Université de Poitiers 2023-2024 M1 MFA et STDV

Analyse Numérique

Kodzo LIMA
César DJIBA

1 Introduction

Dans le cadre de la formation de Matser 1 Statistique Données Du Vivant pour apprendre l'implémentation des algorithmes pour résoudre des sytèmes d'équations avec en python , un devoir de maison nous a été donné qui consiste à écrire des fonctions qui permettent à résoudre des systèmes d'équation avec la décomposition LU et QR.

Exercice 1 Décomposition LU

1. Ecrire une fonction qui effectue la décomposition LU d'une matrice inversible.

```
def LU(A):
    m, n = A.shape
    U = np.zeros((m, n))
    L = np.eye(m)

    for i in range(m):
        for j in range(i, n):
            som = 0
            for k in range(i):
                som += L[i, k] * U[k, j]
            U[i, j] = A[i, j] - som

        for j in range(i+1, m):
            som = 0
            for k in range(i):
                som += L[j, k] * U[k, i]
            L[j, i] = (A[j, i] - som) / U[i, i]

    return L, U
```

2. Pour écrire une fonction qui effectue la résolution d'un système linéaire à l'aide de la décomposition LU ,une fonction de descente nomée "descente" pour résoudre le sythème d'équation $LY = B$ ensuite la fonction de remonté nommée "remontée" pour résoudre le sythème d'équation $UX = Y$.Et cette dernière sera donc la solution de $AX = B$ avec $A = LU$,L triangulaire inférieure et U triangulaire supérieure

```
   #Algorithme de remontée
def descente (L,B):

        m = len(L)

        X = np.copy(B)

        for i in range(m): #On parcourt les lignes de L
            s=0
            for j in range(i):

                s=s+ L[i, j]*X[j]

            X[i]= (X[i]-s)/L[i,i]

        return X
#Solution de l'équation LY=B
def solution_equation1(L,B):
    Y=descente(L,B)
    return Y

#Algorithme de Remontée
def remontee(R, B):
    m = len(R)
    X = np.copy(B)

    for i in range(m - 1, -1, -1):  # On parcourt les lignes de U en sens
       inverse
        s = 0
        for j in range(i + 1, m):  # On utilise i + 1 pour exclure les élé
           ments au-dessus de la diagonale
            s += R[i, j] * X[j]

        X[i] = (X[i] - s) / R[i, i]

    return X

#Solution de l'équation UX=Y
def solution_finale(U,Y):
    X=remontee(U, Y)
    return X
```

* Comparaison des résultats avec ceux obtenus par np.linalg.solve

La solution avec ma décomposition Lu est : [3 0 1]

La solution avec np.linalg.solve est : [3 0 1]

3. Présentation de plusieurs cas tests.

```
#3 CAS DE TEST

#Exemple 1
A = np.array([[2, 0, 0], [0, 3, 0], [0, 0, 3]],dtype = float)
B=np.array([[3], [0], [1]],dtype = float)
L,U=LU(A)
X=solution_finale(U,solution_equation1(L,B))
print("La solution avec ma décomposition Lu est:",X)
print("La solution avec np.linalg.solve est:",X)
#Exemple 2
K = np.array([[-1, 4, 0], [-1, 3, 1], [0, 3, -2]],dtype = float)
J=np.array([[3], [0], [1]],dtype = float)
L,U=LU(K)
X=solution_finale(U,solution_equation1(L,B))
print("La solution avec ma décomposition Lu est:",X)
print("La solution avec np.linalg.solve est:",X)
#Exemple 3
M = np.array([[2, 1, 3], [0, -1, 4], [-1, 2, 0]],dtype = float)
N=np.array([[3], [0], [1]],dtype = float)
L,U=LU(M)
X=solution_finale(U,solution_equation1(L,B))
print("La solution avec ma décomposition Lu est:",X)
print("La solution avec np.linalg.solve est:",X)
```

On a les résultats suivants :

* Exemple 1 :

La solution de exemple 1 avec ma décomposition LU est : [[1.5]
[0.]
[0.33333333]]
La solution avec np.linalg.solve est : [[1.5]
[0.]
[0.33333333]]

* Exemple 2 :

La solution de exemple 2 avec ma décomposition LU est : [[-0.5]
[1.33333333]
[0.33333333]]

La solution avec np.linalg.solve est : [[-0.5]
[1.33333333]
[0.33333333]]

* Exemple 3 :

La solution de exemple 3 avec ma décomposition LU est : [[-1.]
[-0.33333333]
[0.33333333]]
La solution avec np.linalg.solve est : [[-1.]
[-0.33333333]
[0.33333333]]

Tous les résulats conccordent bien comme souhaité.

Exercice 2 Décomposition QR

1. Ecrire une fonction qui effectue la décomposition QR d'une matrice inversible.

```
def QR(A):

    A = np.array(A, dtype=float, copy=True)
    m, n = A.shape
    Q = np.eye(m)

    for j in range(min(m-1, n)):
        # Calculer la réflexion de Householder pour la colonne j de A
        v = A[j:, j]
        norm_v = np.linalg.norm(v)
        beta = -np.sign(v[0]) * norm_v

        u = np.zeros_like(v)
        u[0] = v[0] - beta
        u[1:] = v[1:]

        u = u / np.linalg.norm(u)

        # Mettre    jour A et Q avec la réflexion de Householder
        A[j:, j:] -= 2.0 * np.outer(u, np.dot(u, A[j:, j:]))
        Q[:, j:] -= 2.0 * np.outer(Q[:, j:].dot(u), u)
    R= A[:m, :n]
    return Q, R
```

* Proposition d'un cas test.

```
A = np.array([[1, -1, 4], [1, 4, -2], [2, -1, 1]])
Q, R = QR(A)
print("\nMatrice A:")
print(A)
print("\nMatrice Q:")
print(Q)
print("\nMatrice R:")
print(R)
```

Nous obtenons les résultat suivant :

Matrice A :
[[1 -1 4]
[1 4 -2]
[2 -1 1]]

Matrice Q :
[[-0,40824829 0,27626796 -0,87006284]
[-0,40824829 -0,90773759 -0,09667365]
[-0,81649658 0,31573482 0,48336824]]

Matrice R :
[[-2.44948974e+00 -4.08248290e-01 -1.63299316e+00]
[0,00000000e+00 -4,22295315e+00 3,23628186e+00]
[0,00000000e+00 1,11022302e-16 -2,80353582e+00]]

* Vérification de l décomposition avec la fonction np.linalg.qr()

```
#Vérification de l  décomposition avec la fonction np.linalg.qr()

Q,R= np.linalg.qr(A)
print("La solution avec np.linalg.qr(A) est :")
print("\nMatrice Q:")
print(Q)
print("\nMatrice R:")
print(R)
```

La solution avec np.linalg.qr(A) est :
Matrice Q :
[[-0,40824829 0,27626796 -0,87006284]
[-0,40824829 -0,90773759 -0,09667365]
[-0,81649658 0,31573482 0,48336824]]

Matrice R :
[[-2.44948974 -0.40824829 -1.63299316]
[0.-4.22295315 3.23628186]
[0. 0. -2.80353582]]

Les résultats concordent bien comme souhaité.

* On veut vérifier mon QR avec np.dot(Q,R) qui doit donné A

```
X=np.dot(Q,R)
print("le produit entre Q et R est la matrice :", X)
```

* Le produit entre Q et R est la matrice : [[1. -1. 4.]
[1. 4. -2.]
[2.-1. 1.]]

Ce qui est bien la matrice A avec R triangulaire supérieure.

2. Ecrire une fonction qui qui effectue la résolution d'un système linéaire à l'aide de la décomposition QR

```
def solution_avec_householder(A,b):
    Q, R=QR(A)
    X=remontee(R, np.dot(Q.T,b))

    return X
```

Pour la résolution du système d'équation nous avons utilisé dans la fonction précédente la fonction "remontee" de remontée puisque R est triangulaire supérieure.

* Un cas Test

```
#Cas TEST
A = np.array([[2, -1, 1], [3, 3, 9], [3, 3, 5]])
b = np.array([8, 18, 14])

X=solution_avec_householder(A,b)

print("La solution de  AX = b  d'équation par la décomposition QR est:",X)

#Vérification avec np.linalg.solve

X=np.linalg.solve(A,b)
print("La solution pour vérifier est:", X)
```

On a testé notre fonction de résolution et on veut comparer le résultat avec "np.linalg.solve" qui permet de résoudre un sytème d'équation

La solution de AX = b d'équation par ma décomposition QR est : [3.33333333 -0.33333333 1.]

La solution pour vérifier avec np.linalg.solve est : [3.33333333 -0.33333333 1.]

Les résultats concordent bien comme souhaité.

3. On se propose de tester notre décomposition QR sur des matrices programmées

```
def Defaut_orthogonalite(N, a):
    A = np.ones((N, N)) + a * np.random.rand(N, N)

    # Décomposition QR
    Q, R = QR(A)

    # Calculer le défaut d'orthogonalité
    I = np.eye(N)
    defaut= np.linalg.norm(I - np.dot(Q.T, Q))

    return defaut

# Exemple d'utilisation avec N=3 et a=0.1
N = 3
a = 0.1
defaut = Defaut_orthogonalite(N, a)
print(f"Défaut d'orthogonalité pour N={N} et a={a}: {defaut}")

# Exemple d'utilisation avec N=3 et a=100
N = 3
a = 100
defaut = Defaut_orthogonalite(N, a)
print(f"Défaut d'orthogonalité pour N={N} et a={a}: {defaut}")
```

* Commentaire des résultats.
Le défaut d'orthogonalité est utilisé comme mesure de la qualité de la décomposition QR d'une matrice, notamment dans le contexte de résolution numérique de systèmes linéaires.

une petite valeur (valeur proche de 0) du défaut d'orthogonalité indique que la décomposition QR a bien fonctionné et que la matrice Q est proche d'une matrice orthogonale.

Ici nous avons des résulats suivants :

Défaut d'orthogonalité pour N=3 et a=0.1 : 1.9845449856571236e-14

Défaut d'orthogonalité pour N=3 et a=100 : 4.079927388680004e-16

Nous voyons que lorque le scalaire a varie le defaut d'orthogonalité est toujours proche de 0 . On conclut que ma décomposition QR a bien fonctionné et que Q est proche d'une matrice orthogonale plus loin la solution de mon sytème d'équations vont se rapporcher de la solution exacte.