DS n° 03

Les codes en python doivent être commentés et les indentations dans le code doivent être visibles.

I Question de cours – Méthode d'Euler

Soit l'équation différentielle :

$$\forall t \in \mathbb{R}, \quad y'(t) = y(t) \qquad \text{avec} \qquad y(0) = 1$$

Ecrire une fonction `Euler` qui prend comme entrée une liste `t` de flottants (la liste des abscisses) et renvoie une liste `y` (la liste des ordonnées) qui contient les valeurs de la fonction y calculée en t_i à l'aide de la méthode d'Euler.

Solution 1.

```
def methode_euler(t):
    y = [0]*len(t)
    y[0] = 0
    for i in range(len(t)-1):
        y[i+1] = y[i]+(t[i+1]-t[i])*y[i]
    return y
```

II Exercice de TP – Résolution d'équations du second ordre

L'objectif est de résoudre les équations de type (E): $ax^2 + bx + c = 0$ où a, b et c sont des réels, (donc seront des flottants dans vos programmes).
Dans cet exercice, on suppose que $a \neq 0$.

1. Écrire une fonction `solution(a,b,c)` qui renvoie les solutions de (E): $ax^2 + bx + c = 0$ et précise la nature de ses solutions. Par exemple :

```
>>> solution(2,-6,4)
Deux solutions reelles  x1=1.0 et x2=2.0
>>> solution(4,-4,1)
Une solution double  x=0.5
>>> solution(1,-2,2)
Deux solutions complexes  x1=1+1j et x2=1-1j
```

Rappel : le nombre complexe i se code `1j`.

Solution 2.

```
from math import sqrt
def solution(a,b,c):
        delta=b**2-4*a*c
        if delta >0:
                x1=(-b+sqrt(delta))/float(2*a)
                x2=(-b-sqrt(delta))/float(2*a)
                return "Deux solutions reelles", x1,x2
        elif delta==0:
                x=-b/float(2*a)
                return "Une solution double" ,x
        else:
                x1=(-b+sqrt(-delta)*1j)/float(2*a)
                x2=(-b-sqrt(-delta)*1j)/float(2*a)
                return "Deux solutions complexes", x1,x2
```

2. Est-ce que la fonction `solution` renvoie toujours la bonne réponse ? Quels problèmes peuvent se poser ?

Solution 3. Le test `Delta==0` va poser problème. Si a, b, c sont des flottants, Δ est aussi un flottant, donc une valeur approchée.
Si Δ est non nul mais inférieur à la précision de Python, il sera évalué comme nul.
Si Δ est nul mais calculé à l'aide de flottants, il peut être évalué comme non nul.

III Exercice – Matrices semi-magiques

Une matrice carrée de taille n, $A = (a_{i,j})_{\substack{0\leqslant i\leqslant n-1\\0\leqslant j\leqslant n-1}}$ dans $\mathcal{M}_n(\mathbb{R})$ est dite "semi-magique" si :

$$a_{0,0} + a_{0,1} + \cdots + a_{0,n-1} = a_{1,0} + a_{1,1} + \cdots + a_{1,n-1} = \cdots = a_{n-1,0} + \cdots + a_{n-1,n-1}$$
$$= a_{0,0} + a_{1,0} + \cdots + a_{n-1,0} = a_{0,1} + a_{1,1} + \cdots + a_{n-1,1} = \cdots = a_{0,n-1} + \cdots + a_{n-1,n-1}$$

autrement dit si la somme sur chaque colonne et la somme sur chaque ligne donne toujours la même valeur. On note alors $\sigma(A)$ la valeur commune de ces sommes.
Dans cet exercice, les matrices seront des objets du type `array` de la bibliothèque `numpy`.

1. Soit $B = \begin{pmatrix} 8 & 2 \\ -17 & \sqrt{6} \end{pmatrix}$. Que renvoie `B[0,1]` ?

   Solution 4. 2

2. Ecrire une fonction `somme_ligne`, d'argument une matrice A et un entier i et qui renvoie la somme des coefficients de la $i$ème ligne de A.
   Ecrire de même une fonction `somme_colonne`, d'argument A et j, qui renvoie la somme des coefficients de la $j$ème colonne de A.

   Solution 5.

```python
def somme_ligne(A,i):
    n=len(A[0])
    somme=0
    for j in range(n):
        somme=somme+A[i,j]
    return(somme)

def somme_colonne(A,j):
    n=len(A[0])
    somme=0
    for i in range(n):
        somme=somme+A[i,j]
    return(somme)
```

3. Ecrire une fonction `test`, d'argument A, renvoyant la valeur $\sigma(A)$ si A est semi-magique et `False` sinon.

   Solution 6.

```python
def test(A):
    n=len(A[0])
    sigma=somme_ligne(A,0)
    numero_ligne=1
    while numero_ligne<n and somme_ligne(A,numero_ligne)==sigma:
        numero_ligne=numero_ligne+1
    numero_colonne=0
    while numero_colonne<n and somme_colonne(A,numero_colonne)==sigma:
        numero_colonne=numero_colonne+1
    if numero_colonne==numero_ligne and numero_ligne==n:
        return(sigma)
    else:
        return(False)
```

4. Montrer que la complexité de la fonction `test` en $O(n^2)$ où n est la taille de la matrice.
   Indication : On se placera dans le pire des cas.

Solution 7. Première constatation : pour savoir si une matrice est magique, il faut connaitre toutes les cases de la matrice. Donc, a minima, on va parcourir les termes n^2 de A : la complexité est au minimum en $O(n^2)$.
Plus précisément :
la complexité des deux fonctions `somme_ligne` et `somme_colonne` est : $2n + 2 = O(n)$.
Dans la fonction `test`, dans le pire des cas, on entre n fois dans la première boucle `while`. Cela veut dire qu'on effectue n fois le test `somme_ligne(A,numero_ligne)==sigma`. Chaque test est en $O(n)$. Donc chaque boucle `while` est en $nO(n) = O(n^2)$.
La complexité est donc : $1 + O(n) + 1 + O(n^2) + 1 + O(n^2) + 3 = O(n^2)$.

$\boxed{\text{La complexité est en } O(n^2).}$