TD : Matrices et graphes

On rappelle qu'il est nécessaire d'importer la bibliothèque numpy.linalg pour utiliser les commandes relatives aux opérations sur les matrices et numpy pour les array
import numpy.linalg as al
import numpy as np

Pour les exercices sur les graphes, travailler avec votre cours car il y a besoin des définitions et des résultats.

1. Soient $(u_n)_{n\in\mathbb{N}}, (v_n)_{n\in\mathbb{N}}$ et $(w_n)_{n\in\mathbb{N}}$ les suites réelles définies par $u_0 = 0, v_0 = 1$ et $w_0 = 2$ et pour tout entier n :

   $\begin{cases} u_{n+1} = u_n + v_n + w_n \\ v_{n+1} = v_n + w_n \\ w_{n+1} = w_n \end{cases}$ De plus, on note $C_n = \begin{pmatrix} u_n \\ v_n \\ w_n \end{pmatrix}$

   (a) Déterminer une matrice A telle que pour tout entier n, $C_{n+1} = A \cdot C_n$. En déduire que $C_n = A^n C_0$ (à faire sur papier).
   (b) Écrire un programme Python affichant la matrice C_n où n est rentré par l'utilisateur.
   (c) Compléter le programme précédent pour qu'il affiche u_n.

2. * Que dois afficher la fonction suivante lorsqu'on rentre une matrice d'adjacence A ?

```
def mystere(A) :
    S=0     # S est un simple compteur
    n=np.shape(A)[0]            #le nombre de ligne de A
    for i in range(0,n):
        for j in range(0,n):
            if i==j:
                if A[i,j]!=0:     # != veut dire different de
                    S=S+1
            else :
                if A[I,j]>1:
                    S=S+1
    if S==0:
        return 'le graphe est…'
```

3. Soit $A \in \mathcal{M}_n(\mathbb{R})$. Pour tout $p \in \mathbb{N}^*$, on pose $S_p(A) = I_n + A + A^2 + \cdots + A^p$
   (a) Écrire une fonction def S(p,A) qui prend en arguments un entier p et une matrice carrée A et qui renvoie $S_p(A)$.
   (b) Comment utiliser cette fonction pour savoir si un graphe A est connexe ou non (on ne demande pas d'écrire un programme) ?

4. Soit G un graphe de matrice d'adjacence A. On suppose que G est **simple**.
   (a) Comment obtenir le degré d'un sommet i à l'aide de la matrice d'adjacence ? En déduire une fonction def deg(A,i) qui prend en argument une matrice

d'adjacence A d'un graphe **simple**, un indice de sommet i et qui renvoie d_i, le degré du i-ème sommet.

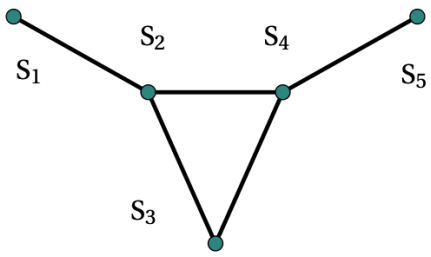
(b) Qu'affiche le programme suivant :

```
n=np.shape(A)[0]          #le nombre de ligne de A, donc l'ordre du graphe
for i in range(1,,n+1):
    S=S+deg(A,i)
print(S/2)
```

(c) * En déduire un procédé qui prend en argument une matrice d'adjacence A d'un graphe connexe et indique si le graphe est eulérien ou non.

5. *Soit $\mathcal{G}$ un graphe non-orienté connexe d'ordre n.
Pour S_i, S_j deux sommets distincts de $\mathcal{G}$, on définit la distance entre S_i et S_j, notée $d(S_i, S_j)$ comme la plus petite longueur parmi les chaînes reliant S_i et S_j. On définit de plus le diamètre du graphe comme la plus grande distance parmi les sommets du graphe.

1 Exemple. Le graphe du taureau



Compléter le tableau qui précise les distances entre les sommets. En déduire le diamètre du graphe.

	S_1	S_2	S_3	S_4	S_5
S_1					
S_2	–				
S_3	–	–			
S_4	–	–	–		
S_5	–	–	–	–	

2 Cas général

(a) Justifier que le diamètre est toujours inférieur à $n-1$.

(b) Prouver que le diamètre est la plus petite valeur entière d telle que $I_n + A + A^2 + \cdots + A^d$ soit une matrice dont tous les coefficients sont strictement positifs.

(c) En déduire une fonction qui prend une matrice d'adjacence d'un graphe connexe et renvoie son diamètre.

Corrigé

1. Voici le programme :
```
n=int(input('rentrer l'entier n souhaité'))
A=np.array([[1,1,1],[0,1,1],[0,0,1]])        # on rentre A
C=np.array([0,1,2])                          # on rentre C0
B=al.matrix_power(A,n)      # on calcule A**n
C=np.dot(B,C)               # on calcule C_n = A^n C_0
print(C[0])                 # affiche un qui est le premier coefficient de C_n
```

2. Ce programme compte le nombre de zéro (avec la variable S) que contient la matrice A en dehors de sa diagonale. Si elle n'a pas de zéro c'est que le graphe est complet (tous les sommets sont adjacents).

3. (a) def S(p,A) :
```
    S= al.matrix_power(A,0)
    for k in range(1,p+1):
        S=S+al.matrix_power(A,k)
    return S
```
   (c) D'après le cours (condition nécessaire et suffisante de connexité) il suffit de regarder si S(n-1,A) contient des zéros ou non (s'il y a des zéros la matrice n'est pas connexe)

4. (a) def deg(A,i)
```
    d=0
    n=np.shape(A)[0]    # nombre de ligne de A donc l'ordre du graphe
    for k in range(0,n):
        d=d+A[i-1,k]  # compte les 1 sur la colonne i
    return d
```
   (b) Ce programme calcule la somme des degrés et renvoie la moitié, donc, d'après la formule d'Euler il calcule le nombre d'arêtes.
   (c) Il suffit de regarder si tous les degrés sont pairs (résultat de cours).

5. 2(a) Le graphe est connexe et d'ordre n donc tous les sommets sont reliés par une chaîne, et la chaîne de longueur minimale reliant deux sommets a au maximum n-1 arêtes.
   (b) Tant qu'il y a un zéro sur cette matrice cela signifie qu'il existe deux chaînes dont la distance est plus grande que d, donc le diamètre est bien le plus petit d tel que la matrice n'a aucun zéro.
   (c) def diam(A) :
```
    n=np.shape(A,0) ; k=1
    while prod (S(k,A))==0:          #fonction définie dans l'ex 3
        k+=1
    return k
```