

UNIVERSITÉ SAINT QUENTIN EN YVELINES  
UNIVERSITÉ PARIS SACLAY



RAPPORT TP2 TP3 CN  
M1 CALCUL HAUTE HAUTE PERFORMANCE, SIMULATION

---

## TP Calcul Numérique

---

*Étudiant:*  
Anis MEHIDI

*Responsable:*  
T.DUFAUD

Novembre 2021

# Contents

1	Introduction	2
2	Exercice 6 Tp2 (exo6TP2.sci)	2
3	Exercice 7 Tp2 (exo7TP2.sci)	5
4	Exercice 8 Tp2 (exo8TP2.sci)	7
5	Exercice 2 Tp3 (exo2TP3.sci)	12
6	Exercice 3 Tp3 (exo3TP3.sci)	15
7	Exercice 4 Tp3 (exo4TP3.sci)	17
8	Annexe	19

# 1 Introduction

Ce rapport a pour but de montrer tout les travaux réalisé durant les trois dernières séances de TD/TP. On a pu découvrir le langage Scilab ainsi que son utilisation pour implémenter divers algorithmes déjà vu au cours ainsi que de faire des tests et aussi calculer l'erreur avant, l'erreur arrière, le conditionnement...

## 2 Exercice 6 Tp2 (exo6TP2.sci)

1)- Vecteur X à 1 ligne et 4 colonnes

- $x=[1,2,3,4]$

Result:

$x =$

1. 2. 3. 4.

2)- Vecteur Y à 4 lignes et 1 colonnes

- $y=[1;2;3;4]$

Result:

$y =$

1.

2.

3.

4.

3)- La somme et le produit

- Données:

$x=[1,2,3,4]$ ,  $y=[4,5,6,7]$

- $z=x+y$

Result:

$z =$

5. 7. 9. 11.

- Données:

$x=[1,2,3,4]$ ,  $y=[1;2;3;4]$

- $s=x*y$

Result:

$s =$

30.

4)- La taille des vecteurs

- `size(x)`

Données:

`x=[1,2,3,4]`

Result:

`ans =`

1. 4.

- `size(y)`

Données:

`y=[1;2;3;4]`

Result:

`ans =`

4. 1.

5)- Norme 2 de x

- `norm(x)`

Données:

`x=[1,2,3,4]`

Result:

`ans =`

5.4772256

6)- Matrice A à 4 lignes et 3 colonnes

- Entrée

`A=[1,2,3;4,5,6;7,8,9;10,11,12]`

- Result:

`A =`

1. 2. 3.

4. 5. 6.

7. 8. 9.

10. 11. 12.

7)- Transposée de A

- A'

Entrée

A=[1,2,3;4,5,6;7,8,9;10,11,12]

Result:

ans =

1. 4. 7. 10.

2. 5. 8. 11.

3. 6. 9. 12.

8)- Les opérations de bases avec deux matrices carrées A et B

- Entrée A=[1,2,3;4,5,6;7,8,9]

B=[10,11,12;13,14,15;16,17,18]

- A+B

Result:

ans =

11. 13. 15.

17. 19. 21.

23. 25. 27.

- A-B

Result:

-9. -9. -9.

-9. -9. -9.

-9. -9. -9.

- A\*B

Result:

ans =

84. 90. 96.

201. 216. 231.

318. 342. 366.

9)- Conditionnement de la Matrice A

- cond(A)

Entrée: A=[1,2,3;4,5,6;7,8,9]

Result:

ans =

3.813D+16

### 3 Exercice 7 Tp2 (exo7TP2.sci)

- Pour le cas  $n=3$

1)- Matrice A de taille  $3 \times 3$  en utilisant la fonction rand()

- **A=rand(3,3)**

Result:

A =

```
0.2113249 0.3303271 0.8497452
0.7560439 0.6653811 0.685731
0.0002211 0.6283918 0.8782165
```

2)- Vecteur xex avec la fonction rand()

- **xex=rand(1:3)**

Result:

xex =

```
0.068374 0.5608486 0.6623569
```

- Vérification que x est un vecteur colonne

**xex=xex'**

Result:

xex =

```
0.068374
0.5608486
0.6623569
```

3)- Écriture  $b = A * xex$

- **b=A\*xex**

Result:

b =

```
0.7625473
0.8790705
0.9341406
```

4)- Résolution du système  $Ax=b$  avec la fonction "\"

On a :  $Ax=b$  du coup  $x=A \backslash b$  ou  $x=inv(A)*b$

- Entrée:

**x=A\b**

- Result:

x =

```
0.068374
0.5608486
0.6623569
```

5)- Calcul des erreurs

- Calcul de l'erreur avant

**err=norm(xex-x)/norm(xex)**

Result:

err =

5.640D-16

- Calcul de l'erreur arrière

**r = b - A \* x**

Result:

r =

-1.110D-16

0.

-2.220D-16

**relres=norm(r)/(norm(A)\*norm(x))**

Result:

relres =

1.604D-16

- L'erreur avant est inférieure à l'erreur arrière

- Pour le cas n=100

- err =

1.218D-12

- relres =

3.573D-16

- L'erreur avant est toujours inférieure à l'erreur arrière mais lorsque on augmente le n, l'erreur arrière deviens plus petite et l'erreur avant deviens plus grande

- Pour le cas n=1000

- err =

9.564D-13

- relres =

1.739D-15

- L'erreur avant est toujours inférieure à l'erreur arrière mais lorsque on augmente le n, l'erreur arrière deviens plus petite et l'erreur avant deviens plus grande

- Pour le cas n=10000

- Mon pc ne supporte pas cet entier, saturation

Il prend tellement du temps pour executer pour cet entier

## 4 Exercice 8 Tp2 (exo8TP2.sci)

1)- Fonction matmat3b :

```
function [C]=matmat3b(A,B)
    m=size(A,1)
    n=size(B,2)
    p=size(B,1)
    C=zeros(m,n)
    for i = 1 : m
        for j = 1 : n
            for k = 1 : p
                C(i, j) = A(i, k )*B(k , j) + C(i, j);
            end
        end
    end
endfunction
```

2)- Fonction matmat2b et matmat1b

- Fonction matmat2b

```
function [C]=matmat2b(A,B)
    m=size(A,1)
    n=size(B,2)
    C=zeros(m,n)
    for i = 1 : m
        for j = 1 : n
            C(i, j) = A(i, :)*B(:, j) + C(i, j);
        end
    end
endfunction
```



- Fonction matmat1b :

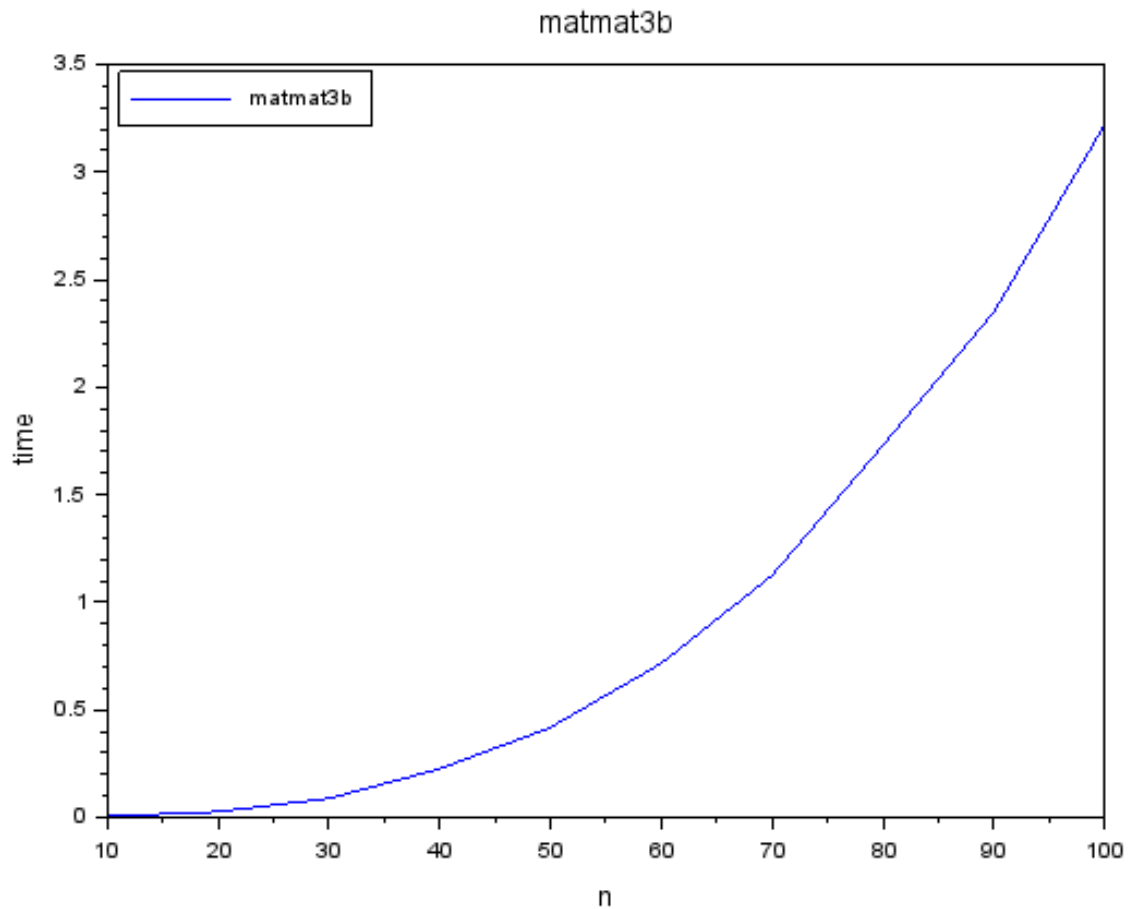
```
function [C]=matmat1b(A,B)
    m=size(A,1)
    n=size(B,2)
    C=zeros(m,n)
    for i = 1 : m
        C(i, :) = A(i, :)*B + C(i, :);
    end
endfunction
```

3)- Mesure avec les fonctions tic et toc()

```
tic;
[C3] = matmat3b(A, B);
t3(i) = toc();
tic;
[C2] = matmat2b(A, B);
t2(i) = toc();
tic;
[C1] = matmat1b(A, B);
t1(i) = toc();
```

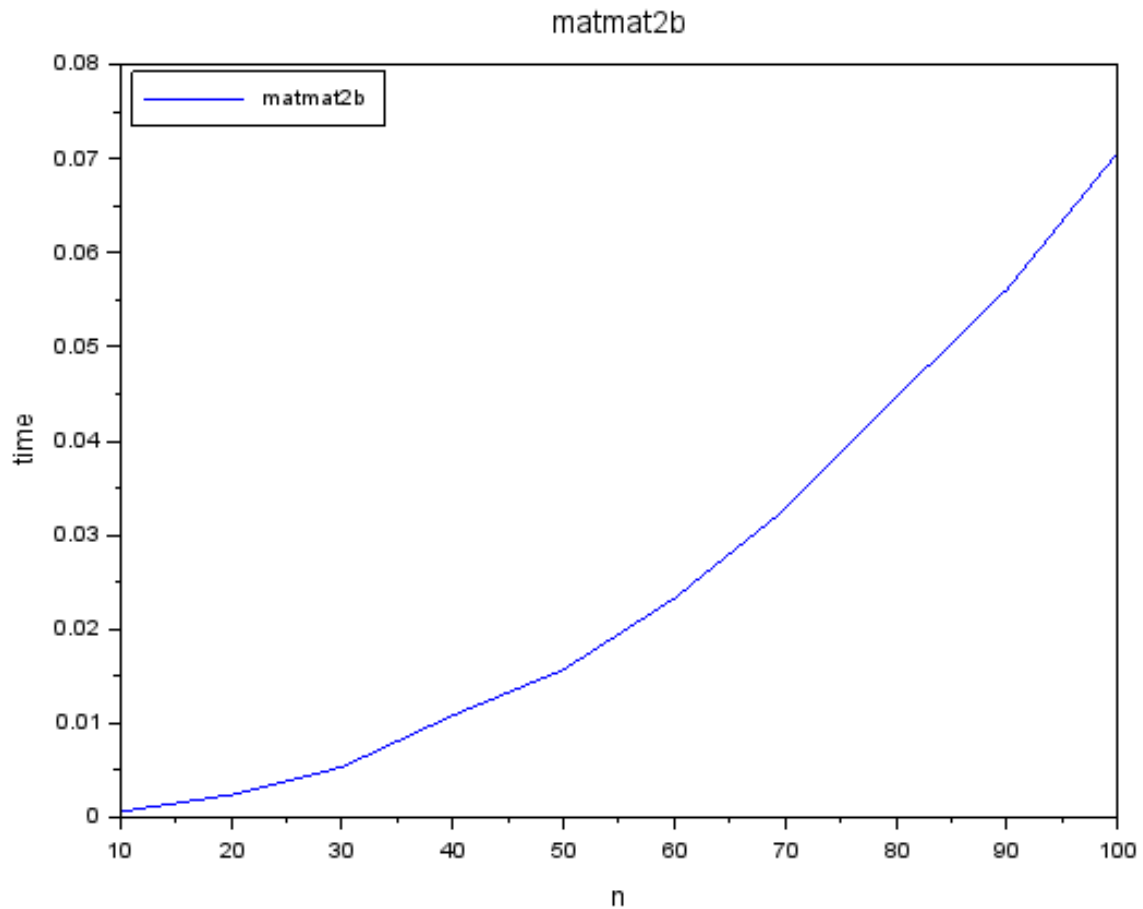
- Graphe pour la fonction `matmat3b`

On remarque qu'à chaque incrémentation de  $n$ , les mesures des fonctions `tic` et `toc` par rapport au temps augmente d'une manière lente car on exécute 3 boucles pour résoudre la fonction `matmat3b`.



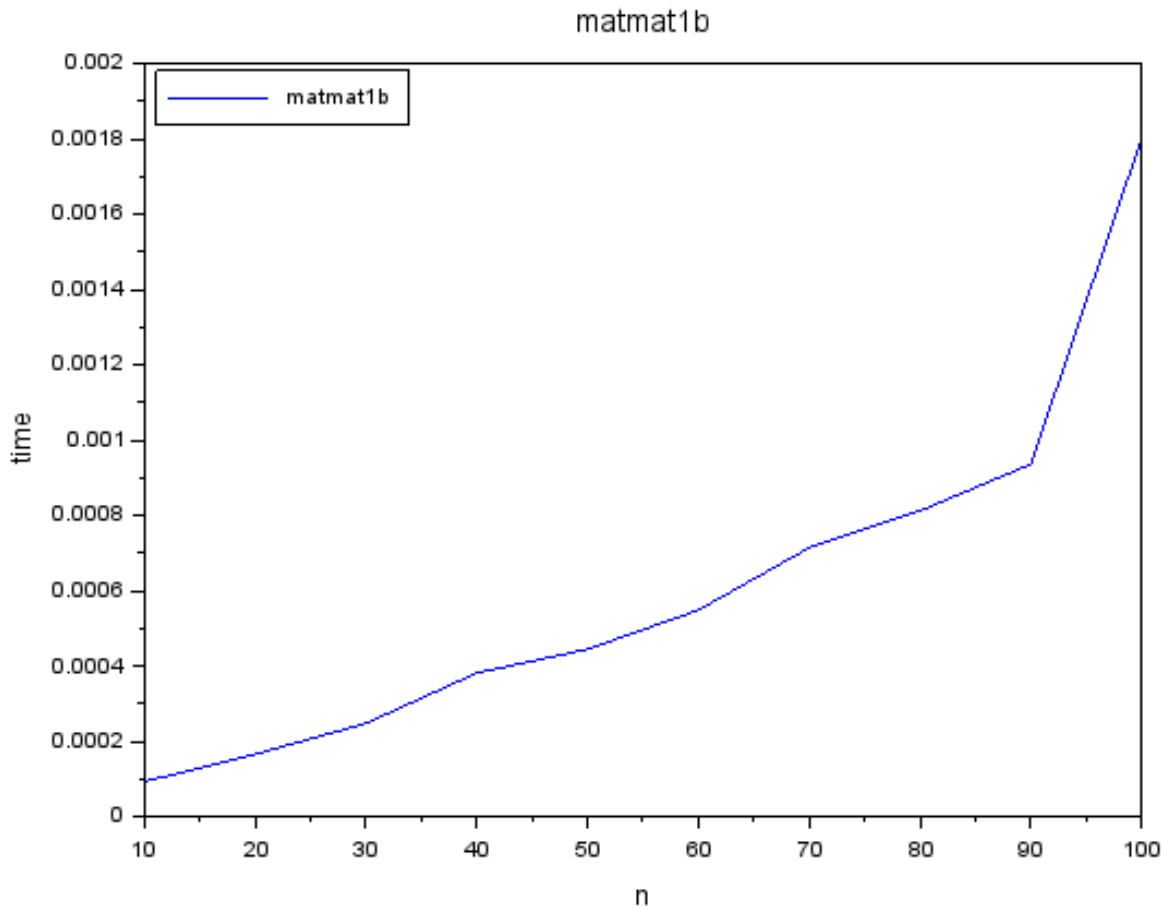
- Graphe pour la fonction `matmat2b`

On remarque qu'à chaque incrémentation de  $n$ , les mesures des fonctions `tic` et `toc` par rapport au temps augmente d'une manière plus rapide que `matmat3b` car on execute que deux boucles pour résoudre la fonction `matmat2b`.



- **Graphe pour la fonction matmat1b**

On remarque qu'à chaque incrémentation de  $n$ , les mesures des fonctions tic et toc par rapport au temps augmente d'une manière plus rapide que matmat2b et matmat3b car on exécute qu'une boucle pour résoudre la fonction matmat1b du coup on réduit l'ordre de complexité par rapport à la matrice matmat2b et matmat3b.



4)- Le résultat le plus performant c'est les résultats de la fonction matmat1b  
Le code source de cette exercice se trouve dans le fichier exo8TP2.sci

## 5 Exercice 2 Tp3 (exo2TP3.sci)

1)- Les algorithmes de résolution par remontée et descente

- Fonction Usolve

```
function [x] = usolve(U,b)
    n = size(U,1);
    x = zeros(n);
    x(n)=b(n)/U(n,n);
    for i = n-1:-1:1
        x(i) = (b(i) - U(i, (i + 1):n) * x((i + 1):n)) / U(i, i);
    end
endfunction
```

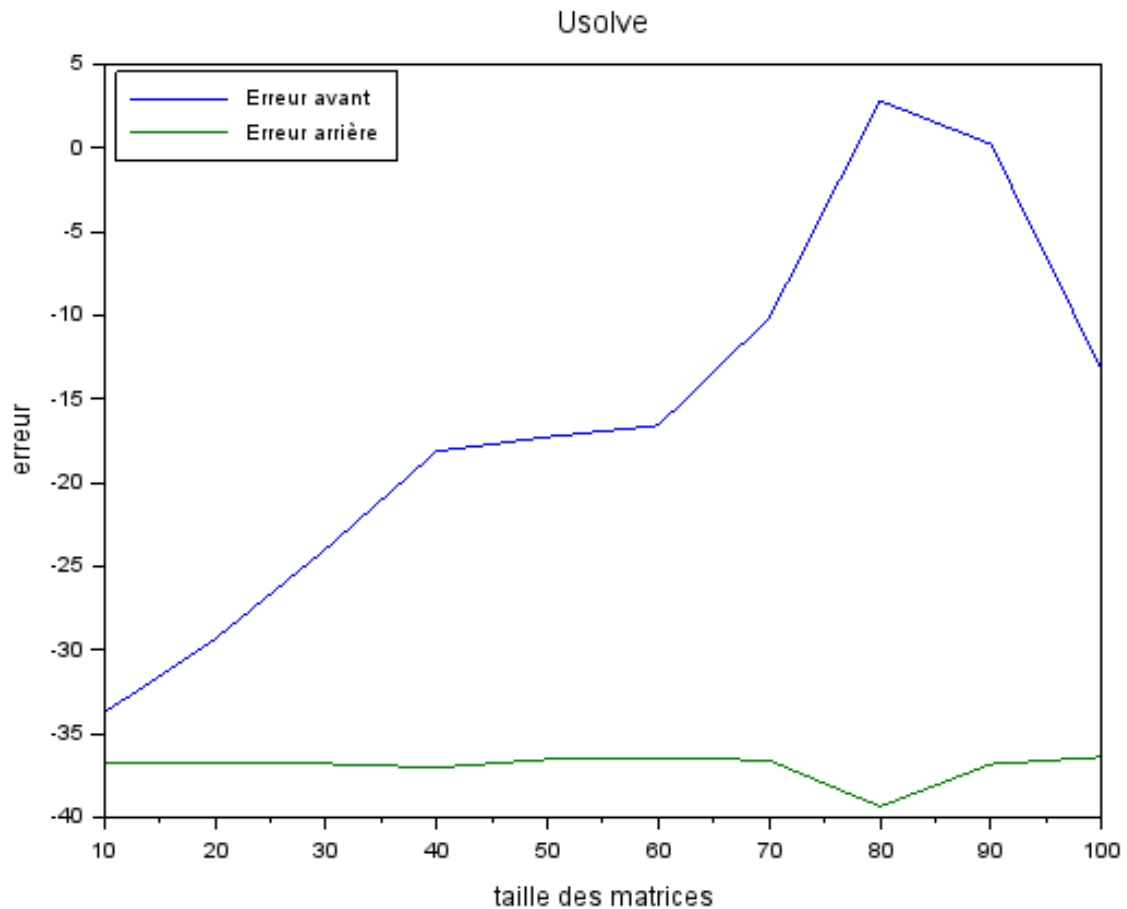
- Fonction Lsolve

```
function [x] = lsolve(L,b)
    n = size(L,1);
    x = zeros(n);
    x(1) = b(1) / L(1, 1);
    for i = 2:n
        x(i) = (b(i) - (L(i, 1:(i - 1)) * x(1:(i - 1)))) / L(i, i);
    end
endfunction
```

## 2)- Test et validation des algorithmes

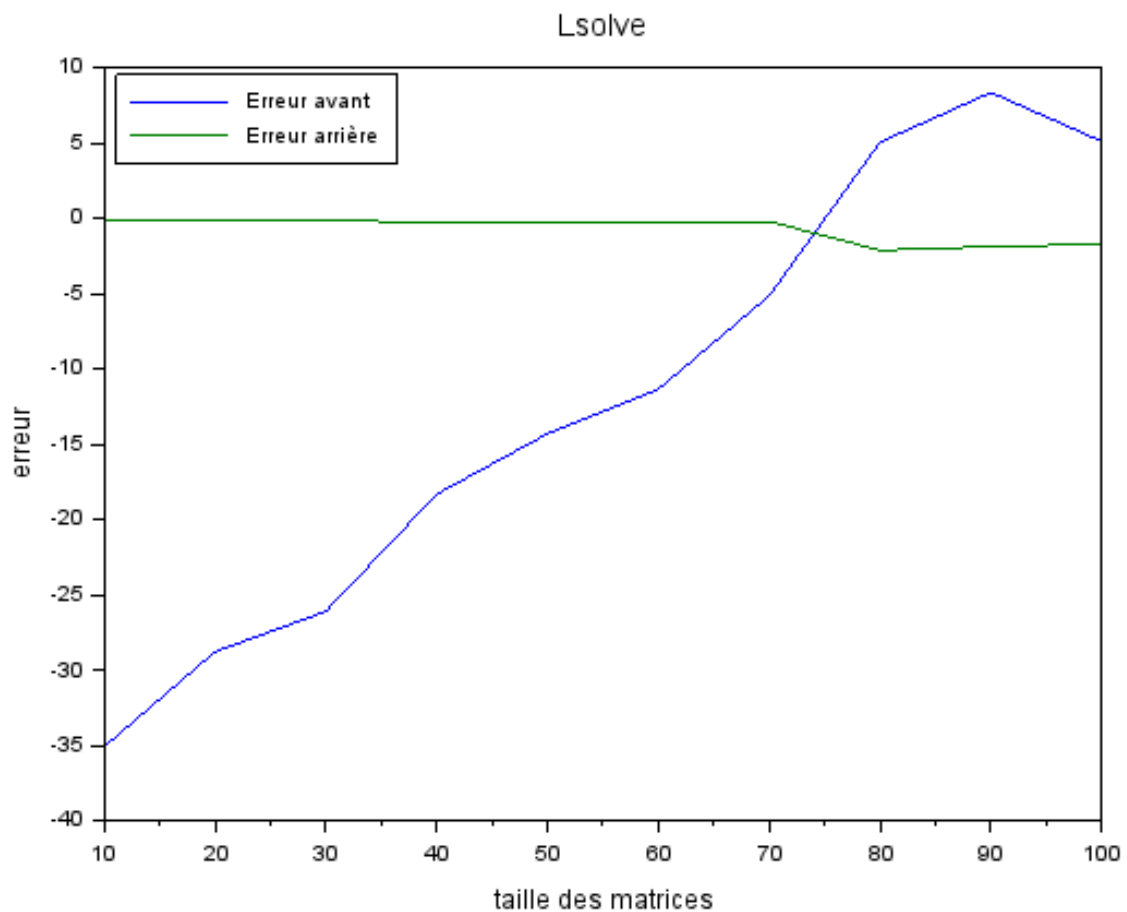
### • Pour la fonction **Usolve**

- L'erreur avant augmente très rapidement jusqu'à un certain degré de la taille de la matrice (Jusqu'à  $n \approx 80$ ) et puis elle redescend.
- L'erreur arrière reste stable jusqu'à un certain degré de la taille de la matrice ( $n \approx 70$ ) après ça elle baisse d'un petit peu jusqu'au niveau ( $n \approx 80$ ) après ça elle redevient stable.
- De ça on remarque que toujours l'erreur arrière est beaucoup mieux que l'erreur avant. - L'erreur arrière est acceptable car elle est presque stable au  $10^{-37}$ .
- Lorsque la taille de la matrice est petite, l'erreur avant est acceptable.



- **Pour la fonction Usolve**

- L'erreur avant augmente très rapidement jusqu'à un certain degré de la taille de la matrice (Jusqu'à  $n \approx 90$ ) et puis elle diminue.
- L'erreur arrière reste presque stable.
- De ça on remarque que l'erreur avant est beaucoup mieux que l'erreur arrière jusqu'à que leur courbe se croise (vers  $n \approx 75$ ) après ça redeviens le contraire, l'erreur arrière est beaucoup mieux que l'erreur avant.
- L'erreur arrière n'est pas acceptable car elle est grande .
- Lorsque la taille de la matrice est petite, l'erreur avant est acceptable.



## 6 Exercice 3 Tp3 (exo3TP3.sci)

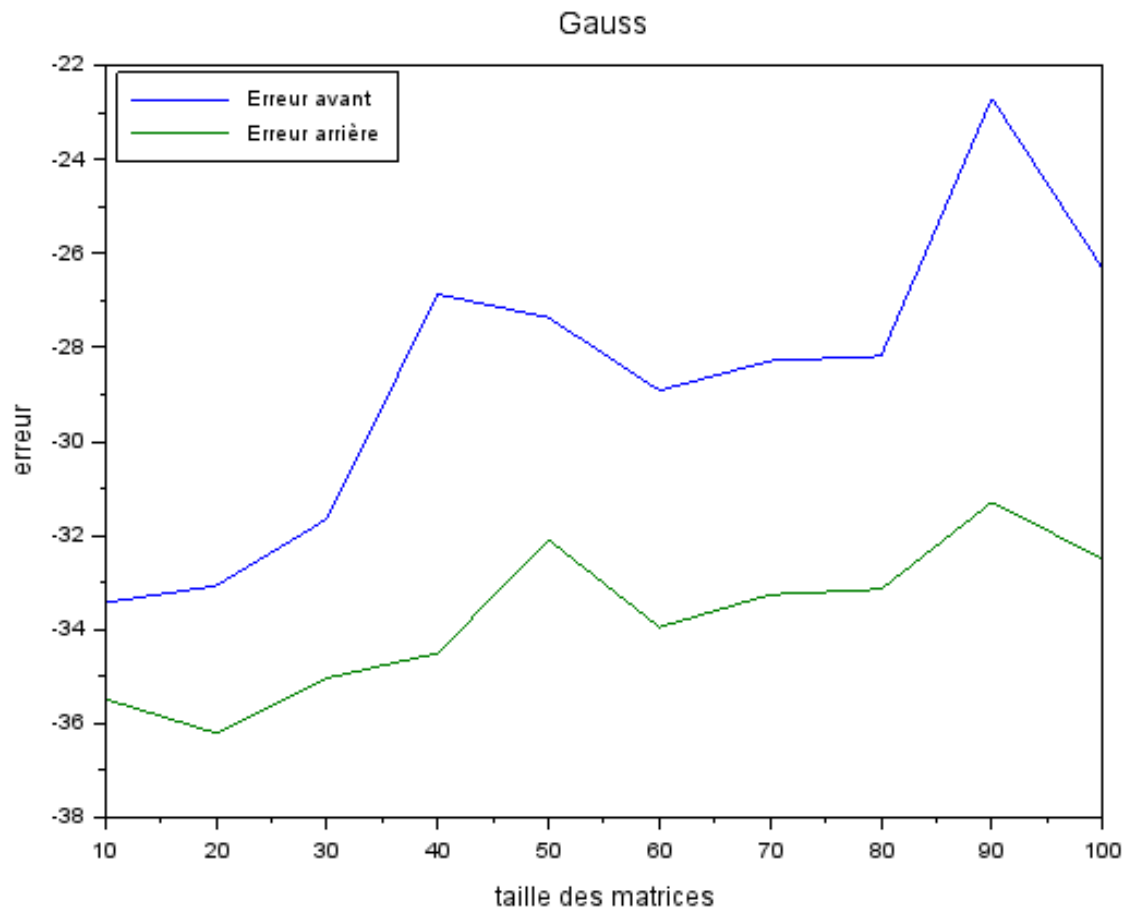
1)- Algorithme de résolution par élimination de Gauss sans pivotage

```
function [x] = gausskij3b(A,b)
    n = size ( A, 1);
    for k = 1:n-1
        for i = k+1:n
            mik = A(i, k)/A(k,k);
            b(i) = b(i) - mik * b(k);
            for j = k+1:n
                A(i, j) = A(i, j) - mik * A(k, j);
            end
        end
    end
    x = usolve ( A, b);
endfunction
```

2)- Test et validation de l'algorithme

- L'erreur arrière reste presque stable.
- De ça on remarque que l'erreur arrière est beaucoup mieux que l'erreur avant.
- L'erreur arrière est acceptable.
- Lorsque la taille de la matrice est petite, l'erreur avant est acceptable.





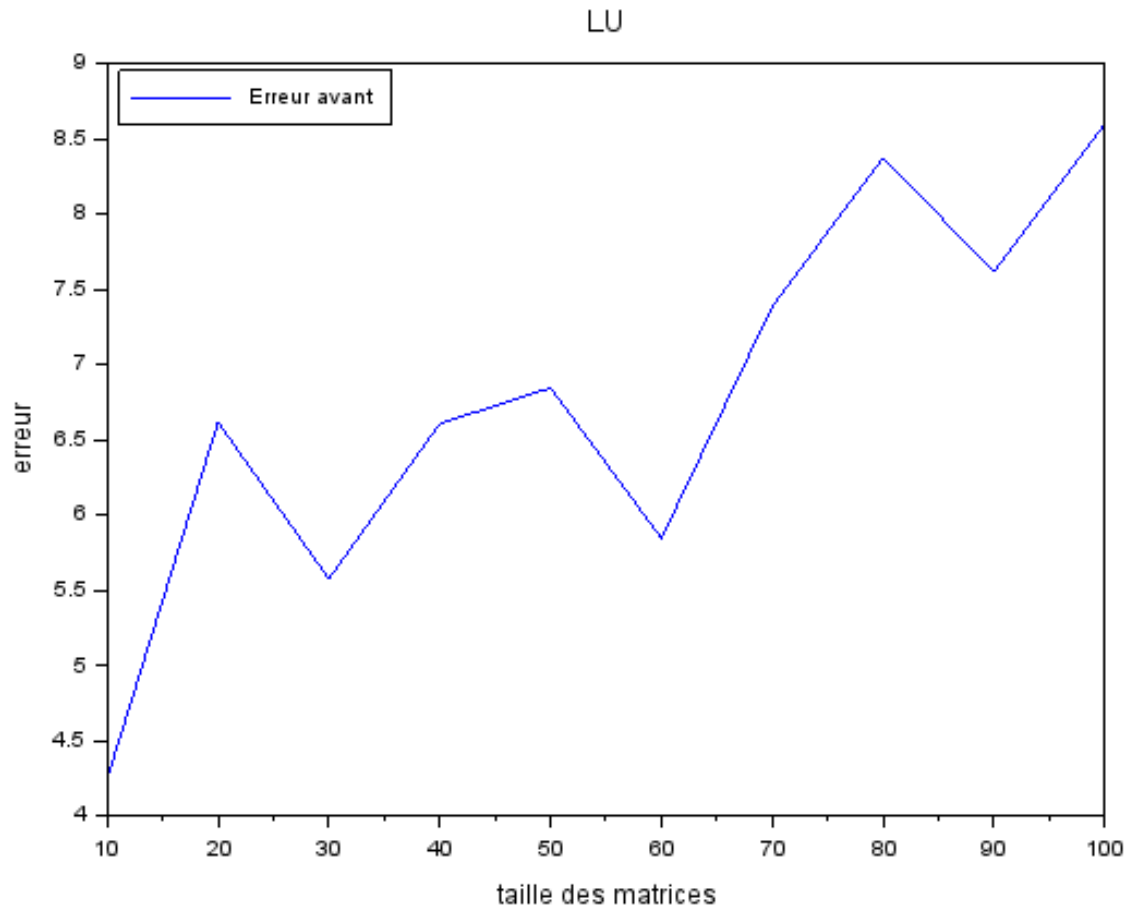
## 7 Exercice 4 Tp3 (exo4TP3.sci)

1)- Algorithme de factorisation LU

```
function [L,U] = mylu3b(A)
    n = size ( A, 1);
    for k = 1:n-1
        for i = k+1:n
            A(i,k) = A(i,k)/A(k,k);
        end
        for i = k+1:n
            for j = k+1:n
                A(i,j) = A(i,j) - A(i,k)*A(k,j);
            end
        end
    end
    L = usolve ( A, b);
    U = usolve ( A, b);
    disp(A);
endfunction
```

2)- Test et validation de l'algorithme

- L'erreur arrière n'est pas acceptable car elle est grande.
- Lorsque la taille de la matrice devient grande, l'erreur avant redevient de plus en plus très grande .



### 3)- Amélioration de l'algorithme de factorisation LU

```

function [L,U] = mylubl(A)
    n = size ( A, 1);
    for k = 1:n-1
        A(k+1:n,k) = A(k+1:n,k)/A(k,k);
        A(k+1:n,k+1:n) = A(k+1:n,k+1:n) - A(k+1:n,k)*A(k,k+1:n);
    end
    L = usolve ( A, b);
    U = usolve ( A, b);
    disp(A);
endfunction

```

## 8 Annexe

Code sur github:

[https://github.com/anisus07/Calcul\\_Numerique\\_MEHIDI\\_ANIS.git](https://github.com/anisus07/Calcul_Numerique_MEHIDI_ANIS.git)