

# pyOM 1.0 documentation

*Carsten Eden, KlimaCampus, University Hamburg, Germany*

12. June 2011

## Contents

<b>1</b>	<b>Introduction</b>	<b>2</b>
1.1	Summary . . . . .	2
1.2	Navier-Stokes equations . . . . .	3
1.3	Discretisation . . . . .	3
<b>2</b>	<b>Installation</b>	<b>5</b>
2.1	Fortran front end . . . . .	5
2.2	Python front end . . . . .	5
<b>3</b>	<b>Directory structure and model configuration</b>	<b>6</b>
3.1	Directory structure . . . . .	6
3.2	Model configuration . . . . .	7
3.3	Running and restarting the model . . . . .	8
3.3.1	Restarting Fortran front end . . . . .	8
3.3.2	Restarting Python front end . . . . .	8
3.4	Sample configurations for Fortran front end . . . . .	8
3.4.1	Non-hydrostatic configurations . . . . .	8
3.4.2	Hydrostatic configurations . . . . .	9
3.5	Sample configurations for Python front end . . . . .	10
<b>4</b>	<b>Parameterisations and boundary conditions</b>	<b>10</b>
4.1	Rigid lid and surface pressure . . . . .	10
4.2	Implicit free surface . . . . .	11
4.3	Harmonic friction and diffusivity . . . . .	11
4.4	Biharmonic friction and diffusivity . . . . .	11
4.5	No-slip or free-slip boundary condition . . . . .	12
4.6	Other boundary and initial conditions . . . . .	12
4.7	Bottom and interior friction . . . . .	12
4.8	Advection schemes . . . . .	12
4.9	Hydrostatic approximation and convective adjustment . . . . .	13

4.10	Meso-scale eddy parameterisation . . . . .	13
4.10.1	Residual Mean formulation . . . . .	13
4.10.2	Bolus velocity . . . . .	14
4.11	Large eddy simulations (LES) . . . . .	14
4.12	Non-linear viscosity . . . . .	14
4.13	Setting a background state . . . . .	14
<b>5</b>	<b>Diagnostic output</b>	<b>15</b>
5.1	Fortran front end . . . . .	15
5.1.1	Snapshots . . . . .	15
5.1.2	Passive tracer . . . . .	15
5.1.3	Lagrangian particles . . . . .	15
5.1.4	Zonal averages . . . . .	16
5.1.5	Time averages . . . . .	16
5.2	Python front end . . . . .	16

# 1 Introduction

## 1.1 Summary

pyOM (Python Ocean Model) is a numerical circulation ocean model which was written for educational purpose. It is not meant to be a realistic ocean model, but a simple and easy to use numerical tool to configure and to integrate idealized numerical simulations. The main assumptions or restrictions are

- the neglect of any thermodynamic equation, i.e. the density budget is considered only, although passive tracers and also salinity can be included
- the Boussinesq approximation in its form applicable to the ocean, i.e. mass conservation is replaced by volume conservation and buoyancy perturbations act in the gravity term of the Navier-Stokes equation only
- the rotating coordinate system is Cartesian together with a  $\beta$ -plane approximation.

On the other hand, fully non-hydrostatic situations as well as large-scale oceanic flows can be considered. Many idealised experiments and examples are preconfigured and can be easily chosen and modified using two alternative configuration methods based on Fortran90 or Python. Prerequisites for the installation is a Fortran 90 compiler and for the Fortran front end the NetCDF-library (since IO is realized mainly using the NetCDF format). For the Python front end, the numerical module `numpy` is required and a graphical user interface is provided. Both version are based on identical Fortran90 code which is vectorised and for which a moderate parallelisation is realized based on the MPI-library to enhance performance.

## 1.2 Navier-Stokes equations

The Navier-Stokes equations, the density conservation equation and the mass (or volume) conservation equation within the Boussinesq approximation in a rotating Cartesian coordinate system are solved numerically in pyOM. These equations are given here for reference

$$\frac{\partial u}{\partial t} = F_u - \frac{\partial p}{\partial x} \quad (1)$$

$$\frac{\partial v}{\partial t} = F_v - \frac{\partial p}{\partial y} \quad (2)$$

$$\epsilon \frac{\partial w}{\partial t} = \epsilon F_w - \frac{\partial p}{\partial z} - b \quad (3)$$

$$\frac{\partial b}{\partial t} = -\nabla \cdot (\mathbf{u}b) + K_h \nabla_h^2 b + K_v \frac{\partial^2 b}{\partial z^2} \quad (4)$$

$$0 = \nabla \cdot \mathbf{u} \quad (5)$$

with the (scaled) pressure  $p = p^*/\rho_0$ , where  $\rho_0$  is a constant reference density, the scaled density  $b = g\rho/\rho_0$  (corresponding to *negative* buoyancy) and

$$F_u = fv - f_h w - \mathbf{u} \cdot \nabla u + A_h \nabla_h^2 u + A_v \frac{\partial^2 u}{\partial z^2} - A_{biha}^h \nabla_h^4 u - A_{biha}^v \frac{\partial^2}{\partial z^2} \frac{\partial^2 u}{\partial z^2} \quad (6)$$

$$F_v = -fu - \mathbf{u} \cdot \nabla v + A_h \nabla_h^2 v + A_v \frac{\partial^2 v}{\partial z^2} - A_{biha}^h \nabla_h^4 v - A_{biha}^v \frac{\partial^2}{\partial z^2} \frac{\partial^2 v}{\partial z^2} \quad (7)$$

$$F_w = f_h u - \mathbf{u} \cdot \nabla w + A_h \nabla_h^2 w + A_v \frac{\partial^2 w}{\partial z^2} - A_{biha}^h \nabla_h^4 w - A_{biha}^v \frac{\partial^2}{\partial z^2} \frac{\partial^2 w}{\partial z^2} \quad (8)$$

The Boussinesq approximation is made, i.e. a constant reference density  $\rho_0$  is used everywhere except for the gravity force in the vertical momentum equation. Note that  $f$  is related to the vertical component of the Coriolis force while  $f_h$  to the horizontal component of the Coriolis force. Note also that the parameter  $\epsilon$  is set to one, and that the hydrostatic approximation can be made by setting  $\epsilon = f_h = 0$ . Since a rotating Cartesian coordinate system is used, a  $\beta$ -plane approximation is employed as well. That means that the Coriolis parameters  $f$  and  $f_h$  are given by

$$f = 2\Omega \sin \phi_0 + \frac{2\Omega}{r} y \cos \phi_0, \quad f_h = 2\Omega \cos \phi_0 - \frac{2\Omega}{r} y \sin \phi_0 \quad (9)$$

where  $r$  denotes Earth's radius,  $\phi_0$  a reference latitude and  $\Omega$  the rotational frequency of the earth.

## 1.3 Discretisation

Variables are discretised on a Arakawa C-grid. Pressure  $p$  and buoyancy are centered in a tracer box. On the eastern, northern and upper sides of these boxes, the zonal, meridional and vertical velocities are placed. The horizontal grid arrangement is shown in Fig. 1.

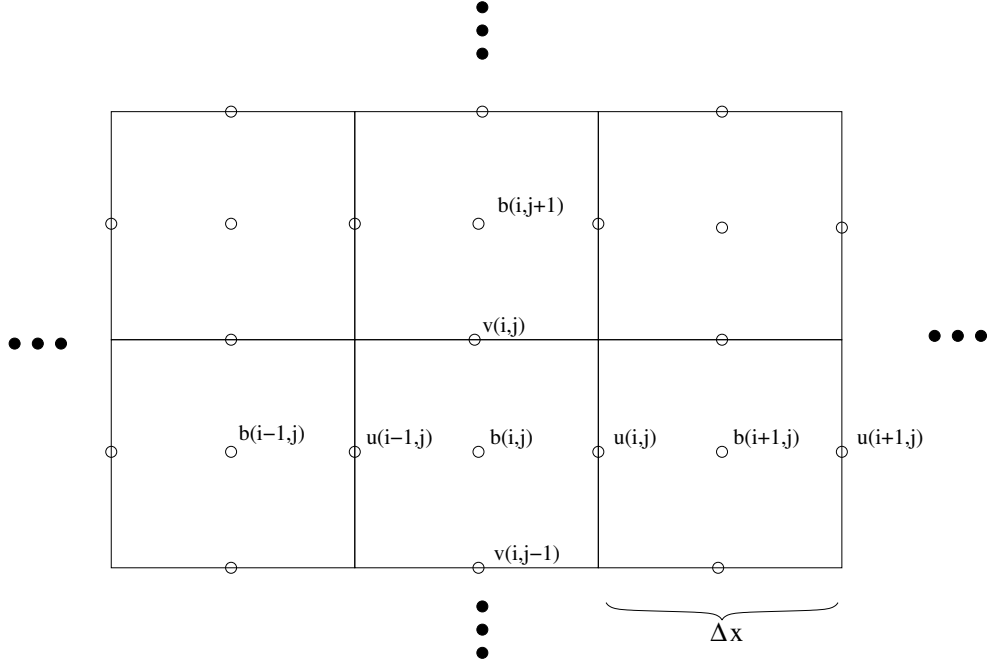


Figure 1: C-grid

Advective fluxes of density across the eastern cell border,  $F_{i,j,k}^E$  and the corresponding fluxes across the meridional and vertical boundary can be computed as

$$F_{i,j,k}^E = \Delta x \Delta z u_{i,j,k} (b_{i,j,k} + b_{i+1,j,k}) / 2 \quad (10)$$

$$F_{i,j,k}^N = \Delta x \Delta z v_{i,j,k} (b_{i,j,k} + b_{i,j+1,k}) / 2 \quad (11)$$

$$F_{i,j,k}^T = \Delta x^2 w_{i,j,k} (b_{i,j,k} + b_{i,j,k+1}) / 2 \quad (12)$$

Note that this discretisation represents the standard second order scheme, but that other higher order discretisation are possible. It is also obvious that diffusive fluxes can be discretised in a similar way. In any case, however, the convergence of these fluxes leads to a change in density content in the grid cell

$$(\Delta x^2 \Delta z) \frac{\partial}{\partial t} b_{i,j,k} = (F_{i,j,k}^E - F_{i-1,j,k}^E) + (F_{i,j,k}^N - F_{i,j-1,k}^N) + (F_{i,j,k}^T - F_{i,j,k-1}^T) \quad (13)$$

It is clear that fluxes of density across the boundaries must be set to zero. For momentum, free-slip or no-slip boundary conditions can be used.

The temporal discretisation is explicit (except for vertical diffusion, see below) using a so-called leapfrog time stepping for the advective terms as default (other advection schemes are possible) and a Robert's time filter to eliminate the numerical mode. The two-dimensional Poisson equation for the surface pressure (or implicit free surface, see below) and the three-dimensional equation for the non-hydrostatic pressure are solved using a simple preconditioned conjugate gradient solver.

## 2 Installation

This installation procedure assumes a Unix system and that all libraries and compilers are available. For both the Fortran and Python front ends of pyOM you first need to follow the following two steps:

- Make a new directory where the model code shall be placed. That directory is called `dir` hereafter.
- Extract the tar-archive containing pyOM in `dir`.

### 2.1 Fortran front end

The following steps are needed to use the Fortran front end of pyOM

- Edit the platform dependent options in the file `dir/for_src/Makefile`. See below for further comments.
- Change directory to `dir/for_config` and type `make <model>`, where `<model>` is set to one of the example Fortran files in `dir/for_src` (excluding the suffix `.f90`).
- Run the executable `<model>.x` in `dir/bin`, investigate the output and modify the model setup.

The platform dependent options in the file `dir/for_src/Makefile` contain information about the path to netcdf library and header files in the variable `CDFFLAGS` and information for the path to MPI library and header in `MPIFLAGS`. The name of the Fortran 90 compiler (usually `gfortran` or `ifort`) should be given by the variable `F90`. Some examples for common platforms are given in `dir/for_src/Makefile`. When the MPI library is not available, a non-parallel version of the Fortran front end of pyOM can be build by `make target=without_mpi <model>`. The netcdf library is, however, mandatory at the moment.

### 2.2 Python front end

For the Python front end of pyOM, a number of Python modules are needed. The numerical module `numpy` is mandatory, the modules `mpi4py`, `scipy`, `Tkinter`, `matplotlib` and `netcdf4` or `Scientific.IO.netcdf` are optional. For the Python front end of pyOM, it is necessary to compile an extension module for Python from the Fortran subroutines and modules. This can be done by executing `python setup.py install --install-lib=.` in the directory `dir`. Two extension modules will be build automatically, one with and one without MPI support. For the former, it is necessary to provide the path to the MPI library and include header files in the configuration file `dir/setup.py`.

When the extension module `dir/pyOM_code.so` is build, the Python front end can be used by executing `python <model>.py` in the directory `dir/py_config`, where `<model>`

denotes one of the example configurations in the directory `dir/py_config`. The extension module `dir/pyOM_code_MPI.so` contains MPI support and is loaded automatically when available. The examples in `dir/py_config` are based on different classes, which differ by using additional Python modules. The basic class is contained in the Python module `pyOM.py` and should work as long as one of the extension modules have been built. Extensions of this basic class are given by

- `pyOM_cdf.py` which adds netcdf based IO using the Python modules `netcdf4` or `Scientific.IO.netcdf`.
- `pyOM_mat.py` which adds Matlab file format IO, using the Python module `scipy`
- `pyOM_gui.py` which adds a graphical user interface using `Tkinter.py` and plotting during model execution using `matplotlib`. Note that parallel execution is not possible using this class.

The classes are imported by each of the examples in `dir/py_config`.

## 3 Directory structure and model configuration

### 3.1 Directory structure

The numerical code is distributed over several directories. Here is a list of these directories and a short characterisation of its content. All references to files or directories are relative to the directory in which `pyOM` is located.

- `./for_src`: Fortran subroutines used by both Fortran and Python front end
- `./py_src`: Python modules and extension modules
- `./py_config`: configuration examples of Python front end
- `./for_config`: configuration examples for Fortran front end
- `./doc`: contains documentation
- `./bin`: contains executable of Fortran front end after successful compilation

The top-level program of the Fortran front end is `./for_src/pyOM.f90`. Here, the work flow of the main routines can be seen. The basic class for the Python front end is in `./py_src/pyOM.py`, which does essentially the same as `./for_src/pyOM.f90`. The main module `./model/pyOM_module.f90` is available to both front ends and contains all important parameters and model variables. A short description of each model variable can be found in this file.

## 3.2 Model configuration

The model configuration is realized by template subroutines (Fortran front end) or template methods (Python front end) for which specific realisations are supplied for each specific experiment. The idea is to keep the whole model configuration in a single file containing all relevant routines. Specific examples can be found in the directory `./py_config` and `./for_config`. Template Fortran routines and Python methods are named identical. A list of them is given here.

- **set\_parameter:**  
sets all important fixed model parameters, like domain size, resolution, etc. The configuration is specified mainly by logical switches, implementing mixing parameterisation, hydrostatic approximation, etc. This routine/method is called only once before allocating the model variables.
- **set\_coriolis:**  
sets the (vertical and horizontal) Coriolis parameter. This routine/method is only called once during the model setup.
- **initial\_conditions**  
sets the initial conditions. This routine/method is only called once during the model setup.
- **topography**  
set the topography. This routine/method is only called once during the model setup.
- **boundary\_conditions**  
set the surface boundary conditions. This routine/method is called for each time step.
- **restoring\_zones**  
sets the interior sources and sinks for density. This routine/method is called for each time step.
- **momentum\_restoring\_zones**  
sets the interior sources and sinks for momentum. This routine/method is called for each time step.
- **tracer\_sources**  
set interior sources and sinks for the passive tracer. Also sets the surface boundary conditions. This routine/method is called for each time step.

The Fortran template routines have to be present in any case, even when they are empty. For the Python front end, only the methods which are actually changed are necessary.

### 3.3 Running and restarting the model

After successful compilation of the Fortran front end, the executable can be found in the directory `./bin`. The executable can be run directly in `./bin` or in any other directory. No further files are needed for a model integration. For the Python front end, the location of the module files have to be specified. This is done in the first line of each example.

#### 3.3.1 Restarting Fortran front end

The model integration will start from the initial conditions specified in the configuration templates and will integrate over a specified time period. After that period the integration stops and a restart file `restart.dta` will automatically be written by the Fortran front end, containing the last two time steps of each model variable. This restart file will be read automatically when the model is restarted. Note that the Fortran front end will always try to read a restart file, which then overrides the initial conditions specified in the template configuration routines. If there is no restart file present, the Fortran front end will use the initial conditions. Note also that at the end of each integration, any existing restart file will be overwritten.

#### 3.3.2 Restarting Python front end

Methods are implemented for the Python front end for reading and writing the restarts in the same format as the Fortran front end. In the GUI-version, the restarts can be written at any time.

### 3.4 Sample configurations for Fortran front end

This is an incomplete list of sample configurations which can be found in the directory `./for_config`.

#### 3.4.1 Non-hydrostatic configurations

- `kelv_helm1.f90`

This is a simple two-layer system with a large shear between the layers to demonstrate Kelvin-Helmholtz instabilities as in [Eden et al. \(2009\)](#). The domain is periodic in  $x$  and  $y$ , 14  $m$  deep, 16  $m$  long and there are no surface fluxes at the top or bottom. The initial conditions are two layers of equal thickness with a buoyancy difference corresponding to about 10  $K$  moving with 1  $m/s$  to the east and west. At the layer interface a small disturbance in buoyancy is introduced such that in the simulation Kelvin-Helmholtz instability will show up. The domain is two-dimensional, i.e. in the  $x$ - $z$  plane. The variable `fac` can be increased in order to increase the spatial (and temporal) resolution. For `fac=1.0` the spatial resolution is 25  $cm$  in each direction.

- `internal_wave1.f90`



This is a setup to demonstrate internal wave beams. The domain is two-dimensional, i.e. in the  $x$ - $z$  plane and in the center a wave maker is placed. The variable `fac` can be increased in order to increase the spatial (and temporal) resolution. The background state in terms of a stratification is prescribed using `enable_back_state` (see also below).

### 3.4.2 Hydrostatic configurations

- `barbi_exp1.f90`

Experiment DISTURB in [Olbers and Eden \(2003\)](#), i.e. a large scale buoyancy anomaly propagating westward in an ocean basin. The background state is prescribed using `enable_back_state` (see also below).

- `barbi_exp2.f90`

Experiment RIDGE in [Olbers and Eden \(2003\)](#), i.e. adjustment of a wind-driven gyre over topography resembling the North Atlantic Ridge to the flat-bottom solution. The background state is prescribed using `enable_back_state` (see also below).

- `wardlemarshall1.f90`

First experiment in [Wardle and Marshall \(2000\)](#), i.e. a reentrant channel driven by eastward wind stress profile.

- `jets1.f90`

A wide channel model with relaxation at the side walls and interior damping as in [Eden \(2010\)](#), simulating strong eddy-driven zonal jets.

- `eady1.f90`

The classical Eady problem ([Eady, 1949](#)). A narrow channel on an  $f$ -plane with prescribed stratification and vertically sheared background zonal flow. The background state is prescribed using `enable_back_state` (see also below).

- `acc1.f90`

A model with a channel attached to a closed basin, similar to the Southern and Atlantic Ocean as in [Viebahn and Eden \(2010\)](#). There is wind forcing over the channel part and buoyancy relaxation driving a large-scale meridional overturning circulation.

- `THC1.f90`

Thermohaline circulation using temperature relaxation and fixed freshwater flux. Salinity is implemented using a passive tracer (impacting the surface boundary condition for density only) using the switch `enable_diag_tracer` (see below).

### 3.5 Sample configurations for Python front end

to be continued

## 4 Parameterisations and boundary conditions

In this section, the most important sub-grid-scale parameterisations and boundary conditions which are used in the model and also other important aspect of the numerical implementation are listed and briefly discussed. Note that all references to files containing code are relative to the directory in which pyOM is located. It is also assumed that all parameters are given in SI units.

### 4.1 Rigid lid and surface pressure

To obtain the pressure in the Boussinesq system of equations, it is necessary to solve a diagnostic relation for the pressure. In case of the hydrostatic assumption, this relation simplified to one of the surface pressure only. The default treatment of the pressure is to assume a rigid lid, i.e.  $w = 0$  at  $z = 0$ , which is briefly explained here.

It is useful to split the pressure in a hydrostatically balanced part and deviation from that. Within the hydrostatic approximation, the pressure  $p$  is given by

$$\int_z^0 \frac{\partial}{\partial z'} p dz' = p(0) - p(z) = - \int_z^0 b dz' \quad , \quad p = p(0) + \int_z^0 b dz' \equiv p_s(x, y) + p_h(x, y, z) \quad (14)$$

where  $p_s(x, y)$  denotes the surface pressure and  $p_h = \int_z^0 b dz'$  the hydrostatic pressure. The surface pressure can be found from an elliptic equation given below. In general, however, the non-hydrostatic pressure  $p'$  will also add to give the full pressure  $p$

$$p(x, y, z) = p_s(x, y) + p_h(x, y, z) + \epsilon p'(x, y, z) \quad (15)$$

The non-hydrostatic pressure  $p'$  can be found by taking divergence of momentum equation

$$\frac{\partial}{\partial x} F_u + \frac{\partial}{\partial y} F_v + \frac{\partial}{\partial z} F_w - \nabla_h^2 (p_h + p_s) = \nabla^2 p' \quad (16)$$

A diagnostic equation for the surface pressure  $p_s$  can be found in case of a rigid lid from vertical integration of the momentum equation and the continuity equation

$$\nabla \cdot h \nabla p_s = \frac{\partial}{\partial x} \int_h^0 (F_u - (p_h + \epsilon p')_x) dz + \frac{\partial}{\partial y} \int_h^0 (F_v - (p_h + \epsilon p')_y) dz \quad (17)$$

Note that it is also possible to relax the rigid-lid assumption and to use a linearised free surface formulation.

## 4.2 Implicit free surface

An alternative surface boundary formulation to the rigid lid is given by the implicit free surface, which is activated by the switch `enable_free_surface` in the template configuration subroutine/method `set_parameter`. Using the implicit free surface option, the surface pressure is replaced by the free surface  $\eta$ . The free surface equation is given by

$$\frac{\partial}{\partial t}\eta + \nabla_h \cdot \int_{-h}^{\eta} \mathbf{u}_h dz \approx \frac{\partial}{\partial t}\eta + \nabla_h \cdot \int_{-h}^0 \mathbf{u}_h dz = 0 \quad (18)$$

where the free surface  $\eta$  is related to the surface pressure by  $p_s = g\eta$ . Discretisation of the time dependency yields

$$\eta^{n+1} + 2\Delta t \nabla_h \cdot \int_{-h}^0 \mathbf{u}_h^{n+1} dz = \eta^{n-1} \quad (19)$$

and with  $\mathbf{u}_h^{n+1} = \mathbf{u}_h^{n-1} + 2\Delta t(\mathbf{F}_h - \nabla_h p)$  this gives

$$\nabla_h \cdot g h \nabla_h \eta^{n+1} - \epsilon \eta^{n+1} = \nabla_h \cdot \int_{-h}^0 (\mathbf{F}_h - \nabla_h p_h - \nabla_h p') dz - \epsilon \eta^{n-1} \quad (20)$$

with  $\epsilon = (2\Delta t)^{-2}$ .

## 4.3 Harmonic friction and diffusivity

Harmonic friction acting on the horizontal velocity ( $u, v$ ) and harmonic diffusivity acting on the buoyancy ( $b$ ) is always present and controlled by the horizontal viscosity `a_h`, the vertical viscosity `a_v`, the horizontal diffusivity `k_h`, and the vertical diffusivity `k_v`. Default values are zero. Note that in case of a non-hydrostatic simulation, the harmonic vertical and horizontal viscosity is also acting on the vertical velocity ( $w$ ). Note also that all viscosities and diffusivities should be specified in the template configuration subroutine/method `set_parameter`.

## 4.4 Biharmonic friction and diffusivity

Biharmonic horizontal friction acting on the horizontal velocity (given e.g. by  $a_{biha} \nabla^2 u$  in Eq. (6)) and biharmonic horizontal diffusion acting on buoyancy can be implemented by setting the logical variable `enable_biharmonic_friction` and `enable_biharmonic_diffusion` respectively to a true value in the configuration subroutine/method `set_parameter`. The horizontal biharmonic viscosity and diffusivity are given by `ahbi` and `khbi` respectively. Default values are zero.

Vertical biharmonic friction and diffusion can be implemented by setting the logical variable `enable_vert_biha_friction` and `enable_vert_biha_diffusion` respectively to a true value in the configuration subroutine/method `set_parameter`. The vertical biharmonic viscosity and diffusivity are given by `avbi` and `kvbi` respectively. Note that in case of a non-hydrostatic simulation, the biharmonic vertical and horizontal viscosity is also acting on the vertical velocity ( $w$ ).

## 4.5 No-slip or free-slip boundary condition

Since the spatial discretisation is realized using a C-grid (see also section 1.3) it is natural to implement zero diffusive fluxes of momentum across boundaries of the model. This choice is equivalent a free-slip boundary condition and the default choice. No-slip lateral boundary conditions, assuming zero tangential velocities at the lateral boundaries implying a diffusive momentum flux across the lateral boundaries, are implemented setting the logical variable `enable_noslip` to a true value in the configuration subroutine/method `set_parameter`. Note that for the upper and lower boundaries free-slip conditions are always used. Note also that the no/free-slip conditions are implemented for harmonic and biharmonic friction.

## 4.6 Other boundary and initial conditions

The rigid-lid boundary condition is used as a default but a linearised implicit free surface formulation can be switched on by setting the the logical variable `enable_free_surface` to a true value in the configuration subroutine/method `set_parameter`. Lateral boundary conditions are either no-flux boundary condition for buoyancy and passive tracers (and no/free-slip for momentum as discussed above) or periodic boundary conditions. The latter can be chosen by setting the logical variables `enable_cyclic_x` and `enable_cyclic_y` to a true value in the configuration subroutine/method `set_parameter` to apply periodic boundary conditions in zonal and meridional direction respectively. Surface and eventually bottom fluxes of buoyancy and momentum can be specified in the configuration subroutine `boundary_conditions` and method `surface_boundary_conditions` The same holds for the initial conditions and the configuration subroutine `initial_conditions` and method `setup_initial_cond`.

## 4.7 Bottom and interior friction

Bottom friction can be implemented by setting the logical variable `enable_bottom_stress` to a true value in the configuration subroutine/method `set_parameter`. The inverse time scale for bottom friction is given by `cdbot`. Default value is zero. Adding interior Rayleigh damping is implemented by the switch `enable_interior_stress` and the parameter `cdint`. For the bottom, a no-slip boundary condition is implemented by the switch `enable_bottom_noslip`.

## 4.8 Advection schemes

Although the choice of the advection scheme is not a sub-grid-scale parameterisation, it strongly affects the dissipation in the model and is therefore discussed here as well. It is possible to choose between the classical second-order central differences scheme as the default, the first order upwind schemen, a forth-order accurate central difference scheme and the Quicker advection scheme. They are implemented by setting the logical variables `enable_upwind_advection`, `enable_4th_advection` or `enable_quicker_advection` to a

true value, respectively, in the configuration subroutine/method `set_parameter`. Note that this choice refers to the advection scheme of the tracers (buoyancy, passive tracer, TKE, etc) but not to the advection of momentum. The advection scheme for momentum is controlled by the variables `enable_4th_mom_advection` and `enable_quicker_mom_advection`, default is also the classical second-order central differences scheme.

Note that Quicker introduces a certain amount of numerical diffusion on buoyancy, such that sometimes no additional diffusion is needed in the model simulation. The standard and the forth-order accurate scheme do not contain any numerical diffusion, but are dispersive. The upwind scheme is very diffusive.

## 4.9 Hydrostatic approximation and convective adjustment

The hydrostatic approximation is switched on by setting the logical variable `enable_hydrostatic` to a true value in the configuration subroutine/method `set_parameter`. Note that when using the hydrostatic approximation the three-dimensional Poisson equation for the full pressure does not have to be solved, such that the model integration can be much faster.

Using the hydrostatic approximation, static instabilities are removed by setting the vertical diffusivity to a large value. This procedure is sometimes called convective adjustment. Vertical diffusion is calculated with a fully implicit time stepping scheme in that case. Find the code in the file `./for_src/convection.f90`. The model variable containing the large vertical diffusivity is called `K_b` and written in the snapshot NetCDF file.

## 4.10 Meso-scale eddy parameterisation

### 4.10.1 Residual Mean formulation

The effect of meso-scale eddies on the density budget is given by an additional eddy-driven advection velocity in the density and tracer equations. In pyOM, this eddy-driven velocity is implemented using the Residual Mean formulation by [Andrews et al. \(1987\)](#). In this formulation, the momentum budget is considered as the one for the Residual velocity, which is given by the sum of the eddy-driven velocity of the [Gent and McWilliams \(1990\)](#) parameterisation and the normal Eulerian Mean velocity. The necessary modification of the momentum budget is given by adding a vertical friction term with viscosity  $Kf^2/N^2$ , where  $K$  denotes the thickness diffusivity of the [Gent and McWilliams \(1990\)](#) parameterisation. Since  $Kf^2/N^2$  can become large, an implicit formulation for the vertical friction is necessary, analogous to the parameterisation of convection. The formulation is implemented in the file `./for_src/vert_friction.f90` and activated by the switch `enable_vert_friction` in the template routine/method `set_parameter`. The thickness diffusivity `k_gm` has to be specified in  $\text{m}^2/\text{s}$  in `set_parameter`, its default value is  $1000 \text{ m}^2/\text{s}$ . Further, a minimal threshold for the stability frequency  $N$  can be specified in  $1/\text{s}$  (variable `n_min`) and a maximal threshold for  $f^2/N^2$  (`fnsqr_max`).

Note that while the buoyancy equation does not need a further parameterisation for meso-scale eddies using the Residual Mean formulation, in the tracer equations isopyc-

nal diffusion should be added in addition to the eddy-driven velocity using the switch `enable_bolus_isopycnal_diffusion` in the template routine/method `set_parameter`. The corresponding isopycnal diffusivity is set to `k_gm`. Further, two parameters controlling the maximal allowed isopycnal slopes are specified, `iso_slopec` and `iso_dslope`, for which reasonable default values are prescribed (there is not much reason to modify these parameters). For the case that tracers like concentrations become negative by spurious undershooting caused by the isopycnal mixing scheme, it is possible to prevent this by the switch `delimit_tracer_fluxes`. Note that the implementation of isopycnal diffusion in pyOM follows the standard one by [Redi \(1982\)](#), which was shown by [Griffies \(1998\)](#) to be slightly unstable. [Griffies \(1998\)](#) also suggested a stable scheme, which is, however, rather complicated and is therefore not implemented.

#### 4.10.2 Bolus velocity

The standard implementation of the eddy-driven velocity and the [Gent and McWilliams \(1990\)](#) parameterisation can be found in the file `./for_src/bolus_velocity.f90` but is only available to the Fortran front end. It is activated by the switch `enable_bolus_velocity` in the template routine `set_parameter`. The thickness diffusivity `k_gm` has to be specified in  $\text{m}^2/\text{s}$ . Default value is  $1000 \text{ m}^2/\text{s}$ . Diagnostic output in form of the vector streamfunction for the eddy-driven velocity and the thickness diffusivity written to the NetCDF file `bolus.cdf` by activating by the switch `enable_bolus_diag` locally in the file `./for_src/bolus_velocity.f90`. Note that the implementation of the [Gent and McWilliams \(1990\)](#) parameterisation in pyOM follows the standard one as in [Pacanowski \(1995\)](#), which was shown by [Griffies \(1998\)](#) to be slightly unstable.

### 4.11 Large eddy simulations (LES)

to be continued

### 4.12 Non-linear viscosity

to be continued

### 4.13 Setting a background state

To add the effect of a prescribed background state use the switch `enable_back_state`. Zonal or meridional background can be implemented by the switches `enable_back_zonal_flow` and `enable_back_meridional_flow`. This background state is given by a stationary field of buoyancy and zonal or meridional flow (but not meridional and zonal flow) and is defined by the configuration subroutine `initial_conditions` and method `setup_initial_cond`. Density, pressure and velocity predicted by the model are perturbations on the specified background state for this formulation. Note that it is important that the background state

is balanced with respect to the underlying equations. Some examples are given in the examples.

## 5 Diagnostic output

### 5.1 Fortran front end

For each time step, the time and the number of iterations of the two- and three-dimensional Poisson solver are written to standard output (i.e on the screen). More complex diagnostic output is written by the Fortran front end in NetCDF format following usual conventions. The output can be easily visualised by e.g. the free software **ferret** available at <http://ferret.wrc.noaa.gov/Ferret>.

#### 5.1.1 Snapshots

Snapshots of the model variables are written by the Fortran front end subsequently to a NetCDF file **py0M.cdf**. Any existing file of this name will be overwritten during model start. The frequency of the output is controlled by the variable **snapint** in the configuration subroutine **set\_parameter**. The value of **snapint** gives the number of second between subsequent snapshots written to **py0M.cdf**.

#### 5.1.2 Passive tracer

The contemporary simulation of one or several passive tracer can be enabled by the switch **enable\_diag\_tracer**. Initial and surface boundary conditions and relaxation zones, sources and sinks, etc of the passive tracer are specified in template subroutines. Output is written to the NetCDF file **tracer.cdf** with frequency given by **snapint** in the configuration subroutine **set\_parameter**.

Note that at the beginning of the simulation, a file **tracer\_restart.dta** will be read if it exists, which contains the last two time steps for the passive tracers from a proceeding simulation with the same configuration. At the end of each simulation, the restart file **tracer\_restart.dta** will be overwritten.

#### 5.1.3 Lagrangian particles

This diagnostic is only available to the Fortran front end. The contemporary simulation of a number of Lagrangian particles (following  $d\mathbf{x}/dt = \mathbf{u}$ ) can be enabled by the switch **enable\_diag\_particles**. Initial distribution and total number of the particles are specified locally in the file **./for\_src/particles.f90**. Output is written to the NetCDF file **float.cdf** with frequency given by **snapint** in the configuration subroutine **set\_parameter**.

Note that at the beginning of the simulation, a file **restart\_float.dta** will be read if it exists, which contains the last two time steps for the passive tracers from a proceeding

simulation with the same configuration. At the end of each simulation, the restart file `restart_float.dta` will be overwritten.

#### 5.1.4 Zonal averages

This diagnostic is only available to the Fortran front end. Zonal average diagnostic can be enabled by the switch `enable_diag_zonalave`. Several local switches are also located in the file `./for_src/zonal_averages.f90` defining the specific zonally averaged output. Output is written to the NetCDF file `zonal_ave.cdf` with frequency given by `snapint` in the configuration subroutine `set_parameter`.

#### 5.1.5 Time averages

This diagnostic is only available to the Fortran front end. Time average diagnostic can be enabled by the respective switch `enable_diag_timeave`. Several local switches are also located in the file `./for_src/time_averages.f90` defining the specific output. Output is written to the NetCDF file `time_ave.cdf` with frequency given by `snapint` in the configuration subroutine `set_parameter`.

Note that at the beginning of the simulation, a file `restart_ave.dta` will be read if it exists, which contains the unfinished time averages from a proceeding simulation with the same configuration. At the end of each simulation, the restart file `restart_ave.dta` will be overwritten.

## 5.2 Python front end

The Python front end offers the possibility of online plotting by specifying the method `make_plot`. Examples are provided.

## Acknowledgments

The conjugate gradient solvers are taken from GFDL MOM-2 and MOM-3, although modified. The same holds for some auxilliary routines.

## References

- Andrews, D. G., J. R. Holton, and C. B. Leovy, 1987: *Middle Atmosphere Dynamics*. Academic Press.
- Eady, E., 1949: Long waves and cyclone waves. *Tellus*, **1**, 33–52.
- Eden, C., 2010: Parameterising meso-scale eddy momentum fluxes based on potential vorticity mixing and a gauge term. *Ocean Modelling*, **32**(1-2), 58–71.



- Eden, C., D. Olbers, and R. J. Greatbatch, 2009: A generalised Osborn-Cox relation. *J. Fluid Mech.*, **632**, 457–474.
- Gent, P. R. and J. C. McWilliams, 1990: Isopycnal mixing in ocean circulation models. *J. Phys. Oceanogr.*, **20**, 150–155.
- Griffies, S. M., 1998: The gent-mcwilliams skew flux. *J. Phys. Oceanogr.*, **28**, 831–841.
- Olbers, D. and C. Eden, 2003: A model with simplified circulation dynamics for a baroclinic ocean with topography. Part I: Waves and wind-driven circulations. *J. Phys. Oceanogr.*, **33**, 2719–2737.
- Pacanowski, R. C., 1995: MOM 2 Documentation, User’s Guide and Reference Manual. Technical report, GFDL Ocean Group, GFDL, Princeton, USA.
- Redi, M. H., 1982: Oceanic isopycnal mixing by coordinate rotation. *J. Phys. Oceanogr.*, **12**, 1154–1158.
- Viebahn, J. and C. Eden, 2010: Towards the impact of eddies on the response of the southern ocean to climate change. *Ocean Modelling*, **34**(3-4), 150–165.
- Wardle, R. and J. Marshall, 2000: Representation of Eddies in Primitive Equation Models by a PV Flux. *J. Phys. Oceanogr.*, **30**, 2481–2503.