Université Lyon 1 Année 2020-2021
Licence Mathématiques 3e année
Analyse matricielle

TP 1

On aura besoin des modules numpy, matplotlib.pyplot et scipy.stats. Il faudra charger ces modules, en tapant les commandes

```
import numpy
import matplotlib.pyplot
import scipy.stats
import scipy.linalg
```

Exercice 1. (*Manipulation de matrices avec Python*)

On considère les vecteurs et matrices suivants

$$\ell = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \end{pmatrix} \quad v = \begin{pmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{pmatrix} \quad A = \begin{pmatrix} 1 & 2 & 0 & 0 & 0 \\ 0 & 0 & 2 & 3 & 1 \\ 0 & 0 & 0 & 2 & 2 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 0 \end{pmatrix} \quad B = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 2 & 3 & 4 & 5 & 6 \\ 3 & 4 & 5 & 6 & 7 \\ 4 & 5 & 6 & 7 & 8 \\ 5 & 6 & 7 & 8 & 9 \end{pmatrix}$$

1. Vérifier qu'on peut les définir sous Python à l'aide des commandes

```
l = numpy.array([1,2,3,4,5])
m = numpy.array([[1,2,3,4,5]])
v = numpy.transpose(m)
v = numpy.array([[1],[2],[3],[4],[5]])
A = numpy.array([[1,2,0,0,0],[0,0,2,3,1],[0,0,0,2,2],[0,0,0,0,1],[1,1,1,0,0]])
B = numpy.array([[1,2,3,4,5],[2,3,4,5,6],[3,4,5,6,7],[4,5,6,7,8],[5,6,7,8,9]])
```

2. Commenter le résultat produit par les instructions suivantes :

```
l*v, v*l, l/v, A*B, B/A, numpy.transpose(l), numpy.transpose(A),
numpy.dot(A,l), numpy.dot(l,A), numpy.dot(A,v), numpy.dot(v,A),
numpy.dot(A,B), numpy.sin(l), numpy.exp(A), scipy.linalg.expm(A),
numpy.linalg.inv(A) et l + v - 1
```

(on utilisera la commande `print()`).

RÉPONSE :

3. Comment déterminer le format (nombres de lignes et de colonnes) de ces tableaux ? Comparer les résultats de `numpy.size, numpy.shape`.

RÉPONSE :

4. Python permet la génération automatique de tableaux particuliers. À titre d'illustration, analyser et commenter le résultat produit par chacune des instructions suivantes.

```
r = numpy.linspace(1.3,15.8,10)
s = numpy.arange(1,4,0.3)
t = numpy.ones(5)
u = numpy.zeros(10)
C = numpy.eye(3,3)
D = numpy.ones((3,3))
y = numpy.random.rand(5,2)
```

RÉPONSE :

5. Pour $n \geq 2$ on considère la matrice tridiagonale d'ordre n définie par

$$G_n = \begin{pmatrix} 2 & -1 & 0 & \dots & 0 \\ -1 & 2 & -1 & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & -1 & 2 & -1 \\ 0 & \dots & 0 & -1 & 2 \end{pmatrix}$$

(a) Que produit la suite d'instructions ci-dessous ?

```
n = 5
D = numpy.diag(numpy.ones(n))
S = numpy.diag(numpy.ones(n-1),1)
G = 2*D - S - numpy.transpose(S)
```

RÉPONSE :

(b) Que produit la suite d'instructions suivante ?

```
H = A[1,3]
I = A[1:3,3]
J = A[:2,1:]
K = A[2,:]
```

RÉPONSE :

6. En s'inspirant des instructions de la question précédente, extraire la première ligne, la deuxième colonne et la sous-matrice $(a_{ij})_{\substack{1 \leq i \leq 3 \\ 1 \leq j \leq 5}}$ de la matrice A, ainsi que les termes diagonaux de la matrice B.

RÉPONSE :

7. On s'intéressera aux méthodes de recherche de valeurs propres lors d'un prochain TP. Ici, quand on aura besoin de calculer les valeurs propres d'une matrice, ou son rayon spectral, on utilisera les fonctions préprogrammées de Python. À l'aide de la commande `numpy.linalg.eig`, déterminer le rayon spectral de la matrice G_5.

RÉPONSE :

Exercice 2. (*Définitions de fonctions*)

Une fonction se déclare en Python comme suit :

```
def nom-de-fonction(variable_1,variable_2,...):
   instructions...
   return(...)
```

Voici un exemple de fonction permettant de définir la matrice tridiagonale des différences finies pour la dérivée seconde, pour un nombre de points n :

```
def DF(n):
    D = numpy.diag(numpy.ones(n))
    S = numpy.diag(numpy.ones(n-1),1)
    return(2*D - S - numpy.transpose(S))
```

Appeler la fonction pour $n = 6$ avec la commande `A = DF(6)`. Qu'obtient-on ?

RÉPONSE :

Exercice 3. (*Résolution de systèmes linéaires et perturbation des données*)

On s'intéresse à la solution du problème $Ax = b$ où

$$A = \begin{pmatrix} 1 & 0.1 & 0.01 & 0.001 & 0.0001 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1.5 & 2.25 & 3.375 & 5.0625 \\ 1 & 2 & 4 & 8 & 16 \\ 1 & 3 & 9 & 27 & 81 \end{pmatrix} \quad \text{et} \quad b = \begin{pmatrix} 0.01 \\ 1 \\ 2.25 \\ 4 \\ 9 \end{pmatrix}$$

1. Le module numpy de python a une fonction de résolution de systèmes linéaires : la commande `numpy.linalg.solve(A,b)` renvoie le résultat `x` du système ci-dessus. Quel est le résultat obtenu ? Aurait-on pu le trouver sans calcul ?

   RÉPONSE :

2. Rappeler la définition de $\text{cond}_2(A)$ et en donner une valeur approchée en utilisant la commande `numpy.linalg.cond`.

   RÉPONSE :

3. À l'aide de la commande `10**-3*numpy.random.rand(numpy.shape(A)[0],numpy.shape(A)[1])`, définir une matrice $B = A + \delta A$ où δA est définie « aléatoirement ». Résoudre $By = b$ et évaluer la quantité

   $$\frac{\|y - x\|_2}{\|y\|_2} \frac{\||A\||_2}{\||\delta A\||_2}.$$

   en utilisant la commande `numpy.linalg.norm`. Faire ceci pour différentes perturbations δA. Avec quoi peut-on majorer cette quantité (résultat de cours et travaux dirigés) ?

   RÉPONSE :

4. En s'inspirant de la question précédente, calculer la solution z du problème $Az = b + \delta b$ où δb est défini aléatoirement, puis évaluer la quantité

$$\frac{\|z - x\|_2}{\|x\|_2} \frac{\|b\|_2}{\|\delta b\|_2}.$$

Faire ceci pour différentes perturbations δb. Avec quoi peut-on majorer cette quantité ?

RÉPONSE :

5. Soit D la matrice diagonale obtenue à partir de la diagonale de la matrice A avec la commande `D=numpy.diag(numpy.diag(A))`. On pose $E = D^{-1}A$ et $c = D^{-1}b$ et on considère le système linéaire $Ex = c$. Reprendre les questions 3. et 4. avec $\delta E = D^{-1}\delta A$ et $\delta c = D^{-1}\delta b$. Que peut-on en conclure ?

RÉPONSE :

Exercice 4. (*Résolution de systèmes linéaires par méthodes itératives*)

Une méthode itérative pour résoudre le système linéaire $Ax = b$ consiste en une décomposition de la matrice A sous la forme $A = M - N$ puis la définition de la suite

$$\begin{cases} x_0 \in \mathbb{R}^n \\ x_{k+1} = M^{-1}Nx_k + M^{-1}b \quad \text{pour tout } k \in \mathbb{N}. \end{cases}$$

On vérifie sans peine que si la suite est stationnaire ($x_{k+1} = x_k$ pour tout k), x_k est la solution cherchée, car $(M - N)x_k = b$ pour tout k. On identifiera en cours diverses conditions sur M et N sous lesquelles la suite *converge* (pour tout x_0) vers la solution cherchée (quelques indications apparaissent déjà dans la suite de cet exercice).

La méthode de *Gauss-Seidel* est obtenue en posant $M = D - E$ et $N = F$ où D est la matrice diagonale dont la diagonale est celle de A, $-E$ correspond à la partie sous-diagonale de A et $-F$ à sa partie sur-diagonale. On définit donc la suite approximante par

$$\begin{cases} x_0 \quad \text{donné} & \text{(Gauss-Seidel)} \\ \forall k \in \mathbb{N},\ (D-E)x_{k+1} = Fx_k + b \end{cases}$$

Pour la méthode de *Jacobi*, l'itération s'écrit

$$\begin{cases} x_0 \quad \text{donné} & \text{(Jacobi)} \\ \forall k \in \mathbb{N},\ Dx_{k+1} = (E+F)x_k + b \end{cases}$$

1. Utiliser les commandes Python `numpy.tril` et `numpy.triu` pour définir $D,\ E,\ F$ et vérifier que $D - E - F = A$. **Attention aux signes**.

   RÉPONSE :

2. Soit $n \in \mathbb{N}^*$, $B \in \mathcal{M}_n(\mathbb{R})$ et $c \in \mathbb{R}^n$. Soit la suite récurrente définie par :

   $$\begin{cases} x_0 \in \mathbb{R}^n \\ x_{k+1} = Bx_k + c \quad \text{pour tout } k \in \mathbb{N}. \end{cases}$$

   La suite $(x_k)_{k\in\mathbb{N}}$ converge, quel que soit x_0, si et seulement si $\rho(B) < 1$.

   En déduire une condition nécessaire et suffisante pour que les méthodes de Jacobi et Gauss-Seidel convergent. Prédire, à l'aide de Python, le comportement (convergence ou non) de ces deux méthodes pour la matrice A de l'exercice précédent.

   RÉPONSE :

3. Voici une fonction qui calcule `niter` itérations de la méthode de Gauss-Seidel, et qui renvoie la solution approchée `u` et la matrice `res` contenant la suite des itérés calculés : $x_0,\ x_1,\ x_2, \ldots$.

```
def gauss_seidel(A,b,x0,niter):
    D = numpy.diag(numpy.diag(A))
    E = -numpy.tril(A) + D
    F = -numpy.triu(A) + D
    sol = numpy.linalg.solve(A,b)
    u = x0
    res = x0
    resn = numpy.linalg.norm(u-sol)
    for k in range(niter):
        u = numpy.linalg.solve((D-E),(numpy.dot(F,u)+b))
        res = numpy.vstack((res, u))
        resn = numpy.vstack((resn,numpy.linalg.norm(u-sol)))
    return(u,res,resn)
```

Tester cette fonction (choisir `x0` et `niter`). Que contiennent `u`, `res` et `resn` ? Comment afficher l'erreur entre la solution approchée et la solution exacte ?

RÉPONSE :

4. *Vitesse de convergence.* On note pour tout $k \in \mathbb{N}$, $e_k = \|x_k - x\|$, où x est la solution de $Ax = b$, une norme de l'erreur à l'itération k. Lors de l'utilisation d'une méthode itérative, s'il existe deux constantes positives α et C telles que

$$\lim_{k \to +\infty} \frac{e_{k+1}}{(e_k)^\alpha} = C,$$

on dit que l'ordre de la méthode est α et que sa vitesse de convergence est C.
On s'intéresse à l'ordre et la vitesse de convergence de la méthode de Gauss-Seidel. Pour cela, on va tracer des fonctions de l'erreur. Écrire la suite d'instructions suivante

```
x0 = numpy.array([1,5,1,5,1])
n = 50
u,res,resn = gauss_seidel(A,b,x0,n)

matplotlib.pyplot.figure("Étude de la vitesse de convergence de la\
                         méthode de Gauss-Seidel",figsize=(14,8))
matplotlib.pyplot.plot(numpy.log(resn[:n]),numpy.log(resn[1:]))
matplotlib.pyplot.grid()
matplotlib.pyplot.show()

resn = resn.flatten()
pente,oao,_,_,_ = \
     scipy.stats.linregress(numpy.log(resn[:n]),numpy.log(resn[1:]))
print("pente : ",pente)
print("ordonnée à l'origine : ",oao)
```

Exécuter cette suite d'instructions et interpréter la figure tracée.

RÉPONSE :

5. Que calcule la commande `linregress` ? Où, sur la figure, peut-on retrouver les nombres calculés ?

RÉPONSE :

6. Pour accélérer la convergence, on a recours à la méthode de *relaxation* qui consiste à remplacer la récurrence de Gauss-Seidel par

$$\forall k \in \mathbb{N},\ \left(\frac{1}{\omega}D - E\right) x_{k+1} = \left(\frac{1-\omega}{\omega}D + F\right) x_k + b.$$

On se restreindra au cas $0 < \omega < 2$. On peut par ailleurs montrer qu'il s'agit là d'une condition nécessaire à la convergence de la méthode. On admet que la méthode de relaxation est de même ordre que la méthode de Gauss-Seidel. On s'intéresse à sa vitesse de convergence.

(a) En s'inspirant de la fonction `gauss_seidel(A,b,x0,niter)`, écrire une fonction `relaxation(A,b,x0,niter,omega)` qui effectue `niter` itérations de la méthode de relaxation et qui comme au-dessus renvoie un triplet `[u,res,resn]`, où `u` est le vecteur obtenu après `niter` itérations, `res` contient tous les itérés x_k calculés et `resn` la suite des erreurs $\|x_k - x\|_2$ (x solution de référence).

(b) On souhaite comparer la vitesse de convergence pour différentes valeurs du paramètre ω. Le script suivant permet de tracer les différentes courbes de convergence que l'on obtient pour

$$\omega = 0.5,\ 0.8,\ 1.2,\ 1.4,\ 1.6,\ 1.9$$

que l'on compare au cas $\omega = 1$ qui correspond à l'algorithme de Gauss-Seidel classique.

```
x0 = numpy.array([1,5,1,5,1])
n=50
u1,res1,resn1 = gauss_seidel(A,b,x0,n)
u05,res05,resn05 = relaxation(A,b,x0,n,0.5)
u08,res08,resn08 = relaxation(A,b,x0,n,0.8)
u12,res12,resn12 = relaxation(A,b,x0,n,1.2)
u14,res14,resn14 = relaxation(A,b,x0,n,1.4)
u16,res16,resn16 = relaxation(A,b,x0,n,1.6)
u19,res19,resn19 = relaxation(A,b,x0,n,1.9)

matplotlib.pyplot.figure("Étude de la vitesse de convergence de la\
         méthode de relaxation",figsize=(14,8))
matplotlib.pyplot.plot(numpy.log(resn1),"*", label = "omega = 1")
matplotlib.pyplot.plot(numpy.log(resn05),"b:",label = "omega = 0.5")
matplotlib.pyplot.plot(numpy.log(resn08),"r:",label = "omega = 0.8")
matplotlib.pyplot.plot(numpy.log(resn12),"g:",label = "omega = 1.2")
matplotlib.pyplot.plot(numpy.log(resn14),"y:",label = "omega = 1.4")
matplotlib.pyplot.plot(numpy.log(resn16),"m:",label = "omega = 1.6")
matplotlib.pyplot.plot(numpy.log(resn19),"black",label = "omega = 1.9")
leg = matplotlib.pyplot.legend(loc='best', shadow=True, fancybox=True)
matplotlib.pyplot.grid()
matplotlib.pyplot.show()
```

Quel est, parmi ces paramètres, le choix le plus intéressant ? Calculer (à l'aide de Python) le rayon spectral de $\left(\frac{1}{\omega}D - E\right)^{-1}\left(\frac{1-\omega}{\omega}D + F\right)$ pour toutes ces valeurs de ω et conclure.

RÉPONSE :