# Solving System of Linear Equation using MATLAB

## Introduction

We will be discussing the topic of solving systems of linear equations using MATLAB. Linear equations are a fundamental topic in mathematics, science, and engineering. They have a wide range of applications in various fields, including physics, chemistry, economics, and finance. Solving systems of linear equations can be a tedious task, especially when dealing with large matrices. However, with the use of MATLAB, we can solve these systems quickly and efficiently. MATLAB is a powerful tool that allows us to perform numerical computations, analyze data, and visualize results. It has built-in functions and tools that can be used to solve various types of linear systems of equations. In this presentation, we will discuss the basics of linear equations and their solutions. We will also explore the various methods used to solve linear systems of equations and how MATLAB can be used to perform these computations. We will cover topics such as matrix operations, linear system solvers, and visualization techniques.

**Contents**

# 1.Gauss Elimination Method

In mathematics, **Gaussian elimination**, also known as **row reduction**, is an algorithm for solving systems of linear equations. It consists of a sequence of operations performed on the corresponding matrix of coefficients. This method can also be used to compute the rank of a matrix, the determinant of a square matrix, and the inverse of an invertible matrix. The method is named after Carl Friedrich Gauss (1777–1855) although some special cases of the method—albeit presented without proof—were known to Chinese mathematicians as early as circa 179 AD.[1]

```
n = input('Please Enter the size of the equation system  n =   ') ;
C = input('Please Enter the elements of the Matrix C ' ) ;
b = input('Please Enter the elements of the Matrix b ' ) ;
dett = det(C);
if dett == 0
    fprintf('This system unsolvable because det(C) = 0 ')
else
    fprintf('The Matrix form is \n')
```

```
disp(C)
b = b';
    fprintf('The corresponding Cx=b, b values are')
disp(b)
    fprintf('The Augmented Matrix form is')
A = [ C  b ]
for j = 1:(n-1)
    %for the diagonal element
        for i= (j+1) : n
            %For the ratio
            mult = A(i,j)/A(j,j) ;
            %for row-echelon form
            for k= j:n+1
                A(i,k) = A(i,k) - mult*A(j,k) ;
                A
            end
        end
end
fprintf('The upper Triangular Matrix form is');
A
%this is nth element value
x(n) = A(n,n+1)/A(n,n);
%for back-substituition
    for i = n-1 : -1 : 1
     sm = 0;
    for j= i+1 : n
        sm = sm +A (i , j)*x(j);
    end
        x(i) = (A(i,n+1)-sm)/A(i,i);
    end
end
```

```
The Matrix form is
     1    -1     1
    -3     2    -3
     2    -5     4
The corresponding Cx=b, b values are
     1
    -6
     5
The Augmented Matrix form is
A = 3×4
     1    -1     1     1
    -3     2    -3    -6
     2    -5     4     5
A = 3×4
     1    -1     1     1
     0     2    -3    -6
     2    -5     4     5
A = 3×4
     1    -1     1     1
     0    -1    -3    -6
     2    -5     4     5
A = 3×4
     1    -1     1     1
     0    -1     0    -6
     2    -5     4     5
A = 3×4
```

```
     1    -1     1     1
     0    -1     0    -3
     2    -5     4     5
A = 3×4
     1    -1     1     1
     0    -1     0    -3
     0    -5     4     5
A = 3×4
     1    -1     1     1
     0    -1     0    -3
     0    -3     4     5
A = 3×4
     1    -1     1     1
     0    -1     0    -3
     0    -3     2     5
A = 3×4
     1    -1     1     1
     0    -1     0    -3
     0    -3     2     3
A = 3×4
     1    -1     1     1
     0    -1     0    -3
     0     0     2     3
A = 3×4
     1    -1     1     1
     0    -1     0    -3
     0     0     2     3
A = 3×4
     1    -1     1     1
     0    -1     0    -3
     0     0     2    12
The upper Triangular Matrix form is
A = 3×4
     1    -1     1     1
     0    -1     0    -3
     0     0     2    12
```

```
fprintf('Solutions Are from the Gauss Elimination Method');
```

```
Solutions Are from the Gauss Elimination Method
```

```
disp(x)
```

```
    -2     3     6
```

## 2.Gauss-Jordan Elimination Method

auss-Jordan elimination is a method used in linear algebra to solve systems of linear equations and to find the inverse of a matrix. The method involves performing a series of row operations on an augmented matrix, which is a matrix consisting of the coefficients of the variables in the equations and the constants on the right-hand side.

The row operations include swapping rows, multiplying a row by a non-zero scalar, and adding a multiple of one row to another row. The goal of these operations is to transform the matrix into a form called the reduced row echelon form, which has the following properties:

1. The first non-zero element (known as the pivot) of each row is 1.

2. The pivots are located in columns that are to the right of the pivots of the rows above them.
3. All other elements in the pivot columns are zero.

Once the matrix is in reduced row echelon form, the solution to the system of linear equations can be easily read off from the last column of the matrix. If the matrix has a row of zeros, it indicates that there are infinitely many solutions to the system of equations. If a row has a pivot in the last column, it indicates that the system has no solution.

Gauss-Jordan elimination is particularly useful for finding the inverse of a matrix, as the inverse can be found by augmenting the matrix with an identity matrix and applying the same row operations until the left-hand side becomes the identity matrix.

```
n = input('Please Enter the size of the equation system  n =   ') ;
C = input('Please Enter the elements of the Matrix C ' ) ;
b = input('Please Enter the elements of the Matrix b ' ) ;
dett = det(C);
if dett == 0
    fprintf('This system unsolvable because det(C) = 0 ')
else
    fprintf('The Matrix form is \n')
disp(C)
b = b';
    fprintf('The corresponding Cx=b, b values are \n')
disp(b)
    fprintf('The Augmented Matrix form is \n')
A = [ C  b ] ;
disp(A)
for j = 1:n
    A( j , : ) = A( j , : )/A( j , j ) ;
    %for the diagonal element
        for i= 1 : n
            if( i ~=  j )
            %For the ratio
            mult = A( i , j ) ;
            %for row-echelon form
              A( i , : ) = A( i , : ) - mult * A( j , : )
            end
        end
end
fprintf('The Diagonal Matrix form is \n');
disp(A);
%this is nth element value
x(n) = A(n,n+1)/A(n,n);
%for back-substituition
    for i = 1 : n
      x(i) = A(i,n+1);
    end
end
```

```
The Matrix form is
```

```
     1    -1     1
    -3     2    -3
     2    -5     4
The corresponding Cx=b, b values are
     1
    -6
     5
The Augmented Matrix form is
     1    -1     1     1
    -3     2    -3    -6
     2    -5     4     5
A = 3×4
     1    -1     1     1
     0    -1     0    -3
     2    -5     4     5
A = 3×4
     1    -1     1     1
     0    -1     0    -3
     0    -3     2     3
A = 3×4
     1     0     1     4
     0     1     0     3
     0    -3     2     3
A = 3×4
     1     0     1     4
     0     1     0     3
     0     0     2    12
A = 3×4
     1     0     0    -2
     0     1     0     3
     0     0     1     6
A = 3×4
     1     0     0    -2
     0     1     0     3
     0     0     1     6
The Diagonal Matrix form is
     1     0     0    -2
     0     1     0     3
     0     0     1     6
```

```
fprintf('Solutions Are from the Gauss-Jordan Elimination Method');
```

```
Solutions Are from the Gauss-Jordan Elimination Method
```

```
disp(x)
```

```
    -2     3     6
```

## 3.Gauss-Jacobi Method

In numerical linear algebra, the **Jacobi method** (a.k.a. the **Jacobi iteration method**) is an iterative algorithm for determining the solutions of a strictly diagonally dominant system of linear equations. Each diagonal element is solved for, and an approximate value is plugged in. The process is then iterated until it converges. This algorithm is a stripped-down version of the Jacobi transformation method of matrix diagonalization. The method is named after Carl Gustav Jacob Jacobi.

```
% enter the coefficient matrix and RHS of the system
```

```
a = input('Enter the coefficinet matrix: ');
b = input('Enter the RHS of the system: ');
n = length(b);
it = input('Enter the iteration you want \n');

% checking if the system is valid
ok = 1;
for i = 1: n
    check_sum = 0;
    for j = 1: n
        if i ~= j
            check_sum = check_sum + a(i, j);
        end
    end
    if check_sum >= a(i, i)
        ok = 0;
        break;
    end
end

% setting up tolerance and maximum number of iterations
tol = 1e-3;


% setting up the initial guess for the solution
x0 = zeros(1, n);

% performing Jacobi iteration method to find solutions for the system
if ok == 0
    fprintf('The system is not valid to solve with this method!!!\n');
else
    x = x0;
    for k = 1: it
        for i = 1: n
            sum = 0;
            for j = 1: n
                if i ~= j
                    sum = sum + a(i, j) * x0(j);
                end
            end
            x0(i) = (b(i) - sum) / a(i, i)
        end

        % checking convergence
        if (norm(x - x0) )< tol
            break;
        end
    end
    x = x0;
end
```

```
x0 = 1×3
    3.1481         0         0
x0 = 1×3
```

```
    3.1481    3.5407         0
x0 = 1×3
    3.1481    3.5407    1.9132
x0 = 1×3
    2.4322    3.5407    1.9132
x0 = 1×3
    2.4322    3.5720    1.9132
x0 = 1×3
    2.4322    3.5720    1.9258
x0 = 1×3
    2.4257    3.5720    1.9258
x0 = 1×3
    2.4257    3.5729    1.9258
x0 = 1×3
    2.4257    3.5729    1.9260
x0 = 1×3
    2.4255    3.5729    1.9260
x0 = 1×3
    2.4255    3.5730    1.9260
x0 = 1×3
    2.4255    3.5730    1.9260
x0 = 1×3
    2.4255    3.5730    1.9260
x0 = 1×3
    2.4255    3.5730    1.9260
x0 = 1×3
    2.4255    3.5730    1.9260
```

```
% printing the solutions
fprintf('The solutins of the system are in Jacobi Method: \n');
```

```
The solutins of the system are in Jacobi Method:
```

```
disp(x);
```

```
    2.4255    3.5730    1.9260
```

## 4.Gauss-Seidal Method

In numerical linear algebra, the **Gauss–Seidel method**, also known as the **Liebmann method** or the **method of successive displacement**, is an iterative method used to solve a system of linear equations. It is named after the German mathematicians Carl Friedrich Gauss and Philipp Ludwig von Seidel, and is similar to the Jacobi method. Though it can be applied to any matrix with non-zero elements on the diagonals, convergence is only guaranteed if the matrix is either strictly diagonally dominant,[1] or symmetric and positive definite. It was only mentioned in a private letter from Gauss to his student Gerling in 1823.[2] A publication was not delivered before 1874 by Seidel.

The Gauss–Seidel method is an iterative technique for solving a square system of *n* linear equations. Let **Ax=b** be a square system of *n* linear equations

```
% enter the coefficient matrix and RHS of the system
a = input('Enter the coefficinet matrix: ');
b = input('Enter the RHS of the system: ');
```

```
n = length(b);
it = input('Enter the iteration you want \n');

% checking if the system is valid
ok = 1;
for i = 1: n
    check_sum = 0;
    for j = 1: n
        if i ~= j
            check_sum = check_sum + a(i, j);
        end
    end
    if check_sum >= a(i, i)
        ok = 0;
        break;
    end
end

% setting up tolerance and maximum number of iterations
tol = 1e-6;

% setting up the initial guess for the solution
x0 = zeros(1, n);

% performing Gauss-Seidal iteration method to find solutions for the system
if ok == 0
    fprintf('The system is not valid to solve with this method!!!\n');
else
    x = x0;
    for k = 1: it
        for i = 1: n
            sum = 0;
            for j = 1: n
                if i ~= j
                    sum = sum + a(i, j) * x(j);
                end
            end
            x(i) = (b(i) - sum) / a(i, i)
        end

        % checking convergence
        if norm(x - x0) <  tol
            break;
        end
        x0 = x;
    end
end
```

```
x = 1×3
    3.1481         0         0
x = 1×3
    3.1481    3.5407         0
x = 1×3
    3.1481    3.5407    1.9132
x = 1×3
```

```
    2.4322    3.5407    1.9132
x = 1×3
    2.4322    3.5720    1.9132
x = 1×3
    2.4322    3.5720    1.9258
x = 1×3
    2.4257    3.5720    1.9258
x = 1×3
    2.4257    3.5729    1.9258
x = 1×3
    2.4257    3.5729    1.9260
x = 1×3
    2.4255    3.5729    1.9260
x = 1×3
    2.4255    3.5730    1.9260
x = 1×3
    2.4255    3.5730    1.9260
x = 1×3
    2.4255    3.5730    1.9260
x = 1×3
    2.4255    3.5730    1.9260
x = 1×3
    2.4255    3.5730    1.9260
```

```
% printing the solution
fprintf('The solutins of the system are in Gauss Seidal Method: \n');
```

```
The solutins of the system are in Gauss Seidal Method:
```

```
disp(x0);
```

```
    2.4255    3.5730    1.9260
```

## 5.Tri-diagonal System

In numerical linear algebra, the **tridiagonal matrix algorithm**, also known as the **Thomas algorithm** (named after Llewellyn Thomas), is a simplified form of Gaussian elimination that can be used to solve tridiagonal systems of equations.

```
% taking input for number of equations in the system
n = input('Enter the number of equations: ');

% taking input for the coefficients as column matrix
a = input('Enter the first coefficients of the system: ');
b = input('Enter the secound coefficients of the system: ');
c = input('Enter the third coefficients of the system: ');
d = input('Enter the RHS coefficients of the system: ');

% initial solution
alpha = b;
s = d;

% calculating values of alpha and s
```

```
for i = 2:n
    m = a(i) / alpha(i - 1);
    alpha(i) = b(i) - (c(i - 1) * m);
    s(i) = d(i) - (s(i - 1) * m);
end

% back-substituion to find the solution
x(n) = s(n) / alpha(n);
for i = n - 1:-1:1
    x(i) = (s(i) - (c(i) * x(i + 1))) / alpha(i);
end

% printing the solution
for i = 1:n
    disp('The value of x in tri-diagonal System ')
    fprintf('x(%d): %f\n', i, x(i));
end
```

```
The value of x in tri-diagonal System
x(1): 2.086895
The value of x in tri-diagonal System
x(2): 1.291240
The value of x in tri-diagonal System
x(3): 1.078064
The value of x in tri-diagonal System
x(4): 1.021017
The value of x in tri-diagonal System
x(5): 1.006005
The value of x in tri-diagonal System
x(6): 1.003002
```

## 6.L-U Decomposition Method

In numerical analysis and linear algebra, **lower–upper** (**LU**) **decomposition** or **factorization** factors a matrix as the product of a lower triangular matrix and an upper triangular matrix (see matrix decomposition). The product sometimes includes a permutation matrix as well. LU decomposition can be viewed as the matrix form of Gaussian elimination. Computers usually solve square systems of linear equations using LU decomposition, and it is also a key step when inverting a matrix or computing the determinant of a matrix. The LU decomposition was introduced by the Polish mathematician Tadeusz Banachiewicz in 1938.[1] It's also referred to as **LR** decomposition (factors into left and right triangular matrices).

```
% The Matrix form must be in Ax = b
A = input('Enter the coefficient matrix A : \n');
B = input('Enter the RHS of the system: ');

% A = LU so LUX = B let UX =Y
%then first we need to find the value of Y and
%then we  need to find X by guass elimination

[L , U ] = lu(A) ;
```

```
disp('The lower Triangular Matrix L is ') ;
```

```
The lower Triangular Matrix L is
```

```
disp(L) ;
```

```
    1.0000         0         0
    0.2857    1.0000         0
    0.7143    0.6232    1.0000
```

```

disp('The upper Triangular Matrix U is ') ;
```

```
The upper Triangular Matrix U is
```

```
disp(U) ;
```

```
    7.0000   -3.0000    1.0000
         0    9.8571   -3.2857
         0         0   12.3333
```

```

%The new augmented matrix
P = [L B] ;

[row , col ] = size(P);

for m = 1 : row
    a = P (m,m);
    if a == 0
        disp ('LU decompostion method cannot be applicable');
        return
    end
    P( m , : ) = P (m , : ) / a;
end
s = 0;
%finding thesolution of Y
for m=1:row-1
    for k = row-m :row-1
        s = s + P (m+1 , col-k-1) * P(col-k-1,col);
        P (m+1 ,col ) = P(m+1,col) - s ;
        s = 0 ;
    end
end
Y = P (:,col);
disp (' The solution for Y is :');
```

```
 The solution for Y is :
```

```
disp(Y);
```

```
   21.0000
   31.0000
  -19.3188
```

```
%construction the new augmented matrix Q
Q = [U Y] ;
for m = 1 : row
```

```
    b = Q ( m , m );
    if b == 0
        disp('LU decompostion method cannot be applicable');
        return
    end
    Q(m,:) = Q(m,:)/b;

end
%Finding the final solution
for m =row : -1 : 2
    for k= m+1 : col
        s = s + Q(m-1,k-1) * Q (k-1 ,col);
        Q (m-1 , col) = Q (m-1 ,col)- s ;
    end
end
X = Q (: , col);
disp('The required Solution X is in LU decomposition');
```

```
The required Solution X is in LU decomposition
```

```
disp(X)
```

```
    4.4276
    2.6228
   -1.5664
```

**THE END**