Chapter 2
Visualization

Create your own visual style... let it be unique for yourself and yet identifiable for others.

—Orson Welles

Abstract Examples are introduced here in which Python can be used to visualize Scientific data in 2D and 3D. Data can originate from both observation and numerical models. Examples in this chapter focus more on observational data. The most used visualization library in Python, *MatPlotLib*, is introduced from the historical point of view. Datasets are downloaded from the web, the length of the day through several years. A mathematical 3D object called *Moebius stream* is created and visualized with several techniques. Python is mostly used as a wrapper for general visualization software such as *Paraview* and *VisIt*. More in general there exists a *Scientific Python Ecosystem*, embedding tools such as *iPython* (now *Jupyter*), MatPlotLib, and many more. This is a set of tools that allows practicing, visualizing, and sharing technologies applied to Science.

When Python was initially created, the main software used in the academia and many private data analysis/numerical modeling companies was MATLAB. For this reason, it became natural to the developers who worked on a visualization library that would work with Python to create a library with an environment that was as similar as possible to MATLAB. In this way, scientists would have found natural to transition to the open software.

As in many other Numerical Python projects, the initiator was a scientist. In this case the Neuroscientist John D. Hunter, who did not find an open-source option for visualizing his data, the Electrocorticography of patients with epilepsy, contacted Fernando Perez, the creator of iPython, in order to add a specific patch to iPython. As Perez himself recalled, however, Perez was too busy writing his Ph.D. dissertation, which brought John D. Hunter to just create a new library from scratch [107]. This library was called *MatplotLib*, and remained his main project until his sudden death in 2012. Since then a team of developers continues to expand the library.

Although *MatplotLib* can plot some features in 3D, most of its power lies on its 2D capabilities. Practically every datasets that can be visualized in 2D can be efficiently

G. Morra, *Pythonic Geodynamics*, Lecture Notes in Earth System Sciences,
DOI 10.1007/978-3-319-55682-6_2

and beautifully visualized with Matplotlib. For example, almost all the figures of the now historical paper that announced the discovery of the Gravitational Waves have been done with Python and MatplotLib [111], and more in general Numerical Python is used daily by the LIGO and VIRGO projects, in control room, Signal Processing, parameter estimation, etc.

In 3D, instead, only few Python-based softwares exist, and Python can be employed more proficiently as a wrapper to collect data, organize them, and finally interface with the major existing visualization software. I will cover here two of the main open source software available: Paraview and VisIt.

Visualization is not the most difficult problem to solve in Geodynamics, certainly not as much as the parallel programming of multi-scale nonlinear Stokes of Darcy flow equation, however without an efficient and effective visualization framework it is impossible to quickly understand the outcome of the development efforts themselves. Ultimately, it is essential for every modeler to have a clear understanding of what the options available in visualization and how to quickly implement them.

From this point of view, visualization also enters into the more general problem of scientific reproducibility. Two projects, heavily based on MatPlotLib, that address this are *iPython* and its newer version, *Jupyter*. Here Visualization is intended in its most general form, as representation of a scientific result. To use iPython Notebooks or Jupyter Notebooks to reproduce a theoretical or numerical work allows (i) giving all the details on its reproducibility and (ii) checking its validity. Many years ago a scientific result was validated by its reproducibility by the calculation on some pieces of paper. With the advent of computing many numerical and data-based results cannot be manually reproduced anymore. Following many before me, I strongly advocate for the adoption of a general framework of openness in data and procedures behind every scientific result, and I believe that the Python powered scientific environment will play a key role in the next future of this revolution.

2.1 The MatPlotLib Visualization Library

Psychologists have proven that more than 50% of our attention is captured by our visual input. This implies that the development of the superb, flexible, and stable MatPlotLib library has played an essential role in the development of the Scientific Python project. *MatplotLib* is today the most commonly used visualization library in Python. It was started by John D Hunter [72], who has built most of it by himself, until he passed away in 2013, and left an enourmous imprint to the Python community of developers.

There are fundamentally two ways to call the *MatplotLib* functions: (1) the *PyLab* interface that facilitates the transition of experienced MATLAB users into *MatplotLib*, whose development was the initial goal of John Hunter, and (2) a python interface that can be used in a much more general framework. The quickest and smartest way to make a new 2D or 3D plot is to look at the vast gallery of graphs available at `http://matplotlib.org/gallery.html`, each with the

associated Python scripts. In most cases one can find a figure close to the style in which one wants to show data or results and use it as a template. Let us look now at some easy examples.

2.1.1 Plotting a 2D Field

The best way to learn how *MatplotLib* works is by playing with the examples available in the online *gallery*. Let us, for example, assume that we know a function z of x and y: $z = f(x, y)$. In this case, we assume that $f(x, y)$ is the sum of two Gaussian functions, called in 2D *Bivariate Normal*. We will center them one in $(-2, -2)$ and the other in $(1, 1)$, with sizes $(1, 2)$ and $(2, 1)$. These functions are already implemented in Python:

```
import numpy as np
import matplotlib.pyplot as plt

x = y = np.arange(-5.0, 5.0, 0.1)
X, Y = np.meshgrid(x, y)

Z1 = plt.mlab.bivariate_normal(X, Y, 1.0, 2.0, -2.0, -2.0)
Z2 = plt.mlab.bivariate_normal(X, Y, 2.0, 1.0, 1.0, 1.0)
Z = (Z1 - Z2)
```

Here after importing *NumPy* and *MatPlotLib*, I define a background regular mesh. On this *lattice*, I calculate the Gaussian Normal function and operate on them. To plot the difference Z between the Gaussians with a filling contour plot, I need the intuitive command:

```
CS = plt.contourf(X, Y, Z)
```

where however the however requires a *colorbar* to be better interpreted:

```
CS = plt.contourf(X, Y, Z); plt.colorbar(CS)
```

the result is on the left of Fig. 2.1. A large number of options exist to plot more accurately a 2D field with *MatPlotLib*. For example to add contour lines, define values and write them on the plot itself one can program:

```
levels = np.arange(-0.095,0.095,0.02)
CS = plt.contourf(X, Y, Z, levels)
CS4 = plt.contour(X, Y, Z, levels, colors='k')
plt.clabel(CS4, colors='k', fontsize=16)
```

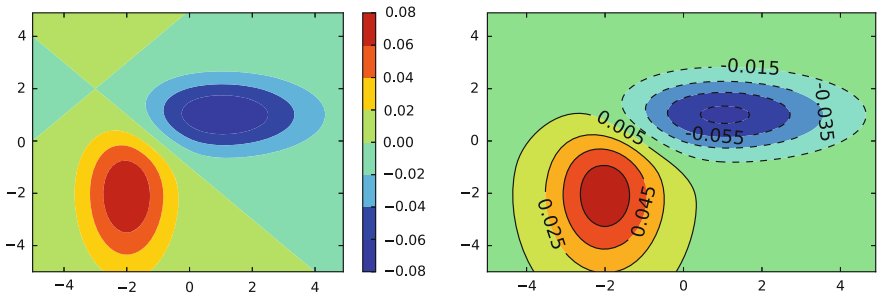

Fig. 2.1 *Left* filled contour plot of the difference of two bivariate normal distributions with a colorbar on the side. *Right* the same functions plotted now with labeled values on the contour boundaries

which gives the result on the right of Fig. 2.1. A figure like this can be saved and shown on the screen with:

```
plt.savefig('filename.pdf')
plt.show()
```

The sequence is important. If `plt.show()` is shown before, `plt.savefig()` will save a blank image.

2.1.2 Plotting a Map

In Geodynamics we often want to map our numerical models on the Earth surface, or plot our data on a physical Map. Aimed at doing this MatplotLib has a specific module called *mpl_toolkits.basemap*. This can be installed, e.g., in Anaconda with the command *conda install basemap*. Let us first create a very simple projection of a map on a sphere of a simple function (decays exponentially toward the poles, oscillated like a wave with the longitude). First we load the module, create the coastlines, color the continents versus the oceans and plot the latitude and longitude:

```
import mpl_toolkits.basemap as bm
import matplotlib.pyplot as plt
import numpy as np

# Let us set our view above Eurasia. And use low resolution. myMap =
bm.Basemap(projection='ortho',lat_0=30,lon_0=60,resolution='l')

# draw coastlines, countries and continents.
myMap.drawcoastlines(linewidth=0.25)
myMap.drawcountries(linewidth=0.25)
myMap.fillcontinents(color='orange',lake_color='aqua')
```

```
# draw the edge of the map, the meridians and the parallels
myMap.drawmapboundary(fill_color='aqua')
myMap.drawmeridians(np.arange(0,360,30))
myMap.drawparallels(np.arange(-90,90,30))
```

You can already run this script in *iPython* and visualize it with `plt.show()`. We are now ready to add a function on top, and I will create them by adding a baseline to a wave on a 100 × 100 grid (I use here some *NumPy* functions that I will explain in detail later):

```
# make up some data on a regular lat/lon grid.
n = 100; d = 2*np.pi/(n-1)
[lats,lons] = d*np.indices((n,n))
wave = np.exp(-lats/10) * np.cos(lons)
mean = np.cos(2.*lats) * np.sin(2.*lats)
x, y = myMap(lons*180./np.pi, lats*180./np.pi) #project the lat/lon on the grid
cs = myMap.contour(x,y,mean+wave,15,linewidths=1.5) #project a contour plot on
  ↪ the map
plt.title('Example of a plot over a global map')
plt.show()
```

And one obtains the result shown in Fig. 2.2.

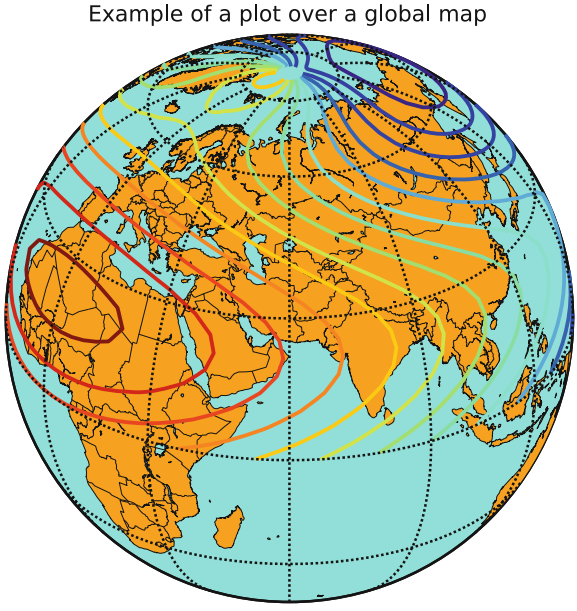

Fig. 2.2 Example of a simple made-up function plot over a spherical projection of the Earth

2.1.3 NetCDF and ETOPO

In the next example, I will use the standard storage system called *NetCDF4*. NetCDF stands for Network Common Data Form and it is an interface for storing and retrieving data in form of arrays, that are the main format of data that we use in our models and for data representation. To store the data in arrays has numerous advantages that we will exploit when learning how to use NumPy. Among them that array values may be accessed directly, ignoring the details of how the data are stored. This means that netCDF datasets can be only partially accessed and transformed, without locally storing and managing the entire datasets, which therefore can be very large.

The physical representation of netCDF data is designed to be independent of the computer on which the data were written. In particular in the version four of NetCDF the modern data format HDF5 was introduced, allowing unlimited datatypes and specifically designed for efficiently transmit high volume and complex data. Examples on how to create, store and retrieve HDF5 data in Python are in https://support.hdfgroup.org/HDF5/examples/intro.html#python.

In order to learn to use NetCDF4, dataset let us play with them and plot a public topography/bathymetry dataset of *NOAA*. First, we have to access it. For this exercise you can also download the entire dataset, but it is huge, while the idea of accessing NetCDF data is to extract only the data that we are interested in. We will use the database of the software *Ferret*. In Anaconda, the netCDF library can be installed with the command *conda install netCDF4*.

```
import mpl_toolkits.basemap as bm
import numpy as np
import matplotlib.pyplot as plt
from netCDF4 import Dataset

#loading the data and extracting the latitude, longitude and topography
#if this dataset is not available anymore, it can be downloaded from many
  ↪ sources
#as well as the more recent higher resolution versions
url = 'http://ferret.pmel.noaa.gov/ thredds/dodsC/data/PMEL/etopo5.nc'
topography = Dataset(url) #extract data using NetCDF
topoin = topography.variables['ROSE'][:]
lons = topography.variables['ETOPO05_X'][:]
lats = topography.variables['ETOPO05_Y'][:]

# ETOPO and basemap are shifted of 180 degrees in longitude, so we need to
  ↪ shift the reference
topoin,lons = bm.shiftgrid(180.,topoin,lons,start=False)
```

Let us plot the topography above the Himalaya, without country boundaries, but only with latitude and longitude plot every 20 degrees. The key function that creates the map is again *basemap*, that we now use with parameters to define the plotting window (min and max lat and lon) and viewpoint from the space.

```
fig = plt.figure()
ax = fig.add_axes([0.1,0.1,0.8,0.8])
m = bm.Basemap(llcrnrlon=40.,llcrnrlat=0., urcrnrlon=140,urcrnrlat=60.,\
            resolution='l',area_thresh=1000., projection='lcc',\
            lat_1=30.,lon_0=90.,ax=ax) #viewpoint from space

n = 1000
nx = 1 + int( (m.xmax-m.xmin)/n )
ny = 1 + int( (m.ymax-m.ymin)/n )
topodat = m.transform_scalar(topoin,lons,lats,nx,ny)
# plot image over map with imshow.
im = m.imshow(topodat)
m.drawcoastlines()
par = np.arange(0.,80.,20.); m.drawparallels(par,labels=[1,0,0,0])
mer = np.arange(10.,360.,20.); m.drawmeridians(mer,labels=[0,0,0,1])
cb = m.colorbar(im,"right", size="5
ax.set_title('Topography above Himalaya')
plt.show()
```

And one obtains the result shown in Fig. 2.3.

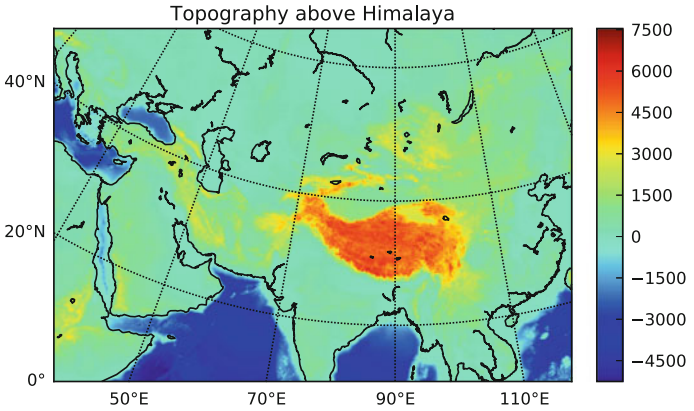

Fig. 2.3 Plot of the topography map over the Himalayan region using NOAA public Data

These were really two very rudimentary examples. Many more and more sophisticated ones can be found in the webpage of the *basemap toolkit* of *MatplotLib*.

There are many other tools for plotting Maps. For example, *PyFerret* allows to extract and manipulate the above dataset with greater speed and care. And the most used Plotting tool in Geophysics, *GMT* can be used as well, through the binding of *PyGMT* and *GmtPy*, although they are not so updated. While I write there is a new project, called gmt-python, lead by Leonardo Uieda and his supervisor Paul Wessel, who promises to become a definitive binding package. *GMT* commands can be also called from the shell line within Python. For example, all the plots in the publication [113] have been made in this way.

There are really advanced projects for global visualization of data related to plate tectonics reconstruction, with the most prominent one being probably *GPlates*, that has been written in many parts using Python and allows interfacing with external scripts by using Python, as explained in its tutorial at https://www.gplates.org/docs.html. Although GPlates presently only plots maps on the two-dimensional surface of the Earth, plans exist for its extension to visualizing it in combination with the Earth's interiors.

2.1.4 Plotting a Seismic Waveform

One of the most inspiring stories of geophysics refers to Bernard Chouet. Born in Switzerland, after graduating in Electrical Engineering, was hired at MIT to work on the NASA's Apollo's missions, therefore at the age of 30 took a Master in Geophysics and started looking at the seismic signatures of volcanoes, with the ambition of predicting the behavior of volcanoes. After obtaining a Ph.D. again at MIT he moved to USGS, where he worked until he discovered in the seismic records of the eruption of 1985 in Nevado del Ruiz, Colombia, that by filtering the seismic signals of the volcano for long-period waves, he could detect and increase of the activity before the eruption. He used then this insight to successfully predict several other eruptions in Alaska and again in Colombia. The life of many people has been saved in Mexico in 2000 by predicting the eruption of Popocatepetl in Mexico using his method.

Today thanks to open Internet databases of worldwide seismic records, we can repeat his analysis in many locations.

The community of seismology made a great effort to deliver extensive data based that could be used by seismologists to test their software, their algorithms, and build seismological and tomographic models. I will show here an example from the *IRIS* database. *IRIS* stands for *Incorporated Research Institutions for Seismology* and its data can be accessed using the *client* of its Python library *obspy*. The following code plots a seismic event at the Hawaii in 2016. The events happened near Kilauea, Hawaii and was detected at the *West Rim* site. I show here a simple filtering of the seismogram where only the frequencies between 1.5Hz and 2.5Hz are selected and used. the results is shown in Fig. 2.4. Again, there is not enough space to list here all the plotting possibilities of *obspy*, furthermore these libraries evolve very rapidly, so the best way to learn to use them is to access the tutorial that for *obspy* is located at the address https://docs.obspy.org/tutorial/.

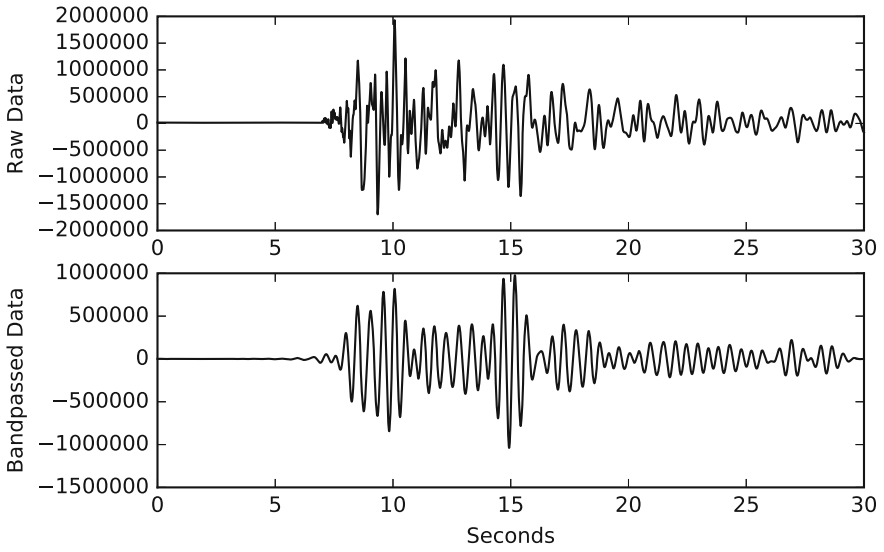

Fig. 2.4 Seismogram of an event on August 24, 2016, detected at the West Rim, Hawaii. The above plot shows the original data, and the one below the filtered one

```
#Plotting a seismic event near Kilauea
import matplotlib.pyplot as plt
import numpy as np
from obspy.clients.fdsn import Client
from obspy import UTCDateTime
from obspy.signal import freqattributes

# Load the data from the web
Network = "HV"; Station = "WRM"; Location = "--"; Channel = "HHE"
t1 = UTCDateTime("2016-08-24T22:17:50.000")
client = Client("IRIS")
st = client.get_waveforms(Network, Station, Location, Channel, t1, t1 + 30)

# There is only one trace in the Stream object, let's work on that trace...
tr = st[0]; df=tr.stats.sampling_rate; dD=tr.stats.delta

# Filtering a copy of the original Trace
fMin=1.5; fMax=2.5
tr_filt = tr.copy()
tr_filt.filter('bandpass', freqmin=fMin, freqmax=fMax, corners=3,
  ↪  zerophase=True)
tr_spec = freqattributes.spectrum(tr_filt.data,df,18001)

# Plottnig raw and filtered data
t = np.arange(0, tr.stats.npts / tr.stats.sampling_rate, tr.stats.delta)
plt.subplot(211); plt.plot(t, tr.data, 'k'); plt.ylabel('Raw Data')
plt.subplot(212); plt.plot(t, tr_filt.data, 'k')
plt.ylabel('Bandpassed Data'); plt.xlabel('Seconds')

plt.show()
```

2.2 Plotting in 3D with MatPlotLib

MatplotLib allows also plotting in 3D. For example, a set of 1000 randomly placed points can be displayed with the script:

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.cm as cm

n = 1000
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
color=cm.rainbow(np.linspace(0,1,n))
xs = np.random.rand(n)
ys = np.random.rand(n)
zs = np.random.rand(n)
ax.scatter(xs, ys, zs, c=color)
plt.show()
```

One outcome is in Fig. 2.5. Colors are assigned sequentially using the function `plt.cm.rainbow()`. From this simple snipped, one can see how the 3D plots can be rotated and seen from any angle.

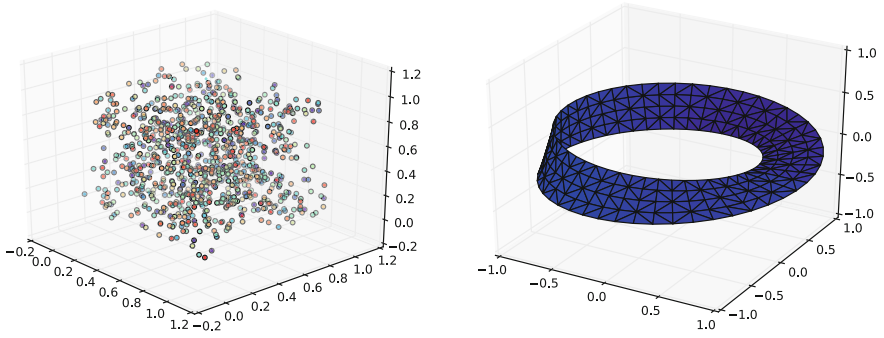

Fig. 2.5 *Left* 1000 dots randomly placed in the space. *Right* the Moebius strip obtained using the `tri.Triangulation()` function of MatPlotLib

2D surfaces in the 3D space can be plot as well. For example, the celebrated Möbius strip, which is a surface with only one side and only one boundary, can be created by the 2D to 3D mapping:

$$x(u,v) = \left(1 + \frac{v}{2}\cos\frac{u}{2}\right)\cos u$$
$$y(u,v) = \left(1 + \frac{v}{2}\cos\frac{u}{2}\right)\sin u$$
$$z(u,v) = \frac{v}{2}\sin\frac{u}{2}$$

I show here how to redo it with Numerical Python. Some of the tools (`np.arange(); np.ones(); np.reshape()`) that I employ in the snipped below will be explained in Chap. 3. Let us instead focus on the Matplot lib functions.

One is *matplotlib.tri.Triangulation*, that takes a collection of points in the 2D space and turns it into a triangulated mesh. If the mesh is not explicitly indicated, Python will build the Delaunay mesh, which is a triangulated mesh that maximizes the size of the angles of the mesh, i.e., avoids thin and long triangles. This is an extremely useful tool that we will use again.

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import tri

n = 50; l = 5

u = np.arange(n)*2*np.pi/(n-1)
u = u * np.ones((l,1))
u = u.reshape(n*l)

v = np.arange(l)/(l-1)-0.5
v = v * np.ones((n,1))
v = v.transpose().reshape(n*l)

# Mobius mapping, (u, v) -> (x, y, z)
x = (1 + 0.5 * v * np.cos(u / 2.0)) * np.cos(u)
y = (1 + 0.5 * v * np.cos(u / 2.0)) * np.sin(u)
z = 0.5 * v * np.sin(u / 2.0)

# Create the triangulated surface
surface = tri.Triangulation(u, v)

fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.plot_trisurf(x, y, z, triangles=surface.triangles)
ax.set_xlim(-1, 1); ax.set_ylim(-1, 1); ax.set_zlim(-1, 1);
```

In the above example, n and l are the parameters controlling the density of the mesh. You can play with their values if their meaning is not immediately clear.

MatPlotLib plots triangulated surfaces by using the `matplotlib.plot\_trisurf()` function. A view of the Möbius stream created by the above program is in Fig. 2.5.

2.2.1 VTK File Format

While in 2D Matplotlib allows plotting almost anything, its 3D features are limited, in particular when dealing with large amounts of data, as is often the case in 3 dimensions. For this reason very large models can be better visualized using third part software. The most common tool used in geosciences is probably *Paraview*, and open and free software, which also runs in parallel when the size of the dataset requires it. For simple datasets as the one above, it is possible to simply use ASCII or binary *VTK* files.

VTK stands for Visualization Toolkit and is associated to a large number of popular data file formats. In *VTK* there are mainly two different styles of file formats. The classical one is the simplest and consists in a serial list of points, connectivity, data. This structure has the advantage of being easy to read and write, normally with a program, and even by hand. We will look here only into this format. An alternative is the *XML* format, that supports random access and parallel input/output. For more information on this last format, the *VTK* website `www.vtk.org` is very well documented.

Here below, I show a routine that allows creating a *VTK* file with any surface created with `matplotlib.tri.Triangulation()`:

```
def writeVTKSurface(outFileName,x,y,z,triangles):
    nodesNumber=x.size
    trianglesNumber=int(triangles.size/3)
    m=open(outFileName,'w')
    m.write('# vtk DataFile Version 2.0\n')
    m.write('Moebius surface\n')
    m.write('ASCII\n')
    m.write('DATASET POLYDATA\n')
    m.write('POINTS '+str(nodesNumber)+' float\n')
    for node in np.arange(nodesNumber):
        m.write(str(x[node])+' '+str(y[node])+' '+str(z[node])+' \n')

    m.write('POLYGONS '+str(trianglesNumber)+' '+str(trianglesNumber*4)+'\n')
    for triangle in np.arange(trianglesNumber):
        m.write('3 '+str(triangles[triangle,0])+'
  ↪  '+str(triangles[triangle,1])+' '+str(triangles[triangle,2])+' \n')

    m.close()
    return()
```

This function can be called in our case, for example, with the command:

```
writeVTKSurface('moebius.vtk',x,y,z,surface.triangles)
```

2.3 Example: Length of the Day

The ability to find and process datasets is essential in geophysics. I show here an example on how to download a dataset from the web (evolution of the Length of the Day for one year) from an official repository and how to plot these data using MatPlotLib.

Earth–Moon dynamics, variations of the Earth's inertial axis, seasonal oscillations, local tides, and other phenomena make the Earth's rotation rate slightly vary in time, causing a fluctuation of the Length of the Day. Present GPS data allow estimating the Earth's rotation rate every day, and therefore the Length of the data (LOD) on a daily bases.

As an example, I illustrate here a plot of a recent dataset from the International Earth Rotation and Reference System Service (`www.iers.org`). I download first the raw data from January 2016 that are available at the address `http://datacenter.iers.org/eop/-/somos/5Rgv/latestXL/207/bulletinb-337/csv`. *csv* is a format for data storage and in our case it is in *ascii*, therefore we can simply parse the file to find necessary data.

In Python 3.x the important Module *urllib* has been introduced, which allows requesting, downloading and writing a file online, if permitted. We will use this module to download the data. Generally in *csv* files the fields are divided by `,` or `;` or a similar symbol. In our case, a direct inspection of the data file shows that it is the symbol `;`. Furthermore the first line of the *csv* file shows the meaning of every field and that the first field is the day and the Length of the Day is the thirteenth, which are indexes 0 and 12 in Python.

We will store the cardinal day number and the Length of each Day in the lists *day* and *LOD*. It is important, when reading an unknown file, to allow the possibility that one line could be not read. This is normally done with the instructions **`try`** and **`except`** that control this case and avoid an error that would irremediably stop the execution:

```
import urllib.parse
import urllib.request
import matplotlib.pyplot as plt

day=[]; LOD=[]
url='http://datacenter.iers.org/eop/-/
  ↪  somos/5Rgv/latestXL/207/bulletinb-337/csv'

with urllib.request.urlopen(url) as response:
lines = response.read()
lines=str(lines)
lines = lines.split('\\n')

for line in lines:
    words = line.split(';')
    try:
      (thisDay, thisLOD)=( float(words[0]),  float(words[12]) )
        day.append(thisDay); LOD.append(thisLOD)
    except:
        print('line not readable')
```

```
plt.plot(day,LOD)
plt.show()
```

Here we have extract the data from the file as lines of text and then used the function *split()* to divide different fields. Different lines are divided by the *newline* special character (*n* in our case). Different fields by ;. Finally, we transform the text into float numbers and append to the initially empty *day* and *LOD* lists. The plot associated to this script is on the left of Fig. 2.6.

To visually detect cycles in the evolution of the LOD we need to look at several months, or better years. This can be done by extending the above script to reading a series of files, one for each month, and then putting all the data together. For example the following could be a way (only the modified part is shown):

```
numberOfYears=5
day=[]; LOD=[]
fileNames=[]
url = 'http://datacenter.iers.org/eop/-/somos/5Rgv/latestXL/207/'
for i in range(338-12*numberOfYears, 338, 1):
    fileNames.append('bulletinb-'+str(i))

for fileName in fileNames:
    with urllib.request.urlopen(url+fileName+'/csv') as response:
        lines = response.read()
        lines=str(lines)
        lines = lines.split('\\n')
        for line in lines:
            words = line.split(';')
            try:
                (thisDay, thisLOD)=( float(words[0]) ,  float(words[12]) )
                day.append(thisDay); LOD.append(thisLOD)
            except:
                print('line not readable')
```

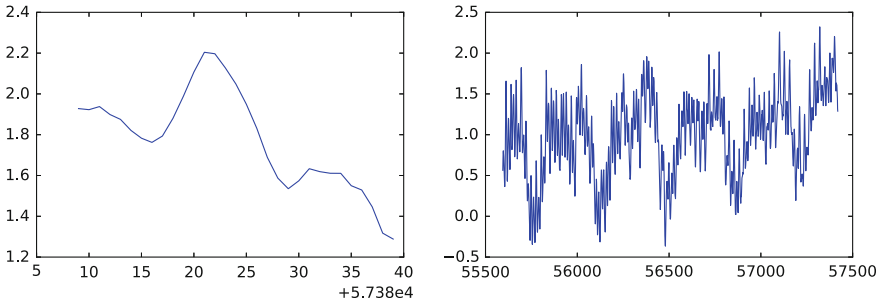

Fig. 2.6 *Left* time series extracted from the *IERS* website related to the variations of the Length of the Day in January 2016. *Right* the same plot for a period of 5 years, combining the data extracted by 60 (5*12) monthly files from the same source. In this second plot at least two overlapping cycles are clearly visible

The result of this script is on the left of Fig. 2.6 and now several cycles are clearly visible. We will see in the next chapter, using NumPy how we can extract more information from this time series.

2.4 IPython and Jupyter Notebooks

IPython, *Interactive Python*, initially started as a very practical and handy interface for Python, in particular aimed at prototyping new programs. Today it is a growing, language-agnostic, project. Agnostic in the sense that it is based on a *Notebook* format. To learn iPython is like to learn MatPlotLib: the best way is to start from examples, which can be all found presently at the page `ipython.org`. An even greater set of examples can be found in the *A gallery of interesting IPython Notebooks*, presently available on the `github.com` platform [108].

Among the many features of iPython, it is important to mention the existance of the *magic commands* that allow to quickly perform *extra Python* operations. These commands are of two types, either *line magics* or *cell magics*, the first being anticipated by % and the second by %%. Example of the first case is *%time*, that allows testing the speed of a certain command. For example, to benchmark the time necessary to create an array of sequentially 1 billion integers one can write *%time sum(range(1000000000))* (27 s on my laptop). Examples of the seconds are *%%script*, that allows writing a shell command and possibly run other languages (e.g., *%%script ruby*) inside a Python script.

In 2015, the iPython project has evolved into the Jupyter project, where the *Notebook* is now at the center. This application allows creating and sharing documents in the form of a *Notebook*. These documents can contain live code, equations, visualizations, and explanatory text. The idea behind their creation is extremely ambitious and in many ways aims at transferring the framework of the Open-Source Software (OSS) into Academia. Jupyter Notebooks have the characteristic of being very technical and detailed. In this sense, they allow to distribute knowledge in a new form compared to technical publications where only general directives about implementations are given, allowing to increase the speed at which science can spread.

Jupyter Notebooks do not exist only for Python but with many other languages as well, however here I will show only examples with Python. On printing this book, I plan to transfer most of the examples from this book into Jupyter Notebooks and to share them online.

The reason for inserting *iPython* and *Jupyter* in the visualization chapter is that these tools aim at reproducing and sharing techniques. They are therefore a way to *visualize* them.

2.5 Paraview and VisIt

One of the best Python integrated powerful visualization software, openly and freely available to anyone, is ParaView. Personally it has been my preferred choice for over a decade. Paraview is multiplatform, working on Linux, OS-X and Windows. On every platform it builds 3D visualizations at an amazing speed plotting any kind of data on volumes, surfaces, sections and other more complex integrated geometries. Normally it is used in an interactive mode, to visualize ongoing simulations, however using Python is possible to programmatically process it and create figures with a consistent outlook, which is particularly useful when writing publications, e.g. to use the same color-scale and plot 3D data from the same angle for large datasets.

The name ParaView clearly comes from Parallel, because it has been developed in order to analyze extremely large datasets on a distributed computing system. It can run on supercomputers and analyze petabytes of data, well beyond what can be stored in the hard-drive of most desktops, but it results extremely fast and efficient on a laptop as well. A detailed and updated guide to Paraview is available at `http://www.paraview.org/paraview-guide/` in PDF. Here I only show few examples to illustrate its potential.

The simplest way to combine Python and Paraview is to create VTK files and open them with Paraview. For example the Moebius stream that I created above can be visualized immediately (see Fig. 2.7).

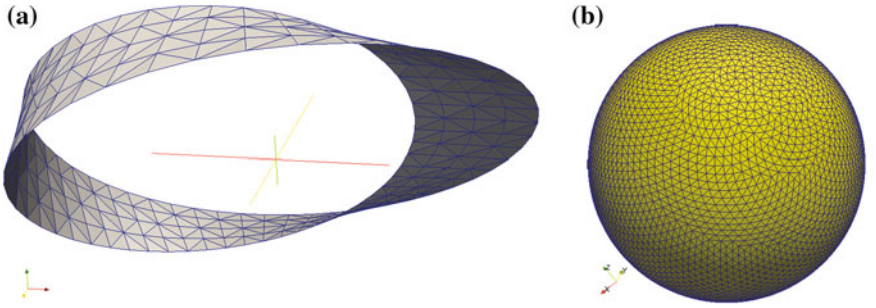

Fig. 2.7 Left: Snapshot of the visualization of the Moebius stream done with Paraview. Right: visualization of the mesh of a sphere obtained with a Fibonacci spiraling technique, again visualized from a VTK file in Paraview

One can similarly create much more complex setup and visualize results. For example I can create a set of bubbles in space, at random position and of random size (with certain limits), that has been used for research (e.g., [27, 123, 124]. Over there, the mesh was created by recursively subdividing the faces of a regular polyhedron in 4 parts until the desired resolution is achieved. I show here instead a simpler approach based on spiraling around a sphere to setup the points of the mesh and then using the *Convex Hull* to triangulate the surface:

```
import numpy as np
from scipy.spatial import ConvexHull

samples = 5000
increment = np.pi * ( 3 - 5**0.5 )
points = np.zeros((samples,3))

phi = np.arange(0., samples*increment, increment)
points[:,1] = ( 2. * np.arange(samples) + 1 )/samples - 1.
r = ( 1 - points[:,1]**2)**0.5
points[:,0] = np.cos(phi) * r
points[:,2] = np.sin(phi) * r

hull = ConvexHull(points)
```

Here the key is in using the interval equal to the angle in radiants $\pi(3-\sqrt{5})$, derived from the golden number, and obtaining a sequence like the Fibonacci one. This creates a perfect covering of a sphere, with any given number of number. The result of this mesh is shown in Fig. 2.7. In Exercise 2.5 it is suggested to use a number of spheres created with this technique to create volume filled with spherical beads. Techniques to create such filling, important in many fields of science of heterogeneous media, and in many fields of petrology and sedimentology, can be found in [45].

Alternatively one can create a Python script and run it with *pvpython*, the dedicated python interpreter created for Paraview. In *pvpython* both NumPy and MatplotLib can be called, making it a powerful tool, however most of its command are complex and its strict object oriented structure make it difficult to use for the beginner programmer.

Another large open source project in which Python plays a big role is *VisIt*. VisIt, like ParaView, is designed for post processing of mesh based data, mostly scientific. Python was adopted early by the VisIt team as the VisIt's primary scripting interface [73]. VisIt's developers have embedded Python interpreters into their data flow network pipelines, allowing users to write custom algorithms in Python to manipulate the data in the mesh. Once installed *VisIt*, any python program can be sent to *VisIt* just with the command *./visit -nowin -cli -s <script.py >*. On the *VisIt* platform many examples are given on how to plot functions and dataset from VTK sets. Generally the outcome is similar to Paraview, with however more attention to visualizing fluid-dynamics, streamlines and movies.

2.6 Python as a wrapper: SEATREE and Underworld

Large collaborative projects specifically applied to geodynamics also exist. Many computational geodynamicists use commonly Python as the wrapper for integrating existing open source software. *SEATREE* (Solid Earth Teaching and Research Environment) is general project that aims at connecting in the most efficient way very different computational modules ranging from a mantle convection code, to flow visualization, from mantle tomography sampling and subsampling to 3D visualiza-

tion, from body wave mantle seismic tomography to earthquake location inversions. The goal of SEATREE is its application in the classroom, and as a platform for scientific collaboration. Here Python is used mainly as a wrapper to combine together existing well-known codes that have been used by the scientific community for many years.

The purpose of SEATREE is different from most of the examples shown in this book, which have mainly the purpose of understanding the techniques behind geodynamic numerical modeling. SEATREE is instead an attempt to provide an easy and graphically supported *black box* tool, where however what lies under the hood is well explained.

Underworld is also project aiming at combining modeling and visualization, but by using original and innovative Particles in Cell implementations. It is Python-friendly by providing a programmable and flexible front end that allow to quickly setup standard geodynamic setups for running parallel HPC simulations. It is specifically suited for modellers who do not have an in-depth knowledge of the code development and want to focus on complex geodynamic simulations. Tutorials to learn how to use Underworld also use the Jupyter Notebooks, which makes easy to integrate it with the techniques introduced in this book.

Summary

- MatPlotLib is the most used visualization library in Python. It is particularly powerful in 2D, but limited in 3D, particularly for visualizing the results of complex numerical models.
- Maps and other geometrical features are already implemented in MatPlotLib. It is straightforward to plot fields on geographical projections in 2D and 3D.
- MatPlotLib allows plotting features in 0D (points), 1D (segments), 2D (triangles, squares) as well as 3D (spheres, blocks, etc.).
- Paraview and VisIt are much more powerful visualization tools but need specific file formats (such as VTK) to be used efficiently to plot numerical results.

Problems

2.1 Plot the topography of Africa, using the same `http://ferret.pmel.noaa.gov/thredds/dodsC/data/PMEL/etopo5.nc'` of Sect. 2.1.3. The topography data are called *ROSE* and latitude and longitude are called *ETOPO05_X* and *ETOPO05_Y* respectively. By setting to zero all negative topography values and plotting only the `np.log10()` of the topography the continental morphology appears very neat.

2.2 Using the strategy shown in Sect. 2.1.3 for accessing large NetCDF files, extract the global precipitation data from the NOAA website (e.g. `http://www.esrl.noaa.gov/psd/thredds/dodsC/Datasets/cmap/std/precip.mon.mean.nc`) and plot a precipitation map for a continent for Jan. 2009. A careful look at the NetCDF file reveals that the data are organized in *time*, *longitude* and *latitude*, and averaged per month, starting in January 1979, therefore to obtain the data for January 2009 one needs to extract the month after exactly 30 years.

2.3 Extract and visualize the sea surface Temperature from the Sect. 2.1.3 file at the NOAA website (e.g. `ftp://ftp.cdc.noaa.gov/Datasets/COBE2/sst.mon.mean.nc`). The data file is about 500 Megabytes large, therefore extract only the necessary data. The dataset ranges from 1850 to 2015. Make one plot per year to show the time progression.

2.4 Download a NetCDF file of Paleo Age Crustal Data from the `http://www.earthbyte.org/Resources/agegrid2008.html` website. The data are organized in *age*, *longitude* and *latitude* as z, x, y. Plot the seafloor age for an ocean, such as *Atlantic*, *Pacific* or *Indian*. Use the `mpl\_toolkits.basemap` as shown in Sect. 2.1.3.

2.5 Build a VTK file (Sect. 2.2.1) and show in Paraview (Sect. 2.5) the highly compact set of spheres that occupies the volume $1 \times 1 \times 1$ with a compaction of 70%. You will need to use sphere with different dimensions as such a high compaction cannot possibly be achieved with sphere of the same size. Use first large sphere and then fill the remaining space with smaller spheres, where possible. The setup can be all identified before building the mesh, since the distance between two spheres is given by the distance between the centers mines the sum of the two radii. Use Paraview to visualize the sphere surface mesh.

http://www.springer.com/978-3-319-55680-2

Pythonic Geodynamics
Implementations for Fast Computing
Morra, G.
2018, XVI, 227 p. 46 illus., 40 illus. in color., Hardcover
ISBN: 978-3-319-55680-2