Chapitre I
Rappels sur la programmation sous Python

Introduction à Python

Python est un langage de programmation polyvalent et accessible (open source), qui a gagné en popularité grâce à sa syntaxe claire et ses vastes bibliothèques, il est idéal pour les débutants. Dans ce qui suit, nous allons présenter **l'essentiel pour faire des calculs scientifiques**.

Syntaxe et Accessibilité

Bibliothèques

Python bénéficie d'un large éventail de bibliothèques comme NumPy, Pandas et Matplotlib etc.., qui facilitent les tâches de manipulation de données, de calcul numérique et de visualisation. MATLAB, bien qu'extrêmement puissant pour les applications mathématiques et d'ingénierie, peut nécessiter des licences coûteuses et se concentre principalement sur ces domaines. En revanche, C, bien que très performant pour les applications nécessitant une optimisation poussée, demande une grande quantité de code pour réaliser des tâches simples, ce qui peut alourdir le développement.

I.0 Entrées et Sorties

Comme ce résumé est basé l'apprentissage par des exemples, il est impossible de le faire sans introduire au préalable d'une manière élémentaire comment on introduit une valeur à une variable et comment on l'affiche sur écran.

* Entreées

Les variables peuvent être introduite simplement par l'affection (algorithmique←) qui prend le symbôle = en python.

On peut aussi utiliser la fonction **input**(), il faut cependant, préciser qu'en Python la valeur est considérée comme une **chaine de caractère** (String str), il faut la faire suivre par une **conversion** dans le type désiré (**integer** ou **float**) pour être considéré comme un nombre.

```
a=input("introduire un nombre  ")

af=float(a)
```

* Sortie

Pour afficher les différents résultats, on utilise la fonction **print**( ) dont la syntaxe est :

print(objet,sep=' ',end='\n',file=sys.stout,flush=False)

objet: la variable à afficher

sep (facultative): le séparateur par défaut espace ' ' on peut rencontrer la tabulation '\t' ou ' ;' ou autre

end='\n' : passer à une nouvelle ligne sinon end= ''

file : où envoyer les résultats par défaut file=sys.stout sortie vers écran

flush=True force l'écriture immédiate dans le flux de sortie

Exemple

```
x=3
#differente façon pour afficher x
print('x1=',x)
print("x2=",x)
print(f'x3={x}')
```

affichage

```
x1= 3
x2= 3
x3=3
>>>
```

I.1 Types de données et expressions

Python ne nécessite pas la déclaration des variables, leurs type est déterminé automatiquement lors de leurs affectations.

a) Types de données de base

Text Type:	`str`
Numeric Types:	`int`, `float`, `complex`
Sequence Types:	`list`, `tuple`, `range`
Mapping Type:	`dict`
Set Types:	`set`, `frozenset`
Boolean Type:	`bool`
Binary Types:	`bytes`, `bytearray`, `memoryview`
None Type:	`NoneType`

- ***type numérique***

int (entiers >0 ou <0) exemples n=10 q=-5

float (réels : IR) exemples x=0.25 y=1e-5)

complex (nombres complexes) exemples Z=1+3j ou bien Z=complex(1,3)

- ***Chaines de caractère (String)***

str suite de caractères exemple :

str= " ceci mon premier programme en Python "

b) les types de données complexes (liste tuple set dictionary)

- ***listes***

Les listes sont utilisées pour stocker plusieurs éléments dans une seule variable.

Les listes sont l'un des 4 types de données intégrés à Python utilisés pour stocker des collections de données, les 3 autres sont Tuple, Set et Dictionary, tous avec des qualités et des utilisations différentes.

Les listes sont créées à l'aide de crochets :

Les éléments de la liste sont ordonnés, modifiables et autorisent les valeurs en double.

Les éléments de la liste sont indexés, le premier élément a un index [0], le deuxième élément a un index [1], etc.

Exemples de listes

List1= ["pomme", "banane", "orange"]

List2= [1, 2, 3]

List3=[1, 0.25,'vert'] # les élément ne sont pas forcément du même type.

On accède aux différents éléments d'une liste par :

Nom_liste[indice]

Exemple

print(List1[0]) → 'pomme' print(List2[2]) → 3 print(List3[-1]) → 'vert'

print() c'est une fonction qui permet d'afficher le résultat sur écran.

Pour les matrices on peut utiliser une liste à deux dimensions.

Exemples

A=[[1, 2, 3], [1, 2, 3],[3, 2, 1]]

Et on accède aux différents éléments de A par :

A[i][j] avec i et j sont respectivement les numéros de ligne et colonne ou bien A[i,j]

Nous verrons dans la partie réservée à la bibiothèque numpy comment faire les calculs qui peuvent nous intéressées dans ce cours sur les matrices.

-Tuples

Les tuples sont utilisés pour stocker plusieurs éléments dans une seule variable. Se sont des collections ordonnées et **immuables** (**on ne peut pas les modifier**) Ils sont identiques aux listes aux seules différences :

1) Les tuples sont écrits entre parenthèses
2) Ils sont immuables

Exemples

```
fruits= ("pomme", "banane", "orange")
nombre= (1, 2, 3)
divers=(1, 0.25,"vert") # les élément ne sont pas forcément du même type.
print(fruits[0])
print(nombre[2])
print(divers[-1])
```

On accède aux différents éléments d'un tuple par :
Nom_tuple[indice]
Exemple
print(fruits[0]) → 'pomme' print(nombre[2]) → 3 print(divers[-1]) → 'vert'
fruits[0]= "orange"→erreur car un tuple ne peut être modifié

Ensembles (**set**)

C'est une structure de donnée équivalente aux ensembles en mathématique. Cette structure n'est pas ordonnée et donc **non indicée**, elle permet la répétition des éléments.

Exemple de sets

S1={1,2,3} S2={"vert", "blanc"," rouge"}
Les ensembles (sets) sont beaucoup utilisés dans le calcul de probabilités.
I.1.2Les expressions dans Python
Les expressions sont identiques à matlab à quelques exceptions près qui seront indiquées ci-dessous.

b) Les expressions arithmétiques

Se sont l'addition (+) la soustraction (-)

la multiplication (*) la division (/) dans IR, dans Python il y a la division dans IN (//)

la puissance (**) : n**2 donne n à la puissance 2 (différente de matlab) on y accède aussi par la fonction pow(x,a) qui donne x^a.

le reste de la division: par exemple le reste de la division de n par 2 dans IN (n%2) (différente de matlab).

Les priorités (identiques à ce qui vous a été enseigné) qui peuvent être retenu par : PEMDAS

P : parenthèses E : Exposant M : Multiplication D : Division A : Addition S : Soustraction

Priorités= Priorités=

Sens des priorités du moins vers le plus élevé

Dans le cas de priorités égales (*,/) ou (+,-) les calcul se font de gauche à droite

c) Les opérateurs relationnels

Se sont les opérateurs de comparaison supérieur, inférieur etc…

>	supérieur	<	inférieur	>=	supérieur ou égal	<=	inférieur ou égal
= =	égale	! =	différent	=	affectation		

d) Les opérateurs logiques

Se sont le OU , le ET et le NON logiques. Ils sont exprimés par :

ET logique	OU logique	NON logique
and	**or**	**not**

```
a=1;b=0;c=0;
c=a>0 or b>0
d=a==1 and b<0
e=not d
print("c=",c)
print("d=",d)
print("e=",e)
résultats:
c= True
d= False
e= True
```

I.2 Instructions conditionnelles

Contrairement aux autres langages de programmation les instructions de contrôles (les instructions conditionnelles et répétitives) dans Python ne nécessitent pas des éléments explicites (visibles) qui marquent le début et la fin du bloc d'instructions tels que (begin end) et { }.

Le début et la fin d'un bloc d'instruction sont marqués dans python **implicitement** et ceci est expliqué en abordant les instructions conditionnelles **if, if – else,** if - **elif** - **else**.

Sytaxe de if simple

if condition**:**

: marque le début du bloc d'instruction

instruction 1
instruction 2
instruction 3

bloc d'instruction marqué par un retrait obligatoire par rapport au **if** (indentation)

Exemple

```
# resolution de l'équation ax=b
a,b=1,5
if a!=0:
    x=b/a
    print(x)
print('fin du programme')
```

Comme on peut mettre l'ensemble des instructions sur la même ligne en séparant les instructions par un point virgule (;)

```
# resolution de l'équation ax=b
a,b=1,5
if a!=0:      x=b/a;     print(x)
print('fin du programme')
```

if-else

Ceci représente le si – sinon

syntaxe if else

```
if      condition :
        instruction1 ou bloc d'instructions 1
else:
        instruction2 ou bloc d'instructions 2
```

Si la condition est vérifiée alors l'instruction1 (ou bloc d'instructions 1) sera exécutée.

Si la condition n'est pas vérifiée instruction1 ne sera pas et exécutée et instruction2 (ou l'ensemble d'instructions 2) après le else qui sera exécutée.

Remarque comme pour le if simple les bocs sont mis en retrait par rapport au if et else

exemple

```
# calcul de la valeur absolue d'un nombre reel
x=-5
if x>=0: print('la valeur absolue de x=',x, 'est', x)
else: print('|',x,'|','est',-x)
print("fin du calcul")
```

if elif else

Ceci représente : si condition1 instruction 1 sinon si condition 2 instruction 2 sinon instruction 3

Syntaxe

```
if condition 1 :
        instruction1
elif condition 2 :
        Instruction 2
else :
        Instruction3
```

Exemple résolution d'une équation du second degré : $ax^2+bx+c=0$

```
from math import * #pour l'utilisation de la fonction sqrt() : racine carre
a,b,c=1,1,1
print("a=",a,"b=",b,"c=",c)
delta=b*b-4*a*c

if delta>0:
    x1=(-b-sqrt(delta))/(2*a)
    x2=(-b+sqrt(delta))/(2*a)
    print("deux solutions distinctes",x1,x2)
elif delta==0:
    x=-b/(2*a)
    print("une solution double",x)
else:
    print("pas de solution dans IR")
print('fin du programme')
```

I.3 Les instructions Répétitives (Boucles)

Les instructions répétitives ont une importance majeure dans la programmation. Lorsque nous avons un même calcul qui se répète un certain nombre de fois "N " connu ou pas, on a recours aux instructions répétitives qui sont aussi connues sous le nom de boucles. Python dispose de deux boucles : **while** et **for**

La boucle for

Syntaxe

```
for variable in sequence :
    # bloc d'instruction à exécuter
```

Particularité de la boucle for sous Python

Dans les langages C, C++, Fortran, Matlab ou Pascal, la variable de la boucle for appelée aussi variable indice (index) doit être un nombre entier. Dans python ce n'est plus le cas, ça peut-être de tout type déjà présentés dans les paragraphes précédents, nous présentons ci-dessous quelques exemples.

Exemples ou la variable index est un entier :

```
#Exemple boucle for variable entier
vecteur=[1.5,2.6,13.8,1e-6]

n=len(vecteur)
for i in range(n):
   print(vecteur[i],end='\t')
   #on peut aussi ecrire sans l'utilisation de la boucle tout simplement par:
print('\n vecteur=',vecteur)
```

- Calcul de la somme

Exemple2

Soit à calculer la somme suivante : $S=\sum_{i=2,4,6}^{100}\frac{1}{i}$ (i dans cette exemple va prendre les valeurs paires de 2 jusqu'à 100)

algoritme

```
S←0

pour i=2 :100
    S←S+1/i
    i←i+2
```

```
#calcul de la somme S=somme(1/i),pour i=2,4 ...100
S=0;
for i in range(2,102,2): #102 ne sera pas prise
    S=S+1/i # la division se fait dans IR
print('i=',i,'S=',S)
```

- On veut dans cet exemple afficher les valeurs d'une liste de string

```
#exemple for liste de string

s='mohamed'
fruit=["banane" , "pomme", "fraise", "orange"]
for i in s:
   print(i,end='')
# comme on peut simplement faire:
print('\n',s)
for i in fruit:
        print(i)
divers=["dimanche", 2, 2.15, "fatima"]
for i in divers:
    print(i)
```

La boucle while (tant que)

La boucle **while** est utilisée, pour les instructions répétitives surtout quand le nombre de fois, où le calcul est répété, n'est pas connu à l'avance.

Syntaxe

```
while condition :
    instructions
```

Tant que la condition **est vérifiée** les instructions (après : marqué par le retrait) sont exécutées. Une fois que la condition n'est plus vérifiée, on sort de la boucle.

Exemple

On veut calculer la somme $S=1-\frac{1}{2}+\frac{1}{3}-\frac{1}{4}+\ldots\ldots$ avec une précision de 10^{-6}

programme

```
S=1;S0=0; i=2
while abs(S-S0)>1e-6: #abs la valeur absolue
    S0=S;
    S=S-(-1)**i/i;
    i=i+1;
print("S=",S)
```

résultat

i=1000002 S=0.6931476805592526

L'instruction break

Elle permet de sortir d'une boucle. Il arrive dans un programme où il faut sortir de la boucle même si elle n'est pas encore arrivée à sa fin. Si à titre d'exemple dans la boucle while de l'exemple précédent quand i dépasse une certaine valeur soit 1000 on souhaite sortir de la boucle même si la condition de while est toujours vérifiée. Pour faire on utilise l'instruction **break** :

Programme

```
S=1;S0=0; i=2
while abs(S-S0)>1e-6: #abs la valeur absolue
    S0=S;
    S=S-(-1)**i/i;
    i=i+1;
    if i>=1000: break;
print(f"i={i} S={S}")
```

résultat i=1000 S=0.6936474305598223

L'instruction continue

Elle permet de sauter une itération si une condition est vérifiée à l'intérieure d'une boucle.

Exemple

on veut calculer S=some(1/i) i=-10:10 en évitant la division par 0, pour faire on utilise l'instruction continue.

```
i=-11;S=0;
while i<10:
  i=i+1  #incréméntation de i juste après le while! Sinon boucle infinie (i==0 sans modification)
  if i==0:
        print("i==0");
        continue
  S=S+1/i;
```

I.4 Les fonctions et procédures – Variables locales – variables globales

I.4.1 Les fonctions

Le mot fonction englobe les fonctions intégrées (telles que print() abs() len()), les fonctions mathématiques telles que (abs() sin() etc ainsi que les fonction définies par l'utilisateur. Dans ce paragraphe nous allons voir comment développer ces fonctions.

Syntaxe d'une fonction définie par l'utilisateur :

```
def nom_fonction(paramètre1,paramètre2,.....) :
#corps de la fonction
        Instruction1
        Instruction2
        ……
        return valeur    #la valeur de retour return est facultative
```

Exemple

On veut transformer le programme déjà établi pour la résolution d'une équation du 2nd degrés sous forme d'une fonction :

```
1  from math import sqrt
2  def eq2(a,b,c):
3      delta=b**2-4*a*c
4      if delta >0:
5          x1=(-b-sqrt(delta))/(2*a)
6          x2=(-b+sqrt(delta))/(2*a)
7          print("deux solutions distinctes (x1!=x2)")
8          return x1,x2
9      elif delta==0:
10         x1=x2=(-b)/(2*a)
11         print("une solution double")
12         return x1,x2
13     else:
14         print("pas de solution dans IR") # return omise -->affichage none
15
16 #appel 1 de la fonction eq2 a=1 b=2 c=1
17 y=eq2(1,2,1) # y sera un tuple (x1,x2)
18 print("\t",y)
19 #appel 2 de la fonction eq2 a=1 b=1 c=1
20 y=eq2(1,1,1)
21 print("\t",y)
22 #appel 3 de la fonction eq2 a=1 b=0 c=-9
23 y=eq2(1,0,-9)
24 print("\t",y)
```

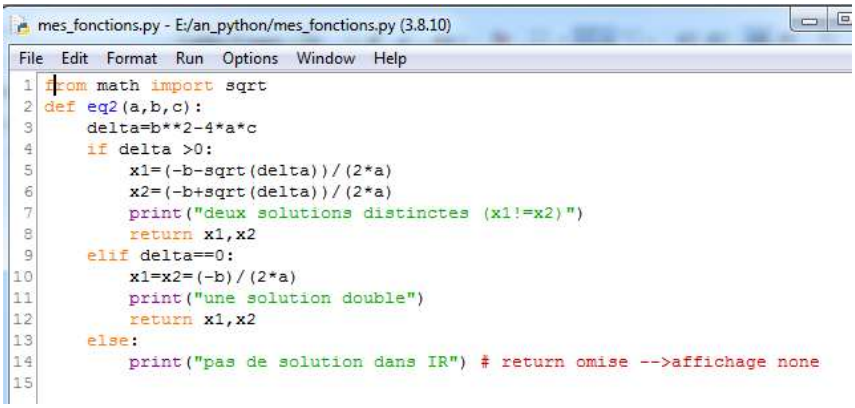
Résultats :

```
une solution double
        (-1.0, -1.0)
pas de solution dans IR
        None
deux solutions distinctes (x1!=x2)
        (-3.0, 3.0)
>>>
```

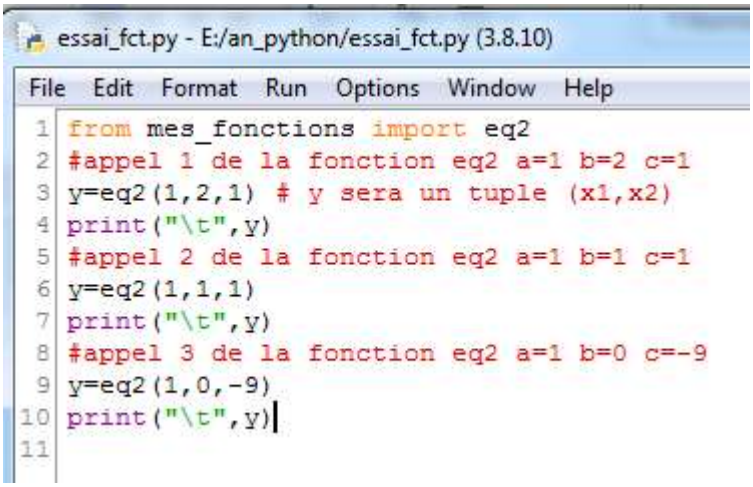
I.4.2 Les modules Python

Un module Python est un fichier ayant l'extension **.py** contenant du code python, généralement des fonctions. Les modules peuvent être utilisés à chaque fois que nous aurons besoin des fonctions qu'il contient. Essayons de voir comment procéder avec l'exemple précédant.

i) On crée un fichier (module) contenant la fonction eq2 on lui donne le nom mes_fonction.py



ii) On import la fonction eq2 dans le fichier du programme où nous avons besoin par l'instruction : from mes_fonctions import eq2



On a les mêmes résultats que précédemment.

```
>>>
====================== RESTART: E:/an_python/essai_fct.py =============
une solution double
        (-1.0, -1.0)
pas de solution dans IR
        None
deux solutions distinctes (x1!=x2)
        (-3.0, 3.0)
>>>
```

Nous aurons l'occasion de développer des fonctions et les mettre dans des modules au cours des travaux pratiques.

I.4.3 Variables Locales et Variables Globales

Toutes variables utilisées à l'intérieur d'une fonction a par défaut une portée locale. Ceci va permettre d'utiliser les mêmes noms de variables d'une fonction sans aucun conflit, par exemple les variables x et y à l'intérieur de la fonction f et le programme principal n'ont aucune relation.

Pour donner une portée globale à une variable, il faut utiliser le mot clé global à l'intérieur de la fonction.

Exemples

```
def f(x):
    a=2
    y=a*x;
    a+=10; # a est local
    print(f'dans f a={a} x={x} y={y}')
    return y

def g(x):
    global a
    a +=4  # a est global elle est modifiée par g(x)
    y=a*x**2-1;
    print(f'dans g a={a} x={x} y={y}')
    return y
a=1
print(f'dans principal valeur de a initiale a={a}')
#appel de la fonction f
y1=f(a) #a=1
#appel de la fonction g
y1=g(2)
print(f'dans principal valeur de a finale a={a}')
```

Résultats

```
dans principal valeur de a initiale a=1
dans f a=12 x=1 y=2
dans g a=5 x=2 y=19
dans principal a finale a=5
```

I.5 Les fichiers (Lectures et écritures)

Un fichier permet de stocker des informations sur une mémoire de masse (disque dur flash-disque etc). Il existe plusieurs types de fichiers mais nous nous limitons ici à l'étude des fichiers de type texte. Quel que soit le langage utilisé, les opérations concernant l'utilisation des fichiers texte sont les mêmes : tout fichier avant de le lire ou d'y écrire des données doit tout d'abord être ouvert. Une fois la tâche de lecture ou d'écriture est effectuée il doit être fermé.
En résumé les opérations suivantes sont indispensables lors de l'utilisation des fichiers texte :

Tâche	Interprétation en Python
Ouvrir (fichier) <en lecture ou en écriture>	f=open(nom__fichier.txt,rt) (r : read; w :write; t :texte)
lire ou écrire (fichier, X)	print(f.read()) print(var,file=f)
fermer (fichier)	f.close

Exemple

```
f1=open("essai.txt","rt") #il faut que le fichier soit dans le repertoire
                          #courant, sinon donner le chemail
f2=open("res.txt","w")
print(f1.read())
for i in range(10):
    #print(f2.write(f"{i}\n"))
    print(i,i**2,sep=' ',file=f2)
f1.close()
f2.close()
```

I.6 Graphisme

Pour pouvoir tracer des courbes en Python, on utilise la bibliothèque matplotlib. Pour faire, il faut d'abord l'installer en lançant la commande sous dos (système d'explotation de windows) :

```
C:\Users\Nae>pip install matplotlib
```

Dans le script, on import la bibliothèque matplotlib.pyplot, généralement, on lui donnant un alias (nom raccourci généralement plt).

```
import matplotlib.pyplot as plt #plt alias pyplot
```

Voir les détails et les exemples dans la section bibliothèques : matplotlib

I.7 Bibliothèques Numpy SciPy Matplotlib

Dans ce paragraphe, nous allons nous intéresser aux bibliothèques de Python les plus utilisées dans le calcul scientifique. Nous avons indiqué au préalable qu'il faut l'installer avec la commande pip sous MS-Dos. Dans ce qui suit, nous allons voir comment accéder aux différents modules et fonctions.

I.7.1 Numpy

Cette bibliothèque est dédiée aux calculs scientifiques et manipulations de tableaux multidimensionnels. Pour pouvoir l'utiliser, il faut l'importer. Souvent en utilisant l'alias np pour alléger l'écriture :

```
import numpy as np          #np alias numpy
```

Ce package permet la création de tableau (array) à une dimension (vecteurs) ou deux dimensions (matrices). Comme dans tous langages de programmation, ces tableaux doivent avoir le même type (int float …) Il offrent des fonctions permettant les différents calculs sur les tableaux.

Nous donnons des fonctions de bases dans ce qui suit et nous recommandons l'excellent site : **https://www.w3schools.com/python/numpy** pour un apprentissage plus approfondie.

Le fait d'importer le package en entier va nous obliger à chaque utilisation d'une de ses fonctions d'utiliser le préfixe (**np**) suivie (**.**) suivie du nom de la fonction (**array**) → np.array()

Par contre, si on rajoute l'instruction **from numpy import array,** on utilise la fonction array sans préfixe.

```
import numpy as np              #np alias numpy
from numpy import array
A=array([[1,1,1],[2,2,2]])
print("A=",A)
B=np.reshape(A,(3,2)) # la function reshape est accessible en utilisant le
préfixe np
print("B=",B)
#on peut aussi écrire:
C=A.reshape(6,1)
print("C=",C)
```

Résultats :

```
A= [[1 1 1]
 [2 2 2]]
B= [[1 1]
 [1 2]
 [2 2]]
```

C= [[1]
 [1]
 [1]
 [2]
 [2]
 [2]]
Exemple d'utilisation de numpy

Dans l'exemple ci-dessous, nous utilisons quelques fonctions de numpy, nous avons voir comment on effectue le produit matriciel, le calcul du déterminant et l'inverse d'une matrice donné.

```
import numpy as np
#création matrice A et vecteur b avec np.array
A=np.array([[1, 2,3],[1, 1,1],[3, 3,3]])
print("A=",A)
b=np.array([3,2,1])
print("b=",b)
# création par range
c=np.array(range(0,10,2))
print("c=",c)
d=b.reshape(3,1)
print('d=',d)
At=A.T  #transposé d'une matrice
print('At=',At)
bt=b.T
print("bt=",bt) #ne marche pas pour un vecteur!
#produit de deux matrices
pAAt=A.dot(At)
print('pAAt=',pAAt)
#Nb:depuis la version 3.5 le symbole @ est utiliser pour le produit matriciel
# C=A@B
C=A@At
print('C=',C)
# * est utiliser pour faire le produit élément par élement D=A*At
# donne un résultat diff
D=A*At; print("D=",D)
de=np.linalg.det(D)#calcul du determinant
x=np.linalg.solve(D,b) # résolution du sys linéaire [D][x]=[b]
D_1=np.linalg.inv(D)
print('\n')
print('D_1=',D_1)
#oubien
from numpy.linalg import inv
D2=inv(D)
print('\n')
print('D2=',D2)
x2=D_1.dot(b)
print('\n')
print('solution=')
print(x2)
print(x)
```

I.7.2 SciPy

On trouve dans ce package différentes fonctions mathématiques. On y trouve les fonctions pour le calcul d'intégral, l'interpolation polynômiale, de résolution de problèmes d'optimisation, comme pour les autres bibliothèques il faut l'installer par :

```
C:\Users\Nae>pip install scipy
```

N.B qu'il faut d'abord importer numpy avant d'importer scipy car cette dernière utilise numpy.

Exemple calcul d'intégral de la fonction [3]

Calcul de $\int_0^{\pi/2} \sin(x)\, dx$

```
#calcul de l'integral
import numpy as np
from scipy import integrate
resultat=integrate.quad(np.sin,0,np.pi/2)
print('type=',type(resultat))
print(f'integrale={resultat[0]}')
```

Résultas :

type= <class 'tuple'>

integrale=0.9999999999999999

I.7.3 Matplotlib :

Création de graphiques statiques et interactifs

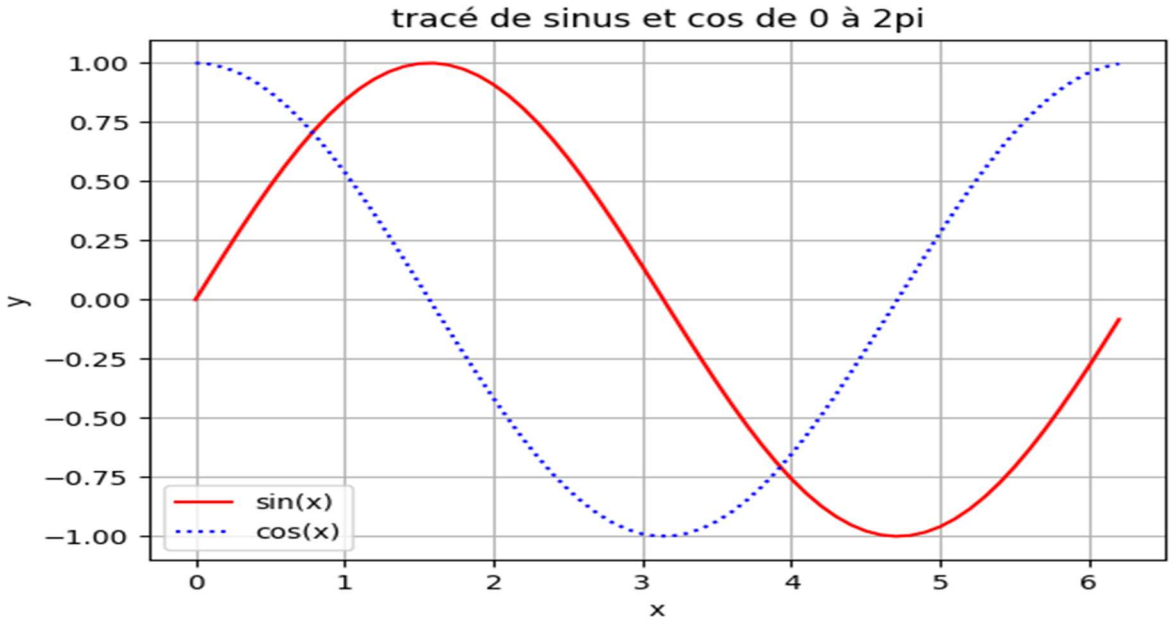
Il faut d'abord installer matplotlib, sous MS-DOS on tape la commande :

```
pip install matplotlib
```

Nous expliquons comment tracer une courbe à partir de l'exemple ci-dessous:

Exemple d'utilisation de matplotlib courbe 2D

```
import numpy as np
import matplotlib.pyplot as plt
x=np.arange(0,2*np.pi,0.1)
y1=np.sin(x) ;y2=np.cos(x)
plt.xlabel('x') ;plt.ylabel('y') ;plt.title('tracé de sinus et cos de 0 à
2pi')
plt.plot(x,y1,'r',x,y2,'b:')
plt.legend(['sin(x)','cos(x)'])
plt.grid() ;plt.show()
```



Nous pouvons constater que c'est les mêmes fonctions de Matlab à la seule différence de rajouter le préfixe **plt.,** il faut préciser d'introduire l'instruction `plt.show()` afin de faire apparaitre le graphique

I.8. Annexes (fonctions numpy les plus utilisées dans la matière)

np.arange(10) np.arange(0,10,2)	[0,1,2…….9] [0,2,4,6,8]	**range** n'existe pas dans numpy
b=np.array([1,2,3]) a=np.array([1,1,1],[2,2,2])	[1,2,3] a=[[1,1,1],[2,2,2]]	Génération de vecteur et matrice
a.shape	(2,3)	Donne le nombre de lignes et colonne
bt=b.reshape(1,3)	b=[1 2 3]	Transforme vecteur. Ligne en vecteur. Colonne
x=dot(b,b)	x=14	Produit scalaire : Attention (vecteur ligne*vecteur ligne il fait lui-même la transposition)
C=np.arra([3,1,-1]) y=cross(b,c)	y = [-5 10 -5]	Calcul du produit vectoriel de deux vecteurs
np.zeros((2,3))	[[0,0,0],[0,0,0]	Génère une matrice (2*3) éléments nuls
np.ones((3,3))	[[1.,1.,1.],[1.,1.,1.],[1.,1.,1.]]	Génère une matrice (3*3) éléments = unité(1)
np.eye(n)		Matrice unité d'ordre n
np.diag(A)		Renvoie un vecteur contenant la diagonae de A
np.triu		Renvoie la matrice triangulaire supérieure (diagonale comprise)
np.tril		Renvoie la matrice triangulaire inférieure (diagonale comprise)
np.linspace(0,1,11) np.linspace(0,1,10,endpoint=**T**rue)	[0,0.1,0.2……..1] [0,0.1,0.2……..1]	

np.dot(A,B) A@B A.dot(B)		Produit matriciel (AXB)
A.T		Calcul de la matrice transposée
np.linalg.inv(A)		Calcul de l'inverse de A (A^{-1})
np.linalg.solve(A,b)		Résout le système [A][x]=[b]
determinant=np.linalg.det(A)		Calcul le déterminant de A
P=np.poly1d([1,-2,1]) print(np.poly1d(p))	$P=1x^2-2x+1$	Donne la forme polynomiale
P(0.5)	0.25	Calcul la valeur de P pour x=0.5
np.roots(P)	Array([1.,1.])	Trouve les racines de P
np.polyadd(p1,p2)		Addition de deux polynômes p1 et p2
np.polysub		Soustration de p1-p2
np.polymul(p1,p2)		Produit de p1 et p2
np.polydiv(p1,p2)	Retourne: (Le résultat de la division, Le reste de la division)	Division Euclidienne p1/p2
x = np.array([0.0, 1.0, 2.0]) np.polyval(p,x)	Retourne un vecteur de p(x)	Calcul la valeur de P pour chaque valeur de x

Table des matières