RENDU_TP/TD

Algèbre linéaire et résolution de système linéaire par des méthodes directes.

Réalisé par:
Mr Lougani Faouzi

Résumé :

Le but de ce TD/TP est d'implémenter des algorithmes numériques pour la résolution de problèmes .Pour se faire il faut connaître le champ d'application de ce dernier et son comportent numérique .On a essayé de comprendre l'effet d'une variation des paramètres sur le temps de calcul et sur la précision des résultats en resaluant les systèmes numériques AX=B,problème jouet ….. les exercices ci-joint.
En résumé il faut savoir évaluer la complexité des algorithmes en termes de complexité arithmétiques et de stockage en mémoire afin d'implémenter des algorithmes efficaces en termes de performance .

Les fichiers/rapport sont disponibles en annexe ou sur :
https://github.com/lougani-faouzi/calcul_numerique

Exercice 2 :

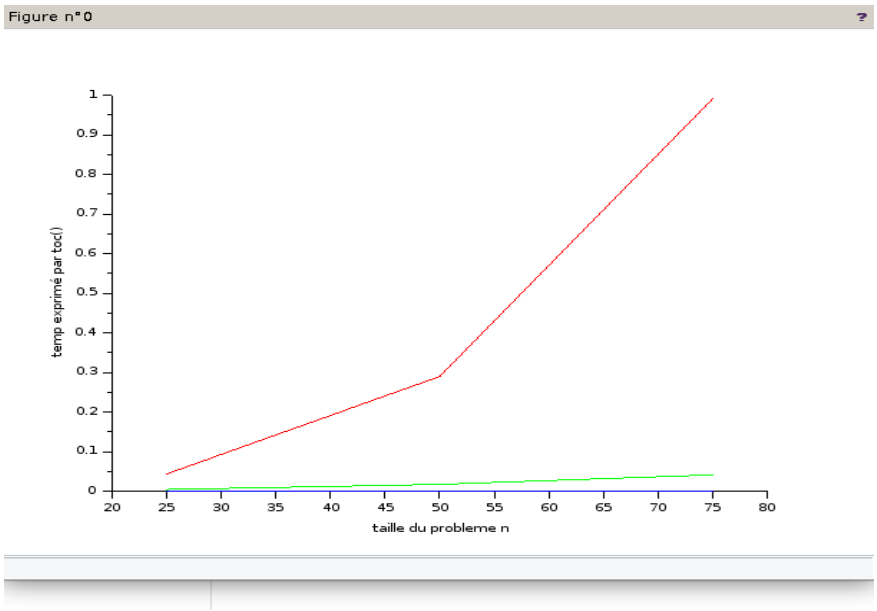
5-L'erreur avant et l'erreur arrière. Que dire des résultats obtenus
On voit que l'erreur avant > l'erreur arrière
pour un exemple d'application de la fonction exo2(n,ex) avec n=3 et ex=16 ,
cela est du au fait que lorsque on calcule la solution (systeme Ax=b par exemple comme notre cas) avec un algorithme (la fonction "\" de scilab comme exemple) nous donne pas forcement un résultat exact .Car chaque algorithme a des propriétés sur la convergence et sa complexité arithmétique.

En augmentant la taille des matrices a chaque fois (n=10,100,1000) on voit que le **conditionnement augmente** ,un résultat mème s'impose a l'exécution ,on **prend plus du temps .**
Donc on déduit que : **augmenter la taille du problème =>augmenter le conditionnement .**

Si on dépasse un certain n(n=1 00 000 mon cas)pour notre fonction exo2() (**n= un certain grand nombre)**cela permet une saturation de mémoire .et même non exécution du script .

La précision numérique (float point précision ou double point précision) des résultats,dépend de plusieurs facteur notamment la taille du problème ,ici n **est grand nous donne par exemple moins de précision .**

Exercice3 :



En faisant un plot2d on aura le graphe précédent
Analyse des résultats obtenus :
Pour n=25 puis 50 puis 75
-couleur rouge=graphe matmat3b(A,B)
-couleur vert=graphe matmat2b(A,B)
-couleur bleu=graphe matmat1b(A,B)

On déduit que :matmat1b(A,B)>matmat2b(A,B)>matmat3b(A,B)
- Mesure de temps avec tic et toc en stockant dans une variable t=toc() ; a chaque fois
-Complexité et temps :

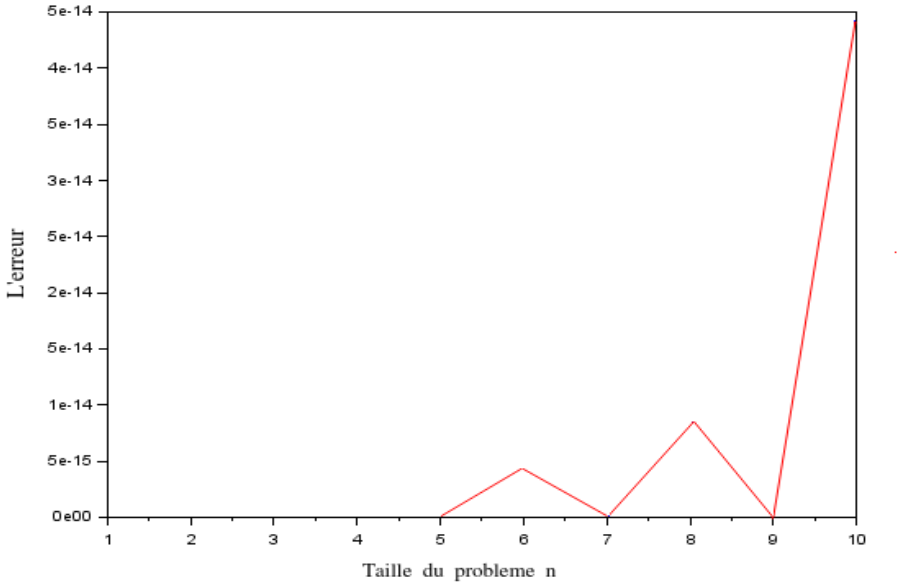
Donc :
la complexité de matmat1b(A,B)=O(n)
la complexité de matmat2b(A,B)=O(n^2) c'est une complexité quadratique
la complexité de matmat3b(A,B)=O(n^3) c'est une complexité cubique
ceci justifié par le nombre de boucles de chacune des fonctions

Donc :
On déduit que :
1/augmenter la taille du problème =>augmenter le temps d'exécution
2/un code optimale est un code qui a moins de complexité **matmat1b(A,B)=O(n) pour notre cas**

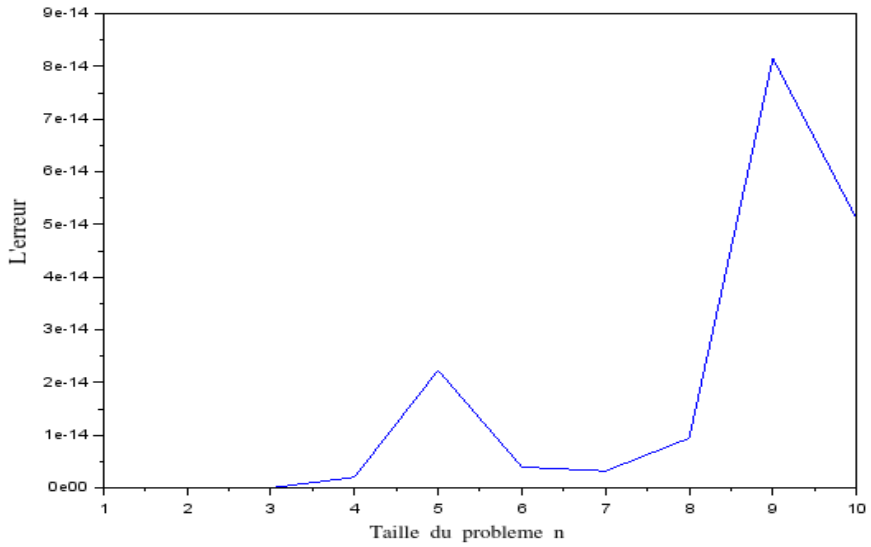
Conclusion : **matmat1b(A,B)>temps matmat2b(A,B)>temps matmat3b(A,B)**

Exercice 4 :

-Erreur ?
On varie **n (la taille du problème)** a chaque fois et on étudie les résultats des graphes obtenus
Graphe **(Usolve)** :



Graphe **(Lsolve):**



Une comparaison nous conduit à dire que **(Lsolve)** donne moins d'erreurs (si la taille du problème augmente) contrairement a **(Usolve)** les erreurs augmentent en fonction de n (plus grand).
Donc **(Lsolve)** est meilleure par apport à **(Usolve).**

-Étude par apport au conditionnement ?
le conditionnement en n=5 :
Usolve=60,9 Pour des valeurs **xex** générées aléatoirement
Lsolve=10
-le conditionnement en n=10:
Usolve=56 Pour des valeurs **xex** générées aléatoirement
Lsolve=39
Puisque **Lsolve** donne <u>un conditionnement bas</u> donc c'est la méthode à choisir .

-Erreur avant arrière ?

Usolve :l'erreur avant >l'erreur arrière mais avec plus d'écart entre les deux
Lsolve :l'erreur avant >l'erreur arrière mais avec moins d'écart entre les deux
pour n=5 ou n=10
Lsolve est meilleur

-Quelle complexité ?
On a une complexité de O(n) pour les deux fonctions ,car elles ont une seule boucle chacune .

Exercice 5 :

1. Écriture de l'algorithme 12 (diapo 35) de résolution par élimination de Gauss sans pivotage.
   On crée 1 fichier **gausskij3b.sci**

```
function [A, b]=gausskij3b(A, b)
```

```
    n=size(A,1);
    m=eye(n,n);

    for k=1:n-1
        for i=k+1:n
            m(i,k)=A(i,k)/A(k,k);
            b(i)=b(i)-m(i,k)*b(k);
            for j=k+1:n
                A(i,j)=A(i,j)-m(i,k)*A(k,j);
            end
        end

    end
endfunction
```

On voit que l'elimintaiton est d'une puissance de cette eliminitaion est de : $2n^3$

2. Testez et validez votre algorithme sur de petites matrices.

Pour tester/valider il faut utiliser la fonction **usolve()** développée à l'exercice 4 :

```
function x=Usolve(U, b)
    n=size(U,1);
    x=zeros(n,1);
    x(n)=b(n)/U(n,n);
    for i=n-1:-1:1
        x(i)=(b(i)-U(i,(i+1):n)*x(i+1:n))/U(i,i);
    end
endfunction
```

En saisie les valeurs de **testgauss.sci** on a comme résultats :

Pour A=[3,2;4,5]; B=[8;6];
la solution val=[4 ; -2] ;
Ce qui est vérifiée et correct .

Exercice 6 :

1.Écrire l'algorithme 13 (diapo 38) de factorisation LU .
Le fichier **exo6.sci** contient les fonctions, les test (valeurs …) sont le fichier **test_exo6.sci**

```
function A=mylu3b(A)
    n=size(A,1);

    for k=1:n-1
        for i=k+1:n
            A(i,k)=A(i,k)/A(k,k);
        end
          for i=k+1:n
            for j=k+1:n
                A(i,j)=A(i,j)-A(i,k)*A(k,j);
            end le
        end

    end
endfunction
```

2.Testez et validez votre algorithme sur de petites matrices. On calculera l'erreur commise sur la factorisation LU :A−LU avec le script suivant

```
M1=zeros(2,1);
B=[3,2;4,5];
b=[8;6];
U=zeros(2,2);
L=zeros(2,2);
  A=mylu3b(B)
  disp(A)
  U=triu(A);
  disp(U)
  Z=A-U
  L=Z+eye(2,2)
  disp(L)
  M1=inv(L)*b
  disp(M1)
  ereur=B-L*U
disp(ereur,'erreur:')
   x=inv(U)*M1
  disp(x)
```

On obtient une erreur =0 ici donc par apport aux résultats précédents (en l'appliquant pour les mêmes valeurs A et B de la question précédents) ce qui est vrai .mais en donnant des matrices grandes on aura un autre résultat (une erreur considérable).

3. Améliorez l'algorithme 13 (diapo 38) de factorisation LU de sorte à n'obtenir qu'une boucle.

```
function A=mylu3b1(A)

      n=size(A,1);
      for k=1:n-1
                A(k+1:n,k)=A(k+1:n,k)/A(k,k);
                A(k+1:n,k+1:n)=A(k+1:n,k+1:n)-A(k+1:n,k)*A(k,k+1:n);
       end
endfunction
```

4.Ajoutez la méthode de pivot partiel :

```
function A=mylu(A)

      n=size(A,1);
      // on ajoute la partie pivot
      for j=1:n-1
            for i=1:n-1

              A(i,:)=A(i,:)-A(i,j)/A(j,j)*A(j;:)

            end
      end
      // fin parite pivot
      for k=1:n-1

                A(k+1:n,k)=A(k+1:n,k)/A(k,k);
                A(k+1:n,k+1:n)=A(k+1:n,k+1:n)-A(k+1:n,k)*A(k,k+1:n);
      end

endfunction
```

TD Résolution de systèmes linéaires: Méthodes directes

Exercice 1 :

1. Appliquer la méthode de Gauss à la matrice tridiagonale suivante:

$$A = \begin{pmatrix} a_1 & c_1 & 0 & \cdots & 0 & 0 \\ b_1 & a_2 & c_2 & \cdots & 0 & 0 \\ & & & \ddots & & \\ 0 & 0 & 0 & \cdots & a_{n-1} & c_{n-1} \\ 0 & 0 & 0 & \cdots & b_{n-1} & a_n \end{pmatrix}.$$

Annexe :

Exo2 :

```
function exo2(n, ex)
        format("e",ex);
        A = rand(n,n);
        disp("A=",A);
        xex = rand(n,1);
        disp("xex=",xex);
        b = A*xex;
        disp("b=",b);
        x=A\b;
        disp("x=",x);
        frelres = norm(x-xex)/norm(xex); //erreur avant
        disp("frelres=",frelres);
        brelres = norm(b-A*x)/norm(b); //erreur arrière
        disp("brelres=",brelres);
        capa=cond(A);
        disp("cap=",capa);
        borne=cond(A)*brelres;
        disp("borne=",borne);
endfunction
```

Exo3 :

```
function [C]=matmat3b(A, B)
      m=size(A,1);
      n=size(B,2);
      p=size(B,1);

      C=zeros(m,n);
      for i=1:m
            for j=1:n
                  for k=1:p
                              C(i,j)=A(i,k)+B(k,j)+C(i,j);
                  end
             end
      end
endfunction

function [C]=matmat2b(A, B)
        m=size(A,1);
        n=size(B,2);
        p=size(B,1);

        C=zeros(m,n);

        for i=1:m
            for j=1:n
                    C(i,j)=A(i,:)*B(:,j)+C(i,j);
              end
        end

endfunction

function [C]=matmat1b(A, B)
        m=size(A,1);
        n=size(B,2);
        p=size(B,1);
```

```
        C=zeros(m,n);

      for i=1:m
            C(i,:)=A(i,:)*B+C(i,:);
      end
endfunction

function [C]=matmat1bkij(A, B)
          m=size(A,1);
          n=size(B,2);
          p=size(B,1);

          C=zeros(m,n);
          for k=1:p
                C=A(:,k)*B(k,:)+C;
          end

endfunction
```

Exo4 :

```
function x=Usolve(U, b)
      n=size(U,1);
      x=zeros(n,1);
      x(n)=b(n)/U(n,n);
      for i=n-1:-1:1
            x(i)=(b(i)-U(i,(i+1):n)*x(i+1:n))/U(i,i);
      end

endfunction

function x=Lsolve(L, b)
      n=size(L,1);
      x=zeros(n,1);
      x(1)=b(1)/L(1,1);
      for i=2:n
            x(i)=(b(i)-L(i,1:(i-1))*x(1:(i-1)))/L(i,i);
      end

endfunction
```