TP 1 – Révisions – Matrices et programmation

On travaille avec les modules `numpy` et `numpy.linalg`.

```
import numpy as np
import numpy.linalg as alg
```

1 Création de matrices

Pour définir une matrice, on utilise la fonction array du module numpy.

```
A = np.array([[1,2,3],[4,5,6]])
A
> array([[1,2,3],
         [4,5,6]])
```

L'attribut `shape` donne la taille d'une matrice : nombre de lignes, nombre de colonnes. On peut redimensionner une matrice, sans modifier ses termes, à l'aide de la méthode `reshape`.

```
np.shape(A)
> (2,3)
A=np.reshape(A,(3,2))
> array([[1,2],
         [3,4],
         [5,6]])
```

L'accès à un terme de la matrice `A` se fait à l'aide de l'opération d'indexage `A[i,j]` où `i` désigne la ligne et `j` la colonne. Attention, les indices commencent à zéro ! À l'aide d'intervalles, on peut également récupérer une partie d'une matrice : ligne, colonne, sous-matrice.

```
A[1,0]
> 3
np.shape(A[0,:])
> (2,)
A[0:1, :]
> array([[1,2]])
A[:, 1]
> array([2,4,6])
A[:, 1:2]
> array([[2],
         [4],
         [6]])
A[1:3, 0:2]
> array([[3,4],
         [5,6]])
```

Les fonctions `zeros` et `ones` permettent de créer des matrices remplies de 0 ou de 1. La fonction `eyes` permet de créer une matrice du type I_n où n est un entier. La fonction `diag` permet de créer une matrice diagonale.

```
np.zeros((2,3))
> array([[0., 0., 0.],
         [0., 0., 0.]])
np.ones((2,2))
> array([[1., 1.],
         [1., 1.]])
np.eye(2)
> array([[1., 0.],
         [0., 1.]])
np.diag([1,2,3])
> array([[1, 0, 0],
         [0, 2, 0],
         [0, 0, 3]])
```

Enfin la fonction `concatenate` permet de créer des matrices par blocs.

```
A = np.ones((2,3))
B = np.zeros((1,3))
C = np.zeros((2,2))
np.concatenate((A,B), axis=0)
> array([[1., 1., 1.],
         [1., 1., 1.],
         [0., 0., 0.]])
np.concatenate((A,C),axis=1)
> array([[1., 1., 1., 0., 0.],
         [1., 1., 1., 0., 0.]])
```

2 Calcul matriciel

Les opérations d'ajout et de multiplication par un scalaire se font avec les opérateurs `+` et `*`.

```
A=np.array([[1,2], [3,4]])
B=np.eye(2)
A + 3*B
> array([[4., 2.],
         [3., 7.]])
```

Pour effectuer un produit matriciel (lorsque que cela est possible), il faut employer la fonction `dot`.

```
A=np.array([[1,2], [3,4]])
B=np.array([[1,1,1], [2,2,2]])
np.dot(A,B)
> array([[5, 5, 5],
         [11, 11, 11]])
```

La fonction `matrix_power` de la librairie `numpy.linalg` permet de calculer des puissances de matrices.

```
alg.matrix_power(A, 3)
> array([[37, 54],
         [81, 118]])
```

La transposée s'obtient avec la fonction `transpose` de la librairie `bumpy`.

```
np.transpose(B)
> array([[1, 2],
         [1, 2],
         [1, 2]])
```

Le déterminant, le rang et la trace d'une matrice s'obtiennent par les fonctions `det`, `matrix_rank` du module `numpy.linalg` et `trace` du module `numpy`. Enfin la fonction `inv` du module `numpy.linalg` renvoie la matrice inverse si elle existe.

```
alg.det(A)
> -2.0000000000004
alg.matrix_rank(A)
> 2
np.trace(A)
> 5
alg.inv(A)
> matrix([[-2., 1.],
          [1.5, -0.5]])
```

Pour résoudre le système linéaire $Ax = b$ lorsque la matrice A est inversible, on peut employer la fonction `solve` du module `numpy.linalg`.

```
b=np.array([1,5])
alg.solve(A,b)
> array([3., -1.])
```

La fonction `eigvals` du module `numpy.linalg` renvoie les valeurs propres de la matrice. Pour obtenir en plus les vecteurs propres associés, il faut employer la fonction `eig`.

```
alg.eigvals(A)
> array([-0.37228132, 5.37228132])
```

La fonction `vdot` permet de calculer le produit scalaire de deux vecteurs de $\mathbb{R}^n$.

```
u = np.array([1, 2])
v = np.array([3, 4])
np.vdot(u,v)
> 11
```

3 Exercices

Exercice 1. Sans rentrer les coefficients un à un, déclarer les matrices $\begin{pmatrix} 5 & 3 & 3 \\ 3 & 5 & 3 \\ 3 & 3 & 5 \end{pmatrix}$ et $\begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{pmatrix}$.

Exercice 2. Que vaut le produit matriciel $\begin{pmatrix} a_1 \\ \vdots \\ a_n \end{pmatrix} \begin{pmatrix} 1 & \dots & 1 \end{pmatrix}$? En déduire une manière de déclarer la matrice

$$\begin{pmatrix} 1 & 1 & \dots & 1 \\ 2 & 2 & \dots & 2 \\ \vdots & \vdots & & \vdots \\ 10 & 10 & \dots & 10 \end{pmatrix}.$$

Exercice 3. On définit la matrice $A = \begin{pmatrix} 1 & 1 & 2 & 0 \\ 1 & -1 & 0 & -2 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & -1 \end{pmatrix}$.

1. Déclarer la matrice A.
2. Écrire une fonction `lignes(A,i,j,a)` qui prend entrée une matrice `A`, les indices de lignes `i` et `j`, ainsi qu'un réel `a`, et qui retourne la matrice obtenue à partir de A en soustrayant `a` fois la ligne `j` à la ligne `i`.
3. Écrire une fonction `echange(A,i,j)` qui prend en entrée une matrice `A`, et des indices de lignes `i` et `j`, et retourne la matrice obtenue en échangeant les lignes `i` et `j` de `A`.
4. Appliquer l'algorithme du pivot de Gauss sur la matrice A à l'aide des fonctions des questions précédentes. En déduire le rang de A.
5. Vérifier le résultat à l'aide de la fonction `matrix_rank` du module `numpy.linalg`.
6. Écrire une fonction `pivot(A)` qui prend en entrée une matrice `A` et retourne la matrice obtenue après avoir effectué l'algorithme du pivot de Gauss sur la matrice `A`.

Exercice 4.

1. *a.* En une seule ligne de commande, créer le vecteur $x = \left(1, \frac{1}{4}, \frac{1}{9}, \frac{1}{16}, \ldots, \frac{1}{100}\right)$ sans saisir un à un les éléments.

   b. Compléter la commande précédente pour qu'elle renvoie $\sum_{k=1}^{10} \frac{1}{k^2}$.

2. En une seule ligne de commande, calculer la somme $\sum_{n=0}^{10} \frac{1}{2^n}$.

Exercice 5.

1. Écrire une commande affichant tous les multiples de 7 inférieurs ou égaux à 1000.
2. Compléter la commande précédente pour qu'elle renvoie le plus grand multiple de 7 inférieur ou égal à 1000.
3. Ecrire une commande renvoyant le plus petit multiple de 7 supérieur strictement à 1000 .

Exercice 6. On modélise le nombre de véhicules passant sur une petite route de campagne en une journée par une variable aléatoire X qui suit une loi de Poisson $\mathscr{P}(5)$.

1. Introduire une matrice A de taille 52×7 qui simule une année de circulation, chaque ligne contenant le trafic de chaque jour d'une semaine. La matrice A contient alors 52×7 nombres tirés suivant la loi $\mathscr{P}(5)$.
2. Déterminer le nombre maximal de véhicules circulant sur la route en une journée durant l'année, puis le nombre maximal de véhicules un mercredi durant l'année.
3. Déterminer les numéros des semaines où la route a connu son maximum de fréquentation.
4. Déterminer le nombre de jours où la route a connu son minimum de fréquentation.
5. En une seule ligne de commande, calculer le nombre moyen de véhicules par jour durant ces 364 jours. Recommencez plusieurs fois en recréant la matrice A. Que constate-t-on ? Était-ce prévisible ?
6. En une seule ligne de commande, évaluer la probabilité qu'une journée voie passer 8 véhicules ou plus. Calculer la valeur exacte de cette probabilité.

Exercice 7. 1. Écrire un programme permettant de calculer $\sum_{k=1}^{n} \frac{(-1)^{k-1}}{k}$ pour une valeur de n entrée par l'utilisateur. On proposera pour cela deux méthodes, l'une utilisant la fonction `sum`, l'autre à l'aide d'une boucle `for`.

2. On peut montrer que la série $\sum \frac{(-1)^{n-1}}{n}$ converge et que sa somme vaut $\ln(2)$. Écrire un programme permettant de déterminer le plus petit entier naturel n pour lequel $|S_n - \ln(2)| \leqslant 10^{-3}$, où (S_n) désigne la suite des sommes partielles de cette série.

Exercice 8. Soient $(u_n)_{n\in\mathbb{N}^*}$ et $(v_n)_{n\in\mathbb{N}^*}$ les suites définies par $u_n = \sum_{k=1}^{n} \frac{1}{k^2}$ et $v_n = u_n + \frac{1}{n}$, pour tout $n \in \mathbb{N}^*$.

1. Montrer que les suites $(u_n)_{n\in\mathbb{N}^*}$ et $(v_n)_{n\in\mathbb{N}^*}$ sont adjacentes.

2. Écrire un programme qui donne une approximation de $\sum_{k=1}^{+\infty} \frac{1}{k^2}$ à ε près pour $\varepsilon > 0$ donné.

Exercice 9. Ecrire une fonction `trace(A)` qui prend en entrée une matrice A, et renvoie la trace de A si la matrice est carrée, et un message d'erreur sinon.

Exercice 10.

1. Pour k et n des entiers, écrire $\binom{n}{k}$ sous la forme d'un quotient de deux produits, et en déduire une fonction `binomial(k,n)` qui prend en entrée des entiers `k` et `n` et retourne $\binom{n}{k}$, en ayant recours aux fonctions `prod` et `arange` de la bibliothèque `numpy`.

2. À l'aide de la fonction de la question précédente, créer la matrice carrée de taille n suivante :

$$C = \begin{pmatrix} \binom{0}{0} & \binom{0}{1} & \cdots & \binom{0}{n} \\ \binom{1}{0} & \binom{1}{1} & \cdots & \binom{1}{n} \\ \vdots & \vdots & \vdots & \vdots \\ \binom{n-1}{0} & \binom{n-1}{1} & \cdots & \binom{n-1}{n-1} \end{pmatrix}.$$

3. Proposer un autre moyen de construire la matrice C, en utilisant la formule de Pascal.

Exercice 11. Écrire une fonction `Prod(A,B)` qui prend en entrée deux matrices `A` et `B`, et qui renvoie leur produit matriciel AB lorsqu'il est bien défini (sans utiliser la fonction `np.dot`), et renvoie un message d'erreur lorsqu'il n'est pas défini.

Exercice 12. 1. On dit qu'une matrice est stochastique si tous ses coefficients sont positifs, et la somme de chacune de ses lignes vaut 1. Écrire une fonction `stoch(A)` qui prend en entrée une matrice `A` et renvoie `True` si la matrice est stochastique, et `False` sinon.

2. On dit qu'une matrice est bistochastique si elle est stochastique et que la somme de chacune de ses colonnes vaut 1. Écrire une fonction `bistoch(A)` qui prend en entrée une matrice `A` et renvoie `True` si la matrice est bistochastique, et `False` sinon.

Exercice 13. On dit qu'une matrice carrée $A \in \mathscr{M}_n(\mathbb{R})$ est à diagonale strictement dominante si pour tout $i \in [\![1, n]\!]$, on a $|a_{i,i}| > \sum_{j\neq i} |a_{i,j}|$.

Écrire une fonction `Diagdom(A)` qui prend en entrée une matrice `A` et qui renvoie `True` si `A` est à diagonale strictement dominante, et `False` sinon.