



---

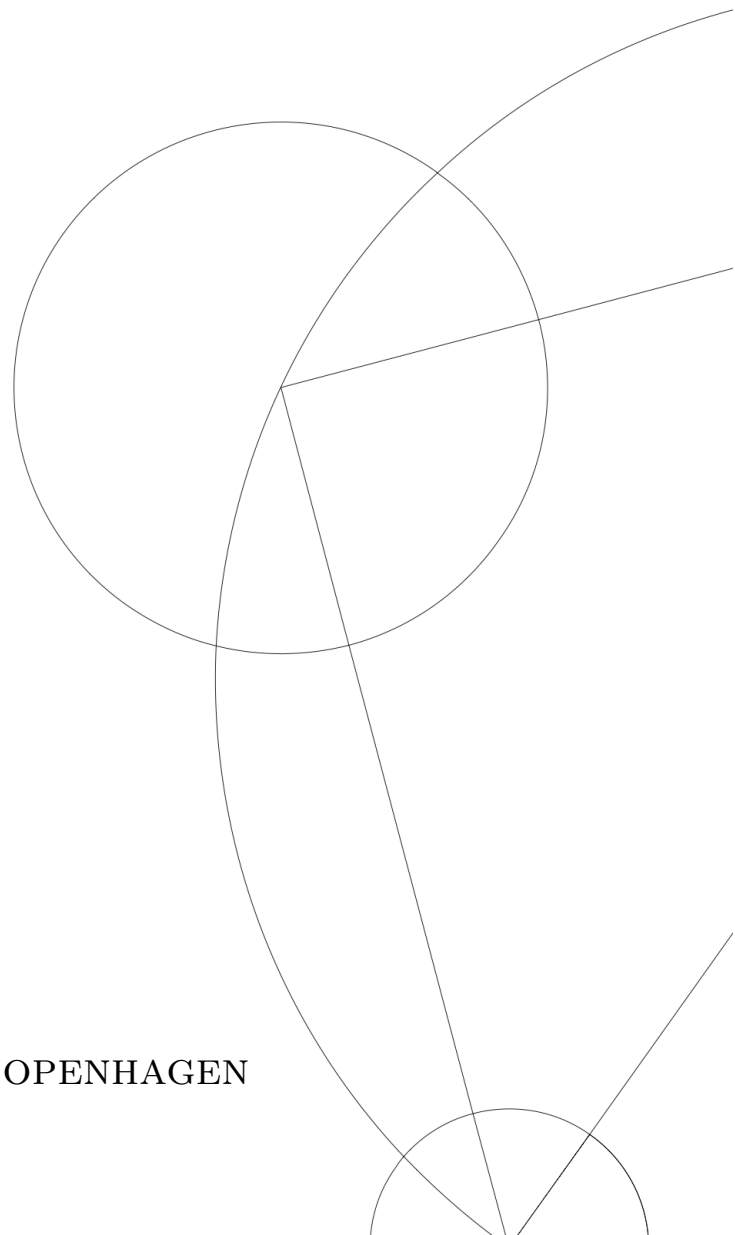
# NO SLIP LATERAL BOUNDARY CONDITIONS FOR VEROS

BACHELORS THESIS

Written by *Jesper R. Pedersen*

June 11, 2018

Supervised by  
Markus Jochum



UNIVERSITY OF COPENHAGEN



UNIVERSITY OF  
COPENHAGEN

NAME OF INSTITUTE: Niels Bohr Institute  
NAME OF DEPARTMENT: Climate and Geophysics  
AUTHOR(S): Jesper R. Pedersen  
EMAIL: xhn538@ku.dk  
TITLE AND SUBTITLE: No slip lateral boundary conditions for Veros  
-  
SUPERVISOR(S): Markus Jochum  
HANDED IN: 11.06.2018  
DEFENDED: 21.06.2018

NAME Jesper Rask Pedersen

SIGNATURE 

DATE 11/06/2018

## **Abstract**

Veros is an open-source general circulation ocean model completely written in Python, and the aim of this thesis is to add the option of no slip lateral boundary conditions to Veros, the versatile ocean simulator. Previously the Veros only had the option of using free slip, but by introducing no slip it is possible to choose the condition one thinks serves one best. The implementation is analogue to one by Carsten Eden for a shallow water model, shared with the author. I run the ACC model, one of the standard setups of Veros, for both kinds of boundary condition, and find that the velocity close to the boundary is significantly reduced when using no slip compared to free slip.

# Contents

<b>Abstract</b>	<b>1</b>
<b>Contents</b>	<b>2</b>
<b>List of Figures</b>	<b>2</b>
<b>1 Introduction</b>	<b>3</b>
<b>2 Theory</b>	<b>3</b>
2.1 Interior circulation . . . . .	4
2.2 The homogeneous model . . . . .	4
<b>3 Veros</b>	<b>7</b>
3.1 Implementing no slip boundary conditions . . . . .	7
3.2 The ACC model . . . . .	9
<b>4 Comparison of free- and no slip</b>	<b>11</b>
<b>5 Discussion and Conclusion</b>	<b>12</b>
<b>Acknowledgements</b>	<b>13</b>
<b>References</b>	<b>14</b>
<b>Apx. A; Veros friction.py</b>	<b>15</b>
<b>Apx. B; ACC model setup</b>	<b>23</b>
<b>Apx. C; Code for Plots</b>	<b>26</b>
<b>Apx. D; Carsten Eden’s implementation of no slip</b>	<b>32</b>

## List of Figures

1 Munk’s solution . . . . .	6
2 Arakawa C-grid illustration . . . . .	7
3 The wind stress forcing of the ACC model . . . . .	9
4 Time series of key phenomena . . . . .	10
5 Comparison of ACC . . . . .	11
6 Streamlines of northern basin . . . . .	12
7 Western boundary current . . . . .	13

# 1 Introduction

The aim of this thesis is to add the option of no slip lateral boundary conditions to Veros, the versatile ocean simulator. Veros is an open-source general circulation ocean model completely written in Python [1], [2]. Historically Fortran has been the customary language for ocean models, but by writing it in Python, Veros integrates easily with the great amount of open-source projects in Python, and should be easier to access for people new to the field. Veros is based on the Fortran model pyOM2.1, created by Carsten Eden [3] [4], and in the translation of the model only the option of free slip lateral boundary conditions was available. Such boundary conditions are normally used in coarse ocean models, since the sheer scale of these models makes the standard physical arguments for using no slip irrelevant. However it is not entirely clear which boundary condition to use, and this answer might depend on the scale of the model, adding no slip to Veros gives experimenters the option to decide for themselves. Determining what the correct boundary condition to use (assuming that question have a definite answer), is **not** the objective of this thesis. That is beyond the scope of this project.

One famous western boundary current is the gulf stream, which has a major impact on the weather and climate of northern Europe. It is interesting to see the difference in western boundary currents due to boundary conditions. Being able to simulate this current is essential, if you would want to make any predictions on the future of the climate of northern Europe. In order to explore this I will introduce Munk's solution to the dynamics of a homogenous general ocean circulation model as it is presented in [[5]], and compare it to a simple model I have simulated.

To thesis is laid out as follows: In section 2 I will introduce the theoretical background behind Munks solution. Section 3 will concern the problem of implementing no slip boundary conditions in Veros. Veros is using an Arakawa C-grid, and this grid will be introduced along with a description of what no slip translates to here. Finally I will describe the setup of the ACC model I run to compare the boundary conditions. Section 4 introduce the results of running the ACC model in Veros. I will take a qualitative look at the differences in the ACC, followed by a direct comparison of the western boundary currents. Section 5 will be discussion and conclusion. Furthermore I have include the appendices: A; Veros friction.py, which contains almost all of the edits I made to Veros, B; ACC model setup, the standard setup, C; Code for Plots and D; Carsten Eden's implementation of no slip.

## 2 Theory

This section redoes and draws heavily on what Joseph Pedlosky lays out in [5] in chapters one and two, in order to give us an idea of how a western boundary current might look different depending on its boundary conditions. For a fuller exposition on the subject I recommend looking there.

I will use the homogeneous, constant density, model as a reference point to make a comparison between free slip and no slip boundary conditions, since the homogeneous model is a simple model that has a relation between interior circulation and the dynamics of a western boundary current.

## 2.1 Interior circulation

If we start with the general equation of motion in the ocean:

$$\frac{D\bar{u}}{Dt} + 2\bar{\Omega} \times \bar{u} = -\frac{\nabla p}{\rho} + \bar{g} + \frac{\tau}{\rho}, \quad (1)$$

where  $\frac{D\bar{u}}{Dt}$  is the relative acceleration,  $2\bar{\Omega} \times \bar{u}$  is the Coriolis term,  $\frac{\nabla p}{\rho}$  is the pressure gradient,  $\bar{g}$  represents gravity and  $\frac{\tau}{\rho}$  a turbulent momentum mixing, which occurs on small scale and acts as diffusion on larger. The first thing of interest is size of the relative acceleration to the Coriolis terms. If the basin has a length scale of  $L$ , and a scale of velocity  $U$ , then the scale for time is  $L/U$  giving us the scale of the relative acceleration as  $U^2/L$ . The Coriolis term will have the scale  $2\Omega \sin \theta U$  giving us the ratio:

$$\frac{U}{fL} \equiv R_0 \quad \text{the Rossby number,} \quad (2)$$

$$f = 2\Omega \sin \theta.$$

In the ocean the scale of the flows makes the Rossby number of the order  $10^{-4}$ . Thus the Coriolis term dominates the left side of Eq. 1. If one looks at a region where the turbulent momentum mixing is insignificant, the horizontal part of the momentum equation reduces to:

$$\rho f \hat{k} \times \bar{u} = -\nabla p. \quad (3)$$

Which is the geostrophic balance.

## 2.2 The homogeneous model

The homogeneous model is a three layered model with a homogeneous layer of thickness  $D$  under a mixed layer affected by wind stress. At the bottom we have a boundary layer coupling the fluid to the bottom. The fluid is taken to be in geostrophic balance, with a small Rossby number and small north-to-south extent such that we can use the  $\beta$  plane approximation:

$$f = f_0 + \beta y. \quad (4)$$

And here we assume that  $f_0 \gg \beta y$ . This, and the fact that we have geostrophic

balance, allows us to write :

$$\begin{aligned}\psi &= \frac{p}{\rho f_0}, \\ u &= -\frac{\partial\psi}{\partial y}, \\ v &= \frac{\partial\psi}{\partial x},\end{aligned}\tag{5}$$

where  $\psi$  is the stream function,  $p$  the pressure,  $\rho$  the density and  $f_0$  is a characteristic parameter of the Coriolis term in the gyre. Now the relative vorticity  $\zeta$ :

$$\zeta = \frac{\partial v}{\partial x} - \frac{\partial u}{\partial y},\tag{6}$$

has the governing equation:

$$\frac{d(\zeta + \beta y)}{dt} = (f_0 + \beta y + \zeta) \frac{\partial w}{\partial z} + A_H \nabla^2 \zeta.\tag{7}$$

Here  $(f_0 + \beta y + \zeta) \frac{\partial w}{\partial z}$  represents the increase in vorticity by stretching and  $A_H \nabla^2 \zeta$  is diffusion. Here we can approximate:

$$f_0 + \beta y + \zeta \approx f_0.\tag{8}$$

At the same time  $\frac{\partial w}{\partial z}$  is independent of depth since the model is homogeneous, so by integrating Eq. 9 vertically, and diving by  $D$ , we get:

$$\frac{d\zeta}{dt} + \beta v = \frac{f_0}{D}(w_E - w_B) + A_H \nabla^2 \zeta.\tag{9}$$

When written in terms of the stream function gives us the governing equation for the homogenous as:

$$\frac{\partial}{\partial t} \nabla^2 \psi + J(\psi, \nabla^2 \psi) + \beta \frac{\partial \psi}{\partial x} = \frac{f_0}{D} w_E - r \nabla^2 \psi + A_H \nabla^4 \psi.\tag{10}$$

In the region of the western boundary current is reasonable to make the following assumption:

$$\frac{\partial}{\partial x} \gg \frac{\partial}{\partial y}.\tag{11}$$

Which, in the steady state, reduces 10 to:

$$J(\psi, \frac{\partial^2 \psi}{\partial x^2}) + \beta \frac{\partial \psi}{\partial x} = \frac{f_0}{D} w_E - r \frac{\partial^2 \psi}{\partial x^2} + A_H \frac{\partial^4 \psi}{\partial x^4}.\tag{12}$$

This can be solved for the boundary layer by ignoring nonlinearity in order to create a simple solution for a western boundary current. Ignoring nonlinearity requires the

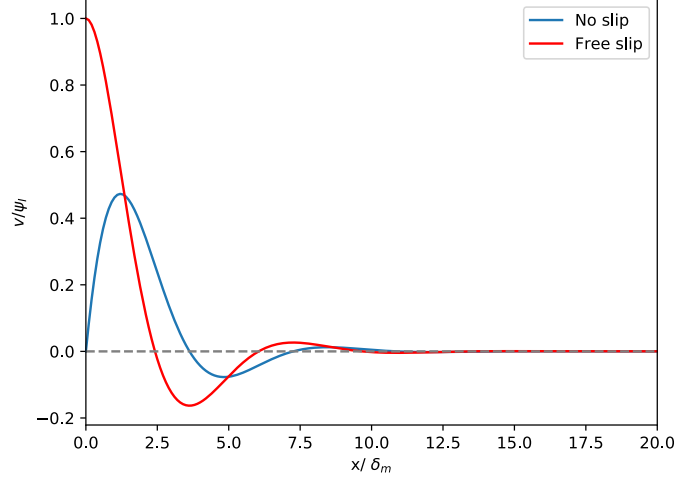


Figure 1: A plot of  $v$  for a western boundary current obtained by ignoring nonlinearity in the boundary layer for both free- and no slip boundary conditions.

Reynolds number to be small, which demands that the forcing is weak or the horizontal turbulent mixing is large. Eq. 12 reduces to:

$$A_H \frac{\partial^4 \psi}{\partial x^4} - \beta \frac{\partial \psi}{\partial x} = 0. \quad (13)$$

If the boundary is at  $x = 0$  and the solution for  $\psi$  has to go tend toward  $\psi_I$  we get:

$$\psi = \psi_I(x, y) [1 - \exp(-x/2\delta_M) \cos(\frac{\sqrt{3}x}{2\delta_M})] + C(y) \exp(-x/2\delta_M) \sin(\frac{\sqrt{3}x}{2\delta_M}). \quad (14)$$

Imposing the no slip boundary condition leaves us with:

$$\begin{aligned} \psi &= \psi_I(x, y) [1 - \exp(-x/2\delta_M) (\cos(\frac{\sqrt{3}x}{2\delta_M}) + \frac{1}{\sqrt{3}} \sin(\frac{\sqrt{3}x}{2\delta_M}))], \\ v &= \frac{2}{\sqrt{3}} \frac{\psi_I(x, y)}{\delta_M} \exp(-x/2\delta_M) \sin(\frac{\sqrt{3}x}{2\delta_M}). \end{aligned} \quad (15)$$

If we instead chose the free slip condition we would have had:

$$\begin{aligned} \psi &= \psi_I(x, y) [1 - \exp(-x/2\delta_M) (\cos(\frac{\sqrt{3}x}{2\delta_M}) - \frac{1}{\sqrt{3}} \sin(\frac{\sqrt{3}x}{2\delta_M}))], \\ v &= \frac{2}{\sqrt{3}} \frac{\psi_I(x, y)}{\delta_M} \exp(-x/2\delta_M) (\cos(\frac{\sqrt{3}x}{2\delta_M}) + \frac{1}{\sqrt{3}} \sin(\frac{\sqrt{3}x}{2\delta_M})). \end{aligned} \quad (16)$$

In the limit  $x \gg \delta_M$  these all tend towards the interior solution.  $v$  is plotted in Fig. 1 for both cases.



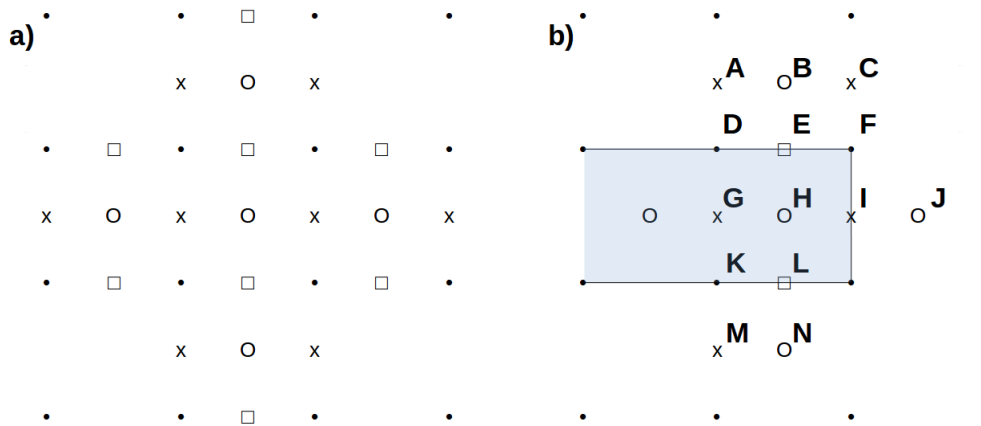


Figure 2: A plot of the Arakawa C-grid. The keys are: Circles are the tracers. Crosses are zonal velocity,  $u$ , and squares meridional velocity,  $v$ . Since the grid is staggered the keys doesn't overlap. In part b) various points are labelled to be used in explaining no slip boundary complications.

### 3 Veros

The versatile ocean simulator [1]. Veros is a translation of pyOM2 (v2.1.0) [3], [4] a finite-difference ocean model, originally created by Carsten Eden, to Python. The idea behind Veros was to make a general circulation ocean model that is, as written in Veros documentation; easy to access, use, verify and modify [6]. Veros is open source, accessible on github [2].

Generally in fluid dynamics you assume no slip boundary conditions, but for ocean models which are usually much more coarse than your typical fluid dynamics problem, free slip boundary conditions are the standard. The difference between no slip and free slip is the possible tangential value of the velocity of the fluid at the boundary. For no slip the tangential velocity has to be zero, whereas for free slip it can be non-zero.

#### 3.1 Implementing no slip boundary conditions

Veros is based on an Arakawa C-grid, and was initially implemented with only the free-slip condition. The Arakawa C-grid is illustrated in Fig. 2a.

In Veros the land is masked and it's impermeable. The mask is implemented in such a way as to easily set all velocities normal to a sea-land boundary to zero. In Veros the land mask is defined as grid squares with centre in the tracer grid points, where in the land area all values are zero. The boundary between land and sea is thus always defined on the velocity points, in between two tracer grid points. This is illustrated in Fig. 2b with the points (D,E,F,G,H,I,K,L) being land, and the land-sea boundary drawn explicitly. Since the Arakawa C-grid is staggered, meaning that velocities are evaluated in half a grid space between the tracers, the tangential velocities are never defined on

the boundary directly. The boundary contains only the normal velocities which were set to zero by the mask. The tangential velocities are defined half a grid space away, which for ocean models can be many kilometers. As such the tangential velocities only exist implicitly when one calculates any differentials in a direction across a boundary. An example being Point A in Fig. 2b, where  $u$  is defined.

The way no slip is implemented is then as follows. The difference between the values of  $u$  at A and G is just  $u_A$ , since G is on the land. This implies that the gradient between A and G is  $u_A / \delta y$ , and if you then extrapolate linearly to the value of the velocity at the boundary (D), you get, in Cartesian coordinates:

$$\frac{\partial u_D}{\partial y} = \frac{u_G - u_A}{\delta y} = -\frac{u_A}{\delta y} \quad (17)$$

which is what we would like for free slip, but not for no slip. The way to fix this is to multiply any gradient crossing a land-sea boundary by 2:

$$\frac{\partial u_D}{\partial y} = 2 * \frac{u_G - u_A}{\delta y} = -\frac{2 * u_A}{\delta y}. \quad (18)$$

In this way the value at D is instead:

$$u_D = u_A + \frac{\partial u_D}{\partial y} \cdot \frac{\delta y}{2} = 0. \quad (19)$$

This is what Carsten Eden did in his scripts, attached in Apx D, which was the basis for my implementation. It's also what has been done in other ocean models I looked up, respectively Nemo [7], and ROMS [8].

In the code it resulted in the addition of the option of no slip lateral boundary conditions as:

---

```
1 enable_noslip_lateral = True/False
```

---

In the code this switches an additional calculation on, exemplified below:

---

```
1 vs.flux_east[:-1] = vs.A_h * fxa[:, :, np.newaxis] * (vs.v[1:, :, :, vs.tau] -
2     vs.v[:-1, :, :, vs.tau]) \
3     / (vs.cosu * vs.dxu[:-1, np.newaxis])[:, :, np.newaxis] * vs.maskV[1:]
4     * vs.maskV[:-1]
3 if vs.enable_noslip_lateral:
4     vs.flux_east[:-1] = vs.flux_east[:-1] \
5         + 2 * vs.A_h * fxa[:, :, np.newaxis] * vs.v[1:, :, :, vs.tau] \
6         / (vs.cosu * vs.dxu[:-1, np.newaxis])[:, :, np.newaxis] *
7         vs.maskV[1:] * (1 - vs.maskV[:-1]) \
8         - 2 * vs.A_h * fxa[:, :, np.newaxis] * vs.v[:-1, :, :, vs.tau] \
         / (vs.cosu * vs.dxu[:-1, np.newaxis])[:, :, np.newaxis] * (1 -
         vs.maskV[1:]) * vs.maskV[:-1]
```

---

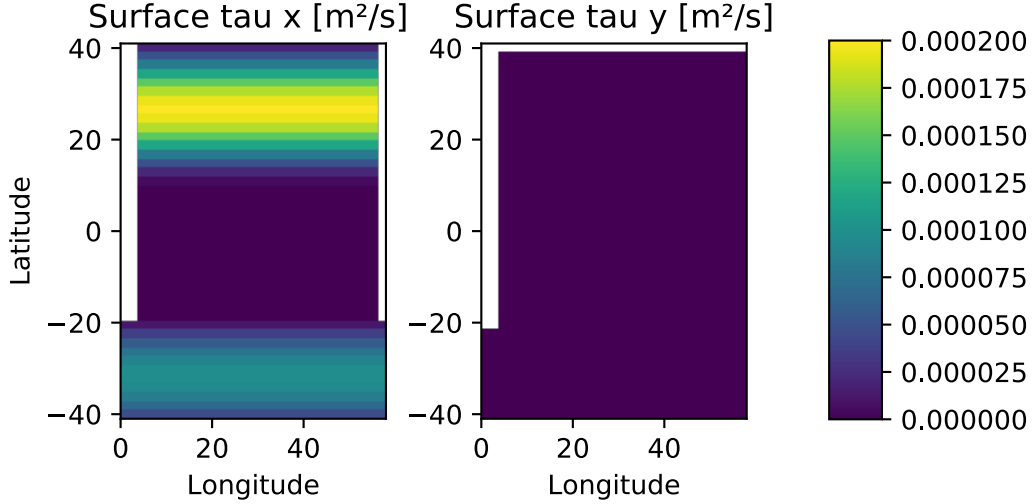


Figure 3: A plot of the wind stress forcing on the ACC model. On the left I have plotted the wind stress in the zonal direction, and on the right the meridional. The model has periodic boundary conditions in the zonal direction, and has walls in the meridional. The wind stress is constant in time.

Here we are calculating the flux. First the flux east is calculated using the zonal difference between  $v$ . This means that we are possibly calculating across land-sea boundaries but a mask:

$$vs.maskV[1:] * vs.maskV[:-1], \quad (20)$$

assures that we look at difference between sea and sea. Then in the no slip case, we specifically add an instance for the case where we calculate across boundaries by the masks:

$$(1 - vs.maskV[1:]) * vs.maskV[:-1], \quad (21)$$

and

$$vs.maskV[1:] * (1 - vs.maskV[:-1]). \quad (22)$$

Here we add 2 times the difference, thus creating the "implicit" no slip boundary conditions.

### 3.2 The ACC model

In order to compare free slip to no slip, and as a proof of concept, I used a simple model of the Antarctic Circumpolar Current, ACC, one of the standard setups in Veros. The only difference between the two runs of the model was the lateral boundary condition.

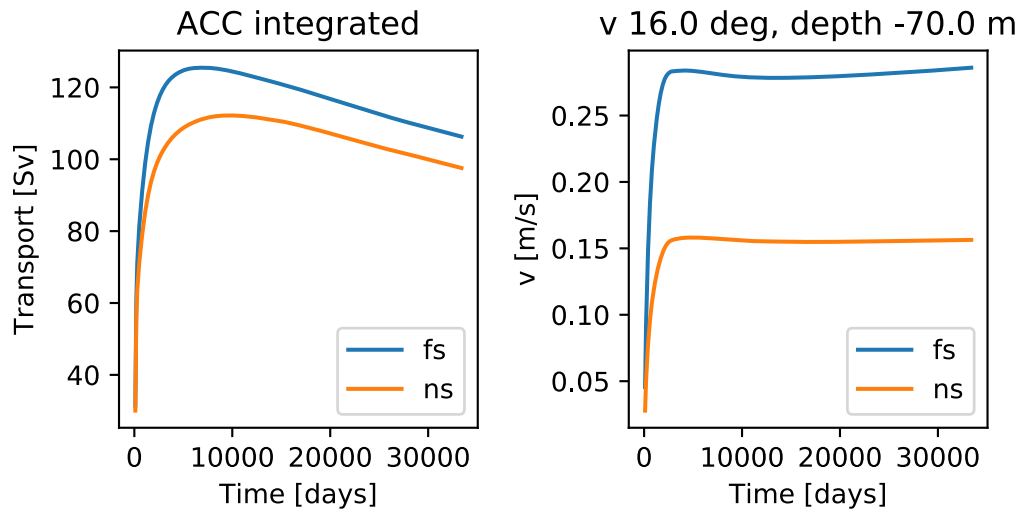


Figure 4: On the left the total transport of the ACC in the model is plotted, and on the right the value of  $v$  at the western boundary.

The ACC model has periodic boundary conditions in the zonal direction, and has land walls in the meridional. I have tried to visualize the grid by plotting the surface wind stress in both the zonal and meridional direction in Fig. 3. The wind stress is constant. In the  $y$  direction it's zero, and its' value in the  $x$  direction is constant in time. In the model there is a continent, effectively creating a norther basin, and a simple ACC below it. I assume that the gyre created in the northern basin is comparable to the one analyzed in the theory section.

The model is in degree coordinates from 0 to 60 West-East and 40 to -42 North-South, with a grid spacing of 2 degrees. It uses linear bottom friction, without no slip, and horizontal harmonic friction, where one can have either free- or no slip. In order to try to determine whether I reached a steady state solution I simulated the model for approximately 35000 days (95 years). In Fig. 4 I plotted a time series of the total transport of the ACC and the value of  $v$  representing the western boundary current. On my laptop it takes about 3 hours to simulate that far, using the Bohrium framework. Veros can run with both Numpy and Bohrium, with Bohrium being developed locally at The Niels Bohr Institute [9]. While the ACC is not entirely stable by this point, the value of  $v$  seems more so. Going much beyond 100 years of simulation seemed excessive to me. I sample every 100 days and for the results in the next section I use the time average of the last 8 samples.

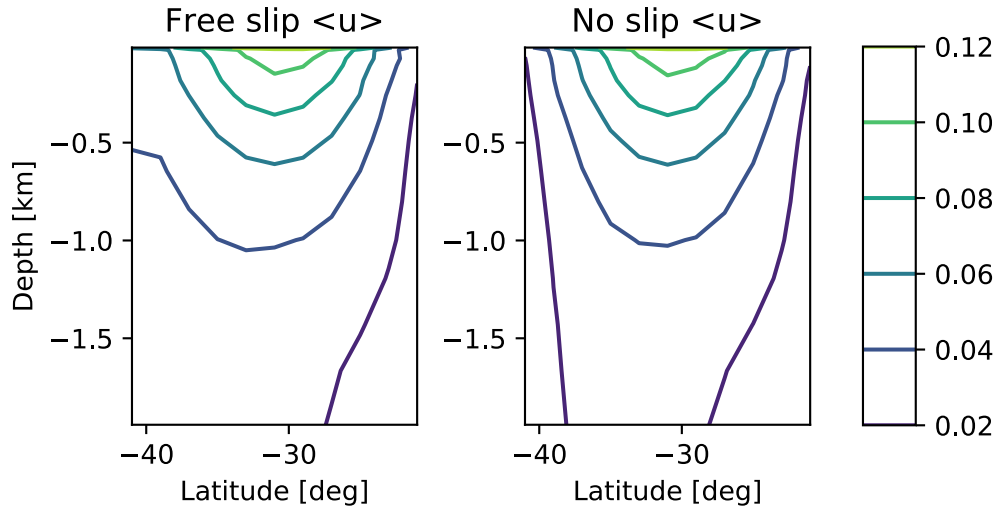


Figure 5: A comparison of the contours of  $u$  for the ACC between free slip, on the left, and no slip, on the right. The comparison is on the average of  $u$  for 20 degrees longitude. The latitude is from -41 degrees which is also the end of the model. The boundary is thus at -42 degrees, and the difference in  $u$  is immediately clear from looking at the figures.  $u$  is not exactly 0 here since we are 1 degree away from the boundary still, since we are using the Arakawa C-grid.

#### 4 Comparison of free- and no slip

I looked at the zonal flow of the part of the model meant to represent the ACC, averaged over 3 years and 20 longitudinal degrees. The contours of the flow is plotted in Fig. 5. The land-sea boundary with Antarctica is the reason for the difference between the two sets of flow, with the no slip condition greatly reducing the tangential flow nearby the southern boundary. The flow is strongest  $10^\circ$  away from the boundary, and here it is very similar between the two experiments.

The stream function is plotted for both cases in Fig. 6. The plot begins from  $10^\circ$  S in order that the spectrum is dominated by the stream function of the ACC. From the plot of the stream functions we can see that the overall circulation is very similar.

The  $v$  of western boundary current plotted in Fig. 7 at  $16^\circ$  N. If we compare Fig. 7 to the value of  $v$  in Munk's solution 1 we see that the free slip case has maximum velocity close to the boundary, before dropping under 0, whereas for no slip the velocity is lower, but decreases less steeply away from the boundary.

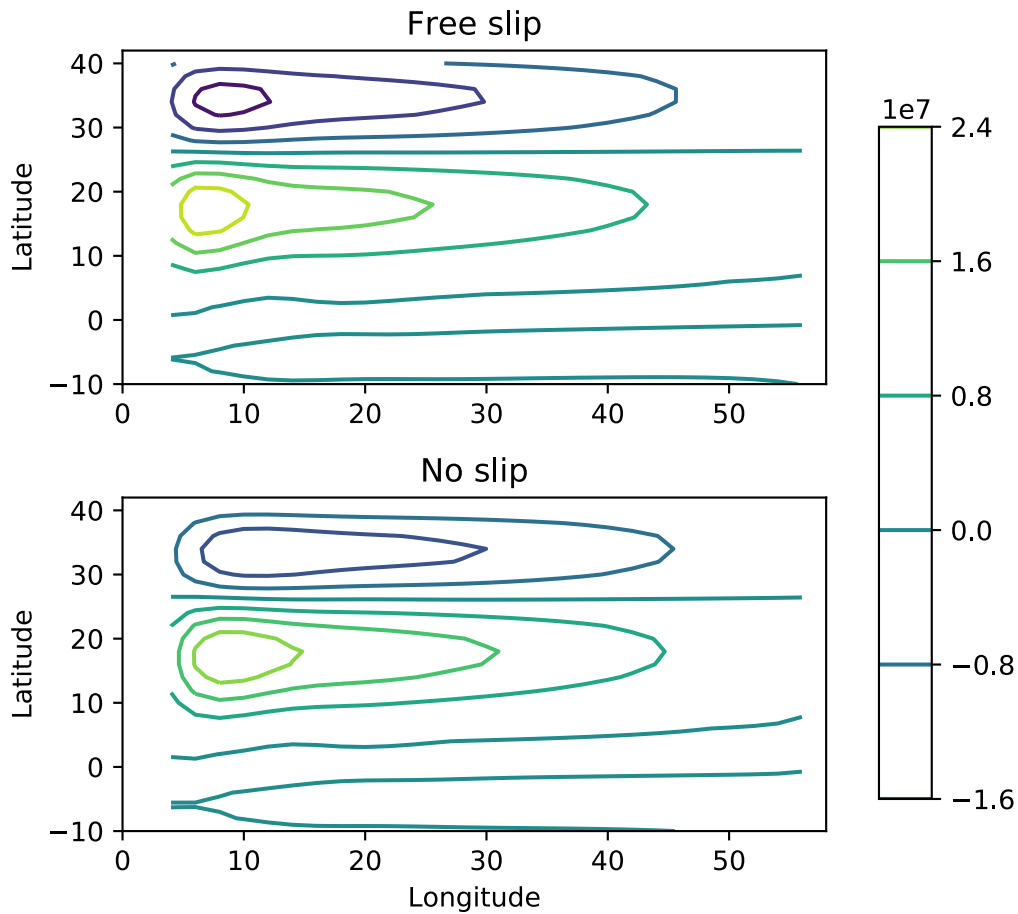


Figure 6: The streamlines plotted starting from the bottom of the continent, at -10 south. The western boundary current is visible around latitude 10 to 20.

## 5 Discussion and Conclusion

The first thing to comment upon is that even though I simulated almost a hundred years for the model, it looks like we are not yet in a stable state. Whether this is because such a state doesn't exist or that I merely ended it too quickly is unknown to me. If I had had more time I would have liked to run the model further in order to investigate this.

I would have liked to see how my results depended on the scale of the model. Whether they would look more, or less alike if the resolution was doubled or tripled.

I would also have liked to try to calculate explicitly how applicable the homogeneous model was to the ACC model. The assumptions of a characteristic value for the Coriolis

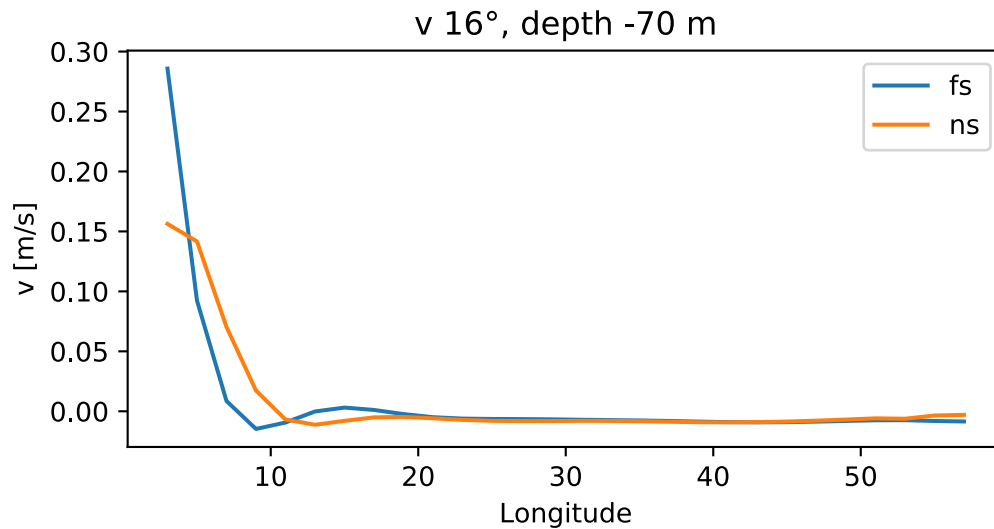


Figure 7: A comparison of the western boundary current for free slip, blue, and no slip, orange. The value of  $v$  for the free slip case looks like Munks solution, having maximum close to the boundary, and dropping below zero a short distance away. In the no slip case the velocity is way lower, but not zero since we only have values 1 degree away from the boundary.

force  $f_0$  and the geostrophic balance would have been nice to test.

A thing one could add to Veros was to make the parameter controlling the amount of slip to be adjustable, as in [7], from zero and upwards. 0 corresponds to free slip and at 2 we get no slip.

The aim of the project was to add the option of no slip boundary conditions to Veros. This has been done. With no slip enabled the tangential velocity, half a grid-spacing away, at the boundary is greatly reduced.

## Acknowledgements

This thesis wouldn't have seen the light of the day if it wasn't for the supervision of Markus Jochum, and the help I received from the people affiliated with Veros, Dion Häfner and Roman Nuterman.

## References

- [1] Dion Häfner et al. “Veros v0.1 – a Fast and Versatile Ocean Simulator in Pure Python”. Unpublished, review at: <https://www.geosci-model-dev-discuss.net/gmd-2018-3/>. 2018.
- [2] Dion Häfner. *Veros: The Versatile Ocean Simulator in Pure Python*. 2018. URL: <https://github.com/dionhaefner/veros> (visited on 04/24/2018).
- [3] Carsten Eden. *Python Ocean Model 2.2 (pyOM2) wiki*. 2017. URL: <https://wiki.cen.uni-hamburg.de/ifm/TO/pyOM2/> (visited on 06/01/2018).
- [4] Carsten Eden and Dirk Olbers. “An Energy Compartment Model for Propagation, Nonlinear Interaction, and Dissipation of Internal Gravity Waves”. In: *Journal of Physical Oceanography* 44 (Aug. 2014), pp. 2093–2106.
- [5] Joseph Pedlosky. *Ocean Circulation Theory*. 2nd. Springer, 1998.
- [6] Dion Häfner. *Veros: ReadTheDocs*. 2018. URL: <https://veros.readthedocs.io/> (visited on 04/24/2018).
- [7] Nemo European Consortium. *Nemo Docs: Boundary Condition at the Coast*. 2017. URL: <https://www.nemo-ocean.eu/doc/node58.html> (visited on 06/01/2018).
- [8] ROMS. *MyROMS Wiki: Numerical Solution Technique*. 2016. URL: [https://www.myroms.org/wiki/Numerical\\_Solution\\_Technique](https://www.myroms.org/wiki/Numerical_Solution_Technique) (visited on 06/01/2018).
- [9] Mads R. B. Kristensen et al. “Bohrium: Unmodified NumPy Code on CPU, GPU, and Cluster”. In: *Python for High Performance and Scientific Computing (PyHPC)* (2013).



## Apx. A; Veros friction.py

The lines I added was: {255 – 259, 265 – 269, 302 – 306, 314 – 318, 362 – 366, 382 – 386, 420 – 424, 439 – 443}

---

```
1 import math
2
3 from .. import veros_method
4 from . import numerics, utilities, cyclic
5
6
7 @veros_method
8 def harmonic_friction(vs):
9     """
10     horizontal harmonic friction
11     dissipation is calculated and added to K_diss_h
12     """
13     diss = np.zeros((vs.nx + 4, vs.ny + 4, vs.nz), dtype=vs.default_float_type)
14
15     """
16     Zonal velocity
17     """
18     if vs.enable_hor_friction_cos_scaling:
19         fxa = vs.cost**vs.hor_friction_cosPower
20         vs.flux_east[:-1] = vs.A_h * fxa[np.newaxis, :, np.newaxis] * (vs.u[1:,
21             :, vs.tau] - vs.u[:-1, :, :, vs.tau]) \
22             / (vs.cost * vs.dxt[1:, np.newaxis])[:, :, np.newaxis] *
23             vs.maskU[1:] * vs.maskU[:-1]
24         fxa = vs.cosu**vs.hor_friction_cosPower
25         vs.flux_north[:, :-1] = vs.A_h * fxa[np.newaxis, :-1, np.newaxis] *
26             (vs.u[:, 1:, :, vs.tau] - vs.u[:, :-1, :, vs.tau]) \
27             / vs.dyu[np.newaxis, :-1, np.newaxis] * vs.maskU[:, 1:] *
28             vs.maskU[:, :-1] * vs.cosu[np.newaxis, :-1, np.newaxis]
29         if vs.enable_noslip_lateral:
30             vs.flux_north[:, :-1] += 2 * vs.A_h * fxa[np.newaxis, :-1,
31                 np.newaxis] * (vs.u[:, 1:, :, vs.tau]) \
32                 / vs.dyu[np.newaxis, :-1, np.newaxis] * vs.maskU[:, 1:] * (1 -
33                 vs.maskU[:, :-1]) * vs.cosu[np.newaxis, :-1, np.newaxis] \
34                 - 2 * vs.A_h * fxa[np.newaxis, :-1, np.newaxis] * (vs.u[:, :-1,
35                 :, vs.tau]) \
36                 / vs.dyu[np.newaxis, :-1, np.newaxis] * (1 - vs.maskU[:, 1:]) *
37                 vs.maskU[:, :-1] * vs.cosu[np.newaxis, :-1, np.newaxis]
38     else:
39         vs.flux_east[:-1, :, :] = vs.A_h * (vs.u[1:, :, :, vs.tau] - vs.u[:-1,
40             :, :, vs.tau]) \
41             / (vs.cost * vs.dxt[1:, np.newaxis])[:, :, np.newaxis] *
42             vs.maskU[1:] * vs.maskU[:-1]
43         vs.flux_north[:, :-1, :] = vs.A_h * (vs.u[:, 1:, :, vs.tau] - vs.u[:,
44             :-1, :, vs.tau]) \
```

```

34         / vs.dyu[np.newaxis, :-1, np.newaxis] * vs.maskU[:, 1:] *
           vs.maskU[:, :-1] * vs.cosu[np.newaxis, :-1, np.newaxis]
35     if vs.enable_noslip_lateral:
36         vs.flux_north[:, :-1] += 2 * vs.A_h * vs.u[:, 1:, :, vs.tau] /
           vs.dyu[np.newaxis, :-1, np.newaxis] \
37         * vs.maskU[:, 1:] * (1 - vs.maskU[:, :-1]) * vs.cosu[np.newaxis,
           :-1, np.newaxis] \
38         - 2 * vs.A_h * vs.u[:, :-1, :, vs.tau] / vs.dyu[np.newaxis, :-1,
           np.newaxis] \
39         * (1 - vs.maskU[:, 1:]) * vs.maskU[:, :-1] * vs.cosu[np.newaxis,
           :-1, np.newaxis]
40
41     vs.flux_east[-1, :, :] = 0.
42     vs.flux_north[:, -1, :] = 0.
43
44     """
45     update tendency
46     """
47     vs.du_mix[2:-2, 2:-2, :] += vs.maskU[2:-2, 2:-2] * ((vs.flux_east[2:-2,
           2:-2] - vs.flux_east[1:-3, 2:-2])
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
           / (vs.cost[2:-2] *
             vs.dxu[2:-2,
                 np.newaxis])[ :, :,
                 np.newaxis]
           + (vs.flux_north[2:-2,
                 2:-2] -
             vs.flux_north[2:-2,
                 1:-3])
           / (vs.cost[2:-2] *
             vs.dyt[2:-2])[np.newaxis,
                 :, np.newaxis])
51
52     if vs.enable_conserve_energy:
53         """
54         diagnose dissipation by lateral friction
55         """
56         diss[1:-2, 2:-2] = 0.5 * ((vs.u[2:-1, 2:-2, :, vs.tau] - vs.u[1:-2,
           2:-2, :, vs.tau]) * vs.flux_east[1:-2, 2:-2]
57         + (vs.u[1:-2, 2:-2, :, vs.tau] - vs.u[:-3, 2:-2, :,
           vs.tau]) * vs.flux_east[:-3, 2:-2]) \
58         / (vs.cost[2:-2] * vs.dxu[1:-2, np.newaxis])[ :, :, np.newaxis] \
59         + 0.5 * ((vs.u[1:-2, 3:-1, :, vs.tau] - vs.u[1:-2, 2:-2, :, vs.tau])
           * vs.flux_north[1:-2, 2:-2]
60         + (vs.u[1:-2, 2:-2, :, vs.tau] - vs.u[1:-2, 1:-3, :, vs.tau])
           * vs.flux_north[1:-2, 1:-3]) \
61         / (vs.cost[2:-2] * vs.dyt[2:-2])[np.newaxis, :, np.newaxis]
62     vs.K_diss_h[...] = 0.
63     vs.K_diss_h[...] += numerics.calc_diss(vs, diss, 'U')
64

```

```

65     """
66     Meridional velocity
67     """
68     if vs.enable_hor_friction_cos_scaling:
69         fxa = (vs.cosu ** vs.hor_friction_cosPower) * np.ones((vs.nx + 3, 1),
70             dtype=vs.default_float_type)
71         vs.flux_east[:-1] = vs.A_h * fxa[:, :, np.newaxis] * (vs.v[1:, :, :,
72             vs.tau] - vs.v[:-1, :, :, vs.tau]) \
73             / (vs.cosu * vs.dxu[:-1, np.newaxis])[:, :, np.newaxis] *
74             vs.maskV[1:] * vs.maskV[:-1]
75         if vs.enable_noslip_lateral:
76             vs.flux_east[:-1] += 2 * vs.A_h * fxa[:, :, np.newaxis] * vs.v[1:,
77                 :, :, vs.tau] \
78                 / (vs.cosu * vs.dxu[:-1, np.newaxis])[:, :, np.newaxis] *
79                 vs.maskV[1:] * (1 - vs.maskV[:-1]) \
80                 - 2 * vs.A_h * fxa[:, :, np.newaxis] * vs.v[:-1, :, :, vs.tau] \
81                 / (vs.cosu * vs.dxu[:-1, np.newaxis])[:, :, np.newaxis] * (1 -
82                 vs.maskV[1:]) * vs.maskV[:-1]
83
84         fxa = (vs.cost[1:] ** vs.hor_friction_cosPower) * np.ones((vs.nx + 4,
85             1), dtype=vs.default_float_type)
86         vs.flux_north[:, :-1] = vs.A_h * fxa[:, :, np.newaxis] * (vs.v[:, 1:,
87             :, vs.tau] - vs.v[:, :-1, :, vs.tau]) \
88             / vs.dyt[np.newaxis, 1:, np.newaxis] * vs.cost[np.newaxis, 1:,
89                 np.newaxis] * vs.maskV[:, :-1] * vs.maskV[:, 1:]
90     else:
91         vs.flux_east[:-1] = vs.A_h * (vs.v[1:, :, :, vs.tau] - vs.v[:-1, :, :,
92             vs.tau]) \
93             / (vs.cosu * vs.dxu[:-1, np.newaxis])[:, :, np.newaxis] *
94             vs.maskV[1:] * vs.maskV[:-1]
95         if vs.enable_noslip_lateral:
96             vs.flux_east[:-1] += 2 * vs.A_h * vs.v[1:, :, :, vs.tau] / (vs.cosu
97                 * vs.dxu[:-1, np.newaxis])[:, :, np.newaxis] \
98                 * vs.maskV[1:] * (1 - vs.maskV[:-1]) \
99                 - 2 * vs.A_h * vs.v[:-1, :, :, vs.tau] / (vs.cosu * vs.dxu[:-1,
100                 np.newaxis])[:, :, np.newaxis] \
101                 * (1 - vs.maskV[1:]) * vs.maskV[:-1]
102         vs.flux_north[:, :-1] = vs.A_h * (vs.v[:, 1:, :, vs.tau] - vs.v[:, :-1,
103             :, vs.tau]) \
104             / vs.dyt[np.newaxis, 1:, np.newaxis] * vs.cost[np.newaxis, 1:,
105                 np.newaxis] * vs.maskV[:, :-1] * vs.maskV[:, 1:]
106     vs.flux_east[-1, :, :] = 0.
107     vs.flux_north[:, -1, :] = 0.
108
109     """
110     update tendency
111     """
112     vs.dv_mix[2:-2, 2:-2] += vs.maskV[2:-2, 2:-2] * ((vs.flux_east[2:-2, 2:-2]
113         - vs.flux_east[1:-3, 2:-2])

```

```

98         / (vs.cosu[2:-2] * vs.dxt[2:-2,
99             np.newaxis])[:, :, np.newaxis]
100     + (vs.flux_north[2:-2, 2:-2] -
        vs.flux_north[2:-2, 1:-3])
101     / (vs.dyu[2:-2] *
        vs.cosu[2:-2])[np.newaxis, :,
        np.newaxis])
102
103     if vs.enable_conserve_energy:
104         """
105         diagnose dissipation by lateral friction
106         """
107         diss[2:-2, 1:-2] = 0.5 * ((vs.v[3:-1, 1:-2, :, vs.tau] - vs.v[2:-2,
108             1:-2, :, vs.tau]) * vs.flux_east[2:-2, 1:-2]
109             + (vs.v[2:-2, 1:-2, :, vs.tau] - vs.v[1:-3, 1:-2,
110             :, vs.tau]) * vs.flux_east[1:-3, 1:-2]) \
111         / (vs.cosu[1:-2] * vs.dxt[2:-2, np.newaxis])[:, :, np.newaxis] \
112         + 0.5 * ((vs.v[2:-2, 2:-1, :, vs.tau] - vs.v[2:-2, 1:-2, :, vs.tau])
113             * vs.flux_north[2:-2, 1:-2]
114             + (vs.v[2:-2, 1:-2, :, vs.tau] - vs.v[2:-2, :-3, :, vs.tau]) *
115             vs.flux_north[2:-2, :-3]) \
116         / (vs.cosu[1:-2] * vs.dyu[1:-2])[np.newaxis, :, np.newaxis]
117         vs.K_diss_h[...] += numerics.calc_diss(vs, diss, 'V')
118
119     @veros_method
120     def biharmonic_friction(vs):
121         """
122         horizontal biharmonic friction
123         dissipation is calculated and added to K_diss_h
124         """
125         fxa = math.sqrt(abs(vs.A_hbi))
126
127         """
128         Zonal velocity
129         """
130         vs.flux_east[:-1, :, :] = fxa * (vs.u[1:, :, :, vs.tau] - vs.u[:-1, :, :,
131             vs.tau]) \
132         / (vs.cost[np.newaxis, :, np.newaxis] * vs.dxt[1:, np.newaxis,
133             np.newaxis]) \
134         * vs.maskU[1:, :, :] * vs.maskU[:-1, :, :]
135         vs.flux_north[:, :-1, :] = fxa * (vs.u[:, 1:, :, vs.tau] - vs.u[:, :-1, :,
136             vs.tau]) \
137         / vs.dyu[np.newaxis, :-1, np.newaxis] * vs.maskU[:, 1:, :] \
138         * vs.maskU[:, :-1, :] * vs.cosu[np.newaxis, :-1, np.newaxis]
139         if vs.enable_noslip_lateral:
140             vs.flux_north[:, :-1] += 2 * fxa * vs.u[:, 1:, :, vs.tau] /
141             vs.dyu[np.newaxis, :-1, np.newaxis] \
142             * vs.maskU[:, 1:] * (1 - vs.maskU[:, :-1]) * vs.cosu[np.newaxis,

```

```

135         :-1, np.newaxis]\
- 2 * fxa * vs.u[:, :-1, :, vs.tau] / vs.dyu[np.newaxis, :-1,
    np.newaxis] \
136     * (1 - vs.maskU[:, 1:]) * vs.maskU[:, :-1] * vs.cosu[np.newaxis,
    :-1, np.newaxis]
137 vs.flux_east[-1, :, :] = 0.
138 vs.flux_north[:, -1, :] = 0.
139
140 del2 = np.zeros((vs.nx + 4, vs.ny + 4, vs.nz), dtype=vs.default_float_type)
141 del2[1:, 1:, :] = (vs.flux_east[1:, 1:, :] - vs.flux_east[:-1, 1:, :]) \
142     / (vs.cost[np.newaxis, 1:, np.newaxis] * vs.dxu[1:, np.newaxis,
    np.newaxis]) \
143     + (vs.flux_north[1:, 1:, :] - vs.flux_north[1:, :-1, :]) \
144     / (vs.cost[np.newaxis, 1:, np.newaxis] * vs.dyt[np.newaxis, 1:,
    np.newaxis])
145
146 vs.flux_east[:-1, :, :] = fxa * (del2[1:, :, :] - del2[:-1, :, :]) \
147     / (vs.cost[np.newaxis, :, np.newaxis] * vs.dxt[1:, np.newaxis,
    np.newaxis]) \
148     * vs.maskU[1:, :, :] * vs.maskU[:-1, :, :]
149 vs.flux_north[:, :-1, :] = fxa * (del2[:, 1:, :] - del2[:, :-1, :]) \
150     / vs.dyu[np.newaxis, :-1, np.newaxis] * vs.maskU[:, 1:, :] \
151     * vs.maskU[:, :-1, :] * vs.cosu[np.newaxis, :-1, np.newaxis]
152 if vs.enable_noslip_lateral:
153     vs.flux_north[:, :-1, :] += 2 * fxa * del2[:, 1:, :] / vs.dyu[np.newaxis,
    :-1, np.newaxis] \
154     * vs.maskU[:, 1:, :] * (1 - vs.maskU[:, :-1, :]) *
    vs.cosu[np.newaxis, :-1, np.newaxis] \
155     - 2 * fxa * del2[:, :-1, :] / vs.dyu[np.newaxis, :-1, np.newaxis] \
156     * (1 - vs.maskU[:, 1:, :]) * vs.maskU[:, :-1, :] *
    vs.cosu[np.newaxis, :-1, np.newaxis]
157 vs.flux_east[-1, :, :] = 0.
158 vs.flux_north[:, -1, :] = 0.
159
160 """
161 update tendency
162 """
163 vs.du_mix[2:-2, 2:-2, :] += -vs.maskU[2:-2, 2:-2, :] * ((vs.flux_east[2:-2,
    2:-2, :] - vs.flux_east[1:-3, 2:-2, :])
164
    / (vs.cost[np.newaxis, 2:-2,
    np.newaxis] *
    vs.dxu[2:-2, np.newaxis,
    np.newaxis])
165
    + (vs.flux_north[2:-2, 2:-2,
    :] - vs.flux_north[2:-2,
    1:-3, :])
166
    / (vs.cost[np.newaxis, 2:-2,
    np.newaxis] *
    vs.dyt[np.newaxis, 2:-2,

```

```

                                                    np.newaxis]))
167 if vs.enable_conserve_energy:
168     """
169     diagnose dissipation by lateral friction
170     """
171     if vs.enable_cyclic_x:
172         cyclic.setcyclic_x(vs.flux_east)
173         cyclic.setcyclic_x(vs.flux_north)
174         diss = np.zeros((vs.nx + 4, vs.ny + 4, vs.nz),
175                         dtype=vs.default_float_type)
176         diss[1:-2, 2:-2, :] = -0.5 * ((vs.u[2:-1, 2:-2, :, vs.tau] - vs.u[1:-2,
177                                     2:-2, :, vs.tau]) * vs.flux_east[1:-2, 2:-2, :]
178                                     + (vs.u[1:-2, 2:-2, :, vs.tau] - vs.u[:-3,
179                                         2:-2, :, vs.tau]) * vs.flux_east[:-3, 2:-2,
180                                         :]) \
181                                     / (vs.cost[np.newaxis, 2:-2, np.newaxis] * vs.dxu[1:-2, np.newaxis,
182                                         np.newaxis]) \
183                                     - 0.5 * ((vs.u[1:-2, 3:-1, :, vs.tau] - vs.u[1:-2, 2:-2, :, vs.tau])
184                                     * vs.flux_north[1:-2, 2:-2, :]
185                                     + (vs.u[1:-2, 2:-2, :, vs.tau] - vs.u[1:-2, 1:-3, :, vs.tau])
186                                     * vs.flux_north[1:-2, 1:-3, :]) \
187                                     / (vs.cost[np.newaxis, 2:-2, np.newaxis] * vs.dyt[np.newaxis, 2:-2,
188                                         np.newaxis])
189         vs.K_diss_h[...] = 0.
190         vs.K_diss_h[...] += numerics.calc_diss(vs, diss, 'U')
191
192     """
193     Meridional velocity
194     """
195     vs.flux_east[:-1, :, :] = fxa * (vs.v[1:, :, :, vs.tau] - vs.v[:-1, :, :,
196                                     vs.tau]) \
197     / (vs.cosu[np.newaxis, :, np.newaxis] * vs.dxu[:-1, np.newaxis,
198         np.newaxis]) \
199     * vs.maskV[1:, :, :] * vs.maskV[:-1, :, :]
200     if vs.enable_noslip_lateral:
201         vs.flux_east[:-1, :, :] += 2 * fxa * vs.v[1:, :, :, vs.tau] /
202         (vs.cosu[np.newaxis, :, np.newaxis] * vs.dxu[:-1, np.newaxis,
203             np.newaxis]) \
204         * vs.maskV[1:, :, :] * (1 - vs.maskV[:-1, :, :]) \
205         - 2 * fxa * vs.v[:-1, :, :, vs.tau] / (vs.cosu[np.newaxis, :,
206             np.newaxis] * vs.dxu[:-1, np.newaxis, np.newaxis]) \
207         * (1 - vs.maskV[1:, :, :]) * vs.maskV[:-1, :, :]
208     vs.flux_north[:, :-1, :] = fxa * (vs.v[:, 1:, :, vs.tau] - vs.v[:, :-1, :,
209                                     vs.tau]) \
210     / vs.dyt[np.newaxis, 1:, np.newaxis] * vs.cost[np.newaxis, 1:,
211         np.newaxis] \
212     * vs.maskV[:, :-1, :] * vs.maskV[:, 1:, :]
213     vs.flux_east[-1, :, :] = 0.
214     vs.flux_north[:, -1, :] = 0.

```

```

200
201 del2[1:, 1:, :] = (vs.flux_east[1:, 1:, :] - vs.flux_east[:-1, 1:, :]) \
202     / (vs.cosu[np.newaxis, 1:, np.newaxis] * vs.dxt[1:, np.newaxis,
203         np.newaxis]) \
204     + (vs.flux_north[1:, 1:, :] - vs.flux_north[1:, :-1, :]) \
205     / (vs.dyu[np.newaxis, 1:, np.newaxis] * vs.cosu[np.newaxis, 1:,
206         np.newaxis])
207
208 vs.flux_east[:-1, :, :] = fxa * (del2[1:, :, :] - del2[:-1, :, :]) \
209     / (vs.cosu[np.newaxis, :, np.newaxis] * vs.dxu[:-1, np.newaxis,
210         np.newaxis]) \
211     * vs.maskV[1:, :, :] * vs.maskV[:-1, :, :]
212
213 if vs.enable_noslip_lateral:
214     vs.flux_east[:-1, :, :] += 2 * fxa * del2[1:, :, :] /
215         (vs.cosu[np.newaxis, :, np.newaxis] * vs.dxu[:-1, np.newaxis,
216             np.newaxis]) \
217         * vs.maskV[1:, :, :] * (1 - vs.maskV[:-1, :, :]) \
218         - 2 * fxa * del2[:-1, :, :] / (vs.cosu[np.newaxis, :, np.newaxis] *
219             vs.dxu[:-1, np.newaxis, np.newaxis]) \
220         * (1 - vs.maskV[1:, :, :]) * vs.maskV[:-1, :, :]
221
222 vs.flux_north[:, :-1, :] = fxa * (del2[:, 1:, :] - del2[:, :-1, :]) \
223     / vs.dyt[np.newaxis, 1:, np.newaxis] * vs.cost[np.newaxis, 1:,
224         np.newaxis] \
225     * vs.maskV[:, :-1, :] * vs.maskV[:, 1:, :]
226
227 vs.flux_east[-1, :, :] = 0.
228 vs.flux_north[:, -1, :] = 0.
229
230 """
231
232 update tendency
233 """
234
235 vs.dv_mix[2:-2, 2:-2, :] += -vs.maskV[2:-2, 2:-2, :] * ((vs.flux_east[2:-2,
236     2:-2, :] - vs.flux_east[1:-3, 2:-2, :])
237
238     / (vs.cosu[np.newaxis,
239         2:-2, np.newaxis] *
240         vs.dxt[2:-2,
241             np.newaxis, np.newaxis])
242
243     + (vs.flux_north[2:-2,
244         2:-2, :] -
245         vs.flux_north[2:-2,
246             1:-3, :])
247
248     / (vs.dyu[np.newaxis, 2:-2,
249         np.newaxis] *
250         vs.cosu[np.newaxis,
251             2:-2, np.newaxis]))
252
253
254 if vs.enable_conserve_energy:
255     """
256
257     diagnose dissipation by lateral friction
258     """

```

```

232     if vs.enable_cyclic_x:
233         cyclic.setcyclic_x(vs.flux_east)
234         cyclic.setcyclic_x(vs.flux_north)
235     diss[2:-2, 1:-2, :] = -0.5 * ((vs.v[3:-1, 1:-2, :, vs.tau] - vs.v[2:-2,
236         1:-2, :, vs.tau]) * vs.flux_east[2:-2, 1:-2, :]
                + (vs.v[2:-2, 1:-2, :, vs.tau] - vs.v[1:-3,
                1:-2, :, vs.tau]) * vs.flux_east[1:-3, 1:-2,
                :]) \
237     / (vs.cosu[np.newaxis, 1:-2, np.newaxis] * vs.dxt[2:-2, np.newaxis,
        np.newaxis]) \
238     - 0.5 * ((vs.v[2:-2, 2:-1, :, vs.tau] - vs.v[2:-2, 1:-2, :, vs.tau])
        * vs.flux_north[2:-2, 1:-2, :]
239         + (vs.v[2:-2, 1:-2, :, vs.tau] - vs.v[2:-2, :-3, :, vs.tau]) *
        vs.flux_north[2:-2, :-3, :]) \
240     / (vs.cosu[np.newaxis, 1:-2, np.newaxis] * vs.dyu[np.newaxis, 1:-2,
        np.newaxis])
241     vs.K_diss_h[...] += numerics.calc_diss(vs, diss, 'V')

```

---



## Apx. B; ACC model setup

Here I have only included the model once. The only difference between my two runs was the value of enable noslip lateral.

---

```
1 #!/usr/bin/env python
2
3 import veros
4 import veros.tools
5
6
7 class ACC(veros.Veros):
8     """A model using spherical coordinates with a partially closed domain
9         representing the Atlantic and ACC.
10
11     Wind forcing over the channel part and buoyancy relaxation drive a
12     large-scale meridional overturning circulation.
13
14     This setup demonstrates:
15     - setting up an idealized geometry
16     - updating surface forcings
17     - basic usage of diagnostics
18
19     `Adapted from pyOM2
20     <https://wiki.cen.uni-hamburg.de/ifm/TO/pyOM2/ACC%202020>`.
21     """
22
23     @veros.veros_method
24     def set_parameter(self):
25         self.identifier = "acc_ns"
26
27         self.nx, self.ny, self.nz = 30, 42, 15
28         self.dt_mom = 4800
29         self.dt_tracer = 86400 / 2.
30         self.runlen = 86400 * 365 * 100
31
32         self.coord_degree = True
33         self.enable_cyclic_x = True
34
35         self.congr_epsilon = 1e-12
36         self.congr_max_iterations = 5000
37
38         self.enable_neutral_diffusion = True
39         self.K_iso_0 = 1000.0
40         self.K_iso_steep = 500.0
41         self.iso_dslope = 0.005
42         self.iso_slopec = 0.01
43         self.enable_skew_diffusion = True
44
45         self.enable_noslip_lateral = True
```

```

42
43     self.enable_hor_friction = True
44     self.A_h = (2 * self.degtom)**3 * 2e-11
45     self.enable_hor_friction_cos_scaling = True
46     self.hor_friction_cosPower = 1
47
48     self.enable_bottom_friction = True
49     self.r_bot = 1e-5
50
51     self.enable_implicit_vert_friction = True
52
53     self.enable_tke = True
54     self.c_k = 0.1
55     self.c_eps = 0.7
56     self.alpha_tke = 30.0
57     self.mxl_min = 1e-8
58     self.tke_mxl_choice = 2
59     # self.enable_tke_superbee_advection = True
60
61     self.K_gm_0 = 1000.0
62     self.enable_eke = True
63     self.eke_k_max = 1e4
64     self.eke_c_k = 0.4
65     self.eke_c_eps = 0.5
66     self.eke_cross = 2.
67     self.eke_crhin = 1.0
68     self.eke_lmin = 100.0
69     self.enable_eke_superbee_advection = True
70     self.enable_eke_isopycnal_diffusion = True
71
72     self.enable_idemix = True
73     self.enable_idemix_hor_diffusion = True
74     self.enable_eke_diss_surfbot = True
75     self.eke_diss_surfbot_frac = 0.2
76     self.enable_idemix_superbee_advection = True
77
78     self.eq_of_state_type = 3
79
80     @veros.veros_method
81     def set_grid(self):
82         ddz = np.array([50., 70., 100., 140., 190., 240., 290., 340.,
83                        390., 440., 490., 540., 590., 640., 690.])
84         self.dxt[...] = 2.0
85         self.dyt[...] = 2.0
86         self.x_origin = 0.0
87         self.y_origin = -40.0
88         self.dzt[...] = ddz[::-1] / 2.5
89
90     @veros.veros_method

```

```

91     def set_coriolis(self):
92         self.coriolis_t[:, :] = 2 * self.omega * np.sin(self.yt[None, :] / 180.
          * self.pi)
93
94     @veros.veros_method
95     def set_topography(self):
96         x, y = np.meshgrid(self.xt, self.yt, indexing="ij")
97         self.kbot[...] = np.logical_or(x > 1.0, y < -20).astype(np.int)
98
99     @veros.veros_method
100    def set_initial_conditions(self):
101        # initial conditions
102        self.temp[:, :, :, 0:2] = ((1 - self.zt[None, None, :] / self.zw[0]) *
          15 * self.maskT)[..., None]
103        self.salt[:, :, :, 0:2] = 35.0 * self.maskT[..., None]
104
105        # wind stress forcing
106        tau_x = np.zeros(self.ny + 4, dtype=self.default_float_type)
107        tau_x[self.yt < -20] = 1e-4 * np.sin(self.pi * (self.yu[self.yt < -20] -
          self.yu.min()) / (-20.0 - self.yt.min()))
108        tau_x[self.yt > 10] = 1e-4 * (1 - np.cos(2 * self.pi * (self.yu[self.yt
          > 10] - 10.0) / (self.yu.max() - 10.0)))
109        self.surface_tau_x[:, :] = tau_x * self.maskU[:, :, -1]
110
111        # surface heatflux forcing
112        self._t_star = 15 * np.ones(self.ny + 4, dtype=self.default_float_type)
113        self._t_star[self.yt < -20] = 15 * (self.yt[self.yt < -20] -
          self.yt.min()) / (-20 - self.yt.min())
114        self._t_star[self.yt > 20] = 15 * (1 - (self.yt[self.yt > 20] - 20) /
          (self.yt.max() - 20))
115        self._t_rest = self.dzt[None, -1] / (30. * 86400.) * self.maskT[:, :,
          -1]
116
117        if self.enable_tke:
118            self.forc_tke_surface[2:-2, 2:-2] = np.sqrt((0.5 *
          (self.surface_tau_x[2:-2, 2:-2] + self.surface_tau_x[1:-3,
          2:-2]))**2
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999

```

```

127     self.forc_temp_surface[...] = self._t_rest * (self._t_star -
128         self.temp[:, :, -1, self.tau])
129
130     @veros.veros_method
131     def set_diagnostics(self):
132         self.diagnostics["snapshot"].output_frequency = 86400 * 365
133         self.diagnostics["averages"].output_variables = (
134             "salt", "temp", "u", "v", "w", "psi", "surface_taux", "surface_tauy"
135         )
136         self.diagnostics["averages"].output_frequency = 100 * 86400.
137         self.diagnostics["averages"].sampling_frequency = self.dt_tracer * 10
138         self.diagnostics["overturning"].output_frequency = 365 * 86400. / 48.
139         self.diagnostics["overturning"].sampling_frequency = self.dt_tracer * 10
140         self.diagnostics["tracer_monitor"].output_frequency = 365 * 86400. / 12.
141         self.diagnostics["energy"].output_frequency = 365 * 86400. / 48
142         self.diagnostics["energy"].sampling_frequency = self.dt_tracer * 10
143
144     def after_timestep(self):
145         pass
146
147     @veros.tools.cli
148     def run(*args, **kwargs):
149         simulation = ACC(*args, **kwargs)
150         simulation.setup()
151         simulation.run()
152
153     if __name__ == "__main__":
154         run()
155

```

---

## Apx. C; Code for Plots

```

1  import numpy as np
2  import matplotlib.pyplot as plt
3  from netCDF4 import Dataset
4
5  def theory(_save=False):
6      """
7      The theoretical western boundary current.
8      """
9      deltam = 1
10     x = np.arange(0,100,0.1)
11     # No slip
12     v1 = np.exp(-x/(2*deltam))* np.sin(np.sqrt(3)*x/(2*deltam))
13     # Free slip

```

```

14     v2 = np.exp(-x/(2*deltam))*(np.cos(np.sqrt(3)*x/(2*deltam)) + 1/np.sqrt(3)*
15         np.sin(np.sqrt(3)*x/(2*deltam)) )
16     fig, ax = plt.subplots(1,1,figsize = (3.2,2.9))
17     ax.plot(x,v1,label='No slip')
18     ax.plot(x,v2, 'r',label='Free slip')
19     ax.plot(x,[0]*len(x), '--',color='gray',label='')
20     ax.legend()
21     ax.set_xlim([0,20])
22     ax.set_xlabel(r'x/  $\delta_m$ ')
23     ax.set_ylabel(r' $v/ \psi_I$ ')
24     #plt.show()
25     if _save:
26         plt.savefig('theory.pdf')
27     return(fig)
28
29 def forcing(_save =False):
30     """
31     A plot of the wind force in respectively the x and y direction.
32     """
33     vmax_tot = np.amax([np.amax(surface_tau_x),np.amax(surface_tau_y)])
34     vmin_tot = np.amin([np.amin(surface_tau_x),np.amin(surface_tau_y)])
35
36     fig, ax = plt.subplots(1,2,figsize = (5.5,2.9))
37
38     viz = ax[0].imshow(surface_tau_x[-1,:,:], extent =
39         [xu[0],xu[-1],yt[0],yt[-1]], origin = 'lower', interpolation='nearest',
40         vmin = vmin_tot, vmax = vmax_tot)
41     ax[0].set_xlabel("Longitude")
42     ax[0].set_title("Surface tau x [m2/s] ")
43     ax[0].set_ylabel("Latitude")
44     tau_y=ax[1].imshow(surface_tau_y[-1,:,:], extent =
45         [xu[0],xu[-1],yt[0],yt[-1]], origin = 'lower', interpolation='nearest',
46         vmin = vmin_tot, vmax = vmax_tot)
47     ax[1].set_xlabel("Longitude")
48     ax[1].set_title("Surface tau y [m2/s]")
49     fig.tight_layout()
50     fig.subplots_adjust(right=0.75)
51
52     cbar_ax = fig.add_axes([0.80, 0.20, 0.05, 0.68])
53     fig.colorbar(tau_y, cax = cbar_ax)
54
55     if _save:
56         plt.savefig('forcing.pdf')
57     return fig
58
59 def time_series(_time_bound,_save=False):
60     """
61     The ACC integrated and plotted as a time series to show convergence.

```

```

57     And the value of the velocity of the western boundary current at the
        boundary.
58     """
59     # For the ACC
60     # Cross area.
61     delta_z = -np.append((zw[0]-zw[1])-20,(zw[:-1] - zw[1:]))
62     radius = 6370e3 # Earth radius in m
63     degtom = radius / 180. * np.pi # Conversion degrees latitude to meter
64     delta_y = np.append(-42.0,yu)
65     cosu = np.cos(delta_y * np.pi / 180.)
66     delta_y = - (delta_y[:-1]*cosu[:-1] - delta_y[1:]*cosu[1:])*degtom
67     area_u = delta_z[:,np.newaxis]*delta_y[np.newaxis,:]
68
69     cut_off_south = 11
70     cur = np.mean(u[:, :, :cut_off_south, 5:25], axis=3)
71     cur_ns = np.mean(u_ns[:, :, :cut_off_south, 5:25], axis=3)
72     cur_time_series = np.sum(np.sum(cur*area_u[np.newaxis, :, :cut_off_south],
        axis=2),axis=1) /1000000
73     cur_time_series_ns =
        np.sum(np.sum(cur_ns*area_u[np.newaxis, :, :cut_off_south],
        axis=2),axis=1)/1000000
74
75     # For the WBC.
76     lower = 28
77     depth = -3
78     first_idx_for_wbc= 2
79
80     fig,axes = plt.subplots(1,2, figsize = (5.5,2.9))
81
82     axes[0].plot(Time[:_time_bound[1]],cur_time_series[:_time_bound[1]], label
        = 'fs')
83     axes[0].plot(Time_ns[:_time_bound[1]],cur_time_series_ns[:_time_bound[1]],
        label = 'ns')
84     axes[0].set_title('ACC integrated')
85     axes[0].set_xlabel('Time [days]')
86     axes[0].set_ylabel('Transport [Sv]')
87     axes[0].legend()
88
89     axes[1].plot(Time[:_time_bound[1]],
        v[:_time_bound[1],depth,lower,first_idx_for_wbc], label = 'fs')
90     axes[1].plot(Time_ns[:_time_bound[1]],
        v_ns[:_time_bound[1],depth,lower,first_idx_for_wbc], label = 'ns')
91     axes[1].set_title('v '+str(yu[lower])+' deg, depth ' +str(zt[depth])+' m')
92     axes[1].set_xlabel('Time [days]')
93     axes[1].set_ylabel('v [m/s]')
94     axes[1].legend()
95
96     fig.tight_layout()
97

```

```

98     if _save:
99         plt.savefig('AAC_and_WBC_ts.pdf')
100    return(fig)
101
102    def acc_comparison(_time_bound,_save=False):
103        """
104        A comparison of the structure of the ACC.
105        """
106        cut_off_south = 11
107        cur =
108            np.mean(np.sum(u[_time_bound[0]:_time_bound[1],:,:cut_off_south,5:25],
109                        axis=3) / 20.0, axis = 0)
110        cur_ns =
111            np.mean(np.sum(u_ns[_time_bound[0]:_time_bound[1],:,:cut_off_south,5:25],
112                        axis=3) / 20.0, axis = 0)
113
114        vmax_tot = np.amax([np.amax(cur),np.amax(cur_ns)])
115        vmin_tot = np.amin([np.amin(cur),np.amin(cur_ns)])
116
117        fig_cur, ax_cur = plt.subplots(1,2,figsize = (5.5,2.9))
118
119        con = ax_cur[0].contour(yt[0:11],zt/1000.0, cur, vmin = vmin_tot, vmax =
120            vmax_tot)
121        con_ns = ax_cur[1].contour(yt[0:11],zt/1000.0, cur_ns, vmin = vmin_tot,
122            vmax = vmax_tot)
123
124        ax_cur[0].set_title('Free slip <u>')
125        ax_cur[1].set_title('No slip <u>')
126
127        ax_cur[0].set_ylabel("Depth [km]")
128        ax_cur[0].set_xlabel("Latitude [deg]")
129        ax_cur[1].set_xlabel("Latitude [deg]")
130
131        fig_cur.tight_layout()
132
133        fig_cur.subplots_adjust(right=0.8)
134        cbar_con = fig_cur.add_axes([0.85, 0.20, 0.05, 0.68])
135
136        fig_cur.colorbar(con, cax = cbar_con)
137
138    if _save:
139        plt.savefig('acc_comp.pdf')
140    return(fig_cur)
141
142    def west_boundary_current(_time_bound,_save=False):
143        """
144        A comparison of the western boundary current, to be compared with
145        theoretical solution.
146        """

```

```

140     lower = 28
141     depth = -3
142
143     fig_close, ax_close = plt.subplots(1,1, figsize = (5.5,3))
144
145     ax_close.plot(xt,np.mean(v[_time_bound[0]:_time_bound[1],depth,lower,:],axis=0),
146                  label = 'fs')
147     ax_close.plot(xt,np.mean(v_ns[_time_bound[0]:_time_bound[1],depth,lower,:],axis=0),
148                  label = 'ns')
149
150     ax_close.set_title('v '+str(int(yu[lower]))+'° , depth '
151                       +str(int(zt[depth]))+' m')
152     ax_close.set_ylabel('v [m/s]')
153     ax_close.set_xlabel('Longitude')
154     ax_close.legend()
155     fig_close.tight_layout()
156     if _save:
157         plt.savefig('westb_cur.pdf')
158     return(fig_close)
159
160 def psiplot_western_boundary(_time_bound,_save=False):
161     """
162     A plot of psi at the western boundary.
163     I cut of the ACC in order to be able to see the variations in the northern
164     basin.
165     """
166     index_from = 15
167     _psi_wa = np.mean(psi[_time_bound[0]:_time_bound[1],index_from:,:], axis =0)
168     _psin_wa = np.mean(psi_ns[_time_bound[0]:_time_bound[1],index_from:,:],
169                       axis =0)
170     vmax_psi = np.amax([np.amax(_psi_wa),np.amax(_psin_wa)])
171     vmin_psi = np.amin([np.amin(_psi_wa),np.amin(_psin_wa)])
172
173     fig, ax = plt.subplots(2,1,figsize = (5.5,5))
174
175     psip = ax[0].contour(xu,yu[index_from:],(_psi_wa), vmax = vmax_psi, vmin =
176                        vmin_psi)#, extent = [xu[0],xu[-1],yu[index_a],yu[-1]], origin =
177                        'lower')
178     ax[0].set_title('Free slip')
179     ax[0].set_ylabel("Latitude")
180
181     psinp = ax[1].contour(xu,yu[index_from:],(_psin_wa), vmax = vmax_psi, vmin
182                          = vmin_psi)#, extent = [xu[0],xu[-1],yu[index_a],yu[-1]], origin =
183                          'lower')
184     ax[1].set_title('No slip')
185     ax[1].set_ylabel("Latitude")
186     ax[1].set_xlabel("Longitude")
187
188     fig.tight_layout()

```



```

180 fig.subplots_adjust(right=0.8)
181
182 cbar_ax = fig.add_axes([0.85, 0.15, 0.05, 0.7])
183 fig.colorbar(psinp, cax = cbar_ax)
184 if _save:
185     plt.savefig('psi_comp.pdf')
186 return(fig)
187
188
189 if __name__ == '__main__':
190     with Dataset("acc_averages.nc", "r") as datafile:
191         # read variable "u" and save it to a NumPy array
192
193         xt = datafile.variables["xt"][...]
194         xu = datafile.variables["xu"][...]
195         yt = datafile.variables["yt"][...]
196         yu = datafile.variables["yu"][...]
197         zt = datafile.variables["zt"][...]
198         zw = datafile.variables["zw"][...]
199         temp = datafile.variables["temp"][...]
200         Time = datafile.variables["Time"][...]
201         surface_taux = datafile.variables["surface_taux"][...]
202         surface_tauy = datafile.variables["surface_tauy"][...]
203
204         u = datafile.variables["u"][...]
205         v = datafile.variables["v"][...]
206         psi = datafile.variables["psi"][...]
207
208
209     with Dataset("acc_ns_averages.nc", "r") as datafile:
210         # read variable "u" and save it to a NumPy array
211         Time_ns = datafile.variables["Time"][...]
212         u_ns = datafile.variables["u"][...]
213         v_ns = datafile.variables["v"][...]
214         psi_ns = datafile.variables["psi"][...]
215         # Due to unforeseen end of Dataset acc_ns I have created:
216         time_bound = [len(Time_ns)-10, len(Time_ns)-1]
217         #print(u[360,-1,6,6])
218         #print(np.abs(u[360,:,:,:]-u_ns[360,:,:,:]) < 0.0001)
219
220         theo = theory()
221         sf = forcing()
222         ts = time_series(time_bound)
223         acc_comp = acc_comparison(time_bound)
224         wbc= west_boundary_current(time_bound)
225         fig_psi_wa = psiplot_western_boundary(time_bound)
226         plt.show()

```

---

## Apx. D; Carsten Eden's implementation of no slip

```
1 #include "options.inc"
2
3
4
5     subroutine momentum_tendency
6 c=====
7 c     tendencies for momentum stored in F_u, F_v and F_w
8 c=====
9     use cpflame_module
10    implicit none
11    integer :: i,j,k,js,je
12    real :: adv_fe(imt,jmt,km), adv_ft(imt,jmt,km)
13    real :: adv_fn(imt,jmt,km), diff_fn(imt,jmt,km)
14    real :: diff_fe(imt,jmt,km),diff_ft(imt,jmt,km)
15    real :: fxa,fxb,fxc
16
17 #ifdef enable_smagorinsky_friction
18     call smagorinsky
19 #endif
20
21     js=max(2,js_pe); je = min(je_pe,jmt-1)
22 c-----
23 c     Zonal momentum equation: advective and diffusive fluxes
24 c-----
25     adv_fe(:,js_pe:je_pe,:)=0.0;
26     adv_fn(:,js_pe:je_pe,:)=0.0;
27     adv_ft(:,js_pe:je_pe,:)=0.0;
28     if (enable_4th_mom_advection) then
29         call setcyclic3D_j2(u(:, :, :, 1, tau) )
30         call adv_flux_u_4th(adv_fe,adv_fn,adv_ft)
31     elseif (enable_quicker_mom_advection) then
32         call setcyclic3D_j2(u(:, :, :, 1, taum1) )
33         call adv_flux_u_quicker(adv_fe,adv_fn,adv_ft)
34     else
35         call adv_flux_u_2nd(adv_fe,adv_fn,adv_ft)
36     endif
37
38     diff_fe(:,js_pe:je_pe,:)=0.0;
39     diff_fn(:,js_pe:je_pe,:)=0.0;
40 #ifdef enable_smagorinsky_friction
41     call smagorinsky_fric_u(diff_fe,diff_fn)
42 #else
43     call harm_hfric_u(diff_fe,diff_fn)
44 #endif
45 c-----
46 c     vertical friction
```

```

47 c-----
48     diff_ft(:,js_pe:je_pe,:)=0.0;
49     do k=1,km-1
50         do j=js,je
51             do i=2,imt-1
52                 diff_ft(i,j,k)=A_v*(u(i,j,k+1,1,taum1)-u(i,j,k,1,taum1))/dz
53             &                 *maskU(i,j,k+1)*maskU(i,j,k)
54             enddo
55         enddo
56     enddo
57 c-----
58 c     add surface and bottom boundary conditions
59 c-----
60     do j=js,je
61         diff_ft(:,j,km-1)= surf_tau(:,j,1)*maskU(:,j,km-1)
62         diff_ft(:,j,1 )= bott_tau(:,j,1)*maskU(:,j,2 )
63     enddo
64 c-----
65 c     account for no slip at bottom
66 c-----
67     if (enable_bottom_noslip) then
68         do k=1,km-1
69             do j=js,je
70                 diff_ft(:,j,k)=diff_ft(:,j,k)+2*A_v*u(:,j,k+1,1,taum1)/dz
71             &                 *(1-maskU(:,j,k))*maskU(:,j,k+1)
72             enddo
73         enddo
74     endif
75 c-----
76 c     F_u = - u u_x - v u_y - w u_z + A_h u_xx + A_v u_zz + f_vert v - f_hor w
77 c-----
78     do k=2,km-1
79         do j=js,je
80             do i=2,imt-1
81                 fu(i,j,k)= maskU(i,j,k)*(
82             &     -(adv_fe(i,j,k) - adv_fe(i-1,j,k) )/dx
83             &     -(adv_fn(i,j,k) - adv_fn(i,j-1,k) )/dx
84             &     -(adv_ft(i,j,k) - adv_ft(i,j,k-1) )/dz
85             &     +(diff_ft(i,j,k) - diff_ft(i,j,k-1))/dz
86             &     +(diff_fe(i,j,k) - diff_fe(i-1,j,k))/dx
87             &     +(diff_fn(i,j,k) - diff_fn(i,j-1,k))/dx
88             &     )
89         enddo
90     enddo
91 enddo
92 c-----
93 c     Meridional momentum equation: advective and diffusive fluxes
94 c-----
95     adv_fe(:,js_pe:je_pe,:)=0.0;

```

```

96     adv_fn(:,js_pe:je_pe,:)=0.0;
97     adv_ft(:,js_pe:je_pe,:)=0.0;
98     if (enable_4th_mom_advection) then
99         call setcyclic3D_j2(u(:, :, :, 2, tau) )
100        call adv_flux_v_4th(adv_fe,adv_fn,adv_ft)
101    elseif (enable_quicker_mom_advection) then
102        call setcyclic3D_j2(u(:, :, :, 2, taum1) )
103        call adv_flux_v_quicker(adv_fe,adv_fn,adv_ft)
104    else
105        call adv_flux_v_2nd(adv_fe,adv_fn,adv_ft)
106    endif
107
108    diff_fe(:,js_pe:je_pe,:)=0.0;
109    diff_fn(:,js_pe:je_pe,:)=0.0;
110 #ifndef enable_smagorinsky_friction
111    call smagorinsky_fric_v(diff_fe,diff_fn)
112 #else
113    call harm_hfric_v(diff_fe,diff_fn)
114 #endif
115 c-----
116 c     vertical friction
117 c-----
118     diff_ft(:,js_pe:je_pe,:)=0.0;
119     do k=1,km-1
120         do j=js,je
121             do i=2,imt-1
122                 diff_ft(i,j,k)=A_v*(u(i,j,k+1,2,taum1)-u(i,j,k,2,taum1))/dz
123                 & *maskV(i,j,k+1)*maskV(i,j,k)
124             enddo
125         enddo
126     enddo
127 c-----
128 c     add surface and bottom boundary conditions
129 c-----
130     do j=js,je
131         diff_ft(:,j,km-1)= surf_tau(:,j,2)*maskV(:,j,km-1)
132         diff_ft(:,j,1 )= bott_tau(:,j,2)*maskV(:,j,2)
133     enddo
134 c-----
135 c     account for no slip at bottom
136 c-----
137     if (enable_bottom_noslip) then
138         do k=1,km-1
139             do j=js,je
140                 diff_ft(:,j,k)=diff_ft(:,j,k)+2*A_v*u(:,j,k+1,2,taum1)/dz
141                 & *(1-maskV(:,j,k))*maskV(:,j,k+1)
142             enddo
143         enddo
144     endif

```

```

145 c-----
146 c      F_v = - u v_x - v v_y - w v_z + A_h v_xx + A_v v_zz - f_vert u
147 c-----
148     do k=2,km-1
149     do j=js,je
150     do i=2,imt-1
151         fv(i,j,k)= maskV(i,j,k)*(
152     &   -(adv_fe(i,j,k) - adv_fe(i-1,j,k) )/dx
153     &   -(adv_fn(i,j,k) - adv_fn(i,j-1,k) )/dx
154     &   -(adv_ft(i,j,k) - adv_ft(i,j,k-1) )/dz
155     &   +(diff_ft(i,j,k) - diff_ft(i,j,k-1))/dz
156     &   +(diff_fe(i,j,k) - diff_fe(i-1,j,k))/dx
157     &   +(diff_fn(i,j,k) - diff_fn(i,j-1,k))/dx
158     &   )
159     enddo
160     enddo
161     enddo
162 c-----
163 c      vertical momentum equation: advective and diffusive fluxes
164 c-----
165     if (.not. enable_hydrostatic) then
166     adv_fe(:,js_pe:je_pe,:)=0.0;
167     adv_fn(:,js_pe:je_pe,:)=0.0;
168     adv_ft(:,js_pe:je_pe,:)=0.0;
169     if (enable_4th_mom_advection) then
170     call setcyclic3D_j2(u(:,:,:,3,tau) )
171     call adv_flux_w_4th(adv_fe,adv_fn,adv_ft)
172     elseif (enable_quicker_mom_advection) then
173     call setcyclic3D_j2(u(:,:,:,3,taum1) )
174     call adv_flux_w_quicker(adv_fe,adv_fn,adv_ft)
175     else
176     call adv_flux_w_2nd(adv_fe,adv_fn,adv_ft)
177     endif
178     diff_fe(:,js_pe:je_pe,:)=0.0;
179     diff_fn(:,js_pe:je_pe,:)=0.0;
180     call harm_hfric_w(diff_fe,diff_fn)
181 c-----
182 c      vertical friction
183 c-----
184     diff_ft(:,js_pe:je_pe,:)=0.0;
185     do k=1,km-1
186     do j=js,je
187     do i=2,imt-1
188         diff_ft(i,j,k)=
189     &   A_v*(u(i,j,k+1,3,taum1)-u(i,j,k,3,taum1))/dz
190     &   *maskW(i,j,k+1)*maskW(i,j,k)
191     enddo
192     enddo
193     enddo

```

```

194 c-----
195 c      F_w = - u w_x - v w_y - w w_z + A_h w_xx + A_v w_zz +f_hor u
196 c-----
197     do k=2,km-1
198     do j=js,je
199     do i=2,imt-1
200         fw(i,j,k)= maskW(i,j,k)*(
201     &   -(adv_fe(i,j,k) - adv_fe(i-1,j,k) )/dx
202     &   -(adv_fn(i,j,k) - adv_fn(i,j-1,k) )/dx
203     &   -(adv_ft(i,j,k) - adv_ft(i,j,k-1) )/dz
204     &   +(diff_fe(i,j,k) - diff_fe(i-1,j,k))/dx
205     &   +(diff_fn(i,j,k) - diff_fn(i,j-1,k))/dx
206     &   +(diff_ft(i,j,k) - diff_ft(i,j,k-1))/dz
207     &   )
208     enddo
209     enddo
210     enddo
211     endif
212 c-----
213 c      Add coriolis force to F_u, F_v and F_w
214 c-----
215     call coriolis_force
216 c-----
217 c      add bottom drag : u_t = - c_D u
218 c-----
219     if (enable_bottom_stress) then
220     do j=js,je
221     do i=2,imt-1
222         k=max(1,kmu(i,j))
223         fxa = cdbot*u(i,j,k,1,taum1)
224         fu(i,j,k) = fu(i,j,k)-maskU(i,j,k)*fxa
225         k=max(1,kmv(i,j))
226         fxa = cdbot*u(i,j,k,2,taum1)
227         fv(i,j,k) = fv(i,j,k)-maskV(i,j,k)*fxa
228     enddo
229     enddo
230     endif
231 c-----
232 c      add interior drag : u_t = - c_D u
233 c-----
234     if (enable_interior_stress) then
235     do k=2,km-1
236     do j=js,je
237     do i=2,imt-1
238         fxa = cdint*u(i,j,k,1,taum1)
239         fu(i,j,k) = fu(i,j,k)-maskU(i,j,k)*fxa
240         fxa = cdint*u(i,j,k,2,taum1)
241         fv(i,j,k) = fv(i,j,k)-maskV(i,j,k)*fxa
242     enddo

```

```

243     enddo
244     enddo
245     endif
246 c-----
247 c   add biharmonic friction
248 c-----
249     if (enable_biharmonic_friction) then
250       call biha_hfric_u(diff_fe,diff_fn)
251       call biha_hfric_v(diff_fe,diff_fn)
252       if (.not.enable_hydrostatic) call biha_hfric_w(diff_fe,diff_fn)
253     endif
254     if (enable_vert_biha_friction) then
255       call biha_vfric (diff_ft,maskU,fu,1)
256       call biha_vfric (diff_ft,maskV,fv,2)
257       if (.not.enable_hydrostatic) then
258         call biha_vfric(diff_ft,maskW,fw,3)
259       endif
260     endif
261 c-----
262 c     Nudging terms
263 c-----
264     call momentum_restoring_zones
265 c-----
266 c     boundary exchange for result
267 c-----
268     call border_exchg3D(fu,1)
269     call setcyclic3D(fu)
270     call border_exchg3D(fv,1)
271     call setcyclic3D(fv)
272     if (.not. enable_hydrostatic) then
273       call border_exchg3D(fw,1)
274       call setcyclic3D(fw)
275     endif
276     end subroutine momentum_tendency
277
278
279
280
281
282     subroutine coriolis_force
283 c=====
284 c   Add coriolis force to F_u, F_v and F_w
285 c=====
286     use cpflame_module
287     implicit none
288     integer :: i,j,k,js,je
289     js=max(2,js_pe); je = min(je_pe,jmt-1)
290
291 c-----

```

```

292 c      F_u = A_h u_xx + A_v u_zz + f_vert v - f_hor w + ...
293 c-----
294     do k=2,km-1
295     do j=js,je
296     do i=2,imt-1
297         fu(i,j,k)=fu(i,j,k)+maskU(i,j,k)*
298 &     coriolis_t(j)*(u(i,j ,k,2,tau)+u(i+1,j ,k,2,tau)+
299 &     u(i,j-1,k,2,tau)+u(i+1,j-1,k,2,tau))/4.0
300     enddo
301     enddo
302     enddo
303     if (.not. enable_hydrostatic) then
304     do k=2,km-1
305     do j=js,je
306     do i=2,imt-1
307         fu(i,j,k)=fu(i,j,k)-maskU(i,j,k)*coriolis_hor(j)*
308 &     (u(i,j,k ,3,tau) + u(i+1,j,k ,3,tau)+
309 &     u(i,j,k-1,3,tau) + u(i+1,j,k-1,3,tau) )/4.0
310     enddo
311     enddo
312     enddo
313     endif
314 c-----
315 c      F_v = A_h v_xx + A_v v_zz - f_vert u
316 c-----
317     do k=2,km-1
318     do j=js,je
319     do i=2,imt-1
320         fv(i,j,k)= fv(i,j,k)-maskV(i,j,k)*
321 &     (coriolis_t(j )*(u(i-1,j ,k,1,tau)+u(i,j ,k,1,tau))
322 &     +coriolis_t(j+1)*(u(i-1,j+1,k,1,tau)+u(i,j+1,k,1,tau)))/4.0
323     enddo
324     enddo
325     enddo
326 c-----
327 c      F_w = - u w_x - v w_y - w w_z + A_h w_xx + A_v w_zz + f_hor u
328 c-----
329     if (.not. enable_hydrostatic) then
330     do k=2,km-1
331     do j=js,je
332     do i=2,imt-1
333         fw(i,j,k)=fw(i,j,k)+maskW(i,j,k)*
334 &     coriolis_hor(j)*(u(i,j,k ,1,tau)+u(i-1,j,k ,1,tau)+
335 &     u(i,j,k+1,1,tau)+u(i-1,j,k+1,1,tau))/4.0
336     enddo
337     enddo
338     enddo
339     endif
340     end subroutine coriolis_force

```



```

341
342
343
344     subroutine harm_hfric_u(diff_fe,diff_fn)
345 c=====
346 c     horizontal harmonic friction for u
347 c     diff. fluxes are stored in diff_fe and diff_fn
348 c     account for no slip boundary condition if requested
349 c=====
350     use cpflame_module
351     implicit none
352     integer :: i,j,k,js,je
353     real :: diff_fn(imt,jmt,km), diff_fe(imt,jmt,km)
354
355     js=max(2,js_pe); je = min(je_pe,jmt-1)
356     do k=2,km-1
357         do j=js,je
358             do i=1,imt-1
359                 diff_fe(i,j,k)=A_h*(u(i+1,j,k,1,taum1)-u(i,j,k,1,taum1))/dx
360             enddo
361         enddo
362     enddo
363     call setcyclic3D(diff_fe)
364     do k=2,km-1
365         do j=js,je
366             do i=2,imt-1
367                 diff_fn(i,j,k)=A_h*(u(i,j+1,k,1,taum1)-u(i,j,k,1,taum1))/dx
368 &                 *maskU(i,j+1,k)*maskU(i,j,k)
369             enddo
370         enddo
371     enddo
372     if (enable_noslip) then
373         do j=js-1,je
374             diff_fn(:,j,:)=diff_fn(:,j,:)-2*A_h*u(:,j,:,1,taum1)/dx
375 &             *(1-maskU(:,j+1,:))*maskU(:,j,:)
376             diff_fn(:,j,:)=diff_fn(:,j,:)+2*A_h*u(:,j+1,:,1,taum1)/dx
377 &             *(1-maskU(:,j,:))*maskU(:,j+1,:)
378         enddo
379     endif
380     call border_exchg3D(diff_fn,1)
381     call setcyclic3D(diff_fn)
382     end subroutine harm_hfric_u
383
384
385
386
387
388     subroutine harm_hfric_v(diff_fe,diff_fn)
389 c=====

```

```

390 c    horizontal harmonic friction for v
391 c    diff. fluxes are stored in diff_fe and diff_fn
392 c    account for no slip boundary condition if requested
393 c=====
394     use cpflame_module
395     implicit none
396     integer :: i,j,k,js,je
397     real :: diff_fn(imt,jmt,km), diff_fe(imt,jmt,km)
398     js=max(2,js_pe); je = min(je_pe,jmt-1)
399     do k=2,km-1
400         do j=js,je
401             do i=1,imt-1
402                 diff_fe(i,j,k)=A_h*(u(i+1,j,k,2,taum1)-u(i,j,k,2,taum1))/dx
403             & *maskV(i+1,j,k)*maskV(i,j,k)
404             enddo
405         enddo
406     enddo
407     if (enable_noslip) then
408         do j=js,je
409             do i=1,imt-1
410                 diff_fe(i,j,:)=diff_fe(i,j,:)-2*A_h*u(i,j,:,2,taum1)/dx
411             & *(1-maskV(i+1,j,:))*maskV(i,j,:)
412                 diff_fe(i,j,:)=diff_fe(i,j,:)+2*A_h*u(i+1,j,:,2,taum1)/dx
413             & *(1-maskV(i,j,:))*maskV(i+1,j,:)
414             enddo
415         enddo
416     endif
417     call setcyclic3D(diff_fe)
418     do k=2,km-1
419         do j=js,je
420             do i=2,imt-1
421                 diff_fn(i,j,k)=A_h*(u(i,j+1,k,2,taum1)-u(i,j,k,2,taum1) )/dx
422             enddo
423         enddo
424     enddo
425     call border_exchg3D(diff_fn,1)
426     call setcyclic3D(diff_fn)
427     end subroutine harm_hfric_v
428
429
430
431
432     subroutine harm_hfric_w(diff_fe,diff_fn)
433 c=====
434 c    horizontal harmonic friction for w
435 c    diff. fluxes are stored in diff_fe and diff_fn
436 c=====
437     use cpflame_module
438     implicit none

```

```

439     integer :: i,j,k,js,je
440     real :: diff_fn(imt,jmt,km), diff_fe(imt,jmt,km)
441     js=max(2,js_pe); je = min(je_pe,jmt-1)
442     do k=2,km-1
443         do j=js,je
444             do i=1,imt-1
445                 diff_fe(i,j,k)=
446 &         A_h*(u(i+1,j,k,3,taum1)-u(i,j,k,3,taum1))/dx
447 &         *maskW(i+1,j,k)*maskW(i,j,k)
448             enddo
449         enddo
450     enddo
451     call setcyclic3D(diff_fe)
452     do k=2,km-1
453         do j=js,je
454             do i=2,imt-1
455                 diff_fn(i,j,k)=
456 &         A_h*(u(i,j+1,k,3,taum1)-u(i,j,k,3,taum1))/dx
457 &         *maskW(i,j+1,k)*maskW(i,j,k)
458             enddo
459         enddo
460     enddo
461     call border_exchg3D(diff_fn,1)
462     call setcyclic3D(diff_fn)
463     end subroutine harm_hfric_w

```

---

---

```

1 #include "options.inc"
2
3 C=====
4 c    Biharmonic friction and diffusion
5 C=====
6
7     subroutine biha_hfric_u (diff_fe,diff_fn)
8 C-----
9 c    horizontal biharmonic friction for zonal momentum
10 c    no slip condition is possible
11 C-----
12     use cpflame_module
13     implicit none
14     integer :: i,j,k,js,je
15     real :: diff_fe(imt,jmt,km), diff_fn(imt,jmt,km)
16     real :: del2(imt,jmt,km),diff_tx,diff_ty,diffx
17     DIFF_Tx(i,j,k) = (diff_fe(i,j,k)-diff_fe(i-1,j,k))/dx
18     DIFF_Ty(i,j,k) = (diff_fn(i,j,k)-diff_fn(i,j-1,k))/dx
19
20     js=max(2,js_pe); je = min(je_pe,jmt-1)
21     diffx = sqrt(abs(ahbi))
22     diff_fe(:,js_pe:je_pe,:)=0.0;diff_fn(:,js_pe:je_pe,:)=0.0
23     do k=1,km
24         do j=js,je
25             do i=1,imt-1
26                 diff_fe(i,j,k)=diffx*(u(i+1,j,k,1,taum1)-u(i,j,k,1,taum1))/dx
27             enddo
28         enddo
29     enddo
30     call border_exchg3D(diff_fe,1); call setcyclic3D(diff_fe)
31     do k=1,km-1
32         do j=js,je
33             do i=1,imt
34                 diff_fn(i,j,k)=diffx*(u(i,j+1,k,1,taum1)-u(i,j,k,1,taum1))/dx
35             &                *maskU(i,j+1,k)*maskU(i,j,k)
36         enddo
37     enddo
38     enddo
39     if (enable_noslip) then
40         do j=js-1,je
41             diff_fn(:,j,:)=diff_fn(:,j:)-2*diffx*u(:,j,:,1,taum1)/dx
42             &                *(1-maskU(:,j+1,:))*maskU(:,j,:)
43             &                +2*diffx*u(:,j+1,:,1,taum1)/dx
44             &                *(1-maskU(:,j,:))*maskU(:,j+1,:)
45         enddo
46     endif
47     call border_exchg3D(diff_fn,1); call setcyclic3D(diff_fn)
48     del2(:,js_pe:je_pe,:)=0.0

```

```

49     do k=2,km-1
50         do j=js,je
51             do i=2,imt-1
52                 del2(i,j,k) = (DIFF_Tx(i,j,k) + DIFF_Ty(i,j,k))*maskU(i,j,k)
53             enddo
54         enddo
55     enddo
56     call border_exchg3D(del2,1); call setcyclic3D(del2)
57     diff_fe(:,js_pe:je_pe,:)=0.0;diff_fn(:,js_pe:je_pe,:)=0.0
58     do k=2,km-1
59         do j=js,je
60             do i=1,imt-1
61                 diff_fe(i,j,k) =diffx*(del2(i+1,j,k)-del2(i,j,k))/dx
62             enddo
63         enddo
64     enddo
65     call border_exchg3D(diff_fe,1); call setcyclic3D(diff_fe)
66     do k=1,km-1
67         do j=js-1,je
68             do i=2,imt-1
69                 diff_fn(i,j,k) = diffx*(del2(i,j+1,k) - del2(i,j,k))/dx
70 &                                     *maskU(i,j+1,k)*maskU(i,j,k)
71             enddo
72         enddo
73     enddo
74     if (enable_noslip) then
75         do j=js-1,je
76             diff_fn(:,j,:)=diff_fn(:,j:)-2*diffx*del2(:,j:)/dx
77 &                                     *(1-maskU(:,j+1,:))*maskU(:,j,:)
78 &                                     +2*diffx*del2(:,j+1:)/dx
79 &                                     *(1-maskU(:,j,:))*maskU(:,j+1,:)
80         enddo
81     endif
82     call border_exchg3D(diff_fn,1); call setcyclic3D(diff_fn)
83     do k=2,km-1
84         do j=js,je
85             do i=2,imt-1
86                 fu(i,j,k)=fu(i,j,k)
87 &                 -maskU(i,j,k)*(DIFF_Tx(i,j,k)+DIFF_Ty(i,j,k))
88             enddo
89         enddo
90     enddo
91     end subroutine biha_hfric_u
92
93
94
95
96     subroutine biha_hfric_v (diff_fe,diff_fn)
97 c-----

```

```

98 c    horizontal biharmonic friction for meridional momentum
99 c    no slip condition is possible
100 c-----
101     use cpflame_module
102     implicit none
103     integer :: i,j,k,js,je
104     real :: diff_fe(imt,jmt,km), diff_fn(imt,jmt,km)
105     real :: del2(imt,jmt,km),diff_tx,diff_ty,diffx
106     DIFF_Tx(i,j,k)=(diff_fe(i,j,k)-diff_fe(i-1,j,k))/dx
107     DIFF_Ty(i,j,k)=(diff_fn(i,j,k)-diff_fn(i,j-1,k))/dx
108
109     js=max(2,js_pe); je = min(je_pe,jmt-1)
110     diffx = sqrt(abs(ahbi))
111     diff_fe(:,js_pe:je_pe,:)=0.0;diff_fn(:,js_pe:je_pe,:)=0.0
112     do k=1,km
113         do j=js,je
114             do i=1,imt-1
115                 diff_fe(i,j,k)=diffx*(u(i+1,j,k,2,taum1)-u(i,j,k,2,taum1))/dx
116             & *maskV(i,j,k)*maskV(i+1,j,k)
117             enddo
118         enddo
119     enddo
120     if (enable_noslip) then
121         do j=js,je
122             do i=1,imt-1
123                 diff_fe(i,j,:)=diff_fe(i,j,:)-2*diffx*u(i,j,:,2,taum1)/dx
124             & *(1-maskV(i+1,j,:))*maskV(i,j,:)
125             & +2*diffx*u(i+1,j,:,2,taum1)/dx
126             & *(1-maskV(i,j,:))*maskV(i+1,j,:)
127             enddo
128         enddo
129     endif
130     call border_exchg3D(diff_fe,1); call setcyclic3D(diff_fe)
131     do k=1,km-1
132         do j=js,je
133             do i=1,imt
134                 diff_fn(i,j,k)=diffx*(u(i,j+1,k,2,taum1)-u(i,j,k,2,taum1))/dx
135             enddo
136         enddo
137     enddo
138     call border_exchg3D(diff_fn,1); call setcyclic3D(diff_fn)
139     del2(:,je_pe:je_pe,:)=0.0
140     do k=2,km-1
141         do j=js,je
142             do i=2,imt-1
143                 del2(i,j,k) = (DIFF_Tx(i,j,k) + DIFF_Ty(i,j,k))*maskV(i,j,k)
144             enddo
145         enddo
146     enddo

```

```

147 call border_exchg3D(del2,1); call setcyclic3D(del2)
148 diff_fe(:,js_pe:je_pe,:)=0.0;diff_fn(:,js_pe:je_pe,:)=0.0
149 do k=2,km-1
150   do j=js,je
151     do i=1,imt-1
152       diff_fe(i,j,k) =diffx*(del2(i+1,j,k)-del2(i,j,k))/dx
153     &          *maskV(i+1,j,k)*maskV(i,j,k)
154     enddo
155   enddo
156 enddo
157 if (enable_noslip) then
158   do j=js,je
159     do i=1,imt-1
160       diff_fe(i,j,:)=diff_fe(i,j,:)-2*diffx*del2(i,j,:)/dx
161     &          *(1-maskV(i+1,j,:))*maskV(i,j,:)
162     &          +2*diffx*del2(i+1,j,:)/dx
163     &          *(1-maskV(i,j,:))*maskV(i+1,j,:)
164     enddo
165   enddo
166 endif
167 call border_exchg3D(diff_fe,1); call setcyclic3D(diff_fe)
168 do k=1,km-1
169   do j=js,je
170     do i=2,imt-1
171       diff_fn(i,j,k) = diffx*(del2(i,j+1,k) - del2(i,j,k))/dx
172     enddo
173   enddo
174 enddo
175 call border_exchg3D(diff_fn,1); call setcyclic3D(diff_fn)
176 do k=2,km-1
177   do j=js,je
178     do i=2,imt-1
179       fv(i,j,k)= fv(i,j,k)+maskV(i,j,k)*(
180     &          -DIFF_Tx(i,j,k) - DIFF_Ty(i,j,k))
181     enddo
182   enddo
183 enddo
184 end subroutine biha_hfric_v
185
186
187
188
189
190 subroutine biha_hfric_w (diff_fe,diff_fn)
191 c-----
192 c   horizontal biharmonic friction for vertical momentum
193 c-----
194 use cpflame_module
195 implicit none

```

```

196     integer :: i,j,k,js,je
197     real :: diff_fe(imt,jmt,km), diff_fn(imt,jmt,km)
198     real :: del2(imt,jmt,km),diff_tx,diff_ty,diffx
199     DIFF_Tx(i,j,k)=(diff_fe(i,j,k)-diff_fe(i-1,j,k))/dx
200     DIFF_Ty(i,j,k)=(diff_fn(i,j,k)-diff_fn(i,j-1,k))/dx
201
202     js=max(2,js_pe); je = min(je_pe,jmt-1)
203     diffx = sqrt(abs(ahbi))
204     diff_fe(:,js_pe:je_pe,:)=0.0;diff_fn(:,js_pe:je_pe,:)=0.0
205     do k=1,km
206         do j=js,je
207             do i=1,imt-1
208                 diff_fe(i,j,k)=diffx*(u(i+1,j,k,3,taum1)-u(i,j,k,3,taum1))/dx
209             & *maskW(i,j,k)*maskW(i+1,j,k)
210             enddo
211         enddo
212     enddo
213     call border_exchg3D(diff_fe,1); call setcyclic3D(diff_fe)
214     do k=1,km-1
215         do j=js,je
216             do i=1,imt
217                 diff_fn(i,j,k)=diffx*(u(i,j+1,k,3,taum1)-u(i,j,k,3,taum1))/dx
218             & *maskW(i,j,k)*maskW(i,j+1,k)
219             enddo
220         enddo
221     enddo
222     call border_exchg3D(diff_fn,1); call setcyclic3D(diff_fn)
223     del2(:,je_pe:je_pe,:)=0.0
224     do k=2,km-1
225         do j=js,je
226             do i=2,imt-1
227                 del2(i,j,k) = (DIFF_Tx(i,j,k) + DIFF_Ty(i,j,k))*maskW(i,j,k)
228             enddo
229         enddo
230     enddo
231     call border_exchg3D(del2,1); call setcyclic3D(del2)
232     diff_fe(:,js_pe:je_pe,:)=0.0;diff_fn(:,js_pe:je_pe,:)=0.0
233     do k=2,km-1
234         do j=js,je
235             do i=1,imt-1
236                 diff_fe(i,j,k) =diffx*(del2(i+1,j,k)-del2(i,j,k))/dx
237             & *maskW(i+1,j,k)*maskW(i,j,k)
238             enddo
239         enddo
240     enddo
241     call border_exchg3D(diff_fe,1); call setcyclic3D(diff_fe)
242     do k=1,km-1
243         do j=js,je
244             do i=2,imt-1

```



```

245     diff_fn(i,j,k) = diffx*(del2(i,j+1,k) - del2(i,j,k))/dx
246 &         *maskW(i,j+1,k)*maskW(i,j,k)
247     enddo
248     enddo
249     enddo
250     call border_exchg3D(diff_fn,1); call setcyclic3D(diff_fn)
251     do k=2,km-1
252         do j=js,je
253             do i=2,imt-1
254                 fw(i,j,k)= fw(i,j,k)+maskW(i,j,k)*(
255 &                     -DIFF_Tx(i,j,k) - DIFF_Ty(i,j,k))
256             enddo
257         enddo
258     enddo
259     end subroutine biha_hfric_w
260
261
262
263     subroutine biha_vfric (diff_ft,mask,ff,n)
264 c-----
265 c     vertical biharmonic friction for momentum component n
266 c     result is added to ff
267 c-----
268     use cpflame_module
269     implicit none
270     integer :: i,j,k,js,je,n
271     real,dimension(imt,jmt,km) :: diff_ft,mask,ff
272     real :: del2(imt,jmt,km),diff_tz,diffx
273     DIFF_Tz(i,j,k)=(diff_ft(i,j,k)-diff_ft(i,j,k-1))/dz
274
275     js=max(2,js_pe); je = min(je_pe,jmt-1)
276     diffx = sqrt(abs(Avbi))
277     diff_ft(:,je_pe:je_pe,:)=0.0
278     do k=1,km-1
279         do j=js,je
280             do i=2,imt-1
281                 diff_ft(i,j,k)=diffx*(u(i,j,k+1,n,taum1)-u(i,j,k,n,taum1))/dz
282 &                 *mask(i,j,k)*mask(i,j,k+1)
283             enddo
284         enddo
285     enddo
286     del2(:,je_pe:je_pe,:)=0.0
287     do k=2,km-1
288         do j=js,je
289             do i=2,imt-1
290                 del2(i,j,k) = DIFF_Tz(i,j,k)*mask(i,j,k)
291             enddo
292         enddo
293     enddo

```

```

294     diff_ft(:,je_pe:je_pe,:)=0.0
295     do k=2,km-1
296         do j=js,je
297             do i=2,imt-1
298                 diff_ft(i,j,k) =diffx*(del2(i,j,k+1)-del2(i,j,k))/dz
299             &                 *mask(i,j,k+1)*mask(i,j,k)
300         enddo
301     enddo
302 enddo
303 do k=2,km-1
304     do j=js,je
305         do i=2,imt-1
306             ff(i,j,k)= ff(i,j,k)-mask(i,j,k)*DIFF_Tz(i,j,k)
307         enddo
308     enddo
309 enddo
310 end subroutine biha_vfric
311
312
313
314     subroutine biha_mix (diff_fe,diff_fn,diff_ft,var)
315 c-----
316 c     horizontal biharmonic diffusion of buoyancy
317 c     also vertical biha. diff. if requested
318 c-----
319     use cpflame_module
320     implicit none
321     integer :: i,j,k,js,je
322     real :: var(imt,jmt,km,0:2)
323     real :: diff_fe(imt,jmt,km), diff_fn(imt,jmt,km)
324     real :: diff_ft(imt,jmt,km) ,diffz
325     real :: del2(imt,jmt,km),diff_tx,diff_ty,diffx,diff_tz
326     DIFF_Tx(i,j,k)=(diff_fe(i,j,k)-diff_fe(i-1,j,k))/dx
327     DIFF_Ty(i,j,k)=(diff_fn(i,j,k)-diff_fn(i,j-1,k))/dx
328     DIFF_Tz(i,j,k)=(diff_ft(i,j,k)-diff_ft(i,j,k-1))/dz
329
330     js=max(2,js_pe); je = min(je_pe,jmt-1)
331     diffx = sqrt(abs(Khbi))
332     diffz = sqrt(abs(Kvbi))
333     diff_fe(:,js_pe:je_pe,:)=0.0; diff_fn(:,js_pe:je_pe,:)=0.0
334     do k=1,km
335         do j=js,je
336             do i=1,imt-1
337                 diff_fe(i,j,k)=diffx*(var(i+1,j,k,taum1)-var(i,j,k,taum1))/dx
338             &                 *maskU(i,j,k)
339         enddo
340     enddo
341 enddo
342 call border_exchg3D(diff_fe,1); call setcyclic3D(diff_fe)

```

```

343     do k=1,km-1
344     do j=js,je
345     do i=1,imt
346         diff_fn(i,j,k)=diffx*(var(i,j+1,k,taum1)-var(i,j,k,taum1))/dx
347     & *maskV(i,j,k)
348     enddo
349     enddo
350     enddo
351     call border_exchg3D(diff_fn,1); call setcyclic3D(diff_fn)
352     del2(:,js_pe:je_pe,:)=0.0
353     do k=2,km-1
354     do j=js,je
355     do i=2,imt-1
356     del2(i,j,k) = (DIFF_Tx(i,j,k) + DIFF_Ty(i,j,k))*maskT(i,j,k)
357     enddo
358     enddo
359     enddo
360     call border_exchg3D(del2,1); call setcyclic3D(del2)
361     diff_fe(:,js_pe:je_pe,:)=0.0; diff_fn(:,js_pe:je_pe,:)=0.0
362     do k=2,km-1
363     do j=js,je
364     do i=1,imt-1
365         diff_fe(i,j,k) =diffx*(del2(i+1,j,k)-del2(i,j,k))/dx
366     & *maskU(i,j,k)
367     enddo
368     enddo
369     enddo
370     call border_exchg3D(diff_fe,1); call setcyclic3D(diff_fe)
371     do k=1,km-1
372     do j=js,je
373     do i=2,imt-1
374         diff_fn(i,j,k) = diffx*(del2(i,j+1,k) - del2(i,j,k))/dx
375     & *maskV(i,j,k)
376     enddo
377     enddo
378     enddo
379     call border_exchg3D(diff_fn,1); call setcyclic3D(diff_fn)
380     do k=2,km-1
381     do j=js,je
382     do i=2,imt-1
383         var(i,j,k,taup1)= var(i,j,k,taup1)+c2dt*maskT(i,j,k)*(
384     & -DIFF_Tx(i,j,k) - DIFF_Ty(i,j,k))
385     enddo
386     enddo
387     enddo
388     if (enable_vert_biha_diffusion) then
389     diff_ft(:,js_pe:je_pe,:)=0.0
390     do k=1,km-1
391     do j=js,je

```

```

392     do i=2,imt-1
393         diff_ft(i,j,k)=diffz*(var(i,j,k+1,taum1)-var(i,j,k,taum1))/dz
394     &         *maskW(i,j,k)
395     enddo
396     enddo
397     enddo
398     del2(:,js_pe:je_pe,:)=0.0
399     do k=2,km-1
400         do j=js,je
401             do i=2,imt-1
402     del2(i,j,k) = DIFF_Tz(i,j,k)*maskT(i,j,k)
403             enddo
404             enddo
405             enddo
406     diff_ft(:,js_pe:je_pe,:)=0.0
407     do k=2,km-1
408         do j=js,je
409             do i=2,imt-1
410     diff_ft(i,j,k) =diffz*(del2(i,j,k+1)-del2(i,j,k))/dz
411     &         *maskW(i,j,k)
412             enddo
413             enddo
414             enddo
415     do k=2,km-1
416         do j=js,je
417             do i=2,imt-1
418     var(i,j,k,taup1)=var(i,j,k,taup1)
419     &         -c2dt*maskT(i,j,k)*DIFF_Tz(i,j,k)
420             enddo
421             enddo
422             enddo
423     endif
424     end subroutine biha_mix

```

---