Programmation sous `python` – 2024-2025

Feuille d'exercices n° 3

Remarque. Pour les utilisateurs de `spyder`: si on veut avoir les dessins dans une fenêtre séparée, il faut utiliser la commande `%matplotlib qt` dans la console. Pour revenir aux dessins intégrés dans la fenêtre principale (en haut à droite), il faut utiliser la commande `%matplotlib inline`.

Utilisation des bibliothèques scientifiques `numpy` et `matplotlib`

Exercice 1 (numpy)**.** La commande `import numpy as np` donne la possibilité d'utiliser les éléments de la bibliothèque `numpy` ; la bibliothèque est importée comme un objet[1] nommé `np` et met en œuvre une structure de tableaux (ou *arrays* en anglais) qui représentent les vecteurs, matrices et super-matrices habituels.

1) Expérimentez les commandes suivantes dans la console (éventuellement utiliser la commande `print` pour visualiser les résultats).

```
a = [1, 2, 3, -4]
b = [5, -7, 9, 11]
v = np.array(a)
A = np.array([[1, 2, 3, -4]])
M = np.array([a, b, [0, 0, 0, 2]])
type(a), type(v), type(A)
np.shape(v)
np.shape(A)
np.shape(M), np.shape(M.T)
np.size(M)
```

```
w = v + 2*v
v3 = v**3
C = v + A
Mv = M.dot(v)
N = Mv.dot(M)
MAt = M.dot(A.T)
M.dot(A)
A*v

v.T, np.shape(v.T)
```

Que signifie `M.dot(v)` ? Quelle type de variable est le résultat ? Pourquoi la septième commande dans la colonne de droite ne fonctionne-elle pas ? Les vecteurs v et $v.T$ sont-ils égaux ? Comment pourrait-on en décider ?

2) Que représentent $M[0,3]$, $M[2,3]$, $M[2,:]$ et $M[1:,4]$?

3) Étudier les méthodes `np.eye(...)`, `np.zeros(...)` et `np.linspace(...)`.

Exercice 2 (*)**.** On considère la matrice M de l'exercice précédent et on veut résoudre le système linéaire homogène $Mx = 0$ en appliquant le pivot de Gauss. Le premier pivot est identifié comme étant l'élément $(3,4)$ de M.

1) Que représentent du point de vue de l'algorithme de Gauss les opérations $M_1 = D_3(\frac{1}{2})\,M$, $M_2 = P_{1,3}(4)\,M_1$ et $M_3 = P_{2,3}(-11)\,M_2$, où

$$D_3(\tfrac{1}{2}) = \begin{pmatrix} 1 & & \\ & 1 & \\ & & \frac{1}{2} \end{pmatrix}, \qquad P_{1,3}(4) = \begin{pmatrix} 1 & & 4 \\ & 1 & \\ & & 1 \end{pmatrix} \qquad \text{et} \qquad P_{2,3}(-11) = \begin{pmatrix} 1 & & \\ & 1 & -11 \\ & & 1 \end{pmatrix} ?$$

2) Écrire une fonction `tmpP` qui prend en argument n, i, j et x, avec n un entier non nul, i et j des entiers $1 \leq i \neq j \leq n$ et x un réel, et qui rend la matrice carrée de taille $n \times n$, $P_{i,j}(x)$.

3) Écrire une fonction `unPivot` qui prend en argument $A = (a_{ij})$ et pos, avec A une matrice (`np.array`) et $pos = [\alpha, \beta]$ une liste définissant le pivot, et qui rend la matrice obtenue après avoir utilisé le pivot $a_{\alpha\beta}$.

4) Écrire une fonction `pivot` qui prend en argument une matrice A et qui rend une matrice échelonnée associée à A en appliquant l'algorithme du pivot de Gauss.

5) (facultatif) Modifier les fonctions précédentes pour calculer le déterminant d'une matrice carrée.

[1] Tout objet en `python` est muni d'un certain nombre de *propriétés* (ou attributs) et de *méthodes* (ou fonctions). Par exemple, si L est une liste, alors la commande `L.append(1)` invoque la méthode `append` de L avec l'argument 1. On peut voir la liste de toutes les caractéristiques et méthodes d'un objet Ob en utilisant la commande `print(dir(Ob))`, bien qu'elle peut être un peu intimidante.

Exercice 3 (numpy)**.** On veut écrire des fonctions `python` qui renvoie des matrices spéciales.

1) En algèbre linéaire, une *matrice circulante* est une matrice carrée dans laquelle on passe d'une ligne à la suivante par décalage vers la droite des coefficients. Écrire une fonction qui prend en argument une liste (ou une matrice ligne) et qui renvoie la matrice circulante dont la première ligne est donnée par les éléments de la liste. Vérifier la formule

$$C(a_1,\dots,a_n)=\sum_{k=1}^{n} a_k J^{k-1},\quad \text{où}\quad J=\begin{pmatrix}0&1&0&\dots&0\\0&0&1&\dots&0\\\vdots&&&&\\0&0&0&\dots&1\\1&0&0&\dots&0\end{pmatrix}.$$

2) En algèbre linéaire, une *matrice Vandermond* est une matrice carrée dans laquelle chaque ligne est une progression géométrique commençant avec 1. Écrire une fonction qui prend en argument une liste (ou une matrice ligne) et qui renvoie la matrice Vandermond dont les raisons sont données par les éléments de la liste. Vérifier la formule $\det V(a_1,\dots,a_n)=\prod_{i<j}(a_j-a_i)$ où `det` est la fonction déterminant. Voir la commande `det` du module `np.linalg`.

Exercice 4 (matplotlib)**.** Insérer dans le fichier la commande `import matplotlib.pyplot as plt` qui permet l'utilisation des éléments de la bibliothèque `matplotlib.pyplot`. L'outil principal pour le tracé de courbes est la méthode `plot`.

1) Tracer le graphe de $x\mapsto\cos x$ sur $[0,2\pi]$. Pour cela, on construira un vecteur X contenant les coordonnées des abscisses de points entre 0 et 2π (utiliser `linspace`), on construira le vecteur Y des ordonnées en appliquant la fonction `np.cos` à X, et on effectuera le dessin avec `plt.plot`.

2) Écrire une fonction `plotSommeSin2` qui prend en argument un nombre réel A et qui trace (dans le même système de coordonnées) les graphes des fonctions $x\mapsto\sin x$ et de $x\mapsto\sin x+\sin(2x)$ sur l'intervalle $[-A,A]$.

3) Écrire une fonction `plotSommeSin` qui prend en entrée un entier N et un nombre réel A et qui trace le graphe de $x\mapsto\sum_{j=1}^{N}\sin(jx)$ sur l'intervalle $[-A,A]$.

4) Regarder la documentation (sur internet) des méthodes `matplotlib.pyplot.xlabel`, `-.ylabel`, `-.title` et `-.legend` et s'en servir pour détailler les graphiques précédents.

Exercice 5 (lemniscate)**.** Le lemniscate de Bernoulli est la courbe paramétrée donnée en coordonnées cartésiennes par

$$\big(x(t),y(t)\big)=\frac{a\sqrt{2}\,\cos t}{\sin^2 t+1}\,(1,\sin t),$$

où $a>0$ et $t\in[0,2\pi]$.

1) Écrire une fonction `plotLemniscate` qui prend en argument une constante a et qui trace la courbe lemniscate basée sur une subdivision de $[0,2\pi]$ en 1000 sous-intervalles équidistants. (On pourrait utiliser les fonctions `np.cos` et `np.sin` appliquées à un vecteur avec `linspace` — on appelle cela, des *opérations vectorisées*).

2) Écrire une fonction `plotLemniscates` qui prend en argument une liste de nombres réels (différentes valeurs de a) et qui superpose les lemniscates correspondantes. On veillera à ajouter une légende adaptée.

Exercice 6. On considère la parabole $\Gamma : y=x^2$, une direction $u=(\sin\alpha,\cos\alpha)$, $\alpha\in\left]-\frac{\pi}{2},\frac{\pi}{2}\right[$, et un entier N (pour commencer on peut prendre $N=10$). On veut dessiner dans un même système de coordonnées (dans le rectangle $[-2,2]\times[-0.2,5]$) les $2N+1$ courbes suivantes.

1) La parabole Γ pour $x\in[-2,2]$.

2) On considère la subdivision de $[-2,2]$ en N intervalles équidistants (utiliser la commande `linspace` de `numpy`) donnés par les points $x_0=-2,\dots,x_N=2$.

a) Pour chaque x_k, dessiner la demi droite passant par $P_k=(x_k,x_k^2)$ de direction u, à partir de P_k vers $y>0$. Utiliser la couleur bleu et une opacité assez petite.

b) On considère cette droite comme étant un rayon lumineux (provenant d'une source lumineuse à l'infini dans la direction u) qui sera réfléchi par la parabole en P_k. Dessiner la demi droite réfléchie à partir de P_k vers l'intérieur de la parabole. Utiliser la couleur rouge.

La courbe obtenue à partir des rayons réfléchie est une *caustique*.

3) Que remarquez-vous quand $\alpha = 0$?

4) Comment varie la caustique lorsque la direction de la source tourne dans le sens trigonométrique ?

Exercice 7. Nous allons apprendre comment importer et transformer une image dans `python`. Commencer par télécharger une image au format `.png` ou `.jpg` et la placer dans le même dossier du fichier `python` actuel. On suppose par la suite que le nom du fichier est `image.png`.

1) Regarder (sur internet) la documentation des fonctions `matplotlib.pyplot.imread` et `-.imshow`.

2) Utiliser la commande `x = plt.imread("image.png")` pour importer l'image dans `python`. Quelle est la nature de `x` ?

3) On peut rendre une image plus "lisse" en interpolant de nouvelles valeurs à partir des anciennes. Essayer par exemple `plt.imshow(x, interpolation='bilinear')`.

4) Retrouver la dimension de l'image à l'aide de la fonction `np.shape` ; la première valeur est le nombre de pixels selon la hauteur, la deuxième le nombre de pixels selon la largeur et la troisième le nombre de composantes pour coder une couleur[2]. Zoomer sur une partie de l'image en choisissant une partie de `x`.

5) Utiliser la fonction `numpy.flipud` pour inverser l'image.

6) Essayer des opérations vectorielles sur `x` et afficher les résultats sous forme d'images. Par exemple remplacer les composantes de chaque couleur par leurs moyenne, ou bien enlever la composante *rouge*...

Exercice 8 (déterminant)**.**

1) Écrire une fonction `det2d(M)` prenant une matrice M de taille 2×2 en entrée et renvoyant son déterminant.

2) Écrire une fonction `det3d(M)` prenant une matrice M de taille 3×3 en entrée et renvoyant son déterminant, qui utilise `det2d` pour faire ses calculs.

3) Généraliser le programme pour obtenir une fonction `detNd(M)` prenant une matrice M de taille $n \times n$ en entrée et renvoyant son déterminant par une approche récursive. Qu'observez-vous ?

4) Comparer vos résultats avec la commande `det` du module `np.linalg`.

[2] Il y a trois composantes pour chaque couleur, trois entiers R, G et B compris entre 0 et $255 = 2^8 - 1$.