TP Python – 1

Révisions : suites et fonctions

L'objet de ce TP est d'une part de revenir sur les fondamentaux du langage Python (boucles, fonctions) et quelques algorithmes élémentaires (calculs de sommes, dichotomie) et, d'autre part d'illustrer les chapitres de révision sur les suites et fonctions.

1 Pour bien débuter

Commencer par créer un *dossier de travail* `TP` **sur le serveur** pour y **enregistrer très régulièrement** l'avancée de votre travail.

Quelques conseils :

1. structurez votre code en *cellules*, c'est-à-dire en zones de code exécutables indépendamment :

```
1 # %% Cellule 1 Script principal
2
3 # %% Cellule 2 Script principal
```

2. utilisez la `docstring` (entre `"""`) décrivant ce que fait le script, les fonctions ...et des commentaires (débutant par `#`) signalant les différentes zones du script.

2 Tableau Numpy

La bibliothèque `Numpy` introduit un nouveau type appelé `array`, semblable à une liste à la différence près que tous les éléments doivent être du même type (entier, flottant,...).

Ce type présente cependant de nombreux avantages notamment pour la manipulation de matrices ou de fonctions.

2.1 Création d'un tableau `Numpy`

Définir une matrice ou un vecteur

1. Pour définir un vecteur ou une matrice en entrant les coordonnées on utilise la commande `np.array([[...],[...]])` (voir l'exemple ci-dessous).
2. La commande `np.zeros([n,p])` permet de créer la matrice nulle de taille $n \times p$.
   La commande `np.ones([n,p])` permet de créer la matrice de taille $n \times p$ dont tous les coefficients valent 1.
   La commande `np.eye(n)` permet de créer la matrice identité de taille n.
3. La commande `np.linspace(a,b,n)` permet d'obtenir un vecteur de taille `n` dont les coordonnées sont régulièrement espacées de `a` à `b`.
4. La commande `np.arange(a,b,c)` permet d'obtenir un vecteur dont les coordonnées sont espacées de `a` à `b` par pas de `c`.

1. La commande `np.array([[1,1,0],[1,2,3],[0,1,0]])` crée la matrice $\begin{pmatrix} 1 & 1 & 0 \\ 1 & 2 & 3 \\ 0 & 1 & 0 \end{pmatrix}$.
2. La commande `np.linspace(2,12,5)` crée le vecteur $(2, 4.5, 7, 9.5, 12)$.
3. La commande `np.arange(0, 1.3,0.2)` crée le vecteur $(0, 0.2, 0.4, 0.6, 0.8, 1, 1.2)$.

Travail demandé

1. Étant donné deux réels $a < b$ et un entiers naturel n écrire une ligne de programme créant un vecteur, sous forme d'un tableau `numpy`, de n nombres régulièrement espacés entre a et b.
2. Étant donné deux réels $a < b$ et un entiers naturel n écrire une ligne de programme créant une liste de n nombres régulièrement espacés entre a et b (sans utiliser de commande `numpy`).

2.2 Manipulation d'un tableau `Numpy`

Manipulation des matrices

Soit `M` un tableau préalablement défini dans Python.

1. (a) La commande `M[i,j]` donne le coefficient de la i-ième ligne et j-ième colonne de `M`.
   (b) La commande `M[i,:]` donne la i-ième ligne de `M`.
   (c) La commande `M[:,j]` donne la j-ième colonne de `M`.

   Attention, en python la numérotation des lignes et des colonnes commence à zéro !
2. Soient M et N deux tableaux préalablement définis dans Python et `a` un réel.
   (a) `M+N` : somme des matrices (qui doivent donc être de la même taille).
   (b) `M*N` : produit coefficients par coefficients des matrices (qui doivent donc être de la même taille).
   (c) `M+a` : ajout de `a` à tous les coefficients de `M`.
   (d) `M*a` : multiplication par `a` de tous les coefficients de `M`.
   (e) `M**a` : passage à la puissance `a` de tous les coefficients de `M`.
   (f) `np.dot(M,N)` : produit matriciel de `M` et de `N` (à condition que les tailles des matrices le permettent).
   (g) `np.shape(M)` : taille de la matrice.
   (h) `np.transpose(M)` : transposée de la matrice.

Travail demandé

1. Créer la matrice A, de taille 6×6, ne contenant que des 5 (sans rentrer les coefficients à la main).

2. Créer la matrice B, de taille 6×6, ne contenant que des 5 sauf sur la diagonale qui contient des 6 (sans rentrer les coefficients à la main).
3. Calculer les produits AB et BA.

2.3 Avantages des tableaux Numpy

— L'un des avantages des tableaux est que la plupart des fonctions `numpy` peuvent prendre en argument un tableau.
Par exemple les commandes

```
X = np.linspace(0,1,6)
Y = np.cos(X)
```

donnent

```
X = [0,0.2,0.4,0.6,0.8,1]
Y = [cos(0),cos(0.2),cos(0.4),cos(0.6),cos(0.8),cos(1)]
```

Cela est très pratique lorsqu'on veut tracer la courbe représentative d'une fonction.

— Un autre avantage vient avec la librairie `numpy.linalg` qui permet de réalisé beaucoup d'opération algébrique sur les tableaux :

```
import numpy.linalg as al

a.inv(M) # inverse de M si inversible
al.matrix_power(M,n) # puissance d'une matrice
al.solve(A,B)} # résolution du système AX=B
```

Travail demandé

1. Soit $A = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$. Calculer avec Python A^n pour $n = 2, 3, 4$ et 5.
2. Conjecturer la forme de A^n et démontrer votre conjecture par récurrence.

3 Fonctions

Quand on écrit un programme sous Python, il est fréquent d'utiliser des fonctions. Elles peuvent avoir 0,1 ou plusieurs arguments (certains optionnels).
Par exemple, dans la définition de la fonction `nom_fonction` ci-dessous, les arguments `arg_1`,...,`arg_n` sont nécessaires et l'argument `arg_option1` est optionnel et vaudra 10 par défaut :

```
def ma_fonction(arg_1,...,arg_n, arg_option1 = 10):
    ...
    return resultat_1,...,resultat_p
```

Travail demandé

On considère la fonction f définie sur $\mathbb{R}$ par : $\forall x \in \mathbb{R}, \quad f(x) = (x-1)^2$.
On définit une suite (u_n) en posant : $u_0 \in \mathbb{R}$ et $\forall n \in \mathbb{N}, \quad u_{n+1} = f(u_n)$.

1. Programmer la fonction f.
2. Écrire une fonction `suite(n,u0)` qui prend en argument un entier naturel n et le terme initial u_0 et renvoie la valeur de u_n.
3. Modifier la fonction précédente pour qu'elle renvoie la liste $[u_0, \ldots, u_n]$.

4 Algorithmes à accumulateur

Pour calculer la somme des éléments d'une liste `L`, on se base sur une boucle et un « accumulateur » `S`. L'algorithme se décrit en mots de la façon suivante :

1. Initialiser l'accumulateur à 0
2. Boucler sur les éléments de la liste `L` et, dans la boucle, ajouter l'élément courant de la liste à l'accumulateur `S`.

Des codes Python typiques pour cela sont donc :

```
1 #Liste L de nombres, préexiste
2
3 S=0 #Accumulateur pour la somme
4 for i in range(len(L)): #indices pour L
5     S+L[i] #ou S+=L[i]
6 #A ce point S= la somme des éléments de L
```

```
1 S=0
2 for elt in L: #Possibilité propre à Python
3     S+=elt
```

```
1 S=0
2 i=0 #indices pour L
3 while i< len(L):
4     S+=L[i]
5     i+=1 #incrementation de l'indice
```

```
1 S=0
2 i=len(L) #indices pour L
3 while i>0: #En décrémentant
4     i=i-1 #décrementation de l'indice
5     S+=L[i]
```

Les algorithmes accumulateurs peuvent aussi être utilisés pour déterminer le minimum des éléments d'une liste de nombre `L` de la façon suivante :

1. initialiser `m` l'accumulateur à l'un des éléments de la liste ;

2. boucler sur les éléments de la liste `L` et, dans la boucle, tester si l'élément courant de la liste est inférieur à l'accumulateur `m`, si c'est le cas, affecter cette valeur à l'accumulateur.

Des codes Python typiques pour cela sont donc

```
#La liste L est composée de nombres, préexiste
m=L[0]  #Accumulateur pour le minimum
for i in range(len(L)):  #indices pour L
    if L[i] < m:
        m=L[i]
#A ce point m = minimum des éléments de L
```

```
m=L[-1]  #Accumulateur pour le minimum
i=len(L)#indices pour L
while i>0:
    i=i-1#On décrémente l'indice
    if L[i] <= m:
        m=L[i]
```

Travail demandé

1. Écrire une fonction qui prend en argument un entier naturel `N` et qui renvoie la valeur de $\sum_{n=0}^{N} u_n$ où (u_n) est la suite définie au paragraphe 3.
2. Écrire une fonction `min(n,g,a,b)` qui prend en argument un entier naturel non nul `n` et une fonction g définie sur $[a,b]$, qui créer le vecteur $\left(g\left(a+\frac{k(b-a)}{n}\right)\right)_{k\in[\![0,n]\!]}$ et en détermine la coordonnée minimum.
3. En déduire une valeur approchée du minimum sur $[0,5]$ de $x\mapsto f(x)-x$ (où f est la fonction définie au paragraphe 3).
4. Modifier le programme précédent pour qu'il renvoie également une valeur approchée d'une abscisse x où le minimum est atteint.

5 Affichage graphique

L'affichage de graphique nécessite d'importer un module spécifique :

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

Si `X` et `Y` sont des vecteurs (ou des listes) la commande :

```
plt.plot(X,Y)
plt.show()
```

trace `Y` en fonction de `X`.
Si f est une fonction préalablement définie dans Python, pour tracer la représentation graphique de f sur un intervalle $[a,b]$, on procède alors de la façon suivante :

```
Abs = ...   #vecteur avec bcp de point régulièrement espacés entre a et b
Ord = ...   # vecteur contenant les images des éléments de Abs
plt.plot(Abs,Ord)
plt.show()
```

Travail demandé

On considère la fonction et la suite définies au paragraphe 3.

1. (a) Tracer la représentation graphique de f sur $[0,3]$ et conjecturer ses variations.
   (b) Prouver ces conjectures.
   (c) Ajouter une ligne au script ci-dessus pour qu'il affiche également la droite d'équation $y = x$ sur le même graphique.
   Qu'en déduit-on sur les points fixes de f ?
2. Tracer la représentation graphique des 20 premiers termes de la suite (u_n) avec $u_0 = 3$.
3. Tracer la représentation graphique des 20 premiers termes de la suite $(\ln(u_n))$ avec $u_0 = 3$.
   Que peut-on conjecturer sur la nature de la suite ?

6 Pour aller plus loin...

On considère la fonction f définie sur $\mathbb{R}$ par : $\forall x \in \mathbb{R}, \quad f(x) = (x-1)^2$.
On définit une suite (u_n) en posant : $u_0 \in \mathbb{R}$ et $\forall n \in \mathbb{N}, \quad u_{n+1} = f(u_n)$.

1. On prend toujours $u_0 = 3$. Démontrer la conjecture précédente.
   Indication : on pourra commencer par montrer que $[3, +\infty[$ *est stable par* f *puis étudier la monotonie de la suite.*
2. On admet que f admet deux points fixes $0 < a < 1 < b < 3$.
   (a) Dans une nouvelle cellule, écrire une fonction `dicho(eps)` qui prend en argument un réel `esp`> 0 et détermine, par un algorithme de dichotomie, une valeur approchée de a à `eps` près.
   (b) Déterminer la valeur exacte de a et comparer avec `dicho(10**(-3))`.
3. Dans cette question on suppose que $u_0 \in [0,1]$ (pour les simulations numériques, on prendra $u_0 = 0.5$).
   (a) A l'aide de la représentation graphique des 100 premiers termes de la suite $(u_n)_n$, conjecturer la nature de (u_{2n}), de (u_{2n+1}) et de (u_n).
   (b) Tracer sur le même graphique les fonctions f, $f \circ f$ et la droite d'équation $y = x$ entre $[0,3]$. En déduire les points fixes de $f \circ f$ et ses variations.
   (c) Retrouver ces résultats par le calcul.
   (d) Montrer que $[0,1]$ est stable par $f \circ f$ et en déduire que (u_{2n}) et (u_{2n+1}) sont monotones.
   (e) Déterminer la limite de (u_{2n}) et (u_{2n+1}).
4. Étudier le cas $u_0 \in]1, b[$. On pourra étudier numériquement quelques cas ($b = \dfrac{3+\sqrt{5}}{2}$).