TP13 Python - Manipulations de matrices

Ouvrir Pyzo (ou votre IDE favori), ouvrir un fichier .py dans l'éditeur et le sauvegarder dans un dossier approprié.

Le but de ce TP est d'appréhender le calcul matriciel à l'aide de Python. Nous utiliserons d'abort la structure de liste, centrale en Python, pour reconstruire les outils du calcul matriciel. Puis nous présenterons le module *numpy.array*, sur lequel est prédéfini un certain nombres d'opérations. Ce sera votre outil de référence pour manipuler les matrices par la suite. Enfin, nous présenterons la bibliothèque *linalg*, permettant d'obtenir un certain nombre de notions liées à l'alg´bre linéaire.

1 Matrices à travers le type *list*

1.1 Rappels sur le type *list*

Une liste en Python est une séquence d'oblets qui peuvent être hétérogènes. Elle se définit par des crochets, et les objets sont séparés par des virgules. On peut définir une liste :

— Explicitement : `M1=[1, 2, 3, 4, 5]`
— Ou en compréhension : `M2=[k**2 for k in M1]`

Il existe un certain nombres de *méthode* que l'on peut appliquer à une liste (vous pouvez les afficher avec la commande `dir(lit)`). On retiendra en particulier :

— Ajouter un élément à la fin de la liste `M1` : `M1.append(6)`.
— Ajouter une liste à la fin de la liste `M2` : `M2.extend([25, 36])`.

Notez bien que ces deux commandes modifient la liste en place : la variable `M2` réfère la liste modifiée, mais celle-ci ne sera pas affichée si vous ne le demandez pas.
Pour accéder aux élèments d'une liste, ceux-ci sont numérotés. Attention : la numérotation en Python commence toujours à 0 !

— `M2[3]` renverra 16.
— `M1[1:4]` renverra la lsite des éléments de `M1` entre les positions 1 et 4-1=3, soit ? ?
— On peut les modifier en place : `M2[1]='Saint-Jean de Passy'`
— Pour obtenir la longueur d'une liste, on al la commande universelle `len(M2)`.
— La commande `for` permet de parcourir une liste :

```
s=0
for k in M1:
     s=s+k
print(s)
```

1.2 Matrices comme listes de listes

On peut choisir de représenter une matrice par une liste de listes : chaque entrée correspond à une ligne. Ainsi,

```
M=[[1, 2, 3],[4, 5, 6]]
```

représenterait la matrice $\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$. Que renverra `M[1][2]` ? Et `M[2][2]` ?

Création d'une matrice Si on veut créer la matrice nulle de taille $n \times p$, on peut procéder comme suit :

```
def nulle(n,p):
    M=[None]*n          # on crée un tableau de taille n, initialisé avec None
    for i in range(n):
         M[i]=[0]*p     # on affecte à chaque ligne un tableau de taille p rempli de 0
    return M
```

Exercice 1

1. Créer une fonction `dim(M)` qui prend en argument une liste `M` et renvoie un couple d'entiers donnant la dimension de la matrice représentée par `M`.

2. Tester votre fonctions avec les matrices définies ci-dessus.

Exercice 2

créer une fonction `id(n)` qui prend en argument un entier n et renvoie une liste représentant I_n.

Exercice 3

1. Rappeler la formule du produit matriciel.
2. Créer une fonction `prod(A,B)` qui renvoie le produit matriciel de deux matrices A et B si celui-ci existe, et un message d'erreur sinon.
3. Tester votre fonction avec les matrices

$$A = \begin{pmatrix} 2 & -4 \\ 1 & 2 \end{pmatrix},\ B = \begin{pmatrix} 1 & 1 & 2 \\ 3 & -1 & -2 \end{pmatrix},\ \text{ et } C = \begin{pmatrix} -1 & 1 \\ -2 & 0 \\ 2 & -1 \end{pmatrix},$$

et avec vos fonctions `id` et `nulle`.

2 Le type *array*

La bibliothèque *numpy* offre l'environnement *array*, très similaire à la liste, mais avec quelques fonctionnalités assez pratiques en sus.

2.1 Création de matrices avec *array*

Commencez, comme toujours, par importer la bibliothèque *numpy* :

```
import numpy as np
```

Une matrice peut se créer avec le type *array* comme avec le type *list*, en précisant en plus qu'on utilise ce type :
— De manière explicite : `A=np.array([[1, 2, 3],[4, 5, 6]])`
— En compréhension : `B=np.array([[i for j in range(3)] for i in range (4)])`
Mais il existe aussi des commandes pour certaines matrices. On retiendra :
— `np.zeros((n,p))` : la matrice nulle.
— `np.ones((n,p))` : une matrice qui ne contient que des 1.
— `np.eye(n)` : la matrice identité.

2.2 Opérations sur les matrices

Une bonne partie des opérations usuelles sont déjà implémentées dans la bibliothèque *numpy*.
— Dimension d'une matrice : `n,p=np.shape(A)`
— Transposition : `np.transpose(A)`.
— Les opérations usuelles `+ , - , * , ** , /`, posées entre deux matrices, sont eefectuées **coefficient par coefficient.**
— Le **produit matriciel** d'obtient par la commande `np.dot(A,B)`.
— Accéder au coefficient d'indice (i, j) d'une matrice A : `A[i][j]`. On peut modifier sa valeur par affectation (`A[i][j]=5` par exemple).

Exercice 4

1. En utilisant la commande *np.array*, créer deux objets `A` et `B` représentant les matrices

$$A = \begin{pmatrix} -2 & 6 \\ 3 & -4 \end{pmatrix} \text{ et } B = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}.$$

2. Prédire les objets obtenus avec les commandes suivantes, puis tester vos réponses.

```
A+B    A-B    A*B    A**B    A/B    np.dot(A,B)
```

Exercice 5

On pose $A = \begin{pmatrix} 1 & 3 & -2 \\ 0 & 4 & -3 \\ -2 & 1 & 0 \end{pmatrix}$.

1. À l'aide de Python, calculer $A^3 - 5A^2 + 3A$.
2. En déduire que A est inversible et exprimer son inverse en fonction de A.
3. Calculer A^{-1} à l'aide de Python.

Exercice 6 Trace d'une matrice

1. Écrire une fonction `trace(M)` qui prend en entrée une matrice et renvoie sa trace.
2. Tester votre fonction avec la matrice identité, la matrice de l'exercice précédant, et la matrice $\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$.

3 Le pivot de Gauß

Dans cette section, on souhaite implémenter la méthode du pivot pour aboutir, en partant d'une matrice quelconque, à sa forme réduite échelonnée. On rappelle que la forme réduite échelonnée d'une matrice A est l'unique matrice que l'on peut obtenir avec les opérations élémentaires vérifiant les propriétés suivantes :

P1 Si une ligne est non nulle, le premier coefficient non nul est un 1. C'est le *pivot* de la ligne.

P2 Si une colonne contient un pivot, alors tous les autres coefficients de la colonne sont nuls.

P3 Si une ligne i contient un pivot, toutes les lignes au-dessus contiennent un pivot, et ceux-ci sont à gauche du pivot de la ligne i.

Les opérations élémentaires permettant d'obtenir la forme réduite échelonnée sont :

— La *permutation* de deux lignes : $L_i \leftrightarrow L_j$.

— La multiplication par un scalaire non nul d'une ligne :$L_i \leftarrow \mu L_i$.

— La *transvection* de lignes : $L_i \leftarrow L_i + \mu L_j$

Exercice 7

1. Écrire une fonction `echangeL(A,i,j)` qui échange les lignes i et j d'une matrice A. Tester votre fonction sur des matrices déjà définies.
2. Écrire une fonction `transvection(A,i,j,mu)` qui effectue la transvection $L_i \leftarrow L_i + \mu L_j$ dans la matrice A.

* 3. Écrire une fonction `rref(A)` qui renvoie la forme réduite échelonnée d'une matrice. Tester votre fonction avec les matrices suivantes :

$$A_1 = \begin{pmatrix} 2 & 4 & -2 & 2 & 4 \\ 1 & 2 & -1 & 2 & 0 \\ 3 & 6 & -2 & 1 & 9 \\ 5 & 10 & -4 & 5 & 9 \end{pmatrix}, \quad A_2 = \begin{pmatrix} 1 & -3 & 0 & -5 \\ 3 & -12 & -2 & -27 \\ -2 & 10 & 2 & 24 \\ -1 & 6 & 1 & 14 \end{pmatrix}.$$

En déduire le noyau de chacune de ces matrices.

4 La bibliothèque *numpy.linalg*

La bibliothèque *numpy.linalg* propose un certain nombre de routines fort utiles en algèbre linéaire. On l'appellera :

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

Il y a quatre commandes que vous devez connaître de la bibliothèque *numpy.linalg*. À chaque fois, les matrices ou vecteurs impliqués doivent être de type *np.array*

1. Résolution d'un système linéaire $Ax = b$: `al.solve(A,b)`. La matrice A doit être inversible, les dimensions respectives de A et b doivent correspondre.

2. Calcul du rang d'une matrice A : `al.matrix_rank(A)`.
3. Calcul de l'inverse d'une matrice inversible A : `al.inv(A)`
4. Calcul de puissances d'une matrice carrée A, pour calculer A^5 par exemple : `al.matrix_power(A,5)`

Exercice 8

On considère les deux systèmes suivants :

$$\begin{cases} 2x+8y+4z=2 \\ 2x+5y+z=5 \\ 4x+10y-z=1 \end{cases} \qquad \begin{cases} 3x+21y-3z=0 \\ -6x-2y-z=62 \\ 2x-3y+8z=32 \end{cases}$$

1. Les écrire sous forme matricielle.
2. Vérifier, à l'aide de la commande `rank`, qu'ils sont de Cramer.
3. Les résoudre en utilisant la commande `solve`.
4. Les résoudre en utilisant la commande `inv`.
5. On considère maintenant la matrice $A_1 = \begin{pmatrix} 2 & 4 & -2 & 2 & 4 \\ 1 & 2 & -1 & 2 & 0 \\ 3 & 6 & -2 & 1 & 9 \\ 5 & 10 & -4 & 5 & 9 \end{pmatrix}$.
   a) Donner le rang de cette matrice à l'aide de la commande `rank`.
   b) La commande `solve` vous permet-elle de calculer le noyau de cette matrice ?

Exercice 9 Suites récurrentes linéaires

1. Ordre 1 : soit (u_n) définie par :

$$u_0 \in \mathbb{R}, \quad \text{et } \forall n \in \mathbb{N},\ u_{n+1} = qu_n.$$

   a) Rappeler une expression explicite de u_n en fonction de n, la raison q, et le premier terme u_0.
   b) Écrire une fonction Python `geom(q,u0,n)` qui prend en entrée une raison q, un premier u_0, et un entier n, et renvoie le $n^{\text{ème}}$ terme de la suite géométrique correspondante. On donnera deux scripts différents, l'un utilisant la relation de récurrence, l'autre la formule explicite.
2. Ordre 2 : on considère une suite récurrente lnéaire d'ordre 2 définie par $u_0, u_1 \in \mathbb{R}$ et :

$$\forall n \in \mathbb{N},\ u_{n+2} = u_{n+1} + u_n. \tag{1}$$

   a) Rappeler ou retrouver l'expression explicite de u_n en partant de $u_0 = 0, u_1 = 1$.
   b) On retourne au cas général : écrire une fonction `fibo(u0,u1,n)` qui renvoie le $n^{\text{ème}}$ terme de la suite définie par (1).
   c) Formulation matricielle : pour $n \in \mathbb{N}$, on pose $U_n = \begin{pmatrix} u_n \\ u_{n+1} \end{pmatrix} \in \mathbb{R}^2$. Montrer qu'il existe une matrice $A \in \mathcal{M}_2(\mathbb{R})$, que l'on déterminera, telle que $U_{n+1} = AU_n$.
   d) En déduire une expression de U_n en fonction de A et U_0.
   e) En utilisant la commande `al.matrix_power`, donner le vingtième terme de la suite de Fibonacci.
3. On considère maintenant la suite récurrente définie par :

$$\forall n \in \mathbb{N},\ u_{n+2} = u_{n+1} - u_n. \tag{2}$$

   Pour $n \in \mathbb{N}$, on pose $U_n = \begin{pmatrix} u_n \\ u_{n+1} \end{pmatrix} \in \mathbb{R}^2$.
   a) Expliciter la matrice B telle que $U_{n+1} = BU_n$.
   b) Afficher les trente premiers termes de la suite en partant de $(u_0, u_1) = (1, 3)$, puis de $(u_0, u_1) = (-5, 4)$. Conjecturer le comportement de la suite (u_n) dans le cas général.
   c) À l'aide de la commande `al.matrix_power`, vérifier votre conjecture.