TP. Résolution des systèmes linéaire. $4^{ème}$ partie: Méthodes itératives

Abdelfatah Bouziani

1. Méthodes itératives

1.1. Méthode d'itération de Jacobi. La méthode d'itération de Jacobi est une procédure numérique utilisée pour résoudre des systèmes d'équations linéaires de la forme $Ax = b$, où A est une matrice carrée, x est le vecteur des inconnues, et b est le vecteur des termes constants.

Voici comment elle fonctionne :

Formulation itérative : L'équation $Ax = b$ peut être reformulée en isolant la matrice diagonale:

$$Dx = b + (L + U)\, x = b + Rx.$$

Ou encore,

$$x = D^{-1}\left[b + (L + U)\, x\right] = D^{-1}\left[b + Rx\right].$$

Cela donne lieu à la formule itérative suivante pour chaque composante i du vecteur x:

$$x_i^{(k+1)} = \frac{1}{a_{ii}}\left(b_i - \sum_{\substack{j=1 \\ j\neq i}}^{n} a_{ij} x_j^{(k)}\right), \quad i = 1, 2, \ldots, n \quad k = 0, 1, 2, \ldots$$

où $x^{(k)}$ est la valeur de x à l'itération k.

Initialisation et itération : Le vecteur x est initialisé, souvent à zéro ou à une autre estimation. Ensuite, on utilise la formule itérative pour calculer x à chaque étape jusqu'à ce qu'une certaine condition d'arrêt soit satisfaite (comme un nombre maximum d'itérations ou un seuil pour la norme de l'erreur).

Convergence : La convergence de la méthode de Jacobi est garantie si la matrice A est diagonalement dominante, ou symétrique et définie positive. Cependant, il peut y avoir convergence même si ces conditions ne sont pas satisfaites, bien que cela ne soit pas garanti.

La méthode de Jacobi est appréciée pour sa simplicité et son potentiel de parallélisation, mais elle peut être lente à converger pour certains systèmes. Dans la pratique, d'autres méthodes comme la méthode de Gauss-Seidel ou des méthodes basées sur la décomposition LU peuvent être préférées en raison de leur convergence plus rapide.

Voici un exemple de MATLAB illustrant **la méthode d'itération de Jacobi**

```
% Matrice des coefficients
A = [3, 1, -1;
1, 5, 2;
2, -1, -6];
% Vecteur de résultats
b = [2; 17; -18];
% Vecteur d'approximation initial
x_jacobi = zeros(size(b));
% Nombre maximum d'itérations
max_iter = 1000;
% Tolérance pour la convergence
tolerance = 10e-6;
% Afficher le vecteur initial
fprintf('Vecteur d"approximation initial :\n');
disp(x_jacobi);
% Méthode de Jacobi
for iter = 1:max_iter
x_jacobi_new = zeros(size(x_jacobi));
for i = 1:length(b)
x_jacobi_new(i) = (b(i) - A(i,:) * x_jacobi + A(i,i) * x_jacobi(i)) / A(i,i);
end
% Afficher l'itération actuelle
fprintf('Iteration %d :\n', iter);
disp(x_jacobi_new);
 % Vérifier la convergence
if norm(x_jacobi_new - x_jacobi, inf) < tolerance
fprintf('Convergence atteinte après %d itérations pour Jacobi.\n', iter);
break;
end
x_jacobi = x_jacobi_new;
end
% Afficher la solution finale
fprintf('La solution du système avec la méthode de Jacobi est :\n');
disp(x_jacobi);
```

2. Forme matricielle de la méthode Jacobi

La forme matricielle de la méthode de Jacobi permet une compréhension plus claire de son fonctionnement global, surtout lorsqu'il s'agit de systèmes d'équations linéaires de grande taille. Voici comment la méthode de Jacobi peut être représentée en utilisant la notation matricielle :

Formulation itérative en matricielle :

La réarrangement initial $Ax = b$ peut être réécrit en isolant la matrice diagonale D:

$$Dx = b + Rx.$$

En résolvant pour x, on obtient:

$$x = D^{-1}\left(b + Rx\right).$$

Cette équation peut être exprimée sous une forme itérative, où à chaque itération k, le nouveau vecteur x est calculé à partir de l'ancien vecteur x :

$$x^{(k+1)} = D^{-1}\left(b + Rx^{(k)}\right).$$

Initialisation et processus itératif :

On commence généralement avec une estimation initiale $x^{(0)}$, souvent le vecteur nul ou une autre valeur simple. Ensuite, on applique l'équation itérative pour mettre à jour x jusqu'à atteindre une convergence suffisante, c'est-à-dire que la différence entre deux itérations successives est inférieure à un seuil prédéfini, ou après un nombre fixé d'itérations.

Vectorisation et calculs :

La beauté de cette forme matricielle est qu'elle permet des calculs parallèles et vectorisés, ce qui est particulièrement avantageux pour les systèmes de grande taille traités sur des ordinateurs modernes.

En résumé, la méthode de Jacobi en forme matricielle montre clairement comment les éléments de la matrice sont utilisés pour mettre à jour de manière itérative le vecteur des inconnues x, en utilisant uniquement des opérations sur les matrices et les vecteurs. Cette approche rend la méthode particulièrement adaptée à l'implémentation sur des systèmes informatiques parallèles et à haute performance.

2.1. Exemple en MATLAB.

```
% Définition de la matrice des coefficients
A = [4 -2 1;
1 -4 1;
1 2 5];
% Définition du vecteur de résultats
b = [4; 0; 7];
% Vecteur d'approximation initial
x_jacobi = [0; 0; 0];
% Extraction des matrices D, L, et U
D = diag(diag(A));
L = tril(A, -1);
U = triu(A, 1);
% Nombre maximum d'itérations
max_iter = 1000;
% Tolérance pour la convergence
tolerance = 1e-6;
% Itération de la méthode de Jacobi
for iter = 1:max_iter
x_jacobi_new = D \ (b - (L + U) * x_jacobi);
% Vérifier la convergence
if norm(x_jacobi_new - x_jacobi, inf) < tolerance
fprintf('Convergence atteinte après %d itérations.\n', iter);
break;
end
x_jacobi = x_jacobi_new;
end
% Afficher la solution finale
disp('La solution est :');
```

```
disp(x_jacobi);
```

3. Méthode d'itération de Gauss – Seidel

La méthode d'itération de Gauss-Seidel est une technique utilisée pour résoudre des systèmes d'équations linéaires, similaire à la méthode de Jacobi, mais avec une approche légèrement différente qui peut conduire à une convergence plus rapide. Elle est particulièrement utile pour les matrices larges et creuses où la factorisation directe serait coûteuse en termes de temps et de mémoire.

3.1. Description de la méthode. Décomposition de la matrice A:

La matrice A est décomposée en une matrice diagonale D, une matrice triangulaire inférieure $-L$, et une matrice triangulaire supérieure $-U$. Ainsi, $A = -L + D - U$.

Formulation itérative:

En réarrangeant l'équation initiale $Ax = b$ pour isoler le terme x sur la gauche, on obtient:

$$(-L + D)\, x = b + Ux.$$

Ce qui conduit à la formule itérative suivante:

$$x = (-L + D)^{-1} (b + Ux).$$

Processus itératif:

On part d'une estimation initiale $x^{(0)}$. À chaque itération, on utilise les nouvelles valeurs de x dès qu'elles sont disponibles:

$$x_i^{(k+1)} = \frac{1}{a_{ii}} \left[b_i - \sum_{j=1}^{i-1} a_{ij} x_i^{(k+1)} - \sum_{j=i+1}^{n} a_{ij} x_i^{(k)} \right]$$

pour $i = 1, 2, \ldots, n$. Chaque composante est mise à jour séquentiellement en utilisant les valeurs les plus récentes de x.

Convergence:

La convergence de la méthode de Gauss-Seidel est généralement plus rapide que celle de Jacobi, particulièrement si la matrice A est symétrique et définie positive ou diagonalement dominante. Toutefois, pour certaines matrices, la convergence peut être lente ou la méthode peut ne pas converger du tout.

3.2. Avantages et inconvénients. Avantages:

Potentiellement plus rapide à converger que la méthode de Jacobi.

Peut être efficace pour les systèmes de grande taille et creux.

Inconvénients:

Peut être plus difficile à paralléliser efficacement en raison de la dépendance entre les calculs des composantes de x.

Ne garantit pas la convergence pour toutes les matrices.

La méthode de Gauss-Seidel est fréquemment utilisée en calcul scientifique et en ingénierie pour des problèmes où la convergence rapide est nécessaire et où la structure de la matrice est favorable à son application.

3.3. Exemple en MATLAB.

```
% Matrice des coefficients
A = [4, -2, 1;
1, -4, 1;
1, 2, 5];
% Vecteur de résultats
b = [4; 0; 7];
% Vecteur d'approximation initial
x = [0; 0; 0];
% Nombre maximum d'itérations
max_iter = 1000;
% Tolérance pour la convergence
tolerance = 1e-6;
% Itération de Gauss-Seidel
for i = 1:max_iter
x_old = x;
for j = 1:length(x)
x(j) = (b(j) - A(j,1:j-1)*x(1:j-1) - A(j,j+1:end)*x_old(j+1:end)) / A(j,j);
end
% Vérifier la convergence
if norm(x - x_old, inf) < tolerance
fprintf('Convergence atteinte après %d itérations.\n', i);
break;
end
end
% Afficher la solution
fprintf('La solution du système est :\n');
disp(x);
```

4. Forme matricielle de la méthode Gauss-Seidel

Processus itératif

Le processus commence avec une valeur initiale $x^{(0)}$. À chaque itération, les valeurs de x sont mises à jour séquentiellement en utilisant les nouvelles valeurs dès qu'elles sont disponibles. La formule itérative matricielle peut être visualisée comme suit pour une mise à jour complète de x :

$$x^{(k+1)} = (-L + D)^{-1} \left(b + U x^{(k)} \right),$$

où $x^{(k)}$ représente le vecteur de solutions à l'itération k.

Convergence

La convergence de la méthode de Gauss-Seidel dépend de la structure de la matrice A. Elle est généralement plus rapide que celle de Jacobi pour les matrices qui sont symétriques et définies positives ou diagonalement dominantes. Cependant, comme dans le cas de Jacobi, il n'y a pas de garantie de convergence pour toutes les matrices.

Avantages

- Utilisation efficace des calculs intermédiaires pour une convergence plus rapide.
- Peut être particulièrement efficace pour des matrices bien conditionnées.

Inconvénients

- Difficulté de parallélisation due à la dépendance des calculs de chaque itération sur les résultats précédents.
- Convergence non garantie pour certaines classes de matrices.

En résumé, la méthode de Gauss-Seidel en forme matricielle permet une approche systématique et séquentielle, utilisant les informations les plus récentes disponibles pour accélérer la convergence vers la solution du système linéaire.

4.1. Exemple en MATLAB.

```
% Définition de la matrice des coefficients
A = [4 -2 1;
1 -4 1;
1 2 5];
% Définition du vecteur de résultats
b = [4; 0; 7];
% Vecteur d'approximation initial
x = zeros(3, 1);
% Extraction des matrices D, L, et U
D = diag(diag(A));
L = tril(A, -1);
U = triu(A, 1);
% Inverse de la matrice (D+L)
DL_inv = inv(D + L);
% Nombre maximum d'itérations
max_iter = 1000;
% Tolérance pour la convergence
tolerance = 1e-6;
% Itération de la méthode de Gauss-Seidel
for iter = 1:max_iter
x_new = DL_inv * (b - U * x);
 % Vérifier la convergence
if norm(x_new - x, inf) < tolerance
fprintf('Convergence atteinte après %d itérations.\n', iter);
break;
end
x = x_new;
end
% Afficher la solution finale
disp('La solution est :');
disp(x);
```