Analyse Nmérique TP2

DUQUENOY Cyrile CALLEA Angelo *

Université d'Aix-Marseille
2021-2022

L3 de Mathématiques
Second semestre

tp2, Analyse Numérique : Normes et Conditionnement

Abstract

Le but de ce TP est de mettre en pratique les normes et conditionnements de matrices.

A savoir que le conditionnement d'une matrice permet d'évaluer la dépendance du résultat aux données initiales du système.

1 Rappel de cours

On définit le conditionnement d'une matrice par :

$$cond(A) = \|A\|\|A^{-1}\|$$

2 Exercice 1

On prends ici le système à résoudre Ax=b et on regarde le conditionnement de A sans puis avec perturbation de b.

$$A = \begin{pmatrix} 10 & 7 & 8 & 7 \\ 7 & 5 & 6 & 5 \\ 8 & 6 & 10 & 9 \\ 7 & 5 & 9 & 10 \end{pmatrix}$$

*Université Aix Marseille

$$b = \begin{pmatrix} 32 \\ 23 \\ 33 \\ 31 \end{pmatrix}$$

On remarque tout d'abord que A est symétrique définie positive, car toute ses valeurs propres sont strictement positives.

On a $cond(A) = 3009.578708058694$.

On résout ensuite le système $Ax = b$ en utilisant numpy. Ici $x = (1, 1, 1, 1)^t$.

On remarque au passage que la solution est la même en utilisant l'implémentation du Pivot de Gauss vu au TP précédent.

Ci-dessous le code python ainsi que les résultats obtenus.

```python
import numpy as np
import pivot as pv


#_____________________EXO_1___________________________#

A=np.array([[10,7,8,7],[7,5,6,5],[8,6,10,9],[7,5,9,10]], dtype=float)
print('A :',A)
print('')
t=np.linalg.eigvals(A)
print('Valeurs propres de A :',t)

b=np.array([[32],[23],[33],[31]], dtype=float)

def sdp(A):
    print('')
    t=np.linalg.eigvals(A)
    for i in t:
        if i <= 0:
            return(print('La matrice nest pas symétrique définie positive'))
    return(print('La matrice est symetrique défnie positive'))

sdp(A)

def sol(A,b):
    x=np.linalg.solve(A,b)
    return x


""""
################TEST#####################
"""
print('')
x=sol(A,b)
print('La solution x du système Ax=b est x= : \n',x,)
print('')


X1=pv.gauss_partiel(A, b)
print('X1 :', X1)

def dx(x,y):
    t=[]
    for i in range(len(x)):
        t.append(x[i] - y[i])
    return t

```

```
A : [[10.  7.  8.  7.]
 [ 7.  5.  6.  5.]
 [ 8.  6. 10.  9.]
 [ 7.  5.  9. 10.]]

Valeurs propres de A : [3.02886853e+01 3.85805746e+00 1.01500484e-02 8.43107150e-01]

La matrice est symetrique défnie positive

La solution x du système Ax=b est x= :
 [[1.]
 [1.]
 [1.]
 [1.]]

X1 : [[1.]
 [1.]
 [1.]
 [1.]]
```

2.0.1 Perturbation du système

On perturbe le système en prenant $b = b + \delta_b$, où $\delta_b = (0.1, -0.1, 0.1, -0.1)^t$.

On a alors une nouvelle solution $x + \delta_x = (1, 1, 1, 1)^t$.

On vérifie ensuite l'inégalité suivante :

$$\frac{\|\delta_x\|}{\|x\|} \leq cond(A) \frac{\|\delta_b\|}{\|b\|}$$

L'inégalité est bien vérifiée (cf. résultats python ci-dessous).

```python
"""
##################Perturbation##################
"""

print('')
print('PERTURBATION')
print('')

db=[0.1,-0.1,0.1,-0.1]

def vect_perturb(y,y1):
    x=[]
    for i in range(len(y)):
        x.append(y[i]+y1[i])
    return x

b1=vect_perturb(b, db)

print('b perturbé :', b1)

x1=sol(A,b1)
print('x perturbé :',x)

print('')
dx=dx(x, x1)
print('dx :', dx)

#Verif Conditionnement#
def cond(A):
    x=np.linalg.norm(A)
    y=np.linalg.norm(np.linalg.inv(A))
    return x*y

print('')
cond=cond(A)
print("cond(A) =",cond)

def norm(x):
    return(np.linalg.norm(x))

DX=norm(dx) / norm(x)
print('DX :', DX)

DB=norm(db)/norm(b)
print('DB :', DB)


print('')

def verif_cond(cond,DX,DB):
    print('Inegalité ',DX <= cond*DB )



verif_cond(cond, DX, DB)
```

```
PERTURBATION

b perturbé : [array([32.1]), array([22.9]), array([33.1]), array([30.9])]
x perturbé : [[1.]
 [1.]
 [1.]
 [1.]]

dx : [array([-8.2]), array([13.6]), array([-3.5]), array([2.1])]

cond(A) = 3009.578708058694
DX : 8.19847546803699
DB : 0.003331945311897623

Inegalité  True
```

3 Exercice 2

Soit $A \in \mathbb{M}_n(\mathbb{R})$ et $n = 10$.
Le but de cette partie est de résoudre l'équation $(QDQ^t)x = b$ où D est une matrice diagonale aléatoire dont les coefficients diagonaux sont compris entre 1 et 10 (choisis aléatoirement). La matrice Q de la décomposition QR d'une matrice à coefficients aléatoires inversible.

On pose $A = QDQ^t$, b la dernière colonne de Q et δ_b la première colonne de Q.
On perturbera ensuite b tel que b perturbé vaut b plus la première colonne de Q pour trouver

x + δ_x.

Puis on vérifiera que $\frac{\|\delta_x\|}{\|x\|} = cond(A)\frac{\|\delta_b\|}{\|b\|}$.

Ci-dessous le code python nous permettant de faire ce qui est précisé au-dessus :

```python
import numpy as np

N=10

#Construction de D# Question 1
def Diag(n):
    d=np.random.rand(n)*10
    d=np.sort(d)
    A=np.diag(d)
    return A

D=Diag(N)
print('D :', D)
print("")

#################################################

#Construction matrice carée aléatoire#
def Mat(n):
    A=np.random.rand(n,n)*10
    if(np.linalg.det(A)==0):
        print('Matrice non inversible')
        return Mat(n)
    return A

#Retourne Q de la décomposition QR#
def QR(X):
    Q, R = np.linalg.qr(X)
    return Q

# Y Matrice aléatoire
Y=Mat(N)
print('Y :', Y)
print('')

# On sort Q de la décompo. QR de Y
Q=QR(Y)
print('Q :', Q)

#####################################################

#Fait le produit Q D Q^t#
def Mat_3(Q,D):
    return np.dot( Q , np.dot(D, np.transpose(Q)))

print('')

A2=Mat_3(Q,D)
print('A=QD :', A2)

print('')

######################################################

def col_n(Q):
    b=[]
    for i in range(N):
        b.append(Q[i][N-1])
    return b

b=col_n(Q)

```

```python
def col_n(Q):
    b=[]
    for i in range(N):
        b.append(Q[i][N-1])
    return b

b=col_n(Q)

print('b', b)
print('')

x=np.linalg.solve(np.linalg.inv(A2),b)
#x=np.dot(A2,b)
print('x :', x)
print("")

def col_1(Q):
    db=[]
    for i in range(N):
        db.append(Q[i][0])
    return db

db=col_1(Q)
print('db :', db)
print('')

b1=[]
for i in range(N):
    b1.append(b[i] + db[i])
print('b + db :', b1)

print('')

x2=np.dot(A2,b1)
print('x+dx', x2)
print('')

dx=[]
for i in range(N):
    dx.append(x2[i] - x[i])

print('dx',dx)
print('')

##########################################################################

def cond(A):
    x=np.linalg.norm(A)
    y=np.linalg.norm(np.linalg.inv(A))
    return x*y

def norm(x):
    return(np.linalg.norm(x))

#################Probleme : Egalité FAUSSE !!!!!!!!!!!!!!! ###############


cond_A=cond(A2)

print('cond(A)',cond_A)
print('')

a=norm(dx)/norm(x)
c=norm(db)/norm(b)

c1=norm(dx)/norm(x) * norm(b)/norm(db)

print('a',a)
```

```
b [-0.4088123231950093, -0.3482422524616913, 0.4655563986180672, 0.45654568423113157, 0.1426178745697422,
-0.032816733713888435, 0.02537481715709934, -0.4977485061256522, 0.03411581798647578, 0.12427772941524853]

x : [-3.99653566 -3.40440467  4.55126385  4.46317541  1.39422759 -0.3208153
  0.24806337 -4.86597282  0.3335151   1.21493494]

db : [-0.3075347539354265, -0.501766545817585, -0.12436201579945436, -0.3673186996801838, -0.33041772750692605,
-0.4496758081216726, -0.17230846356339305, -0.020433721631085533, -0.24003888282000188, -0.32272767868630675]

b + db : [-0.7163470771304359, -0.8500087982792763, 0.3411943828186128, 0.08922698455094774,
-0.18779985293718385, -0.48249254183556106, -0.1469336464062937, -0.5181822277567377, -0.2059230648335261,
-0.19844994927105822]

x+dx [-4.42003773e+00 -4.09538078e+00  4.38000655e+00  3.95734565e+00
  9.39213681e-01 -9.40057944e-01  1.07796460e-02 -4.89411183e+00
  2.96070841e-03  7.70510898e-01]

dx [-0.42350206978859095, -0.6909761189105716, -0.17125729830589886, -0.5058297561363587, -0.4550139120972584,
-0.6192426483069033, -0.237283721684708, -0.028139009635525625, -0.3305543923187192, -0.4444240468848485]

cond(A) 21.34545490167735

a 0.14086452065182015

c*cond(A) 21.345454901677353

c1 0.14086452065182015
```

3.1 Edit :

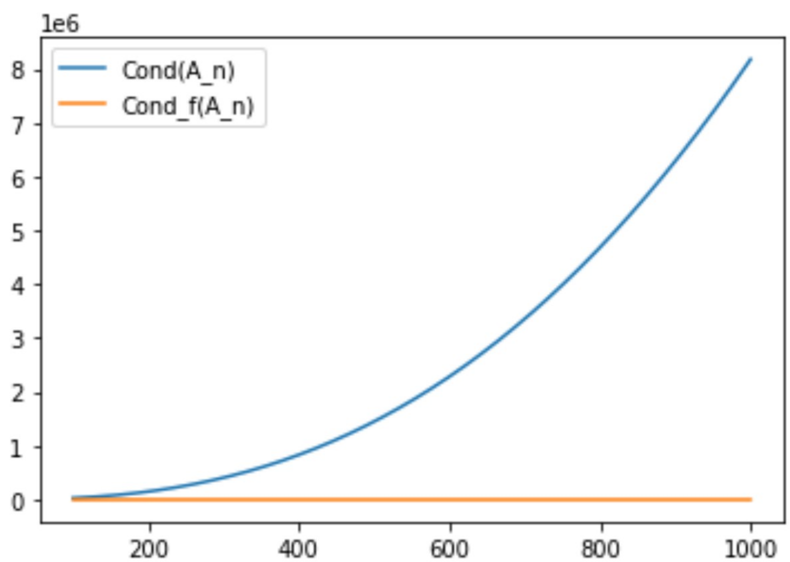
L'égalité n'est pas vérifiée, malgré plusieurs vérifications de notre code.

4 Exercice 3

On reprend la matrice du Laplacien vu au TP 1, avec $f(x) = \pi^2 sin(\pi x)$. On note, pour tout $n \in \mathbb{N}^*$, A_n cette matrice.
On s'intéresse ici au conditionnement de A_n pour chaque n allant de 100 à 1 000.

On calcul alors d'abord le conditionnement des A_n noté $cond(A_n)$ puis celui vérifiant l'égalité $\frac{\|\delta_{x_n}\|}{\|x_n\|} = cond_f(A_n)\frac{\|\delta_{b_n}\|}{\|b_n\|}$.

On a alors le graphe suivant (donné par le code python juste après) :



On remarque alors que $cond(A_n)$ est strictement croissante alors que $cond_f(A_n)$ est très proche de zéro (il est quasi nul).

```python
import numpy as np
import pivot as pv
import matplotlib.pyplot as plt

def norm(x):
    return(np.linalg.norm(x))

def dx(x,y):
    t=[]
    for i in range(len(x)):
        t.append(x[i] - y[i])
    return t

N=3
h=1/(N+1)
def Mat(N):
    h=1/(N+1)
    d=(2/(h*h))*np.ones(N)
    d1= (-1/(h*h))*np.ones(N-1)
    A=np.diag(d,0)+np.diag(d1,1)+np.diag(d1,-1)
    return A

def f(x):
    y=(np.pi*np.pi)*np.sin(np.pi*x)
    return (y)

A=Mat(N)

x=np.linspace(0,1,N+2)

B=f(x)

X=np.linalg.solve(A,B[1:-1])
print('X :', X, '\n')

def cond(A):
    x=np.linalg.norm(A)
    y=np.linalg.norm(np.linalg.inv(A))
    return x*y

cond_n=[]
y=[]
X=[]
cond_F=[]

for n in range(100,1001):
    A=Mat(n)
    x=np.linspace(0,1,n+2)
    B=f(x)
    X=np.linalg.solve(A,B[1:-1])
    cond_A=cond(A)
    #cA.append(cond_A)
    dB=np.random.rand(n+2)*0.1

    B1=B+dB

    X1=np.linalg.solve(A,B1[1:-1])

    dX=dx(X,X1)
    cond_f= (norm(B)/norm(dB))*(norm(dX)/norm(X))
    cond_F.append(cond_f)

    cond_n.append(cond(A))
    y.append(n)

plt.plot(y,cond_n, label="Cond(A_n)")
plt.plot(y,cond_F, label="Cond_f(A_n)")
plt.legend()
plt.show
```