Analyse numérique

1ère année de l'ISIMA, semestre 2

TP Analyse spectrale (mise à niveau)

Jean-Philippe Gayon et Viet Hung Nguyen

2019–2020

[1]:
```
# Nous commençons par importer la librairie numpy et lui donner un alias np
import numpy as np
```

[2]:
```
## Création des arrays dans numpy
#Contrairement aux listes en Python, les tableaux Numpy ne peuvent contenir
#des membres que d'un seul type. On peut aussi spécifier le type manuellement ou laisser Python
# détecter automatiquement.
```

[3]:
```
#On peut créer un tableau à partir d'une liste. Voici trois exemples où le premier
#est un tableau d'entiers, le deuxième est un tableau de flottants double précision (64 bits)
#et le troisième est un tableau de flottant simple précision (32 bits)
A=np.array([4,7,12,15])
print("A=",A)
B=np.array([4.5,6,8,43])
print("B=",B)
C=np.array([2.3,4.2,0.8,12], dtype='float32')
print("C=",C)
```

```
A= [ 4  7 12 15]
B= [ 4.5  6.   8.  43. ]
C= [ 2.3  4.2  0.8 12. ]
```

[4]:
```
#Creation de matrices (tableau de deux dimensions)
A=np.array([[2,3,5],[4,7,9]])
print("A=",A)
```

```
A= [[2 3 5]
 [4 7 9]]
```

[5]:
```
#Accès aux éléments d'une matrice
print("Element en 1ère ligne et en 1ère colonne de A =",A[0,0])
print("Element en 2ème ligne et en 2ème colonne de A =",A[1,1])
print("Element en dernière ligne et en dernière colonne=",A[-1,-1])
```

```
Element en 1ère ligne et en 1ère colonne de A = 2
Element en 2ème ligne et en 2ème colonne de A = 7
Element en dernière ligne et en dernière colonne= 9
```

[6]:
```
#Creation par l'initialisation
#Matrices avec les éléments de valeur nulle
A=np.zeros([4,3])
print("A=",A)
```

```
A= [[0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]]
```

[7]:
```
#Matrices avec les éléments de valeur 1
A=np.ones([2,3])
print("A=",A)
#Matrice d'identité
A=np.eye(3)
print("A=",A)
```

```
#Matrice avec les éléments dont les valeurs sont tirées␣
↪aléatoirement
#entre 1 et 10
A=np.random.randint(1,10,[4,4])
print("A=",A)
```

```
A= [[1. 1. 1.]
 [1. 1. 1.]]
A= [[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]
A= [[3 4 9 1]
 [1 9 7 9]
 [7 4 7 2]
 [9 6 8 1]]
```

[8]:
```
#Extraction (par adresse) de sous-matrice
B=A[0:2,1:3]
print("B=",B)
B[1,1]=20
print("B=",B)
print("A=",A)
#extraction de la 3ème colonne de A
print("La troisième colonne de A=",A[:,2])
#extraction de la 3ème ligne de A
print("La troisième ligne de A=",A[2,:])
```

```
B= [[4 9]
 [9 7]]
B= [[ 4  9]
 [ 9 20]]
A= [[ 3  4  9  1]
 [ 1  9 20  9]
 [ 7  4  7  2]
 [ 9  6  8  1]]
La troisième colonne de A= [ 9 20  7  8]
La troisième ligne de A= [7 4 7 2]
```

[9]:
```
#Extraction (par valeur) de sous-matrice
A[1,2]=6
B=np.copy(A[0:2,1:3])
print("B=",B)
B[1,1]=20
print("B=",B)
print("A=",A)
```

```
B= [[4 9]
 [9 6]]
B= [[ 4  9]
 [ 9 20]]
A= [[3 4 9 1]
 [1 9 6 9]
 [7 4 7 2]
 [9 6 8 1]]
```

[10]:
```
#Pour connaître les dimensions d'une matrice
print("Les dimensions de A = ", np.shape(A))
```

```
Les dimensions de A =  (4, 4)
```

[11]:
```
#On peut redimensionner une matrice
print(np.reshape(A,(2,8)))
```

```
[[3 4 9 1 1 9 6 9]
 [7 4 7 2 9 6 8 1]]
```

[12]:
```
#Création de matrice par itérator
A=np.reshape(np.array([i+j for i in range(2) for j in␣
↪range(2)]),(2,2))
print("A=",A)
```

```
A= [[0 1]
 [1 2]]
```

[13]:
```
#Création de matrice par superposition de vecteurs
u=np.array([1,2,3,4])
v=np.array([5,6,7,8])
M=np.vstack([u,v])
print("M=",M)
N=np.hstack([u,v])
print("N=",N)
```

```
M= [[1 2 3 4]
 [5 6 7 8]]
N= [1 2 3 4 5 6 7 8]
```

[14]:
```
#Opérations vectorielles et matricielle
#Produit scalaire
w=np.dot(u,v)
print("w=",w)
```

```
w= 70
```

[15]:
```
#l'opérateur usuel "*" donne un produit élément par élément
print(u*v)
```

```
[ 5 12 21 32]
```

[16]:
```
#Produit matriciel
A=np.array([[1,0],[1,2]])
print("A=",A)
B=np.array([[2,1],[3,1]])
print("B=",B)
C=np.dot(A,B)
print("C=",C)
#on peut aussi remplacer la fonction np.dot par l'opérateur @
C=A@B
print("C=",C)
```

```
A= [[1 0]
 [1 2]]
B= [[2 1]
 [3 1]]
C= [[2 1]
 [8 3]]
C= [[2 1]
 [8 3]]
```

[17]:
```
#Matrice transposée
print(A.T)
```

```
[[1 1]
 [0 2]]
```

[18]:
```
#Le package linalg de numpy.
#Le package linalg de numpy permet d'exécuter des opérations␣
↪usuelles
#de l'algèbre linéaire sur les matrices (inverser,␣
↪triangulariser, décomposer,
#calculer de diverses propriétés d'une matrices : rang,␣
↪dérterminant, valeurs et vecteurs propres, etc.)
import numpy.linalg as lg
print("Le rang de A=",lg.matrix_rank(A))
print("Le déterminant de A=",lg.det(A))
C=lg.inv(A)
print("L'inverse de A=",C)
```

```
Le rang de A= 2
Le déterminant de A= 2.0
L'inverse de A= [[ 1.   0. ]
 [-0.5  0.5]]
```

[19]:
```
#La fonction eig retourne un tuple qui contient un vecteur␣
↪dont les composantes sont les valeurs propres
```

```
#et une matrices dont les colonnes correspondent aux vecteur␣
 ↪propres.
E=lg.eig(A)
print("Les valeurs de propres et vecteurs propres de A=\n",␣
 ↪E)
#On peut vérifier que la première valeur propre est E[0][0]␣
 ↪et le vecteur propre correspondant
#est E[1][:,0]
print(A@E[1][:,0])
print(E[0][0]*E[1][:,0])
#Création de la matrice diagonale avec les valeurs propres␣
 ↪de A
D=np.diag(E[0])
print("La matrice diagonale D avec les valeurs propres de␣
 ↪A=\n",D)
```

```
Les valeurs de propres et vecteurs propres de A=
 (array([2., 1.]), array([[ 0.        ,  0.70710678],
       [ 1.        , -0.70710678]]))
[0. 2.]
[0. 2.]
La matrice diagonale D avec les valeurs propres de A=
 [[2. 0.]
 [0. 1.]]
```

[20]:
```
#Pour trouver la dimension de l'espace propre lié à la␣
 ↪valeur propre 1.
print(lg.matrix_rank(A-np.eye(2)))
```

```
1
```

[21]:
```
#Calculer une norme vectorielle ou matricielle
#Pour un vecteur, la norme euclidienne est calculée par␣
 ↪défaut
print("Norme euclidienne du vecteur \"ones\" =", lg.norm(np.
 ↪ones(2)))
#On peut calculer une autre norme en la spécifiant
print("Norme infinie du vecteur \"ones\" =", lg.norm(np.
 ↪ones(2),np.inf))
#Pour une matrice, la norme frobenus est calculée par défaut
print("Norme frobenus de la matrice A=",lg.norm(A))
# Pour calculer la norme 2 de A
print("Norme 2 de la matrice A=",lg.norm(A,2))
#Calculer le conditionnement d'une matrice
print("Le conditionnement de la matrice A=",lg.cond(A))
#On peut vérifier que le conditionnement de A est le produit
#de la norme 2 de A avec celle de l'inverse de A
print("Le conditionnement de la matrice A par la norme␣
 ↪2=",lg.norm(A,2)*lg.norm(lg.inv(A),2))
```

```
Norme euclidienne du vecteur "ones" = 1.4142135623730951
Norme infinie du vecteur "ones" = 1.0
Norme frobenus de la matrice A= 2.449489742783178
Norme 2 de la matrice A= 2.2882456112707374
Le conditionnement de la matrice A= 2.6180339887498953
Le conditionnement de la matrice A par la norme 2= 2.
 ↪6180339887498953
```

[22]:
```
#Solution d'un système linéaire Ax=b
b=np.ones(2)
x=lg.solve(A,b)
print(x)
```

```
[1. 0.]
```

[23]:
```
#Décomposition QR de A
[Q,R]=lg.qr(A)
print("Q=",Q)
print("R=",R)
#Vérification QR==A
```

```
#Comparaison directe QR==A n'est pas souhaitée à cause de la␣
 ↪précision numérique
print("Est ce que QR=A ?", Q@R==A)
#On préfère la comparaison suivante
print("Est ce que QR=A ?",np.max(np.abs(Q@R-A))<1E-7)
#A1 est A modifiée afin d'obtenir une matrice symétrique␣
 ↪définie positive
A1=np.copy(A)
A1[0,1]=1
#Décomposition de Cholesky de A1
L=lg.cholesky(A1)
print("L=",L)
print("Est ce que LxL^T=A1 ?", np.max(np.abs(L@L.T-A1))<1E-7)
```

```
Q= [[-0.70710678 -0.70710678]
 [-0.70710678  0.70710678]]
R= [[-1.41421356 -1.41421356]
 [ 0.          1.41421356]]
Est ce que QR=A ? [[False False]
 [ True  True]]
Est ce que QR=A ? True
L= [[1. 0.]
 [1. 1.]]
Est ce que LxL^T=A1 ? True
```

[24]:
```
#Pour la décomposition PLU, on doit importer le package␣
 ↪linalg de scipy et non celui de numpy
import scipy.linalg as slg
[P,L,U]=slg.lu(A)
print("P=",P)
print("L=",L)
print("U=",U)
#On peut vérifier que PA=LU
print("PA=",P@A)
print("LU=",L@U)
```

```
P= [[1. 0.]
 [0. 1.]]
L= [[1. 0.]
 [1. 1.]]
U= [[1. 0.]
 [0. 2.]]
PA= [[1. 0.]
 [1. 2.]]
LU= [[1. 0.]
 [1. 2.]]
```

[25]:
```
#Mesurer le temps d'exécution
from time import time
N=10000000
a=np.random.randint(1,100,N)
temps_avant=time()
a1=np.sort(a,-1,'quicksort')
temps_apres=time()
print("Temps de quicksort pour trier ",N,"␣
 ↪entiers=",temps_apres-temps_avant, "secondes")
temps_avant=time()
a1=np.sort(a,-1,'heapsort')
temps_apres=time()
print("Temps de heapsort pour trier ",N,"␣
 ↪entiers=",temps_apres-temps_avant, "secondes")
```

```
Temps de quicksort pour trier  10000000  entiers= 0.
 ↪6188819408416748 secondes
Temps de heapsort pour trier  10000000  entiers= 1.
 ↪8348851203918457 secondes
```

[ ]:

1 Diagonalisation avec numpy

Pour l'exercice 1 de la fiche de TD "Mise à niveau", retrouver avec Python les valeurs propres, vecteurs propres et diagonalisation des matrices proposées.

2 Suite de Fibonnaci

On reprend l'exercice 2 de la fiche de TD.

Q1. Écrivez un programme qui calcule le n-ième terme de la suite de Fibonacci par un algorithme récursif puis par un algorithme itératif. Calculez le 40-ème terme par les deux méthodes en mesurant le temps de calcul. Commentez.

Q2. Vérifier par ailleurs numériquement que le rapport de deux termes consécutifs converge rapidement vers le nombre d'or. Donner une explication mathématique à la vitesse de convergence.

3 Conditionnement et instabilité numérique d'un système linéaire

On utilisera la norme euclidienne dans cet exercice. Soit

$$M = \begin{pmatrix} 0.4 & 0.3 \\ 0.9 & 0.7 \end{pmatrix}, d = \begin{pmatrix} 0.3 \\ 0.7 \end{pmatrix}$$

Q1. Trouver la solution x de $Mx = d$ par linalg.

Q2. Supposons que par erreur, au lieu de recevoir le vecteur d, on reçoit

$$d_1 = \begin{pmatrix} 0.31 \\ 0.7 \end{pmatrix}$$

Trouver la solution x_1 de $Mx_1 = d_1$ par linalg.

Q3. La matrice M est-elle mal conditionnée ?

Q4. Calculer les erreurs relatives $\delta d = \dfrac{\|d_1 - d\|}{\|d\|}$ et $\delta x = \dfrac{\|x_1 - x\|}{\|x\|}$ ainsi que le facteur d'amplification de l'erreur $\dfrac{\delta x}{\delta d}$.

Q5. Comparer à la borne supérieure du facteur d'amplification donnée par le conditionnement de la matrice.

4 Matrices de Hilbert

Les matrices de Hilbert $\mathcal{H}_n$ sont les matrices définies comme suit : $\mathcal{H}_n(i,j) = \dfrac{1}{i+j-1}$ pour $i, j = 1, \ldots, n$.

Q1. Créer la matrice $\mathcal{H}_n$ avec des itérators (item [12] de l'introduction).

Q2. Programmer avec un nombre linéaire (par rapport à n) d'opérations de multiplications la formule exacte pour le déterminant Δ_n de $\mathcal{H}_n$ donnée par

$$\Delta_n = \prod_{k=0}^{n-1} \frac{(k!)^3}{(n+k)!}.$$

Remarque : la fonction math.factorial(k) nécessite $(k-1)$ multiplications.

Q3. Calculer en utilisant le module linalg de numpy le déterminant de la matrice de Hilbert d'ordre 11. Comparer avec le résultat donnée par la formule exacte.

Q4. Soit A la matrice de Hilbert d'ordre 5. Calculer et afficher A^{-1}.

Q5. Changer la valeur de A[0,0] de 1 à 999/1000 et recalculer A^{-1}. Y-a-t-il de l'instabilité numérique ? Justifier votre réponse. Quelle est la cause éventuelle de cette instabilité ?