Algèbre linéaire et bilinéaire I
TDTP$_{12}$ – Pivot de Gauss

6 Décembre 2022

On souhaite faire une fonction Python qui calcule le déterminant d'une matrice carrée d'ordre n. Pour simplifier, **toutes les matrices sont considérées inversibles**.
Comme exemples, on prendra

$$A_1 = \begin{bmatrix} 3 & 7 & -7 & 4 \\ 1 & -2 & -7 & 2 \\ -3 & 5 & -3 & -3 \\ 6 & 0 & -4 & 4 \end{bmatrix} \in \mathrm{M}_4(\mathbb{R}) \quad \text{et} \quad A_2 = \begin{bmatrix} 3 & 7 & -7 & 4 \\ 0 & 0 & 0 & 1 \\ -3 & 5 & -3 & -3 \\ 6 & 0 & -4 & 4 \end{bmatrix} \in \mathrm{M}_4(\mathbb{R}).$$

Partie 1 : manipulation de matrices

1. Importer la bibliothèque `sympy`. Recopier votre code ci-dessous :

```
from sympy import *
```

2. Pour les matrices, on utilise le type `Matrix` de la bibliothèque `sympy`. Recopier votre code ci-dessous et exécuter le :

```
A = Matrix([[1,2],[3,4],[5,6]])
display(A)
display(A.shape)
```

Que fait la fonction `display` ? Elle affiche l'objet.

Que fait la méthode `.shape` ? Elle envoie la taille de la matrice.

3. Recopier les lignes de code suivantes et les exécuter :

```
B = zeros(2,2)
display(B)
B1 = B
B2 = B.copy()
B1[0,0] = 1
B2[0,0] = 2
display(B1,B2)
display(B)
```

Si on veut faire des opérations à partir de la matrice B sans la changer, faut-il les faire sur la matrice `B1 = B` ou sur la matrice `B2 = B.copy()` ? Il faut le faire sur la matrice `B.copy()` pour éviter de modifier `B`.

4. Compléter le tableau suivant (les lignes de la matrice A sont appelées L_i) :

Opération élémentaire sur la matrice A	Code correspondant
Dilatation : $L_i \longleftarrow aL_i$ avec $a \in \mathbb{R}$	`A[i,:] = a*A[i,:]`
Transvection : $L_i \longleftarrow L_i + bL_j$ avec $b \in \mathbb{R}$	`A[i,:] = A[i,:] + b*A[j,:]`
Échange : $L_i \longleftrightarrow L_j$ $C_i \longleftrightarrow C_j$	`A[i,:],A[j,:] = A[j,:],A[i,:]` `A[:,i],A[:,j] = A[:,j],A[:,i]`

Partie 2 : calcul du déterminant d'une matrice triangulaire

5. Soit $M = [m_{i,j}]_{(i,j)\in\{1,\dots,n\}^2} \in \mathrm{M}_n(\mathbb{K})$ une matrice triangulaire. On a

$$\det M = \prod_{k=1}^{n} m_{k,k}$$

6. Compléter puis implémenter la fonction suivante pour qu'elle calcule le déterminant d'une matrice triangulaire :

```python
def determinantTriangulaire(M):
    d = 1
    for k in range(M.shape[0])
        d = d*M[k,k]
    return d
```

Tester cette fonction avec $M = \begin{bmatrix} 2 & 3 & 7 \\ 0 & -4 & 5 \\ 0 & 0 & 9 \end{bmatrix}$ et vérifier avec le résultat de la fonction `det`.

Partie 3 : pivot de Gauss

Le pivot de Gauss consiste à se ramener à une matrice triangulaire à l'aide d'opérations élémentaires sur les lignes.

7. Appliquer ci-dessous et à la main, la première étape de l'algorithme du pivot de Gauss sur la matrice A_2 :

$$A_2 = \begin{bmatrix} 3 & 7 & -7 & 4 \\ 0 & 0 & 0 & 1 \\ -3 & 5 & -3 & -3 \\ 6 & 0 & -4 & 4 \end{bmatrix} \longrightarrow \begin{bmatrix} 3 & 7 & -7 & 4 \\ 0 & 0 & 0 & 1 \\ 0 & 12 & -10 & 1 \\ 0 & -14 & 10 & -4 \end{bmatrix} \begin{matrix} \\ \\ (L_3 \leftarrow L_3 + L_1) \\ (L_4 \leftarrow L_4 - 2L_1) \end{matrix}$$

Quelle opération doit-on faire pour continuer l'algorithme ? Pour éliminer les coefficients en dessous du coefficient en $(1,1)$, il faut échanger L_2 et L_3.

8. Créer une fonction `echange(A,k)` qui a pour arguments une matrice $A = [a_{i,j}]$ et un entier k, qui cherche un coefficient $a_{i,k} \neq 0$ pour $i > k$ et qui échange les lignes L_k et L_i de A.

```python
def echange(B,k):
    for i in range(k+1,B.shape[0]):
        if B[i,k] != 0:
            B[k,:],B[i,:] = B[i,:],B[k,:]
            break
```

Tester votre fonction sur une copie de A_2 avec $k = 1$.

9. Écrire la fonction `pivot(A)` en Python qui correspond à au pseudo-code suivant :

```
pivot(A)
n ← nombre de lignes/colonnes de A
faire une copie B de A
d ← 1
pour k allant de 1 à n
        si b_{k,k} = 0
                echange(B,k)
                d ← -d
        pour i allant de k+1 à n
                L_i ← L_i − (b_{i,k}/b_{k,k}) L_k
retourner B et d
```

```python
def pivot(A):
    n = A.shape[0]
    d = 1
    B = A.copy()
    for k in range(n):
        if B[k,k] == 0:
            echange(B,k)
            d = -d
        for i in range(k+1,n):
                B[i,:] = B[i,:] - (B[i,k]/B[k,k])*B[k,:]
    return B,d
```

10. **Application du pivot de Gauss**

(a) Dans la fonction `pivot`, à quoi correspond `d` ? On a $d = (-1)^s$ où s est le nombre d'échanges.

En déduire l'expression du déterminant d'une matrice A en fonction des résultats `B` et `d` de `pivot(A)` : $\det A = d \det B$

(b) Écrire une fonction `determinant(A)` qui calcule le déterminant d'une matrice A en utilisant les fonctions `pivot` et `determinantTriangulaire`.

```python
def determinant(A):
    B,d = pivot(A)
    return d*determinantTriangulaire(B)
```

(c) Calculer le déterminant des matrices A_1 et A_2 avec cette fonction. Comparer avec le résultat de la fonction `det`.

```
A1 = Matrix([[3,7,-7,4],[1,-2,-7,2],[-3,5,-3,-3],[6,0,-4,4]])
print(determinant(A1),det(A1))
A2 = Matrix([[3,7,-7,4],[0,0,0,1],[-3,5,-3,-3],[6,0,-4,4]])
print(determinant(A2),det(A2))
```

Résultat :
-906 -906
-60 -60

Partie 4 : efficacité du pivot de Gauss

Essayons maintenant de comprendre en quoi le pivot de Gauss est « plus efficace » que d'utiliser la formule

$$\forall A = [a_{i,j}] \in \mathrm{M}_n(\mathbb{K}), \quad \det A = \sum_{s \in \mathfrak{S}_n} \varepsilon(s) \prod_{j=1}^{n} a_{s(j),j}. \tag{1}$$

Pour mesure l'*efficacité* d'un algorithme, on compte le nombre de multiplications effectuées.

11. Rappeler le nombre d'éléments du groupe symétrique $\mathfrak{S}_n$: $n!$ éléments

    Combien de multiplications fait-on lors du calcul de $\det A$ par la formule (1) ? n multiplications dans le produit et cela $n!$ fois donc il y a en tout $n! \times n$ multiplications.

    Pour $n = 10$, cela fait 36288000 multiplications.

Sur Moodle, récupérer le fichier correspondant au TP qui contient une fonction `determinant2(n)` qui choisit au hasard une matrice de $\mathrm{M}_n(\mathbb{R})$, qui calcule son déterminant avec la méthode du pivot de Gauss et renvoie le nombre de multiplications qui ont été effectuées.

12. Éxécuter `determinant2(10)` et comparer ce résultat avec la question précédente.

```
determinant2(10)
```

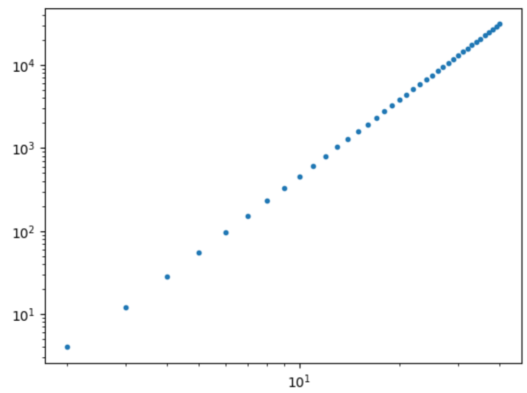
Résultat :
(460, 0.10789098989098876)

460 est bien plus petit que 36288000.

13. Pour n allant de 2 à 40, tracer le résultat de `determinant2(n)` en fonction de n **en échelle logarithmique** (on pourra utiliser la fonction `loglog` de la bibliothèque `matplotlib.pyplot`).

```python
X = range(2,41)
Y = []
for n in X:
    C,d = determinant2(n)
    Y.append(C)
plt.loglog(X,Y,'.')
```

Résultat :

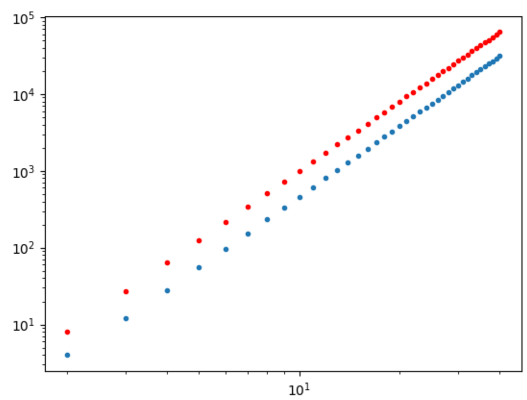


Que remarque-t-on ? On obtient une droite, le nombre de multiplications est donc de la forme $C(n) = An^k$ où k est la pente de la droite.

14. Tracer sur le même graphe la fonction $n \mapsto n^3$.

```python
YY = [x**3 for x in X]
plt.loglog(X,YY,'.r')
```

Résultat :



Que remarque-t-on ? Les deux droites sont parallèles, donc $k = 3$.

Ainsi, le nombre de multiplications nécessaires pour calculer un déterminant avec la méthode du pivot de Gauss est de l'ordre de n^3 ce qui est bien meilleur que $n! \times n$.

Les scientifiques estiment qu'il y a environ 10^{80} atomes (1 suivi de 80 zéros...) dans l'univers observable. Comparer ce nombre avec le nombre de multiplications nécessaire pour calculer le déterminant d'une matrice carrée de taille 60×60, d'abord avec la formule (1) puis avec la fonction `determinant2` Avec `determinant2`, on trouve 106260 multiplications (ce qui n'est rien pour un ordinateur), alors qu'avec (1) on trouve un nombre de multiplications supérieur à 10^{80}...

Partie 5 : résolution de systèmes linéaires (approfondissement)

On souhaite résoudre des systèmes linéaires de la forme

$$AX = Y \tag{2}$$

d'inconnue $X \in \mathrm{M}_{n,1}(\mathbb{K})$, avec $A \in \mathrm{M}_n(\mathbb{K})$ et $Y \in \mathrm{M}_{n,1}(\mathbb{K})$ données.

15. Écrire une fonction Python qui résout (2) dans le cas où A est triangulaire supérieure.

    Tester votre fonction avec la matrice M de la partie 2 en comparant avec la fonction `linsolve`.

```python
def resoudreTriangulaire(A,Y):
    n = A.shape[0]
    X = zeros(n,1)
    X[-1] = Y[-1]/A[-1,-1]
    for i in reversed(range(n-1)):
        s = 0
        for k in range(i+1,n):
            s = s + A[i,k]*X[k]
        X[i] = (Y[i]-s)/A[i,i]
    return X
M = Matrix([[2,3,7],[0,-4,5],[0,0,9]])
Y = Matrix([7,9,-2])
display(resoudreTriangulaire(M,Y))
display(linsolve((M,Y)))
```

Résultat :

$$\begin{bmatrix} \frac{581}{72} \\ -\frac{91}{36} \\ -\frac{2}{9} \end{bmatrix}$$

$$\left\{\left(\frac{581}{72},\ -\frac{91}{36},\ -\frac{2}{9}\right)\right\}$$

16. Écrire une fonction Python qui résout (2) en utilisant la méthode du pivot de Gauss en se ramenant au cas d'une matrice triangulaire.

    Tester votre fonction avec les matrices A_1 et A_2 en comparant avec la fonction `linsolve`.

```python
def pivotSysteme(A,Y):
    n = A.shape[0]
    B = A.copy()
    Z = Y.copy()
    for k in range(n):
        if B[k,k] == 0:
            for i in range(k+1,B.shape[0]+1):
                if B[i,k] != 0:
                    B[k,:],B[i,:] = B[i,:],B[k,:]
                    Z[k],Z[i] = Z[i],Z[k]
                    break
        for i in range(k+1,n):
                a = B[i,k]/B[k,k]
                B[i,:] = B[i,:] - a*B[k,:]
                Z[i] = Z[i] - a*Z[k] # on fait les mêmes opérations sur Z
    return B,Z

def resoudre(A,Y):
    B,Z = pivotSysteme(A,Y)
    X = resoudreTriangulaire(B,Z)
    return X

A1 = Matrix([[3,7,-7,4],[1,2,-7,2],[-3,5,-3,-3],[6,0,-4,4]])
A2 = Matrix([[3,7,-7,4],[0,0,0,1],[-3,5,-3,-3],[6,0,-4,4]])
Y = Matrix([7,9,-2,3])
display(resoudre(A1,Y))
display(linsolve((A1,Y)))
display(resoudre(A2,Y))
display(linsolve((A2,Y)))
```