L’objectif du TP est de résoudre le SEL d’inconnu x suivant l’équation AX=b. A est une matrice carrée de dimension n, tandis que b et x sont des matrices colonnes à n lignes. Pour répondre à ce problème, on utilisera trois méthodes : la méthode directe, itérative et enfin de relaxation. On observera l’influence des différents paramètres sur nos résultats.

1. Méthodes et programme
2. Facteurs d’influence de la résolution du SEL

Il existe deux facteurs d’influence pour la résolution d’un SEL : le conditionnement de la matrice et sa taille.

A travers les méthodes que l’on va utiliser pour résoudre ce SEL, on veut essayer de mettre en évidence ces facteurs d’influence.

On cherche alors à créer un programme nous permettant de choisir à notre guise la valeur du conditionnement de la matrice et la valeur de sa taille.

* Conditionnement d’une matrice :

Le conditionnement d’une matrice relativement à une norme matricielle *||.||alpha* est défini comme suit :

Pour *A appartenant à R^N\*N*

*kalpha(A)=||A||alpha ||A^(-1)||alpha*

On fixe alpha=2.

Pour étudier l’influence de ce conditionnement, on fera varier k en prenant les valeurs 10, 1000, 10^8 et 10^16.

* Taille de la matrice

Pour étudier l’influence de la taille n, on la fera varier en prenant n=10, 50, 100, 150.

* Programme :

On construit la matrice A selon les démarches définies dans l’énoncé

*// Construction d'une matrice diagonale de le forme (1,K^(-1/n-1),..., 1/K)*

*On pourra choisir n la taille et k le conditionnement de la matrice*

function D=diagonale(n, k)

*//on crée une matrice carrée de zéros de taille n*

 D = zeros(n,n)

 *//on défini le premier terme de la mtrice c'est-à-dire (1,1) comme étant 1*

 D(1,1) = 1;

 *//on remplis notre matrice en respectant la méthode de l'énoncé*

 for i = 2 : n

 D(i,i) = k^(-1/(n-(i-1)));

 end

endfunction

*// Construction de la matrice tridiagonale (H dans l’énoncé):*

function D=tridiagonale(n, k)

 D = zeros(n,n)

 *//on remplit notre matrice*

 for i = 1 : n-1

 *//les éléments au dessus et en dessous de la diagonales sont égaux à -1*

 D(i+1,i) = -1;

 D(i,i+1) = -1;

 *//les éléments de la diagonale sont égales à 2*

 D(i,i) = 2;

 end

 *//on définit le dernier élément de la diagonale étant égale à 2*

 D(n,n) = 2;

endfunction

*// Conditionnement d'une matrice k(A)=||A||\*||A^(-1)||*

function condi=conditionnement(A)

 condi = norm(A) \* norm(inv(A));

endfunction

*//on fait la décomposition en valeurs singulière de notre matrice diagonale D*

[U,S,V] = svd(D);

*//à l'aide des matrices orthogonales U et V on construit notre matrice A*

A = U\*diagonale(n,n)\*V';

*A ce stade, on peut vérifier le conditionnement de A :*

conditionnement(A);

*// Construction de b (dimension : n lignes et une colonne)*

B = A \* ones(n,1)

1. Résolution directe

La résolution directe consiste à demander à Scilab de résoudre l’équation Ax=b grâce à la fonction linsolve. En effet, cette fonction permet de résoudre un système d’équations linéaires, c'est-à-dire une équation de type Ax=b, où A est la matrice réelle des coefficients du système d'équations, et b un vecteur de constantes.

Programme réalisé :

*//grâce à la fonction linsolve on calcule la solution numérique x du système Ax=b*

X1 = linsolve(A, -B);

1. Méthode itérative

Les méthodes itératives consistent à utiliser un vecteur initial X0 afin de produire une suite de vecteurs du type : X(k+1)=F(X(k)) où X(k) représente la solution obtenue au bout de k itérations.

Il faut décomposer la matrice A telle que A = D+L+U où D est la diagonale de A, L, sa partie strictement inférieure et U, sa partie strictement supérieure.

On construit les matrices D, L et U :

Programme réalisé :

*// Construction de la matrice L :*

function L=matriceL(A)

 l = length(A(1,:));

 L = zeros(l,l);

 for i=1:l

 for j=1:l

 if i>j then

 L(i,j)=A(i,j);

 else

 L(i,j)=0;

 end

 end

 end

endfunction

*// Construction de la matrice U :*

function U=matriceU(A)

 l = length(A(1,:));

 U = zeros(l,l);

 for i=1:l

 for j=1:l

 if i<j then

 U(i,j)=A(i,j);

 else

 U(i,j)=0;

 end

 end

 end

endfunction

*// Construction de la matrice D :*

function D=matriceD(A)

 l = length(A(1,:));

 D = zeros(l,l);

 for i=1:l

 D(i,i)=A(i,i);

 end

endfunction

1. Méthode de Jacobi

Pour tout k entier naturel différent de zéro (représentant le nombre d’itérations), on a :

X(k+1)=-D^(-1) (L+U) X(k) + D^(-1)b

Avec D, L, U et b matrices définies précédemment.

Par ailleurs, l’énoncé impose une erreur résiduelle (de l’ordre de 10^(-8)) ce qui permet de définir une condition d’arrêt de l’itération.

Notre critère d’arrêt est ainsi défini :

||r(k)|| / ||b|| < 10^(-8)

Où r(k)=b-AX(k)

Ainsi, tant que ||r(k)|| / ||b|| ne dépasse pas 10^(-8), on a une boucle for qui va réitérer la suite X(k+1)=-D^(-1) (L+U) X(k) + D^(-1)b à partir du rang k=1.

Dans notre programme, on laisse l’utilisateur choisir l’ordre de valeur de l’erreur résiduelle et on note cette variable seuil.

Programme réalisé :

function X=jacobi(A, b, k, seuil)

 D = matriceD(A);

 L = matriceL(A);

 U = matriceU(A);

 X = zeros(length(A(1,:)),1);

 x = X;

 for i=1:k

 X = -inv(D) \* (L + U)\* x + inv(D)\*b;

 erreurResiduel = norm( (A\*X) - b) ;

 erreurResiduel=erreurResiduel / norm(b);

 *//on impose la condition d'arrêt lorque notre erreur résiduel est inférieur au seuil*

 if erreurResiduel < seuil then

 break;

 end

 x = X;

 end

endfunction

A présent, il faut connaître le nombre d’itérations m effectués pour trouver notre solution avec l’erreur résiduelle imposée. On sait qu’en diminuant e(k) de H par rapport à e(0) on a :

m>= H/log(ro(Rj)) où ro(Rj) est le rayon spectrale de la matrice d’itération de Jacobi Rj avec Rj=-D^(-1) (L+U).

Programme réalisé :

*Calcul du rayon spectral d'une matrice!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!! utiliser spec*

function r=RayonSpectral(A)

 Dr = svd(A);

 r = max(Dr)

endfunction

function m=nb\_iteration(seuil, A, e0)

 D = matriceD(A);

 L = matriceL(A);

 U = matriceU(A);

 H = log10( seuil) / log10(10);

 Rj = -inv(D) \* (L + U); // Définition de la matrice d’itération de Jacobi

 m = log10(seuil) / log10(RayonSpectral(Rj)) ; //La fonction RayonSpectrale a préalablement été définie dans le programme comme ci-dessus.

endfunction

1. Méthode de relaxation
2. Résultats et interprétation :

 Après avoir fait fonctionner notre programme, on obtient pour chaque valeur de n et de k la matrice la matrice B, la matrice de Jacobi, l’erreur directe, la norme de l’erreur e ainsi que le nombre d’itérations m.

 Nous avons joint à notre compte-rendu un fichier Excel où nous avons donné tous les résultats que nous avons obtenus en modifiant les valeurs de k et n.

 Observation du tableau 1: plus la taille de la matrice est grande plus la valeur de l’erreur est grande. Cela était prévisible dans la mesure où le nombre d’équations et d’inconnus augmente avec la taille de la matrice, cela engendrerait plus d’erreurs et donc plus de calculs.

De même, plus le conditionnement augmente plus l’erreur grandit. On en déduit que notre résultat est plus précis lorsque nous choisissons n et k petit.

 Observation du tableau 2 : Le nombre d’itération augmente en fonction de la taille de la matrice. Ce qui s’explique par le fait que le nombre de calculs à faire augmente par l’augmentation du nombre d’équations et d’inconnus.

 Cependant, une incohérence apparait quant aux nombre d’itérations. En effet, pour n=10, une première valeur incohérente apparait pour k=1000 : m, le nombre d’itérations, est négatif (-25.53). Pour la même valeur de n, le nombre d’itérations restera négatif mais augmentera lorsque k augmente. Pour les valeurs suivantes de n des valeurs négatives vont aussi apparaitre mais pour un k plus grand. Notre analyse de ce problème est qu’il s’agit d’une divergence du nombre d’itération lorsque k augmente.

 De plus, pour les valeurs de n=50, 100,150 et de k=1000, la valeur de m augmente dans un premier temps (par rapport au résultat de m pour k=10), puis pour les valeurs suivantes de k(qui augmente) les valeurs de m diminue (sauf dans le cas n=100 et k=10^8). Cela montre les limites du calculateur : à partir d’une certaine valeur de conditionnement, il ne fait pas beaucoup d’itérations ce qui explique aussi l’augmentation de l’erreur observée au tableau1.