ICT for Health
Python

Monica Visintin

Politecnico di Torino

ICT4SS
Information and Communications Technologies for Smart Societies

2018/19

Table of Contents

Table of Contents

Python [1]

NumPy numericalpy

- Python is a high level programming language, somehow similar to Matlab since it allows to add two vectors/arrays `a` and `b` by simply writing `c=a+b`
- like Matlab, it is not necessary to declare the variables (which is an advantage and a disadvantage...)
- Python is free (Matlab costs around 10 keuros per year)
- Python scripts run faster than corresponding Matlab scripts, but slightly slower than corresponding C programs
- Python uses extensions/libraries that can be imported, if necessary. For example **NumPy** is required for working with vectors and matrices, **scipy** for using the FFT, **matplotlib** is an extension for generating plots with a syntax similar to that of Matlab.

Python [2]

- You can use Python
  1. interactively (like Matlab): you simply enter the command `$python` in a console (not so nice, but useful)
  2. by running a Python script `myscript.py` from a console with the command line `$ python myscript.py`
  3. through the *jupyter notebook*, which allows you to write and comment your code and execute portions of it or the entire code
  4. through an IDE (Integrated Development Environment) like *Spyder* or *PyCharm Edu*, which allows you to write your code, automatically check the grammar and "good writing", and run portions of code or the entire code.

latex export

Python [3]

- You can write a Python script with
  1. a normal editor (but then you can only run the entire script, and you must use the command `$ python myscript.py` in a shell)
  2. an IDE like *Spyder* or *PyCharm Edu*: the advantage is that you can autocomplete your code, get help, inspect your variables and objects. Basically these IDEs make Python very similar to Matlab
  3. *jupyter notebook* (you can run the entire script, or only portions of it; numerical results, pictures etc are embedded in the file, and, even if you close your session and you reopen it later, you find not only the script, but also the figures and the results). Jupyter notebook allows you to directly write the report of the lab while you write the script.
- There are two versions of Python: Python 2 (old version) and Python 3 (new version, partially incompatible with Python 2). In the future, only Python 3 will be maintained (at least, this is what they say...).

Python [4]

- Linux distributions include Python (it might be Python2 or Python3). If you have Python2, you can install Python3; if you have both installed and you want to use Python3, the command in the shell must be `$ python3 myscript.py`, otherwise Python2 will be used. Pycharm Edu, Spyder and Jupyter notebook ask you which version of Python you want to use.
- Python installation. Installation might be a complex task, we suggest to first install **Anaconda**, then use Anaconda to install Python and all its extensions/libraries; Anaconda allows you to have different environments and different settings for each environment; otherwise install Python and use pip to install the needed extensions (including TensorFlow)

Object Oriented Programming (OOP)

Very briefly: Python allows to define “**class**es” for which you can define “**method**s”; once you “**instantiate**” an “**object**” belonging to a given “class”, the object “**inherits**” the “methods” of that “class”. An example will show the meaning of this.

Some preliminar details

- **Comments** in Python are preceded by **#**. If a comment takes more than one line, you must use **#** at the beginning of each line
- Python uses **indentations** (tabs) to separate “sections”. If blanks are not correctly placed, you get an error and the code does not run.

Table of Contents

Example of a Python-3 script [1]

Goal:

1. We want to solve the minimization program

$$\min_{\mathbf{w}} \|\mathbf{y} - \mathbf{A}\mathbf{w}\|^2$$

   where $A \in \mathbb{R}^{N_p \times N_f}$, $\mathbf{w} \in \mathbb{R}^{N_f \times 1}$, $\mathbf{y} \in \mathbb{R}^{N_p \times 1}$

2. We write Python-3 classes that implement Linear Least Squares (LLS), Gradient Algorithm (GA), Steepest Descent (SD) and a main script to test these algorithms

Example of a Python-3 script [2]

- Vectors and matrices are managed by the Python extension/library called Numpy. The first line of the Python script is

```
import numpy as np
```

  If you get an error, you probably forgot to download and install Numpy, do it!

- In Numpy vectors and matrices are called Ndarrays (Nd stands for N-dimension). How do you create an Ndarray?

```
y=np.ones((Np,1),dtype=float)# column vector of Np floats all equal to 1
y=np.zeros((1,Np),dtype=int)# row vector of Np integers all equal to 0
A=np.eye(4)# 4x4 identity matrix
y=np.array([1,2,3])# vector (neither column nor row) with values 1,2,3
A=np.array([[1,2,3],[4,5,6]])# 2x3 matrix
z=np.arange(5)# z=[0,1,2,3,4], 5 elements starting from 0
z=np.arange(5,7,dtype=float)# z=[5.0,6.0]
```

Example of a Python-3 script [3]

- Operations on Ndarrays:
  - if `A` and `B` have the same size, `A+B` is the elementwise sum of the two Ndarrays
  - if `A` and `B` have the same size, `A*B` is the elementwise product of the two Ndarrays (WARNING: in Matlab this corresponds to `A.*B`)
  - `np.dot(A,B)` is the (linear algebra) dot product between Ndarrays `A` and `B`, i.e. `A*B` (sizes of the Ndarrays must be correct)
  - if `A` and `B` have adequate size, `A@B` is the product of the two Ndarrays (WARNING 1: in Matlab this corresponds to `A*B`)(WARNING 2: operator `@` have been only recently included, you can use `np.dot(A,B)`)
  - if `A` is an Ndarray, `M,N=A.shape` gives the shape of the Ndarray: `M` rows and `N` columns
  - if `A` is an Ndarray with shape `(M1,N1)` and `(M2,N2)` is another possible shape (`M2` and `N2` integers and `M2*N2=M1*N1`), then `B=np.reshape(A,(M2,N2))` gives `B` with the elements of `A` and shape `M2,N2` (check how this is done)
  - `np.kron(A,B)` is the Kronecker product between two Ndarrays `A` and `B`

Example of a Python-3 script [4]

- Numpy has sub-libraries: `linalg` for linear algebra and `random` to generate samples of random variables, etc.

Example of a Python-3 script [5]

- Useful functions available in the `linalg` sub-library:
  - `np.linalg.norm(x)` square norm of 1D array `x` (i.e. $\sqrt{\sum_k x^2(k)}$)
  - `np.linalg.inv(A)` inverse of the square matrix `A`
  - `np.linalg.det(A)` determinant of the Ndarray `A`
  - `lambda, U = np.linalg.eig(A)` eigenvalues and eigenvectors of 2D array `A`

Example of a Python-3 script [6]

- Useful functions available in the `random` sub-library:
  - `np.random.rand(M,N)` generates an Ndarray with `M` row and `N` columns with random numbers extracted from a uniform probability density function (pdf) in the range $[0, 1]$
  - `np.random.randn(M,N)` generates an Ndarray with `M` row and `N` columns with random numbers extracted from a Gaussian probability density function (pdf) with 0 mean and variance 1
  - `np.random.randint(K,size=(M,N)` generates an Ndarray with `M` row and `N` columns with random integer numbers uniformly distributed in the range $[0, K-1]$
  - `np.random.shuffle(x)`, where `x` is an Ndarray (vector) randomly permutes/shuffles the elements of `x`

Example of a Python-3 script [7]

- Let us generate a matrix `A` and a vector y. We generate a fake case, in which we know $\mathbf{w}$ (random), we know $\mathbf{A}$ and we generate $\mathbf{y} = \mathbf{A}\mathbf{w}$. Then we pretend we don't know $\mathbf{w}$ and we use the algorithms to find it.

```
Np=5 # Number of rows
Nf=4 # Number of columns
A=np.random.randn(Np,Nf) # generate a Gaussian random matrix A
w_id=np.random.randn(Nf) # generate a Gaussian random vector w_id
y=np.dot(A,w_id)
```

Example of a Python-3 script [8]

- Let us now apply the linear least square method:

$$\hat{\mathbf{w}} = (\mathbf{A}\mathbf{A}^T)^{-1}\mathbf{A}^T\mathbf{y}$$

```
AAT=np.dot(A,A.T) # generate A*AT
AATinv=np.linalg.inv(AAT) # generate (A*AT)**(-1)
AATinvAT=np.dot(AATinv,A.T) # generate (A*AT)**(-1)*AT
w=np.dot(AATinvAT,y) # generate w=(A*AT)**(-1)*AT*y
```

or you can use `np.linalg.pinv` which directly generates the pseudoinverse:

```
w=np.dot(np.linalg.pinv(A),y) # generate w
```

Carefully search the web when you have to do an operation: maybe the operation is already included in a Python library

Example of a Python-3 script [9]

- Now we want to print the result:

```python
print(w)
```

WARNING: in Python 2, this would be `print w` (without the brackets). If you want something nicer, you can use this code:

```python
print('Result obtained by applying the LLS method:')
print('vector w is: ',w)
```

Example of a Python-3 script [10]

- Now we want to plot the result. We need Python library matplotlib

```
import matplotlib.pyplot as plt
plt.figure()# create a new figure
plt.plot(w)# plot w with a line
plt.xlabel('n')# label on the x-axis
plt.ylabel('w(n)')# label on the y-axis
plt.grid() # set the grid
plt.title('Solution of the problem')# set the title
plt.show()# show the figure on the screen
```

Search `https://matplotlib.org/` for the details about plots (really many possibilities).

Table of Contents

What do we want to do?

- We want to use more than one optimization technique, and for all these techniquess we want to print and plot the resulting solution vector `w`.
- Then we define a class `SolveMinProbl` in which we define the methods to print and to plot vector `w` and maybe other methods. Then we define the classes `SolveLLS`, `SolveGrad`, `SolveSteepDesc` that belong to class `SolveMinProbl` and all inherit the methods to plot and to print `w` (so you do not have to repeat the methods for each of the classes).
- We start by writing in file `minimization.py` the lines in the next slide.

Class SolveMinProbl (1)

```python
import numpy as np
import matplotlib.pyplot as plt
class SolveMinProbl:
    def __init__(self,y=np.ones((3,1)),A=np.eye(3)): #initialization
        self.matr=A # matrix A
        self.Np=y.shape[0] # number of rows
        self.Nf=A.shape[1] # number of columns
        self.vect=y # column vector y
        self.sol=np.zeros((self.Nf,1),dtype=float) # column vector w
        return
    def plot_w(self,title='Solution'): # method to plot
        w=self.sol
        n=np.arange(self.Nf)
        plt.figure()
        plt.plot(n,w)
        plt.xlabel('n')
        plt.ylabel('w(n)')
        plt.title(title)
        plt.grid()
        plt.show()
        return
    def print_result(self,title): # method to print
        print(title,' :')
        print('the optimum weight vector is: ')
        print(self.sol)
        return
```

Class SolveLLS (2)

```
class SolveLLS(SolveMinProbl):# class SolveLLS belongs to class SolveMinProb
    """
    Comments...
    """

    def run(self):# method that finds w
        A=self.matr
        y=self.vect
        w=np.dot(np.dot(np.linalg.inv(np.dot(A.T,A)),A.T),y)
        self.sol=w
        self.min=np.linalg.norm(np.dot(A,w)-y)
```

Note that initialization is inherited from class `SolveMinProbl` and therefore the construct `def __init__(self,...):` is missing. On the other side we introduce method `run` (without parameters) that sets the values of `self.sol` and `self.min`, according to the formula of minimum square error.

Use of the defined classes in the main [1]

- We have just one file `minimization.py` that already contains classes `SolveMinProbl` and `SolveLLS`, we have now to tell Python what it has to do if you write in the shell `$ python3 minimization.py`: we add (at the bottom of file `minimization.py`) the following lines:

```
if __name__ == "__main__":
    Np=4 #number of rows
    Nf=4 #number of columns
    A=np.random.randn(Np,Nf) # matrix/Ndarray A
    y=np.random.randn(Np,1)# column vector y
    m=SolveLLS(y,A) # instantiate the object
    m.run() #run LLS
    m.print_result('LLS') # print the results (inherited method)
    m.plot_w('LLS')# plot w (inherited method)
```

This solution thus only requires one file, in which you write the entire code (both the main and the called classes/methpods).

Use of the defined classes in the main [2]

- You might want to have one file with the classes/methods (like a library, in folder `./sub`) and a separate file for the main. You write another Python file `lab0.py` that uses `SolveLLS` but you have to import file `./sub/minimization.py` that includes `SolveLLS`. Your file `lab0.py` will be:

```
from sub.minimization import * # import (entirely) file minimization.py
import numpy as np
Np=4 #number of rows
Nf=4 #number of columns
A=np.random.randn(Np,Nf) # matrix/Ndarray A
y=np.random.randn(Np,1)# column vector y
m=SolveLLS(y,A) # instantiate the object
m.run() #run LLS
m.print_result('LLS') # print the results (inherited method)
m.plot_w('LLS')# plot w (inherited method)
```

Use of the defined classes in the main [3]

Even if you import `minimization.py`, which includes the lines `if __name__ == "__main__":` etc., these lines will not be executed (they are executed only if you write in the shell `$ python3 minimization.py`). For this solution to work, it is necessary that you create an empty file `__init__.py` in the folder `./sub` that stores `minimization.py`. After you have run at least once `lab0.py`, you'll see file `minimization.pyc` in folder `./sub`: this file is the **compiled** version of `minimization.py`

Class SolveGrad [1]

Now we add the class that implements the gradient algorithm:

```
class SolveGrad(SolveMinProb):
    """ Comments...
    """

    def run(self,gamma=1e-3,Nit=100):# we need to specify the params.
        self.err=np.zeros((Nit,2),dtype=float)
        # value of the function to be minimized: the 1st column stores the
        # iteration step, the 2nd column stores the value of the error
        A=self.matr
        y=self.vect
        w=np.random.rand(self.Nf,1)# random initialization of w
        for it in range(Nit):
            grad=2*np.dot(A.T,(np.dot(A,w)-y))
            w=w-gamma*grad
            self.err[it,0]=it
            self.err[it,1]=np.linalg.norm(np.dot(A,w)-y)
        self.sol=w
        self.min=self.err[it,1]
```

Class SolveGrad [2]

- Class `SolveGrad` must be added to file `minimization.py`.
- If everything is correct, the error $\|\mathbf{y} - \mathbf{A}\mathbf{w}(k)\|^2$ should decrease as the iteration step k increases, and it is convenient to check it is like that, because this is a way to understand if the learning coefficient is too small or too large.

Class SolveGrad [3]

- Then we want to plot the error versus k. Since also the steepest descent technique has a decreasing error, it is convenient to add method `plot_err` in class `SolveMinProb`:

```
    def plot_err(self,title='Square error',logy=0,logx=0):
        """           """
        err=self.err
        plt.figure()
        if (logy==0) & (logx==0):
            plt.plot(err[:,0],err[:,1])
        if (logy==1) & (logx==0):
             plt.semilogy(err[:,0],err[:,1])
        if (logy==0) & (logx==1):
             plt.semilogx(err[:,0],err[:,1])
        if (logy==1) & (logx==1):
             plt.loglog(err[:,0],err[:,1])
        plt.xlabel('n')
        plt.ylabel('e(n)')
        plt.title(title)
        plt.margins(0.01,0.1)# leave some space
        plt.grid()
        plt.show()
        return
```

Class SolveGrad [4]

- In the main part of the file, we have

```
Nit=1000
gamma=1e-5
g=SolveGrad(y,A)
g.run(gamma,Nit)
g.print_result('Gradient algo.') # inherited method
logx=0
logy=1
g.plot_err('Gradient algo: square error',logy,logx) # inherited method
```

Table of Contents

To do (at home, before next Wednesday)

- Write the Python code in your file/files (you can copy and paste from this pdf file)
- Add the class that implements the steepest descent (write it on your own)
- Compare the results you get with the LLS, gradient algorithm and steepest descent algorithm
- Learn (search the web) how to save a picture in a file with Matplotlib (this is necessary to write reports)
- Answer these questions:
    1. From a theoretical point of view, should the solution you find with the gradient algorithm be equal to the solution you find with LLS?
    2. From a theoretical point of view, should the solution you find with the steepest descent algorithm be equal to the solution you find with LLS?
    3. Do you get numerically equal solutions with the three techniques? why?
- This part will be used to solve the regression problem of Lab 1.