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 Jesper R Pedersen

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

Abstract **1**

Contents **2**

List of Figures **2**

1 Introduction **3**

2 Theory **3**
2.1 Interior circulation 4
2.2 The homogeneous model 4

3 Veros **7**
3.1 Implementing no slip boundary conditions 7
3.2 The ACC model 9

4 Comparison of free- and no slip **11**

5 Discussion and Conclusion **12**

Acknowledgements **13**

References **14**

Apx. A; Veros friction.py **15**

Apx. B; ACC model setup **23**

Apx. C; Code for Plots **26**

Apx. D; Carsten Eden's implementation of no slip **32**

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 homogenious 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:

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

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:

$$\begin{aligned} \frac{U}{fL} &\equiv R_0 \quad \text{the Rossby number,} \\ f &= 2\Omega \sin\theta. \end{aligned} \tag{2}$$

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:

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

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 β plane approximation:

$$f = f_0 + \beta y. \tag{4}$$

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 ψ is the stream function, p the pressure, ρ the density and f_0 is a characteristic parameter of the Coriolis term in the gyre. Now the relative vorticity ζ:

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

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 homogenious 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

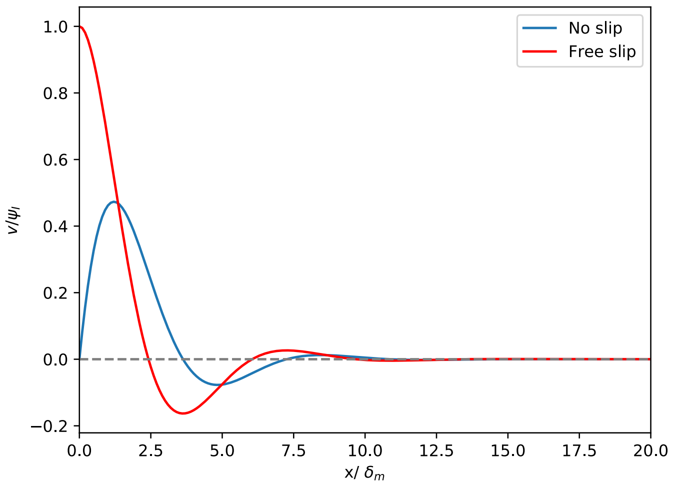

Figure 1: A plot of v for a western boundary current obtained by ignoring nonlinearity in the boundary layer for both free- and no slip boundary conditions.

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

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

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

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

Imposing the no slip boundary condition leaves us with:

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

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

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

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

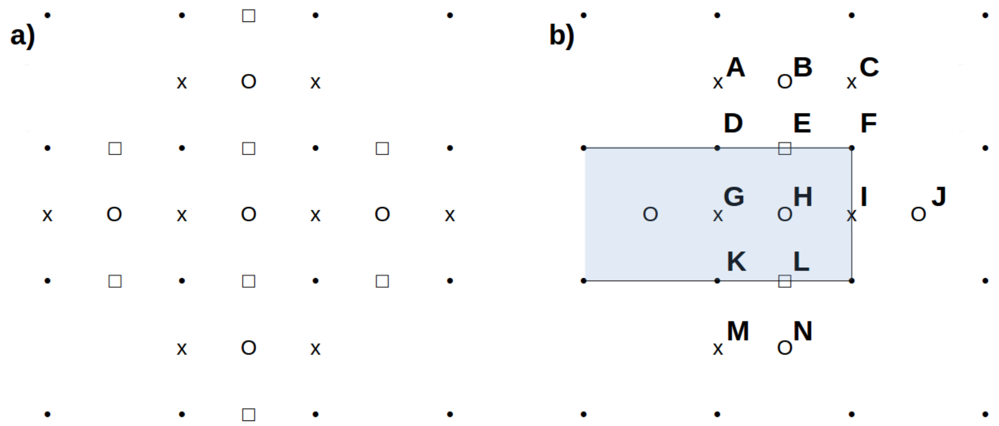

Figure 2: A plot of the Arakawa C-grid. The keys are: Cirles are the tracers. Crosses are zonal velocity, u, and squares meridional velocity, v. Since the grid is staggered the keys doesn't overlab. In part b) vaiurs points are labelled to be used in explaining no slip boundary complications.

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 docmentation; 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 initial 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 where 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 A, since G is on the land. This implies that the gradient between A and G is A, and if you then extrapolate linearly to the value of the velocity at the boundary (D), you get, in Cartesian coordinates:

$$\frac{\partial u_D}{\partial y} = \frac{u_G - u_A}{\delta y} = -\frac{u_A}{\delta y} \tag{17}$$

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:

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

In this way the value at D is instead:

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

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:

```
enable_noslip_lateral = True/False
```

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

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

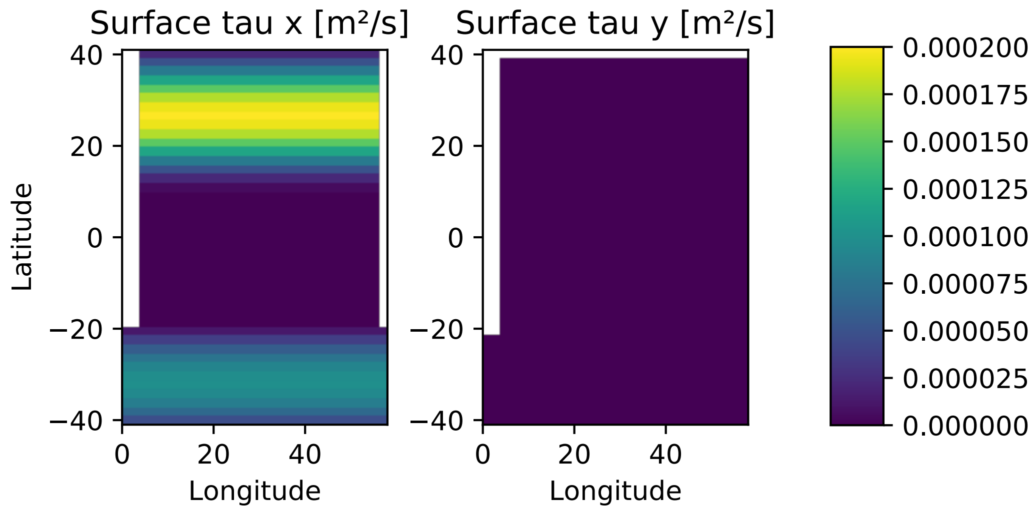

Figure 3: A plot of the wind stress forcing on the ACC model. On the left I have plotted the wind stress in the zonal direction, and on the right the meridional. The model has periodic boundary conditions in the zonal direction, and has walls in the meridional. The wind stress is constant in time.

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:

$$\text{vs.maskV[1:]} * \text{vs.maskV[:-1]}, \tag{20}$$

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:

$$(1 - \text{vs.maskV[1:]}) * \text{vs.maskV[:-1]}, \tag{21}$$

and

$$\text{vs.maskV[1:]} * (1 - \text{vs.maskV[:-1]}). \tag{22}$$

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.

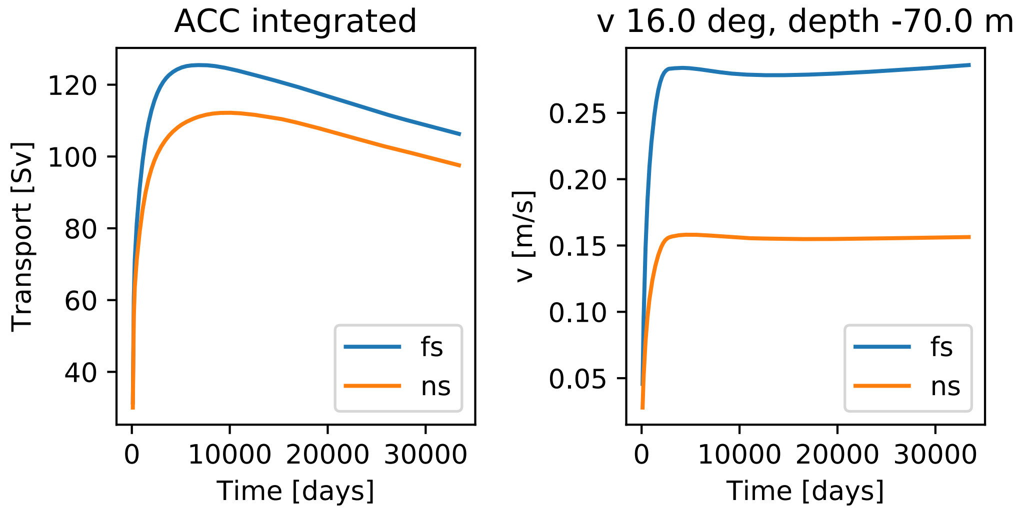

Figure 4: On the left the total transport of the ACC in the model is plotted, and on the right the value of v at the western boundary.

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.

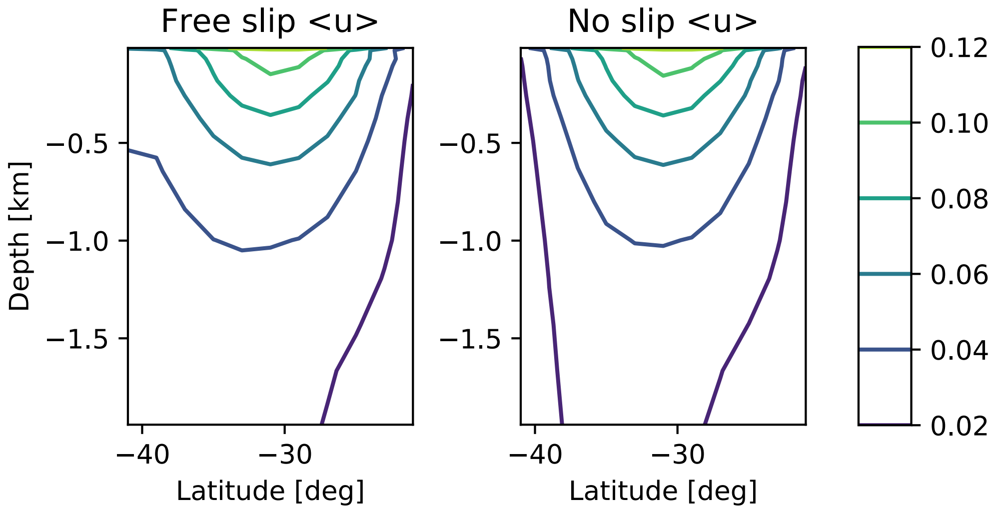

Figure 5: A comparison of the contours of u for the ACC between free slip, on the left, and no slip, on the right. The comparison is on the average of u for 20 degrees longitude. The latitude is from -41 degrees which is also the end of the model. The boundary is thus at -42 degrees, and the difference in u is immediately clear from looking at the figures. u is not excatly 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° 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° 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° 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, where as for no slip the the velocity is lower, but decreases less steeply away from the boundary.

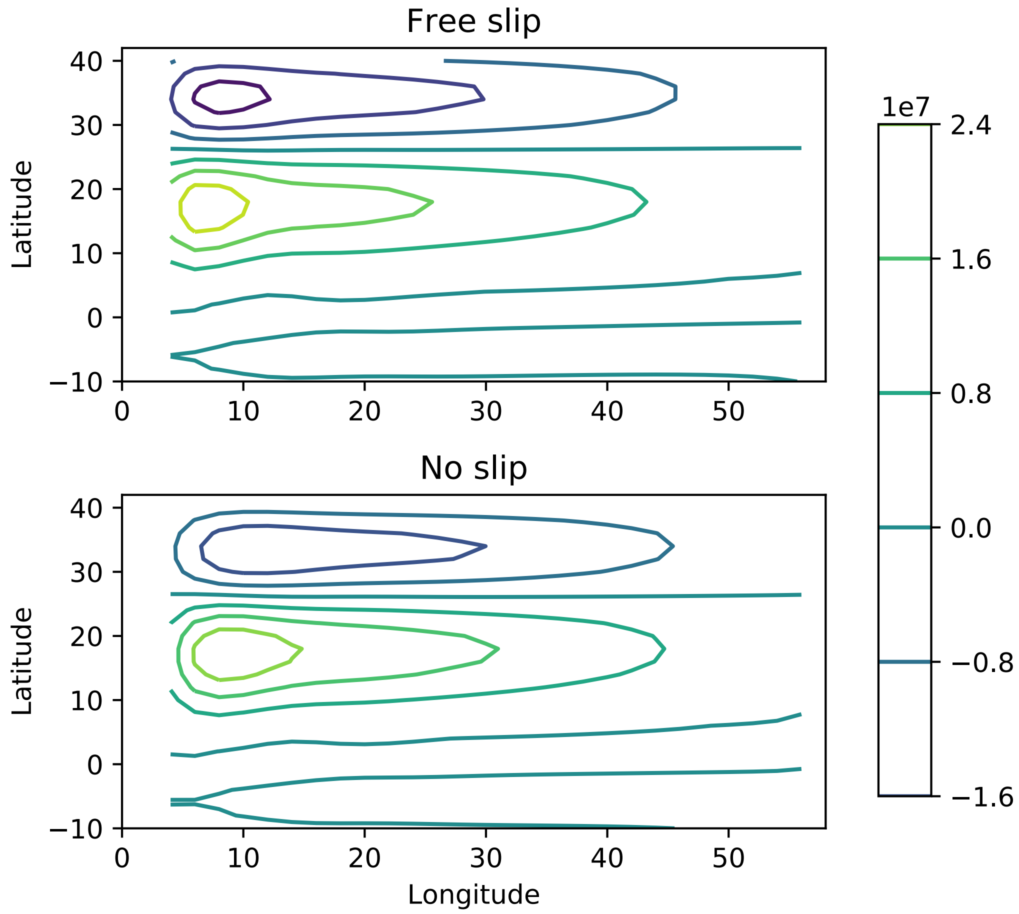

Figure 6: The streamlines plotted starting from the bottom of the continent, at -10 south. The western boundary current is visible around latitude 10 to 20.

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 exists 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

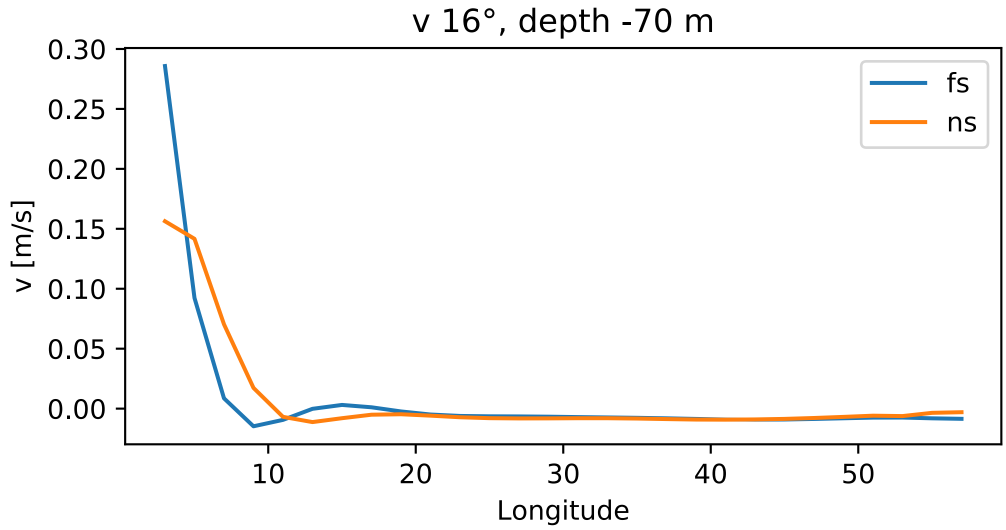

Figure 7: A comparison of the western boundary current for free slip, blue, and no slip, orange. The value of v for the free slip case looks like Munks solution, having maximum close to the boundary, and dropping below zero a short distance away. In the no slip case the velocity is way lower, but not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

Ackowledgements

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` (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:, :, :, vs.tau] - vs.u[:-1, :, :, vs.tau]) \
21             / (vs.cost * vs.dxt[1:, np.newaxis])[:, :, np.newaxis] * vs.maskU[1:] * vs.maskU[:-1]
22         fxa = vs.cosu**vs.hor_friction_cosPower
23         vs.flux_north[:, :-1] = vs.A_h * fxa[np.newaxis, :-1, np.newaxis] * (vs.u[:, 1:, :, vs.tau] - vs.u[:, :-1, :, vs.tau]) \
24             / vs.dyu[np.newaxis, :-1, np.newaxis] * vs.maskU[:, 1:] * vs.maskU[:, :-1] * vs.cosu[np.newaxis, :-1, np.newaxis]
25         if vs.enable_noslip_lateral:
26              vs.flux_north[:, :-1] += 2 * vs.A_h * fxa[np.newaxis, :-1, np.newaxis] * (vs.u[:, 1:, :, vs.tau]) \
27                 / vs.dyu[np.newaxis, :-1, np.newaxis] * vs.maskU[:, 1:] * (1 - vs.maskU[:, :-1]) * vs.cosu[np.newaxis, :-1, np.newaxis]\
28                 - 2 * vs.A_h * fxa[np.newaxis, :-1, np.newaxis] * (vs.u[:, :-1, :, vs.tau]) \
29                 / vs.dyu[np.newaxis, :-1, np.newaxis] * (1 - vs.maskU[:, 1:]) * vs.maskU[:, :-1] * vs.cosu[np.newaxis, :-1, np.newaxis]
30     else:
31         vs.flux_east[:-1, :, :] = vs.A_h * (vs.u[1:, :, :, vs.tau] - vs.u[:-1, :, :, vs.tau]) \
32             / (vs.cost * vs.dxt[1:, np.newaxis])[:, :, np.newaxis] * vs.maskU[1:] * vs.maskU[:-1]
33         vs.flux_north[:, :-1, :] = vs.A_h * (vs.u[:, 1:, :, vs.tau] - vs.u[:, :-1, :, vs.tau]) \
```

```python
           / vs.dyu[np.newaxis, :-1, np.newaxis] * vs.maskU[:, 1:] * vs.maskU[:, :-1] * vs.cosu[np.newaxis, :-1, np.newaxis]
        if vs.enable_noslip_lateral:
            vs.flux_north[:, :-1] += 2 * vs.A_h * vs.u[:, 1:, :, vs.tau] / vs.dyu[np.newaxis, :-1, np.newaxis] \
              * vs.maskU[:, 1:] * (1 - vs.maskU[:, :-1]) * vs.cosu[np.newaxis, :-1, np.newaxis]\
              - 2 * vs.A_h * vs.u[:, :-1, :, vs.tau] / vs.dyu[np.newaxis, :-1, np.newaxis] \
              * (1 - vs.maskU[:, 1:]) * vs.maskU[:, :-1] * vs.cosu[np.newaxis, :-1, np.newaxis]

    vs.flux_east[-1, :, :] = 0.
    vs.flux_north[:, -1, :] = 0.

    """
    update tendency
    """
    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])
                                                      / (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])

    if vs.enable_conserve_energy:
        """
        diagnose dissipation by lateral friction
        """
        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]
                                 + (vs.u[1:-2, 2:-2, :, vs.tau] - vs.u[:-3, 2:-2, :, vs.tau]) * vs.flux_east[:-3, 2:-2]) \
           / (vs.cost[2:-2] * vs.dxu[1:-2, np.newaxis])[:, :, np.newaxis]\
           + 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]
             + (vs.u[1:-2, 2:-2, :, vs.tau] - vs.u[1:-2, 1:-3, :, vs.tau]) * vs.flux_north[1:-2, 1:-3]) \
           / (vs.cost[2:-2] * vs.dyt[2:-2])[np.newaxis, :, np.newaxis]
        vs.K_diss_h[...] = 0.
        vs.K_diss_h[...] += numerics.calc_diss(vs, diss, 'U')

```

```python
    """
    Meridional velocity
    """
    if vs.enable_hor_friction_cos_scaling:
        fxa = (vs.cosu ** vs.hor_friction_cosPower) * np.ones((vs.nx + 3, 1), dtype=vs.default_float_type)
        vs.flux_east[:-1] = vs.A_h * fxa[:, :, np.newaxis] * (vs.v[1:, :, :, vs.tau] - vs.v[:-1, :, :, vs.tau]) \
            / (vs.cosu * vs.dxu[:-1, np.newaxis])[:, :, np.newaxis] * vs.maskV[1:] * vs.maskV[:-1]
        if vs.enable_noslip_lateral:
            vs.flux_east[:-1] += 2 * vs.A_h * fxa[:, :, np.newaxis] * vs.v[1:, :, :, vs.tau] \
                / (vs.cosu * vs.dxu[:-1, np.newaxis])[:, :, np.newaxis] * vs.maskV[1:] * (1 - vs.maskV[:-1]) \
                - 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]

        fxa = (vs.cost[1:] ** vs.hor_friction_cosPower) * np.ones((vs.nx + 4, 1), dtype=vs.default_float_type)
        vs.flux_north[:, :-1] = vs.A_h * fxa[:, :, np.newaxis] * (vs.v[:, 1:, :, vs.tau] - vs.v[:, :-1, :, vs.tau]) \
            / vs.dyt[np.newaxis, 1:, np.newaxis] * vs.cost[np.newaxis, 1:, np.newaxis] * vs.maskV[:, :-1] * vs.maskV[:, 1:]
    else:
        vs.flux_east[:-1] = vs.A_h * (vs.v[1:, :, :, vs.tau] - vs.v[:-1, :, :, vs.tau]) \
            / (vs.cosu * vs.dxu[:-1, np.newaxis])[:, :, np.newaxis] * vs.maskV[1:] * vs.maskV[:-1]
        if vs.enable_noslip_lateral:
            vs.flux_east[:-1] += 2 * vs.A_h * vs.v[1:, :, :, vs.tau] / (vs.cosu * vs.dxu[:-1, np.newaxis])[:, :, np.newaxis] \
                * vs.maskV[1:] * (1 - vs.maskV[:-1]) \
                - 2 * vs.A_h * vs.v[:-1, :, :, vs.tau] / (vs.cosu * vs.dxu[:-1, np.newaxis])[:, :, np.newaxis] \
                * (1 - vs.maskV[1:]) * vs.maskV[:-1]
        vs.flux_north[:, :-1] = vs.A_h * (vs.v[:, 1:, :, vs.tau] - vs.v[:, :-1, :, vs.tau]) \
            / vs.dyt[np.newaxis, 1:, np.newaxis] * vs.cost[np.newaxis, 1:, np.newaxis] * vs.maskV[:, :-1] * vs.maskV[:, 1:]
    vs.flux_east[-1, :, :] = 0.
    vs.flux_north[:, -1, :] = 0.

    """
    update tendency
    """
    vs.dv_mix[2:-2, 2:-2] += vs.maskV[2:-2, 2:-2] * ((vs.flux_east[2:-2, 2:-2] - vs.flux_east[1:-3, 2:-2])
```

```
                                          / (vs.cosu[2:-2] * vs.dxt[2:-2,
                                            np.newaxis])[:, :, np.newaxis]
                                          + (vs.flux_north[2:-2, 2:-2] -
                                            vs.flux_north[2:-2, 1:-3])
                                          / (vs.dyu[2:-2] *
                                            vs.cosu[2:-2])[np.newaxis, :,
                                            np.newaxis])

    if vs.enable_conserve_energy:
        """
        diagnose dissipation by lateral friction
        """
        diss[2:-2, 1:-2] = 0.5 * ((vs.v[3:-1, 1:-2, :, vs.tau] - vs.v[2:-2,
            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]) \
            / (vs.cosu[1:-2] * vs.dxt[2:-2, np.newaxis])[:, :, np.newaxis] \
            + 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]
                  + (vs.v[2:-2, 1:-2, :, vs.tau] - vs.v[2:-2, :-3, :, vs.tau]) *
                      vs.flux_north[2:-2, :-3]) \
            / (vs.cosu[1:-2] * vs.dyu[1:-2])[np.newaxis, :, np.newaxis]
        vs.K_diss_h[...] += numerics.calc_diss(vs, diss, 'V')


@veros_method
def biharmonic_friction(vs):
    """
    horizontal biharmonic friction
    dissipation is calculated and added to K_diss_h
    """
    fxa = math.sqrt(abs(vs.A_hbi))

    """
    Zonal velocity
    """
    vs.flux_east[:-1, :, :] = fxa * (vs.u[1:, :, :, vs.tau] - vs.u[:-1, :, :,
        vs.tau]) \
        / (vs.cost[np.newaxis, :, np.newaxis] * vs.dxt[1:, np.newaxis,
            np.newaxis]) \
        * vs.maskU[1:, :, :] * vs.maskU[:-1, :, :]
    vs.flux_north[:, :-1, :] = fxa * (vs.u[:, 1:, :, vs.tau] - vs.u[:, :-1, :,
        vs.tau]) \
        / vs.dyu[np.newaxis, :-1, np.newaxis] * vs.maskU[:, 1:, :] \
        * vs.maskU[:, :-1, :] * vs.cosu[np.newaxis, :-1, np.newaxis]
    if vs.enable_noslip_lateral:
        vs.flux_north[:, :-1] += 2 * fxa * vs.u[:, 1:, :, vs.tau] /
            vs.dyu[np.newaxis, :-1, np.newaxis] \
            * vs.maskU[:, 1:] * (1 - vs.maskU[:, :-1]) * vs.cosu[np.newaxis,
```

```python
            :-1, np.newaxis]\
        - 2 * fxa * vs.u[:, :-1, :, vs.tau] / vs.dyu[np.newaxis, :-1,
            np.newaxis] \
        * (1 - vs.maskU[:, 1:]) * vs.maskU[:, :-1] * vs.cosu[np.newaxis,
            :-1, np.newaxis]
    vs.flux_east[-1, :, :] = 0.
    vs.flux_north[:, -1, :] = 0.

    del2 = np.zeros((vs.nx + 4, vs.ny + 4, vs.nz), dtype=vs.default_float_type)
    del2[1:, 1:, :] = (vs.flux_east[1:, 1:, :] - vs.flux_east[:-1, 1:, :]) \
        / (vs.cost[np.newaxis, 1:, np.newaxis] * vs.dxu[1:, np.newaxis,
            np.newaxis]) \
        + (vs.flux_north[1:, 1:, :] - vs.flux_north[1:, :-1, :]) \
        / (vs.cost[np.newaxis, 1:, np.newaxis] * vs.dyt[np.newaxis, 1:,
            np.newaxis])

    vs.flux_east[:-1, :, :] = fxa * (del2[1:, :, :] - del2[:-1, :, :]) \
        / (vs.cost[np.newaxis, :, np.newaxis] * vs.dxt[1:, np.newaxis,
            np.newaxis]) \
        * vs.maskU[1:, :, :] * vs.maskU[:-1, :, :]
    vs.flux_north[:, :-1, :] = fxa * (del2[:, 1:, :] - del2[:, :-1, :]) \
        / vs.dyu[np.newaxis, :-1, np.newaxis] * vs.maskU[:, 1:, :] \
        * vs.maskU[:, :-1, :] * vs.cosu[np.newaxis, :-1, np.newaxis]
    if vs.enable_noslip_lateral:
        vs.flux_north[:,:-1,:] += 2 * fxa * del2[:, 1:, :] / vs.dyu[np.newaxis,
            :-1, np.newaxis] \
            * vs.maskU[:, 1:, :] * (1 - vs.maskU[:, :-1, :]) *
                vs.cosu[np.newaxis, :-1, np.newaxis] \
            - 2 * fxa * del2[:, :-1, :] / vs.dyu[np.newaxis, :-1, np.newaxis] \
            * (1 - vs.maskU[:, 1:, :]) * vs.maskU[:, :-1, :] *
                vs.cosu[np.newaxis, :-1, np.newaxis]
    vs.flux_east[-1, :, :] = 0.
    vs.flux_north[:, -1, :] = 0.

    """
    update tendency
    """
    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, :])
                                                  / (vs.cost[np.newaxis, 2:-2,
                                                      np.newaxis] *
                                                      vs.dxu[2:-2, np.newaxis,
                                                      np.newaxis])
                                                  + (vs.flux_north[2:-2, 2:-2,
                                                      :] - vs.flux_north[2:-2,
                                                      1:-3, :])
                                                  / (vs.cost[np.newaxis, 2:-2,
                                                      np.newaxis] *
                                                      vs.dyt[np.newaxis, 2:-2,
```

```
                                                        np.newaxis]))
    if vs.enable_conserve_energy:
        """
        diagnose dissipation by lateral friction
        """
        if vs.enable_cyclic_x:
            cyclic.setcyclic_x(vs.flux_east)
            cyclic.setcyclic_x(vs.flux_north)
        diss = np.zeros((vs.nx + 4, vs.ny + 4, vs.nz),
            dtype=vs.default_float_type)
        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, :]
                                     + (vs.u[1:-2, 2:-2, :, vs.tau] - vs.u[:-3,
                                         2:-2, :, vs.tau]) * vs.flux_east[:-3, 2:-2,
                                         :]) \
            / (vs.cost[np.newaxis, 2:-2, np.newaxis] * vs.dxu[1:-2, np.newaxis,
                np.newaxis]) \
            - 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, :]
                  + (vs.u[1:-2, 2:-2, :, vs.tau] - vs.u[1:-2, 1:-3, :, vs.tau])
                      * vs.flux_north[1:-2, 1:-3, :]) \
            / (vs.cost[np.newaxis, 2:-2, np.newaxis] * vs.dyt[np.newaxis, 2:-2,
                np.newaxis])
        vs.K_diss_h[...] = 0.
        vs.K_diss_h[...] += numerics.calc_diss(vs, diss, 'U')

    """
    Meridional velocity
    """
    vs.flux_east[:-1, :, :] = fxa * (vs.v[1:, :, :, vs.tau] - vs.v[:-1, :, :,
        vs.tau]) \
        / (vs.cosu[np.newaxis, :, np.newaxis] * vs.dxu[:-1, np.newaxis,
            np.newaxis]) \
        * vs.maskV[1:, :, :] * vs.maskV[:-1, :, :]
    if vs.enable_noslip_lateral:
        vs.flux_east[:-1, :, :] += 2 * fxa * vs.v[1:, :, :, vs.tau] /
            (vs.cosu[np.newaxis, :, np.newaxis] * vs.dxu[:-1, np.newaxis,
            np.newaxis]) \
            * vs.maskV[1:, :, :] * (1 - vs.maskV[:-1, :, :]) \
            - 2 * fxa * vs.v[:-1, :, :, vs.tau] / (vs.cosu[np.newaxis, :,
                np.newaxis] * vs.dxu[:-1, np.newaxis, np.newaxis]) \
            * (1 - vs.maskV[1:, :, :]) * vs.maskV[:-1, :, :]
    vs.flux_north[:, :-1, :] = fxa * (vs.v[:, 1:, :, vs.tau] - vs.v[:, :-1, :,
        vs.tau]) \
        / vs.dyt[np.newaxis, 1:, np.newaxis] * vs.cost[np.newaxis, 1:,
            np.newaxis] \
        * vs.maskV[:, :-1, :] * vs.maskV[:, 1:, :]
    vs.flux_east[-1, :, :] = 0.
    vs.flux_north[:, -1, :] = 0.
```

```python

    del2[1:, 1:, :] = (vs.flux_east[1:, 1:, :] - vs.flux_east[:-1, 1:, :]) \
        / (vs.cosu[np.newaxis, 1:, np.newaxis] * vs.dxt[1:, np.newaxis, np.newaxis]) \
        + (vs.flux_north[1:, 1:, :] - vs.flux_north[1:, :-1, :]) \
        / (vs.dyu[np.newaxis, 1:, np.newaxis] * vs.cosu[np.newaxis, 1:, np.newaxis])

    vs.flux_east[:-1, :, :] = fxa * (del2[1:, :, :] - del2[:-1, :, :]) \
        / (vs.cosu[np.newaxis, :, np.newaxis] * vs.dxu[:-1, np.newaxis, np.newaxis]) \
        * vs.maskV[1:, :, :] * vs.maskV[:-1, :, :]
    if vs.enable_noslip_lateral:
        vs.flux_east[:-1, :, :] += 2 * fxa * del2[1:, :, :] / (vs.cosu[np.newaxis, :, np.newaxis] * vs.dxu[:-1, np.newaxis, np.newaxis]) \
            * vs.maskV[1:, :, :] * (1 - vs.maskV[:-1, :, :]) \
            - 2 * fxa * del2[:-1, :, :] / (vs.cosu[np.newaxis, :, np.newaxis] * vs.dxu[:-1, np.newaxis, np.newaxis]) \
            * (1 - vs.maskV[1:, :, :]) * vs.maskV[:-1, :, :]
    vs.flux_north[:, :-1, :] = fxa * (del2[:, 1:, :] - del2[:, :-1, :]) \
        / vs.dyt[np.newaxis, 1:, np.newaxis] * vs.cost[np.newaxis, 1:, np.newaxis] \
        * vs.maskV[:, :-1, :] * vs.maskV[:, 1:, :]
    vs.flux_east[-1, :, :] = 0.
    vs.flux_north[:, -1, :] = 0.

    """
    update tendency
    """
    vs.dv_mix[2:-2, 2:-2, :] += -vs.maskV[2:-2, 2:-2, :] * ((vs.flux_east[2:-2, 2:-2, :] - vs.flux_east[1:-3, 2:-2, :])
                                                           / (vs.cosu[np.newaxis, 2:-2, np.newaxis] * vs.dxt[2:-2, np.newaxis, np.newaxis])
                                                           + (vs.flux_north[2:-2, 2:-2, :] - vs.flux_north[2:-2, 1:-3, :])
                                                           / (vs.dyu[np.newaxis, 2:-2, np.newaxis] * vs.cosu[np.newaxis, 2:-2, np.newaxis]))

    if vs.enable_conserve_energy:
        """
        diagnose dissipation by lateral friction
        """
```

```
        if vs.enable_cyclic_x:
            cyclic.setcyclic_x(vs.flux_east)
            cyclic.setcyclic_x(vs.flux_north)
        diss[2:-2, 1:-2, :] = -0.5 * ((vs.v[3:-1, 1:-2, :, vs.tau] - vs.v[2:-2, 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, :]) \
            / (vs.cosu[np.newaxis, 1:-2, np.newaxis] * vs.dxt[2:-2, np.newaxis, np.newaxis]) \
            - 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, :]
                + (vs.v[2:-2, 1:-2, :, vs.tau] - vs.v[2:-2, :-3, :, vs.tau]) * vs.flux_north[2:-2, :-3, :]) \
            / (vs.cosu[np.newaxis, 1:-2, np.newaxis] * vs.dyu[np.newaxis, 1:-2, np.newaxis])
        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 to runs was the value of enable noslip lateral.

```python
#!/usr/bin/env python

import veros
import veros.tools


class ACC(veros.Veros):
    """A model using spherical coordinates with a partially closed domain
        representing the Atlantic and ACC.

    Wind forcing over the channel part and buoyancy relaxation drive a
        large-scale meridional overturning circulation.

    This setup demonstrates:
     - setting up an idealized geometry
     - updating surface forcings
     - basic usage of diagnostics

    `Adapted from pyOM2
        <https://wiki.cen.uni-hamburg.de/ifm/TO/pyOM2/ACC%202>`_.
    """
    @veros.veros_method
    def set_parameter(self):
        self.identifier = "acc_ns"

        self.nx, self.ny, self.nz = 30, 42, 15
        self.dt_mom = 4800
        self.dt_tracer = 86400 / 2.
        self.runlen = 86400 * 365 * 100

        self.coord_degree = True
        self.enable_cyclic_x = True

        self.congr_epsilon = 1e-12
        self.congr_max_iterations = 5000

        self.enable_neutral_diffusion = True
        self.K_iso_0 = 1000.0
        self.K_iso_steep = 500.0
        self.iso_dslope = 0.005
        self.iso_slopec = 0.01
        self.enable_skew_diffusion = True

        self.enable_noslip_lateral = True
```

```

        self.enable_hor_friction = True
        self.A_h = (2 * self.degtom)**3 * 2e-11
        self.enable_hor_friction_cos_scaling = True
        self.hor_friction_cosPower = 1

        self.enable_bottom_friction = True
        self.r_bot = 1e-5

        self.enable_implicit_vert_friction = True

        self.enable_tke = True
        self.c_k = 0.1
        self.c_eps = 0.7
        self.alpha_tke = 30.0
        self.mxl_min = 1e-8
        self.tke_mxl_choice = 2
        # self.enable_tke_superbee_advection = True

        self.K_gm_0 = 1000.0
        self.enable_eke = True
        self.eke_k_max = 1e4
        self.eke_c_k = 0.4
        self.eke_c_eps = 0.5
        self.eke_cross = 2.
        self.eke_crhin = 1.0
        self.eke_lmin = 100.0
        self.enable_eke_superbee_advection = True
        self.enable_eke_isopycnal_diffusion = True

        self.enable_idemix = True
        self.enable_idemix_hor_diffusion = True
        self.enable_eke_diss_surfbot = True
        self.eke_diss_surfbot_frac = 0.2
        self.enable_idemix_superbee_advection = True

        self.eq_of_state_type = 3

    @veros.veros_method
    def set_grid(self):
        ddz = np.array([50., 70., 100., 140., 190., 240., 290., 340.,
                       390., 440., 490., 540., 590., 640., 690.])
        self.dxt[...] = 2.0
        self.dyt[...] = 2.0
        self.x_origin = 0.0
        self.y_origin = -40.0
        self.dzt[...] = ddz[::-1] / 2.5

    @veros.veros_method
```

```python
    def set_coriolis(self):
        self.coriolis_t[:, :] = 2 * self.omega * np.sin(self.yt[None, :] / 180. * self.pi)

    @veros.veros_method
    def set_topography(self):
        x, y = np.meshgrid(self.xt, self.yt, indexing="ij")
        self.kbot[...] = np.logical_or(x > 1.0, y < -20).astype(np.int)

    @veros.veros_method
    def set_initial_conditions(self):
        # initial conditions
        self.temp[:, :, :, 0:2] = ((1 - self.zt[None, None, :] / self.zw[0]) * 15 * self.maskT)[..., None]
        self.salt[:, :, :, 0:2] = 35.0 * self.maskT[..., None]

        # wind stress forcing
        taux = np.zeros(self.ny + 4, dtype=self.default_float_type)
        taux[self.yt < -20] = 1e-4 * np.sin(self.pi * (self.yu[self.yt < -20] - self.yu.min()) / (-20.0 - self.yt.min()))
        taux[self.yt > 10] = 1e-4 * (1 - np.cos(2 * self.pi * (self.yu[self.yt > 10] - 10.0) / (self.yu.max() - 10.0)))
        self.surface_taux[:, :] = taux * self.maskU[:, :, -1]

        # surface heatflux forcing
        self._t_star = 15 * np.ones(self.ny + 4, dtype=self.default_float_type)
        self._t_star[self.yt < -20] = 15 * (self.yt[self.yt < -20] - self.yt.min()) / (-20 - self.yt.min())
        self._t_star[self.yt > 20] = 15 * (1 - (self.yt[self.yt > 20] - 20) / (self.yt.max() - 20))
        self._t_rest = self.dzt[None, -1] / (30. * 86400.) * self.maskT[:, :, -1]

        if self.enable_tke:
            self.forc_tke_surface[2:-2, 2:-2] = np.sqrt((0.5 * (self.surface_taux[2:-2, 2:-2] + self.surface_taux[1:-3, 2:-2]))**2
                                                        + (0.5 * (self.surface_tauy[2:-2, 2:-2] + self.surface_tauy[2:-2, 1:-3]))**2)**(1.5)

        if self.enable_idemix:
            self.forc_iw_bottom[...] = 1e-6 * self.maskW[:, :, -1]
            self.forc_iw_surface[...] = 1e-7 * self.maskW[:, :, -1]

    @veros.veros_method
    def set_forcing(self):
```

```python
        self.forc_temp_surface[...] = self._t_rest * (self._t_star -
            self.temp[:, :, -1, self.tau])

    @veros.veros_method
    def set_diagnostics(self):
        self.diagnostics["snapshot"].output_frequency = 86400 * 365
        self.diagnostics["averages"].output_variables = (
            "salt", "temp", "u", "v", "w", "psi", "surface_taux", "surface_tauy"
        )
        self.diagnostics["averages"].output_frequency = 100 * 86400.
        self.diagnostics["averages"].sampling_frequency = self.dt_tracer * 10
        self.diagnostics["overturning"].output_frequency = 365 * 86400. / 48.
        self.diagnostics["overturning"].sampling_frequency = self.dt_tracer * 10
        self.diagnostics["tracer_monitor"].output_frequency = 365 * 86400. / 12.
        self.diagnostics["energy"].output_frequency = 365 * 86400. / 48
        self.diagnostics["energy"].sampling_frequency = self.dt_tracer * 10

    def after_timestep(self):
        pass


@veros.tools.cli
def run(*args, **kwargs):
    simulation = ACC(*args, **kwargs)
    simulation.setup()
    simulation.run()


if __name__ == "__main__":
    run()
```

Apx. C; Code for Plots

```python
import numpy as np
import matplotlib.pyplot as plt
from netCDF4 import Dataset

def theory(_save=False):
    """
    The theoretical western boundary current.
    """
    deltam = 1
    x = np.arange(0,100,0.1)
    # No slip
    v1 = np.exp(-x/(2*deltam))* np.sin(np.sqrt(3)*x/(2*deltam))
    # Free slip
```

```python
    v2 = np.exp(-x/(2*deltam))*(np.cos(np.sqrt(3)*x/(2*deltam)) + 1/np.sqrt(3)*
        np.sin(np.sqrt(3)*x/(2*deltam)) )
    fig, ax = plt.subplots(1,1,figsize = (3.2,2.9))
    ax.plot(x,v1,label='No slip')
    ax.plot(x,v2,'r',label='Free slip')
    ax.plot(x,[0]*len(x),'--',color='gray',label='')
    ax.legend()
    ax.set_xlim([0,20])
    ax.set_xlabel(r'x/ $\delta_m$')
    ax.set_ylabel(r'$v/ \psi_I$')
    #plt.show()
    if _save:
       plt.savefig('theory.pdf')
    return(fig)

def forcing(_save =False):
    """
    A plot of the wind force in respectively the x and y direction.
    """
    vmax_tot = np.amax([np.amax(surface_taux),np.amax(surface_tauy)])
    vmin_tot = np.amin([np.amin(surface_taux),np.amin(surface_tauy)])

    fig, ax = plt.subplots(1,2,figsize = (5.5,2.9))

    viz = ax[0].imshow(surface_taux[-1,:,:], extent =
        [xu[0],xu[-1],yt[0],yt[-1]], origin = 'lower', interpolation='nearest',
        vmin = vmin_tot, vmax = vmax_tot)
    ax[0].set_xlabel("Longitude")
    ax[0].set_title("Surface tau x [m²/s] ")
    ax[0].set_ylabel("Latitude")
    tauy=ax[1].imshow(surface_tauy[-1,:,:], extent =
        [xu[0],xu[-1],yt[0],yt[-1]], origin = 'lower', interpolation='nearest',
        vmin = vmin_tot, vmax = vmax_tot)
    ax[1].set_xlabel("Longitude")
    ax[1].set_title("Surface tau y [m²/s]")
    fig.tight_layout()
    fig.subplots_adjust(right=0.75)

    cbar_ax = fig.add_axes([0.80, 0.20, 0.05, 0.68])
    fig.colorbar(tauy, cax = cbar_ax)

    if _save:
       plt.savefig('forcing.pdf')
    return fig

def time_series(_time_bound,_save=False):
    """
    The ACC integrated and plotted as a time series to show convergence.
```

```python
    And the value of the velocity of the western boundary current at the boundary.
    """
    # For the ACC
    # Cross area.
    delta_z = -np.append((zw[0]-zw[1])-20,(zw[:-1] - zw[1:]))
    radius = 6370e3 # Earth radius in m
    degtom = radius / 180. * np.pi # Conversion degrees latitude to meter
    delta_y = np.append(-42.0,yu)
    cosu = np.cos(delta_y * np.pi / 180.)
    delta_y = - (delta_y[:-1]*cosu[:-1] - delta_y[1:]*cosu[1:])*degtom
    area_u = delta_z[:,np.newaxis]*delta_y[np.newaxis,:]

    cut_off_south = 11
    cur = np.mean(u[:,:,:cut_off_south,5:25], axis=3)
    cur_ns = np.mean(u_ns[:,:,:cut_off_south,5:25], axis=3)
    cur_time_series = np.sum(np.sum(cur*area_u[np.newaxis,:,:cut_off_south], axis=2),axis=1) /1000000
    cur_time_series_ns = np.sum(np.sum(cur_ns*area_u[np.newaxis,:,:cut_off_south], axis=2),axis=1)/1000000

    # For the WBC.
    lower = 28
    depth = -3
    first_idx_for_wbc= 2

    fig,axes = plt.subplots(1,2, figsize = (5.5,2.9))

    axes[0].plot(Time[:_time_bound[1]],cur_time_series[:_time_bound[1]], label = 'fs')
    axes[0].plot(Time_ns[:_time_bound[1]],cur_time_series_ns[:_time_bound[1]], label = 'ns')
    axes[0].set_title('ACC integrated')
    axes[0].set_xlabel('Time [days]')
    axes[0].set_ylabel('Transport [Sv]')
    axes[0].legend()

    axes[1].plot(Time[:_time_bound[1]], v[:_time_bound[1],depth,lower,first_idx_for_wbc], label = 'fs')
    axes[1].plot(Time_ns[:_time_bound[1]], v_ns[:_time_bound[1],depth,lower,first_idx_for_wbc], label = 'ns')
    axes[1].set_title('v '+str(yu[lower])+' deg, depth ' +str(zt[depth])+' m')
    axes[1].set_xlabel('Time [days]')
    axes[1].set_ylabel('v [m/s]')
    axes[1].legend()

    fig.tight_layout()

```

```
    if _save:
        plt.savefig('AAC_and_WBC_ts.pdf')
    return(fig)

def acc_comparison(_time_bound,_save=False):
    """
    A comparison of the structure of the ACC.
    """
    cut_off_south = 11
    cur = np.mean(np.sum(u[_time_bound[0]:_time_bound[1],:,:cut_off_south,5:25], axis=3) / 20.0, axis = 0)
    cur_ns = np.mean(np.sum(u_ns[_time_bound[0]:_time_bound[1],:,:cut_off_south,5:25], axis=3) / 20.0, axis = 0)

    vmax_tot = np.amax([np.amax(cur),np.amax(cur_ns)])
    vmin_tot = np.amin([np.amin(cur),np.amin(cur_ns)])

    fig_cur, ax_cur = plt.subplots(1,2,figsize = (5.5,2.9))

    con = ax_cur[0].contour(yt[0:11],zt/1000.0, cur, vmin = vmin_tot, vmax = vmax_tot)
    con_ns = ax_cur[1].contour(yt[0:11],zt/1000.0, cur_ns, vmin = vmin_tot, vmax = vmax_tot)

    ax_cur[0].set_title('Free slip <u>')
    ax_cur[1].set_title('No slip <u>')

    ax_cur[0].set_ylabel("Depth [km]")
    ax_cur[0].set_xlabel("Latitude [deg]")
    ax_cur[1].set_xlabel("Latitude [deg]")

    fig_cur.tight_layout()

    fig_cur.subplots_adjust(right=0.8)
    cbar_con = fig_cur.add_axes([0.85, 0.20, 0.05, 0.68])

    fig_cur.colorbar(con, cax = cbar_con)

    if _save:
        plt.savefig('acc_comp.pdf')
    return(fig_cur)

def west_boundary_current(_time_bound,_save=False):
    """
    A comparison of the western boundary current, to be compared with theoretical solution.
    """
```

```
    lower = 28
    depth = -3

    fig_close, ax_close = plt.subplots(1,1, figsize = (5.5,3))

    ax_close.plot(xt,np.mean(v[_time_bound[0]:_time_bound[1],depth,lower,:],axis=0),
        label = 'fs')
    ax_close.plot(xt,np.mean(v_ns[_time_bound[0]:_time_bound[1],depth,lower,:],axis=0),
        label = 'ns')

    ax_close.set_title('v '+str(int(yu[lower]))+'°, depth '
        +str(int(zt[depth]))+' m')
    ax_close.set_ylabel('v [m/s]')
    ax_close.set_xlabel('Longitude')
    ax_close.legend()
    fig_close.tight_layout()
    if _save:
        plt.savefig('westb_cur.pdf')
    return(fig_close)

def psiplot_western_boundary(_time_bound,_save=False):
    """
    A plot of psi at the western boundary.
    I cut of the ACC in order to be able to see the variations in the northern
        basin.
    """
    index_from = 15
    _psi_wa = np.mean(psi[_time_bound[0]:_time_bound[1],index_from:,:], axis =0)
    _psin_wa = np.mean(psi_ns[_time_bound[0]:_time_bound[1],index_from:,:],
        axis =0)
    vmax_psi = np.amax([np.amax(_psi_wa),np.amax(_psin_wa)])
    vmin_psi = np.amin([np.amin(_psi_wa),np.amin(_psin_wa)])

    fig, ax = plt.subplots(2,1,figsize = (5.5,5))

    psip = ax[0].contour(xu,yu[index_from:],(_psi_wa), vmax = vmax_psi, vmin =
        vmin_psi)#, extent = [xu[0],xu[-1],yu[index_a],yu[-1]], origin =
        'lower')
    ax[0].set_title('Free slip')
    ax[0].set_ylabel("Latitude")

    psinp = ax[1].contour(xu,yu[index_from:],(_psin_wa), vmax = vmax_psi, vmin
        = vmin_psi)#, extent = [xu[0],xu[-1],yu[index_a],yu[-1]], origin =
        'lower')
    ax[1].set_title('No slip')
    ax[1].set_ylabel("Latitude")
    ax[1].set_xlabel("Longitude")

    fig.tight_layout()
```

```
    fig.subplots_adjust(right=0.8)

    cbar_ax = fig.add_axes([0.85, 0.15, 0.05, 0.7])
    fig.colorbar(psinp, cax = cbar_ax)
    if _save:
       plt.savefig('psi_comp.pdf')
    return(fig)


if __name__ == '__main__':
    with Dataset("acc.averages.nc", "r") as datafile:
    # read variable "u" and save it to a NumPy array

       xt = datafile.variables["xt"][...]
       xu = datafile.variables["xu"][...]
       yt = datafile.variables["yt"][...]
       yu = datafile.variables["yu"][...]
       zt = datafile.variables["zt"][...]
       zw = datafile.variables["zw"][...]
       temp = datafile.variables["temp"][...]
       Time = datafile.variables["Time"][...]
       surface_taux = datafile.variables["surface_taux"][...]
       surface_tauy = datafile.variables["surface_tauy"][...]

       u = datafile.variables["u"][...]
       v = datafile.variables["v"][...]
       psi = datafile.variables["psi"][...]


    with Dataset("acc_ns.averages.nc", "r") as datafile:
    # read variable "u" and save it to a NumPy array
       Time_ns = datafile.variables["Time"][...]
       u_ns = datafile.variables["u"][...]
       v_ns = datafile.variables["v"][...]
       psi_ns = datafile.variables["psi"][...]
    # Due to unforseen end of Dataset acc_ns I have created:
    time_bound = [len(Time_ns)-10, len(Time_ns)-1]
    #print(u[360,-1,6,6])
    #print(np.abs(u[360,:,:,:]-u_ns[360,:,:,:]) < 0.0001)

    theo = theory()
    sf = forcing()
    ts = time_series(time_bound)
    acc_comp = acc_comparison(time_bound)
    wbc= west_boundary_current(time_bound)
    fig_psi_wa = psiplot_western_boundary(time_bound)
    plt.show()
```

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

```
#include "options.inc"



      subroutine momentum_tendency
c=======================================================================
c       tendencies for momentum stored in F_u, F_v and F_w
c=======================================================================
      use cpflame_module
      implicit none
      integer :: i,j,k,js,je
      real :: adv_fe(imt,jmt,km), adv_ft(imt,jmt,km)
      real :: adv_fn(imt,jmt,km), diff_fn(imt,jmt,km)
      real :: diff_fe(imt,jmt,km),diff_ft(imt,jmt,km)
      real :: fxa,fxb,fxc

#ifdef enable_smagorinsky_friction
      call smagorinsky
#endif

      js=max(2,js_pe); je = min(je_pe,jmt-1)
c-----------------------------------------------------------------------------------
c      Zonal momentum equation: advective and diffusive fluxes
c-----------------------------------------------------------------------------------
      adv_fe(:,js_pe:je_pe,:)=0.0;
      adv_fn(:,js_pe:je_pe,:)=0.0;
      adv_ft(:,js_pe:je_pe,:)=0.0;
      if (enable_4th_mom_advection) then
       call setcyclic3D_j2(u(:,:,:,1,tau) )
       call adv_flux_u_4th(adv_fe,adv_fn,adv_ft)
      elseif (enable_quicker_mom_advection) then
       call setcyclic3D_j2(u(:,:,:,1,taum1) )
       call adv_flux_u_quicker(adv_fe,adv_fn,adv_ft)
      else
       call adv_flux_u_2nd(adv_fe,adv_fn,adv_ft)
      endif

      diff_fe(:,js_pe:je_pe,:)=0.0;
      diff_fn(:,js_pe:je_pe,:)=0.0;
#ifdef enable_smagorinsky_friction
      call smagorinsky_fric_u(diff_fe,diff_fn)
#else
      call harm_hfric_u(diff_fe,diff_fn)
#endif
c-----------------------------------------------------------------------------------
c      vertical friction
```

```
c-----------------------------------------------------------------------------------
      diff_ft(:,js_pe:je_pe,:)=0.0;
      do k=1,km-1
       do j=js,je
        do i=2,imt-1
          diff_ft(i,j,k)=A_v*(u(i,j,k+1,1,taum1)-u(i,j,k,1,taum1))/dz
     &                  *maskU(i,j,k+1)*maskU(i,j,k)
        enddo
       enddo
      enddo
c-----------------------------------------------------------------------------------
c      add surface and bottom boundary conditions
c-----------------------------------------------------------------------------------
      do j=js,je
       diff_ft(:,j,km-1)= surf_tau(:,j,1)*maskU(:,j,km-1)
       diff_ft(:,j,1 )= bott_tau(:,j,1)*maskU(:,j,2 )
      enddo
c-----------------------------------------------------------------------------------
c    account for no slip at bottom
c-----------------------------------------------------------------------------------
      if (enable_bottom_noslip) then
       do k=1,km-1
        do j=js,je
         diff_ft(:,j,k)=diff_ft(:,j,k)+2*A_v*u(:,j,k+1,1,taum1)/dz
     &                  *(1-maskU(:,j,k))*maskU(:,j,k+1)
        enddo
       enddo
      endif
c-----------------------------------------------------------------------------------
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
c-----------------------------------------------------------------------------------
      do k=2,km-1
       do j=js,je
        do i=2,imt-1
         fu(i,j,k)= maskU(i,j,k)*(
     &    -(adv_fe(i,j,k) - adv_fe(i-1,j,k) )/dx
     &    -(adv_fn(i,j,k) - adv_fn(i,j-1,k) )/dx
     &    -(adv_ft(i,j,k) - adv_ft(i,j,k-1) )/dz
     &    +(diff_ft(i,j,k) - diff_ft(i,j,k-1))/dz
     &    +(diff_fe(i,j,k) - diff_fe(i-1,j,k))/dx
     &    +(diff_fn(i,j,k) - diff_fn(i,j-1,k))/dx
     &                            )
        enddo
       enddo
      enddo
c-----------------------------------------------------------------------------------
c      Meridional momentum equation: advective and diffusive fluxes
c-----------------------------------------------------------------------------------
      adv_fe(:,js_pe:je_pe,:)=0.0;
```

```
      adv_fn(:,js_pe:je_pe,:)=0.0;
      adv_ft(:,js_pe:je_pe,:)=0.0;
      if (enable_4th_mom_advection) then
       call setcyclic3D_j2(u(:,:,:,2,tau) )
       call adv_flux_v_4th(adv_fe,adv_fn,adv_ft)
      elseif (enable_quicker_mom_advection) then
       call setcyclic3D_j2(u(:,:,:,2,taum1) )
       call adv_flux_v_quicker(adv_fe,adv_fn,adv_ft)
      else
       call adv_flux_v_2nd(adv_fe,adv_fn,adv_ft)
      endif

      diff_fe(:,js_pe:je_pe,:)=0.0;
      diff_fn(:,js_pe:je_pe,:)=0.0;
#ifdef enable_smagorinsky_friction
      call smagorinsky_fric_v(diff_fe,diff_fn)
#else
      call harm_hfric_v(diff_fe,diff_fn)
#endif
c-----------------------------------------------------------------------------------
c      vertical friction
c-----------------------------------------------------------------------------------
      diff_ft(:,js_pe:je_pe,:)=0.0;
      do k=1,km-1
       do j=js,je
        do i=2,imt-1
          diff_ft(i,j,k)=A_v*(u(i,j,k+1,2,taum1)-u(i,j,k,2,taum1))/dz
     &                 *maskV(i,j,k+1)*maskV(i,j,k)
        enddo
       enddo
      enddo
c-----------------------------------------------------------------------------------
c      add surface and bottom boundary conditions
c-----------------------------------------------------------------------------------
      do j=js,je
       diff_ft(:,j,km-1)= surf_tau(:,j,2)*maskV(:,j,km-1)
       diff_ft(:,j,1 )= bott_tau(:,j,2)*maskV(:,j,2)
      enddo
c-----------------------------------------------------------------------------------
c    account for no slip at bottom
c-----------------------------------------------------------------------------------
      if (enable_bottom_noslip) then
       do k=1,km-1
        do j=js,je
         diff_ft(:,j,k)=diff_ft(:,j,k)+2*A_v*u(:,j,k+1,2,taum1)/dz
     &                 *(1-maskV(:,j,k))*maskV(:,j,k+1)
        enddo
       enddo
      endif
```

```fortran
c--------------------------------------------------------------------------------
c     F_v = - u v_x - v v_y - w v_z + A_h v_xx + A_v v_zz - f_vert u
c--------------------------------------------------------------------------------
     do k=2,km-1
      do j=js,je
       do i=2,imt-1
        fv(i,j,k)= maskV(i,j,k)*(
    &    -(adv_fe(i,j,k) - adv_fe(i-1,j,k) )/dx
    &    -(adv_fn(i,j,k) - adv_fn(i,j-1,k) )/dx
    &    -(adv_ft(i,j,k) - adv_ft(i,j,k-1) )/dz
    &    +(diff_ft(i,j,k) - diff_ft(i,j,k-1))/dz
    &    +(diff_fe(i,j,k) - diff_fe(i-1,j,k))/dx
    &    +(diff_fn(i,j,k) - diff_fn(i,j-1,k))/dx
    &                             )
       enddo
      enddo
     enddo
c--------------------------------------------------------------------------------
c       vertical momentum equation: advective and diffusive fluxes
c--------------------------------------------------------------------------------
     if (.not. enable_hydrostatic) then
      adv_fe(:,js_pe:je_pe,:)=0.0;
      adv_fn(:,js_pe:je_pe,:)=0.0;
      adv_ft(:,js_pe:je_pe,:)=0.0;
      if (enable_4th_mom_advection) then
       call setcyclic3D_j2(u(:,:,:,3,tau) )
       call adv_flux_w_4th(adv_fe,adv_fn,adv_ft)
      elseif (enable_quicker_mom_advection) then
       call setcyclic3D_j2(u(:,:,:,3,taum1) )
       call adv_flux_w_quicker(adv_fe,adv_fn,adv_ft)
      else
       call adv_flux_w_2nd(adv_fe,adv_fn,adv_ft)
      endif
      diff_fe(:,js_pe:je_pe,:)=0.0;
      diff_fn(:,js_pe:je_pe,:)=0.0;
      call harm_hfric_w(diff_fe,diff_fn)
c--------------------------------------------------------------------------------
c      vertical friction
c--------------------------------------------------------------------------------
      diff_ft(:,js_pe:je_pe,:)=0.0;
      do k=1,km-1
       do j=js,je
        do i=2,imt-1
         diff_ft(i,j,k)=
    &    A_v*(u(i,j,k+1,3,taum1)-u(i,j,k,3,taum1))/dz
    &         *maskW(i,j,k+1)*maskW(i,j,k)
        enddo
       enddo
      enddo
```

```
c-----------------------------------------------------------------------
c       F_w = - u w_x - v w_y - w w_z + A_h w_xx + A_v w_zz +f_hor u
c-----------------------------------------------------------------------
       do k=2,km-1
        do j=js,je
         do i=2,imt-1
          fw(i,j,k)= maskW(i,j,k)*(
     &     -(adv_fe(i,j,k) - adv_fe(i-1,j,k) )/dx
     &     -(adv_fn(i,j,k) - adv_fn(i,j-1,k) )/dx
     &     -(adv_ft(i,j,k) - adv_ft(i,j,k-1) )/dz
     &     +(diff_fe(i,j,k) - diff_fe(i-1,j,k))/dx
     &     +(diff_fn(i,j,k) - diff_fn(i,j-1,k))/dx
     &     +(diff_ft(i,j,k) - diff_ft(i,j,k-1))/dz
     &                          )
         enddo
        enddo
       enddo
      endif
c-----------------------------------------------------------------------
c     Add coriolis force to F_u, F_v and F_w
c-----------------------------------------------------------------------
      call coriolis_force
c-----------------------------------------------------------------------
c      add bottom drag : u_t = - c_D u
c-----------------------------------------------------------------------
      if (enable_bottom_stress) then
       do j=js,je
        do i=2,imt-1
         k=max(1,kmu(i,j))
         fxa = cdbot*u(i,j,k,1,taum1)
         fu(i,j,k) = fu(i,j,k)-maskU(i,j,k)*fxa
         k=max(1,kmv(i,j))
         fxa = cdbot*u(i,j,k,2,taum1)
         fv(i,j,k) = fv(i,j,k)-maskV(i,j,k)*fxa
        enddo
       enddo
      endif
c-----------------------------------------------------------------------
c      add interior drag : u_t = - c_D u
c-----------------------------------------------------------------------
      if (enable_interior_stress) then
       do k=2,km-1
        do j=js,je
         do i=2,imt-1
          fxa = cdint*u(i,j,k,1,taum1)
          fu(i,j,k) = fu(i,j,k)-maskU(i,j,k)*fxa
          fxa = cdint*u(i,j,k,2,taum1)
          fv(i,j,k) = fv(i,j,k)-maskV(i,j,k)*fxa
         enddo
```

```
          enddo
         enddo
        endif
c-----------------------------------------------------------------------
c      add biharmonic friction
c-----------------------------------------------------------------------
        if (enable_biharmonic_friction) then
         call biha_hfric_u(diff_fe,diff_fn)
         call biha_hfric_v(diff_fe,diff_fn)
         if (.not.enable_hydrostatic) call biha_hfric_w(diff_fe,diff_fn)
        endif
        if (enable_vert_biha_friction) then
         call biha_vfric (diff_ft,maskU,fu,1)
         call biha_vfric (diff_ft,maskV,fv,2)
         if (.not.enable_hydrostatic) then
          call biha_vfric(diff_ft,maskW,fw,3)
         endif
        endif
c-----------------------------------------------------------------------
c       Nudging terms
c-----------------------------------------------------------------------
        call momentum_restoring_zones
c-----------------------------------------------------------------------
c       boundary exchange for result
c-----------------------------------------------------------------------
        call border_exchg3D(fu,1)
        call setcyclic3D(fu)
        call border_exchg3D(fv,1)
        call setcyclic3D(fv)
        if (.not. enable_hydrostatic) then
         call border_exchg3D(fw,1)
         call setcyclic3D(fw)
        endif
        end subroutine momentum_tendency





        subroutine coriolis_force
c=================================================================
c      Add coriolis force to F_u, F_v and F_w
c=================================================================
        use cpflame_module
        implicit none
        integer :: i,j,k,js,je
        js=max(2,js_pe); je = min(je_pe,jmt-1)

c-----------------------------------------------------------------------
```

```
c      F_u = A_h u_xx + A_v u_zz + f_vert v - f_hor w +...
c-----------------------------------------------------------------------------------------------
      do k=2,km-1
       do j=js,je
        do i=2,imt-1
         fu(i,j,k)=fu(i,j,k)+maskU(i,j,k)*
     &     coriolis_t(j)*(u(i,j ,k,2,tau)+u(i+1,j ,k,2,tau)+
     &                    u(i,j-1,k,2,tau)+u(i+1,j-1,k,2,tau))/4.0
        enddo
       enddo
      enddo
      if (.not. enable_hydrostatic) then
       do k=2,km-1
        do j=js,je
         do i=2,imt-1
          fu(i,j,k)=fu(i,j,k)-maskU(i,j,k)*coriolis_hor(j)*
     &           (u(i,j,k ,3,tau) + u(i+1,j,k ,3,tau)+
     &            u(i,j,k-1,3,tau) + u(i+1,j,k-1,3,tau) )/4.0
         enddo
        enddo
       enddo
      endif
c-----------------------------------------------------------------------------------------------
c      F_v = A_h v_xx + A_v v_zz - f_vert u
c-----------------------------------------------------------------------------------------------
      do k=2,km-1
       do j=js,je
        do i=2,imt-1
         fv(i,j,k)= fv(i,j,k)-maskV(i,j,k)*
     &     (coriolis_t(j )*(u(i-1,j ,k,1,tau)+u(i,j ,k,1,tau))
     &     +coriolis_t(j+1)*(u(i-1,j+1,k,1,tau)+u(i,j+1,k,1,tau)))/4.0
        enddo
       enddo
      enddo
c-----------------------------------------------------------------------------------------------
c        F_w = - u w_x - v w_y - w w_z + A_h w_xx + A_v w_zz + f_hor u
c-----------------------------------------------------------------------------------------------
      if (.not. enable_hydrostatic) then
       do k=2,km-1
        do j=js,je
         do i=2,imt-1
          fw(i,j,k)=fw(i,j,k)+maskW(i,j,k)*
     &      coriolis_hor(j)*(u(i,j,k ,1,tau)+u(i-1,j,k ,1,tau)+
     &                       u(i,j,k+1,1,tau)+u(i-1,j,k+1,1,tau))/4.0
         enddo
        enddo
       enddo
      endif
      end subroutine coriolis_force
```

```
      subroutine harm_hfric_u(diff_fe,diff_fn)
c=======================================================================
c      horizontal harmonic friction for u
c      diff. fluxes are stored in diff_fe and diff_fn
c      account for no slip boundary condition if requested
c=======================================================================
      use cpflame_module
      implicit none
      integer :: i,j,k,js,je
      real :: diff_fn(imt,jmt,km), diff_fe(imt,jmt,km)

      js=max(2,js_pe); je = min(je_pe,jmt-1)
      do k=2,km-1
       do j=js,je
        do i=1,imt-1
          diff_fe(i,j,k)=A_h*(u(i+1,j,k,1,taum1)-u(i,j,k,1,taum1))/dx
        enddo
       enddo
      enddo
      call setcyclic3D(diff_fe)
      do k=2,km-1
       do j=js,je
        do i=2,imt-1
          diff_fn(i,j,k)=A_h*(u(i,j+1,k,1,taum1)-u(i,j,k,1,taum1))/dx
     &                 *maskU(i,j+1,k)*maskU(i,j,k)
        enddo
       enddo
      enddo
      if (enable_noslip) then
       do j=js-1,je
         diff_fn(:,j,:)=diff_fn(:,j,:)-2*A_h*u(:,j,:,1,taum1)/dx
     &                 *(1-maskU(:,j+1,:))*maskU(:,j,:)
         diff_fn(:,j,:)=diff_fn(:,j,:)+2*A_h*u(:,j+1,:,1,taum1)/dx
     &                 *(1-maskU(:,j,:))*maskU(:,j+1,:)
       enddo
      endif
      call border_exchg3D(diff_fn,1)
      call setcyclic3D(diff_fn)
      end subroutine harm_hfric_u




      subroutine harm_hfric_v(diff_fe,diff_fn)
c=======================================================================
```

```
c      horizontal harmonic friction for v
c      diff. fluxes are stored in diff_fe and diff_fn
c      account for no slip boundary condition if requested
c=======================================================================
      use cpflame_module
      implicit none
      integer :: i,j,k,js,je
      real :: diff_fn(imt,jmt,km), diff_fe(imt,jmt,km)
      js=max(2,js_pe); je = min(je_pe,jmt-1)
      do k=2,km-1
       do j=js,je
        do i=1,imt-1
          diff_fe(i,j,k)=A_h*(u(i+1,j,k,2,taum1)-u(i,j,k,2,taum1))/dx
     &                  *maskV(i+1,j,k)*maskV(i,j,k)
        enddo
       enddo
      enddo
      if (enable_noslip) then
       do j=js,je
       do i=1,imt-1
         diff_fe(i,j,:)=diff_fe(i,j,:)-2*A_h*u(i,j,:,2,taum1)/dx
     &                  *(1-maskV(i+1,j,:))*maskV(i,j,:)
         diff_fe(i,j,:)=diff_fe(i,j,:)+2*A_h*u(i+1,j,:,2,taum1)/dx
     &                  *(1-maskV(i,j,:))*maskV(i+1,j,:)
       enddo
       enddo
      endif
      call setcyclic3D(diff_fe)
      do k=2,km-1
       do j=js,je
        do i=2,imt-1
          diff_fn(i,j,k)=A_h*(u(i,j+1,k,2,taum1)-u(i,j,k,2,taum1) )/dx
        enddo
       enddo
      enddo
      call border_exchg3D(diff_fn,1)
      call setcyclic3D(diff_fn)
      end subroutine harm_hfric_v




      subroutine harm_hfric_w(diff_fe,diff_fn)
c=======================================================================
c      horizontal harmonic friction for w
c      diff. fluxes are stored in diff_fe and diff_fn
c=======================================================================
      use cpflame_module
      implicit none
```

```
      integer :: i,j,k,js,je
      real :: diff_fn(imt,jmt,km), diff_fe(imt,jmt,km)
      js=max(2,js_pe); je = min(je_pe,jmt-1)
      do k=2,km-1
       do j=js,je
        do i=1,imt-1
          diff_fe(i,j,k)=
     &    A_h*(u(i+1,j,k,3,taum1)-u(i,j,k,3,taum1))/dx
     &         *maskW(i+1,j,k)*maskW(i,j,k)
        enddo
       enddo
      enddo
      call setcyclic3D(diff_fe)
      do k=2,km-1
       do j=js,je
        do i=2,imt-1
          diff_fn(i,j,k)=
     &     A_h*(u(i,j+1,k,3,taum1)-u(i,j,k,3,taum1))/dx
     &         *maskW(i,j+1,k)*maskW(i,j,k)
        enddo
       enddo
      enddo
      call border_exchg3D(diff_fn,1)
      call setcyclic3D(diff_fn)
      end subroutine harm_hfric_w
```

```
#include "options.inc"

c=======================================================================
c     Biharmonic friction and diffusion
c=======================================================================

      subroutine biha_hfric_u (diff_fe,diff_fn)
c-----------------------------------------------------------------------------------
c     horizontal biharmonic friction for zonal momentum
c     no slip condition is possible
c-----------------------------------------------------------------------------------
      use cpflame_module
      implicit none
      integer :: i,j,k,js,je
      real :: diff_fe(imt,jmt,km), diff_fn(imt,jmt,km)
      real :: del2(imt,jmt,km),diff_tx,diff_ty,diffx
      DIFF_Tx(i,j,k) = (diff_fe(i,j,k)-diff_fe(i-1,j,k))/dx
      DIFF_Ty(i,j,k) = (diff_fn(i,j,k)-diff_fn(i,j-1,k))/dx

      js=max(2,js_pe); je = min(je_pe,jmt-1)
      diffx = sqrt(abs(ahbi))
      diff_fe(:,js_pe:je_pe,:)=0.0;diff_fn(:,js_pe:je_pe,:)=0.0
      do k=1,km
       do j=js,je
        do i=1,imt-1
         diff_fe(i,j,k)=diffx*(u(i+1,j,k,1,taum1)-u(i,j,k,1,taum1))/dx
        enddo
       enddo
      enddo
      call border_exchg3D(diff_fe,1); call setcyclic3D(diff_fe)
      do k=1,km-1
       do j=js,je
        do i=1,imt
         diff_fn(i,j,k)=diffx*(u(i,j+1,k,1,taum1)-u(i,j,k,1,taum1))/dx
     &                 *maskU(i,j+1,k)*maskU(i,j,k)
        enddo
       enddo
      enddo
      if (enable_noslip) then
       do j=js-1,je
         diff_fn(:,j,:)=diff_fn(:,j,:)-2*diffx*u(:,j,:,1,taum1)/dx
     &                  *(1-maskU(:,j+1,:))*maskU(:,j,:)
     &                              +2*diffx*u(:,j+1,:,1,taum1)/dx
     &                  *(1-maskU(:,j,:))*maskU(:,j+1,:)
       enddo
      endif
      call border_exchg3D(diff_fn,1); call setcyclic3D(diff_fn)
      del2(:,js_pe:je_pe,:)=0.0
```

```
      do k=2,km-1
       do j=js,je
        do i=2,imt-1
      del2(i,j,k) = (DIFF_Tx(i,j,k) + DIFF_Ty(i,j,k))*maskU(i,j,k)
        enddo
       enddo
      enddo
      call border_exchg3D(del2,1); call setcyclic3D(del2)
      diff_fe(:,js_pe:je_pe,:)=0.0;diff_fn(:,js_pe:je_pe,:)=0.0
      do k=2,km-1
       do j=js,je
        do i=1,imt-1
         diff_fe(i,j,k) =diffx*(del2(i+1,j,k)-del2(i,j,k))/dx
        enddo
       enddo
      enddo
      call border_exchg3D(diff_fe,1); call setcyclic3D(diff_fe)
      do k=1,km-1
       do j=js-1,je
        do i=2,imt-1
         diff_fn(i,j,k) = diffx*(del2(i,j+1,k) - del2(i,j,k))/dx
     &                     *maskU(i,j+1,k)*maskU(i,j,k)
        enddo
       enddo
      enddo
      if (enable_noslip) then
       do j=js-1,je
         diff_fn(:,j,:)=diff_fn(:,j,:)-2*diffx*del2(:,j,:)/dx
     &                   *(1-maskU(:,j+1,:))*maskU(:,j,:)
     &                                +2*diffx*del2(:,j+1,:)/dx
     &                   *(1-maskU(:,j,:))*maskU(:,j+1,:)
       enddo
      endif
      call border_exchg3D(diff_fn,1); call setcyclic3D(diff_fn)
      do k=2,km-1
       do j=js,je
        do i=2,imt-1
         fu(i,j,k)=fu(i,j,k)
     &        -maskU(i,j,k)*(DIFF_Tx(i,j,k)+DIFF_Ty(i,j,k))
        enddo
       enddo
      enddo
      end subroutine biha_hfric_u




      subroutine biha_hfric_v (diff_fe,diff_fn)
c---------------------------------------------------------------------------------------------
```

```
c     horizontal biharmonic friction for meridional momentum
c     no slip condition is possible
c---------------------------------------------------------------------------------
      use cpflame_module
      implicit none
      integer :: i,j,k,js,je
      real :: diff_fe(imt,jmt,km), diff_fn(imt,jmt,km)
      real :: del2(imt,jmt,km),diff_tx,diff_ty,diffx
      DIFF_Tx(i,j,k)=(diff_fe(i,j,k)-diff_fe(i-1,j,k))/dx
      DIFF_Ty(i,j,k)=(diff_fn(i,j,k)-diff_fn(i,j-1,k))/dx

      js=max(2,js_pe); je = min(je_pe,jmt-1)
      diffx = sqrt(abs(ahbi))
      diff_fe(:,js_pe:je_pe,:)=0.0;diff_fn(:,js_pe:je_pe,:)=0.0
      do k=1,km
       do j=js,je
        do i=1,imt-1
         diff_fe(i,j,k)=diffx*(u(i+1,j,k,2,taum1)-u(i,j,k,2,taum1))/dx
     &                *maskV(i,j,k)*maskV(i+1,j,k)
        enddo
       enddo
      enddo
      if (enable_noslip) then
       do j=js,je
       do i=1,imt-1
         diff_fe(i,j,:)=diff_fe(i,j,:)-2*diffx*u(i,j,:,2,taum1)/dx
     &                 *(1-maskV(i+1,j,:))*maskV(i,j,:)
     &                            +2*diffx*u(i+1,j,:,2,taum1)/dx
     &                 *(1-maskV(i,j,:))*maskV(i+1,j,:)
       enddo
       enddo
      endif
      call border_exchg3D(diff_fe,1); call setcyclic3D(diff_fe)
      do k=1,km-1
       do j=js,je
        do i=1,imt
         diff_fn(i,j,k)=diffx*(u(i,j+1,k,2,taum1)-u(i,j,k,2,taum1))/dx
        enddo
       enddo
      enddo
      call border_exchg3D(diff_fn,1); call setcyclic3D(diff_fn)
      del2(:,je_pe:je_pe,:)=0.0
      do k=2,km-1
       do j=js,je
        do i=2,imt-1
      del2(i,j,k) = (DIFF_Tx(i,j,k) + DIFF_Ty(i,j,k))*maskV(i,j,k)
        enddo
       enddo
      enddo
```

```
      call border_exchg3D(del2,1); call setcyclic3D(del2)
      diff_fe(:,js_pe:je_pe,:)=0.0;diff_fn(:,js_pe:je_pe,:)=0.0
      do k=2,km-1
       do j=js,je
        do i=1,imt-1
         diff_fe(i,j,k) =diffx*(del2(i+1,j,k)-del2(i,j,k))/dx
     &                  *maskV(i+1,j,k)*maskV(i,j,k)
        enddo
       enddo
      enddo
      if (enable_noslip) then
       do j=js,je
       do i=1,imt-1
         diff_fe(i,j,:)=diff_fe(i,j,:)-2*diffx*del2(i,j,:)/dx
     &                *(1-maskV(i+1,j,:))*maskV(i,j,:)
     &                             +2*diffx*del2(i+1,j,:)/dx
     &                *(1-maskV(i,j,:))*maskV(i+1,j,:)
       enddo
       enddo
      endif
      call border_exchg3D(diff_fe,1); call setcyclic3D(diff_fe)
      do k=1,km-1
       do j=js,je
        do i=2,imt-1
         diff_fn(i,j,k) = diffx*(del2(i,j+1,k) - del2(i,j,k))/dx
        enddo
       enddo
      enddo
      call border_exchg3D(diff_fn,1); call setcyclic3D(diff_fn)
      do k=2,km-1
       do j=js,je
        do i=2,imt-1
          fv(i,j,k)= fv(i,j,k)+maskV(i,j,k)*(
     &            -DIFF_Tx(i,j,k) - DIFF_Ty(i,j,k))
        enddo
       enddo
      enddo
      end subroutine biha_hfric_v





      subroutine biha_hfric_w (diff_fe,diff_fn)
c---------------------------------------------------------------------------------
c     horizontal biharmonic friction for vertical momentum
c---------------------------------------------------------------------------------
      use cpflame_module
      implicit none
```

```fortran
       integer :: i,j,k,js,je
       real :: diff_fe(imt,jmt,km), diff_fn(imt,jmt,km)
       real :: del2(imt,jmt,km),diff_tx,diff_ty,diffx
       DIFF_Tx(i,j,k)=(diff_fe(i,j,k)-diff_fe(i-1,j,k))/dx
       DIFF_Ty(i,j,k)=(diff_fn(i,j,k)-diff_fn(i,j-1,k))/dx

       js=max(2,js_pe); je = min(je_pe,jmt-1)
       diffx = sqrt(abs(ahbi))
       diff_fe(:,js_pe:je_pe,:)=0.0;diff_fn(:,js_pe:je_pe,:)=0.0
       do k=1,km
        do j=js,je
         do i=1,imt-1
          diff_fe(i,j,k)=diffx*(u(i+1,j,k,3,taum1)-u(i,j,k,3,taum1))/dx
      &                 *maskW(i,j,k)*maskW(i+1,j,k)
         enddo
        enddo
       enddo
       call border_exchg3D(diff_fe,1); call setcyclic3D(diff_fe)
       do k=1,km-1
        do j=js,je
         do i=1,imt
          diff_fn(i,j,k)=diffx*(u(i,j+1,k,3,taum1)-u(i,j,k,3,taum1))/dx
      &                 *maskW(i,j,k)*maskW(i,j+1,k)
         enddo
        enddo
       enddo
       call border_exchg3D(diff_fn,1); call setcyclic3D(diff_fn)
       del2(:,je_pe:je_pe,:)=0.0
       do k=2,km-1
        do j=js,je
         do i=2,imt-1
       del2(i,j,k) = (DIFF_Tx(i,j,k) + DIFF_Ty(i,j,k))*maskW(i,j,k)
         enddo
        enddo
       enddo
       call border_exchg3D(del2,1); call setcyclic3D(del2)
       diff_fe(:,js_pe:je_pe,:)=0.0;diff_fn(:,js_pe:je_pe,:)=0.0
       do k=2,km-1
        do j=js,je
         do i=1,imt-1
          diff_fe(i,j,k) =diffx*(del2(i+1,j,k)-del2(i,j,k))/dx
      &                    *maskW(i+1,j,k)*maskW(i,j,k)
         enddo
        enddo
       enddo
       call border_exchg3D(diff_fe,1); call setcyclic3D(diff_fe)
       do k=1,km-1
        do j=js,je
         do i=2,imt-1
```

```
          diff_fn(i,j,k) = diffx*(del2(i,j+1,k) - del2(i,j,k))/dx
     &                     *maskW(i,j+1,k)*maskW(i,j,k)
         enddo
        enddo
       enddo
       call border_exchg3D(diff_fn,1); call setcyclic3D(diff_fn)
       do k=2,km-1
        do j=js,je
         do i=2,imt-1
           fw(i,j,k)= fw(i,j,k)+maskW(i,j,k)*(
     &               -DIFF_Tx(i,j,k) - DIFF_Ty(i,j,k))
         enddo
        enddo
       enddo
       end subroutine biha_hfric_w



       subroutine biha_vfric (diff_ft,mask,ff,n)
c-----------------------------------------------------------------------------------
c      vertical biharmonic friction for momentum component n
c      result is added to ff
c-----------------------------------------------------------------------------------
       use cpflame_module
       implicit none
       integer :: i,j,k,js,je,n
       real,dimension(imt,jmt,km) :: diff_ft,mask,ff
       real :: del2(imt,jmt,km),diff_tz,diffx
       DIFF_Tz(i,j,k)=(diff_ft(i,j,k)-diff_ft(i,j,k-1))/dz

       js=max(2,js_pe); je = min(je_pe,jmt-1)
       diffx = sqrt(abs(Avbi))
       diff_ft(:,je_pe:je_pe,:)=0.0
       do k=1,km-1
        do j=js,je
         do i=2,imt-1
          diff_ft(i,j,k)=diffx*(u(i,j,k+1,n,taum1)-u(i,j,k,n,taum1))/dz
     &                  *mask(i,j,k)*mask(i,j,k+1)
         enddo
        enddo
       enddo
       del2(:,je_pe:je_pe,:)=0.0
       do k=2,km-1
        do j=js,je
         do i=2,imt-1
       del2(i,j,k) = DIFF_Tz(i,j,k)*mask(i,j,k)
         enddo
        enddo
       enddo
```

```fortran
      diff_ft(:,je_pe:je_pe,:)=0.0
      do k=2,km-1
       do j=js,je
        do i=2,imt-1
         diff_ft(i,j,k) =diffx*(del2(i,j,k+1)-del2(i,j,k))/dz
     &                   *mask(i,j,k+1)*mask(i,j,k)
        enddo
       enddo
      enddo
      do k=2,km-1
       do j=js,je
        do i=2,imt-1
          ff(i,j,k)= ff(i,j,k)-mask(i,j,k)*DIFF_Tz(i,j,k)
        enddo
       enddo
      enddo
      end subroutine biha_vfric



      subroutine biha_mix (diff_fe,diff_fn,diff_ft,var)
c-----------------------------------------------------------------------------------
c      horizontal biharmonic diffusion of buoyancy
c      also vertical biha. diff. if requested
c-----------------------------------------------------------------------------------
      use cpflame_module
      implicit none
      integer :: i,j,k,js,je
      real :: var(imt,jmt,km,0:2)
      real :: diff_fe(imt,jmt,km), diff_fn(imt,jmt,km)
      real :: diff_ft(imt,jmt,km) ,diffz
      real :: del2(imt,jmt,km),diff_tx,diff_ty,diffx,diff_tz
      DIFF_Tx(i,j,k)=(diff_fe(i,j,k)-diff_fe(i-1,j,k))/dx
      DIFF_Ty(i,j,k)=(diff_fn(i,j,k)-diff_fn(i,j-1,k))/dx
      DIFF_Tz(i,j,k)=(diff_ft(i,j,k)-diff_ft(i,j,k-1))/dz

      js=max(2,js_pe); je = min(je_pe,jmt-1)
      diffx = sqrt(abs(Khbi))
      diffz = sqrt(abs(Kvbi))
      diff_fe(:,js_pe:je_pe,:)=0.0; diff_fn(:,js_pe:je_pe,:)=0.0
      do k=1,km
       do j=js,je
        do i=1,imt-1
         diff_fe(i,j,k)=diffx*(var(i+1,j,k,taum1)-var(i,j,k,taum1))/dx
     &                *maskU(i,j,k)
        enddo
       enddo
      enddo
      call border_exchg3D(diff_fe,1); call setcyclic3D(diff_fe)
```

```
      do k=1,km-1
       do j=js,je
        do i=1,imt
         diff_fn(i,j,k)=diffx*(var(i,j+1,k,taum1)-var(i,j,k,taum1))/dx
     &               *maskV(i,j,k)
        enddo
       enddo
      enddo
      call border_exchg3D(diff_fn,1); call setcyclic3D(diff_fn)
      del2(:,js_pe:je_pe,:)=0.0
      do k=2,km-1
       do j=js,je
        do i=2,imt-1
      del2(i,j,k) = (DIFF_Tx(i,j,k) + DIFF_Ty(i,j,k))*maskT(i,j,k)
        enddo
       enddo
      enddo
      call border_exchg3D(del2,1); call setcyclic3D(del2)
      diff_fe(:,js_pe:je_pe,:)=0.0; diff_fn(:,js_pe:je_pe,:)=0.0
      do k=2,km-1
       do j=js,je
        do i=1,imt-1
         diff_fe(i,j,k) =diffx*(del2(i+1,j,k)-del2(i,j,k))/dx
     &                  *maskU(i,j,k)
        enddo
       enddo
      enddo
      call border_exchg3D(diff_fe,1); call setcyclic3D(diff_fe)
      do k=1,km-1
       do j=js,je
        do i=2,imt-1
         diff_fn(i,j,k) = diffx*(del2(i,j+1,k) - del2(i,j,k))/dx
     &                  *maskV(i,j,k)
        enddo
       enddo
      enddo
      call border_exchg3D(diff_fn,1); call setcyclic3D(diff_fn)
      do k=2,km-1
       do j=js,je
        do i=2,imt-1
          var(i,j,k,taup1)= var(i,j,k,taup1)+c2dt*maskT(i,j,k)*(
     &             -DIFF_Tx(i,j,k) - DIFF_Ty(i,j,k))
        enddo
       enddo
      enddo
      if (enable_vert_biha_diffusion) then
       diff_ft(:,js_pe:je_pe,:)=0.0
       do k=1,km-1
        do j=js,je
```

```
        do i=2,imt-1
         diff_ft(i,j,k)=diffz*(var(i,j,k+1,taum1)-var(i,j,k,taum1))/dz
    &                 *maskW(i,j,k)
        enddo
       enddo
      enddo
      del2(:,js_pe:je_pe,:)=0.0
      do k=2,km-1
       do j=js,je
        do i=2,imt-1
     del2(i,j,k) = DIFF_Tz(i,j,k)*maskT(i,j,k)
        enddo
       enddo
      enddo
      diff_ft(:,js_pe:je_pe,:)=0.0
      do k=2,km-1
       do j=js,je
        do i=2,imt-1
         diff_ft(i,j,k) =diffz*(del2(i,j,k+1)-del2(i,j,k))/dz
    &                    *maskW(i,j,k)
        enddo
       enddo
      enddo
      do k=2,km-1
       do j=js,je
        do i=2,imt-1
         var(i,j,k,taup1)=var(i,j,k,taup1)
    &             -c2dt*maskT(i,j,k)*DIFF_Tz(i,j,k)
        enddo
       enddo
      enddo
     endif
     end subroutine biha_mix
```