

High Performance Computing

Project : Jacobi Method in Parallel for Equation resolution

Plessia Stanislas

Mars 2018

Table des matières

1	Introduction	3
2	Methode de Jacobi	3
2.1	Problème	3
2.2	Méthode de résolution	3
2.3	Preuve et Convergence	4
2.4	Vecteur erreur et résidu	5
3	Implémentation	6
3.1	Architecture	6

1 Introduction

2 Methode de Jacobi

2.1 Problème

Soit $n \in \mathbb{N}$, $A = (a_{i,j})_{i,j \in \llbracket 1,n \rrbracket^2}$ matrice carrée de taille n , $b = (b_i)_{i \in \llbracket 1,n \rrbracket}$ vecteur de taille n .
On cherche alors le vecteur $x = (x_i)_{i \in \llbracket 1,n \rrbracket}$ tel que :

$$Ax = b \quad (1)$$

2.2 Méthode de résolution

Afin de résoudre ce problème, on va utiliser une méthode itérative appelée Méthode de Jacobi.

Tout d'abord, on va séparer la matrice A en deux sous matrices D et R de la manière suivante :

$$A = \begin{pmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n,1} & a_{n,2} & \cdots & a_{n,n} \end{pmatrix} = \begin{pmatrix} a_{1,1} & 0 & \cdots & 0 \\ 0 & a_{2,2} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & a_{n,n} \end{pmatrix} + \begin{pmatrix} 0 & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & 0 & \cdots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n,1} & a_{n,2} & \cdots & 0 \end{pmatrix}$$

Notons D la matrice diagonale, et R la matrice du reste. Comme la matrice D est diagonale (qu'on suppose à coefficients non nuls dans notre problème), D est trivialement inversible d'inverse :

$$D^{-1} = \begin{pmatrix} \frac{1}{a_{1,1}} & 0 & \cdots & 0 \\ 0 & \frac{1}{a_{2,2}} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \frac{1}{a_{n,n}} \end{pmatrix}$$

On peut donc réécrire la formule (1) de la manière qui suit :

$$(D + R)x = b \quad (2)$$

$$\Leftrightarrow Dx = b - Rx \quad (3)$$

$$\Leftrightarrow x = D^{-1}(b - Rx) \quad (4)$$

La Méthode de Jacobi est alors une méthode itérative qui va chercher à trouver un point fixe à cet équation. On peut alors définir la suite $x^{(k)}$, $k \in \mathbb{N}$ telle que :

$$\begin{cases} x^{(0)} = \vec{0} \\ x^{(k+1)} = D^{-1}(b - Rx^{(k)}) \end{cases}$$

L'avantage flagrant de cette méthode de calcul, est que les éléments n'ont aucune dépendance verticale. Elle est donc très simple à paralléliser.

En effet, en écrivant la formule de récurrence pour un élément du vecteur $x^{(k+1)}$, on obtient :

$$\forall k \in \mathbb{N}, \forall i \in \llbracket 1, n \rrbracket, \quad x_i^{k+1} = \frac{1}{a_{i,i}} (b_i - \sum_{i \neq j} a_{i,j} \times x_j^{(k)}) \quad (5)$$

La seule contrainte devient alors que chaque processeur doit connaître toutes les composantes de $x^{(k)}$ afin de pouvoir itérer, et il faut donc les communiquer, ce que nous ferons en utilisant la bibliothèque MPI.

2.3 Preuve et Convergence

D'après le théorème de la méthode du point fixe, on sait que la suite $x_{n+1} = f(x_n)$ converge si la fonction f est contractante aka k -lipshitzienne avec $k < 1$

La matrice $D^{-1}R$ étant une application linéaire, trouvons une condition nécessaire et suffisante pour qu'elle soit contractante

Soit a, b deux vecteurs dans \mathbb{R}^n , $k \in [0, 1[$ et notons $C = D^{-1}R$,

$$|Ca - Cb| \leq k|a - b| \quad (6)$$

$$\Leftrightarrow \|C - kI\| \leq 0 \quad (7)$$

Cela revient à dire que le rayon spectral de la matrice C noté $\rho(C)$ doit être strictement inférieur à 1

On peut donc affirmer que :

$$\lim_{k \rightarrow \infty} x^{(k)} = x \Leftrightarrow \rho(C) = \rho(D^{-1}R) < 1$$

Cherchons une condition simple suffisante pour affirmer que le rayon spectral de la matrice C soit inférieur à 1.

Soit λ valeur propre de C ie $\forall y \in \mathbb{R}^n, Cy = \lambda y$

On a alors, $\forall (i, j) \in \llbracket 1, n \rrbracket^2$

$$\|\lambda y\|_\infty = \left| \sum_{j=1}^n c_{i,j} y_j \right| \quad (8)$$

$$\Leftrightarrow \lambda \|y\|_\infty = \left| \sum_{j \neq i} \frac{a_{i,j}}{a_{i,i}} y_j \right| \quad (9)$$

$$\Leftrightarrow \lambda \|y\|_\infty = \left| \frac{1}{a_{i,i}} \right| \times \left| \sum_{j \neq i} a_{i,j} y_j \right| \quad (10)$$

$$\Leftrightarrow \lambda \|y\|_\infty \leq \left| \frac{1}{a_{i,i}} \right| \times \sum_{j \neq i} |a_{i,j} y_j| \quad (11)$$

$$\Leftrightarrow \lambda \|y\|_\infty \leq \left| \frac{1}{a_{i,i}} \right| \times \sum_{j \neq i} |a_{i,j}| \times \|y\|_\infty \quad (12)$$

$$\Leftrightarrow \lambda \leq \left| \frac{1}{a_{i,i}} \right| \times \sum_{j \neq i} |a_{i,j}| \quad (13)$$

Une condition suffisante pour que $\lambda < 1$ serait alors :

$$\forall (i, j) \in \llbracket 1, n \rrbracket^2 \quad |a_{i,i}| > \sum_{i \neq j} |a_{i,j}|$$

ie A est à diagonale strictement dominante

2.4 Vecteur erreur et résidu

Afin de mesurer la convergence de la méthode de Jacobi (et d'avoir une condition d'arrêt), on définit le vecteur erreur suivant :

$$e^{(k+1)} = x^{(k+2)} - x^{(k+1)} = C(x^{(k+1)} - x^{(k)}) = Ce^{(k)}$$

On a donc $e^{(k)} = C^k e^{(0)}$

e est l'erreur relative de la méthode, mais comme on a pas connaissance du véritable vecteur x , on l'utilise comme référence pour le test d'arrêt.

Le test d'arrêt le plus courant de la méthode Jacobi est sur l'erreur relative du vecteur résidu :

$$\epsilon > \frac{\|r^{(k)}\|}{\|b\|} = \frac{\|De^{(k)}\|}{\|b\|}$$

3 Implémentation

3.1 Architecture

Le projet est divisé en plusieurs sous-dossiers

- **/lib** qui contient des librairies pour manipuler les matrices, les vecteurs ainsi que les opérations basiques de manipulation de ces deux structures de données.
- **/src** qui contient le fichier *main* qui fait tourner le programme de l'algorithme de Jacobi.
- **/data** qui contient les fichiers des matrices et des vecteurs ainsi que les métadonnées contenant la taille des matrices considérées.
- **/bin** qui contient les binaires *main* et *test* afin de faire tourner le programme ou de tester les librairies
- **/test** qui contient le code source du binaire de test dont l'objectif est de tester les fonctions des librairies.

Le projet est de plus constitué d'un Makefile qui contient les procédures suivantes :

- *make all* (ou simplement *make*) pour compiler les librairies, main et les tests
- *make test* pour compiler les tests
- *make clean* pour supprimer les fichiers objets