V.12. Exercices d'application sous Matlab

Exercice V.12.1 : (Méthode d'élimination de Gauss)

Résoudre le système linéaire suivant en utilisant la méthode d'élimination de Gauss :

$$\begin{cases} 2x & +4y & -6z = & 4 \\ 4x & +4y & -7z = & 3 \\ 2x & -y & +2z = & -2 \end{cases}$$

Solution :

Le système peut être écrit sous forme matricielle Ax = b avec :

Le code comporte deux étapes principales :

1- Élimination de Gauss : il s'agit de transformer la matrice A en une matrice triangulaire supérieure en utilisant des opérations élémentaires sur les lignes. Pour cela, on boucle sur les colonnes de la matrice A (ou les lignes de la matrice triangulaire supérieure obtenue), et on utilise un facteur qui permet d'éliminer les éléments en dessous de la diagonale. On effectue la même opération sur le vecteur b.

2- Rétro-substitution : il s'agit de résoudre le système d'équations linéaires à partir de la matrice triangulaire supérieure obtenue à l'étape précédente. Pour cela, on boucle sur les lignes de la matrice, en partant de la dernière ligne, et on utilise les valeurs déjà calculées de x pour déterminer la valeur de la variable courante.

```
% Définition du système d'équations linéaires
A = [2, 4, -6; 4, 3, -7; 2, -1, 2];
b = [4; 3; -2];

% Étape 1 : Élimination de Gauss
n = length(b);
for k = 1:n-1
    for i = k+1:n
        factor = A(i,k) / A(k,k);
        A(i,k:n) = A(i,k:n) - factor * A(k,k:n);
        b(i) = b(i) - factor * b(k);
    end
end
```

```
% Étape 2 : Rétro-substitution
x = zeros(n,1);
x(n) = b(n) / A(n,n);
for k = n-1:-1:1
    x(k) = (b(k) - A(k,k+1:n)*x(k+1:n)) / A(k,k);
end

% Affichage de la solution
disp('La solution du système est :')
disp(x)
```

Enfin, le code affiche la solution du système dans le command window :

```
La solution du système est :
   -0.3333
    0.6667
   -0.3333
```

Exercice V.12.2 : (Décomposition LU)

Résoudre le système linéaire suivant en utilisant la méthode de Factorisation LU :

Solution

Le système peut être écrit sous forme matricielle Ax = b avec :

Le code Matlab correspondant est le suivant :

```
% Définition du système d'équations linéaires
A = [2, 4, -6; 4, 3, -7; 2, -1, 2];
b = [4; 3; -2];

% Étape 1 : Factorisation LU
n = length(b);
L = eye(n);
U = A;
for k = 1:n-1
    for i = k+1:n
        factor = U(i,k) / U(k,k);
        L(i,k) = factor;
        U(i,k:n) = U(i,k:n) - factor * U(k,k:n);
    end
end

% Étape 2 : Résolution des systèmes triangulaires
y = zeros(n,1);
y(1) = b(1) / L(1,1);
for k = 2:n
    y(k) = (b(k) - L(k,1:k-1)*y(1:k-1)) / L(k,k);
end
```

```
x = zeros(n,1);
x(n) = y(n) / U(n,n);
for k = n-1:-1:1
    x(k) = (y(k) - U(k,k+1:n)*x(k+1:n)) / U(k,k);
end

% Affichage de la solution
disp('La solution du système est :')
disp(x)
```

Exercice V.12.3 : (Méthode de Cholesky)

Ecrire un script qui va vous permettre de trouver la décomposition de Cholesky de la matrice définie positive suivante : $A = \begin{pmatrix} 4 & 2 & -1 \\ 2 & 5 & 3 \\ -1 & 3 & 9 \end{pmatrix}$

Solution :

Nous avons définir une matrice symétrique définie positive A, puis nous avons initialisé une matrice L de la même taille que A avec des zéros. Nous avons ensuite implémenté la décomposition de Cholesky en utilisant une boucle pour parcourir chaque élément de L et en calculant les éléments diagonaux et hors diagonale de L en fonction de A.

Enfin, nous avons vérifié la décomposition en reconstruisant A à partir de L et en affichant les matrices L et A_reconstructed.

```
% Définir la matrice symétrique définie positive A
A = [4 2 -1; 2 5 3; -1 3 9];
% Initialiser la matrice L
L = zeros(size(A));
% Calculer la décomposition de Cholesky
for i = 1:size(A,1)
    L(i,i) = sqrt(A(i,i) - sum(L(i,1:i-1).^2));
    for j = i+1:size(A,1)
        L(j,i) = (A(j,i) - sum(L(j,1:i-1).*L(i,1:i-1))) /
L(i,i);
    end
end
% Vérifier la décomposition en reconstruisant A
A_reconstructed = L*L'
% Afficher les résultats
disp('La matrice L de la décomposition de Cholesky est:')
disp(L)
disp('La matrice A reconstruite est:')
disp(A_reconstructed)
```

Exercice V.12.4 : (Méthode de Jacobi)

Soit le système linéaire suivant : $\begin{cases} 3x & +y & -2z = & 1 \\ -2x & +4y & +z = & 4 \\ x & +2y & +5z = & 7 \end{cases}$

Appliquez la méthode de Jacobi pour trouver la solution, avec une précision de $\varepsilon = 10^{-6}$ et en partant du vecteur initial (0, 0, 0).

Solution :

La méthode de Jacobi consiste à réécrire la matrice A sous la forme A = D - L - U, où D est la matrice diagonale de A, L est la matrice triangulaire inférieure de A (avec des zéros sur la diagonale), et U est la matrice triangulaire supérieure de A (avec des zéros sur la diagonale). Nous pouvons alors écrire le système sous la forme : $x^{(k+1)} = D^{-1}(L + U)x^{(k)} + D^{-1}b$, où $x^{(k)}$ est une approximation de la solution à l'étape k. En Matlab, nous pouvons écrire le code suivant :

```
% Définition de la matrice A et du vecteur b
A = [3 1 -2; -2 4 1; 1 2 5];
b = [1; 4; 7];

% Initialisation des variables
x = [0; 0; 0];
x_prev = [0; 0; 0];
tol = 1e-6;
err = inf;
k = 0;

% Calcul des matrices D, L et U
D = diag(diag(A));
L = -tril(A,-1);
U = -triu(A,1);

% Boucle de Jacobi
while err > tol
    x_prev = x;
    x = D \ ((L+U)*x + b);
    err = norm(x - x_prev);
    k = k + 1;
end

% Affichage de la solution
disp(['La solution est : x = ', num2str(x(1)), ', y = ',
num2str(x(2)), ', z = ', num2str(x(3))])
```

Le résultat affiché est :

```
La solution est : x = 0.5555, y = 1.0617, z = 0.8642
Nombre d'itérations : 29
```

Exercice V.12.5 :

Résoudre le système linéaire suivant à l'aide de la méthode de Gauss-Seidel :

$$\begin{cases} 3x & +y & +z = & 10 \\ x & +4\,y & +z = & 12 \\ 2x & +3y & +8z = & 0 \end{cases}$$

Le vecteur initial est : $x^{(0)} = (0, 0, 0)$ et le critère d'arrêt est fixé à : $\varepsilon = 10^{-6}$

Solution :

Nous pouvons réécrire le système sous forme matricielle : Ax = b, où

```
A = [3 1 1; 1 4 1; 2 3 8]
x = [x; y; z]
b = [10; 12; 0]
```

La méthode de Gauss-Seidel consiste à résoudre itérativement le système Ax = b en utilisant la formule de mise à jour suivante :

```
x(i+1) = D^(-1) * (b - (L + U) * x(i))
```

Où D est la matrice diagonale de A, L est la partie triangulaire inférieure de A, et U est la partie triangulaire supérieure de A.

Le nombre d'itérations nécessaire pour atteindre une solution précise dépend de la convergence de la méthode. Dans cet exemple, nous allons fixer le nombre d'itérations à 50.

Dans ce code, nous allons définir la matrice A et le vecteur b, ainsi que les variables nécessaires pour la méthode de Gauss-Seidel (valeurs initiales, nombre maximal d'itérations et tolérance). Nous avons ensuite appliqué la méthode de Gauss-Seidel en utilisant deux boucles for : une pour itérer sur les valeurs de x, et une pour itérer sur le nombre d'itérations.

La méthode de Gauss-Seidel converge vers une solution lorsque la matrice A est à diagonale dominante.

```
% Initialiser les variables
x0 = [0; 0; 0]; % vecteur des valeurs initiales
x = x0; % vecteur des valeurs courantes
n = length(x0); % nombre d'inconnues
iter_max = 50; % nombre maximal d'itérations
tol = 1e-6; % tolérance

% Appliquer la méthode de Gauss-Seidel
for k = 1:iter_max
    for i = 1:n
        x(i) = (b(i) - A(i,1:i-1)*x(1:i-1) -
A(i,i+1:n)*x0(i+1:n)) / A(i,i);
    end
    if norm(x - x0) < tol
        fprintf('Solution trouvée après %d itérations:\n', k);
        disp(x);
        break;
    end
    x0 = x;
end
```