PROJET 1
MMSN-SUJET 3

Yan Tonglin
Deregnaucourt Lucas

17 October 2019

Table des matières

Chapitre 1

Présentation du problème, des méthodes utilisées et des théorèmes classiques à vérifier

1.1 Objectif

Ce projet consiste à comparer l'implémentation de deux méthodes classiques de factorisation : la factorisation LU et la factorisation avec pivotage. En effet ces méthodes sont censées être équivalentes, mais les problèmes liés à la représentation des flottants et du 0 en machine pourraient peser dans la balance. Pour ce faire, nous nous appuierons sur les deux systèmes linéaires suivants :

$$\underbrace{\begin{pmatrix} 2 & 4 & 6 \\ -2 & 1 & 1 \\ -1 & -1 & 2 \end{pmatrix}}_{A} \underline{x} = \begin{pmatrix} 6 \\ -1 \\ -2 \end{pmatrix} \text{ et } \underbrace{\begin{pmatrix} 1 & 1+0.5\times 10^{-15} & 3 \\ 2 & 2 & 20 \\ 3 & 6 & 4 \end{pmatrix}}_{B} \underline{x} = \begin{pmatrix} 5+0.5\times 10^{-15} \\ 24 \\ 18 \end{pmatrix}$$

Tout d'abord, nous montrerons que les matrices A et B sont inversibles. Cette condition est nécessaire pour utiliser les méthodes de factorisation voulues. Nous vérifierons alors si les vecteurs donnés dans l'énoncé sont bien solutions de ces systèmes afin d'avoir une idée du résultat attendu. Ensuite, nous examinerons les résultats obtenus par l'implémentation des deux méthodes. Enfin, nous remplacerons le 10^{-15} par une valeur paramètre ϵ afin de comparer l'implémentation des deux méthodes en fonction de ϵ.

1.2 Inversibilté et vérification des solutions

Pour savoir si A et B sont inversibles, nous avons utiliser le théorème d'inversiblilité qui nous dit qu'une matrice carrée est inversible si, et seulement si,

son déterminant est non nul :

$$\begin{aligned} det(A) &= \begin{vmatrix} 2 & 4 & 6 \\ -2 & 1 & 1 \\ -1 & -1 & 2 \end{vmatrix} \\ &= 2 \times \begin{vmatrix} 1 & 1 \\ -1 & 2 \end{vmatrix} + 2 \times \begin{vmatrix} 4 & 6 \\ -1 & 2 \end{vmatrix} - 1 \times \begin{vmatrix} 4 & 6 \\ 1 & 1 \end{vmatrix} \\ &= 2 \times 3 + 2 \times 14 - 1 \times (-2) \\ &= 36 \neq 0 \end{aligned}$$

Donc A est inversible. Voyons pour B :

$$\begin{aligned} det(B) &= \begin{vmatrix} 1 & 1 + 0.5 \times 10^{-15} & 3 \\ 2 & 2 & 20 \\ 3 & 6 & 4 \end{vmatrix} \\ &= (1 + 0.5 \times 10^{-15}) \times 52 + 2 \times (-5) - 6 \times 14 \\ &= -42 + 26 \times 10^{-15} \neq 0 \end{aligned}$$

Les matrices A et B sont inversibles, nous allons donc pouvoir utiliser la factorisation LU et la factorisation avec pivotage.

Avant celà, vérifions que $\begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix}$ et $\begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$ sont solutions des systèmes linéaires :

$$A \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 2 \times 1 + 4 \times 1 + 6 \times 0 \\ -2 \times 1 + 1 \times 1 + 0 \times 1 \\ -1 \times 1 - 1 + 1 + 0 \times 2 \end{pmatrix} = \begin{pmatrix} 6 \\ -1 \\ -2 \end{pmatrix}$$

$$B \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \times 1 + (1 + 0.5 \times 10^{-15}) \times 1 + 3 \times 1 \\ 2 \times 1 + 2 \times 1 + 20 \times 1 \\ 3 \times 1 + 6 \times 1 + 4 \times 1 \end{pmatrix} = \begin{pmatrix} 5 + 0.5 \times 10^{-15} \\ 24 \\ 13 \end{pmatrix}$$

Les vecteurs proposés sont bien solutions des systèmes linéaires.

1.3 Méthodes utilisées

Etant donné que A et B sont carrées et inversibles, nous pouvons utiliser la méthode de factorisation de Gauss. De plus, toutes les sous-matrices de A et B sont inversibles, donc, par théorème, les décompositions LU des matrices A et B sont uniques et L est à diagonale unité, c'est-à-dire que $L_{ii} = 1 \; \forall i \in \{1, ..., n\} (*)$.

1.3.1 Factorisation LU

Le principe de cette méthode consiste à trouver L_A, L_B, U_A et U_B tels que

$A = L_A U_A$ et $B = L_B U_B$ avec L_A, L_B triangulaires inférieures et U_A, U_B triangulaires supérieures.

Résoudre, par exemple, $A\underline{x} = \underline{b}$ reviendrait alors à résoudre ces systèmes linéaires plus simples :

$$\begin{cases} L_A \underline{y} = \underline{b} \\ U_A \underline{x} = \underline{y} \end{cases}$$

Examinons la méthode pour le système $A\underline{x} = \underline{b}$:

Nous savons déjà que L_A à diagonale unité, donc :

$$L_A = \begin{pmatrix} 1 & 0 & 0 \\ L_{A21} & 1 & 0 \\ L_{A31} & L_{A32} & 1 \end{pmatrix}$$

Prenons ensuite arbitrairement A_{11} comme pivot. Pour obtenir L_{A21} et L_{A31}, il suffit de diviser A_{21} et A_{31} par le pivot :

$$L_{A21} = \frac{A_{21}}{A_{11}} = -1$$
$$L_{A31} = \frac{A_{31}}{A_{11}} = -\frac{1}{2}$$

On remarque alors aisément que la première colonne de U_A est nécessairement la même que celle de A. Voici où nous en sommes à ce stade :

$$L_A = \begin{pmatrix} 1 & 0 & 0 \\ -1 & 1 & 0 \\ -\frac{1}{2} & L_{A32} & 1 \end{pmatrix}$$
$$U_A = \begin{pmatrix} 2 & 4 & 6 \\ 0 & U_{22} & U_{23} \\ 0 & 0 & U_{33} \end{pmatrix}$$

Or $A_{ik} = \sum_{j=1}^{min(i,k)} L_{Aij} U_{Ajk}$. La suite de la méthode devient de suite purement calculatoire :

$$\begin{cases} A_{22} = L_{A21}U_{A12} + L_{A22}U_{A22} \iff 1 = -1 * 4 + U_{A22} \iff U_{A22} = 5 \\ A_{23} = L_{A21}U_{A13} + L_{A22}U_{A23} \iff 1 = -1 * 6 + U_{A23} \iff U_{A23} = 7 \\ A_{32} = L_{A31}U_{A12} + L_{A32} + U_{A22} \iff -1 = -\frac{1}{2} * 4 + L_{A32} * 5 \iff L_{A32} = \frac{1}{5} \\ A_{33} = L_{A31}U_{A13} + L_{A32}U_{A23} + L_{A33}U_{A33} \iff 2 = -\frac{1}{2} * 6 + \frac{1}{5} * 7 + U_{A33} \iff U_{A33} = \frac{18}{5} \end{cases}$$

Nous avons ainsi déterminé entièrement les matrices L_A et U_A :

$$L_A = \begin{pmatrix} 1 & 0 & 0 \\ -1 & 1 & 0 \\ -\frac{1}{2} & \frac{1}{5} & 1 \end{pmatrix}$$
$$U_A = \begin{pmatrix} 2 & 4 & 6 \\ 0 & 5 & 7 \\ 0 & 0 & \frac{18}{5} \end{pmatrix}$$

La méthode pour trouver L_B et U_B serait strictement la même.

Nous venons de voir la factorisation LU pour des matrices de dimension 3*3, mais le principe reste le même pour des matrices de dimension n*n :

-Initialiser la matrice L : diagonale unité si le théorème $(*)$ est vérifié, puis on détermine la première colonne en fixant le pivot. On a donc $L_{i1} = \frac{A_{i1}}{A_{11}}$.

-Initialiser la matrice U : si L est à diagonale unité, la première colonne de U est celle de A.

-Décomposer chaque élément de A en somme de produits d'éléments de L et de U : $A_{ik} = \sum_{j=1}^{min(i,k)} L_{Aij} U_{Ajk}$. On détermine ainsi un à un les éléments inconnus de L et U.

1.3.2 Factorisation avec pivotage

Le principe consiste à triangulariser le système en utilisant l'élimination de Gauss. Prenons comme exemple le système A :

La première étape consiste à créer une matrice $E^{(2)} = \begin{pmatrix} 1 & 0 & 0 \\ -L_{21}^{(2)} & 1 & 0 \\ -L_{31}^{(2)} & 0 & 1 \end{pmatrix}$ avec $L_{i1}^{(2)} = \frac{A_{i1}}{A_{11}}$, $i \geq 2$. On obtient ainsi $E^{(2)} = \begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ \frac{1}{2} & 0 & 1 \end{pmatrix}$.

On crée ensuite $A^{(2)} = E^{(2)}A = \begin{pmatrix} 2 & 4 & 6 \\ 0 & 5 & 7 \\ 0 & 1 & 5 \end{pmatrix}$ et $\underline{b}^{(2)} = E^{(2)}\underline{b} = \begin{pmatrix} 6 \\ 5 \\ 1 \end{pmatrix}$.

Le système $A\underline{x} = \underline{b}$ est équivalent au système $A^{(2)}\underline{x} = \underline{b}^{(2)}$. La création de ce nouveau système nous a permis d'éliminer l'inconnue x_1 des équations 2 et 3. Il ne nous reste alors plus qu'à répéter l'opération pour éliminer l'inconnue x_2 de la troisième équation afin d'obtenir un système triangulaire facile à résoudre.

Le pivot ne sera cette fois-ci pas A_{11}, mais A_{22}. On crée donc $E^{(3)} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -L_{32}^{(3)} & 1 \end{pmatrix}$ avec $L_{32}^{(3)} = \frac{A_{32}^{(2)}}{A_{22}^{(2)}} = \frac{1}{5}$. Donc $E^{(3)} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -\frac{1}{5} & 1 \end{pmatrix}$.

On crée ensuite $A^{(3)} = E^{(3)}A^{(2)} = \begin{pmatrix} 2 & 4 & 6 \\ 0 & 5 & 7 \\ 0 & 0 & \frac{18}{5} \end{pmatrix}$ et $\underline{b}^{(3)} = E^{(3)}\underline{b}^{(2)} = \begin{pmatrix} 6 \\ 5 \\ 0 \end{pmatrix}$. Nous obtenons le système suivant :

$$A\underline{x} = \underline{b} \Longleftrightarrow A^{(3)}\underline{x} = \underline{b}^{(3)} \Longleftrightarrow E^{(3)}E^{(2)}A\underline{x} = \underline{b}^{(3)}$$

On en déduit une décomposition LU avec $U = A^{(3)}$ et $L = (E^{(3)}E^{(2)})^{-1} = \begin{pmatrix} 1 & 0 & 0 \\ L_{21}^{(2)} & 1 & 0 \\ L_{31}^{(2)} & L_{32}^{(3)} & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ -1 & 1 & 0 \\ -\frac{1}{2} & \frac{1}{5} & 1 \end{pmatrix}$.

Il ne reste plus qu'à résoudre le système triangularisé $U\underline{x} = \underline{b}^{(3)}$.

Pour des matrices de dimension n*n, le principe ne change pas :

-Créer $E^{(2)} = \begin{pmatrix} 1 & 0 & \cdots & 0 \\ -L_{21}^{(2)} & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ -L_{n1}^{(2)} & 0 & \cdots & 1 \end{pmatrix}$ avec $L_{i1}^{(2)} = \frac{A_{i1}}{A11}$ $\forall i \geq 2$.

-Créer $A^{(2)} = E^{(2)}A$ ainsi que $\underline{b}^{(2)} = E^{(2)}\underline{b}$ afin d'obtenir un système équivalent tout en éliminant l'inconnue x_1 des équations 2 à n.

-Recommencer jusqu'à ce que le pivot soit $A_{(n-1)(n-1)}^{(n-1)}$. On obtient alors $U = A^{(n)}$ triangulaire supérieure, $L = (E^{(n)}E^{(n-1)} \cdots E^{(2)})^{-1}$ triangulaire inférieure ainsi que $\underline{b}^{(n)} = E^{(n)}\underline{b}^{(n-1)}$.

-Résoudre le système $U\underline{x} = \underline{b}^{(n)}$.

Nous retrouvons théoriquement les mêmes matrices L et U avec ces deux méthodes. Cependant, il n'y a plus qu'un système à résoudre en utilisant la méthode avec pivotage contre deux avec la factorisation LU.

Chapitre 2

Programmation

Nous avons décidé d'implémenter ces méthodes en utilisant python car c'est un langage assez proche du pseudo code, il est donc à la fois simple à manipuler et simple à comprendre. De plus, le module numpy nous permet d'effectuer facilement des calculs matriciels. Nous présenterons les algorithmes en prenant l'exemple $A\underline{x} = \underline{b}$.

2.1 Factorisation LU

```
# initialisation de la matrice triangulaire supérieure
U=np.zeros((3,3))
# initialisation de la matrice triangulaire inférieure à
↪  diagonale unité
L=np.eye(3)
# boucle d'étape 1 à étape n
for k in range(3):
# nous calculons d'abord la matrice U à chaque étape
        for c in range(k,3):
                U[k,c]=A1[k,c]
                for j in range(k):
                        U[k,c]=U[k,c]-L[k,j]*U[j,c] #7 cycles
# puis nous résolvons la matrice L
        for d in range(k+1,3):
                L[d,k]=A1[d,k]
                for j in range(k):
                        L[d,k]=L[d,k]-L[d,j]*U[j,k] #7 cycles
                L[d,k]=L[d,k]/U[k,k] #10 cycles
#65 cycles
```

2.2 Factorisation avec pivotage

Dans cette section, nous développons deux méthodes pour coder la factorisation avec pivotage :

La première méthode est celle qui se rapproche le plus du calcul théorique, elle est donc plus facile à comprendre. Dans cette fonction, nous avons deux matrices importantes $E1$ et E qui servent à effectuer des boucles. Nous initialisons $E = I_3$ dans la boucle i pour obtenir $E^{(i)}$. Ensuite, nous faisons le produit matriciel $E = E^{(i)}E$ et $A = E^{(i)}A$. Le résultat de cette fonction nous donne

$$E = E^{(3)}E^{(2)}$$

$$A\underline{x} = \underline{b} \iff E^{(3)}E^{(2)}A\underline{x} = \underline{b}^{(3)}$$

```
A0=A1
E1=np.eye(3)
for i in range(2):
        E=np.eye(3)
        for j in range(i+1,3):
                E[j][i]=-A0[j][i]/A0[i][i] #11 cycles
        E1=E@E1 #180 cycles
        A0=E@A0 #180 cycles
#753 cycles
```

Puisque le point clé de la factorisation avec pivotage est de créer une matrice triangulaire inférieure, nous codons de façon plus directe. En effet, au lieu de faire le produit scalaire, nous ne modifions que des éléments.

```
A0=A1
L=np.eye(3)
for i in range(2):
        for j in range(i+1,3):
                L[j,i]=A0[j,i]/A0[i,i] #10 cycles
                for k in range(i,3):
                        A0[j,k]=A0[j,k]-L[j,i]*A0[i,k] #7 cycles
#65 cycles
```

2.3 Résoudre le système

2.3.1 Méthode 1

Après avoir obtenu deux matrice L et U, nous cherchons à déterminer $\underline{x}$ tel que $A\underline{x} = \underline{b}$. Celà revient à résoudre ces systèmes linéaires plus simples :

$$\begin{cases} L_A\underline{y} = \underline{b} \\ U_A\underline{x} = \underline{y} \end{cases}$$

Nous résolvons la première équation par l'algorithme de descente et la deuxième équation par l'algorithme de remontée.

```
for i in range(3):
        y[i]=B1[i]
        for j in range (i):
                y[i]=y[i]-L[i,j]*y[j] #7 cycles
        y[i]=y[i]/L[i,i] #10 cycles
for i in range(2,-1,-1):
        X[i]=y[i]
        for j in range(i+1,3):
                X[i]=X[i]-U[i,j]*X[j] #7 cycles
        X[i]=X[i]/U[i,i] #10 cycles
#102 cycles
```

2.3.2 Méthode 2

Par la factorisation LU, le système $A\underline{x} = \underline{b}$ est transformé en $L_A U_A \underline{x} = \underline{b}$. La deuxième méthode consiste à multiplier L_A^{-1} des deux côtés de l'équation de sorte qu'il ne reste qu'une matrice triangulaire supérieur à gauche.

$$L_A U_A \underline{x} = \underline{b} \iff L_A^{-1} L_A U_A \underline{x} = L_A^{-1} \underline{b} \iff U_A \underline{x} = L_A^{-1} \underline{b}$$

Maintenant, nous avons besoin d'utiliser l'algorithme de remontée.

```
#A0=L(-1)LU
A0=np.linalg.inv(L)@L@U #360 cycles
#B0=L(-1)B1
B0=np.linalg.inv(L)@B1 #180 cycles
for i in range(2,-1,-1):
        X[i]=B0[i]
        for j in range(i+1,3):
                X[i]=X[i]-X[j]*A0[i,j] #7 cycles
        X[i]=X[i]/A0[i,i] #10 cycles
#591 cycles
```

Chapitre 3

Résultats numériques

3.1 Coût de calcul des algorithmes utilisés

Nous avons calculé le coût d'opérations suivant le code. Le principe est que l'addition et la soustraction ne prennent que 1 cycle, la multiplication prend 6 cycles et la division en prend 10. Pour le produit matriciel, il prend 20 cycles pour chaque élément. Puisque les matrices sont de taille $3 * 3$, il faut alors 180 cycles pour un produit scalaire dans notre programme. Si nous faisons la comparaison sur les différentes méthodes que nous avons développées, il est facile de conclure que le produit matriciel prend trop de cycles.

Puisque les méthodes avec produit scalaire prennent trop de cycles, nous ne l'utilisons que dans le premier système pour économiser l'espace. Les deux systèmes suivants sont présentés par la méthode plus efficace.

3.2 Choix des exemples numériques et fiabilité

Pour vérifier que notre programme fonctionne correctemment, nous l'avons testé avec les systèmes demandés. Ces systèmes sont en effet de très bons exemples car, d'une part, les matrices A et B satisfont la condition d'inversibilité nécessaire pour être certains que la factorisation LU est unique. D'autre part, les solutions sont simples : $\begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix}$ pour le premier système et $\begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$ pour le second.

De plus, le premier système ne comporte que des entiers de taille résonnable. Les résultats du code devrait donc être assez proches des résultats théoriques. Cependant, nous avons de suite émis des doutes quant à la fiabilité du code par rapport au deuxième système puisque, bien que la solution soit simple, les matrices B et $\underline{b}_2$ comportent des valeurs qui peuvent être problématiques pour la représentation des flottants en machine : $B_{12} = 1 + 0.5 * 10^{-15}$ et $\underline{b}_{2_1} = 5 + 0.5 * 10^{-15}$. Ces doutes se sont avérés être justifiés lors de l'exécution du

code. En effet, les solutions obtenues avec la factorisation LU et la factorisation avec pivotage étaient différentes.

Nous avons ensuite remplacé $05 * 10^{-15}$ par une valeur paramètre ϵ afin d'étudier son poids dans l'erreur de la solution obtenue et d'en déduire l'ordre de l'intervalle de valeurs que peut prendre ϵ pour que notre code puisse être considéré comme fiable.

Chapitre 4

Conclusion

La solution du code pour le premier système ne diffère pas selon la méthode utilisée et nous semble satisfaisante par rapport aux résultats théoriques :

$$\begin{pmatrix} 1.00000000e+00 \\ 1.00000000e+00 \\ -4.62592927e-17 \end{pmatrix}$$

L'erreur du troisième élément vient certainement du problème de la représentation du 0 en machine.

Quant à la solution obtenue pour le deuxième système, nous sommes bien loins des résultats théoriques :
$\begin{pmatrix} 2. \\ -0. \\ 1. \end{pmatrix}$ avec la factorisation LU et $\begin{pmatrix} 0. \\ 2. \\ 1. \end{pmatrix}$ avec pivotage. Nous pouvons remarquer qu'ici, le résultat diffère selon la méthode utilisée. Nous avons donc émis l'hypothèse que ces erreurs sont dues à la représentation des flottants assez proches de 0.

Pour valider ou non cette hypothèse, nous avons adapté le code avec une valeur paramètre ϵ. Celà nous a permis de constater que les résultats obtenus étaient satisfaisants lorque ϵ était comprise entre $0.5 * 10^{-11}$ et $0.5 * 10^{12}$.

Celà nous permet ainsi de conclure en validant notre hypothèse, ce qui vérifie les problèmes liés à la représentation des flottants en machine étudiés en MAO.