Requirements and Design
Implicit Vertical Mixing in the Ocean Core

MPAS Development Team

May 2, 2012

Contents

Chapter 1

Summary

Implicit vertical mixing is required in ocean models because vertical mixing between unstably stratified layers occurs at timescales faster that other model processes. The timestep requirement for explicit timestepping is usually set by the horizontal advective CFL condition. In order to include realistic vertical mixing without very small time steps, we must use operator splitting so that the vertical tracer diffusion term is treated with an implicit timestep, while the remaining terms of the tracer equation use an explicit method. This is almost identical to the implementation in POP (see Reference Manual p. 25), except that POP uses leapfrog for the explicit timestep, while MPAS-Ocean uses fourth-order Runge-Kutta. A reasonable test of implicit vertical mixing is simply vertical tracer diffusion with zero advection, which can be compared against one-dimensional analytic solutions.

Chapter 2

Requirements

2.1 Requirement: Pacanowski and Philander vertical mixing scheme

Date last modified: 2011/02/28
Contributors: Mark Petersen

This simply computes the value of the vertical viscosity and tracer diffusivity based on the local Richardson number, calculated from velocity and density fields.

2.2 Requirement: operator splitting and implicit solve

Date last modified: 2011/02/28
Contributors: Mark Petersen

The momentum vertical and tracer vertical diffusion terms must be solved implicitly. This will normally be used in z-level mode, but coding should work in both z-level and isopycnal modes.

Chapter 3

Algorithmic Formulations

3.1 Design Solution: Pacanowski and Philander vertical mixing scheme

Date last modified: 2011/02/28
Contributors: Mark Petersen

The formulation is identical to that presented in the POP reference manual, p. 49, except that velocities must be averaged from from cell edges to cell centers, rather than cell corners to cell edges, and in MPAS the column index loops within the cell index.

In this parameterization, the vertical diffusivity and viscosity are functions of the Richardson Number,

$$Ri = N^2 \left(\frac{\partial V}{\partial z} \right)^{-2} = -\frac{g}{\rho_0} \frac{\partial \rho}{\partial z} \left(\frac{\partial V}{\partial z} \right)^{-2}, \tag{3.1}$$

where $V = \sqrt{u^2 + v^2} = \sqrt{ke}$ is the velocity magnitude. The discrete version is

$$Ri_k^{top} = -\frac{g}{\rho_0} \frac{\rho^*_{k-1} - \rho^*_k}{\frac{1}{2}(h_{k-1} + h_k)} \left(\frac{u_{k-1} - u_k}{\frac{1}{2}(h_{k-1} + h_k)} \right)^{-2} \tag{3.2}$$

$$= -\frac{g}{\rho_0} \frac{(\rho^*_{k-1} - \rho^*_k)\frac{1}{2}(h_{k-1} + h_k)}{(u_{k-1} - u_k)^2 + \epsilon} \tag{3.3}$$

where *top* indicates a layer interface, ke is the cell-centered kinetic energy, ρ^*_k is the density in layer k adiabatically displaced to the surface, and ϵ is a small number to avoid dividing by zero. We will use $\rho_0 = 1000 m^3/kg$ as in POP.

The variable Ri^{top} must be available at cell edges for the viscoscity ν_v and at cell centers for the tracer diffusion κ_v. In addition, the computation of shear is native to the edges, while density is native to the cell centers. The design chapter lays out the steps to compute these variables.

The functional forms for vertical viscosity and diffusivity at each layer interface will be identical to POP:

$$\nu_v = \nu_{bkrd} + Rich\_mix/(1 + 5Ri)^2 \tag{3.4}$$

$$\kappa_v = \kappa_{bkrd} + (\nu_{bkrd} + Rich\_mix/(1 + 5Ri)^2)/(1 + 5Ri) \tag{3.5}$$

for $Ri >= 0$. For unstable stratification, $Ri < 0$ and the viscosity and diffusion are set to be very high. For implicit vertical mixing, it is set by the config_convective_visc and config_convective_diff input variables. For explicit vertical diffusion, it is based on the vertical diffusive CFL condition,

$$\nu_v = \kappa_v = \frac{1}{4}\frac{dz^2}{dt} \tag{3.6}$$

3.2 Design Solution: operator splitting and implicit solve

Date last modified: 2012/05/02
Contributors: Mark Petersen, Qingshan Chen

The continuous MPAS-Ocean tracer equation with operator splitting is

$$\frac{\partial h\varphi}{\partial t} + \nabla\cdot(h\varphi\mathbf{u}) + \frac{\partial}{\partial z}(h\varphi w) = \nabla\cdot(h\kappa_h\nabla\varphi) + (1-\lambda)h\frac{\partial}{\partial z}\left(\kappa_v\frac{\partial\varphi}{\partial z}\right) + \lambda h\frac{\partial}{\partial z}\left(\kappa_v\frac{\partial\varphi}{\partial z}\right), \tag{3.7}$$

where $\lambda = 1$ for fully implicit vertical mixing. Consolidating all of the terms to be solved explicitly and implicitly, we have

$$\frac{\partial h\varphi}{\partial t} = F_{exp}(t,h,\phi) + F_{imp}(t,h,\phi), \tag{3.8}$$

$$F_{imp}(t,h,\phi) = \lambda h\frac{\partial}{\partial z}\left(\kappa_v\frac{\partial\varphi}{\partial z}\right). \tag{3.9}$$

We now choose Runge-Kutta 4 for the explicit timestep, and backward Euler for the implicit timestep,

$$(h\varphi)^{n+1} = (h\varphi)^n + \frac{\Delta t}{6}(k_1+2k_2+2k_3+k_4) + \Delta t\lambda h^{n+1}\frac{\partial}{\partial z}\left(\kappa_v\frac{\partial\varphi^{n+1}}{\partial z}\right), \tag{3.10}$$

where $k_1 \ldots k_4$ are the RK4 slopes. Because h and φ are separate in the last term, we must solve for φ^{n+1} rather than $(h\varphi)^{n+1}$ in the implicit solve. To deal with this, we define a provisional variable $\widetilde{(h\varphi)}^{n+1}$ and separate the solve into two steps,

$$\widetilde{(h\varphi)}^{n+1} = (h\varphi)^n + \frac{\Delta t}{6}(k_1+2k_2+2k_3+k_4) \tag{3.11}$$

$$\varphi^{n+1} = \frac{\widetilde{(h\varphi)}^{n+1}}{h^{n+1}} + \Delta t\lambda\frac{\partial}{\partial z}\left(\kappa_v\frac{\partial\varphi^{n+1}}{\partial z}\right). \tag{3.12}$$

Note that $\widetilde{(h\varphi)}^{n+1}$ is the solution from the explicit RK4 in the current code, and h^{n+1} is known from the solution of the thickness equation. Rewrite 3.12 as

$$\left(1 - \Delta t\lambda\frac{\partial}{\partial z}\left(\kappa_v\frac{\partial}{\partial z}\right)\right)\varphi^{n+1} = \frac{\widetilde{(h\varphi)}^{n+1}}{h^{n+1}} \tag{3.13}$$

and add spacial discretization,

$$\left(1 - \Delta t\lambda\delta_z\kappa_v\delta_z\right)\varphi_k^{n+1} = \frac{\widetilde{(h\varphi)}_k^{n+1}}{h_k^{n+1}} \tag{3.14}$$

where δ_z is a first order finite difference $\delta_z\varphi_k = (\varphi_{k-1/2} - \varphi_{k+1/2})/h_k$ (index k increases downward). The second order derivative is then:

$$\delta_z \kappa \delta_z \varphi_k = \frac{\kappa_k^{top}(\varphi_{k-1} - \varphi_k)}{h_k^{n+1}(h_{k-1}^{n+1} + h_k^{n+1})/2} - \frac{\kappa_{k+1}^{top}(\varphi_k - \varphi_{k+1})}{h_k^{n+1}(h_k^{n+1} + h_{k+1}^{n+1})/2} \tag{3.15}$$

so that (3.14) is then

$$\varphi_k^{n+1} - \frac{2\Delta t\lambda}{h_k^{n+1}}\left[\frac{\kappa_k^{top}(\varphi_{k-1}^{n+1} - \varphi_k^{n+1})}{h_{k-1}^{n+1} + h_k^{n+1}} - \frac{\kappa_{k+1}^{top}(\varphi_k^{n+1} - \varphi_{k+1}^{n+1})}{h_k^{n+1} + h_{k+1}^{n+1}}\right] = \frac{\widetilde{(h\varphi)}_k^{n+1}}{h_k^{n+1}} \tag{3.16}$$

which can be rewritten as

$$A_k\varphi_{k-1}^{n+1} + B_k\varphi_k^{n+1} + C_k\varphi_{k+1}^{n+1} = \frac{\widetilde{(h\varphi)}^{n+1}}{h^{n+1}}, \quad k = 1 \ldots N \tag{3.17}$$

$$A_k = -\frac{2\Delta t\lambda\kappa_k^{top}}{h_k^{n+1}(h_{k-1}^{n+1} + h_k^{n+1})}, \quad k = 2 \ldots N \tag{3.18}$$

$$C_k = -\frac{2\Delta t\lambda\kappa_{k+1}^{top}}{h_k^{n+1}(h_k^{n+1} + h_{k+1}^{n+1})}, \quad k = 1 \ldots N-1 \tag{3.19}$$

$$A_1 = C_N = 0, \tag{3.20}$$

$$B_k = 1 - A_k - C_k \tag{3.21}$$

where κ_{k+1}^{top} is the tracer diffusion on the interface between layers k and $k+1$, and the bottom cell is N. This equation set is nearly identical to that in the POP reference manual, section 4.2.3. The boundary condition for the implicit tracer solve is no flux,

$$\frac{\partial\varphi}{\partial z} = 0 \tag{3.22}$$

at the top of cell 1 and the bottom of cell N (top of cell $N+1$). When this is implemented in the derivation from (3.17) to (3.22), the result is $A_1 = C_N = 0$ above.

We now proceed to operator splitting for the momentum equation. The continuous equation is

$$\begin{aligned}\frac{\partial u}{\partial t} + q(hu^{\perp}) + w\frac{\partial u}{\partial z} = &-\frac{1}{\rho_0}\nabla p - \nabla K + \nu_h(\nabla\delta + \mathbf{k}\times\nabla\eta) \\ &+(1-\lambda)\frac{\partial}{\partial z}\left(\nu_v\frac{\partial u}{\partial z}\right) + \lambda\frac{\partial}{\partial z}\left(\nu_v\frac{\partial u}{\partial z}\right).\end{aligned} \tag{3.23}$$

Continuing as in (3.8-3.10), we define a provisional variable $\tilde{u}^{n+1}$ and separate the solve into two steps,

$$\tilde{u}^{n+1} = u^n + \frac{\Delta t}{6}(k_1 + 2k_2 + 2k_3 + k_4) \tag{3.24}$$

$$u^{n+1} = \tilde{u}^{n+1} + \Delta t\lambda\frac{\partial}{\partial z}\left(\nu_v\frac{\partial u^{n+1}}{\partial z}\right). \tag{3.25}$$

Note that $\tilde{u}^{n+1}$ is the solution from the explicit RK4 in the current code. Rewrite (3.25) as

$$\left(1 - \Delta t \lambda \frac{\partial}{\partial z}\left(\nu_v \frac{\partial}{\partial z}\right)\right) u^{n+1} = \tilde{u}^{n+1} \tag{3.26}$$

and add spacial discretization,

$$\left(1 - \Delta t \lambda \delta_z \nu_v \delta_z\right) u_k^{n+1} = \tilde{u}_k^{n+1} \tag{3.27}$$

where δ_z is a first order finite difference. The tridiagonal solve is then

$$A_k u_{k-1}^{n+1} + B_k u_k^{n+1} + C_k u_{k+1}^{n+1} = \tilde{u}_k^{n+1}, \quad k = 1 \ldots N \tag{3.28}$$

$$A_k = -\frac{2\Delta t \lambda \nu_k^{top}}{h_k^{n+1}(h_{k-1}^{n+1} + h_k^{n+1})}, \quad k = 2 \ldots N \tag{3.29}$$

$$C_k = -\frac{2\Delta t \lambda \nu_{k+1}^{top}}{h_k^{n+1}(h_k^{n+1} + h_{k+1}^{n+1})}, \quad k = 1 \ldots N-1 \tag{3.30}$$

$$A_1 = C_N = 0, \tag{3.31}$$

$$B_k = 1 - A_k - C_k \tag{3.32}$$

Again, no flux boundary conditions $\partial u/\partial z = 0$ imply that $A_0 = C_N = 0$. In practice λ, the fraction vertical mixing done implicitly, is either one (fully implicit) or zero (fully explicit) so that variable does not show up in the code. Rather, there is a logical flag `config_implicit_vertical_mix` that serves that purpose.

3.2.1 Inclusion of bottom drag in the momentum implicit solve

Date last modified: 2012/05/02
Contributors: Mark Petersen, Mathew Maltrud, Qingshan Chen

The bottom boundary condition for the viscous term is

$$\nu_v \frac{\partial u}{\partial z} \rightarrow c_{drag}|u|u. \tag{3.33}$$

where c_{drag} is the dimensionless drag coefficient of the order 10^{-3}. Note that this is nonlinear in u, so we linearize by considering the speed $|u|$ to be at time level n and the velocity to at be time level $n+1$:

$$\nu_v \frac{\partial u^{n+1}}{\partial z} \rightarrow c_{drag}|u^n|u^{n+1}. \tag{3.34}$$

Applying the bottom boundary condition to (3.26) at level N, and discritizing the outside derivative first,

$$u_k^{n+1} - \Delta t \left[\frac{\left(\nu_v \frac{\partial u}{\partial z}\right)^{k-1/2} - \left(\nu_v \frac{\partial u}{\partial z}\right)^{k+1/2}}{h_k^{n+1}}\right] = \tilde{u}_k^{n+1} \tag{3.35}$$

Level $N + 1/2$ is the bottom boundary, and can be replaced with the drag term,

$$u_k^{n+1} - \Delta t \left[\left(\frac{\nu_v}{h_k^{n+1}} \frac{\partial u}{\partial z} \right)^{k-1/2} - \frac{c_{drag}|u^n|u^{n+1}}{h_k^{n+1}} \right] = \tilde{u}_k^{n+1}, \quad k = N \tag{3.36}$$

Discretizing the remaining derivative,

$$u_k^{n+1} - \Delta t \left[\frac{\nu_k^{top}(u_{k-1}^{n+1} - u_k^{n+1})}{h_k^{n+1}(h_{k-1}^{n+1} + h_k^{n+1})/2} - \frac{c_{drag}|u^n|u^{n+1}}{h_k^{n+1}} \right] = \tilde{u}_k^{n+1}, \quad k = N \tag{3.37}$$

so that the coefficients of the tridiagonal solve are revised as:

$$A_k u_{k-1}^{n+1} + B_k u_k^{n+1} + C_k u_{k+1}^{n+1} = \tilde{u}_k^{n+1}, \quad k = 1 \ldots N \tag{3.38}$$

$$A_k = -\frac{2\Delta t \lambda \nu_k^{top}}{h_k^{n+1}(h_{k-1}^{n+1} + h_k^{n+1})}, \quad k = 2 \ldots N \tag{3.39}$$

$$C_k = -\frac{2\Delta t \lambda \nu_{k+1}^{top}}{h_k^{n+1}(h_k^{n+1} + h_{k+1}^{n+1})}, \quad k = 1 \ldots N-1 \tag{3.40}$$

$$A_1 = C_N = 0, \tag{3.41}$$

$$B_k = 1 - A_k - C_k, \quad k = 1 \ldots N-1 \tag{3.42}$$

$$B_k = 1 - A_k + \Delta t c_{drag}|u^n|/h_k^{n+1}, \quad k = N \tag{3.43}$$

3.2.2 Inclusion of implicit terms in the surface forcing.

Date last modified: 2011/05/20
Contributors: Mark Petersen, Mathew Maltrud

Surface forcing terms often take on a similar form as the drag term in the previous section. For example, simple restoring of surface temperature takes the following form,

$$R = (T_{ref} - T)/\tau, \tag{3.44}$$

where T_{ref} is a reference temperature, and τ is a restoring timescale. Clearly, this linear form can easily be incorporated into the tridiagonal implicit solve. However, POP does not have this feature. Also, complications may occur when running fully coupled. We therefore defer this aspect of the problem to the future.

Chapter 4

Design and Implementation

4.1 Implementation: Pacanowski and Philander vertical mixing scheme

Date last modified: 2011/02/28
Contributors: Mark Petersen

Add the "rich" option to `config_vert_visc_type`. Add separate namelist for each vertical mixing type, so the vmix namelist is less confusing. That part of the Registry will look like:

```
namelist character vmix     config_vert_visc_type    const
namelist character vmix     config_vert_diff_type    const
namelist logical   vmix     config_implicit_vertical_mix  .true.
namelist real      vmix     config_convective_visc  1.0
namelist real      vmix     config_convective_diff  1.0
namelist real      vmix_const   config_vert_visc        2.5e-4
namelist real      vmix_const   config_vert_diff        2.5e-5
namelist real      vmix_rich    config_bkrd_vert_visc   1.0e-4
namelist real      vmix_rich    config_bkrd_vert_diff   1.0e-5
namelist real      vmix_rich    config_rich_mix         50
namelist real      vmix_tanh    config_max_visc_tanh    2.5e-1
namelist real      vmix_tanh    config_min_visc_tanh    1.0e-4
namelist real      vmix_tanh    config_max_diff_tanh    2.5e-2
namelist real      vmix_tanh    config_min_diff_tanh    1.0e-5
namelist real      vmix_tanh    config_zMid_tanh        -100
namelist real      vmix_tanh    config_zWidth_tanh      100
```

Right now, the vertical viscosity variable `vertViscTop(k)` is computed columnwise in subroutine `compute_tend`, and the vertical diffusivity variable `vertDiffTop(k)` is computed columnwise in subroutine `compute_scalar_tend`. We propose to change these to 3D variables computed in `compute_solve_diagnostics`. The revised registry would include:

```
var persistent real   RiTopOfCell ( nVertLevelsP1 nCells Time )       1 o  diagnostics
var persistent real   RiTopOfEdge ( nVertLevelsP1 nEdges Time )       1 o  diagnostics
```

```
var persistent real   vertViscTopOfEdge ( nVertLevelsP1 nEdges Time ) 1 o  diagnostics
var persistent real   vertDiffTopOfCell ( nVertLevelsP1 nCells Time ) 1 o  diagnostics
var persistent real   rhoDisplaced ( nVertLevels nCells Time )        2 o  state
```

A note will be added that in the future, arrays could be saved by computing these values columnwise on the fly. We feel that the full 3D allocation allows for clear and consolidated code, so is worthwhile in the prototype version. A new variable class, named diagnostics, with only one time level is used for these variables, in order to avoid the memory required for two time-level state class variables.

Right now subroutine `equation_of_state_jm` computes the density for all cells and levels. It will need to be modified to compute ρ^* values, density of a parcel after adiabatic displacement.

The following steps will be used to ensure that variables are computed at the correct locations:

1. compute density of parcel displaced to surface, ρ^*
2. drhoTopOfCell(k) $= \rho^*_{k-1} - \rho^*_k$
3. interpolate drhoTopOfCell to drhoTopOfEdge
4. duTopOfEdge(k) $= u_{k-1} - u_k$
5. interpolate duTopOfEdge to duTopOfCell
6. compute RiTopOfCell using drhoTopOfCell and duTopOfCell
7. compute vertDiffTopOfCell from RiTopOfCell
8. compute RiTopOfEdge using drhoTopOfEdge and duTopOfEdge
9. compute vertViscTopOfEdge from RiTopOfEdge

4.2 Implementation: operator splitting and implicit solve

Date last modified: 2011/02/28
Contributors: Mark Petersen

Add the following namelist option to the registry:

```
namelist logical   vmix     config_implicit_vertical_mix  .true.
```

so that .true. sets $\lambda = 1$ in 3.7. In the code the variable λ will not appear. Rather, the namelist variable `config_implicit_vertical_mix` would be used in if statements around the tridiagonal solve and the explicit vertical tracer diffusion term.

We will add eqn (3.14) and (3.27) to subroutine `rk4`, here within `if (config_implicit_vertical_mix)`

```
! END RK loop

block => domain % blocklist
do while (associated(block))
```

```
   if (config_implicit_vertical_mix) then

      call compute_vertical_mix_coeff(dt, state, mesh, diag)

      do iCell=1,nCells
         N=maxLevelCell(iCell)

         do k=1,N
            ! Compute A(k), C(k), R(k) for momentum
         enddo
         call tridiagonal_solve(u(:,iCell),A(2:N),C(1:N),A(1:N-1),R)

         do k=1,N
            ! Compute A(k), C(k), R(iTracer,k) for tracers
         enddo
         call tridiagonal_solve_mult(tracers(:,:,iCell),A(2:N),C(1:N),A(1:N-1),R)
      end do
   else
      do iCell=1,nCells
         do k=1,maxLevelCell
            tracers(:,k,iCell) = tracers(:,k,iCell) / h(k,iCell)
         end do
      end do
   endif

   call compute_solve_diagnostics(dt, state, mesh)
   call reconstruct(state, mesh)

   block => block % next
end do
```

The variables ρ^*, Ri, ν_v, and κ_v are computed in the new subroutine `compute_vertical_mix_coeff` rather than `compute_solve_diagnostics` in order to save computation. For implicit vertical mixing ($\lambda = 1$) one only needs ν_v and κ_v after the fourth RK stage.

We propose to use a standard tridiagonal solver for `tridiagonal_solve`. An additional subroutine `tridiagonal_solve_mult`, will include an internal loop over tracers to save computation. I think we should add both of these to the operator subdirectory. The POP tridiag solver, in `impvmixt` in module `vertical_mix.F90`, should not be copied, because it has horizontal indices as the innermost loop, includes partial bottom cells, and so is not general in its interface.

Chapter 5

Testing

5.1 Testing and Validation: Pacanowski and Philander vertical mixing scheme

Date last modified: 2011/02/28
Contributors: Mark Petersen

For an artificial initial condition with velocities and densities that are linear with depth, one can compute the expected values for Ri, vertical viscosity, and vertical diffusion, and make sure the model matches this. This would require a linear equation of state to predict ρ^*.

The equation of state routine with adiablatic displacement should be tested by itself.

5.2 Testing and Validation: operator splitting and implicit solve

Date last modified: 2011/02/28
Contributors: Mark Petersen

The tridiagonal solver subroutine may be tested with a stand-alone matrix and be compared with a solution from Matlab.

A reasonable test of implicit vertical mixing is simply vertical tracer diffusion with zero advection, which can be compared against one-dimensional analytic solutions.

In the end, a full global ocean simulation with topography should be run, and implicit vertical mixing should produce reasonable results.