Devoir surveillé n°5

Devoir du **samedi 14 mars 2026**.

L'usage de documents de cours et d'appareils électroniques est interdit.
Consignes de présentation : Écrire avec une encre bleue sombre ou noire. Encadrer les résultats. Ne pas utiliser la couleur rouge. Écrire de façon lisible. Éviter les ratures. **Utiliser une règle**.
Consignes de rédaction : Rédiger les raisonnements en français, dans une langue correcte et soignée, sans utiliser d'abréviation.
On rappelle que le connecteur $\Longrightarrow$ ne peut pas être utilisé comme synonyme de "donc".

Exercice 1 *Un exercice proche du cours*

On considère la fonction f définie sur $\mathbb{R}_+$ par : $f(x) = \begin{cases} xe^{-1/x} & \text{si } x > 0 \\ 0 & \text{si } x = 0 \end{cases}$

1. (a) Calculer $\lim\limits_{x\to 0^+} f(x)$ et en déduire que f est continue à droite en 0.

   (b) Calculer $\lim\limits_{x\to 0^+} \dfrac{f(x)}{x}$ et en déduire que f est dérivable à droite en 0 et donner le nombre dérivé de f en 0, noté $f'_d(0)$.

2. (a) Déterminer, pour tout réel x de $\mathbb{R}_+^*$, l'expression de $f'(x)$ en fonction de x où f' désigne la fonction dérivée de f.

   (b) Étudier le signe de $f'(x)$ sur $\mathbb{R}_+^*$ puis donner les variations de f sur $\mathbb{R}_+$.

   (c) Calculer les limites de f aux bornes de son domaine de définition puis dresser le tableau de variations de f.

   (d) Est-ce que f est deux fois dérivable en 0 ? Justifier.

   (e) Vérifier que pour tout réel x de $\mathbb{R}_+^*$, on a $f''(x) = \dfrac{1}{x^3}e^{-1/x}$.
   La fonction f est-elle convexe ou concave sur $\mathbb{R}_+$?

3. (a) Calculer $\lim\limits_{u\to 0^+} \dfrac{e^{-u}-1}{u}$.

   (b) En déduire que $\lim\limits_{x\to+\infty} (f(x) - (x-1)) = 0$.

   (c) On note $(\mathcal{C})$ la courbe représentative de f dans un repère orthonormé. Donner l'équation de la droite asymptote à $(\mathcal{C})$ au voisinage de $+\infty$ et tracer l'allure de $(\mathcal{C})$.

On définit la suite $(u_n)_{n\in\mathbb{N}}$ par la donnée de son premier terme $u_0 = 1$ et par la relation de récurrence $u_{n+1} = f(u_n)$, valable pour tout entier naturel n.

4. (a) Montrer que pour tout entier naturel n, u_n existe et $u_n > 0$.

   (b) Montrer que la suite $(u_n)_{n\in\mathbb{N}}$ est décroissante.

   (c) En déduire que la suite $(u_n)_{n\in\mathbb{N}}$ converge et donner sa limite.

   (d) Compléter les commandes suivantes pour qu'elles affichent le rang n à partir duquel $u_n \leq 10^{-3}$.

```
n = 0
u = 1
while u > 0.001 :
    u = -----
    n = -----
print(n)
```

5. (a) Montrer que, pour tout entier naturel n, on a la relation $\displaystyle\sum_{k=0}^{n} \frac{1}{u_k} = -\ln u_{n+1}$.

(*b*) En déduire que la série de terme général $\dfrac{1}{u_n}$ est divergente.

Exercice 2 *Autour des applications contractantes*

1. Donner un exemple, d'une fonction f de $\mathbb{R}$ dans $\mathbb{R}$ pour laquelle il existe un réel K élément de $]0,1[$ tel que, pour tout couple (x,y) de réels, on ait :

$$|f(x)-f(y)| \leq K \times |x-y| \qquad (\bigstar)$$

On considère pour toute la suite une fonction f vérifiant la condition précédente pour un réel $K \in]0,1[$. On dit que f est K-contractante.

2. Montrer que f est continue sur $\mathbb{R}$.

3. À l'aide de la relation $(\bigstar)$, montrer par l'absurde que l'équation $f(x)=x$ admet au plus une solution.

4. On considère la suite $(u_n)_{n\in\mathbb{N}}$ définie par la donnée du réel u_0 et la relation de récurrence $u_{n+1}=f(u_n)$, valable pour tout entier naturel n.

   (*a*) Montrer par récurrence que, pour tout entier naturel n, on a

$$|u_{n+1}-u_n| \leq K^n \times |u_1-u_0|$$

   (*b*) Établir la convergence de la série de terme général $u_{n+1}-u_n$, puis en déduire que la suite $(u_n)_{n\in\mathbb{N}}$ est convergente. On note a sa limite.

   (*c*) Conclure que l'équation $f(x)=x$ admet une unique solution.

5. On désigne par n et p des entiers naturels (avec $p \geq 1$).

   (*a*) Justifier que l'on a : $\displaystyle\sum_{i=n}^{n+p-1} |u_{i+1}-u_i| \leq \sum_{i=n}^{n+p-1} K^i \times |u_1-u_0|$

   (*b*) En déduire l'inégalité :

$$|u_{n+p}-u_n| \leq K^n \times \frac{1-K^p}{1-K}|u_1-u_0|$$

   (*c*) Établir enfin l'inégalité suivante :

$$|a-u_n| \leq \frac{K^n}{1-K} \times |u_1-u_0|$$

6. Étude d'un exemple : on considère la fonction f définie par :

$$\forall t \in \mathbb{R},\ f(t)=\frac{1}{1+e^t}$$

   (*a*) Justifier que f est de classe C^2 sur $\mathbb{R}$ puis calculer $f'(t)$ et $f''(t)$, pour tout réel t.

   (*b*) Déterminer les variations de f' sur $\mathbb{R}$ et établir enfin l'inégalité :

$$\forall t \in \mathbb{R},\ |f'(t)| \leq \frac{1}{4}$$

   (*c*) En déduire que f est $\dfrac{1}{4}$-contractante.

   (*d*) On considère la suite $(u_n)_{n\in\mathbb{N}}$ définie par la donnée du réel $u_0=0$ et par la relation de récurrence $u_{n+1}=f(u_n)$, valable pour tout entier naturel n. Montrer que cette suite est convergente. On note toujours a sa limite.

   (*e*) Compléter la fonction Python ci-dessous afin qu'elle renvoie, pour une valeur donnée de n, la valeur de u_n à l'appel de `suite(n)` :

```
def suite(n) :
    u = -----
    for k in range(1,n+1) :
        u = -----
    return u
```

(f) En s'appuyant sur le résultat de la question 5c), établir que u_n est une valeur approchée de a à moins de 10^{-3} près dès que n vérifie $4^n > 2000/3$.

(g) En déduire un programme Python, utilisant la fonction précédente, qui calcule et affiche la valeur approchée de a qui en résulte.

Exercice 3 *Graphes et réseaux sociaux*

Pour analyser un réseau social, on peut former son graphe dans le but d'utiliser l'outil informatique pour avoir des résultats. Dans cet exercice, on considérera un réseau social "symétrique" : deux individus sont ou bien amis, ou bien non amis. On considérera de plus qu'un individu n'est pas ami avec lui même.

Le graphe d'un tel réseau social est construit de la manière suivante :

- Les sommets de ce graphe sont les individus du réseau social (les différents comptes).
- Deux sommets i et j sont reliés par une arête si et seulement si i et j sont amis.

On rappelle qu'un graphe est dit ***simple*** s'il est sans boucle (et sans multi-arête). Le graphe associé à un tel réseau social est donc simple et non orienté. Deux sommets d'un graphe sont dits ***voisins*** s'il existe une arête entre ces sommets.

Dans tout cet exercice, les graphes considérés sont simples et non orientés.

Dans les questions d'informatique, on supposera les sommets des graphes considérés numérotés à partir de 0, et on pourra donc parler de ***la*** matrice d'adjacence d'un graphe. On identifiera librement les sommets à leur numéro. On aussi supposera le module `numpy` importé à l'aide de la commande :

```
import numpy as np
```

Une annexe informatique est présente en fin d'exercice, contenant des rappels sur les commandes Python.

Partie I : Théorie des graphes et modélisation

Soit G un graphe non orienté. Notons n son ordre, et numérotons $s_1, \dots, s_n$ ses sommets.

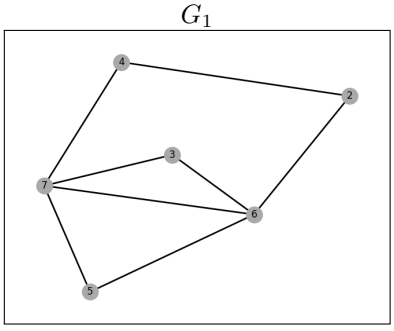
1. (a) Définir la matrice d'adjacence M de G associée à la numérotation $s_1, \dots, s_n$ des sommets de G.

   (b) Rappeler sans démonstration le lien entre M^k et les chaînes de longueur k de G (pour $k \in \mathbb{N}$).

   (c) Démontrer que si deux sommets s_i et s_j de G sont reliés par une chaîne, alors ils sont reliés par une chaîne de longueur au plus $n-1$ (où $1 \leq i, j \leq n$).

2. Déterminer les ensembles S et A tels que le graphe représenté ci-dessous, noté G_1 dans la suite, soit le graphe (S, A).

G_1



3. Soit $n \in \mathbb{N}$. On appelle graphe complet d'ordre n le graphe K_n dont les sommets sont les entiers de 0 à $n-1$, et tel que :

- K_n est un graphe simple non orienté, et
- deux sommets distincts de K_n sont toujours reliés par une arête.

(a) Déterminer la matrice d'adjacence M de K_4.

(b) Écrire le code d'une fonction `Python` d'entête `def K(n):` prenant en entrée un entier `n` et renvoyant en sortie la matrice d'adjacence de K_n.

(c) On pose $J = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix}$. Calculer J^k pour tout entier naturel k.

(d) Montrer que M est combinaison linéaire de J et de la matrice identité I_4.

(e) En déduire le calcul de M^k pour tout entier naturel k, à l'aide de la formule du binôme de Newton.

Une manière assez naïve d'analyser un réseau social afin d'en dégager des sources d'influence est de considérer que plus un individu a d'amis, plus il est influent.

4. Écrire le code d'une fonction `Python` d'entête `def degre(M,i):` prenant en entrée la matrice d'adjacence `M` d'un graphe et le numéro `i` de l'un de ses sommets et renvoyant en sortie le degré de ce sommet.

5. En déduire la code d'une fonction d'entête `def listeDegMax(M):` prenant en entrée la matrice d'adjacence `M` d'un graphe et renvoyant en sortie la liste des numéros des sommets de ce graphe dont le degré est maximal.

Partie II : Ensemble dominant de sommets

Dans le but de répandre une fake-news (ou, de manière équivalente, une publicité) sur les fils de publication, on peut chercher à identifier certains individus tel que : tout individu du réseau social est ami avec au moins un des individus identifiés. En termes de théorie des graphes, cela donne la définition suivante :

Définition 1. Soit G un graphe non orienté, notons S l'ensemble de ses arêtes. On dit qu'un ensemble de sommets $D \subset S$ est ***dominant*** si tout sommet de G est ou bien élément de D, ou bien voisin d'un élément de D.

Dans un objectif d'efficacité, on cherche à déterminer un ensemble de sommets dominant ayant le moins d'éléments possibles (au sens où s'il on retire un élément de cet ensemble de sommets, il n'est plus dominant). On dira dans ce cas que l'ensemble dominant trouvé est minimal.

6. Déterminer un ensemble de sommets dominant minimal pour le graphe G_1 de la partie I (on démontrera le caractère minimal de celui-ci).

7. Écrire une fonction `Python` d'entête `def estVoisin(M,i,j):` prenant en entrée la matrice d'adjacence `M` d'un graphe et les numéros `i` et `j` de deux de ses sommets, et renvoyant en sortie `True` si ces sommets sont voisins dans ce graphe, et `False` sinon.

8. Que fait la fonction `Mystere` suivante, prenant en entrée la matrice d'adjacence `M` d'un graphe, le numéro `j` de l'un de ses sommets et une liste `A` constituée de (numéros de) sommets de ce graphe ?

```
def Mystere(M,j,A):
    if j in A:
        return True
    for a in A:
        if estVoisin(M,j,a) :
            return True
    return False
```

9. En déduire une fonction d'entête `def estDominant(M,A):` prenant en entrée la matrice d'adjacence `M` d'un graphe et une liste `A` constituée de sommets de ce graphe, et renvoyant en sortie `True` si `A` est la liste des sommets d'un ensemble dominant, et `False` sinon.

10. En déduire une fonction d'entête `def minimiseDominant(M,A):` prenant en entrée la matrice d'adjacence `M` d'un graphe et la liste `A` des sommets d'un ensemble dominant, et renvoyant en sortie :

- Une liste `L` obtenue à partir de `A`, en retirant un de ses éléments de sorte que `L` soit toujours la liste des sommets d'un ensemble dominant, si c'est possible,
- `False` si aucun sommet ne peut être enlevé à la liste `A` sans obtenir une liste de sommets d'un ensemble dominant.

La fonction `minimiseDominant` ne devra pas modifier la liste A.

11. Expliquer le code suivant, dans lequel `M` est la matrice d'adjacence d'un graphe.

```
n,p = np.shape(M)
L = [i for i in range(n)]
while minimiseDominant(M,L) != False:
    L = minimiseDominant(M,L)
print(L)
```

Partie III : Centralité de proximité

On peut détecter des sources d'influence avec un indicateur appelé le degré de proximité. L'idée est de dire que plus un individu est proche d'un grand nombre d'individus, plus il est influent.

Dans cette partie, on fixe un graphe G d'ordre n. On suppose que l'ensemble des sommets de G est $[\![0, n-1]\!]$ et on note `M` sa matrice d'adjacence. Si i et j sont deux sommets de G, on note $d(i,j)$ la longueur de la plus petite chaîne reliant les sommets i et j si une telle chaîne existe, et on pose $d(i,j) = +\infty$ si i et j ne sont pas reliés par une chaine. En particulier, $d(i,i) = 0$.

On dit que $d(i,j)$ est la distance du sommet i au sommet j.

Enfin, pour tout sommet i, on pose :

$$D(i) = \sum_{\substack{j=0 \\ j\neq i}}^{n-1} \frac{1}{d(i,j)}$$

avec la convention $\dfrac{1}{+\infty} = 0$.

On dit que $D(i)$ est le degré de proximité du sommet i.

12. (a) On se place dans le cas du graphe G_1 de la partie I. Déterminer $D(2)$ sans justification.

    On revient au cas général.

    (b) Montrer que pour tout sommet i, $0 \leq D(i) \leq n-1$.

    (c) Que dire du sommet i si $D(i) = 0$? Que dire si $D(i) = n-1$?

    (d) Soient i, j et k trois sommets de G.

    Démontrer que $d(i,j) \leq d(i,k) + d(k,j)$ avec la convention $+\infty \leq +\infty$ et $x \leq +\infty$ pour tout réel x.

13. À l'aide des résultats de la partie I, écrire une fonction d'entête `def dist(M,i,j):` prenant en entrée la matrice `M` et les numéros `i` et `j` de deux sommets de G, et renvoyant en sortie $d($`i`$,$`j`$)$ si ces sommets sont reliés par une chaîne, et -1 sinon. On justifiera que le code proposé est correcte.

14. En déduire le code d'une fonction d'entête `def D(M,i):` prenant en entrée la matrice `M` et le numéro `i` d'un des sommets de G, et renvoyant en sortie le degré de proximité $D($`i`$)$ du sommet `i`.

15. Écrire le code d'une fonction d'entête `def ClassementCentralite(M):` prenant en entrée la matrice `M` et renvoyant en sortie la liste des sommets de G triée par degré de proximité décroissant.

    On pourra utiliser la méthode du tri à bulles.

Partie IV : Détection de cliques

Définition 2. Soit $G = (S, A)$ un graphe non orienté. On dit qu'un ensemble de sommets $C \subset S$ est ***une clique*** si pour tous sommets i et j distincts et appartenant à C, ces sommets sont voisins.

Dans un graphe, une clique est donc la donnée de sommets qui sont "tous voisins entre eux". Identifier les cliques du graphe d'un réseau social permet de détecter des communautés (des groupes d'individus partageant un même intérêt).

On rappelle que le sous graphe induit par un ensemble S' de sommets d'un graphe G est le sous-graphe obtenu à partir de G en ne gardant que les sommets de S' et toutes les arêtes de G entre les sommets de S'.

16. Déterminer une clique de cardinal 3 du graphe G_1 de la partie I. Donner sans justification la matrice d'adjacence du sous-graphe induit par cette clique. Que remarque-t-on?

17. Écrire le code d'une fonction d'entête `def gardeLignes(M,L):` prenant en entrée une matrice `M` et une liste `L` formée d'entiers distincts entre 0 et $n-1$, où n est le nombre de lignes de `M`, et renvoyant en sortie la matrice obtenue à partir de `M` en ne gardant que les lignes dont les indices sont présents dans `L`.

    Par exemple, si `M` est la matrice $\begin{pmatrix}1&2&3&4\\3&4&5&6\\5&6&7&8\end{pmatrix}$ et si `L=[0,2]`, alors cette fonction devra renvoyer la matrice

    $$\begin{pmatrix}1&2&3&4\\5&6&7&8\end{pmatrix}$$

18. Compléter le code suivant permettant de définir une fonction d'entête `def gardeColonnes(M,L):` effectuant la même opération que dans la question précédente, mais sur les colonnes de `M`.

```
1        def gardeColonnes(M,L):
2            A = gardeLignes(np.transpose(M),L)
3            return(...)
```

19. Écrire une fonction d'entête `def sousMatrice(M,L):` prenant en entrée une matrice carrée `M` et une liste `L` formée d'entiers distincts entre 0 et $n-1$, où n est la taille de `M`, et renvoyant en sortie la matrice obtenue à partir de `M` en ne gardant que les lignes et les colonnes dont les indices sont présents dans `L`.

    Par exemple, si `M` est la matrice $\begin{pmatrix}1&2&3&4\\2&3&4&5\\5&6&7&8\\6&7&8&9\end{pmatrix}$ et si `L= [0,3]`, alors cette fonction devra renvoyer la matrice

    $$\begin{pmatrix}1&4\\6&9\end{pmatrix}$$

20. En déduire le code d'une fonction d'entête `def verifieClique(M,L):` prenant en entrée la matrice d'adjacence `M` d'un graphe et une liste `L` de sommets (distincts) de ce graphe, et renvoyant en sortie `True` si `L` est la liste des sommets d'une clique, et `False` sinon.

Annexe informatique

Si `L` est une liste de longueur n :

- `del L[i]` modifie `L` en enlevant l'élément en `i`-ième position (pour $0 \leq$ `i` $\leq n-1$)
- `L.copy()` renvoie une copie indépendante de la liste `L`

Si `M` est une matrice de taille `(n,p)` :

- `M[i,:]` renvoie la matrice qui est la `i`-ième ligne de `M` (sous réserve de bonne définition)
- `M[:,i]` renvoie la matrice qui est la `i`-ième colonne de `M`
- `np.transpose(M)` renvoie la transposée de M.
- Si `N` est une autre matrice, la commande `np.dot(M,N)` renvoie le résultat du produit matriciel `MN` si celui-ci est bien défini.
- Si `N` est une autre matrice, la commande `(M==N).all()` renvoie `True` si `M` et `N` sont égales, `False` sinon.
- `np.ones((n,p))` renvoie la matrice de taille (n,p) dont tous les coefficients valent 1.

Si le module numpy.linalg est importé à l'aide de la commande `import numpy.linalg as al` :

- `al.matrix_power(M,k)` renvoie la puissance M^k de `M`.

— Fin de l'énoncé —