TP 20 : algèbre linéaire en Python, SVD et pseudo-inverse

L'objectif du TP est de résoudre numériquement des systèmes linéaires. Pour ce faire, on utilisera la représentation matricielle des systèmes. Autrement dit, si on cherche à résoudre le système

$$\begin{cases} a_{11}x_1 & + & \cdots & + & a_{1n}x_n & = & b_1 \\ \vdots & & \ddots & & \vdots & & \vdots \\ a_{p1}x_1 & + & \cdots & + & a_{pn}x_n & = & b_p \end{cases}$$

d'inconnues $x_1, x_2, \ldots, x_n$, il faudra donner à Python les **matrices colonnes représentant ces objets dans les bases canoniques**. Ici on définira donc un vecteur numpy `Y` et une matrice `A` de tailles respectives $p \times 1$ et $p \times n$, et on considérera l'équation matricielle $AX = Y$ d'inconnue $X \in \mathcal{M}_{n,1}(\mathbf{R})$.

1. Résolvez à la main le système suivant

$$\begin{cases} x & + & y & - & z & = & 8 \\ x & + & 2y & + & 3z & = & 5 \\ 3x & - & 3y & - & z & = & 2 \end{cases}$$

2. La commande `solve` du module numpy.linsolve de Python permet de résoudre un système linéaire. Elle s'utilise de la manière suivante :

```python
import numpy as np
import numpy.linalg as al

A = np.arrray([[1,3],[2,4]])
Y = np.array([[-1],[1]])

X = al.solve(A,Y)
```

On obtient l'unique solution X de l'équation $AX = Y$
(après avoir chargé le module d'algèbre linéaire de numpy sous le nom `al`).

Utilisez `al.solve` pour résoudre le système de la question 1. numériquement et vérifiez votre résultat.

3. La solution dans le cas où A est inversible est facile à obtenir : en effet

$$AX = Y \;\Leftrightarrow\; X = A^{-1}Y \; .$$

À l'aide de la commande `al.inv(A)`, qui renvoie l'inverse de la matrice A, vérifiez que la solution proposée par Python dans les questions 1. et 2. sont bien les bonnes.
(on rappelle que le produit matriciel de A par B s'obtient avec `np.dot(A,B)` ou bien `A.dot(B)`)

4. On considère le système suivant :

A n'est pas inversible.
(rang 4)

$$\begin{cases} 3x & + & y & - & 5z & + & v & - & 12u & = & 6 \\ & & 5y & & & & & + & 3u & = & 15 \\ 4x & - & 5y & & & & +3v & & & = & 2 \\ 6x & - & y & + & 5z & - & 3v & - & 5u & = & 0 \end{cases}$$

Tentez de résoudre ce système avec Python. Que se passe-t-il ? Vérifiez l'inversibilité de la matrice. La commande Python `al.matrix_rank(A)` renvoie le rang de la matrice. Nous verrons qu'une matrice carrée de taille n est inversible si et seulement si son rang est égal à n. Calculez avec Python le rang des matrices vues précédemment.

5. Si le rang d'une matrice n'est pas maximal, alors son noyau n'est pas réduit à 0_E. Vérifiez à l'aide du produit matriciel que la matrice de la question 4. a dans son noyau la colonne
${}^t\begin{pmatrix} 225 & -105 & -401 & -475 & 175 \end{pmatrix}$.

6. On va utiliser Python pour obtenir le noyau de A. La décomposition en valeurs singulières ou SVD est une technique qui permet d'écrire **toute** matrice A (quelque soit sa taille et son rang) sous la forme :

$$A = U\ \Sigma\ V$$

ou Σ est une matrice diagonale, et U et V sont carrées et inversibles, d'inverse égal à leur transposée. Si A n'est pas carrée, ou bien si Σ a des 0 sur la diagonale, alors A ne sera pas inversible et on peut obtenir une base du noyau de A à l'aide de la matrice V. Recopiez et testez la fonction suivante :

```python
def noyau(mat):
    U, S, V = al.svd(mat)                    # fait la SVD sur la matrice
    p, n = mat.shape                         # taille de la matrice
    noyau = []                              # les futurs elements de la base du noyau
    for k in range(len(S)):             # detecte les eventuels 0 de S
        if S[k] == 0 :
            L = np.array([V[k,:]]) # prend la ligne k de V
            noyau.append(L.transpose()) # en fait une colonne et l'ajoute au noyau
    if p<n :
        for k in range(p,n):
            L = np.array([V[k,:]])
            noyau.append(L.transpose())
    return noyau
```

7. Calculez le noyau de la matrice de la question 4. à l'aide de la fonction `noyau` et vérifiez que le vecteur obtenu est bien colinéaire au vecteur proposé en question 6.

8. Résolvez à la main le système suivant :

$$\left\{\begin{array}{rcrcrcrcr} 3x & + & 4y & + & z & + & 2t & = & 3 \\ 6x & + & 8y & + & 2z & + & 6t & = & 7 \\ 9x & + & 12y & + & 3z & + & 10t & = & 11 \end{array}\right.$$

Calculez à l'aide de votre résultat une base du noyau de la matrice

$$A = \begin{pmatrix} 3 & 4 & 1 & 2 \\ 6 & 8 & 2 & 6 \\ 9 & 12 & 3 & 10 \end{pmatrix}$$

9. Calculez le noyau de A à l'aide de Python. Y'a t-il un problème ? Le résoudre.

10. À partir d'une solution particulière du système, comment obtenir l'ensemble des solutions du système ?

11. Le pseudo-inverse de Moore-Penrose, noté A^+ permet d'obtenir une solution particulière du système. Il s'obtient à l'aide de la commande `al.pinv(A)`. Si Y est le second membre du système, alors A^+Y est une solution particulière du système $AX = Y$. Vérifiez que cela fonctionne bien sur le système de la question 9.

12. **Calcul du pseudo-inverse à partir de la SVD**
Si dans la SVD de A, la matrice Σ est inversible, alors A est inversible et son pseudo-inverse est son inverse. Dans le cas général, on a

$$\Sigma = \begin{pmatrix} D & 0_{r,n-r} \\ 0_{p-r,r} & 0_{p-r,n-r} \end{pmatrix} \quad \text{ou} \quad \Sigma = \begin{pmatrix} D & 0_{r,n-r} \end{pmatrix} \quad \text{ou} \quad \Sigma = \begin{pmatrix} D \\ 0_{p-r,r} \end{pmatrix}$$

où D est diagonale et inversible, et r est le rang de A. Le premier des trois cas inclut les deux autres si on autorise $p - r = 0$ ou $n - r = 0$. Le pseudo inverse de Σ alors donné par

$$\Sigma^+ = {}^t\begin{pmatrix} D^{-1} & 0_{r,n-r} \\ 0_{p-r,r} & 0_{p-r,n-r} \end{pmatrix}$$

Le pseudo-inverse de A est alors ${}^tV\ \Sigma^+\ {}^tU$. Codez une fonction Pyhton `pseudo_inverse` et comparez son résultat avec le résultat de `al.pinv`.

Formulaire :

Vous aurez besoin des commandes suivantes (dans le désordre)

```
### LES MODULES USUELS
from math import *
import numpy as np
import matplotlib.pyplot as plt
import numpy.random as rd

### LES VARIABLES ALEATOIRES
rd.randint(a,b,N)
rd.random(N)
rd.binomial(n,p,N)
rd.geometric(p,N)
rd.poisson(lambda,N)

## LES LISTES
np.linspace(a,b,n)
for compteur in liste :
for compteur in range(a,b):
[ f(compteur) for compteur in liste]
[ f(compteur) for compteur in range(a,b)]

### COMMANDES CLASSIQUES
while condition:
def nom_fonction( variables ) :
return resultat
print( 'message', variable )
if condition :
else :
log(x)
plt.plot( abscisses , ordonnees )

### CLASSES EN PYTHON
class nom_de_classe :
class nom_de_classe(classe_d_heritage):
def __init__(self, autres_parametres):
def __str__(self):

### MATRICES EN PYTHON
np.array([[...],...,[...]])
A.shape
np.zeros([p,n])
np.eye(n)
np.ones([p,n])
A.dot(B)
np.dot(A,B)
A.transpose()
A.all()
```