2022

Linear Equations Solver

[AREEJ SALAHUDDIN 6389]
[SOHAILA HAZEM 6388]
[MANAR ABDELKADER 6485]

Contents

Overview

We want to Solve Systems of Linear Equations Program. We have direct and iterative methods:

A) Direct Methods:

1. Gaussian Elimination
2. Gaussian Jordan
3. LU Decomposition

B) Iterative Methods:

1. Gauss Seidel
2. Jacobi Method

The aim of this project is to implement different methods and to compare and analyze their behavior. We implemented the direct methods.

A Detailed Analysis of Each Method

1. Gaussian Elimination:

- ***Behavior:***

  The main idea of the method is to reduce the system of equations to an upper triangular matrix as this shows:

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 &= b_1 \\ a_{21}x_1 + a_{22}x_2 + a_{23}x_3 &= b_2 \\ a_{31}x_1 + a_{32}x_2 + a_{33}x_3 &= b_3 \end{aligned} \quad \Rightarrow \quad \begin{aligned} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 &= b_1 \\ a'_{22}x_2 + a'_{23}x_3 &= b'_2 \\ a''_{33}x_3 &= b''_3 \end{aligned}$$

1. Forward Elimination:

$$\left[\begin{array}{ccc|c} a_{11} & a_{12} & a_{13} & b_1 \\ a_{21} & a_{22} & a_{23} & b_2 \\ a_{31} & a_{32} & a_{33} & b_3 \end{array}\right]$$

$$\Downarrow$$

$$\left[\begin{array}{ccc|c} a_{11} & a_{12} & a_{13} & b_1 \\ & a'_{22} & a'_{23} & b'_2 \\ & & a''_{33} & b''_3 \end{array}\right]$$

2. Backward Substitution:

$$\begin{aligned} x_3 &= b''_3 / a''_{33} \\ x_2 &= (b'_2 - a'_{23}x_3) / a'_{22} \\ x_1 &= (b_1 - a_{12}x_2 - a_{13}x_3) / a_{11} \end{aligned}$$

- ***Pseudocode:***

```
function [x, flag] = gaussianElimination(A, B)
    flag = 0;
    [n,~] = size(A);
    x = zeros(n,1);
    %foward elimination
    for i = 1 :n-1
        if(A(i,i) == 0)
            flag = 1;
            return;
        end
       for k = i+1 : n
           factor = A(k,i)/A(i,i);
           for j = i+1: n
               A(k,j) = A(k,j) - factor * A(i,j);
           end
           B(k) = B(k)- factor * B(i);
       end
    end
    %backward substitution
    if(A(n,n) == 0)
        flag = 1;
        return;
    end
    x(n)= B(n)/A(n,n);
    for i = n-1:-1:1
        if(A(i,i) == 0)
            flag = 1;
            return;
        end
        sum = 0;
        for j = i+1 : n
            sum = sum + A(i,j) * x(j);
        end
        x(i) = (B(i) - sum) / A(i,i);
    end
end
```

- ***Conclusion:***

  From the pseudo code, We can see that it costs O (n^3) in elimination step, but O (n^2) in back substitution step.

- ***Misbehavior:***

  1. Division by zero in both main steps.
  2. Round-off errors will occur, and they will propagate from one iteration to the next one. When there are too many equations, this problem gets worse and more sensitive.

- ***Suggestions:***

  1. For division by zero, pivoting strategy must be used, where rows are swapped to prevent that case.
  2. Check the results if they're correct by substituting in the original system of equations.

3. Scaling is used to reduce round-off errors and improve accuracy.
4. Always use double-precision arithmetic because they're more accurate.

- ***Pseudocode with pivot:***

```
function [x,flag] = gaussianEliminationPivot(a,b)
flag=0;
n = size(a,1);
x = zeros(n,1);
a = [a,b];

%forward elimination
for i = 1:n-1
      if(a(i,i) == 0)
            flag = 1;
            return;
      end
    max = i;
    for row = i+1:n
        if abs(a(row,i)) > abs(a(max,i))
            max=row;
        end
    end
    if(max~=i)
        temp = a(i,:);
        a(i,:) = a(max,:);
        a(max,:) = temp;
    end
    m = a(i+1:n,i)/a(i,i);
    a(i+1:n,:) = a(i+1:n,:) - m*a(i,:);
end

% back substitution
 if(a(n,n) == 0)
        flag = 1;
        return;
 end
x(n) = a(n,n+1)/a(n,n);
for i = n-1:-1:1
    x(i) = (a(i,n+1) - a(i,i+1:n)*x(i+1:n))/a(i,i);
end
end
```

2. Gaussian Jordan:

- ***Behavior:***
  1. Elimination is applied to all equations (excluding the pivot equation) instead of just the subsequent equations.
  2. All rows are normalized by dividing them by their pivot elements
  3. No back substitution is required

- ***Pseudocode:***

```
function [C, flag] = gaussianJordan(A,B)
    flag = 0;
    X = [A B];
    [n,m]=size(X);
    i=1;
    while i <= n
        if X(i,i)==0
             C = [];
             flag = 1;
             break;
         end

    [r,m]= size(X);
    a=X(i,i);
    X(i,:)=X(i,:)/a;

    for k = 1:r
        if k == i
            continue;
        end
        X(k,:) = X(k,:)-X(i,:)*X(k,i);
    end
        i = i+1;
    end
    C = X(:,m);
end
```

- ***Conclusion:***
    1. Almost 50% more arithmetic operations than Gaussian elimination
    2. Gauss Jordan Elimination is preferred when the inverse of a matrix is required. [A | I]
    3. Apply Gauss Jordan elimination to convert A into an identity matrix. [I |inverse of A]

- ***Misbehavior:***
    1. Division by zero
    2. Round off error
    3. Ill conditioned systems.(Infinite solutions/No solution)

- ***Suggestions:***
    1. To avoid division by zero use pivoting
    2. To reduce round off error use pivoting and scaling

- ***Pseudocode with pivot:***

```
function  [x,flag]  = gaussianJordanPivot(A,B )
flag = 0;
X = [A B];
[n,m]=size(X);
x = zeros(n,1);
for i = 1:n
     if X(i,i)==0
             x = [];
             flag = 1;
             break;
     end
max=i;
    for row = i+1:n
        if abs(X(row,i)) > abs(X(max,i))
            max=row;
        end
    end
    if(max~=i)
        temp = X(i,:);
        X(i,:) = X(max,:);
        X(max,:) = temp;
    end

    for j = 1:n
        if  i ~= j
            X(j,:) = X(j,:) - X(i,:).*X(j,i)./X(i,i);
        end
    end
end
for i = 1:n
    x(i) = X(i,m)/X(i,i);
end
end
```

3. LU Decomposition:

- ***Behavior:***

  You have to solve a system of equations has the form:
  Ax = b

  1. Decompose [A] into [L] & [U], where LU = A
  2. Solve LUx = b using forward and back substitution.
  3. Assume that Ux = y, and then solve Ly = b to compute y by forward elimination.
  4. Then solve Ux = y to compute x by back substitution

- ***Pseudocode:***

```
function [x] = LUDecomposition(A, B)
    [n,~] = size(A);
    scalingFactors = zeros(n, 1);
    [L, U, B] = decompose(A, B, n, scalingFactors);
    [X] = substitute(L, U, B, n);
    x = zeros(n,1);
    for i = 1 : n
        x(i) = X(i);
    end
end

function [L, A, B] = decompose(A, B, n, scalingFactors)
    for i = 1 : n
        scalingFactors(i) = abs(A(i, 1));
        for j = 2 : n
            if abs(A(i, j)) > scalingFactors(i)
                scalingFactors(i) = abs(A(i, j));
            end
        end
    end

    L = zeros(n);
    for i = 1 : n
        L(i,i) = 1;   %set diagonal elements = 1
    end
    % decompose A into U and L
    for k = 1 : n - 1
        [L, A, B, scalingFactors] =  pivot(L, A, B, scalingFactors, n, k);
        % forward elimination for A to get U
        for i = k + 1 : n
            factor = A(i, k) / A(k, k);
            for j = k + 1 : n
                A(i, j) = A(i, j) - factor * A(k, j);
            end
            L(i, k) = factor;
        end
    end
end
```

```
function [L, U, B, scalingFactors] =  pivot(L, U, B, scalingFactors, n, row)
    pivot = row;
    big = abs(U(row, row) / scalingFactors(row));
    for i = row + 1 : n
        dummy = abs(U(i, row) / scalingFactors(i));
        if (dummy > big)
            big = dummy;
            pivot = i;
        end
    end

    if (pivot ~= row)
        % swap in U Matrix
        for j = row : n
            dummy = U(pivot, j);
            U(pivot, j) = U(row, j);
            U(row, j) = dummy;
        end
        % swap in L Matrix
        for j = 1 : row-1
            dummy = L(pivot, j);
            L(pivot, j) = L(row, j);
            L(row, j) = dummy;
        end
        % swap in B Matrix
        dummy = B(pivot);
        B(pivot) = B(row);
        B(row) = dummy;

        % swap in scaling factors
        dummy = scalingFactors(pivot);
        scalingFactors(pivot) = scalingFactors(row);
        scalingFactors(row) = dummy;
    end
end
```

```
function [X] = substitute(L, U, B, n)
    Y = zeros(n, 1);
    Y(1) = B(1) / L(1,1);
    for i = 2 : n
        sum = 0;
        for j = 1 : i - 1
            sum = sum + L(i, j) * Y(j);
        end
        Y(i) = (B(i) - sum) / L(i,i);
    end

    X(n) = Y(n) / U(n, n);
    for i = n - 1 : -1 : 1
        sum = 0;
        for j = i + 1 : n
            sum = sum + U(i, j) * X(j);
        end
        X(i) = (Y(i) - sum) / U(i, i);
    end
end
```

- ***Conclusion:***

  From the pseudo code that such a method costs O (n^3) to compute both [L] & [U], and costs O (n^2) to apply forward elimination & back substitution.

- ***Misbehavior:***

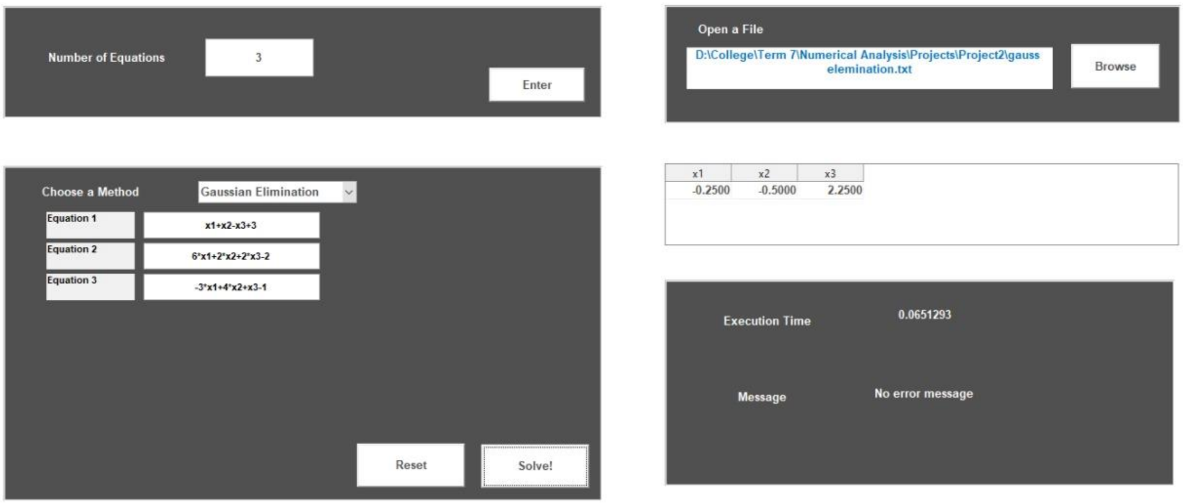
  1. Dividing by zero in the process of pivoting
  2. Reordering of the elements in [x] during Swapping rows.

- ***Suggestions:***

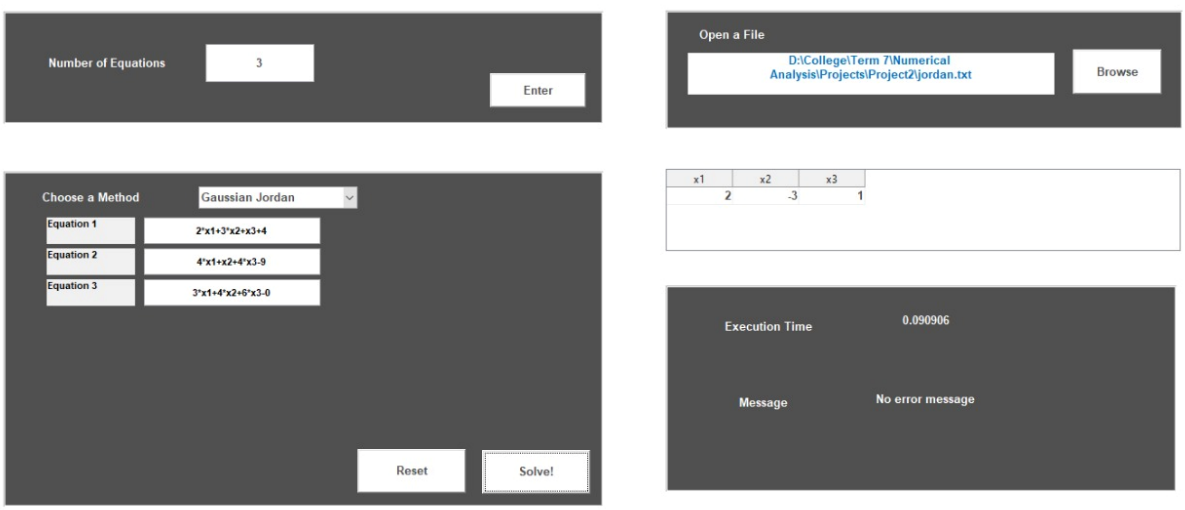
  1. Concerning the space, [L] & [U] can be stored in one matrix, where L takes the lower triangle, and U takes the upper one of the matrix.

Sample Runs:

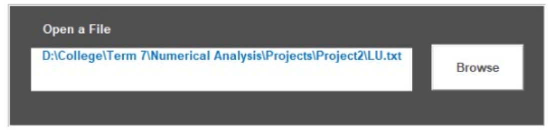
1. Gauss Elimination:



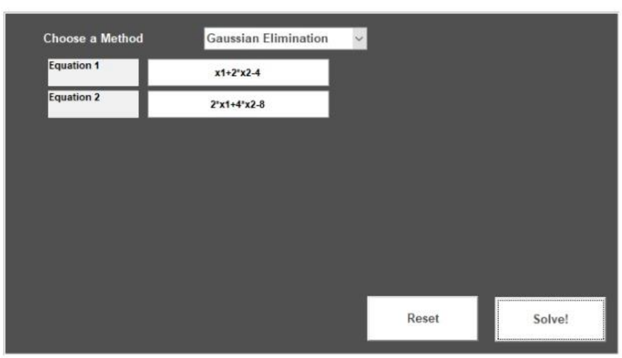
2. Gauss Jordan:



3. LU Decomposition:



4. Infinite Solutions:



5. No Solution: