Devoir surveillé numéro 7 : Concours blanc 2

*Devoir du **2 mai 2025**.*

L'usage de documents de cours et d'appareils électroniques est interdit. Le soin de la copie et la qualité de la rédaction seront pris en compte de manière importante dans la notation. Les résultats démontrés non encadrés pourront être ignorés par le correcteur. On rappelle que la qualité de l'argumentation et la rigueur (donc le soin accordé aux détails) sont au cœur des mathématiques. Sauf mention explicite du contraire, tout résultat demandé doit être démontré.

Bon courage à toutes et à tous!

Exercice 1

Pour tout entier naturel n, on pose $u_n = \prod_{k=0}^{n} \left(1 + \frac{1}{2^k}\right) = (1+1)(1+\frac{1}{2})(1+\frac{1}{4})...(1+\frac{1}{2^n})$.

1. Donner, sous forme de fractions simplifiées, les valeurs de u_0, u_1 et u_2.
2. (a) Montrer que, pour tout entier naturel n, $u_n \geq 2$.
   (b) Exprimer u_{n+1} en fonction de u_n et en déduire les variations de la suite (u_n).
3. (a) Montrer que, pour tout $x > -1$, on a : $\ln(1+x) \leq x$.
   (b) En déduire, pour tout entier naturel n, un majorant de $\ln(u_n)$.
4. En utilisant les questions précédentes, montrer que (u_n) converge vers un réel ℓ élément de $[2, e^2]$.
5. On se propose dans cette question de déterminer la nature de la série de terme général $(\ell - u_n)$.
   (a) Justifier que la suite $(\ln(u_n))_{n\in\mathbb{N}}$ converge et que l'on a :
   $$\ln(\ell) = \sum_{k=0}^{+\infty} \ln\left(1 + \frac{1}{2^k}\right).$$
   (b) Montrer que, pour tout n de $\mathbb{N}$, on a : $\ln\left(\frac{\ell}{u_n}\right) = \sum_{k=n+1}^{+\infty} \ln\left(1 + \frac{1}{2^k}\right)$.
   (c) Vérifier, en utilisant le résultat de la question 3a), que : $\forall n \in \mathbb{N}, 0 \leq \ln\left(\frac{\ell}{u_n}\right) \leq \frac{1}{2^n}$.
   (d) En déduire que : $\forall n \in \mathbb{N}, 0 \leq \ell - u_n \leq \ell\left(1 - \mathrm{e}^{-\frac{1}{2^n}}\right)$.
   (e) Justifier que, pour tout réel x, on a $1 - \mathrm{e}^{-x} \leq x$. En déduire : $\forall n \in \mathbb{N}, 0 \leq \ell - u_n \leq \frac{\ell}{2^n}$.
   (f) Déterminer la nature de la série de terme général $(\ell - u_n)$.

Exercice 2 *Problème*

Dans ce problème, on désigne par ***graphe*** tout graphe ***non orienté*** et ***simple*** (sans boucles ni multi-arêtes).

Ainsi, un graphe est un couple $G = (S, A)$ où S est un ensemble fini donnant l'ensemble des sommets de G, et A est un ensemble de parties de S toutes de cardinal 2, qui donne l'ensemble des arêtes.

Si i et j sont deux sommets d'un graphe G, on notera $i \sim_G j$ si i et j sont voisins, c'est-à-dire reliés par une arête, dans G.

Pour tout graphe $G = (S, A)$, on rappelle que :

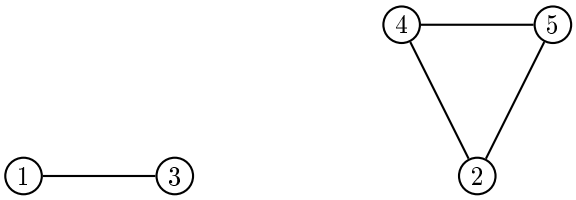
- Une ***chaîne*** de G de longueur $k \in \mathbb{N}$ est la donnée d'une suite finie $(s_1, ..., s_{k+1})$ de sommets de G tels que $s_i \sim_G s_{i+1}$ pour tout $i \in [\![1, k]\!]$. On dit que cette chaîne ***relie*** s_1 et s_{k+1}.
- Une chaine $(s_1, ..., s_{k+1})$ est dite ***fermée*** si $s_1 = s_{k+1}$, et est dite ***simple*** si elle n'emprunte pas deux fois la même arête ($\{s_i, s_{i+1}\} \neq \{s_j, s_{j+1}\}$ pour tous i et j distincts entre 1 et k).
- On appelle ***cycle*** de G toute chaîne fermée simple de G de longueur non nulle.
- Un graphe est dit ***acyclique*** s'il ne contient aucun cycle.

On définit également les notions suivantes :

- On appelle ***sous-graphe*** de G tout graphe $G' = (S', A')$ tel que $S' \subset S$ et $A' \subset A$. Autrement dit, un sous-graphe de G est un graphe obtenu à partir de G en conservant un certain nombre de ses sommets et un certain nombre d'arêtes entre les sommets choisis.
- Pour toute partie V de S, on appelle ***sous-graphe (de G) induit par*** V le sous-graphe de G noté $G[V]$ donné par
$$G[V] = (V, A_V)$$
où $A_V = \{a \in A | \exists (i, j) \in V^2, a = \{i, j\}\}$. Autrement dit, $G[V]$ est le sous-graphe obtenu en gardant les sommets éléments V et toutes les arêtes entre ces sommets. En particulier, $G[S] = G$.
- Soit $V \subset S$ non vide. On dit que $G[V]$ est une ***composante connexe*** de G si $G[V]$ est connexe, et si $G[V']$ est non-connexe pour toute partie V' de S contenant V et distincte de V.

  On admet qu'il existe un entier c et des parties deux à deux disjointes et non vides $V_1, ..., V_c$ de S telles que $S = V_1 \cup V_2 \cup ... \cup V_c$ et telle que $G[V_i]$ soit une composante connexe de G pour tout $i \in [\![1, c]\!]$.

  Les sous-graphes $G[V_1]$,...,$G[V_c]$ sont appelés les composantes connexes de G et on admet également leur unicité, à l'ordre près de celles-ci. Par exemple, si G est connexe alors il admet $G = G[S]$ comme unique composante connexe, et le graphe G représenté ci-dessous admet deux composantes connexes : $G[\ \{1, 3\}\ ]$ et $G[\ \{2, 4, 5\}\ ]$.



Enfin, pour toutes les questions d'informatique, on supposera que les graphes considérés ont pour ensemble de sommets $[\![0, n-1]\!]$, où n est l'ordre du graphe considéré. Une annexe est présente en fin d'énoncé et rappelle quelques commandes informatiques.

On supposera que les commandes suivantes ont été préalablement rentrées :

```
import numpy as np
import numpy.linalg as al
```

Les parties I, II et III de ce problème sont indépendantes, à l'exception d'une question partie II mentionnant la dépendance utilisée.

Partie I : Arbres couvrant d'un graphe

1. On souhaite montrer que les sommets d'un graphe connexe G (dont on note n l'ordre) peuvent être énumérés $v_1, ..., v_n$ de telle sorte que pour tout $i \in [\![1, n]\!]$, $G[\ \{v_1, ..., v_i\}\ ]$ soit connexe.

   On propose l'algorithme glouton suivant :

   - On choisit pour v_1 n'importe quel sommet de G.
   - Si $v_1, ..., v_i$ sont déjà construits et $i < n$, on choisit pour v_{i+1} l'un quelconque des sommets de G non présent dans la liste $[v_1, ..., v_i]$ et adjacent à l'un de ces sommets.
   - On répète l'opération précédente pour $i = 1, ..., n-1$ afin de construire une liste $[v_1, ..., v_n]$.

   (*a*) Justifier que cet algorithme construit une énumération $v_1, ..., v_n$ répondant au problème posé. On pourra procéder par l'absurde.

   (*b*) Écrire le code d'une fonction Python d'entête `def Intersection(L,N):` prenant en entrée deux listes `L` et `N` et renvoyant `True` si ces listes ont un élément en commun, et `False` sinon.

   (*c*) Écrire le code d'une fonction Python d'entête `def EnumereConnexe(V):` prenant en entrée la liste des listes d'adjacences `V` de G et renvoyant en sortie la liste $[v_1, ..., v_n]$ ainsi construite.

On appelle ***arbre*** tout graphe acyclique connexe.

On souhaite démontrer le théorème suivant :

Théorème 1. *Soit T un graphe, notons n son ordre. Il est équivalent de dire :*
(i) T est un arbre,
(ii) T est connexe et tout sous-graphe obtenu à partir de T en gardant tous ses sommets et en ne supprimant qu'une arête est non-connexe,
(iii) T est connexe et possède exactement $n-1$ arêtes.

2. Dans cette question, on fixe un graphe T et on démontre $(i) \iff (ii)$.

   (*a*) Justifier que dans tout graphe, deux sommets reliés par une chaîne sont reliés par une chaîne simple. *On pourra considérer une chaîne de longueur minimale reliant ces sommets.*

   (*b*) Soit G un graphe connexe. Soient a une arête de G, et G' le sous-graphe de G obtenu en retirant l'arête a. Montrer que si G' est connexe, alors G n'est pas acyclique. En déduire $(i) \implies (ii)$.

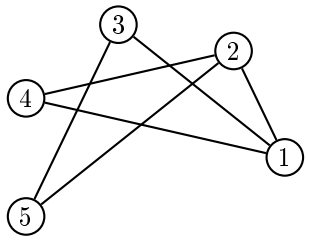
   (*c*) Montrer $(ii) \implies (i)$.

On appelle ***arbre couvrant*** d'un graphe G tout sous-graphe T de G ayant les mêmes sommets que G et qui est un arbre.

3. Montrer que si un graphe G admet un arbre couvrant, alors il est connexe.

4. On souhaite montrer que toute graphe G connexe admet au moins un arbre couvrant.

   (*a*) Représenter un arbre couvrant du graphe ci-dessous.



   (*b*) On se propose l'algorithme glouton suivant pour construire un arbre couvrant T d'un graphe connexe G donné en entrée :

   - Initialement, on pose $T = G$.
   - On parcours les arêtes de T. Si l'on peut enlever une arête à T tout en conservant un graphe connexe, on le fait et on met fin à cette étape.
   - On répète l'étape précédente jusqu'à ce qu'on ne puisse plus enlever d'arête à T, et on renvoie T lorsque c'est le cas.

Justifier que le graphe obtenu à la fin de cet algorithme est un arbre couvrant de G.

(c) Énoncer un théorème caractérisant la connexité d'un graphe G à l'aide de sa matrice d'adjacence.

(d) En déduire le code d'une fonction Python `EstConnexe(M)` prenant en entrée la matrice d'adjacence `M` de G et renvoyant en sortie `True` si G est connexe, et `False` sinon.

(e) En déduire le code d'une fonction `TestRetrait(M,i,j)` prenant en entrée la matrice d'adjacence `M` de G et les numéros `i` et `j` de deux de ses sommets adjacents, et renvoyant en sortie `True` si le sous-graphe obtenu en supprimant l'arête de `i` vers `j` est connexe, et `False` sinon.

(f) Écrire le code d'une fonction Python `ListeAretes(M)` prenant en entrée la matrice d'adjacence d'un graphe G et renvoyant en sortie la liste `[ a1, a2,...,ak ]` des arêtes (sans répétitions) de G, où une arête reliant deux sommets i et j avec $i < j$ est donnée en python par la liste $[i, j]$.

(g) Recopier et compléter le code suivant pour qu'il implémente l'algorithme ci-dessus, la fonction prenant en entrée la matrice d'adjacence `M` de G et renvoyant en sortie la matrice d'adjacence d'un arbre couvrant T de G.

```
def ArbreCouvrant(M):
    n,p=np.shape(M)
    T=np.copy(M) # créer une copie de M
    estArbre=False #Vaut True si T est un arbre
    while not(estArbre):
        retrait=False # Vaut True si une arête a été retirée
        # Parcours des arêtes de T
        for a in ListeAretes(T):
            i,j=...
            if ... :
                # retrait de l'arête de i vers j
                retrait=True
                ...=...
                break # interrompt la boucle for
        if not(retrait):
            ...=...
    return(T)
```

5. Soit T un arbre, notons n son ordre. Montrer qu'il existe une énumération $v_1, ..., v_n$ de ses sommets telle que : pour tout $i \geq 2$, v_i a un unique voisin parmi $v_1, ..., v_{i-1}$.

6. On veut enfin montrer l'équivalence entre (i) et (iii) dans le théorème 1.

   (a) A l'aide de la question précédente, montrer par récurrence sur n que tout arbre d'ordre n possède exactement $n - 1$ arêtes.

   (b) Réciproquement, montrer que (iii) $\implies$ (i). On pourra utiliser l'existence d'un arbre couvrant pour tout graphe connexe.

7. Montrer que tout arbre d'ordre au moins 2 possède au moins deux sommets de degré 1.

Partie II : Graphes aléatoires

Soit $n \geq 2$ un entier, fixé dans cette partie.

On considère le graphe complet K ayant pour ensemble de sommets $[\![1, n]\!]$. Cela signifie que deux sommets distincts de K sont toujours reliés par une arête dans K.

On procède à l'expérience aléatoire suivante : pour chaque arête de K, on lance une pièce à Pile ou Face et on marque l'arête considérée si et seulement si la pièce tombe sur Pile. Ces lancers sont bien-sûr considérés mutuellement indépendants.

On note G le graphe obtenu en gardant tous les sommets de K et uniquement les arêtes marquées suite à cette expérience aléatoire.

On note $\mathcal{SG}(K)$ l'ensemble des sous-graphes de K ayant pour ensemble de sommets $[\![1, n]\!]$.

8. Quel est le nombre d'arêtes de K ? On note N ce nombre dans la suite.

9. Dans cette question, on suppose que la pièce est équilibrée.

   (a) Montrer que $\text{Card}\,(\mathcal{SG}(K)) = 2^{\frac{n(n-1)}{2}}$.

   (b) Soit $T \in \mathcal{SG}(K)$. On note $[G = T]$ l'événement "le graphe G obtenu est T". Calculer $\mathbb{P}([G = T])$.

   (c) En déduire, à l'aide du théorème 1 de la partie I, que la probabilité que G soit un arbre est inférieure à $\binom{N}{n-1} 2^{-\frac{n(n-1)}{2}}$.

Dans toute la suite de cette partie, on suppose que la pièce tombe sur Pile avec probabilité $p \in]0,1[$. On s'intéresse à la probabilité que G soit connexe pour de grandes valeurs de n.

On note C l'événement "G est connexe".

10. Pour tout $k \in [\![1, n]\!]$, soit A_k l'événement "il existe un sous-ensemble de k sommets de G tel qu'aucun de ces k sommets ne soit relié à l'un des $n-k$ autres par une chaîne de G".

    (a) Soit $k \in [\![1, n-1]\!]$ et soit G_k une partie de $[\![1, n]\!]$ de cardinal k. On pose $D_k = [\![1, n]\!] \setminus G_k$. Quel est le nombre d'arêtes dans K reliant un élément de G_k à un élément de D_k ?

    (b) Montrer que la probabilité qu'aucun sommet de G_k ne soit adjacent à l'un des sommet de D_k dans G est $(1-p)^{k(n-k)}$.

    (c) Démontrer l'inégalité de Boole : pour tous événements $B_1, ..., B_r$ d'un espace probabilisé fini $(\Omega, \mathcal{P}(\Omega), \mathbb{P})$, on a :
    $$\mathbb{P}(B_1 \cup B_2 \cup \ldots \cup B_r) \leq \sum_{i=1}^{r} \mathbb{P}(B_i).$$

    (d) En déduire que $\mathbb{P}(A_k) \leq \binom{n}{k}(1-p)^{k(n-k)}$.

11. (a) Justifier que $\overline{C} \subset \bigcup_{k=1}^{\lfloor n/2 \rfloor} A_k$.

    (b) En déduire que $\mathbb{P}(\overline{C}) \leq \sum_{k=1}^{\lfloor n/2 \rfloor} \binom{n}{k} \mathrm{e}^{-pk(n-k)}$. On pourra utiliser que $1 - x \leq e^{-x}$ pour tout réel x.

12. Dans cette question, on considère le cas où la probabilité p de faire Pile avec la pièce vérifie $p > 2\dfrac{\ln(n)}{n}$ et on veut montrer que la probabilité c_n que G soit connexe tend vers 1 lorsque le nombre n de sommets considérés tend vers $+\infty$.

    On pose $\alpha = \dfrac{pn}{\ln(n)}$ de sorte que $p = \alpha\dfrac{\ln(n)}{n}$ et $\alpha > 2$.

    (a) Justifier que, si n est assez grand, il est bien possible d'avoir $p > 2\dfrac{\ln(n)}{n}$.

    (b) Justifier que $\binom{n}{k} \leq n^k$ pour tout $k \in [\![1, n]\!]$.

    (c) En déduire que $\mathbb{P}(\overline{C}) \leq \sum_{k=1}^{\lfloor n/2 \rfloor} \exp\left((1 - \dfrac{\alpha}{2}) k \ln(n)\right)$.

    (d) Montrer que $\sum_{k \geq 1} \exp\left((1 - \dfrac{\alpha}{2}) k \ln(n)\right)$ converge et calculer sa somme.

    (e) En déduire l'existence d'une constante $\gamma > 0$ telle que $\mathbb{P}(\overline{C}) \leq \dfrac{1}{n^\gamma - 1}$.

    (f) En déduire une minoration de la probabilité c_n que G soit connexe, puis montrer que $c_n \xrightarrow[n \to +\infty]{} 1$.

Partie III : Composantes connexes et matrice laplacienne

On fixe un graphe $G = (S, A)$, on note n son ordre, et on supposera que l'ensemble S des sommets de G est $S = [\![1, n]\!]$. On suppose de plus $n \geq 2$.

On note M la matrice d'adjacence de G. On appelle ***matrice des degrés*** de G la matrice diagonale, notée D, dont les coefficients diagonaux sont, dans l'ordre, $\deg(1), ..., \deg(n)$ (où $\deg(i)$ est le degré du sommet i pour tout i).

Enfin, on appelle ***matrice laplacienne*** de G la matrice notée L donnée par :

$$L = D - M.$$

Pour toute matrice $A \in \mathcal{M}_n(\mathbb{R})$, on pose $\mathrm{Ker}(A) = \{X \in \mathcal{M}_{n,1}(\mathbb{R}), AX = 0_{n,1}\}$.

13. *On rappelle que, pour les questions informatiques, les sommets sont numérotés de* 0 *à* $n-1$.

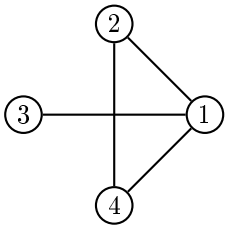
    (a) Écrire le code d'une fonction Python `Deg(M,i)` prenant en entrée la matrice d'adjacence `M` de G et le numéro `i` de l'un de ses sommets, et renvoyant en sortie le degré du sommet numéro `i`.

    (b) En déduire le code d'une fonction `Laplace(M)` prenant en entrée la matrice d'adjacence `M` de G et renvoyant en sortie sa matrice laplacienne.

14. Montrer que L est symétrique.

15. Montrer que, pour tout $A \in \mathcal{M}_n(\mathbb{R})$, $\mathrm{Ker}(A)$ est un sous-espace vectoriel de $\mathcal{M}_{n,1}(\mathbb{R})$.

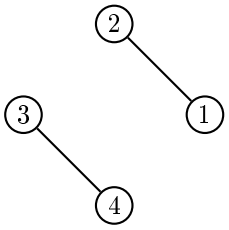
16. Un premier exemple : on considère le graphe G ci-dessous.



    (a) Donner M, D et L.

    (b) Montrer que $\begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix} \in \mathrm{Ker}(L)$.

17. Un second exemple : on considère le graphe G ci-dessous.



    (a) Donner la matrice L et montrer $\mathrm{Vect}\left(\begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \\ 1 \\ 1 \end{pmatrix}\right) \subset \mathrm{Ker}(L)$.

    (b) Montrer $\mathrm{Ker}(L) = \mathrm{Vect}\left(\begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \\ 1 \\ 1 \end{pmatrix}\right)$.

    (c) En déduire la dimension de $\mathrm{Ker}(L)$.

On retourne au cas général.

18. On pose $U = \begin{pmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{pmatrix} \in \mathcal{M}_{n,1}(\mathbb{R})$. Montrer que $U \in \mathrm{Ker}(L)$ puis que $\mathrm{Vect}(U) \subset \mathrm{Ker}(L)$.

On veut montrer que si G est connexe, alors $\mathrm{Ker}(L) = \mathrm{Vect}(U)$.

19. On appelle ***fonction harmonique*** sur G toute application $f : S \to \mathbb{R}$ telle que, pour tout sommet v de G ayant au moins un voisin, $f(v)$ est la moyenne des valeurs prises par f sur les voisins de v :

$$f(v) = \frac{1}{\deg(v)} \sum_{\substack{i=1 \\ v \sim_G i}}^{n} f(i).$$

On veut montrer que si G est connexe, alors toute fonction harmonique f sur G est constante : $\exists C \in \mathbb{R}, \forall v \in S, f(v) = C$. On fixe une telle fonction harmonique f et on suppose G connexe.

(a) On pose $M = \max_{v \in S}(f(v))$. Montrer que, pour tout sommet v tel que $f(v) = M$, les voisins v' de v vérifient aussi $f(v') = M$.

(b) En déduire le résultat voulu.

20. Montrer que, si G est connexe, alors $\mathrm{Ker}(L) = \mathrm{Vect}(U)$.

On pourra considérer l'application $\phi : \left| \begin{array}{ccc} S = [\![1, n]\!] & \longrightarrow & \mathbb{R} \\ i & \longmapsto & x_i \end{array} \right.$, où $\begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}$ est un élément fixé de $\mathrm{Ker}(L)$.

21. Passons au cas général d'un graphe non nécessairement connexe. Notons c le nombre de composantes connexes de G, et soient $G[V_1], ..., G[V_c]$ les composantes connexes de G.

On considère, pour tout $i \in [\![1, c]\!]$, le vecteur $U_i = (u_{k,\ell}^{(i)})_{\substack{1 \leq k \leq n \\ 1 \leq \ell \leq 1}} \in \mathcal{M}_{n,1}(\mathbb{R})$ donné par :

$$\forall k \in [\![1, n]\!], u_{k,1}^{(i)} = \begin{cases} 1 \text{ si } k \in V_i \\ 0 \text{ sinon} \end{cases}.$$

Par exemple, si $n = 5$, $c = 2$, $V_1 = \{1, 3\}$ et $V_2 = \{2, 4, 5\}$, alors $U_1 = \begin{pmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}$ et $U_2 = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 1 \\ 1 \end{pmatrix}$.

(a) Montrer que les vecteurs $(U_1, ..., U_c)$ forment une famille libre.

(b) Montrer que, pour tout $i \in [\![1, c]\!]$, $U_i \in \mathrm{Ker}(L)$ puis que $\mathrm{Vect}(U_1, ..., U_c) \subset \mathrm{Ker}(L)$.

(c) En déduire $\dim(\mathrm{Ker}(L)) \geq c$.

(d) Soit $\begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} \in \mathrm{Ker}(L)$. Soit $i \in [\![1, c]\!]$. Posons $n_i = \mathrm{Card}(V_i)$ et soient $s_1, ..., s_{n_i}$ les sommets deux à deux distincts éléments de V_i. Montrer que, si $n_i \geq 2$, alors $x_{s_1} = x_{s_2} = \ldots = x_{s_{n_i}}$. On pourra s'inspirer de la méthode de la question 20.

(e) Montrer que $\mathrm{Ker}(L) = \mathrm{Vect}(U_1, ..., U_c)$ puis que $\dim(\mathrm{Ker}(L)) = c$.

Annexe informatique

Voici une liste non-exhaustive de commandes que vous pouvez utiliser.

- Si `L` est une liste, `L[i]` renvoie l'élément d'indice `i` de `L`.
- La commande `break` interrompt la dernière boucle `while` ou `for` en cours.
- Si `L` est une liste de nombres ou un vecteur numpy, `np.sum(L)` renvoie la somme des éléments de `L`.
- Si `L` est une liste et `a` une variable, la commande `a in L` renvoie `True` si une entrée de `L` est égale à `a`, et `False` sinon.

Si `M` est une matrice, de type `array`:

- Si `n` est un entier, `np.zeros((n,p))` renvoie une matrice nulle de taille (`n`,`p`)
- `np.shape(M)` renvoie les dimensions de `L` sous la forme d'un uplet.
- `np.copy(M)` renvoie une copie indépendante de la matrice `M`.
- Si `M` est une matrice formée de booléens, `M.all()` renvoie `True` si tous les coefficients de `M` sont `True` et `False` sinon.
- `np.dot(M,N)` renvoi le résultat du produit matriciel `MN`, si `N` est une autre matrice convenable.
- `al.matrix_power(M,k)` renvoie la puissance `k`-ième de `M`.

— Fin de l'énoncé —