Introduction to graphics and math packages

4TTV313U - Introduction to simulation of dynamic process

University of Bordeaux - 2022

Graphs with Matplotlib

Even before analysing and modeling observations, we need to look at them to have a general idea of the dynamics of the system, sometimes detect peculiar processes or even outliers. We also need to provide graphical outputs from our models. For all figures, we will use the package `matplotlib.pyplot`.

Even though this package has numerous uses (see matplotlib gallery), we will focus here on the most basic one: the scatter (x-y) plot. If we get back to the values extracted in the previous script, we can plot a yearly value of NAO.

```
import matplotlib.pyplot as plt

# We check the types of data are reals or integers.
type(year[0])
type(val[0])

#We want to plot the NAO values for the first month of every year
val_bis=[val[i*12] for i in range(0,70)]
year_bis=[year[i*12] for i in range(0,70)]

plt.figure()
plt.plot(year_bis,val_bis,color="blue",linestyle="-",linewidth=2,
... marker="o",markeredgecolor="black",markerfacecolor="darkblue",markersize=5)
plt.xlabel("Year")
plt.ylabel("NAO")
plt.title("NAO variation")

plt.savefig("NAO_plot.pdf",bbox_inches="tight")
```

Exercise

Plot the population growth with data from `World_population_simplified.csv`. Don't forget title, labels... all information that are needed for someone not familiar with your data to understand what they are looking at.

Maths with NumPy

NumPy is the reference package for mathematical tools and easier-to-handle formats for vectors and matrices.

```
import numpy as np

listx =[0,1,2,3,4,5]; listy =[4,5,6,7,8,9]
listx+listy
x = np.array (listx); y = np.array (listy)
```

```
x+y

#New lists / 1D-array
x=list(range(10))
type(x)
x_np_1=np.arange(10)
type(x_np_1)
x_np_2=np.linspace(0,9,10)  #What is the difference between the two commands?

##Arrays
x = np.zeros((2,2), dtype = np.float)
x.ndim
x.shape
y=x.flat[:]
y.shape
x.shape=(4,1)
x.shape #Notice the difference between x and y
```

Other interesting features of the NumPy package include usual mathematical functions (sin, cos, exp...) and random number generation.

```
x=np.random.uniform(0,10,(1,10))
x_2=np.random.uniform(0,10,(1,10))

np.random.seed(42)
x_3=np.random.uniform(0,10,(1,10))
np.random.seed(42)
x_4=np.random.uniform(0,10,(1,10)) ###What does seed() do?

exp_x=np.exp(x)
```

There are a lot of features in the NumPy package: you will discover some of them during the classes and your projects.

Exercises

1. Draw a two panel-plot (use `matplotlib.pyplot.subplot(1,2,1)`): in the first panel, plot the following functions from -3 to 3 with a 0.01 time step: $f_1(x) = x^2$ and $f_2(x) = x^3$. On the second panel, plot the function $f_3 = \sin(x)\cos(x)^2$. Don't forget the legend.
2. Model a seasonal temperature with a daily timestep.
   (a) Assume that the average temperature over a year is 12.5°C, the amplitude of variation is 20°C and the signal is sinusoidal.
   (b) There are small, daily variations in temperature. Add a random error on the temperature for every day. We assume that this error is normally distributed with mean=0 and standard deviation=1.
   (c) We know that there has been a global warming over the last 150 years. This has been estimated around 0.2°C per decade. Add this to your temperature signal. Plot a year of temperatures at the beginning and at the end of your simulation (after 150 years).

Cellular automaton: Conway's game of life

The system is defined as a two-dimensional grid of cells, each of which is in one of two possible states, alive or dead. Every cell interacts with its eight neighbours, which are the cells that are horizontally, vertically, or diagonally adjacent. At each timestep, the following transitions occur:

- Any live cell with fewer than two live neighbours dies, as if by underpopulation.
- Any live cell with two or three live neighbours lives on to the next generation.
- Any live cell with more than three live neighbours dies, as if by overpopulation.
- Any dead cell with exactly three live neighbours becomes a live cell, as if by reproduction.

You will produce the initial state randomly. Each generation is a function of the preceding one. The rules continue to be applied repeatedly to create further generations. Stop at 20 iterations. Save your first and final iterations in a text file and plot them (you will need the `matplotlib.pyplot.imshow` function to draw the matrix).