TP. Résolution des systèmes linéaires. 1 ère partie

1. La méthode de Cramer

La méthode de Cramer est une méthode algébrique permettant de résoudre un système linéaire de n équations à n inconnues en utilisant les déterminants. Cette méthode est applicable uniquement lorsque la matrice des coefficients du système (le coefficient devant les inconnues) est inversible. Le système linéaire général peut être écrit sous la forme matricielle $Ax = B$, où A est la matrice des coefficients, x est le vecteur des inconnues, et B est le vecteur des termes constants.

Le système linéaire peut être écrit de manière générale comme suit :

$$Ax = B$$

La solution pour chaque inconnue x_i peut être trouvée en utilisant la formule de Cramer :

$$x_i = \frac{\det(A_i)}{\det(A)}$$

où A_i est la matrice obtenue en remplaçant la $i - ème$ colonne de A par le vecteur B, et $\det(A)$ est le déterminant de la matrice A.

Voici les étapes pour résoudre un système linéaire par la méthode de Cramer :

1. Calculez le déterminant de la matrice des coefficients A, c'est-à-dire $\det(A)$.
2. Pour chaque inconnue x_i, remplacez la $i - ème$ colonne de A par le vecteur des termes constants B pour obtenir la matrice A_i.
3. Calculez le déterminant de A_i, c'est-à-dire $\det(A_i)$.
4. La solution pour x_i est donnée par $x_i = \frac{\det(A_i)}{\det(A)}$.

Voici un exemple de MATLAB illustrant la méthode de Cramer :

```
% Définition de la matrice A et du vecteur b
A = [2, -1, 0; -1, 2, -1; 0, -1, 2];
b = [1; 0; 1];
% Vérification que A est carrée et que det(A) n'est pas nul
if det(A) == 0
disp('La matrice A doit être carrée et invertible.');
else
n = size(A,1); % Nombre d'inconnues
x = zeros(n,1); % Initialisation du vecteur solution
for i = 1:n
% Copie de A dans Ai
Ai = A;
% Remplacement de la i-ème colonne par le vecteur b
Ai(:,i) = b;
```

```
% Calcul de la i-ème variable
x(i) = det(Ai)/det(A);
end
% Affichage du vecteur solution
disp('La solution du système est :');
disp(x);
end
```

Autres exemples :

1. A = [2, -1, 1; -3, -1, 4; -1, 1, 3]; b = [8; 2; 6];
2. A = [1, 1, 1; 2, -1, 3; -1, 2, -1]; b = [6; 14; -2];
3. A = [2, -3, 1; 4, 1, 2; -2, 4, -3]; b = [7; 4; -10];

Assurez-vous que la matrice A est inversible avant d'appliquer la méthode de Cramer.

2. Système à matrices triangulaires

Un système d'équations linéaires dont les matrices des coefficients sont triangulaires est souvent plus facile à résoudre que les systèmes linéaires généraux. Il existe deux types de systèmes à matrices triangulaires : supérieure et inférieure. Un système triangulaire supérieur a des zéros en dessous de la diagonale, tandis qu'un système triangulaire inférieur a des zéros au-dessus de la diagonale.

2.1. Système triangulaire supérieur. Un système triangulaire supérieur peut être écrit sous la forme :

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n-1}x_{n-1} + a_{1n}x_n = b_1 \\ a_{22}x_2 + \cdots + a_{1n-1}x_{n-1} + a_{2n}x_n = b_2 \\ \vdots \\ a_{(n-1)(n-1)}x_{n-1} + a_{(n-1)n}x_n = b_{n-1} \\ a_{nn}x_n = b_n \end{cases}$$

La résolution d'un tel système peut être effectuée par substitution arrière, en partant de la dernière équation et en remontant vers la première.

2.2. Système triangulaire inférieure. Un système triangulaire inférieur peut être écrit sous la forme :

$$A = \begin{pmatrix} a_{11} & 0 & \cdots & & 0 \\ a_{21} & a_{22} & & & 0 \\ & & \ddots & & \vdots \\ \vdots & \vdots & & \ddots & 0 \\ a_{n1} & a_{n2} & \cdots & & a_{nn} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ \vdots \\ b_n \end{pmatrix}.$$

La résolution d'un tel système peut être effectuée par substitution avant, en partant de la première équation et en descendant vers la dernière.

2.3. Exemple en MATLAB. Voici un exemple de résolution d'un système linéaire avec **une matrice triangulaire supérieure** en utilisant MATLAB :

```
% Définition de la matrice des coefficients (triangulaire supérieure) et du vecteur des termes constants
A = [3, 2, -1; 0, 1, 2; 0, 0, 1];
b = [1; 2; 3];
% Nombre d'inconnues
n = size(A, 1);
% Initialisation du vecteur solution
x = zeros(n, 1);
% Substitution arrière
for i = n:-1:1
x(i) = (b(i) - A(i, i+1:n) * x(i+1:n)) / A(i, i);
end
% Affichage du vecteur solution
disp('La solution du système est :');
disp(x);
```

La résolution d'un système triangulaire inférieur se ferait de manière similaire, mais en utilisant la substitution avant (**méthode de descente**):

```
% Définition de la matrice des coefficients (triangulaire inférieure) et du vecteur des termes constants
A = [1, 0, 0; -2, 3, 0; 4, -5, 6];
b = [3; 2; 1];
% Nombre d'inconnues
n = size(A, 1);
% Initialisation du vecteur solution
x = zeros(n, 1);
% Substitution avant
for i = 1:n
x(i) = (b(i) - A(i, 1:i-1) * x(1:i-1)) / A(i, i);
end
% Affichage du vecteur solution
disp('La solution du système est :');
disp(x);
```

3. Méthode de Décomposition LU

La décomposition LU (Lower-Upper) est une méthode de factorisation d'une matrice A en deux matrices L (inférieure) et U (supérieure), telles que $A = LU$. Cette décomposition est souvent utilisée pour résoudre des systèmes linéaires de manière efficace.

La décomposition LU consiste à exprimer une matrice A sous la forme $A = LU$ où L est une matrice triangulaire inférieure avec des uns sur la diagonale, et U est une matrice triangulaire supérieure. La décomposition LU est particulièrement utile pour résoudre des systèmes linéaires $Ax = B$ de manière itérative.

Étapes de la Décomposition LU

1. Décomposition : Trouvez les matrices L et U telles que $A = LU$. Il existe plusieurs algorithmes pour accomplir cette décomposition, dont la méthode de Doolittle, de Crout, et de Cholesky (pour les matrices symétriques positives définies).

2. Résolution du système $Lz = b$: Une fois A décomposée en L et U, le système original $Ax = b$ est équivalent à $LUx = b$. Le premier pas consiste à résoudre $Lz = b$ pour z en utilisant la méthode de substitution avant, L étant une matrice triangulaire inférieure.

3. Résolution du système $Ux = z$: Ensuite, résolvez $Ux = z$ pour x en utilisant la méthode de substitution arrière, U étant une matrice triangulaire supérieure.

3.1. La factorisation de Doolittle. La factorisation de Doolittle est une méthode spécifique de décomposition LU où la matrice L (triangulaire inférieure) a tous ses éléments diagonaux égaux à 1. Voici un exemple de code MATLAB pour effectuer **la factorisation de Doolittle d'une matrice** A :

```
function [L, U] = doolittleFactorization(A)
% Vérification que A est carrée
[n, m] = size(A);
if n ~= m
error('La matrice A doit être carrée.');
end
% Initialisation des matrices L et U
L = eye(n); % L est initialisée à la matrice identité
U = zeros(n); % U est initialisée à une matrice de zéros
for i = 1:n
% Calcul des éléments de U (ligne i)
for j = i:n
sum = 0;
for k = 1:i-1
sum = sum + L(i,k) * U(k,j);
end
U(i,j) = A(i,j) - sum;
end
% Calcul des éléments de L (colonne i, en dessous de la diagonale)
for j = i+1:n
sum = 0;
for k = 1:i-1
sum = sum + L(j,k) * U(k,i);
end
L(j,i) = (A(j,i) - sum) / U(i,i);
end
end
end
```

3.2. Utilisation du code. Pour utiliser cette fonction, vous devez la sauvegarder dans un fichier nommé '**doolittleFactorization.m**'. Ensuite, vous pouvez l'appeler depuis la fenêtre de commande MATLAB ou dans un script en passant la matrice A comme argument.

Exemple :

```
1. A = [4, 3; 6, 3];
[L, U] = doolittleFactorization(A)
2. A = [2, -1, 0, 0; -1, 2, -1, 0; 0, -1, 2, -1; 0, 0, -1, 2];
```

```
[L, U] = doolittleFactorization(A)
```

3.3. Méthode de Doolittle pour résoudre un système linéaire. Après avoir obtenu L et U, le système original $Ax = b$ est résolu en deux étapes principales :

1. Résoudre $Lz = b$ pour y en utilisant la substitution avant, car L est une matrice triangulaire inférieure.
2. Résoudre $Ux = z$ pour x en utilisant la substitution arrière, car U est une matrice triangulaire supérieure.

Voici un exemple de code MATLAB qui illustre l'ensemble du processus, en utilisant une fonction fictive '**doolittleFactorization**' pour effectuer la factorisation de Doolittle, suivie de la résolution des systèmes $Lz = b$ et $Ux = z$.

```
function x = solveUsingDoolittle(A, b)
% Factorisation de Doolittle de A
[L, U] = doolittleFactorization(A); % Assurez-vous que cette fonction est définie
ou disponible
% Résolution de Lz = b pour z en utilisant la substitution avant
z = forwardSubstitution(L, b);
% Résolution de Ux = z pour x en utilisant la substitution arrière
x = backwardSubstitution(U, z);
end
function z = forwardSubstitution(L, b)
n = length(b);
z = zeros(n, 1);
for i = 1:n
z(i) = (b(i) - L(i, 1:i-1) * z(1:i-1)) / L(i, i);
end
end
function x = backwardSubstitution(U, z)
n = length(z);
x = zeros(n, 1);
for i = n:-1:1
x(i) = (z(i) - U(i, i+1:n) * x(i+1:n)) / U(i, i);
end
end
```

Ce code inclut deux fonctions supplémentaires, '**forwardSubstitution**' et '**backwardSubstitution**', pour résoudre respectivement $Lz = b$ et $Ux = z$. Ces fonctions implémentent les algorithmes de substitution avant et arrière en parcourant les équations de manière ordonnée pour isoler chaque variable.

```
A = [2, -1, 1; -3, -1, 4; -1, 1, 3];
B = [8; 2; 6];
[L, U] = doolittleFactorization(A)
```

3.4. La factorisation de Crout. La factorisation de Crout est une autre méthode de factorisation LU (Lower-Upper) d'une matrice, similaire à la factorisation de Doolittle. Cependant, dans la factorisation de Crout, la diagonale de la matrice supérieure U est composée de 1s.

Voici comment vous pouvez obtenir la factorisation de Crout d'une matrice $A = LU$ où L est une matrice inférieure et U est une matrice supérieure. Chaque

élément de L et U est obtenu en suivant ces étapes :

$$\begin{aligned} l_{ij} &= a_{ij} - \sum_{k=1}^{i-1} l_{ik}u_{kj}, \qquad \text{pour } j \leqslant i, \\ u_{ij} &= \frac{1}{l_{jj}}\left(a_{ij} - \sum_{k=1}^{i-1} l_{ik}u_{kj}\right), \qquad \text{pour } j > i. \end{aligned}$$

Voici un exemple de code MATLAB pour effectuer la factorisation de Crout d'une matrice A :

```
function [L, U] = croutFactorization(A)
n = size(A, 1);
L = zeros(n);
U = eye(n); % U commence avec des 1 sur la diagonale
for i = 1:n
for j = 1:i
L(i, j) = A(i, j) - L(i, 1:j-1) * U(1:j-1, j);
end
for j = i+1:n
U(i, j) = (A(i, j) - L(i, 1:i-1) * U(1:i-1, j)) / L(i, i);
end
end
end
```

3.5. Utilisation du code. Ce code définit une fonction '**croutFactorization**' qui prend une matrice carrée A et retourne deux matrices, L et U, résultant de la factorisation de Crout de A. Cette fonction peut être utilisée pour préparer la résolution de systèmes d'équations linéaires en décomposant la matrice des coefficients une seule fois, puis en appliquant des substitutions avant et arrière pour résoudre pour différents vecteurs de termes constants.

Application de la Factorisation de Crout

Nous allons maintenant utiliser la fonction '**croutFactorization**' avec cette matrice A.

D'abord, assurez-vous que la fonction '**croutFactorization**' est sauvegardée dans un fichier nommé '**croutFactorization.m**' dans votre répertoire de travail MATLAB. Ensuite, exécutez ce code dans MATLAB :

```
% Définition de la matrice A
A = [4, 12, -16; 12, 37, -43; -16, -43, 98];
% Appel de la fonction croutFactorization
[L, U] = croutFactorization(A)
% Affichage des résultats
disp('Matrice L (Triangulaire Inférieure) :');
disp(L);
disp('Matrice U (Triangulaire Supérieure) :');
disp(U);
```

3.6. Résultats Attendus. Après avoir exécuté le code, MATLAB affichera les matrices L et U obtenues par la factorisation de Crout. La matrice L sera une matrice triangulaire inférieure avec des éléments non unitaires sur sa diagonale, et la matrice U sera une matrice triangulaire supérieure avec des 1 sur sa diagonale.

3.7. Vérification de la Factorisation. Pour vérifier que la factorisation a été effectuée correctement, vous pouvez multiplier L et U et comparer le produit avec la matrice A originale :

```
% Vérification que LU est égal à A
A_reconstructed = L * U;
disp('Matrice A reconstruite à partir de LU :');
disp(A_reconstructed);
```

Si la factorisation a été réalisée correctement, '**A_reconstructed**' devrait être très proche de la matrice A originale, à l'intérieur des limites de précision numérique de MATLAB. Ce processus de vérification est un bon moyen de s'assurer de l'exactitude des calculs effectués lors de la factorisation de Crout.

4. Méthode de Cholesky

La méthode de Cholesky est une méthode de factorisation d'une matrice symétrique et définie positive A en un produit de deux matrices L et L^T, où L est une matrice triangulaire inférieure avec des éléments diagonaux strictement positifs. La factorisation de Cholesky est particulièrement utile pour résoudre des systèmes d'équations linéaires ou inverser des matrices symétriques définies positives de manière plus efficace que d'autres méthodes.

La factorisation de Cholesky s'exprime comme suit :

$$A = LL^T$$

où L est une matrice triangulaire inférieure, et L^T est la transposée de L.

4.1. Conditions d'Application. La méthode de Cholesky s'applique uniquement aux matrices qui sont :

1. **Symétriques:** $A = A^T$.
2. **Positives définies :** $x^T Ax > 0$ pour tout vecteur non nul x

4.2. Procédure de Décomposition de Cholesky. La décomposition de Cholesky calcule les éléments de L en suivant ces étapes :

Pour chaque ligne $i = 1, 2, \ldots, n$ de L,

- Calculez d'abord l'élément diagonal

$$l_{ii} = \sqrt{a_{ii} - \sum_{k=1}^{i-1} l_{ik}^2}.$$

- Ensuite, pour chaque élément sous-diagonal

$$l_{ji} = \frac{1}{l_{ii}} \left(a_{ji} - \sum_{k=1}^{i-1} l_{jk} u_{ik} \right)$$

4.3. Avantages de la Méthode de Cholesky.

- **Efficacité :** Moins de la moitié des opérations requises par la décomposition LU pour des matrices symétriques générales, car elle exploite la symétrie de la matrice.
- **Stabilité :** La décomposition est numériquement stable.

4.4. Exemple de Code MATLAB pour la Décomposition de Cholesky. MATLAB fournit une fonction intégrée chol pour effectuer la décomposition de Cholesky. Voici comment vous pouvez l'utiliser :

```
% Définition d'une matrice symétrique positive définie A
A = [4, 12, -16; 12, 37, -43; -16, -43, 98];
% Décomposition de Cholesky de A
L = chol(A, 'lower'); % 'lower' indique que MATLAB retourne la matrice
triangulaire inférieure L
% Affichage de L
disp('Matrice L (Triangulaire Inférieure) :');
disp(L);
% Vérification que A = LL^T
A_reconstructed = L * L';
disp('Matrice A reconstruite (LL^T) :');
disp(A_reconstructed);
```

Dans cet exemple, '**chol(A, 'lower')**' effectue la décomposition de Cholesky de la matrice A et retourne la matrice triangulaire inférieure L. La vérification que $A = LL^T$ sert à confirmer l'exactitude de la décomposition.

4.5. Méthode de Cholesky pour résoudre un système linéaire. Une fois la décomposition $A = LL^T$ obtenue, résoudre $Ax = b$ se fait en deux étapes :

1. Résolvez $Lz = b$ pour z en utilisant la substitution avant.
2. Résolvez $L^T x = z$ pour x en utilisant la substitution arrière.

La méthode de Cholesky offre une approche efficace et stable pour travailler avec des matrices spécifiques, réduisant le coût computationnel et améliorant la précision des solutions des systèmes d'équations linéaires.

```
function x = solveLLT(A, b)
% Vérifie que A est symétrique
if ~isequal(A, A')
error('A doit être une matrice symétrique.');
end
% Décomposition de Cholesky de A en L et L^T
% 'lower' spécifie que L est une matrice triangulaire inférieure
L = chol(A, 'lower');
% Résolution de Lz = b pour z en utilisant la substitution avant
z = forwardSubstitution(L, b);
% Résolution de L^Tx = z pour x en utilisant la substitution arrière
x = backSubstitution(L', z);
end
function x = forwardSubstitution(L, b)
n = length(b);
x = zeros(n, 1);
for i = 1:n
x(i) = (b(i) - L(i, 1:i-1) * x(1:i-1)) / L(i, i);
end
end
function x = backSubstitution(U, b)
n = length(b);
```

```
x = zeros(n, 1);
for i = n:-1:1
x(i) = (b(i) - U(i, i+1:n) * x(i+1:n)) / U(i, i);
end
end
```

Explications

Décomposition de Cholesky ('chol'): La fonction MATLAB **chol(A, 'lower')** décompose la matrice A en LL^T, où L est une matrice triangulaire inférieure. La décomposition de Cholesky nécessite que A soit symétrique et définie positive.

Substitution avant ('forwardSubstitution'): Cette fonction résout $Lz = b$ pour z en parcourant les équations une par une de haut en bas, en isolant l'inconnue à gauche de chaque équation.

Substitution arrière ('backSubstitution'): Cette fonction résout $Ux = z$ (où $U = L$ pour x en parcourant les équations une par une de bas en haut, en isolant l'inconnue à gauche de chaque équation.

Ces fonctions **'forwardSubstitution'** et **'backSubstitution'** sont des implémentations basiques des processus de substitution avant et arrière respectivement, conçues pour fonctionner avec des matrices triangulaires. La résolution de systèmes d'équations linéaires via la décomposition de Cholesky est efficace pour les matrices symétriques définies positives, et ce processus en deux étapes minimise le nombre d'opérations requises comparé à d'autres méthodes directes.

Vous pouvez utiliser ces fonctions en combinant la décomposition de Cholesky avec la résolution du système linéaire. Voici des exemples:

Exemple 1

Prenons la matrice A et le vecteur b suivants:

```
A = [4, 12, -16; 12, 37, -43; -16, -43, 98];
B = [8; 2; 6];
x = solveLLT(A, b);
disp(x);
```

Exemple 2

Prenons une autre matrice A et un vecteur b :

```
A = [1 -1 0; -1 2 -1; 0 -1 3];
b = [1; 0; 1];
x = solveLLT(A, b);
disp(x);
```

Exemple 3

Prenons une autre matrice A et un vecteur b :

```
A = [4 12 -16 0; 12 37 -43 0; -16 -43 98 0; 0 0 0 5];
b = [2; 3; -1; 1];
% Appel de la fonction solveLLT pour résoudre Ax = b
x = solveLLT(A, b);
% Affichage de la solution
disp('La solution x est :');
disp(x);
```