
Rapport Master M1

**Aicha Maaoui**

Master Calcul Haute Performance Simulation (CHPS)

# TP4 Calcul Numerique

Décembre 2021

Institut des Sciences et Techniques des Yvelines (ISTY)

# Abstract

Le TP 4 "Exploitation des Structures et Calcul Creux" est associé au cours de Calcul Numérique "Vers l'optimisation d'algorithmes numériques et algèbre creuse".
Il a pour but:

- Implémentation de l'algorithme de la factorisation $LDL^T$ pour une matrice symétrique,
- Comparison de $LDL^T$ à $LU$,
- Implémentation des algorithmes de stockage COO, CSR et CSC pour les matrices creuses,
- Algorithme de calcul du transposée d'une matrice creuse,
- Algorithme de calcul du Produit Matrice-Vecteur creux pour le format CSR,
- Test et validation, étude de compléxité, ainsi que des mesures de performance.

# Contents

# List of Tables

# Listings

# Chapter 1

# TP4 de Calcul Numérique

Les exercices 1.b, 4 et 5 sont réalisés sur scilab. Ce compte rendu comprend l'analyse des algorithmes à implémenter et les mesures de performances.

## 1.1 Exercice 1.b: Factorisation $LDL^T$ pour une matrice symétrique

1/ **Algorithme de la factorisation $LDL^T$ pour une matrice symétrique:**

## 1.2 Exercice 4: Stockage CSR et CSC

0/ **Stockage des matrices creuses:** Pour des matrices avec des éléments égaux à zéro, on peut faire recourt au différentes formats de stockage caractéristiques des matrices creuses.
Soit $nnz$ le nombre des éléments non nuls d'une matrice creuse $A \in R^{m \times n}$. On peut citer comme format de stockage des éléments non nuls d'une matrice creuse $A$:

- **COO (Coordinate Storage):** On stocke les éléments non nuls d'une matrice creuse dans $AA \in R^{1 \times nnz}$ (selon les lignes). Pour l'accès aux éléments, les indices sur les lignes i et sur les colonnes j sont stockées dans $JR \in R^{1 \times nnz}$ et $JC \in R^{1 \times nnz}$, respectivement,
- **CSR (Compressed Sparse Matrix):** On stocke les éléments non nuls d'une matrice creuse dans $AA \in R^{1 \times nnz}$. Pour l'accès aux éléments, on stocke dans $JA \in R^{1 \times nnz}$ le numéro $j$ de la colonne respective a l'élément et dans $IA \in R^{1 \times (n+1)}$ les pointeurs sur les lignes.
- **CSC (Compressed Sparse Column):** On stocke les éléments non nuls d'une matrice creuse dans $AA \in R^{1 \times nnz}$ (selon les colonnes). Pour l'accès aux éléments, on stocke dans $JA \in R^{1 \times nnz}$ le numéro $i$ de la ligne respective a l'élément et dans $IA \in R^{1 \times (m+1)}$ les pointeurs sur les colonnes.

1/ Soit la matrice $A \in R^{8 \times 6}$ comme suit:

$$\text{A}= \begin{pmatrix} 15 & 0 & 0 & 22 & 0 & -15 & 0 & 0 \\ 0 & 11 & 3 & 0 & 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & -6 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 91 & 0 & 0 & 0 & 0 & 0 & 25 & 7 \\ 0 & 0 & 28 & 0 & 0 & 0 & 0 & -2 \end{pmatrix}$$

On a $nnz = 12$ éléments non nuls de la matrice $A$.

Stockage CSR de $A$:

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **AA** | 15 | 22 | -15 | 11 | 3 | 2 | -6 | 91 | 25 | 7 | 28 | -2 |
| **JA** | 1 | 4 | 6 | 2 | 3 | 7 | 4 | 1 | 7 | 8 | 3 | 8 |
| **IA** | 1 | 4 | 7 | 8 | 8 | 11 | 13 | - | - | - | - | - |

Table 1.1: Stockage CSR de la matrice A, exo.4.

2/ Stockage CSC de $A$:

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **AA** | 15 | 19 | 11 | 3 | 28 | 22 | -6 | -15 | 2 | 25 | 7 | -2 |
| **JA** | 1 | 4 | 2 | 2 | 5 | 1 | 3 | 1 | 2 | 4 | 4 | 5 |
| **IA** | 1 | 3 | 4 | 6 | 8 | 8 | 9 | 11 | 13 | - | - | - |

Table 1.2: Stockage CSC de la matrice A, exo.4.

3/ **Transposée de la matrice creuse $A$:**

Desciption de l'algorithme: Soit la matrice $A \in R^{6\times 8}$. la transposée de la matrice $A$ est $A^T \in R^{8\times 6}$. On considère le stockage COO. Soit la matrice triplet $\in R^{12\times 3}$, dont la repartition des colonnes est la suivante:

- **Première colonne:** On stocke les indices des lignes des éléments non nuls de $A$, i,
- **Deuxième colonne:** On stocke les indices des colonnes des éléments non nuls de $A$, j,
- **Troisième colonne:** On stocke les éléments non nuls de $A$.

Soit la matrice triplet obtenue à partir de la matrice $A$:

$$\text{triplet}= \begin{pmatrix} 1 & 1 & 15 \\ 1 & 4 & 22 \\ 1 & 6 & -15 \\ 2 & 2 & 11 \\ 2 & 3 & 3 \\ 2 & 7 & 2 \\ 3 & 4 & -6 \\ 5 & 1 & 91 \\ 5 & 7 & 25 \\ 5 & 8 & 7 \\ 6 & 3 & 28 \\ 8 & 6 & -2 \end{pmatrix}$$

Listing 1.1: Algorithme pour calculer le triplet d'une matrice creuse $A$

```
%Extract Triplet from matrix A
function [triplet]=Triplet(A)

[m n]=size(A); %taille de la matrice A (m*n)
nnz = 0; %nnz= elements non nuls de la matrice A

    for i=1:m
        for j=1:n
            if (A(i,j)~=0) then %condition: element non nul de la matrice A
                nnz = nnz+1; %nombre des elements non nuls augmente de 1
                %1ere colonne de triplet (indices i des lignes)
                triplet(nnz,1) = i;
                 %2eme colonne de triplet (indices j des colonnes)
                triplet(nnz,2) = j;
                %3eme colonne de triplet (elements non nuls de la matrice A)
                triplet(nnz,3) = A(i,j);
            end
        end
    end
endfunction
funcprot(0)
```

L'étape suivante consiste à inverser la première et la deuxième colonne dans triplet. Soit la matrice triplet' $\in R^{12\times 3}$, tel que:

$$\text{triplet'}= \begin{pmatrix} 1 & 1 & 15 \\ 4 & 1 & 22 \\ 6 & 1 & -15 \\ 2 & 2 & 11 \\ 3 & 2 & 3 \\ 7 & 2 & 2 \\ 4 & 3 & -6 \\ 1 & 5 & 91 \\ 7 & 5 & 25 \\ 8 & 5 & 7 \\ 3 & 6 & 28 \\ 8 & 6 & -2 \end{pmatrix}$$

Pour obtenir la matrice transposée de $A$ à partir de triplet', il faut organiser cette dernière selon les valeurs minimales des lignes de la première colonne. Si deux valeurs de lignes sont égales, on compare le minimum de la deuxième colonne. Soit la matrice tripletInvSor le résultat de cet algorithme. Alors:

$$\text{tripletInvSor}= \begin{pmatrix} 1 & 1 & 15 \\ 1 & 5 & 91 \\ 2 & 2 & 11 \\ 3 & 2 & 3 \\ 3 & 6 & 28 \\ 4 & 1 & 22 \\ 4 & 3 & -6 \\ 6 & 1 & -15 \\ 7 & 2 & 2 \\ 7 & 5 & 25 \\ 8 & 5 & 7 \\ 8 & 6 & -2 \end{pmatrix}$$

L'algorithme qui permet de calculer la tranposée d'une matrice creuse $A$ est donné en listing 1.2.

Listing 1.2: Algorithme pour calculer la tranposée d'une matrice creuse $A$

```
%Extract Triplet from matrix A to proceed
function [tripletInvSort]=SortedInverseTriplet(triplet)

nnz = size(triplet,1); %nnz= nombre des elements non nuls de la matrice triplet
temp = zeros(nnz,3); %Initialisation de la matrice temp
%Calcul du transposee de la matrice Triplet
    for i=1:nnz %de 1 a nnz= nbre des elements non nuls
        for j=1:3
            temp = triplet(i,1); %stockage de la 1ere colonne de triplet
            triplet(i,1)=triplet(i,2); %inverser les colonnes 1 et 2
            triplet(i,2)=temp; %triplet(i,2)=triplet(i,1)
        end
    end
%Sorting du Transposee de la matrice Triplet
max= triplet(1,1); %inialisation de la valeur max de triplet
    for i=1:(nnz-1)
            if (triplet(i,1) < triplet(i+1,1)) then
                max = triplet(i+1,1); %Recherche de la valeur max de triplet
            end
    end
k=1; %initialisation du scalaire k
    for i=1:max
        for j=1:nnz
            if(triplet(j,1)==i) then
                tripletInvSort(k,1) = triplet(j,1); %sorting ligne1
                tripletInvSort(k,2) = triplet(j,2); %sorting ligne2
                tripletInvSort(k,3) = triplet(j,3); %sorting ligne3
                k=k+1; %increment k de 1
            end
        end
    end
endfunction
funcprot(0)
```

Le programme écrit en Scilab est testé avec la matrice $A$ de l'exercice 4, comme indiqué dans listing 1.3.

Listing 1.3: Tranposée de la matrice creuse $A$ de l'exerice 4 et validation avec Scilab

```
%Matrice A de lexercice 4
A=[15,0,0,22,0,-15,0,0;0,11,3,0,0,0,2,0;0,0,0,-6,0,0,0,0;0,0,0,0,0,0,0,0;
91,0,0,0,0,0,25,7;0,0,28,0,0,0,0,-2];

[triplet]=Triplet(A); %Triplet de la matrice A, listing 1.1

[tripletInvSort]=SortedInverseTriplet(triplet); %Inverse Triplet organise de la
    matrice A, listing 1.2
disp(tripletInvSort); %display tripletInvSort

%Validation avec Scilab --> Validated
A1=sparse (A); %obtention de la matrice creuse A1 a partir de A
At=A1'; %Tranposee de la matrice creuse A
disp(At); %Display Tranposee de la matrice creuse A
```

Avec Scilab, on obtient la matrice transposée de $A$ suivante (de la forme "(i,j) val"):

( 1, 1) 15.
( 1, 5) 91.
( 2, 2) 11.
( 3, 2) 3.
( 3, 6) 28.
( 4, 1) 22.
( 4, 3) -6.
( 6, 1) -15.
( 7, 2) 2.
( 7, 5) 25.
( 8, 5) 7.
( 8, 6) -2.

**Il est à noter que la transposée d'une matrice creuse $A$ avec la format de stockage CSR est une matrice avec la format de stockage CSC.**

4/ La complexité de l'algorithme de calcul de la tranposée d'une matrice $A$ est: $O(3 \times nnz)$.

Le tableau 1.3 représente la moyenne de 10 mesures du temps de calcul de l'algorithme tranposée de la matrice creuse $A$ de l'exercice.

| **Matrice** | **Moyenne du temps de calcul** |
|---|---|
| $A \in R^{8\times6}$ | 0.0002152 s |

Table 1.3: Moyenne de temps de calcul de l'algorithme tranposée de $A$.

## 1.3 Exercice 5: Produit Matrice-Vecteur Creux

0/ **Les algorithmes de Stockages:** Les algorithmes des différents formats de stockages d'une matrice creuse $A$ sont donnés dans les listings de 1.5 à 1.7. Ils se basent sur $nnz$ le nombre des éléments non nuls d'une matrice creuse $A$ (listing 1.4).

Listing 1.4: Eléments non nuls d'une matrice creuse $A$

```
function [nnz]=NNZ_SPMat(A) %Calcul des elements ~=0 d'une matrice creuse A
[m n]=size(A); %taille de la matrice creuse A (m*n)
nnz = 0; %initialisation, nnz= elements non nuls de la matrice creuse A

    for i=1:m
        for j=1:n
            if (A(i,j)~=0) then %element non nul
                nnz = nnz+1; %incrementation du nombre des elements ~=0 de A
            end
        end
    end
endfunction
funcprot(0)
```

Listing 1.5: Stockage COO d'une matrice creuse $A$

```
%Stockage COO d'une matrice creuse A
function [AA,JA,IA]=COO_SPMat(A)

[m n]=size(A);  %taille de la matrice A (m*n)
[nnz]=NNZ_SPMat(A); %nombre des elements non nuls d'une matrice creuse A
AA=zeros(nnz,1); %initialisation du vecteur AA (nnz*1)
JA=zeros(nnz,1); %initialisation du vecteur JA (nnz*1)
IA=zeros(nnz,1); %initialisation du vecteur IA (nnz*1)

k=1; %initialisation du compteur k
    for i=1:m
        for j=1:n
            if (A(i,j)~=0) then %element non nul de A
                AA(k) = A(i,j); %remplissage du vecteur AA: elements ~=0 de A
                JA(k) = i; %remplissage du vecteur JA: i des elements ~=0 de A
                IA(k) = j; %remplissage du vecteur IA: j des elements ~=0 de A
                k=k+1; %incrementation de k
            end
        end
    end
endfunction
funcprot(0)
```

Listing 1.6: Stockage CSR d'une matrice creuse $A$

```
%Stockage CSR d'une matrice creuse A
function [AA,JA,IA]=CSR_SPMat(A)

[m n]=size(A);  %taille de la matrice A (m*n)
[nnz]=NNZ_SPMat(A); %nombre des elements non nuls d'une matrice creuse A
AA=zeros(nnz,1); %initialisation du vecteur AA (nnz*1)
JA=zeros(nnz,1); %initialisation du vecteur JA (nnz*1)
IA=zeros(n+1,1); %initialisation du vecteur IA (n+1*1)
k=1; %initialisation du compteur k
    for i=1:m
        for j=1:n
            if (A(i,j)~=0) then %element non nul de A
                AA(k) = A(i,j); %remplissage du vecteur AA (selon les lignes)
                JA(k) = j; %remplissage du vecteur JA: j des elements ~=0 de A
                k=k+1; %incrementation de k
            end
        end
    end
%Remplissage du vecteur IA
nnz1=1; %initialisation du compteur sur les nbres des elements non nuls
IA(1)=1; %1er element du vecteur IA
IA(n+1)=nnz+1; %Element (n+1) du vecteur IA
    for i=1:m
        for j=1:n
            if (A(i,j)~=0) then %element non nul de A
                nnz1=nnz1+1; %nombre des elements non nuls de A
            end
        end
IA(i+1)=nnz1; %Remplissage du vecteur IA par des pointeurs sur les lignes
    end
endfunction
funcprot(0)
```

Listing 1.7: Stockage CSC d'une matrice creuse $A$

```
%Stockage CSC d'une matrice creuse A
function [AA,JA,IA]=CSC_SPMat(A)

[m n]=size(A);  %taille de la matrice A (m*n)
[nnz]=NNZ_SPMat(A); %nombre des elements non nuls d'une matrice creuse A
AA=zeros(nnz,1); %initialisation du vecteur AA (nnz*1)
JA=zeros(nnz,1); %initialisation du vecteur JA (nnz*1)
IA=zeros(m+1,1); %initialisation du vecteur IA (m+1*1)
nnz1=1; %initialisation du compteur sur les lignes
k=1; %intialisation du compteur k
IA(1)=1; %1er element du vecteur IA
IA(n+1)=nnz+1; %Element (n+1) du vecteur IA
    for j=1:n %boucle sur les lignes
        for i=1:m %boucle sur les colonnes
            if (A(i,j)~=0) then %elements non nuls de A
                AA(k) = A(i,j); %vecteur AA (selon les colonnes)
                JA(k) = i; %vecteur JA: i des elements ~=0 de A
                k=k+1; %incrementation de k
                nnz1=nnz1+1; %incrementation de nnz1
            end
        end
IA(j+1)= nnz1; %Remplissage du vecteur IA par des pointeurs sur les colonnes
    end
endfunction
funcprot(0)
```

On peut également tester les différentes formats de stockage d'une matrice creuse $A$ et la comparer avec les fonctions de Scilab pour une matrice creuse, comme montré dans listing1.8.

Listing 1.8: Tests et Validation des algorithmes de stockage d'une matrice creuse $A$

```
%Test des formats de stockage d'une matrice creuse A: COO, CSR et CSC

A=[1,2,0,11,0;0,3,4,0,0;0,5,6,7,0;0,0,0,8,0;0,0,0,9,10]; %matrice creuse A(5*5)

%Stockage COO
[AA_COO,JA_COO,IA_COO]=COO_SPMat(A); %Algo COO

%Stockage COO dans Scilab
%Pour tester le stockage COO on utilise space(A) --> format(JA(i),IA(i),AA(i))
D=sparse(A); %(JA(i),IA(i),AA(i)) --> Algo validated

%On peut aussi utiliser [ij,AA_COO1,mn]=spget(D), ou D = matrice creuse de A
[ij,AA_COO1,mn]=spget(D); % COO avec Scilab

%Stockage CSR
[AA_CSR,JA_CSR,IA_CSR]=CSR_SPMat(A); %Algo CSR

%Stockage CSC
[AA_CSC,JA_CSC,IA_CSC]=CSC_SPMat(A); %Algo CSC

%Test avec Scilab de CSC
B=sparse(A);
[IA_CSC1,JA_CSC1,AA_CSC1]=sp2adj(B); %CSC avec Scilab

%Erreurs entre l'algo CSC et Scilab:
Err_AA = norm(AA_CSC-AA_CSC1); %Erreur entre AA de l'algo implemente et Scilab
Err_JA = norm(JA_CSC-JA_CSC1); %Erreur entre JA de l'algo implemente et Scilab
Err_IA = norm(IA_CSC-IA_CSC1); %Erreur entre IA de l'algo implemente et Scilab
disp(Err_AA); %Erreur =0 --> Algo validated
disp(Err_JA); %Erreur =0 --> Algo validated
disp(Err_IA); %Erreur =0 --> Algo validated

A1=[1,2,0,11,0;0,3,4,0,0;0,5,6,7,0]; %matrice A(5*3)
[AA_COO,JA_COO,IA_COO]=COO_SPMat(A1); %COO
[AA_CSR,JA_CSR,IA_CSR]=CSR_SPMat(A1); %CSR
[AA_CSC,JA_CSC,IA_CSC]=CSC_SPMat(A1); %CSC
```

1/ Soit $A \in R^{m \times n}$ une matrice creuse et $x \in R^{n \times 1}$. On cherche à calculer le produit $y = A \times x$. Donc, le vecteur résultat $y \in R^{m \times 1}$.

L'algorithme dans listing 1.9 est basé sur le stockage COO d'une matrice creuse $A$.
Cet algorithme prend en compte les algorithmes de calcul des éléments non nuls de la matrice creuse $A$ dans listing 1.4 et de stockage COO de la matrice creuse $A$ dans listing 1.5.

Listing 1.9: Produit Matrice creuse Vecteur

```
%A partir du Stockage COO, on calcule: y=A*x

function [y]= COO_SPMatVec(A,x)

[m n]= size(A); %dimensions de la matrice A(m,n)
y=zeros(m,1); %initialisation du vecteur y

%Appel des fonctions NNZ_SPMat et COO_SPMat
[nnz]=NNZ_SPMat(A); %Fonction qui calcule nnz, le nombre des elements non nuls
    de la matrice A
[AA,JA,IA]=COO_SPMat(A); %Fonction qui calcule AA, JA, IA du stockage COO

for i=1:nnz
    y(JA(i)) = y(JA(i))+AA(i)*x(IA(i)); %Calcul du produit y=A*x
end

endfunction
funcprot(0)
```

(i) et (ii)/ L'algorithme de multiplication matrice creuse $A$ vecteur $x$ est testé et validé avec Scilab, comme montré dans listing 1.10.

Listing 1.10: Tests et Validation du produit matrice creuse vecteur

```
%Test de l'algo produit matrice creuse vecteur

%Test n1 (i)
A= [1,0,0,2,0;3,4,0,5,0;6,0,7,8,9;0,0,10,11,0;0,0,0,0,12]; %matrice A(5*5)
x = [1,1,1,1,1]'; %n=5 (Ex5,i)
[y]= COO_SPMatVec(A,x); %Algo produit Mat Vec
disp(y); %Display le produit de A*x

%Test de l'algo avec Scilab
y1=A*x; %produit de A et x
err=norm(y-y1); %erreur entre y de l'algo et y1 de Scilab
disp(err); %err=0 --> Algo valide

%Test n2 (ii)
A2= [5,0,0,22,0,-15;0,11,3,0,0,4;0,0,0,-6,0,0]; %matrice A(6*3)
x2 = [1,0,0,1,0,0]'; %n=6 (Ex5,ii)
[y2]= COO_SPMatVec(A2,x2); %Algo produit Mat Vec
disp(y2); %Display le produit de A2*x2

%Test de l'algo avec Scilab
y3=A2*x2; %produit de A2 et x2
err2=norm(y3-y2); %erreur entre y2 de l'algo et y3 de Scilab
disp(err2); %err=0 --> Algo valide

funcprot(0)
```

La complexité de l'algorithme de produit matrice creuse vecteur dans listing 1.9 est évaluée à O($nnz$).

On peut calculer la moyenne de 10 mesures du temps de calcul de cet algorithme, comme indiqué dans le tableau 1.4.

| **Algorithme** | **Moyenne du temps de calcul** |
|---|---|
| $y = A \times x$ | 0.0003155 s |

Table 1.4: Moyenne de temps de calcul de l'algorithme Produit Matrice creuse Vecteur.

# Appendix A

# Appendix Chapter

Le TP4 du calcul numérique est déposé dans le dépot git **"TPs-TDs-Calcul-Numerique-CHPS-M1-"**. Le code SSH de ce dépot est le suivant:

**git@github.com:Chaichas/TPs-TDs-Calcul-Numerique-CHPS-M1-.git**