Introduction à l'analyse numérique
Année : 2021-2022
Formation : L2 Mathématiques

TP1 : Initiation à la programmation en Python pour l'analyse numérique

Le but de ce TP est de rappeler quelques bases de Python et d'introduire les modules NUMPY et MATPLOTLIB de Python, qui seront indispensables pour ce module d'analyse numérique. Nous présenterons l'essentiel dans ce TP grâce à des exemples que vous devrez saisir afin de comprendre les commandes qui vous seront prÃ©sentÃ©es. Il se termine par des exercices.

1 Rappels de quelques bases de Python

1. Les variables sont définies très naturellement. Essayez par exemple les commandes suivantes :
In [1] :

```
a = 1.54
b = 1
c = 2 + 3j
print(a)
print(b)
print(c)
type(a)
type(b)
type(c)
```

La fonction type renvoie le type d'une variable (i.e. entier, réel, complexe, booléen...)

2. Opérateurs : Voici la liste des opérattions disponibles dans Python :

Opérateur	Opérations
=	Affectation de variable
+	Addition (ou concaténation) de variables
*	Multiplication
−	Soustraction
/	Division
//	Division entière
**	Exposant
%	Modulo
not or and	Opérations booléennes
< <= == != >= >	Comparaisons ($<$ $\leq$ $=$ $\neq$ $\geq$ $>$)

3. Quelques fonctions utiles

(a) **range()** génère des suites arithmétiques

(b) **list()** permet de convertir les résultats en liste.
Exemple : la commande ci-dessous

```
list(range(10))
```

génére une liste contenant tous les nombres entiers de 0 inclus à 9.

(c) La fonction **len()** permet de connaître la longueur d'une liste
Exemple :

```
T = ["alpha","beta","gamma"]
len(T)
```

(d) La fonction **input()** permet de lire une valeur/information que l'utilisateur a entré. Attention, par défaut, la fonction input() renvoie une chaîne de caractères. Il faut donc penser à convertir au type souhaité.
Exemple :

```
print("Quel age avez vous?")
age = input()
type(age)
age = int(input())
type(age)
```

4. Structures de contrôles

(a) **Expression conditionnelle**
La syntaxe est la suivante :
if condition1 **:**
    instructions
elif condition2 **:**
    instructions
elif condition3 **:**
    instructions
else :
    instructions

Exemple :

```
if a < 0 :
        print("a est negatif")
elif a >0 :
        print("a est positif")

else:
        print("a est nul")
```

(b) **Boucle for**
La syntaxe est la suivante :
for variable **in** iterable **:**
    instructions

Exemple :

```
for i in range(5):
        print(i)
```

(c) **Boucle while**
while condition **:**
instructions

Exemple :

```
i = 0
while i<5:
        i = i + 1
        print(i)
```

5. **Les fonctions**
La syntaxe Python pour définir une fonction est la suivante :

def nom_fonction(liste de paramètres) **:**
bloc d'instructions

Exemple :

```
def funct(x):
        return x**2

def funct2(n):
        print("le carre de n est",n)
```

6. **Les modules**
Pour importer un module, il faut utiliser **import**. Toutes les constantes et fonctions de votre module sont importées. Par contre, leur utilisation doit être précédée d'un préfixe. Par exemple pour le module math qui permet d'avoir accès aux fonctions mathématiques, on écrit :

```
import math
math.sqrt(2)
math.exp(2)
```

Une autre solution est d'utiliser la commande **from** et dans ce cas, nous n'avons pas besoin de préfixe pour les constantes et les fonctions importées :

```
from math import sqrt,cos,pi
sqrt(2)
cos(pi/4)
```

On peut aussi utiliser la commande from module import * qui importe toutes les fonctions du module.

2 Numpy

NumPy est une bibliothèque pour le langage de programmation Python permettant de manipuler des matrices ou des tableaux multidimensionnels ainsi que des fonctions mathématiques opérant sur ces tableaux. Elle est particulièrement adaptée pour faire du calcul scientifique.

Le chargement de la bibliothèque se fait habituellement avec l'alias np :

```
import numpy as np
```

2.1 Vecteurs et matrices

2.1.1 Création de vecteurs et de matrices

Un vecteur est une matrice à une ligne ou une colonne. Pour entrer un vecteur ou une matrice explicitement, on peut faire :

```
import numpy as np
A = np.array([ 7,9,8]) # creation d'un vecteur
T = np.array([[1, 2, 3], [4, 5, 6]])  # creation d'une matrice 3x2
```

On peut ensuite accéder aux éléments de la même façon qu'en mathématiques pour les matrices et vecteurs :

```
A[0]    # Premier element du vecteur A. Les indices commencent a 0.
T[0,1]
```

Pour créer une matrice de taille quelconque remplie de zéros ou de 1 :

```
A = np.zeros((3,3)) # renvoie un tableau 2D (matrice) de 3x3 zeros
A = np.ones(5) # renvoie un tableau 1D de taille 5 avec que des uns
```

On peut aussi générer une matrice identité :

```
A = np.eye(3) #  renvoie  la matrice Identite de taille 3 x 3
```

On peut aussi générer des vecteurs particuliers avec les fonctions arange et linspace :

```
np.arange(3, 10) # Entiers de 3 a 9
np.arange(0, 20, 2)  # Entier de 0 a 18 avec un pas de 2
np.linspace(0,1,10) #  On obtient un tableau de 10 elements allant  de 0 a 1
```

Remarquez que la fonction arange étend le range de Python (il s'agit de la contraction de array(range())).

2.2 Manipulations des éléments d'une matrice

Comme nous l'avons vu pour accéder aux éléments d'une matrice, il suffit de préciser leurs numéros de ligne et de colonne entre les crochets :

```
A = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
A[1,2]
```

Pour extraire une ligne ou une colonne d'une matrice, on peut faire :

```
l = A[0,:]   # l recoit la premiere ligne de la matrice A
c =  A[:,1] # c recoit la deuxieme colonne de la matrice A
```

Pour ajouter une nouvelle ligne et colonne à A :

```
v =np.array([[10,11,12]])
B = np.concatenate((A,v)) # Ajout du vecteur v a A
```

Pour obtenir la taille, la dimension d'une matrice ainsi que le nombre d'éléments :

```
[n,m] = np.shape(A)
np.ndim(A)
np.size(A)
np.size(A,0)
```

2.3 Opérations sur les matrices

Les opérations usuelles +,-,*,/ et ** agissent sur les matrices élément par élément à partir du moment ou les opérations sont bien définies :

```
A+ v
A**2
A + 1.5
A/2
A*A
```

Les fonctions usuelles s'appliquent aussi élément par élément sur les vecteurs et matrices :

```
np.cos(A)
np.exp(A)
np.sqrt(A)
```

La transposée est obtenue de la façon suivante :

```
A.T
```

Il est aussi possible de faire la multiplication matricielle et le produit scalaire entre deux vecteurs

```
B = A + 2
np.dot(A,B)
w = np.array([[1,2,3]])
np.dot(v,w)
```

Les fonctions sum et prod renvoient respectivement la somme et le produit des éléments du tableau (i.e vecteur, matrice)

```
np.sum(A)
np.prod(v)
```

Pour obtenir le déterminant ou l'inverse d'une matrice, il faudra utiliser le module **numpy.linalg** :

```
from numpy.linalg import *
a = np.array([[1, 2],
              [3, 4]])
det(a)
a = np.array([[1, 3, 3],
              [1, 4, 3],
              [1, 3, 4]])
inv(a)
```

D'autres fonctions sont disponibles sur `http://www.numpy.org/`.

2.3.1 Matplotlib

Le module matplotlib permet de faire des graphiques. Le chargement de la bibliothèque se fait habituellement avec l'alias plt :

```
import matplotlib.pyplot as plt
```

Pour tracer un graphique en deux dimensions, on utilise la commande plot :

```
import numpy as np
x = np.linspace(-10, 10, 50)
y = np.exp(x)
plt.plot(x,y,label="exp(x)")
plt.legend()
plt.title("Fonction exponentielle")
plt.xlabel("abscisses")
plt.ylabel("ordonnees")
```

Les commandes **xlabel()** et **ylabel()** permettent d'ajouter des labels sur les axes des abscisses et ordonnées. La commande **title()** permet d'ajouter un titre et **legend()** une légende Il est aussi possible de tracer plusieurs courbes de couleurs différentes sur une même figure :

```
import numpy as np
import matplotlib.pyplot as plt
x = np.linspace(0, 2*np.pi, 50)
y1 = np.cos(x)
y2 = np.sin(x)
plt.plot(x, y1, "r", label="cos(x)")
plt.plot(x, y2, "b--", label="sin(x)")
plt.axis("equal")
```

```
plt.xlim(-1.5,1.5)
plt.legend()
```

Les commandes **xlim()** et **ylim()** permettent de fixer les domaines des abscisses et ordonnées. La commande **axis("equal")** permet d'imposer une échelle identique sur les deux axes de coordonées.

3 Exercices de manipulations

1. La conjecture de Syracuse

Soit $\phi : \mathbb{N}^* \to \mathbb{N}^*$ l'application telle que $\phi(n) = \frac{n}{2}$ si n est pair et $\phi(n) = 3n + 1$ si n est impair. On appelle suite de Syracuse toute suite $(s_n)_{n\in\mathbb{N}}$ définie par son premier terme $s_0 \in \mathbb{N}^*$ et par la relation $\forall n \in \mathbb{N}$, $s_{n+1} = \phi(s_n)$.
La conjecture de Syracuse postule que $(s_n)_{n\in\mathbb{N}}$ finit toujours par atteindre la valeur 1 et répète le motif 1, 4, 2, 1, 4, 2 ... [1]

(a) Ecrivez une fonction `syracuse(a,n)` qui calcule le n-ième terme de la suite de Syracuse initiée par a.

(b) Combien de temps faut-il pour atteindre 1 et quelle est la valeur maximale de la suite lorsque $s_0 = 15$ et $s_0 = 127$? Modifiez la fonction précédente pour répondre à ces questions.

2. Mouvement brownien

L'objectif des deux exercices est d'étudier numériquement des phénomènes de marches aléatoires, qui sont - entre autres - un moyen d'approximer le mouvement brownien [2]

(a) **Marche aléatoire en une dimension.** On considère une particule dans un tube très fin. Cette particule va se déplacer entre chaque pas de temps dt aléatoirement vers la gauche ou la droite.

i. Ecrire une fonction `pile_ou_face()` qui renvoie aléatoirement les valeurs +1 ou -1. On utilisera pour cela le module random et la fonction random.uniform pour obtenir une valeur aléatoire entre 0 et 1.

ii. La position de la particule à l'instant initial t_1 est $x_1 = 0$. Sa position à l'instant $t_n = ndt$ est donnée par $x_n = x_{n-1}+$ `pile_ou_face()`. Ecrire une fonction qui prend un entier N en argument et envoie un tableau `position` donnant les positions de la particule aux instants successifs $t_1, \cdots, t_N$.

iii. Afficher le résultat de la fonction précédente.

iv. Ecrire une nouvelle fonction qui calcule les positions de la particule jusqu'à ce que celle-ci atteigne la position -30 ou +30, ou que le nombre d'itérations dépasse 10 000.

1. En dépit de la simplicité de son énoncé, cette conjecture n'a pas encore été prouvée. Elle mobilisa tant les mathématiciens américains durant les années 1960, en pleine guerre froide, qu'une plaisanterie courut selon laquelle ce problème faisait partie d'un complot soviétique visant à ralentir la recherche américaine (source Wikipedia).

2. Le mouvement brownien est une description mathématique du mouvement aléatoire d'une "grosse" particule immergée dans un fluide et qui n'est soumise à aucune autre interaction que des chocs avec les "petites" molécules du fluide environnant. Il en résulte un mouvement très irrégulier de la grosse particule, qui a été décrit pour la première fois en 1827 par le botaniste Robert Brown en observant des mouvements de particules à l'intérieur de grains de pollen d'une espèce de fleur sauvage nord-américaine. La description physique la plus élémentaire du phénomène est la suivante : entre deux chocs, la grosse particule se déplace en ligne droite avec une vitesse constante ; la grosse particule est accélérée lorsqu'elle rencontre une molécule de fluide ou une paroi. Ce mouvement permet de décrire avec succès le comportement thermodynamique des gaz, ainsi que le phénomène de diffusion. Il est aussi très utilisé dans des modèles de mathématiques financières. (source Wikipedia). Pour visualiser : https ://www.canal-u.tv/video/cerimes/le_mouvement_brownien.10217.

(b) **Marche aléatoire en deux dimensions.** La particule est sortie du tube et peut maintenant se déplacer dans quatre directions.

i. Ecrire une fonction `boussole` qui renvoie aléatoirement les valeurs [1,0], [-1,0], [0,-1] ou [0,1].

ii. Pour représenter les positions successives de la particule, on va utiliser deux tableaux : `X` pour les abcisses et `Y` pour les ordonnées. A l'instant initial t_1 la position de la particule est $X(0) = 0, Y(0) = 0$. Ecrire une fonction qui prend un entier N en argument et envoie les tableaux `X` et `Y` donnant les positions de la particule aux instants successifs $t_1, \cdots, t_N$.

iii. Afficher le résultat de la fonction précédente.