Maîtrise de Mathématiques Fondamentales et Applications
Grands Systèmes Linéaires
feuille de TP 5

8 Octobre 2009

Exercice - 1 *Convergence des méthodes Jacobi, Gauss-Seidel et SOR dans* **un** *cas non symétrique*

On rappelle que les méthodes itératives considérées ici sont définies par :
$\|x_0$ étant un vecteur initial donné, $Mx_{k+1} = Nx_k + b$, où $A = M - N$.
Elles sont alors entièrement définies par la donnée de M car $N = M - A$, ce qui conduit à l'itération $Mx_{k+1} = Mx_k + (b - Ax_k)$. La table 1 définit les différentes méthodes ; On y a posé

- D la matrice diagonale de A : $D_{ii} = A_{ii}, D_{ij} = 0,$ si $i \neq j$,
- E la matrice triangulaire inférieure stricte de $-A$: $E_{ij} = -A_{ij}$ si $i < j, E_{ij} = 0$ si $i \geq j$.

Méthode	matrice M
Jacobi	D
Gauss-Seidel	$D - E$
Relaxation (SOR)	$D/\omega - E$

TAB. 1 –

1- Rappeler les conditions nécessaires de convergence de ces méthodes.

Montrer que si la matrice A est à diagonale strictement dominante ($|a_{ii}| > \sum_{\substack{j=1 \\ j\neq i}}^{n} |a_{ij}|$), alors cette condition est satisfaite pour les méthodes Jacobi, Gauss-Seidel, et pour la méthode de relaxation (lorsque $0 < \omega \leq 1$) même si la matrice A n'est pas hermitienne.

(*On essaiera d'évaluer* $\|M^{-1}N\|_\infty$*, comme étant* $\sup_{\substack{x\neq 0 \\ My=Nx}} \frac{\|y\|_\infty}{\|x\|_\infty}$)

2- Sans effectuer de calcul, justifier que la méthode de relaxation avec un paramètre ω optimal converge toujours plus vite que la méthode de Gauss-Seidel.

3- Montrer sur un cas particulier que si la méthode de Jacobi et de Gauss-Seidel convergent simultanément, alors la méthode de Gauss-Seidel converge toujours plus vite que la méthode de Jacobi.

4- Justifier en se basant sur le cas particlier de la matrice du Laplacien en 1D, pourquoi pour une matrice tridiagonale hermitienne, il vaut mieux sur-relaxer ($\omega > 1$) que sous-relaxer ($\omega < 1$) la relaxation.

On rappelle que pour une matrice tridiagonale, hermitienne, définie positive, le paramètre optimal $\omega_{opt} = \frac{2}{1+\sqrt{1-\rho(\mathcal{J})^2}}$*, où* $\mathcal{J}$ *est la matrice d'itération de la méthode itérative de Jacobi.*

Exercice - 2 *Programmation des méthodes Jacobi, Gauss-Seidel et SOR*

Cet exercice a pour but de programmer et de comparer les méthodes Jacobi, Gauss-Seidel et SOR. Toutes les fonctions auront pour arguments principaux :

- `A` la matrice du système et `b` le vecteur second membre,
- `x0` est le vecteur initial, `tol` la valeur ε du test d'arrêt et `iterMax` le nombre maximal d'itérations,

et fourniront en retour

- `x` la solution approchée,
- `res` un vecteur stockant la valeur de la norme du résidu à chaque itération.

La longeur de `res` est donc égale au nombre d'itérations effectuées. Il n'y a convergence que si `length(res) < itermax` ou si `length(res) == itermax` et `res(length(res)) <= tol`.

1- Programmation
a- Ecrire une fonction MATLAB de prototype `[x,res] = Jacobi(A,b,x0, tol,iterMax)`, qui résoud le système linéaire $Ax = b$ selon la méthode Jacobi.
b- Ecrire une fonction MATLAB de prototype
`[x,res] = GaussSeidel(A,b,x0, tol,iterMax)` qui résoud le système $Ax = b$ par une méthode de Gauss-Seidel.
c- Ecrire une fonction MATLAB de prototype `[x,res] = sor(A,b,x0, tol,iterMax, w)` qui résoud le système $Ax = b$ par une méthode de relaxation. Ici, `w` désigne le paramètre de relaxation.

2- Comparaison
a- Ecrire une fonction `[A,b] = laplace2Df(nx,ny)` qui retourne la matrice A et le second membre b du problème du laplacien sur le domaine $\Omega =]0,1[\times]0,1[$ avec un second membre $f = 1$ et une condition de Dirichlet homogène sur toute la frontière du domaine (*on pourra utiliser les fonctions* `laplace2D` *et* `Smlaplace2D` *du TP 4 ; il suffit pour cela d'ajouter le chemin vers le repertoire du TP 4 via la commande* `addpath`*).*

b- Pour comparer ces méthodes, on prend `tol = 1e-6`, le vecteur initial `x0` nul, `iterMax=1E4` et pour la méthode de relaxation, on prend `w = 1.6`.
Générer la matrice et le second membre du laplacien `[A,b] = laplace2Df(15,10)` et représenter sur un même graphique la courbe du logarithme des normes des résidus retournés par les méthodes Jacobi, Gauss-Seidel et SOR, en fonction du nombre d'itérations. Commenter les observations.
(*On pourra utiliser la commande* `semilogy`).

Exercice - 3 *Quelques formats de stockage de matrices pour les méthodes itératives*

Les méthodes itératives ne nécessitent que les produits matrice-vecteur et la résolution des systèmes dont les matrices sont de formes simples (diagonales ou triangulaires). On peut alors stocker sous forme condensée (sparse) les matrices utilisées.
1- Stockage par coordonnées (COO) ou stockage IJV.
Ici on ne stocke que les éléments non nuls de la matrice dans un vecteur `VA`. Pour chaque élément `VA(k)`, son indice de ligne est donné par `IA(k)` et son indice de colonne est donné par `JA(k)`. Les vecteurs `IA`, `JA` et `VA` sont donc de même taille.

a- Ecrire une fonction MATLAB `function [IA,JA,VA] = stockCOO(A)` qui réalise le stockage par coordonnées d'une matrice pleine A.
Pour vérifier votre programmation, écrire aussi une fonction `function [A] = COO2full(IA,JA,VA)` (*lire COO to full*) qui effectue la conversion réciproque.

b- Ecrire une fonction MATLAB `function [x] = descenteCOO (IA,JA,VA,b)` qui résoud le système $Ax = b$ où A est une matrice triangulaire inférieure stockée au format coordonnées.
Ecrire une fonction MATLAB `function [ax] = amultCOO (IA,JA,VA,x)` qui effectue le produit matrice vecteur $ax = Ax$ où A est une matrice stockée sous format coordonnées.

2- Stockage CSR (en anglais Compressed Sparse Row) ou stockage Morse.
Dans ce format on diminue la taille du vecteur `IA` du format précédent. Pour cela, `VA` est créé en parcourant la matrice ligne par ligne de la première à la dernière en stockant au même moment les indices de colonne des éléments de `VA` dans `JA`. Le vecteur `IA` est ici de taille `n+1` et est défini de la manière suivante : `IA(i)` est la position dans `VA` du premier élément non nul de la ligne `i`. Une autre interprétation qui justifie la taille `n+1` de `IA` est la suivante : On a `IA(1)=1` et `IA(i+1) - IA(i)` est le nombre d'éléments non nuls de la ligne `i` ; ainsi `IA(n+1) = nnz+1`, où `nnz` est le nombre d'éléments non nuls de la matrice `A`, c'est à dire la taille de `VA`.

a- Ecrire une fonction MATLAB `function [IA,JA,VA] = stockMorse(A)` qui réalise le stockage Morse d'une matrice pleine A. Pour vérifier votre programmation, écrire aussi une fonction `function [A] = Morse2full(IA,JA,VA)` qui effectue la conversion réciproque.

b- Ecrire une fonction MATLAB `function [x] = descenteMorse (IA,JA,VA,b)` qui résoud le système $Ax = b$ où A est une matrice triangulaire inférieure stockée au format CSR.
Ecrire une fonction MATLAB `function [ax] = amultMorse (IA,JA,VA,x)` qui effectue le produit matrice vecteur $ax = Ax$ où A est une matrice stockée au format CSR.