SWAMPE

Release 1.0.0

Ekaterina Landgren

Dec 05, 2022

CONTENTS

SWAMPE is a 2D shallow-water general circulation model designed for simulating exoplanet atmospheres. SWAMPE is a fully in Python implementation of a spectral algorithm described in Hack and Jakob (1992) and previously available only in Fortran.

SWAMPE's features currently include:

- Exoplanet atmosphere simulations that can be run on your laptop.
- Tuned for synchronously rotating hot Jupiters and sub-Neptunes.
- Can be easily adapted to model a variety of substellar objects.

This public release contains tutorials for running atmospheric simulations, working with SWAMPE output, and plotting.

CHAPTER

ONE

CONTRIBUTING

To report a bug or request a new feature, please open a new issue.

If you would like to get more actively involved in SWAMPE development, you are invited to get in touch ek672-at-cornell-dot-edu, and I will be happy to help you understand the codebase as you work on your contribution.

1.1 Installation

1.1.1 Install SWAMPE with Pip

```
pip install SWAMPE
```

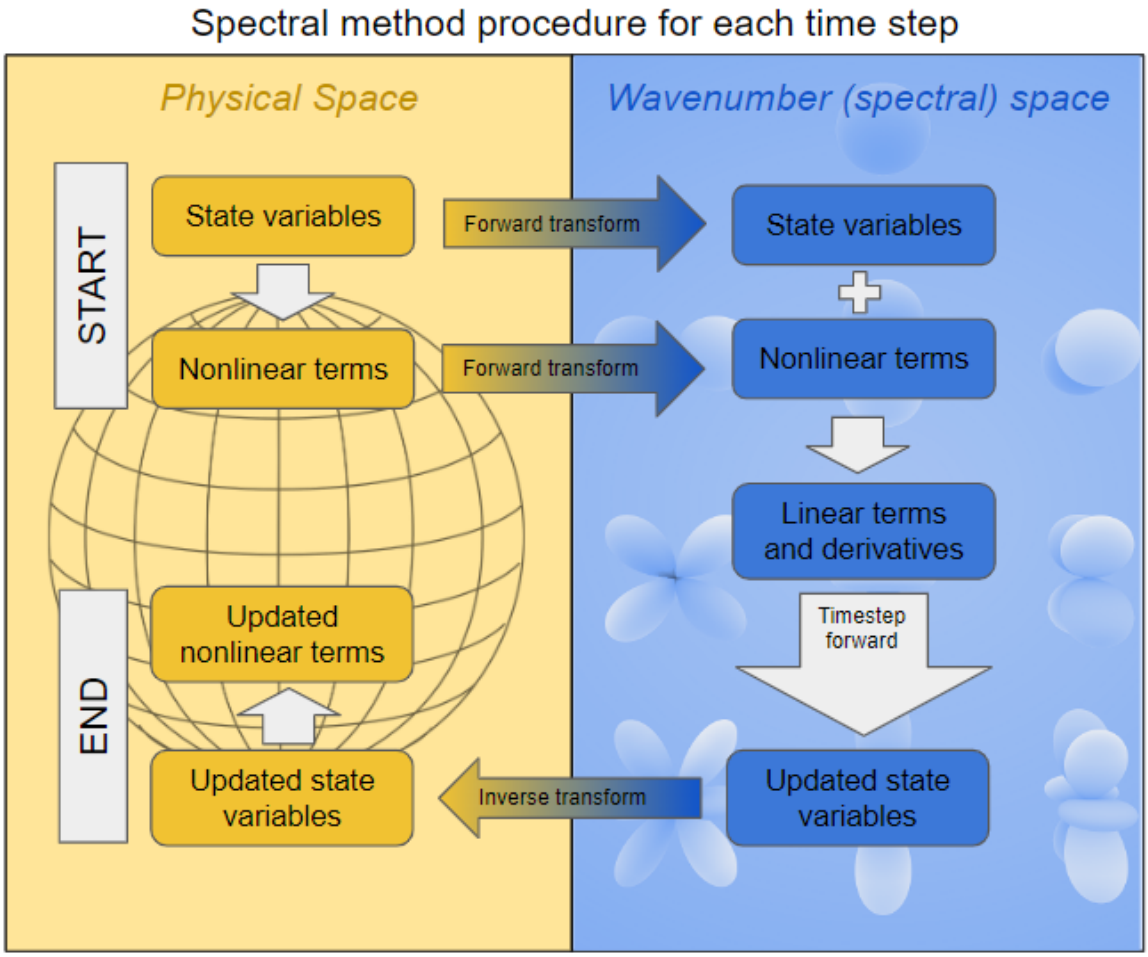
1.1.2 Install SWAMPE with Git

```
git clone https://github.com/kathlandgren/SWAMPE.git
cd SWAMPE
python setup.py install
```

1.2 Methods

SWAMPE is based on a spectral method for solving shallow-water equations on a sphere described in Hack and Jakob (1992). *SWAMPE* is designed to replace the previous Fortran implementations of this method.

> "The basic idea behind the spectral transform method is to locally evaluate all nonlinear terms (including diabatic physical processes) in physical space on an associated finite-difference-like grid, most often referred to as the transform grid. These terms are then transformed back into wavenumber space to calculate linear terms and derivatives, and to obtain tendencies for the time-dependent state variables."



1.2.1 Governing equations

The shallow-water equations on the sphere are given by:

$\frac{d\mathbf{V}}{dt} = -f\mathbf{k} \times \mathbf{V} - \nabla\Phi$

$\frac{d\Phi}{dt} = -\Phi\nabla \cdot \mathbf{V}$

where $\mathbf{V} = u\mathbf{i} + v\mathbf{j}$ is the velocity vector on the surface of the sphere and $\mathbf{i}$ and $\mathbf{j}$ are the unit eastward and northward directions, respectively. The free surface geopotential is given by $\Phi \equiv gh$, where g is the gravitational acceleration.

The atmosphere is assumed to be a fluid that is incompressible and hydrostatically balanced. For a derivation of the shallow water equations from the continuity equation and the equation of motion, see, e.g. Kaper and Engel (2013).

For the spherical harmonic transform method that we use, it is convenient to multiply the velocity $\mathbf{V}$ by the cosine of latitude so that the velocity arguments are smooth at the poles. We will also rewrite the governing equations in terms of geopotential, the vertical component of relative vorticity $\zeta = \mathbf{k} \cdot (\nabla \times \mathbf{V})$, and horizontal divergence $\delta = \nabla \cdot \mathbf{V}$. Taking the curl ($\mathbf{k} \cdot \nabla \times [\,]$) and divergence ($\nabla \cdot [\,]$) of the momentum equation yields the following:

$\frac{\partial\zeta}{\partial t} = -\nabla \cdot (\zeta + f)\mathbf{V}$

$\frac{\partial\delta}{\partial t} = \mathbf{k} \cdot \nabla \times ((\zeta + f)\mathbf{V}) - \nabla^2\left(\Phi + \frac{\mathbf{V}\cdot\mathbf{V}}{2}\right).$

Writing the continuity equation so that the partial time derivative is the only term on the left hand side yields $\frac{\partial\Phi}{\partial t} = -(\mathbf{V} \cdot \nabla)\Phi - \Phi\nabla \cdot \mathbf{V} = -\nabla \cdot (\Phi\mathbf{V})$.

It is now convenient to use absolute vorticity $\eta = \zeta + f$, as opposed to relative vorticity $\eta = \zeta + f$, as one of the state variables in addition to divergence δ and geopotential Φ. The zonal winds U and meridional winds V, on the other hand, are diagnostic variables: they will not be time-stepped directly. The winds enter the time-stepping scheme via nonlinear components.

We follow Hack and Jakob (1992) in our notation for the nonlinear terms, namely:

$A = U\eta,$

$B = V\eta,$

$C = U\Phi,$

$D = V\Phi,$

$E = \frac{U^2+V^2}{2(1-\mu^2)}.$

1.2.2 Time stepping

While Hack and Jakob (1992) describe a semi-implicit time-stepping scheme, we follow Langton (2008) and implement a modified Euler's method scheme. In practice, this method proves more stable, especially for strongly irradiated planets.

The modified Euler's method for differential equations has the form:

$y_{n+1} = y_n + (\Delta t/2)[f(t_n, y_n) + f(t_{n+1}, y_n + \Delta t f(t_n, y_n)].$

Equivalently, we can write:

$K_1 = \Delta t f(t_n, y_n),$

$K_2 = \Delta t f(t_{n+1}, y_n + K_1),$

$y_{n+1} = y_n + \frac{K_1+K_2}{2}.$

We will now derive these coefficients for the state variables in our timestepping scheme.

Firstly, we can split the unforced shallow-water system into the linear and nonlinear components by rewriting it as follows:

$$\frac{d}{dt}\begin{bmatrix}\eta_n^m\\ \delta_n^m\\ \Phi_n^m\end{bmatrix} = \begin{bmatrix}0 & 0 & 0\\ 0 & 0 & \frac{n(n+1)}{a^2}\\ 0 & -\overline{\Phi} & 0\end{bmatrix}\begin{bmatrix}\eta_n^m\\ \delta_n^m\\ \Phi_n^m\end{bmatrix} + \begin{bmatrix}E(t)\\ D(t)\\ P(t)\end{bmatrix}$$

where $\begin{bmatrix}E(t)\\ D(t)\\ P(t)\end{bmatrix}$ represents the nonlinear time-dependent components. We will evaluate the first component of the right hand side implicitly, while evaluating the second component explicitly.

The *unforced* nonlinear components can be expressed as follows:

$E(t) = -\frac{1}{a(1-\mu^2)}\frac{\partial A}{\partial \lambda} - \frac{1}{a}\frac{\partial B}{\partial \mu}$

$D(t) = \frac{1}{a(1-\mu^2)}\frac{\partial B}{\partial \lambda} - \frac{1}{a}\frac{\partial A}{\partial \mu} - \nabla^2 E$

$P(t) = -\frac{1}{a(1-\mu^2)}\frac{\partial C}{\partial \lambda} - \frac{1}{a}\frac{\partial D}{\partial \mu}.$

Let F_Φ be the geopotential forcing (for *SWAMPE*, due to stellar irradiation, but more general in theory). Let $F_U = F_u \cos\phi$ and $F_V = F_v \cos\phi$ be momentum forcing. Then the *forced* nonlinear components are as follows:

$E(t) = -\frac{1}{a(1-\mu^2)}\frac{\partial}{\partial \lambda}(A - F_V) - \frac{1}{a}\frac{\partial}{\partial \mu}(B + F_U),$

$D(t) = \frac{1}{a(1-\mu^2)}\frac{\partial}{\partial \lambda}(B + F_U) - \frac{1}{a}\frac{\partial}{\partial \mu}(A - F_V) - \nabla^2 E,$

$P(t) = -\frac{1}{a(1-\mu^2)}\frac{\partial C}{\partial \lambda} - \frac{1}{a}\frac{\partial D}{\partial \mu} + F_\Phi$.

Following the notation of the modified Euler's method, we write $K^1 = \Delta t f(t, y_t)$:

$K^1_\eta = \Delta t(E(t))$,

$K^1_\delta = \Delta t\left(\frac{n(n+1)}{a^2}\Phi_n^{m(t)} + D(t)\right)$,

$K^1_\Phi = \Delta t\left(-\overline{\Phi}\delta_n^{m(t)} + P(t)\right)$.

Then we can write the $K^2 = \Delta t(f(t+1, y_t + K^1))$ coefficients.

$K^2_\eta = \Delta t(E(t+1))$,

$K^2_\delta = \Delta t\left(D(t+1) + \frac{n(n+1)}{a^2}(\Phi_n^m + K^1_\Phi)\right)$,

$K^2_\Phi = \Delta t\left(P(t+1) - \overline{\Phi}(\delta_n^m + K^1_\delta)\right)$.

Expanding the equations for K^2_δ and K^2_Φ, we obtain:

$K^2_\delta = \Delta t\left(D(t+1) + \frac{n(n+1)}{a^2}(P(t)) + \frac{n(n+1)}{a^2}\Phi_n^m - \overline{\Phi}\frac{n(n+1)}{a^2}\delta_n^m\right)$,

$K^2_\Phi = \Delta t\left(P(t+1) - \overline{\Phi}(D(t)) - \overline{\Phi}\delta_n^m - \overline{\Phi}\frac{n(n+1)}{a^2}\Phi_n^m\right)$.

We evaluate the time-dependent terms explicitly, assuming $\begin{bmatrix} E(t) \\ D(t) \\ P(t) \end{bmatrix} = \begin{bmatrix} E(t+1) \\ D(t+1) \\ P(t+1) \end{bmatrix}$ to first order. This is what is done in the semi-implicit method in Hack and Jakob (1992). An alternative variant would be to approximate η, δ, Φ, U, and V by a different method, such as forward Euler's method or a semi-implicit one. This would result in a higher computational cost and hopefully higher accuracy as well, while maintaining the stability properties of modified Euler's method.

Note that in the current implementation, η time-stepping is equivalent to forward Euler's method, since η does not depend linearly on other state variables, only nonlinearly in the $E(t)$ term. Writing $(K^1 + K^2)/2$ in order to evaluate the modified Euler scheme, we can simplify:

$\frac{K^1_\delta + K^2_\delta}{2} = \Delta t\left(\frac{n(n+1)}{a^2}\Phi_n^m + D(t) + \frac{1}{2}\left(\frac{n(n+1)}{a^2}(P(t) - \overline{\Phi}\delta_n^m\right)\right)$,

and

$\frac{K^1_\Phi + K^2_\Phi}{2} = \Delta t\left(-\overline{\Phi}\delta_n^m + P(t)\right) - \frac{\Delta t}{2}\overline{\Phi}\left(D(t) + \frac{n(n+1)}{a^2}\right)$.

1.2.3 Filters

To ensure numerical stability, SWAMPE applies the following filters:

- a modal-splitting filter as described in Hack and Jakob (1992).
- a sixth-degree hyperviscosity filter. We use the formulation based on Gelb and Gleeson (2001).

Note: *SWAMPE*'s default hyperviscosity coefficient has been tested for hot Jupiter and sub-Neptune simulations but might require further tuning for drastically different stellar forcings. The modal-splitting coefficient typically does not need to be adjusted from its default value.

1.2.4 Testing

To ensure the correct operation of the spectral transforms, a series of unit tests are performed via continuous integration with Github Actions.

SWAMPE has been benchmarked against end-to-end tests 1 and 2 from a standard test set for numerical shallow-water solvers (see Williamson and Drake (1992)). as well as strongly irradiated hot Jupiters described by Perez-Becker and Showman (2013).

1.3 SWAMPE Quick Start

In this tutorial, we will explore how to use SWAMP-E to simulate planetary atmospheres.

All tutorials are available for download here.

[1]:
```
import numpy
import SWAMPE
```

1.3.1 Running SWAMPE

Planetary Parameters

First, let's specify some key physical inputs:

[2]:
```
# planetary radius a in meters

a=8.2*10**(7)

# planetary rotation rate omega in radians/s

omega=3.2*10**(-5)

# the reference geopotential height, m^2/s^2, typically based on scale height

Phibar=4*(10**6)
```

Simulation Parameters

Now let's specify the timestep and the number of timesteps. For stability, a reasonable timestep will range from 30 to 180 seconds, with strongly forced planets needing a shorter timestep.

[3]:
```
#timestep length in seconds
dt=30
#number of timesteps for the simulation
tmax=10
```

We will also specify the spectral resolution M. This resolution determines the size of the resulting lat-lon grid.

[4]:
```
M=42
```

Forcing parameters

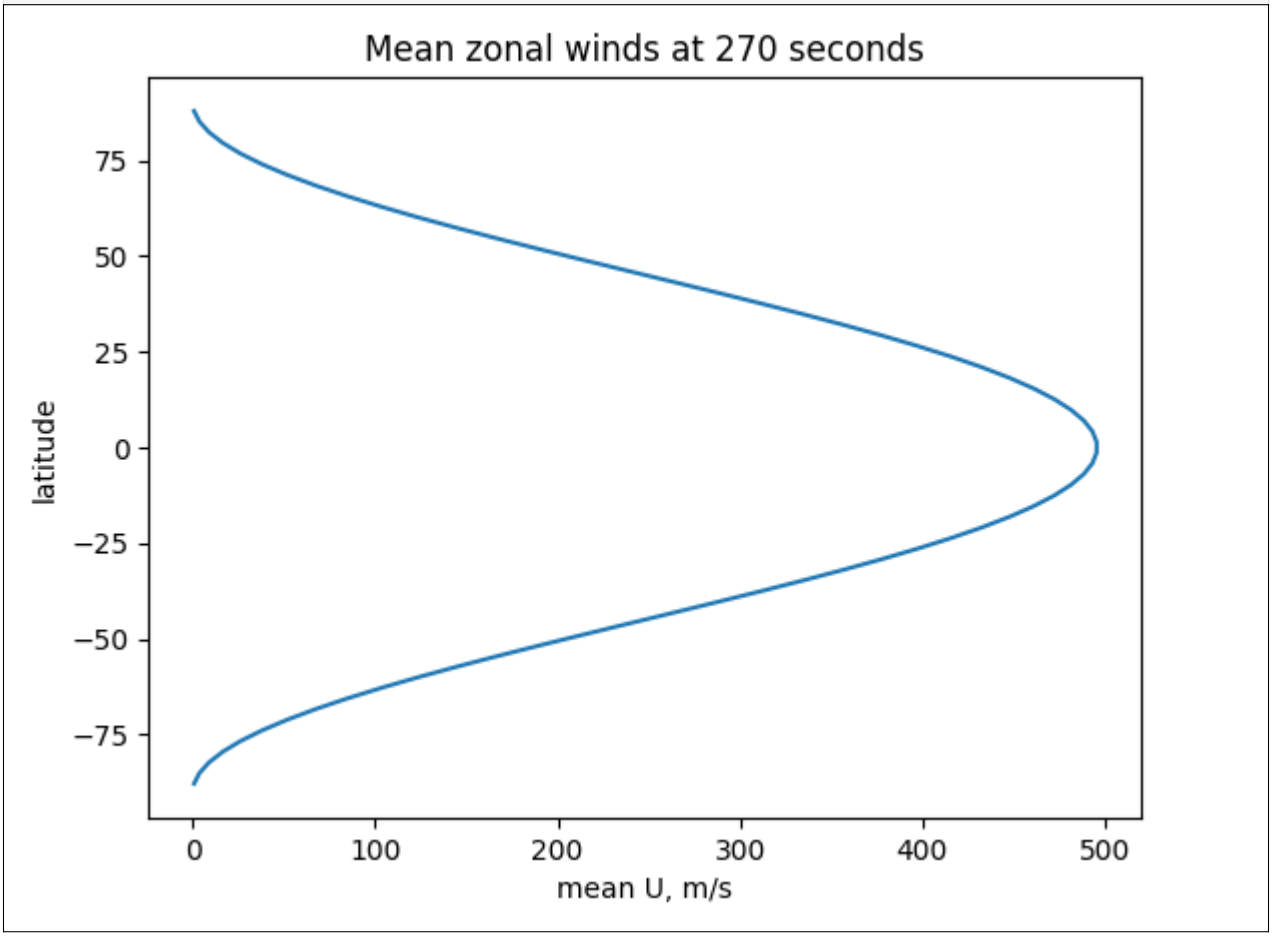
Let's specify forcing parameters. The forcing strength is controlled via DPhieq, which is the day/night amplitude of the local radiative equilibrium geopotential. The default forcing scheme in SWAMPE is based on Perez-Becker and Showman (2013). **The current default forcing scheme supports synchronous rotators with time-constant stellar irradiation.** If you'd like to simulate other forcing patterns, download SWAMPE from the repository and change `SWAMPE.forcing.Phieqfun` (prescribing local radiative equilibroum) and/or `SWAMPE.forcing.Qfun` (the Newtonian relaxation to local radiative equilibrium).

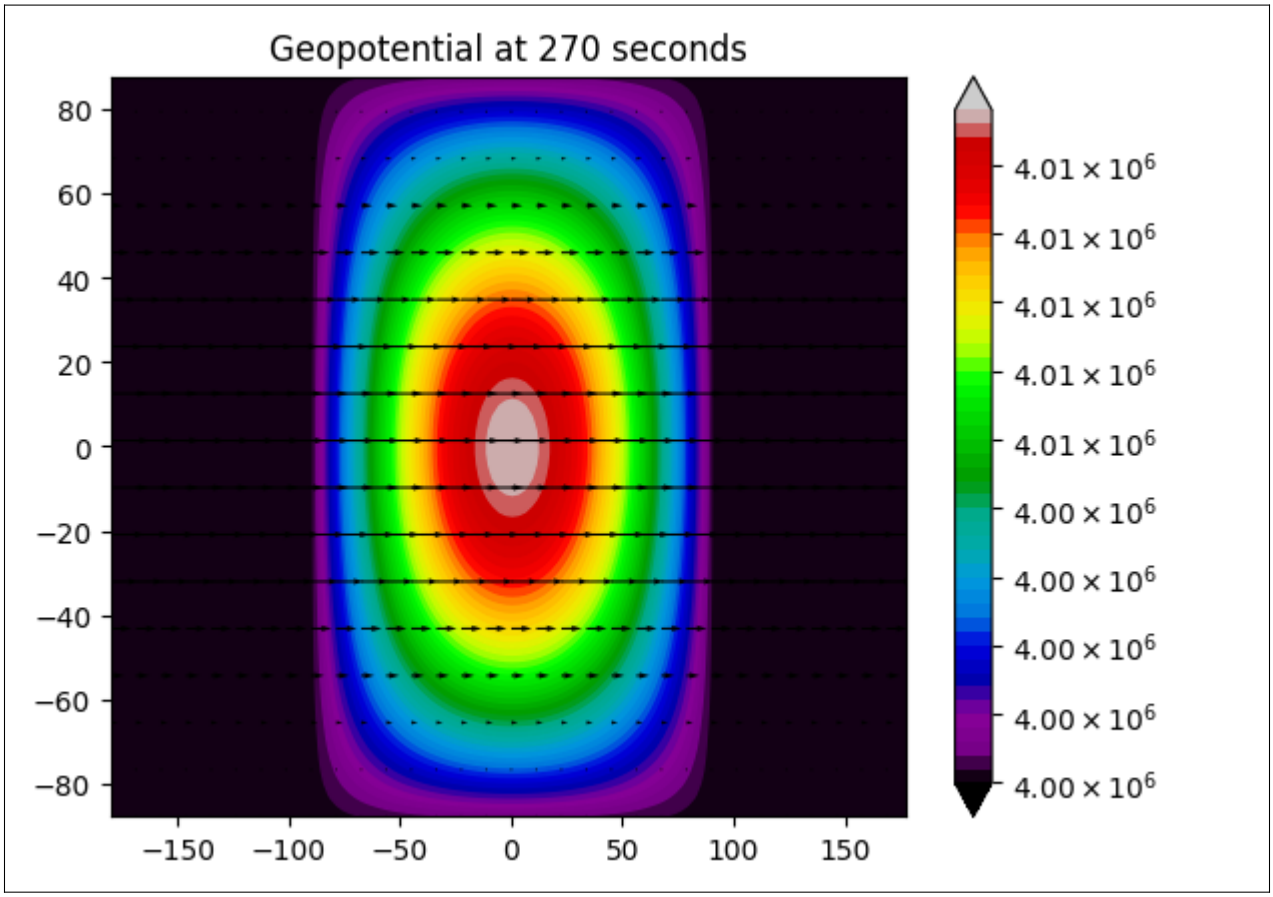
We also specify two timescales: a radiative timescale of one day and a drag timescale of ten days.

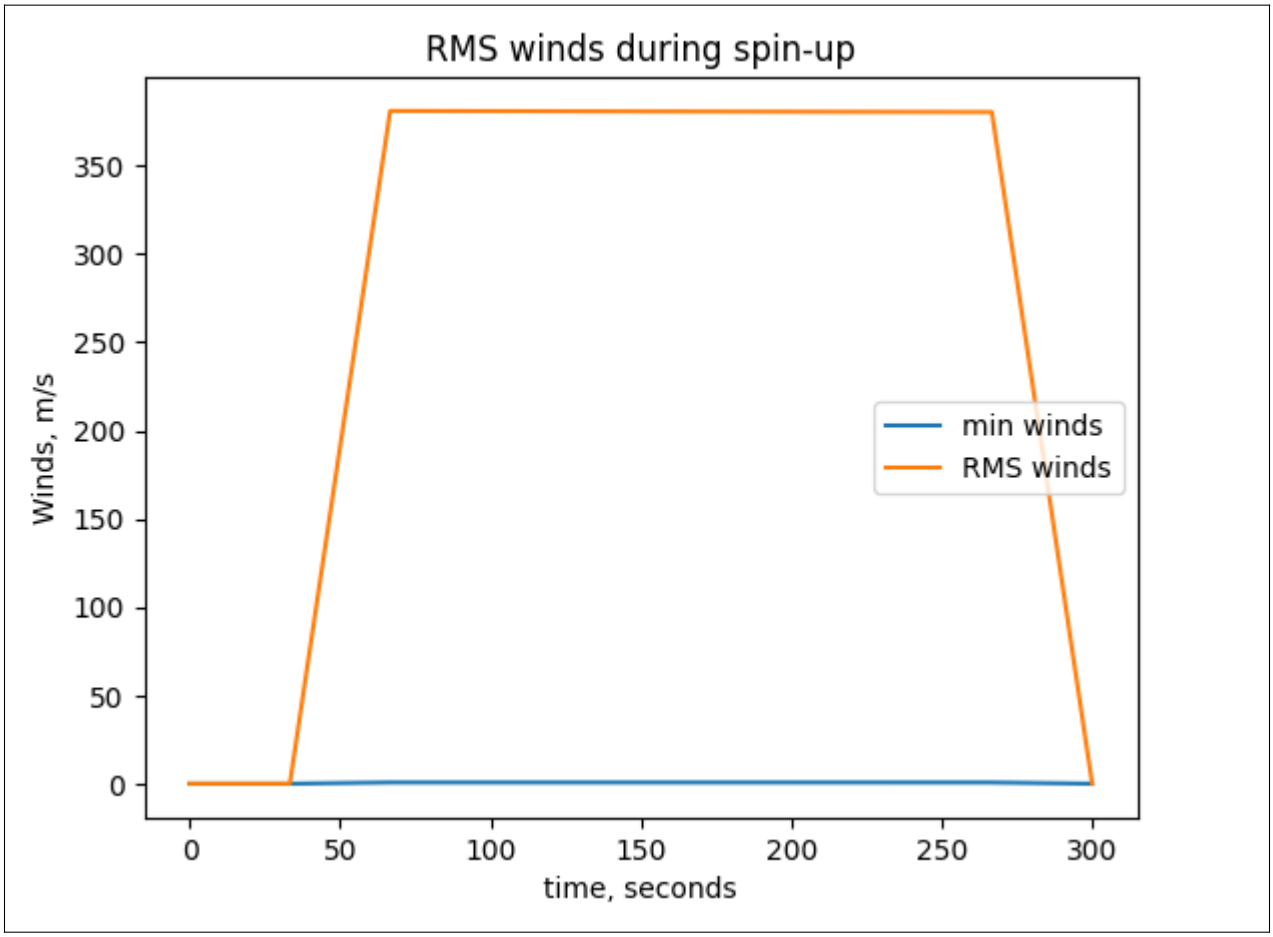
[5]:
```
DPhieq=Phibar

taurad=3600*24
taudrag=10*3600*24
```

Now, we can use SWAMP-E to run the simulation. We can specify whether we want to see plots as the simulation is conducted and control the frequency of output using plotflag and plotfreq.

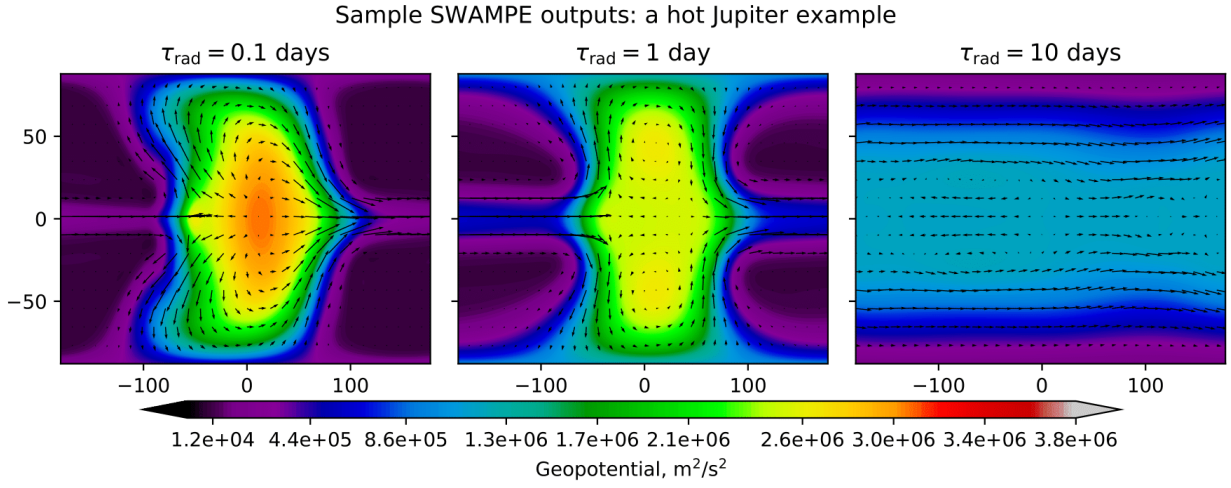
[6]:
```
plotflag=True
plotfreq=9 #output plots every 9 time steps

SWAMPE.run_model(M,dt,tmax,Phibar, omega, a,  taurad=taurad, taudrag=taudrag,␣
↪DPhieq=DPhieq, plotflag=plotflag, plotfreq=plotfreq,saveflag=False,verbose=False,
↪timeunits='seconds')
```




Geopotential at 270 seconds
80
60
40
20
0
−20
−40
−60
−80
−150
−100
−50
0
50
100
150
4.01 × 10^6
4.01 × 10^6
4.01 × 10^6
4.01 × 10^6
4.01 × 10^6
4.00 × 10^6
4.00 × 10^6
4.00 × 10^6
4.00 × 10^6
4.00 × 10^6



Note that a 2D model like SWAMP-E will need a few hundred time steps to spin up, so with a real-world example, we recommend running the model using a script rather than a Jupyter notebook. Here are sample outputs for the same planetary parameters, but without drag (equivalent to $\tau_{\mathrm{drag}} = \infty$) and with τ_{rad} equal to 0.1, 1, and 10 Earth days. The scripts and the reference data are provided here.



1.3.2 Saving and loading SWAMPE data

Saving the data

Since the simulations can take a few hundred timeteps to spin up, it is helpful to save the data to be loaded later. We can specify whether to save the data using saveflag. We can also specify savefrequency. The data will be stored in the pickle format with the name 'variable-timestamp', where timestamp can be specified in seconds, minutes, or hours.

The saved data files will look like this:

```
Phi-10
eta-10
delta-10
U-10
V-10
```

This will save the data every 120 time steps in the folder C:/your/Path/here/

```
SWAMPE.run_model(..., saveflag=True, savefreq=120, custompath='C:/your/Path/here/',
↪timeunits='minutes')
```

For a timestep of 30 seconds, the first output will look like this, with the timestamp of sixty minutes:

```
Phi-60
eta-60
delta-60
U-60
V-60
```

If no custom path is provided, SWAMPE will create a local 'data/' directory and store the data there.

Continuation

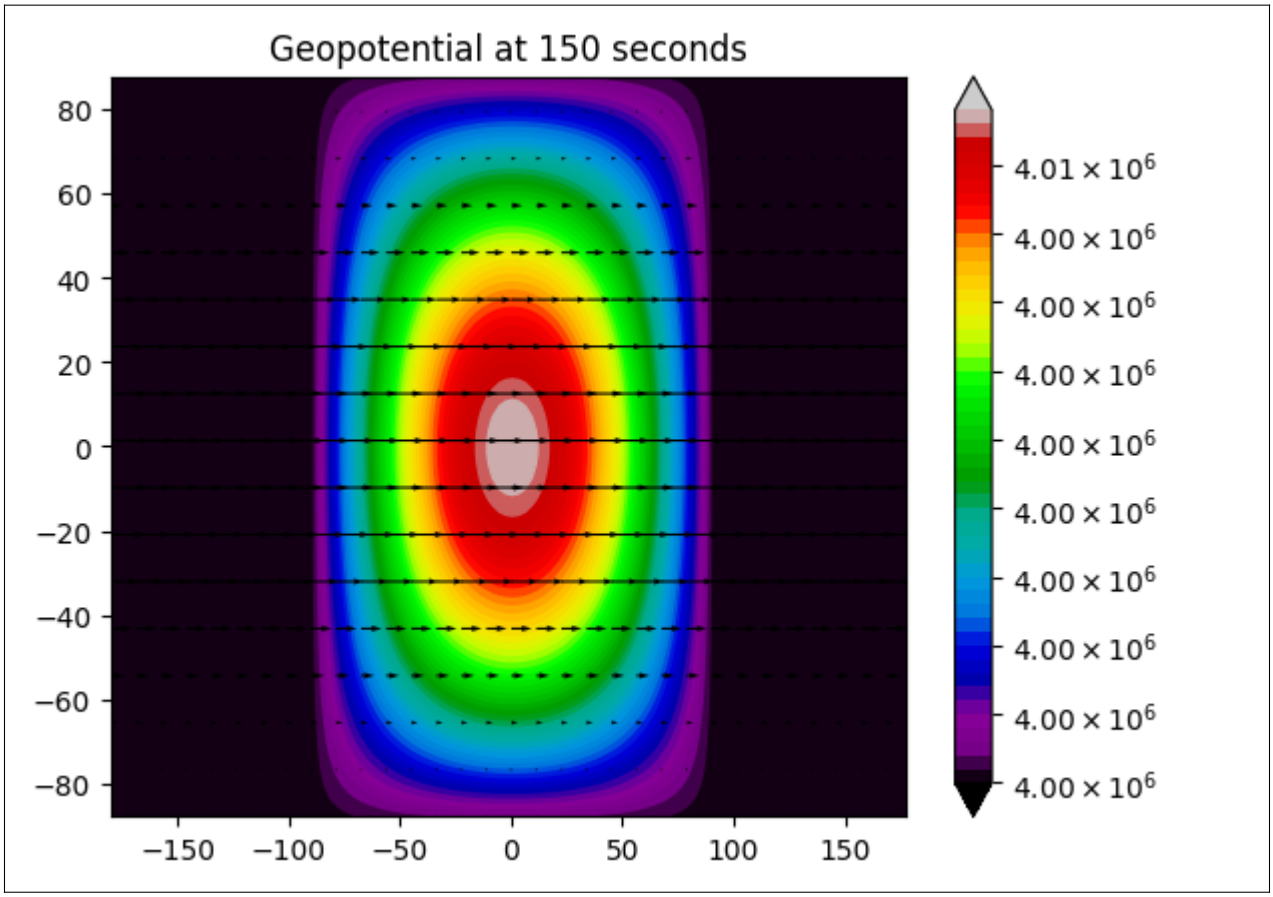
We can also start the simulation from any suitable initial condition. Let's save some data and restart the simulation from the saved data.

[14]:
```
#saving the data every 5 timesteps
SWAMPE.run_model(M,dt,tmax,Phibar, omega, a,  taurad=taurad, taudrag=-1, DPhieq=Phibar,␣
↪plotflag=False, plotfreq=plotfreq,saveflag=True,savefreq=5,verbose=False,timeunits=
↪'seconds')
#note taudrag=-1 corresponds to taudrag=infinity, or no drag
```

Now we can load the saved data. Let's say we want to load the data from 150 seconds.

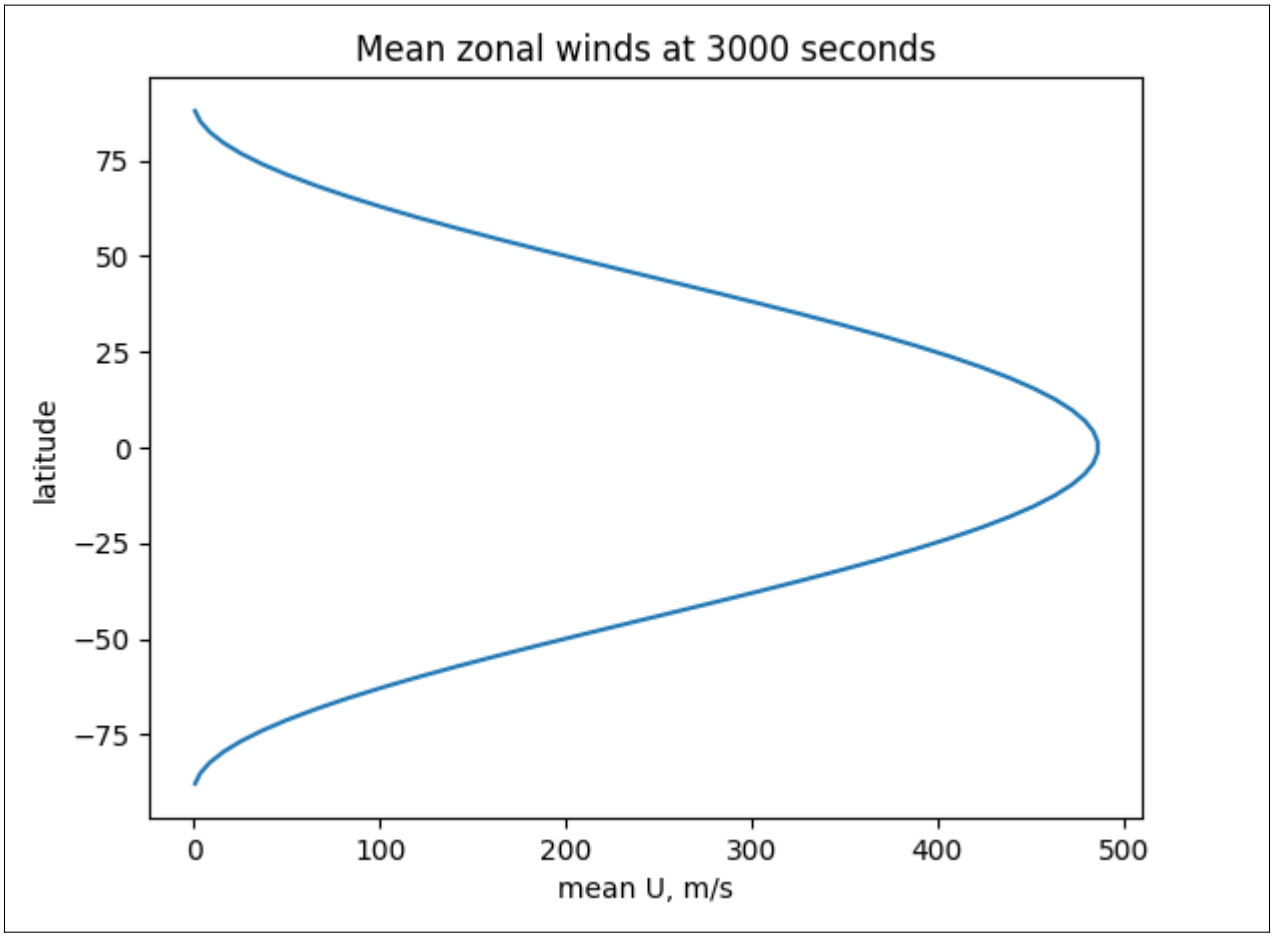
[8]:
```
timestamp='150'
eta, delta, Phi, U, V =SWAMPE.continuation.load_data(str(timestamp))
```

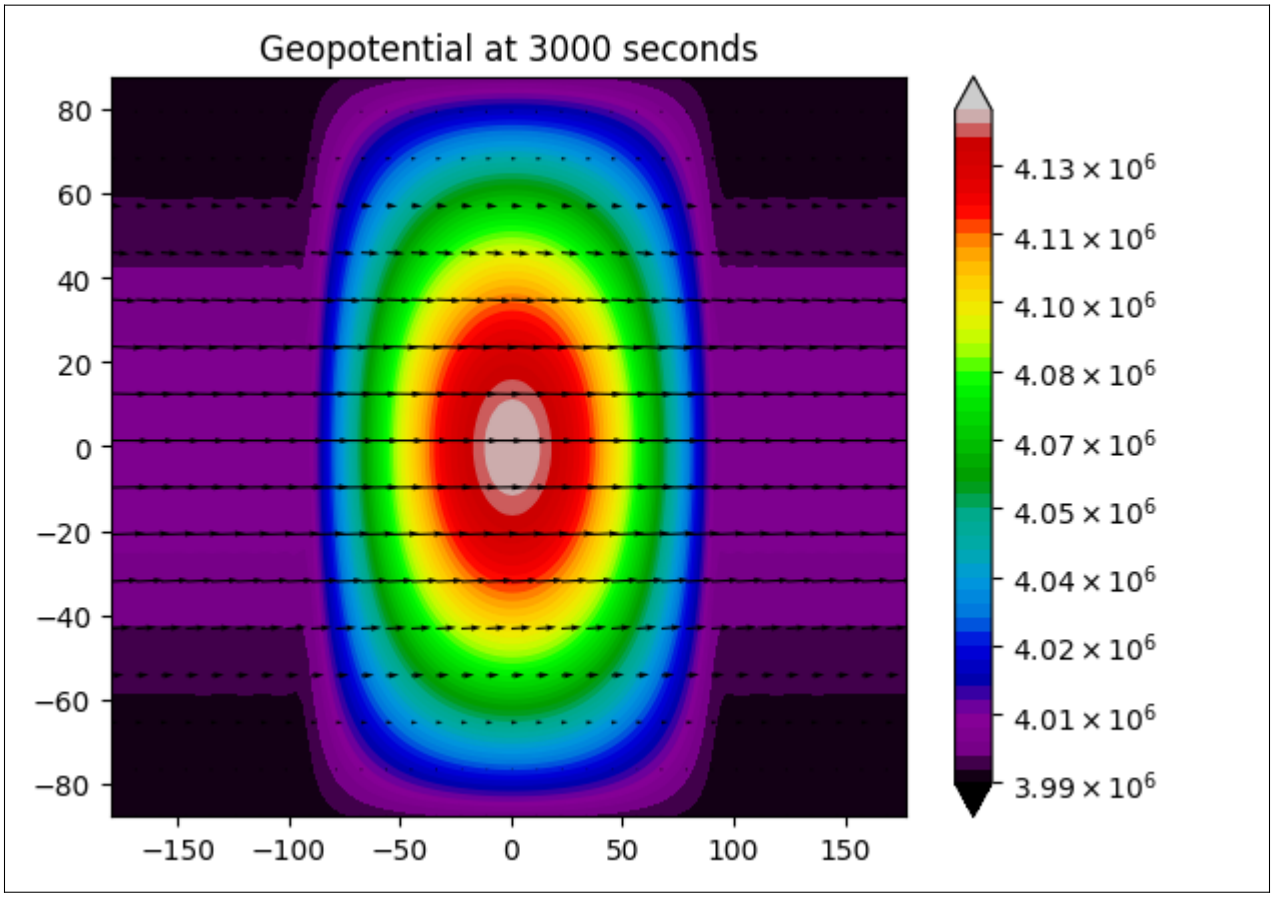
[9]:
```
N,I,J,otherdt,lambdas,mus,w=SWAMPE.initial_conditions.spectral_params(M)
fig=SWAMPE.plotting.quiver_geopot_plot(U,V,Phi+4*10**6,lambdas,mus,timestamp,
↪sparseness=4,minlevel=None,maxlevel=None,units='seconds')
```

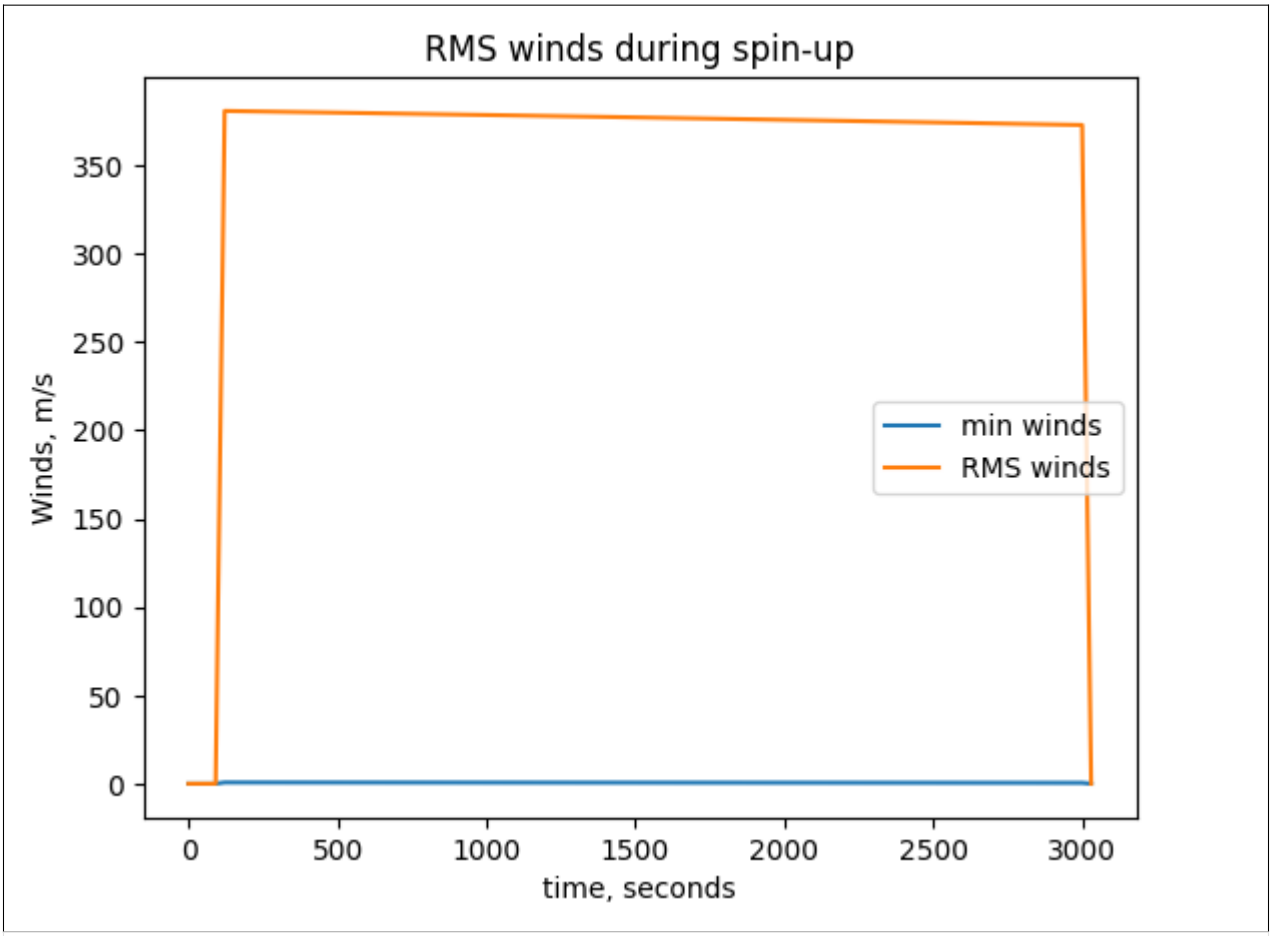


[10]:
```
SWAMPE.run_model(M,30,101,Phibar, omega, a,  taurad=taurad, taudrag=-1, DPhieq=Phibar,␣
↪plotflag=True, plotfreq=100,saveflag=False,verbose=True,timeunits='seconds',
↪contflag=True,contTime=150)
```

```
t=10, 9.900990099009901% complete
t=20, 19.801980198019802% complete
t=30, 29.702970297029704% complete
t=40, 39.603960396039604% complete
t=50, 49.504950495049506% complete
t=60, 59.40594059405941% complete
t=70, 69.3069306930693% complete
t=80, 79.20792079207921% complete
t=90, 89.10891089108911% complete
t=100, 99.00990099009901% complete
```


Mean zonal winds at 3000 seconds
latitude
75
50
25
0
−25
−50
−75
0
100
200
300
400
500
mean U, m/s


Geopotential at 3000 seconds
80
60
40
20
0
−20
−40
−60
−80
−150
−100
−50
0
50
100
150
4.13 × 10^6
4.11 × 10^6
4.10 × 10^6
4.08 × 10^6
4.07 × 10^6
4.05 × 10^6
4.04 × 10^6
4.02 × 10^6
4.01 × 10^6
3.99 × 10^6




```
GCM run completed!
```

1.4 Plots

In this tutorial, we will explore how to plot the simulations generated by SWAMP-E.

All tutorials are available for download here.

[1]:
```
import os
import imageio
import numpy as np
from matplotlib import cm
import matplotlib.pyplot as plt
import SWAMPE
```

Let's load the reference data. **To make the path to the reference data work, it is easiest to download the notebooks by cloning the repository.**

[2]:
```
# load  reference data
data_dir1 = os.path.abspath('../../../SWAMPE/reference_data/HJ_taurad_100/')+'\\'
timestamp1=710
eta1, delta1, Phi1, U1, V1 =SWAMPE.continuation.load_data(timestamp1,custompath=data_
```

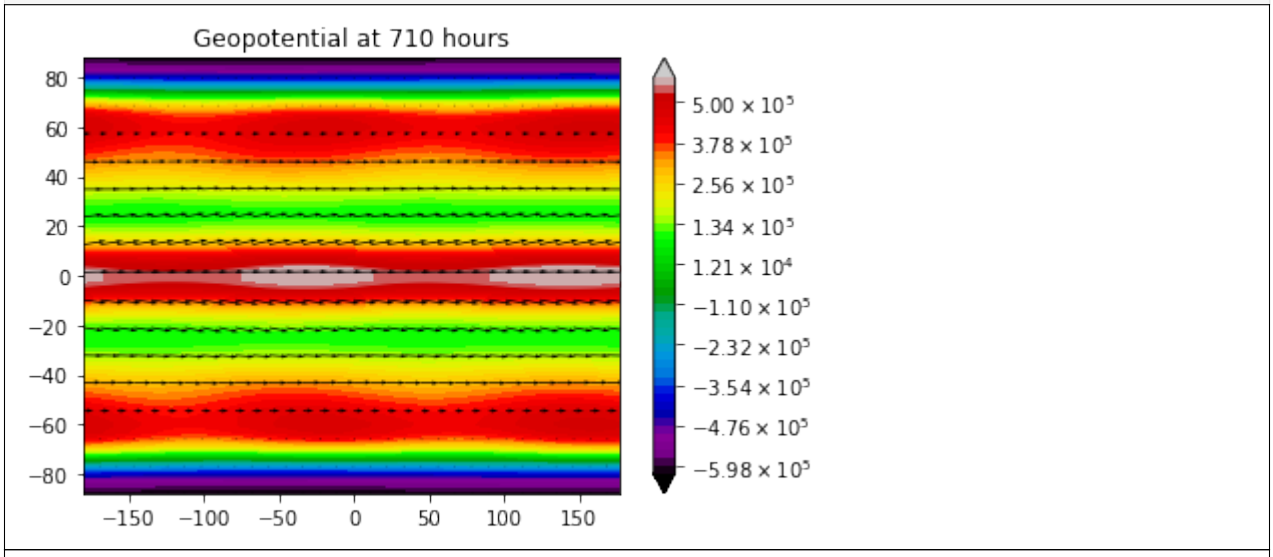
(continues on next page)

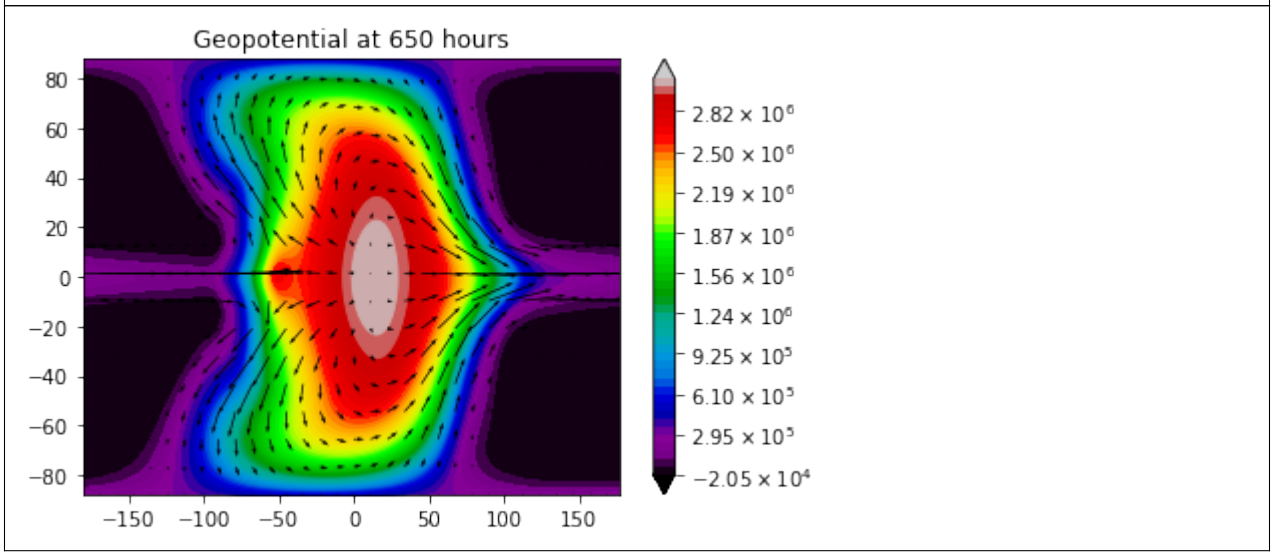
(continued from previous page)

```
↪dir1)
data_dir2 = os.path.abspath('../../../SWAMPE/reference_data/HJ_taurad_0p1/')+'\\'
timestamp2=650
eta2, delta2, Phi2, U2, V2 =SWAMPE.continuation.load_data(timestamp2,custompath=data_
↪dir2)

#generate the latitudes and longitude of matching resolution
M=42
N,I,J,dt,lambdas,mus,w=SWAMPE.initial_conditions.spectral_params(M)
```

1.4.1 Plotting the geopotential plot with a wind vector field

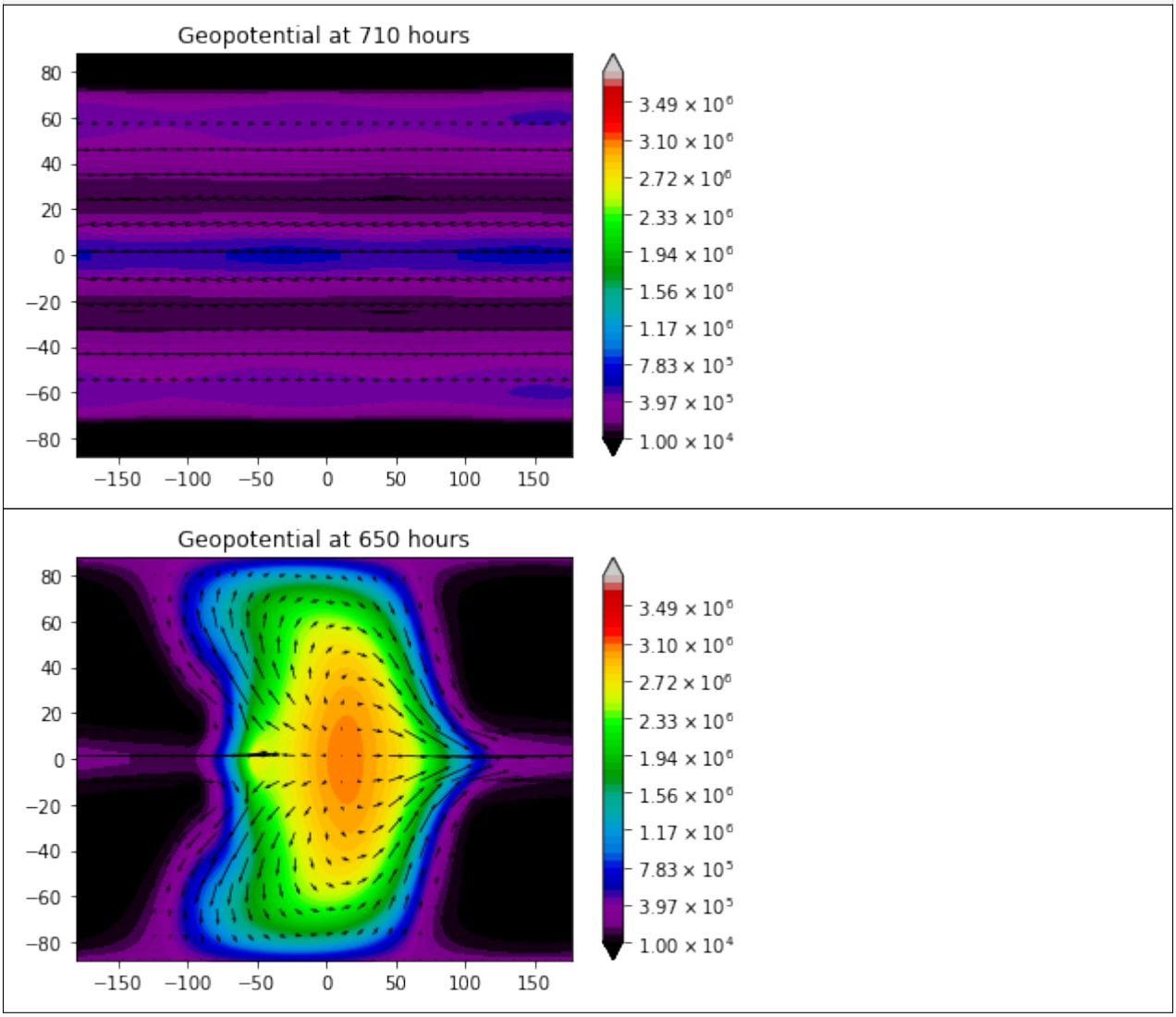
[3]:
```
# plot the geopotential field with the overlayed wind field

fig1=SWAMPE.plotting.quiver_geopot_plot(U1,V1,Phi1,lambdas,mus,timestamp1)
fig2=SWAMPE.plotting.quiver_geopot_plot(U2,V2,Phi2,lambdas,mus,timestamp2)
```



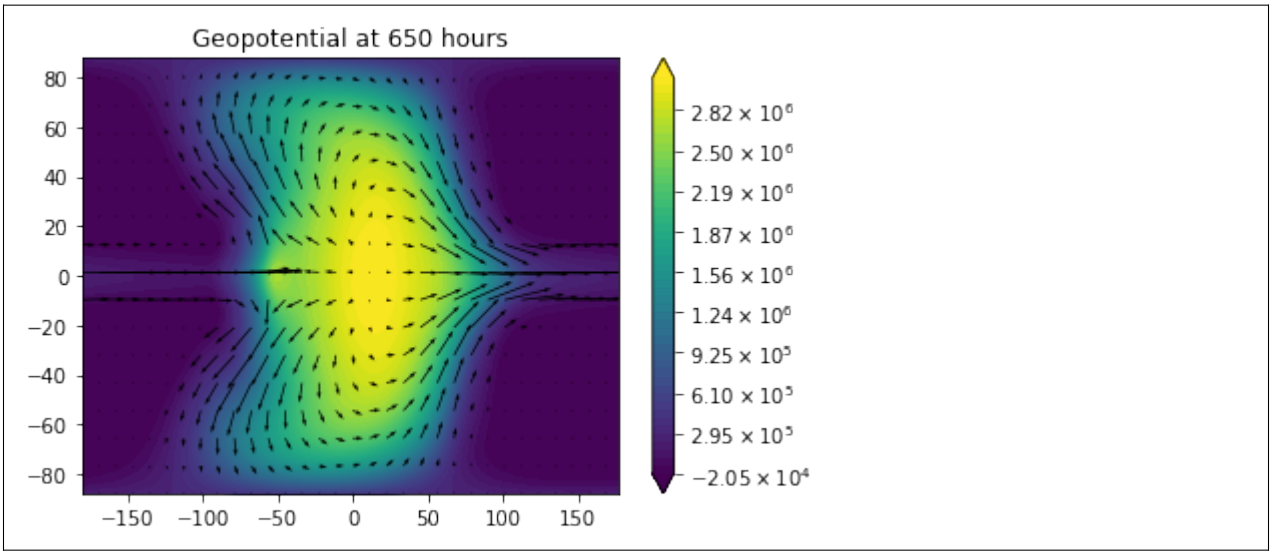


We can change the limits of the colorbar. This can be useful for comparing the outputs of multiple simulations.

[4]:
```
#change min/maxlevel
Phibar=4*10**6
fig1=SWAMPE.plotting.quiver_geopot_plot(U1,V1,Phi1,lambdas,mus,timestamp1,minlevel=10**4,
↪maxlevel=3.8*10**6)
fig2=SWAMPE.plotting.quiver_geopot_plot(U2,V2,Phi2,lambdas,mus,timestamp2,minlevel=10**4,
↪maxlevel=3.8*10**6)
```

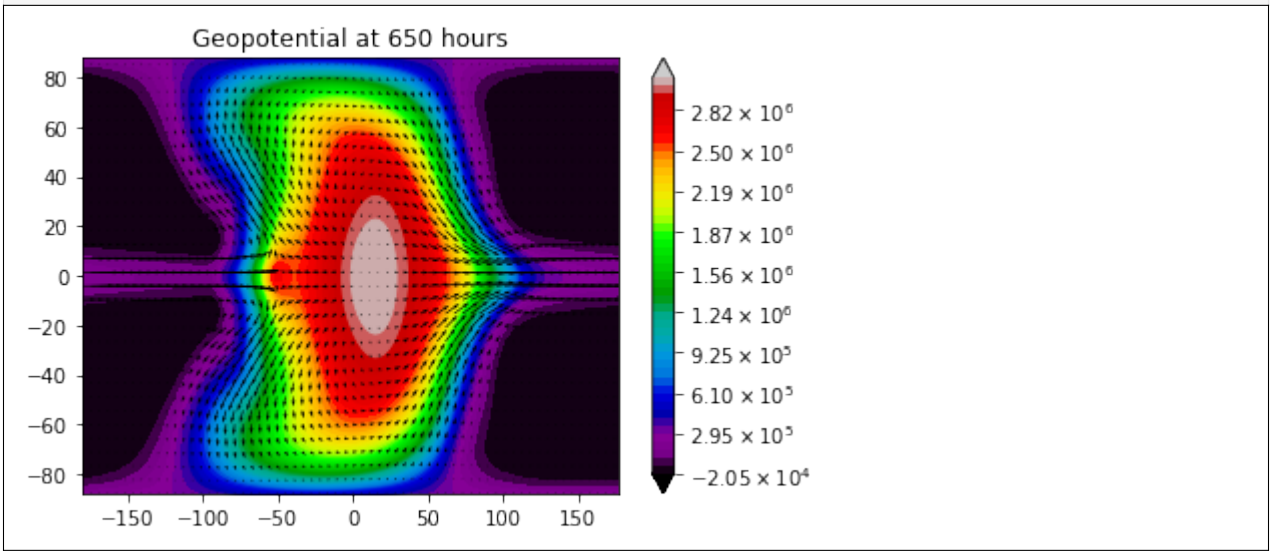


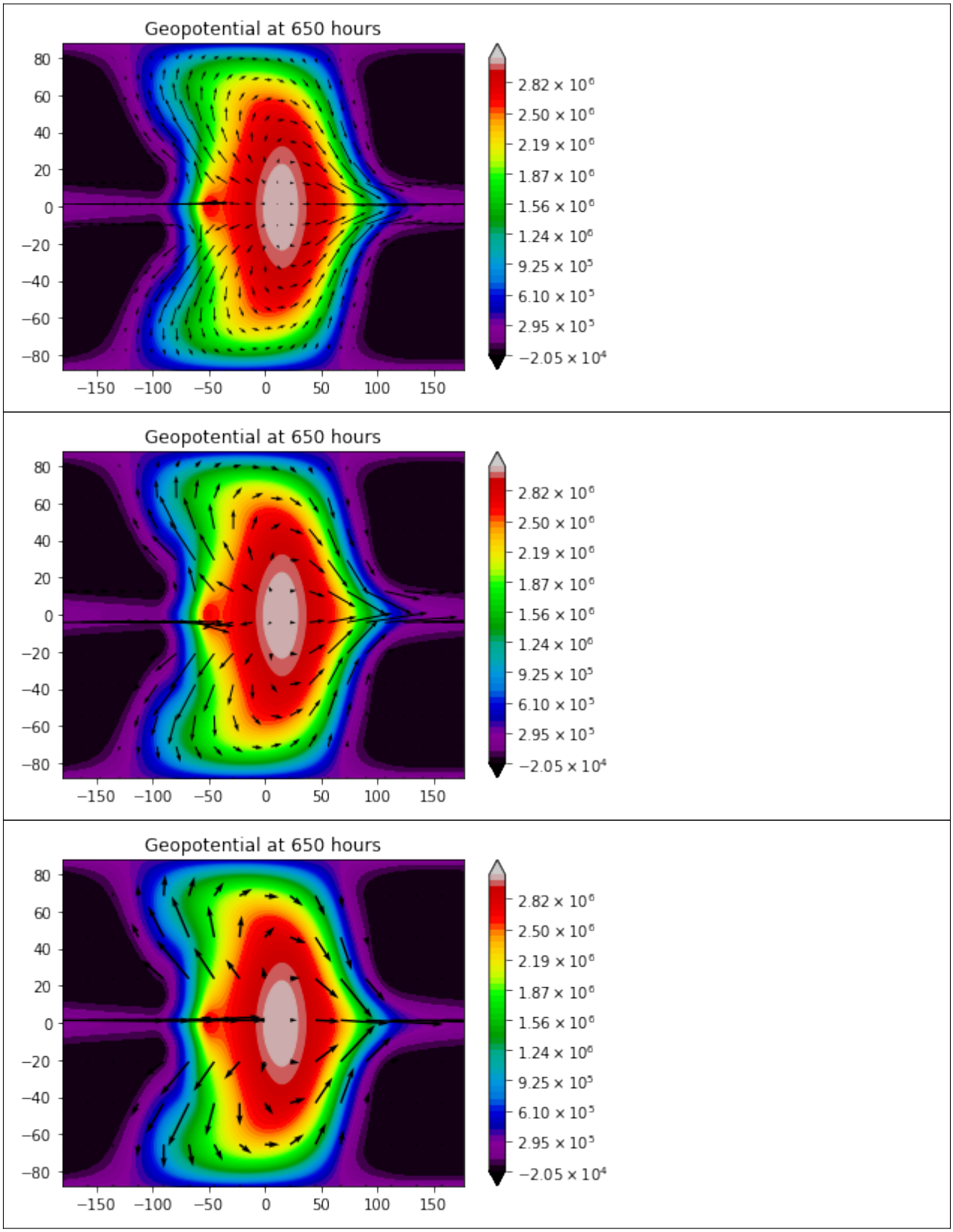
[5]:
```
#change colormap
colormap=cm.viridis
fig=SWAMPE.plotting.quiver_geopot_plot(U2,V2,Phi2,lambdas,mus,timestamp2,
↪colormap=colormap)
```



We can also change the density of the wind vector field using sparseness.

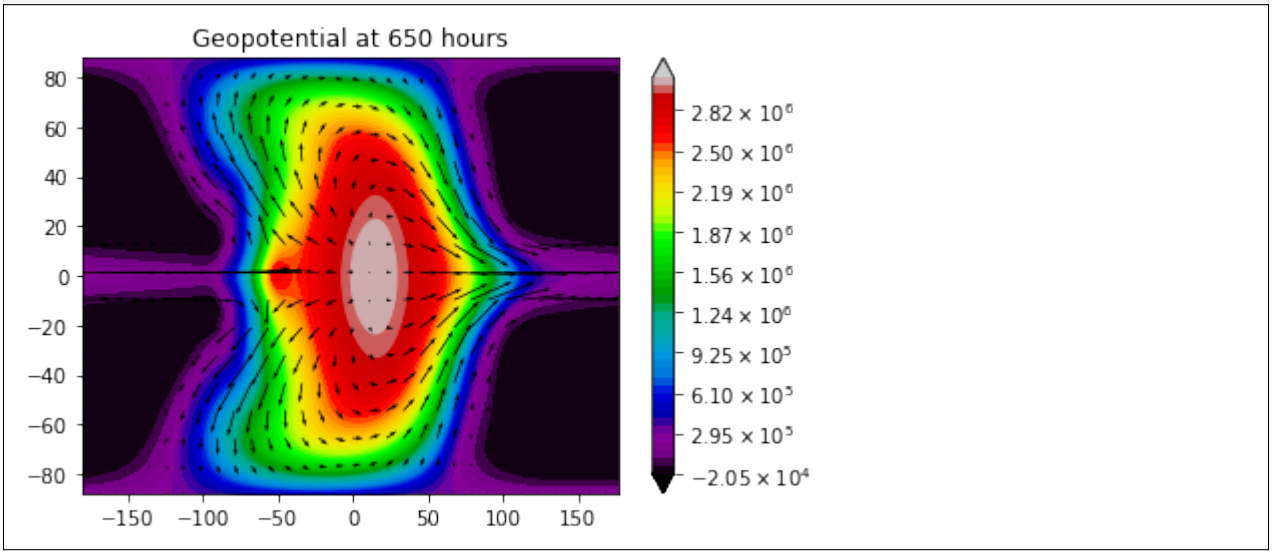
[6]:
```
#change sparseness
sparsenessvec=[2,4,6,8]
for i in range(len(sparsenessvec)):
    fig=SWAMPE.plotting.quiver_geopot_plot(U2,V2,Phi2,lambdas,mus,timestamp2,
↪sparseness=sparsenessvec[i])
```





We can also save the figure. By default, SWAMPE will create a "plots" directory in the current folder. You can also provide a custom path using the optional argument "custompath".

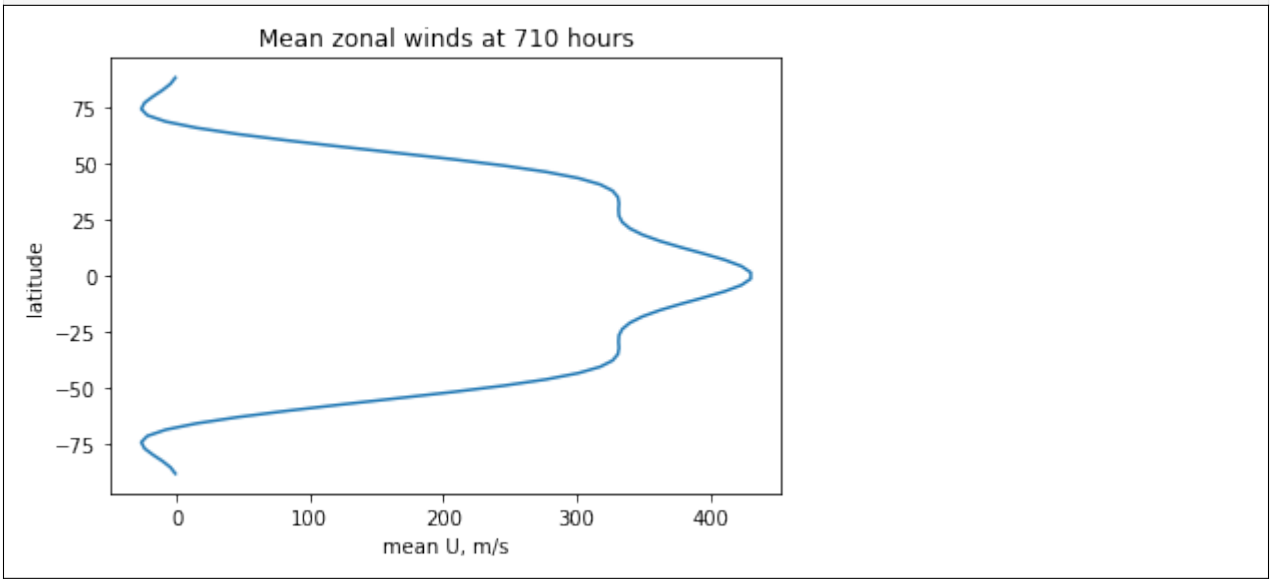
[7]:
```
#save figure tutorial

fig=SWAMPE.plotting.quiver_geopot_plot(U2,V2,Phi2,lambdas,mus,timestamp2,savemyfig=True,
↪filename='geopotfig.pdf')
```

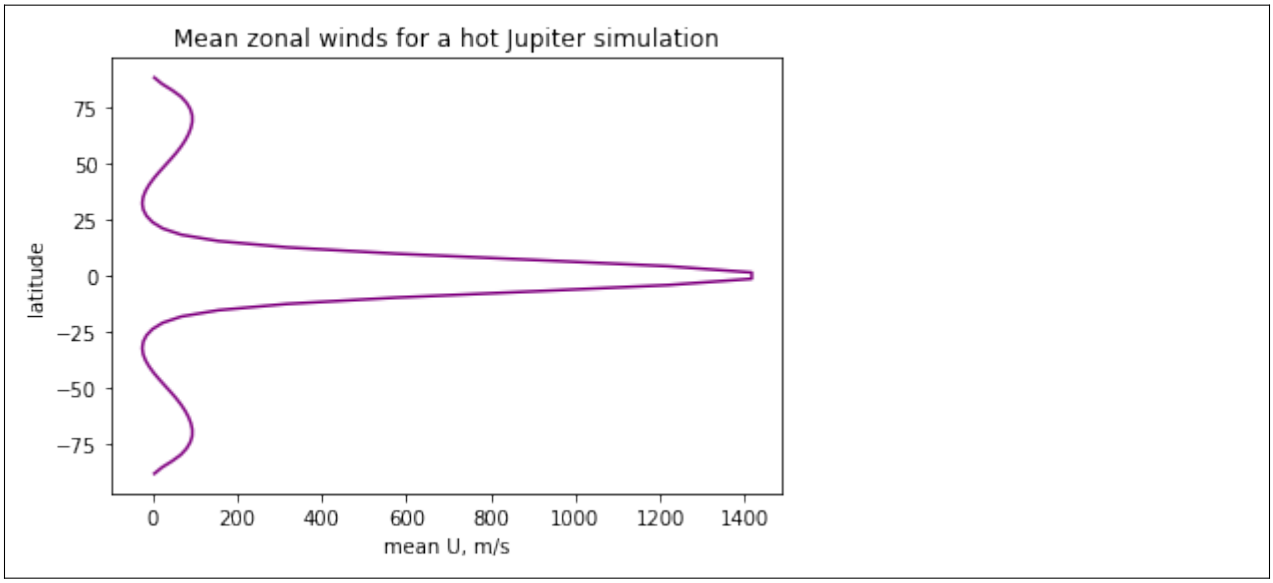


1.4.2 Plotting mean-zonal winds

Now let's plot mean zonal winds:

[8]:
```
# plot mean zonal winds
fig1=SWAMPE.plotting.mean_zonal_wind_plot(U1,mus,timestamp1)
plt.show()
fig2=SWAMPE.plotting.mean_zonal_wind_plot(U2,mus,timestamp2,color='purple',customtitle=
↪'Mean zonal winds for a hot Jupiter simulation')
```





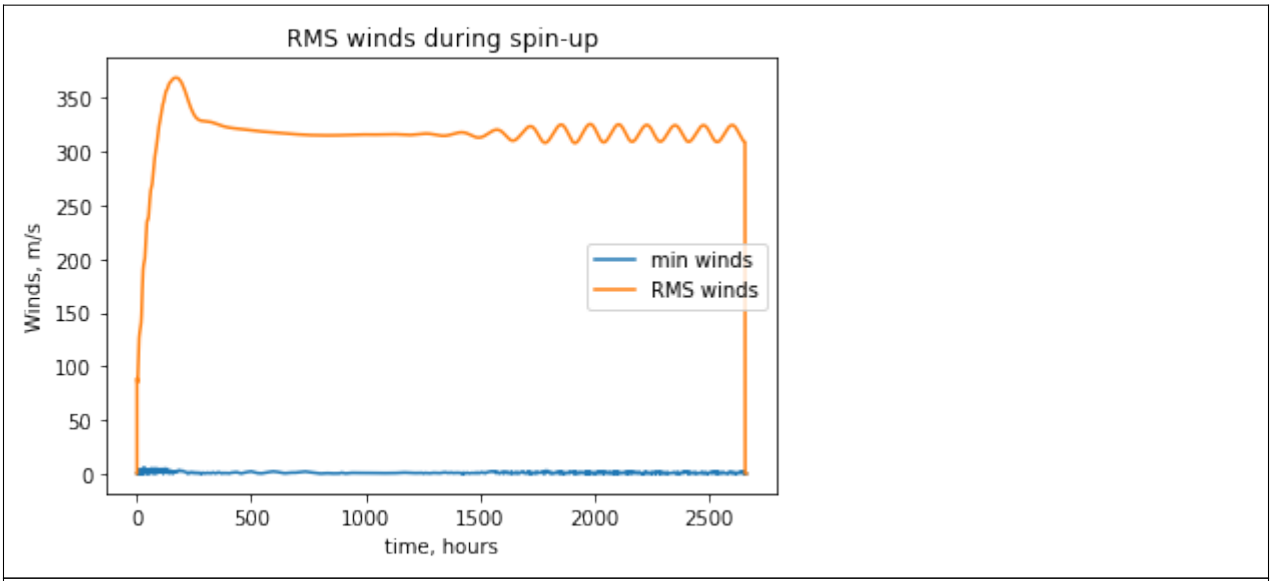
These plots can also be saved similarly to the quiver geopotential plots using "savemyfig".

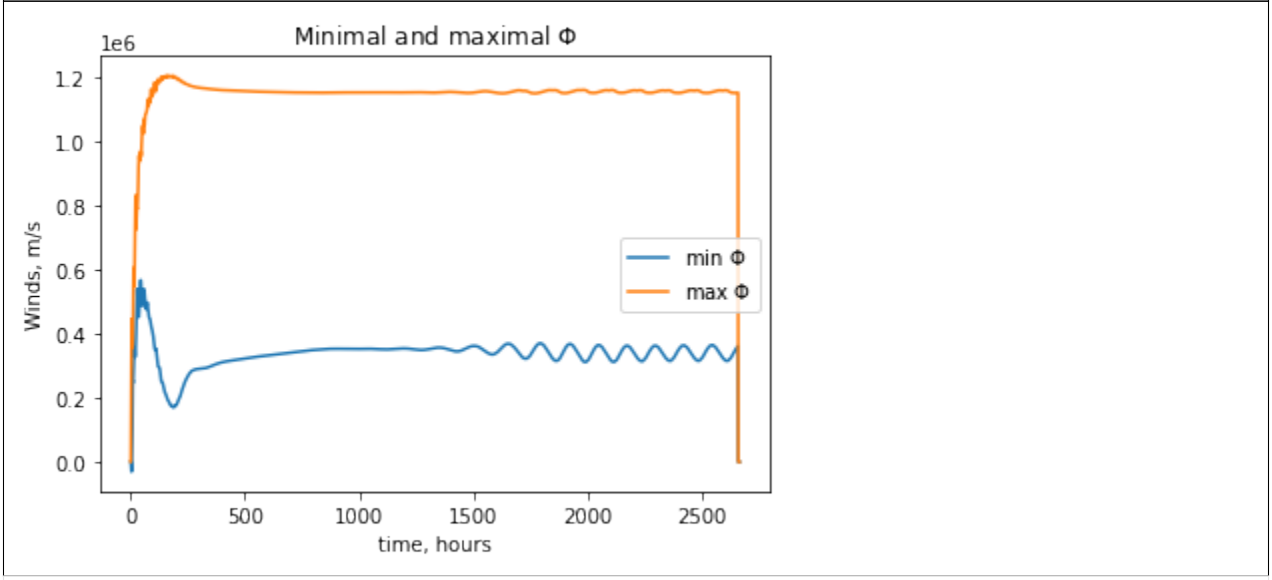
1.4.3 Plotting RMS winds during spinup

GCM simulations can take a while to run. SWAMPE can save spin-up data, specifically RMS wind data as well as the minimum and maximum geopotential.

[9]:
```
#load RMS winds
data_dir_spinup = os.path.abspath('../../../SWAMPE/reference_data/spinup_data/')+'\\'
spinup_winds=SWAMPE.continuation.read_pickle('spinup-winds',custompath=data_dir_spinup)
spinup_geopot=SWAMPE.continuation.read_pickle('spinup-geopot',custompath=data_dir_spinup)
```

[10]:
```
# plot RMS winds
SWAMPE.plotting.spinup_plot(spinup_winds,30)
#plot minimal and maximal geopotential
SWAMPE.plotting.spinup_plot(spinup_geopot,30,customlegend=['min $\Phi$','max $\Phi$'],
↪customtitle='Minimal and maximal $\Phi$')
```





[10]:
```
[<matplotlib.lines.Line2D at 0x2fb13effbe0>]
```

1.4.4 Creating gifs

While many simulations converge to a constant steady-state, in many settings, oscillations are present. In these circumstances, it can be helpful to combine several snapshots into a gif.

[12]:
```
## make a geopotential gif
num_snapshots=11
timestamps=np.zeros(num_snapshots)

Phidata=np.zeros((num_snapshots,J,I))
Udata=np.zeros((num_snapshots,J,I))
Vdata=np.zeros((num_snapshots,J,I))
```

(continues on next page)

(continued from previous page)

```
#load data
data_dir3 = os.path.abspath('../../../SWAMPE/reference_data/SN_for_gif/')+'\\'


for i in range(num_snapshots):
    timestamp=int(2500+10*i)
    timestamps[i]=timestamp
    eta, delta, Phi, U, V =SWAMPE.continuation.load_data(timestamp,custompath=data_dir3)
    Phidata[i,:,:]=Phi
    Udata[i,:,:]=U
    Vdata[i,:,:]=V
```

To generate a gif, we run the following code:

```
SWAMPE.plotting.write_quiver_gif(lambdas,mus,Phidata,Udata,Vdata,timestamps,'testgif.gif
↪',sparseness=4,frms=5)
```

The resulting gif should look like this:

1.5 Continuation

This module contains the functions needed to save and read SWAMPE data, as well as for continuation.

SWAMPE.continuation.**compute_t_from_timestamp**(*units*, *timestamp*, *dt*)

Computes the current timestamp t based on timestamp, units, and timestep size.

Parameters

- **units** (*str*) – Units of timestamps on the savefile: 'hours','minutes', or 'seconds'
- **timestamp** (*int*) – Timestamp in specified units
- **dt** (*float*) – timestep length, in second s

Returns

number of timestep to continue the simulation

Return type

int

SWAMPE.continuation.**compute_timestamp**(*units*, *t*, *dt*)

Computes timestamp in appropriate units to append to the saved data files.

Parameters

- **units** (*str*) – Units of timestamps on the savefile: 'hours','minutes', or 'seconds'
- **t** (*int*) – number of current timestep
- **dt** (*float*) – timestep length, in seconds

Returns

timestamp in desired units

Return type
string

SWAMPE.continuation.**load_data**(*timestamp*, *custompath=None*)

Loads the data necessary for continuation based on timestamp.

Parameters

- **timestamp** (*string*) – timestamp used for naming saved files
- **custompath** (*string*) – path to the custom directory, defaults to None. If None, files will be saved in the parent_directory/data/

Returns
arrays of eta, delta, Phi, U, V

Return type
arrays of float JxI

SWAMPE.continuation.**read_pickle**(*filename*, *custompath=None*)

Loads a pickle file.

Parameters

- **filename** (*string*) – name of the pickle file to be read
- **custompath** (*string, optional*) – path to the custom directory, defaults to None

Returns
any Python type from the pickle file

Return type
any Python type

SWAMPE.continuation.**save_data**(*timestamp*, *etadata*, *deltadata*, *Phidata*, *U*, *V*, *spinupdata*, *geopotdata*, *custompath=None*)

Saves the data for plotting and continuation purposes.

Parameters

- **timestamp** (*string*) – timestamp to be used for naming saved files
- **etadata** (*array of float JxI*) – python array of the data for absolute vorticity eta
- **deltadata** (*array of float JxI*) – python array of the data for divergence delta
- **Phidata** (*array of float JxI*) – python array of the data for geopotential Phi
- **U** (*array of float JxI*) – python array of the data for zonal winds U
- **V** (*array of float JxI*) – python array of the data for meridional winds V
- **spinupdata** (*array of float*) – time series array of minimum length of wind vector and RMS winds
- **geopotdata** (*array of float*) – time series array of minimum and maximum of the geopotential Phi
- **custompath** (*string, optional*) – path to the custom directory, defaults to None. If None, files will be saved in the parent_directory/data/

SWAMPE.continuation.**write_pickle**(*filename*, *data*, *custompath=None*)

Writes a pickle file from the data.

Parameters

- **filename** (*string*) – name of the pickle file to be saved
- **data** (*any Python type*) – a Python array of data to be saved
- **custompath** (*string, optional*) – path to the custom directory, defaults to None. If None, files will be saved in the parent_directory/data/.

1.6 Filters

This module contains the functions associated with filters needed for numerical stability.

SWAMPE.filters.**diffusion**(*Ximn*, *sigma*)

Applies the diffusion filter described in Gelb and Gleeson (eq. 12).

Parameters

- **Ximn** (*list*) – the spectral coefficient data to be filtered
- **sigma** (*float*) – the hyperviscosity coefficient

Return newXimn
filtered spectral coefficient

Return type
array of float

SWAMPE.filters.**modal_splitting**(*Xidataslice*, *alpha*)

Applies the modal splitting filter from Hack and Jacob (1992).

Parameters

- **Xidata** (*list*) – data array to be filtered
- **alpha** (*float*) – filter coefficient

Return newxi
filtered data slice

Return type
array of float

SWAMPE.filters.**sigma**(*M*, *N*, *K4*, *a*, *dt*)

Computes the coefficient for the fourth degree diffusion filter described in Gelb and Gleeson (eq. 12) for vorticity and divergence.

1.6.1 Parameters

param M
spectral dimension

type M
int

param N
highest degree of associated Legendre polynomials

type N
int

param K4
hyperviscosity coefficient

type K4
float

param a
planetary radius, m

type a
float

param dt
time step,s

type dt
float

1.6.2 Returns

return sigma
coefficient for the diffusion filter for geopotential

rtype
array of float

SWAMPE.filters.**sigma6**(*M*, *N*, *K4*, *a*, *dt*)

Computes the coefficient for the sixth degree diffusion filter for vorticity and divergence.

Parameters

- **M** (*int*) – spectral dimension
- **N** (*int*) – highest degree of associated Legendre polynomials
- **K4** (*float64*) – hyperviscosity coefficient
- **a** (*float64*) – planetary radius, m
- **dt** (*float64*) – time step,s

Return sigma
coefficient for the diffusion filter for geopotential

Return type
array of float64

SWAMPE.filters.**sigma6Phi**(*M*, *N*, *K4*, *a*, *dt*)

Computes the coefficient for the fourth degree diffusion Filter described in Gelb and Gleeson (eq. 12) for geopotential.

Parameters

- **M** (*int*) – spectral dimension
- **N** (*int*) – highest degree of associated Legendre polynomials
- **K4** (*float*) – hyperviscosity coefficient
- **a** (*float*) – planetary radius, m
- **dt** (*float*) – time step,s

Return sigma
coefficient for the diffusion filter for geopotential

Return type
array of float

SWAMPE.filters.**sigmaPhi**(*M*, *N*, *K4*, *a*, *dt*)

Computes the coefficient for the fourth degree diffusion Filter described in Gelb and Gleeson (eq. 12) for geopotential.

Parameters

- **M** (*int*) – spectral dimension
- **N** (*int*) – highest degree of associated Legendre polynomials
- **K4** (*float*) – hyperviscosity coefficient
- **a** (*float*) – planetary radius, m
- **dt** (*float*) – time step,s

Return sigma
coefficient for the diffusion filter for geopotential

Return type
array of float

1.7 Forcing

This module contains the functions used for the evaluation of stellar forcing (insolation).

SWAMPE.forcing.**Phieqfun**(*Phibar*, *DPhieq*, *lambdas*, *mus*, *I*, *J*, *g*)

Evaluates the equilibrium geopotential from Perez-Becker and Showman (2013).

1.7.1 Parameters

param Phibar
Mean geopotential

type Phibar
float

param DPhieq
The difference between mean geopotential and the maximum geopotential

type DPhieq
float

param lambdas
Uniformly spaced longitudes of length I.

type lambdas
array of float

param mus
Array of Gaussian latitudes of length J.

type mus
array of float

param I
number of longitudes

type I
int

param J
number of latitudes.

type J
int

param g
Surface gravity, m/s^2.

type g
float

1.7.2 Returns

return PhieqMat
the equilibrium geopotential, array (J,I)

rtype
array of float

SWAMPE.forcing.**Qfun**(*Phieq*, *Phi*, *Phibar*, *taurad*)

Evaluates the radiative forcing on the geopotential. Corresponds to the Q from Perez-Becker and Showman (2013), but has an extra factor of g as we are evaluating the geopotential, and they are evaluating the geopotential height.

1.7.3 Parameters

param Phieq
Equilibrium geopotential, (J,I)

type Phieq
array of float64

param Phi
Geopotential with the mean subtracted, (J,I)

type Phi
array of float64

param Phibar
Mean geopotential

type Phibar
float64

param taurad
radiative time scale, s

type taurad
float64

1.7.4 Returns

return Q
Geopotential forcing, (J,I)

rtype
array of float64

SWAMPE.forcing.**Qfun_with_rampup**(*Phieq*, *Phi*, *Phibar*, *taurad*, *t*, *dt*)

Evaluates the radiative forcing on the geopotential, but slowly ramps up the forcing to improve stability for short radiative timescales.

1.7.5 Parameters

param Phieq
Equilibrium geopotential, (J,I)

type Phieq
array of float64

param Phi
Geopotential with the mean subtracted, (J,I)

type Phi
array of float64

param Phibar
Mean geopotential

type Phibar
float64

param taurad
radiative time scale, s

type taurad
float64

param t
number of current timestep

type t
int

param dt
timestep length

type dt
float64

1.7.6 Returns

return Q
Geopotential forcing, (J,I)

rtype
array of float64

SWAMPE.forcing.**Rfun**(*U*, *V*, *Q*, *Phi*, *Phibar*, *taudrag*)

Evaluates the first and second component of the vector that expresses the velocity forcing in Perez-Becker and Showman. The divergence and vorticity (F,G) correspond to the forcing on the state variables delta and zeta, respectively.

1.7.7 Parameters

param U
zonal velocity component, (J,I)

type U
array of float

param V
meridional velocity component, (J,I)

type V
array of float

param Q
radiative forcing of geopotential, (J,I)

type Q
array of float

param Phi
geopotential with the mean subtracted, (J,I)

type Phi
array of float

param Phibar
mean geopotential

type Phibar
float

param taudrag
drag timescale,in seconds

type taudrag
float

1.7.8 Returns

return

- **F**
  Zonal component of the velocity forcing vector field
- **G**
  Meridional component of the velocity forcing vector field

rtype
arrays of float (J,I)

1.8 Initial conditions

This module contains the initialization functions.

SWAMPE.initial_conditions.**ABCDE_init**(*Uic*, *Vic*, *etaic0*, *Phiic0*, *mus*, *I*, *J*)

Initializes the auxiliary nonlinear components.

Parameters

- **Uic** (*array of float or complex*) – zonal velocity component
- **Vic** (*array of float or complex*) – meridional velocity component
- **etaic0** (*array of float or complex*) – initial eta
- **Phiic0** (*array of float or complex*) – initial Phi
- **mustile** (*array of float or complex*) – reshaped mu array to fit the dimensions

Returns
J by I data arrays for eta0, eta1, delta0, delta1, and phi0, phi1

Return type
array of float or complex

SWAMPE.initial_conditions.**coriolismn**(*M*, *omega*)

Initializes the Coriolis parameter in spectral space.

1.8.1 Parameters

param M
Spectral dimension.

type M
int

param omega
Planetary rotation rate, in radians per second.

type omega
float

1.8.2 Returns

return
fmn, The Coriolis parameter in spectral space.

rtype
array of float, size (M+1, M+1)

SWAMPE.initial_conditions.**spectral_params**(*M*)

Generates the resolution parameters according to Table 1 and 2 from Jakob et al. (1993). Note the timesteps are appropriate for Earth-like forcing. More strongly forced planets will need shorter timesteps.

Parameters
M (*int*) – spectral resolution

1.8.3 Returns

return

- N - int - the highest degree of the Legendre functions for m = 0
- I - int - number of longitudes
- J - int - number of Gaussian latitudes
- dt - float - timestep length, in seconds
- lambdas - array of float of length I - evenly spaced longitudes
- mus - array of float of length J - Gaussian latitudes
- w - array of float of length J - Gaussian weights

SWAMPE.initial_conditions.**state_var_init**(*I*, *J*, *mus*, *lambdas*, *test*, *etaamp*, **args*)

Initializes state variables.

1.8.4 Parameters

param I
number of longitudes.

type I
int

param J
number of latitudes.

type J
int

param mus
Array of Gaussian latitudes of length J.

type mus
array of float

param lambdas
Uniformly spaced longitudes of length I.

type lambdas
array of float

param test
The number of the regime being tested from Williamson et al. (1992)

type test
int

param etaamp
Amplitude of absolute vorticity.

type etaamp
float

param *args
Additional initialization parameters for tests from Williamson et al. (1992)

type *args
arrays of float

1.8.5 Returns

return

- etaic0 - Initial condition for absolute vorticity, (J,I).
- etaic1 - Second initial condition for absolute vorticity, (J,I).
- deltaic0 - Initial condition for divergence, (J,I).
- deltaic1 - Second initial condition for divergence, (J,I).
- Phiic0 - Initial condition for geopotential, (J,I).
- Phiic1 - Second initial condition for geopotential, (J,I).

rtype
tuple of arrays of float

`SWAMPE.initial_conditions.`**`test1_init`**`(`*a*, *omega*, *a1*`)`

Initializes the parameters from Test 1 in Williamson et al. (1992), Advection of Cosine Bell over the Pole.

1.8.6 Parameters

param a
Planetary radius, in meters.

type a
float

param omega
Planetary rotation rate, in radians per second.

type omega
float

param a1
Angle of advection, in radians.

type a1
float

1.8.7 Returns

return

- **SU0**
  Amplitude parameter from Test 1 in Williamson et al. (1992)
- **sina**
  sine of the angle of advection.
- **cosa**
  cosine of the angle of advection.
- **etaamp**
  Amplitude of absolute vorticity.
- **Phiamp**
  Amplitude of geopotential.

rtype
float

`SWAMPE.initial_conditions.`**`velocity_init`**`(`*I*, *J*, *SU0*, *cosa*, *sina*, *mus*, *lambdas*, *test*`)`

Initializes the zonal and meridional components of the wind vector field.

1.8.8 Parameters

param I
number of latitudes.

type I
int

param J
number of longitudes.

type J
int

param SU0
Amplitude parameter from Test 1 in Williamson et al. (1992)

type SU0
float

param cosa
cosine of the angle of advection.

type cosa
float

param sina
sine of the angle of advection.

type sina
float

param mus
Array of Gaussian latitudes of length J

type mus
array of float

param lambdas
Array of uniformly spaces longitudes of length I.

type lambdas
array of float

param test
when applicable, number of test from Williamson et al. (1992).

type test
int

1.8.9 Returns

return

- Uic - (J,I) array - the initial condition for the latitudinal velocity component,
- Vic - (J,I) array - the initial condition for the meridional velocity component.

rtype
array of float

1.9 Model

This module contains the main SWAMPE function which runs the 2D shallow-water general circulation model.

`SWAMPE.model.`**`run_model`**(*M*, *dt*, *tmax*, *Phibar*, *omega*, *a*, *test=None*, *g=9.8*, *forcflag=True*, *taurad=86400*, *taudrag=86400*, *DPhieq=4000000*, *a1=0.05*, *plotflag=True*, *plotfreq=5*, *minlevel=None*, *maxlevel=None*, *diffflag=True*, *modalflag=True*, *alpha=0.01*, *contflag=False*, *saveflag=True*, *expflag=False*, *savefreq=150*, *K6=1.24e+33*, *custompath=None*, *contTime=None*, *timeunits='hours'*, *verbose=True*)

This is the main SWAMPE function which runs the model.

Parameters

- **`M`** (*`int`*) – spectral resolution
- **`dt`** (*`float`*) – time step length in seconds
- **`tmax`** (*`int`*) – number of timesteps to run in the simulation
- **`Phibar`** (*`float`*) – reference geopotential (a good rule of thumb is Phibar=gH, where H is scale height in meters)
- **`omega`** (*`float`*) – planetary rotation rate in radians/s
- **`a`** (*`float`*) – planetary radius in meters
- **`test`** (*`int, optional`*) – number of test from Jakob & Hack (1993), tests 1 and 2 are supported, defaults to None
- **`g`** (*`float, optional`*) – surface gravity, defaults to 9.8 m/s^2

- **forcflag** (*bool, optional*) – option to implement radiative forcing from the host star, defaults to True
- **taurad** (*float, optional*) – radiative timescale for Newtonian relaxation in the forcing scheme, defaults to 86400 s
- **taudrag** (*float, optional*) – drag timescale for wind forcing, defaults to 86400 s
- **DPhieq** (*float, optional*) – day/night amplitude of prescribed local radiative equilibrium geopotential, defaults to 4*(10**6) m^2/s^2.
- **a1** (*float, optional*) – angle for tests 1 and 2 from Jakob and Hack (1993), defaults to 0.05
- **plotflag** (*bool, optional*) – option to display progress plots over the course of the simulation run, defaults to True
- **plotfreq** (*int, optional*) – frequency of plot output during the simulation run, defaults to once every 5 timesteps
- **minlevel** (*float, optional*) – minimum level of colorbar for geopotential plotting, defaults to minimum geopotential
- **maxlevel** (*float, optional*) – maximum level of colorbar for geopotential plotting, defaults to minimum geopotential
- **diffflag** (*bool, optional*) – option to turn on the diffusion/hyperviscosity filter, defaults to True (strongly recommended)
- **modalflag** (*bool, optional*) – option to turn on the modal splitting filter from Hack and Jakob (1992), defaults to True
- **alpha** (*float, optional*) – parameter for the modal splitting filter from Hack and Jakob (1992), defaults to 0.01
- **contflag** (*bool, optional*) – option to continue the simulation from saved data, defaults to False
- **saveflag** (*bool, optional*) – option to save data as pickle files, defaults to True
- **expflag** (*bool, optional*) – option to use explicit time-stepping scheme instead of modified Euler, defaults to False (strongly recommended to use the modified Euler scheme)
- **savefreq** (*int, optional*) – frequency of saving the data, defaults to once every 150 timesteps
- **K6** (*float, optional*) – sixth order hyperviscosity filter parameter, defaults to 1.24*10**33
- **custompath** (*str, optional*) – option to specify the path for saving the data. By default SWAMPE will make a local directory data/ and store files there.
- **contTime** (*int, optional*) – (if continuing from saved data) timestamp of the data, defaults to None
- **timeunits** (*str, optional*) – time units, defaults to 'hours', also supports 'minutes' and 'seconds'
- **verbose** (*bool, optional*) – option to print progress statements, defaults to True

1.10 Plotting

This module contains functions related to generating figures and gifs with SWAMPE.

SWAMPE.plotting.**fmt**(*x*, *pos*)

Generates the format for scientific notation in axis and colorbar labels.

SWAMPE.plotting.**gif_helper**(*fig*, *dpi=200*)

Converts the figure to image format for gif generation.

Parameters

- **fig** (*matplotlib figure*) – figure
- **dpi** (*int, optional*) – resolution, defaults to 200

Returns
image

Return type
numerical image

SWAMPE.plotting.**mean_zonal_wind_plot**(*plotdata*, *mus*, *timestamp*, *units='hours'*, *customtitle=None*, *customxlabel=None*, *savemyfig=False*, *filename=None*, *custompath=None*, *color=None*)

Generates a plot of mean zonal winds (averaged across all longitudes).

Parameters

- **plotdata** (*array of float*) – Wind data, ususally U, of size (J,I).
- **mus** (*array of float*) – Array of Gaussian latitudes of length J.
- **timestamp** (*float*) – time of snapshot
- **units** (*str, optional*) – units of timestamp, defaults to 'hours'
- **customtitle** (*string, optional*) – option to change the title, defaults to None
- **customxlabel** (*str, optional*) – option to change the label of the x-axis, defaults to None
- **savemyfig** (*bool, optional*) – option to save the figure, defaults to False
- **filename** (*str, optional*) – file name for saving the figure, defaults to None
- **custompath** (*str, optional*) – path for saving the figure, defaults to None
- **color** (*str, optional*) – option to change the color of the plot, defaults to None

Returns
figure

Return type
matplotlib figure

SWAMPE.plotting.**quiver_geopot_plot**(*U*, *V*, *Phi*, *lambdas*, *mus*, *timestamp*, *sparseness=4*, *minlevel=None*, *maxlevel=None*, *units='hours'*, *customtitle=None*, *savemyfig=False*, *filename=None*, *custompath=None*, *axlabels=False*, *colormap=None*)

Generates a quiver plot with the geopotential field and overlayed wind vectors.

Parameters

- **U** (*array of float*) – lat-lon zonal wind component, (J,I)

- **V** (*array of float*) – lat-lon meridional wind component, (J,I)
- **Phi** (*array of float*) – lat-lon geopotential field, (J,I)
- **lambdas** (*array of float*) – Uniformly spaced longitudes of length I.
- **mus** (*array of float*) – Gaussian latitudes of length J.
- **timestamp** (*float*) – time of snapshot
- **sparseness** (*int, optional*) – spacing of overlayed wind vector field, defaults to 4
- **minlevel** (*float, optional*) – colorbar minimum for geopotential plotting, defaults to minimum of Phi
- **maxlevel** (*float, optional*) – colorbar maximum for geopotential plotting, defaults to maximum of Phi
- **units** (*str, optional*) – units of timestamp, defaults to 'hours'
- **customtitle** (*string, optional*) – option to change the title, defaults to None
- **savemyfig** (*bool, optional*) – option to save the figure, defaults to False
- **filename** (*str, optional*) – file name for saving the figure, defaults to None
- **custompath** (*str, optional*) – path for saving the figure, defaults to None
- **axlabels** (*bool, optional*) – option to display axis labels for latitude and longitude, defaults to False
- **colormap** (*matplotlib colormap, optional*) – option to change the colormap, defaults to nipy.spectral

Returns
figure

Return type
matplotlib figure

SWAMPE.plotting.**spinup_plot**(*plotdata*, *dt*, *units='hours'*, *customtitle=None*, *customxlabel=None*, *customylabel=None*, *savemyfig=False*, *filename=None*, *custompath=None*, *color=None*, *legendflag=True*, *customlegend=None*)

Generates a plot of RMS winds and minimal winds over time. Can be useful for monitoring spinup.

Parameters

- **plotdata** (*array of float*) – Spinup winds, of size (2,tmax).
- **dt** (*float*) – timestep length, in seconds
- **units** (*str, optional*) – units of timestamp, defaults to 'hours'
- **customtitle** (*string, optional*) – option to change the title, defaults to None
- **customxlabel** (*str, optional*) – option to change the label of the x-axis, defaults to None
- **customylabel** (*str, optional*) – option to change the label of the y-axis, defaults to None
- **savemyfig** (*bool, optional*) – option to save the figure, defaults to False
- **filename** (*str, optional*) – file name for saving the figure, defaults to None
- **custompath** (*str, optional*) – path for saving the figure, defaults to None

- **color** (*array of string, optional*) – option to specify array of two colors ["color1", "color2"], defaults to None
- **legendflag** (*bool, optional*) – option to display legend, defaults to True
- **customlegend** (*array of string, optional*) – option to customize the legend, defaults to None

Returns
figure

Return type
matplotlib figure

SWAMPE.plotting.**write_quiver_gif**(*lambdas*, *mus*, *Phidata*, *Udata*, *Vdata*, *timestamps*, *filename*, *frms=5*, *sparseness=4*, *dpi=200*, *minlevel=None*, *maxlevel=None*, *units='hours'*, *customtitle=None*, *custompath=None*, *axlabels=False*, *colormap=None*)

Writes a gif generated from a series of geopotential quiver plots.

Parameters

- **lambdas** (*array of float*) – Uniformly spaced longitudes of length I.
- **mus** (*array of float*) – Gaussian latitudes of length J.
- **Phidata** (*array of float*) – array of geopotential snapshots (num_snapshots,J,I)
- **Udata** (*array of float*) – array of zonal wind component snapshots (num_snapshots,J,I)
- **Vdata** (*array of float*) – array of meridional wind component snapshots (num_snapshots,J,I)
- **timestamps** (*array of float*) – array of timestamps for the snapshots, length corresponding to (num_snapshots)
- **filename** (*str*) – name of the gif file, should end in ".gif"
- **frms** (*int, optional*) – frames per second, defaults to 5
- **sparseness** (*int, optional*) – spacing of the wind vector field, defaults to 4
- **dpi** (*int, optional*) – resolution, defaults to 200
- **minlevel** (*float, optional*) – colorbar minimum for geopotential plotting, defaults to minimum of Phi
- **maxlevel** (*float, optional*) – colorbar maximum for geopotential plotting, defaults to maximum of Phi
- **units** (*str, optional*) – units of timestamp, defaults to 'hours'
- **customtitle** (*string, optional*) – option to change the title, defaults to None
- **custompath** (*str, optional*) – path for saving the figure, defaults to None
- **axlabels** (*bool, optional*) – option to display axis labels for latitude and longitude, defaults to False
- **colormap** (*matplotlib colormap, optional*) – option to change the colormap, defaults to nipy.spectral

1.11 Spherical Harmonic Transforms

SWAMPE.spectral_transform.**PmnHmn**(*J*, *M*, *N*, *mus*)

Calculates the values of associated Legendre polynomials and their derivatives evaluated at Gaussian latitudes (mus) up to wavenumber M.

Parameters

- **J** (*int*) – number of latitudes
- **M** (*int*) – highest wavenumber for associated Legendre polynomials
- **N** (*int*) – highest degree of the Legendre functions for m=0
- **mus** (*array of floats*) – Gaussian latitudes

Returns

Pmn array (associated legendre polynomials), Hmn array (derivatives of Pmn*(1-x^2)), both evaluated at the Gaussian latitudes mus

Return type

array of float

SWAMPE.spectral_transform.**diagnostic_eta_delta**(*Um*, *Vm*, *fmn*, *I*, *J*, *M*, *N*, *Pmn*, *Hmn*, *w*, *tstepcoeff*, *mJarray*, *dt*)

Computes vorticity and divergence from zonal and meridional wind fields. This is a diagnostic relationship. For details, see Hack and Jakob (1992) equations (5.26)-(5.27).

1.11.1 Parameters

param Um
Fourier coefficient of zonal winds

type Um
array of float

param Vm
Fourier coefficient of meridional winds

type Vm
array of float

param fmn
spectal coefficients of the Coriolic force

type fmn
array of float

param I
number of longitudes

type I
int

param J
number of Gaussian latitudes

type J
int

param M
highest wavenumber for associated Legendre polynomials

type M
int

param N
highest degree of associated Legendre polynomials

type N
int

param Pmn
values of the associated Legendre polynomials at Gaussian latitudes mus up to wavenumber M

type Pmn
array of float64

param Hmn
values of the associated Legendre polynomial derivatives at Gaussian latitudes up to wavenumber M

type Hmn
array of float

param w
Gauss Legendre weights

type w
array of float

param tstepcoeff
a coefficient for time-stepping of the form 2dt/(a(1-mus^2) from Hack and Jakob (1992)

type tstepcoeff
array of float

param mJarray
coefficients equal to m=0,1,. . . ,M

type mJarray
array of float

param dt
time step, in seconds

type dt
float Returns

Absolute vorticity

- **newdelta**
  Divergence
- **etamn**
  Spectral coefficients of absolute vorticity
- **deltamn**
  Spectral coefficients of divergence

rtype
array of float

SWAMPE.spectral_transform.**fwd_fft_trunc**(*data*, *I*, *M*)

Calculates and truncates the fast forward Fourier transform of the input.

Parameters

- **data** (*array of float*) – array of dimension IxJ (usually the values of state variables at lat-long coordinates)
- **I** (*int*) – number of longitudes
- **M** (*int*) – highest wavenumber for associated Legendre polynomials

Return datam
Fourier coefficients 0 through M

Rtype datam
array of complex

SWAMPE.spectral_transform.**fwd_leg**(*data*, *J*, *M*, *N*, *Pmn*, *w*)

Calculates the forward legendre transform.

Parameters

- **data** (*array of float or array of complex*) – input to be transformed (usually output of fft)
- **J** (*int*) – number of latitudes
- **M** (*int*) – highest wavenumber for associated Legendre polynomials
- **N** (*int*) – highest degree of the Legendre functions for m=0
- **Pmn** (*array of float*) – associated legendre functions evaluated at the Gaussian latitudes mus up to wavenumber M
- **w** (*array of float*) – Gauss Legendre weights

Return legcoeff
Legendre coefficients (if data was output from FFT, then legcoeff are the spectral coefficients)

Rtype legcoeff
array of complex

SWAMPE.spectral_transform.**invrsUV**(*deltamn*, *etamn*, *fmn*, *I*, *J*, *M*, *N*, *Pmn*, *Hmn*, *tstepcoeffmn*, *marray*)

Computes the wind velocity from the values of vorticity and divergence. This is a diagnostic relationship. For details, see Hack and Jakob (1992) equations (5.24)-(5.25).

1.11.2 Parameters

param deltamn
Fourier coefficients of divergence

type deltamn
array of complex

param etamn
Fourier coefficients of vorticity

type etamn
array of complex

param fmn
spectral coefficients of the Coriolis force

type etamn
array of float

param I
number of longitudes

type I
int

param J
number of latitudes

type J
int

param M
highest wavenumber for associated Legendre polynomials

type M
int

param N
highest degree of associated Legendre polynomials

type N
int

param Pmn
values of the associated Legendre polynomials at Gaussian latitudes mus up to wavenumber M

type Pmn
array of float

param Hmn
values of the associated Legendre polynomial derivatives at Gaussian latitudes up to wavenumber M

type Hmn
array of float

param tstepcoeffmn
coefficient to scale spectral components

type tstepcoeffmn
array of float

param marray
array to multiply a quantity by a factor of m ranging from 0 through M.

type marray
array of float

1.11.3 Returns

return

- **Unew**
  Zonal velocity component
- **Vnew**
  Meridional velocity component

rtype
array of float

SWAMPE.spectral_transform.**invrs_fft**(*approxXim*, *I*)

Calculates the inverse Fourier transform.

Parameters

- **approxXim** (*array of complex*) – Fourier coefficients
- **I** (*int*) – number of longitudes

Returns
long-lat coefficients

Return type
array of complex

SWAMPE.spectral_transform.**invrs_leg**(*legcoeff*, *I*, *J*, *M*, *N*, *Pmn*)

Calculates the inverse Legendre transform.

Parameters

- **legcoeff** (*array of complex*) – Legendre coefficients
- **J** (*int*) – number of latitudes
- **M** (*int*) – highest wavenumber for associated Legendre polynomials
- **Pmn** (*array of float*) – associated legendre functions evaluated at the Gaussian latitudes mus up to wavenumber M

Returns
transformed spectral coefficients

Return type
array of complex

1.12 Time stepping

This module contains the function that calls an explicit or an implicit time-stepping scheme, as well as the functions that compute the arrays of coefficients involved in time-stepping.

SWAMPE.time_stepping.**RMS_winds**(*a*, *I*, *J*, *lambdas*, *mus*, *U*, *V*)

Computes RMS winds based on the zonal and meridional wind fields.

Parameters

- **a** (*float*) – planteray radius, m

- **I** (*int*) – number of longitudes
- **J** (*int*) – number of Gaussian latitudes
- **mus** (*array of float*) – Array of Gaussian latitudes of length J
- **lambdas** (*array of float*) – Array of uniformly spaces longitudes of length I.
- **U** (*array of float*) – zonal wind field, JxI
- **V** (*array of float*) – meridional wind field, JxI

Returns
RMS winds value

Return type
float

SWAMPE.time_stepping.**mJarray**(*J*, *M*)

Computes coefficients equal to m=0,1,...,M.

Parameters

- **J** (*int*) – number of Gaussian latitudes
- **M** (*int*) – spectral dimension

Return mJarray
coefficient m in a matrix of size (J, M+1)

Return type
array of float

SWAMPE.time_stepping.**marray**(*M*, *N*)

Computes coefficients equal to m=0,1,...,M.

Parameters

- **M** (*int*) – highest wavenumber of associated Legendre polynomials
- **N** (*int*) – highest degreee of associated Legendre polynomials

Return marray
coefficient m in a matrix of size (M+1, N+1)

Return type
array of float

SWAMPE.time_stepping.**narray**(*M*, *N*)

Computes the array n(n+1).

Parameters

- **M** (*int*) – spectral dimension
- **N** (*int*) – highest degreee of associated Legendre polynomials

Return narray
coefficients n(n+1) in a matrix of size (M+1,N+1)

Return type
array of float

SWAMPE.time_stepping.**tstepcoeff**(*J*, *M*, *dt*, *mus*, *a*)

Computes a coefficient for time-stepping of the form 2dt/(a(1-mus^2)) from Hack and Jakob (1992).

Parameters

- **J** (*int*) – number of Gaussian latitudes
- **M** (*int*) – spectral dimension
- **mus** (*array of float*) – array of Gaussian latitudes of length J
- **a** (*float*) – planetary radius, m

Return tstepcoeff
coefficients 2dt/(a(1-mus^2)) in a matrix of size (J,M+1)

Return type
array of float

SWAMPE.time_stepping.**tstepcoeff2**(*J*, *M*, *dt*, *a*)

Computes the time stepping coefficient of the form 2dt/a^2 from Hack and Jakob (1992).

Parameters

- **J** (*int*) – number of Gaussian latitudes
- **M** (*int*) – spectral dimension
- **dt** (*float*) – time step length, s
- **a** (*float*) – planetary radius, m

Return tstepcoeff2
time stepping coefficients of size (J,M+1)

Return type
array of float

SWAMPE.time_stepping.**tstepcoeffmn**(*M*, *N*, *a*)

Generates the coefficient that multiplies spectral components in invrsUV.

Parameters

- **M** (*int*) – highest wave number
- **N** – highest degree of associated legendre polynomial for m=0
- **a** (*float*) – radius of the planet, in meters

Returns
an array of coefficients a/(n(n+1))

Return type
list

SWAMPE.time_stepping.**tstepping**(*etam0*, *etam1*, *deltam0*, *deltam1*, *Phim0*, *Phim1*, *I*, *J*, *M*, *N*, *Am*, *Bm*, *Cm*, *Dm*, *Em*, *Fm*, *Gm*, *Um*, *Vm*, *fmn*, *Pmn*, *Hmn*, *w*, *tstepcoeff*, *tstepcoeff2*, *tstepcoeffmn*, *marray*, *mJarray*, *narray*, *PhiFm*, *dt*, *a*, *Phibar*, *taurad*, *taudrag*, *forcflag*, *diffflag*, *expflag*, *sigma*, *sigmaPhi*, *test*, *t*)

Calls the timestepping scheme.

Parameters

- **etam0** (*array of float*) – Fourier coefficents of absolute vorticity for one time step

- **etam1** (*array of float*) – Fourier coefficents of absolute vorticity for the following time step
- **deltam0** (*array of float*) – Fourier coefficents of divergence for one time step
- **deltam1** (*array of float*) – Fourier coefficents of divergence for the following time step
- **Phim0** (*array of float*) – Fourier coefficents of geopotential for one time step
- **Phim1** (*array of float*) – Fourier coefficents of geopotential for the following time step
- **I** (*int*) – number of longitudes
- **J** (*int*) – number of Gaussian latitudes
- **N** (*int*) – highest degree of the Legendre functions for m=0
- **Am** (*array of float*) – Fourier coefficients of the nonlinear component A=U*eta
- **Bm** (*array of float*) – Fourier coefficients of the nonlinear component B=V*eta
- **Cm** (*array of float*) – Fourier coefficients of the nonlinear component C=U*Phi
- **Dm** (*array of float*) – Fourier coefficients of the nonlinear component D=V*Phi
- **Em** (*array of float*) – Fourier coefficients of the nonlinear component E=(U^2+V^2)/(2(1-mu^2))
- **Fm** (*array of float*) – Fourier coefficients of the zonal component of wind forcing
- **Gm** (*array of float*) – Fourier coefficients of the meridional component of wind forcing
- **Um** (*array of float*) – Fourier coefficients of the zonal component of wind
- **Vm** (*array of float*) – Fourier coefficients of the meridional component of wind
- **fmn** (*array of float*) – spectral coefficients of the Coriolis force
- **Pmn** (*array of float*) – associated legendre functions evaluated at the Gaussian latitudes mus up to wavenumber M
- **Hmn** (*array of float*) – derivatives of the associated legendre functions evaluated at the Gaussian latitudes mus up to wavenumber M
- **w** (*array of float*) – Gauss Legendre weights
- **tstepcoeff** (*array of float*) – coefficient for time-stepping of the form 2dt/(a(1-mus^2))
- **tstepcoeff2** (*array of float*) – time stepping coefficient of the form 2dt/a^2
- **tstepcoeffmn** (*array of float*) – an array of coefficients a/(n(n+1))
- **marray** (*array of float*) – coefficients equal to m=0,1,...,M in a matrix M+1xN+1
- **mJarray** (*array of float*) – coefficients equal to m=0,1,...,M in a matrix M+1xJ
- **narray** (*array of float*) – array n(n+1) in a matrix M+1xN+1
- **PhiFm** – Fourier coefficients of the geopotential forcing
- **dt** (*float*) – time step, in seconds
- **a** (*float*) – planetary radius, m
- **Phibar** (*float*) – time-invariant spatial mean geopotential, height of the top layer
- **taurad** (*float*) – radiative timescale
- **taudrag** (*float*) – drag timescale

- **forcflag** (*float*) – forcing flag
- **diffflag** (*float*) – hyperdiffusion filter flag
- **sigma** (*array of float*) – hyperdiffusion filter coefficients for absolute vorticity and divergence
- **sigmaPhi** (*array of float*) – hyperdiffusion filter coefficients for geopotential
- **test** (*int*) – number of test, defaults to None
- **t** (*int*) – number of current time step

Returns

- **newetamn**
  Updated spectral coefficients of absolute vorticity
- **newetatstep**
  Updated absolute vorticity
- **newdeltamn**
  Updated spectral coefficients of divergence
- **newdeltatstep**
  Updated divergence
- **newPhimn**
  Updated spectral coefficients of geopotential
- **newPhitstep**
  Updated geopotential
- **Unew**
  Updated zonal winds
- **Vnew**
  Updated meridional winds

Return type
array of float

1.13 Modified Euler's Method

This module contains the functions associated with the modified Euler time-stepping scheme. The associated coefficients and the method are outlined in the Methods section of this documentation.

SWAMPE.modEuler_tdiff.**delta_timestep**(*etam0*, *etam1*, *deltam0*, *deltam1*, *Phim0*, *Phim1*, *I*, *J*, *M*, *N*, *Am*, *Bm*, *Cm*, *Dm*, *Em*, *Fm*, *Gm*, *Um*, *Vm*, *Pmn*, *Hmn*, *w*, *tstepcoeff1*, *tstepcoeff2*, *mJarray*, *narray*, *PhiFm*, *dt*, *a*, *Phibar*, *taurad*, *taudrag*, *forcflag*, *diffflag*, *sigma*, *sigmaPhi*, *test*, *t*)

This function timesteps the divergence delta forward.

Parameters

- **etam0** (*array of float*) – Fourier coefficents of absolute vorticity for one time step
- **etam1** (*array of float*) – Fourier coefficents of absolute vorticity for the following time step
- **deltam0** (*array of float*) – Fourier coefficents of divergence for one time step

- **deltam1** (*array of float*) – Fourier coefficents of divergence for the following time step
- **Phim0** (*array of float*) – Fourier coefficents of geopotential for one time step
- **Phim1** (*array of float*) – Fourier coefficents of geopotential for the following time step
- **I** (*int*) – number of longitudes
- **J** (*int*) – number of Gaussian latitudes
- **N** (*int*) – highest degree of the Legendre functions for m=0
- **Am** (*array of float*) – Fourier coefficients of the nonlinear component A=U*eta
- **Bm** (*array of float*) – Fourier coefficients of the nonlinear component B=V*eta
- **Cm** (*array of float*) – Fourier coefficients of the nonlinear component C=U*Phi
- **Dm** (*array of float*) – Fourier coefficients of the nonlinear component D=V*Phi
- **Em** (*array of float*) – Fourier coefficients of the nonlinear component E=(U^2+V^2)/(2(1-mu^2))
- **Fm** (*array of float*) – Fourier coefficients of the zonal component of wind forcing
- **Gm** (*array of float*) – Fourier coefficients of the meridional component of wind forcing
- **Um** (*array of float*) – Fourier coefficients of the zonal component of wind
- **Vm** (*array of float*) – Fourier coefficients of the meridional component of wind
- **fmn** (*array of float*) – spectral coefficients of the Coriolis force
- **Pmn** (*array of float*) – associated legendre functions evaluated at the Gaussian latitudes mus up to wavenumber M
- **Hmn** (*array of float*) – derivatives of the associated legendre functions evaluated at the Gaussian latitudes mus up to wavenumber M
- **w** (*array of float*) – Gauss Legendre weights
- **tstepcoeff** (*array of float*) – coefficient for time-stepping of the form 2dt/(a(1-mus^2))
- **tstepcoeff2** (*array of float*) – time stepping coefficient of the form 2dt/a^2
- **tstepcoeffmn** (*array of float*) – an array of coefficients a/(n(n+1))
- **marray** (*array of float*) – coefficients equal to m=0,1,...,M in a matrix M+1xN+1
- **mJarray** (*array of float*) – coefficients equal to m=0,1,...,M in a matrix M+1xJ
- **narray** (*array of float*) – array n(n+1) in a matrix M+1xN+1
- **PhiFm** – Fourier coefficients of the geopotential forcing
- **dt** (*float*) – time step, in seconds
- **a** (*float*) – planetary radius, m
- **Phibar** (*float*) – time-invariant spatial mean geopotential, height of the top layer
- **taurad** (*float*) – radiative timescale
- **taudrag** (*float*) – drag timescale
- **forcflag** (*float*) – forcing flag
- **diffflag** (*float*) – hyperdiffusion filter flag

- **sigma** (*array of float*) – hyperdiffusion filter coefficients for absolute vorticity and divergence
- **sigmaPhi** (*array of float*) – hyperdiffusion filter coefficients for geopotential
- **test** (*int*) – number of test, defaults to None
- **t** (*int*) – number of current time step

Returns

- **deltamntstep**
  Updated spectral coefficients of divergence
- **newdeltatstep**
  Updated divergence

Return type
array of float

SWAMPE.modEuler_tdiff.**eta_timestep**(*etam0*, *etam1*, *deltam0*, *deltam1*, *Phim0*, *Phim1*, *I*, *J*, *M*, *N*, *Am*, *Bm*, *Cm*, *Dm*, *Em*, *Fm*, *Gm*, *Um*, *Vm*, *Pmn*, *Hmn*, *w*, *tstepcoeff1*, *tstepcoeff2*, *mJarray*, *narray*, *PhiFm*, *dt*, *a*, *Phibar*, *taurad*, *taudrag*, *forcflag*, *diffflag*, *sigma*, *sigmaPhi*, *test*, *t*)

This function timesteps the absolute vorticity eta forward.

Parameters

- **etam0** (*array of float*) – Fourier coefficents of absolute vorticity for one time step
- **etam1** (*array of float*) – Fourier coefficents of absolute vorticity for the following time step
- **deltam0** (*array of float*) – Fourier coefficents of divergence for one time step
- **deltam1** (*array of float*) – Fourier coefficents of divergence for the following time step
- **Phim0** (*array of float*) – Fourier coefficents of geopotential for one time step
- **Phim1** (*array of float*) – Fourier coefficents of geopotential for the following time step
- **I** (*int*) – number of longitudes
- **J** (*int*) – number of Gaussian latitudes
- **N** (*int*) – highest degree of the Legendre functions for m=0
- **Am** (*array of float*) – Fourier coefficients of the nonlinear component A=U*eta
- **Bm** (*array of float*) – Fourier coefficients of the nonlinear component B=V*eta
- **Cm** (*array of float*) – Fourier coefficients of the nonlinear component C=U*Phi
- **Dm** (*array of float*) – Fourier coefficients of the nonlinear component D=V*Phi
- **Em** (*array of float*) – Fourier coefficients of the nonlinear component E=(U^2+V^2)/(2(1-mu^2))
- **Fm** (*array of float*) – Fourier coefficients of the zonal component of wind forcing
- **Gm** (*array of float*) – Fourier coefficients of the meridional component of wind forcing
- **Um** (*array of float*) – Fourier coefficients of the zonal component of wind
- **Vm** (*array of float*) – Fourier coefficients of the meridional component of wind
- **fmn** (*array of float*) – spectral coefficients of the Coriolis force

- **Pmn** (*array of float*) – associated legendre functions evaluated at the Gaussian latitudes mus up to wavenumber M
- **Hmn** (*array of float*) – derivatives of the associated legendre functions evaluated at the Gaussian latitudes mus up to wavenumber M
- **w** (*array of float*) – Gauss Legendre weights
- **tstepcoeff** (*array of float*) – coefficient for time-stepping of the form 2dt/(a(1-mus^2))
- **tstepcoeff2** (*array of float*) – time stepping coefficient of the form 2dt/a^2
- **tstepcoeffmn** (*array of float*) – an array of coefficients a/(n(n+1))
- **marray** (*array of float*) – coefficients equal to m=0,1,...,M in a matrix M+1xN+1
- **mJarray** (*array of float*) – coefficients equal to m=0,1,...,M in a matrix M+1xJ
- **narray** (*array of float*) – array n(n+1) in a matrix M+1xN+1
- **PhiFm** – Fourier coefficients of the geopotential forcing
- **dt** (*float*) – time step, in seconds
- **a** (*float*) – planetary radius, m
- **Phibar** (*float*) – time-invariant spatial mean geopotential, height of the top layer
- **taurad** (*float*) – radiative timescale
- **taudrag** (*float*) – drag timescale
- **forcflag** (*float*) – forcing flag
- **diffflag** (*float*) – hyperdiffusion filter flag
- **sigma** (*array of float*) – hyperdiffusion filter coefficients for absolute vorticity and divergence
- **sigmaPhi** (*array of float*) – hyperdiffusion filter coefficients for geopotential
- **test** (*int*) – number of test, defaults to None
- **t** (*int*) – number of current time step

Returns

- **etamntstep**
  Updated spectral coefficients of absolute vorticity
- **newetatstep**
  Updated absolute vorticity

Return type
array of float

SWAMPE.modEuler_tdiff.**phi_timestep**(*etam0*, *etam1*, *deltam0*, *deltam1*, *Phim0*, *Phim1*, *I*, *J*, *M*, *N*, *Am*, *Bm*, *Cm*, *Dm*, *Em*, *Fm*, *Gm*, *Um*, *Vm*, *Pmn*, *Hmn*, *w*, *tstepcoeff1*, *tstepcoeff2*, *mJarray*, *narray*, *PhiFm*, *dt*, *a*, *Phibar*, *taurad*, *taudrag*, *forcflag*, *diffflag*, *sigma*, *sigmaPhi*, *test*, *t*)

This function timesteps the geopotential Phi forward.

Parameters

- **etam0** (*array of float*) – Fourier coefficents of absolute vorticity for one time step

- **etam1** (*array of float*) – Fourier coefficents of absolute vorticity for the following time step
- **deltam0** (*array of float*) – Fourier coefficents of divergence for one time step
- **deltam1** (*array of float*) – Fourier coefficents of divergence for the following time step
- **Phim0** (*array of float*) – Fourier coefficents of geopotential for one time step
- **Phim1** (*array of float*) – Fourier coefficents of geopotential for the following time step
- **I** (*int*) – number of longitudes
- **J** (*int*) – number of Gaussian latitudes
- **N** (*int*) – highest degree of the Legendre functions for m=0
- **Am** (*array of float*) – Fourier coefficients of the nonlinear component A=U*eta
- **Bm** (*array of float*) – Fourier coefficients of the nonlinear component B=V*eta
- **Cm** (*array of float*) – Fourier coefficients of the nonlinear component C=U*Phi
- **Dm** (*array of float*) – Fourier coefficients of the nonlinear component D=V*Phi
- **Em** (*array of float*) – Fourier coefficients of the nonlinear component E=(U^2+V^2)/(2(1-mu^2))
- **Fm** (*array of float*) – Fourier coefficients of the zonal component of wind forcing
- **Gm** (*array of float*) – Fourier coefficients of the meridional component of wind forcing
- **Um** (*array of float*) – Fourier coefficients of the zonal component of wind
- **Vm** (*array of float*) – Fourier coefficients of the meridional component of wind
- **fmn** (*array of float*) – spectral coefficients of the Coriolis force
- **Pmn** (*array of float*) – associated legendre functions evaluated at the Gaussian latitudes mus up to wavenumber M
- **Hmn** (*array of float*) – derivatives of the associated legendre functions evaluated at the Gaussian latitudes mus up to wavenumber M
- **w** (*array of float*) – Gauss Legendre weights
- **tstepcoeff** (*array of float*) – coefficient for time-stepping of the form 2dt/(a(1-mus^2))
- **tstepcoeff2** (*array of float*) – time stepping coefficient of the form 2dt/a^2
- **tstepcoeffmn** (*array of float*) – an array of coefficients a/(n(n+1))
- **marray** (*array of float*) – coefficients equal to m=0,1,...,M in a matrix M+1xN+1
- **mJarray** (*array of float*) – coefficients equal to m=0,1,...,M in a matrix M+1xJ
- **narray** (*array of float*) – array n(n+1) in a matrix M+1xN+1
- **PhiFm** – Fourier coefficients of the geopotential forcing
- **dt** (*float*) – time step, in seconds
- **a** (*float*) – planetary radius, m
- **Phibar** (*float*) – time-invariant spatial mean geopotential, height of the top layer
- **taurad** (*float*) – radiative timescale
- **taudrag** (*float*) – drag timescale

- **forcflag** (*float*) – forcing flag
- **diffflag** (*float*) – hyperdiffusion filter flag
- **sigma** (*array of float*) – hyperdiffusion filter coefficients for absolute vorticity and divergence
- **sigmaPhi** (*array of float*) – hyperdiffusion filter coefficients for geopotential
- **test** (*int*) – number of test, defaults to None
- **t** (*int*) – number of current time step

Returns

- **Phimntstep**
  Updated spectral coefficients of geopotential
- **newPhitstep**
  Updated geopotential

Return type
array of float

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

S

INDEX

T

V

W