**Daedalus Science Study**

**The DaedalusMAZEModeling Environment:**

**User’s Manual**

**Document Information:**

|  |  |
| --- | --- |
| **Name** | **DAED\_MDL\_DaedalusMAZE\_Manual\_2019-09-21\_v01.docx** |
| **Date** | 2019-11-08 |
| **Version** | V04 |
| **Subject** | * DaedalusMAZE - Daedalus modeling environment user’s manual |
| **Resp.** | * DaedalusMAZE administrator ([lfsc.daedalus@gmail.com](mailto:lfsc.daedalus@gmail.com)), cc to kostispapadakis ([kostispapadak@gmail.com](mailto:kostispapadak@gmail.com)) and Theodoros Sarris ([tsarris@ee.duth.gr](mailto:tsarris@ee.duth.gr)) |
| **Location** | <http://evros.ee.duth.gr:8080/user/daedalusgit/tree/DaedalusMaze/>  (if hyperlink does not work, copy/paste the link to your browser) |

**Version History:**

|  |  |  |
| --- | --- | --- |
| **Version** | **Date** | **Description** |
| v01 | 2019-09-21 | * Instructions to obtain account, login, to clone, branch and push project |
| v02 | 2019-09-23 | * Updated with examples of TIEGCM interpolations along orbit |
| v03 | 2019-10-16 | * Updated with examples of calling empirical models along orbit |
| v04 | 2019-11-08 | * Includes descriptions of Daedalus NetCDF files and how to read/write them |

**Table of Contents**

1. Instructions on how to start working on DaedalusMAZE 3

1.1 Signing up and logging on 3

1.1.1 Obtaining a GitHub Account 3

1.1.2 Signing in to DaedalusMAZE 3

1.2 Cloning a project, creating a new branch and pushing a branch 4

1.3 Links for more information on Jupyter and GitHub 5

2. Guidelines for the coding of modules in DaedalusMaze 6

3. Sample execution of empirical model runs along orbit 7

3.1 Available Models and input parameters 7

3.2 HWM14 Model 7

3.3 IGRF 8

3.4 IRI16 8

3.5 NRLMSISE00 8

3.6 Plot Globe function 8

3.7 Other Plotting Examples 10

4. Sample TIEGCM interpolations along orbit 14

4.1 Using the Interpolator Outside of DaedalusMAZE 14

4.2 Interpolation Module Instructions 15

4.3 Examples of runs 16

5. Daealus NetCDF files - User Manual 18

5.1 Daedalus NetCDF file structure and contents 18

5.2 Panoply NetCDF viewer 18

5.3 How to read a Daedalus NetCDF file 18

5.4 How to write a Daedalus NetCDF file 19

5.5 The Variables inside a Daedalus NetCDF file 19

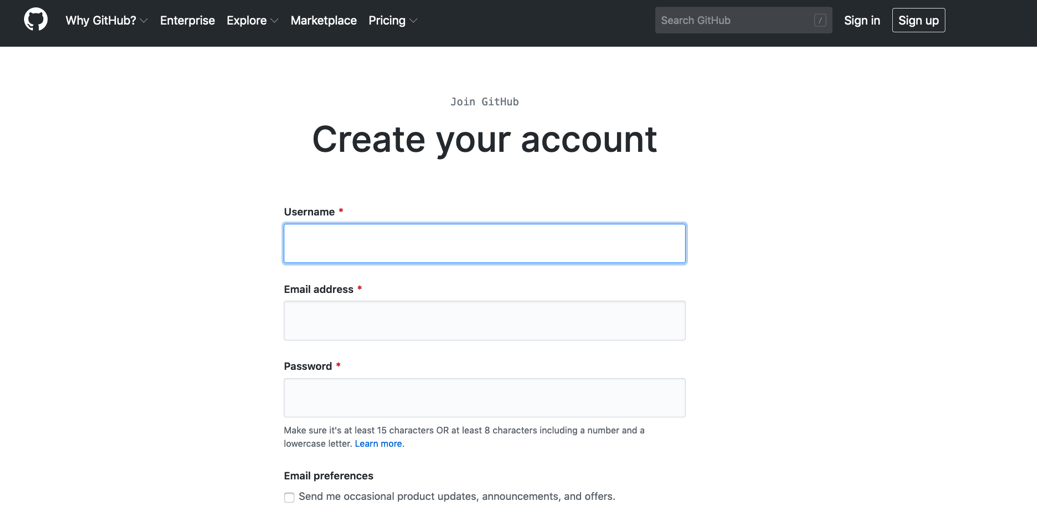
# Instructions on how to start working on DaedalusMAZE

## Signing up and logging on

### Obtaining a GitHub Account

* **Step 1:**DaedalusMAZE uses GitHub for code repository and version control. As a first step, users will need to sign up to obtain a GitHub account, at:

[https://github.com](https://github.com/).



* **Step 2:** In order to add a user as collaborators to the DaedalusMAZE project, e-mail yourGitHub username to[*lfsc.daedalus@gmail.com*](mailto:lfsc.daedalus@gmail.com).(**Note:** Please cc**both**[*kostispapadak@gmail.com*](mailto:kostispapadak@gmail.com)and[*tsarris@ee.duth.gr*](mailto:tsarris@ee.duth.gr))
* **Step 3:** You will be sent an automatic Githubinvitation by e-mail. Please accept it while logged in on Github.

### Signing in to DaedalusMAZE

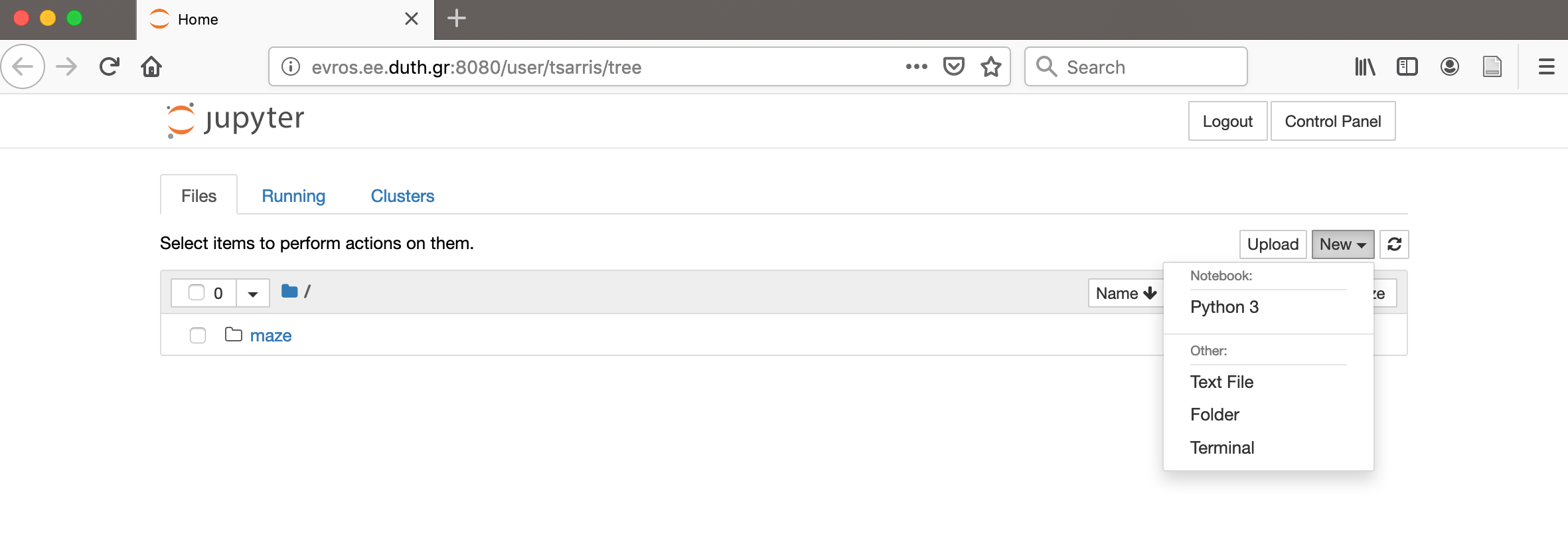
* **Step 4:** You will also be provided with a username and password for accessing DaedalusMAZE from the above email, [lfsc.daedalus@gmail.com](mailto:lfsc.daedalus@gmail.com).
* **Step 5:** You can visit the DaedalusMAZE environment atthe following link:

<http://evros.ee.duth.gr:8080>

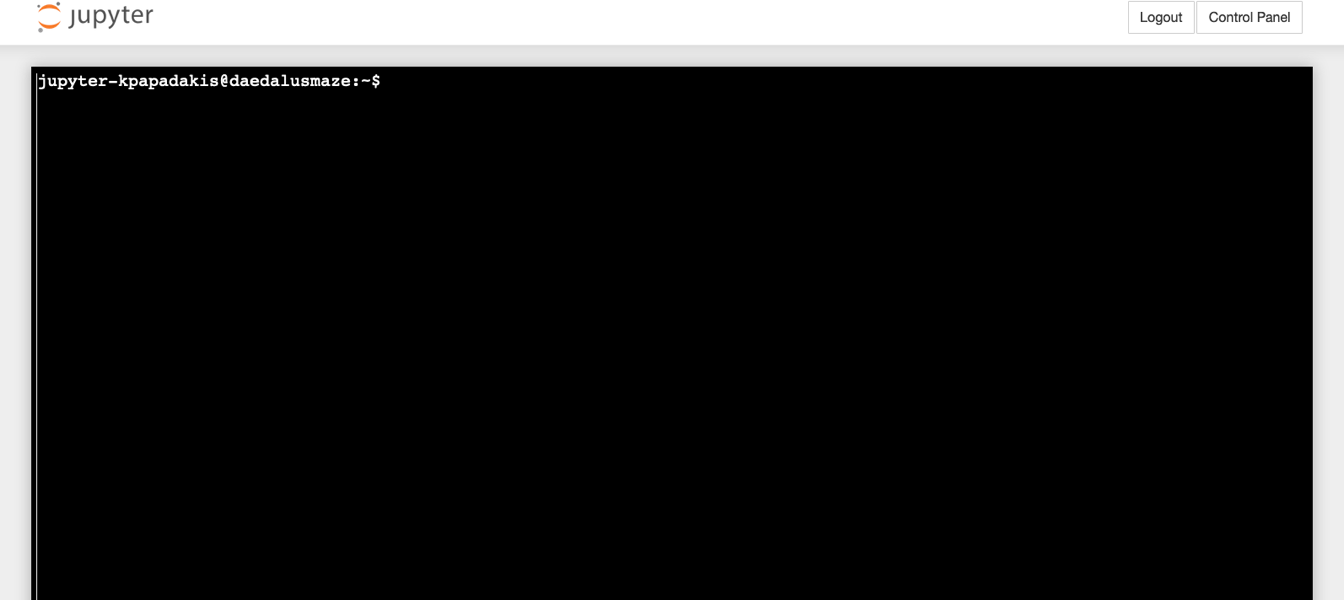
At the prompt, sign in to the DaedalusMAZE environment using the username and password you were provided with in **Step 4**.

## Cloning a project, creating a new branch and pushing a branch

* **Step 6:** Select “New” and then “Terminal”, as shown in the following screenshots:



The following Terminal environment should appear:

­

* **Step 7:** In order to clone DaedalusMAZE,in the jupyter terminal that appears write the following command and press enter:

git clone <https://github.com/DaedalusGIT/DaedalusMAZE.git>

* **Step 8:** You will be asked for an authentication, at which you enter your Github username and password. You have successfully cloned DaedalusMAZE.
* **Step 9:** In order to create your personal branch, write the following command and press enter:

git checkout -b <my-branch-name>

* **Step 10:** In order to push the branch, write the following command and press enter:

git push origin <my-branch-name>

* **Step 11:** In order to check what branch, you are on, write the following command and press enter:

git branch -a

## Links for more information on Jupyter and GitHub

* For more info on Jupyter please refer to: <https://jupyter.org/documentation>
* For help on github commands please refer to the github documentation: <https://github.github.com/training-kit/downloads/github-git-cheat-sheet.pdf>

# Guidelines for the coding of modules in DaedalusMaze

1. A module is regarded as a black box with one or more inputs, one or more outputs and an internal logic, which processes the inputs and produces the outputs.
2. In coding terms the logic corresponds to a function, the inputs to the function’s arguments and the outputs to the function’s return values.
3. For this document we suppose that the module we want to create will be named “Mod1”
4. The source code of Mod1 must reside in a file named:

DaedalusMaze/SourceCode/ModulesSourceCode/Mod1/Mod1.py ,

which must contain a python function named “Mod1”

1. The same folder can hold whatever other files or folders are necessary for the module, as well.
2. After the new module is created, the DaedalusMAZEadministrator at DUTH needs the following information in order to incorporate it into DaedalusMaze:
   1. The module’s name
   2. The names and data-types of the module’s inputs and outputs.
   3. A description of the module’s function, inputs and outputs, so that others can understand how to use it.
   4. The module’s connections with other modules. All these modules will form a group. The group will be available for selection and execution through the GUI.
3. Note: In case the Globe plotting module will be used, then Firefox is recommended.

# Sample execution of empirical model runs along orbit

## Available Models and input parameters

The available models through DaedalusMaze are IRI16, HWM14, MSISE00 and IGRF. All these models are provided through the pyglow library written in python. Pyglow is already installed in DaedalusMaze. If further tests are needed to be run outside the DaedalusMaze environment, all the available information on how to install it are given in the link below.

<https://github.com/timduly4/pyglow>

All the models require four parameters in order to return the values/parameters they calculate. These are time, latitude, longitude and altitude. All these must be in the same .csv file and in the order mentioned above.

Time must be given in this format “**Jan 01 2015 00:00:01.000000**” and if possible with no more than 6 decimal units. If time is given in another format, changes must be made in the code in order for the models to run.

Latitude, longitude and altitude must be given in geodetic coordinates. In figure 1 below, is shown the appropriate content of a .csv file meant to be used for the models.

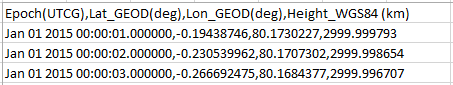


Figure 1

The order in which these input parameters are given in the models (time, latitude, longitude and altitude), must be like figure 1. If otherwise, changes must be made in the source code.

When a NAME.csv file, containing the abovementioned format, is given as input in DaedalusMaze and a model is chosen, DaedalusMaze produces a new file containing all the abovementioned format plus all the parameters calculated through that model.

## HWM14 Model

HWM14 model calculates **u** and **v** which are the zonal wind and meridional wind respectively. Input parameters are time, latitude, longitude and altitude as mentioned above. Do note that HWM14 does not provide any values for altitude below 0 km and 450 km, so do not be confused in some spots of the orbit there are no values for the respective altitudes i.e an orbit’s file with perigee equal to 120km and apogee equal to 3000km.

## IGRF

IGRF model calculates the magnetic field components and the total magnetic field (Bx, By, Bz and B). Its input parameters are the same as HWM14’s model and DaedalusMaze, when running this model produces a new file with the abovementioned format plus B (Total Magnetic field).

## IRI16

IRI16 model when run in DaedalusMaze environment gives the following parameters: NO+ (cm-3), O+ (cm-3), (cm-3), Te (K), Ti (K) and Ne (cm-3). Its input parameters are the same as IGRF’s and DaedalusMaze, when running this model produces a new file with the abovementioned format plus NO+ (cm-3), O+ (cm-3), (cm-3), Te (K), Ti (K) and Ne (cm-3).

## NRLMSISE00

NRLMSISE describes the neutral temperature and densities in Earth's atmosphere from ground to thermospheric heights. It is valid for altitudes ranging from 0 to 1000km. Its input parameters are the same as IGRF’s. DaedalusMaze, when running this model, produces a new file with the abovementioned format plus N2 (cm-3), O2 (cm-3), O (cm-3), TN (K) and rho (g cm-3).

## Plot Globe function

Plot globe function is executed, in DaedalusMaze environment, when the button **execute** is clicked.

Plot globe requires, in order to run, certain files as inputs. These files are the **DataCSVfilename,OrbitDataCSVfilename.** Also the user of DaedalusMaze may “fill” the cells with names:**PlotTitle, ColorbarTitle, ColorscaleName.**

**DataCSVfilename** is used in order to plot a parameter of a model along the spherical surface of the earth. Models are the abovementioned and parameters each parameter produced by each model. In example, if the IGRF model is chosen, only the parameter B may be plotted. **DataCSVfilename**filetypically has values for the whole surface of the surface at a given altitude and one or more parameters. In figure 2 it is shown the cell in which the name of the file is filled.



Figure 2

**OrbitDataCSVfilename**is used in order to plot the values of a parameter along a given orbit. The typical file of the **OrbitDataCSVfilename**includes the geodetic coordinates of the orbit as well as the value or values of the model studied. In figure 3 below, it is shown where the name of the orbit’s file is filled.



Figure 3

It is not mandatory **DataCSVfilename**and**OrbitDataCSVfilename**to study the same parameter.

In **PlotTitle**the end user may “fill” the title of the plot.

In **ColorbarTitle** the end user may “fill” the name of the parameter to be plot as well as its measurement units.

In **ColorscaleName**the end user may “fill” the hue the plot will have. The available values are: ‘Blackbody’, ‘Bluered’, ‘Blues’, ‘Earth’, ‘Electric’, ‘Greens’, ‘Greys’, ‘Hot’, ‘Jet’, ‘Picnic’, ‘Portland’, ‘Rainbow’, ‘RdBu’, ‘Reds’, ‘Viridis’, ‘YlGnBu’, ‘YlOrRd’.

In case nothing is “filled” in the appropriate box, then a default HeatMapcolorscale will be applied. In case None is “filled” then all points will be black irrespective of value.

In figure 4 it is shown the cells where one can fill the appropriate values for PlotTitle, ColorbarTitle and ColorscaleName.

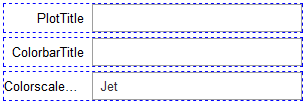


Figure 4

## Other Plotting Examples

**2D LINE PLOT WITH COLORED LINE**

The following code reads a .csv file containing NRLMSISE00 model outputs along a Daedalus orbit and plots the temperature of the neutrals. In the example shown below neutrqal temperature (Tn) is ploted in the y axis; on the x axis the first line of labels shows time along the orbit in mins and the second label shows the geodetic latitude that corresponds to the given time. Furthermore, there is a colorbar at the left showing the different altitude of the Daedalus orbit and this is also shown on the curve of the temperature, meaning that information on temperature is given with respect to time, latitude and altitude.

**CODE**

importnumpy as np

importmatplotlib.pyplot as plt

import pandas as pd

frommatplotlib.collections import LineCollection

degree\_sign= u'\N{DEGREE SIGN}'

#read file to plot

file\_to\_read=pd.read\_csv("Deadalus Orbits/MSISE00\_Output\_DAED\_ORB\_Evt0\_LLA\_Per120\_Lat00\_Srt01Hz\_Msc.csv")

#read time and simply convert it in seconds in the form 1,2,3,....5000 etc

x1\_axis\_value\_from\_file = file\_to\_read["Epoch(UTCG)"]

pointer = len(x1\_axis\_value\_from\_file)

x1\_axis\_value\_from\_file\_array = []

fori in range(0,pointer):

x1\_axis\_value\_from\_file\_array.append(i+1)

#read the value to be plotted

y\_axis\_value\_from\_file = file\_to\_read ["Tn\_msise00\_K"]

#read the values for the 2nd x axis to be shown

x2\_axis\_value\_from\_file = file\_to\_read["Lat\_GEOD(deg)"]

x2\_axis\_value\_from\_file\_array = np.array(x2\_axis\_value\_from\_file)

#read the values that will color the line. In this example it is height

color\_value\_from\_file = file\_to\_read ["Height\_WGS84 (km)"]

color\_value\_from\_file\_array = np.array(color\_value\_from\_file,dtype=float)

X = np.array(x1\_axis\_value\_from\_file\_array,dtype=float)

Y = np.array(y\_axis\_value\_from\_file,dtype=float)

# Create a set of line segments so that we can color them individually

# This creates the points as a N x 1 x 2 array so that we can stack points

# together easily to get the segments. The segments array for line collection

# needs to be (numlines) x (points per line) x 2 (for x and y)

# its important for points and segments to work, that all arrays are float (at

# the moment, that's how its working)

points = np.array([X, Y]).T.reshape(-1, 1, 2)

segments = np.concatenate([points[:-1], points[1:]], axis=1)

fig = plt.figure()

#necessary for the second x axis

ax1 = fig.add\_subplot(111)

ax2 = ax1.twiny()

ax1.grid(True)

# Add some extra space for the second axis at the bottom

fig.subplots\_adjust(bottom=0.2)

#plotting data

#1st set the boundaries of the plot

ax1.set\_xlim(X[0], X[7079])

upperYlim = Y.max() + 100

ax1.set\_ylim(0, upperYlim) #since temp is in K, 0 is the bottom boundary

# Create a continuous norm to map from data points to colors

norm = plt.Normalize(color\_value\_from\_file\_array.min(),color\_value\_from\_file\_array.max())

lc = LineCollection(segments, cmap='plasma', norm=norm)

# Set the values used for colormapping

lc.set\_array(color\_value\_from\_file\_array)

lc.set\_linewidth(2)

line = ax1.add\_collection(lc)

fig.colorbar(line,ax=ax1, label='Height\_WGS84 (km)')

array\_length = len(x2\_axis\_value\_from\_file\_array)

ax1.set\_xlabel("Time in mins")

ax1.set\_ylabel("Temperature of neutrals in "+degree\_sign+"K")

#converting seconds to mins and getting the new values for ticks

first\_tick\_1st\_axis = int ((x1\_axis\_value\_from\_file\_array[0])/60)

second\_tick\_1st\_axis = int ((x1\_axis\_value\_from\_file\_array[int(array\_length\*0.125)])/60)

third\_tick\_1st\_axis = int ((x1\_axis\_value\_from\_file\_array[int(array\_length\*0.25)])/60)

fourth\_tick\_1st\_axis = int ((x1\_axis\_value\_from\_file\_array[int(array\_length\*0.375)])/60)

fifth\_tick\_1st\_axis = int ((x1\_axis\_value\_from\_file\_array[int(array\_length\*0.5)])/60)

sixth\_tick\_1st\_axis = int ((x1\_axis\_value\_from\_file\_array[int(array\_length\*0.625)])/60)

seventh\_tick\_1st\_axis = int ((x1\_axis\_value\_from\_file\_array[int(array\_length\*0.75)])/60)

eigth\_tick\_1st\_axis = int ((x1\_axis\_value\_from\_file\_array[int(array\_length\*0.875)])/60)

ninth\_tick\_1st\_axis = int ((x1\_axis\_value\_from\_file\_array[int(array\_length-1)])/60)

x1axis\_tick\_labels = (first\_tick\_1st\_axis, second\_tick\_1st\_axis, third\_tick\_1st\_axis, fourth\_tick\_1st\_axis, fifth\_tick\_1st\_axis, sixth\_tick\_1st\_axis, seventh\_tick\_1st\_axis, eigth\_tick\_1st\_axis, ninth\_tick\_1st\_axis)

ax1.set\_xticklabels(x1axis\_tick\_labels)

ax1.set\_xlim(xmin=0,xmax=array\_length)

#creating the number of ticks in the first x-axis in the graph. This can be set accordingly

ax1.set\_xticks([0,array\_length\*0.125,array\_length\*0.25,array\_length\*0.375,array\_length\*0.5,array\_length\*0.625,array\_length\*0.75,array\_length\*0.875,array\_length-1])

#getting the new ticks,assuming that the first diagram has 8 ticks including

#0 in the beginning of the axes

first\_tick\_2nd\_axis = "["+str("{:.1f}".format(x2\_axis\_value\_from\_file\_array[0]))+"]"

second\_tick\_2nd\_axis = "[" + str("{:.1f}".format(x2\_axis\_value\_from\_file\_array[int(array\_length\*0.125)])) + degree\_sign+"]"

third\_tick\_2nd\_axis = "[" + str("{:.1f}".format(x2\_axis\_value\_from\_file\_array[int(array\_length\*0.25)]))+degree\_sign + "]"

fourth\_tick\_2nd\_axis = "[" + str("{:.1f}".format(x2\_axis\_value\_from\_file\_array[int(array\_length\*0.375)]))+degree\_sign + "]"

fifth\_tick\_2nd\_axis = "[" + str("{:.1f}".format(x2\_axis\_value\_from\_file\_array[int(array\_length\*0.5)]))+degree\_sign + "]"

sixth\_tick\_2nd\_axis = "[" + str("{:.1f}".format(x2\_axis\_value\_from\_file\_array[int(array\_length\*0.625)]))+degree\_sign + "]"

seventh\_tick\_2nd\_axis = "[" + str("{:.1f}".format(x2\_axis\_value\_from\_file\_array[int(array\_length\*0.75)])) + degree\_sign + "]"

eigth\_tick\_2nd\_axis = "[" +str("{:.1f}".format(x2\_axis\_value\_from\_file\_array[int(array\_length\*0.875)])) + degree\_sign + "]"

ninth\_tick\_2nd\_axis = "[" + str("{:.1f}".format(x2\_axis\_value\_from\_file\_array[int(array\_length-1)])) + degree\_sign + "]"

NoTicks = 9

new\_tick\_labels = (first\_tick\_2nd\_axis, second\_tick\_2nd\_axis, third\_tick\_2nd\_axis, fourth\_tick\_2nd\_axis, fifth\_tick\_2nd\_axis, sixth\_tick\_2nd\_axis, seventh\_tick\_2nd\_axis, eigth\_tick\_2nd\_axis, ninth\_tick\_2nd\_axis)

# Move twinned axis ticks and label from top to bottom

ax2.xaxis.set\_ticks\_position("bottom")

ax2.xaxis.set\_label\_position("bottom")

# Offset the twin axis below the host

ax2.spines["bottom"].set\_position(("axes", -0.15))

# Turn on the frame for the twin axis, but then hide all

# but the bottom spine

ax2.set\_frame\_on(True)

ax2.patch.set\_visible(False)

#creating the number of ticks in the first x-axis in the graph

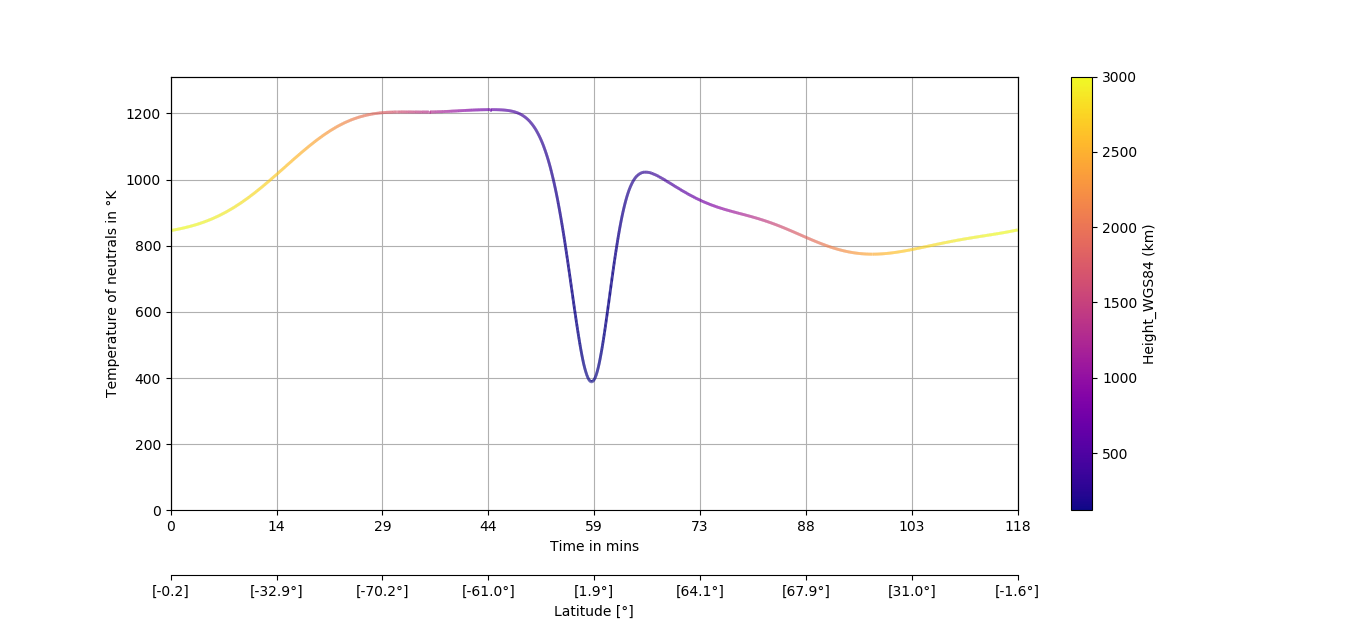
ax2.set\_xticks([0,1,2,3,4,5,6,7,8])

ax2.set\_xticklabels(new\_tick\_labels)

ax2.set\_xlabel("Latitude ["+degree\_sign+"]")

plt.show()

By running this code the result is the following figure:



**2D LINE PLOT & SURFACE PLOT WITH COLORED LINE**

The following code reads two “name.csv” files: The first one contains a Daedalus orbit and the parameters from the HWM14 model along this orbit. The second contains a “slice” of the earth’s atmosphere at 150km and the parameters from the HWM14 model. The code plots the slice with a scatter plot and a colorbar on the left of the plot. It also plots Daedalus orbit, and along its orbit produces different colors according to the value of the parameter measured.

**CODE**

importnumpy as np

importmatplotlib.pyplot as plt

import pandas as pd

frommatplotlib.collections import LineCollection

#read file to plot

file\_to\_read = pd.read\_csv("Deadalus Orbits/HWM14\_Output\_DAED\_ORB\_Evt0\_LLA\_Per120\_Lat00\_Srt01Hz\_Msc.csv")

file\_to\_read\_2 = pd.read\_csv("Deadalus Orbits/HWM14\_Output\_earth\_sample\_150km\_05deg.csv")

y\_orbit\_file = file\_to\_read["Lat\_GEOD(deg)"]

x\_orbit\_file = file\_to\_read["Lon\_GEOD(deg)"]

z\_orbit\_file = file\_to\_read["Height\_WGS84 (km)"]

orbit\_param\_file = file\_to\_read["u\_hwm14\_m/s"]

#before plotting the orbit, check if the csv file has values in longitude from 0-360 degrees

#and convert these values to -180 degrees till 180. If there is no need for conversion, comment

#the lines below till the next comments and change input in plot command from y\_orbit\_array to y\_orbit

x\_orbit\_array\_file = np.array(x\_orbit\_file,dtype=float)

z\_orbit\_array\_file = np.array(z\_orbit\_file)

fori in range(0,len(x\_orbit\_array\_file)):

x\_orbit\_array\_file[i]=x\_orbit\_array\_file[i]-180

#make the treaty for the markers

index\_array = []

fori in range (0,len(y\_orbit\_file)):

ifz\_orbit\_file[i]<150:

index\_array.append(i)

last\_point\_index\_array = len(index\_array)-1

#create new arrays for the colored line from the orbit file.

#create half arrays for the plot of the orbit.

new\_y\_orbit = []

new\_x\_orbit = []

new\_orbit\_param = []

x\_orbit\_array = []

y\_orbit = []

index\_orbit\_array = []

fori in range (0,len(y\_orbit\_file)):

ifz\_orbit\_array\_file[i] <= 450:

new\_y\_orbit.append(y\_orbit\_file[i])

new\_x\_orbit.append(x\_orbit\_array\_file[i])

new\_orbit\_param.append(orbit\_param\_file[i])

index\_orbit\_array.append(i)

elifz\_orbit\_array\_file[i] > 450:

x\_orbit\_array.append(x\_orbit\_array\_file[i])

y\_orbit.append(y\_orbit\_file[i])

pointer\_orbit = len(x\_orbit\_array) # define a pointer to "cut" the orbit in two files

x\_orbit\_array\_1st\_half = []

y\_orbit\_1st\_half = []

x\_orbit\_array\_2nd\_half = []

y\_orbit\_2nd\_half = []

fori in range(0,index\_orbit\_array[0]):

x\_orbit\_array\_1st\_half.append(x\_orbit\_array\_file[i])

y\_orbit\_1st\_half.append(y\_orbit\_file[i])

zstart=index\_orbit\_array[len(index\_orbit\_array)-1]

zend=len(y\_orbit\_file)-1

for k in range(zstart,zend):

x\_orbit\_array\_2nd\_half.append(x\_orbit\_array\_file[k])

y\_orbit\_2nd\_half.append(y\_orbit\_file[k])

color\_value\_from\_file\_array = np.array(new\_orbit\_param,dtype=float)

X = np.array(new\_x\_orbit,dtype=float)

Y = np.array(new\_y\_orbit,dtype=float)

# Create a set of line segments so that we can color them individually

# This creates the points as a N x 1 x 2 array so that we can stack points

# together easily to get the segments. The segments array for line collection

# needs to be (numlines) x (points per line) x 2 (for x and y)

# its important for points and segments to work, that all arrays are float (at

# the moment, that's how its working)

points = np.array([X, Y]).T.reshape(-1, 1, 2)

segments = np.concatenate([points[:-1], points[1:]], axis=1)

fig=plt.figure()

ax1 = fig.add\_subplot(111)

ax1.grid()

# Create a continuous norm to map from data points to colors

norm = plt.Normalize(color\_value\_from\_file\_array.min(),color\_value\_from\_file\_array.max())

lc = LineCollection(segments, cmap='rainbow', norm=norm)

# Set the values used for colormapping

lc.set\_array(color\_value\_from\_file\_array)

lc.set\_linewidth(2)

line = ax1.add\_collection(lc)

fig.colorbar(line,ax=ax1, label='U-Meridional winds across orbit (m/s)')

#plotting commands

plt.rcParams['figure.figsize'] = (10, 10)

plt.xlim(-181, 181) #limits are set from -181 to 181 to have a better view in the plot

plt.ylim(-91, 91) #limits are set from -91 to 91 to have a better view in the plot

sc=plt.scatter(file\_to\_read\_2 ["Lon (deg)"],file\_to\_read\_2 ["Lat (deg)"], c=file\_to\_read\_2 ["u\_hwm14\_m/s"], cmap='rainbow')

plt.title("HWM14 Model u-meridional winds at 150km \n Daedalus Orbit with Lat00 and Per120 \n U-meridional winds across Daedalus orbit")

plt.xlabel("Longitude(deg) \n black dots show the intersection of Daedalus orbit at 150km")

plt.ylabel("Latitude(deg) \n black dots show the intersection of Daedalus orbit at 150km ")

plt.colorbar(sc,label='U-Meridional winds at 150km (m/s)')

plt.plot(x\_orbit\_array\_1st\_half,y\_orbit\_1st\_half,color='black')

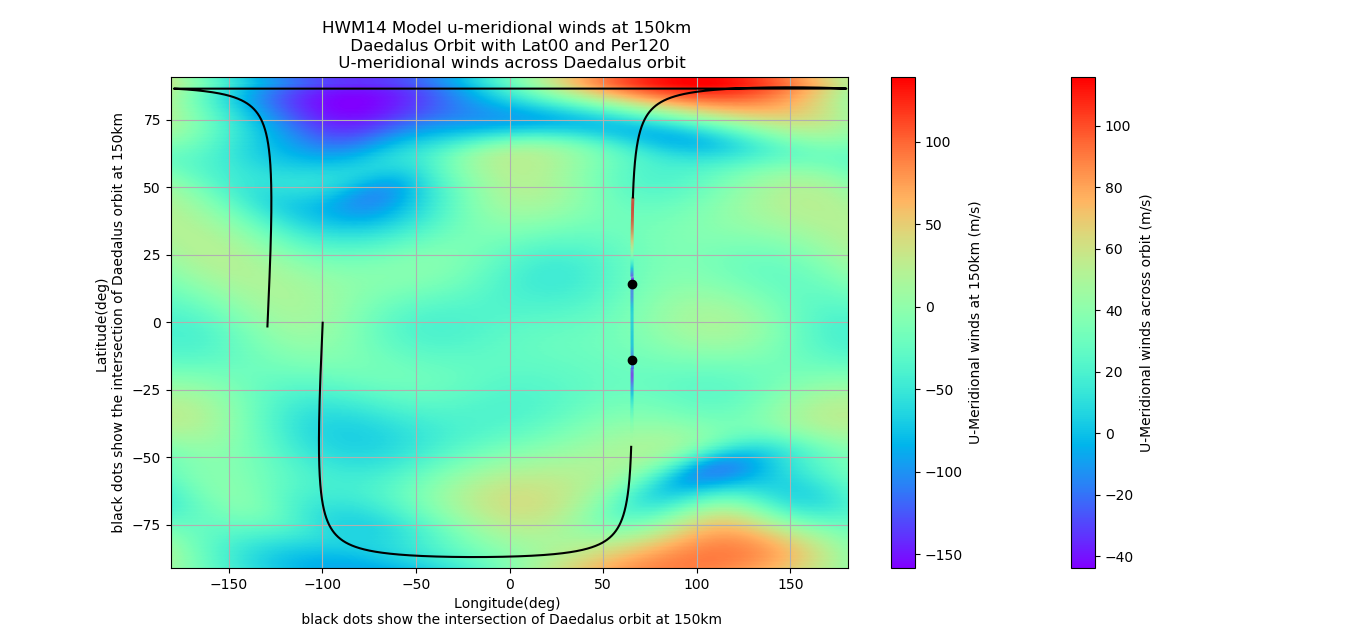
plt.plot(x\_orbit\_array\_2nd\_half,y\_orbit\_2nd\_half,color='black')

plt.plot(x\_orbit\_array\_file[index\_array[0]],y\_orbit\_file[index\_array[0]],marker="o",color='black')

plt.plot(x\_orbit\_array\_file[index\_array[last\_point\_index\_array]],y\_orbit\_file[index\_array[last\_point\_index\_array]],marker="o",color='black')

plt.show()

By running this code the result is the following image. The two dots show the locations where Daedalus’ orbit crosses the altitude where the surface plot is drawn. The colored part of the orbit shows the region where Daedalus is located below 500 km.



# Sample TIEGCM interpolations along orbit

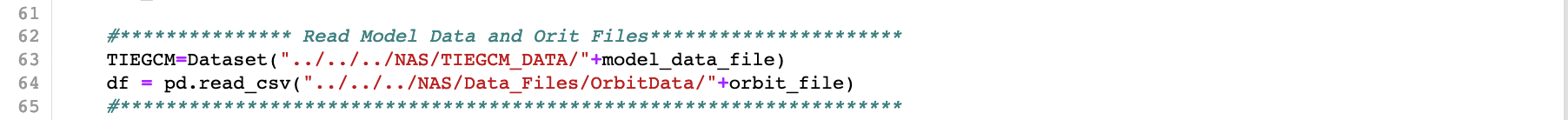
## Using the Interpolator Outside of DaedalusMAZE

To use the Interpolator as a standalone module the following steps are to be followed:

1. Copy DaedalusInterpolator.py and Interpolation\_P2P.f90 to your preferred directory:
2. Open a new terminal window and visit your current path. Write the following command to compile the FORTRAN subroutine and hit enter. You can choose O1, O2, O3 optimization flags as preferred. By default, O2 is used.

f2py3 -c Interolation\_P2P.f95 -lgomp --opt='-O2' -m interpolation

1. Adjust the paths shown below, in DaedalusInterpolator.py according to your current directory. These paths are valid for the directory /DaedalusMAZE





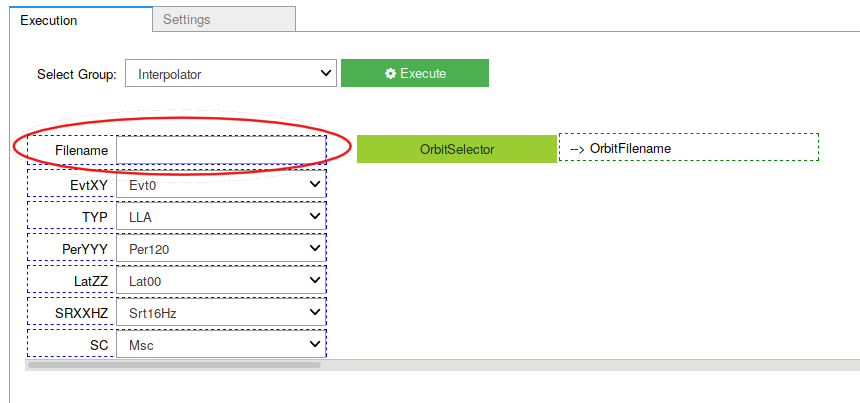
1. Finally, to import the library use the following in your code:

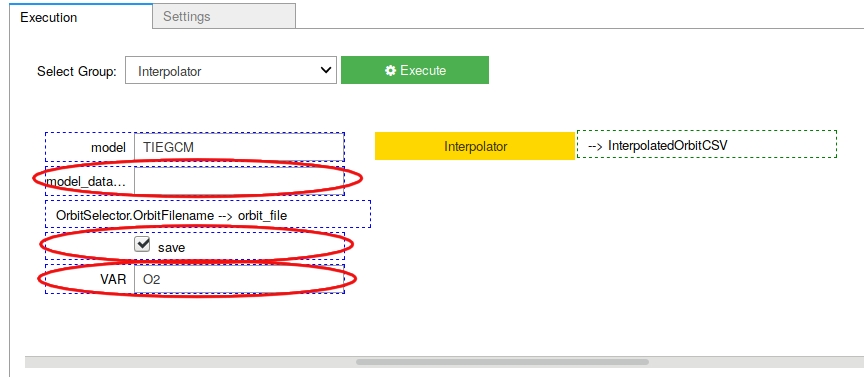
fromDaedalusInterpolator import \*

1. The Example Call Shown in Mod\_Interpolator/Interpolator.ipynbcan be now used in your code to call the Interpolator. After the code finishes a plot similar to the one in**4.1** will be displayed in the output window. The interpolated values will be saved in DaedalusNAS.

## 4.2 Interpolation Module Instructions

The interpolation module uses a trilinear interpolation scheme to interpolate scalars such as Electron Denisty and Oxygen Density, to Daedalus positions along orbit. The user is asked to pick the orbit filename, the TIEGCM set of outputs based on event time and the variable to interpolate. The process is shown visually in the screenshots below. The files can be browsed either through the bash terminal on DaedalusMAZE or using the DaedalusNAS web app at[evros.ee.duth.gr:5000/](evros.ee.duth.gr:5000/volume1/Data_Files/ModelsOutput/Interpolation)



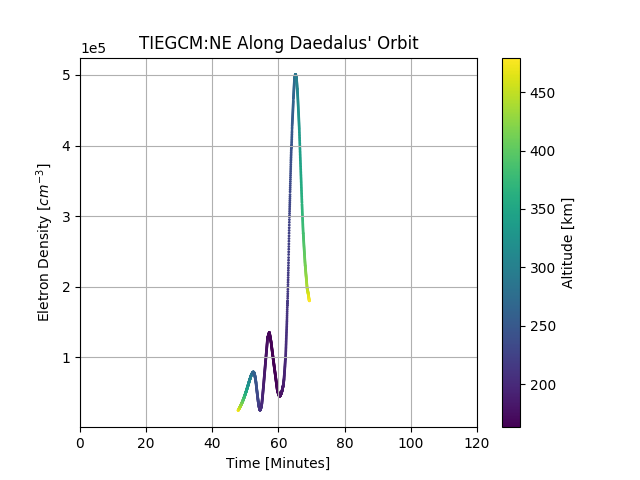


## 4.3Examples of runs

Below are depicted the Electron Density and Oxygen Density as interpolated along track for our event 0 case.

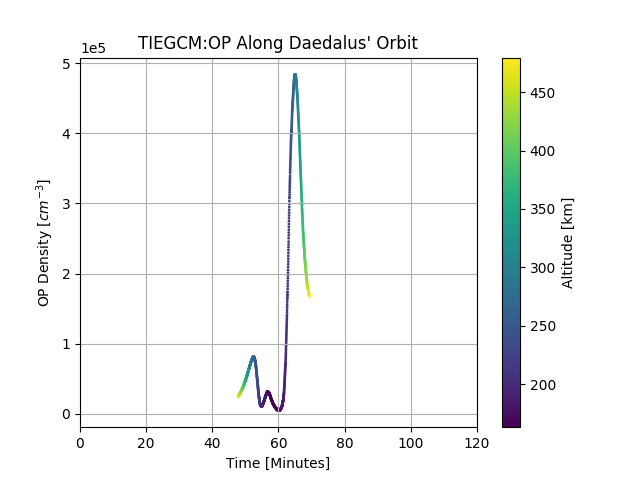
**Example Run1:**

|  |  |
| --- | --- |
| Filename Used | DAED\_ORB\_Evt0\_LLA\_Per150\_Lat80\_Srt01Hz\_Msc.csv |
| Model Data Used | tiegcm\_s\_214900.nc |
| Variable Used | NE |



**Example Run2:**

|  |  |
| --- | --- |
| Filename Used | DAED\_ORB\_Evt0\_LLA\_Per150\_Lat80\_Srt01Hz\_Msc.csv |
| Model Data Used | tiegcm\_s\_214900.nc |
| Variable Used | OP |



After the interpolation finishes, the interpolated data are stored on DaedalusNAS at ([evros.ee.duth.gr:5000/](evros.ee.duth.gr:5000/volume1/Data_Files/ModelsOutput/Interpolation))

# DaealusNetCDF files - User Manual

## Daedalus NetCDF file structure and contents

Daedalus NetCDF4 files reside at /home/NAS/Data\_Files/OrbitData/. Their names follow the rules described at chapter 6.1, their extension is .nc and they hold data which describe a certain orbit of the satellite each.

The NetCDF file has 4 unlimited dimensions: time, lat, lon, level and several variables in conjunction with time, a list of which can be found at Appendix A.

Plotting the “ALFA” variable displays the orbit upon the Earth map. This variable contains random data for this purpose of previewing the orbit.

There are also a few global attributes:

Type = "orbit" # it can also be “surface”

CreationTime = "2019-11-08 051430.041468" # for 8 Nov 2019 05:14 UTC

## Panoply NetCDF viewer

Panoply application can open, preview and plot NetCDF files. It requires Java Run-time Enviroment 8 or later installed. It can be downloaded from:

<https://www.giss.nasa.gov/tools/panoply/download/>

Panoply can display all variable names available inside the NetCDF file.

For this document, “TEC” is used as an example for a variable name.

## How to read a Daedalus NetCDF file

**Source Code**

from netCDF4 import Dataset

CDFroot = Dataset( “aFileName.nc”, “r” ) # open file for read

Times = CDFroot.variables["time"][:]

Lats = CDFroot.variables["lat"][:]

Lons = CDFroot.variables["lon"][:]

Elevations = CDFroot.variables["level"][:]

TECs = CDFroot.variables["TEC"][:] # for Total Electron Content

print( CDFroot.CreationTime ) # read an attribute

CDFroot.close()

**Surface files**

NetCDF files which describe a surface (not an orbit) contain a bit altered values at Latitude and Longitude variables, so that Panoply can plot them:

|  |  |  |  |
| --- | --- | --- | --- |
|  | Lat Lon |  | Lat Lon |
| Instead of | 87.5 0  87.5 5  87.5 10  ... | The values are | 87.50000001 0.0000001  87.500000025.0000002  87.5000000310.0000003  ... |

In case you want to remove those decimal digits you can apply a rounding function before using the values. It can be used indiscriminately to both orbit and surface files:

Lats = CDFroot.variables["lat"][:]

fori in range(0, len(Lats)):

Lats[i] = round( Lats[i], 2 )

**Python Tip**

If you need a 2D array with columns Lat,Lon,Elevation,Variable, you can type:

AllData = np.vstack( (CDFroot.variables["lat"][:],  
CDFroot.variables["lon"][:],   
CDFroot.variables["level"][:],   
CDFroot.variables[VariableName][:] ) ).T

## How to write a Daedalus NetCDF file

Consider copying the file to a specific folder before you edit it. Please consult the Daedalus team for a relative decision.

**Source Code**

from netCDF4 import Dataset

# open file for append - **do not use “w”,** it erases the whole file

CDFroot = Dataset( “aFileName.nc”, “a” )

CDFroot.variables["lat"][0] = 10

CDFroot.variables["lon"][0] = 20

CDFroot.variables["level"][0] = 3000

CDFroot.variables["TEC"][0] = 3.45

CDFroot.CreationTime = str(datetime.now()) # change an attribute

CDFroot.close()

## The Variables inside a Daedalus NetCDF file

ulong time(time)

doublelat(time)

doublelon(time)

double level(time)

double ALFA(time)

description = "Aurora Characteristic Energy"

units = "keV"

double BARM(time)

description = "Mean Molecular Weight"

units = "g/mol"

double DEN(time)

description = "Total Density"

units = "g/cm^3"

double EFLUX(time)

description = "Aurora Energy Flux"

units = "erg/cm^2/s"

double HE(time)

description = "Helium"

units = "mmr"

double LAMDA\_HAL(time)

description = "Hall Ion Drag Coefficient"

units = "1/s"

double LAMDA\_PED(time)

description = "Pedersen Ion Drag Coefficient"

units = "1/s"

double N4S(time)

description = ""

units = "mmr"

double NE(time)

description = "Electron Density"

units = "cm^-3"

double NO(time)

description = "Nitric Oxide"

units = "mmr"

double O1(time)

description = "Atomic Oxygen"

units = "mmr"

double O2(time)

description = "Molecular Oxygen"

units = "mmr"

double OP(time)

description = "O+ ION Density"

units = "cm^-3"

double PEDERSEN(time)

description = ""

units = ""

double POTEN(time)

description = "Electric Potential"

units = "Volts"

double QAMIE(time)

description = ""

units = ""

double QAURORA(time)

description = ""

units = "cm^-3 s^-1"

double QWIND(time)

description = ""

units = ""

doublesdtide(time)

description = "amplitudes and phases of semidiurnal tide"

units = ""

double TBAR(time)

description = ""

units = ""

double Te(time)

description = "Electron Temperature"

units = "Kelvin"

double Ti(time)

description = "Ion Temperature"

units = "Kelvin"

double TEC(time)

description = "Total Electron Content"

units = "1/cm^2"

double TN(time)

description = "Neutral Temperature"

units = "Kelvin"

doubleUI\_ExB(time)

description = "Zonal ExB Velocity"

units = "cm/s"

doubleVI\_ExB(time)

description = " Meridional ExB Velocity"

units = "cm/s"

double VN(time)

description = "Neutral Meridional Wind (+North)"

units = "cm/s"

double UN(time)

description = "Neutral Zonal Wind (+East)"

units = "cm/s"

doubleWI\_ExB(time)

description = "Vertical ExB Velocity"

units = "cm/s"

double WN(time)

description = "Neutral Vertical Wind (plus up) "

units = "cm/s"

double Z(time)

description = "Geopotential Height"

units = "cm"

double ZG(time)

description = "Geometric Height ZG"

units = "cm"

double ZMAG(time)

description = "ZMAG from pdynamo"

units = "km"

double ZGMID(time)

description = "Geometric Height at midpoints"

# DaealusNetCDF files – Names and Locations

## Orbit Filenames and Location

# Orbit data is stored in a fixed length filename. The name is made up of characteristics of the specific orbit. Its general format is:

DAED\_ORB\_EVT**XY**\_**TYP**\_PER**YYY**\_LAT**ZZ**\_SRT**QQ**Hz\_**X**SC

The various parameters in bold are described in the following table:

|  |  |  |
| --- | --- | --- |
|  | **Value** | **Description** |
| **EvtXY** | EVTX**S** | X Event, **S**ingle Orbit |
| EVTX**A** | X Event, **A**ll Orbit |
| EVT**1**Y | **1**st Event: St Patrick’s day event [17 Mar 2015 – 20 Mar 2015] |
| EVT**2**Y | **2**nd Event |
| EVT**3**Y | **3**rd Event |
| **….** | … |
| **TYP** | **LLA** | Time,**L**atitude **L**ongitude **A**ltitude |
| **VEL** | Time,**V**magVxVyVz |
|  | **PTG** | Time, X\_GSE, Y\_GSE, Z\_GSE, RamX\_GSE, RamY\_GSE, RamZ\_GSE |
| **PerYYY** | PER**120** | Perigee Altitude at **120**km |
| PER**150** | Perigee Altitude at **150**km |
| **LatZZ** | LAT**00** | Perigee Latitude at **0**° |
| LAT**40** | Perigee Latitude at **40**° |
| LAT**80** | Perigee Latitude at **80**° |
| **SRXXHZ** | SRT**16**Hz | Sampling rate **16**Hz |
| SRT**01**Hz | Sampling rate **1**Hz |
| **SC** | **MSC** | **M**other **S**pace**c**raft |
| **SSC** | **S**ub-**S**pace**c**raft |

All Daedalus orbit files are available in CSV (.csv extension) and NetCDF4 format (.nc extension). The project mainly uses the NetCDF format. All orbit files reside in folder:

/home/NAS/Data\_Files/OrbitData/

This folder can be accessed using the following ways:

1. Through the Network Attached Storage environment at <http://evros.ee.duth.gr:5000/> and its application “File Station”.
2. From within the DaedalusMaze source code, with a reference to the folder path.  
   Note: It is advisable to import the file “SourceCode/DaedalusGlobals.py”, which contain all useful path names. You can type the following command at the top of your python code:  
    import SourceCode.DaedalusGlobals as DaedalusGlobals

## Data files – Naming and Locations

The execution of the End-to-End simulation produces several data files. The simulation starts from an orbit file and proceeds with subsequent calculations separated in stages. Each stage produces its respective data file:

* Stage A, named “Interpolation”, produces   
  /home/NAS/Data\_Files/ModelOutputs/A\_Interpolated/<orbit-name>\_Interpolated.nc
* Stage B, named “Sub-Grid Variability”, produces   
  /home/NAS/Data\_Files/ModelOutputs/B\_SyntheticTruth/<orbit-name>\_SyntheticTruth.nc
* Stage C, named “Environment-to-Instrument Transfer”, produces   
  /home/NAS/Data\_Files/ModelOutputs/C\_Stimuli/<orbit-name>\_ Stimuli.nc
* Stage D, named “Instrument Outputs”, produces   
  /home/NAS/Data\_Files/ModelOutputs/D\_SyntheticScience/<orbit-name>\_ SyntheticScience.nc

These files can be accessed in the same ways as the orbit files in the previous chapter 6.1.

The events which take place during the creation of the above files are:

1. The user selects an orbit to work with.
2. Stage A copies the orbit file into its subfolder, with the above stated filename.
3. Stage A executes calculations and stores the values into the freshly copied file.
4. Stage B again copies the file produced at Stage A into its own subfolder and the process is continued until the final stage.