Sorbonne Université

UE 4IN910 - « Algorithmique numérique »
Master Informatique - M1 SFPN

Rapport de TME
TME n°4 - Méthodes Itératives pour la résolution de systèmes linéaires

Calle Viera Andersson

Enseignant
GRAILLAT Stef

Table des matières

Introduction

Dans ce T.M.E nous allons voir comment résoudre des systèmes linéaires par des méthodes itératives. Pour cela nous allons implémenter plusieurs algorithmes et comparer leurs performances.

Le code est réalisé en MatLab et est fournit avec le rapport, ci-dessous se trouvent des exemples d'utilisation des fonctions pour $N = 100000$(premier argument).
Tous les fichiers portent de le nom de l'algorithme utilisé dedans et se lancent à peu près de la même manière pour une matrice A, des vecteurs y et $x0$ un nombre d'itérations maximum $maxit$ et un seuil d'erreur eps. Choissison dans la suite $maxit = 1000$ et $eps = 1e - 4$ on a donc pour la méthode de Jacobi, de Gausse Seidel et du gradient conjugué :

```
[x_jac, X_jac, E_jac, iter_jac] = Jacobi(A, y, x0, maxit, eps);
```

Pour SOR il suffit juste de rajouter le paramètre w.

```
[x_sor, X_sor, E_sor, iter_sor] = Sor(A, y, x0, 1.8, Nmax, eps);
```

Si on veut afficher l'évolution des valeurs estimées selon le choix du nombre d'itérations maximum on peut lancer le scripts **Erreur.m** avec comme paramètres le seuil eps, la taille de la matrice de test N et une option opt qui permet de choisir entre la matrice de poisson et binomiale.

1 Exercice 5 (Implémentation des méthodes) :

Dans toute la suite on poser A une matrice carré d'ordre n et x, $b \in \mathbb{R}^n$.
On va aussi noter M et N deux matrices telles que $A = M - N$, on peut ainisi écrire les itérations des méthodes itératives sous la forme :

$$Mx_{k+1} = Nx_k + b \iff x_{k+1} = M^{-1}Nx_k + M^{-1}b$$

Nous tirons partie de cette forme et de l'implémentation rapide et efficace de l'inversion et de la multiplication en **MATLAB** pour coder les algorithmes sous leur forme matricielle sans passer par des boucles for.

1.1 Méthode de Jacobi :

Principe :
Ici on va utiliser la décomposition $A = D - U - L$ où D est la matrice diagonale de A, $-U$ la matrice strictement trinagulaire inférieure de A et $-F$ la matrice strictement triangulaire supérieure de A.
On va donc poser $M = D$ et $N = L + U$, ce qui nous donne la relation matricielle :

$$Dx_{k+1} = (L + U)x_k + b \iff x_{k+1} = D^{-1}((L + U)x_k + b)$$

1.2 Méthode de Gauss Seidel :

Principe :
Cette fois-ci la décomposition utilisée est la même que pour la méthode de Jacobi mais on va poser $M = D - L$ et $N = U$, ce qui nous donne la relation :

$$(D - L)x_{k+1} = Ux_k + b \iff x_{k+1} = (D - L)^{-1}(Ux_k + b)$$

1.3 Méthodes de ralaxation SOR :

Principe :
Pour cette méthode il faut choisir un paramètre $\omega \in]1,\ 2[$ qui dépend de la matrice A et qui va influer sur la vitesse de convergence en plus de la décomposition $A = D - U - L$, ensuite on pose :
$M = \dfrac{D}{\omega} - L$ et $N = \dfrac{D}{\omega} - L$,et on a donc la relation :

$$(D - \omega L)x_{k+1} = (\omega U + (1 - \omega)D)x_k + \omega b \iff x_{k+1} = (D - \omega L)^{-1}((\omega U + (1 - \omega)D)x_k + \omega b)$$

1.4 Méthode du gradient conjugué :

Principe :
C'est la seul méthode du devoir qui ne repose pas pas sur la décomposition de A mais sur une méthode du gradient en utilisant des directions A-conjuguées.

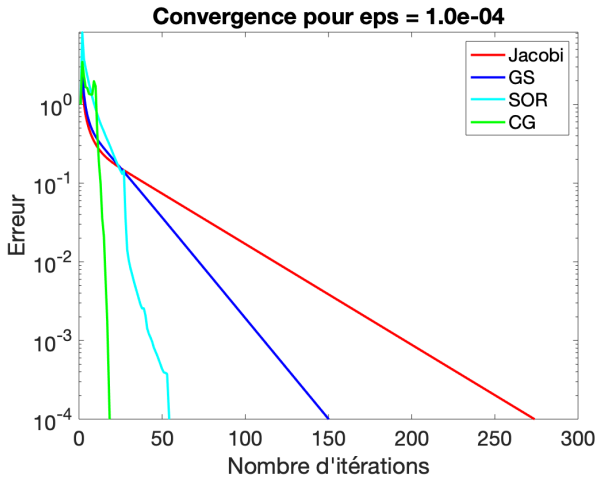
1.5 Comparaison des algorithmes :

Pour la comparaison des algorithmes on va définir le seuil $\varepsilon = 10^{-4}$ ainqi que le nombre max d'itérations $N_{max} = 1000$. Sauf mention du contraire n va utiliser y le vecteur rempli de 0 et x_0 le vecteur rempli de 1.

La première matrice de test une matrice de poisson de taille $12^2 \times 12^2$ définit comme suit :

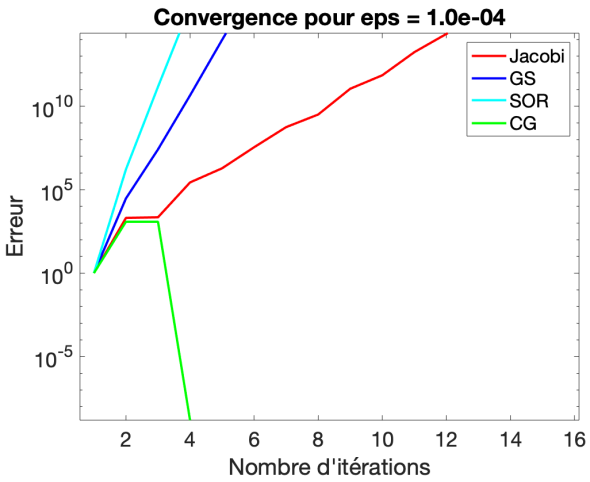
$$A = \begin{bmatrix} D & -I & 0 & 0 & 0 & \dots & 0 \\ -I & D & -I & 0 & 0 & \dots & 0 \\ 0 & -I & D & -I & 0 & \dots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & \dots & 0 & -I & D & -I & 0 \\ 0 & \dots & \dots & 0 & -I & D & -I \\ 0 & \dots & \dots & \dots & 0 & -I & D \end{bmatrix}, \qquad D = \begin{bmatrix} 4 & -1 & 0 & 0 & 0 & \dots & 0 \\ -1 & 4 & -1 & 0 & 0 & \dots & 0 \\ 0 & -1 & 4 & -1 & 0 & \dots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & \dots & 0 & -1 & 4 & -1 & 0 \\ 0 & \dots & \dots & 0 & -1 & 4 & -1 \\ 0 & \dots & \dots & \dots & 0 & -1 & 4 \end{bmatrix},$$

Pour la méthode SOR on choisit ici de faire la comparaison avec $\omega = 1.8$ et on obtient les résultats suivants :



Résultat de notre `Erreur.m` en échelle semi-logarithmique

On peut donc remarquer que même si tout les algorithmes ont convergés avant le nombre maximum d'itérations il y a une grande différence entre la méthode du Gradient Conjugué (18 itérations) et celle de Jacobi (289 itérations). On peut donc remarquer que la plus efficace semble être celle reposant sur la méthode du gradient même si la méthode SOR s'en rapproche avec un nombre d'itérations de 52.
La deuxième matrice de test est une matrice binomiale qui est un multiple d'une matrice d'involution et telle que $A^2 = 2^{n-1}I$. ON utilise ici $y =$ `randn(12, 1)`.



Résultat de notre `Erreur.m` en échelle semi-logarithmique

Ce résultat me fait me poser des questions soit sur la matrice utilisée soit sur mes fonctions. En effet il semble que les trois premières méthodes divergent tandis que seul la méthode du gradient conjugué semble converger au bout de 4 itérations.
Pour conclure on peut dire que la méthode du gradient conjugué est la méthode la plus efficace en terme de nombre d'itérations et elle efficace pour un plus grand nombre de matrices. C'est donc celle à préférer en terme d'implémentation.