TP n° 4 : Exercices d'oral avec numpy et scipy

MP Lycée Camille Jullian

2-9 décembre 2021

Ce nouveau TP de révisions est constitué d'exercices posés à divers oraux faisant intervenir de la programmation sur Python. Ils nécessitent parfois d'utiliser certaines fonctionnalités des modules numpy ou scipy, ce qui me donne l'occasion de faire quelques rappels les concernant. Commençons donc par une liste de choses utiles concernant le module numpy :

- on importe traditionnellement le module numpy à l'aide de la commande **import numpy as np**, ce qui le renomme donc np.
- le type de base quand on utilise numpy est le tableau (**array**) qui peut être multidimensionnel et sert donc aussi bien à gérer des listes que des matrices. Pour créer un tel tableau, on utilise tout simplement la commande **np.array(L)**, l'argument L étant une liste (si on veut créer un tableau à une seule dimension), ou une liste de listes (pour créer une matrice). Attention, un tableau numpy ne doit contenir que des éléments du même type, contrairement aux listes Python. Par ailleurs, la taille d'un tableau numpy n'est pas modifiable
- les fonctions **np.zeros((n,p))** et **np.ones((n,p))** permettent de créer une matrice n lignes p colonnes dont tous les éléments sont égaux à 0 ou 1, respectivement.
- la fonction **np.eye(n)** permet de créer une matrice identité de taille n.
- la fonction **np.arange(debut,fin,pas)** fonctionne comme le classique **range** de Python, avec la possibilité supplémentaire d'indiquer des pas qui ne sont pas nécessairement entiers.
- la fonction **np.linspace(debut,fin,N)** permet de créer une liste de N valeurs régulièrement espacées entre début et fin (particulièrement pratique pour créer des listes de valeurs dans le but de tracer une courbe avec matplotlib.pyplot par exemple). La valeur fin est incluse dans la liste (contrairement aux range ou arange).
- les opérations du type $M + x$, $M * x$, M/x, où M un tableau numpy et x un flottant, seront interprétées par Python comme une opération à effectuer terme à terme sur tous les éléments du tableau.
- le produit des matrices M et N s'effectue à l'aide de la commande **M.dot(N)** (attention, $M * N$ renverra le produit terme à terme des deux matrices, aussi connu sous le nom de produit de Hadamard).
- les commandes **M.size()** et **M.shape()** renvoient respectivement le nombre d'éléments et le couple (n, p) (nombre de lignes et de colonnes) de la matrice M.
- le module numpy contient toutes les fonctions mathématiques usuelles, qu'on peut appliquer directement à tous les éléments d'une matrice, par exemple **np.exp(M)** remplacera tous les éléments de M par leur exponentielle (mais ne calculera pas l'exponentielle matricielle de M !).
- les commandes **np.any** et **np.all** permettent de tester facilement si au moins un élément (respectivement tous les éléments) d'une matrice vérifie une certaine condition. Par exemple **np.any(M>10)** renverra True si et seulement si la matrice M contient au moins un élément strictement supérieur à 10.
- les méthodes **sum**, **prod**, **max**, **min** font ce à quoi on s'attend sur un tableau numpy.
- on dispose également dans numpy d'une quantité de fonctions ou méthodes permettant de faire de l'algèbre linéaire avec nos matrices, souvent disponibles dans le sous-module linalg : **M.trace()**, **np.linalg.matrix_power(M,k)**, **np.linalg.det(M)**, **np.linalg.inv(M)**,

np.linalg.eigvals(M) (valeurs propres), **np.linalg.eig(M)** (pareil, mais fournit les vecteurs propres en supplément gratuit)

Pour ceux qui souhaiteraient une référence un peu plus détaillée et complète sur ce sujet, un très bon poly en français disponible sur le web :
https ://perso.univ-perp.fr/langlois/images/pdf/mp/www.mathprepa.fr-une-petite-reference-numpy.pdf

Tant qu'on y est, rappelons l'existence du module scipy et de quelques fonctionnalités intéressantes pour tout ce qui touche à l'analyse numérique :

- le sous-module **scipy.integrate** permet à la fois de faire du calcul numérique d'intégrales, et de la résolution d'équations différentielles. Parmi les fonctions classiques de ce sous-module, **quad** (calcul d'intégrales), **trapz** (calcul d'intégrales par la méthode des trapèzes), **simps** (calcul d'intégrales par la méthode de Simpson), et surtout **odeint** (résolution d'équas diffs). Je vous laisse consulter la documentation facilement trouvable en ligne pour retrouver la syntaxe précise de ces fonctions si besoin.
- le sous-module **scipy.optimize** contient des outils de résolution d'équation : **newton** cherche une solution d'équation du type $f(x) = 0$ par la méthode de Newton, **brentq** fait la même chose avec une méthode plus obscure mais encore plus efficace. Sinon, **root** fonctionne également.

Il est temps de passer aux exercices. Attention, dans certains exercices, il y a de vraies questions de maths (puisque ces exercices ont été pour certains posés à des oraux de type maths/info), si vous ne voulez vraiment faire que du Python, sautez les questions correspondantes.

Exercice 1 : Pour s'échauffer avec Numpy.

On s'intéresse à la matrice $M = \begin{pmatrix} 4 & 5 & 6 & -1 \\ 5 & 10 & 15 & 2 \\ 6 & 15 & 1 & 4 \\ -1 & 2 & 4 & -2 \end{pmatrix}$.

1. Pourquoi M est-elle diagonalisable ? Donner les valeurs propres et vecteurs propres de M.
2. Calculer l'inverse de A en exploitant les résultats précédents, puis en utilisant directement la fonction Python idoine. Comparer les résultats obtenus.
3. Effectuer le même travail sur la matrice $M = \begin{pmatrix} 2 & -1 & 0 & \dots & \dots & 0 \\ -1 & 2 & -1 & \ddots & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & -1 & 2 & -1 \\ 0 & \dots & \dots & 0 & -1 & 2 \end{pmatrix}$. On testera pour une matrice de taille 5, puis 10, puis 50.
4. Comparer les valeurs propres de la matrice précédente avec les valeurs de $4\sin^2\left(\frac{k\pi}{2(n+1)}\right)$.

Exercice 2 : Manipulations faciles de nombres entiers.

1. Soit l'entier $n = 1\,234$. Quel est le quotient, noté q, dans la division euclidienne de n par 10 ? Quel est le reste ? Que se passe-t-il si on recommence la division par 10 à partir de q ?
2. Écrire une suite d'instructions calculant la somme des cubes des chiffres de 1 234.

3. Écrire une fonction **somcube** renvoyant la somme des cubes des chiffres d'un entier quelconque.
4. Trouver tous les entiers inférieurs à 1 000 égaux à la somme des cubes de leurs chiffres.
5. Écrire une nouvelle version de la fonction somcube utilisant une conversion en chaîne de caractères.

Exercice 3 : Manipulations de listes.

Soit n un entier naturel et l une liste de longueurs n constituée uniquement de 0 et de 1. On veut écrire un programme déterminant le nombre maximal de 0 consécutifs dans la liste l.

1. Écrire une fonction **nombreszeros(l,i)** prenant en paramètres une liste l et un entier i, et renvoyant 0 si $l[i] = 1$, le nombre de zéros consécutifs de la liste l à partir de $l[i]$ sinon.
2. En déduire une fonction **nombrezerosmax(l)** renvoyant le nombre maximal de 0 consécutifs dans une liste l (on utlisera la fonction programmée à la question précédente).
3. Quelle est la complexité de la fonction précédente ?
4. Trouver un moyen simple, toujours en utilisant la fonction nombrezeros (Ndprof : ou sans l'utiliser, ce qui me semble à vrai dire plus simple), d'obtenir un algorithme plus performant.

Exercice 4 : Matrices et polynômes.

Soit n un entier supérieur ou égal à 1 et $a \in \mathbb{R}^*$. On note $M_{n,a} = \begin{pmatrix} 0 & \frac{1}{a} & 0 & \dots & 0 \\ a & 0 & \frac{1}{a} & \ddots & \vdots \\ 0 & a & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & 0 & \frac{1}{a} \\ 0 & \dots & 0 & a & 0 \end{pmatrix}$.

1. Écrire une fonction Python affichant la matrice $M_{n,a}$.
2. Donner des valeurs approchées des valeurs propres de $M_{n,a}$ pour $3 \leqslant n \leqslant 8$ et $a \in \{-2, -1, 1, 2, 3\}$.
3. On note (P_n) les polynômes définis par $P_1 = X$, $P_2 = X^2 - 1$ et $\forall n \in \mathbb{N}$, $P_{n+2} = XP_{n+1} - P_n$.
   (a) Calculer les coefficients des polynômes $P_3, \dots, P_8$.
   (b) Donner des valeurs approchées des racines de ces polynômes.
   (c) Conjecturer un lien avec les matrices $M_{n,a}$ et le démontrer.
4. Les matrices $M_{n,a}$ sont-elles inversibles ? Diagoonalisables ?
5. Trouver un segment de $\mathbb{R}$ contenant toutes les valeurs propres de toutes les matrices $M_{n,a}$.

Exercice 5 : Avec un peu de probabilités.

Pour tous réels (a, b), on définit une matrice $M_{a,b} = \begin{pmatrix} 3a - 2b & -6a + 6b + 3 \\ a - b & -2a + 3b + 1 \end{pmatrix}$. On notera par ailleurs $e(a, b)$ l'écart (en module) entre les deux valeurs propres complexes de $M_{a,b}$.

1. Écrire une fonction Python **ecart** qui renvoie la valeur de $e(a, b)$.
2. Soient A et B deux variables aléatoires indépendantes de même loi $\mathcal{G}(p)$, avec $p \in]0, 1[$. Écrire une fonction **hasard** qui simule une occurrence du couple (A, B).
3. Écrire une fonction **ecartalea** qui réalise la simulation de 500 couples de valeurs (a, b) du couple de variables (A, B) et renvoie le nombre de fois où ecart$(a, b) \geqslant 0.1$.

4. Pour $p = \dfrac{1}{100}, \dfrac{2}{100}, \ldots, \dfrac{99}{100}$, relier les points de coordonnées $\left(p, \dfrac{ecartalea(p)}{500}\right)$.
5. Sur le même graphe, tracer la courbe de la fonction $p \mapsto \dfrac{2-2p+p^2}{2-p}$, commenter.

Exercice 6 : Suite de fonctions.

Soit N un nombre entier (par défaut égal à 101), qui correspondra à un échantillonage de l'intervalle $[0,1]$ en N points. Pour tout entier n, on définit sur $[0,1]$ une fonction f_n par $f_n(x) = n\sin^2(n\pi x)(1-x)^n$.

1. Écrire une fonction **maxiliste** qui renvoie à la fois le maximum de la liste donnée en argument, et le premier indice où ce maximum est atteint.
2. Écrire une fonction **fonctionsup(f,N)** prenant comme un arguments une fonction f définie sur $[0,1]$ et un entier N et retournant le maximum de f sur les points d'échantillonage de $[0,1]$ correspondant à l'entier N, ainsi que l'abscisse correspondante.
3. Écrire une fonction **F(n,x)** calculant simplement $f_n(x)$.
4. Écrire une fonction **Graph** qui affiche simultanément le graphe sur $[0,1]$ de toutes les fonctions f_n, où n appartient à une liste d'entiers donnée en argument. On testera avec $l = [0,1,2,3,4,5,6]$.
5. Écrire une fonction **PlusGrand(M,N)** qui renvoie l'indice n de la première fonction f_n dont le maximum sur $[0,1]$ est supérieur à M, ainsi que l'abscisse et l'ordonnée du point correspondant. Tester cette fonction avec $M = 2$ puis avec $M = 10$.
6. Écrire une suite d'instructions Python permettant de trouver la plus petite valeur de n pour laquelle $\max_{x\in[0,1]} f_n(x) \geqslant 20$ et $\forall x \in [0.1, 1]$, $f_n(x) \leqslant 10^{-8}$. Essayer avec plusieurs valeurs de N et commenter.

Exercice 7 : Matrices et polynômes (bis).

On note χ_n le polynôme caractéristique de la matrice $A_n = \begin{pmatrix} 0 & 0 & \ldots & 0 & \frac{1}{n} \\ 1 & \ddots & \ddots & \vdots & \vdots \\ 0 & \ddots & \ddots & 0 & \vdots \\ \vdots & \ddots & \ddots & 0 & \vdots \\ 0 & \ldots & 0 & 1 & \frac{1}{n} \end{pmatrix}$.

1. Écrire une fonction Python prenant pour paramètre n et renvoyant les coefficients de χ_n sous forme de tableau numpy (on pourra consulter la documentation du sous-module **numpy.poly** pour y trouver des choses intéressantes).
2. Afficher les coefficients de χ_n pour $2 \leqslant n \leqslant 8$ et conjecturer la valeur de χ_n. Démontrer ce résultat.
3. Afficher les modules des racines de χ_n (pour les mêmes valeurs de n que précédemment). Que peut-on conjecturer ? Démontrer ce résultat.