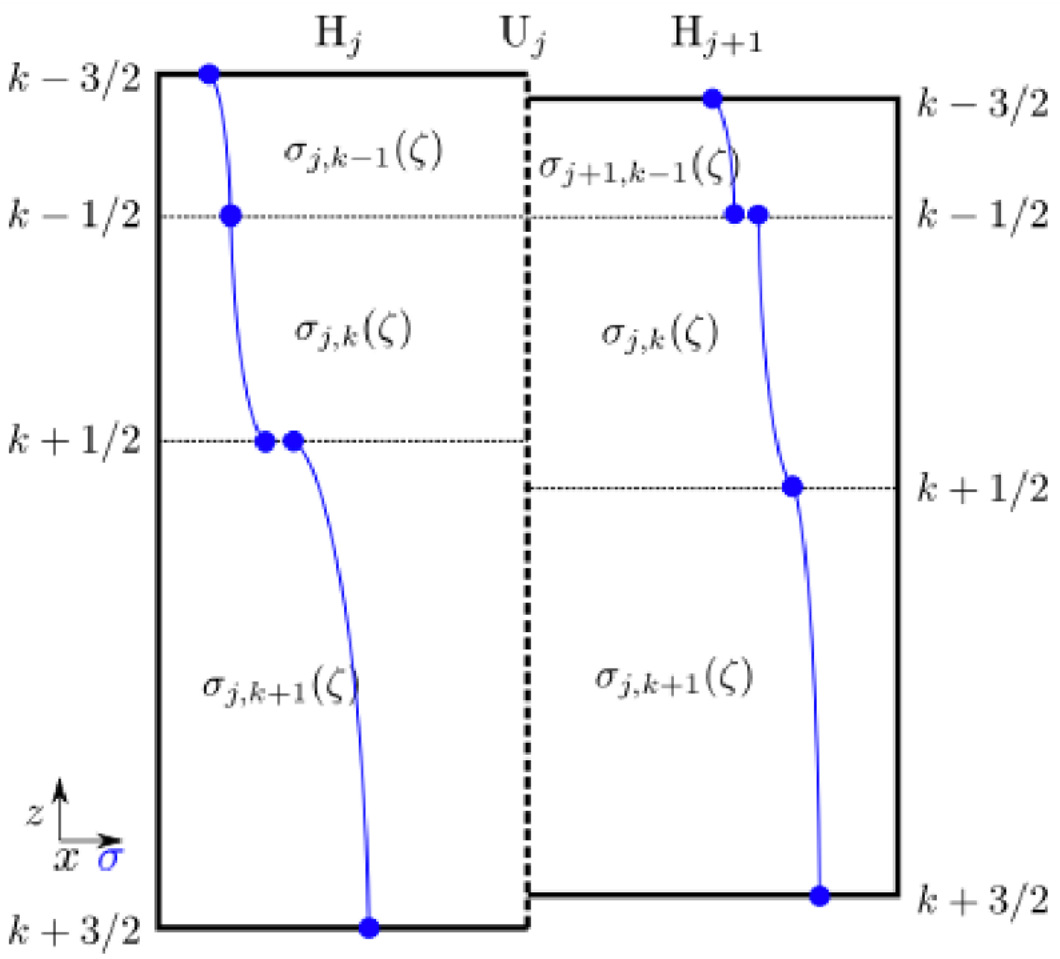

Fig. 4: Polynomial reconstructions of two adjacent water columns.

Because we are looking to mix along epineutral surfaces, we will need to find surfaces of uniform density by using the temperature, salinity, and their effect on the density, α and β . The next step is to find the values of α and β at the interfaces.

Also during the initialization, the unstable parts of the water column are set aside to be skipped by this algorithm.

Sorting

The epineutral surfaces have constant density, where we use this equation:

$$\Delta\rho = \rho_1 - \rho_2 = \frac{\alpha_1 + \alpha_2}{2}(T_1 - T_2) + \frac{\beta_1 + \beta_2}{2}(S_1 - S_2)$$

When calculating α and β , there's more than one way to do it. Using a midpoint pressure gives neutral density while using a reference pressure gives isopycnal values.

Given two adjacent water columns, we are going to be looking to match densities. The match does not need to be at the same level or even near each other in depth. Starting from the top two interfaces, search the column with the lighter surface water (second column) downward to find which layer contains water matching that of the first column at the surface:

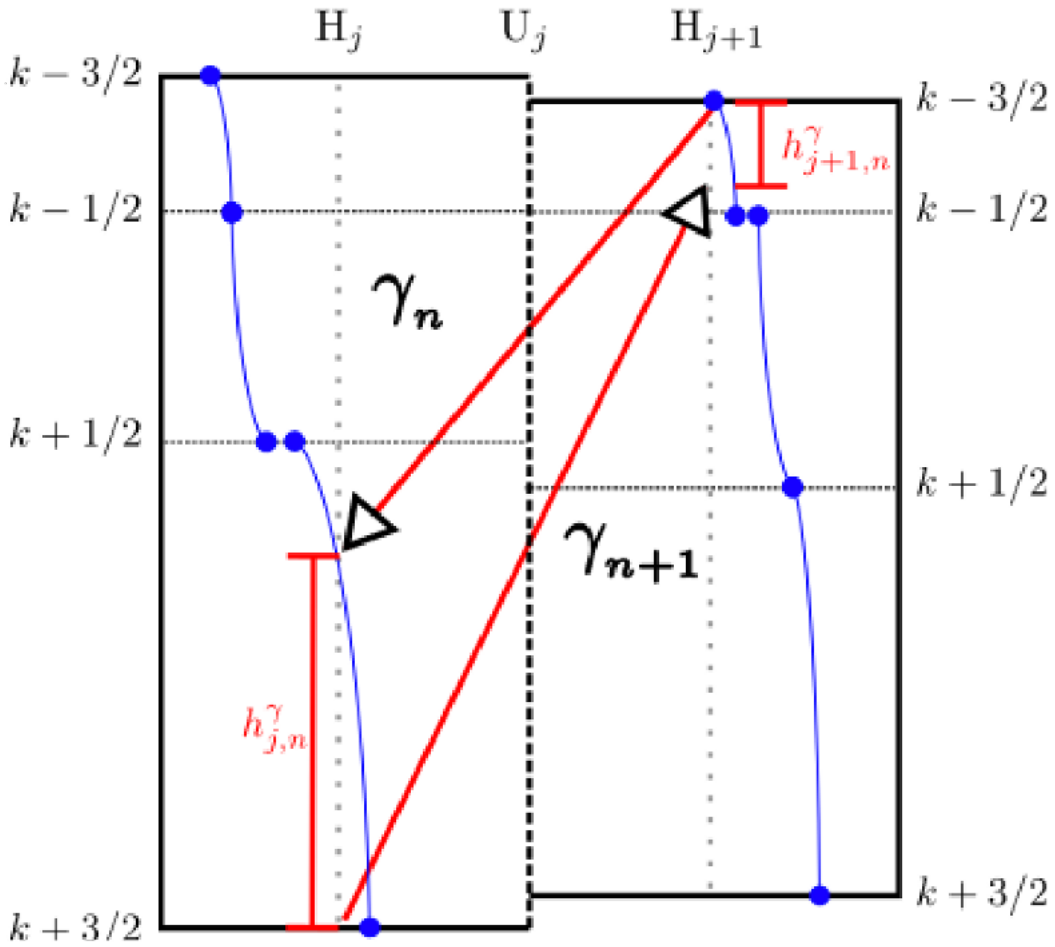

Fig. 5: Searching the column with the lighter surface for the water matching the other column's surface water.

If the surface density matches that of an interface, point to the interface. Otherwise, solve for the matching density along the polynomial reconstruction for that layer. There are again some choices:

- Use Newton's method to find the root with higher order polynomials.
- Assume α and β vary linearly from top to bottom (cubic if T and S are parabolic).
- Equation of state is linear from top to bottom interface (parabolic of T and S are parabolic).
- $\Delta\rho$ is linear in the vertical.

Once the location of the first column's surface density is found in the second column, one goes to the next interface below to find the bottom density of the water to be mixed. Then find that density within the first column. Iterate downward until no more matches are found. These pairs of surfaces make up what is known as a sublayer along which the diffusion can take place.

Flux Calculation

For each sublayer, the fluxes are based on the mean tracer quantities within that sublayer in each column. For a tracer C , compute the vertical average of that tracer within the sublayer to form $\overline{C}$. The flux can then be computed based on:

$$F = Kh_{\text{eff}} \frac{\overline{C_{j,k+1}} - \overline{C_{j+1,k-1}}}{\Delta x} \Delta t$$

where the effective thickness of the sublayer is:

$$h_{\text{eff}} = \frac{2h^{\gamma}_{j,n} h^{\gamma}_{j,n+1}}{h^{\gamma}_{j,n} + h^{\gamma}_{j,n+1}}$$

and as shown in this figure:

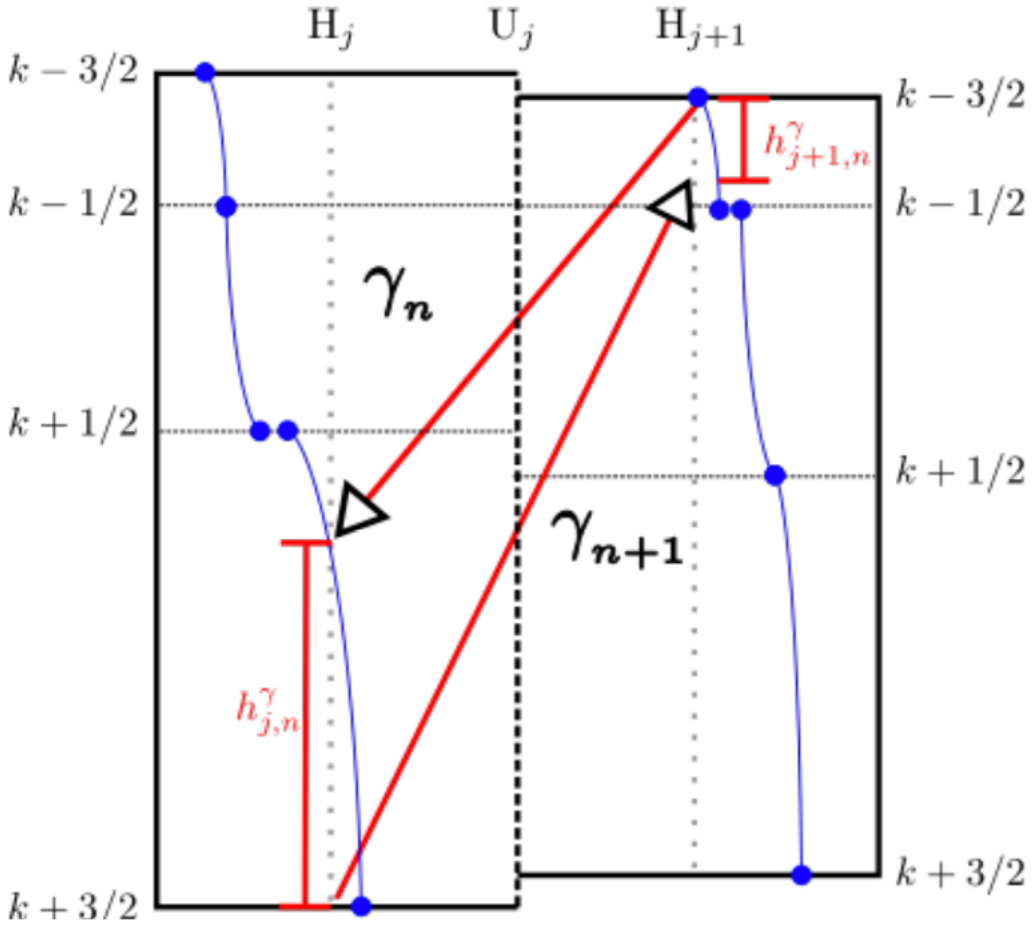

Fig. 6: Diagram of sublayer thickness for the sublayer bounded by surfaces $$ and ${n+1}$.

When updating the tracer state, one needs to accumulate all the fluxes through each face as shown here:

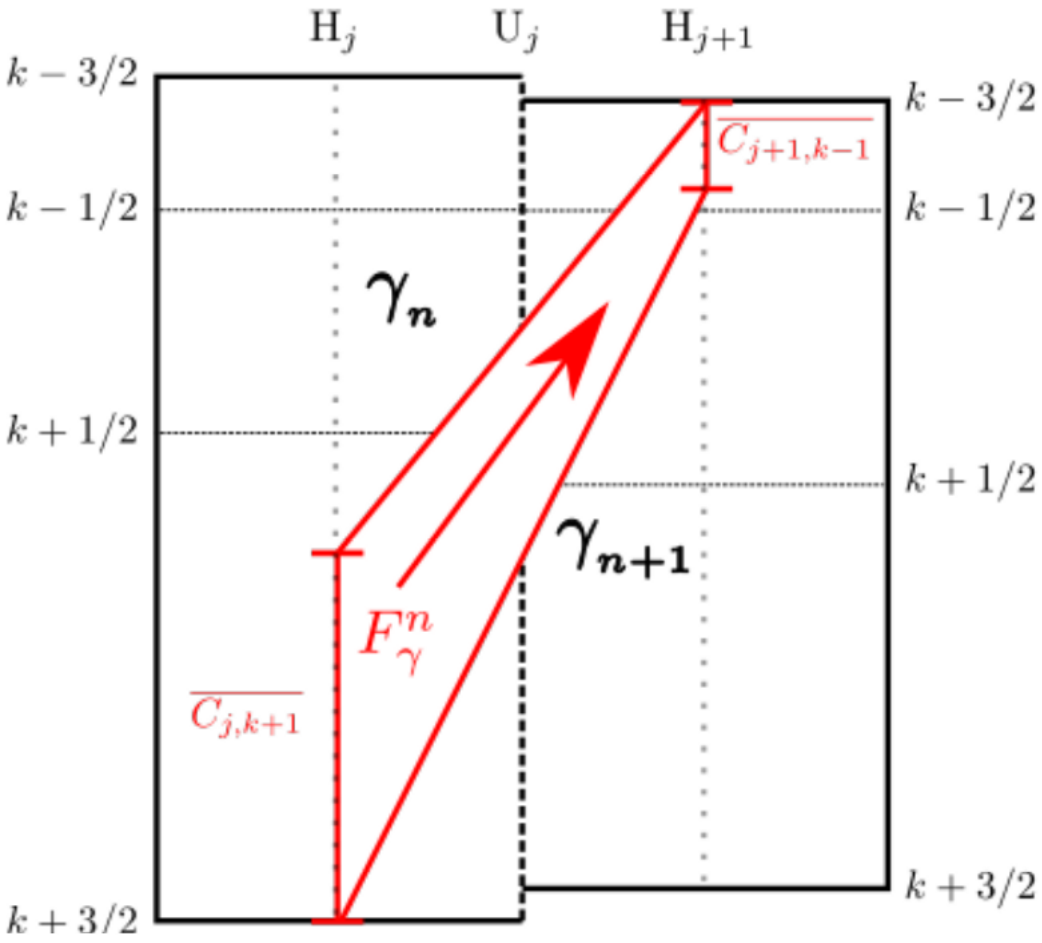

Fig. 7: Flux of tracer C along the sublayer.

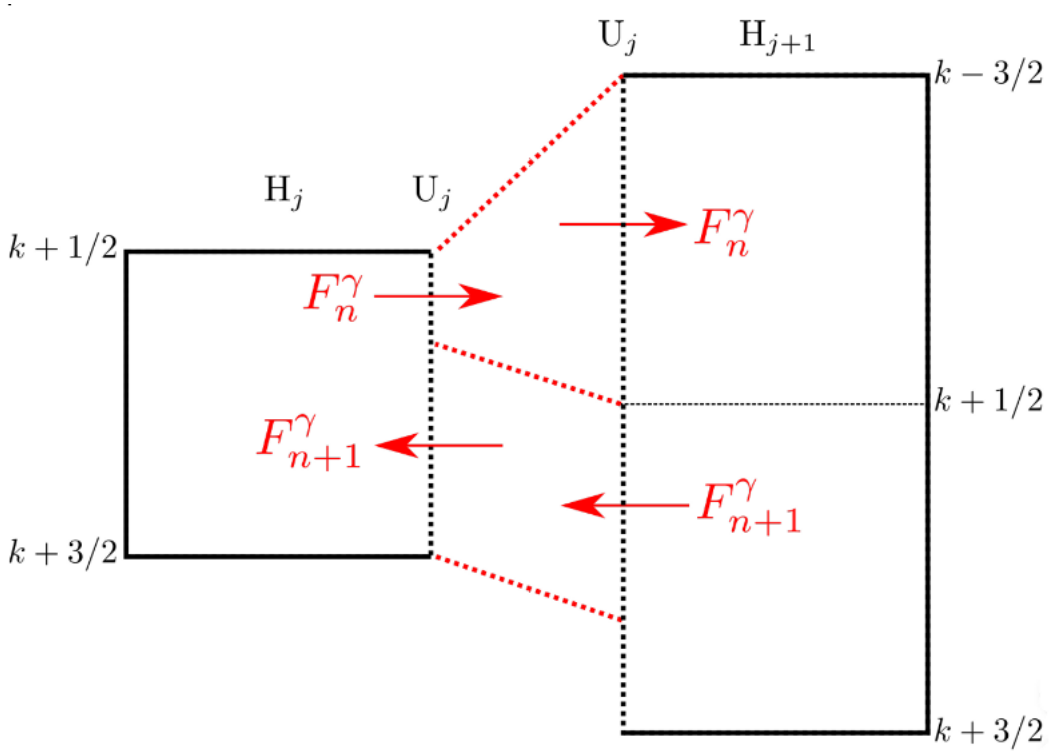

Fig. 8: Accumulate all the fluxes across a face from all the layers in the next column contributing to it.

5.3.2 Surface Diffusion

As shown in figure *Horizontal surface boundary layer fluxes and interior epineutral fluxes.* of the eddy fluxes, the diffusion in the surface boundary layer is assumed to be purely horizontal. A bulk scheme was explored but found wanting, so a layer-by-layer approach has been implemented instead. It is this layer-by-layer code which is described here.

For each water column, the boundary layer thickness is determined first. This can be either via the CVMIX boundary layer thickness or through some other means. Next, determine how many of the model layers are within this boundary layer thickness. It is common for neighboring cells to have differing numbers of layers within the surface boundary layer, such as shown here:

In this case, the cell on the left has four layers within the boundary layer while the cell on the right has just two. The layer-by-layer scheme computes fluxes for the first two layers, then has linearly reduced fluxes for the next two layers below as shown here:

In all cases, the tracer flux is always down-gradient.

$$F(k) = K h_{\text{eff}(k)} \left[\phi_L(k) - \phi_R(k)\right]$$

where the effective thickness of the layer k is:

$$h_{\text{eff}(k)} = \frac{2 h_L(k) h_R(k)}{h_L(k) + h_R(k)}$$

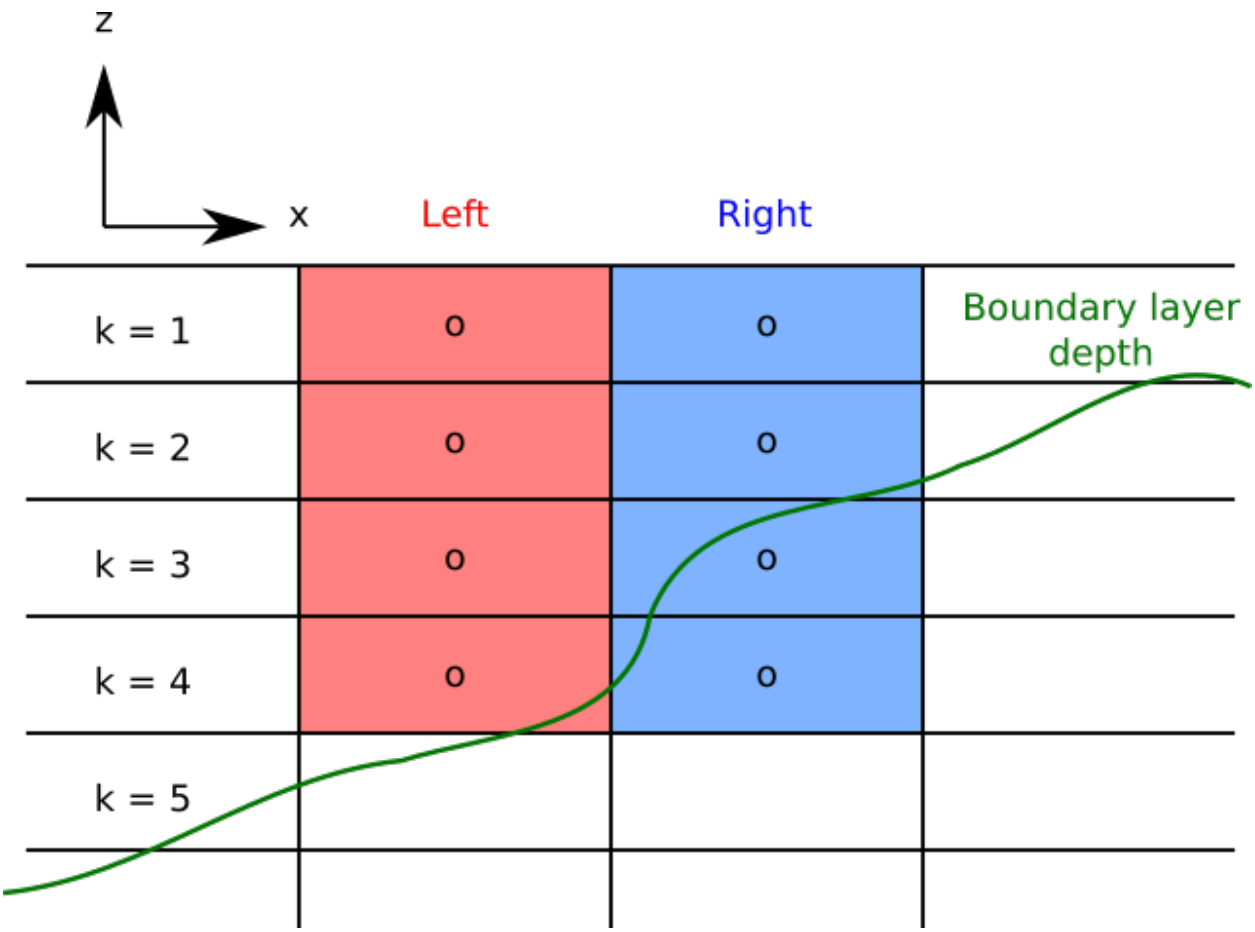

Fig. 9: Two cells within the surface mixed layer, red on the left, blue on the right. The mixed layer depth is shown in green.

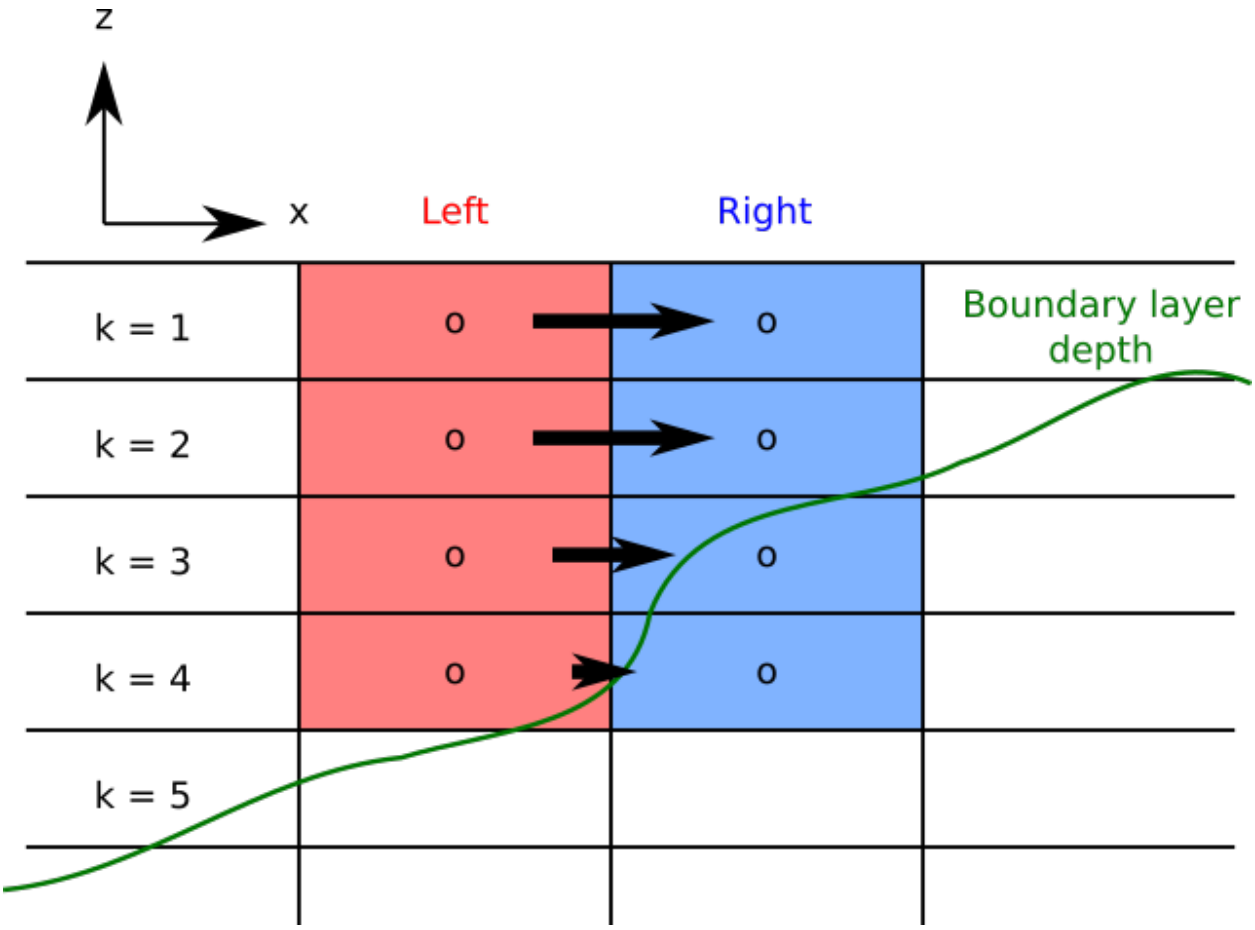

Fig. 10: Two cells within the surface mixed layer with down-gradient fluxes as shown by the black arrows.

5.4 Vertical Diffusion

Vertical diffusion of tracers

The MOM6 tracer registry takes care of the tracer advection as well as horizontal diffusion, but it is up to each individual tracer package to define its own vertical diffusion.

5.5 Passive and Other User-defined Tracers

Tracers beyond temperature and salinity

5.5.1 Passive Tracers

5.5.2 Generic Tracers

5.5.3 User-defined Tracers

CHAPTER

SIX

GRIDS

We love grids.

6.1 Global Orthogonal Grids

Global Orthogonal Grids

6.1.1 Dipole Grids

6.1.2 Tripole Grids

6.2 Regional Orthogonal Grids

Regional Orthogonal Grids

6.2.1 Map Projections

6.2.2 Open Boundary Segments

6.3 Vertical Orthogonal Grids

Vertical Orthogonal Grids

6.3.1 Layered

6.3.2 Z-Star

6.3.3 Sigma

6.3.4 Rho

6.3.5 Hybrid

CHAPTER

SEVEN

PARAMETERIZATIONS

Sub-grid scale parameterizations are broadly grouped into vertical and lateral directions, referring to the direction in which fluxes are predominantly oriented.

7.1 Vertical Parameterizations

The following sub-grid scale parameterizations generally yield fluxes that act in the vertical direction, with no lateral components resolved by the model.

7.1.1 Upper boundary

K-profile parameterization (KPP) Provided by module MOM_KPP, uses the CVmix implementation of KPP.

CVMix_KPP

Energetic Planetary Boundary Layer (ePBL) A energetically constrained boundary layer scheme following [36]. Implemented in MOM_energetic_PBL.

Bulk mixed layer (BML) A 2-layer bulk mixed layer used in pure-isopycnal model. Implemented in MOM_bulk_mixed_layer.

7.1.2 Interior and bottom-driven mixing

Kappa-shear MOM_kappa_shear implement the shear-driven mixing of [27].

Internal-tide driven mixing The schemes of [42], [35], and [33], are all implemented through MOM_set_diffusivity and MOM_diabatic_driver.

7.1.3 Vertical friction

Vertical viscosity is implemented in MOM_vert_frict and coefficient computed in MOM_set_viscosity, although contributions to viscosity from other parameterizations are calculated in those respective modules (e.g. MOM_kappa_shear, MOM_KPP, MOM_energetic_PBL).

7.1.4 Vertical diffusion

Vertical diffusion of scalars is implemented in MOM_diabatic_driver although contributions to diffusion from other parameterizations are calculated in those respective modules (e.g. MOM_kappa_shear, MOM_KPP, MOM_energetic_PBL).

7.1.5 Radiation

Opacity Ocean color is prescribed or dynamically calculated in converted into optical properties in MOM_opacity.

Short-wave absorption Optical properties from MOM_opacity are used to calculate the convergence of shortwave radiation penetrating from the upper surface in MOM_shortwave_abs.

7.1.6 Geothermal heating

Geothermal heat fluxes are implemented in MOM_geothermal.

7.1.7 Isopycnal-mode entrainment and diapycnal diffusion

Diapycnal diffusion in a layered isopycnal mode following [21], is implemented in MOM_entrain_diffuse.

7.2 Lateral Parameterizations

The following sub-grid scale parameterizations generally yield fluxes that act in the lateral direction.

7.2.1 Lateral viscosity

Laplacian and bi-harmonic viscosities with linear and Smagorinsky options are implemented in MOM_hor_visc.

7.2.2 Gent-McWilliams/TEM/isopycnal height diffusion

Lagrangian mean eddy mass transport is parameterized following [16], in MOM_thickness_diffuse.

The diffusivity coefficients are calculated in MOM_lateral_mixing_coeffs and MOM_thickness_diffuse and includes constants and the [45] scaling.

A model of sub-grid scale Mesoscale Eddy Kinetic Energy (MEKE) is implement in MOM_MEKE and the associated diffusivity added in MOM_thickness_diffuse. See [28] and [31].

The Mesoscale Eddy Kinetic Energy (MEKE) framework

7.2.3 Backscatter

A parameterization of the upscale unresolved cascade utilizes MOM_MEKE and negative Laplacian viscosity in MOM_hor_visc.

7.2.4 Mixed layer restratification by sub-mesoscale eddies

Mixed layer restratification from [15] and [14] is implemented in MOM_mixed_layer_restrat.

7.2.5 Lateral diffusion

See *Horizontal Diffusion*.

7.2.6 Tidal forcing

Astronomical tidal forcings and self-attraction and loading are implement in MOM_tidal_forcing.

CHAPTER

EIGHT

OTHER PHYSICS

8.1 Equation of State

Within MOM6, there is a wrapper for the equation of state, so that all calls look the same from the rest of the model. The equation of state code has to calculate not just in situ density, but also the compressibility and various derivatives of the density. There is also code for computing specific volume and the freezing temperature.

8.1.1 Linear Equation of State

Compute the required quantities with uniform values for $\alpha = \frac{\partial \rho}{\partial T}$ and $\beta = \frac{\partial \rho}{\partial S}$, (DRHO_DT, DRHO_DS in MOM_input, also uses RHO_T0_S0).

8.1.2 Wright Equation of State

Compute the required quantities using the equation of state from [48] . This equation of state is in the form:

8.1.3 NEMO Equation of State

Compute the required quantities using the equation of state from [37] .

8.1.4 UNESCO Equation of State

Compute the required quantities using the equation of state from [26] .

8.1.5 TEOS-10 Equation of State

Compute the required quantities using the equation of state from TEOS-10 .

8.1.6 Freezing Temperature of Sea Water

There are three choices for computing the freezing point of sea water:

- Linear The freezing temperature is a linear function of the salinity and pressure:

$$T_{Fr} = (T_{Fr0} + a\,S) + b\,P$$

where T_{Fr0}, a, b are contants which can be set in MOM_input (TFREEZE_S0_P0, DTFREEZE_DS, DTFREEZE_DP).

- Millero The [34] equation is used, but modified so that it is a function of potential temperature rather than *in situ* temperature:

$$T_{Fr} = S(a + (b\sqrt{\max(S, 0.0)} + c\,S)) + d\,P$$

where a, b, c, d are fixed contants.

- TEOS-10 The TEOS-10 package is used to compute the freezing conservative temperature [degC] from absolute salinity [g/kg], and pressure [Pa]. This one must be used if you are using the NEMO or TEOS-10 equation of state.

8.2 Sea Ice Considerations

Sea Ice Coupling

8.2.1 Ice Formation

CHAPTER

NINE

WORKING WITH MOM6

9.1 Organization of the code

The MOM6 source code is divided into a tree of directories (folders) to group related code (e.g. *src/core*) or similar modules (e.g. *src/parametizations/vertical*).

The highest level of directories are:

src/ Code underneath *src/* is always required and compiled.

config_src/ Under *config_src* are various drivers and memory configuration sources that can only be compiled in limited configurations. See *config_src/*

pkg/ Packages (source code) from other sources/parties only some of which might be used. We include the entire package as a sub-module but use symbolic-links to extract the parts the MOM6 uses.

docs/ The directory that contains this documentation, namely that beyond the in-code API documentation. Some of the files are configuration files needed for running doxygen and sphinx. Most documentation in this folder is in the form of reStructuredText (.rst) files.

.testing/ A directory for running tests on MOM6. These are some of the smaller/simpler examples that MOM6 can run out of the box, without large netCDF files.

The directory tree is:

```
MOM6
├── config_src
│   ├── coupled_driver
│   ├── dynamic
│   ├── dynamic_symmetric
│   ├── ice_solo_driver
│   ├── solo_driver
│   └── unit_drivers
├── docs
│   └── images
├── pkg
│   ├── CVMix-src
│   │   ├── ...
│   │   └── src
│   │       ├── drivers
│   │       └── shared
│   └── GSW-Fortran
├── src
│   ├── ALE
│   ├── core
│   ├── diagnostics
```

(continues on next page)

(continued from previous page)

```
│   ├── equation_of_state
│   │   └── TEOS10
│   ├── framework
│   ├── ice_shelf
│   ├── initialization
│   ├── parameterizations
│   │   ├── CVmix -> ../../pkg/CVMix-src/src/shared
│   │   ├── lateral
│   │   └── vertical
│   ├── tracer
│   └── user
└── .testing
    ├── tc0
    ├── tc1
    ├── tc1.a
    ├── tc1.b
    ├── tc2
    ├── tc2.a
    ├── tc3
    └── tc4
```

9.1.1 *config_src/*

dynamic/*, *dynamic_symmetric/ One or none of *config_src/dynamic/* or *config_src/dynamic_symmetric/* can be included at compile time. If neither is used then a *MOM_memory.h* file specific to the model configuration must be present - this is known as a "static" compile with fixed layout and domain shape.

solo_driver/ This driver produces an ocean-only executable with no other coupled components (no sea-ice, no atmosphere, etc.). It is the simplest configuration and fastest to compile and thus used for a lot of testing.

coupled_driver/ This driver provides an interface for the GFDL coupler to call. It requires compiling MOM6 along with at least a sea-ice model and possibly all other components in a coupled model.

9.1.2 *src/*

core/ The dynamical core modules (except for the ALE remapping/regridding).

ALE/ Functions for remapping from between arbitrary vertical grids and generating grids.

diagnostics/ Some diagnostic calculations

equation_of_state/ Various equations of state (linear; Wright, 1997; TEOS-10; . . .).

framework/ Low-level wrappers for communication, diagnostics management, parsing of input parameters, time management, CPU clocks.

initialization/ Initialization of the horizontal grid, vertical coordinate, and the state.

parameterizations/lateral Sub-grid scale parameterization with fluxes primarily oriented in the lateral direction.

parameterizations/vertical Sub-grid scale parameterization with fluxes primarily oriented in the vertical direction, including the top and bottom boundary layer schemes.

tracer/ Everything to do with tracers, including advection and isopycnal stirring.

user/ Initialization and forcing for specific (coded) configurations.

9.2 Run-time Parameter System

How run-time parameters work in MOM6

MOM6 has an extensive set of parameters that are set at run-time by parsing an input file. Many parameters have default values and are not required to be in the input file, although there are a number of parameters that must be set for the model to run. The numerous examples provided with the MOM6 code mostly differ in their run-time parameters (although some add other components, like sea-ice), and comparison between these examples is an excellent way to get a broad overview of many of MOM6's parameters and how they might be set.

9.2.1 Getting parameters into MOM6

Run-time parameters are provided to the model in two phases:

1. A very small set of logistical parameters are read as namelist variables from the FMS parameter file `input.nml` . One of these logistical parameters is a list of ascii files that contain all the other run-time parameters.
2. All of the above-named parameter files are scanned for MOM6 model parameters, default values assigned and replaced, conflicts detected and various parameter summaries logged to files and/or the standard output.

Namelist parameters (*input.nml*)

All FMS derived MOM6 parameters reside in the namelist `MOM_input_nml` in the file `input.nml` . The parameters are:

- `input_filename` - If equal to "n" will run a new run (i.e. will not read a restart file). If equal to "r" MOM6 will attempt to read a restart file.
- `parameter_filename` - A list of file names containing the MOM6 internal run-time parameters. Typically `param_files="MOM6_input","MOM6_override"` where the file MOM6_input contains all the non-default parameters that define a "baseline" experiment and MOM6_override will be either empty (for baseline) or contain a few parameters that define a "derived" experiment (that differs from the baseline). This helps keep the parameter lists concise and enables easy comparison of parameters in related experiments.
- `restart_input_dir` , `restart_output_dir` , and `output_directory` - These specify the directories for reading input files, writing restart files, and writing many non-restart files.

Other MOM6-relevant FMS parameters

The namelist ocean_solo_nml may have the integer parameters secs, hours, days, months and years, which dictate how long the FMS ocean driver will try to run the model each run-segment.

MOM6 parameter file syntax

The general syntax for an entry in a MOM6 parameter file is

```
[!]#[override]  PARAMETER_NAME  =  value[,value][...][!comments]
```

Parameter names must be constructed from the characters `[A-Za-z0-9_]` and by soft convention are upper case. The `!` character is a remark or comment indicator; all subsequent text on that line is ignored.

Parameters that are not specified in the parameter files may assume a default value. It is not an error to specify a parameter more than once with the same value. It is an error to specify different values.

The keyword #override indicates that this parameter specification takes precedence over other specifications. It is an error to have two #override specifications for a single parameter with the same values. It is an error to have two #override statements with different values.

Some illustrations:

```
DO_THIS = True ! Set the Boolean to .TRUE.
DO_THAT = False ! Set the Boolean to .FALSE.
NXYZ = 5 ! Set the value to NXYZ to 5
HALF = 0.5 ! Set the value of HALF to 0.5
NAME = "abc" ! Set the string NAME to 'abc'
VECTOR = 1.0,2.0 ! Set the array VECTOR to [1.0, 2.0]
NAMES = 'abc','xyz' ! Set the strings NAMES to 'abc','xyz'
#override DO_THIS = False ! Set the Boolean to .FALSE., ignoring the␣
↪above specification
#override HALF = 0.25 ! Set the value of HALF to 0.25, ignoring the␣
↪above value
#override HALF = 0.25 ! Same as the above value of HALF to 0.25 so is␣
↪accepted
```

Logging of parameters

The subroutine that reads MOM6 parameters has also serves to log every parameter to a file set by DOCUMENT_FILE, usually "MOM6_parameter_doc". In addition to the name of the variable being read, these calls contain a brief description, along with a description of the units and the default value (if any) or an indication that there is no default and that the variable must be present. For example, `DT` is always required to be present:

```
call get_param(param_file, module, "DT", CS%dt, &
             "The (baroclinic) dynamics time step. The time-step that \n" //&
             "is actually used will be an integer fraction of the \n" //&
             "forcing time-step (DT_FORCING in ocean-only mode or the \n" //&
             "coupling timestep in coupled mode.)" , units= "s" , &
             fail_if_missing=.true.)
```

At run-time, two levels of logging are performed, depending on the value of the parameter `MINIMAL_DOCUMENTATION`:

- (TRUE) The end result of the combination of default values, assignments and overrides are recorded with default and current values, description and units, for all parameters.
- (FALSE) The minimal list of required and non-default value parameters are recorded with current values, description and units only for those parameters needed to reproduce the configuration.

Either of the generated logging files can be used as inputs and yield the same configuration.

In addition, there are also calls that log derived quantities (e.g., a time-step that is derived from a CFL number, or the full path to an input file) without reading anything in.

Error checking of parameters and parameter files

There are several techniques that are used for error checking on MOM6 parameters:

- Some parameters have internal error messages if they are set to nonsensical values.
- No parameter can be set twice inconsistently without an explicit #override specification.
- If the run-time parameter REPORT_UNUSED_PARAMS is true, a warning will be issued for any entries in the input parameter files that are not read in, for instance if they are misspelled.
- Setting the run-time parameter FATAL_UNUSED_PARAMS to true causes a fatal error that will bring down the model if there are any unused entries in the input parameter files.

9.3 Diagnostics

Controlling run-time diagnostics and how to add diagnostics to the code

MOM6 diagnostics are orchestrated via the FMS diag_manager, as for previous versions of MOM. However, because MOM6 is a general coordinate model, the model native-coordinae output can be less familiar to users of earlier generations of MOM. To alleviate this problem, MOM6 provides both "native" and "remapped" diagnostics; the former being diagnostics in the actual model coordinate space, and the latter in user-defined coordinates.

9.3.1 The "diag_table"

At run-time, diagnostics are controlled by the input file `diag_table` which is interpreted but the FMS package diag_manager.

The diag_table file has three kinds of section: Title, File and Field. The title section is mandatory and always the first. There can be multiple file and field sections, typically either in pairs or grouped in to all files and all fields, but always with the file section preceding the corresponding field section.

Title section

The first two lines are mandatory and comprise a line with a title and a line with six integers defining a base date against which time will be referenced.

```
"My  ocean-only  test  case"
1900  1  1  0  0  0
```

File section

This section defines an arbitrary number of files that will be created. Each file is limited to a single rate of either sampling or time-averaging.

```
"file_name",  output_freq,  "output_freq_units",  file_format,  "time_axis_units",
↪"time_axis_name"
```

- `file_name` : The name of the file that contains diagnostics at the given frequency (excluding the ".nc" extension).
- `output_freq` : The period between records in `file_name` , if positive. Special values of 0 mean write every time step and -1 write only at the end of the run.

- `output_freq_units` : The units in which `output_freq` is given. Valid values are "years", "months", "days", "hours", "minutes" or "seconds".
- `file_format` : Always set to 1, meaning netcdf.
- `time_axis_units` : The units to use for the time-axis in the file. Valid values are "years", "months", "days", "hours", "minutes" or "seconds".
- `time_axis_name` : The name of the time-axis (usually "Time").

Optional entries in the file line allow the generation of multiple files are intervals:

```
"file_name", output_freq, "output_freq_units", file_format, "time_axis_units",
↪"time_axis_name"[, new_file_freq, "new_file_freq_units"[, "start_time"[, file_
↪duration, "file_duration_units"]]]
```

- file_name : The base name of the file that contains diagnostics at the given frequency (excluding the ".nc" extension). The strings %yr, %mo, %dy, %hr %mi, %sc are expanded to the current year, month, day, hour, minute and second respectively, with new files created every new_file_freq.
- `new_file_freq` : The period between generation of new files.
- `new_file_freq_units` : The units in which `new_file_freq` is given.
- `start_time` , `file_duration` , `file_duration_units` : Even finer grain control of output files.

Field section

An arbitrary number of lines, one per diagnostic field:

```
"module_name", "field_name", "output_name", "file_name", "time_sampling",
↪"reduction_method", "regional_section", packing
```

- `module_name` : Name of the component model. For native ocean variables this should be "ocean_model". See *Vertically remapped diagnostics* for non-native vertical-grid diagnostics in the ocean model.
- `field_name` : The name of the variable as registered in the model.
- `output_name` " The name of the variable as it will appear in the file. This is usually the same as the `field_name` but can be used to rename a diagnostic.
- `file_name` : One of the files defined above in the section *File section* .
- `time_sampling` : Always set to "all".
- `reduction_method` : "none" means sample or snapshot. "average" or "mean" performs a time-average. "min" or "max" diagnose the minium or maxium over each time period.
- `regional_section` : "none" means global output. A string of six space separated numbers, "lat_min, lat_max, lon_min, lon_max, vert_min, vert_max", limits the diagnostic to a region.
- `packing` : Data representation in the file. 1 means "real*8", 2 means "real*4", 4 mean 16-bit integers, 8 means 1-byte.

Example

```
"OM4  1/4  degree"
1900  1  1  0  0  0

#  Static  file
"ocean_static",  -1,  "months",  1,  "days",  "time"  #  ocean_static  is  a␣
↪protected  name.  Do  not  change  this  line.
"ocean_model",  "deptho",     "deptho",     "ocean_static",  "all",  "none",  "none",␣
↪ 2
"ocean_model",  "geolon",     "geolon",     "ocean_static",  "all",  "none",  "none",␣
↪ 2
"ocean_model",  "geolat",     "geolat",     "ocean_static",  "all",  "none",  "none",␣
↪ 2
"ocean_model",  "wet",        "wet",        "ocean_static",  "all",  "none",  "none",␣
↪ 2

#  High-frequency  file
"surf_%4yr_%3dy",  1,  "hours",  1,  "days",  "time",  1,  "months"
"ocean_model","SSH","SSH","surf_%4yr_%3dy","all","none","none",2

#  Daily  averages
"ocean_daily",  1,  "days",  1,  "days",  "time"
"ocean_model",  "tos",  "tos",  "ocean_daily",  "all",  "mean",  "none",2

#  Monthly  averages
"ocean_month",  1,  "months",  1,  "days",  "time"
"ocean_model",  "thetao",  "thetao",  "ocean_month",  "all",  "mean",  "none",2

#  Annual  averages
"ocean_annual",  12,  "months",  1,  "days",  "time"
"ocean_model",  "thetao",  "thetao",  "ocean_annual",  "all",  "mean",  "none",2

#  Vertical  section
"ocean_Bering_Strait",  5,  "days",  1,  "days",  "time"
"ocean_model",  "thetao","thetao",  "ocean_Bering_Strait",  "all",  "mean",  "-171.4␣
↪-168.7   66.1  66.1  -1  -1",2
```

9.3.2 Native diagnostics

The list of available diagnostics is dependent on the particular configuration of the model. For this reason the model writes a record of the available diagnostic fields at run-time into a file "available_diags.*". See, for example, available_diags.000000 for the global_ALE z-coordinate ocean-only test case.

Diagnostic fields in the module "ocean_model" refer to the native variables or diagnostics in the native grid. Since the model can be run in arbitrary coordinates, say in hybrid-coordinate mode, then native-space diagnostics can be potentially confusing. Native diagnostics are useful when examining exactly what the model is doing, or if the vertical coordinate of the model is configured to be a natural coordinate such as pure isopycnal or z* geopotential.

9.3.3 Vertically remapped diagnostics

Alternative vertical coordinates can be configured for diagnostic purposes only.

The run-time parameter NUM_DIAG_COORDS controls how many diagnostic coordinates to use.

The run-time parameter DIAG_COORDS defines the mapping between each coordinate, the name of the module in the diag_table and run-time parameter names that define the coordinate. A list of string tuples, separated by commas, with each tuple in the form of MODULE_SUFFIX PARAMETER_SUFFIX COORDINATE_NAME. MODULE_SUFFIX is the string appended to "ocean_model" to create a module in the diag_table. PARAMETER_SUFFIX is the string appended to "DiAG_COORD_DEF", and other parameters, used to control the generation of the named coordinate. COORDINATE_NAME is a name understood by the MOM6 regridding module. Valid examples are "ZSTAR", "SIGMA", "RHO", etc.

By default, NUM_DIAG_COORDS=1 and DIAG_COORDS="z Z ZSTAR" , meaning the module "ocean_model_z" provides diagnostics in "z*" coordinates and uses the parameter DIAG_COORD_DEF_Z .

For example, multiple z*-coordinates could be used for diagnostics with

```
NUM_DIAG_COORDS = 2
DIAG_COORDS = "z 01 ZSTAR,abc 02 ZSTAR"
DIAG_COORD_DEF_01 = "WOA09"
DIAG_COORD_DEF_02 = "UNIFORM:10,20."
```

would create the diag_manager modules "ocean_model_z" and "ocean_model_abc".

The above is equivalent to

```
NUM_DIAG_COORDS = 2
DIAG_COORDS = "z ZA ZSTAR,abc ZB ZSTAR"
DIAG_COORD_DEF_ZA = "WOA09"
DIAG_COORD_DEF_ZB = "UNIFORM:10,20."
```

To obtain a diagnostic of monthly-averaged potential temperature in both these coordinate systems the diag_table must include the lines

```
"ocean_month_z", 1, "months", 1, "days", "time"
"ocean_month_abc", 1, "months", 1, "days", "time"
"ocean_model_z", "temp", "temp", "ocean_month_z", "all", "mean", "none",2
"ocean_model_abc", "temp", "temp", "ocean_month_abc", "all", "mean", "none",2
```

Diagnostic vertical coordinates

For each of the NUM_DIAG_COORDS vertical coordinates listed in DIAG_COORDS the corresponding DIAG_COORD_DEF_% parameter must be provided. It can take the following values:

- PARAM : In this case, a corresponding parameter DIAG_COORD_RES_% is read that lists the deltas for each level in the coordinate. For example, DIAG_COODS="z Z ZTAR", DIAG_COORD_DEF_Z="PARAM", DIAG_COORD_RES_Z=5,10,10,15 creates z*-level with 5,10,10,15 meters thicknesses.
- UNIFORM : Uniform distribution down to the maximum depth of the model using the same number of levels as he model.
- UNIFORM:N : Uniform distribution down to the maximum depth of the model using N levels.
- UNIFORM:N,D : Uniform distribution of N levels with thickness D .
- FILE:filename,varname : Reads vector of coordinate thicknesses from field "varname" from file "filename".

- FILE:filename,interfaces=varname : Reads vector of coordinate positions from field "varname" from file "filename".
- WOA09 : Z-levels that correspond to the World Ocean Atlas, 2009, standard levels down to and including the maximum depth of the model.
- WOA09:N : The first `N` levels of the World Ocean Atlas, 2009, standard levels.

9.3.4 APIs for diagnostics

The multiple diagnostic-coordinates are implemented in a layer that sits on top of the FMS diag_manager known as the mom6_diag_mediator.

A diagnostic is registered with register_diag_field() which is an API that looks similar to the FMS diag_manager routine of the same name:

```
diag_id = register_diag_field(module_name, diag_name, axes, ...)
```

The MOM6 version of this routine optionally allows the specification of CMOR names in addition to the native names which then registers the diagnostic twice, once with each name.

For each of the native and CMOR names, the diagnostic is registered in the native module "ocean_model", corresponding to the native model coordinate, and a module associated with each of the diagnostic coordinates.

For each diagnostic coordinate, a horizontally-averaged diagnostic is also registered.

In all, for each 3D diagnostic, the are 2 + 4N diagnostics registered, where N is the number of diagnostic coordinates. As a result, the global_ALE examples have of order 900 total diagnostics available in the shipped configuration.

The data is made available to the diag_manager via a call to post_data() which is a wrapper that does all the vertical remapping before calling FMS's send_data():

```
call post_data(diag_id, data, diag_control)
```

Artifacts of posting frequency for diagnostics

Variables area "posted" for i/o or averaging to the diag_manager (via MOM6's diag_mediator) at different frequencies relative to each other. This is because the MOM6 algorithm takes the form of nested sub-cycles with different time-steps in each loop (e.g. barotropic solver with dynamics). A consequence of this is that a time average of a related quantities may appear to be inconsistent since the diagnostic posted with higher frequency may not vary linearly between the end-points seen on the longer time-step. The differences are usually small, but if you see large differences it might indicate you should re-examine the time-steps used for the various sub-cycles.

9.4 Horizontal indexing and memory

Conventions for staggering of variables and loops over 2d/3d arrays

MOM6 is written in Fortran90 and uses the `i,j,k` order of indexing. `i` corresponds to the fastest index (stride-1 in memory) and thus should be the inner-most loop variable. We often refer to the i-direction as the x- or zonal direction, and similarly to the j-direction as y- or meridional direction. The model can use curvilinear grids/coordinates in the horizontal and so these labels have loose meanings but convenient.

9.4.1 Loops and staggered variables

Many variables are staggered horizontally with respect to each other. The dynamics and tracer equations are discretized on an Arakawa C grid. Staggered variables must still have integer indices and we use a north-east convention centered on the h-points. These means a variable with indices `i,j` will be either collocated, to the east, to the north, or to the north-east of the h-point with the same indices.

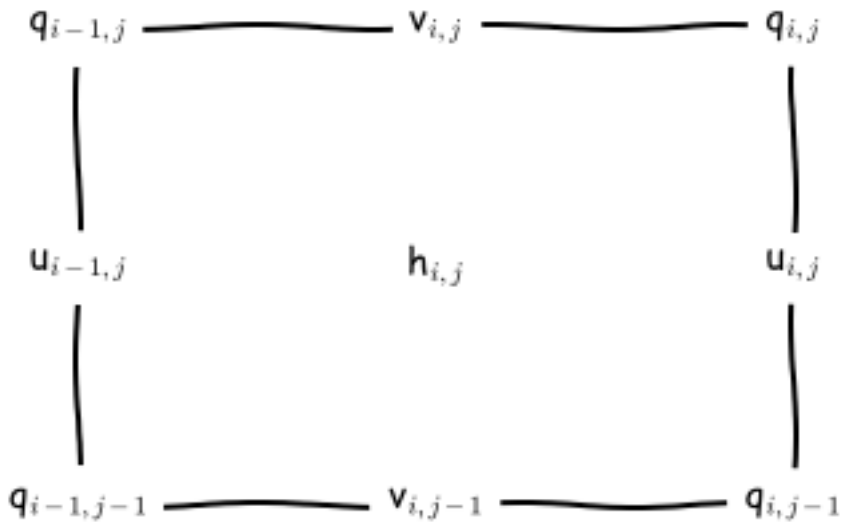

Fig. 1: MOM6 uses an Arakawa C grid staggering of variables with a North-East indexing convention. "Cells" refer to the control volumes around tracer- or h-point located variables unless labelled otherwise.

Soft convention for loop variables

To ease reading the code we use a "soft" convection (soft because there is no syntax checking) where an upper-case index variable can be interpreted as the lower-case index variable plus $\frac{1}{2}$.

For example, when a loop is over h-points collocated variables

- the do-loop statements will be for lower-case `i,j` variables
- references to h-point variables will be `h(i,j)` , `D(i+1,j)` , etc.
- references to u-point variables will be `u(I,j)` (meaning $u_{i+\frac{1}{2},j}$), `u(I-1,j)` (meaning $u_{i-\frac{1}{2},j}$), etc.
- references to v-point variables will be `v(i,J)` (meaning $v_{i,j+\frac{1}{2}}$), `u(I-1,j)` (meaning $u_{i,j-\frac{1}{2}}$), etc.
- references to q-point variables will be `q(I,J)` (meaning $q_{i+\frac{1}{2},j+\frac{1}{2}}$), etc.

In contrast, when a loop is over u-points collocated variables

- the do-loop statements will be for upper-case `I` and lower-case `j` variables
- the expression $u_{i+\frac{1}{2},j}(h_{i,j} + h_{i+1,j})$ is `u(I,j) * ( h(i,j) + h(i+1,j)` .

9.4.2 Declaration of variables

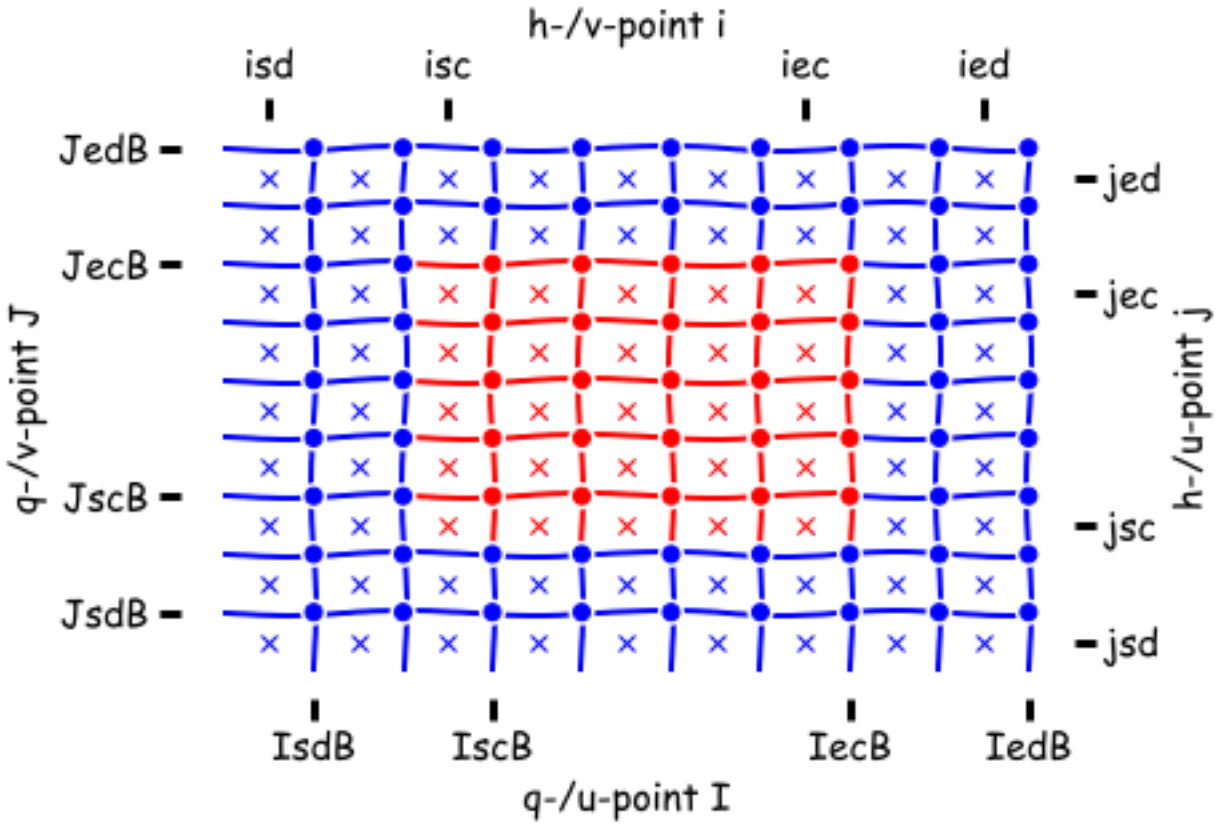

Fig. 2: Non-symmetric mode: All arrays are declared with the same shape (isd:ied,jsd:jed).

A field is described by 2D or 3D arrays which are distributed across parallel processors. Each processor only sees a small window of the global field. The processor "owns" the computational domain (red in above figure) but arrays are extended horizontally with halos which are intermittently updated with the values from neighboring processors. The halo regions (blue in above figure) may not always be up-to-date. Data in halo regions (blue in above figure) will be overwritten my mpp_updates.

MOM6 has two memory models, "symmetric" and "non-symmetric". In non-symmetric mode all arrays are given the same shape. The consequence of this is that there are fewer staggered variables to the south-west of the computational domain. An operator applied at h-point locations involving u- or v- point data can not have as wide a stencil on the south-west side of the processor domain as it can on the north-east side.

In symmetric mode, declarations are dependent on the variables staggered location on the Arakawa C grid. This allows loops to be symmetric and stencils to be applied more uniformly.

In the code, declarations are consistent with the symmetric memory model. The non-symmetric mode is implemented by setting the start values of the staggered data domain to be the same as the non-staggered start value.

The horizontal index type (mom_hor_index::hor_index_type) provides the data domain start values. The values are also copied into the mom_grid::ocean_grid_type for convenience although we might deprecate this convenience in the future.

Declarations of h-point data take the form:

- `real, dimension(HI%isd:HI%ied, HI%jsd:HI%jed) :: D !< Depth at h-points (m)`

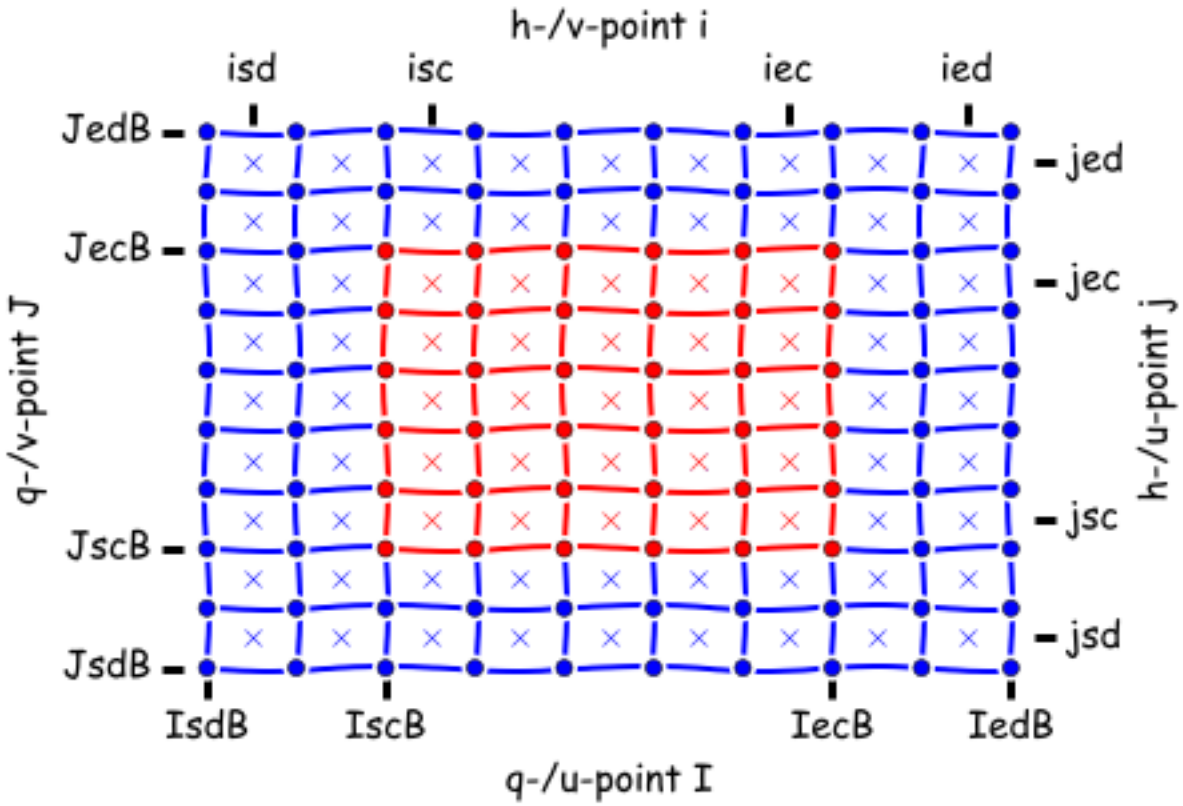

Fig. 3: Symmetric mode: Arrays have different shapes depending on their staggering location on the Arakawa C grid.

- `real, dimension(HI%isd:HI%ied, HI%jsd:HI%jed, GV%ke) :: h !< Layer thickness (H units)`

Declarations of u-point data take the form:

- `real, dimension(HI%IsdB:HI%IedB, HI%jsd:HI%jed) :: Du !< Depth at u-points (m)`
- `real, dimension(HI%IsdB:HI%IedB, HI%jsd:HI%jed, GV%ke) :: h !< Zonal flow (m/s)`

Declarations of v-point data take the form:

- `real, dimension(HI%isd:HI%ied, HI%JsdB:HI%JedB) :: Dv !< Depth at v-points (m)`
- `real, dimension(HI%isd:HI%ied, HI%JsdB:HI%JedB, GV%ke) :: h !< Zonal flow (m/s)`

Declarations of q-point data take the form:

- `real, dimension(HI%IsdB:HI%IedB, HI%JsdB:HI%JedB) :: Dq !< Depth at q-points (m)`
- `real, dimension(HI%IsdB:HI%IedB, HI%JsdB:HI%JedB, GV%ke) :: vort !< Vertical componentof vorticity (s-1)`

The file MOM_memory_macros.h provides the macros `SZI_` , `SZJ_` , `SZIB_` and `SZJB_` that help make the above more concise:

- `real, dimension(SZI_(HI), SZJ_(HI)) :: D !< Depth at h-points (m)`
- `real, dimension(SZIB_(HI), SZJ_(HI)) :: Du !< Depth at u-points (m)`
- `real, dimension(SZI_(HI), SZJB_(HI)) :: Dv !< Depth at v-points (m)`
- `real, dimension(SZIB_(HI), SZJB_(HI)) :: Dq !< Depth at q-points (m)`

See MOM_memory_macros.h for the complete list of macros used in various memory modes.

9.4.3 Calculating a global index

For the most part MOM6 code should be independent of an equivalent absolute global index. There are exceptions and when the global index of a cell `i,j` is needed is can be calculated as follows: `i_global = i + HI%idg_offset`

Before the mom_hor_index::hor_index_type was introduced, this conversion was done use variables in the mom_grid::ocean_grid_type: `i_global = (i-G%isd) + G%isd_global`

which is no longer preferred.

Note that a global index only makes sense for a rectangular global domain. If the domain is a Mosaic of connected tiles (e.g. size tiles of a cube) the global indices (i,j) become meaningless.

CHAPTER

TEN

FORCING

10.1 Solar Radiation

Penetration of Solar Radiation

10.1.1 Jerlov water type

10.1.2 Absorption by Chlorophyll

10.2 Tracer Fluxes

Tracer Fluxes

10.2.1 Tracer Fluxes

10.2.2 River Runoff

10.2.3 Ice Runoff

CHAPTER

ELEVEN

PARALLEL IMPLEMENTATION

11.1 Domain Decomposition

11.1.1 Domain Decomposition

MOM6 supports serial, OpenMP, and MPI computations, with the user choosing between them at run time. All are accomplished through domain decomposition in the horizontal. All of the horizontal operations are explicit with a relatively small footprint, so the tiling is a logical choice. Some goals in the parallel design of MOM6 were:

- Don't hard-code the number of processes.
- MPI and OpenMP share the same basic structure.
- Don't break the serial optimizations.
- Same result as serial code for any number of processes.
- Portability - able to run on any (Unix) system.

The whole horizontal MOM6 grid is shown in *Declaration of variables* . The computations are done over the cells inside the darker line; the cells are numbered 1 to NIGLOBAL in the x -direction and 1 to NJGLOBAL in the y -direction. Those looking ahead to running in parallel would be wise to include factors of two and three in their choice of NIGLOBAL and NJGLOBAL when building new grids. MOM6 will run in parallel with any values of these, but the computations might not be load-balanced.

11.1.2 Wide Halos

11.2 Parallel I/O

Parallel I/O

The model can be told to write a different output file per process. This may or may not save time, and is a bad idea on Lustre filesystems. If the model is writing individual files per process, one can combine them using the mppnccombine program from the FRE-nctools package .

CHAPTER

TWELVE

TESTING OF MOM6

12.1 Testing

MOM6 Validation and Verification

In the software engineering world, people talk about validation and verification of their codes. Verification is the confirmation of design specifications, such as:

- Does it compile on the target platform?
- Is it dimensionally consistent?
- Do answers change with the number of processes?
- Do answers change after a restart?

Validation is a little trickier:

- Does the model meet operational needs?
- Does it produce realistic simulations?
- Are relevant physical features present?
- Can I reproduce my old simulations?

There are a number of ways in which MOM6 is tested before each commit, especially commits to the shared dev/main branch.

12.1.1 Travis Testing

When pushing code to github, it is possible to set it up so that testing is performed automatically by travis. For MOM6, the .travis.yml file is executed, causing the code to be compiled and then run on all the tests in the .testing directory. It is also possible to run these tests on your local machine, but you might have to do some setup first. See

See also

../../../.testing/README.md for more information.

12.1.2 Consortium Testing

For commits to the dev/main branch, there is an opportunity for all consortium members to weigh in on proposed updates. A view of the consortium is shown here:

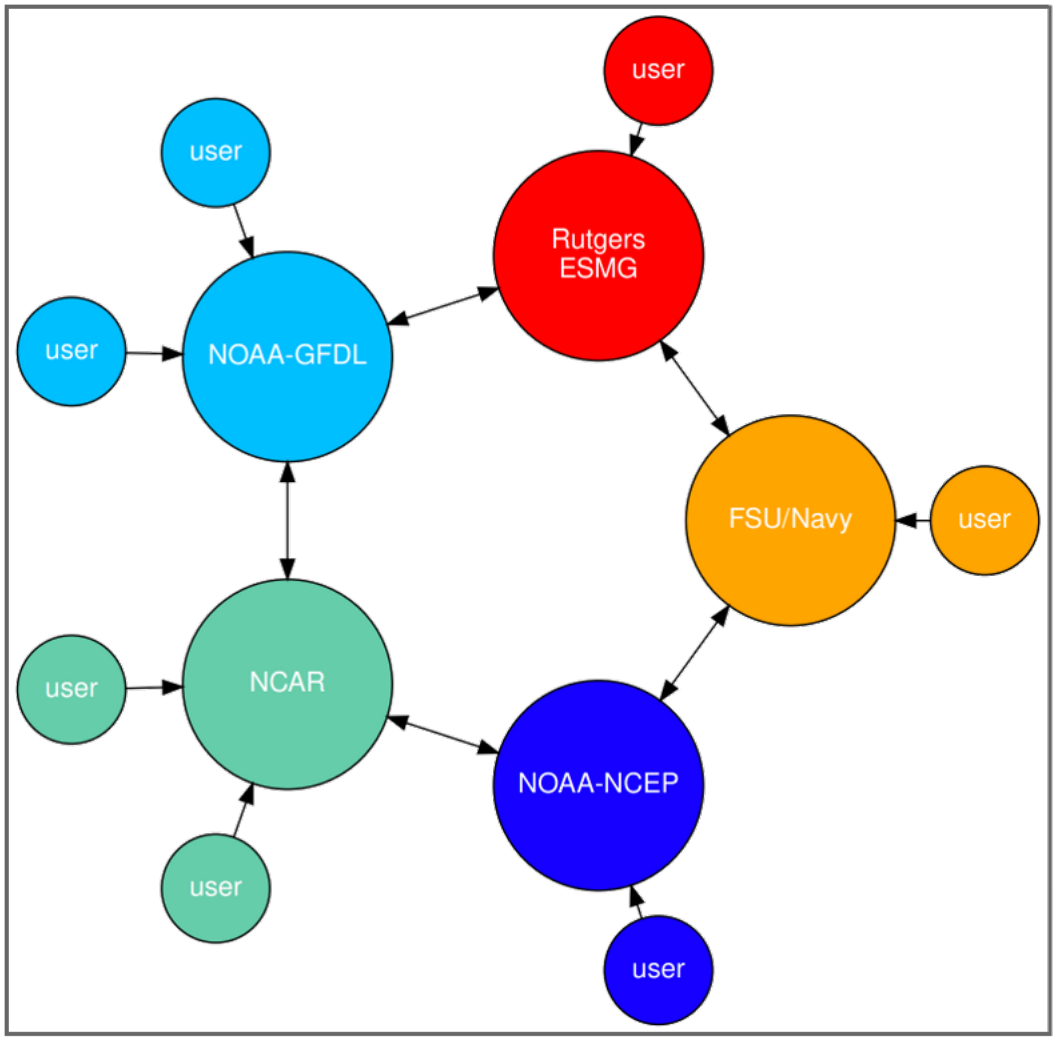

Fig. 1: The MOM6 consortium.

Each group is expected to have their own tests and to keep track of expected answers when these tests are run to be compared to prior answers after the code is updated. Answer-changing updates have to be evaluated carefully, though there are circumstances in which the new answers may well be "better".

12.1.3 Novel Tests

There are two classes of tests which MOM6 performs within the .testing suite which could be considered unusual, but which can be quite useful for finding bugs.

Scaling tests

The equations of motion can be multiplied by factors of two without changing answers. One can use that to scale each of six units by a different factor of two to check for consistent use of units. For instance, the equation:

$$u^{n+1} = u^n + \Delta t \times \mathcal{F}$$

can be scaled as:

$$2^{L-T} u^{n+1} = 2^{L-T} u^n + 2^T \Delta t \times 2^{L-2T} \mathcal{F}$$

MOM6 has been recoded to include six different scale factors:

Unit	Scaling	Name
s	T	Time
m	L	Horizontal length
m	H	Layer thickness
m	Z	Vertical length
kg/m3	R	Density
J/kg	Q	Enthalpy

You can add these integer scaling factors through the runtime parameters X_RESCALE_POWER, where X is one of T, L, H, Z, R, or Q. The valid range for these is -300 to 300.

When adding contributions to MOM6, this coding style with the scale factors must be maintained. For example, if you add new parameters to read from the input file:

```
call get_param(..., "DT", ... , scale=US%s_to_T)
```

This is also required for explicit contants, though we are trying to move those out of the code:

```
ustar = 0.01 * US%m_to_Z * US%T_to_s
```

or for adding diagnostics:

```
call register_diag_field(..., "u", ... , &
       conversion=US%L_T_to_m_s)
```

See also

mom_unit_scaling()

Rotational tests

By setting the runtime option ROTATE_INDEX to True, the model rotates the domain by some number of 90 degree turns. This option can be used to look for bugs in which east-west operations do not match north-south operations. It changes the order of array elements as shown here:

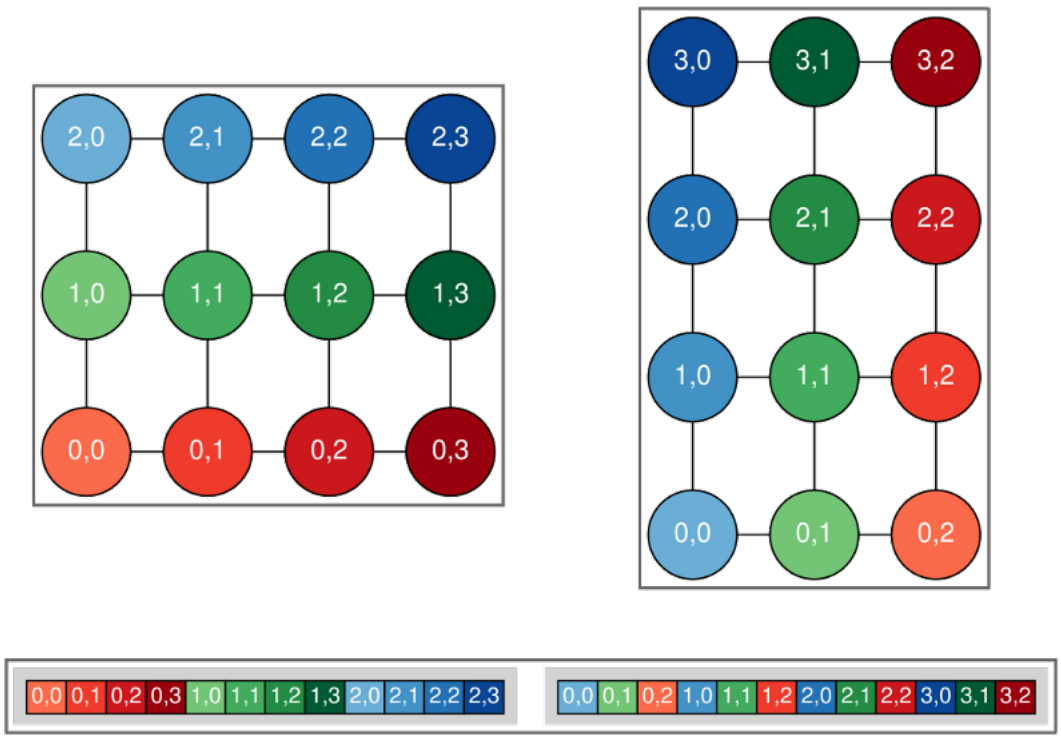

Fig. 2: The original non-rotated domain is shown on the left while the right shows the domain rotated counterclockwise by 90 degrees. The array values are shown by the (invariant) colors, while the array indices (and dimensions) change.

It only currently runs in serial mode. One can ask for rotations of 90, 180, or 270 degrees, but only 90 degree turns are supported if there are open boundaries.

Because order matters in numerical computations, care must be taken for four-way averages to match between rotated and non-rotated runs. Say you want to compute the following quantity:

$$\phi^{(c)}_{i,j} = \frac{1}{4}(\phi_A + \phi_B + \phi_C + \phi_D)$$

as shown in this diagram:

You might write this first as:

$$\frac{1}{4}((\phi_A + \phi_B) + (\phi_C + \phi_D))$$

as shown on the left in this figure:

However, the round-off errors could give differing answers when rotated. Instead, you want to group the terms on the diagonal as shown in the right of the above figure and here:

$$\frac{1}{4}((\phi_A + \phi_D) + (\phi_B + \phi_C))$$

CHAPTER

THIRTEEN

API REFERENCE

This API reference is a partial image of the complete API documentation. The pages you find here are linkable - the url for the page is static. The complete API documentation is generated with doxygen and can be found at http://noaa-gfdl.github.io/MOM6/APIs/.

13.1 Modules

mom	The central module of the MOM6 ocean model.
mom_ale	This module contains the main regridding routines.
mom_eos	Provides subroutines for quantities specific to the equation of state.
mom_ice_shelf	Implements the thermodynamic aspects of ocean / ice-shelf interactions, along with a crude placeholder for
mom_meke	Implements the Mesoscale Eddy Kinetic Energy framework with topographic beta effect included in compu
mom_unit_scaling	Provides a transparent unit rescaling type to facilitate dimensional consistency testing.

13.1.1 mom module reference

The central module of the MOM6 ocean model.

More...

Data Types

mom_control_struct	Control structure for the MOM module, including the variables that describe the state of the ocean.
mom_diag_ids	A structure with diagnostic IDs of the state variables.

Functions/Subroutines

step_mom()	This subroutine orchestrates the time stepping of MOM.
step_mom_dynamics()	Time step the ocean dynamics, including the momentum and continuity equations.
step_mom_tracer_dyn()	step_MOM_tracer_dyn does tracer advection and lateral diffusion, bringing the tracers up
step_mom_thermo()	MOM_step_thermo orchestrates the thermodynamic time stepping and vertical remapping.
step_offline()	step_offline is the main driver for running tracers offline in MOM6. This has been primaril
initialize_mom()	Initialize MOM, including memory allocation, setting up parameters and diagnostics, initia
finish_mom_initialization()	Finishes initializing MOM and writes out the initial conditions.

register_diags()	Register certain diagnostics.
mom_timing_init()	Set up CPU clock IDs for timing various subroutines.
set_restart_fields()	Set the fields that are needed for bitwise identical restarting the time stepping scheme.
adjust_ssh_for_p_atm()	Apply a correction to the sea surface height to compensate for the atmospheric pressure (th
extract_surface_state()	Set the surface (return) properties of the ocean model by setting the appropriate fields in sf
rotate_initial_state()	Rotate initialization fields from input to rotated arrays.
mom_state_is_synchronized()	Return true if all phases of step_MOM are at the same point in time.
get_mom_state_elements()	This subroutine offers access to values or pointers to other types from within the MOM_co
get_ocean_stocks()	Find the global integrals of various quantities.
mom_end()	End of ocean model, including memory deallocation.

Detailed Description

Modular Ocean Model (MOM) Version 6.0 (MOM6)

Additional contributions from:

- Whit Anderson
- Brian Arbic
- Will Cooke
- Anand Gnanadesikan
- Matthew Harrison
- Mehmet Ilicak
- Laura Jackson
- Jasmine John
- John Krasting
- Zhi Liang
- Bonnie Samuels
- Harper Simmons
- Laurent White
- Niki Zadeh

MOM ice-shelf code was developed by

- Daniel Goldberg
- Robert Hallberg
- Chris Little
- Olga Sergienko

Overview of MOM

This program (MOM) simulates the ocean by numerically solving the hydrostatic primitive equations in generalized Lagrangian vertical coordinates, typically tracking stretched pressure (p*) surfaces or following isopycnals in the ocean's interior, and general orthogonal horizontal coordinates. Unlike earlier versions of MOM, in MOM6 these equations are horizontally discretized on an Arakawa C-grid. (It remains to be seen whether a B-grid dynamic core will be revived in MOM6 at a later date; for now applications requiring a B-grid discretization should use MOM5.1.) MOM6 offers a range of options for the physical parameterizations, from those most appropriate to highly idealized models for geophysical fluid dynamics studies to a rich suite of processes appropriate for realistic ocean simulations. The thermodynamic options typically use conservative temperature and preformed salinity as conservative state variables and a full nonlinear equation of state, but there are also idealized adiabatic configurations of the model that use fixed density layers. Version 6.0 of MOM continues in the long tradition of a commitment to climate-quality ocean simulations embodied in previous versions of MOM, even as it draws extensively on the lessons learned in the development of the Generalized Ocean Layered Dynamics (GOLD) ocean model, which was also primarily developed at NOAA/GFDL. MOM has also benefited tremendously from the FMS infrastructure, which it utilizes and shares with other component models developed at NOAA/GFDL.

When run is isopycnal-coordinate mode, the uppermost few layers are often used to describe a bulk mixed layer, including the effects of penetrating shortwave radiation. Either a split- explicit time stepping scheme or a non-split scheme may be used for the dynamics, while the time stepping may be split (and use different numbers of steps to cover the same interval) for the forcing, the thermodynamics, and for the dynamics. Most of the numerics are second order accurate in space. MOM can run with an absurdly thin minimum layer thickness. A variety of non-isopycnal vertical coordinate options are under development, but all exploit the advantages of a Lagrangian vertical coordinate, as discussed in detail by Adcroft and Hallberg (Ocean Modelling, 2006).

Details of the numerics and physical parameterizations are provided in the appropriate source files. All of the available options are selected at run-time by parsing the input files, usually MOM_input and MOM_override, and the options choices are then documented for each run in MOM_param_docs.

MOM6 integrates the equations forward in time in three distinct phases. In one phase, the dynamic equations for the velocities and layer thicknesses are advanced, capturing the propagation of external and internal inertia-gravity waves, Rossby waves, and other strictly adiabatic processes, including lateral stresses, vertical viscosity and momentum forcing, and interface height diffusion (commonly called Gent-McWilliams diffusion in depth- coordinate models). In the second phase, all tracers are advected and diffused along the layers. The third phase applies diabatic processes, vertical mixing of water properties, and perhaps vertical remapping to cause the layers to track the desired vertical coordinate.

The present file (`MOM.F90`) orchestrates the main time stepping loops. One time integration option for the dynamics uses a split explicit time stepping scheme to rapidly step the barotropic pressure and velocity fields. The barotropic velocities are averaged over the baroclinic time step before they are used to advect thickness and determine the baroclinic accelerations. As described in Hallberg and Adcroft (2009), a barotropic correction is applied to the time-mean layer velocities to ensure that the sum of the layer transports agrees with the time-mean barotropic transport, thereby ensuring that the estimates of the free surface from the sum of the layer thicknesses agrees with the final free surface height as calculated by the barotropic solver. The barotropic and baroclinic velocities are kept consistent by recalculating the barotropic velocities from the baroclinic transports each time step. This scheme is described in Hallberg, 1997, J. Comp. Phys. 135, 54-65 and in Hallberg and Adcroft, 2009, Ocean Modelling, 29, 15-26.

The other time integration options use non-split time stepping schemes based on the 3-step third order Runge-Kutta scheme described in Matsuno, 1966, J. Met. Soc. Japan, 44, 85-88, or on a two-step quasi-2nd order Runge-Kutta scheme. These are much slower than the split time-stepping scheme, but they are useful for providing a more robust solution for debugging cases where the more complicated split time-stepping scheme may be giving suspect solutions.

There are a range of closure options available. Horizontal velocities are subject to a combination of horizontal biharmonic and Laplacian friction (based on a stress tensor formalism) and a vertical Fickian viscosity (perhaps using the kinematic viscosity of water). The horizontal viscosities may be constant, spatially varying or may be dynamically calculated using Smagorinsky's approach. A diapycnal diffusion of density and thermodynamic quantities is also allowed, but not required, as is horizontal diffusion of interface heights (akin to the Gent-McWilliams closure of

geopotential coordinate models). The diapycnal mixing may use a fixed diffusivity or it may use the shear Richardson number dependent closure, like that described in Jackson et al. (JPO, 2008). When there is diapycnal diffusion, it applies to momentum as well. As this is in addition to the vertical viscosity, the vertical Prandtl always exceeds 1. A refined bulk-mixed layer is often used to describe the planetary boundary layer in realistic ocean simulations.

MOM has a number of noteworthy debugging capabilities. Excessively large velocities are truncated and MOM will stop itself after a number of such instances to keep the model from crashing altogether. This is useful in diagnosing failures, or (by accepting some truncations) it may be useful for getting the model past the adjustment from an ill-balanced initial condition. In addition, all of the accelerations in the columns with excessively large velocities may be directed to a text file. Parallelization errors may be diagnosed using the DEBUG option, which causes extensive checksums to be written out along with comments indicating where in the algorithm the sums originate and what variable is being summed. The point where these checksums differ between runs is usually a good indication of where in the code the problem lies. All of the test cases provided with MOM are routinely tested to ensure that they give bitwise identical results regardless of the domain decomposition, or whether they use static or dynamic memory allocation.

Structure of MOM

About 115 other files of source code and 4 header files comprise the MOM code, although there are several hundred more files that make up the FMS infrastructure upon which MOM is built. Each of the MOM files contains comments documenting what it does, and most of the file names are fairly self-evident. In addition, all subroutines and data types are referenced via a module use, only statement, and the module names are consistent with the file names, so it is not too hard to find the source file for a subroutine.

The typical MOM directory tree is as follows:

```
../MOM
|-- config_src
|   |-- coupled_driver
|   |-- dynamic
|   `-- solo_driver
|-- examples
|   |-- CM2G
|   |-- ...
|   `-- torus_advection_test
`-- src
    |-- core
    |-- diagnostics
    |-- equation_of_state
    |-- framework
    |-- ice_shelf
    |-- initialization
    |-- parameterizations
    |   |-- lateral
    |   `-- vertical
    |-- tracer
    `-- user
```

Rather than describing each file here, each directory contents will be described to give a broad overview of the MOM code structure.

The directories under config_src contain files that are used for configuring the code, for instance for coupled or ocean-only runs. Only one or two of these directories are used in compiling any, particular run.

- config_src/coupled_driver: The files here are used to couple MOM as a component in a larger run driven by the FMS coupler. This includes code that converts various forcing fields into the code structures and flux and

unit conventions used by MOM, and converts the MOM surface fields back to the forms used by other FMS components.

- config_src/dynamic: The only file here is the version of MOM_memory.h that is used for dynamic memory configurations of MOM.
- config_src/solo_driver: The files here are include the _main driver that is used when MOM is configured as an ocean-only model, as well as the files that specify the surface forcing in this configuration.

The directories under examples provide a large number of working configurations of MOM, along with reference solutions for several different compilers on GFDL's latest large computer. The versions of MOM_memory.h in these directories need not be used if dynamic memory allocation is desired, and the answers should be unchanged.

The directories under src contain most of the MOM files. These files are used in every configuration using MOM.

- src/core: The files here constitute the MOM dynamic core. This directory also includes files with the types that describe the model's lateral grid and have defined types that are shared across various MOM modules to allow for more succinct and flexible subroutine argument lists.
- src/diagnostics: The files here calculate various diagnostics that are anciliary to the model itself. While most of these diagnostics do not directly affect the model's solution, there are some, like the calculation of the deformation radius, that are used in some of the process parameterizations.
- src/equation_of_state: These files describe the physical properties of sea-water, including both the equation of state and when it freezes.
- src/framework: These files provide infrastructure utilities for MOM. Many are simply wrappers for capabilities provided by FMS, although others provide capabilities (like the file_parser) that are unique to MOM. When MOM is adapted to use a modeling infrastructure distinct from FMS, most of the required changes are in this directory.
- src/initialization: These are the files that are used to initialize the MOM grid or provide the initial physical state for MOM. These files are not intended to be modified, but provide a means for calling user-specific initialization code like the examples in src/user.
- src/parameterizations/lateral: These files implement a number of quasi-lateral (along-layer) process parameterizations, including lateral viscosities, parameterizations of eddy effects, and the calculation of tidal forcing.
- src/parameterizations/vertical: These files implement a number of vertical mixing or diabatic processes, including the effects of vertical viscosity and code to parameterize the planetary boundary layer. There is a separate driver that orchestrates this portion of the algorithm, and there is a diversity of parameterizations to be found here.
- src/tracer: These files handle the lateral transport and diffusion of tracers, or are the code to implement various passive tracer packages. Additional tracer packages are readily accommodated.
- src/user: These are either stub routines that a user could use to change the model's initial conditions or forcing, or are examples that implement specific test cases. These files can easily be hand edited to create new analytically specified configurations.

Most simulations can be set up by modifying only the files MOM_input, and possibly one or two of the files in src/user. In addition, the diag_table (MOM_diag_table) will commonly be modified to tailor the output to the needs of the question at hand. The FMS utility mkmf works with a file called path_names to build an appropriate makefile, and path_names should be edited to reflect the actual location of the desired source code.

There are 3 publicly visible subroutines in this file (`MOM.F90`).

- step_MOM steps MOM over a specified interval of time.
- MOM_initialize calls initialize and does other initialization that does not warrant user modification.
- extract_surface_state determines the surface (bulk mixed layer if traditional isoycnal vertical coordinate) properties of the current model state and packages pointers to these fields into an exported structure.

The remaining subroutines in this file (`src/core/MOM.F90`) are:

- find_total_transport determines the barotropic mass transport.
- register_diags registers many diagnostic fields for the dynamic solver, or of the main model variables.
- MOM_timing_init initializes various CPU time clocks.
- write_static_fields writes out various time-invariant fields.
- set_restart_fields is used to specify those fields that are written to and read from the restart file.

Diagnosing MOM heat budget

Here are some example heat budgets for the ALE version of MOM6.

Depth integrated heat budget

Depth integrated heat budget diagnostic for MOM.

- OPOTTEMPTEND_2d = T_ADVECTION_XY_2d + OPOTTEMPPMDIFF_2d + HFDS + HFGEOU
- T_ADVECTION_XY_2d = horizontal advection
- OPOTTEMPPMDIFF_2d = neutral diffusion
- HFDS = net surface boundary heat flux
- HFGEOU = geothermal heat flux
- HFDS = net surface boundary heat flux entering the ocean = rsntds + rlntds + hfls + hfss + heat_pme + hfsifrazil
- More heat flux cross-checks
  - hfds = net_heat_coupler + hfsifrazil + heat_pme
  - heat_pme = heat_content_surfwater = heat_content_massin + heat_content_massout = heat_content_fprec + heat_content_cond + heat_content_vprec
    - hfrunoffds + hfevapds + hfrainds

Depth integrated heat budget

Here is an example 3d heat budget diagnostic for MOM.

- OPOTTEMPTEND = T_ADVECTION_XY + TH_TENDENCY_VERT_REMAP + OPOTTEMPDIFF + OPOTTEMPPMDIFF
  - BOUNDARY_FORCING_HEAT_TENDENCY + FRAZIL_HEAT_TENDENCY
- OPOTTEMPTEND = net tendency of heat as diagnosed in `MOM.F90`
- T_ADVECTION_XY = heating of a cell from lateral advection
- TH_TENDENCY_VERT_REMAP = heating of a cell from vertical remapping
- OPOTTEMPDIFF = heating of a cell from diabatic diffusion
- OPOTTEMPPMDIFF = heating of a cell from neutral diffusion
- BOUNDARY_FORCING_HEAT_TENDENCY = heating of cell from boundary fluxes
- FRAZIL_HEAT_TENDENCY = heating of cell from frazil

- TH_TENDENCY_VERT_REMAP has zero vertical sum, as it redistributes heat in vertical.
- OPOTTEMPDIFF has zero vertical sum, as it redistributes heat in the vertical.
- BOUNDARY_FORCING_HEAT_TENDENCY generally has 3d structure, with k > 1 contributions from penetrative shortwave, and from other fluxes for the case when layers are tiny, in which case MOM6 partitions tendencies into k > 1 layers.
- FRAZIL_HEAT_TENDENCY generally has 3d structure, since MOM6 frazil calculation checks the full ocean column.
- FRAZIL_HEAT_TENDENCY[k=@sum] = HFSIFRAZIL = column integrated frazil heating.
- HFDS = FRAZIL_HEAT_TENDENCY[k=@sum] + BOUNDARY_FORCING_HEAT_TENDENCY[k=@sum]

Here is an example 2d heat budget (depth summed) diagnostic for MOM.

- OPOTTEMPTEND_2d = T_ADVECTION_XY_2d + OPOTTEMPPMDIFF_2d + HFDS

Here is an example 3d salt budget diagnostic for MOM.

- OSALTTEND = S_ADVECTION_XY + SH_TENDENCY_VERT_REMAP + OSALTDIFF + OSALTPMDIFF
  - BOUNDARY_FORCING_SALT_TENDENCY
- OSALTTEND = net tendency of salt as diagnosed in `MOM.F90`
- S_ADVECTION_XY = salt convergence to cell from lateral advection
- SH_TENDENCY_VERT_REMAP = salt convergence to cell from vertical remapping
- OSALTDIFF = salt convergence to cell from diabatic diffusion
- OSALTPMDIFF = salt convergence to cell from neutral diffusion
- BOUNDARY_FORCING_SALT_TENDENCY = salt convergence to cell from boundary fluxes
- SH_TENDENCY_VERT_REMAP has zero vertical sum, as it redistributes salt in vertical.
- OSALTDIFF has zero vertical sum, as it redistributes salt in the vertical.
- BOUNDARY_FORCING_SALT_TENDENCY generally has 3d structure, with k > 1 contributions from the case when layers are tiny, in which case MOM6 partitions tendencies into k > 1 layers.
- SFDSI = BOUNDARY_FORCING_SALT_TENDENCY[k=@sum]

Here is an example 2d salt budget (depth summed) diagnostic for MOM.

- OSALTTEND_2d = S_ADVECTION_XY_2d + OSALTPMDIFF_2d + SFDSI (+ SALT_FLUX_RESTORE)

Type Documentation

type mom/**mom_control_struct**

Control structure for the MOM module, including the variables that describe the state of the ocean.

Type fields

- **%** **real** (* eta_av_bc [*) :: layer thickness [H ~> m or kg m-2]
- **%** **real** :: potential temperature [degC]
- **%** **real** :: salinity [ppt]
- **%** **real** :: zonal velocity component [L T-1 ~> m s-1]
- **%** **real** :: uh = u * h * dy at u grid points [H L2 T-1 ~> m3 s-1 or kg s-1]

- **% real** :: accumulated zonal thickness fluxes to advect tracers [H L2 ~> m3 or kg]
- **% real** :: meridional velocity [L T-1 ~> m s-1]
- **% real** :: vh = v * h * dx at v grid points [H L2 T-1 ~> m3 s-1 or kg s-1]
- **% real** :: accumulated meridional thickness fluxes to advect tracers [H L2 ~> m3 or kg]
- **% real** :: A running time integral of the sea surface height [T m ~> s m].
- **% real** :: time-averaged (over a forcing time step) sea surface height with a correction for the inverse barometer [m]
- **% real** :: free surface height or column mass time averaged over the last baroclinic dynamics time step [H ~> m or kg m-2]
- **% hml** *[real(:,:),pointer]* :: active mixed layer depth [Z ~> m]
- **% time_in_cycle** *[real]* :: The running time of the current time-stepping cycle in calls that step the dynamics, and also the length of the time integral of ssh_rint [T ~> s].
- **% time_in_thermo_cycle** *[real]* :: The running time of the current time-stepping cycle in calls that step the thermodynamics [T ~> s].
- **% g_in** *[type(ocean_grid_type)]* :: Input grid metric.
- **% g** *[type(ocean_grid_type),pointer]* :: Model grid metric.
- **% rotate_index** *[logical]* :: True if index map is rotated.
- **% gv** *[type(verticalgrid_type),pointer]* :: structure containing vertical grid info
- **% us** *[type(unit_scale_type),pointer]* :: structure containing various unit conversion factors
- **% tv** *[type(thermo_var_ptrs)]* :: structure containing pointers to available thermodynamic fields
- **% t_dyn_rel_adv** *[real]* :: The time of the dynamics relative to tracer advection and lateral mixing [T ~> s], or equivalently the elapsed time since advectively updating the tracers. t_dyn_rel_adv is invariably positive and may span multiple coupling timesteps.
- **% t_dyn_rel_thermo** *[real]* :: The time of the dynamics relative to diabatic processes and remapping [T ~> s]. t_dyn_rel_thermo can be negative or positive depending on whether the diabatic processes are applied before or after the dynamics and may span multiple coupling timesteps.
- **% t_dyn_rel_diag** *[real]* :: The time of the diagnostics relative to diabatic processes and remapping [T ~> s]. t_dyn_rel_diag is always positive, since the diagnostics must lag.
- **% preadv_h_stored** *[logical]* :: If true, the thicknesses from before the advective cycle have been stored for use in diagnostics.
- **% diag** *[type(diag_ctrl)]* :: structure to regulate diagnostic output timing
- **% visc** *[type(vertvisc_type)]* :: structure containing vertical viscosities, bottom drag viscosities, and related fields
- **% meke** *[type(meke_type),pointer]* :: structure containing fields related to the Mesoscale Eddy Kinetic Energy
- **% adiabatic** *[logical]* :: If true, there are no diapycnal mass fluxes, and no calls to routines to calculate or apply diapycnal fluxes.
- **% diabatic_first** *[logical]* :: If true, apply diabatic and thermodynamic processes before time stepping the dynamics.

- **% use_ale_algorithm** *[logical]* :: If true, use the ALE algorithm rather than layered isopycnal/stacked shallow water mode. This logical is set by calling the function useRegridding() from the MOM_regridding module.
- **% offline_tracer_mode** *[logical]* :: If true,
- **% time** *[type(time_type),pointer]* :: pointer to the ocean clock
- **% dt** *[real]* :: (baroclinic) dynamics time step [T ~> s]
- **% dt_therm** *[real]* :: thermodynamics time step [T ~> s]
- **% thermo_spans_coupling** *[logical]* :: If true, thermodynamic and tracer time steps can span multiple coupled time steps.
- **% nstep_tot** *[integer]* :: The total number of dynamic timesteps tcaaken so far in this run segment.
- **% count_calls** *[logical]* :: If true, count the calls to step_MOM, rather than the number of dynamics steps in nstep_tot.
- **% debug** *[logical]* :: If true, write verbose checksums for debugging purposes.
- **% ntrunc** *[integer]* :: number u,v truncations since last call to write_energy
- **% cont_stencil** *[integer]* :: The stencil for thickness from the continuity solver.
- **% do_dynamics** *[logical]* :: If false, does not call step_MOM_dyn_*. This is an undocumented run-time flag that is fragile.
- **% split** *[logical]* :: If true, use the split time stepping scheme.
- **% use_rk2** *[logical]* :: If true, use RK2 instead of RK3 in unsplit mode (i.e., no split between barotropic and baroclinic).
- **% thickness_diffuse** *[logical]* :: If true, diffuse interface height w/ a diffusivity KHTH.
- **% thickness_diffuse_first** *[logical]* :: If true, diffuse thickness before dynamics.
- **% mixedlayer_restrat** *[logical]* :: If true, use submesoscale mixed layer restratifying scheme.
- **% usemeke** *[logical]* :: If true, call the MEKE parameterization.
- **% usewaves** *[logical]* :: If true, update Stokes drift.
- **% use_p_surf_in_eos** *[logical]* :: If true, always include the surface pressure contributions in equation of state calculations.
- **% dtbt_reset_period** *[real]* :: The time interval between dynamic recalculation of the barotropic time step [s]. If this is negative dtbt is never calculated, and if it is 0, dtbt is calculated every step.
- **% dtbt_reset_interval** *[type(time_type)]* :: A time_time representation of dtbt_reset_period.
- **% dtbt_reset_time** *[type(time_type)]* :: The next time DTBT should be calculated.
- **% h_pre_dyn** *[real(:,:,:),pointer]* :: The thickness before the transports [H ~> m or kg m-2].
- **% t_pre_dyn** *[real(:,:,:),pointer]* :: Temperature before the transports [degC].
- **% s_pre_dyn** *[real(:,:,:),pointer]* :: Salinity before the transports [ppt].

- **% adp** *[type(accel_diag_ptrs)]* :: structure containing pointers to accelerations, for derived diagnostics (e.g., energy budgets)
- **% cdp** *[type(cont_diag_ptrs)]* :: structure containing pointers to continuity equation terms, for derived diagnostics (e.g., energy budgets)
- **% u_prev** *[real(:,:,:),pointer]* :: previous value of u stored for diagnostics [L T-1 ~> m s-1]
- **% v_prev** *[real(:,:,:),pointer]* :: previous value of v stored for diagnostics [L T-1 ~> m s-1]
- **% interp_p_surf** *[logical]* :: If true, linearly interpolate surface pressure over the coupling time step, using specified value at the end of the coupling step. False by default.
- **% p_surf_prev_set** *[logical]* :: If true, p_surf_prev has been properly set from a previous time-step or the ocean restart file. This is only valid when interp_p_surf is true.
- **% p_surf_prev** *[real(:,:),pointer]* :: surface pressure [R L2 T-2 ~> Pa] at end previous call to step_MOM
- **% p_surf_begin** *[real(:,:),pointer]* :: surface pressure [R L2 T-2 ~> Pa] at start of **step_MOM_dyn_**...
- **% p_surf_end** *[real(:,:),pointer]* :: surface pressure [R L2 T-2 ~> Pa] at end of **step_MOM_dyn_**...
- **% write_ic** *[logical]* :: If true, then the initial conditions will be written to file.
- **% ic_file** *[character(len=120)]* :: A file into which the initial conditions are written in a new run if SAVE_INITIAL_CONDS is true.
- **% calc_rho_for_sea_lev** *[logical]* :: If true, calculate rho to convert pressure to sea level.
- **% hmix** *[real]* :: Diagnostic mixed layer thickness over which to average surface tracer properties when a bulk mixed layer is not used [Z ~> m], or a negative value if a bulk mixed layer is being used.
- **% hfrz** *[real]* :: If HFrz > 0, the nominal depth over which melt potential is computed [Z ~> m]. The actual depth over which melt potential is computed is min(HFrz, OBLD), where OBLD is the boundary layer depth. If HFrz <= 0 (default), melt potential will not be computed.
- **% hmix_uv** *[real]* :: Depth scale over which to average surface flow to feedback to the coupler/driver [Z ~> m] when bulk mixed layer is not used, or a negative value if a bulk mixed layer is being used.
- **% check_bad_sfc_vals** *[logical]* :: If true, scan surface state for ridiculous values.
- **% bad_val_ssh_max** *[real]* :: Maximum SSH before triggering bad value message [Z ~> m].
- **% bad_val_sst_max** *[real]* :: Maximum SST before triggering bad value message [degC].
- **% bad_val_sst_min** *[real]* :: Minimum SST before triggering bad value message [degC].
- **% bad_val_sss_max** *[real]* :: Maximum SSS before triggering bad value message [ppt].
- **% bad_val_col_thick** *[real]* :: Minimum column thickness before triggering bad value message [Z ~> m].

- **% answers_2018** *[logical]* :: If true, use expressions for the surface properties that recover the answers from the end of 2018. Otherwise, use more appropriate expressions that differ at roundoff for non-Boussinsq cases.
- **% ids** *[type(mom_diag_ids)]* :: Handles used for diagnostics.
- **% transport_ids** *[type(transport_diag_ids)]* :: Handles used for transport diagnostics.
- **% sfc_ids** *[type(surface_diag_ids)]* :: Handles used for surface diagnostics.
- **% diag_pre_sync** *[type(diag_grid_storage)]* :: The grid (thicknesses) before remapping.
- **% diag_pre_dyn** *[type(diag_grid_storage)]* :: The grid (thicknesses) before dynamics.
- **% dyn_unsplit_csp** *[type(mom_dyn_unsplit_cs),pointer]* :: Pointer to the control structure used for the unsplit dynamics.
- **% dyn_unsplit_rk2_csp** *[type(mom_dyn_unsplit_rk2_cs),pointer]* :: Pointer to the control structure used for the unsplit RK2 dynamics.
- **% dyn_split_rk2_csp** *[type(mom_dyn_split_rk2_cs),pointer]* :: Pointer to the control structure used for the mode-split RK2 dynamics.
- **% thickness_diffuse_csp** *[type(thickness_diffuse_cs),pointer]* :: Pointer to the control structure used for the isopycnal height diffusive transport. This is also common referred to as Gent-McWilliams diffusion.
- **% mixedlayer_restrat_csp** *[type(mixedlayer_restrat_cs),pointer]* :: Pointer to the control structure used for the mixed layer restratification.
- **% set_visc_csp** *[type(set_visc_cs),pointer]* :: Pointer to the control structure used to set viscosities.
- **% diabatic_csp** *[type(diabatic_cs),pointer]* :: Pointer to the control structure for the diabatic driver.
- **% meke_csp** *[type(meke_cs),pointer]* :: Pointer to the control structure for the MEKE updates.
- **% varmix** *[type(varmix_cs),pointer]* :: Pointer to the control structure for the variable mixing module.
- **% barotropic_csp** *[type(barotropic_cs),pointer]* :: Pointer to the control structure for the barotropic module.
- **% tracer_reg** *[type(tracer_registry_type),pointer]* :: Pointer to the MOM tracer registry.
- **% tracer_adv_csp** *[type(tracer_advect_cs),pointer]* :: Pointer to the MOM tracer advection control structure.
- **% tracer_diff_csp** *[type(tracer_hor_diff_cs),pointer]* :: Pointer to the MOM along-isopycnal tracer diffusion control structure.
- **% tracer_flow_csp** *[type(tracer_flow_control_cs),pointer]* :: Pointer to the control structure that orchestrates the calling of tracer packages.
- **% update_obc_csp** *[type(update_obc_cs),pointer]* :: Pointer to the control structure for updating open boundary condition properties.
- **% obc** *[type(ocean_obc_type),pointer]* :: Pointer to the MOM open boundary condition type.

- **% sponge_csp** *[type(sponge_cs),pointer]* :: Pointer to the layered-mode sponge control structure.
- **% ale_sponge_csp** *[type(ale_sponge_cs),pointer]* :: Pointer to the ALE-mode sponge control structure.
- **% ale_csp** *[type(ale_cs),pointer]* :: Pointer to the Arbitrary Lagrangian Eulerian (ALE) vertical coordinate control structure.
- **% sum_output_csp** *[type(sum_output_cs),pointer]* :: Pointer to the globally summed output control structure.
- **% diagnostics_csp** *[type(diagnostics_cs),pointer]* :: Pointer to the MOM diagnostics control structure.
- **% offline_csp** *[type(offline_transport_cs),pointer]* :: Pointer to the offline tracer transport control structure.
- **% ensemble_ocean** *[logical]* :: if true, this run is part of a larger ensemble for the purpose of data assimilation or statistical analysis.
- **% odacs** *[type(oda_cs),pointer]* :: a pointer to the control structure for handling ensemble model state vectors and data assimilation increments and priors

type mom/**mom_diag_ids**

A structure with diagnostic IDs of the state variables.

Type fields

- **% id_u** *[integer,private]* :: 3-d state field diagnostic IDs
- **% id_v** *[integer,private]* :: 3-d state field diagnostic IDs
- **% id_h** *[integer,private]* :: 3-d state field diagnostic IDs
- **% id_ssh_inst** *[integer,private]* :: 2-d state field diagnotic ID

Function/Subroutine Documentation

subroutine mom/**step_mom**(*forces_in*, *fluxes_in*, *sfc_state*, *Time_start*, *time_int_in*, *CS*, *Waves*, *do_dynamics*, *do_thermodynamics*, *start_cycle*, *end_cycle*, *cycle_length*, *reset_therm*)

This subroutine orchestrates the time stepping of MOM. The adiabatic dynamics are stepped by calls to one of the **step_MOM_dyn_**... routines. The action of lateral processes on tracers occur in calls to advect_tracer and tracer_hordiff. Vertical mixing and possibly remapping occur inside of diabatic.

Parameters

- **forces_in** :: [inout] A structure with the driving mechanical forces
- **fluxes_in** :: [inout] A structure with pointers to themodynamic, tracer and mass exchange forcing fields
- **sfc_state** :: [inout] surface ocean state
- **time_start** :: [in] starting time of a segment, as a time type
- **time_int_in** :: [in] time interval covered by this run segment [s].
- **cs** :: control structure from initialize_MOM
- **waves** :: An optional pointer to a wave property CS
- **do_dynamics** :: [in] Present and false, do not do updates due to the dynamics.

- **do_thermodynamics** :: [in] Present and false, do not do updates due to the thermodynamics or remapping.
- **start_cycle** :: [in] This indicates whether this call is to be treated as the first call to step_MOM in a time-stepping cycle; missing is like true.
- **end_cycle** :: [in] This indicates whether this call is to be treated as the last call to step_MOM in a time-stepping cycle; missing is like true.
- **cycle_length** :: [in] The amount of time in a coupled time stepping cycle [s].
- **reset_therm** :: [in] This indicates whether the running sums of thermodynamic quantities should be reset. If missing, this is like start_cycle.

Call to *adjust_ssh_for_p_atm* *extract_surface_state* id_clock_diagnostics id_clock_dynamics id_clock_ocean id_clock_other id_clock_pass *mom_state_is_synchronized* *step_mom_dynamics* *step_mom_thermo* *step_mom_tracer_dyn*

subroutine mom/**step_mom_dynamics**(*forces*, *p_surf_begin*, *p_surf_end*, *dt*, *dt_thermo*, *bbl_time_int*, *CS*, *Time_local*, *Waves*)

Time step the ocean dynamics, including the momentum and continuity equations.

Parameters

- **forces** :: [in] A structure with the driving mechanical forces
- **p_surf_begin** :: A pointer (perhaps NULL) to the surface pressure at the beginning of this dynamic step, intent in [R L2 T-2 ~> Pa].
- **p_surf_end** :: A pointer (perhaps NULL) to the surface pressure at the end of this dynamic step, intent in [R L2 T-2 ~> Pa].
- **dt** :: [in] time interval covered by this call [T ~> s].
- **dt_thermo** :: [in] time interval covered by any updates that may span multiple dynamics steps [T ~> s].
- **bbl_time_int** :: [in] time interval over which updates to the bottom boundary layer properties will apply [T ~> s], or zero not to update the properties.
- **cs** :: control structure from initialize_MOM
- **time_local** :: [in] End time of a segment, as a time type
- **waves** :: Container for wave related parameters; the

Call to id_clock_bbl_visc id_clock_diagnostics id_clock_dynamics id_clock_ml_restrat id_clock_other id_clock_pass id_clock_thick_diff

Called from *step_mom*

subroutine mom/**step_mom_tracer_dyn**(*CS*, *G*, *GV*, *US*, *h*, *Time_local*)

step_MOM_tracer_dyn does tracer advection and lateral diffusion, bringing the tracers up to date with the changes in state due to the dynamics. Surface sources and sinks and remapping are handled via step_MOM_thermo.

Parameters

- **cs** :: [inout] control structure
- **g** :: [inout] ocean grid structure
- **gv** :: [in] ocean vertical grid structure

- **us** :: [in] A dimensional unit scaling type
- **h** :: [in] layer thicknesses after the transports [H ~> m or kg m-2]
- **time_local** :: [in] The model time at the end of the time step.

Call to `id_clock_diagnostics` `id_clock_other` `id_clock_pass` `id_clock_thermo` `id_clock_tracer`

Called from *step_mom*

subroutine mom/**step_mom_thermo**(*CS*, *G*, *GV*, *US*, *u*, *v*, *h*, *tv*, *fluxes*, *dtdia*, *Time_end_thermo*, *update_BBL*, *Waves*)

MOM_step_thermo orchestrates the thermodynamic time stepping and vertical remapping, via calls to diabatic (or adiabatic) and ALE_main.

Parameters

- **cs** :: [inout] Master MOM control structure
- **g** :: [inout] ocean grid structure
- **gv** :: [inout] ocean vertical grid structure
- **us** :: [in] A dimensional unit scaling type
- **u** :: [inout] zonal velocity [L T-1 ~> m s-1]
- **v** :: [inout] meridional velocity [L T-1 ~> m s-1]
- **h** :: [inout] layer thickness [H ~> m or kg m-2]
- **tv** :: [inout] A structure pointing to various thermodynamic variables
- **fluxes** :: [inout] pointers to forcing fields
- **dtdia** :: [in] The time interval over which to advance [T ~> s]
- **time_end_thermo** :: [in] End of averaging interval for thermo diags
- **update_bbl** :: [in] If true, calculate the bottom boundary layer properties.
- **waves** :: Container for wave related parameters

Call to `id_clock_adiabatic` `id_clock_ale` `id_clock_bbl_visc` `id_clock_diabatic` `id_clock_pass` `id_clock_thermo`

Called from *step_mom*

subroutine mom/**step_offline**(*forces*, *fluxes*, *sfc_state*, *Time_start*, *time_interval*, *CS*)

step_offline is the main driver for running tracers offline in MOM6. This has been primarily developed with ALE configurations in mind. Some work has been done in isopycnal configuration, but the work is very preliminary. Some more detail about this capability along with some of the subroutines called here can be found in tracers/MOM_offline_control.F90

Parameters

- **forces** :: [in] A structure with the driving mechanical forces
- **fluxes** :: [inout] pointers to forcing fields
- **sfc_state** :: [inout] surface ocean state
- **time_start** :: [in] starting time of a segment, as a time type
- **time_interval** :: [in] time interval
- **cs** :: control structure from initialize_MOM

Call to *adjust_ssh_for_p_atm* *extract_surface_state* id_clock_ale id_clock_offline_tracer

subroutine mom/**initialize_mom**(*Time*, *Time_init*, *param_file*, *dirs*, *CS*, *restart_CSp*, *Time_in*, *offline_tracer_mode*, *input_restart_file*, *diag_ptr*, *count_calls*, *tracer_flow_CSp*)

Initialize MOM, including memory allocation, setting up parameters and diagnostics, initializing the ocean state variables, and initializing subsidiary modules.

Parameters

- **time** :: [inout] model time, set in this routine
- **time_init** :: [in] The start time for the coupled model's calendar
- **param_file** :: [out] structure indicating parameter file to parse
- **dirs** :: [out] structure with directory paths
- **cs** :: pointer set in this routine to MOM control structure
- **restart_csp** :: pointer set in this routine to the restart control structure that will be used for MOM.
- **time_in** :: [in] time passed to MOM_initialize_state when model is not being started from a restart file
- **offline_tracer_mode** :: [out] True is returned if tracers are being run offline
- **input_restart_file** :: [in] If present, name of restart file to read
- **diag_ptr** :: A pointer set in this routine to the diagnostic regulatory structure
- **tracer_flow_csp** :: A pointer set in this routine to
- **count_calls** :: [in] If true, nstep_tot counts the number of calls to step_MOM instead of the number of dynamics timesteps.

Call to id_clock_init id_clock_mom_init id_clock_pass_init id_clock_unit_tests *mom_timing_init* *register_diags* *rotate_initial_state* *set_restart_fields*

subroutine mom/**finish_mom_initialization**(*Time*, *dirs*, *CS*, *restart_CSp*)

Finishes initializing MOM and writes out the initial conditions.

Parameters

- **time** :: [in] model time, used in this routine
- **dirs** :: [in] structure with directory paths
- **cs** :: pointer to MOM control structure
- **restart_csp** :: pointer to the restart control structure that will be used for MOM.

Call to id_clock_init

subroutine mom/**register_diags**(*Time*, *G*, *GV*, *US*, *IDs*, *diag*)

Register certain diagnostics.

Parameters

- **time** :: [in] current model time
- **g** :: [in] ocean grid structure
- **gv** :: [in] ocean vertical grid structure

- **us** :: [inout] A dimensional unit scaling type
- **ids** :: [inout] A structure with the diagnostic IDs.
- **diag** :: [inout] regulates diagnostic output

Called from *initialize_mom*

subroutine mom/**mom_timing_init**(*CS*)

Set up CPU clock IDs for timing various subroutines.

Parameters **cs** :: [in] control structure set up by initialize_MOM.

Call to `id_clock_adiabatic id_clock_ale id_clock_bbl_visc id_clock_continuity id_clock_diabatic id_clock_diagnostics id_clock_dynamics id_clock_ml_restrat id_clock_mom_init id_clock_ocean id_clock_offline_tracer id_clock_other id_clock_pass id_clock_pass_init id_clock_thermo id_clock_thick_diff id_clock_tracer id_clock_z_diag`

Called from *initialize_mom*

subroutine mom/**set_restart_fields**(*GV*, *US*, *param_file*, *CS*, *restart_CSp*)

Set the fields that are needed for bitwise identical restarting the time stepping scheme. In addition to those specified here directly, there may be fields related to the forcing or to the barotropic solver that are needed; these are specified in sub- routines that are called from this one.

This routine should be altered if there are any changes to the time stepping scheme. The CHECK_RESTART facility may be used to confirm that all needed restart fields have been included.

Parameters

- **gv** :: [inout] ocean vertical grid structure
- **us** :: [inout] A dimensional unit scaling type
- **param_file** :: [in] opened file for parsing to get parameters
- **cs** :: [in] control structure set up by initialize_MOM
- **restart_csp** :: pointer to the restart control structure that will be used for MOM.

Called from *initialize_mom*

subroutine mom/**adjust_ssh_for_p_atm**(*tv*, *G*, *GV*, *US*, *ssh*, *p_atm*, *use_EOS*)

Apply a correction to the sea surface height to compensate for the atmospheric pressure (the inverse barometer).

Parameters

- **tv** :: [in] A structure pointing to various thermodynamic variables
- **g** :: [in] ocean grid structure
- **gv** :: [in] ocean vertical grid structure
- **us** :: [in] A dimensional unit scaling type
- **ssh** :: [inout] time mean surface height [m]
- **p_atm** :: Ocean surface pressure [R L2 T-2 ~> Pa]
- **use_eos** :: [in] If true, calculate the density for the SSH correction using the equation of state.

Called from *step_mom step_offline*

subroutine mom/**extract_surface_state**(*CS*, *sfc_state_in*)
Set the surface (return) properties of the ocean model by setting the appropriate fields in sfc_state. Unused fields are set to NULL or are unallocated.

Parameters

- **cs** :: Master MOM control structure
- **sfc_state_in** :: [inout] transparent ocean surface state structure shared with the calling routine data in this structure is intent out.

Called from *step_mom step_offline*

subroutine mom/**rotate_initial_state**(*u_in*, *v_in*, *h_in*, *T_in*, *S_in*, *use_temperature*, *turns*, *u*, *v*, *h*, *T*, *S*)
Rotate initialization fields from input to rotated arrays.

Called from *initialize_mom*

function mom/**mom_state_is_synchronized**(*CS*, *adv_dyn*) *[logical]*
Return true if all phases of step_MOM are at the same point in time.

Parameters

- **cs** :: MOM control structure
- **adv_dyn** :: [in] If present and true, only check whether the advection is up-to-date with the dynamics.

Return **undefined** :: True if all phases of the update are synchronized.

Called from *step_mom*

subroutine mom/**get_mom_state_elements**(*CS*, *G*, *GV*, *US*, *C_p*, *C_p_scaled*, *use_temp*)
This subroutine offers access to values or pointers to other types from within the MOM_control_struct, allowing the MOM_control_struct to be opaque.

Parameters

- **cs** :: MOM control structure
- **g** :: structure containing metrics and grid info
- **gv** :: structure containing vertical grid info
- **us** :: A dimensional unit scaling type
- **c_p** :: [out] The heat capacity [J kg degC-1]
- **c_p_scaled** :: [out] The heat capacity in scaled units [Q degC-1 ~> J kg degC-1]
- **use_temp** :: [out] True if temperature is a state variable

subroutine mom/**get_ocean_stocks**(*CS*, *mass*, *heat*, *salt*, *on_PE_only*)
Find the global integrals of various quantities.

Parameters

- **cs** :: MOM control structure
- **heat** :: [out] The globally integrated integrated ocean heat [J].
- **salt** :: [out] The globally integrated integrated ocean salt [kg].
- **mass** :: [out] The globally integrated integrated ocean mass [kg].
- **on_pe_only** :: [in] If present and true, only sum on the local PE.

subroutine mom/**mom_end**(*CS*)
End of ocean model, including memory deallocation.

Parameters **cs** :: MOM control structure

13.1.2 mom_ale module reference

This module contains the main regridding routines.

More...

Data Types

ale_cs	ALE control structure.

Functions/Subroutines

ale_init()	This routine is typically called (from initialize_MOM in file MOM.F90) before the main
ale_register_diags()	Initialize diagnostics for the ALE module.
adjustgridforintegrity()	Crudely adjust (initial) grid for integrity.
ale_end()	End of regridding (memory deallocation).
ale_main()	Takes care of (1) building a new grid and (2) remapping all variables between the old grid
ale_main_offline()	Takes care of (1) building a new grid and (2) remapping all variables between the old grid
ale_offline_inputs()	Regrid/remap stored fields used for offline tracer integrations.
ale_offline_tracer_final()	Remaps all tracers from h onto h_target.
check_grid()	Check grid for negative thicknesses.
ale_build_grid()	Generates new grid.
ale_regrid_accelerated()	For a state-based coordinate, accelerate the process of regridding by repeatedly applying
remap_all_state_vars()	This routine takes care of remapping all variable between the old and the new grids.
ale_remap_scalar()	Remaps a single scalar between grids described by thicknesses h_src and h_dst.
ts_plm_edge_values()	Calculate edge values (top and bottom of layer) for T and S consistent with a PLM recons
ale_plm_edge_values()	Calculate edge values (top and bottom of layer) 3d scalar array.
ts_ppm_edge_values()	Calculate edge values (top and bottom of layer) for T and S consistent with a PPM recons
ale_initregridding()	Initializes regridding for the main ALE algorithm.
ale_getcoordinate()	Query the target coordinate interfaces positions.
ale_getcoordinateunits()	Query the target coordinate units.
ale_remap_init_conds()	Returns true if initial conditions should be regridded and remapped.
ale_update_regrid_weights()	Updates the weights for time filtering the new grid generated in regridding.
ale_updateverticalgridtype()	Update the vertical grid type with ALE information.
ale_writecoordinatefile()	Write the vertical coordinate information into a file.
ale_initthicknesstocoord()	Set h to coordinate values for fixed coordinate systems.

Detailed Description

Regridding comprises two steps:

1. Interpolation and creation of a new grid based on target interface densities (or any other criterion).
2. Remapping of quantities between old grid and new grid.

Original module written by Laurent White, 2008.06.09

Type Documentation

type mom_ale/**ale_cs**
ALE control structure.

Type fields

- **% remap_uv_using_old_alg** *[logical]* :: If true, uses the old "remapping via a delta z" method. If False, uses the new method that remaps between grids described by h.
- **% regrid_time_scale** *[real]* :: The time-scale used in blending between the current (old) grid and the target (new) grid [T ~> s].
- **% regridcs** *[type(regridding_cs)]* :: Regridding parameters and work arrays.
- **% remapcs** *[type(remapping_cs)]* :: Remapping parameters and work arrays.
- **% nk** *[integer]* :: Used only for queries, not directly by this module.
- **% remap_after_initialization** *[logical]* :: Indicates whether to regrid/remap after initializing the state.
- **% answers_2018** *[logical]* :: If true, use the order of arithmetic and expressions for remapping that recover the answers from the end of 2018. Otherwise, use more robust and accurate forms of mathematically equivalent expressions.
- **% show_call_tree** *[logical]* :: For debugging.
- **% diag** *[type(diag_ctrl),pointer]* :: structure to regulate output
- **% id_tracer_remap_tendency** *[integer(:),allocatable]* :: diagnostic id
- **% id_htracer_remap_tendency** *[integer(:),allocatable]* :: diagnostic id
- **% id_htracer_remap_tendency_2d** *[integer(:),allocatable]* :: diagnostic id
- **% do_tendency_diag** *[logical(:),allocatable]* :: flag for doing diagnostics
- **% id_dzregrid** *[integer]* :: diagnostic id
- **% id_u_preale** *[integer]* :: diagnostic id for zonal velocity before ALE.
- **% id_v_preale** *[integer]* :: diagnostic id for meridional velocity before ALE.
- **% id_h_preale** *[integer]* :: diagnostic id for layer thicknesses before ALE.
- **% id_t_preale** *[integer]* :: diagnostic id for temperatures before ALE.
- **% id_s_preale** *[integer]* :: diagnostic id for salinities before ALE.
- **% id_e_preale** *[integer]* :: diagnostic id for interface heights before ALE.
- **% id_vert_remap_h** *[integer]* :: diagnostic id for layer thicknesses used for remapping
- **% id_vert_remap_h_tendency** *[integer]* :: diagnostic id for layer thickness tendency due to ALE

Function/Subroutine Documentation

subroutine mom_ale/**ale_init**(*param_file*, *GV*, *US*, *max_depth*, *CS*)

This routine is typically called (from initialize_MOM in file MOM.F90) before the main time integration loop to initialize the regridding stuff. We read the MOM_input file to register the values of different regridding/remapping parameters.

Parameters

- **param_file** :: [in] Parameter file
- **gv** :: [in] Ocean vertical grid structure
- **us** :: [in] A dimensional unit scaling type
- **max_depth** :: [in] The maximum depth of the ocean [Z ~> m].
- **cs** :: Module control structure

Call to *ale_initregridding*

subroutine mom_ale/**ale_register_diags**(*Time*, *G*, *GV*, *US*, *diag*, *CS*)

Initialize diagnostics for the ALE module.

Parameters

- **time** :: [in] Time structure
- **g** :: [in] Grid structure
- **us** :: [in] A dimensional unit scaling type
- **gv** :: [in] Ocean vertical grid structure
- **diag** :: [in] Diagnostics control structure
- **cs** :: Module control structure

subroutine mom_ale/**adjustgridforintegrity**(*CS*, *G*, *GV*, *h*)

Crudely adjust (initial) grid for integrity. This routine is typically called (from initialize_MOM in file MOM.F90) before the main time integration loop to initialize the regridding stuff. We read the MOM_input file to register the values of different regridding/remapping parameters.

Parameters

- **cs** :: Regridding parameters and options
- **g** :: [in] Ocean grid informations
- **gv** :: [in] Ocean vertical grid structure
- **h** :: [inout] Current 3D grid thickness that are to be adjusted [H ~> m or kg-2]

subroutine mom_ale/**ale_end**(*CS*)

End of regridding (memory deallocation). This routine is typically called (from MOM_end in file MOM.F90) after the main time integration loop to deallocate the regridding stuff.

Parameters **cs** :: module control structure

subroutine mom_ale/**ale_main**(*G*, *GV*, *US*, *h*, *u*, *v*, *tv*, *Reg*, *CS*, *OBC*, *dt*, *frac_shelf_h*)

Takes care of (1) building a new grid and (2) remapping all variables between the old grid and the new grid. The creation of the new grid can be based on z coordinates, target interface densities, sigma coordinates or any arbitrary coordinate system.

Parameters

- **g** :: [in] Ocean grid informations

- **gv** :: [in] Ocean vertical grid structure
- **us** :: [in] A dimensional unit scaling type
- **h** :: [inout] Current 3D grid obtained after the last time step [H ~> m or kg m-2]
- **u** :: [inout] Zonal velocity field [L T-1 ~> m s-1]
- **v** :: [inout] Meridional velocity field [L T-1 ~> m s-1]
- **tv** :: [inout] Thermodynamic variable structure
- **reg** :: Tracer registry structure
- **cs** :: Regridding parameters and options
- **obc** :: Open boundary structure
- **dt** :: [in] Time step between calls to ALE_main [T ~> s]
- **frac_shelf_h** :: Fractional ice shelf coverage

Call to *ale_update_regrid_weights check_grid remap_all_state_vars*

subroutine mom_ale/**ale_main_offline**(*G*, *GV*, *h*, *tv*, *Reg*, *CS*, *OBC*, *dt*)

Takes care of (1) building a new grid and (2) remapping all variables between the old grid and the new grid. The creation of the new grid can be based on z coordinates, target interface densities, sigma coordinates or any arbitrary coordinate system.

Parameters

- **g** :: [in] Ocean grid informations
- **gv** :: [in] Ocean vertical grid structure
- **h** :: [inout] Current 3D grid obtained after the last time step [H ~> m or kg-2]
- **tv** :: [inout] Thermodynamic variable structure
- **reg** :: Tracer registry structure
- **cs** :: Regridding parameters and options
- **obc** :: Open boundary structure
- **dt** :: [in] Time step between calls to ALE_main [T ~> s]

Call to *ale_update_regrid_weights check_grid remap_all_state_vars*

subroutine mom_ale/**ale_offline_inputs**(*CS*, *G*, *GV*, *h*, *tv*, *Reg*, *uhtr*, *vhtr*, *Kd*, *debug*, *OBC*)

Regrid/remap stored fields used for offline tracer integrations. These input fields are assumed to have the same layer thicknesses at the end of the last offline interval (which should be a Zstar grid). This routine builds a grid on the runtime specified vertical coordinate.

Parameters

- **cs** :: Regridding parameters and options
- **g** :: [in] Ocean grid informations
- **gv** :: [in] Ocean vertical grid structure
- **h** :: [inout] Layer thicknesses
- **tv** :: [inout] Thermodynamic variable structure
- **reg** :: Tracer registry structure
- **uhtr** :: [inout] Zonal mass fluxes

- **vhtr** :: [inout] Meridional mass fluxes
- **kd** :: [inout] Input diffusivites
- **debug** :: [in] If true, then turn checksums
- **obc** :: Open boundary structure

Call to *ale_remap_scalar check_grid remap_all_state_vars*

subroutine mom_ale/**ale_offline_tracer_final**(*G*, *GV*, *h*, *tv*, *h_target*, *Reg*, *CS*, *OBC*)
Remaps all tracers from h onto h_target. This is intended to be called when tracers are done offline. In the case where transports don't quite conserve, we still want to make sure that layer thicknesses offline do not drift too far away from the online model.

Parameters

- **g** :: [in] Ocean grid informations
- **gv** :: [in] Ocean vertical grid structure
- **h** :: [inout] Current 3D grid obtained after the last time step [H ~> m or kg-2]
- **tv** :: [inout] Thermodynamic variable structure
- **h_target** :: [inout] Current 3D grid obtained after last time step [H ~> m or kg-2]
- **reg** :: Tracer registry structure
- **cs** :: Regridding parameters and options
- **obc** :: Open boundary structure

Call to *check_grid remap_all_state_vars*

subroutine mom_ale/**check_grid**(*G*, *GV*, *h*, *threshold*)
Check grid for negative thicknesses.

Parameters

- **g** :: [in] Ocean grid structure
- **gv** :: [in] Ocean vertical grid structure
- **h** :: [in] Current 3D grid obtained after the last time step [H ~> m or kg m-2]
- **threshold** :: [in] Value below which to flag issues, [H ~> m or kg m-2]

Called from *ale_main ale_main_offline ale_offline_inputs ale_offline_tracer_final*

subroutine mom_ale/**ale_build_grid**(*G*, *GV*, *regridCS*, *remapCS*, *h*, *tv*, *debug*, *frac_shelf_h*)
Generates new grid.

Parameters

- **g** :: [in] Ocean grid structure
- **gv** :: [in] Ocean vertical grid structure
- **regridcs** :: [in] Regridding parameters and options
- **remapcs** :: [in] Remapping parameters and options
- **tv** :: [inout] Thermodynamical variable structure
- **h** :: [inout] Current 3D grid obtained after the last time step [H ~> m or kg-2]
- **debug** :: [in] If true, show the call tree

- **frac_shelf_h** :: Fractional ice shelf coverage

subroutine mom_ale/**ale_regrid_accelerated**(*CS*, *G*, *GV*, *h*, *tv*, *n*, *u*, *v*, *OBC*, *Reg*, *dt*, *dzRegrid*, *initial*)

For a state-based coordinate, accelerate the process of regridding by repeatedly applying the grid calculation algorithm.

Parameters

- **cs** :: ALE control structure
- **g** :: [inout] Ocean grid
- **gv** :: [in] Vertical grid
- **h** :: [inout] Original thicknesses [H ~> m or kg-2]
- **tv** :: [inout] Thermo vars (T/S/EOS)
- **n** :: [in] Number of times to regrid
- **u** :: [inout] Zonal velocity [L T-1 ~> m s-1]
- **v** :: [inout] Meridional velocity [L T-1 ~> m s-1]
- **obc** :: Open boundary structure
- **reg** :: Tracer registry to remap onto new grid
- **dt** :: [in] Model timestep to provide a timescale for regridding [T ~> s]
- **dzregrid** :: [inout] Final change in interface positions
- **initial** :: [in] Whether we're being called from an initialization routine (and expect diagnostics to work)

Call to *ale_update_regrid_weights* *remap_all_state_vars*

subroutine mom_ale/**remap_all_state_vars**(*CS_remapping*, *CS_ALE*, *G*, *GV*, *h_old*, *h_new*, *Reg*, *OBC*, *dxInterface*, *u*, *v*, *debug*, *dt*)

This routine takes care of remapping all variable between the old and the new grids. When velocity components need to be remapped, thicknesses at velocity points are taken to be arithmetic averages of tracer thicknesses. This routine is called during initialization of the model at time=0, to remap initiali conditions to the model grid. It is also called during a time step to update the state.

Parameters

- **cs_remapping** :: [in] Remapping control structure
- **cs_ale** :: [in] ALE control structure
- **g** :: [in] Ocean grid structure
- **gv** :: [in] Ocean vertical grid structure
- **h_old** :: [in] Thickness of source grid [H ~> m or kg-2]
- **h_new** :: [in] Thickness of destination grid [H ~> m or kg-2]
- **reg** :: Tracer registry structure
- **obc** :: Open boundary structure
- **dxinterface** :: [in] Change in interface position
- **u** :: [inout] Zonal velocity [L T-1 ~> m s-1]
- **v** :: [inout] Meridional velocity [L T-1 ~> m s-1]

- **debug** :: [in] If true, show the call tree
- **dt** :: [in] time step for diagnostics [T ~> s]

Called from *ale_main* *ale_main_offline* *ale_offline_inputs* *ale_offline_tracer_final* *ale_regrid_accelerated*

subroutine mom_ale/**ale_remap_scalar**(*CS*, *G*, *GV*, *nk_src*, *h_src*, *s_src*, *h_dst*, *s_dst*, *all_cells*, *old_remap*, *answers_2018*)

Remaps a single scalar between grids described by thicknesses h_src and h_dst. h_dst must be dimensioned as a model array with GVke layers while h_src can have an arbitrary number of layers specified by nk_src.

Parameters

- **cs** :: [in] Remapping control structure
- **g** :: [in] Ocean grid structure
- **gv** :: [in] Ocean vertical grid structure
- **nk_src** :: [in] Number of levels on source grid
- **h_src** :: [in] Level thickness of source grid [H ~> m or kg-2]
- **s_src** :: [in] Scalar on source grid
- **h_dst** :: [in] Level thickness of destination grid [H ~> m or kg-2]
- **s_dst** :: [inout] Scalar on destination grid
- **all_cells** :: [in] If false, only reconstruct for non-vanished cells. Use all vanished layers otherwise (default).
- **old_remap** :: [in] If true, use the old "remapping_core_w" method, otherwise use "remapping_core_h".
- **answers_2018** :: [in] If true, use the order of arithmetic and expressions that recover the answers for remapping from the end of 2018. Otherwise, use more robust forms of the same expressions.

Called from *ale_offline_inputs*

subroutine mom_ale/**ts_plm_edge_values**(*CS*, *S_t*, *S_b*, *T_t*, *T_b*, *G*, *GV*, *tv*, *h*, *bdry_extrap*)

Calculate edge values (top and bottom of layer) for T and S consistent with a PLM reconstruction in the vertical direction. Boundary reconstructions are PCM unless bdry_extrap is true.

Parameters

- **g** :: [in] ocean grid structure
- **gv** :: [in] Ocean vertical grid structure
- **cs** :: [inout] module control structure
- **s_t** :: [inout] Salinity at the top edge of each layer
- **s_b** :: [inout] Salinity at the bottom edge of each layer
- **t_t** :: [inout] Temperature at the top edge of each layer
- **t_b** :: [inout] Temperature at the bottom edge of each layer
- **tv** :: [in] thermodynamics structure
- **h** :: [in] layer thickness [H ~> m or kg m-2]
- **bdry_extrap** :: [in] If true, use high-order boundary extrapolation within boundary cells

Call to *ale_plm_edge_values*

subroutine mom_ale/**ale_plm_edge_values**(*CS*, *G*, *GV*, *h*, *Q*, *bdry_extrap*, *Q_t*, *Q_b*)
Calculate edge values (top and bottom of layer) 3d scalar array. Boundary reconstructions are PCM unless bdry_extrap is true.

Parameters

- **cs** :: [in] module control structure
- **g** :: [in] ocean grid structure
- **gv** :: [in] Ocean vertical grid structure
- **h** :: [in] layer thickness [H ~> m or kg m-2]
- **q** :: [in] 3d scalar array
- **bdry_extrap** :: [in] If true, use high-order boundary extrapolation within boundary cells
- **q_t** :: [inout] Scalar at the top edge of each layer
- **q_b** :: [inout] Scalar at the bottom edge of each layer

Called from *ts_plm_edge_values*

subroutine mom_ale/**ts_ppm_edge_values**(*CS*, *S_t*, *S_b*, *T_t*, *T_b*, *G*, *GV*, *tv*, *h*, *bdry_extrap*)
Calculate edge values (top and bottom of layer) for T and S consistent with a PPM reconstruction in the vertical direction. Boundary reconstructions are PCM unless bdry_extrap is true.

Parameters

- **g** :: [in] ocean grid structure
- **gv** :: [in] Ocean vertical grid structure
- **cs** :: [inout] module control structure
- **s_t** :: [inout] Salinity at the top edge of each layer
- **s_b** :: [inout] Salinity at the bottom edge of each layer
- **t_t** :: [inout] Temperature at the top edge of each layer
- **t_b** :: [inout] Temperature at the bottom edge of each layer
- **tv** :: [in] thermodynamics structure
- **h** :: [in] layer thicknesses [H ~> m or kg m-2]
- **bdry_extrap** :: [in] If true, use high-order boundary extrapolation within boundary cells

subroutine mom_ale/**ale_initregridding**(*GV*, *US*, *max_depth*, *param_file*, *mdl*, *regridCS*)
Initializes regridding for the main ALE algorithm.

Parameters

- **gv** :: [in] Ocean vertical grid structure
- **us** :: [in] A dimensional unit scaling type
- **max_depth** :: [in] The maximum depth of the ocean [Z ~> m].
- **param_file** :: [in] parameter file
- **mdl** :: [in] Name of calling module
- **regridcs** :: [out] Regridding parameters and work arrays

Called from *ale_init*

function mom_ale/**ale_getcoordinate**(*CS*) *[real]*
Query the target coordinate interfaces positions.

Parameters **cs** :: module control structure

function mom_ale/**ale_getcoordinateunits**(*CS*) *[character(len=20)]*
Query the target coordinate units.

Parameters **cs** :: module control structure

function mom_ale/**ale_remap_init_conds**(*CS*) *[logical]*
Returns true if initial conditions should be regridded and remapped.

Parameters **cs** :: module control structure

subroutine mom_ale/**ale_update_regrid_weights**(*dt*, *CS*)
Updates the weights for time filtering the new grid generated in regridding.

Parameters

- **dt** :: [in] Time-step used between ALE calls [T ~> s]
- **cs** :: ALE control structure

Called from *ale_main ale_main_offline ale_regrid_accelerated*

subroutine mom_ale/**ale_updateverticalgridtype**(*CS*, *GV*)
Update the vertical grid type with ALE information. This subroutine sets information in the verticalGrid_type to be consistent with the use of ALE mode.

Parameters

- **cs** :: ALE control structure
- **gv** :: vertical grid information

subroutine mom_ale/**ale_writecoordinatefile**(*CS*, *GV*, *directory*)
Write the vertical coordinate information into a file. This subroutine writes out a file containing any available data related to the vertical grid used by the MOM ocean model when in ALE mode.

Parameters

- **cs** :: module control structure
- **gv** :: [in] ocean vertical grid structure
- **directory** :: [in] directory for writing grid info

subroutine mom_ale/**ale_initthicknesstocoord**(*CS*, *G*, *GV*, *h*)
Set h to coordinate values for fixed coordinate systems.

Parameters

- **cs** :: [inout] module control structure
- **g** :: [in] module grid structure
- **gv** :: [in] Ocean vertical grid structure
- **h** :: [out] layer thickness [H ~> m or kg m-2]

13.1.3 mom_eos module reference

Provides subroutines for quantities specific to the equation of state.

More...

Data Types

eos_type	A control structure for the equation of state.

Functions/Subroutines

calculate_density_scalar()	Calls the appropriate subroutine to calculate density of sea water for sc
calculate_stanley_density_scalar()	Calls the appropriate subroutine to calculate density of sea water for sc
calculate_density_array()	Calls the appropriate subroutine to calculate the density of sea water fo
calculate_stanley_density_array()	Calls the appropriate subroutine to calculate the density of sea water fo
calculate_density_1d()	Calls the appropriate subroutine to calculate the density of sea water fo
calculate_stanley_density_1d()	Calls the appropriate subroutine to calculate the density of sea water fo
calculate_spec_vol_array()	Calls the appropriate subroutine to calculate the specific volume of sea
calc_spec_vol_scalar()	Calls the appropriate subroutine to calculate specific volume of sea wat
calc_spec_vol_1d()	Calls the appropriate subroutine to calculate the specific volume of sea
calculate_tfreeze_scalar()	Calls the appropriate subroutine to calculate the freezing point for scala
calculate_tfreeze_array()	Calls the appropriate subroutine to calculate the freezing point for a 1-I
calculate_density_derivs_array()	Calls the appropriate subroutine to calculate density derivatives for 1-D
calculate_density_derivs_1d()	Calls the appropriate subroutine to calculate density derivatives for 1-D
calculate_density_derivs_scalar()	Calls the appropriate subroutines to calculate density derivatives by pro
calculate_density_second_derivs_array()	Calls the appropriate subroutine to calculate density second derivatives
calculate_density_second_derivs_scalar()	Calls the appropriate subroutine to calculate density second derivatives
calculate_spec_vol_derivs_array()	Calls the appropriate subroutine to calculate specific volume derivative
calc_spec_vol_derivs_1d()	Calls the appropriate subroutine to calculate specific volume derivative
calculate_compress_array()	Calls the appropriate subroutine to calculate the density and compressil
calculate_compress_scalar()	Calculate density and compressibility for a scalar.
eos_domain()	This subroutine returns a two point integer array indicating the domain
analytic_int_specific_vol_dp()	Calls the appropriate subroutine to calculate analytical and nearly-analy
analytic_int_density_dz()	This subroutine calculates analytical and nearly-analytical integrals of p
query_compressible()	Returns true if the equation of state is compressible (i.e.
eos_init()	Initializes EOS_type by allocating and reading parameters.
eos_manual_init()	Manually initialized an EOS type (intended for unit testing of routines
eos_allocate()	Allocates EOS_type.
eos_end()	Deallocates EOS_type.
eos_use_linear()	Set equation of state structure (EOS) to linear with given coefficients.
convert_temp_salt_for_teos10()	Convert T&S to Absolute Salinity and Conservative Temperature if usi
eos_quadrature()	Return value of EOS_quadrature.
extract_member_eos()	Extractor routine for the EOS type if the members need to be accessed

Detailed Description

The MOM_EOS module is a wrapper for various equations of state (e.g. Linear, Wright, UNESCO) and provides a uniform interface to the rest of the model independent of which equation of state is being used.

Type Documentation

type mom_eos/**eos_type**

A control structure for the equation of state.

Type fields

- **% form_of_eos** *[integer]* :: The equation of state to use.
- **% form_of_tfreeze** *[integer]* :: The expression for the potential temperature of the freezing point.
- **% eos_quadrature** *[logical]* :: If true, always use the generic (quadrature) code for the integrals of density.
- **% compressible** *[logical]* :: If true, in situ density is a function of pressure.
- **% rho_t0_s0** *[real]* :: The density at T=0, S=0 [kg m-3].
- **% drho_dt** *[real]* :: The partial derivative of density with temperature [kg m-3 degC-1].
- **% drho_ds** *[real]* :: The partial derivative of density with salinity [kg m-3 ppt-1].
- **% tfr_s0_p0** *[real]* :: The freezing potential temperature at S=0, P=0 [degC].
- **% dtfr_ds** *[real]* :: The derivative of freezing point with salinity [degC ppt-1].
- **% dtfr_dp** *[real]* :: The derivative of freezing point with pressure [degC Pa-1].
- **% m_to_z** *[real]* :: A constant that translates distances in meters to the units of depth.
- **% kg_m3_to_r** *[real]* :: A constant that translates kilograms per meter cubed to the units of density.
- **% r_to_kg_m3** *[real]* :: A constant that translates the units of density to kilograms per meter cubed.
- **% rl2_t2_to_pa** *[real]* :: Convert pressures from R L2 T-2 to Pa.
- **% l_t_to_m_s** *[real]* :: Convert lateral velocities from L T-1 to m s-1.

Function/Subroutine Documentation

subroutine mom_eos/**calculate_density_scalar**(*T*, *S*, *pressure*, *rho*, *EOS*, *rho_ref*, *scale*)

Calls the appropriate subroutine to calculate density of sea water for scalar inputs. If rho_ref is present, the anomaly with respect to rho_ref is returned. The pressure and density can be rescaled with the US. If both the US and scale arguments are present the density scaling uses the product of the two scaling factors.

Parameters

- **t** :: [in] Potential temperature referenced to the surface [degC]
- **s** :: [in] Salinity [ppt]
- **pressure** :: [in] Pressure [Pa] or [R L2 T-2 ~> Pa]
- **rho** :: [out] Density (in-situ if pressure is local) [kg m-3] or [R ~> kg m-3]

- **eos** :: Equation of state structure
- **rho_ref** :: [in] A reference density [kg m-3]
- **scale** :: [in] A multiplicative factor by which to scale density in combination with scaling given by US [various]

Call to `eos_linear` `eos_nemo` `eos_teos10` `eos_unesco` `eos_wright`

subroutine mom_eos/**calculate_stanley_density_scalar**(*T*, *S*, *pressure*, *Tvar*, *TScov*, *Svar*, *rho*, *EOS*, *rho_ref*, *scale*)

Calls the appropriate subroutine to calculate density of sea water for scalar inputs including the variance of T, S and covariance of T-S. The calculation uses only the second order correction in a series as discussed in Stanley et al., 2020. If rho_ref is present, the anomaly with respect to rho_ref is returned. The density can be rescaled using rho_ref.

Parameters

- **t** :: [in] Potential temperature referenced to the surface [degC]
- **s** :: [in] Salinity [ppt]
- **tvar** :: [in] Variance of potential temperature referenced to the surface [degC2]
- **tscov** :: [in] Covariance of potential temperature and salinity [degC ppt]
- **svar** :: [in] Variance of salinity [ppt2]
- **pressure** :: [in] Pressure [Pa]
- **rho** :: [out] Density (in-situ if pressure is local) [kg m-3] or [R ~> kg m-3]
- **eos** :: Equation of state structure
- **rho_ref** :: [in] A reference density [kg m-3].
- **scale** :: [in] A multiplicative factor by which to scale density from kg m-3 to the desired units [R m3 kg-1]

Call to `eos_linear` `eos_teos10` `eos_wright`

subroutine mom_eos/**calculate_density_array**(*T*, *S*, *pressure*, *rho*, *start*, *npts*, *EOS*, *rho_ref*, *scale*)

Calls the appropriate subroutine to calculate the density of sea water for 1-D array inputs. If rho_ref is present, the anomaly with respect to rho_ref is returned.

Parameters

- **t** :: [in] Potential temperature referenced to the surface [degC]
- **s** :: [in] Salinity [ppt]
- **pressure** :: [in] Pressure [Pa] or [R L2 T-2 ~> Pa]
- **rho** :: [inout] Density (in-situ if pressure is local) [kg m-3] or [R ~> kg m-3]
- **start** :: [in] Start index for computation
- **npts** :: [in] Number of point to compute
- **eos** :: Equation of state structure
- **rho_ref** :: [in] A reference density [kg m-3]
- **scale** :: [in] A multiplicative factor by which to scale density in combination with scaling given by US [various]

Call to `eos_linear` `eos_nemo` `eos_teos10` `eos_unesco` `eos_wright`

Called from *calculate_density_1d*

subroutine mom_eos/**calculate_stanley_density_array**(*T*, *S*, *pressure*, *Tvar*, *TScov*, *Svar*, *rho*, *start*, *npts*, *EOS*, *rho_ref*, *scale*)

Calls the appropriate subroutine to calculate the density of sea water for 1-D array inputs including the variance of T, S and covariance of T-S. The calculation uses only the second order correction in a series as discussed in Stanley et al., 2020. If rho_ref is present, the anomaly with respect to rho_ref is returned.

Parameters

- **t** :: [in] Potential temperature referenced to the surface [degC]
- **s** :: [in] Salinity [ppt]
- **pressure** :: [in] Pressure [Pa]
- **tvar** :: [in] Variance of potential temperature referenced to the surface [degC2]
- **tscov** :: [in] Covariance of potential temperature and salinity [degC ppt]
- **svar** :: [in] Variance of salinity [ppt2]
- **rho** :: [inout] Density (in-situ if pressure is local) [kg m-3]
- **start** :: [in] Start index for computation
- **npts** :: [in] Number of point to compute
- **eos** :: Equation of state structure
- **rho_ref** :: [in] A reference density [kg m-3].
- **scale** :: [in] A multiplicative factor by which to scale density from kg m-3 to the desired units [R m3 kg-1]

Call to eos_linear eos_teos10 eos_wright

subroutine mom_eos/**calculate_density_1d**(*T*, *S*, *pressure*, *rho*, *EOS*, *dom*, *rho_ref*, *scale*)

Calls the appropriate subroutine to calculate the density of sea water for 1-D array inputs, potentially limiting the domain of indices that are worked on. If rho_ref is present, the anomaly with respect to rho_ref is returned.

Parameters

- **t** :: [in] Potential temperature referenced to the surface [degC]
- **s** :: [in] Salinity [ppt]
- **pressure** :: [in] Pressure [R L2 T-2 ~> Pa]
- **rho** :: [inout] Density (in-situ if pressure is local) [R ~> kg m-3]
- **eos** :: Equation of state structure
- **dom** :: [in] The domain of indices to work on, taking into account that arrays start at 1.
- **rho_ref** :: [in] A reference density [kg m-3]
- **scale** :: [in] A multiplicative factor by which to scale density in combination with scaling given by US [various]

Call to *calculate_density_array*

subroutine mom_eos/**calculate_stanley_density_1d**(*T*, *S*, *pressure*, *Tvar*, *TScov*, *Svar*, *rho*, *EOS*, *dom*, *rho_ref*, *scale*)

Calls the appropriate subroutine to calculate the density of sea water for 1-D array inputs including the variance of T, S and covariance of T-S, potentially limiting the domain of indices that are worked on. The calculation

uses only the second order correction in a series as discussed in Stanley et al., 2020. If rho_ref is present, the anomaly with respect to rho_ref is returned.

Parameters

- **t** :: [in] Potential temperature referenced to the surface [degC]
- **s** :: [in] Salinity [ppt]
- **pressure** :: [in] Pressure [R L2 T-2 ~> Pa]
- **tvar** :: [in] Variance of potential temperature [degC2]
- **tscov** :: [in] Covariance of potential temperature and salinity [degC ppt]
- **svar** :: [in] Variance of salinity [ppt2]
- **rho** :: [inout] Density (in-situ if pressure is local) [R ~> kg m-3]
- **eos** :: Equation of state structure
- **dom** :: [in] The domain of indices to work on, taking into account that arrays start at 1.
- **rho_ref** :: [in] A reference density [kg m-3]
- **scale** :: [in] A multiplicative factor by which to scale density in combination with scaling given by US [various]

Call to `eos_linear` `eos_teos10` `eos_wright`

subroutine mom_eos/**calculate_spec_vol_array**(*T*, *S*, *pressure*, *specvol*, *start*, *npts*, *EOS*, *spv_ref*, *scale*)

Calls the appropriate subroutine to calculate the specific volume of sea water for 1-D array inputs.

Parameters

- **t** :: [in] potential temperature relative to the surface [degC]
- **s** :: [in] salinity [ppt]
- **pressure** :: [in] pressure [Pa]
- **specvol** :: [inout] in situ specific volume [kg m-3]
- **start** :: [in] the starting point in the arrays.
- **npts** :: [in] the number of values to calculate.
- **eos** :: Equation of state structure
- **spv_ref** :: [in] A reference specific volume [m3 kg-1]
- **scale** :: [in] A multiplicative factor by which to scale specific volume in combination with scaling given by US [various]

Call to `eos_linear` `eos_nemo` `eos_teos10` `eos_unesco` `eos_wright`

Called from *calc_spec_vol_1d* *calc_spec_vol_scalar*

subroutine mom_eos/**calc_spec_vol_scalar**(*T*, *S*, *pressure*, *specvol*, *EOS*, *spv_ref*, *scale*)

Calls the appropriate subroutine to calculate specific volume of sea water for scalar inputs.

Parameters

- **t** :: [in] Potential temperature referenced to the surface [degC]
- **s** :: [in] Salinity [ppt]
- **pressure** :: [in] Pressure [Pa] or [R L2 T-2 ~> Pa]

- **specvol** :: [out] In situ? specific volume [m3 kg-1] or [R-1 ~> m3 kg-1]
- **eos** :: Equation of state structure
- **spv_ref** :: [in] A reference specific volume [m3 kg-1] or [R-1 m3 kg-1]
- **scale** :: [in] A multiplicative factor by which to scale specific volume in combination with scaling given by US [various]

Call to *calculate_spec_vol_array*

subroutine mom_eos/**calc_spec_vol_1d**(*T*, *S*, *pressure*, *specvol*, *EOS*, *dom*, *spv_ref*, *scale*)

Calls the appropriate subroutine to calculate the specific volume of sea water for 1-D array inputs, potentially limiting the domain of indices that are worked on.

Parameters

- **t** :: [in] Potential temperature referenced to the surface [degC]
- **s** :: [in] Salinity [ppt]
- **pressure** :: [in] Pressure [R L2 T-2 ~> Pa]
- **specvol** :: [inout] In situ specific volume [R-1 ~> m3 kg-1]
- **eos** :: Equation of state structure
- **dom** :: [in] The domain of indices to work on, taking into account that arrays start at 1.
- **spv_ref** :: [in] A reference specific volume [R-1 ~> m3 kg-1]
- **scale** :: [in] A multiplicative factor by which to scale output specific volume in combination with scaling given by US [various]

Call to *calculate_spec_vol_array*

subroutine mom_eos/**calculate_tfreeze_scalar**(*S*, *pressure*, *T_fr*, *EOS*, *pres_scale*)

Calls the appropriate subroutine to calculate the freezing point for scalar inputs.

Parameters

- **s** :: [in] Salinity [ppt]
- **pressure** :: [in] Pressure [Pa] or [other]
- **t_fr** :: [out] Freezing point potential temperature referenced to the surface [degC]
- **eos** :: Equation of state structure
- **pres_scale** :: [in] A multiplicative factor to convert pressure into Pa

Call to `tfreeze_linear` `tfreeze_millero` `tfreeze_teos10`

subroutine mom_eos/**calculate_tfreeze_array**(*S*, *pressure*, *T_fr*, *start*, *npts*, *EOS*, *pres_scale*)

Calls the appropriate subroutine to calculate the freezing point for a 1-D array.

Parameters

- **s** :: [in] Salinity [ppt]
- **pressure** :: [in] Pressure [Pa] or [other]
- **t_fr** :: [inout] Freezing point potential temperature referenced to the surface [degC]
- **start** :: [in] Starting index within the array
- **npts** :: [in] The number of values to calculate
- **eos** :: Equation of state structure

- **pres_scale** :: [in] A multiplicative factor to convert pressure into Pa.

Call to `tfreeze_linear` `tfreeze_millero` `tfreeze_teos10`

subroutine mom_eos/**calculate_density_derivs_array**(*T*, *S*, *pressure*, *drho_dT*, *drho_dS*, *start*, *npts*, *EOS*, *scale*)

Calls the appropriate subroutine to calculate density derivatives for 1-D array inputs.

Parameters

- **t** :: [in] Potential temperature referenced to the surface [degC]
- **s** :: [in] Salinity [ppt]
- **pressure** :: [in] Pressure [Pa] or [R L2 T-2 ~> Pa]
- **drho_dt** :: [inout] The partial derivative of density with potential temperature [kg m-3 degC-1] or [R degC-1 ~> kg m-3 degC-1]
- **drho_ds** :: [inout] The partial derivative of density with salinity, in [kg m-3 ppt-1] or [R degC-1 ~> kg m-3 ppt-1]
- **start** :: [in] Starting index within the array
- **npts** :: [in] The number of values to calculate
- **eos** :: Equation of state structure
- **scale** :: [in] A multiplicative factor by which to scale density in combination with scaling given by US [various]

Call to `eos_linear` `eos_nemo` `eos_teos10` `eos_unesco` `eos_wright`

Called from *calculate_density_derivs_1d*

subroutine mom_eos/**calculate_density_derivs_1d**(*T*, *S*, *pressure*, *drho_dT*, *drho_dS*, *EOS*, *dom*, *scale*)

Calls the appropriate subroutine to calculate density derivatives for 1-D array inputs.

Parameters

- **t** :: [in] Potential temperature referenced to the surface [degC]
- **s** :: [in] Salinity [ppt]
- **pressure** :: [in] Pressure [R L2 T-2 ~> Pa]
- **drho_dt** :: [inout] The partial derivative of density with potential temperature [R degC-1 ~> kg m-3 degC-1]
- **drho_ds** :: [inout] The partial derivative of density with salinity [R degC-1 ~> kg m-3 ppt-1]
- **eos** :: Equation of state structure
- **dom** :: [in] The domain of indices to work on, taking into account that arrays start at 1.
- **scale** :: [in] A multiplicative factor by which to scale density in combination with scaling given by US [various]

Call to *calculate_density_derivs_array*

subroutine mom_eos/**calculate_density_derivs_scalar**(*T*, *S*, *pressure*, *drho_dT*, *drho_dS*, *EOS*, *scale*)

Calls the appropriate subroutines to calculate density derivatives by promoting a scalar to a one-element array.

Parameters

- **t** :: [in] Potential temperature referenced to the surface [degC]

- **s** :: [in] Salinity [ppt]
- **pressure** :: [in] Pressure [Pa] or [R L2 T-2 ~> Pa]
- **drho_dt** :: [out] The partial derivative of density with potential temperature [kg m-3 degC-1] or [R degC-1 ~> kg m-3 degC-1]
- **drho_ds** :: [out] The partial derivative of density with salinity, in [kg m-3 ppt-1] or [R ppt-1 ~> kg m-3 ppt-1]
- **eos** :: Equation of state structure
- **scale** :: [in] A multiplicative factor by which to scale density in combination with scaling given by US [various]

Call to `eos_linear` `eos_teos10` `eos_wright`

subroutine mom_eos/**calculate_density_second_derivs_array**(*T*, *S*, *pressure*, *drho_dS_dS*, *drho_dS_dT*, *drho_dT_dT*, *drho_dS_dP*, *drho_dT_dP*, *start*, *npts*, *EOS*, *scale*)

Calls the appropriate subroutine to calculate density second derivatives for 1-D array inputs.

Parameters

- **t** :: [in] Potential temperature referenced to the surface [degC]
- **s** :: [in] Salinity [ppt]
- **pressure** :: [in] Pressure [Pa] or [R L2 T-2 ~> Pa]
- **drho_ds_ds** :: [inout] Partial derivative of beta with respect to S [kg m-3 ppt-2] or [R ppt-2 ~> kg m-3 ppt-2]
- **drho_ds_dt** :: [inout] Partial derivative of beta with respect to T [kg m-3 ppt-1 degC-1] or [R ppt-1 degC-1 ~> kg m-3 ppt-1 degC-1]
- **drho_dt_dt** :: [inout] Partial derivative of alpha with respect to T [kg m-3 degC-2] or [R degC-2 ~> kg m-3 degC-2]
- **drho_ds_dp** :: [inout] Partial derivative of beta with respect to pressure [kg m-3 ppt-1 Pa-1] or [R ppt-1 Pa-1 ~> kg m-3 ppt-1 Pa-1]
- **drho_dt_dp** :: [inout] Partial derivative of alpha with respect to pressure [kg m-3 degC-1 Pa-1] or [R degC-1 Pa-1 ~> kg m-3 degC-1 Pa-1]
- **start** :: [in] Starting index within the array
- **npts** :: [in] The number of values to calculate
- **eos** :: Equation of state structure
- **scale** :: [in] A multiplicative factor by which to scale density in combination with scaling given by US [various]

Call to `eos_linear` `eos_teos10` `eos_wright`

subroutine mom_eos/**calculate_density_second_derivs_scalar**(*T*, *S*, *pressure*, *drho_dS_dS*, *drho_dS_dT*, *drho_dT_dT*, *drho_dS_dP*, *drho_dT_dP*, *EOS*, *scale*)

Calls the appropriate subroutine to calculate density second derivatives for scalar nputs.

Parameters

- **t** :: [in] Potential temperature referenced to the surface [degC]
- **s** :: [in] Salinity [ppt]
- **pressure** :: [in] Pressure [Pa] or [R L2 T-2 ~> Pa]
- **drho_ds_ds** :: [out] Partial derivative of beta with respect to S [kg m-3 ppt-2] or [R ppt-2 ~> kg m-3 ppt-2]
- **drho_ds_dt** :: [out] Partial derivative of beta with respect to T [kg m-3 ppt-1 degC-1] or [R ppt-1 degC-1 ~> kg m-3 ppt-1 degC-1]
- **drho_dt_dt** :: [out] Partial derivative of alpha with respect to T [kg m-3 degC-2] or [R degC-2 ~> kg m-3 degC-2]
- **drho_ds_dp** :: [out] Partial derivative of beta with respect to pressure [kg m-3 ppt-1 Pa-1] or [R ppt-1 Pa-1 ~> kg m-3 ppt-1 Pa-1]
- **drho_dt_dp** :: [out] Partial derivative of alpha with respect to pressure [kg m-3 degC-1 Pa-1] or [R degC-1 Pa-1 ~> kg m-3 degC-1 Pa-1]
- **eos** :: Equation of state structure
- **scale** :: [in] A multiplicative factor by which to scale density in combination with scaling given by US [various]

Call to `eos_linear` `eos_teos10` `eos_wright`

subroutine mom_eos/**calculate_spec_vol_derivs_array**(*T*, *S*, *pressure*, *dSV_dT*, *dSV_dS*, *start*, *npts*, *EOS*)

Calls the appropriate subroutine to calculate specific volume derivatives for an array.

Parameters

- **t** :: [in] Potential temperature referenced to the surface [degC]
- **s** :: [in] Salinity [ppt]
- **pressure** :: [in] Pressure [Pa]
- **dsv_dt** :: [inout] The partial derivative of specific volume with potential temperature [m3 kg-1 degC-1]
- **dsv_ds** :: [inout] The partial derivative of specific volume with salinity [m3 kg-1 ppt-1]
- **start** :: [in] Starting index within the array
- **npts** :: [in] The number of values to calculate
- **eos** :: Equation of state structure

Call to `eos_linear` `eos_nemo` `eos_teos10` `eos_unesco` `eos_wright`

Called from *calc_spec_vol_derivs_1d*

subroutine mom_eos/**calc_spec_vol_derivs_1d**(*T*, *S*, *pressure*, *dSV_dT*, *dSV_dS*, *EOS*, *dom*, *scale*)

Calls the appropriate subroutine to calculate specific volume derivatives for 1-d array inputs, potentially limiting the domain of indices that are worked on.

Parameters

- **t** :: [in] Potential temperature referenced to the surface [degC]
- **s** :: [in] Salinity [ppt]
- **pressure** :: [in] Pressure [R L2 T-2 ~> Pa]
- **dsv_dt** :: [inout] The partial derivative of specific volume with potential temperature [R-1 degC-1 ~> m3 kg-1 degC-1]
- **dsv_ds** :: [inout] The partial derivative of specific volume with salinity [R-1 ppt-1 ~> m3 kg-1 ppt-1]
- **eos** :: Equation of state structure
- **dom** :: [in] The domain of indices to work on, taking into account that arrays start at 1.
- **scale** :: [in] A multiplicative factor by which to scale specific volume in combination with scaling given by US [various]

Call to *calculate_spec_vol_derivs_array*

subroutine mom_eos/**calculate_compress_array**(*T*, *S*, *press*, *rho*, *drho_dp*, *start*, *npts*, *EOS*)

Calls the appropriate subroutine to calculate the density and compressibility for 1-D array inputs. If US is present, the units of the inputs and outputs are rescaled.

Parameters

- **t** :: [in] Potential temperature referenced to the surface [degC]
- **s** :: [in] Salinity [PSU]
- **press** :: [in] Pressure [Pa] or [R L2 T-2 ~> Pa]
- **rho** :: [inout] In situ density [kg m-3] or [R ~> kg m-3]
- **drho_dp** :: [inout] The partial derivative of density with pressure (also the inverse of the square of sound speed) [s2 m-2] or [T2 L-2]
- **start** :: [in] Starting index within the array
- **npts** :: [in] The number of values to calculate
- **eos** :: Equation of state structure

Call to `eos_linear` `eos_nemo` `eos_teos10` `eos_unesco` `eos_wright`

Called from *calculate_compress_scalar*

subroutine mom_eos/**calculate_compress_scalar**(*T*, *S*, *pressure*, *rho*, *drho_dp*, *EOS*)

Calculate density and compressibility for a scalar. This just promotes the scalar to an array with a singleton dimension and calls calculate_compress_array. If US is present, the units of the inputs and outputs are rescaled.

Parameters

- **t** :: [in] Potential temperature referenced to the surface [degC]
- **s** :: [in] Salinity [ppt]
- **pressure** :: [in] Pressure [Pa] or [R L2 T-2 ~> Pa]
- **rho** :: [out] In situ density [kg m-3] or [R ~> kg m-3]

- **drho_dp** :: [out] The partial derivative of density with pressure (also the inverse of the square of sound speed) [s2 m-2] or [T2 L-2]
- **eos** :: Equation of state structure

Call to *calculate_compress_array*

function mom_eos/**eos_domain**(*HI*, *halo*) *[integer]*

This subroutine returns a two point integer array indicating the domain of i-indices to work on in EOS calls based on information from a hor_index type.

Parameters

- **hi** :: [in] The horizontal index structure
- **halo** :: [in] The halo size to work on; missing is equivalent to 0.

Return **undefined** :: The index domain that the EOS will work on, taking into account that the arrays inside the EOS routines will start at 1.

subroutine mom_eos/**analytic_int_specific_vol_dp**(*T*, *S*, *p_t*, *p_b*, *alpha_ref*, *HI*, *EOS*, *dza*, *intp_dza*, *intx_dza*, *inty_dza*, *halo_size*, *bathyP*, *dP_tiny*, *useMassWghtInterp*)

Calls the appropriate subroutine to calculate analytical and nearly-analytical integrals in pressure across layers of geopotential anomalies, which are required for calculating the finite-volume form pressure accelerations in a non-Boussinesq model. There are essentially no free assumptions, apart from the use of Boole's rule to do the horizontal integrals, and from a truncation in the series for log(1-eps/1+eps) that assumes that |eps| < 0.34.

Parameters

- **hi** :: [in] The horizontal index structure
- **t** :: [in] Potential temperature referenced to the surface [degC]
- **s** :: [in] Salinity [ppt]
- **p_t** :: [in] Pressure at the top of the layer [R L2 T-2 ~> Pa] or [Pa]
- **p_b** :: [in] Pressure at the bottom of the layer [R L2 T-2 ~> Pa] or [Pa]
- **alpha_ref** :: [in] A mean specific volume that is subtracted out to reduce the magnitude of each of the integrals [R-1 ~> m3 kg-1] The calculation is mathematically identical with different values of alpha_ref, but this reduces the effects of roundoff.
- **eos** :: Equation of state structure
- **dza** :: [inout] The change in the geopotential anomaly across
- **intp_dza** :: [inout] The integral in pressure through the layer of the
- **intx_dza** :: [inout] The integral in x of the difference between the
- **inty_dza** :: [inout] The integral in y of the difference between the
- **halo_size** :: [in] The width of halo points on which to calculate dza.
- **bathyp** :: [in] The pressure at the bathymetry [R L2 T-2 ~> Pa] or [Pa]
- **dp_tiny** :: [in] A miniscule pressure change with the same units as p_t [R L2 T-2 ~> Pa] or [Pa]
- **usemasswghtinterp** :: [in] If true, uses mass weighting to interpolate T/S for top and bottom integrals.

Call to eos_linear eos_wright

subroutine mom_eos/**analytic_int_density_dz**(*T*, *S*, *z_t*, *z_b*, *rho_ref*, *rho_0*, *G_e*, *HI*, *EOS*, *dpa*, *intz_dpa*, *intx_dpa*, *inty_dpa*, *bathyT*, *dz_neglect*, *useMassWghtInterp*)

This subroutine calculates analytical and nearly-analytical integrals of pressure anomalies across layers, which are required for calculating the finite-volume form pressure accelerations in a Boussinesq model.

Parameters

- **hi** :: [in] Ocean horizontal index structure
- **t** :: [in] Potential temperature referenced to the surface [degC]
- **s** :: [in] Salinity [ppt]
- **z_t** :: [in] Height at the top of the layer in depth units [Z ~> m]
- **z_b** :: [in] Height at the bottom of the layer [Z ~> m]
- **rho_ref** :: [in] A mean density [R ~> kg m-3] or [kg m-3], that is subtracted out to reduce the magnitude of each of the integrals.
- **rho_0** :: [in] A density [R ~> kg m-3] or [kg m-3], that is used to calculate the pressure (as p~=-z*rho_0*G_e) used in the equation of state.
- **g_e** :: [in] The Earth's gravitational acceleration [L2 Z-1 T-2 ~> m s-2] or [m2 Z-1 s-2 ~> m s-2]
- **eos** :: Equation of state structure
- **dpa** :: [inout] The change in the pressure anomaly
- **intz_dpa** :: [inout] The integral through the thickness of the
- **intx_dpa** :: [inout] The integral in x of the difference between
- **inty_dpa** :: [inout] The integral in y of the difference between
- **bathyt** :: [in] The depth of the bathymetry [Z ~> m]
- **dz_neglect** :: [in] A miniscule thickness change [Z ~> m]
- **usemasswghtinterp** :: [in] If true, uses mass weighting to interpolate T/S for top and bottom integrals.

Call to eos_linear eos_wright

function mom_eos/**query_compressible**(*EOS*) *[logical]*

Returns true if the equation of state is compressible (i.e. has pressure dependence)

Parameters **eos** :: Equation of state structure

subroutine mom_eos/**eos_init**(*param_file*, *EOS*, *US*)

Initializes EOS_type by allocating and reading parameters.

Parameters

- **param_file** :: [in] Parameter file structure
- **eos** :: Equation of state structure
- **us** :: [in] A dimensional unit scaling type

Call to *eos_allocate* eos_default eos_linear eos_linear_string eos_nemo eos_nemo_string eos_teos10 eos_teos10_string eos_unesco eos_unesco_string eos_wright eos_wright_string tfreeze_default tfreeze_linear tfreeze_linear_string tfreeze_millero tfreeze_millero_string tfreeze_teos10 tfreeze_teos10_string

Called from *mom_ice_shelf::initialize_ice_shelf*

subroutine mom_eos/**eos_manual_init**(*EOS*, *form_of_EOS*, *form_of_TFreeze*, *EOS_quadrature*, *Compressible*, *Rho_T0_S0*, *drho_dT*, *dRho_dS*, *TFr_S0_P0*, *dTFr_dS*, *dTFr_dp*)

Manually initialized an EOS type (intended for unit testing of routines which need a specific EOS)

Parameters

- **eos** :: Equation of state structure
- **form_of_eos** :: [in] A coded integer indicating the equation of state to use.
- **form_of_tfreeze** :: [in] A coded integer indicating the expression for the potential temperature of the freezing point.
- **eos_quadrature** :: [in] If true, always use the generic (quadrature) code for the integrals of density.
- **compressible** :: [in] If true, in situ density is a function of pressure.
- **rho_t0_s0** :: [in] Density at T=0 degC and S=0 ppt [kg m-3]
- **drho_dt** :: [in] Partial derivative of density with temperature in [kg m-3 degC-1]
- **drho_ds** :: [in] Partial derivative of density with salinity in [kg m-3 ppt-1]
- **tfr_s0_p0** :: [in] The freezing potential temperature at S=0, P=0 [degC]
- **dtfr_ds** :: [in] The derivative of freezing point with salinity in [degC ppt-1]
- **dtfr_dp** :: [in] The derivative of freezing point with pressure in [degC Pa-1]

subroutine mom_eos/**eos_allocate**(*EOS*)

Allocates EOS_type.

Parameters **eos** :: Equation of state structure

Called from *eos_init*

subroutine mom_eos/**eos_end**(*EOS*)

Deallocates EOS_type.

Parameters **eos** :: Equation of state structure

subroutine mom_eos/**eos_use_linear**(*Rho_T0_S0*, *dRho_dT*, *dRho_dS*, *EOS*, *use_quadrature*)

Set equation of state structure (EOS) to linear with given coefficients.

Parameters

- **rho_t0_s0** :: [in] Density at T=0 degC and S=0 ppt [kg m-3]
- **drho_dt** :: [in] Partial derivative of density with temperature [kg m-3 degC-1]
- **drho_ds** :: [in] Partial derivative of density with salinity [kg m-3 ppt-1]
- **use_quadrature** :: [in] If true, always use the generic (quadrature) code for the integrals of density.
- **eos** :: Equation of state structure

Call to eos_linear

subroutine mom_eos/**convert_temp_salt_for_teos10**(*T*, *S*, *HI*, *kd*, *mask_z*, *EOS*)

Convert T&S to Absolute Salinity and Conservative Temperature if using TEOS10.

Parameters

- **kd** :: [in] The number of layers to work on

- **hi** :: [in] The horizontal index structure
- **t** :: [inout] Potential temperature referenced to the surface [degC]
- **s** :: [inout] Salinity [ppt]
- **mask_z** :: [in] 3d mask regulating which points to convert.
- **eos** :: Equation of state structure

Call to `eos_nemo eos_teos10`

function `mom_eos/`**eos_quadrature**(*EOS*) *[logical]*

Return value of EOS_quadrature.

Parameters **eos** :: Equation of state structure

subroutine `mom_eos/`**extract_member_eos**(*EOS*, *form_of_EOS*, *form_of_TFreeze*, *EOS_quadrature*, *Compressible*, *Rho_T0_S0*, *drho_dT*, *dRho_dS*, *TFr_S0_P0*, *dTFr_dS*, *dTFr_dp*)

Extractor routine for the EOS type if the members need to be accessed outside this module.

Parameters

- **eos** :: Equation of state structure
- **form_of_eos** :: [out] A coded integer indicating the equation of state to use.
- **form_of_tfreeze** :: [out] A coded integer indicating the expression for the potential temperature of the freezing point.
- **eos_quadrature** :: [out] If true, always use the generic (quadrature) code for the integrals of density.
- **compressible** :: [out] If true, in situ density is a function of pressure.
- **rho_t0_s0** :: [out] Density at T=0 degC and S=0 ppt [kg m-3]
- **drho_dt** :: [out] Partial derivative of density with temperature in [kg m-3 degC-1]
- **drho_ds** :: [out] Partial derivative of density with salinity in [kg m-3 ppt-1]
- **tfr_s0_p0** :: [out] The freezing potential temperature at S=0, P=0 [degC]
- **dtfr_ds** :: [out] The derivative of freezing point with salinity [degC PSU-1]
- **dtfr_dp** :: [out] The derivative of freezing point with pressure [degC Pa-1]

13.1.4 mom_ice_shelf module reference

Implements the thermodynamic aspects of ocean / ice-shelf interactions, along with a crude placeholder for a later implementation of full ice shelf dynamics, all using the MOM framework and coding style.

More...

Data Types

ice_shelf_cs	Control structure that contains ice shelf parameters and diagnostics handles.

Functions/Subroutines

shelf_calc_flux()	Calculates fluxes between the ocean and ice-shelf using the three-equations formulation
change_thickness_using_melt()	Changes the thickness (mass) of the ice shelf based on sub-ice-shelf melting.
add_shelf_forces()	This subroutine adds the mechanical forcing fields and perhaps shelf areas, based on the
add_shelf_pressure()	This subroutine adds the ice shelf pressure to the fluxes type.
add_shelf_flux()	Updates surface fluxes that are influenced by sub-ice-shelf melting.
initialize_ice_shelf()	Initializes shelf model data, parameters and diagnostics.
initialize_shelf_mass()	Initializes shelf mass based on three options (file, zero and user)
update_shelf_mass()	Updates the ice shelf mass using data from a file.
ice_shelf_save_restart()	Save the ice shelf restart file.
ice_shelf_end()	Deallocates all memory associated with this module.
solo_step_ice_shelf()	This routine is for stepping a stand-alone ice shelf model without an ocean.

Detailed Description

section_ICE_SHELF

This module implements the thermodynamic aspects of ocean/ice-shelf inter-actions using the MOM framework and coding style.

Derived from code by Chris Little, early 2010.

The ice-sheet dynamics subroutines do the following: initialize_shelf_mass - Initializes the ice shelf mass distribution.

- Initializes h_shelf, h_mask, area_shelf_h
- CURRENTLY: initializes mass_shelf as well, but this is unnecessary, as mass_shelf is initialized based on h_shelf and density_ice immediately afterwards. Possibly subroutine should be renamed update_shelf_mass - updates ice shelf mass via netCDF file USER_update_shelf_mass (TODO). solo_step_ice_shelf - called only in ice-only mode. shelf_calc_flux - after melt rate & fluxes are calculated, ice dynamics are done. currently mass_shelf is updated immediately after ice_shelf_advect in fully dynamic mode.

NOTES: be aware that hmask(:,:) has a number of functions; it is used for front advancement, for subroutines in the velocity solve, and for thickness boundary conditions (this last one may be removed). in other words, interfering with its updates will have implications you might not expect.

Overall issues: Many variables need better documentation and units and the subgrid on which they are discretized.

ICE_SHELF equations

The three fundamental equations are: Heat flux

$$\rho_w C_{pw} \gamma_T (T_w - T_b) = \rho_i \dot{m} L_f$$

Salt flux

$$\rho_w \gamma_s (S_w - S_b) = \rho_i \dot{m} S_b$$

Freezing temperature

$$T_b = aS_b + b + cP$$

where ....

References

Asay-Davis, Xylar S., Stephen L. Cornford, Benjamin K. Galton-Fenzi, Rupert M. Gladstone, G. Hilmar Gudmundsson, David M. Holland, Paul R. Holland, and Daniel F. Martin. Experimental design for three interrelated marine ice sheet and ocean model intercomparison projects: MISMIP v. 3 (MISMIP+), ISOMIP v. 2 (ISOMIP+) and MISOMIP v. 1 (MISOMIP1). Geoscientific Model Development 9, no. 7 (2016): 2471.

Goldberg, D. N., et al. Investigation of land ice-ocean interaction with a fully coupled ice-ocean model: 1. Model description and behavior. Journal of Geophysical Research: Earth Surface 117.F2 (2012).

Goldberg, D. N., et al. Investigation of land ice-ocean interaction with a fully coupled ice-ocean model: 2. Sensitivity to external forcings. Journal of Geophysical Research: Earth Surface 117.F2 (2012).

Holland, David M., and Adrian Jenkins. Modeling thermodynamic ice-ocean interactions at the base of an ice shelf. Journal of Physical Oceanography 29.8 (1999): 1787-1800.

Type Documentation

type mom_ice_shelf/**ice_shelf_cs**
Control structure that contains ice shelf parameters and diagnostics handles.

Type fields

- **% id_melt** *[integer]* :: Diagnostic handles.
- **% id_exch_vel_s** *[integer]* :: Diagnostic handles.
- **% id_exch_vel_t** *[integer]* :: Diagnostic handles.
- **% id_tfreeze** *[integer]* :: Diagnostic handles.
- **% id_tfl_shelf** *[integer]* :: Diagnostic handles.
- **% id_thermal_driving** *[integer]* :: Diagnostic handles.
- **% id_haline_driving** *[integer]* :: Diagnostic handles.
- **% id_u_ml** *[integer]* :: Diagnostic handles.
- **% id_v_ml** *[integer]* :: Diagnostic handles.
- **% id_sbdry** *[integer]* :: Diagnostic handles.
- **% id_h_shelf** *[integer]* :: Diagnostic handles.

- **% id_h_mask** *[integer]* :: Diagnostic handles.
- **% id_surf_elev** *[integer]* :: Diagnostic handles.
- **% id_bathym** *[integer]* :: Diagnostic handles.
- **% id_area_shelf_h** *[integer]* :: Diagnostic handles.
- **% id_ustar_shelf** *[integer]* :: Diagnostic handles.
- **% id_shelf_mass** *[integer]* :: Diagnostic handles.
- **% id_mass_flux** *[integer]* :: Diagnostic handles.
- **% restart_csp** *[type(mom_restart_cs),pointer]* :: A pointer to the restart control structure for the ice shelves.
- **% grid** *[type(ocean_grid_type)]* :: Grid for the ice-shelf model.
- **% us** *[type(unit_scale_type),pointer]* :: A structure containing various unit conversion factors.
- **% ocn_grid** *[type(ocean_grid_type),pointer]* :: A pointer to the ocean model grid The rest is private.
- **% flux_factor** *[real]* :: A factor that can be used to turn off ice shelf melting (flux_factor = 0) [nondim].
- **% restart_output_dir** *[character(len=128)]* :: The directory in which to write restart files.
- **% iss** *[type(ice_shelf_state),pointer]* :: A structure with elements that describe the ice-shelf state.
- **% dcs** *[type(ice_shelf_dyn_cs),pointer]* :: The control structure for the ice-shelf dynamics.
- **% utide** *[real(:,:),pointer]* :: An unresolved tidal velocity [L T-1 ~> m s-1].
- **% ustar_bg** *[real]* :: A minimum value for ustar under ice shelves [Z T-1 ~> m s-1].
- **% cdrag** *[real]* :: drag coefficient under ice shelves [nondim].
- **% g_earth** *[real]* :: The gravitational acceleration [L2 Z-1 T-2 ~> m s-2].
- **% cp** *[real]* :: The heat capacity of sea water [Q degC-1 ~> J kg-1 degC-1].
- **% rho_ocn** *[real]* :: A reference ocean density [R ~> kg m-3].
- **% cp_ice** *[real]* :: The heat capacity of fresh ice [Q degC-1 ~> J kg-1 degC-1].
- **% gamma_t** *[real]* :: The (fixed) turbulent exchange velocity in the 2-equation formulation [Z T-1 ~> m s-1].
- **% salin_ice** *[real]* :: The salinity of shelf ice [ppt].
- **% temp_ice** *[real]* :: The core temperature of shelf ice [degC].
- **% kv_ice** *[real]* :: The viscosity of ice [L4 Z-2 T-1 ~> m2 s-1].
- **% density_ice** *[real]* :: A typical density of ice [R ~> kg m-3].
- **% kv_molec** *[real]* :: The molecular kinematic viscosity of sea water [Z2 T-1 ~> m2 s-1].
- **% kd_molec_salt** *[real]* :: The molecular diffusivity of salt [Z2 T-1 ~> m2 s-1].
- **% kd_molec_temp** *[real]* :: The molecular diffusivity of heat [Z2 T-1 ~> m2 s-1].
- **% lat_fusion** *[real]* :: The latent heat of fusion [Q ~> J kg-1].

- **% gamma_t_3eq** *[real]* :: Nondimensional heat-transfer coefficient, used in the 3Eq. formulation.
- **% gamma_s_3eq** *[real]* :: Nondimensional salt-transfer coefficient, used in the 3Eq. formulation This number should be specified by the user.
- **% col_mass_melt_threshold** *[real]* :: An ocean column mass below the iceshelf below which melting does not occur [R Z ~> kg m-2].
- **% mass_from_file** *[logical]* :: Read the ice shelf mass from a file every dt.
- **% time_step** *[real]* :: this is the shortest timestep that the ice shelf sees, and is equal to the forcing timestep (it is passed in when the shelf is initialized - so need to reorganize MOM driver. it will be the prognistic timestep . . . maybe.
- **% solo_ice_sheet** *[logical]* :: whether the ice model is running without being coupled to the ocean
- **% gl_regularize** *[logical]* :: whether to regularize the floatation condition at the grounding line a la Goldberg Holland Schoof 2009
- **% gl_couple** *[logical]* :: whether to let the floatation condition be determined by ocean column thickness means update_OD_ffrac will be called (note: GL_regularize and GL_couple should be exclusive)
- **% calve_to_mask** *[logical]* :: If true, calve any ice that passes outside of a masked area.
- **% min_thickness_simple_calve** *[real]* :: min. ice shelf thickness criteria for calving [Z ~> m].
- **% t0** *[real]* :: temperature at ocean surface in the restoring region [degC]
- **% s0** *[real]* :: Salinity at ocean surface in the restoring region [ppt].
- **% input_flux** *[real]* :: Ice volume flux at an upstream open boundary [m3 s-1].
- **% input_thickness** *[real]* :: Ice thickness at an upstream open boundary [m].
- **% time** *[type(time_type)]* :: The component's time.
- **% eqn_of_state** *[type(eos_type),pointer]* :: Type that indicates the equation of state to use.
- **% active_shelf_dynamics** *[logical]* :: True if the ice shelf mass changes as a result the dynamic ice-shelf model.
- **% override_shelf_movement** *[logical]* :: If true, user code specifies the shelf movement instead of using the dynamic ice-shelf mode.
- **% isthermo** *[logical]* :: True if the ice shelf can exchange heat and mass with the underlying ocean.
- **% threeeq** *[logical]* :: If true, the 3 equation consistency equations are used to calculate the flux at the ocean-ice interface.
- **% insulator** *[logical]* :: If true, ice shelf is a perfect insulator.
- **% const_gamma** *[logical]* :: If true, gamma_T is specified by the user.
- **% constant_sea_level** *[logical]* :: if true, apply an evaporative, heat and salt fluxes. It will avoid large increase in sea level.
- **% min_ocean_mass_float** *[real]* :: The minimum ocean mass per unit area before the ice shelf is considered to float when constant_sea_level is used [R Z ~> kg m-2].
- **% cutoff_depth** *[real]* :: Depth above which melt is set to zero (>= 0) [Z ~> m].

- **% find_salt_root** *[logical]* :: If true, if true find Sbdry using a quadratic eq.
- **% tfr_0_0** *[real]* :: The freezing point at 0 pressure and 0 salinity [degC].
- **% dtfr_ds** *[real]* :: Partial derivative of freezing temperature with salinity [degC ppt-1].
- **% dtfr_dp** *[real]* :: Partial derivative of freezing temperature with pressure [degC T2 R-1 L-2 ~> degC Pa-1].
- **% id_read_mass** *[integer]* :: An integer handle used in time interpolation of the ice shelf mass read from a file.
- **% id_read_area** *[integer]* :: An integer handle used in time interpolation of the ice shelf mass read from a file.
- **% diag** *[type(diag_ctrl),pointer]* :: A structure that is used to control diagnostic output.
- **% user_cs** *[type(user_ice_shelf_cs),pointer]* :: A pointer to the control structure for user-supplied modifications to the ice shelf code.
- **% debug** *[logical]* :: If true, write verbose checksums for debugging purposes and use reproducible sums.

Function/Subroutine Documentation

subroutine mom_ice_shelf/**shelf_calc_flux**(*sfc_state*, *fluxes*, *Time*, *time_step*, *CS*, *forces*)

Calculates fluxes between the ocean and ice-shelf using the three-equations formulation (optional to use just two equations). See *ICE_SHELF equations* .

Parameters

- **sfc_state** :: [inout] A structure containing fields that describe the surface state of the ocean. The intent is only inout to allow for halo updates.
- **fluxes** :: [inout] structure containing pointers to any possible thermodynamic or mass-flux forcing fields.
- **time** :: [in] Start time of the fluxes.
- **time_step** :: [in] Length of time over which these fluxes will be applied [s].
- **cs** :: A pointer to the control structure returned by a previous call to initialize_ice_shelf.
- **forces** :: [inout] A structure with the driving mechanical forces

Call to *add_shelf_flux* *add_shelf_forces* *change_thickness_using_melt* id_clock_pass id_clock_shelf *update_shelf_mass*

subroutine mom_ice_shelf/**change_thickness_using_melt**(*ISS*, *G*, *US*, *time_step*, *fluxes*, *density_ice*, *debug*)

Changes the thickness (mass) of the ice shelf based on sub-ice-shelf melting.

Parameters

- **g** :: [inout] The ocean's grid structure.
- **iss** :: [inout] A structure with elements that describe the ice-shelf state
- **us** :: [in] A dimensional unit scaling type
- **time_step** :: [in] The time step for this update [T ~> s].
- **fluxes** :: [inout] structure containing pointers to any possible thermodynamic or mass-flux forcing fields.

- **density_ice** :: [in] The density of ice-shelf ice [R ~> kg m-3].
- **debug** :: [in] If present and true, write chksums

Called from *shelf_calc_flux*

subroutine mom_ice_shelf/**add_shelf_forces**(*G*, *US*, *CS*, *forces*, *do_shelf_area*)

This subroutine adds the mechanical forcing fields and perhaps shelf areas, based on the ice state in ice_shelf_CS.

Parameters

- **g** :: [inout] The ocean's grid structure.
- **us** :: [in] A dimensional unit scaling type
- **cs** :: This module's control structure.
- **forces** :: [inout] A structure with the driving mechanical forces
- **do_shelf_area** :: [in] If true find the shelf-covered areas.

Called from *initialize_ice_shelf shelf_calc_flux*

subroutine mom_ice_shelf/**add_shelf_pressure**(*G*, *US*, *CS*, *fluxes*)

This subroutine adds the ice shelf pressure to the fluxes type.

Parameters

- **g** :: [inout] The ocean's grid structure.
- **us** :: [in] A dimensional unit scaling type
- **cs** :: [in] This module's control structure.
- **fluxes** :: [inout] A structure of surface fluxes that may be updated.

Called from *add_shelf_flux initialize_ice_shelf*

subroutine mom_ice_shelf/**add_shelf_flux**(*G*, *US*, *CS*, *sfc_state*, *fluxes*)

Updates surface fluxes that are influenced by sub-ice-shelf melting.

Parameters

- **g** :: [inout] The ocean's grid structure.
- **us** :: [in] A dimensional unit scaling type
- **cs** :: This module's control structure.
- **sfc_state** :: [inout] Surface ocean state
- **fluxes** :: [inout] A structure of surface fluxes that may be used/updated.

Call to *add_shelf_pressure*

Called from *shelf_calc_flux*

subroutine mom_ice_shelf/**initialize_ice_shelf**(*param_file*, *ocn_grid*, *Time*, *CS*, *diag*, *forces*, *fluxes*, *Time_in*, *solo_ice_sheet_in*)

Initializes shelf model data, parameters and diagnostics.

Parameters

- **param_file** :: [in] A structure to parse for run-time parameters
- **ocn_grid** :: The calling ocean model's horizontal grid structure
- **time** :: [inout] The clock that that will indicate the model time

- **cs** :: A pointer to the ice shelf control structure
- **diag** :: [in] A structure that is used to regulate the diagnostic output.
- **fluxes** :: [inout] A structure containing pointers to any possible thermodynamic or mass-flux forcing fields.
- **forces** :: [inout] A structure with the driving mechanical forces
- **time_in** :: [in] The time at initialization.
- **solo_ice_sheet_in** :: [in] If present, this indicates whether a solo ice-sheet driver.

Call to *add_shelf_forces* *add_shelf_pressure* *mom_eos::eos_init* *mom_unit_scaling::fix_restart_unit_scaling* id_clock_pass id_clock_shelf *initialize_shelf_mass* *mom_unit_scaling::unit_scaling_init*

subroutine mom_ice_shelf/**initialize_shelf_mass**(*G*, *param_file*, *CS*, *ISS*, *new_sim*)

Initializes shelf mass based on three options (file, zero and user)

Parameters

- **g** :: [in] The ocean's grid structure.
- **param_file** :: [in] A structure to parse for run-time parameters
- **cs** :: A pointer to the ice shelf control structure
- **iss** :: [inout] The ice shelf state type that is being updated
- **new_sim** :: [in] If present and false, this run is being restarted

Called from *initialize_ice_shelf*

subroutine mom_ice_shelf/**update_shelf_mass**(*G*, *US*, *CS*, *ISS*, *Time*)

Updates the ice shelf mass using data from a file.

Parameters

- **g** :: [inout] The ocean's grid structure.
- **us** :: [in] A dimensional unit scaling type
- **cs** :: [in] A pointer to the ice shelf control structure
- **iss** :: [inout] The ice shelf state type that is being updated
- **time** :: [in] The current model time

Called from *shelf_calc_flux*

subroutine mom_ice_shelf/**ice_shelf_save_restart**(*CS*, *Time*, *directory*, *time_stamped*, *filename_suffix*)

Save the ice shelf restart file.

Parameters

- **cs** :: ice shelf control structure
- **time** :: [in] model time at this call
- **directory** :: [in] An optional directory into which to write these restart files.
- **time_stamped** :: [in] f true, the restart file names include a unique time stamp. The default is false.
- **filename_suffix** :: [in] An optional suffix (e.g., a time-stamp) to append to the restart file names.

subroutine mom_ice_shelf/**ice_shelf_end**(*CS*)
Deallocates all memory associated with this module.

Parameters **cs** :: A pointer to the ice shelf control structure

subroutine mom_ice_shelf/**solo_step_ice_shelf**(*CS*, *time_interval*, *nsteps*, *Time*, *min_time_step_in*)
This routine is for stepping a stand-alone ice shelf model without an ocean.

Parameters

- **cs** :: A pointer to the ice shelf control structure
- **time_interval** :: [in] The time interval for this update [s].
- **nsteps** :: [inout] The running number of ice shelf steps.
- **time** :: [inout] The current model time
- **min_time_step_in** :: [in] The minimum permitted time step [T ~> s].

13.1.5 mom_meke module reference

Implements the Mesoscale Eddy Kinetic Energy framework with topographic beta effect included in computing beta in Rhines scale.

More...

Data Types

meke_cs	Control structure that contains MEKE parameters and diagnostics handles.

Functions/Subroutines

step_forward_meke()	Integrates forward-in-time the MEKE eddy energy equation.
meke_equilibrium()	Calculates the equilibrium solutino where the source depends only on MEKE diffusivity
meke_equilibrium_restoring()	
meke_lengthscales()	Calculates the eddy mixing length scale and γ_b and γ_t functions that are ratios of either
meke_lengthscales_0d()	Calculates the eddy mixing length scale and γ_b and γ_t functions that are ratios of either
meke_init()	Initializes the MOM_MEKE module and reads parameters.
meke_alloc_register_restart()	Allocates memory and register restart fields for the MOM_MEKE module.
meke_end()	Deallocates any variables allocated in MEKE_init or MEKE_alloc_register_restart.

Detailed Description

The Mesoscale Eddy Kinetic Energy (MEKE) framework

The MEKE framework accounts for the mean potential energy removed by the first order closures used to parameterize mesoscale eddies. It requires closure at the second order, namely dissipation and transport of eddy energy.

Monitoring the sub-grid scale eddy energy budget provides a means to predict a sub-grid eddy-velocity scale which can be used in the lower order closures.

MEKE equations

The eddy kinetic energy equation is:

$$\partial_{\tilde{t}} E = \overbrace{\dot{E}_b + \gamma_\eta \dot{E}_\eta + \gamma_v \dot{E}_v}^{\text{sources}} - \overbrace{(\lambda + C_d |U_d| \gamma_b^2) E}^{\text{local dissipation}} + \overbrace{\nabla \cdot ((\kappa_E + \gamma_M \kappa_M) \nabla E - \kappa_4 \nabla^3 E)}^{\text{smoothing}}$$

where E is the eddy kinetic energy (variable `MEKE`) with units of m ^{2}s $^{-2}$, and $\tilde{t} = at$ is a scaled time. The non-dimensional factor $a \geq 1$ is used to accelerate towards equilibrium.

The MEKE equation is two-dimensional and obtained by depth averaging the the three-dimensional eddy energy equation. In the following expressions $\langle \phi \rangle = \frac{1}{H} \int_{-D}^{\eta} \phi \, dz$ maps three dimensional terms into the two-dimensional quantities needed.

MEKE source terms

The source term $\dot{E}_b$ is a constant background source of energy intended to avoid the limit $E \to 0$.

The "GM" source term

$$\dot{E}_\eta = -\left\langle \overline{w'b'} \right\rangle = \left\langle \kappa_h N^2 S^2 \right\rangle \approx \left\langle \kappa_h g\prime |\nabla_\sigma \eta|^2 \right\rangle$$

equals the mean potential energy removed by the Gent-McWilliams closure, and is excluded/included in the MEKE budget by the efficiency parameter $\gamma_\eta \in [0, 1]$.

The "frictional" source term

$$\dot{E}_v = \langle \partial_i u_j \tau_{ij} \rangle$$

equals the mean kinetic energy removed by lateral viscous fluxes, and is excluded/included in the MEKE budget by the efficiency parameter $\gamma_v \in [0, 1]$.

MEKE dissipation terms

The local dissipation of E is parameterized through a linear damping, λ , and bottom drag, $C_d |U_d| \gamma_b^2$. The γ_b accounts for the weak projection of the column-mean eddy velocty to the bottom. In other words, the bottom velocity is estimated as $\gamma_b U_e$. The bottom drag coefficient, C_d is the same as that used in the bottom friction in the mean model equations.

The bottom drag velocity scale, U_d , has contributions from the resolved state and E :

$$U_d = \sqrt{U_b^2 + |u|^2_{z=-D} + |\gamma_b U_e|^2}.$$

where the eddy velocity scale, U_e , is given by:

$$U_e = \sqrt{2E}.$$

U_b is a constant background bottom velocity scale and is typically not used (i.e. set to zero).

Following Jansen et al., 2015, the projection of eddy energy on to the bottom is given by the ratio of bottom energy to column mean energy:

$$\gamma_b^2 = \frac{E_b}{E} = \gamma_{d0} + \left(1 + c_b \frac{L_d}{L_f}\right)^{-\frac{4}{5}},$$

$$\gamma_b^2 \leftarrow \max\left(\gamma_b^2, \gamma_{min}^2\right).$$

MEKE smoothing terms

E is laterally diffused by a diffusivity $\kappa_E + \gamma_M \kappa_M$ where κ_E is a constant diffusivity and the term $\gamma_M \kappa_M$ is a "self diffusion" using the diffusivity calculated in the section *Diffusivity derived from MEKE* . κ_4 is a constant bi-harmonic diffusivity.

Diffusivity derived from MEKE

The predicted eddy velocity scale, U_e , can be combined with a mixing length scale to form a diffusivity. The primary use of a MEKE derived diffusivity is for use in thickness diffusion (module mom_thickness_diffuse) and optionally in along isopycnal mixing of tracers (module mom_tracer_hor_diff). The original form used (enabled with MEKE_OLD_LSCALE=True):

$$\kappa_M = \gamma_\kappa \sqrt{\gamma_t^2 U_e^2 A_\Delta}$$

where A_Δ is the area of the grid cell. Following Jansen et al., 2015, we now use

$$\kappa_M = \gamma_\kappa l_M \sqrt{\gamma_t^2 U_e^2}$$

where $\gamma_\kappa \in [0, 1]$ is a non-dimensional factor and, following Jansen et al., 2015, γ_t^2 is the ratio of barotropic eddy energy to column mean eddy energy given by

$$\gamma_t^2 = \frac{E_t}{E} = \left(1 + c_t \frac{L_d}{L_f}\right)^{-\frac{1}{4}},$$

$$\gamma_t^2 \leftarrow \max\left(\gamma_t^2, \gamma_{min}^2\right).$$

The length-scale is a configurable combination of multiple length scales:

$$l_M = \left(\frac{\alpha_d}{L_d} + \frac{\alpha_f}{L_f} + \frac{\alpha_R}{L_R} + \frac{\alpha_e}{L_e} + \frac{\alpha_\Delta}{L_\Delta} + \frac{\delta[L_c]}{L_c}\right)^{-1}$$

where

$$\begin{aligned}
L_d &= \sqrt{\frac{c_g^2}{f^2 + 2\beta c_g}} \sim \frac{c_g}{f} \\
L_R &= \sqrt{\frac{U_e}{\beta^*}} \\
L_e &= \frac{U_e}{|S|N} \\
L_f &= \frac{H}{c_d} \\
L_\Delta &= \sqrt{A_\Delta}.
\end{aligned}$$

L_c is a constant and $\delta[L_c]$ is the impulse function so that the term $\frac{\delta[L_c]}{L_c}$ evaluates to $\frac{1}{L_c}$ when L_c is non-zero but is dropped if $L_c = 0$.

β^* is the effective β that combines both the planetary vorticity gradient (i.e. $\beta = \nabla f$) and the topographic β effect, with the latter weighed by a weighting constant, c_β , that varies from 0 to 1, so that $c_\beta = 0$ means the topographic β effect is ignored, while $c_\beta = 1$ means it is fully considered. The new β^* therefore takes the form of

$$\beta^* = \sqrt{(\partial_x f - c_\beta \frac{f}{D}\partial_x D)^2 + (\partial_y f - c_\beta \frac{f}{D}\partial_y D)^2}$$

where D is water column depth at T points.

Viscosity derived from MEKE

As for κ_M , the predicted eddy velocity scale can be used to form a harmonic eddy viscosity,

$$\kappa_u = \gamma_u \sqrt{U_e^2 A_\Delta}$$

as well as a biharmonic eddy viscosity,

$$\kappa_4 = \gamma_4 \sqrt{U_e^2 A_\Delta^3}$$

Limit cases for local source-dissipative balance

Note that in steady-state (or when $a >> 1$) and there is no diffusion of E then

$$\overline{E} \approx \frac{\dot{E}_b + \gamma_\eta \dot{E}_\eta + \gamma_v \dot{E}_v}{\lambda + C_d |U_d| \gamma_b^2}.$$

In the linear drag limit, where $U_e << \min(U_b, |u|_{z=-D}, C_d^{-1}\lambda)$, the equilibrium becomes $\overline{E} \approx \frac{\dot{E}_b + \gamma_\eta \dot{E}_\eta + \gamma_v \dot{E}_v}{\lambda + C_d \sqrt{U_b^2 + |u|^2_{z=-D}}}$.

In the nonlinear drag limit, where $U_e >> \max(U_b, |u|_{z=-D}, C_d^{-1}\lambda)$, the equilibrium becomes $\overline{E} \approx \left(\frac{\dot{E}_b + \gamma_\eta \dot{E}_\eta + \gamma_v \dot{E}_v}{\sqrt{2} C_d \gamma_b^3}\right)^{\frac{2}{3}}$.

MEKE module parameters

Symbol	Module parameter
•	USE_MEKE
a	MEKE_DTSCALE
$\dot{E}_b$	MEKE_BGSRC
γ_η	MEKE_GMCOEFF
γ_v	MEKE_FrCOEFF
λ	MEKE_DAMPING
U_b	MEKE_USCALE
γ_{d0}	MEKE_CD_SCALE
c_b	MEKE_CB
c_t	MEKE_CT
κ_E	MEKE_KH
κ_4	MEKE_K4
γ_κ	MEKE_KHCOEFF
γ_M	MEKE_KHMEKE_FAC
γ_u	MEKE_VISCOSITY_COEFF_KU
γ_4	MEKE_VISCOSITY_COEFF_AU
γ^2_{min}	MEKE_MIN_GAMMA2
α_d	MEKE_ALPHA_DEFORM
α_f	MEKE_ALPHA_FRICT
α_R	MEKE_ALPHA_RHINES
α_e	MEKE_ALPHA_EADY
α_Δ	MEKE_ALPHA_GRID
L_c	MEKE_FIXED_MIXING_LENGTH
c_β	MEKE_TOPOGRAPHIC_BETA
•	MEKE_KHTH_FAC
•	MEKE_KHTR_FAC

Symbol	Model parameter
C_d	CDRAG

References

Jansen, M. F., A. J. Adcroft, R. Hallberg, and I. M. Held, 2015: Parameterization of eddy fluxes based on a mesoscale energy budget. Ocean Modelling, 92, 28 41, http://doi.org/10.1016/j.ocemod.2015.05.007.

Marshall, D. P., and A. J. Adcroft, 2010: Parameterization of ocean eddies: Potential vorticity mixing, energetics and Arnold first stability theorem. Ocean Modelling, 32, 188 204, http://doi.org/10.1016/j.ocemod.2010.02.001.

Type Documentation

type mom_meke/**meke_cs**

Control structure that contains MEKE parameters and diagnostics handles.

Type fields

- **% id_meke** *[integer]* :: Diagnostic handles.
- **% id_ue** *[integer]* :: Diagnostic handles.
- **% id_kh** *[integer]* :: Diagnostic handles.
- **% id_src** *[integer]* :: Diagnostic handles.
- **% id_ub** *[integer]* :: Diagnostic handles.
- **% id_ut** *[integer]* :: Diagnostic handles.
- **% id_gm_src** *[integer]* :: Diagnostic handles.
- **% id_mom_src** *[integer]* :: Diagnostic handles.
- **% id_gme_snk** *[integer]* :: Diagnostic handles.
- **% id_decay** *[integer]* :: Diagnostic handles.
- **% id_khmeke_u** *[integer]* :: Diagnostic handles.
- **% id_khmeke_v** *[integer]* :: Diagnostic handles.
- **% id_ku** *[integer]* :: Diagnostic handles.
- **% id_au** *[integer]* :: Diagnostic handles.
- **% id_le** *[integer]* :: Diagnostic handles.
- **% id_gamma_b** *[integer]* :: Diagnostic handles.
- **% id_gamma_t** *[integer]* :: Diagnostic handles.
- **% id_lrhines** *[integer]* :: Diagnostic handles.
- **% id_leady** *[integer]* :: Diagnostic handles.
- **% id_meke_equilibrium** *[integer]* :: Diagnostic handles.
- **% equilibrium_value** *[real(:,:),pointer]* :: The equilbrium value of MEKE to be calculated at each time step [L2 T-2 ~> m2 s-2].
- **% meke_frcoeff** *[real]* :: Efficiency of conversion of ME into MEKE [nondim].
- **% meke_gmcoeff** *[real]* :: Efficiency of conversion of PE into MEKE [nondim].
- **% meke_gmecoeff** *[real]* :: Efficiency of conversion of MEKE into ME by GME [nondim].
- **% meke_damping** *[real]* :: Local depth-independent MEKE dissipation rate [T-1 ~> s-1].

- **% meke_cd_scale** *[real]* :: The ratio of the bottom eddy velocity to the column mean eddy velocity, i.e. sqrt(2*MEKE). This should be less than 1 to account for the surface intensification of MEKE.
- **% meke_cb** *[real]* :: Coefficient in the
- **% meke_min_gamma** *[real]* :: Minimum value of gamma_b^2 allowed [nondim].
- **% meke_ct** *[real]* :: Coefficient in the
- **% visc_drag** *[logical]* :: If true use the vertvisc_type to calculate bottom drag.
- **% meke_geometric** *[logical]* :: If true, uses the GM coefficient formulation from the GEOMETRIC framework (Marshall et al., 2012)
- **% meke_geometric_alpha** *[real]* :: The nondimensional coefficient governing the efficiency of the GEOMETRIC thickness diffusion.
- **% meke_equilibrium_alt** *[logical]* :: If true, use an alternative calculation for the equilibrium value of MEKE.
- **% meke_equilibrium_restoring** *[logical]* :: If true, restore MEKE back to its equilibrium value, which is calculated at each time step.
- **% gm_src_alt** *[logical]* :: If true, use the GM energy conversion form S^2*N^2*kappa rather than the streamfunction for the MEKE GM source term.
- **% rd_as_max_scale** *[logical]* :: If true the length scale can not exceed the first baroclinic deformation radius.
- **% use_old_lscale** *[logical]* :: Use the old formula for mixing length scale.
- **% use_min_lscale** *[logical]* :: Use simple minimum for mixing length scale.
- **% cdrag** *[real]* :: The bottom drag coefficient for MEKE [nondim].
- **% meke_bgsrc** *[real]* :: Background energy source for MEKE [L2 T-3 ~> W kg-1] (= m2 s-3).
- **% meke_dtscale** *[real]* :: Scale factor to accelerate time-stepping [nondim].
- **% meke_khcoeff** *[real]* :: Scaling factor to convert MEKE into Kh [nondim].
- **% meke_uscale** *[real]* :: MEKE velocity scale for bottom drag [L T-1 ~> m s-1].
- **% meke_kh** *[real]* :: Background lateral diffusion of MEKE [L2 T-1 ~> m2 s-1].
- **% meke_k4** *[real]* :: Background bi-harmonic diffusivity (of MEKE) [L4 T-1 ~> m4 s-1].
- **% khmeke_fac** *[real]* :: A factor relating MEKEKh to the diffusivity used for MEKE itself [nondim].
- **% viscosity_coeff_ku** *[real]* :: The scaling coefficient in the expression for viscosity used to parameterize lateral harmonic momentum mixing by unresolved eddies represented by MEKE.
- **% viscosity_coeff_au** *[real]* :: The scaling coefficient in the expression for viscosity used to parameterize lateral biharmonic momentum mixing by unresolved eddies represented by MEKE.
- **% lfixed** *[real]* :: Fixed mixing length scale [L ~> m].
- **% adeform** *[real]* :: Weighting towards deformation scale of mixing length [nondim].
- **% arhines** *[real]* :: Weighting towards Rhines scale of mixing length [nondim].
- **% africt** *[real]* :: Weighting towards frictional arrest scale of mixing length [nondim].

- **% aeady** *[real]* :: Weighting towards Eady scale of mixing length [nondim].
- **% agrid** *[real]* :: Weighting towards grid scale of mixing length [nondim].
- **% meke_advection_factor** *[real]* :: A scaling in front of the advection of MEKE [nondim].
- **% meke_topographic_beta** *[real]* :: Weight for how much topographic beta is considered when computing beta in Rhines scale [nondim].
- **% meke_restoring_rate** *[real]* :: Inverse of the timescale used to nudge MEKE toward its equilibrium value [s-1].
- **% kh_flux_enabled** *[logical]* :: If true, lateral diffusive MEKE flux is enabled.
- **% initialize** *[logical]* :: If True, invokes a steady state solver to calculate MEKE.
- **% debug** *[logical]* :: If true, write out checksums of data for debugging.
- **% diag** *[type(diag_ctrl),pointer]* :: A type that regulates diagnostics output.
- **% id_clock_pass** *[integer]* :: Clock for group pass calls.
- **% pass_meke** *[type(group_pass_type)]* :: Group halo pass handle for MEKEMEKE and maybe MEKEKh_diff.
- **% pass_kh** *[type(group_pass_type)]* :: Group halo pass handle for MEKEKh, MEKEKu, and/or MEKEAu.

Function/Subroutine Documentation

subroutine mom_meke/**step_forward_meke**(*MEKE*, *h*, *SN_u*, *SN_v*, *visc*, *dt*, *G*, *GV*, *US*, *CS*, *hu*, *hv*)

Integrates forward-in-time the MEKE eddy energy equation. See *MEKE equations* .

Parameters

- **meke** :: MEKE data.
- **g** :: [inout] Ocean grid.
- **gv** :: [in] Ocean vertical grid structure.
- **us** :: [in] A dimensional unit scaling type
- **h** :: [in] Layer thickness [H ~> m or kg m-2].
- **sn_u** :: [in] Eady growth rate at u-points [T-1 ~> s-1].
- **sn_v** :: [in] Eady growth rate at v-points [T-1 ~> s-1].
- **visc** :: [in] The vertical viscosity type.
- **dt** :: [in] Model(baroclinic) time-step [T ~> s].
- **cs** :: MEKE control structure.
- **hu** :: [in] Accumlated zonal mass flux [H L2 ~> m3 or kg].
- **hv** :: [in] Accumlated meridional mass flux [H L2 ~> m3 or kg]

Call to *meke_equilibrium meke_equilibrium_restoring meke_lengthscales*

subroutine mom_meke/**meke_equilibrium**(*CS*, *MEKE*, *G*, *GV*, *US*, *SN_u*, *SN_v*, *drag_rate_visc*, *I_mass*)

Calculates the equilibrium solutino where the source depends only on MEKE diffusivity and there is no lateral diffusion of MEKE. Results is in MEKEMEKE.

Parameters

- **g** :: [inout] Ocean grid.
- **gv** :: [in] Ocean vertical grid structure.
- **us** :: [in] A dimensional unit scaling type
- **cs** :: MEKE control structure.
- **meke** :: A structure with MEKE data.
- **sn_u** :: [in] Eady growth rate at u-points [T-1 ~> s-1].
- **sn_v** :: [in] Eady growth rate at v-points [T-1 ~> s-1].
- **drag_rate_visc** :: [in] Mean flow velocity contribution to the MEKE drag rate [L T-1 ~> m s-1]
- **i_mass** :: [in] Inverse of column mass [R-1 Z-1 ~> m2 kg-1].

Call to *meke_lengthscales_0d*

Called from *step_forward_meke*

subroutine mom_meke/**meke_equilibrium_restoring**(*CS*, *G*, *US*, *SN_u*, *SN_v*)

Parameters

- **g** :: [inout] Ocean grid.
- **us** :: [in] A dimensional unit scaling type.
- **cs** :: MEKE control structure.
- **sn_u** :: [in] Eady growth rate at u-points [T-1 ~> s-1].
- **sn_v** :: [in] Eady growth rate at v-points [T-1 ~> s-1].

Called from *step_forward_meke*

subroutine mom_meke/**meke_lengthscales**(*CS*, *MEKE*, *G*, *GV*, *US*, *SN_u*, *SN_v*, *EKE*, *bottomFac2*, *barotrFac2*, *LmixScale*)

Calculates the eddy mixing length scale and γ_b and γ_t functions that are ratios of either bottom or barotropic eddy energy to the column eddy energy, respectively. See *MEKE equations* .

Parameters

- **cs** :: MEKE control structure.
- **meke** :: MEKE data.
- **g** :: [inout] Ocean grid.
- **gv** :: [in] Ocean vertical grid structure.
- **us** :: [in] A dimensional unit scaling type
- **sn_u** :: [in] Eady growth rate at u-points [T-1 ~> s-1].
- **sn_v** :: [in] Eady growth rate at v-points [T-1 ~> s-1].
- **eke** :: [in] Eddy kinetic energy [L2 T-2 ~> m2 s-2].
- **bottomfac2** :: [out] gamma_b^2
- **barotrfac2** :: [out] gamma_t^2
- **lmixscale** :: [out] Eddy mixing length [L ~> m].

Call to *meke_lengthscales_0d*

Called from *step_forward_meke*

subroutine mom_meke/**meke_lengthscales_0d**(*CS*, *US*, *area*, *beta*, *depth*, *Rd_dx*, *SN*, *EKE*, *bottomFac2*, *barotrFac2*, *LmixScale*, *Lrhines*, *Leady*)

Calculates the eddy mixing length scale and γ_b and γ_t functions that are ratios of either bottom or barotropic eddy energy to the column eddy energy, respectively. See *MEKE equations* .

Parameters

- **cs** :: MEKE control structure.
- **us** :: [in] A dimensional unit scaling type
- **area** :: [in] Grid cell area [L2 ~> m2]
- **beta** :: [in] Planetary beta = |grad F| [T-1 L-1 ~> s-1 m-1]
- **depth** :: [in] Ocean depth [Z ~> m]
- **rd_dx** :: [in] Resolution Ld/dx [nondim].
- **sn** :: [in] Eady growth rate [T-1 ~> s-1].
- **eke** :: [in] Eddy kinetic energy [L2 T-2 ~> m2 s-2].
- **bottomfac2** :: [out] gamma_b^2
- **barotrfac2** :: [out] gamma_t^2
- **lmixscale** :: [out] Eddy mixing length [L ~> m].
- **lrhines** :: [out] Rhines length scale [L ~> m].
- **leady** :: [out] Eady length scale [L ~> m].

Called from *meke_equilibrium meke_lengthscales*

function mom_meke/**meke_init**(*Time*, *G*, *US*, *param_file*, *diag*, *CS*, *MEKE*, *restart_CS*) *[logical]*

Initializes the MOM_MEKE module and reads parameters. Returns True if module is to be used, otherwise returns False.

Parameters

- **time** :: [in] The current model time.
- **g** :: [inout] The ocean's grid structure.
- **us** :: [in] A dimensional unit scaling type
- **param_file** :: [in] Parameter file parser structure.
- **diag** :: [inout] Diagnostics structure.
- **cs** :: MEKE control structure.
- **meke** :: MEKE-related fields.
- **restart_cs** :: Restart control structure for MOM_MEKE.

subroutine mom_meke/**meke_alloc_register_restart**(*HI*, *param_file*, *MEKE*, *restart_CS*)

Allocates memory and register restart fields for the MOM_MEKE module.

Parameters

- **hi** :: [in] Horizontal index structure
- **param_file** :: [in] Parameter file parser structure.
- **meke** :: A structure with MEKE-related fields.

- **restart_cs** :: Restart control structure for MOM_MEKE.

subroutine mom_meke/**meke_end**(*MEKE*, *CS*)
Deallocates any variables allocated in MEKE_init or MEKE_alloc_register_restart.

Parameters

- **meke** :: A structure with MEKE-related fields.
- **cs** :: The control structure for MOM_MEKE.

13.1.6 mom_unit_scaling module reference

Provides a transparent unit rescaling type to facilitate dimensional consistency testing.

More...

Data Types

unit_scale_type	Describes various unit conversion factors.

Functions/Subroutines

unit_scaling_init()	Allocates and initializes the ocean model unit scaling type.
fix_restart_unit_scaling()	Set the unit scaling factors for output to restart files to the unit scaling factors for this run.
unit_scaling_end()	Deallocates a unit scaling structure.

Detailed Description

Provides a transparent unit rescaling type to facilitate dimensional consistency testing.

Type Documentation

type mom_unit_scaling/**unit_scale_type**
Describes various unit conversion factors.

Type fields

- **% m_to_z** *[real]* :: A constant that translates distances in meters to the units of depth.
- **% z_to_m** *[real]* :: A constant that translates distances in the units of depth to meters.
- **% m_to_l** *[real]* :: A constant that translates lengths in meters to the units of horizontal lengths.
- **% l_to_m** *[real]* :: A constant that translates lengths in the units of horizontal lengths to meters.
- **% s_to_t** *[real]* :: A constant that translates time intervals in seconds to the units of time.
- **% t_to_s** *[real]* :: A constant that translates the units of time to seconds.
- **% r_to_kg_m3** *[real]* :: A constant that translates the units of density to kilograms per meter cubed.

- **% kg_m3_to_r** *[real]* :: A constant that translates kilograms per meter cubed to the units of density.
- **% q_to_j_kg** *[real]* :: A constant that translates the units of enthalpy to Joules per kilogram.
- **% j_kg_to_q** *[real]* :: A constant that translates Joules per kilogram to the units of enthalpy.
- **% z_to_l** *[real]* :: Convert vertical distances to lateral lengths.
- **% l_to_z** *[real]* :: Convert lateral lengths to vertical distances.
- **% l_t_to_m_s** *[real]* :: Convert lateral velocities from L T-1 to m s-1.
- **% m_s_to_l_t** *[real]* :: Convert lateral velocities from m s-1 to L T-1.
- **% l_t2_to_m_s2** *[real]* :: Convert lateral accelerations from L T-2 to m s-2.
- **% z2_t_to_m2_s** *[real]* :: Convert vertical diffusivities from Z2 T-1 to m2 s-1.
- **% m2_s_to_z2_t** *[real]* :: Convert vertical diffusivities from m2 s-1 to Z2 T-1.
- **% w_m2_to_qrz_t** *[real]* :: Convert heat fluxes from W m-2 to Q R Z T-1.
- **% qrz_t_to_w_m2** *[real]* :: Convert heat fluxes from Q R Z T-1 to W m-2.
- **% rz_to_kg_m2** *[real]* :: Convert mass loads from R Z to kg m-2.
- **% kg_m2s_to_rz_t** *[real]* :: Convert mass fluxes from kg m-2 s-1 to R Z T-1.
- **% rz_t_to_kg_m2s** *[real]* :: Convert mass fluxes from R Z T-1 to kg m-2 s-1.
- **% rz3_t3_to_w_m2** *[real]* :: Convert turbulent kinetic energy fluxes from R Z3 T-3 to W m-2.
- **% w_m2_to_rz3_t3** *[real]* :: Convert turbulent kinetic energy fluxes from W m-2 to R Z3 T-3.
- **% rl2_t2_to_pa** *[real]* :: Convert pressures from R L2 T-2 to Pa.
- **% m_to_z_restart** *[real]* :: A copy of the m_to_Z that is used in restart files.
- **% m_to_l_restart** *[real]* :: A copy of the m_to_L that is used in restart files.
- **% s_to_t_restart** *[real]* :: A copy of the s_to_T that is used in restart files.
- **% kg_m3_to_r_restart** *[real]* :: A copy of the kg_m3_to_R that is used in restart files.
- **% j_kg_to_q_restart** *[real]* :: A copy of the J_kg_to_Q that is used in restart files.

Function/Subroutine Documentation

subroutine mom_unit_scaling/**unit_scaling_init**(*param_file*, *US*)
Allocates and initializes the ocean model unit scaling type.

Parameters

- **param_file** :: [in] Parameter file handle/type
- **us** :: A dimensional unit scaling type

Called from *mom_ice_shelf::initialize_ice_shelf*

subroutine mom_unit_scaling/**fix_restart_unit_scaling**(*US*)
Set the unit scaling factors for output to restart files to the unit scaling factors for this run.

Parameters **us** :: [inout] A dimensional unit scaling type

Called from *mom_ice_shelf::initialize_ice_shelf*

subroutine mom_unit_scaling/**unit_scaling_end**(*US*)
Deallocates a unit scaling structure.

Parameters **us** :: A dimensional unit scaling type

CHAPTER

FOURTEEN

BIBLIOGRAPHY

CHAPTER

FIFTEEN

INDICES AND TABLES

- genindex
- search

BIBLIOGRAPHY

[1] A. Adcroft, W. Anderson, V. Balaji, C. Blanton, M. Bushuk, C. O. Dufour, J. P. Dunne, S. M. Griffies, R. Hallberg, M. J. Harrison, I. M. Held, M. F. Jansen, J. G. John, J. P. Krasting, A. R. Langenhorst, S. Legg, Z. Liang, C. McHugh, A. Radhakrishnan, B. G. Reichl, T. Rosati, B. L. Samuels, A. Shao, R. Stouffer, M. Winton, A. T. Wittenberg, B. Xiang, N. Zadeh, and R. Zhang. The GFDL global ocean and sea ice model OM4.0: model description and simulation features. *J. Adv. Mod. Earth Sys.*, 11(10):3167–3211, 2019. doi:10.1029/2019ms001726.

[2] Alistair Adcroft and Robert Hallberg. On methods for solving the oceanic equations of motion in generalized vertical coordinates. *Ocean Modelling*, 11(1-2):224–233, January 2006. URL: http://www.sciencedirect.com/science/article/pii/S1463500305000090, doi:10.1016/j.ocemod.2004.12.007.

[3] Alistair Adcroft, Robert Hallberg, and Matthew Harrison. A finite volume discretization of the pressure gradient force using analytic integration. *Ocean Modelling*, 22(3-4):106–113, January 2008. URL: http://www.sciencedirect.com/science/article/pii/S1463500308000243, doi:10.1016/j.ocemod.2008.02.001.

[4] Akio Arakawa and Yueh-Jiuan G. Hsu. Energy conserving and potential-enstrophy dissipating schemes for the shallow water equations. *Monthly Weather Review*, 118:1960–1969, 1990. doi:10.1175/1520-0493(1990)118<1960:ECAPED>2.0.CO;2.

[5] Akio Arakawa and Vivian R. Lamb. A Potential Enstrophy and Energy Conserving Scheme for the Shallow Water Equations. *Monthly Weather Review*, 109(1):18–36, January 1981. URL: https://journals.ametsoc.org/doi/abs/10.1175/1520-0493(1981)109%3C0018:APEAEC%3E2.0.CO%3B2, doi:10.1175/1520-0493(1981)109<0018:APEAEC>2.0.CO;2.

[6] Rainer Bleck. An oceanic general circulation model framed in hybrid isopycnic-Cartesian coordinates. *Ocean Modelling*, 4(1):55–88, 2002. URL: http://www.sciencedirect.com/science/article/pii/S1463500301000129, doi:10.1016/S1463-5003(01)00012-9.

[7] Rainer Bleck and Linda T. Smith. A wind-driven isopycnic coordinate model of the north and equatorial atlantic ocean: 1. model development and supporting experiments. *JGR Oceans*, 95:3273–3285, 1990. doi:10.1029/JC095iC03p03273.

[8] Jr. Carpenter, Richard L., Kelvin K. Droegemeier, Paul R. Woodward, and Carl E. Hane. Application of the piecewise parabolic method (ppm) to meteorological modeling. *Monthly Weather Review*, 118:586—612, 1990. doi:https://doi.org/10.1175/1520-0493(1990)118<0586:AOTPPM>2.0.CO;2.

[9] Phillip Colella and Paul R Woodward. The Piecewise Parabolic Method (PPM) for gas-dynamical simulations. *Journal of Computational Physics*, 54(1):174–201, April 1984. URL: https://www.sciencedirect.com/science/article/pii/0021999184901438, doi:10.1016/0021-9991(84)90143-8.

[10] John K. Dukowicz and John R. Baumgardner. Incremental Remapping as a Transport/Advection Algorithm. *Journal of Computational Physics*, 160(1):318–335, May 2000. URL: http://www.sciencedirect.com/science/article/pii/S0021999100964659, doi:10.1006/jcph.2000.6465.

[11] Dale R. Durran. *Numerical Methods for Fluid Dynamics With Applications to Geophysics*. Springer-Verlag New York, 2010. doi:10.1007/978-1-4419-6412-0.

[12] Richard C. Easter. Two modified versions of bott's positive-definite numerical advection scheme. *Monthly Weather Review*, 121:297–304, 1993. doi:10.1175/1520-0493(1993)121<0297:TMVOBP>2.0.CO;2.

[13] B. Fox-Kemper, G. Danabasoglu, R. Ferrari, S. M. Griffies, R. W. Hallberg, M. M. Holland, M. E. Maltrud, S. Peacock, and B. L. Samuels. Parameterization of mixed layer eddies. III: Implementation and impact in global ocean climate simulations. *Ocean Modelling*, 39(1):61–78, January 2011. URL: http://www.sciencedirect.com/science/article/pii/S1463500310001290, doi:10.1016/j.ocemod.2010.09.002.

[14] B. Fox-Kemper and R. Ferrari. Parameterization of mixed layer eddies. part ii: prognosis and impact. *J. Phys. Oceangraphy*, 38:1166–1179, 2008. doi:10.1175/2007JPO3788.1.

[15] B. Fox-Kemper, R. Ferrari, and R. Hallberg. Parameterization of mixed layer eddies. part i: theory and diagnosis. *J. Phys. Oceangraphy*, 38:1145–1165, 2008. doi:10.1175/2007JPO3792.1.

[16] Peter R. Gent and James C. Mcwilliams. Isopycnal mixing in ocean circulation models. *J. Phys. Oceanogr.*, 20:150–155, 1990. doi:10.1175/1520-0485(1990)020<0150:IMIOCM>2.0.CO;2.

[17] Peter R. Gent, Jurgen Willebrand, Trevor J. McDougall, and James C. McWilliams. Parameterizing Eddy-Induced Tracer Transports in Ocean Circulation Models. *Journal of Physical Oceanography*, 25(4):463–474, April 1995. URL: https://journals.ametsoc.org/doi/10.1175/1520-0485%281995%29025%3C0463%3APEITTI%3E2.0.CO%3B2, doi:10.1175/1520-0485(1995)025<0463:PEITTI>2.0.CO;2.

[18] Stephen M. Griffies. *Fundamentals of Ocean Climate Models*. Princeton University Press, Princeton, USA, 2004. 518+xxxiv pages.

[19] Stephen M. Griffies and Robert W. Hallberg. Biharmonic Friction with a Smagorinsky-Like Viscosity for Use in Large-Scale Eddy-Permitting Ocean Models. *Monthly Weather Review*, 128(8):2935–2946, August 2000. URL: https://journals.ametsoc.org/doi/full/10.1175/1520-0493%282000%29128%3C2935%3ABFWASL%3E2.0.CO%3B2, doi:10.1175/1520-0493(2000)128<2935:BFWASL>2.0.CO;2.

[20] Robert Hallberg. Stable split time stepping schemes for large-scale ocean modeling. *Journal of Computational Physics*, 135:54–65, 1997. doi:DOI:10.1006/jcph.1997.5734.

[21] Robert Hallberg. Time integration of diapycnal diffusion and richardson number–dependent mixing in isopycnal coordinate ocean models. *Monthly Weather Review*, 128:1402–1419, 2000.

[22] Robert Hallberg. A thermobaric instability of lagrangian vertical coordinate ocean models. *Ocean Modelling*, 2005. doi:10.1016/j.ocemod.2004.01.001.

[23] Robert Hallberg and Alistair Adcroft. Reconciling estimates of the free surface height in Lagrangian vertical coordinate ocean models with mode-split time stepping. *Ocean Modelling*, 29(1):15–26, January 2009. URL: http://www.sciencedirect.com/science/article/pii/S1463500309000298, doi:10.1016/j.ocemod.2009.02.008.

[24] C. W. Hirt, A. A. Amsden, and J. L. Cook. An Arbitrary Lagrangian-Eulerian Computing Method for All Flow Speeds. *Journal of Computational Physics*, 135(2):203–216, August 1997. URL: http://www.sciencedirect.com/science/article/pii/S0021999197957028, doi:10.1006/jcph.1997.5702.

[25] H. T. Huynh. *Schemes and constraints for advection*. Volume 490. Springer, Berlin, Heidelberg, 1997. doi:10.1007/BFb0107151.

[26] David R. Jackett and Trevor J. McDougall. Minimal adjustment of hydrographic profiles to achieve static stability. *J. Atmos. Ocean. Tech.*, 12:381–389, 1995. doi:10.1175/1520-0426(1995)012<0381:MAOHPT>2.0.CO;2.

[27] L. Jackson, R. Hallberg, and S. Legg. A Parameterization of Shear-Driven Turbulence for Ocean Climate Models. *Journal of Physical Oceanography*, 38(5):1033–1053, May 2008. URL: https://journals.ametsoc.org/doi/10.1175/2007JPO3779.1, doi:10.1175/2007JPO3779.1.

[28] Malte F. Jansen, Alistair J. Adcroft, Robert Hallberg, and Isaac M. Held. Parameterization of eddy fluxes based on a mesoscale energy budget. *Ocean Modelling*, 92:28–41, August 2015. URL: http://www.sciencedirect.com/science/article/pii/S1463500315000967, doi:10.1016/j.ocemod.2015.05.007.

[29] W. G. Large, J. C. McWilliams, and S. C. Doney. Oceanic vertical mixing: A review and a model with a nonlocal boundary layer parameterization. *Reviews of Geophysics*, 32(4):363–403, 1994. URL: https://agupubs.onlinelibrary.wiley.com/doi/abs/10.1029/94RG01872, doi:10.1029/94RG01872.

[30] Shian-Jiann Lin, Winston C. Chao, Y. C. Sud, and G. K. Walker. A class of the van leer-type transport schemes and its application to the moisture transport in a general circulation model. *Mon. Wea. Rev.*, 122:1575–1593, 1994. doi:10.1175/1520-0493(1994)122<1575:ACOTVL>2.0.CO;2.

[31] D. P. Marshall and A. J. Adcroft. Parameterization of ocean eddies: potential vorticity mixing, energetics and arnold first stability theorem. *Ocean Modelling*, 32:188–204, 2010. doi:10.1016/j.ocemod.2010.02.001.

[32] Trevor J. McDougall and Peter C. McIntosh. The Temporal-Residual-Mean Velocity. Part II: Isopycnal Interpretation and the Tracer and Momentum Equations. *Journal of Physical Oceanography*, 31(5):1222–1246, May 2001. URL: https://journals.ametsoc.org/doi/10.1175/1520-0485%282001%29031%3C1222%3ATTRMVP%3E2.0.CO%3B2, doi:10.1175/1520-0485(2001)031<1222:TTRMVP>2.0.CO;2.

[33] Angelique Melet, Robert Hallberg, Sonya Legg, and Kurt Polzin. Sensitivity of the Ocean State to the Vertical Distribution of Internal-Tide-Driven Mixing. *Journal of Physical Oceanography*, 43(3):602–615, December 2012. URL: https://journals.ametsoc.org/doi/full/10.1175/JPO-D-12-055.1, doi:10.1175/JPO-D-12-055.1.

[34] F.J. Millero. Freezing point of seawater. In *Eight report of the Joint Panel on Oceanographic Tables and Standards (JPOTS)*, number 28, 29–35. UNESCO technical papers in marine sciences, 1978. Annex 6. URL: https://www.researchgate.net/publication/292574579_Freezing_point_of_seawater.

[35] Kurt L. Polzin. An abyssal recipe. *Ocean Modelling*, 30(4):298–309, January 2009. URL: http://www.sciencedirect.com/science/article/pii/S1463500309001565, doi:10.1016/j.ocemod.2009.07.006.

[36] Brandon G. Reichl and Robert Hallberg. A simplified energetics based planetary boundary layer (ePBL) approach for ocean climate simulations. *Ocean Modelling*, 132:112–129, December 2018. URL: http://www.sciencedirect.com/science/article/pii/S1463500318301069, doi:10.1016/j.ocemod.2018.10.004.

[37] F. Roquet, G. Madec, T. J. McDougall, and P. M. Barker. Accurate polynomial expressions for the density and specific volume of seawater using the teos-10 standard. *Ocean Modelling*, 90:29–43, 2015.

[38] Gary L. Russell and Jean A. Lerner. A new finite-differencing scheme for the tracer transport equation. *Journal of Applied Meteorology*, 20:1483–1498, 1981. doi:10.1175/1520-0450(1981)020<1483:ANFDSF>2.0.CO;2.

[39] Robert Sadourny. The Dynamics of Finite-Difference Models of the Shallow-Water Equations. *Journal of the Atmospheric Sciences*, 32(4):680–689, April 1975. URL: https://journals.ametsoc.org/doi/abs/10.1175/1520-0469%281975%29032%3C0680%3ATDOFDM%3E2.0.CO%3B2, doi:10.1175/1520-0469(1975)032<0680:TDOFDM>2.0.CO;2.

[40] Andrew Shao, Alistair Adcroft, Robert Hallberg, and Stephen Griffies. A new, general-coordinate, non-local neutral diffusion operator. 2019.

[41] A. F. Shchepetkin and J. C. McWilliams. The regional ocean modeling system (roms): a split-explicit, free-surface, topography-following coordinates oceanic model. *Ocean Modeling*, 9:347–404, 2005.

[42] L. C. St Laurent, H. L. Simmons, and S. R. Jayne. Estimating tidally driven mixing in the deep ocean. *Geophysical Research Letters*, 29(23):21–1–21–4, December 2002. URL: https://agupubs.onlinelibrary.wiley.com/doi/abs/10.1029/2002GL015633, doi:10.1029/2002GL015633.

[43] Shan Sun, Rainer Bleck, Claes Rooth, John Dukowicz, Eric Chassignet, and Peter Killworth. Inclusion of thermobaricity in isopycnic-coordinate ocean models. *Journal of Physical Oceanography*, 29:2719–2729, 1999.

[44] Geoffrey K. Vallis. *Atmospheric and Oceanic Fluid Dynamics: Fundamentals and Large-scale Circulation*. Cambridge University Press, Cambridge, 2nd edition, 2017. 946 + xxv pp.

[45] Martin Visbeck, John Marshall, Tom Haine, and Mike Spall. Specification of eddy transfer coefficients in coarse-resolution ocean circulation models. *J. Phys. Oceanogr.*, 27:381–402, 1997. doi:10.1175/1520-0485(1997)027<0381:SOETCI>2.0.CO;2.

[46] Laurent White and Alistair Adcroft. A high-order finite volume remapping scheme for nonuniform grids: The piecewise quartic method (PQM). *Journal of Computational Physics*, 227(15):7394–7422, July 2008. URL: http://www.sciencedirect.com/science/article/pii/S0021999108002593, doi:10.1016/j.jcp.2008.04.026.

[47] Laurent White, Alistair Adcroft, and Robert Hallberg. High-order regridding-remapping schemes for continuous isopycnal and generalized coordinates in ocean models. *Journal of Computational Physics*, 228(23):8665–8692, December 2009. URL: http://www.sciencedirect.com/science/article/pii/S0021999109004628, doi:10.1016/j.jcp.2009.08.016.

[48] Daniel G. Wright. An Equation of State for Use in Ocean Models: Eckart's Formula Revisited. *Journal of Atmospheric and Oceanic Technology*, 14(3):735–740, June 1997. URL: https://journals.ametsoc.org/doi/abs/10.1175/1520-0426%281997%29014%3C0735%3AAEOSFU%3E2.0.CO%3B2, doi:10.1175/1520-0426(1997)014<0735:AEOSFU>2.0.CO;2.

[49] IOC, SCOR, and IAPSO. *The international thermodynamic equation of seawater-2010: calculation and use of thermodynamic properties*. Intergovernmental Oceanographic Commission, Manuals and Guides No. 56, UNESCO, edition, 2010. 196pp.

FORTRAN MODULE INDEX

m

INDEX

A

C

E

F

G

I

M

Q

R

S

T

U