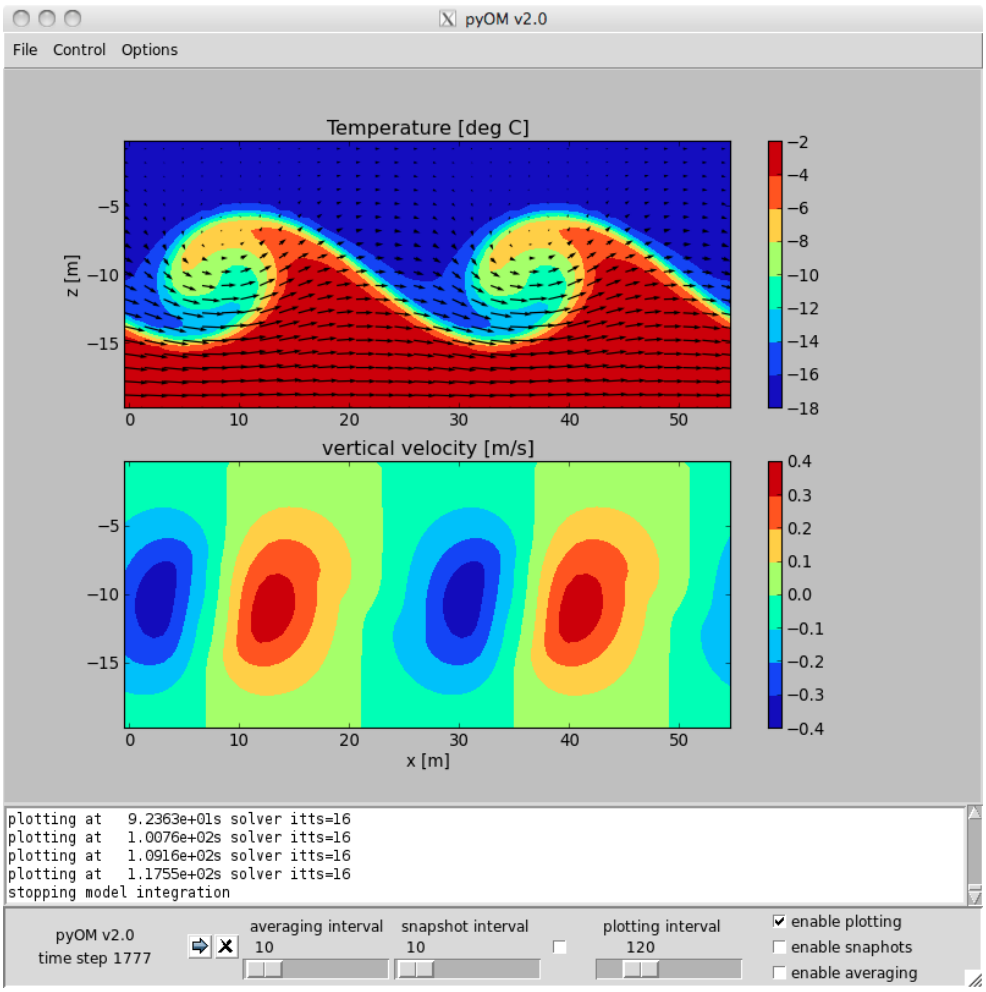
pyOM2.2 documentation

Carsten Eden

Institut für Meereskunde, Universität Hamburg, Germany

May 2017



Contents

1 Summary

pyOM2.2 (Python Ocean Model) is a numerical circulation ocean model which was written for educational purpose. It is meant to be a simple and easy to use numerical tool to configure and to integrate idealized and realistic numerical simulations of the ocean in Boussinesq approximation. Non-hydrostatic situations as well as large-scale oceanic flows can be considered, Cartesian or pseudo-spherical coordinate systems can be used. Several idealized 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 the Lapack library, and for the Fortran front the NetCDF-library (since IO is realized mainly using the NetCDF format). For the Python front end, the numerical module `numpy` is required and several other modules can be used in addition, e.g. to provide a graphical user interface. Both version are based on identical Fortran90 code which is fully parallelized based on the MPI-library to enhance performance. The code can be freely downloaded[1] but there is usually no support provided.

2 Installation

The following installation procedure to compile pyOM2.2 on your system assumes a Unix system and that all libraries and compilers are available. For both the Fortran and Python front ends of pyOM2.2 you first need to follow the following steps:

- Obtain the source code as tar archive from

  `https://wiki.cen.uni-hamburg.de/ifm/TO/pyOM2`

- Make a new directory where the model code shall be placed. The directory is called `dir` hereafter.

- Extract the tar-archive containing pyOM2.2 in `dir`.

[1]

2.1 Fortran front end

The following steps are needed to use the Fortran front end of pyOM2.2

- Determine which site-specific Makefile options will be read by the compiler. Type

  echo `site_specific.mk_${HOSTTYPE}`

  to see the file name that will be read and which needs to be provided. The shell variable `HOSTTYPE` is used to determine the specific system and is usually set to the name of the system you are logged on. Note that names might differ using different shells. Examples for supported systems are provided in `dir` and one of them should be copied to the file `dir/site_specific.mk_${HOSTTYPE}`. If necessarry edit the global variables in the file `dir/site_specific.mk_${HOSTTYPE}` to set compiler names and options and to set directories of libraries. See also comments further below.

- 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/site_specific.mk_${HOSTTYPE}` 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`. Compiler flags and further libraries such as Lapack are contained in the variables `F90FLAGS`. When the MPI library is not available, a non-parallel version of the Fortran front end of pyOM2.2 can be build by `make target=without_mpi <model>`. The netcdf library is, however, mandatory at the moment. Currently, the following files for the corresponding systems are provided:

file name	HOSTTYPE	system
site_specific.mk_i686-linux	i686-linux	generic 32 bit Linux (Ubuntu, etc)
site_specific.mk_x86_64-linux	x86_64-linux	generic 64 bit Linux
site_specific.mk_intel-mac	intel_mac	Mac OSX
site_specific.mk_thunder	x86_64-linux	a specific 64 bit Linux server
site_specific.mk_mistral	x86_64-linux	another specific platform
site_specific.mk_rs6000	rs6000	a specific AIX power pc

2.2 Python front end

The following steps are needed to use the Python front end of pyOM2.2

- Determine which site-specific Makefile options will be read by the compiler. Type

  echo `site_specific.mk_${HOSTTYPE}`

  to see the file name that will be read and which needs to be provided. For details see description of Fortran front end installation.

- Change directory to `dir/py_src` and type `make`.
- Run one of the example Python scripts in `dir/py_config`.

To compile the extension module for Python from the Fortran subroutines and modules 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. Two extension modules will be build automatically, one with and one without MPI support, named `dir/pyOM_code_MPI.so` and/or `dir/pyOM_code.so`, respectively. For the former, it is necessary to provide the path to the MPI library and include header files in the configuration file `dir/site_specific.mk_${HOSTTYPE}` in the variables `F2PY_MPIFLAGS`. Further compiler flags and libraries that are needed are specified in `F2PY_FLAGS`. The other variables in `dir/site_specific.mk_${HOSTTYPE}` are not used.

When the extension modules `dir/pyOM_code.so` and/or `dir/pyOM_code_MPI.so` are 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, otherwise `dir/pyOM_code.so` is loaded. 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 basic netcdf-based IO to `pyOM.py` using the Python modules `netcdf4` or `Scientific.IO.netcdf`.
- `pyOM_ave.py` which adds time averages to `pyOM_cdf.py`
- `pyOM_gui.py` which adds a graphical user interface to `pyOM_ave.py` using `Tkinter.py` and plotting during model execution using `matplotlib`.

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 pyOM2.2 is located.

- `./for_src`: Fortran subroutines used by both Fortran and Python front end
  - `./for_src/main`: Main module and subroutines
  - `./for_src/external`: Solver for surface pressure or streamfunction

  - `./for_src/non_hydrostatic`: Routines for non-hydrostatic terms
  - `./for_src/parallel`: Routines for communication between processors
  - `./for_src/density`: Routines for equation of state of sea water
  - `./for_src/isoneutral`: Routines for mixing along neutral surfaces
  - `./for_src/tke`: Small-scale mixing closure by Gaspar et al. (1990)
  - `./for_src/eke`: Meso-scale eddy mixing closure by Eden and Greatbatch (2008)
  - `./for_src/idemix`: Internal wave mixing closure by Olbers and Eden (2013)
  - `./for_src/obc`: Open boundary formulation
  - `./for_src/advection`: Routines for passive and active tracer advection
  - `./for_src/tracer`: Routines for optional passive tracer
  - `./for_src/diagnostics`: Routines for diagnostics (used by Fortran only)
  - `./for_src/etc`: Miscellaneous routines
- `./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 this documentation
- `./bin`: contains executable of Fortran front end after successful compilation
- `./setups`: contains a few specific realistic setups in subdirectories

The directory `./for_src` is again subdivided into several subdirectories containing certain aspects of the code. The main code is located in `./for_src/main`. The top-level program of the Fortran front end is `./for_src/main/main.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/main/main.f90`. The main module is located in `./for_src/main/main_module.f90`, is available to both front ends and contains all important parameters and model variables. A short description of the most important model variables can be found in `main_module.f90`. Other modules containing documentation are `./for_src/tke/tke_module` containing everything concerning the small-scale mixing closure, `./for_src/tke/eke_module`, containing prognostic meso-scale mixing closures, `./for_src/tke/idemix_module`, containing the IDEMIX closure for internal gravity waves, `./for_src/tke/isoneutral_module`, containing the along-isopycnal mixing package, `./for_src/tke/diagnostic_module`, containing everything related to diagnostics. Several other modules will also be build.

3.2 Model configuration

The model configuration is realized by template subroutines (Fortran front end) or template methods (Python front end) for which specific realizations are supplied for each specific experiment. The idea is to keep the whole model configuration in a single source code 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 complete list of them is given here.

- `set_parameter`:

  sets all important fixed model parameters, like domain size, etc. The configuration is specified mainly by logical switches, implementing mixing parameterizations, hydrostatic approximation, etc. which is explained in detail below. This routine/method is called only once before allocating the model variables.

- `set_grid`:

  sets the (vertical and horizontal) resolution by specifying the variables `dxt`, `dyt`, and `dzt`. Units are m for the latter and either also m or degree longitude or latitude for `dxt`, `dyt`, respectively, if the logical switch `coord_degree` is true. The origin of the grid in x and y can be set by `x_origin` and `y_origin` with corresponding units. Based in this input, the u-centered grid as detailed below is set up. This routine/method is only called once during the model setup after allocating the model variables.

- `set_coriolis`:

  sets the Coriolis parameter in the double precision fields `coriolis_t` and `coriolis_h`. Both are two-dimensional (physical dimensions x and y) to allow for rotated grids. This routine/method is only called once during the model setup after `set_grid`.

- `set_topography`

  sets the topography which is given by the two-dimensional (physical dimensions x and y) integer field `kbot`. A value of zero denotes land, $1 \leq$ `kbot` $\leq$ `nz` denotes the vertical index of the deepest wet grid point. This routine/method is only called once during the model setup after `set_coriolis`.

- `set_initial_conditions`

  sets the initial conditions for all model variables. This routine/method is only called once during the model setup after `set_topography`.

- `set_forcing`

  can be used to set time dependent surface boundary conditions or restoring zones. This routine/method is called at the beginning of each time step.

- `set_diagnostics`

  can be used to register variables to be averaged. In Python front-end only available in class `pyOM_ave`. This routine/method is only called once during the model setup after all configuration is done.

- `set_particles`

  can be used to initialize the position of particles integrated simultaneously. This routine/method is only called once during the model setup after all configuration is done.

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. The following methods are available for the Python front end only

- `user_defined_signal`

  is a method which is called by all processors when the leading processor has signaled so. Can be used to exchange data for plotting with the GUI versions in parallel execution.

- `make_plot`

  makes a plot for the GUI version.

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, except for configurations which read data from specific (binary or netcdf) forcing files. 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 in `./py_config`.

3.3.1 Running and restarting the Fortran front end

The model integration will start from the initial conditions as specified in the configuration templates (if no restart files are present, see below) and will integrate over a time period specified by the variable `runlen` which can be found in `main_module`. After that period the integration stops and a (or several) restart file(s) will automatically be written by the Fortran front end, containing the last two time steps of each model variable. The files are named `restart_PE_'my_pe'.dta` where the variable `my_pe` is identical to the one in `main_modules` and denotes the ordinal number of the respective processor.

The restart files 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 overides 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 Running and restarting the Python front end

After initializing the instance of the class `pyOM` (or derivatives of it) the model is started by the method `run`. The length of the run is specified by the parameter `runlen` for the method `run`, the snapshot intervall by the parameter `snapint`, both in seconds. The methods `read_restart` and `write_restart` 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 `pyOM_gui.py`, the snapshots and restarts can be written at any time using the corresponding buttons.

3.4 Sample configurations

This is an incomplete list of sample configurations which can be found either in the directory `./for_config` or in `./py_config`, or in both.

- `kelv_helm1.py/f90`

  A nonhydrostatic configuration of a two-layer system with a large shear between the layers to demonstrate Kelvin-Helmholtz instabilities. The domain is two-dimensional, i.e. in the x-z plane, periodic in x and y, and there are no surface fluxes at the top or bottom, but a zone in the westernmost part of the domain, where temperature and velocity are relaxed towards the initial conditions. The initial conditions are two layers of equal thickness but different buoyancy moving to the east and relative to each other. At the layer interface a small disturbance is introduced such that in the simulation Kelvin-Helmholtz instability will show up.

- `holmboe1.py/f90`

  Similar to `kelv_helm1.py/f90` but without relaxation zone and different initial condition corresponding to an Holmboe instability. The initial small disturbance is taken as the fastest growing mode of a linear stability analysis of the initial conditions.

- `internal_wave1.py/f90`

  A nonhydrostatic configuration 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 temperature variable of the model is in this case a temperature perturbation, the effect of the background stratification on the perturbation temperature is implemented in the configuration routine `set_forcing`.

- `rayleigh1.py/f90`

A nonhydrostatic configuration to demonstrate Rayleigh-Bernard convection. The domain is two-dimensional, i.e. in the x-z plane, and the are top and bottom heat fluxes. The variable `fac` can be increased in order to increase the spatial (and temporal) resolution.

- `jets1.py/f90`

  A hydrostatic configuration to demonstrate eddy-driven zonal jets. A wide channel model with relaxation at the side walls and interior damping as in Eden (2010), simulating strong eddy-driven zonal jets.

- `eady1.py/f90`

  A hydrostatic configuration to demonstrate the classical Eady problem (Eady, 1949). A narrow channel on an f-plane with prescribed stratification and vertically sheared background zonal flow. The temperature variable of the model is in this case a temperature perturbation, the effect of the background stratification on the perturbation temperature is implemented in the configuration routine `set_forcing`.

- `eady2.py/f90`

  Similar to `eady1.py/f90`, but the small initial perturbation is calculated from the fastest growing mode of a linear stability analysis of the initial condition.

- `acc1.py/f90`

  A hydrostatic 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 temperature relaxation driving a large-scale meridional overturning circulation.

- `acc2.py/f90`

  A hydrostatic model using spherical coordinates with a partially closed domain. There is wind forcing over the channel part and buoyancy relaxation driving a large-scale meridional overturning circulation.

3.5 Realistic configurations

Several realistic model configurations can be found in `./setups`. Forcing files are provided at `https://wiki.cen.uni-hamburg.de/ifm/TO/pyOM2`.

- `flame_low/setup1.py/f90`

  North Atlantic configuration with horizontal resolution of $4/3^o \times 4/3^o$ and 45 levels. Based on the FLAME model configuration used in e.g. Eden and Willebrand (2001).

- `global_4deg_45level/setup1.py/f90`

  Quasi-global configuration with $4^o \times 4^o$ and 45 levels.

- `global_1deg/setup1.py/f90`

  Quasi-global configuration with $1^o \times 1^o$ and 100 levels.

4 Parameterisations and boundary conditions

In this section, the most important sub-grid-scale parameterizations and boundary conditions which are used in the model and also other important aspect of the numerical implementation are listed and briefly discussed. All references to files containing code are relative to the directory in which pyOM2.2 is located. All parameters need to be given in SI units.

Parameterizations or boundary conditions are specified by logical variables which are either contained in the main module `./for_src/main/main_module.f90` or in the modules of the respective parameterization. These logical variables are set to a false value by default, except for some which are noted below.

4.1 Streamfunction or 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 with a rigid lid i.e. $w = 0$ at $z = 0$, this relation simplified to one of the surface pressure only. An alternative formulation involved a streamfunction for the depth-averaged flow as detailed below. Setting the logical switch `enable_streamfunction` to a true value enables a streamfunction formalism, otherwise the surface pressure itself is found. The former algorithm is more stable, but involved a special treatment of island integrals as detailed below. In both cases an iterative conjugate gradient method to solve diagnostic relations is used controlled by the stop criterion `congr_epsilon` and the maximal allowed iterations by `congr_max_iterations`.

It is also possible to relax the rigid-lid assumption and to use a linearized free surface formulation by the switch `enable_free_surface`, but then energy conservation is not given anymore.

4.2 Lateral and surface boundary conditions

Lateral boundary conditions are no-flux boundary condition for tracers and free-slip for momentum or periodic boundary conditions (or open boundary conditions as detailed below). Zonal periodic boundary conditions can be chosen by setting the logical switch `enable_cyclic_x` to a true value in the configuration subroutine/method `set_parameter` to apply periodic boundary conditions for all variables in zonal direction. The logical switch `enable_cyclic_y` applies periodic boundary conditions in meridional direction. Surface and eventually bottom fluxes of buoyancy and momentum can be specified in the configuration subroutine/method `set_initial_conditions` and `set_forcing`.

Open boundary conditions using a radiation condition at the northern, southern, eastern or western wall of the model domain can be set in the module `dir/for_src/obc/obc_module.f90`. Additional damping zones can be implemented there as well.

4.3 Conservation of energy

Energy is exchanged consistently between the different reservoirs setting the logical variables `enable_conserve_energy` to a true value, which is also the default setting. For energy conservation, the dissipation rates by all frictional and mixing effects need to be calculated which considerably increases computing time, such that it might become necessary to disable energy conservation for specific configurations. This should be done in the template configuration subroutine/method `set_parameter` by setting `enable_conserve_energy` to a false value.

There are some choices to be made for energy consistency. Setting the logical switch `enable_store_cabbeling_heat` to a true value, the nonlinear heating rates due to vertical, lateral and isopycnal mixing in the presence of the the non-linear equation of state are transfered to heat (where it can be neglected), otherwise it is transferred to the respective suggrid reservoir (TKE or EKE). Numercal advection of temperature and salinity also needs energy, even using the 2.nd order advection scheme. This energy is taken either from the heat reservoir or from TKE by setting `enable_take_P_diss_adv_from_tke` to a true value. Before taking this energy from TKE, the global mean energy need is calculated and subtracted locally to guarantee positive TKE values. Setting `enable_store_bottom_friction_tke` to a true value, energy release by bottom friction is transferred to TKE instead of internal wave energy.

4.4 Hydrostatic approximation and convective adjustment

The hydrostatic approximation is enabled by the logical variable `enable_hydrostatic`. It is true by default, for a non-hydrostatic configuration it needs to be set to a false 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. Using the TKE parameterization by Gaspar et al. (1990) (see below), convective adjustment is done automatically.

4.5 Spherical coordinates

For realistic configurations, the model can use pseudo-spherical coordinates as specified below in Section 6. These coordinates are enabled with the logical variable `coord_degree`, which is set to a false value be default. For a false value of `coord_degree` all metric terms in the momentum equation are set to zero, and the factor $\cos\phi$ or $1/\cos\phi$ showing up in

the differential operators is set to one, otherwise the full equations as specified below in Section 6 are used.

4.6 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, and a second order scheme with superbee flux limiter. The latter is implemented by setting the logical variable `enable_superbee_advection` to a true value in the configuration subroutine/method `set_parameter`. Note that this choice refers to the advection scheme of the tracers but not to the advection of momentum, which is always the classical second-order central differences scheme. Note also that the positive definite superbee scheme introduces a certain amount of numerical diffusion, such that sometimes no additional diffusion is needed in the model simulation. There also also `enable_dst3_advection`, `enable_upwind3_advection` which enable 3.rd order direct space time and 3.rd order upwind schemes for tracer advection, respectively.

Setting the switch `enable_AB_time_stepping` to a true value enables Adam-Bashforth time stepping for the chosen advection scheme (default), otherwise forward time stepping will be used.

4.7 Constant lateral and vertical friction and mixing

Lateral harmonic friction acting on the horizontal velocity (u, v) can be enable by setting `enable_hor_friction` to a true value and is controlled by the horizontal viscosity `a_h`. Lateral harmonic mixing acting on the salinity and temperature is enabled by `enable_hor_diffusion` and controlled by the horizontal diffusivity `k_h`.

Vertical harmonic friction and mixing is always present any by default treated by an fully implicit formulation to allow for large mixing parameter. An explicit formulation for friction can be enabled by setting `enable_explicit_vert_friction` to a true value. The vertical viscosity is set by `kappam_0`, and the vertical diffusivity by `kappah_0`. Default values for all mixing parameters are zero. Note that all viscosities and diffusivities should be specified in the template configuration subroutine/method `set_parameter`.

4.8 Bottom and interior friction

Linear bottom friction is enabled by setting the logical variable `enable_bottom_friction` to a true value in the configuration subroutine/method `set_parameter`. The inverse time scale for bottom friction is given by `r_bot`. Default value is zero. Quadratic bottom friction is enabled by `enable_quadratic_bottom_friction` with parameter `r_quad_bot`. Adding interior Rayleigh damping is implemented by the switch `enable_ray_friction` and the parameter `r_ray`.

4.9 Small-scale turbulent mixing closure

A prognostic TKE model for vertical mixing first introduced by Gaspar et al. (1990) is enabled with the logical variable `enable_tke` in the configuration subroutine/method `set_parameter`, which then calculates the vertical diffusivities and viscosities. The constant variables `kappam_0` and `kappah_0` are not used in this case. Several parameter and additional options can be set, which are documented in `./for_src/tke/tke_module.f90`.

4.10 Internal wave breaking closure

The internal wave mixing closure IDEMIX (Olbers and Eden, 2013) is enabled by the switch `enable_idemix` in the configuration subroutine/method `set_parameter`. Surface and bottom forcing has to be supplied by the fields `forc_iw_surface` and `forc_iw_bottom`, respectively, which may be set in the template subroutine `set_initial_conditions`. See also the example config files. Several additional options can be set, which are documented in `./for_src/idemix/idemix_module.f90`. The gravity wave drag formulation is contained in the file `idemix3.f90`.

4.11 Meso-scale eddy closure

Meso-scale eddy mixing is implemented by along-isopycnal mixing and an additional eddy-driven advection velocity for tracer, which is usually called the parameterization by Gent et al. (1995). The corresponding diffusivities are either prescribed as constant values of calculated using the prognostic EKE closure by Eden and Greatbatch (2008).

4.11.1 Isoneutral mixing

Lateral mixing of tracers along neutral surfaces following the formulation by Griffies (1998) is enabled by setting the logical variable `enable_neutral_diffusion` to a true value in the configuration subroutine/method `set_parameter`. The isoneutral diffusivity which is used is either given by the constant variable `K_iso_0` or by the diffusivity calculated by the prognostic EKE closure by Eden and Greatbatch (2008) (see below) if it is enabled.

In case of too steep slopes s of the neutral surfaces, the mixing scheme by Griffies (1998) becomes unstable. Therefore the isoneutral diffusivity is multiplied by the factor d_{taper} given by

$$d_{taper} = \frac{1}{2}\left(1 + \tanh\left((s_c - |s|)/s_d\right)\right)$$

where the parameter s_c and s_d are set by the variables `iso_slopec` and `iso_dslope`, respectively. In regions with too steep slopes, the isoneutral mixing is replaced by lateral mixing using the diffusivity given by `K_iso_steep`, multiplied with a factor $1 - d_{taper}$.

4.11.2 Eddy-driven advection velocity

The additional eddy-driven advection velocity is either explicitly calculated in form of an anti-symmetric (skew) component of the isoneutral mixing operator following Griffies (1998), which is the default, or by the Temporal Residual Mean (TRM) form where the velocities from the momentum equations already contain the eddy-driven components. The former version is enabled by the logical `enable_skew_diffusion` in the configuration subroutine/method `set_parameter`. This logical is set to false by default, i.e. the TRM form is the default. For the TRM form, an additional vertical friction term is applied in the momentum equation with viscosity $K_{gm} f^2/N^2$, where N is the local stability frequency, f the Coriolis parameter. To limit this viscosity in case of vanishing stratification, a fixed maximal threshold of 0.01 is applied to the factor f^2/N^2.

K_{gm} denotes the skew diffusivity, either used for the antisymmetric components of the isoneutral mixing operator or for the TRM form, and is given either by the constant variable `K_gm_0` or by the prognostically diffusivity of the EKE closure by Eden and Greatbatch (2008) (see below) if it is enabled.

4.11.3 Prognostic EKE closure

The eddy mixing closure by Eden and Greatbatch (2008) is enabled by setting the switch `enable_eke` to a true value in the configuration subroutine/method `set_parameter`. Several parameter for this closure and additional options can be set, which are documented in `./for_src/eke/eke_module.f90`.

5 Diagnostic output

Simple text based output is 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`. For the python frontend, graphical output during the model integration is possible.

5.1 Time step monitor and snapshots of model variables

5.1.1 Fortran frontend

If the logical variable `enable_diag_ts_monitor` is set to a true value, the model time and the number of iterations of the two- and three-dimensional Poisson solver is written to standard output. Also the maximum horizontal and vertical CFL numbers are diagnosed. The interval of the output is controlled by the variable `ts_monint` in seconds.

If the logical variable `enable_diag_snapshots` is set to a true value, snapshots of the model variables are written by the Fortran front end subsequently to a NetCDF file `pyOM.cdf`. Any existing file of this name will be overwritten during model start. The

frequency of the output is controlled by the variable `snapint`. The value of `snapint` gives the number of seconds between subsequent snapshots written to `pyOM.cdf`.

All diagnostics are inactive by default and need to be enabled by logical variables in the configuration subroutine `set_parameter`. The logical variables can be found in `./for_src/diagnostics/diagnostics_module.f90`.

5.1.2 Python frontend

Model time and the number of iterations of the two- and three-dimensional Poisson solver is written by the method `diagnose` to standard output in intervals specified by the parameter `snapint` given to the method `run` used to start the integration. `pyOM_cdf.py` extents `diagnose` by output of model variables to the NetCDF file `pyOM.cdf` with the same interval.

Using `pyOM_gui.py` extents the diagnosis by online plotting specified in the template method `make_plot`. Examples are provided in `./py_config`.

5.2 Time averages

5.2.1 Fortran frontend

Time average diagnostic can be enabled by the respective switch `enable_diag_timeave` in the Fortran front end. Output is written to the NetCDF file `averages_'itt'.cdf` where `itt` is the time step number, with frequency given by `aveint` in seconds in the configuration subroutine `set_parameter`. The variable `avefreq` gives the frequency of individual averaging operations and can be larger than the time step. Note that at the beginning of the simulation, one or several files with names `unfinished_averages_PE_'my_pe'.dta` will be read if they exist, which contain unfinished time averages written at the end of a proceeding simulation with the same configuration. The variable `my_pe` is the ordinal number of each processor. At the end of each simulation, the restart files for the time averaging will be overwritten.

The variables to be averaged are specified in the template routine `set_diagnostics`. Examples are provided in `./for_config`.

5.2.2 Python frontend

The Python class `pyOM_ave.py` implements also time averages. The interval is given by the parameter `snapint` of the method `run`. The variables to be averaged are specified in the template method `set_diagnostics`. Examples are provided in `./py_config`.

5.3 Variances

5.3.1 Fortran frontend

Second order quantities eddy kinetic energy and eddy density fluxes are calculated by setting `enable_diag_variances` to a true value. Output is written to the NetCDF

file `variances_'itt'.cdf` where `itt` is the time step number, with frequency given by `varint` in seconds in the configuration subroutine `set_parameter`. The variable `varfreq` gives the frequency of individual averaging operations and can be larger than the time step. Note that at the beginning of the simulation, one or several files with names `unfinished_variances_PE_'my_pe'.dta` will be read if they exist, which contain unfinished time averages written at the end of a proceeding simulation with the same configuration. The variable `my_pe` is the ordinal number of each processor. At the end of each simulation, the restart files for the variances will be overwritten.

5.3.2 Python frontend

No such feature implemented.

5.4 Energy diagnostics

5.4.1 Fortran frontend

Globally averaged energies and transfer terms can be diagnosed by setting the logical variable `enable_diag_energy` to a true value. Output is written to standard out and in the NetCDF file `energy.cdf`. The variable `energint` specified the number of second between subsequent output. All related variables are averaged between subsequent output intervals. The variable `energfreq` gives the frequency of individual averaging operations and can be larger than the time step.

5.4.2 Python frontend

There is no python implementation for this diagnosis at the moment.

5.5 Meridional overturning

5.5.1 Fortran frontend

Zonally integrated transports above different (potential) density classes are written in intervals `overint` to a file `over.cdf` if the logical variables `enable_diag_overturning` is set to a true value. Any existing file of this name will be overwritten during model start. The variable `overfreq` gives the frequency of individual transport calculations and can be larger than the time step. The number of density levels, reference depth for potential density, and range of density values are set in the subroutine `init_diag_overturning` in the file `./for_src/diagnostics/diag_over.f90` and might need to be adjusted. Transports interpolated to the modified density are also calculated, the results corresponds to the Quasi-Stokes transport streamfunction by McDougall and McIntosh (2001).

5.5.2 Python frontend

There is no python implementation for this diagnosis at the moment.

5.6 Particles

5.6.1 Fortran frontend

Particles or floats can be integrated using the switch `enable_diag_particles` in the Fortran front end. The initial position of the particles is set in the user defined routine `set_particles`. Output is written to the NetCDF file `float.cdf` with frequency `particles_int`. Restart are written to and read from `particles_restart.dta`.

5.6.2 Python frontend

There is no python implementation at the moment.

6 Continuous equations

The relevant equations for a fluid in Boussinesq approximation on an rotating pseudo–spherical coordinate system are given here for reference and can e.g. be found in Olbers et al. (2012). For the moment the hydrostatic and traditional approximation are applied (which will be relaxed below). The primitive equations for momentum are

$$\begin{aligned}
\partial_t u &= \delta u - \frac{1}{a\cos\phi}\partial_\lambda p_s \quad , \quad \delta u = fv - \frac{1}{a\cos\phi}\partial_\lambda p_{hyd} + \frac{uv}{a}\tan\phi + \partial_z A_v \partial_z u - \boldsymbol{\nabla}\cdot\boldsymbol{u}u \\
\partial_t v &= \delta v - \frac{1}{a}\partial_\phi p_s \quad , \quad \delta v = -fu - \frac{1}{a}\partial_\phi p_{hyd} - \frac{u^2}{a}\tan\phi + \partial_z A_v \partial_z v - \boldsymbol{\nabla}\cdot\boldsymbol{u}v \\
0 &= -\partial_z p - g\rho/\rho_0
\end{aligned}$$

with geographical longitude and latitude λ and ϕ and velocity components u and v in those directions, respectively. Note that a factor ρ_0 is absorbed in the pressure $p = p_s + p_{hyd}$, and that further frictional terms might be added to the time tendencies δu and δv. The non-linear advection terms are defined below. Vertically integrating the vertical momentum equation and using a rigid lid yields

$$\int_z^0 \partial_z p = -\int_z^0 dz g\rho/\rho_0 = p|_0 - p \;\rightarrow\; p = p_s + p_{hyd} \;,\; p_{hyd} = \int_z^0 dz g\rho/\rho_0$$

The continuity equation is given by

$$\frac{1}{a\cos\phi}\left(\partial_\lambda u + \partial_\phi(v\cos\phi)\right) + \partial_z w = 0$$

which can be integrated to obtain w. We use a conservation equation for salinity S

$$\partial_t S + \frac{1}{a\cos\phi}\left(\partial_\lambda(uS) + \partial_\phi(vS\cos\phi)\right) + \partial_z(wS) \;=\; D_S$$

where the right hand side terms in D_S are specified later. A corresponding equation for conservative temperature θ and an equation of state $\rho = \rho(S, \theta, z)$ closes the system.

6.1 Surface pressure

The integration constant p_s is only known diagnostically. Vertically integrating the horizontal momentum equation and taking the divergence yields a Poisson equation for the surface pressure p_s:

$$\frac{1}{a\cos\phi}\left(\partial_\lambda \frac{1}{a\cos\phi} h\partial_\lambda p_s + \partial_\phi \cos\phi \frac{1}{a} h\partial_\phi p_s\right) = \frac{1}{a\cos\phi}\left(\partial_\lambda \int_{-h}^{0} dz\delta u + \partial_\phi \cos\phi \int_{-h}^{0} dz\delta v\right)$$

which can be solved at each time step. Alternatively it is possible to derive a corresponding equation for a streamfunction. Vertically averaging the horizontal momentum equation and taking the curl yields

$$\frac{1}{a\cos\phi}\partial_t\left(\partial_\lambda \frac{1}{h}\int_{-h}^{0} dzv - \partial_\phi \frac{1}{h}\int_{-h}^{0} dzu\cos\phi\right) = \frac{1}{a\cos\phi}\left(\partial_\lambda \frac{1}{h}\int_{-h}^{0} dz\delta v - \partial_\phi \frac{1}{h}\int_{-h}^{0} dz\delta u\cos\phi\right)$$

which is a Poisson equation for the temporal change of the streamfunction of the depth averaged flow. An alternative surface boundary formulation to the rigid lid is given by the implicit free surface. The free surface equation is given by

$$\partial_t\eta + \nabla_h \cdot \int_{-h}^{\eta} \boldsymbol{u}_h dz \approx \partial_t\eta + \nabla_h \cdot \int_{-h}^{0} \boldsymbol{u}_h dz = 0$$

where the free surface η is related to the surface pressure by $p_s = g\eta$.

6.2 Pseudo-Cartesian coordinate system

Pseudo-Cartesian coordinates are now defined as $x = \lambda/a$ and $y = \phi/a$. The relevant equations become

$$\begin{aligned}
\partial_t u &= \delta u - \frac{1}{\cos\phi}\partial_x p_s \quad , \quad \delta u = fv - \frac{1}{\cos\phi}\partial_x p_{hyd} + \frac{uv}{a}\tan\phi - \boldsymbol{\nabla}\cdot\boldsymbol{u}u + \partial_z A_v \partial_z u \\
\partial_t v &= \delta v - \partial_y p_s \quad , \quad \delta v = -fu - \partial_y p_{hyd} - \frac{u^2}{a}\tan\phi + \partial_z A_v \partial_z v - \boldsymbol{\nabla}\cdot\boldsymbol{u}v \\
\partial_t S &= -\boldsymbol{\nabla}\cdot\boldsymbol{u}S + D_S \quad , \quad \boldsymbol{\nabla}\cdot\boldsymbol{u}S = \frac{1}{\cos\phi}\left(\partial_x(uS) + \partial_y(vS\cos\phi)\right) + \partial_z(wS) \\
0 &= \frac{1}{\cos\phi}\left(\partial_x u + \partial_y(v\cos\phi)\right) + \partial_z w \\
0 &= -\left(\partial_x \frac{1}{\cos\phi} h\partial_x p_s + \partial_y h\cos\phi\partial_y p_s\right) + \left(\partial_x \int_{-h}^{0} dz\delta u + \partial_y \cos\phi \int_{-h}^{0} dz\delta v\right) \\
0 &= -\partial_t\left(\partial_x \frac{1}{h}\int_{-h}^{0} dzv - \partial_y \frac{1}{h}\int_{-h}^{0} dzu\cos\phi\right) + \partial_x \frac{1}{h}\int_{-h}^{0} dz\delta v - \partial_y \frac{1}{h}\int_{-h}^{0} dz\delta u\cos\phi \\
0 &= p_{hyd} - \int_{z}^{0} dzg\rho/\rho_0
\end{aligned}$$

6.3 Thermodynamics

We use either the 48-term TEOS equation of state or a model equation of state from Vallis (2006)

$$\rho = p_0/c_{s,0}^2 - \rho_0\beta_T(1+\gamma^\star p_0)(\theta-\theta_0) - \rho_0\beta_T^\star(\theta-\theta_0)^2/2 + \rho_0\beta_S(S-S_0)$$

with $p_0 = -g\rho_0 z$, and parameters

$$\rho_0 = 1024\,\mathrm{kg/m^3} \quad , \quad \theta_0 = 9.85\,^\mathrm{o}\mathrm{C} \quad , \quad S_0 = 35 \quad , \quad g = 9.81\,\mathrm{m/s^2} \quad , \quad c_{s,0} = 1490.0\,\mathrm{m/s}$$
$$\beta_T = 1.67\times10^{-4}\,\mathrm{K^{-1}} \quad , \quad \beta_T^\star = 10^{-5}\,\mathrm{K^{-2}} \quad , \quad \beta_S = 0.78\times10^{-3} \quad , \quad \gamma^\star = 1.1\times10^{-8}\,\mathrm{ms^2/kg}$$

or a simple linear equation of state. The derivatives of density with respect to θ and S are given by

$$\begin{aligned}\frac{\partial\rho}{\partial\theta} &= \rho_0\left(-\beta_T + \beta_T\gamma^\star g\rho_0 z - \beta_T^\star(\theta-\theta_0)\right)\\ \frac{\partial\rho}{\partial S} &= \rho_0\beta_S\end{aligned}$$

Dynamic enthalpy is given by

$$\begin{aligned}H_d(S,\theta,z) &= -g/\rho_0\int_z^0\rho(S,\theta,z')dz'\\ &= +gz^2/2\left(-\frac{g}{c_{s,0}^2} + \beta_T g\rho_0\gamma^\star(\theta-\theta_0)\right) + gz\left(-\beta_T(\theta-\theta_0) - \beta_T^\star(\theta-\theta_0)^2/2 + \beta_S(S-S_0)\right)\end{aligned}$$

with

$$\begin{aligned}\frac{\partial H_d}{\partial\theta} &= -\frac{g}{\rho_0}\int_z^0\frac{\partial\rho}{\partial\theta}dz' = gz(-\beta_T - \beta_T^\star(\theta-\theta_0) + \beta_T\gamma^\star g\rho_0\frac{z}{2})\\ \frac{\partial H_d}{\partial\theta} &= -\frac{g}{\rho_0}\int_z^0\frac{\partial\rho}{\partial S}dz' = gz\beta_S\end{aligned}$$

6.4 Non-hydrostatic terms

The hydrostatic and traditional approximation was applied above. Without these the momentum equation becomes

$$\begin{aligned}\partial_t u &= -\frac{1}{\cos\phi}\partial_x(p_s + p_{hyd} + p_{res}) + fv - f_h w - \boldsymbol{u}\cdot\boldsymbol{\nabla}u + \frac{uv}{a}\tan\phi - \frac{uw}{a}\\ \partial_t v &= -\partial_y(p_s + p_{hyd} + p_{res}) - fu - \boldsymbol{u}\cdot\boldsymbol{\nabla}v - \frac{u^2}{a}\tan\phi - \frac{vw}{a}\\ \partial_t w &= -\partial_z p_{res} + f_h u - \boldsymbol{u}\cdot\boldsymbol{\nabla}w + \frac{u^2+v^2}{a}\end{aligned}$$

plus frictional terms. $f_h = 2\omega \cos\phi$ relates to the horizontal component of the Coriolis force. p_{res} is the non-hydrostatic pressure contribution, which is calculated from the (uncorrected) total divergence of the flow. Note that the last terms on the right hand side of the lateral momentum equations are usually neglected in the pseudo-spherical coordinate system. For consistency, the metric term in the vertical momentum equation should also be neglected.

7 Discretization

7.1 Numerical grids

7.1.1 Spatial grid

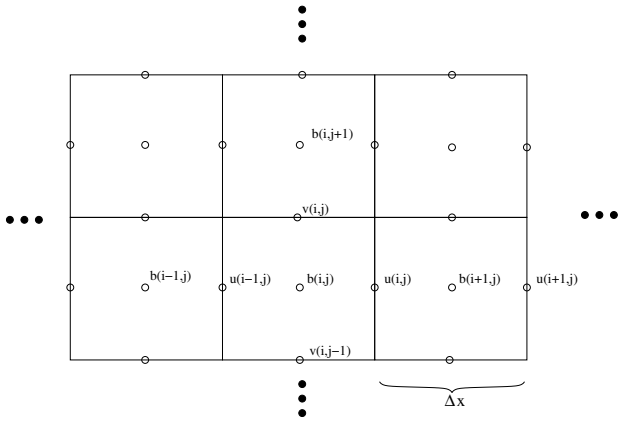


Figure 1: C-grid

All variables are discretized on an Arakawa C-grid. Pressure p and density ρ 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. Advective fluxes of density across the eastern cell border, $F^E_{i,j,k}$ and the corresponding fluxes across the meridional and vertical boundary can be computed as

$$\begin{aligned} F^E_{i,j,k} &= \Delta x \Delta z \, u_{i,j,k}(b_{i,j,k} + b_{i+1,j,k})/2 \\ F^N_{i,j,k} &= \Delta x \Delta z \, v_{i,j,k}(b_{i,j,k} + b_{i,j+1,k})/2 \\ F^T_{i,j,k} &= \Delta x^2 \, w_{i,j,k}(b_{i,j,k} + b_{i,j,k+1})/2 \end{aligned}$$

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

$$(\Delta x^2 \Delta z)\partial_t b_{i,j,k} = (F^E_{i,j,k} - F^E_{i-1,j,k}) + (F^N_{i,j,k} - F^N_{i,j-1,k}) + (F^T_{i,j,k} - F^T_{i,j,k-1})$$

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.

7.1.2 Non-equidistant spatial grid

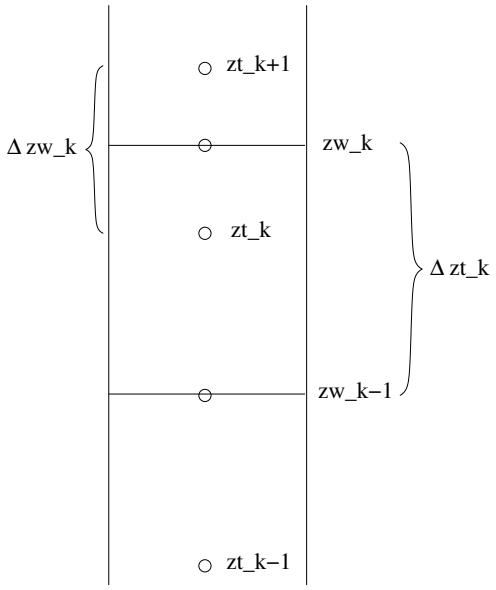


Figure 2: Non-equidistant spatial grid

We count all three spatial indices positive in the respective direction. This also holds for the vertical coordinate. Tracer such as temperature θ is given at points zt_k. Above zt_k ends the grid cell at the level zw_k which is given by $zw_k = (zt_{k+1} + zt_k)/2$. The grid box size for the tracer is thus $\Delta zt_k = zw_k - zw_{k-1}$ and we also define $\Delta zw_k = zt_{k+1} - zt_k$. Figure 2 shows the configuration on the grid. The vertical velocity is defined at zw_k and is thus in the center of its box, while the tracer point zt_k is displaced relative to the center of its box for a non equidistant grid. This is sometimes called a $u-$ centered grid. The grid in i/x and j/y direction follow equivalent rules. The grid sizes Δzt_k, Δxt_i and Δyt_j are specified for each model configuration, from which all levels zt_k and zw_k and zonal positions xt_i, xu_i, etc can be derived. The sea surface is at $zw_N = 0\,\mathrm{m}$, where N is the number of vertical levels.

7.1.3 Time stepping

To integrate $\boldsymbol{u}_h$ an Adam-Bashforth scheme is used but Euler forward for the mixing terms.

$$\boldsymbol{u}_h^{n+1} = \boldsymbol{u}_h^n + \Delta t \left(\delta \boldsymbol{u}_{mix}^n + A_1 \delta \boldsymbol{u}^n - A_2 \delta \boldsymbol{u}^{n-1} - \boldsymbol{\nabla}_h p_s^{n+1} \right)$$

with $A_1 = 1.5 + \epsilon$, $A_2 = 0.5 + \epsilon$ and $\epsilon = 0.1$, and where

$$\delta u^n \;=\; f v^n + \frac{u^n v^n}{a} \tan\phi - \boldsymbol{u}^n \cdot \boldsymbol{\nabla} u^n - \frac{1}{\cos\phi} \partial_x p_{hyd}^n$$

$$
\begin{aligned}
\delta v^n &= -fu^n - \frac{u^n u^n}{a}\tan\phi - \boldsymbol{u}^n \cdot \boldsymbol{\nabla} v^n - \partial_y p^n_{hyd} \\
\delta u^n_{mix} &= \partial_z A_v \partial_z u^{n+1} + \boldsymbol{\nabla}_h \cdot A_h \boldsymbol{\nabla}_h u^n - r u^n \\
\delta v^n_{mix} &= \partial_z A_v \partial_z v^{n+1} + \boldsymbol{\nabla}_h \cdot A_h \boldsymbol{\nabla}_h v^n - r v^n
\end{aligned}
$$

Similar for T and S.

7.1.4 Array dimensions and parallelization

All physical field quantities are discretized only for the domain of the respective processor, which leads to full scalability of the code in terms of memory demands. Array dimensions are of course identical in Fortran and Python front end, but the referencing of the variables is slightly different. First, Fortran array dimensions are detailed: In the Fortran frontend, the total zonal (meridional, vertical) dimension is from 1 to `nx` (`ny`, `nz`). The domain of a processor extends from `is_pe` to `ie_pe` in zonal direction and from `js_pe` to `je_pe` in meridional direction, and over the full depth. If there is only one processor `is_pe` = 1 and `ie_pe` = `nx` and similar for the meridional coordinate. In any case, the actual dimension of each physical variable extents from `is_pe-onx` to `ie_pe+onx`, where `onx` denotes the number of overlapping points between domains in zonal direction and is set currently to 2. The meridional dimension extents from `js_pe-onx` to `je_pe+onx`, with identical parameter `onx` as for the zonal direction. The overlapping points in each direction are used to store the respective information of the neighboring processors during memory exchange, which is done at the end of each time step.

The python module numpy does not allow numerical arrays with other start index as zero. Therefore, the index of a physical dimension needs to be shifted accordingly to access the Fortran arrays in the Python frontend. There are many examples how this is done in the sample configurations.

7.2 External mode

7.2.1 Surface pressure formulation

To solve for the pressure field for a rigid lid

$$
\partial_x \frac{1}{\cos\phi} h \partial_x p_s + \partial_y h \cos\phi \partial_y p_s = \partial_x \int_{-h}^{0} dz \delta u + \partial_y \cos\phi \int_{-h}^{0} dz \delta v
$$

is inverted to yield p_s at each time step. The following algorithm is used

$$
\begin{aligned}
\boldsymbol{u}^*_h &= \boldsymbol{u}^n_h + \Delta t \left(\delta \boldsymbol{u}^n_{mix} + A_1 \delta \boldsymbol{u}^n - A_2 \delta \boldsymbol{u}^{n-1} \right) \\
\boldsymbol{\nabla}_h \cdot h \boldsymbol{\nabla}_h p^{n+1}_s &= \frac{1}{\Delta t} \boldsymbol{\nabla}_h \cdot \int_{-h}^{0} dz \boldsymbol{u}^* \\
\boldsymbol{u}^{n+1}_h &= \boldsymbol{u}^n_h - \Delta t \boldsymbol{\nabla}_h p^{n+1}_s
\end{aligned}
$$

where $\delta \boldsymbol{u}_{mix}$ denote velocity changes related to friction which are excluded from the Adam-Bashforth time stepping. This scheme is identical to MITgcm. The Poisson equation is solved with a simple conjugate gradient solver. Problematic are eigenvalues near zero of the matrix to be inverted in certain configurations, such the solver diverges. This becomes even worse using a preconditioner.

7.2.2 Free surface formulation

To solve for the surface pressure for a free surface

$$\partial_t p_s + g\nabla_h \cdot \int_{-h}^{0} \boldsymbol{u}_h dz = 0 \ , \ p_s^{n+1} = p_s^n - g\Delta t \nabla_h \cdot \int_{-h}^{0} \boldsymbol{u}_h^{n+1} dz$$

the following algorithm is used

$$\begin{aligned}
\boldsymbol{u}_h^* &= \boldsymbol{u}_h^n + \Delta t \left(\delta \boldsymbol{u}_{mix}^n + A_1 \delta \boldsymbol{u}^n - A_2 \delta \boldsymbol{u}^{n-1} \right) \\
p_s^* &= \epsilon p_s^n - g\Delta t \nabla_h \cdot \int_{-h}^{0} \boldsymbol{u}_h^* dz \\
\boldsymbol{\nabla}_h \cdot h \boldsymbol{\nabla}_h p_s^{n+1} - p_s^{n+1} \frac{\epsilon}{g\Delta t^2} &= -p_s^* \frac{1}{g\Delta t^2} = \frac{1}{\Delta t} \boldsymbol{\nabla}_h \cdot \int_{-h}^{0} dz \boldsymbol{u}^* - p_s^n \frac{\epsilon}{g\Delta t^2}
\end{aligned}$$

For $\epsilon = 0$ this becomes the rigid lid version. However, $\epsilon = 1$ converges often much faster. Problematic are the conservation properties in the uppermost grid boxes.

7.2.3 Streamfunction formulation

The transport streamfunction ψ is given by

$$\frac{1}{\cos\phi} \left(\partial_x \frac{1}{h} \frac{1}{\cos\phi} \partial_x \partial_t \psi + \partial_y \frac{1}{h} \cos\phi \partial_y \partial_t \psi \right) = \frac{1}{\cos\phi} \left(\partial_x \frac{1}{h} \int_{-h}^{0} dz \delta v - \partial_y \frac{1}{h} \int_{-h}^{0} dz \delta u \cos\phi \right)$$

with

$$\frac{1}{\cos\phi} \partial_x \psi = \int_{-h}^{0} dz v \quad , \quad -\partial_y \psi = \int_{-h}^{0} dz u$$

The following algorithm is used

$$\begin{aligned}
\boldsymbol{\nabla}_h \cdot \frac{1}{h} \boldsymbol{\nabla}_h \delta\psi^n &= \underset{\neg}{\boldsymbol{\nabla}}_h \cdot \frac{1}{h} \int_{-h}^{0} dz \left(\delta \boldsymbol{u}_{mix}^n + A_1 \delta \boldsymbol{u}^n - A_2 \delta \boldsymbol{u}^{n-1} \right) \\
\psi^{n+1} &= \psi^n + \Delta t \delta \psi^n
\end{aligned}$$

or with AB. ψ is defined on the vorticity grid with

$$\frac{(\psi_{i,j} - \psi_{i-1,j})}{\cos\phi u_j \Delta x t_i} = \int_{-h}^{0} dz v_{ij} \quad , \quad -\frac{(\psi_{i,j} - \psi_{i,j-1})}{\Delta y t_j} = \int_{-h}^{0} dz u_{ij}$$

and

$$\begin{aligned}
\underset{\neg}{\boldsymbol{\nabla}}_h \cdot \frac{1}{h}\int_{-h}^{0} dz\delta\boldsymbol{u} &= \frac{1/h^v_{i+1,j}\int dz\delta v_{i+1,j} - 1/h^v_{ij}\int dz\delta v_{i,j}}{\cos\phi u_j \Delta x u_i} \\
&\quad - \frac{\cos\phi t_{j+1}/h^u_{i,j+1}\int dz\delta u_{i,j+1} - \cos\phi t_j 1/h^u_{ij}\int dz\delta u_{i,j}}{\cos\phi u_j \Delta y u_j} \\
\boldsymbol{\nabla}_h \cdot \frac{1}{h}\boldsymbol{\nabla}_h\delta\psi^n &= \frac{\frac{(\psi_{i+1,j}-\psi_{i,j})}{h^v_{i+1,j}\cos\phi u_j \Delta x t_{i+1}} - \frac{(\psi_{i,j}-\psi_{i-1,j})}{h^v_{ij}\cos\phi u_j \Delta x t_i}}{\cos\phi u_j \Delta x u_i} \\
&\quad + \frac{\cos\phi t_{j+1}\frac{(\psi_{i,j+1}-\psi_{i,j})}{h^u_{i,j+1}\Delta y t_{j+1}} - \cos\phi t_j \frac{(\psi_{i,j}-\psi_{i,j-1})}{h^u_{ij}\Delta y t_j}}{\cos\phi u_j \Delta y u_j} \\
&= \frac{(\psi_{i+1,j}-\psi_{i,j})}{h^v_{i+1,j}(\cos\phi u_j)^2 \Delta x t_{i+1}\Delta x u_i} - \frac{(\psi_{i,j}-\psi_{i-1,j})}{h^v_{ij}(\cos\phi u_j)^2 \Delta x t_i \Delta x u_i} \\
&\quad + \frac{\cos\phi t_{j+1}}{\cos\phi u_j}\frac{(\psi_{i,j+1}-\psi_{i,j})}{h^u_{i,j+1}\Delta y t_{j+1}\Delta y u_j} - \frac{\cos\phi t_j}{\cos\phi u_j}\frac{(\psi_{i,j}-\psi_{i,j-1})}{h^u_{ij}\Delta y t_j \Delta y u_j}
\end{aligned}$$

The surface pressure contribution to the forcing is

$$\begin{aligned}
\underset{\neg}{\boldsymbol{\nabla}}_h \cdot \boldsymbol{\nabla}_h p_s &= \frac{(p_{i+1,j+1}-p_{i+1,j})/\Delta y u_j - (p_{i,j+1}-p_{ij})/\Delta y u_j}{\cos\phi u_j \Delta x u_i} \\
&\quad - \frac{(p_{i+1,j+1}-p_{i,j+1})/(\Delta x u_i) - (p_{i+1,j}-p_{i,j})/(\Delta x u_i)}{\cos\phi u_j \Delta y u_j} = 0
\end{aligned}$$

The Poisson equation is solved using a conjugate gradient solver with simple preconditioner taken from MOM.

7.2.4 Island integrals for streamfunction

Island integrals can be treated as follows: Consider the vertically integrated momentum equation

$$1/h\partial_t \int_{-h}^{0} \boldsymbol{u}_h dz = 1/h\int_{-h}^{0} dz\delta\boldsymbol{u} - \boldsymbol{\nabla}_h p_s = 1/h\partial_t \underset{\neg}{\boldsymbol{\nabla}}\psi$$

For a closed line integral e.g. around an island, the surface pressure contribution drops

$$\int_C 1/h\, d\boldsymbol{\ell}\cdot\underset{\neg}{\boldsymbol{\nabla}}\partial_t\psi = \int_C 1/h\, d\boldsymbol{\ell}\cdot\int_{-h}^{0} dz\delta\boldsymbol{u}$$

This relation holds for each island with number n and perimeter C_n. Now we consider the superposition

$$\psi = \psi_0(t,\boldsymbol{x}_h) + \sum_n \mu_n(t)\psi_n(\boldsymbol{x}_h)$$

with $\psi_0 = 0$ along all boundaries, and $\psi_n = const$ along C_n but $\psi_n = 0$ along $C_{m\neq n}$ and with

$$\boldsymbol{\nabla}_h \cdot \frac{1}{h}\boldsymbol{\nabla}_h \psi_n = 0 \quad , \quad \boldsymbol{\nabla}_h \cdot \frac{1}{h}\boldsymbol{\nabla}_h \partial_t \psi_0 = \underset{\neg}{\boldsymbol{\nabla}}_h \cdot \frac{1}{h}\int_{-h}^{0} dz \delta \boldsymbol{u}$$

The first relation is time independent and is solved only once, the second is easily solved with simple Dirichlet boundary conditions. Using the superposition for ψ in the island integrals yields

$$\sum_i (\partial_t \mu_i) \int_{C_n} 1/h \, d\boldsymbol{\ell} \cdot \underset{\neg}{\boldsymbol{\nabla}} \psi_i \;\; = \;\; \int_{C_n} 1/h \, d\boldsymbol{\ell} \cdot \int_{-h}^{0} dz \delta \boldsymbol{u} - \int_{C_n} 1/h d\boldsymbol{\ell} \cdot \partial_t \underset{\neg}{\boldsymbol{\nabla}} \psi_0$$

This can be solved for each island integral along C_n and yields an algebraic system for all $\partial_t \mu_i$.

7.3 Discrete kinetic energy

7.3.1 Coriolis term

A way to obtain consistent Coriolis terms is the following from MITgcm.

$$\begin{aligned}
\partial_t u_{ij} + ... &= 0.25(f_{ij}\frac{\Delta x t_i}{\Delta x u_i}(v_{i,j} + v_{i,j-1}) + f_{i+1,j}\frac{\Delta x t_{i+1}}{\Delta x u_i}(v_{i+1,j} + v_{i+1,j-1})) \\
\partial_t v_{ij} + ... &= -0.25(f_{ij}\frac{\Delta y t_j \cos\phi t_j}{\cos\phi u_j \Delta y u_j}(u_{i,j} + u_{i-1,j}) + f_{i,j+1}\frac{\Delta y t_{j+1}\cos\phi t_{j+1}}{\cos\phi u_j \Delta y u_j}(u_{i,j+1} + u_{i-1,j+1}))
\end{aligned}$$

or

$$\begin{aligned}
\partial_t u_{ij} + ... &= 0.25(f_{ij}A^t_{ij}(v_{ij} + v_{i,j-1}) + f_{i+1,j}A^t_{i+1,j}(v_{i+1,j} + v_{i+1,j-1}))/A^u_{ij} \\
\partial_t v_{ij} + ... &= -0.25(f_{ij}A^t_{ij}(u_{ij} + u_{i-1,j}) + f_{i,j+1}A^t_{i,j+1}(u_{i,j+1} + u_{i-1,j+1}))/A^v_{ij}
\end{aligned}$$

with

$$A^t_{ij} = \Delta y t_j \Delta x t_i \cos\phi t_j \;\; , \;\; A^u_{ij} = \Delta y t_j \Delta x u_i \cos\phi t_j \;\; , \;\; A^v_{ij} = \Delta y u_j \Delta x t_i \cos\phi u_j$$

Integrating u^2 horizontally on the U grid and v^2 on the V grid we obtain

$$\begin{aligned}
&\sum_{ij} u_{ij}(f_{ij}A^t_{ij}(v_{ij} + v_{i,j-1}) + f_{i+1,j}A^t_{i+1,j}(v_{i+1,j} + v_{i+1,j-1})) \\
-&\sum_{ij} v_{ij}(f_{ij}A^t_{ij}(u_{ij} + u_{i-1,j}) + f_{i,j+1}A^t_{i,j+1}(u_{i,j+1} + u_{i-1,j+1})) \;\; = \;\; 0
\end{aligned}$$

by shifting the sums. Thus, the Coriolis term discretized in this way does no work.

The non-hydrostatic additional terms are

$$\begin{aligned}
\partial_t u_{ijk} + ... &= ... - 0.25(f^h_{ij}A^t_{ij}(w_{ijk} + w_{ij,k-1}) + f^h_{i+1,j}A^t_{i+1,j}(w_{i+1,jk} + w_{i+1,j,k-1}))/A^u_{ij} \\
\partial_t w_{ijk} + ... &= ... + 0.25(f^h_{ij}\Delta z t_k(u_{i,j,k} + u_{i-1,j,k}) + f^h_{ij}\Delta z t_{k+1}(u_{i,j,k+1} + u_{i-1,j,k+1}))/\Delta z w_k
\end{aligned}$$

with the horizontal Coriolis parameter f_h. Integrating u^2 globally on the u grid and w^2 on the w grid we obtain

$$-\sum_{ijk}(f^h_{ij}A^t_{ij}\Delta zt_k u_{ijk}(w_{ijk}+w_{ij,k-1})+f^h_{i+1,j}A^t_{i+1,j}\Delta zt_k u_{ijk}(w_{i+1,jk}+w_{i+1,j,k-1}))$$
$$+\sum_{ijk}(f^h_{ij}A^t_{ij}\Delta zt_k w_{ijk}(u_{i,j,k}+u_{i-1,j,k})+f^h_{ij}A^t_{ij}\Delta zt_{k+1}w_{ijk}(u_{i,j,k+1}+u_{i-1,j,k+1}))=0$$

by shifting the sums.

7.3.2 Metric terms

The metric terms $uva^{-1}\tan\phi$ and $-u^2a^{-1}\tan\phi$ are discretized like in the MITgcm as

$$\begin{aligned}\partial_t u_{ij}+... &= 0.125a^{-1}\tan\phi t_j((u_{ij}+u_{i-1,j})A^t_{ij}(v_{ij}+v_{i,j-1})+(u_{i+1,j}+u_{i,j})A^t_{i+1,j}(v_{i+1,j}+v_{i+1,j-1}))/A^u_{ij}\\ &= 0.125a^{-1}\tan\phi t_j(\frac{\Delta xt_i}{\Delta xu_i}(u_{ij}+u_{i-1,j})(v_{ij}+v_{i,j-1})+\frac{\Delta xt_{i+1}}{\Delta xu_i}(u_{i+1,j}+u_{i,j})(v_{i+1,j}+v_{i+1,j-1}))\end{aligned}$$

$$\begin{aligned}\partial_t v_{ij}+... &= -0.125a^{-1}(A^t_{ij}\tan\phi t_j(u_{ij}+u_{i-1,j})^2+A^t_{i,j+1}\tan\phi t_{j+1}(u_{i,j+1}+u_{i-1,j+1})^2)/A^v_{ij}\\ &= -0.125a^{-1}(\frac{\Delta yt_j\cos\phi t_j}{\cos\phi u_j\Delta yu_j}\tan\phi t_j(u_{ij}+u_{i-1,j})^2\\ &\quad+\frac{\Delta yt_{j+1}\cos\phi t_{j+1}}{\cos\phi u_j\Delta yu_j}\tan\phi t_{j+1}(u_{i,j+1}+u_{i-1,j+1})^2)\end{aligned}$$

The non-hydrostatic metric terms in the vertical momentum equation are

$$\begin{aligned}\partial_t w_{ijk}+... &= 0.125a^{-1}\left(\Delta zt_k\left(u_{i,j,k}+u_{i-1,j,k}\right)^2+\Delta zt_{k+1}\left(u_{i,j,k+1}+u_{i-1,j,k+1}\right)^2\right)/\Delta zw_k\\ &\quad+0.125a^{-1}\left(\Delta zt_k\left(v_{i,j,k}+v_{i,j-1,k}\right)^2+\Delta zt_{k+1}\left(v_{i,j,k+1}+v_{i,j-1,k+1}\right)^2\right)/\Delta zw_k\end{aligned}$$

They should be balanced by corresponding terms in the lateral momentum equations, which are, however, usually neglected in the pseudo-spherical approximation. Thus, we also neglect all metric terms in the vertical momentum quation.

7.3.3 Momentum advection

A kinetic energy conserving momentum advection scheme is given by

$$\begin{aligned}\boldsymbol{u}\cdot\boldsymbol{\nabla}u &= (\delta_i\overline{U}^i\overline{u}^i+\delta_j\overline{V}^i\overline{u}^j)/A^u_{ij}+\delta_k\overline{W}^i\overline{u}^k/(\Delta zt_kA^u_{ij})\\ \boldsymbol{u}\cdot\boldsymbol{\nabla}v &= (\delta_i\overline{U}^j\overline{v}^i+\delta_j\overline{V}^j\overline{v}^j)/A^v_{ij}+\delta_k\overline{W}^j\overline{v}^k/(\Delta zt_kA^v_{ij})\end{aligned}$$

with $U_{ijk}=\Delta yt_ju_{ijk}$ and $V_{ijk}=\Delta xt_i\cos\phi t_jv_{ijk}$ and $W=wA^t_{ij}$. Or write as

$$\begin{aligned}\boldsymbol{u}\cdot\boldsymbol{\nabla}u &= (F^u_{ij}-F^u_{i-1,j})/A^u_{ij}+(F^v_{ij}-F^v_{i,j-1})/A^u_{ij}+\delta_k\overline{W}^i\overline{u}^k/(\Delta zt_kA^u_{ij})\\ F^u_{ij} &= 0.25(\Delta yt_ju_{i+1,j}+\Delta yt_ju_{i,j})(u_{i+1,j}+u_{ij})\\ F^v_{ij} &= 0.25x(\Delta xt_i\cos\phi t_jv_{i+1,j}+\Delta xt_i\cos\phi t_jv_{i,j})(u_{i,j+1}+u_{ij})\end{aligned}$$

7.3.4 Vertical dissipation

For the dissipation due to implicit vertical friction $u\partial_z A_v \partial_z u = \partial_z A_v \partial_z u^2/2 - A_v(\partial_z u)^2$ we obtain

$$\begin{aligned} u_k^n(A_k(u_{k+1}^{n+1} - u_k^{n+1})/\Delta zw_k - A_{k-1}(u_k^{n+1} - u_{k-1}^{n+1})/\Delta zw_{k-1})/\Delta zt_k = \\ 0.5(A_k(u_{k+1}^{n+1}u_{k+1}^n - u_k^{n+1}u_k^n)/\Delta zw_k - A_{k-1}(u_k^{n+1}u_k^n - u_{k-1}^{n+1}u_{k-1}^n)/\Delta zw_{k-1})/\Delta zt_k \\ -0.5(A_k u_k^{n+1}u_k^n/\Delta zw_k + A_{k-1}u_k^{n+1}u_k^n/\Delta zw_{k-1})/\Delta zt_k \\ +(A_k u_k^n u_{k+1}^{n+1}/\Delta zw_k + A_{k-1}u_k^n u_{k-1}^{n+1}/\Delta zw_{k-1})/\Delta zt_k \\ -0.5(A_k u_{k+1}^{n+1}u_{k+1}^n/\Delta zw_k + A_{k-1}u_{k-1}^{n+1}u_{k-1}^n/\Delta zw_{k-1})/\Delta zt_k \end{aligned}$$

so a flux of KE

$$\partial_z A \partial_z u^2/2 = 0.5\overline{(A_k(u_{k+1}^2 - u_k^2)/\Delta zw_k - A_{k-1}(u_k^2 - u_{k-1}^2)/\Delta zw_{k-1})/\Delta zt_k}^i$$

and a dissipation term

$$\begin{aligned} -(A_k u_k^{n+1}u_k^n/\Delta zw_k + A_{k-1}u_k^{n+1}u_k^n/\Delta zw_{k-1}) + (A_k u_k^n u_{k+1}^{n+1}/\Delta zw_k + A_{k-1}u_k^n u_{k-1}^{n+1}/\Delta zw_{k-1}) \\ = A_k u_k^n(u_{k+1}^{n+1} - u_k^{n+1})/\Delta zw_k - u_k^n A_{k-1}(u_k^{n+1} - u_{k-1}^{n+1})/\Delta zw_{k-1} = u_k^n(F_k - F_{k-1}) \end{aligned}$$

plus

$$\begin{aligned} +(A_k u_k u_{k+1}/\Delta zw_k + A_{k-1}u_k u_{k-1}/\Delta zw_{k-1}) - (A_k u_{k+1}^2/\Delta zw_k + A_{k-1}u_{k-1}^2/\Delta zw_{k-1}) \\ = -u_{k+1}A_k(u_{k+1} - u_k)/\Delta zw_k + u_{k-1}A_{k-1}(u_k - u_{k-1})/\Delta zw_{k-1} = -u_{k+1}F_k + u_{k-1}F_{k-1} \end{aligned}$$

such that

$$-A(\partial_z u)^2 = -0.5\overline{((u_{k+1} - u_k)F_k + (u_k - u_{k-1})F_{k-1})}^i/\Delta zt_k < 0$$

and similar for v. At the surface and the bottom the fluxes out/in the domain should be taken out of the dissipation. At $k = 1$

$$\begin{aligned} u_k(A_k(u_{k+1} - u_k)/\Delta zw_k - F_b)/\Delta zt_k = \\ 0.5(A_k(u_{k+1}^2 - u_k^2)/\Delta zw_k)/\Delta zt_k - u_k F_b/\Delta zt_k - 0.5(u_{k+1} - u_k)F_k/\Delta zt_k \end{aligned}$$

and at $k = N$

$$\begin{aligned} u_k(F_t - A_{k-1}(u_k - u_{k-1})/\Delta zw_{k-1})/\Delta zt_k = \\ u_k F_t/\Delta zt_k - 0.5A_{k-1}(u_k^2 - u_{k-1}^2)/\Delta zw_{k-1}/\Delta zt_k - 0.5(u_k - u_{k-1})F_{k-1}/\Delta zt_k \end{aligned}$$

Integrate over z

$$\begin{aligned} 0.5(u_2 - u_1)F_1 + \sum_{k=2}^{N-1} 0.5((u_{k+1} - u_k)F_k + (u_k - u_{k-1})F_{k-1}) + 0.5(u_N - u_{N-1})F_{N-1} = \\ \sum_{k=1}^{N-1} 0.5F_k u_{k+1} - \sum_{k=1}^{N-1} 0.5u_k F_k + \sum_{k=2}^{N} 0.5F_{k-1}u_k - \sum_{k=2}^{N} 0.5u_{k-1}F_{k-1} = \sum_{k=2}^{N} F_{k-1}(u_k - u_{k-1}) \\ = \sum_{k=1}^{N-1} F_k(u_{k+1} - u_k) \end{aligned}$$

with $F_0 = F_N = 0$. In the TKE equation on the W grid it is thus adequate to add

$$\overline{F_k(u^n_{k+1} - u^n_k)/\Delta zw_k}^i$$

as forcing by vertical dissipation.

7.3.5 Dissipation by Rayleigh friction

Rayleigh friction $-r\boldsymbol{u}$ in the momentum equation yields $-ru^2 - rv^2$ as energy sink. The discrete form interpolated horizontally on T grid is

$$S_{ijk} = -r0.5(u^2_{i,j,k} + u^2_{i-1,j,k}) - r0.5(v^2_{i,j,k} + v^2_{i,j-1,k})$$

Integrate vertically

$$\sum_{k=1}^{N} S_{ijk}\Delta zt_k = -0.5r \sum_{k=1}^{N} \Delta zt_k(u^2_{i,j,k} + u^2_{i-1,j,k} + v^2_{i,j,k} + v^2_{i,j-1,k})$$

Integrate also on W grid

$$0.5\Delta zw_0 S_{ij1} + \sum_{k=1}^{N-1} 0.5(S_{ijk} + S_{i,j,k+1})\Delta zw_k + 0.5\Delta zw_N S_{ijN}$$

$$= 0.5S_{ij,1}(\Delta zw_0 + \Delta zw_1) + 0.5\sum_{k=2}^{N-1} S_{ijk}(\Delta zw_k + \Delta zw_{k-1}) + 0.5S_{i,j,N}(\Delta zw_{N-1} + \Delta zw_N) = \sum_{k=1}^{N} S_{ijk}\Delta zt_k$$

with $zw_k = (zt_{k+1} + zt_k)/2$ and $\Delta zw_k = zt_{k+1} - zt_k$ and $\Delta zt_k = zw_k - zw_{k-1} = (zt_{k+1} - zt_{k-1})/2$ and $0.5(\Delta zw_k + \Delta zw_{k-1}) = 0.5(zt_{k+1} - zt_{k-1}) = \Delta zt_k$. It is thus adequate to add

$$-0.5\frac{\Delta zw_0}{\Delta zw_1}S_{ij1} - 0.5(S_{ij1} + S_{i,j,2}) \text{ for } k = 1\ ,\ -0.5(S_{ijk} + S_{i,j,k+1}) \text{ for } k = 2, N-1\ ,\ -S_{ijN} \text{ for } k = N$$

in the TKE equation as forcing, since we do not want to integrate $k = 0$ for TKE.

7.3.6 Lateral dissipation

For the lateral friction terms $u\boldsymbol{\nabla}A \cdot \boldsymbol{\nabla}u = \boldsymbol{\nabla} \cdot A\boldsymbol{\nabla}u^2/2 - A(\boldsymbol{\nabla}u)^2$ we consider

$$\boldsymbol{\nabla}A \cdot \boldsymbol{\nabla}u = (F^x_{ij} - F^x_{i-1,j})/(\cos\phi t_j \Delta x u_i) + (F^y_{i,j} - F^y_{i,j-1})/(\cos\phi t_j \Delta y t_j)$$

with

$$F^x_{ij} = A_{ij}(u_{i+1,j} - u_{i,j})/(\cos\phi t_j \Delta x t_{i+1})\ ,\ F^y_{i,j} = A_{ij}(u_{i,j+1} - u_{i,j})/\Delta y u_j \cos\phi u_j$$

and thus

$$u_{ij}(F^x_{ij} - F^x_{i-1,j}) =$$
$$A_i u_{ij}(u_{i+1,j} - u_{i,j})/(\cos\phi t_j \Delta x t_{i+1}) - A_{i-1}u_{ij}(u_{i,j} - u_{i-1,j})/(\cos\phi t_j \Delta x t_i) =$$
$$0.5(A_i(u^2_{i+1,j} - u^2_{i,j})/(\cos\phi t_j \Delta x t_{i+1}) - A_{i-1}(u^2_{i,j} - u^2_{i-1,j})/(\cos\phi t_j \Delta x t_i))$$
$$-0.5A_i u^2_{i+1,j}/(\cos\phi t_j \Delta x t_{i+1}) + A_i u_{ij}u_{i+1,j}/(\cos\phi t_j \Delta x t_{i+1}) - 0.5A_i u^2_{i,j}/(\cos\phi t_j \Delta x t_{i+1})$$
$$-0.5A_{i-1}u^2_{i,j}(\cos\phi t_j \Delta x t_i) - 0.5A_{i-1}u^2_{i-1,j}/(\cos\phi t_j \Delta x t_i) + A_{i-1}u_{ij}u_{i-1,j}/(\cos\phi t_j \Delta x t_i)$$
$$u_{ij}(F^x_{ij} - F^x_{i-1,j})/(\cos\phi t_j \Delta x u_i) = \partial_x A \partial_x u^2/2$$
$$-0.5(u_{i+1,j} - u_{ij})F^x_{ij}/(\cos\phi t_j \Delta x u_i) - 0.5(u_{ij} - u_{i-1,j})F^x_{i-1,j}/(\cos\phi t_j \Delta x u_i)$$

and for F^y_{ij}

$$u_{ij}(F^y_{i,j} - F^y_{i,j-1}) = u_{ij}(A_{ij}(u_{i,j+1} - u_{i,j})/\Delta y u_j \cos\phi u_j - A_{i,j-1}(u_{i,j} - u_{i,j-1})/\Delta y u_{j-1}\cos\phi u_{j-1}$$
$$u_{ij}(F^y_{ij} - F^y_{i,j-1})/(\cos\phi t_j \Delta y t_j) = \partial_y A \partial_y u^2/2$$
$$-0.5(u_{i,j+1} - u_{ij})F^y_{ij}/(\cos\phi t_j \Delta y t_j) - 0.5(u_{ij} - u_{i,j-1})F^y_{i,j-1}/(\cos\phi t_j \Delta y t_j)$$

so that

$$\begin{aligned} A(\boldsymbol{\nabla} u)^2 &= 0.5\overline{(u_{i+1,j} - u_{i,j})F^x_{ij} + (u_{i,j} - u_{i-1,j})F^x_{i-1,j}}^i/(\cos\phi t_j \Delta x u_i) \\ &\quad +0.5\overline{(u_{i,j+1} - u_{i,j})F^y_{ij} + (u_{i,j} - u_{i,j-1})F^y_{i,j-1}}^i/(\cos\phi t_j \Delta y t_j) > 0 \end{aligned}$$

and for v with

$$\boldsymbol{\nabla} A \cdot \boldsymbol{\nabla} v = (F^x_{ij} - F^x_{i-1,j})/(\cos\phi u_j \Delta x t_i) + (F^y_{i,j} - F^x_{i,j-1})/(\cos\phi u_j \Delta y u_j)$$
$$F^x_{ij} = A_i(u_{i+1,j} - u_{i,j})/(\cos\phi u_j \Delta x u_i)\ ,\ F^y_{i,j} = A_i(u_{i,j+1} - u_{i,j})/\Delta y t_{j+1}\cos\phi t_{j+1}$$

yields

$$\begin{aligned} A(\boldsymbol{\nabla} v)^2 &= 0.5\overline{(v_{i+1,j} - v_{i,j})F^x_{ij} + (v_{i,j} - v_{i-1,j})F^x_{i-1,j}}^i/(\cos\phi u_j \Delta x t_i) \\ &\quad +0.5\overline{(v_{i,j+1} - v_{i,j})F^y_{ij} + (v_{i,j} - v_{i,j-1})F^y_{i,j-1}}^i/(\cos\phi u_j \Delta y u_j) > 0 \end{aligned}$$

For free slip fluxes are simply zero at the boundaries, for no-slip things are more complicated. Vertical interpolation on W grid as for Rayleigh friction.

7.4 Biharmonic friction

Biharmonic friction is given by

$$\partial_t u = ... - \boldsymbol{\nabla} \cdot A^{1/2}\boldsymbol{\nabla}\boldsymbol{\nabla} \cdot A^{1/2}\boldsymbol{\nabla} u$$

and similar for v. Its effect on energy is given by

$$u\partial_t u = ... - \boldsymbol{\nabla} \cdot \left(uA^{1/2}\boldsymbol{\nabla}\boldsymbol{\nabla} \cdot A^{1/2}\boldsymbol{\nabla} u\right) + A^{1/2}\boldsymbol{\nabla}\left(\boldsymbol{\nabla} \cdot A^{1/2}\boldsymbol{\nabla} u\right) \cdot \boldsymbol{\nabla} u$$

Now with $A^{1/2}(\boldsymbol{\nabla}\left(\boldsymbol{\nabla} \cdot A^{1/2}\boldsymbol{\nabla} u\right)) \cdot \boldsymbol{\nabla} u = \boldsymbol{\nabla} \cdot \left(\boldsymbol{\nabla} \cdot A^{1/2}\boldsymbol{\nabla} u\right) A^{1/2}\boldsymbol{\nabla} u - \left(\boldsymbol{\nabla} \cdot A^{1/2}\boldsymbol{\nabla} u\right)^2$

$$u\partial_t u = ... - \boldsymbol{\nabla} \cdot \left[uA^{1/2}\boldsymbol{\nabla}\left(\boldsymbol{\nabla} \cdot A^{1/2}\boldsymbol{\nabla} u\right) - \left(\boldsymbol{\nabla} \cdot A^{1/2}\boldsymbol{\nabla} u\right) A^{1/2}\boldsymbol{\nabla} u\right] - \left(\boldsymbol{\nabla} \cdot A^{1/2}\boldsymbol{\nabla} u\right)^2$$

The last term is sign-definite and the dissipation rate by biharmonic friction.

7.4.1 Buoyancy work

Pressure work is

$$B = -\overline{u_{ij}(p_{i+1,j} - p_{i,j})/(\Delta x u_i \cos\phi t_j)}^{\,i} - \overline{v_{ij}(p_{i,j+1} - p_{ij})/\Delta y u_j}^{\,j}$$

and for

$$-\boldsymbol{u}\cdot\boldsymbol{\nabla}p = -\boldsymbol{\nabla}\cdot(\boldsymbol{u}p) + p\boldsymbol{\nabla}\cdot\boldsymbol{u} = -\boldsymbol{\nabla}\cdot(\boldsymbol{u}p) - p\partial_z w = -\boldsymbol{\nabla}\cdot(\boldsymbol{u}p) - \partial_z pw + w\partial_z p$$

we get for $\boldsymbol{\nabla}\cdot(\boldsymbol{u}p)$

$$(F^x_{ij} - F^x_{i-1,j})/(\Delta x t_i \cos\phi t_j) + (F^y_{ij} - F^y_{i,j-1})/(\Delta y t_j \cos\phi t_j)$$

with

$$F^x_{ij} = (p_{ij} + p_{i+1,j})/2u_{ij} \quad , \quad F^y_{ij} = (p_{ij} + p_{i,j+1})/2v_{ij}\cos\phi u_j$$

Continuity equation is given by

$$(w_k - w_{k-1})/\Delta z t_k + (u_{i,j,k} - u_{i-1,j,k})/(\cos\phi t_j \Delta x t_i) + (\cos\phi u_j v_{i,j,k} - \cos\phi u_{j-1} v_{i,j-1,k})/(\cos\phi t_j \Delta y t_j) = 0$$

Simpler for constant Δx, Δy

$$\begin{aligned}
\boldsymbol{u}\cdot\boldsymbol{\nabla}p &= \overline{u_{ij}\frac{(p_{i+1,j} - p_{i,j})}{\Delta x}}^{\,i} + \overline{v_{ij}\frac{(p_{i,j+1} - p_{ij})}{\Delta y}}^{\,j} \\
&= 0.5u_{ij}\frac{(p_{i+1,j} - p_{i,j})}{\Delta x} + 0.5u_{i-1,j}\frac{(p_{i,j} - p_{i-1,j})}{\Delta x} + 0.5v_{ij}\frac{(p_{i,j+1} - p_{ij})}{\Delta y} + 0.5v_{i,j-1}\frac{(p_{i,j} - p_{i,j-1})}{\Delta y} \\
&= 0.5u_{ij}\frac{(p_{i+1,j} + p_{i,j})}{\Delta x} - 0.5u_{i-1,j}\frac{(p_{i,j} + p_{i-1,j})}{\Delta x} - u_{ij}\frac{p_{i,j}}{\Delta x} + u_{i-1,j}\frac{p_{i,j}}{\Delta x} + ... \\
&= \boldsymbol{\nabla}\cdot(\boldsymbol{u}p) + p_{ij}(w_k - w_{k-1})/\Delta z t_k
\end{aligned}$$

When we sum the last term over k

$$\sum_{k=1}^{N_z} \Delta z t_k p_k (w_k - w_{k-1})/\Delta z t_k = \sum_{k=1}^{N_z} p_k w_k - \sum_{k=2}^{N_z} p_k w_{k-1} = p_N w_N + \sum_{k=1}^{N_z - 1} (p_k - p_{k+1}) w_k$$

or we write

$$\begin{aligned}
\boldsymbol{u}\cdot\boldsymbol{\nabla}p &= \boldsymbol{\nabla}\cdot(\boldsymbol{u}p) + (0.5(p_{k+1} + p_k)w_k - 0.5(p_k + p_{k-1})w_{k-1})/\Delta z t_k \\
&\quad + (-0.5(p_{k+1} - p_k)w_k - 0.5(p_k - p_{k-1})w_{k-1})/\Delta z t_k
\end{aligned}$$

now use $\partial_z p = -g\rho/\rho_0$, or $(p_{k+1} - p_k)/\Delta z w_k = -0.5g(\rho_k + \rho_{k+1})/\rho_0$

$$\boldsymbol{u}\cdot\boldsymbol{\nabla}p = \boldsymbol{\nabla}\cdot(\boldsymbol{u}p) + \partial_z(pw) + 0.25g/\rho_0\left(\frac{\Delta z w_k}{\Delta z t_k}(\rho_{k+1} + \rho_k)w_k + \frac{\Delta z w_{k-1}}{\Delta z t_k}(\rho_k + \rho_{k-1})w_{k-1}\right)$$

With $\boldsymbol{\nabla} \cdot (\boldsymbol{u}p)$

$$(F^x_{ij} - F^x_{i-1,j})/(\Delta x) + (F^y_{ij} - F^y_{i,j-1})/(\Delta y)$$
$$= 0.5((p_{i+1,j} + p_{i,j})u_{ij} - (p_{i,j} + p_{i-1,j})u_{i-1j})/\Delta x + 0.5((p_{ij} + p_{i,j+1})v_{ij} - (p_{i,j-1} + p_{i,j})v_{i,j-1})/\Delta y$$

with

$$F^x_{ij} = 0.5(p_{i+1,j} + p_{i,j})u_{ij} \quad , \quad F^y_{ij} = 0.5(p_{ij} + p_{i,j+1})v_{ij}$$

and with $\boldsymbol{\nabla} \cdot \boldsymbol{u}$

$$(u_{i,j,k} - u_{i-1,j,k})/\Delta x + (v_{i,j,k} - v_{i,j-1,k})/\Delta y = -(w_k - w_{k-1})/\Delta z t_k$$

7.5 Discrete potential energy and dynamic enthalpy

7.5.1 Potential energy

Potential energy $P = g\rho z/\rho_0$ is given by

$$\partial_t P + \boldsymbol{\nabla} \cdot \boldsymbol{u}P = g\frac{d\rho}{dt}z/\rho_0 + g\rho w/\rho_0 = \partial_z(g/\rho_0 K_v z \partial_z \rho) - g/\rho_0 K_v \partial_z \rho + g\rho w/\rho_0$$

With non-linear equation of state we have

$$\frac{dP}{dt} = \frac{g}{\rho_0}(z\rho_T \partial_z K \partial_z T + z\rho_S \partial_z K \partial_z S) + g\rho w/\rho_0 - g^2 z\rho_p w$$

The term $-g^2 z\rho_p w$ results from advection since

$$\begin{aligned} \rho_T \partial_t \theta + \rho_T \boldsymbol{u} \cdot \boldsymbol{\nabla}\theta + \rho_S \partial_t S + \rho_S \boldsymbol{u} \cdot \boldsymbol{\nabla} S &= \rho_S \partial_z(K\partial_z S) + \rho_T \partial_z(K\partial_z \theta) \\ &= \frac{d\rho}{dt} - \rho_p \frac{dp}{dt} \end{aligned}$$

with

$$\frac{d\rho}{dt} = \rho_T \frac{d\theta}{dt} + \rho_S \frac{dS}{dt} + \rho_p \frac{dp}{dt}$$

It is an exchange with mean dynamic internal energy and drops for dynamic enthalpy. The right hand side can be written as

$$\begin{aligned} \frac{dP}{dt} &= \frac{g}{\rho_0}(\partial_z z\rho_T K \partial_z T + \partial_z z\rho_S K \partial_z S) - \frac{g}{\rho_0} K(\rho_T \partial_z T + \rho_S \partial_z S) \\ &\quad -\frac{g}{\rho_0} zK(\partial_z T \partial_z \rho_T + \partial_z S \partial_z \rho_S) + g\rho w/\rho_0 - g^2 z\rho_p w \end{aligned}$$

The first term is a flux, the second the usual exchange with TKE, the third term is cabelling. For the equation of state by Vallis it becomes negative (see below).

7.5.2 Dynamic enthalpy

For an incompressible equation of state dynamic enthalpy $H_d = -g\int_z^0 \rho(S,\theta,z')/\rho_0 dz'$ is identical to potential energy, since ρ does not depend on z

$$H_d = -g\rho(S,\theta)/\rho_0 \int_z^0 dz' = g\rho z/\rho_0 = P$$

With compressibility we find

$$\begin{aligned}
\rho_0 \overline{\frac{d}{dt}} H_d(S,\theta,z) &= g\rho w - g\tilde{\rho}_T \frac{d\theta}{dt} - g\tilde{\rho}_S \frac{dS}{dt} \\
&= g\rho w - \partial_z(g\tilde{\rho}_T K \partial_z\theta) - \partial_z(g\tilde{\rho}_S K \partial_z S) + gK\partial_z\theta\partial_z\tilde{\rho}_T + gK\partial_z S\partial_z\tilde{\rho}_S - g\tilde{\rho}_T\dot{\theta} - g\tilde{\rho}_S\dot{S} \\
\overline{\frac{d}{dt}} H_d(S,\theta,z) &= \frac{g\rho}{\rho_0} w + \partial_z(H_T K\partial_z\theta) + \partial_z(H_S K\partial_z S) - K\partial_z\theta\partial_z H_T - K\partial_z S\partial_z H_S + H_T\dot{\theta} + H_S\dot{S}
\end{aligned}$$

with $\tilde{\rho}_T = \int_z^0 \rho_T(S,\theta,z')dz' = -(\rho_0/g)\partial H_d/\partial\theta$. Using

$$\partial_z\tilde{\rho}|_{x,y} = \partial_z\tilde{\rho}|_{S,H} + \frac{\partial\tilde{\rho}}{\partial\theta}|_{S,z}\partial_z\theta + \frac{\partial\tilde{\rho}}{\partial S}|_{\theta,z}\partial_z S = -\rho + \tilde{\rho}_\theta\partial_z\theta + \tilde{\rho}_S\partial_z S$$

The exchange terms become

$$K\partial_z\theta\partial_z\tilde{\rho}_T + K\partial_z S\partial_z\tilde{\rho}_S = -\rho_T K\partial_z\theta - \rho_S K\partial_z S + \tilde{\rho}_{TT}K(\partial_z\theta)^2 + 2\tilde{\rho}_{TS}K(\partial_z S)(\partial_z\theta) + \tilde{\rho}_{SS}K(\partial_z S)$$

The first two are the usual exchange with TKE, the remaining three are cabbeling terms. For the equation of state by Vallis (2006), only the first of the cabellling terms is active and $\tilde{\rho}_{TT}K(\partial_z\theta)^2 < 0$ since $\rho_{TT} = -\rho_0\beta_T^\star < 0$.

7.5.3 Exchange of potential energy with TKE for a linear equation of state

On a discrete level this becomes

$$\begin{aligned}
\partial_z K_v \partial_z\rho = (F_k^\rho - F_{k-1}^\rho)/\Delta zt_k \quad , \quad F_k^\rho = K_k^v(\rho_{k+1}-\rho_k)/\Delta zw_k \\
z\partial_z K_v\partial_z\rho = zt_k(F_k^\rho - F_{k-1}^\rho)/\Delta zt_k \\
= ((zt_k + zt_{k+1})/2F_k^\rho - (zt_k + zt_{k-1})/2F_{k-1}^\rho)/\Delta zt_k \\
+(zt_k/2F_k^\rho - zt_k/2F_{k-1}^\rho)/\Delta zt_k - (zt_{k+1}/2F_k^\rho - zt_{k-1}/2F_{k-1}^\rho)/\Delta zt_k \\
= (zw_k F_k^\rho - zw_{k-1}F_{k-1}^\rho)/\Delta zt_k - (\Delta zw_k F_k^\rho + \Delta zw_{k-1}F_{k-1}^\rho)/(2\Delta zt_k)
\end{aligned}$$

with $zw_k = (zt_{k+1} + zt_k)/2$ and $\Delta zw_k = zt_{k+1} - zt_k$. The second term is the exchange with TKE, the first is a flux divergence. Integrating the second term over z

$$0.5\sum_{k=1}^{N}(\Delta zw_k F_k^\rho + \Delta zw_{k-1}F_{k-1}^\rho) = 0.5\sum_{k=1}^{N}\Delta zw_k F_k^\rho + 0.5\sum_{k=2}^{N}\Delta zw_{k-1}F_{k-1}^\rho = 0.5\Delta zw_N F_N^\rho + \sum_{k=1}^{N-1}\Delta zw_k F_k^\rho$$

and the first also

$$\sum_{k=1}^{N}(zw_kF_k^\rho - zw_{k-1}F_{k-1}^\rho) = \sum_{k=1}^{N} zw_kF_k^\rho - \sum_{k=1}^{N-1} zw_kF_k^\rho = zw_NF_N^\rho = 0$$

since $zw_N = 0$. Check by summing

$$\sum_{k=1}^{N} zt_k(F_k^\rho - F_{k-1}^\rho) = zt_NF_N^\rho - \sum_{k=1}^{N-1} \Delta zw_kF_k^\rho$$

The forcing which enters the TKE equation defined on W grid is then simply

$$F_k^\rho \text{ for } k = 1, N-1 \quad , \quad F_N^\rho \text{ for } k = N$$

It is possible that the surface density flux F_N^ρ drains out the TKE at the surface. This is because we need to mix the first layer, and this energy has to be taken from somewhere. We need to increase the TKE forcing or to reduce the surface density flux in such a case.

7.5.4 Exchange of potential energy with TKE for a nonlinear equation of state

With a nonlinear equation of state we have

$$z\rho_T\partial_z K\partial_z T = zt_k(\rho_T)_k(F_k^T - F_{k-1}^T)/\Delta zt_k$$
$$= ((zt_k(\rho_T)_k + zt_{k+1}(\rho_T)_{k+1})/2F_k^T - (zt_k(\rho_T)_k + zt_{k-1}(\rho_T)_{k-1})/2F_{k-1}^T)/\Delta zt_k$$
$$+(zt_k(\rho_T)_k/2F_k^T - zt_k(\rho_T)_k/2F_{k-1}^T)/\Delta zt_k - (zt_{k+1}(\rho_T)_{k+1}/2F_k^T - zt_{k-1}(\rho_T)_{k-1}/2F_{k-1}^T)/\Delta zt_k$$

and similar for S. The first term is a flux which integrates to zero, since

$$0.5\sum_{k=1}^{N}((zt_k(\rho_T)_k + zt_{k+1}(\rho_T)_{k+1})F_k^T - (zt_k(\rho_T)_k + zt_{k-1}(\rho_T)_{k-1})F_{k-1}^T) =$$
$$0.5(zt_N(\rho_T)_N + zt_{N+1}(\rho_T)_{N+1})F_N^T = zw_N(\rho_T)_NF_N^T = 0$$

for $\partial\rho_T/\partial p = 0$. For compressible equation of state there is a flux. The second can be written as

$$-(zt_{k+1}(\rho_T)_{k+1} - zt_k(\rho_T)_k)F_k^T/(2\Delta zt_k) - (zt_k(\rho_T)_k - zt_{k-1}(\rho_T)_{k-1})F_{k-1}^T/(2\Delta zt_k)$$

Integrate over z

$$-0.5\sum_{k=1}^{N}(zt_{k+1}(\rho_T)_{k+1} - zt_k(\rho_T)_k)F_k^T - 0.5\sum_{k=1}^{N}(zt_k(\rho_T)_k - zt_{k-1}(\rho_T)_{k-1})F_{k-1}^T$$
$$= -0.5(\rho_T)_N\Delta zw_NF_N^T - \sum_{k=1}^{N-1}(zt_{k+1}(\rho_T)_{k+1} - zt_k(\rho_T)_k)F_k^T$$

The forcing which enters the TKE equation defined on W grid is then

$$\frac{(zt_{k+1}(\rho_T)_{k+1} - zt_k(\rho_T)_k)}{\Delta zw_k}F_k^T \text{ for } k = 1, N-1 \quad , \quad (\rho_T)_NF_N^T \text{ for } k = N$$

7.5.5 Exchange of dynamic enthalpy with TKE

With compressibility we find

$$\begin{aligned}\rho_0\overline{\frac{d}{dt}}H_d(S,\theta,z) &= g\rho w - g\tilde{\rho}_T\partial_z(K\partial_z\theta) - g\tilde{\rho}_S\partial_z(K\partial_z S)\\ &= g\rho w - \partial_z(g\tilde{\rho}_T K\partial_z\theta) - \partial_z(g\tilde{\rho}_S K\partial_z S) + gK\partial_z\theta\partial_z\tilde{\rho}_T + gK\partial_z S\partial_z\tilde{\rho}_S\end{aligned}$$

with $\tilde{\rho}_T = \int_z^0 \rho_T(S,\theta,z')dz'$. On a discrete level this becomes

$$\begin{aligned}\tilde{\rho}_T\partial_z K\partial_z T &= (\tilde{\rho}_T)_k(F_k^T - F_{k-1}^T)/\Delta zt_k\\ &= (((\tilde{\rho}_T)_k + (\tilde{\rho}_T)_{k+1})/2F_k^T - ((\tilde{\rho}_T)_k + (\tilde{\rho}_T)_{k-1})/2F_{k-1}^T)/\Delta zt_k\\ &\quad -0.5[(\tilde{\rho}_T)_{k+1} - (\tilde{\rho}_T)_k]F_k^T/\Delta zt_k - 0.5[(\tilde{\rho}_T)_k - (\tilde{\rho}_T)_{k-1}]F_{k-1}^T\Delta zt_k\end{aligned}$$

and similar for S. The first term is a flux, the second the exchange with TKE. Integrating the flux term over depth yields

$$0.5\sum_{n=1}^{N}[((\tilde{\rho}_T)_k + (\tilde{\rho}_T)_{k+1})F_k - ((\tilde{\rho}_T)_k + (\tilde{\rho}_T)_{k-1})F_{k-1}]$$

$$= 0.5\sum_{n=1}^{N}((\tilde{\rho}_T)_k + (\tilde{\rho}_T)_{k+1})F_k - 0.5\sum_{n=1}^{N-1}((\tilde{\rho}_T)_{k+1} + (\tilde{\rho}_T)_k)F_k = 0.5((\tilde{\rho}_T)_N + (\tilde{\rho}_T)_{N+1})F_N$$

which only vanishes if $\partial\rho/\partial p = 0$. Integrating the second over depth yields

$$\begin{aligned}&-0.5\sum_{n=1}^{N}[(\tilde{\rho}_T)_{k+1} - (\tilde{\rho}_T)_k]F_k^T - 0.5\sum_{n=2}^{N}[(\tilde{\rho}_T)_k - (\tilde{\rho}_T)_{k-1}]F_{k-1}^T\\ &= -0.5[(\tilde{\rho}_T)_{N+1} - (\tilde{\rho}_T)_N]F_N^T - \sum_{n=1}^{N-1}[(\tilde{\rho}_T)_{k+1} - (\tilde{\rho}_T)_k]F_k^T\end{aligned}$$

The forcing which enters the TKE equation defined on W grid is then

$$-\frac{(\tilde{\rho}_T)_{k+1} - (\tilde{\rho}_T)_k}{\Delta zw_k}F_k^T \text{ for } k = 1, N-1 \quad , \quad -\frac{[(\tilde{\rho}_T)_{N+1} - (\tilde{\rho}_T)_N]F_N^T}{\Delta zw_N} \text{ for } k = N$$

At the surface, the flux contribution adds to

$$\frac{2(\tilde{\rho}_T)_N F_N}{\Delta zw_N} \text{ for } k = N$$

7.5.6 Exchange with TKE by horizontal diffusion

Horizontal diffusion of T and S also affects potential energy for a non-linear equation of state.

$$\begin{aligned}\partial_x K\partial_x T = (F_i - F_{i+1})/\Delta xt_i \quad , \quad F_i = K_i(T_{i+1} - T_i)/\Delta xu_i\\ \rho_T z\partial_x K\partial_x T = (\rho_T)_i z(F_i - F_{i+1})/\Delta xt_i\\ = z(((\rho_T)_i + (\rho_T)_{i+1})/2F_i - z((\rho_T)_i + (\rho_T)_{i-1})/2F_{i-1})/\Delta xt_i\\ +z((\rho_T)_i/2F_i - (\rho_T)_i/2F_{i-1})/\Delta xt_i - z((\rho_T)_{i+1}/2F_i - (\rho_T)_{i-1}/2F_{i-1})/\Delta xt_i\end{aligned}$$

and similar for S. The first term is a flux divergence and integrates to zero. The second term can be written as

$$-z((\rho_T)_{i+1} - (\rho_T)_i)F_i/(2\Delta x t_i) - z((\rho_T)_i - (\rho_T)_{i-1})F_{i-1}/(2\Delta x t_i)$$

This forcing has to be interpolated to the W grid as for the frictional terms. The sign of the forcing

$$\rho_T z \partial_x K \partial_x T = \partial_x(\rho_T z K \partial_x T) - zK\partial_x T \partial_x \rho_T = \partial_x(\rho_T z K \partial_x T) - zK(\partial_x T)^2 \rho_{TT}$$

depends on the sign of ρ_{TT}. For the equation of state by Vallis (2006) we have $\rho_{TT} = -\rho_0 \beta_{Ts} < 0$ and thus

$$-zK(\partial_x T)^2 \rho_{TT} < 0$$

It is thus always a forcing of TKE.

7.5.7 Exchange by isopycnal diffusion

The isopycnal diffusion operator in small slope approximation is given by

$$\partial_t T \;=\; ... + \boldsymbol{\nabla} \cdot \boldsymbol{K} \boldsymbol{\nabla} T$$

with

$$\boldsymbol{K} \;=\; K_{iso} \begin{pmatrix} 1 & 0 & s_x \\ 0 & 1 & s_y \\ s_x & s_y & s_x^2 + s_y^2 \end{pmatrix} + K_{gm} \begin{pmatrix} 0 & 0 & -s_x \\ 0 & 0 & -s_y \\ s_x & s_y & 0 \end{pmatrix}$$

with the isopycnal slopes $s_x = -\partial_x \rho / \partial_z \rho$ and $s_y - \partial_y \rho / \partial_z \rho$. The effect of dynamic enthalpy is given by

$$\rho_0 \frac{\overline{d}}{dt} H_d(S, \theta, z) \;=\; g\rho w - g\tilde{\rho}_T \boldsymbol{\nabla} \cdot \boldsymbol{K} \boldsymbol{\nabla} \theta - g\tilde{\rho}_S \boldsymbol{\nabla} \cdot \boldsymbol{K} \boldsymbol{\nabla} S$$

The isopycnal mixing part can be treated as for horizontal fluxes and vertical mixing. This also holds for skew the diffusion. The discrete functional for the flux $\boldsymbol{K}\boldsymbol{\nabla}T$ by Griffies et al is used for isopycnal mixing and skew diffusion. The vector streamfunction $\boldsymbol{B}$ for the eddy driven velocity $\boldsymbol{\nabla} \times \boldsymbol{B}$ is given by

$$\boldsymbol{B} = \begin{pmatrix} -K_{gm} s_y \\ K_{gm} s_x \\ 0 \end{pmatrix} \quad , \quad \boldsymbol{\nabla} \times \boldsymbol{B} = \begin{pmatrix} -\partial_z K_{gm} s_x \\ -\partial_z K_{gm} s_y \\ \partial_x K_{gm} s_x + \partial_y K_{gm} s_y \end{pmatrix}$$

7.5.8 Exchange by advective fluxes

Accept for a change in sign, the advective fluxes behave the same as the diffusive ones.

$$\begin{aligned}
-\tilde{\rho}_T\partial_x(uT) &= -\partial_x(\tilde{\rho}_T uT) + uT\partial_x\tilde{\rho}_T \\
-\tilde{\rho}_T\partial_x(uT) &= -\frac{(\tilde{\rho}_T)_i(F_i - F_{i+1})}{\Delta xt_i} = -\frac{((\tilde{\rho}_T)_i + (\tilde{\rho}_T)_{i+1})/2F_i - ((\tilde{\rho}_T)_i + (\tilde{\rho}_T)_{i-1})/2F_{i-1}}{\Delta xt_i} \\
&\quad + \frac{((\tilde{\rho}_T)_{i+1} - (\tilde{\rho}_T)_i)F_i + ((\tilde{\rho}_T)_i - (\tilde{\rho}_T)_{i-1})F_{i-1}}{2\Delta xt_i}
\end{aligned}$$

where F_i is the advective zonal flux. The first term is a flux which integrates to zero, the second is an exchange with TKE. For the vertical flux we have

$$\begin{aligned}
-\tilde{\rho}_T\partial_z(wT) &= -\partial_z(\tilde{\rho}_T wT) + wT\partial_z\tilde{\rho}_T = -\partial_z(\tilde{\rho}_T wT) + wT(-\rho_T + \tilde{\rho}_{TT}\partial_z\theta + \tilde{\rho}_{TS}\partial_z S) \\
&= -(\tilde{\rho}_T)_k(F_k^T - F_{k-1}^T)/\Delta zt_k \\
&= -\frac{((\tilde{\rho}_T)_k + (\tilde{\rho}_T)_{k+1})F_k^T - ((\tilde{\rho}_T)_k + (\tilde{\rho}_T)_{k-1})F_{k-1}^T}{2\Delta zt_k} \\
&\quad + \frac{[(\tilde{\rho}_T)_{k+1} - (\tilde{\rho}_T)_k]F_k^T + [(\tilde{\rho}_T)_k - (\tilde{\rho}_T)_{k-1}]F_{k-1}^T}{2\Delta zt_k}
\end{aligned}$$

where F_k is the vertical advective flux. Using

$$\partial_z\tilde{\rho}|_{x,y} = -\rho + \tilde{\rho}_\theta\partial_z\theta + \tilde{\rho}_S\partial_z S$$

The equation of potential energy $P = g\rho z/\rho_0$ in the model

$$\partial_t P = g\rho w/\rho_0 + \boldsymbol{\nabla}\cdot(..) + KN^2 + \left(-g/\rho_0\partial_z(z\rho_T)K\partial_z\theta - g/\rho_0\partial_z(z\rho_S)K\partial_z S - KN^2\right)$$

$$-g/\rho_0 z(\rho_T\boldsymbol{\nabla}\cdot(\boldsymbol{u}\theta) + \rho_S\boldsymbol{\nabla}\cdot(\boldsymbol{u}S) + \rho_p\boldsymbol{\nabla}\cdot(\boldsymbol{u}p_0) - \boldsymbol{\nabla}\cdot(\boldsymbol{u}\rho))$$

The equation of dynamic enthalpy $H = -g/\rho_0\int_z^0 \rho dz'$ in the model

$$\partial_t H = g\rho w/\rho_0 + \boldsymbol{\nabla}\cdot(..) + KN^2 + [g/\rho_0\partial_z(\tilde{\rho}_T)K\partial_z\theta + g/\rho_0\partial_z(\tilde{\rho}_S)K\partial_z S - KN^2]$$

$$-(-g/\rho_0\tilde{\rho}_T\boldsymbol{\nabla}\cdot(\boldsymbol{u}\theta) - g/\rho_0\tilde{\rho}_S\boldsymbol{\nabla}\cdot(\boldsymbol{u}S) + g/\rho_0\rho w - \boldsymbol{\nabla}\cdot(\boldsymbol{u}H))$$

7.5.9 Exchange of potential energy with mean kinetic energy

Now calculate

$$\begin{aligned}
z(\partial_t\rho + \boldsymbol{\nabla}\cdot\boldsymbol{u}\rho) + \partial_z w\rho z &= z(\partial_t\rho + \boldsymbol{\nabla}\cdot\boldsymbol{u}\rho) + 0.5(w_k(\rho_k + \rho_{k+1})zw_k - w_{k-1}(\rho_k + \rho_{k-1})zw_{k-1})/\Delta zt_k \\
&= z(\partial_t\rho + \boldsymbol{\nabla}\cdot\boldsymbol{u}\rho) + zt_k 0.5(w_k(\rho_k + \rho_{k+1}) - w_{k-1}(\rho_k + \rho_{k-1}))/\Delta zt_k \\
&\quad - zt_k 0.25(w_k(\rho_k + \rho_{k+1}) - w_{k-1}(\rho_k + \rho_{k-1}))/\Delta zt_k \\
&\quad + 0.25(w_k(\rho_k + \rho_{k+1})zt_{k+1} - w_{k-1}(\rho_k + \rho_{k-1})zt_{k-1})/\Delta zt_k
\end{aligned}$$

$$= z(\partial_t \rho + \boldsymbol{\nabla} \cdot \boldsymbol{u}\rho + \partial_z w\rho)$$
$$+0.25(w_k(\rho_k + \rho_{k+1})\Delta z w_k + w_{k-1}(\rho_k + \rho_{k-1})\Delta z w_{k-1})/\Delta z t_k$$
$$g/\rho_0 \frac{dP}{dt} = g/\rho_0 z \frac{d\rho}{dt} - 0.5(w_k(p_{k+1} - p_k) + w_{k-1}(p_k - p_{k-1}))/\Delta z t_k$$

with $zw_k = (zt_{k+1} + zt_k)/2$ and $\Delta z w_k = zt_{k+1} - zt_k$ and with with $(p_{k+1} - p_k)/\Delta z w_k = -0.5g(\rho_k + \rho_{k+1})/\rho_0$. The second term is the exchange with KE. Or different

$$z(\partial_t \rho + \boldsymbol{\nabla} \cdot \boldsymbol{u}\rho) + \partial_z w\rho z = z(\partial_t \rho + \boldsymbol{\nabla} \cdot \boldsymbol{u}\rho) + 0.5(w_k(zt_k\rho_k + zt_{k+1}\rho_{k+1}) - w_{k-1}(zt_k\rho_k + zt_{k-1}\rho_{k-1}))/\Delta z t_k$$
$$= z(\partial_t \rho + \boldsymbol{\nabla} \cdot \boldsymbol{u}\rho) + zt_k 0.5(w_k(\rho_k + \rho_{k+1}) - w_{k-1}(\rho_k + \rho_{k-1}))/\Delta z t_k$$
$$+0.5(w_k zt_{k+1}\rho_{k+1} - w_{k-1}zt_{k-1}\rho_{k-1})/\Delta z t_k - zt_k 0.5(w_k\rho_{k+1} - w_{k-1}\rho_{k-1})/\Delta z t_k$$
$$= z(\partial_t \rho + \boldsymbol{\nabla} \cdot \boldsymbol{u}\rho + \partial_z w\rho) + 0.5(w_k\rho_{k+1}\Delta z w_k + \rho_{k-1}w_{k-1}\Delta z w_{k-1})/\Delta z t_k$$

We integrate over z.

$$g/\rho_0/4 \sum_{k=1}^{N} (w_k(\rho_k + \rho_{k+1})\Delta z w_k + w_{k-1}(\rho_k + \rho_{k-1})\Delta z w_{k-1})$$
$$= g/\rho_0/4 \sum_{k=1}^{N} w_k(\rho_k + \rho_{k+1})\Delta z w_k + g/\rho_0/4 \sum_{k=2}^{N} w_{k-1}(\rho_k + \rho_{k-1})\Delta z w_{k-1} =$$
$$= g/\rho_0/4 w_N(\rho_N + \rho_{N+1})\Delta z w_N + g/\rho_0/2 \sum_{k=1}^{N-1} w_k(\rho_k + \rho_{k+1})\Delta z w_k$$
$$= -w_N(p_s - p_N) - \sum_{k=1}^{N-1} w_k(p_{k+1} - p_k)$$

Or different

$$g/\rho_0 \sum_{k=1}^{N} 0.5(w_k\rho_{k+1}\Delta z w_k + \rho_{k-1}w_{k-1}\Delta z w_{k-1}) = g/\rho_0 0.5 w_N\rho_{N+1}\Delta z w_N + g/\rho_0 \sum_{k=1}^{N-1} 0.5 w_k(\rho_{k+1} + \rho_k)\Delta z w_k$$

The exchange in the KE budget becomes

$$\sum_{k=1}^{N} p_k(w_k - w_{k-1}) = \sum_{k=1}^{N-1} p_k w_k - \sum_{k=1}^{N-1} p_{k+1} w_k = \sum_{k=1}^{N-1} w_k(p_k - p_{k+1}) = 0.5g/\rho_0 \sum_{k=1}^{N-1} w_k(\rho_k + \rho_{k+1})\Delta z w_k$$

with $(p_{k+1} - p_k)/\Delta z w_k = -0.5g(\rho_k + \rho_{k+1})/\rho_0$, and is thus identical.

7.5.10 Exchange of dynamic enthalpy with mean kinetic energy

For dynamic enthalpy $H = -g/\rho_0 \int_z^0 \rho dz'$ we have

$$\rho_0 \frac{dH}{dt} = g\rho \frac{dz}{dt} - g\tilde{\rho}_T \frac{dT}{dt} - g\tilde{\rho}_S \frac{dS}{dt} = g\rho w - g\tilde{\rho}_T \frac{dT}{dt} - g\tilde{\rho}_S \frac{dS}{dt}$$

On a discrete level this becomes

$$\partial_z(Hw) = 0.5(w_k(H_{k+1} + H_k) - w_{k-1}(H_k + H_{k-1}))/\Delta z t_k$$

7.6 TKE equation and vertical mixing

The TKE equation is given by

$$\partial_t E = \partial_z \kappa_e \partial_z E - c_\epsilon E^{3/2}/L - N^2 \kappa_h + \kappa_m (\partial_z \boldsymbol{u}_h)^2 + A_h (\boldsymbol{\nabla} \boldsymbol{u}_h)^2$$

with $\kappa_m = c_k L E^{1/2}$, $\kappa_e = \alpha_{tke} \kappa_m$ and $\kappa_h = \kappa_m / P_r$ with $P_r = 6.6Ri$ and $1 <= Pr <= 10$.

7.6.1 TKE at W points

TKE E is defined at W points. Similar for EKE and internal wave energy. Both vertical diffusion of E and dissipation are treated implicitly. Discrete form is given by

$$(E_k^n - E_k^{n-1})/\Delta t = (F_k - F_{k-1})/\Delta z w_k - c_\epsilon E_k^n (E_k^{n-1})^{1/2}/L + P_k$$

with $F_k = \kappa_k (E_{k+1}^n - E_k^n)/\Delta z t_{k+1}$ and $\kappa_k = 0.5((\kappa_e)_{k+1} + (\kappa_e)_k)$, and P_k forcing from dissipation and buoyancy work. Rewrite as

$$E_k^n (1 + c_\epsilon \Delta t (E_k^{n-1})^{1/2}/L) - (E_{k+1}^n - E_k^n) \frac{\kappa_k \Delta t}{\Delta z t_{k+1} \Delta z w_k} + (E_k^n - E_{k-1}^n) \frac{\kappa_{k-1} \Delta t}{\Delta z t_k \Delta z w_k} = E_k^{n-1} + P_k \Delta t$$

or

$$A_k E_{k-1}^n + B_k E_k^n + C_k E_{k+1}^n = E_k^{n-1} + P_k \Delta t$$

$$C_k = -\frac{\delta_k}{\Delta z w_k} \quad , \quad B_k = 1 + c_\epsilon \Delta t (E_k^{n-1})^{1/2}/L + \frac{\delta_k}{\Delta z w_k} + \frac{\delta_{k-1}}{\Delta z w_k} \quad , \quad A_k = -\frac{\delta_{k-1}}{\Delta z w_k}$$

with $\delta_k = \frac{\kappa_k \Delta t}{\Delta z t_{k+1}}$. At $k = 1$, near the bottom we have

$$A_k E_{k-1}^n + B_k E_k^n + C_k E_{k+1}^n = E_k^{n-1} + P_k \Delta t - F_b \Delta t / \Delta z w_k$$

$$C_k = -\frac{\delta_k}{\Delta z w_k} \ , \ B_k = 1 + c_\epsilon \Delta t (E_k^{n-1})^{1/2}/L + \frac{\delta_k}{\Delta z w_k} \ , \ A_k = 0$$

F_b is the TKE flux at $zt_1 = -h + \Delta z_1/2$, which is not at the bottom. At $k = N$, near the surface, the W grid box is only half thick. We have

$$A_k E_{k-1}^n + B_k E_k^n + C_k E_{k+1}^n = E_k^{n-1} + P_k \Delta t + F_s \frac{\Delta t}{0.5 \Delta z w_k}$$

$$C_k = 0 \ , \ B_k = 1 + c_\epsilon \Delta t (E_k^{n-1})^{1/2}/L + \frac{\delta_{k-1}}{0.5 \Delta z w_k} \ , \ A_k = -\frac{\delta_{k-1}}{0.5 \Delta z w_k}$$

where F_s is the surface flux of TKE. A Dirichlet boundary condition looks like

$$E_k^n (1 + c_\epsilon \Delta t (E_k^{n-1})^{1/2}/L) - (E_{surf} - E_k^n) \frac{\delta_k}{0.5 \Delta z w_k} + (E_k^n - E_{k-1}^n) \frac{\delta_{k-1}}{0.5 \Delta z w_k} = E_k^{n-1} + P_k \Delta t$$

$$A_k E_{k-1}^n + B_k E_k^n + C_k E_{k+1}^n = E_k^{n-1} + P_k \Delta t + E_{surf} \frac{\delta_k}{0.5 \Delta z w_k}$$

$$C_k = 0 \ , \ B_k = 1 + c_\epsilon \Delta t (E_k^{n-1})^{1/2}/L + \frac{\delta_k}{0.5 \Delta z w_k} + \frac{\delta_{k-1}}{0.5 \Delta z w_k} \ , \ A_k = -\frac{\delta_{k-1}}{0.5 \Delta z w_k}$$

7.6.2 Advection for tracer at W grid

We advect tracer on the W grid such as TKE with the following lateral advection velocity

$$\begin{aligned} U_{i,j,k} &= 0.5\Delta zt_{k+1}/\Delta zw_k u_{k+1} + 0.5\Delta zt_k/\Delta zw_k u_k \\ U_{i,j,k} &= 0.5\Delta zt_{k+1}/\Delta zw_k u_{k+1} + \Delta zt_k/\Delta zw_k u_k \quad , \quad k=1 \\ U_{i,j,k} &= 0.5\Delta zt_k/\Delta zw_k u_k \quad , \quad k=N \end{aligned}$$

and similar for V. In case of topography U and V have to be redirected at the bottom. The vertical velocity is given by continuity

$$W_{i,j,k} = W_{i,j,k-1} - \Delta zw_k \left((U_{i,j,k} - U_{i-1,j,k})/\Delta xt_i - (V_{i,j,k} - V_{i,j-1,k})/\Delta yt_j\right)$$

For u and v it holds from the streamfunction formalism that

$$0 = \sum_{n=1}^{N} \Delta zt_k \left((u_{i,j,k} - u_{i-1,j,k})/\Delta xt_i + (u_{i,j,k} - u_{i,j-1,k})/\Delta yt_j\right) = -\sum_{n=1}^{N}(w_{i,j,k} - w_{i,j,k-1}) = -w_{i,j,N}$$

suppressing the cos factors. The same has to hold for U and V

$$\begin{aligned} -\sum_{n=1}^{N} \Delta zw_k(W_k - W_{k-1})/\Delta zw_k = -\sum_{n=1}^{N} W_k + \sum_{n=0}^{N-1} W_k = \\ -W_{i,j,N} = \sum_{n=1}^{N} \Delta zw_k(U_{i,j,k} - U_{i-1,j,k})/\Delta xt_i + \sum_{n=1}^{N} \Delta zw_k(V_{i,j,k} - V_{i,j-1,k})/\Delta yt_j \quad = \\ \sum_{n=1}^{N-1} 0.5(u_{i,j,k+1} - u_{i-1,j,k+1})\Delta zt_{k+1}/\Delta xt_i + \sum_{n=1}^{N} 0.5(u_{i,j,k} - u_{i-1,j,k})\Delta zt_k/\Delta xt_i \\ + \sum_{n=1}^{N-1} 0.5(v_{i,j,k+1} - v_{i,j-1,k+1})\Delta zt_{k+1}/\Delta yt_j + \sum_{n=1}^{N} 0.5(v_{i,j,k} - v_{i,j-1,k})\Delta zt_k/\Delta yt_j \\ +0.5(u_{i,j,1} - u_{i-1,j,1})\Delta zt_1/\Delta xt_i + 0.5(v_{i,j,1} - v_{i,j-1,1})\Delta zt_1/\Delta yt_j \\ = 0 \end{aligned}$$

7.6.3 Implicit vertical mixing at T points

Consider a tracer E defined at T points such as temperature. Vertical diffusion of E is treated implicitly. Discrete form is given by

$$(E_k^n - E_k^{n-1})/\Delta t = (F_k - F_{k-1})/\Delta zt_k - I_k E_k^n + P_k(E^{n-1})$$

with $F_k = \kappa_k(E_{k+1}^n - E_k^n)/\Delta zw_k$, with a parameter I_k and P_k containing other explicit terms. Rewrite as

$$E_k^n(1 + I_k\Delta t) - (E_{k+1}^n - E_k^n)\frac{\kappa_k \Delta t}{\Delta zw_k \Delta zt_k} + (E_k^n - E_{k-1}^n)\frac{\kappa_{k-1}\Delta t}{\Delta zw_{k-1}\Delta zt_k} = E_k^{n-1} + P_k\Delta t$$

or

$$A_k E^n_{k-1} + B_k E^n_k + C_k E^n_{k+1} = E^{n-1}_k + P_k \Delta t$$
$$C_k = -\frac{\delta_k}{\Delta z t_k} \quad , \quad B_k = 1 + I_k \Delta t + \frac{\delta_k}{\Delta z t_k} + \frac{\delta_{k-1}}{\Delta z t_k} \quad , \quad A_k = -\frac{\delta_{k-1}}{\Delta z t_k}$$

with $\delta_k = \frac{\kappa_k \Delta t}{\Delta z w_k}$. At $k = 1$, at the bottom we have

$$A_k E^n_{k-1} + B_k E^n_k + C_k E^n_{k+1} = E^{n-1}_k + P_k \Delta t - F_b \Delta t / \Delta z t_k$$
$$C_k = -\frac{\delta_k}{\Delta z t_k} \ , \ B_k = 1 + I_k \Delta t + \frac{\delta_k}{\Delta z t_k} \ , \ A_k = 0$$

F_b is the flux of E at $zw_0 = -h$. At $k = N$, at the surface, we have

$$A_k E^n_{k-1} + B_k E^n_k + C_k E^n_{k+1} = E^{n-1}_k + P_k \Delta t + F_s \frac{\Delta t}{\Delta z t_k}$$
$$C_k = 0 \ , \ B_k = 1 + I_k \Delta t + \frac{\delta_{k-1}}{\Delta z t_k} \ , \ A_k = -\frac{\delta_{k-1}}{\Delta z t_k}$$

F_s is the flux of E at $zw_N = 0$.

Acknowledgments

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

References

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. and R. J. Greatbatch, 2008: Towards a mesoscale eddy closure. *Ocean Modelling*, **20**, 223–239.

Eden, C. and J. Willebrand, 2001: Mechanism of interannual to decadal variability of the North Atlantic circulation. *J. Climate*, **14**(10), 2266–2280.

Gaspar, P., Y. Gregoris, and J.-M. Lefevre, 1990: A simple eddy kinetic energy model for simulations of the oceanic vertical mixing: tests at station PAPA and Long-Term Upper Ocean Study site. *J. Geophys. Res.*, **95**, 16179–16193.

Gent, P. R., J. Willebrand, T. J. McDougall, and J. C. McWilliams, 1995: Parameterizing eddy-induced tracer transports in ocean circulation models. *J. Phys. Oceanogr.*, **25**, 463–474.

Griffies, S. M., 1998: The Gent-McWilliams Skew Flux. *J. Phys. Oceanogr.*, **28**, 831–841.

McDougall, T. J. and P. C. McIntosh, 2001: The temporal-residual-mean velocity. Part II: Isopycnal interpretation and the tracer and momentum equations. *J. Phys. Oceanogr.*, **31**(5), 1222–1246.

Olbers, D. and C. Eden, 2013: A global model for the diapycnal diffusivity induced by internal gravity waves. *J. Phys. Oceanogr.*, **43**, 1759–1779.

Olbers, D., J. Willebrand, and C. Eden, 2012: *Ocean Dynamics*. Springer, Heidelberg.

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.