Aronnax Documentation

Release 0.3.0

Ed Doddridge & Alexey Radul

Mar 12, 2019

Contents:

Aronnax is an idealised and easily configurable isopycnal ocean circulation model. Aronnax can be run as either a reduced gravity model with n + 1/2 layers, or with n layers and variable bathymetry.

Aronnax is

- Easy to install on a laptop or a compute node, including without administrative privileges.
- Easy to configure. All parameters, including grid size, are specified at runtime in a simple configuration file or function call.
- Easy to use. Aronnax can be run from the shell as a Python script.
- Easy to learn and understand, with extensive online documentation, including a complete description of the physics and the numerics.
- Verified. Aronnax successfully reproduces published results from idealised models appearing in the literature.
- Fast. The main integration loop is a Fortran program, wrapped in Python for convenient use.

CHAPTER 1

The Aronnax Model

1.1 The Physics

Aronnax can run in two different modes:

- n + 1/2 layer mode, in which the bottom layer is quiescent and infinitely deep
- n layer mode, in which the bottom topography is specified and all layers have finite thickness

1.1.1 n + 1/2 layer mode

The n + 1/2 layer mode is very cheap to run: Each time-step takes time linear in the number of grid points.

1.1.2 n layer mode

The n layer mode is more expensive, because the integration method is party implicit (to wit, solves a system of equations to ensure non-divergence of the barotropic flow). This tends to cost an extra factor of one grid side length in the run time, but admits a more realistic simulation, in that the effect of the ocean floor is included.

1.2 Governing Equations

The model solves the hydrostatic Boussinesq equations within a finite number of discrete isopycnal layers. At the layer interfaces there is a discrete jump in velocity and density, but not in pressure. The general forms of the equations are shown in Section 1.2.1 and Section 1.2.2, while a derivation of these equations for the 1.5 layer version of the model is shown in Section 1.2.3.

1.2.1 Continuity equation

$$\frac{\partial h_n}{\partial t} = \nabla \cdot (h_n \mathbf{v_n}) . \tag{1.1}$$

in which h_n is the thickness of layer n, and $\mathbf{v_n}$ represents the vertically averaged horizontal velocity in layer n.

1.2.2 Momentum equations

$$\frac{D\mathbf{v_n}}{Dt} + \mathbf{f} \times \mathbf{v_n} + g^{'}\nabla h_n = \mathbf{F_n}, \tag{1.2}$$

in which $g^{'}$ is the reduced gravity given by $g(\rho_2 - \rho_1)/\rho_1$. The reduced gravity is dynamically equivalent to gravity, but is scaled to take into account the density difference between the two layers.

This can be rewritten in terms of the Bernoulli Potential to give,

$$\frac{\partial \mathbf{v_n}}{\partial t} - (f + \zeta_n) \times v_n + \nabla \Pi_n + = \kappa \nabla^2 v_n + \frac{\mathbf{F_n}}{\rho_0} \tag{1.3}$$

where Π_n is the Bernoulli potential, $(\mathbf{v_n} \cdot \mathbf{v_n})/2 + p/\rho_0$, and p is the hydrostatic pressure. In this form the non-linearity from the material derivative has been moved into the Bernoulli Potential and the vorticity term.

The model can be used in either reduced gravity mode, with a quiescent abyss, or in n-layer mode with bathymetry. In the n-layer case the model can either be run with a rigid lid, or with a free surface. In n-layer simulations the following equation is also solved

$$\frac{\partial \eta}{\partial t} + \nabla \cdot (H\mathbf{V}) = 0, \tag{1.4}$$

where η is the free surface height, H is the depth from the free-surface to the bathymetry, and V is the vertically averaged flow, the barotropic flow. With a rigid lid η represents the pressure field required to keep the vertically integrated horizontal flow divergence free - the result is not carried from one timestep to the next.

1.2.3 Reduced gravity equations

The most idealised version of Aronnax is a 1.5 layer or 'reduced gravity' model. Reduced gravity models are often referred to as 1.5 layer models because of the limited dynamics in the lowest layer. The bottom layer is assumed to have no motion, which is mathematically equivalent to an infinite thickness. Since there is no motion in the lowest layer, the model is unable to support vertically homogeneous, or barotropic, motions. This means that the gravest mode supported by the model is the first baroclinic mode. This version of Aronnax also features a rigid lid to remove surface gravity waves from the model solutions. This allows for longer time steps in the numerical solver. Reduced gravity models are often used to model the dynamics of the upper ocean, with the interface between the layers taken as an approximation for the oceanic thermocline. A schematic of a 1.5 layer model is shown in Figure 1.1.

The equations for a model with one active layer above a quiescent abyss will be derived here. Extending the derivation to a model with n active layers requires substantially more algebra but gives little additional insight.

If we take the conservation of mass and recast it as conservation of mass in a layer of thickness h we get

$$\frac{D(\rho h)}{Dt} + \rho h \nabla \cdot \mathbf{V} = 0, \tag{1.5}$$

in which $\mathbf{V}$ represents the vertically averaged horizontal velocity in the layer, and ρh is the areal density. By expanding the material derivative, combining terms and assuming that ρ is constant within an isopycnal layer, (1.5) can be rewritten as

$$\frac{\partial h}{\partial t} = \nabla \cdot (h\mathbf{V}) . \tag{1.6}$$

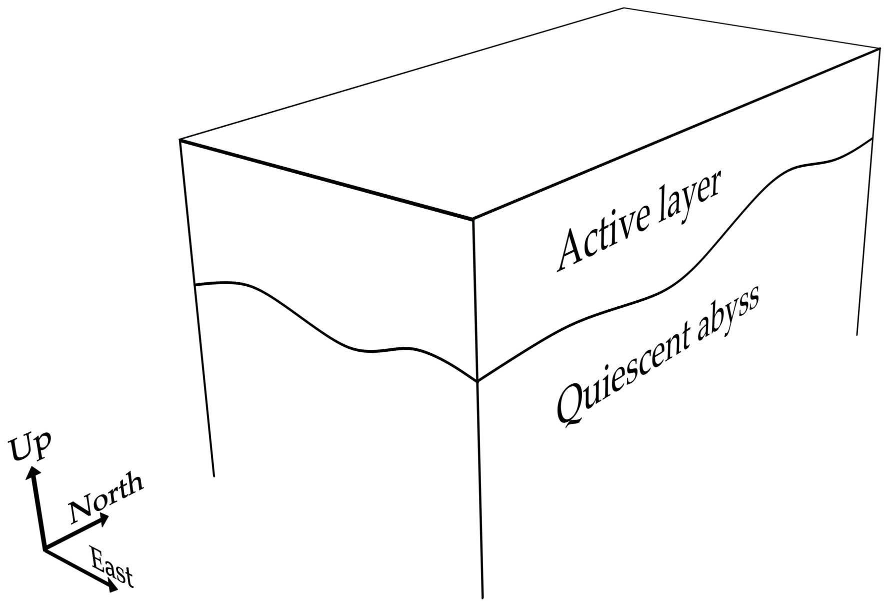


Figure 1.1: Schematic of a reduced gravity model in a rectangular domain. The fluid within the infinitely deep, quiescent abyss is assumed to be at rest.

With a rigid lid the surface is maintained at a constant height of zero, but the pressure is unconstrained. If we let the pressure at the rigid lid be given by $P(x, y, t)$, then the pressure in the upper layer at depth z is

$$P_1(x, y, z, t) = P(x, y, t) - \rho_1 g z, \tag{1.7}$$

where ρ_1 is the density of the upper layer, z is the vertical coordinate which becomes more negative with depth. A defining feature of reduced gravity models is the absence of motion in the lowest layer. This means that the horizontal pressure gradients in layer 2 are identically zero, which we can use to solve for the interface displacement. The pressure in layer 2 is given by

$$P_2(x, y, z, t) = P_1(x, y, h, t) - \rho_2 g(z + h) = P + \rho_1 g h + \rho_2 g(z + h), \tag{1.8}$$

where h is the thickness of the upper layer. Since a central assumption of the reduced gravity framework is that the horizontal gradients of P_2 are zero we can now solve for the horizontal pressure gradient in the upper layer. Taking the gradient of equation (1.8) and setting the left-hand side to zero gives

$$0 = \nabla P + \nabla \rho_1 g h + \nabla \rho_2 g(-h), \tag{1.9}$$

which can be rearranged to give

$$\nabla P = g(\rho_2 - \rho_1)\nabla h, \tag{1.10}$$

which relates the horizontal pressure gradients in the upper layer to displacements of the interface. The momentum equation for the upper layer is therefore

$$\frac{D\mathbf{V}}{Dt} + \mathbf{f} \times \mathbf{V} + g^{'}\nabla h = \mathbf{F}, \tag{1.11}$$

in which $g^{'}$ is the reduced gravity given by $g(\rho_2 - \rho_1)/\rho_1$. The reduced gravity is dynamically equivalent to gravity, but is scaled to take into account the density difference between the two layers.

1.3 Discretisation

Aronnax is discretised on an Arakawa C grid.

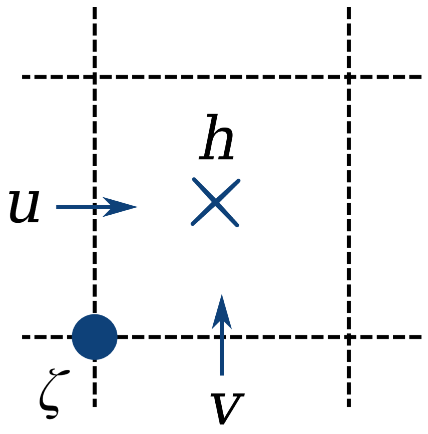


Figure 1.2: A single grid cell from an Arakawa C grid.

1.4 Numerical algorithm

The model solves for two horizontal velocity components and layer thickness in an arbitrary number of layers. The model supports two sets of physics: either a reduced gravity configuration in which the horizontal pressure gradient is set to zero in a quiescent abyss below the lowest active layer; or an n-layer configuration in which bathymetry must be specified.

Aronnax is discretised on an Arakawa C-grid, with the velocity and thickness variables in different locations on the grid cell.

The choice of quiescent abyss or n-layer physics is made by a runtime parameter in the input file. The numerical algorithm for calculating the values at the next time level, $n + 1$, is as follows:

- The Bernoulli Potential is calculated using values from time-level n
  - The function used depends on whether the model is running in reduced gravity mode or n-layer mode
- The relative vorticity is calculated using values from time-level n
- The layer thickness tendencies are calculated using the velocities and layer thicknesses from time-level n
- the velocity tendencies are calculated using values from time-level n
- the layer thicknesses and velocities are stepped forward in time to $n + 1$ using a third-order Adams-Bashforth algorithm and the stored time derivatives from the previous two timesteps. N.B. for the n-layer version these velocities are not strictly at time $n + 1$, let's call it time level $n + *$.
- For the n-layer version:
  - The no-normal flow boundary condition is applied (perhaps unnecessary?)
  - The barotropic velocity required to keep the vertically integrated flow non-divergent in the horizontal is calculated and added to the baroclinic velocities calculated previously. To do this:
    - the barotropic velocities are calculated from the velocities at time-level $n + *$.
    - the divergence of these velocities is used to solve for the free surface elevation at time-level $n + 1$ that makes the barotropic flow non-divergent
      - This is the step that requires the linear system solve, since we solve the equation implicitly to sidestep the issue of requiring a *very* short δt.
    - the barotropic correction is applied to the velocity fields

        * consistency between the sum of the layer thicknesses and the depth of the ocean is forced by applying a uniform inflation/deflation to the layers. (the model currently prints a warning if the discrepancy is larger than a configurable threshold, which defaults to 1%)

- The no normal flow and tangential (no-slip or free-slip) boundary conditions are applied
- The layer thicnkesses are forced to be larger than a configurable minimum. This is for numerical stability and is probably only necessary for the layer receiving the wind forcing. This is discussed in ticket #26
- the arrays are shuffled to prepare for the next timestep.

N.B. To get the Adams-Bashforth method going, two time steps are initially performed using Runge-Kutta 4th order time stepping.

CHAPTER 2

Installation

While we aspire for the installation process for Aronnax to be as simple as `pip install aronnax`, it is not yet that easy.

2.1 Dependencies

Aronnax is tested with Python 2.7 and 3.6, and depends on several external libraries.

The Python components of Aronnax depend on

```
numpy
scipy
```

We recommend installing these with your favourite package manager before installing Aronnax. While the previous dependencies will allow you to run the model, some of the functions for dealing with the output files also depend on

```
xarray
dask
```

Additionally, the Fortran core requires

```
make
gfortran >= 4.7.4
mpi
```

In particular, Aronnax will need the `mpif90` command in order for it to compile its Fortran core. You will need to manually ensure that you have a working installation of each of these.

In addition to these dependencies, the automated tests also depend on `pytest` and `matplotlib`.

2.2 Installation instructions

1. Clone the repository to a local directory
   - `git clone --recursive https://github.com/edoddridge/aronnax.git`
2. Move into the base directory of the repository
   - `cd aronnax`
3. Compile Hypre
   - move to the directory 'lib/hypre/src'
     - `cd lib/hypre/src`
   - configure the Hypre installer
     - `./configure`
   - compile Hypre. This will take a few minutes
     - `make install`
   - move back to root directory of the repository
     - `cd ../../../`
4. install Aronnax
   - `pip install -e ./`

Aronnax is now installed and ready to use. To verify that everything is working, you may wish to run the test suite. Do this by executing `pytest` in the base directory of the repository. This requires that the `pytest` module is installed.

Note: Installing in HPC environments: If your cluster requires programs to be compiled on the compute cores, then you will need to perform step 3 on the compute cores.

CHAPTER 3

Input generation

There are a number of helpers designed to ease the creation of input fields.

3.1 Grid object

The grid object contains the axes for variables on tracer points, both velocity points, and vorticity points. This allows users to create their inputs in a resolution independent manner by specifying functions based on physical location, rather than grid point number. Changing resolution therefore becomes a trivial exercise that requires changing only the *nx*, *ny*, *dx*, and *dy* parameters.

class `aronnax.`**`Grid`**(*nx*, *ny*, *layers*, *dx*, *dy*, *x0=0*, *y0=0*)

Make a grid object containing all of the axes.

Parameters

- **nx** (*int*) – Number of grid points in the x direction
- **ny** (*int*) – Number of grid points in the y direction
- **layers** (*int*) – Number of active layers
- **dx** (*float*) – Grid size in x direction in metres
- **dy** (*float*) – Grid size in y direction in metres
- **x0** (*float*) – x value at lower left corner of domain
- **y0** (*float*) – y value at lower left corner of domain

The initialisation call returns an object containing each of the input parameters as well as the following arrays:

- x: x locations of the tracer points
- y: y locations of the tracer points
- xp1: x locations of the u velocity points and vorticity points
- yp1: y locations of the v velocity points and vorticity points

3.2 Inputs

3.2.1 Pre-existing Fortran files

Fortran unformatted files (often called Fortran binary files) can be used as input. To do this, they should be placed in the 'input' folder, and the name of the file given either in the *aronnax.conf* file, or passed as a keyword argument in the call to *aronnax.driver.simulate*.

The fields need to be the correct shape and size. If they aren't the error messages may be difficult to comprehend or nonexistent depending on whether the field is too big, too small, or the wrong shape. The parameter *DumpWind* can be helpful for ensuring that the wind stress has been set correctly.

3.2.2 Generator functions

The use of input generator functions allows the model to be passed user defined functions describing the inputs. Aronnax will evaluate the user defined functions or constants to create the input fields, and save these fields to the 'input' folder in the format required by the Fortran core.

The generator functions can be called in two ways:

- either directly from the aronnax.conf file, using the syntax *:generator_name:arg1,arg2,. . . ,argn*. In this case the arguments must be numerical (generating very simple layerwise constant value fields).
- or they can be included in the call to `aronnax.driver.simulate()` using the syntax *field_name=[arg1, arg2, . . . , argn]*. In this case the arguments may be either numerical or functions and must be passed as a list. That is, they must be wrapped in square brackets [], even if that list has only one element. Aronnax will check that the length of the list equals the number of layers the field is expected to have and will throw an error if they are not equal.

Regardless of the method used, each argument is evaluated for a single layer. That is, *arg1* is used to create the field for the upper layer.

For use in *aronnax.driver.simulate* call

When using generator functions to create the inputs through the call to `aronnax.driver.simulate()` it is possible to use Python functions to create the fields. This is an extremely powerful method for creating arbitrary forcings and initial conditions.

If a function is passed to the generator functions it must depend on:

- *X* and *Y*, created from a *np.meshgrid* call with the appropriate axis, if it is for a 2D field; or
- *nTimeSteps* and *dt* if it is for a time series.

All other values used within the function must be set in a namespace that the function has access to, or be hard-coded.

For use in *aronnax.conf* file

These generic generator functions can be used in the *aronnax.conf* file to create simple inputs or initial conditions using the syntax *:generator_name:arg1,arg2,. . . ,argn*. The arguments must be numeric.

For specific grid locations

These generator functions are passed one numeric argument per layer.

`aronnax.`**`tracer_point_variable`**(*grid*, *field_layers*, **funcs*)
: Input generator for a variable at the tracer location of the grid. If passed a function, then that function can depend only on X and Y.

`aronnax.`**`u_point_variable`**(*grid*, *field_layers*, **funcs*)
: Input generator for a variable at the u location of the grid. If passed a function, then that function can depend only on X and Y.

`aronnax.`**`v_point_variable`**(*grid*, *field_layers*, **funcs*)
: Input generator for a variable at the v location of the grid. If passed a function, then that function can depend only on X and Y.

`aronnax.`**`time_series_variable`**(*nTimeSteps*, *dt*, *func*)
: Input generator for a time series variable. If passed a function, then that function can depend on the number of timesteps, *nTimeSteps*, and the timestep, *dt*.

Coriolis fields

The f-plane generator functions are passed one numeric argument, the value for f, and the β-plane functions are passed two numeric arguments.

`aronnax.`**`f_plane_f_u`**(*grid*, *field_layers*, *coeff*)
: Define an f-plane approximation to the Coriolis force (u location).

`aronnax.`**`f_plane_f_v`**(*grid*, *field_layers*, *coeff*)
: Define an f-plane approximation to the Coriolis force (v location).

`aronnax.`**`beta_plane_f_u`**(*grid*, *field_layers*, *f0*, *beta*)
: Define a beta-plane approximation to the Coriolis force (u location).

`aronnax.`**`beta_plane_f_v`**(*grid*, *field_layers*, *f0*, *beta*)
: Define a beta-plane approximation to the Coriolis force (v location).

Domain shape

This generator function is passed without arguments in the *aronnax.conf* file.

`aronnax.`**`rectangular_pool`**(*grid*, *field_layers*)
: The wet mask file for a maximal rectangular pool.

CHAPTER 4

Running Aronnax

To run a simulation with Aronnax one needs to have a Python session active in the folder for the simulation. This folder should contain a file, *aronnax.conf* that contains the configuration choices for the simulation. Some, or all, of these choices may be overridden by arguments passed to the simulate function, but it is likely simpler to specify many of the choices in a configuration file.

`aronnax.driver.`**`simulate`**(*work_dir='.'*, *config_path='aronnax.conf'*, ***options*)

Main entry point for running an Aronnax simulation.

A simulation occurs in the working directory given by the *work_dir* parameter, which defaults to the current directory when *simulate* is invoked. The default arrangement of the working directory is as follows:

- aronnax.conf - configuration file for that run
- aronnax-merged.conf - file to save effective configuration, including effects of options passed to *simulate*. This file is automatically generated by merging the aronnax.conf file with the options passed to this function
- parameters.in - relevant portions of aronnax-merged.conf in Fortran namelist format. Also generated automatically
- input/ - subdirectory where Aronnax will save input field files in Fortran raw array format
- output/ - subdirectory where Aronnax will save output field files in Fortran raw array format

The process for a simulation is to

1. Compute the configuration
2. Recompile the Fortran core if necessary
3. Save the computed configuration in aronnax-merged.conf
4. Write raw-format input fields into input/
5. Write parameters.in
6. Execute the Fortran core, which writes progress messages to standard output and raw-format output fields into output/

All the simulation parameters can be controlled from the configuration file aronnax.conf, and additionally can be overridden by passing them as optional arguments to *simulate*.

Calling *simulate* directly provides one capability that cannot be accessed from the configuration file: custom idealized input generators.

As described above, it is possible to define functions that can be passed to *aronnax.driver.simulate* and used to create input or forcing fields. The test suite, found in the 'test' folder uses this functionality to create the zonal wind stress for the β-plane gyre tests. The relevant code is shown below:

```
def wind(_, Y):
    return 0.05 * (1 - np.cos(2*np.pi * Y/np.max(grid.y)))

with working_directory(p.join(self_path, "beta_plane_gyre_red_grav")):
    drv.simulate(zonalWindFile=[wind],
                 nx=10, ny=10, exe="aronnax_test", dx=xlen/10, dy=ylen/10)
```

Warning: Parameters cannot be set to 0 or 1 in the call to *drv.simulate* because the Python wrapper gets confused between numerical and logical variables, as described in https://github.com/edoddridge/aronnax/issues/132

4.1 Executables

Aronnax includes multiple targets for the Makefile. These produce executables intended either for testing or production runs. The different options for *exe* are discussed below. For a comparison on the execution speed of these executables see *Benchmarking*.

For production runs, and simulations that span multiple processors, use either *aronnax_core* (for $n + 1/2$-layer mode) or *aronnax_external_solver* (for n-layer mode).

- *aronnax_core*
  - Uses the internal Fortran code and aggressive compiler optimisation. This is the best executable to use for reduced gravity ($n+1/2$ layer) simulations. It may also be used for n layer simulations but is considerably slower than *aronnax_external_solver* since it does not use the external Hypre library for the pressure solve. *aronnax_core* cannot run on multiple processors in n-layer mode.
- *aronnax_test*
  - Uses the internal Fortran code and no compiler optimisations. This is the best executable to use for assessing code coverage when running tests. The Fortran error messages might be helpful.
- *aronnax_prof*
  - Executable intended solely for profiling the Fortran core. Unlikely to be of general use.
- *aronnax_external_solver_test*
  - Uses the external Hypre library to solve the matrix inversion in the n layer mode. Uses no optimisations for the internal Fortran code and is intended for assessing code coverage when running tests.
- *aronnax_external_solver*
  - Uses the external Hypre library and aggressive compiler optimisations. This is the fastest executable to use in n layer mode. It can also be used in $n + 1/2$ layer mode, but there is no advantage over *aronnax_core*.

4.2 Parameters

Parameters can be passed to the model in two ways. Either they can be included in a file called *aronnax.conf* in the working directory, or they can be passed as keyword arguments to `aronnax.driver.simulate()`. The main directory of the repository includes an example *aronnax.conf* file.

Note: Aronnax takes a deliberately strict approach towards specifying parameter choices. The model contains very few default values, except in situations where a default of zero can sensibly be applied. As such, you will need to specify your parameter choices, either through the configuration file, or the function call. However, parameters that are not required, for example, bottom drag in n+1/2 layer mode, need not be set.

The example file is reproduced here with comments describing the parameters and their units. All possible parameters are included, but they are not all assigned values. After modifying this file for your simulation, any unassigned parameters should be deleted.

```
# Aronnax configuration file. Change the values, but not the names.

#------------------------------------------------------------------------------
# au is the lateral friction coefficient in m^2 / s
# ar is linear drag between layers in 1/s
# kh is thickness diffusivity in m^2 / s
# kv is vertical thickness diffusivity in m^2/s
# dt is time step in seconds
# slip is free-slip (=0), no-slip (=1), or partial slip (something in between)
# niter0: the timestep at which the simulation begins. If not zero, then there
#   must be checkpoint files in the 'checkpoints' directory.
# nTimeSteps: number of timesteps before stopping
# dumpFreq: time between snapshot outputs in seconds
# avFreq: time between averaged output in seconds
# checkpiontFreq: time between checkpoints in seconds
#   (these are used for restarting simulations)
# diagFreq: time between dumping layerwise diagnostics of the simulation. These
#   are mean, min, max, and std of h, u, and v in each layer.
# hmin: minimum layer thickness allowed by model (for stability) in metres
# maxits: maximum iterations for the pressure solver algorithm. Should probably
#   be at least max(nx,ny), and possibly nx*ny
# eps: convergence tolerance for pressure solver. Algorithm stops when error is
#   less than eps*initial_error
# freesurfFac: 1. = linear implicit free surface, 0. = rigid lid.
# botDrag is the linear bottom friction in 1/s
# thickness_error is the  discrepancy between the summed layer thicknesses and
#   the depth above which the model emits a warning. 1e-2 is a 1% discrepancy.
# debug_level controls the level of output produced by the model. When set to
#   zero, or not specified (it defaults to zero), the model outputs h, u, and v
#   at the frequency controlled by dumpFreq and avFreq. Specifying larger
#   integer values results in progressively more output more frequently. See
#   the documentation for details.
# hAdvecScheme selects which advection scheme to use when advecting the
#   thickness field. Current options are:
#   1 first-order centered differencing
#   2 first-order upwind differencing
# TS_algorithm selects the time stepping algorithm used by the model
#   3 (default) is third-order Adams-Bashfort

[numerics]
```

(continues on next page)

(continued from previous page)

```
au = 500.
ar = 0.0
kh = 0.0
kv = 0.0
dt = 600.
slip = 0.0
niter0 = 0
nTimeSteps = 502
dumpFreq = 1.2e5
avFreq = 1.2e5
checkpointFreq = 1.2e5
diagFreq = 6e3
hmin = 100
maxits = 1000
eps = 1e-5
freesurfFac = 0.
botDrag = 1e-6
thickness_error = 1e-2
debug_level = 0
hAdvecScheme = 2
TS_algorithm = 3
#------------------------------------------------------------------------------

# RedGrav selects whether to use n+1/2 layer physics (RedGrav=yes), or n-layer
#   physics with an ocean floor (RedGrav=no)
# depthFile defines the depth of the ocean bottom below the sea surface in metres.
# hmean is a list of initial thicknesses for the layers in metres. Each value is
#   separated by a comma. This input was a useful short cut for specifying
#   initial conditions with constant layer thicknesses, but has been superseded
#   and may be removed in the future.
# H0 is the depth of the ocean basin and is only required in n-layer mode. This
#   input was a useful short cut for specifying a flat bottomed ocean, but has
#   been superseded and may be removed in the future.

[model]
RedGrav = no
depthFile
hmean = 400.,1600.
H0 = 2000.
#------------------------------------------------------------------------------

# these variables set the number of processors to use in each direction.
#   You should ensure that the grid divides evenly into the number of tiles.
#   nx/nProcX and ny/nProcY must be integers.

[pressure_solver]
nProcX = 1
nProcY = 1
#------------------------------------------------------------------------------

# g_vec is the reduced gravity at interfaces in m/s^2. g_vec must have as many
#   entries as there are layers. The values are given by the delta_rho*g/rho_0.
#   In n-layer mode the first entry applies to the surface, i.e. the top of the
#   upper layer. In n+1/2 layer mode the first entry applies to the bottom of
#   the upper layer.
# rho0 is the reference density in kg/m^3, as required by the Boussinesq assumption.
```

(continues on next page)

(continued from previous page)

```
[physics]
g_vec = 9.8, 0.01
rho0 = 1035.
#------------------------------------------------------------------------------

# nx is the number of grid points in the x direction
# ny is the number of grid points in the y direction
# layers is the number of active layers
# OL is the width of the halo region around tiles (must be set to 1)
# dx is the x grid spacing in metres
# dy is the y grid spacing in metres
# fUfile defines the Coriolis parameter on the u grid points in 1/s
# fVfile defines the Coriolis parameter on the v grid points in 1/s
# wetMaskFile defines the computational domain - which grid points are ocean,
#   with wetmask=1, and which are land, with wetmask=0. The wetmask is defined
#   at the centre of each grid cell, the same location as thickness.

[grid]
nx = 10
ny = 10
layers = 2
OL = 1
dx = 2e4
dy = 2e4
fUfile = :beta_plane_f_u:1e-5,2e-11
fVfile = :beta_plane_f_v:1e-5,2e-11
wetMaskFile = :rectangular_pool:
#------------------------------------------------------------------------------

# These files define the values towards which the model variables are relaxed
#   (in metres or m/s), and the timescale for the relaxation, in 1/s.
[sponge]
spongeHTimeScaleFile
spongeUTimeScaleFile
spongeVTimeScaleFile
spongeHFile
spongeUfile
spongeVfile

#------------------------------------------------------------------------------

# These files define the initial values used in the simulation. If no values are
#   defined for the velocities (in m/s) or the free surface elevation (in m),
#   they will be initialised with zeros. Layer thickness (in m) must be initialised,
#   either by passing a file, or using the generator functions.

[initial_conditions]
initUfile
initVfile
initHfile
initEtaFile

#------------------------------------------------------------------------------

# The wind files define the momentum forcing in N/m^2 or m/s
# wind_mag_time_series_file defines the constant factor by which the wind is
#   multiplied by at each timestep.
```

(continues on next page)

(continued from previous page)

```
# wind_depth specifies the depth over which the wind forcing is spread. This
#   only impacts the simulation if the surface layer is thinner than wind_depth.
#   If wind_depth is set to 0 (default) then the wind forcing acts only on the
#   surface layer, no matter how thin it gets.
# DumpWind defines whether the model outputs the wind field when it outputs other
#   variables (at the rate controlled by DumpFreq).
# RelativeWind selects whether the wind forcing is given in
#   N/m^2 (RelativeWind = no), or
#   m/s   (RelativeWind = yes)
# Cd is the quadratic drag coefficient used if RelativeWind = yes

[external_forcing]
zonalWindFile = 'wind_x.bin'
meridionalWindFile = 'wind_y.bin'
wind_mag_time_series_file
wind_depth = 30
DumpWind = no
RelativeWind = no
Cd = 0.

#------------------------------------------------------------------------------
```

Warning: The configuration file shown above includes all of the possible input parameters and fields since it forms part of the documentation. IT WILL NOT WORK AS IS. To use it in an actual simulation the file will need to be modified either by giving values to the parameters that are currently unspecified, or deleting them from the file. If you wish to see a configuration file that corresponds to a successful simulation, look in any of the test, benchmark, or reproduction directories.

4.2.1 debug_level

This parameter determines whether the model produces additional outputs. It should be set to an integer value greater than or equal to zero. The different values have the following effects:

- 0: no additional outputs. Output frequency controlled by *DumpFreq* and *AvFreq*
- 1: output tendencies at frequency given by *DumpFreq*
- 2: output tendencies and convergence diagnostics from the linear solve at frequency given by *DumpFreq* (not implemented)
- 3: output convergence diagnostics and tendencies before and after applying some or all of sponges, barotropic correction, winds, and boundary conditions at frequency controlled by *DumpFreq* (not implemented)
- 4: dump all of the above fields every time step (mostly implemented)
- 5: dump everything every time step including the two initial RK4 steps (not implemented)

4.2.2 niter0

This parameter allows a simulation to be restarted from the given timestep. It requires that the appropriate files are in the 'checkpoints' directory. All parameters, except for the number of grid points in the domain, may be altered when restarting a simulation. This is intended for breaking long simulations into shorter, more manageable chunks, and for running perturbation experiments.

4.2.3 wetMaskFile

The wetmask defines which grid points within the computational domain contain fluid. The wetmask is defined on the tracer points, and a value of 1 defines fluid, while a value of 0 defines land. The domain is doubly periodic in x and y by default. To produce a closed domain the wetmaks should be set to 0 along the edges of the domain. To close the domain it is sufficient to place a strip of land along either the northern or southern boundary and either the western or eastern boundary. You may find it conceptually easier to close both edges.

4.2.4 RelativeWind

If this is false, then the wind input file is given in N m^{-2}. If true, then the wind input file is in m s^{-1} and a quadratic drag law is used to accelerate the fluid with Cd as the quadratic drag coefficient.

4.2.5 hAdvecScheme

hAdvecScheme is an integer that selects the advection scheme used for thickness in the continuity equation. Currently two options are implemented:

- *hAdvecScheme* = 1 (default) uses a first-order centered stencil
- *hAdvecScheme* = 2 uses a first-order upwind stencil

4.2.6 TS_algorithm

TS_algorithm is an integer that selects the timestepping algorithm used by the model. The default behaviour is to use a third-order Adams-Bashforth scheme (*TS_algorithm* = 3), with the initialisation performed by a second-order Runge-Kutta method (*TS_algorithm* = 12).

- TS_algorithm = 1: Forward Euler
- TS_algorithm = 2: Second-order Adams-Bashfort
- TS_algorithm = 3: Third-order Adams-Bashfort (default)
- TS_algorithm = 4: Fourth-order Adams-Bashfort
- TS_algorithm = 5: Fifth-order Adams-Bashfort
- TS_algorithm = 12: Second-order Runge-Kutta
- TS_algorithm = 13: Third-order Runge-Kutta (not implemented)
- TS_algorithm = 14: Fourth-order Runge-Kutta (not implemented)

4.3 Discussion

The following points may be helpful when using the model but don't fit above.

4.3.1 Outcropping

Aronnax allows for layers to become very thin, which simulates outcropping of isopycnals at the surface, and grounding of isopycnals at the sea floor. Unfortunately outcropping adversely impacts layerwise volume conservation. Over a long simulation the change in volume of a layer may be substantial.

To allow for outcropping the following parameters should be set:

- *hmin* very small
- *wind_depth* set to >10 m

CHAPTER 5

Output

Aronnax produces output in two formats. The full fields are saved in Fortran unformatted files, that are compact and efficient, but not particularly user friendly. Layerwise statistics are saved into one csv file per variable.

5.1 Reading the data

Aronnax includes a number of helper functions for dealing with the unformatted Fortran output.

The simplest of these is *interpret_raw_file* which loads a single output of a single variable into a numpy array.

`aronnax.`**`interpret_raw_file`**(*name*, *nx*, *ny*, *layers*)

Read an output file dumped by the Aronnax core.

Each such file contains one array, whose size depends on what, exactly, is in it, and on the resolution of the simulation. Hence, the parameters nx, ny, and layers, as well as the file naming convetion, suffice to interpret the content (assuming it was generated on the same system).

Aronnax also ships with a function for lazily loading multiple timestamps of a single variable. This function uses xarray and dask to create labeled n-dimensional arrays.

`aronnax.`**`open_mfdataarray`**(*files*, *grid*)

Open a number of output files into an xarray dataarray. All files must contain the same variable.

Uses dask.delayed to lazily load data as required.

Parameters

- **`files`** (*`list`*) – a list of file names to load
- **`grid`** – a grid object created by aronnax.Grid

Returns xarray.DataArray of the desired variable

CHAPTER 6

Examples

In the following sections we present some simplified examples and use Aronnax to reproduce a number of published results to show how Aronnax can be easily configured for a range of idealised modelling studies.

6.1 Canonical examples

These simulations use simple domains and and inputs. They highlight some of the features available in Aronnax and can be found in the *examples/* folder. To run these simulations locally change into the *examples/* directory and execute *python run_examples.py* in the terminal.

6.1.1 Gaussian bump on a β-plane

This example is initialised with a Gaussian bump in the layer thickness field. The momentum forcing is set to zero. The initial thickness of the upper layer for both examples is shown in Figure 6.1.

1 + 1/2 layers

2 layers

The upper layer of a two-layer simulation with a flat bottom. This looks very similar to the 1.5 layer simulation.

6.1.2 Twin gyre on a β-plane

These two examples begin with closed rectangular domains that are initially at rest. The momentum forcing shown in Figure 6.4 is applied to the upper layer in both examples. Due to it's computational complexity the n layer configuration takes substantially longer to run than the $n + 1/2$ layer simulation.

The twin gyre simulations are run on a 10 km resolution β-plane with Coriolis parameters equivalent to the midlatitudes of the Northern Hemisphere. At these latitudes 10 km is an eddy resolving resolution, and we expect to see inertial recirculations and internal variability develop as the simulations spin up.

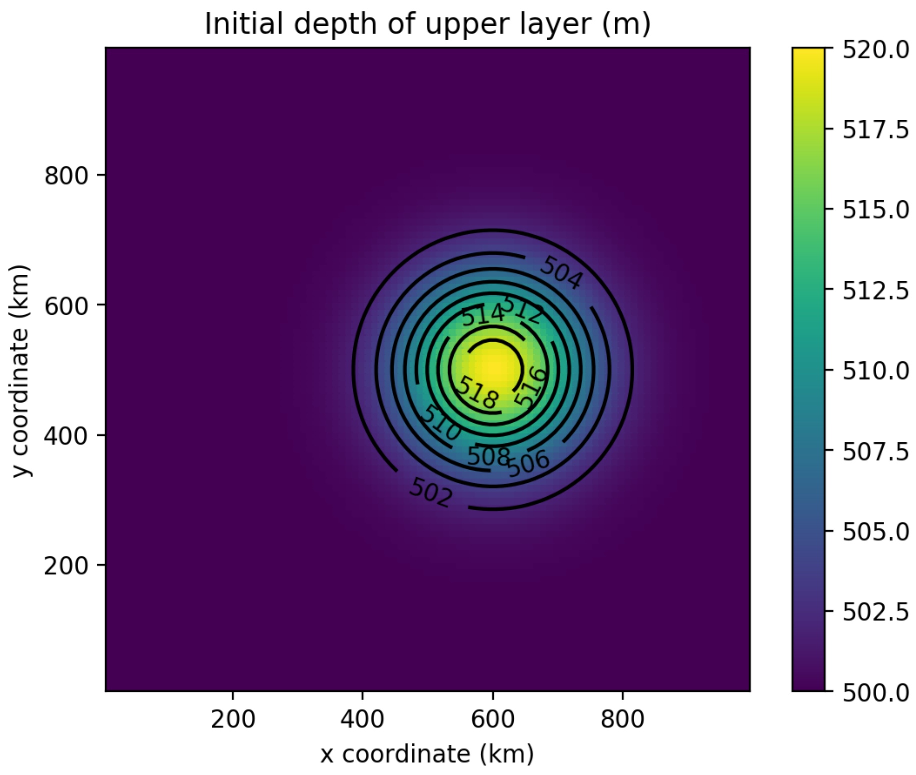


Figure 6.1: Initial thickness of the upper layer for the Gaussian bump examples

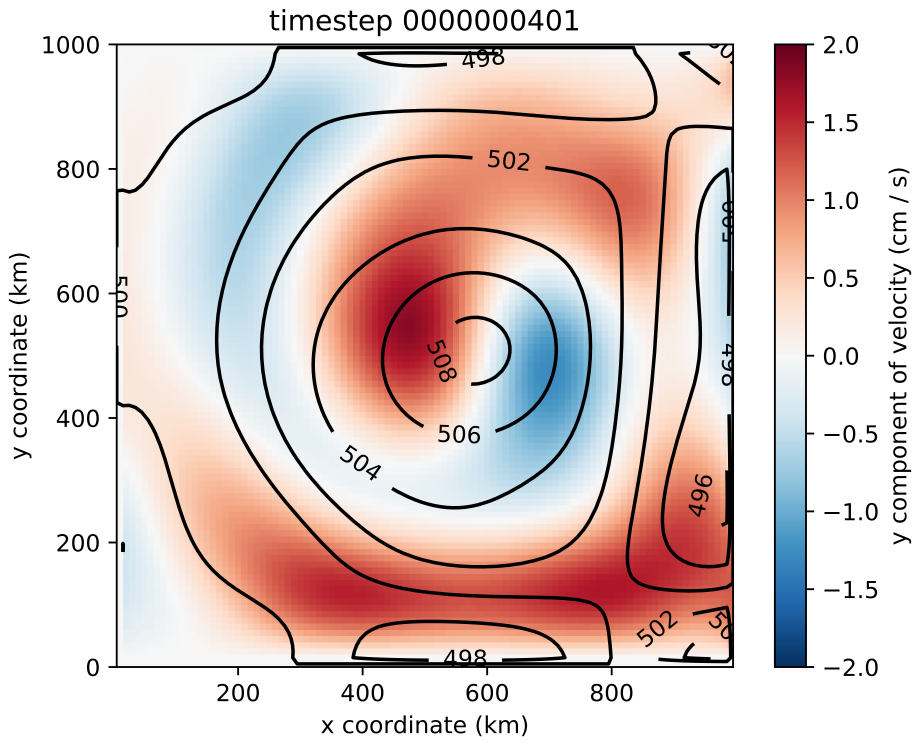


Figure 6.2: Evolution of a Gaussian bump on a 1.5 layer β-plane.

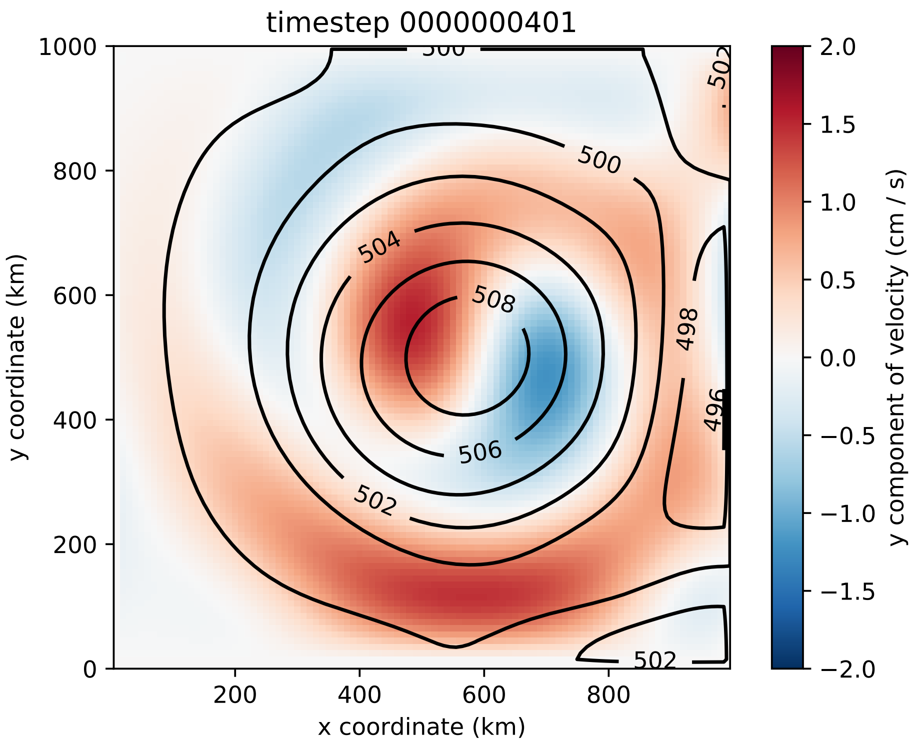


Figure 6.3: Evolution of a Gaussian bump on a 2 layer β-plane.

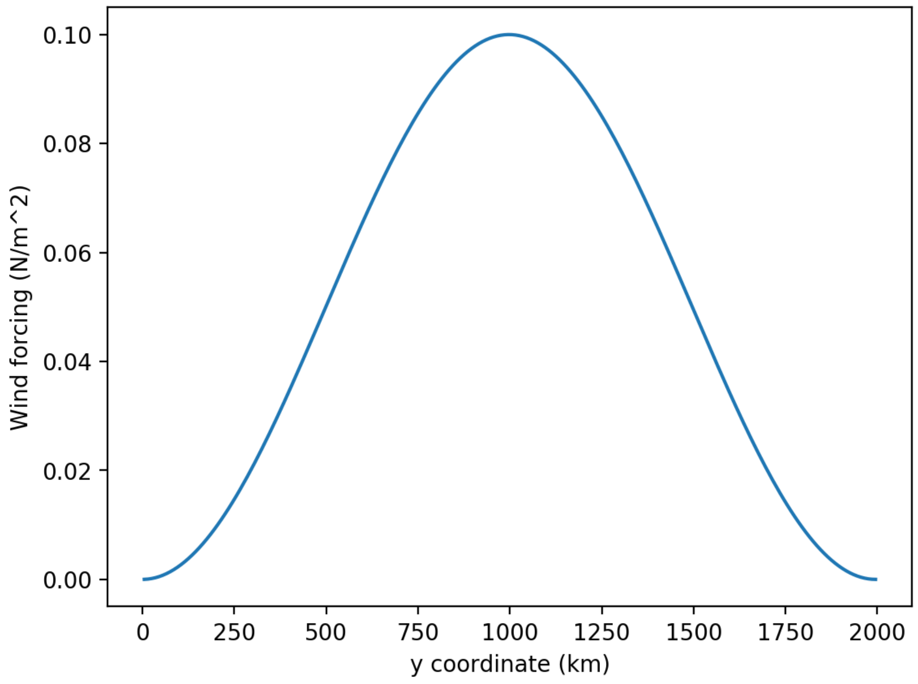


Figure 6.4: Zonal wind forcing applied to twin gyre simulations.

1 + 1/2 layers

This example simulates a twin-gyre on a β-plane with 1 active layer above a quiescent abyss. This simulation runs for almost 140 days of model time, and clearly shows the development of the two gyres and inertial recirculations at the inter-gyre boundary.

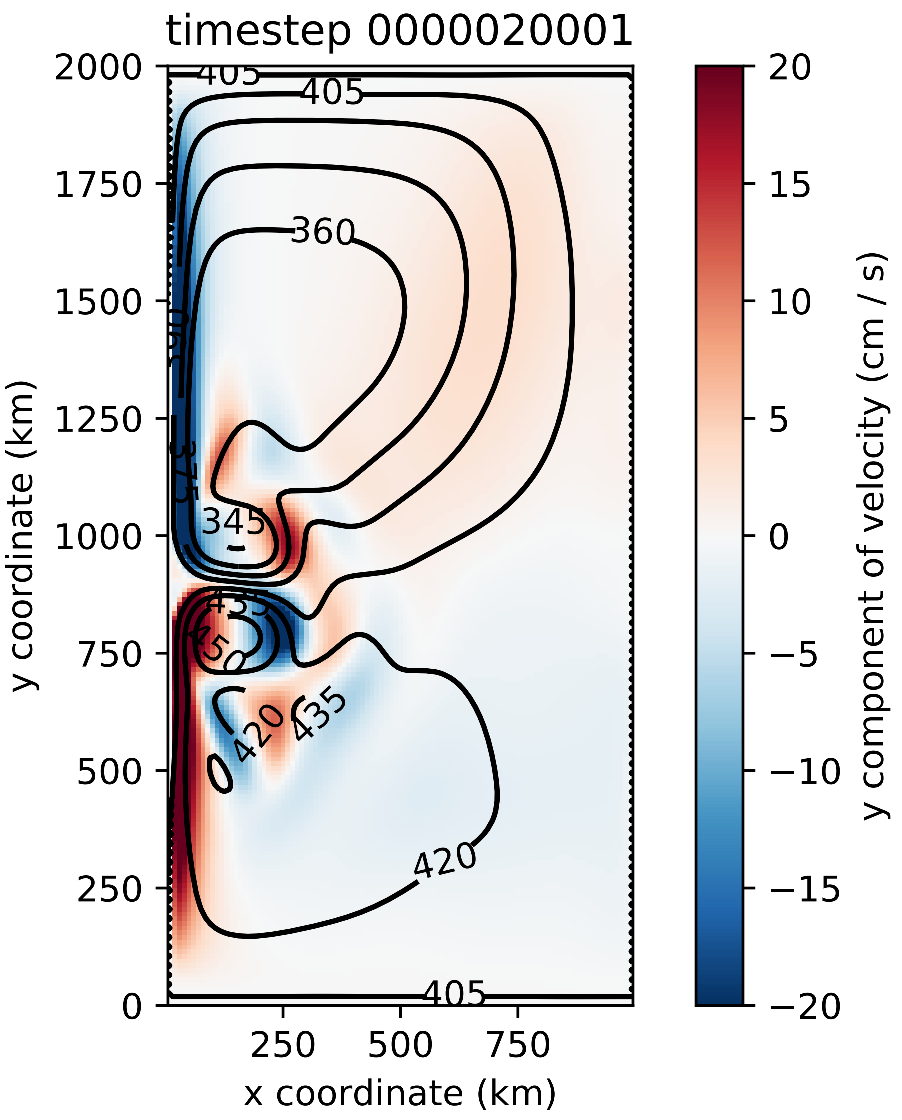


Figure 6.5: Evolution of a twin-gyre on a 1.5 layer β-plane.

2 layers

This is also a twin-gyre simulation, but is run with n layer physics and a flat bottom. Once again the simulation runs for almost 140 model days and clearly shows the development of two gyres, western boundary currents, and inertial recirculation regions.

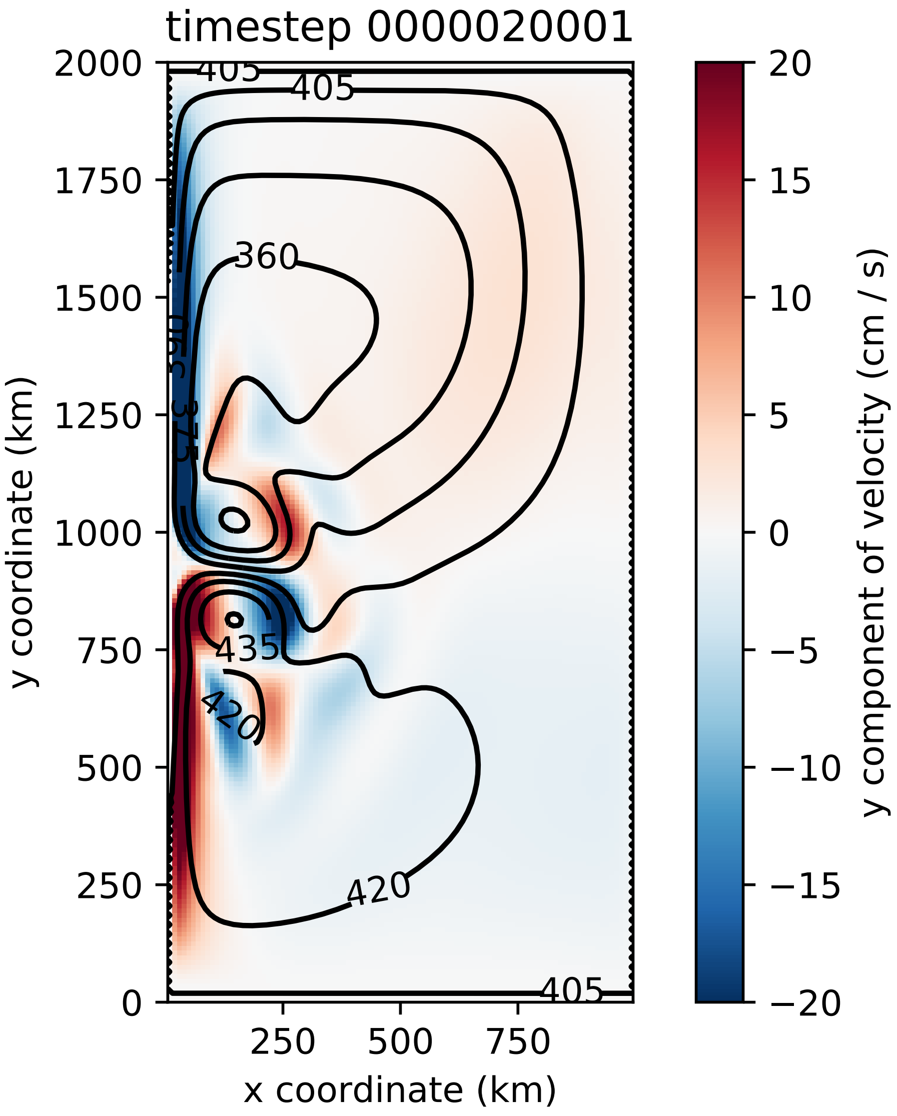


Figure 6.6: Evolution of a twin-gyre on a 2 layer β-plane.

6.2 Reproducing published results

These examples show how Aronnax can be used to reproduce results from the literature.

6.2.1 Davis et al. (2014) - An idealised Beaufort Gyre

Davis et al. (2014) used a reduced gravity model to explore the response of an idealised Beaufort Gyre to changes in the seasonal cycle of wind stress. Here we reproduce their control simulation. The domain is set up as a lollipop, with a circular basin for the Beaufort Gyre and a narrow channel connecting it to a region with sponges.

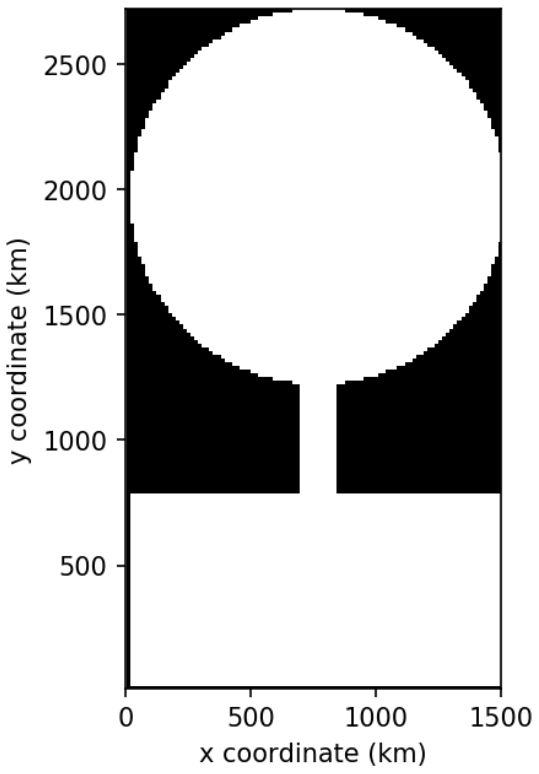


Figure 6.7: The computational domain for Davis et al. (2014). Note: the domain is symmetric, it is the plotting command that makes it look asymmetric.

Over this lollipop basin a wind stress is used to drive an anticyclonic circulation. The magnitude of the wind stress is given as

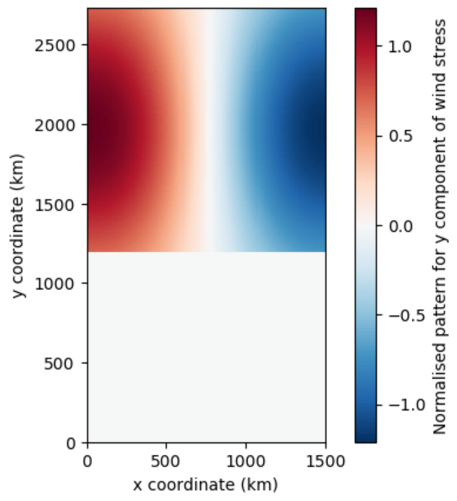
$$\frac{1}{r}\int r\cos^2(r)dr \tag{6.1}$$

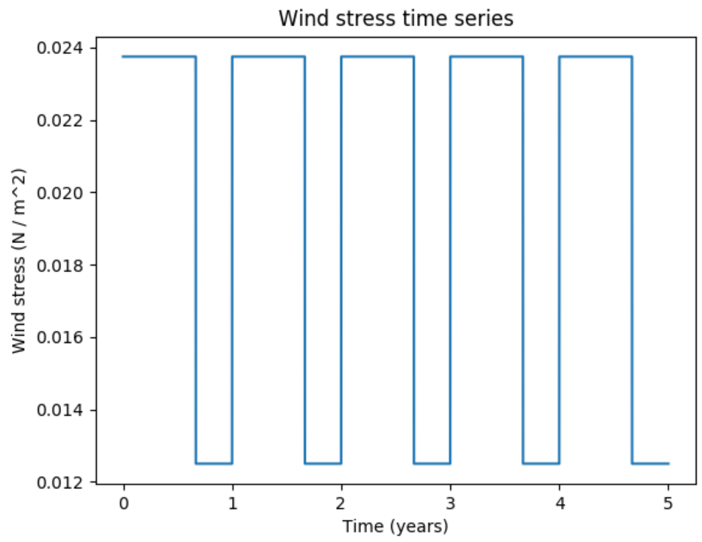
which is multiplied by $\sin(\theta)$ or $-\cos(\theta)$ to give the x and y components of the wind stress. Converting the integral into the wind stress requires evaluating $1/r$ times the integral as

$$\frac{1}{r}\left(\frac{r\sin(2r)}{4} - \frac{\sin^2(r)}{4} + \frac{r^2}{4}\right) \tag{6.2}$$

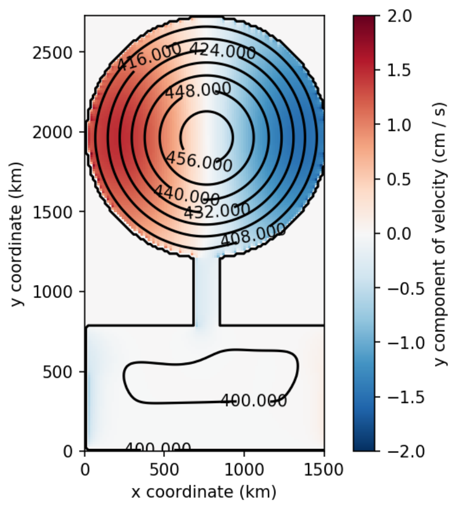
and normalising the result such that the average wind stress magnitude inside the circular domain is equal to one. This normalised wind stress is then converted into its x and y components.

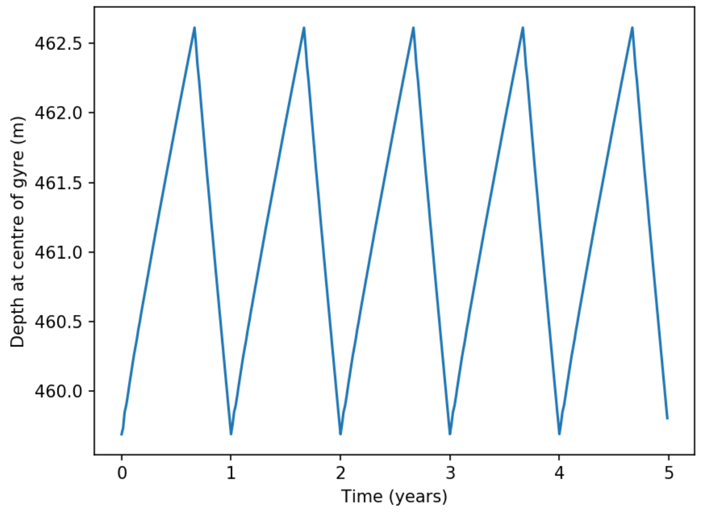
The y component of the normalised wind stress field is shown on the left, and the time series of wind stress magnitude is on the right.





After integrating for 40 model years these inputs produce a steady seasonal cycle in velocity and layer thickness. A snap shot is shown on the left, while a time series of the maximum thickness is shown on the right.





The seasonal cycle in layer thickness requires a time varying transport through the channel. This is shown below.

The paper includes multiple experiments perturbing the seasonal cycle of wind stress. Reproducing the perturbation experiments would require modifying the input variable *wind_mag_time_series_file*.

Note: The configuration used to create these outputs can be found in the reproductions folder of the repository.

6.2.2 Yang et al. (2016)

Yang et al. (2016) run and analyse a series of idealised n-layer models of the Beaufort Gyre in the Arctic. A configuration of Aronnax that mimics their setup can be found in the reproductions folder.

The spin up of the model proceeds as expected, but baroclinic instability sets in sooner than observed in the simulations by Yang et al. (2016). For this reason, the deepening of the upper layer is halted earlier, and it does not become as thick as in their simulations. There are multiple numerical differences between the model described by Yang et al. and Aronnax. For example, they use biharmonic viscosity and a quadratic bottom drag, while Aronnax has Laplacian viscosity and linear bottom drag. Numerical details can make a substantial difference to the output of numerical

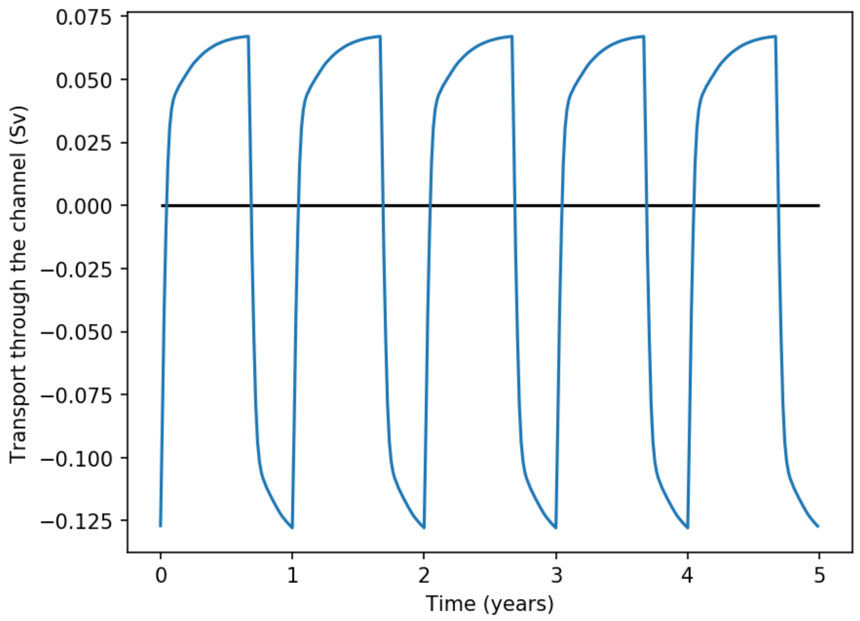


Figure 6.8: Time series of transport through the channel due to the seasonal cycle in wind stress.

simulations, and these differences may be why Aronnax fails to reproduce the mean state of their model. The model state after 14 years of simulation is shown in Figure 6.9. Comparing Figure 6.9 with their Figure 3 highlights that the upper layer in the Aronnax simulation is thinner.

An animation and interactive plot can be found in the online version of this documentation.

Note: The configuration used to run this simulation can be found in the *reproductions/Yang_et_al_2016* folder.

6.2.3 Manucharyan and Spall (2016)

n-layer configuration looking at eddies in the Arctic. (The original experiment was run using a z-level model, but it could also be done in an isopycnal model)

6.2.4 Johnson and Marshall (2002)

Reduced gravity analysis of the adjustment of the MOC to changes in deep water formation rates.

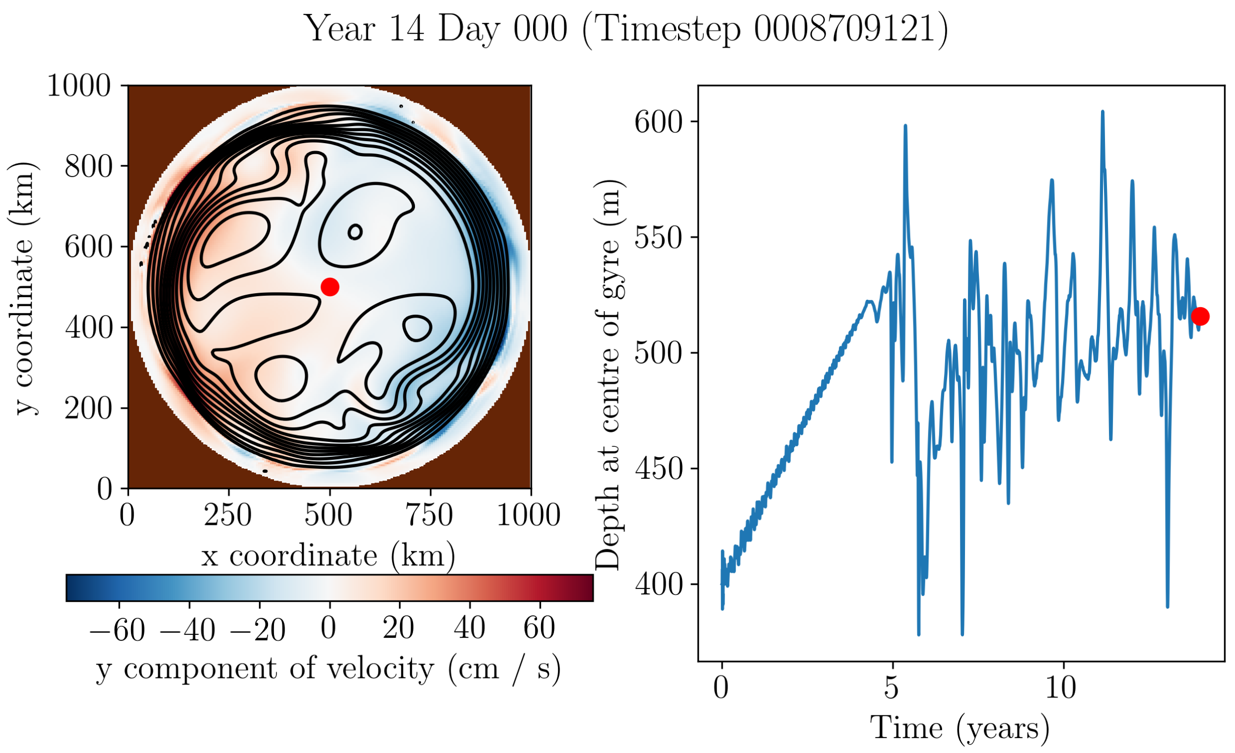


Figure 6.9: Left: Upper layer y velocity (colours) and thickness (contours). Right: Time series of thickness at centre of gyre.

CHAPTER 7

Verification

Aronnax includes a number of diagnostic test to verify that the numerical core is satisfactorily solving the equations of motion.

7.1 Conservation of volume

The test suite checks for volume conservation on each of the test simulations. The extent to which volume is conserved depends on the numerical accuracy of your simulation: large timesteps and coarse grids will reduce the accuracy of the simulation, and hence may affect volume conservation. Additionally, the use of sponge regions can affect both global and layerwise volume conservation, depending on the configuration. Table 7.1 shows when global or layerwise volume conservation can be expected.

Table 7.1: When to expect volume conservation

Physics	Sponge regions?	Volume conservation	
		Global	Layerwise
n-layer	Yes	Yes	No
n-layer	No	Yes	Yes
n + 1/2 layer	Yes	No	No
n + 1/2 layer	No	Yes	Yes

7.2 Momentum forcing

The verification suite runs an ensemble of simulations with different horizontal resolutions to assess the impact of resolution on the treatment of momentum within the model. These simulations are started from rest and a single grid point in the centre of the doubly periodic domain is subjected to a small wind forcing. The model is then integrated forward in time with all explicit viscosity and drag parameters set to zero. The difference between the simulated momentum and the expected momentum provides a measure of how well the model treats momentum.

All of the simulations are integrated for one model year using the same value for δt. This prevents variation due integrating for different amounts of model time or using different timesteps.

7.2.1 Reduced gravity mode

The magnitude of the error varies with δx, suggesting it is likely due to truncation error - the error induced by solving a continuous set of equations on a discrete grid.

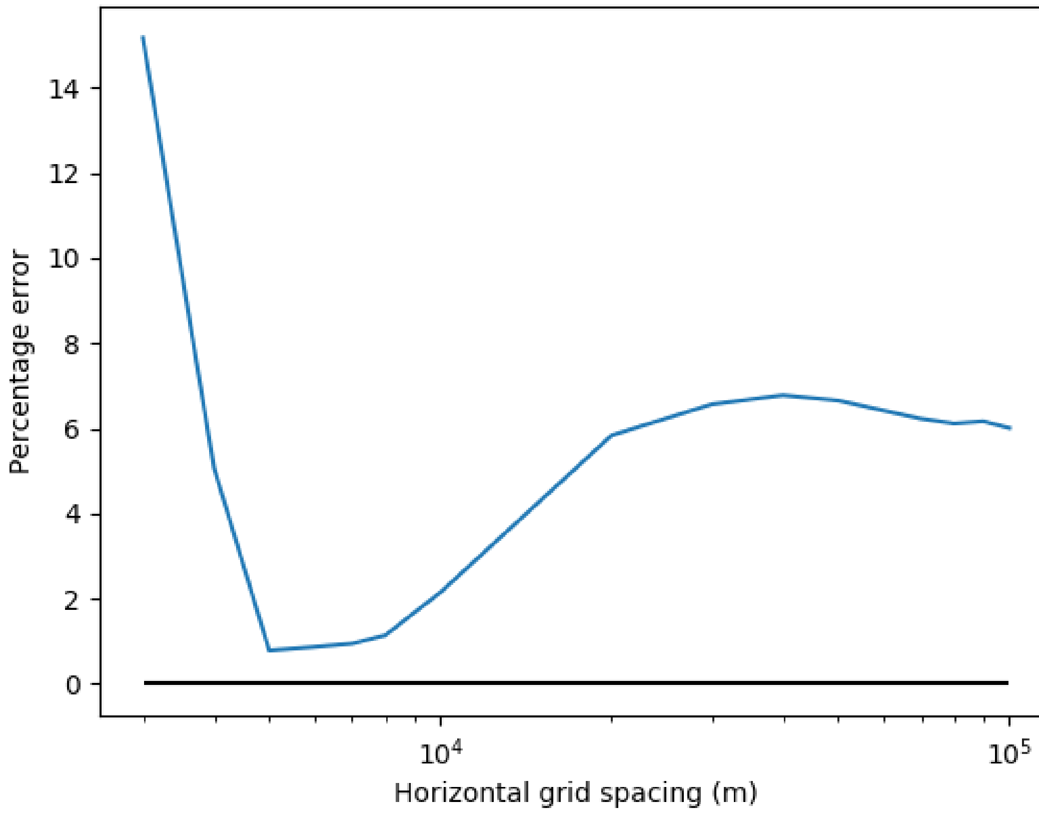


Figure 7.1: Momentum error as a function grid size for simulations using the reduced gravity mode.

The evolution of momentum in the 8 km resolution test simulation using reduced gravity physics.

7.2.2 n-layer mode

When running in the n-layer mode the momentum discrepancy is much larger and exhibits different variation with δx.

The evolution of momentum in the 8 km resolution test simulation using n-layer physics. The simulated momentum increases linearly, as expected, but the slope is much too large - the model obtains more momentum from the wind forcing than expected. This is undesirable, and is discussed in issue #100.

7.3 Momentum convservation

The verification suite also runs simulations to assess how well the model conserves momentum. Once again all explicit drag parameters have been set to zero. The Coriolis parameter has also been set to zero to remove inertial oscillations and allow for a cleaner test.

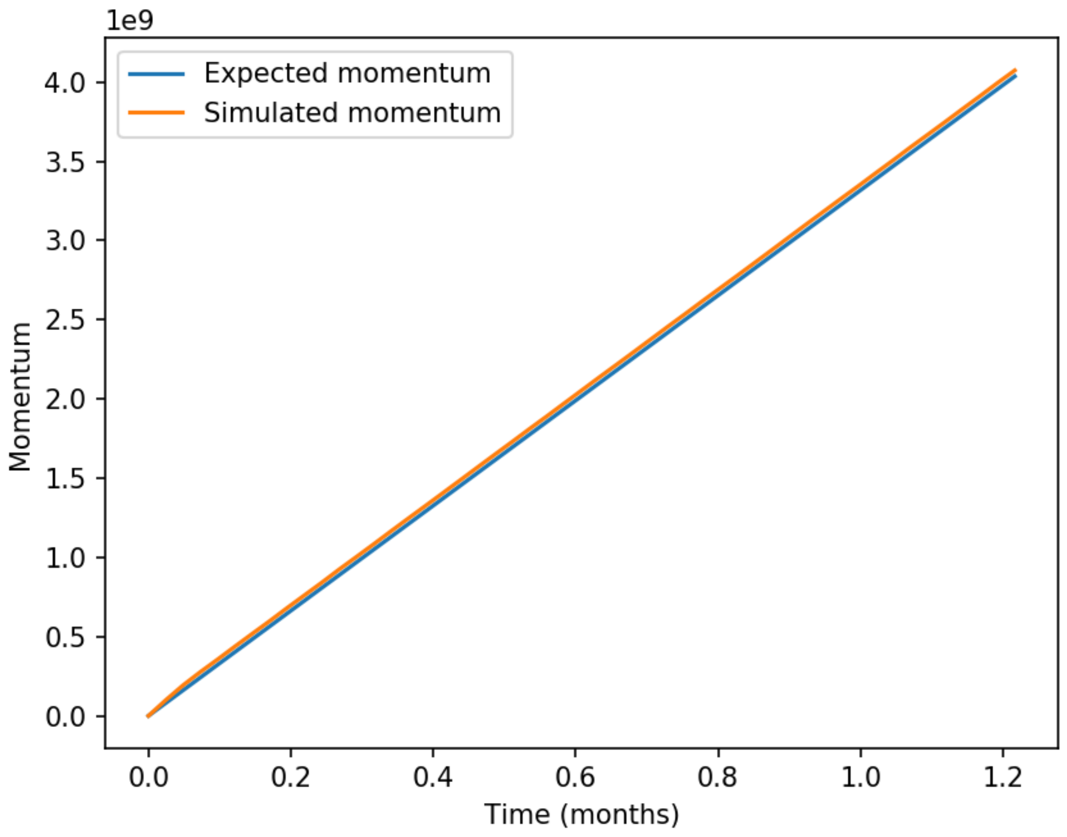


Figure 7.2: Momentum evolution as a function of time at 8 km resolution in the reduced gravity configuration. The two lines are indistinguishable.

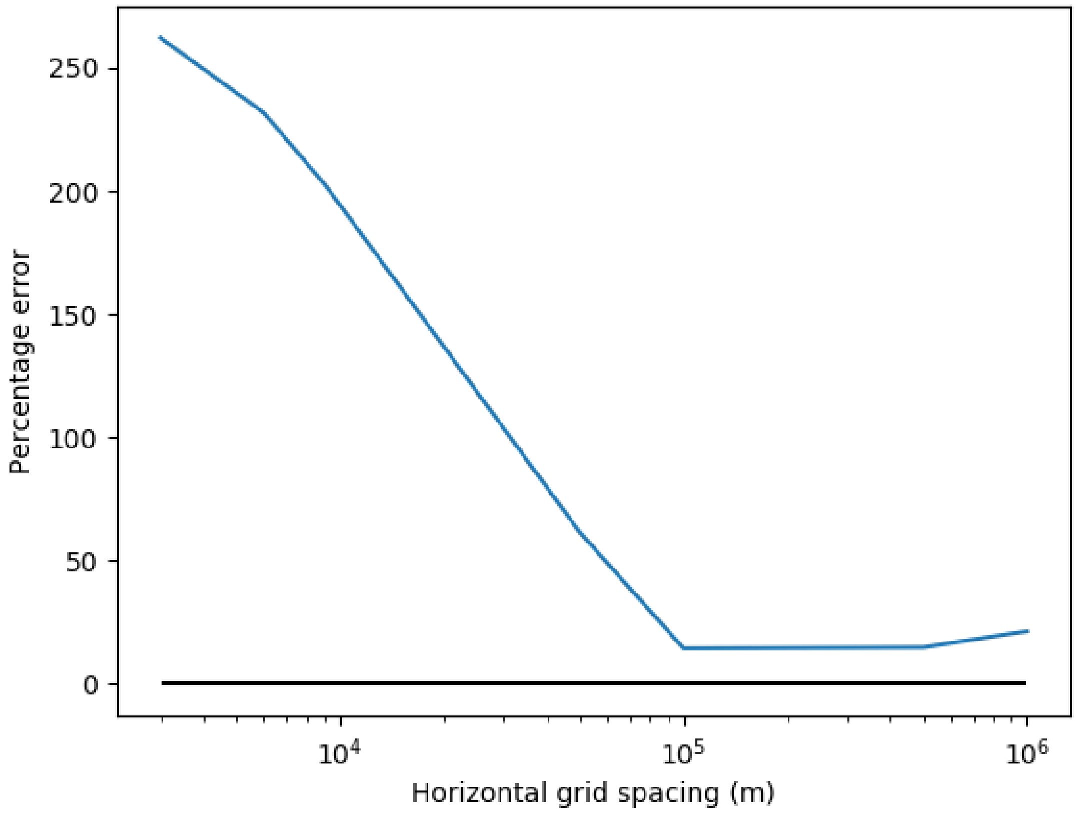


Figure 7.3: Momentum error as a function grid size for simulations using the n-layer mode.

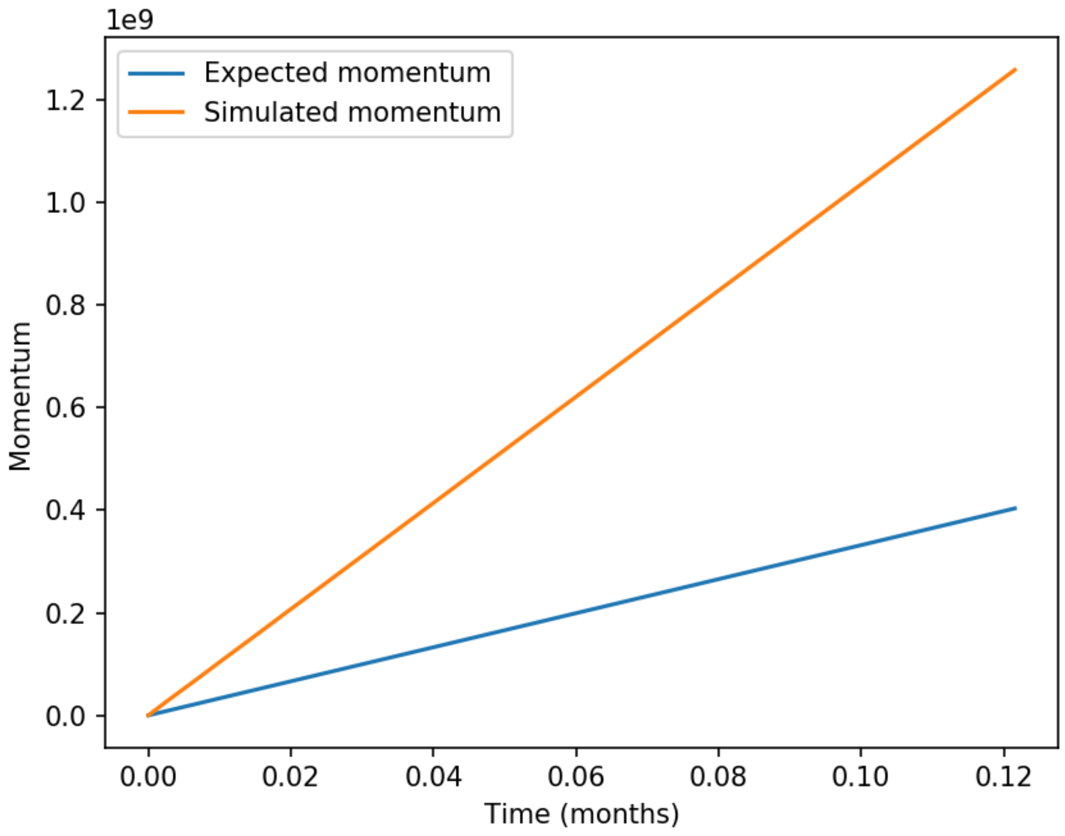


Figure 7.4: Momentum evolution as a function of time at 8 km resolution in the n-layer configuration.

In both the n-layer and reduced gravity modes the model conseres the initial momentum, as shown in Figure 7.5 and Figure 7.6.

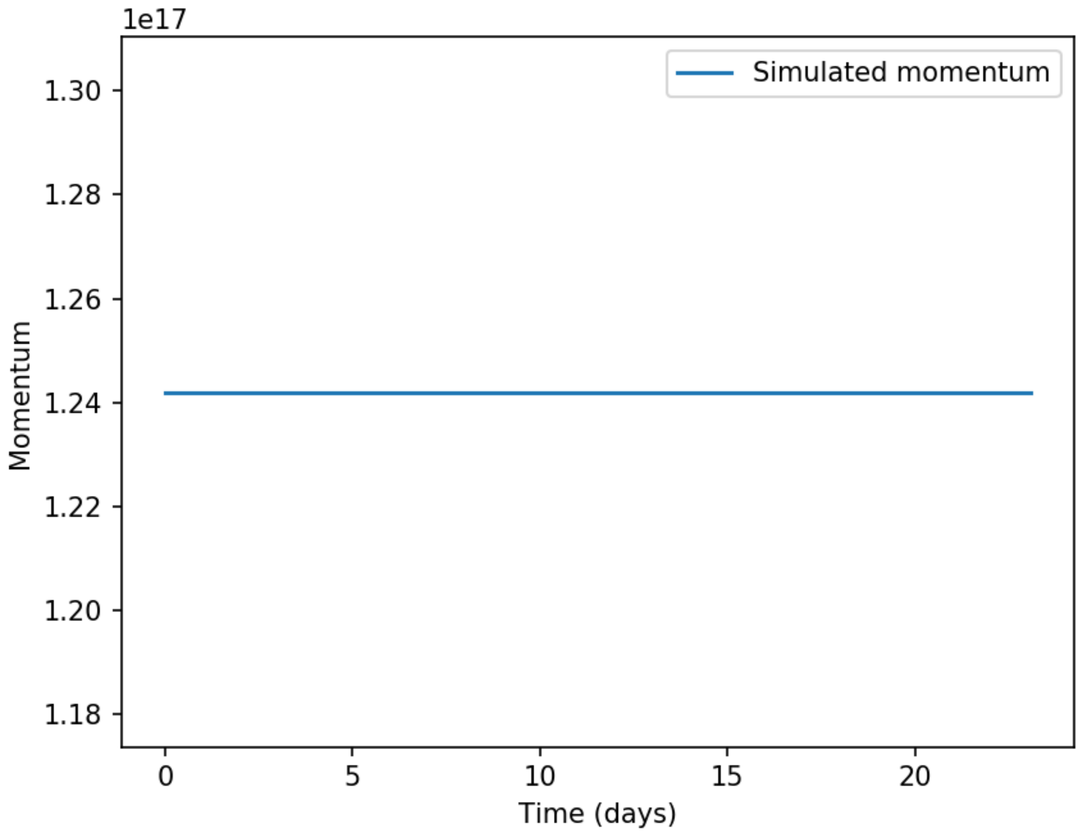


Figure 7.5: A time series of momentum from an unforced n-layer simulation with non-zero initial conditions.

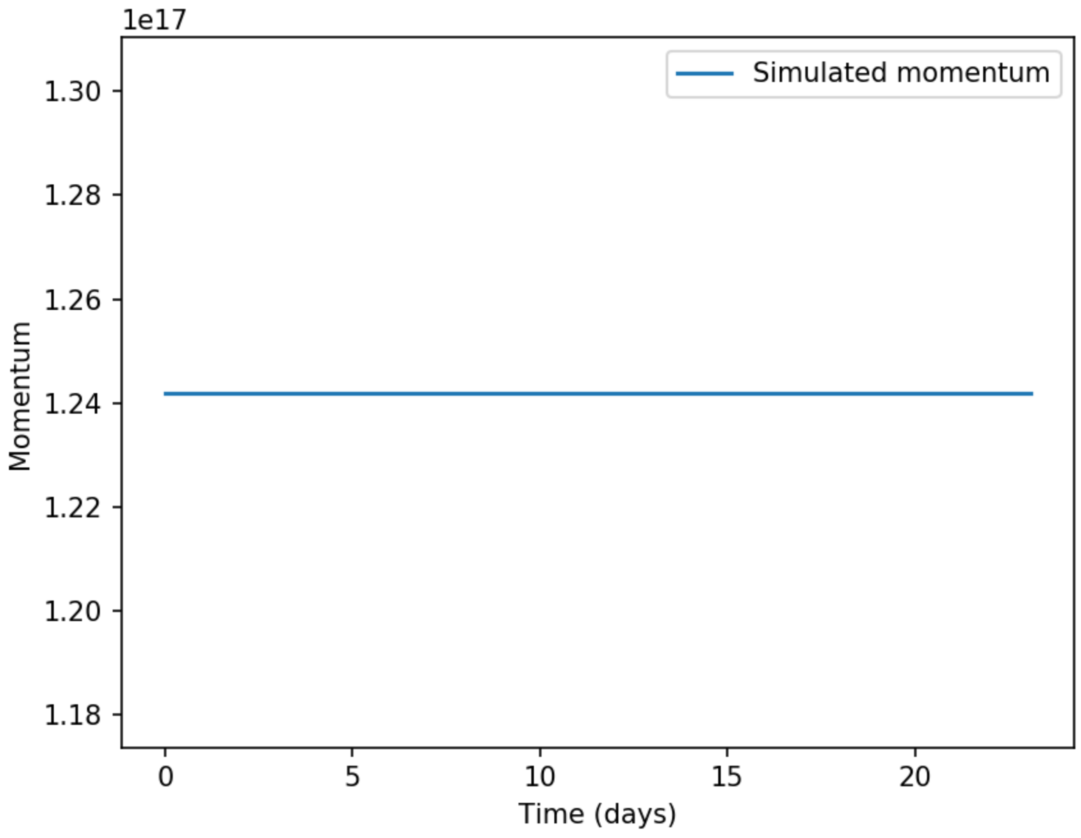


Figure 7.6: A time series of momentum from an unforced reduced gravity simulation with non-zero initial conditions.

CHAPTER 8

Benchmarking

Aronnax runs in two different modes. These two modes have substantially different runtimes due to the equations being solved.

These benchmarks can be reproduced by running the *benchmark.py* script in the *benchmarks/* directory. Be warned, it takes a while to run the full thing.

8.1 n-layer mode

Because it requires a linear equation solve at every timestep, including the ocean floor leads to substantially more expensive simulations.

8.2 Reduced gravity mode

The reduced gravity mode is substantially faster than the n-layer mode.

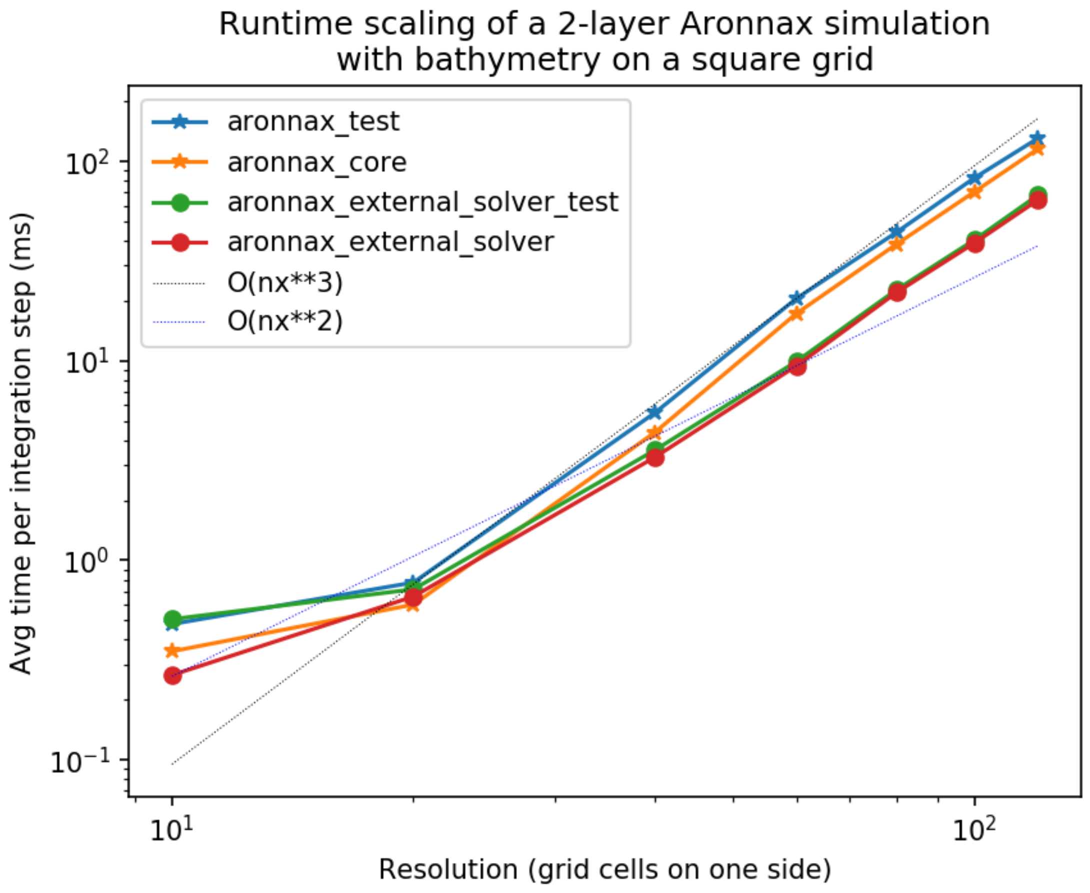


Figure 8.1: n-layer runtime scaling with resolution. These data are timings of different 500-step simulations of a Gaussian depth bump evolving in a β-plane approximation to the Earth's curvature, with different resolutions.

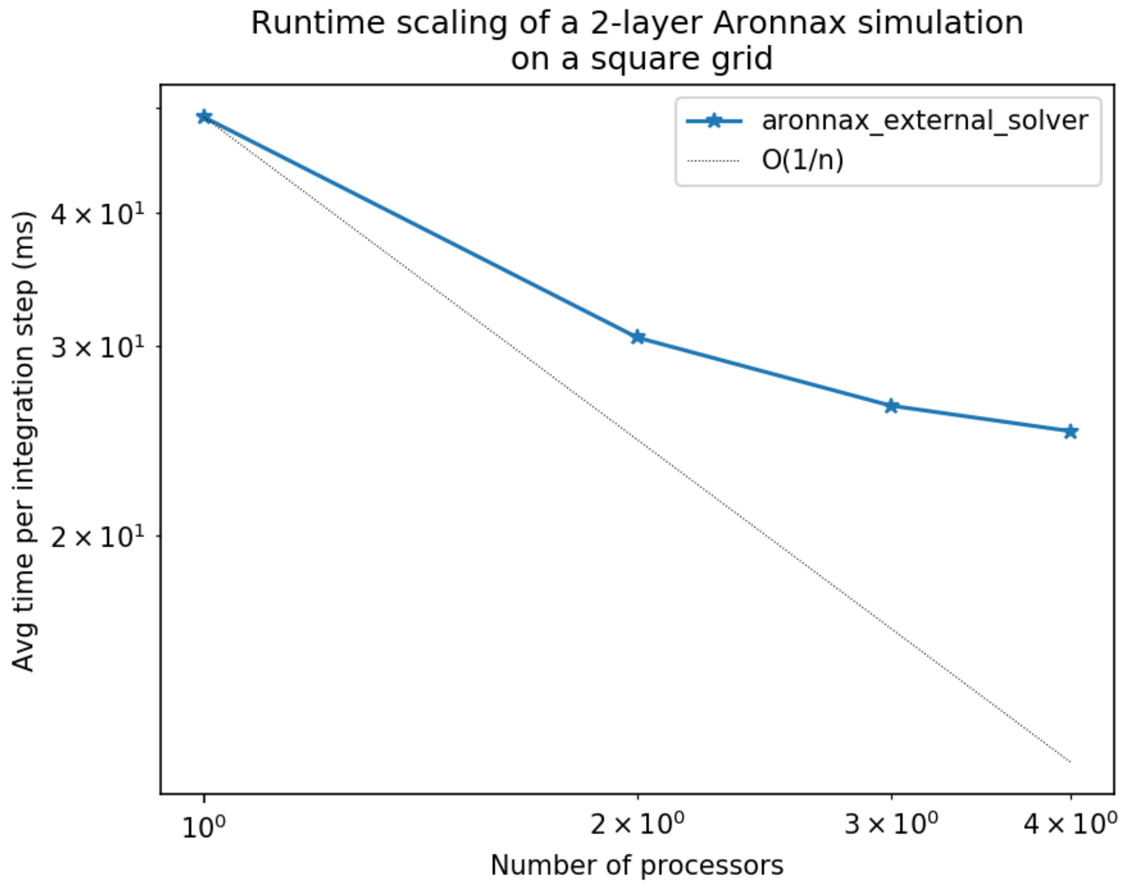


Figure 8.2: n-layer runtime scaling with number of processors. The speed increase with additional processors is not linear, but it is is noticeable.

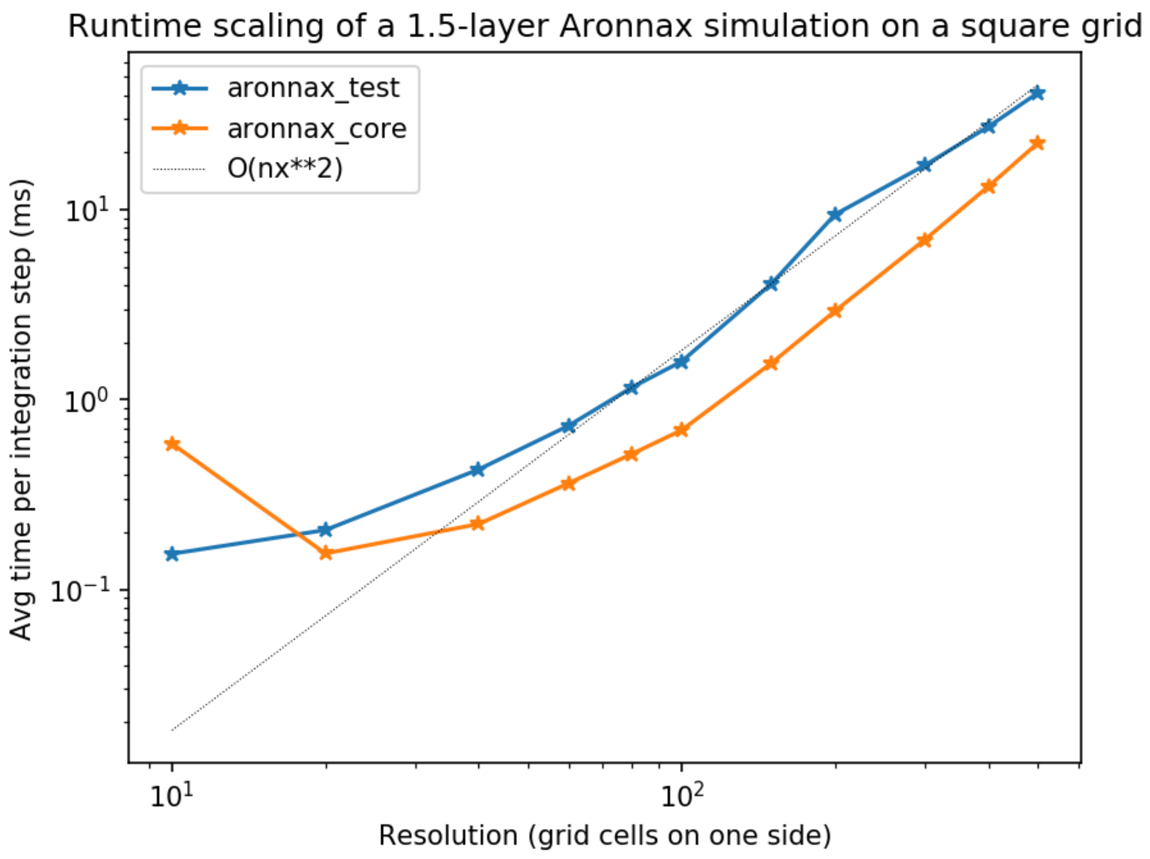


Figure 8.3: Reduced-gravity runtime scaling with resolution. These data are timings of different 500-step simulations of a Gaussian depth bump evolving in a β-plane approximation to the Earth's curvature, with different resolutions.

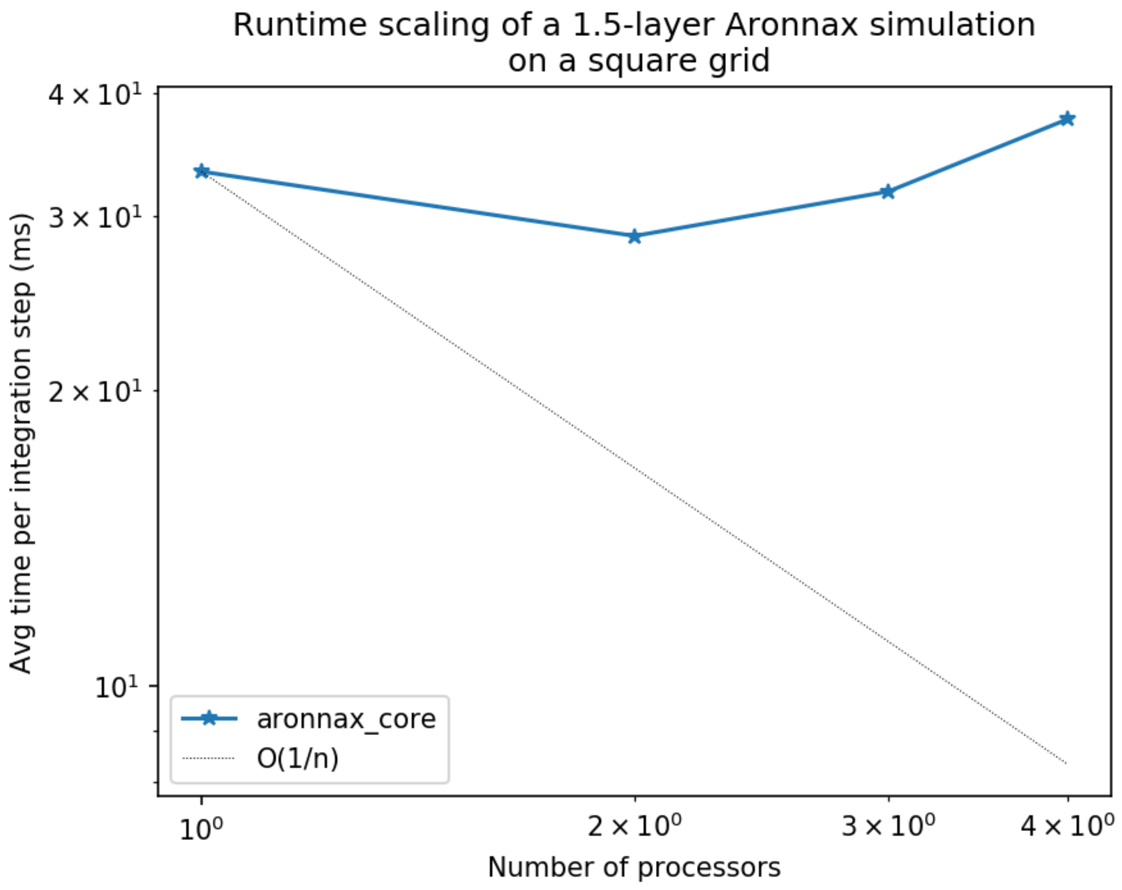


Figure 8.4: Reduced-gravity runtime scaling with number of processors. As you can see, it's not great, at least on my laptop; your mileage may vary. Luckily, this mode is already very fast.

CHAPTER 9

Publications

If you use Aronnax in a publication, please cite this paper:

Doddridge, E. W., & Radul, A. (2018). Aronnax: An idealised isopycnal ocean model. *Journal of Open Source Software*, 3(26), 592. http://doi.org/10.21105/joss.00592

9.1 Publications using Aronnax

None published yet - check back once the reviews are finished.

CHAPTER 10

Development

10.1 Since latest release

10.2 Version 0.3.0 (12 March 2019)

- MULTICORE Aronnax can now run in parallel across multiple processors using MPI GH212 (12 March 2019)
- Remove unused subroutine for enforcing minimum layer thickness GH211 (9 March 2019)
- Bug fix for bottom drag (didn't apply drag in one-layer simulations) GH209 (9 March 2019)
- Bug fix for open_mfdatarray, now respects variable location on C-grid GH208 (21 December 2018)
- Improved initial guess for external solver routine GH206 (20 November 2018)
- Python functions now have axis order specified by Comodo conventions GH204 (13 July 2018)
- Python wrapper can lazily load multiple timestamps into a 4D array (t,z,y,x) GH204 (13 July 2018)
- Aronnax paper published in Journal of Open Source Software (15 June 2018)

10.3 Version 0.2.0 (15 June 2018)

- Allow outcropping GH196 (30 April 2018)
- Add Python 3 compatibility GH189 (23 April 2018)
- Add Adams-Bashforth family of timestepping algorithms up to fifth-order GH184 (15 March 2018)
- Make test suite plot all differences when a test fails, as suggested in GH136 and implemented in GH181 (12 March 2018)
- Refactor Fortran code into modules in src/ directory GH181 (12 March 2018)
- Add option for first-order upwind advection scheme GH179 (8 March 2018)

- Add vertical diffusion of mass between layers GH177 (2 March 2018)

10.4 Version 0.1.0 (11 December 2017)

Initial release

CHAPTER 11

Indices and tables

- genindex
- modindex
- search

Index