TP 8 : tableaux `numpy` et méthodes numériques élémentaires

On a déjà vu des tableaux sur la feuille TP 3 (graphisme). Il s'agit d'une structure de données qui superficiellement ressemble à une liste Python, mais qui a des propriétés bien différentes : les listes servent à stocker des données, les tableaux sont fait pour effectuer des calculs numériques. Pour simplifier la notation, sur cette feuille on omet souvent le préfixe `np` avant les fonctions de `numpy`, mais n'oubliez pas de le mettre dans votre code.

Exercice 1. *Extrait de Terminale S*

Dans un bouquin de Terminale S on peut trouver l'assertion suivante :

2. Les termes de la suite (v_n) définie sur $\mathbb{N}$ par :
$$v_n = \sin n$$
se répartissent uniformément dans l'intervalle $[-1\,;\,1]$.

En utilisant *une* ligne de code (avec un tableau) et une visualisation adéquate, vérifier si c'est vrai. Afficher votre conclusion (`print`).

Exercice 2. *Matrice aléatoire*

Créer et visualiser une matrice carrée $A = (a_{ij})$ de taille 300 où la composante a_{ij} est un entier aléatoire tiré au hasard dans $[0, i]$ $(i, j = 1, 2, \ldots, 300)$. On pourra utiliser la commande `plt.matshow` (de `matplotlib`) avec l'option de jeu de couleurs `cmap='hot'`.

Exercice 3. *Matrices, vecteurs et algèbre linéaire*

1. Définir à l'aide du type `array` du module `numpy` les matrices et vecteurs suivants:

$$A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \quad B = \begin{pmatrix} 2 & -1 \\ 2 & 7 \end{pmatrix} \quad C = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 2 & 2 & 2 & 2 \end{pmatrix} \quad x = \begin{pmatrix} 1 \\ -2 \end{pmatrix} \quad y = \begin{pmatrix} 3 & 1 & -0.5 \end{pmatrix} \quad z = \begin{pmatrix} 7 & 7 \end{pmatrix}$$

$$D = (d_{i,j}) \in \mathcal{M}_{10,20}(\mathbb{R}),\ d_{i,j} = 0 \text{ pour tout } 1 \le i \le 10, 1 \le j \le 20.$$
$$E = (e_{i,j}) \in \mathcal{M}_7(\mathbb{R}),\ e_{i,j} = 7 \text{ pour tout } 1 \le i, j \le 7$$
$$I \text{ la matrice identité pour les matrice } \mathcal{M}_8(\mathbb{R})$$
$$u = (u_i) \in \mathbb{R}^{100}, \quad u_i = i \text{ pour tout } 1 \le i \le 100$$

$L = (\ell_{i,j})$ la matrice diagonale de taille 100×100, $\ell_{ij} = \begin{cases} i, & \text{si } i = j \\ 0, & \text{sinon} \end{cases}$ pour tout $1 \le i, j \le 100$

$M = (m_{i,j}) \in \mathcal{M}_{100}(\mathbb{R})$ la matrice tri-diagonale définie par $m_{ij} = \begin{cases} 2, & \text{si } i = j \\ -1, & \text{si } j = i - 1 \text{ ou } j = i + 1 \\ 0, & \text{sinon.} \end{cases}$

Recommandation : préférer les solutions n'utilisant pas de boucle!

2. Calculer et afficher:

$$A + B, \quad A\ B, \quad G = (g_{ij}) \text{ avec } g_{ij} = a_{ij} b_{ij}, \quad A\ C, \quad A\ x, \quad 10\ C, \quad z\ C, \quad x + z, \quad \langle x, z \rangle .$$

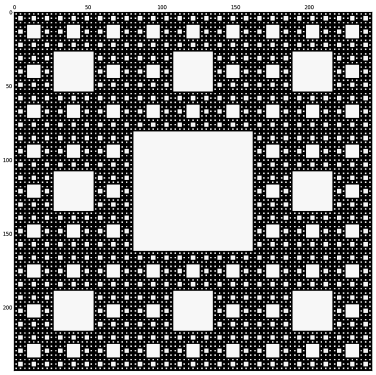
Remarquer que les vecteur `numpy` n'ont pas d'orientation: ils ne sont ni ligne ni colonne.

3. Modifier la matrice A en remplaçant l'élément 4 par 0. Remplacer la deuxième ligne de la matrice C par le vecteur $\begin{pmatrix} 4 & 3 & 2 & 1 \end{pmatrix}$. A partir de la matrice C ainsi obtenue, créer une nouvelle matrice $F \in \mathcal{M}_2(\mathbb{R})$ contenant les deux premières colonnes de C.

4. Calculer et afficher la puissance 10 de A. Proposer une solution en utilisant une boucle `for`. Comparer au résultat obtenu à l'aide de la fonction `matrix_power` du module `numpy.linalg`.

Exercice 4. *Tapis de Sierpiński*

Afficher le tapis de Sierpiński de niveau 6, c'est-à-dire la figure suivante. On utilisera `hstack` et `vstack` de `numpy` et une boucle `for` (et aucune récursivité).



Exercice 5. *Systèmes d'équations linéaires*

Résoudre le système linéaire $Ax = b$ avec

$$A = \begin{pmatrix} 6 & 2 & 8 & 0 \\ 3 & 0 & 1 & 8 \\ 7 & 4 & 0 & 3 \\ 2 & 6 & 7 & 0 \end{pmatrix}, \qquad b = \begin{pmatrix} 1 \\ 2 \\ 3 \\ 4 \end{pmatrix},$$

par trois méthodes suivantes :

1. en utilisant la matrice inverse;
2. en utilisant la commande `solve`.
3. en utilisant la *méthode de Cramer* :

   Soit `A1` une *copie* de `A`;

   on affecte une colonne de la matrice `A1` avec la colonne b, par exemple `A1[:, j] = b` (pourvu que `b` soit de la bonne longueur et "sans forme");

   alors la j-ième inconnue du système et donnée par $\det(A1)/\det(A)$.

   Attention ! on ne modifiera pas la matrice A du système mais on utilisera une copie.

Exercice 6. *Interpolation de Lagrange*

Soit $N \in \mathbb{N}^*$ et soient $x_0, x_1, \ldots, x_N \in \mathbb{R}$ $(N+1)$ nombres réels distincts 2 à 2. Soit f une fonction continue. Dans cet exercice, on se propose de construire un polynôme P de degré N qui approche la fonction f au sens suivant

$$P(x_i) = f(x_i), \qquad i = 0, 1, \ldots, N. \tag{1}$$

Notons $P(X) = a_0 + a_1 X + a_2 X^2 + \cdots + a_N X^N$ avec les coefficients $a_0, \ldots, a_N$ à déterminer. On voit aisément qu'en posant successivement $X = x_0$, puis $X = x_1$, ..., $X = x_n$ dans (1), on obtient le système

d'équations linéaires $L\,a = b$, d'inconnue a, où :

$$L = \begin{pmatrix} 1 & x_0 & x_0^2 & \dots & x_0^N \\ 1 & x_1 & x_1^2 & \dots & x_1^N \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_N & x_N^2 & \dots & x_N^N \end{pmatrix}, \qquad a = \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_N \end{pmatrix}, \qquad b = \begin{pmatrix} f(x_0) \\ f(x_1) \\ \vdots \\ f(x_N) \end{pmatrix}.$$

Posons $f(x) = \sin(x)$ sur $[-\pi, \pi]$. Soit `x` le vecteur composé de $N+1$ points équirépartis sur $[-\pi, \pi]$ (cf. `linspace`).

1. Soit $N = 4$. Construire la matrice L (cf. `hstack`, `reshape`) et résoudre le système linéaire $L\ a = b$.
2. Visualiser, sur un même graphique, la fonction f et le polynôme obtenu dans la question précédente. Pour évaluer un polynôme étant donnés le vecteur de ses coefficients `a = [a_0, ..., a_N]` et des points d'abscisses `t = np.linspace(-pi, pi, 250)` (par exemple), on peut utiliser la commande `polyval(a[::-1], t)` (cette commande du package `numpy` accepte les coefficients dans l'ordre `[a_N, ...,a_0]`, d'où besoin de l'inversion `a[::-1]`).

   Refaire avec $N = 10$.
3. *Facultatif.* La commande `polyfit(x, y, k)`, étant donnés un vecteur de points d'abscisses `x`, un vecteur de valeurs `y = f(x)` et un entier `k`, retourne les coefficients du polynôme de degré `n` passant *au plus près* des points (`x[i], y[i]`), `i = 0, ..., len(x)`. Si de plus `k = len(x)-1`, la commande retourne le même polynôme que celui obtenu "à la main" dans la question précédente (car il existe un unique polynôme de degré k passant par $k+1$ points).

   Refaire la question précédente en utilisant la commande `polyfit` (le resultat de `polyfit` et compatible avec `polyval`, cette fois il n'y a donc pas de besoin d'inverser le vecteur des coefficients).

Exercice 7. *Valeurs propres d'une matrice*

La commande `diag(v, k)` crée une matrice dont la diagonale 'décalée' de `k` est le vecteur `v` (`k` peut être négatif).

1. Construire une matrice carrée L de taille N qui a 1 en dessous et au dessus de la diagonale et 0 ailleurs. Pour l'application numérique dans la suite, on pourra prendre $N = 10$, 50, 100 et 500.
2. Calculer les valeurs propres et les vecteurs propres de L (sans affichage).
3. Afficher la plus grande valeur propre et calculer la valeur absolue (composante par composante) du vecteur propre correspondant. Ensuite, on le normalisera (en multipliant par une constante) pour que sa plus grande composante soit égale à 1.
4. Soit `x = linspace(0, 1, N)`. Visualiser la fonction `sin(pi * x)`. Rajouter, sur le même graphique, le vecteur propre normalisé obtenu dans la question précédente (`plot(x, v)`). Utiliser les valeurs de $N = 10, 50, 100, 500$. Observer le résultats.

Exercice 8. *Dichotomie*

1. Écrire une procédure `dicho(f, a, b, eps)` qui, étant donnés une fonction `f` et deux réels `a` $<$ `b`, renvoie une solution de l'équation $f(x) = 0$ dans l'intervalle $[a, b]$ par la méthode de dichotomie avec le paramètre de précision `eps`.

   Remarque technique : pour passer une fonction mathématique en argument pour la procédure `dicho`, on définit d'abord cette fonction en Python à l'aide de `def`.

2. A l'aide de la fonction `dicho`, trouver avec une précision $\varepsilon = 10^{-8}$

   (a) la solution de l'équation $\ln(x) = \sin(x)$, $x > 0$ (on proposera des valeurs pour a et b);

   (b) les solutions de l'équation $2x + 3 = e^x$, $x \in \mathbb{R}$ (on pourra utiliser une représentation graphique pour trouver des intervalles contenant les solutions).

3. Écrire une procédure `dicho_nbr(f, a, b, eps)` qui renvoie le nombre d'itérations effectuées dans la méthode de dichotomie (avec les arguments habituels `f, a, b, eps`).

   Combien d'itérations faut-il effectuer pour résoudre chaque équation de la question précédente ?

Exercice 9. *Méthode de Newton*

1. Écrire une procédure `newt(f, df, x, eps)` qui implémente la méthode de Newton, où `f` est la fonction à traiter, `df` sa dérivée (calculée et codée à la main), `x` est le point de départ proposé et `eps` est la précision souhaitée. a procédure retournera un zéro (approché) de la fonction et le nombre d'itérations effectuées.

2. Refaire l'exercice 1 en utilisant la méthode de Newton.

   Comparer le nombre d'itérations nécessaires pour la méthode de Newton et celle de la dichotomie.

Exercice 10. *Intégration numérique*

Écrire une procédure `int_rect_g(f, a, b, n)` qui intègre, par la méthode des rectangles à gauche, la fonction f sur l'intervalle $[a, b]$ en utilisant une division avec `n+1` points (`n` rectangles). On utilisera ici la commande `sum` et **aucune** boucle `for`.

Puis écrire une procédure `int_rect_d` qui applique la méthode des rectangles à droite et une procédure `int_trap` qui utilise la méthode de trapèzes (bien entendu, sans aucune boucle `for`).

1. Calculer une valeur approchée de $\displaystyle\int_0^{\pi/2} \sin(x)dx$ à l'aide de vos trois procédures avec $n = 100$ (rectangles ou trapèzes).

   Réponses : $S_g = 0.992125456606$, $S_d = 1.00783341987$, $S_t = 0.99997943824$.

2. Intégrer par une méthode de rectangles les fonction suivantes : $f(x) = e^{\sin(x)}$ sur $[-\pi, \pi]$ puis $g(x) = \exp(-x^2)$ sur $[-10, 2]$. Refaire avec un nombre de rectangles $n = 10^k$, k variant entre 1 et 7 (inclus). Observer les résultats.

3. La commande `quad(f, a, b)` intègre "automagiquement" une fonction `f` sur un intervalle $[a, b]$. Pour l'utiliser, il faut d'abord l'importer : `from scipy.integrate import quad`.

   Comparer les résultats de la question précédente avec ceux donnés par la commande `quad`.