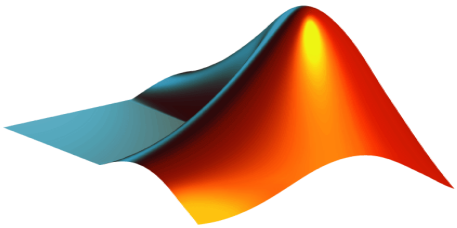
Université EUROMED de Fès
Ecole d'**I**ngénierie **D**igitale et d'**I**ntelligence **A**rtificielle



Rapport de projet Matlab

Résolution des systèmes linéaires Ax = b

Projet réalisé par :

- Khadija BAYOUD
- Achraf RACHID

Projet encadré par :

- Pr. Meryem ELMOUHTADI

EIDIA UEMF
13/06/2021

Sommaire

Introduction

Les systèmes linéaires interviennent à travers leurs applications dans de nombreux domaines : des sciences physiques ou mécaniques, des sciences du vivant, de la chimie, de l'économie, des sciences de l'ingénieur, … car ils forment la base calculatoire de l'algèbre linéaire. Ils permettent également de traiter une bonne partie de la théorie de l'algèbre linéaire en dimension finie.

Le but de ce projet est de réaliser une application, en utilisant Matlab GUI, capable de résoudre des systèmes linéaires.

Matlab GUI (**G**raphical **U**ser **I**nterface) est un outil dédié à la création des interfaces graphiques qui permet au programmeur de créer des interfaces GUI pour mettre à profit les performances du langage Matlab aux utilisateurs qui ne sont pas familiarisés avec ce langage.

- Les objectifs de ce projet sont :
  - La résolution numérique de Ax = b avec différentes méthodes ; directes et itératives.
  - La résolution graphiques de Ax = b
- Les limites de ce projet :
  - On se limite dans ce projet à la résolution graphiques de Ax =b en 2D .

Avant de commencer, nous allons faire un rappel sur les systèmes linéaires.

Notions de bases :

Les systèmes linéaires :

On appelle système linéaire d'ordre n (n entier positif), une expression de la forme Ax = b; Où A = (aij), i ≥ 1 ; j≤n, désigne une matrice de taille n*n (une matrice carrée) de nombres réels ou complexes, et b=(bi),1 ≤ i ≤ n, un vecteur colonne réel ou complexe et x = (xi), 1 ≤ i ≤ n, est le vecteur des inconnues du système.

$$\sum_{j=1}^{n} a_{ij}x_j = b_i, \qquad i = 1, \ldots, n.$$

Si *A* est une **matrice carrée inversible** d'ordre *n (det(A)≠ 0)*, alors le système d'équation dont l'écriture matricielle est *AX = B* admet une **unique solution** : $\boldsymbol{X = A^{-1}B}$ (A^{-1} est l'inverse de A).

Théoriquement, si A est **carrée inversible**, la solution existe et elle est donnée par la formule de Cramer.

$$x_i = \frac{det(A_i)}{det(A)}, \qquad i = 1, \ldots, n,$$

Où Ai est la matrice obtenue en replaçant la i -ème colonne de A par le vecteur b. Cependant l'application de cette formule est inacceptable pour la résolution pratique des systèmes, car elle est couteuse en terme de temps et nombre d'opérations alors nous avons besoins d'utiliser d'autres méthodes afin d'optimiser le temps de la résolution de Ax=b.

- **Matrice symétrique** est une matrice carrée qui est égale à sa propre transposée, c'est-à-dire telle que $a_{i,j} = a_{j,i}$ pour tous *i* et *j* compris entre 1 et *n*, où les $a_{i,j}$ sont les coefficients de la matrice et *n* est son ordre.
- **Matrice définie positive** : Une matrice symétrique A dont les l'éléments sont des nombres réels, est d´définie positive si pour tout vecteur x ∈ R n non nul on a xTAx > 0.
- **Matrice diagonale dominante** : est une matrice carrée à coefficients réels ou complexes dont lorsque le module de chaque terme diagonal est supérieur ou égal à la somme des modules des autres termes de sa ligne. Si $A = (a_{i,j})_{(i,j)\in[1,n]^2}$, alors on a :

$$\forall i \in [\![1, n]\!], \ |a_{i,i}| \geq \sum_{\substack{j=1 \\ j\neq i}}^{n} |a_{i,j}|.$$

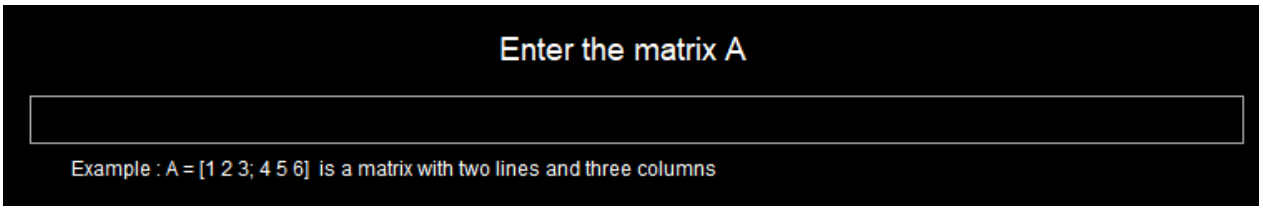
- **Mineur principal** d'ordre k de A désigne le déterminant de la matrice obtenue à partir de A en extrayant les k premières lignes et colonnes.

Présentation d'interface Matlab :

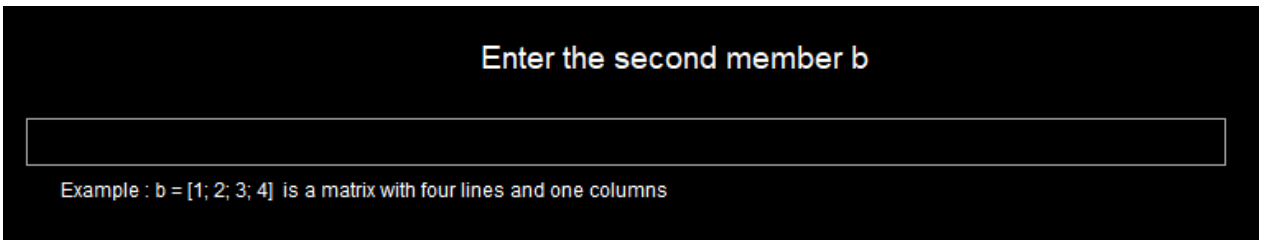
Les items utilisés :

D'abord nous avons commencé par créer un panel qui va contenir tous les items que nous allons utiliser dans la suite.

Puis on a créé 3 items pour la Matrice A ; deux de genre (static text) pour expliquer au utilisateur ce qu'il va faire et une de genre (edit text) où il va entrer sa matrice.



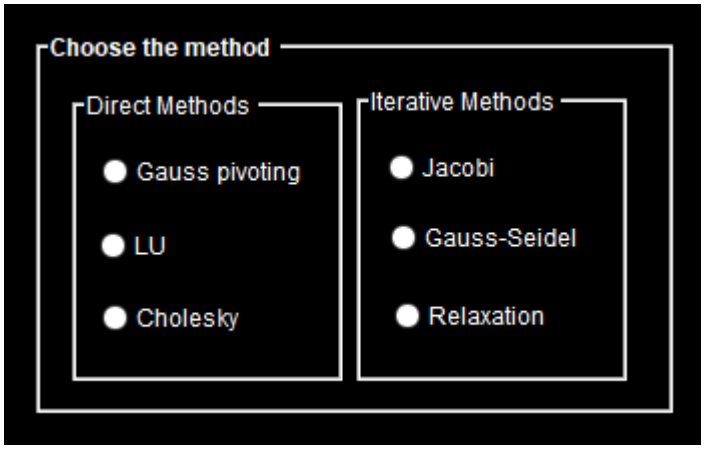
On a répété les mêmes étapes pour l'item dédié à la colonne b.



Ensuite on a créé un panel où nous allons mettre les méthodes à utiliser.

Pour chaque méthode on a utilisé un « radio Button »

Un « radio Button » si elle est cliquée sa valeur vaut 1 sinon elle vaut 0.



Dans chaque callback d'un « radio Button » :

- On associe une variable **val** à un bouton, elle est par défaut à 0 ; Cette variable représente l'état du bouton : cliqué ou non.
  Par exemple dans la fonction callback de gauss pivoting on lui associe une variable nommée **val1** et on récupère l'état du bouton en utilisant la fonction **get** pour savoir si gauss est cliquée ou non. Si oui on va désélectionner les autres boutons et on va rendre les autres vals a 0.
  Pour appliquer l'algorithme de gauss, on doit récupérer la matrice A et la colonne b en utilisant la méthode **get**.

```
% --- Fonction callback du bouton gauss

   function gauss_Callback(hObject, eventdata, handles)

               %Get the value of the button
val1 = get(handles.gauss,'value');

%Check if the button is pressed
if val1 == 1
    set(handles.lu, 'value',0);
    val2 = 0;
    set(handles.cholesky, 'value',0);
    val3 = 0;
    set(handles.jac, 'value',0);
    val4 = 0;
    set(handles.Seidel, 'value',0);
    val5 = 0;
    set(handles.relaxation, 'value',0);
    val6 = 0;
```

```
%Get the matrix A and the vector b

   a = str2num(get(handles.matrixA,'string'));
   [n,m] = size(a);

   b =    str2num(get(handles.vectorb,'string'));
```

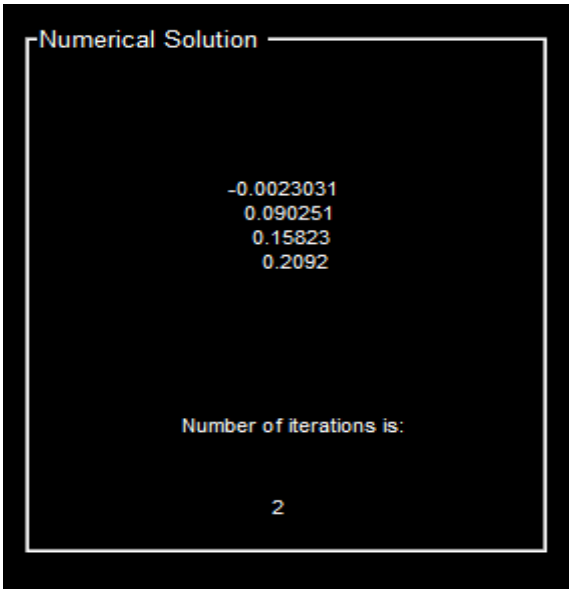
Dans la suite de ce rapport, nous allons voir les exceptions à traiter et l'algorithme de la méthode.

Le même processus va se répéter dans chaque callback des méthodes.

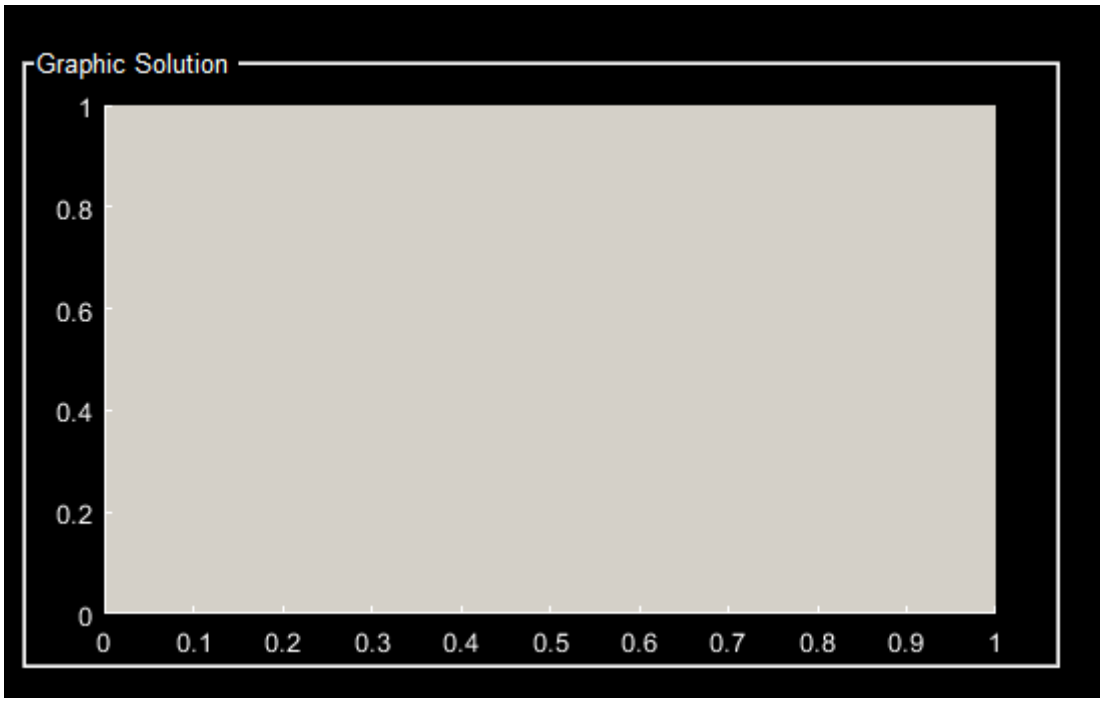
Ensuite on a créé deux régions d'affichage une pour la solution numérique et l'autre pour la solution graphique.

- **L'affichage de la solution numérique :**

On a créé 3 items de type edit text une où nous allons afficher la solution et les autres sont utilisé dans le cas où la méthode utilise est itérative pour afficher le nombre des itérations faites jusqu'à que nous arrivons au résultat.



- **L'affichage de la solution graphique ce fait dans un plan de 2D.**

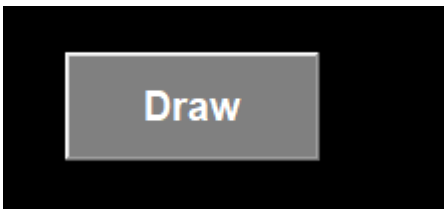


- Button « Resolve »

Il récupère la solution x calculer par une des méthodes et le nombre des itérations si la méthode est itérative puis on les affiche.

```
% --- Executes on button press in resolve.

    function resolve_Callback(hObject, eventdata, handles)

      global itr x val4 val5 val6;

     set(handles.result,'string',num2str(x));

     if  val4 == 1
       set (handles.msg,'string', 'Number of iterations is:');
       set (handles.itr,'string', num2str(itr));

     elseif val5 == 1
        set (handles.msg,'string','Number of iterations is:');
        set (handles.itr,'string', num2str(itr));

     elseif val6 == 1
        set (handles.msg,'string','Number of iterations is:');
        set (handles.itr,'string', num2str(itr));

     end
```

- Button « Draw »



Dont on résoudre graphiquement un système de taille 2x2 par la construction de deux équations des droites qui se rencontrent dans un point qui a comme coordonnées x et y et qui sont en même temps x1 et x2 de la solution qu'on cherche.

```
% --- Executes on button press in Draw.
  function Draw_Callback(hObject, eventdata, handles)

   global a b;
   [n,m] = size(a);
    if n == 2
       y = -3:0.1:3;
      % équation de droite
       x1 = (b(1,1)/a(1,1)) -(a(1,2)/a(1,1))*y;
       x2 = (b(2,1)/a(2,1)) -(a(2,2)/a(2,1)) *y;
       axes (handles.axes1) ;
        % plot des deux droites l'un est de couleur noir et l'autre
rouge
       Plot (y, x1, 'r', y, x2, 'k');
       grid on;
       xlabel('X');
       ylabel('Y');
   else

      error = errordlg ('graphical solution is not available!',
'Note');
   end
```

Si la matrice est de taille plus de 2x2 on ne peut pas afficher la solution graphique alors on affiche à l'utilisateur un message indiquant que la solution graphique n'est pas disponible.

- Button « Clear » : Effacer le contenu de l'interface :

Le contenu de chaque item de l'interface est effacé grâce à la fonction **set :**

```
set(handles.matrixA,'String','');
set(handles.vectorb,'String','');
set(handles.result,'String','');
set(handles.msg,'String','');
set(handles.itr,'String','');
set(handles.gauss, 'value',0);
set(handles.lu, 'value',0);
set(handles.cholesky, 'value',0);
set(handles.jac, 'value',0);
set(handles.Seidel, 'value',0);
set(handles.relaxation, 'value',0);
axes(handles.axes1);
plot(0,0);
clear all;
```

Présentation du programme sous Matlab :

Notre programme se charge de la résolution de Ax=b par des différents méthodes ; directs et itératives :

1. Méthodes directes :

On appelle méthode de résolution directe d'un système linéaire un algorithme qui donnerait la solution en un nombre fini d'opérations.

➢ La méthode du pivot de gauss :

La méthode consiste à rendre la matrice A en une matrice triangulaire supérieure ou inférieure0

Tout d'abord, on va concaténer la matrice **a** et la colonne **b** puisque les opérations que nous allons effectuer sur la matrice « **a** » elles ont aussi touché la colonne « b » . **A=[a b]**

Ensuite, on boucle sur les colonnes et les lignes pour rendre la matrice triangulaire supérieure. Dans cette boucle on fait un teste si le pivot est nul, si oui, on fait une permutation, si un des pivots reste nul après la permutation, la méthode ne marche pas.

```
for i=1:n-1
       for j=i+1:n
           if A(i,i) == 0
               per = 0
               for m=i+1:n
                   if A(m,i)~=0
                       per = m
                       break
                   end
               end
               if per == 0
                   error = errordlg('Non Resovabale','error');
                   set(handles.gauss,'value',0);
               else
                   temp = A(i,:)
                   A(i,:) = A(per,:)
                   A(per,:) = temp
               end
           else
               A(j,:)=A(j,:)-(A(j,i)*A(i,:))/A(i,i);
           end
       end
```

On initialise notre vecteur de solution par des zéros, puis on calcule le n-ième élément de notre solution.

```
x = zeros(n,1);
 x(n) = A(n,n+1)/A(n,n);
```

Dans la suite, on va utiliser le i-ème élément de x pour trouver le i-1 élément de x jusqu'à ce que le vecteur x soit rempli.

```
for i=1:n

A(i,:)=A(i,:)/A(i,i);
   end
   for i=n-1:-1:1

        s=0;
        for k=i+1:n
            s=s+A(i,k)*x(k);
        end
        x(i) = A(i,n+1)-s;
   end
```

➢ La méthode de Cholesky :

La méthode consiste, pour une matrice symétrique définie positive A, à déterminer une matrice triangulaire inférieure L telle que : A=LL'.

Dans notre script, nous allons assurer que la matrice A est une matrice symétrique définie positive sinon une message d'erreur sera affiché.

```
positiveMinor = 1 ;
    for i=1:n
        if det(a(1:i , 1:i))<0
            positiveMinor = 0;
        end
    end
if positiveMinor == 0 || ~isequal(a, a')
       error = errordlg('The matrix is not symetric positive-
definite!','error');
       set(handles.cholesky,'value',0);
end
```

Si la matrice respecte toutes les conditions de la méthode on passe à calculer notre matrice L par le code suivant :

```
L = zeros(n,n);
   for i=1:n
       L(i,i) = sqrt(a(i,i)-L(i,:)*L(i,:)');
       for j=(i+1) : n
           L(j,i) = (a(j,i)-L(i,:)*L(j,:)')/L(i,i);
       end
   end
```

Puis nous calculons la solution x :

```
y=inv(L)*b;
 x=inv(L')*y;
```

➢ La méthode de factorisation LU :

La méthode de décomposition LU exprime la matrice A sous forme de produit d'une matrice triangulaire inférieur L (Lower) à diagonale unité par une matrice supérieur U (Upper), tel que : A = LU . à condition que les mineurs principaux dominants de la matrice A sont non nuls .

- Assurer que la condition est vérifiée :

```
minor = 1 ;
    for i=1:n
        if det(a(1:i , 1:i))==0
            minor = 0;
        end
    end
if minor == 0
       error = errordlg('The matrix doesn't admit LU factorization','error');
       set(handles.lu,'value',0);
end
```

- Décomposer A en L et U : `[L U P] = lu(a);`

Cette ligne de code renvoie la factorisation de la matrice a en L U avec P la matrice de permutation pour rendre la matrice L à diagonal unitaire.

- Trouver la solution :

Le calcul de la solution revient à résoudre les deux systèmes suivants:

$$\begin{cases} y = L\backslash(P*b) \\ x = U\backslash y \end{cases}$$

D'où la solution sur Matlab est :

```
x = U\(L\(P*b));
```

Les méthodes itératives :

➢ La méthode de Jacobi :

Cette méthode consiste à : 1. décomposer A en A=D−E−F, où D est une matrice diagonale (diagonale de A), −E est une matrice triangulaire inférieure de −A, et −F est une matrice triangulaire supérieure de −A.

Le théorème dit que Si A est une matrice à diagonale dominante, alors la suite x(k) converge vers l'unique solution de Ax=b.

Donc on va s'assurer que la matrice saisit par l'utilisateur est une matrice à diagonale dominante .sinon on la rendre à diagonale dominante si il est possible :

```
if ~(all((2*abs(diag(a))) >= sum(abs(a),2)))
          [maxrow,maxind] = max(abs(a),[],2);
          if all(maxrow > (sum(abs(a),2) - maxrow)) &&
isequal(sort(maxind),(1:numel(maxind))')
             a(maxind,:) = a;
             b(maxind,:)= b;
```

si il est impossible de rendre la matrice à diagonale dominante on affiche un message :

```
else
errordlg('The matrix cannot be diagonally dominant!','error');

set(handles.jac,'value',0);
```

Si la matrice respecte toutes les conditions de la méthode, on initialise le vecteur de la solution à zéro. Puis, on demande à l'utilisateur d'entrer la tolérance qui représente la précision de la solution.

```
x=zeros(n,1);
normVal=Inf;
    enterTol = inputdlg({'Enter the tolerance value.
example:10^-2'},'Tolerance');
```

Puis, on boucle sur les lignes pour calculer la solution en utilisant le précèdent **x** jusqu'à ce qu'on atteint la bonne tolérance.

```
while normVal>tol
        x_old=x;
       for i=1:n
         sigma=0;
         for j=1:n
            if j~=i
                sigma=sigma+a(i,j)*x(j);
            end
         end
         x(i)=(1/a(i,i))*(b(i)-sigma);
       end
       itr=itr+1;
       normVal=abs(xold-x);
 end
```

➢ La méthode de Gauss-Seidel :

La méthode de Gauss-Seidel correspond à la décomposition M = D - E (triangulaire inférieure) et N = F, tel que D est une matrice diagonale contenant la diagonale de A, E est la partie triangulaire inférieure stricte (sans la diagonale) de -A et F est la partie triangulaire supérieure stricte de -A.

L'algorithme s'écrit alors :

X(0) donné
(D - E)X(k+1) = FX(k) + b

On suppose que aii ≠ 0; 1 ≤ i ≤ n, alors la matrice L1 =inv((D - E))*F est la matrice de Gauss-Seidel. Autrement dit, pour k ≥ 1 on calcule les composantes de l'itérée X(k+1) à partir des itérées de X(k).

$$x_i^{(k+1)} = \frac{1}{a_{ii}} \left[-\sum_{j=1}^{i-1} (a_{ij} x_j^{(k+1)}) - \sum_{j=i+1}^{n} (a_{ij} x_j^{(k)}) + b_i \right].$$

Condition : Cette méthode converge vers la solution si la matrice A est à diagonale dominante.

- Le code pour assurer la condition :

```
%Check if the matrix is diagonaly dominant
   elseif ~(all((2*abs(diag(a))) >= sum(abs(a),2)))
          [maxrow,maxind] = max(abs(a),[],2);
        if all(maxrow > (sum(abs(a),2) - maxrow)) &&
isequal(sort(maxind),(1:numel(maxind))')
           a(maxind,:) = a;
           b(maxind,:)= b;
else
errordlg('The matrix cannot be diagonally dominant!','error');
```

- Initialiser le vecteur de solution par 0 et demander à l'utilisateur de déterminer la tolérance :

```
x = zeros(n,1);

enterTol = inputdlg({'Enter the tolerance value. example:10^-2'},'Tolerance');
tol = str2num(enterTol{1});
```

- Lignes de codes pour l'algorithme expliqué ci-dessus :

```
normVal=Inf;
itr=0;
x_old=x;
      for i=1:n
          sigma=0;
          for j=1:i-1
                sigma=sigma+a(i,j)*x(j);
          end
          for j=i+1:n
                sigma=sigma+a(i,j)*x_old(j);
          end
          x(i)=(1/a(i,i))*(b(i)-sigma);
      end
      itr=itr+1;
      normVal=norm(x_old-x);
```

➢ La méthode de relaxation :

Cette méthode consiste en la manipulation suivante : on décompose A comme A=D−E−F, où D est une matrice diagonale (diagonale de A), −E est une matrice triangulaire inférieure de −A , et −F est une matrice triangulaire supérieure −A.

Soit aussi ω un nombre réel non nul, appelé paramètre de la méthode de relaxation.

La méthode de relaxation consiste à écrire le système sous la forme.

$$\begin{aligned} Ax = b &\iff \left(\frac{1}{\omega}D - E\right)x + \left(\frac{\omega-1}{\omega}D - F\right)x = b \\ &\iff \left(\frac{1}{\omega}D - E\right)x = \left(\frac{1-\omega}{\omega}D + F\right)x + b. \end{aligned}$$

Puis à définir une suite de vecteurs X(k) par la formule :

$$\left(\frac{1}{\omega}D - E\right)x^{k+1} = \left(\frac{1-\omega}{\omega}D + F\right)x^{k} + b.$$

On espère alors que la suite X(k) converge vers une solution de Ax=b. Sous de bonnes hypothèses concernant la matrice A et le réel ω, c'est effectivement le cas. Par exemple, si A est symétrique et définie positive, et si 0<ω<2, alors la suite X(k) converge effectivement vers l'unique solution de Ax=b. Lorsque ω=1, on retrouve la méthode de Gauss-Seidel.

- Rassurer que symétrique définie positive:

```
positiveMinor = 1 ;
    for i=1:n
        if det(a(1:i , 1:i))<0
            positiveMinor = 0;
        end
if positiveMinor == 0 || ~isequal(a, a')

    errordlg('The matrix is not symetric positive-definite!','error');
       set(handles.relaxation,'value',0);
end
```

- Demander à l'utilisateur d'entrer le paramètre ω et la tolérance en rassurant qu'il est entre 0 et 2

```
input = inputdlg({'Enter the tolerance value. example:10^-2'},{'Enter the parameter
<w>'},'Input');;
          tol = str2num(input{1});
          w = str2num(input{2});
          if w<=0 || w>=2
              error = errordlg('The parameter w must be between 0 and 2 exclusive!','error');
              set(handles.relaxation,'value',0);
          end
```

- L'algorithme de relaxation expliqué ci-dessus :

```
A =[a b];
X = zeros(n,1);

% Limiter le nombre d'itération en 86 itérations
          m = 86;
          itr = 1;
          while  itr <= m
              err = 0;
              for i = 1 : n
                  s = 0;
                  for j = 1 : n
                      s = s-a(i,j)*x(j);
                  end
                  s = w*(s+a(i,n+1))/a(i,i);
                  if abs(s) > err
                     err = abs(s);
                  end
                  x(i) = x(i)+s;
              end

              if err <= tol
                 break;
              else
                 itr = itr+1;
              end
         end
```

Gestion des exceptions :

Si la matrice A n'est pas carrée, n'est pas inversible, ou il ne respecte pas les conditions d'une méthode, le programme donne des résultats indésirables par exemple **NaN** (Not a Number) ou **Inf** (infinie).

Pour éviter ceci, on s'assure au début que la matrice A respecte les conditions suivantes :

- **Matrice carrée :**

La matrice A doit être une matrice carrée :

```
[n,m] = size(a);
if n ~= m
      error = errordlg('The matrix must be a square matrix !','error');
      set(handles.gauss,'value',0);
end
```

- **Matrice inversible :**

La matrice A doit être inversible :

```
if det(a) == 0
      error = errordlg('The matrix is not Invertible !','error');
      set(handles.gauss,'value',0);
end
```

- **Matrice à diagonale dominante :**

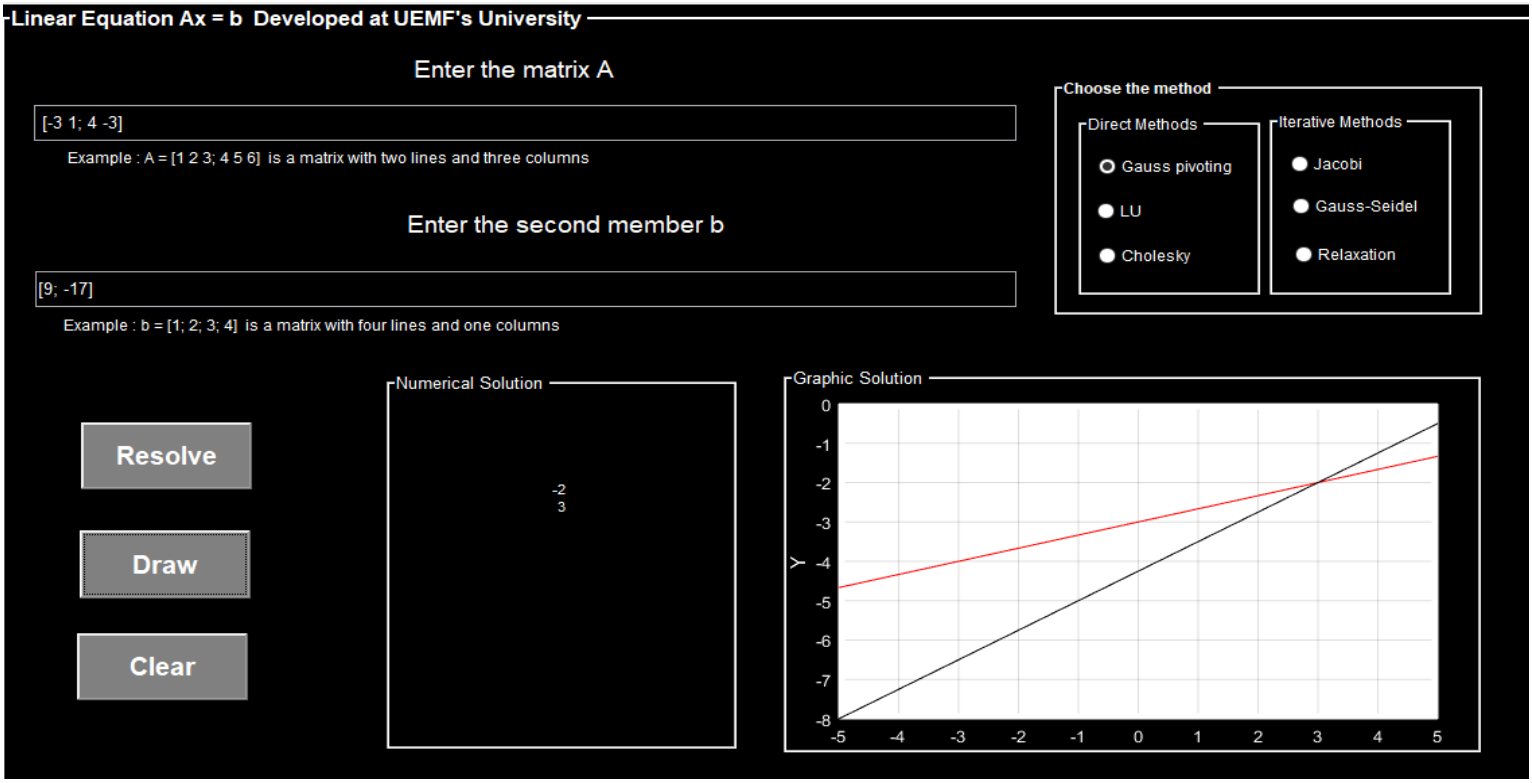
Les deux méthodes Jacobi et Gauss-Seidel convergent si la matrice est à diagonale dominante (La réciproque n'est pas vraie).

```
if ~(all((2*abs(diag(a))) < sum(abs(a),2)))
       while(1)
            if all((2*abs(diag(a))) < sum(abs(a),2))
               break;
            else
               A = A(randperm(size(A, 1)), :);
            end
       end
end
```

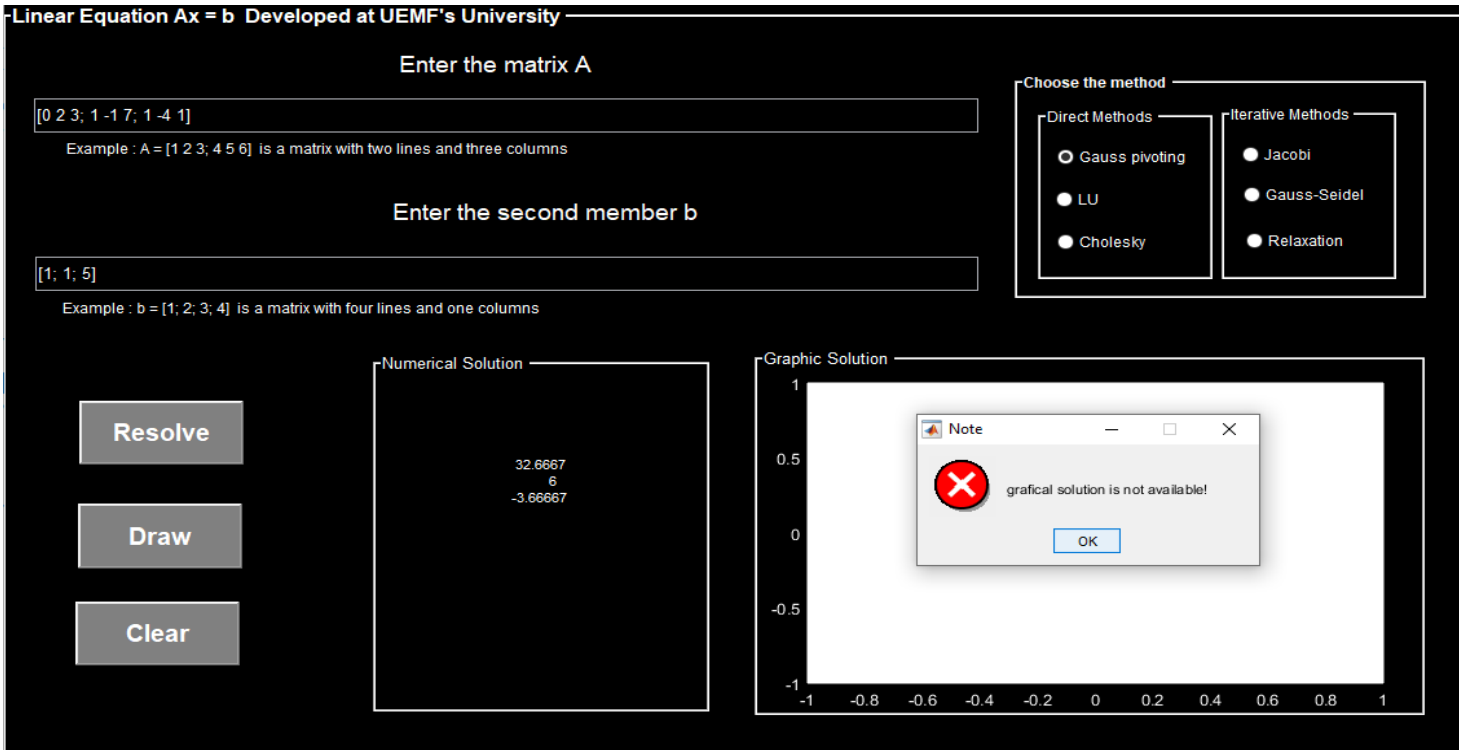
Test du programme sous Matlab :

On teste notre programme par les systèmes linéaires suivantes :

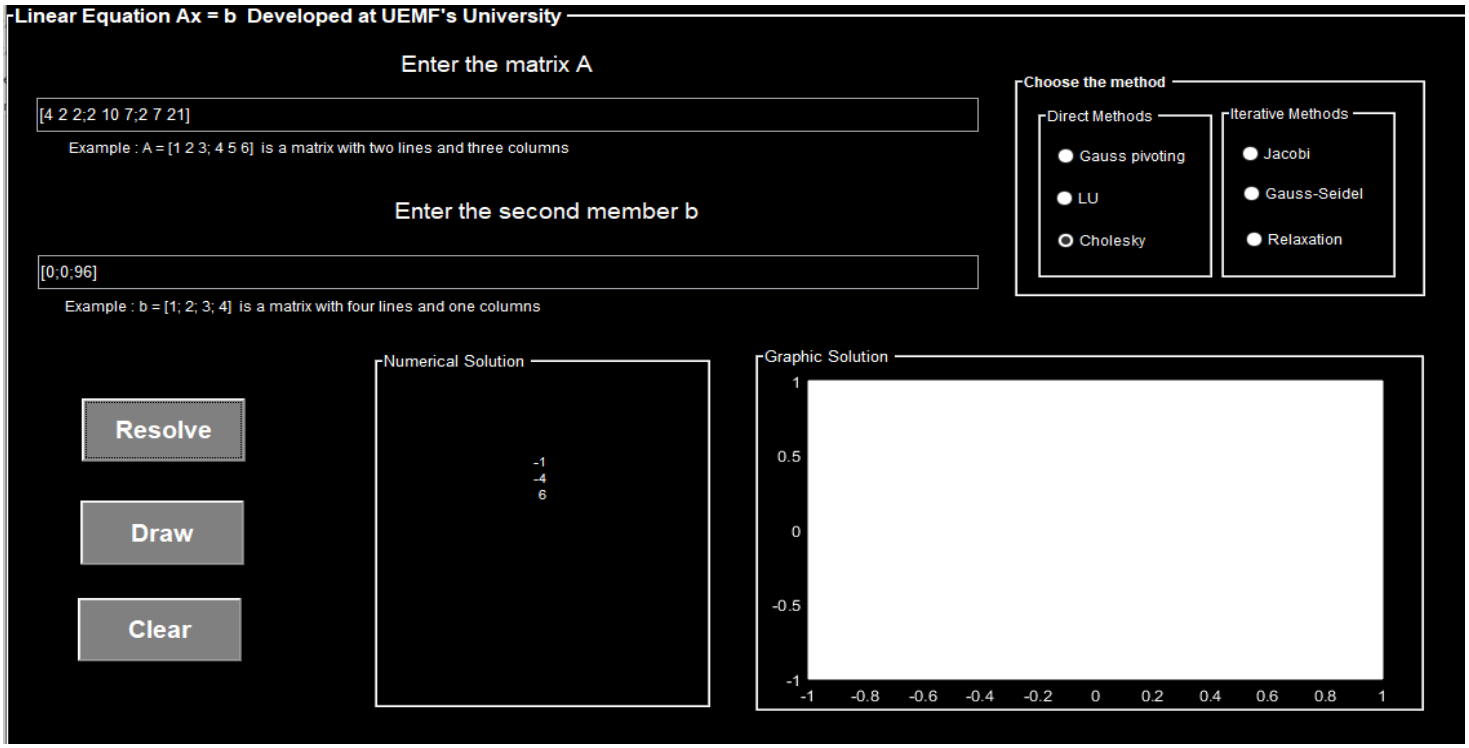
- Matrice 2x2 avec la méthode de Gauss avec résolution graphique.
  - A = [-3 1; 4 -3]
  - B = [9; -17]
  - X = [-2; 3]



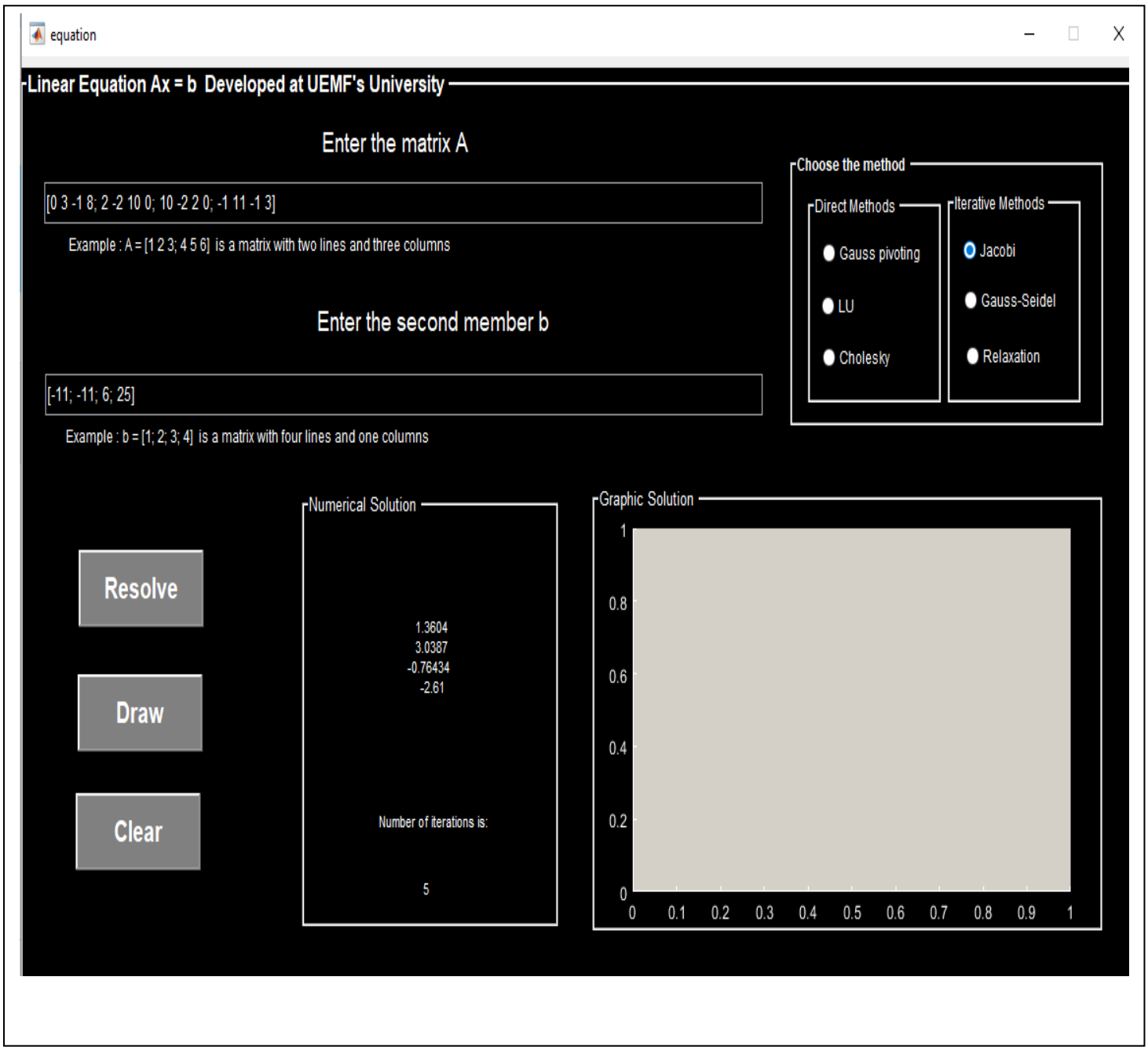
- Matrice 3x3 avec la méthode de Gauss, cas de pivot nul.
  - A = [0 2 3; 1 -1 7; 1 -4 1]
  - B = [1; 1; 5]
  - X = [32.6667; 6.0000; -3.6667]



- Matrice 3x3 avec la méthode de Cholesky.
  - A = [4 2 2;2 10 7;2 7 21]
  - B = [0;0;96]
  - X = [-1 ; -4 ; 6]



- Matrice 3x3 avec la méthode de Jacobi cas de diagonale non-dominante.
  - A = [1 -1 -2; -1 5 -4; 2 -4 6]
  - B = [1;1;2]
  - X = [1.3604 ; 3.0387; -0.76434 ;-2.61]



Bibliographie :

- Cours Langage Matlab :
  Pr.Meryem Elmouhtadi
  UEMF
- Cours Analyse Numérique :
  Pr.Safae Elhaj-ben-ali
  UEMF

- MathWorks : **https://ch.mathworks.com/fr/products/matlab.html**

- Youtube : **https://www.youtube.com/**

Conclusion :

Ce projet s'est révélé très enrichissant dans la mesure où il a consisté en une approche concrète du métier d'ingénieur. En effet, la prise d'initiative, le respect des délais et le travail en équipe seront des aspects essentiels de notre futur métier.

De plus, il nous a permis d'appliquer et enrichir nos connaissances dans le langage Matlab.

Les principaux problèmes, que nous avons rencontrés, concernaient l'implémentation des méthodes de résolution des systèmes linéaires, plus particulièrement la résolution graphique et les cas exceptionnels des méthodes de résolution.

Enfin nous ne prétendons pas avoir résoudre le problème dans son intégralité mais nous sommes par ailleurs convaincus que le travail élaboré n'est qu'une étape primaire pour des études plus approfondies.