TP8 : Quelques méthodes de résolution de systèmes linéaires

Ce TP porte sur les méthodes de résolution des systèmes linéaires. Les matrices utilisées seront celles obtenues par discrétisation par différences finis du Laplacien (les matrices obtenues par une discrétisation par éléments finis sont aussi valables) ou des matrices générées aléatoirement, satisfaisant certaines propriétés particulières (symétrie, définie positive,etc.).

On joint à cet effet des scripts MATLAB *, pour la discrétisation par différences finies du Laplacien en 1D et 2D, et pour la génération aléatoire des matrices de propriétés particulières.*

Cette fiche est constituée de 5 parties dont la partie 5 est un élément de réponse à la partie 4.

PARTIE - 1 Méthodes directes

Exercice - 1 *Méthode de résolution par la factorisation LU*

Algorithme de résolution de systèmes triangulaires inférieures (à gauche) et supérieures (à droite)

```
-----------------------------------------|-----------------------------------------
     ALGORITHME DE DESCENTE              |        ALGORITHME DE REMONTEE
-----------------------------------------|-----------------------------------------
Données: A,b. Résultat:x solution de A x= b| Données: A,b. Résultat:x solution de A x=b
-----------------------------------------|-----------------------------------------
pour i= 1 à n                            | pour i= n à 1
     s= 0                                |      s = 0
     pour j = 1 à i-1                    |        pour j = i+1 à n
       s = s + A(i,j) * x(j)             |             s = s + A(i,j) * x(j)
     fin j                               |        fin
  x(i) = (b(i) - s) /A(i,i)              |    x(i) = (b(i) - s)/A(i,i)
fin                                      |fin
```

Q-1 : Écrire une une fonction MATLAB de prototype `function [x] = Descente (A, b)`, qui résout un système linéaire triangulaire inférieure $Ax = b$.
Tester la fonction en générant aléatoirement des matrices triangulaires inférieures inversibles (voir ci-dessus), de tailles n et en effectuant : `b = rand(n,1), norm(b - Descente(A,A*b))`.

Q-2 : Même question pour les matrices triangulaires supérieures inversibles. La fonction aura pour prototype, `function [x] = Remontee (A, b)`.

Q-3 : Écrire une fonction MATLAB de prototype `function [L,U] = decompLU (A)` qui effectue la décomposition LU d'une matrice carrée.

Algorithme de décomposition A = LU

```
1  Donnée: A.                                |
2  Résultat: U triangulaire supérieure       |     Explications des notations et indications
3  L triangulaire inférieure (diagonale unité)|
4  ------------------------------------------| ---------------------------------------------
5  pour k = 1 à n                            |% : introduit un commentaire
6    pour i = k+1 à n                        |
7            A(i,k)= A(i,k)/ A(k,k)          |
8            pour j = k+1 à n                |
9               A(i,j)=A(i,j)-A(i,k)*A(k,j)  |
10           fin                             |
11   fin                                     |
12 fin                                       | Commande MATLAB pour récupérer L et U
13 Récupérer L et U  à partir de A           | U=triu(A); L = A - U + diag(ones(n,1));
```

Q-3-1 : Évaluer cette fonction sur la matrice $A = \begin{bmatrix} 2 & 4 & -4 & 1 \\ 3 & 6 & 1 & -2 \\ -1 & 1 & 2 & 3 \\ 1 & 1 & 4 & 1 \end{bmatrix}$.
Conclure en calculant tous les mineurs principaux d'ordre $k, k = 1 \dots 3$ de A.

Q-3-2 : On considère la matrice de permutation $P = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$..
Vérifier que le produit PA admet une factorisation LU, en évaluant `[L,U] = decompLU(PA)`.
Expliquer alors comment résoudre un système linéaire dont la matrice est la matrice A ci-dessus.

Q-4 : La question précédente met en évidence la nécessité de pivot dans la factorisation LU de certaines matrices.

Q-4-1 : Montrer que pour toute matrice inversible $A \in \mathcal{M}_m(\mathbb{R})$, il existe une matrice de permutation P telle que la matrice PA admette une factorisation LU. Comparée à la factorisation LU, on a allégé les hypothèses sur la matrice A pour que la factorisation $PA = LU$ soit possible. Lesquelles ?

Q-4-2 : Écrire une fonction MATLAB de prototype `[L,U,P] = decompLUP(A)`, qui effectue une factorisation LU d'une matrice régulière A selon une méthode de Gauss avec pivot partiel, c'est à dire avec une permutation de lignes. (*L'algorithme est donné ci-dessous*).

Algorithme de décomposition PA =LU

```
Donnée: A. Résultats: A contenant L,et U   |     Explications des notations et indications
et P vecteur de permutation des lignes     |
-------------------------------------------| ------------------------------------------------
pour i =1 à n P(i) = i fin                 |% : introduit un commentaire
pour k = 1 à n                             |ipivot est telle que |A(ipivot,k)|=max|A(i,k)| k<= i <= n
  recherche la ligne ipivot du pivot:      | commande MATLAB de recherche du pivot :
 permuter les lignes P(ipivot) et P(k)     |[vpivot,ipivot]=max(abs(A(k:n,k))); ipivot = k - 1 + ipivot;
pour i = k+1 à n                           | commande MATLAB pour permuter les lignes P(ipivot) et P(k) :
           A(i,k)= A(i,k)/AA(k,k)          |ip=P(ipivot); P(ipivot)=P(k);P(k)= ip;
          pour j = k+1 à n                 |v=A(k,:); A(k,:) = A(ipivot,:); A(ipivot,:)= v;
          A(i,j) = A(i,j)-A(i,k) * A(k,j)  |
          fin                              |
    fin                                    |
```

Exercice - 2 *Complexité de la méthode de décomposition LU*

Q-1 : Complexité de la factorisation.

Q-1-1 : Modifier la fonction `[L,U] = decompLU(A)` de l'exercice précédent, en une fonction `[L,U,nop] = decompLU_nop(A)` qui retourne en plus le nombre de multiplications et de divisions effectuées pendant la factorisation LU.
(*On Pourra initialiser nop à* 0 *en début de fonction et l'incrémenter, dans les boucles de la factorisation LU, à chaque fois qu'on rencontre une multiplication ou une division (algo-lignes 7 et 9)*).

Q-1-2 : Justifier l'existence de la factorisation LU d'une matrice symétrique définie positive. Générer des matrices symétriques définies positives de tailles $n = 1, \cdots 30$. Relever $Nop(n)$ fourni par la factorisation `decompLU_nop(A)` et représenter sur un même graphique la courbe $n \longmapsto Nop(n)$ et la courbe $n \longmapsto n^3/3$. Qu'observe-t-on ?
(*On pourra utiliser le script* MATLAB *ci-dessous qui réalise cette expérience*).

fichier nopLU.m de test de la complexité de la factorisation LU

```
x=[]; ni=[]; Nop=[];ind=0;
for i =5:20
```

```
  ind=ind+1; x(ind)= i;
  A=MatSdp(i);            % voir TP1
  [L,U,Nop(ind)] = decompLU_nop(A);
  ni(ind) = (i^3)/3. ;
end
plot(x,Nop,'*',x,ni,'-'); legend('Nop LU','n^3 / 3');
```

Q-1-3 : Reprendre la même expérience en ne représentant que la courbe $n \longmapsto Nop(n)$ en échelle logarithmique. Déterminer la pente de la droite la plus proche de cette courbe et conclure.
(*On utilisera la commande* MATLAB `loglog` *pour la représentation en échelle logarithmique).*

Exercice - 3 *Phénomène de remplissage*

Q-1 : Modifier la factorisation LU de sorte qu'elle retourne les facteurs L, U dans une matrice de la taille de A ; U étant stockée dans la partie supérieure et L (sauf sa diagonale) dans la partie inférieure. Le prototype de la fonction sera `function [A]=decompLU_eco(A).`

Q-2 : On considère la matrice de taille n définie sous MATLAB , pour n donnée, par :
`A = eye(n); A( :,1)=1; A(1, :)=1; A(1,1)=n;`

Q-2-1 : Pour diverses valeurs de n, afficher la matrice A (commande `spy`). Afficher aussi la matrice `B=decompLU_eco(A)`. Qu'observe-t-on ?

Q-2-2 : Que retourne la commande MATLAB `length(find(A))` ?
Comparer `length(find(A))` et `length(find(decompLU_eco(A)))`. Conclure.

PARTIE - 2 Méthodes directes : pratiques

Exercice - 1 *Méthode de résolution par la factorisation Cholesky*

Q-1 **:** Décomposition Cholesky et complexité.

Q-1-1 **:** Écrire une fonction MATLAB de prototype `function [L,Nop] = decompCholesky(A)` qui effectue la décomposition Cholesky d'une matrice carrée symétrique définie positive et retourne le nombre `Nop` de multiplications et de divisions effectuées.

Q-1-2 **:** Générer aléatoirement des matrices symétriques définies positives (*voir* scripts joints) de taille $n = 5, \ldots 30$ et représenter en échelle logarithmique, la courbe $n \longmapsto$ Nop(n), où Nop(n) est le nombre d'opérations fournies par la décomposition Cholesky précédente.
En déduire la complexité de la factorisation Cholesky ; c'est-à-dire les constantes C et α, telles que $Nop(n) \approx Cn^{\alpha}$. Comparer cette complexité à celle obtenue par la factorisation LU (*voir* Parties précédentes).

Q-2 **:** Inversion du système linéaire par la factorisation Cholesky.

Q-2-1 **:** Ecrire une fonction MATLAB de prototype `function [x] = inverseCholesky(L,b)` qui résout le système $LL^T x = b$. Tester cette fonction sur quelques exemples.

Exercice - 2 *Discrétisation par différences finies du Laplacien en une et deux dimensions*

Q-1 **: Matrice du Laplacien en une dimension d'espace**

On considère le problème
chercher $u \in c^2([0,1])$ telle que $-u^{''}(x) = f(x)$ dans$]0,1[$, $\quad u(0) = \alpha, u(1) = \beta$; où $f \in C([0,1])$ et α, β sont deux nombres réels donnés.

On se propose de déterminer une solution approchée de cette équation par la méthode de différences finies.

Q-1-1 **:** Rappeler les différentes étapes à suivre pour discrétiser cette équation par différences finies.
Donner l'expression de la matrice A et du second membre b de la formulation matricielle du problème discrétisé.

Q-1-2 **:** Déterminer analytiquement les valeurs propres de la matrice A. Calculer le rapport entre la plus grande valeur propre et la plus petite valeur propre. Que devient ce rapport lorsque la taille N de la matrice tend vers l'infini. Conclure.

Q-1-3 **:** Écrire deux fonctions MATLAB de prototype `[A] = laplace1D(n)` et `[b] = Smlaplace1D(n,f,alpha,beta)` retournant respectivement la matrice et le second membre de la discrétisation. Et montrer qu'on peut résoudre le système $Au = b$ par la factorisation Cholesky.

Q-1-4 **:** Choisir f, α, β de sorte que la solution exacte soit $u_e(x) = sin(\pi x)$. Représenter sur un même graphique la solution exacte et la solution du système discrétisé.

Q-1-5 : Pour les mêmes expressions de f, α, β, ci-dessus, faites varier le pas de discrétisation $h = 1/(n+1)$ de façon dyadique ($h_i = h \times 2^{-i}, i = 1, 2, 3, 4$) et représenter à l'échelle logarithmique, la courbe $h_i \longmapsto \|u_e - u_{h_i}\|_\infty$, où u_{h_i} désigne la solution approchée obtenue par différences finies pour le pas h_i. Déterminer la pente de la droite (approchée) obtenue et déduire l'ordre de la discrétisation par différences finies adoptée.

Q-2 : Matrice du Laplacien en deux dimensions d'espace

On pose $\Omega =]0,1[\times]0,1[$, on se donne deux fonctions f, g continues sur $\bar{\Omega}$, et on cherche une fonction $u \in C^2(\Omega)$ telle que : $-\Delta u(x,y) = f(x,y)$ dans Ω, $\quad u(x,y) = g(x,y)$ sur $\partial\Omega$, où $\Delta u(x,y) = \frac{\partial^2 u(x,y)}{\partial^2 x} + \frac{\partial^2 u(x,y)}{\partial^2 y}$.

Q-2-1 : En suivant les étapes de discrétisation par différences finies vues en cours, donner l'expression de la matrice A et le second membre b de la formulation matricielle du problème discret.

Q-2-2 : Écrire deux fonctions MATLAB de prototype `[A] = laplace2D(n,m)` et `[b] = Smlaplace2D(n,m,f,g)` retournant la matrice A et le second membre b obtenue ci-dessus, pour les pas de discrétisation $h_x = \frac{1}{n+1}, h_y = \frac{1}{m+1}$. Décrire la matrice obtenue.

Exercice - 3 *Stockage pour les méthodes directes d'inversion de systèmes linéaires*

Le but de cet exercice est d'exhiber un stockage de matrice favorable à l'utilisation des méthodes directes d'inversion des systèmes linéaires.

Q-1 : Matrices bandes (stockage et factorisation)

Q-1-1 : A l'aide de la commande `spy`, représenter graphiquement la matrice A de la discrétisation 1D du Laplacien. Représenter aussi les matrices de sa factorisation LU et Cholesky (*où on a stocké dans une matrice de même taille que A les factorisations LU et Cholesky*).
Comparer les largeurs de bande des différentes matrices et conclure.
.

Q-1-2 : Écrire une fonction MATLAB de prototype `[AA, lbg, lbd, n ] = StockBand(A)` qui calcule les demi-largeurs de bande inférieure `lbg` et supérieure `lbd` d'une matrice pleine A et la stocke sous forme bande dans un vecteur AA de taille $(lbg + lbd + 1) \times n$.

Écrire aussi une fonction MATLAB `[A] = Band2full(AA, lbg, lbd, n)` qui transforme une matrice stockée sous format bande en une matrice pleine.

Q-1-3 : Écrire une fonction MATLAB de prototype `[AA, lbg, lbd, n , Nop] = decompLUBand(AA,lbg, lbd, n)`, qui effectue une factorisation LU d'une matrice stockée bande (*voir algorithme donné en annexe*).

Faites varier l'entier n et générer la matrice du Laplacien 1D. Récupérer les nombres d'opérations donnés par les factorisations `decompLUBand` et `decompLU_nop` (*voir script plus haut*). Représenter sur un même graphique à l'échelle logarithmique ces nombres d'opérations en fonction de la taille n. Calculer les pentes des droites obtenues et conclure.

Q-1-4 : Écrire une fonction MATLAB de prototype `[x] = inverseLUBand(AA,lbg, lbd, n, b)`, qui résout le système linéaire $Ax = b$ pour lequel la matrice A factorisée LU est stockée sous forme bande.

Q-1-5 : Écrire une fonction MATLAB de prototype `[AA, lb, n ] = StockBandSym(A)` qui calcule la demi-largeur de bande `lb` d'une matrice pleine symétrique A et stocke sa partie triangulaire inférieure sous forme bande dans un vecteur AA de taille $(lb + 1) \times n$.

Q-1-6 : Écrire une fonction MATLAB de prototype `[AA, lb, n] = decompCholBand(AA,lb, n)`, qui effectue une factorisation Cholesky d'une matrice symétrique stockée bande (*voir algorithme donné en annexe*)

Q-1-7 : Écrire des fonctions MATLAB de prototype `[x] = inverseCholBand(AA,lb, n, b)`, qui résout le système linéaire $Ax = b$ pour lequel la matrice A factorisée Cholesky est stockée sous forme bande.

Q-2 : Réduction de largeur de bande des matrices

Cette question a pour but de montrer qu'on peut ramener, via une renumérotation, une matrice donnée avec beaucoup de zéros, en une matrice de faible largeur de bande.

Il est fourni, pour cette question, une fonction `[B,p]=rcm(A,i)` à travers un fichier `rcm.m`. Pour son appel, il faut apporter une matrice A pleine de taille n et un indice $1 \leq i \leq n$. On a en retour un vecteur p de permutation de l'ensemble $\{1, \ldots n\}$, et une matrice pleine $B_{ij} = A_{p(i)p(j)}, 1 \leq i, j \leq n$.

Q-2-1 : Tester cette fonction sur certaines matrices (`laplace2D(n,n)`) en affichant, commande `spy`, les matrices entrées et sorties. Commenter les observations.

Pour différentes valeurs de n, exécuter `A=laplace2D(n,n); [B,p]=rcm(A,1); AA=decompLU_eco(A); BB=decompLU_eco(B); length(find(AA)), length(find(BB))`. Conclure.
(*La fonction `decompLU_eco` est celle implémentée plus haut*).

Q-2-2 : Générer pour différentes valeurs de n la matrice du Laplacien en dimension deux. Pour un vecteur b donné de taille n, comparer les temps d'exécution des commandes :
- `A = laplace2D(n,n); x= A\b;`
- `A = laplace2D(n,n); [B,p]=rcm(A,i); bb=b(p); y= B\bb; x(p)=yy;`

Conclure.

PARTIE - 3 Méthodes itératives de base

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 –

Q-1 **:** Rappeler les conditions nécessaires pour la 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* $\sup_{\substack{x\neq 0 \\ My=Nx}} \frac{\|y\|_\infty}{\|x\|_\infty}$)
Q-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.
Q-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.
Q-4 **:** Justifier en se basant sur le cas particulier 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 longueur 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`.

Q-1 **: Programmation**
Q-1-1 **:** Écrire une fonction MATLAB de prototype `[x,res] = Jacobi(A,b,x0, tol,iterMax)`, qui résout le système linéaire $Ax = b$ selon la méthode Jacobi.

Q-1-2 **:** Écrire une fonction MATLAB de prototype
`[x,res] = GaussSeidel(A,b,x0, tol,iterMax)` qui résout le système $Ax = b$ par une méthode de Gauss-Seidel.

Q-1-3 **:** Écrire une fonction MATLAB de prototype `[x,res] = sor(A,b,x0, tol,iterMax, w)` qui résout le système $Ax = b$ par une méthode de relaxation. Ici, `w` désigne le paramètre de relaxation.

Q-2 **: Comparaison**

Q-2-1 **:** Écrire 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` implémentées plus haut*

Q-2-2 **:** 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.

Q-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.

Q-1-1 **:** Écrire 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.

Q-1-2 **:** Écrire une fonction MATLAB `function [x] = descenteCOO (IA,JA,VA,b)` qui résout le système $Ax = b$ où A est une matrice triangulaire inférieure stockée au format coordonnées.
Écrire 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.

Q-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`.

Q-2-1 **:** Écrire 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.

Q-2-2 : Écrire une fonction MATLAB `function [x] = descenteMorse (IA,JA,VA,b)` qui résout le système $Ax = b$ où A est une matrice triangulaire inférieure stockée au format CSR.
Écrire 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.

PARTIE - 4 Méthodes itératives de type gradient

Ce TP a pour but de construire les méthodes itératives (du gradient) pour déterminer la solution x du système linéaire $Ax = b$, où A est une matrice carrée d'ordre n, *symétrique définie positive* et b un vecteur de taille n. Dans tout ce qui suit, on se trouve dans le cas réel. On désigne par $(\cdot, \cdot)$ le produit scalaire euclidien (i.e $(x, y) = \sum_{i=1}^{n} x_i y_i$) et $\| \cdot \|$ la norme associée.

Exercice - 1 *Méthode du gradient à pas fixe (ou de Richardson)*

On considère la fonction f définie de $\mathbb{R}^n$ dans $\mathbb{R}$ par $f(x) = \frac{1}{2}(Ax, x) - (b, x)$.

Q-1 : Montrer que x_0 est solution de $Ax = b$ si et seulement si x_0 minimise la fonction f.

Q-2 : On considère la méthode itérative :
$\|x_0$ étant un vecteur initial donné, $x_{k+1} = x_k - \alpha \nabla f(x_k)$.

Q-2-1 : Donner la matrice d'itération B de cette méthode et conclure que cette méthode converge si et seulement si $0 < \alpha < \frac{2}{\lambda_n}$ où $0 < \lambda_1 \leq \ldots \leq \lambda_n$ sont les valeurs propres de la matrice symétrique définie positive A.

Q-2-2 : Montrer que le meilleur choix de α est : $\alpha_{opt} = \frac{2}{\lambda_n + \lambda_1}$ et qu'alors $\varrho(B) = \frac{\lambda_n - \lambda_1}{\lambda_n + \lambda_1}$.

Q-2-3 : Écrire en pseudo-code l'algorithme de gradient à pas constant, en faisant intervenir le résidu dans les itérations (*on rappelle que le résidu à l'itération k est défini par $r_k = b - Ax_k$*).

Q-2-4 : Programmer cet algorithme sous MATLAB à travers une fonction de prototype
`function [x,res]=Gradient(A, b, x0, tol,iterMax, alpha)`, où
- `A` est 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, `alpha` désigne le paramètre α,
- `x` est la solution approchée,
- `res` un vecteur stockant la valeur de la norme du résidu à chaque itération.

Q-2-5 : On considère la matrice et le second membre du Laplacien `A,b` générés par la commande
`[A,b] = laplace2Df(10,10)` (*voir Parties précédentes*).

On fixe `iterMax = 10000, tol = 1E-4`. Faites varier α entre `1E-3` et `3 E-3` par pas de `1E-4`. Représenter le logarithme du nombre d'itérations en fonction du paramètre α. Déterminer numériquement la valeur α conduisant à un nombre d'itérations minimal. Comparer avec la valeur donnée par la théorie.

Exercice - 2 *Méthode du gradient à pas variable*

Q-1 : On suppose construite une suite $(p_0, p_1, \ldots p_k, \ldots)$ de vecteurs linéairement indépendants. Et on considère la méthode itérative suivante :

x_0 vecteur initial donné,
$x_{k+1} = x_k + \alpha_k p_k$.
Où α_k est choisi tel qu'il réalise le minimum de $f(x_k + \alpha p_k)$.

Q-1-1 : Montrer que $\alpha_k = \frac{(r_k,p_k)}{(Ap_k,p_k)}$, où $r_k = b - Ax_k$, et conclure que $(r_{k+1}, p_k) = 0, \forall k \geq 0$.

Q-1-2 : On pose $E(x_k) = (A(x_k - x), (x_k - x))$ où x est la solution de $Ax = b$.
Montrer que pour la valeur de α_k ci-dessus, on a $E(x_{k+1}) = E(x_k) - \frac{(r_k,p_k)^2}{(Ap_k,p_k)}$.

Q-1-3 : En remarquant que $E(x_k) = (A^{-1}r_k, r_k)$, montrer que $E(x_{k+1}) = E(x_k)\left(1 - \frac{(r_k,p_k)^2}{(A^{-1}r_k,r_k)(Ap_k,p_k)}\right)$.

Q-1-4 : Déduire de la question précédente que $E(x_{k+1}) \leq E(x_k)\left[1 - \frac{1}{cond(A)}\left(\frac{r_k}{\|r_k\|}, \frac{p_k}{\|p_k\|}\right)^2\right]$.
Où $cond(A) = \frac{\lambda_n}{\lambda_1}$ désigne le conditionnement de la matrice A.
on utilisera le fait que $(Ay, y) \leq \lambda_n(y, y), \quad (A^{-1}y, y) \leq \frac{1}{\lambda_1}(y, y) \forall y$.
Conclure qu'une condition suffisante de convergence est de choisir les p_k tels que
$\forall k \geq 0 \quad \left(\frac{r_k}{\|r_k\|}, \frac{p_k}{\|p_k\|}\right) \geq \mu > 0$, où μ est une constante indépendante de k.

Q-1-5 : Déduire de la question précédente que $p_k = r_k, \forall k \geq 0$ est un choix possible assurant la convergence.

Q-2 : On prend dans cette question $p_k = r_k \ \forall k \geq 0$.

Q-2-1 : Écrire l'algorithme ainsi obtenu.

Q-2-2 : Que devient $E(x_{k+1})$ de la question 1-c ci-dessus ?

Q-2-3 : *On admet l'inégalité de Kantorovich suivante :* $\frac{(Ay,y)(A^{-1}y,y)}{(y,y)^2} \leq \frac{(\lambda_n+\lambda_1)^2}{4\lambda_n\lambda_1} \ \forall y \neq 0$..
Montrer qu'on a alors $E(x_{k+1}) = E(x_k)\left(\frac{cond(A)-1}{cond(A)+1}\right)^2$.

Q-2-4 : Conclure que $\|x_k - x\| \leq \sqrt{\frac{E(x_0)}{\lambda_1}}\left(\frac{cond(A)-1}{cond(A)+1}\right)^k$.

Q-2-5 : Programmer cet algorithme sous MATLAB à travers une fonction de prototype
`function [x,res]=GradientV (A, b, x0, tol, iterMax).`
(*Les arguments sont définis comme à la question 2-d de l'exercice 1.*)

Exercice - 3 *Méthode du gradient conjugué*

Une amélioration de la méthode de gradient à pas variable consiste à choisir les directions p_k telles que x_{k+1} réalise le minimum de f sur $x_0 + [p_0, p_1, \ldots p_k]$, en souhaitant que ce problème de minimisation globale soit égale au problème de minimisation locale : $\min_{x=x_0+\bar{x}+y, y\in[p_k]} f(x)$, dans lequel $\bar{x} \in [p_0, p_1, \ldots p_{k-1}]$ est connu et issu d'une précédente minimisation. Les deux premières questions ci-dessous justifient pourquoi on choisit les p_k A-conjugués.

Q-1 : Monter que $f(x_0 + \bar{x} + \alpha p_k) = f(x_0 + \bar{x}) + \alpha(p_k, A\bar{x}) + \frac{\alpha^2}{2}(p_k, Ap_k) - \alpha(p_k, r_0)$.

Q-2 : Conclure qu'un moyen de découpler le problème de minimisation globale en une succession de problèmes de minimisation locale est de prendre $(p_k, A\bar{x}) = 0$, ce qui est équivalent à $(p_k, Ap_i) = 0, \ \forall 0 \leq i \leq k-1$. **On dit dans ce cas que les vecteurs** $p_i, i = 1 \ldots k$ **sont A-conjugués**.

La méthode du gradient conjugué consiste alors en deux points :

- choisir les directions p_k A-conjuguées c'est-à-dire $(Ap_k, p_j) = 0 \; \forall 0 \leq j \leq k-1$,
- $p_0 = r_0$, et prendre p_{k+1} dans le plan contenant r_{k+1} et p_k c'est-à-dire $p_{k+1} = r_{k+1} + \beta_{k+1} p_k$.

Q-3 : Montrer que les vecteurs p_{k+1} et p_k sont A-conjugués si et seulement si $\beta_{k+1} = -\frac{(r_{k+1}, Ap_k)}{(p_k, Ap_k)}$.

Q-4 : En remarquant que $(r_k, p_{k-1}) = 0 \; \forall k$, montrer que $(r_k, p_k) = (r_k, r_k) \; \forall k$.
Simplifier alors l'expression de α_k.

Q-5 : En écrivant $r_k = p_k - \beta_k p_{k-1}$, montrer que $(r_{k+1}, r_k) = 0$.(*on utilisera l'expression de* β_k).

Q-6 : En écrivant $Ap_{k-1} = \frac{1}{\alpha_{k-1}}(r_{k-1} - r_k)$, montrer que
$(Ap_{k-1}, r_k) = -\frac{1}{\alpha_{k-1}}(r_k, r_k)$ et $(Ap_{k-1}, p_{k-1}) = \frac{1}{\alpha_{k-1}}(r_{k-1}, r_{k-1})$.
Simplifier alors l'expression de β_{k+1}.

Q-7 : Montrer que $(r_k, r_j) = 0 \; \forall \, 0 \leq j \leq k-1$. En déduire qu'en arithmétique exacte, l'algorithme du gradient conjugué converge en au plus n itérations, où n est la taille du système.
(*On remarquera que si* $r_k \neq 0, \forall \, 0 \leq k \leq n-1$, *alors* $[r_0, \ldots, r_{n-1}]$ *est une base de* $\mathbb{R}^n$).

Q-8 : Écrire l'algorithme du gradient conjugué.

Q-9 : Programmer cet algorithme sous MATLAB à travers une fonction de prototype
`function [x,res]=GradientC (A, b, x0, tol, iterMax).`
(*Les arguments sont définis comme à la question 2-d de l'exercice 1.*)

Exercice - 4 *Comparaison numérique des méthodes*

On considère A et b la matrice et le second membre obtenus par discrétisation par différences finies du Laplacien sur $]0,1[\times]0,1[$ avec pour second membre $f = 1$ et des conditions aux limites de Dirichlet homogènes. La grille utilisée est définie par $h_x = \frac{1}{n+1}, h_y = \frac{1}{m+1}$.

Q-1 : Exécuter le script suivant, en vous assurant que les fonctions appelées sont bien accessibles. (*Se référer aux précédentes Parties*).

Comparaison des méthodes

```
clear all;
A = laplace2D(10,10);
f = inline('x-x+1','x','y');
g = inline('x-x+0','x','y');
b = Smlaplace2D(10,10,f,g);
x0=b;x0(:,1) = 0;
tol = 1e-9;
iterMax = 100000;
disp(' calcul ');
[x,resJ]=Jacobi(A,b,x0, tol ,iterMax);
[x,resG]=GaussSeidel(A,b,x0, tol ,iterMax);
[x,resS]=sor(A,b,x0, tol ,iterMax, 1.6);
[x,resGradV]=gradientV(A,b,x0, tol ,iterMax);
[x,resGrad]=gradient(A,b,x0, tol ,iterMax,1.);
[x,resGradC]=gradientC(A,b,x0, tol ,iterMax);
disp(' affichage ');
semilogy(1:length(resJ),resJ,'-.r',
         1:length(resG),resG,'--ms',
         1:length(resS),resS,'--s',
         1:length(resGradV),resGradV,':s',
         1:length(resGrad),resGrad,'-b',
         1:length(resGradC),resGradC,'-*r'
        );
legend('Jacobi','Gauss-Seidel','SOR w=1.6','gradient pas variable',
    'gradient pas fixe optimal','gradient conjugue');
```

Q-2 : Interpréter les résultats observés.

PARTIE - 5 Éléments de réponse sur les *méthodes itératives de type gradient*

Exercice - 1 *Méthode du gradient à pas fixe (ou de Richardson)*

Q-1 : On a $\forall t \in \mathbb{R}, \forall y \in \mathbb{R}^n,\ f(x_0+ty) = f(x_0) + t(Ax_0 - b, y) + \frac{t^2}{2}(Ay, y)$. Par conséquent, x_0 minimise f si et seulement si pour tout $t \in \mathbb{R}$ et $y \in \mathbb{R}^n$, on a $t(Ax_0 - b, y) + \frac{t^2}{2}(Ay, y) \geq 0$. Soit $t(Ax_0 - b, y) \leq -\frac{t^2}{2}(Ay, y) \leq 0$. Soit encore $(Ax_0 - b, y) = 0\ \ \forall y \in \mathbb{R}^n$; C'est-à-dire $Ax_0 - b = 0$.

Q-2 :

Q-2-1 : voir cours

Q-2-2 : voir cours

Q-2-3 : Télécharger sur la page web http ://www.math.u-psud.fr/~apoung le code MATLAB . *L'accès est soumis à une authentification.*

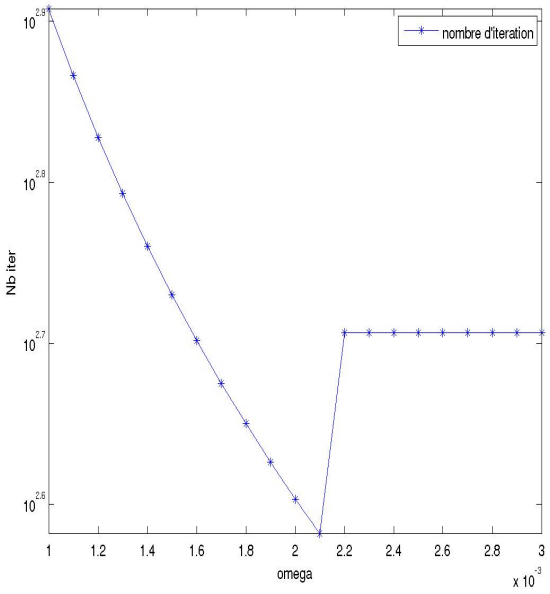
Q-2-4 : Idem que précédemment

script de test du gradient

```
clear all;

A = laplace2D(10,10);
f = inline('x-x+1','x','y');
g = inline('x-x+0','x','y');
b = sndmf2D(10,10,f,g);

w = .001:.0001:.003;
n = length(w);
y=[];
x0=b;x0(:,1) = 0;
tol = 1e-6;
iterMax = 10000;
Niter=[];
disp(' calcul ');
for k=1:n
[x,res]=gradient(A,b,x0, tol ,iterMax, w(k));
Niter(k) = length(res);
y=x;
end
l = eig(A);
wop = 2./(min(l) + max(l))
disp(' affichage ');
semilogy(w,Niter,'*-');
xlabel('omega');ylabel('Nb iter');
legend('nombre d''iteration ');pause();
print -djpeg 'gradient.jpg';
```



La valeur approximative est $\omega_{opt} \approx 0.0021$. On retrouve la valeur théorique $\frac{2}{\lambda_1+\lambda_n}$

Exercice - 2 *Méthode du gradient à pas variable*

Cet exercice a été entièrement traité en cours.

Exercice - 3 *Méthode du gradient conjugué*

Q-1 : Pour $f(x) = \frac{1}{2}(Ax, x) - (b, x)$, un calcul simple donne :

$\nabla f(x) = Ax - b, \nabla^2 f(x) = A$ et $f(x+h) = f(x) + (\nabla f(x), h) + \frac{1}{2}(\nabla^2 f(x)h, h)$

D'où $f(x_0 + \bar{x} + \alpha p_k) = f(x_0 + \bar{x}) + \alpha(A\bar{x}, p_k) - \alpha(r_0, p_k) + \frac{\alpha^2}{2}(p_k, Ap_k)$.

Q-2 : De la question précédente il découle que $f(x_{k+1}) = \min_{y \in [y_0, \dots p_k]} f(y) = \min_{y = x_0 + \bar{x} + \alpha p_k, \bar{x} \in [y_0, \dots p_{k-1}]} f(y)$
$= \min_{\bar{x}} \min_{\alpha} \left[f(x_0 + \bar{x}) + \alpha(A\bar{x}, p_k) - \alpha(r_0, p_k) + \frac{\alpha^2}{2}(p_k, Ap_k) \right]$.
Par suite si $(A\bar{x}, p_k) = 0$ le problème de minimisation global se découple en deux problèmes de minimisation :
$f(x_{k+1}) = \min_{\bar{x}} f(x_0 + \bar{x}) + \min_{\alpha} \left[-\alpha(r_0, p_k) + \frac{\alpha^2}{2}(p_k, Ap_k) \right]$.

Le premier nous fournit x_k, c'est à dire $f(x_k) = \min_{\bar{x}} f(x_0 + \bar{x})$, et le second nous donne $\alpha_k = \frac{(r_0, p_k)}{(Ap_k, p_k)}$. On peut d'ailleurs déduire une première propriété de la méthode :

En effet, pour la valeur de $\bar{x}$, qui réalise le minimum de $\min_{\bar{x}} f(x_0 + \bar{x})$ et qui permet par la même occasion de déterminer x_k, le problème de détermination de x_{k+1} s'écrit :

$f(x_{k+1}) = \min_{\alpha} f(x_k + \alpha p_k)$. Or ce problème a une solution donnée par $\alpha_k = \frac{(r_k, p_k)}{(Ap_k, p_k)}$.
On en déduit alors en égalant les expressions de α_k que :
$$\boxed{(r_0, p_k) = (r_k, p_k) \quad \forall k \geq 0}.$$

Q-3 : $0 = (p_{k+1}, Ap_k) = (r_{k+1}, Ap_k) + \beta_{k+1}(p_k, Ap_k) \Longrightarrow \beta_{k+1} = -\frac{(r_{k+1}, p_k)}{(Ap_k, p_k)}$

Q-4 : $r_k = r_{k-1} - \alpha_{k-1} Ap_k \Longrightarrow (r_k, p_{k-1}) = (r_{k-1}, p_{k-1}) - \alpha_{k-1}(Ap_{k-1}, p_{k-1}) = 0$ par définition de α_{k-1}.
Donc $\boxed{(r_k, p_{k-1}) = 0 \quad \forall k \geq 0}$.

Par conséquent, $(r_k, p_k) = (r_k, r_k + \beta_k p_{k-1}) = (r_k, r_k)$. D'où $\boxed{\alpha_k = \frac{(r_k, p_k)}{(Ap_k, p_k)} = \frac{(r_k, r_k)}{(Ap_k, p_k)}}$.

Q-5 : $(r_{k+1}, r_k) = (r_{k+1}, p_k - \beta_k p_{k-1}) = -\beta_k(r_{k+1}, p_{k-1})$ d'après 4.
Or $(r_{k+1}, p_{k-1}) = (r_k - \alpha_k Ap_k, p_{k-1})$. Et ce dernier s'annule par suite de 4. et de la $A-$conjugaison. Donc $\boxed{(r_{k+1}, r_k) = 0 \quad \forall k \geq 0}$.

Q-6 : $Ap_k = \frac{1}{\alpha_k}(r_k - r_{k+1})$. D'où
$(Ap_k, r_{k+1}) = \frac{1}{\alpha_k}(r_k - r_{k+1}, r_{k+1}) = -\frac{1}{\alpha_k}(r_{k+1}, r_{k+1})$.
$(Ap_k, p_k) = \frac{1}{\alpha_k}(r_k - r_{k+1}, p_k) = \frac{1}{\alpha_k}(r_k, p_k) = \frac{1}{\alpha_k}(r_k, r_k)$

D'où la nouvelle expression de $\boxed{\beta_{k+1} = \frac{(r_{k+1}, r_{k+1})}{(r_k, r_k)}}$.

A ce stade nous avons tous les ingrédients pour écrire l'algorithme du gradient conjugué.

Q-7 : Montrons néanmoins une seconde propriété qui nous permettra de montrer qu'en arithmétique exacte la méthode du gradient conjugué converge en au plus n itération où n est la dimension du système.

On va montrer que les résidus issus des itérations sont tous orthogonaux.

En effet, on a déjà vu que $(r_k, r_{k-1}) = 0$. On initialise ainsi une récurrence. Pour conclure, supposons que $(r_k, r_l) = 0 \quad \forall j+1 \leq l \leq k-1$ et montrons alors que $(r_k, r_j) = 0$.

En effet, on a $r_j = r_{j+1} + \alpha_j A p_j$, et de 5 il vient que $(r_j, r_k) = \alpha_j (A p_j, r_k)$.
Mais $r_k = p_k - \beta_k p_{k-1}$. Par suite de la $A-$conjugaison, on a $(A p_j, r_k) = (A p_j, p_k) - \beta_k (A p_j, p_{k-1}) = 0$. Donc $\boxed{(r_k, r_j) = 0 \quad \forall 0 \leq j \leq k-1}$. Par suite si $r_k \neq 0 \forall 0 \leq k \leq n-1$, on aura $r_n = 0$ car r_n est orthogonal à $[r_0, \ldots r_{n-1}]$ qui est un espace vectoriel de dimension n car de l'orthogonalité des r_k la famille $\{r_0, \ldots, r_k\}$ est libre.
Mais $r_n = b - A x_n$ donc $r_n = 0 \Rightarrow A x_n = b$.

Q-8 : L'algorithme du gradient conjugué s'écrit :

Algorithme du Gradient conjugué

Algorithme du Gradient Conjugue

Initialisation :

x_0

$r_0 = b - A\ x_0$

$p_0 = r_0$

Iteration :

Pour $k = 1 \ \ldots$

$$\alpha_k = \frac{(r_k, r_k)}{(A\ p_k, p_k)}$$

$$x_{k+1} = x_k + \alpha_k\ p_k$$

$$r_{k+1} = r_k - \alpha_k\ A\ p_k$$

$$\beta_{k+1} = \frac{(r_{k+1}, r_{k+1})}{(r_k, r_k)}$$

$$p_{k+1} = r_{k+1} + \beta_{k+1}\ p_{k-1}$$

Fin pour k

Pour programmer cet algorithme on fait intervenir un test d'arrêt basé sur le résidu ($\|r_k\| \leq \varepsilon \|r_0\|$) et sur le nombre maximum d'itérations.

Function du gradient conjugué

```
function [x,res] =gradientC(A,b,x0,tol,iterMax)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% function [x,res] =gradientC(A,b,x0,tol,iterMax)
% fonction qui  resoud le systeme Ax=b
% parla methode de Gradient conjugue
% Entree : A -> matrice
%          b -> vecteur second membre
%          x0 -> vecteur initial
%          tol -> parametre epsilon du test d'arret
%          iterMax -> nombre maximal d'iterations
%
% Sortie : x -> solution
%          res-> vecteur norme du residu par iteration
%
% MAO CALCUL SCIENTIFIQUE
% M1 MFA          2008-2009
% J.-B. APOUNG K. Lab AN-EDP Univ d'ORSAY
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

[n,n]=size(A);
res=[];
%quelques verifications
[n,n]=size(A);
if ((n~=length(b)) | (length(b)~= length(x0)))
    error('erreur sur les dimensions');
end
% initialisation
x = x0;
r = b - A * x;
p=r;
gamma=r'*r;
iter = 1;
res(iter) = norm(r,2);
while ( norm(r,2) > tol && iter <iterMax)
    y = A*p;
    alpha = gamma/(y'*p);
    x = x + alpha*p;
    r = r - alpha*y;
    beta = r'*r / gamma;
    gamma = r'*r;
    p = r + beta * p;
    iter = iter+1;
    res(iter) = norm(r,2);
end
```

Résultat de convergence (voir cours) :

$$\|x_k - x\| \leq 2\sqrt{cond(A)} \left(\frac{\sqrt{cond(A)}-1}{\sqrt{cond(A)}+1} \right)^k \|x_0 - x\| \quad forall k \geq 0.$$

Exercice - 4 *Comparaison numérique des méthodes*

L'exécution du code ci-dessous conduit à la figure ci-après.

Script de comparaison des méthodes

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% script qui compare les differentes methodes
% iteratives
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Indexation des methodes
%
%  1 -> jacobi
%  2 -> Gauss-Seidel
%  3 -> sor avec w = 1.6
%  4 -> gradient pas variable
%  5 -> gradient pas fixe mais parametre optimal
%  6 -> gradient conjugue
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% MAO CALCUL SCIENTIFIQUE
% M1 MFA          2008-2009
% J.-B. APOUNG K. Lab AN-EDP Univ d'ORSAY
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clear all;
A = laplace2D(10,10);
f = inline('x-x+1','x','y');
g = inline('x-x+0','x','y');
b = Smlaplace2D(10,10,f,g);
x0=b;x0(:,1) = 0;
tol = 1e-9;
iterMax = 100000;
disp(' calcul ');
[x,resJ]=Jacobi(A,b,x0, tol ,iterMax);
[x,resG]=GaussSeidel(A,b,x0, tol ,iterMax);
[x,resS]=sor(A,b,x0, tol ,iterMax, 1.6);
[x,resGradV]=gradientV(A,b,x0, tol ,iterMax);
[x,resGrad]=gradient(A,b,x0, tol ,iterMax,1.);
[x,resGradC]=gradientC(A,b,x0, tol ,iterMax);
disp(' affichage ');
semilogy(1:length(resJ),resJ,'-.r',
         1:length(resG),resG,'--ms',
         1:length(resS),resS,'--s',
         1:length(resGradV),resGradV,':s',
         1:length(resGrad),resGrad,'-b',
         1:length(resGradC),resGradC,'-*r'
        );
legend('Jacobi','Gauss-Seidel','SOR w=1.6','gradient pas variable',
    'gradient pas fixe optimal','gradient conjugue');
 xlabel('iterations');ylabel('log(norm(residu))');
print -djpeg 'compareiter.jpg'; %% -> ceci permet d'enregistrer le graphique dans un fichier (ici au format jpeg)
```