Analyse numérique matricielle
TP2-3

Vernet Elliot, Courtois Thibault, Imperatore Valentin

January 3, 2024

Contents

1 Introduction

Ce TP porte sur **deux méthodes de résolutions directes de systèmes linéaires**.

La première est la **factorisation LU**. Elle dérive de l'**élimination de Gauss-Jordan** qui est une méthode de résolution de système linéaire popularisée en Europe par Carl Friedrich Gauss qui l'utilisa pour décrire le mouvement d'un astéroïde en 1810 et par Willhelm Jordan qui explicitera la méthode plus en détail en 1888. Cette méthode consiste à **factoriser la matrice d'un système en un produit de deux matrices triangulaires, l'une inférieure (Lower) et l'autre supérieure (Upper)**.

La deuxième méthode est la **factorisation de Cholesky**. Celle-ci **permet de factoriser une matrice symétrique définie positive en un produit d'une matrice et de sa transposée**. Elle fut utilisée pour la première fois par l'ingénieur français André-Louis Cholesky en 1902 lors de la résolution d'un problème de topographie.

Ces deux méthodes permettent de trouver des solutions approchées d'un système linéaire donné.

Mise en oeuvre de la factorisation LU

Pour écrire l'algorithme de factorisation LU nous avons travaillé sur un cas simple : **une matrice de système linéaire tridiagonale**.

$$\mathrm{A} = \begin{pmatrix} b_1 & c_2 & & & \\ a_1 & b_2 & c_3 & & \\ 0 & \ddots & \ddots & \ddots & \\ \vdots & & a_{n-2} & b_{n-1} & c_n \\ 0 & \cdots & 0 & a_{n-1} & b_n \end{pmatrix}$$

On cherche une matrice triangulaire inférieure de la forme :

$$\mathrm{L} = \begin{pmatrix} 1 & & & & \\ l_1 & 1 & & & \\ 0 & l_2 & \ddots & & \\ \vdots & & \ddots & \ddots & \\ 0 & \cdots & 0 & l_{n-1} & 1 \end{pmatrix}$$

et une matrice triangulaire supérieure de la forme :

$$\mathrm{U} = \begin{pmatrix} d_1 & u_2 & 0 & \cdots & 0 \\ & d_2 & u_3 & \ddots & \\ & & \ddots & \ddots & 0 \\ & & & d_{n-1} & u_n \\ & & & & d_n \end{pmatrix}$$

telles que :

$$A = \begin{pmatrix} b_1 & c_2 & & & \\ a_1 & b_2 & c_3 & & \\ 0 & \ddots & \ddots & \ddots & \\ \vdots & & a_{n-2} & b_{n-1} & c_n \\ 0 & \cdots & 0 & a_{n-1} & b_n \end{pmatrix} = \begin{pmatrix} 1 & & & & \\ l_1 & 1 & & & \\ 0 & l_2 & \ddots & & \\ \vdots & & \ddots & \ddots & \\ 0 & \cdots & 0 & l_{n-1} & 1 \end{pmatrix} . \begin{pmatrix} d_1 & u_2 & 0 & \cdots & 0 \\ & d_2 & u_3 & \ddots & \\ & & \ddots & \ddots & 0 \\ & & & d_{n-1} & u_n \\ & & & & d_n \end{pmatrix}$$

On obtient alors $\mathrm{A} = \begin{pmatrix} d_1 & u_2 & & & \\ l_1 d_1 & l_1 u_2 + d_2 & u_3 & & \\ 0 & \ddots & \ddots & \ddots & \\ \vdots & & \ddots & \ddots & u_n \\ 0 & \cdots & 0 & l_{n-1} d_{n-1} & l_{n-1} u_n + d_n \end{pmatrix}$

On obtient alors le système équivalent :

$$\begin{cases} d_1 &= b_1 \\ u_i &= c_i,\ 1 \leq i \leq n \\ l_1 d_1 &= a_1 \rightarrow l_1 = \frac{a_1}{d_1} \text{ (les } d_i \text{ sont non nuls)} \\ d_1 &= b_1 - l_{i-1}u_i \\ . & \\ . & \\ . & \\ l_n &= \frac{a_n}{d_n} \\ d_n &= b_n - l_{n-1}u_n \end{cases}$$

Soit de manière algorithmique :

$$\begin{cases} d_1 = b_1 \\ \text{Pour i = 2 à n faire :} \\ \quad u_i = c_i \\ \quad l_{i-1} = \frac{ai-1}{di-1} \\ \quad d_i = b_i - l_{i-1}u_i \end{cases}$$

Cet **algorithme nous permet de trouver les coefficients des matrices lower et upper**.

Pour résoudre un système linéaire en utilisant la méthode de factorisation LU, on effectue l'algorithme de Thomas :

$$\begin{aligned} & Ax = f \\ \iff & LUx = f \\ \iff & \begin{cases} Ly &= f \text{ (étape de descente)} \\ Ux &= y \text{ (étape de montée)} \end{cases} \end{aligned}$$

Détermination de l'algotihme de descente :

$$Ly = f \iff \begin{pmatrix} 1 & & & & \\ l_1 & 1 & & & \\ 0 & l_2 & \ddots & & \\ \vdots & & \ddots & \ddots & \\ 0 & \cdots & 0 & l_{n-1} & 1 \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix} = \begin{pmatrix} f_1 \\ f_2 \\ \vdots \\ f_n \end{pmatrix} \iff \begin{cases} y_1 &= f_1 \\ y_{i-1} &= f_i - l_{i-1}y_{i-1},\ i = 2, ..., n \end{cases}$$

Détermination de l'algorithme de montée :

$$Ux = y \iff \begin{pmatrix} d_1 & u_2 & 0 & \cdots & 0 & d_2 & u_3 & \ddots \\ & \ddots & \ddots & 0 & & & & \\ & & d_{n-1} & u_n & & & & \\ & & & d_n & & & & \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}$$

$$\iff \begin{cases} d_n x_n &= \\ u_n x_n + d_{n-1} x_{n-1} &= y_{n-1} \\ & . \\ & . \\ & . \end{cases}$$

$$\iff \begin{cases} x_n &= \frac{y_n}{d_n} \\ x_i &= \frac{y_i - u_{i+1} x_{i+1}}{d_i},\ i = n-1, ..., 2 \end{cases}$$

On peut programmer ces différents algorithmes sur le logiciel Matlab. Nous les réutiliserons par la suite dans une application physique qui nécessite la résolution d'une équation différentielle.

```
function [l, d, u] = lu_facto(a, b, c)
% Factorisation LU d'une matrice tridiagonale
% inputs:
%   a   vecteur contenant la sous-diagonale de A
%   b vecteur contenant la diagonale de A
%   c vecteur contenant la sur-diagonale de A
% outputs:
%   l la sous-diagonale de la matrice triangulaire supérieure
%   d la diagonale de la matrice triangulaire supérieure
%   u la sur-diagonale de la matrice triangulaire supérieure

n = length (b);
d(1) = b(1);
for i = 2 : n
    u(i) = c(i);
    l(i-1) = a(i-1)/d(i-1);
   | d(i) = b(i) - u(i)*l(i-1);
end
end
```

Figure 1: Algorithme de factorisation LU

```
function [l, d, u] = lu_facto(a, b, c)
% Factorisation LU d'une matrice tridiagonale
% inputs:
%   a   vecteur contenant la sous-diagonale de A
%   b vecteur contenant la diagonale de A
%   c vecteur contenant la sur-diagonale de A
% outputs:
%   l la sous-diagonale de la matrice triangulaire supérieure
%   d la diagonale de la matrice triangulaire supérieure
%   u la sur-diagonale de la matrice triangulaire supérieure

n = length (b);
d(1) = b(1);
for i = 2 : n
    u(i) = c(i);
    l(i-1) = a(i-1)/d(i-1);
    d(i) = b(i) - u(i)*l(i-1);
end
end
```

Figure 2: Algorithme de Thomas

On va maintenant mettre en application cet algorithme.

Notre objectif est de déterminer le déplacement des points d'une tige repérés par leur abscisse x suite à la déformation de la tige sur son axe transversal par une force linéique f(x).

Ce problème revient à trouver les solutions du système suivant :

$$(\mathrm{P}) : \begin{cases} -u''(x) + c(x)u(x) &= f(x),\ x \in (0, L) \\ u(0) &= 0 \\ u(L) &= 0 \text{ (Conditions limites de type Dirichlet.)} \end{cases}$$

Avec u(x) : le déplacement et c(x) : une fonction caractéritique de la poutre.

La première étape consiste à linéariser ce système.
Pour cela, on va discrétiser le problème avec une subdivision régulière de pas h de l'intervalle considéré [0, L]

Soit $N \in \mathbb{N}$. On prend $h = \frac{L}{N}$. Alors pour tout $0 \leq i \leq N$, on définit $x_i = ih$, et $u_i = u(x_i)$. Les x_i sont alors les noeuds de la subdivision régulière de pas h.

Si u est solution du problème (P) alors celui-ci est équivalent au problème

$$(\text{P1}) : \begin{cases} -u''(x_i) + c(x_i)u(x_i) &= f(x_i),\ 0 \leq i \leq N-1 \\ u(x_0) &= 0 \\ u(x_N) &= 0 \text{ (le premier et deuxième noeud correspondent à u(0) et u(L).)} \end{cases}$$

En supposant que u est assez régulière, on a :
(A) : $u(x_i + h) = u(x_{i+1}) = u(x_i) + hu'(x_i) + \frac{h^2}{2}u''(x_i) + \frac{h^3}{6}u^{(3)}(x_i) + \bigcirc(h^4)$
(B) : $u(x_i - h) = u(x_{i-1}) = u(x_i) - hu'(x_i) + \frac{h^2}{2}u''(x_i) - \frac{h^3}{6}u^{(3)}(x_i) + \bigcirc(h^4)$

Ainsi, (A) + (B) : $\Longrightarrow u''(x_i) = \frac{u(x_{i-1}) - 2u(x_i) + u(x_{i+1})}{h^2} + \bigcirc(h^2)$

On obtient ainsi une approximation de $u''(x_i)$ à l'ordre 2. Et le système est linéarisé :

$$(\text{P2}) : \begin{cases} -\frac{u(x_{i-1})}{h^2} + 2\frac{u(x_i)}{h^2} - \frac{u(x_i)}{h^2}c(x_i) - \frac{u(x_{i+1})}{h^2} &= f(x_i) + \bigcirc(h^2),\ 0 \leq i \leq N-1 \\ u(x_0) &= 0 \\ u(x_N) &= 0 \end{cases}$$

$\Longleftrightarrow$

$$\begin{pmatrix} \frac{2}{h^2} + c(x_1) & -1 & & & \\ -1 & \frac{2}{h^2} + c(x_2) & -1 & & \\ 0 & \ddots & \ddots & \ddots & \\ \vdots & & -1 & \frac{2}{h^2} + c(x_{N-1}) & -1 \\ 0 & \cdots & 0 & -1 & \frac{2}{h^2} + c(x_N) \end{pmatrix} \begin{pmatrix} u(x_1) \\ u(x_2) \\ \vdots \\ u(x_N) \end{pmatrix} = \begin{pmatrix} f(x_1) + \bigcirc(h^2) \\ f(x_2) + \bigcirc(h^2) \\ \vdots \\ f(x_N) + \bigcirc(h^2) \end{pmatrix}$$

$\Longleftrightarrow A_h U_h = F_h$

On a réussi à approcher le problème par un système linéaire.

On va maintenant observer la précision et la rapidité de la résolution de ce système linéaire et comparer la solution approchée trouvée par l'algorithme avec la solution exacte du problème.
Pour cela, il faut définir à l'avance la solution exacte et définir la fonction caractéristique c(x).
On considère $c(x) = x(1-x)$ et $u(x) = c(x)\exp(x)$
Pour être sûr que u est l'unique solution du problème, il faut que la fonction f vérifie l'équation différentrielle.
On doit donc avoir :
$f(x) = -u''(x) + c(x)u(x)$

Or :
$-u''(x) = -(-2\exp(x) + (1-2x)\exp(x) + (1-2x)\exp(x) + (x-x^2)\exp(x))$

$= 4x\exp(x) - (x - x^2)\exp(x) = (x^2 + 3x)\exp(x)$

Donc on définit f telle que : $f(x) = (x^2 + 3x)\exp(x) + c(x)\exp(x)$

On résout le problème pour N = 10, 50, 100, 1000, 10000, 100000² et on observe la norme de l'erreur et le temps de calcul.

On obtient le tableau suivant :

N	E(N)	T(N)
10	0.0049	0.2686
50	4.352e-04	0.2953
100	1.5392e-04	0.2508
1000	4.8673e-06	0.2898
10000	1.5491e-07	0.4138
100000	8.8956e-08	0.8485

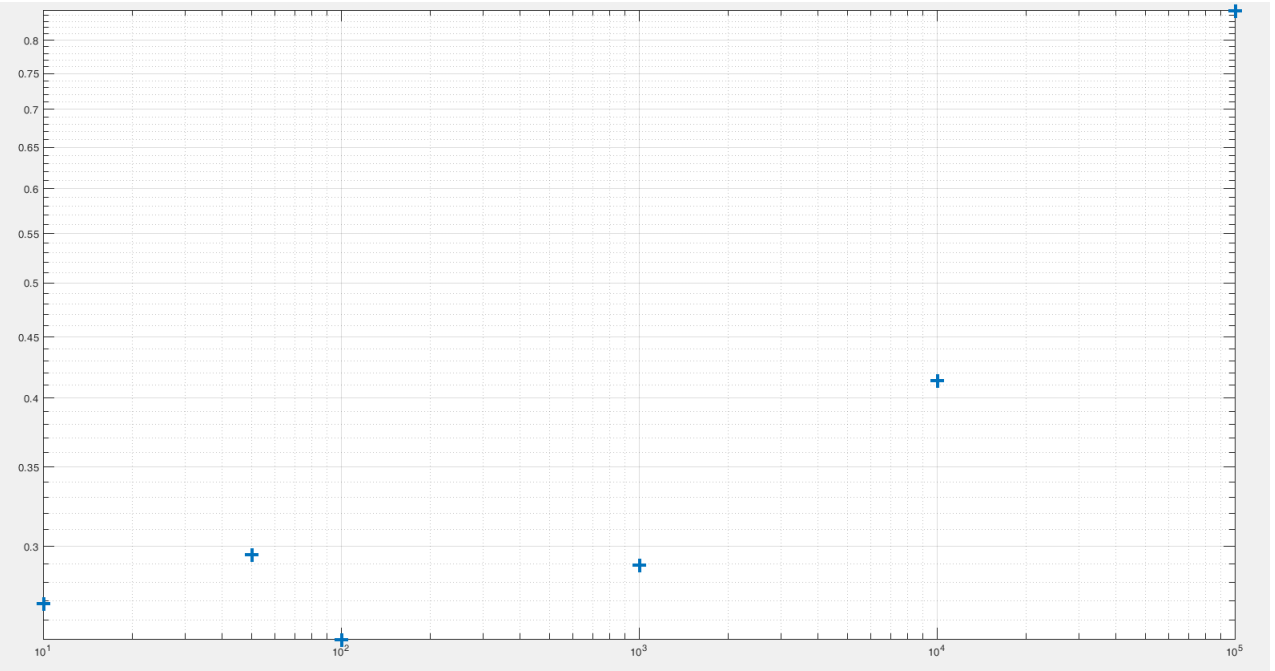

Figure 3: Temps de calcul en fonction de N

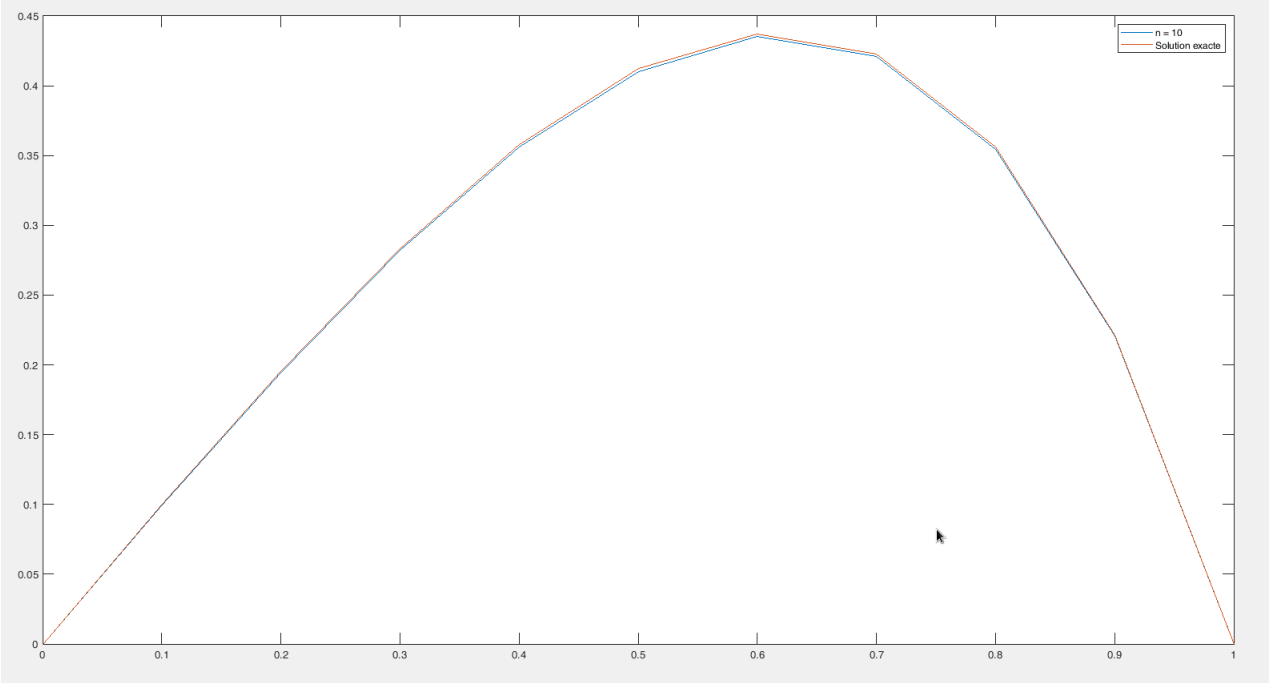

Figure 4: Comparaison entre la solution approchée et la solution exacte n = 10

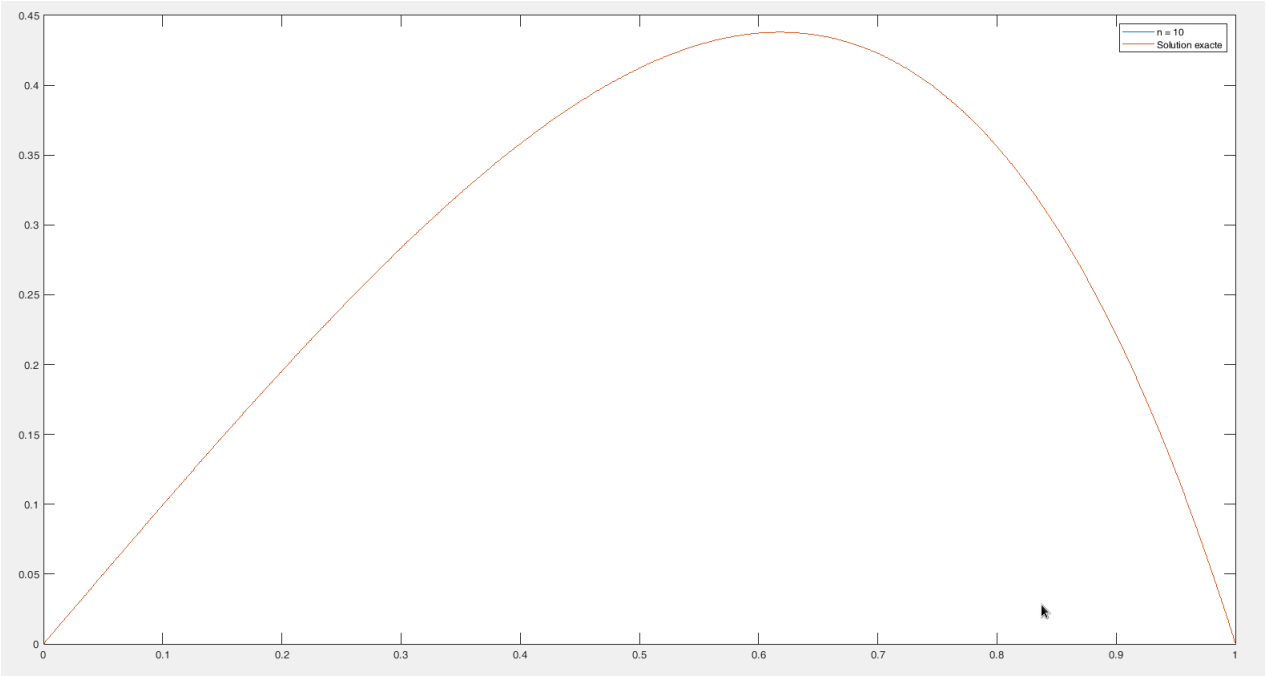

Figure 5: Comparaison entre la solution approchée et la solution exacte n = 1000

En conclusion, on observe que **plus la subdivision est fine et plus la solution approchée est proche de la solution réelle. Le temps de calcul augmente linéairement en fonction de n.**

2 Mise en oeuvre de la factorisation de Cholesky

Nous travaillons encore une fois sur une matrice A tridiagonale. Celle-ci est en plus **symétrique définie positive**. On souhaite décomposer la matrice de la façon suivante :

$A = R^t R$

avec R = $\begin{pmatrix} r_{11} & & \\ \vdots & \ddots & \\ r_{n1} & \cdots & r_{nn} \end{pmatrix}$

On a alors :

$$\text{A} = \begin{pmatrix} a_{11} & a_{12} & & & \\ a_{21} & \ddots & \ddots & & \\ 0 & \ddots & \ddots & \ddots & \\ \vdots & & \ddots & \ddots & a_{(n-1)n} \\ 0 & \cdots & 0 & a_{n(n-1)} & a_{nn} \end{pmatrix}$$

$$= \begin{pmatrix} r_{11} & & \\ \vdots & \ddots & \\ r_{n1} & \cdots & r_{nn} \end{pmatrix} \begin{pmatrix} r_{11} & \cdots & r_{n1} \\ & \ddots & \vdots \\ & & r_{nn} \end{pmatrix}$$

$$= \begin{pmatrix} r_{11}^2 & \cdots & \cdots & r_{11}r_{n1} \\ \vdots & r_{21}^2 + r_{22}^2 & & \\ \vdots & & \ddots & \\ r_{11}r_{n1} & & & r_{n(n-1)}^2 r_{nn}^2 \end{pmatrix}$$

On a donc :

$$\begin{cases} \sqrt{a_{11}} = r_{11} \\ r_{21} = \frac{a_{21}}{r_{11}} \\ r_{22} = \sqrt{a_{22} - r_{21}^2} \\ \vdots \\ r_{ii} = \sqrt{a_{ii} - \sum_{j=1}^{i-1} r_{ij}^2}, \text{ pour i } \in [2, n] \\ r_{ij} = \frac{r_{ij} - \sum_{k=1}^{j-1} r_{ik} r_{jk}}{r_{jj}}, \text{ pour j} < i \end{cases}$$

Soit de façon algorithmique :
donnée : A(n;n)
sortie : R(n;n)
R = zeros(n;n)
for i = 1:n
    for j = 1:i-1
    $R(i,j) = (A(i,j) - \sum_{k=1}^{j-1} R_{i,k}R_{j,k})/R(j,j)$
    end
    $R(i,i) = \sqrt{A(i,i) - \sum_{j=1}^{i-1} R(i,j)^2}$
end

On implémente une fonction Cholesky(A) sur MatLab qui permet de calculer la matrice R :

```
function R = cholesky(A)
    N=size(A);
    N=N(1);
    R=zeros(N, N);
    for i=1:N
        for j=1:i-1
            s=0;
            for k=1:j-1
                s=s+R(i,k)*R(j,k);
            end
            R(i,j)=(A(i,j) - s)/R(j,j);
        end
        s=0;
        for j = 1:i-1
            s=s+R(i,j)^2;
        end
        R(i,i)=sqrt(A(i,i)-s);
    end
end
```

Figure 6: Cholesky(A)

On peut tester cette algorithme sur une matrice carrée de taille 3 :

$$\mathrm{A} = \begin{pmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{pmatrix}$$

On trouve :

```
R =

    1.4142         0         0
   -0.7071    1.2247         0
         0   -0.8165    1.1547
```

On va maintenant utiliser cette factorisation pour résoudre un système linéaire.
On a :
Ax = b $\iff R^t R x = b$

$$\iff \begin{cases} Ry &= b \\ {}^tRx &= y \end{cases}$$

On retrouve l'algorithme de montée et l'algorithme de descente trouvés précedemment.
Cependant, ils ne sont pas exactement pareils.

Algorithme de descente :

$$\begin{pmatrix} r_{11} & & \\ \vdots & \ddots & \\ r_{n1} & \cdots & r_{nn} \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix}$$

$$\iff \begin{cases} y_1 &= \frac{b_1}{r_{11}} \\ r_{21}y_1 + r_{22}y_2 &= b_2 \iff y_2 = \frac{b_2 - r_2 1 y_1}{r_2 2} \\ r_{31}y_1 + r_{33}y_3 &= b_3 \iff y_3 = \frac{b_3 - r_3 1 y_1}{r_3 3} \\ \vdots & \\ r_{n1}y_1 + r_{n2}y_2 + ... + r_{nn}y_n &= b_n \iff y_n = \dfrac{b_n - \sum_{j=1}^{i-1} r_{n,j} y_j}{r_{nn}} \end{cases}$$

Soit sur MatLab pour une matrice L triangulaire inférieure :

```
function z=descente(L,c)
    N = length(c);
    z = zeros(N,1);
    z(1) = c(1)/L(1,1);
    for i=2:N
        s=0;
        for j=1:i-1
            s = s + L(i,j)*z(j);
        end
        z(i)=(c(i)- s)/L(i,i);
    end
end
```

Figure 7: Algorithme de descente

D'une façon similaire, on obtient pour l'algortihme de montée :

```
function x = remonte(u,z)
    N = length(z);
    x = zeros(N,1);
    x(N) = z(N)/u(N,N);
    for i=N-1:-1:1
        s=0;
        for j=i+1:N
            s = s + u(i,j)*x(j);
        end
        x(i)=(z(i)- s)/u(i,i);
    end
    end
```

Figure 8: Algorithme de montée

On peut maintenant appeler ces deux fonctions dans une fonction SolveCholesky(A,b) qui résout le système linéaire Ax=b.

```
function x = SolveCholesky(A,b)
    R = cholesky(A);
    y = descente(R,b);
    x = remonte(R',y);
end
```

Figure 9: SolveCholesky(A,b)

On va maintenant utiliser ce programme pour résoudre le problème de la partie précedente et comparer les résultats des deux méthodes.

On obtient le tableau de résultats suivant :

N	E(N)	T(N)
10	0.0049	0.2934
50	4.352e-04	0.3041
100	1.5392e-04	0.4232
1000	4.8673e-06	0.4525
10000	1.5491e-07	0.4865
100000	8.8956e-08	0.5082

Les précisions sont donc identiques et on obtient les mêmes graphiques que pour la factorisation LU.
Cependant, **les temps de calcul évoluent différemment et sont moins élevés pour n grand :**

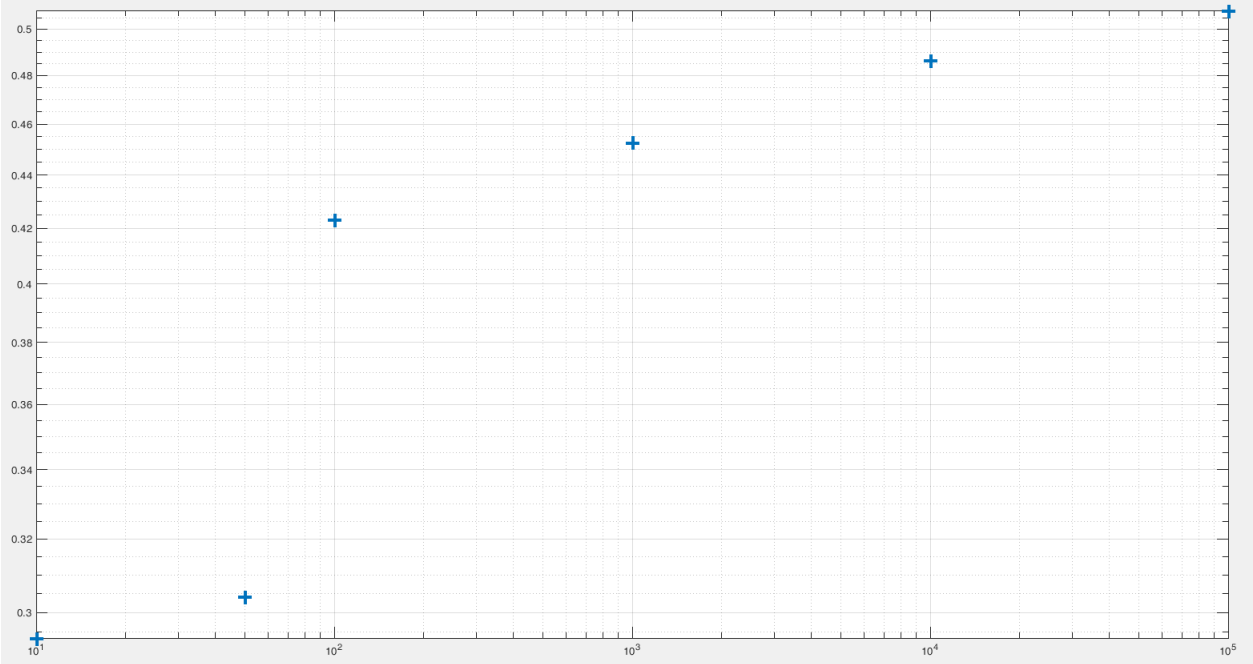

Figure 10: Temps de calcul en fonction de N

3 Conclusion

Ainsi, **les deux méthodes sont équivalentes en terme de précision, mais pas en terme de temps de calcul. Dans le cas où le système est définie par une matrice symétrique définie positive, il semble judicieux d'utiliser la factorisation de Cholesky qui est plus économe en calculs pour les subdivisions les plus fines.**