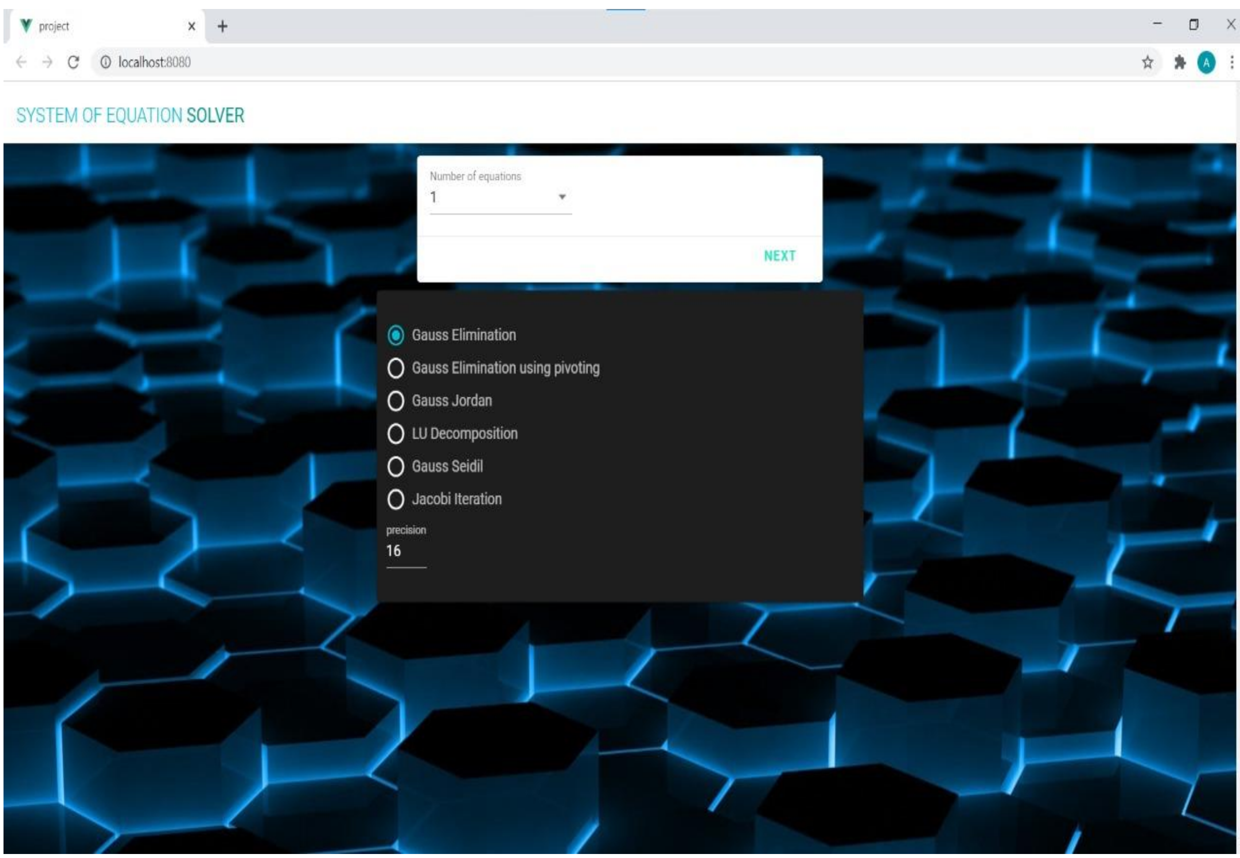
Faculty of engineering **Alexandria university**

CSED **Numerical analysis**

ASSIGNMENT 1

Name	ID
Fares Mohamed Fouad	18011223
Ali Ahmed Ibrahim	18011064
Ahmed Mohamed Abd-Elmonem	18010225
Mohamed Ebrahem El-Sayed	18011333
Fares Waheed Abd-Elhakeem	18011224



1.How to Run the Program:

1.Open the “numericalBackend” folder using any java IDE and Run “NumericalBackendApplication.java” to run the spring Boot App.

2.Run the Vue.js Applications using VS codes:

*Make sure you have installed node.js and yarn

*open the “assignment1” folder in VS codes.

Then write this command in terminal of:

>cd project

>yarn run serve

*Make sure you have the extension “Vuetify for vs”.

*Write the command “yarn add axios” in the terminal of VS code.

*Run the application using the command “yarn run serve”.

*The program will run at: http://localhost:8080/

<u>The First Three Functions:</u>

<u>Pseudo Code</u>:

First of all, some helping methods for Gauss elimination without/with pivoting and for Gauss-Jordan elimination:

```
/**
a: Coefficients matrix
b: Free terms vector (aX=b)
p: precesion
**/
Double[] substitute(a, b, p){
temp = b.length;
sum = 0;
x = new Double[]
x[temp-1] = round (b[temp -1] / round(a[temp-1][temp-1], p), p)
for i = temp-2   downTo   0
        sum = 0;
        for j = i+1   upTo  temp – 1
                sum = round(sum + round(a[i][j] * x[j], p), p);
        end for
        x[i] = round(round(b[i] - sum, p), p);
end for
```

```
return x;
}
/**
This method helps with rounding to certain precision
value: value to be rounded
places: The precision in decimal places
**/
Double round(value, places){
If places < 0 then
	Throw illegalArgumentException
End if
bd = new BigDecimal(Double.toString(value));
bd = bd.setScale(places, roundingMode.HALF_UP);
return bd.doubleValue;
}

/**
This method checks if the given linear equations have a unique solution
pivot: indicate whether we are using pivoting or not
sn:  Scaling factors array
**/
Boolean hasUniqueSolution(a, b, p, pivot, sn){
```

```
Stack s = eliminate(a, b, p, pivot, sn);
If size of s = 0 then
    Return false
end if
a = s.pop();
rank = 0;
n = b. length
for i = 0   upTo  n-1
    for j = i  upTo  n-1
        if a[i][j] != 0 then
            rank ++;
            break;
        end if
    end for
end for
if rank = n then
    return true
end if
return false;
}
/**
Elimination method
```

```
**/
Eliminate(a, b, p, pivot, sn){
factor = 0
for k = 0   upTo  b.length-2
        if pivot = true then
                pivot(a, b, sn, k)
        end if
        for i = k+1 upTo b.length - 1
                factor = round(a[i][k]/a[k][k], p)
                for j = k+1  upTo b.length – 1
                        a[i][j] = round(a[i][j] – round(factor * a[k][j], p));
                end for
                b[i] = round(b[i] –round(factor * b[k], p) , p);
        end for
end for
s = new Stack
s.push(b); s.push(a)
return s;
}
/**
This method is responsible for pivoting
S: scaling factor for each row
```

```
K: current row
**/
void pivot (a, b, s, k){
pivot = k;
big = absolute of (a[k][k] / s[k]);
temp = 0;
for i = k+1  upTo  b.length – 1
        temp = absolute of (a[i][k] / s[i]);
        if temp > big then
                big = temp;
                pivot = I;
        end if
end for
if pivot not equal k then
        for j = k  upTo  b.lenght-1
                swap  a[pivot][j],  a[k][j];
        end for
        swap b[pivot], b[k]
        swap s[pivot], s[k]
end if
}
/**
```

```
This method is responsible for getting the scaling factors
n: number of variables
**/
Double[] scalingFactores(a, n){
Sn = new Double array of size n
For i = 0   upTo  n-1
        Sn[i] = absolute of a[i][0]
        For j = 1 upTo  n-1
                If absolute of a[i][j] > Sn[i] then
                        Sn[i] = absolute of a[i][j]
                End if
        End for
end for
return Sn
}

/**
Backward elimination method
**/
Boolean backwardEliminate(a, b, p){
factor = 0;
for k = b.length  downTo  0
```

```
        for  i = k -1  downTo 0
                factor = round(a[i][k]/a[k][k], p);
                b[i] = round(b[i] –round(factor * b[k], p), p);
        end for
end for
return true
}
```

<u>Gauss elimination without pivoting:</u>

```
/**
a: Coefficients matrix
b: Free terms vector (aX=b)
p: precesion
**/
Double[] gaussElimination (a, b, p){
If aX = b doesn't have a unique solution then
        Return null;
End if
Return substitute(a, b, p);
}
```

<u>Gauss elimination with pivoting:</u>

```
Double gaussEliminationPivot(a, b, p){
```

```
Sn = scalingFactors(a, b.length);
check = hasUniqueSolution(a, b, p, true, sn);
if check  = true then
        return substitute(a, b, p);
end if
return null;
}
```

<u>Gauss-Jordan with pivoting:</u>

```
Double[] gaussJordan(a, b, p){
        Sn = scaling;
        check = true if equations has unique solution
        if check = false then
                return null;
        end if
        for i = 0   upTo   b.length-1
                factor = a[i][j];
                for  j = i   upTo  b.length-1
                        a[i][j] = round(a[i][j] / factor, p);
                end for
                b[i] = round(b[i]/factor, p);
        end for
        check2 = backwardEliminate(a, b, p);
```

```
        if check2 then
                return b;
        end if
        return null;
}
```

<u>Data structures used</u>:

2D array has been used to carry coefficient and, 1D array has been used to carry free terms. This was a good decision because it was easy to manipulate the array elements.

Stack data structure has been used in elimination method. It was useful in sending back 2 or more variables of different types as a response from the method.

<u>Comparison between the methods:</u>

Method	Gauss elimination with/without pivoting	Gauss-Jordan elimination
Elimination steps	Forward Elimination – only needs to eliminate the coefficients below the diagonal. Cost ~ $2n^3/3$	Needs to eliminate coefficients below and above the diagonal. Cost ~ 2 *$2n^3/3$
Substitution steps	Back Substitution Cost ~ $O(n^2)$	No substitution steps
Total	$2n^3/3 + O(n^2)$	$4n^3/3$
Precision	More precise when using a certain number of significant bits	Less precise than Gauss elimination when using a certain number of significant bits

Problematic functions:

For Gauss elimination without pivoting we have a problem when getting the scaling factor, we might divide by zero. However, to overcome this we can apply pivoting with scaling (for extra precision)

For Gauss-Jordan elimination, it doesn't have a problematic function.

4.LU Decomposition:

Data Structure Used:

2D array has been used to carry coefficient and, 1D array has been used to carry free terms. This was a good decision because it was easy to manipulate the array elements.

Problems:

-When there is a zero on the mail diagonal, this introduces some error

As Dividing by that can be avoided using Pivoting with Scaling.

3.1. Doo Little Form:

*This is the pseudocode for decomposition phase:

function Decompose (A, n)

  DO, FOR k = 1, n - 1

    DO, FOR i = k + 1, n

      factor = $A_{i,k}$ / $A_{k,k}$

      $A_{i,k}$ = factor

      DO, FOR j = k + 1, n

        $A_{i,j}$ = $A_{i,j}$ - factor * $A_{k,j}$

```
        END DO
    END DO
 END DO
END Decompose
```

*Then we can get L from the lower bound of A and get U from the upper bound of A. Store them, then using two steps:

1.Forward Substitution with L, B.

2.Backward Substitution with U and result from Forward Substitution.

*Time Complexity:

T($\frac{8n^3}{3} + 12\ n^2 + \frac{4n}{3}$)

Where T = clock cycle time and n = size of the matrix.

3.2. <u>Crout Decomposition:</u>

Pseudo code:

Input is Matrix A and B;

```
 sum;
 L[][] ;
 U[][];
      for i=0, n-1
         U[i][i] = 1.0D; // set elements in the diagonal to 1
      for j = 0, n-1 {
         // set lower triangle values.
          for i = j, n-1 {
            sum = 0;
```

```
        for k = 0, j-1 {
            sum = sum + L[i][k] * U[k][j];
        }
        l[i][j] = A[i][j] - sum;
    }
    // set upper triangle values.
    for i = j, n -1{
        sum = 0;
        for k=0, j {
            sum = sum + L[j][k] * U[k][i];
        }
        if L[j][j] == 0 {
            return null; // Can't divide by zero
        }
        U[j][i] = (A[j][i] - sum)/ L[j][j];
    }
}
```

Then doing last 2 steps:

1.Forward Substitution with L, B.

2.Backward Substitution with U and result from Forward Substitution.

*Time Complexity: (As the first method)

T($\frac{8n^3}{3} + 12\, n^2 + \frac{4n}{3}$)

3.3 Chelosky Form:

*Matrix should be symmetric to use this form.

Pseudocode:

```
Input is Matrix A and B;
 sum;
  L[][];
  U[][]; // transpose of L.
      For i=0, n-1 {
          for j=0, i {
              sum =0;
              if(i==j) {// diagonal elements
                for k=0, j-1
                    sum = sum + L[i][k] * L[i][k];
                L[i][j] = Math.sqrt(A[i][j]-sum);
                U[j][i] = L[i][j];
              }
              else {// non diagonal
                for k=0, j-1
                    sum = sum + L[i][k] * L[j][k];
                if(L[j][j] == 0) {// divide by zero
                    return null;
                }
                L[i][j] = (A[i][j]-sum)/L[j][j];
                U[j][i] = L[i][j];
              }
```

```
        }
    }
```

*Time Complexity:

T($\frac{2n^3}{3} - \frac{n}{3} - 1$)

-Iterative Methods:

<u>-Data structure used and how helpful was your choice:</u>

ArrayLists for the iteration methods to be dynamic according to the absolute relative error the user wants.

<u>-Comparison between different methods (time complexity, convergence, best and worst case for each method and precisions)</u>

-This two iterative methods begin with initial guess for
solution and successively improve it until desired
accuracy attained.
-It might take infinite number of iterations to converge to exact solution theoretically, but in practice iterations are terminated when residual is as small as desired.

<u>The difference between the 2 methods:</u>

-Gauss-Seidel is same as Jacobi technique except with one important difference:
-A newly computed x value is substituted in the subsequent equations in the same iteration.
-So Gauss-Seidel converges faster than Jacobi.

<u>Pitfalls:</u> This two iterative methods not all systems of equations will converge or it converges very slowly.

Best Case: One class of system of equations always converges a diagonally dominant coefficient matrix but not every system of equations can be rearranged to have a diagonally dominant coefficient matrix.

time complexity: Each iteration takes O(n2) time.

Gauss-Siedel pseudo-code:

```
do {
	for ( i < noOfEquations ){
		element = round(b[i]/a[i][i] , p)
		for ( j < n ){
			if (i != j) {
				m = k
				if ( i < j )
					m--
				x[i][k] = round(x[i][k] -round( round(a[i][j]*x[i][m]* ,p)
/a[i][i] ,p) ,p)
			}
		}
	}
	for (i < noOfEquations){	//to calculate relative errors
		relErrors[i] = (x[i][k]- x[i][k-1]) / x[i][k] * 100
	}
	k++
} while ( !relativeErrorTest && k <= noOfIterations)

return x
```

Jacobi pseudo-code:

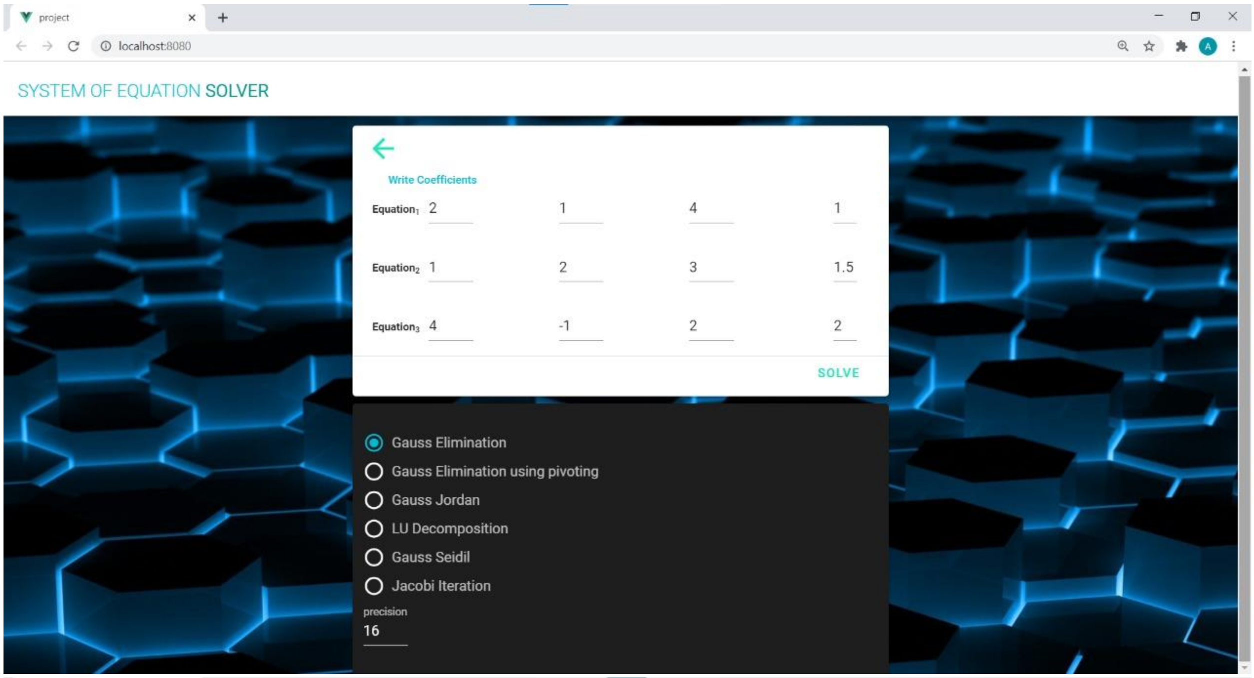
```
do {
	for ( i < noOfEquations ){
```

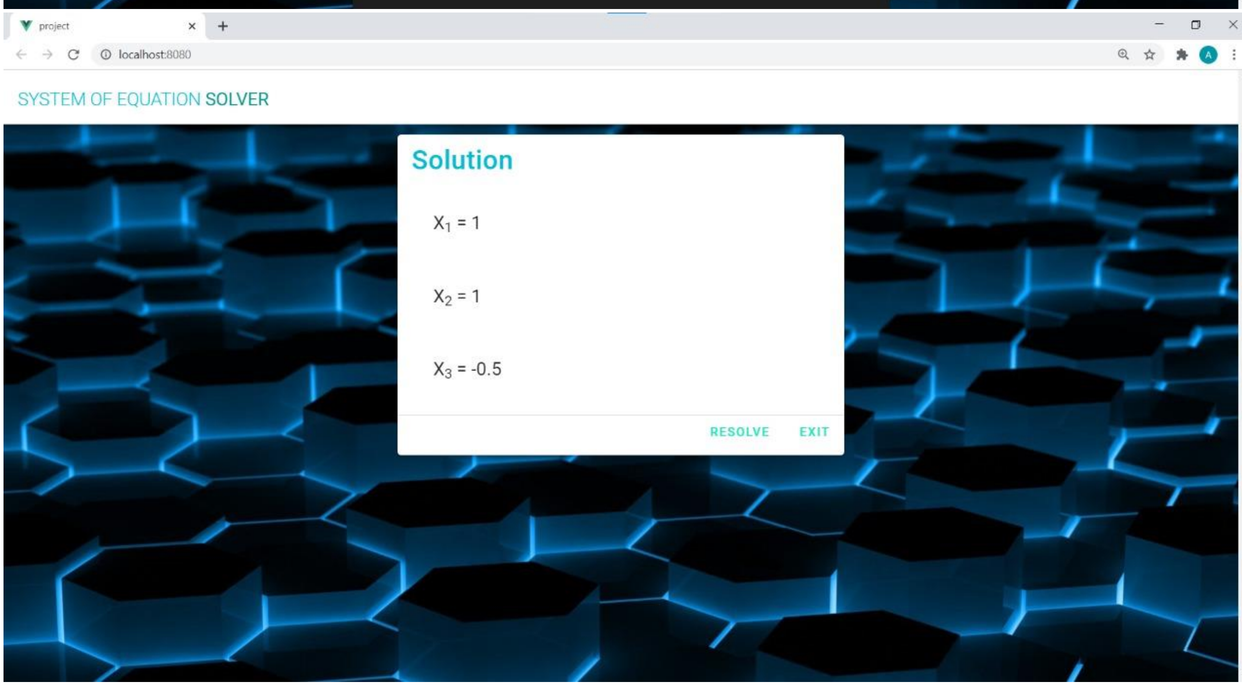
```
            element = round(b[i]/a[i][i] , p)
            for ( j < n ){
                    if (i != j) {
                            m = k
                            x[i][k] = round(x[i][k] -round( round(a[i][j]*x[i][k-1]* ,p)
/a[i][i] ,p) ,p)
                    }
            }
    }
    for (i < noOfEquations){        //to calculate relative errors
            relErrors[i] = (x[i][k]- x[i][k-1]) / x[i][k] * 100
    }
    k++
} while ( !relativeErrorTest && k <= noOfIterations)

return x
```

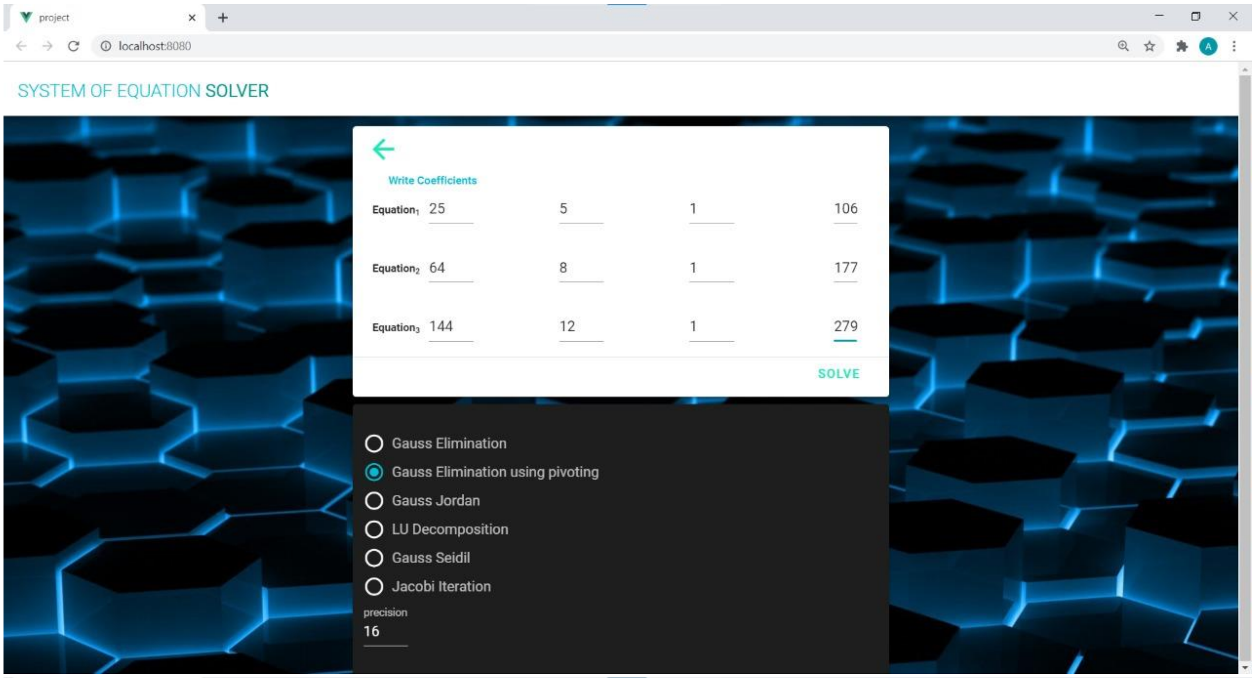
-Analysis of the Runtime of each method with the same data:

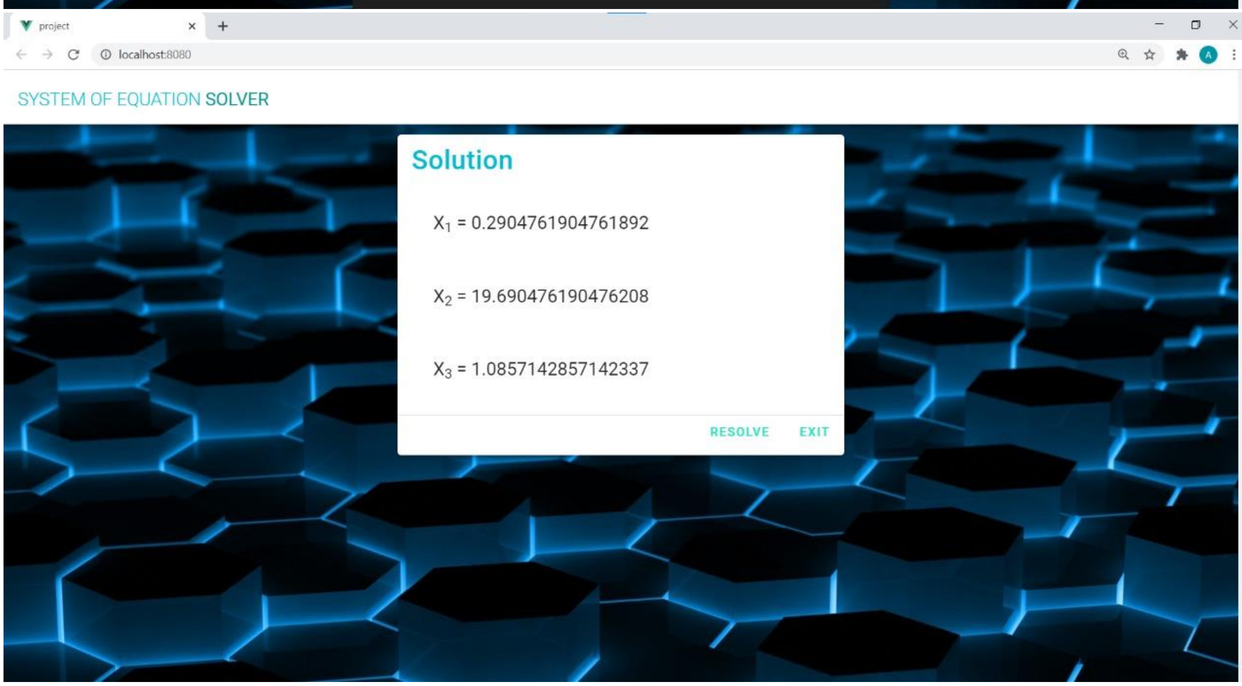
Method	Time(micro sec)
Gauss Elimination	765 micro sec
Gauss with pivoting	1786.2 micro sec
Gauss Jordan	1655.25 micro sec
Doolittle	869.5 micro sec
Crout	793.75 micro sec
Cholesky	873.6 micro sec
Gauss Seidel (30 Itera,0.05 error)	533.16 micro sec
Jacobi(30 Itera,0.05 error)	271 micro sec

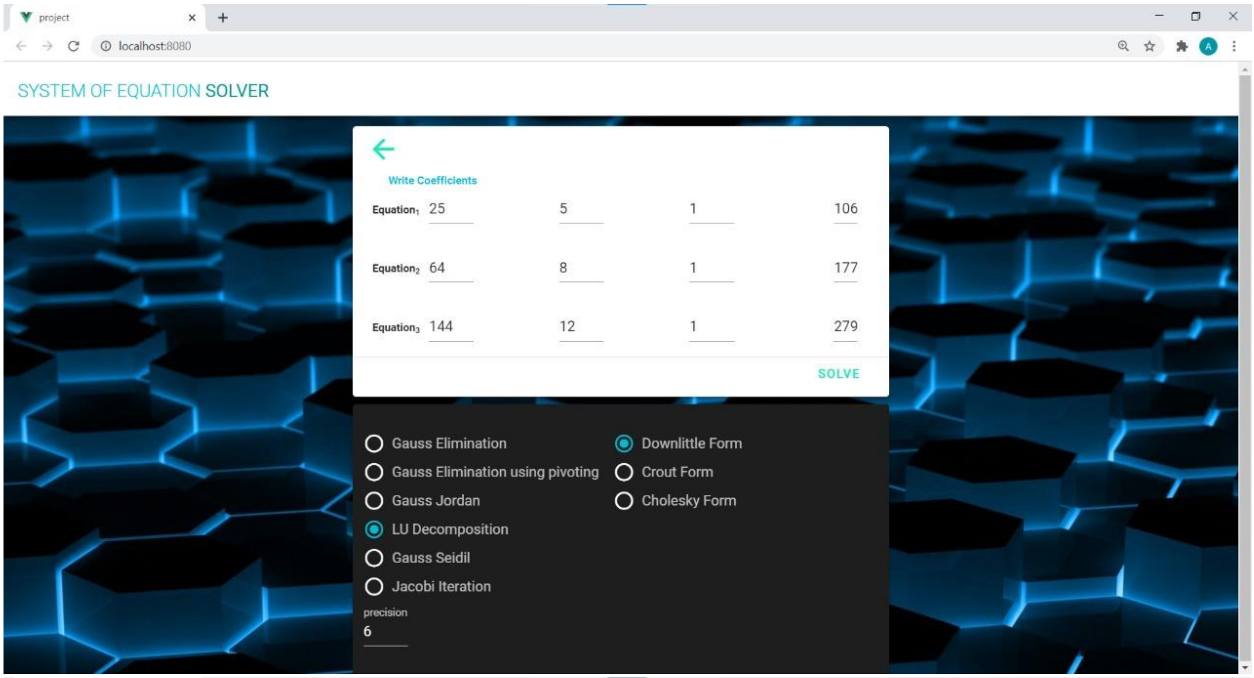
Snapshots:

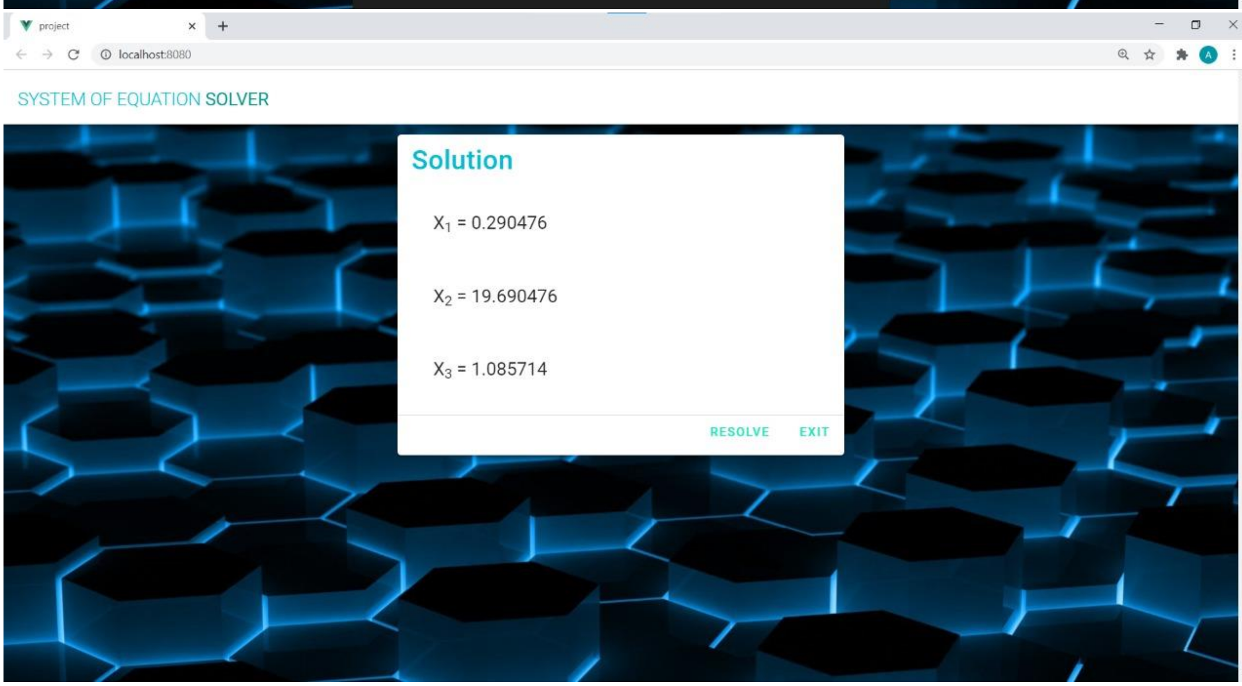

project
localhost:8080
SYSTEM OF EQUATION SOLVER
Write Coefficients
Equation1 2 1 4 1
Equation2 1 2 3 1.5
Equation3 4 -1 2 2
SOLVE
Gauss Elimination
Gauss Elimination using pivoting
Gauss Jordan
LU Decomposition
Gauss Seidil
Jacobi Iteration
precision
16

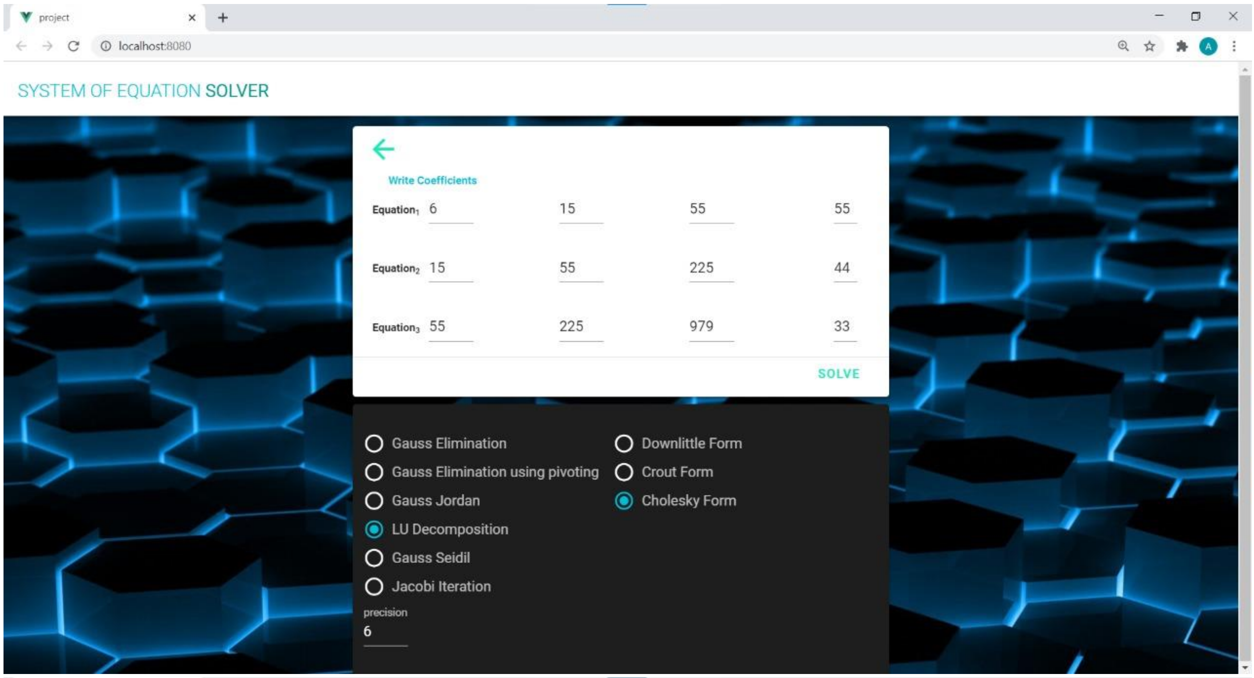

project
localhost:8080
SYSTEM OF EQUATION SOLVER
Solution
X1 = 1
X2 = 1
X3 = -0.5
RESOLVE
EXIT

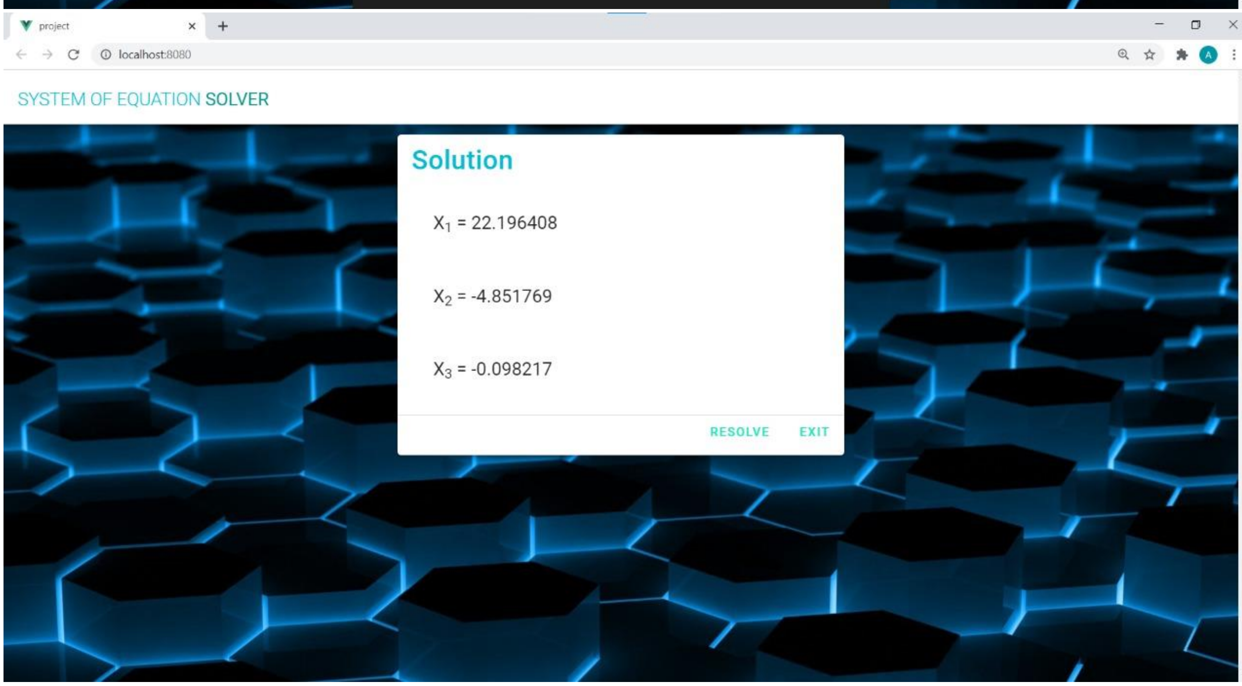

project
localhost:8080
SYSTEM OF EQUATION SOLVER
Write Coefficients
Equation1 25 5 1 106
Equation2 64 8 1 177
Equation3 144 12 1 279
SOLVE
Gauss Elimination
Gauss Elimination using pivoting
Gauss Jordan
LU Decomposition
Gauss Seidil
Jacobi Iteration
precision
16

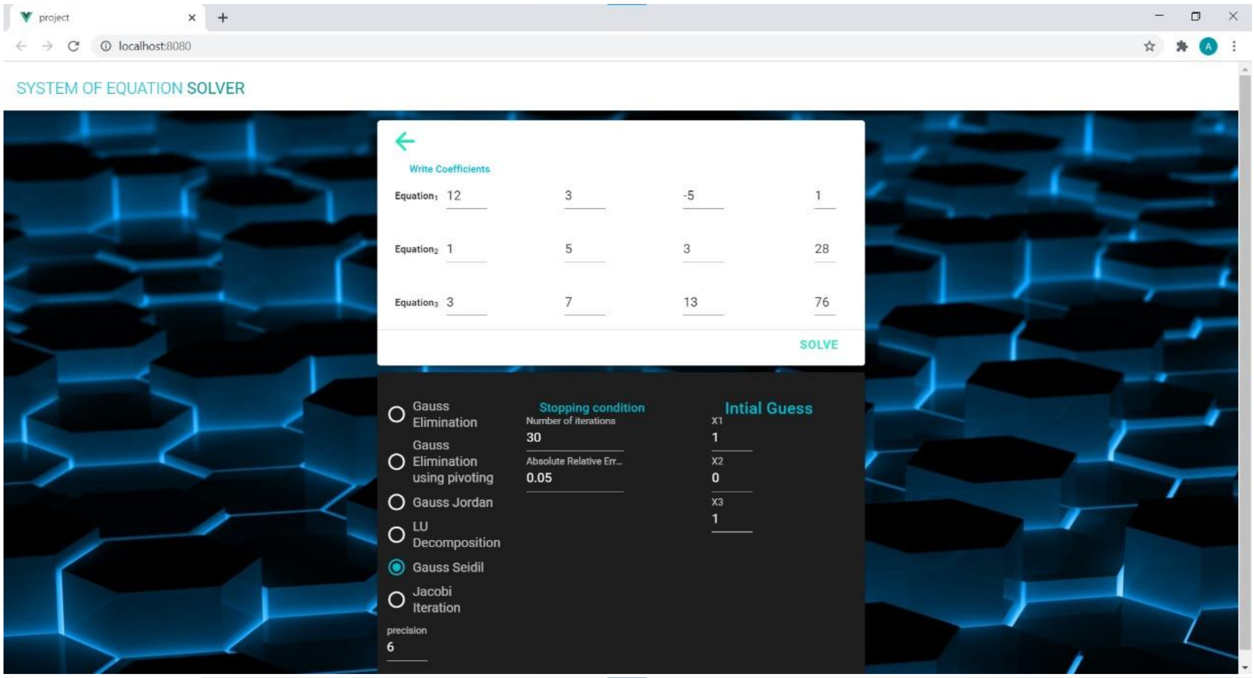

project
localhost:8080
SYSTEM OF EQUATION SOLVER
Solution
X1 = 0.2904761904761892
X2 = 19.690476190476208
X3 = 1.0857142857142337
RESOLVE
EXIT

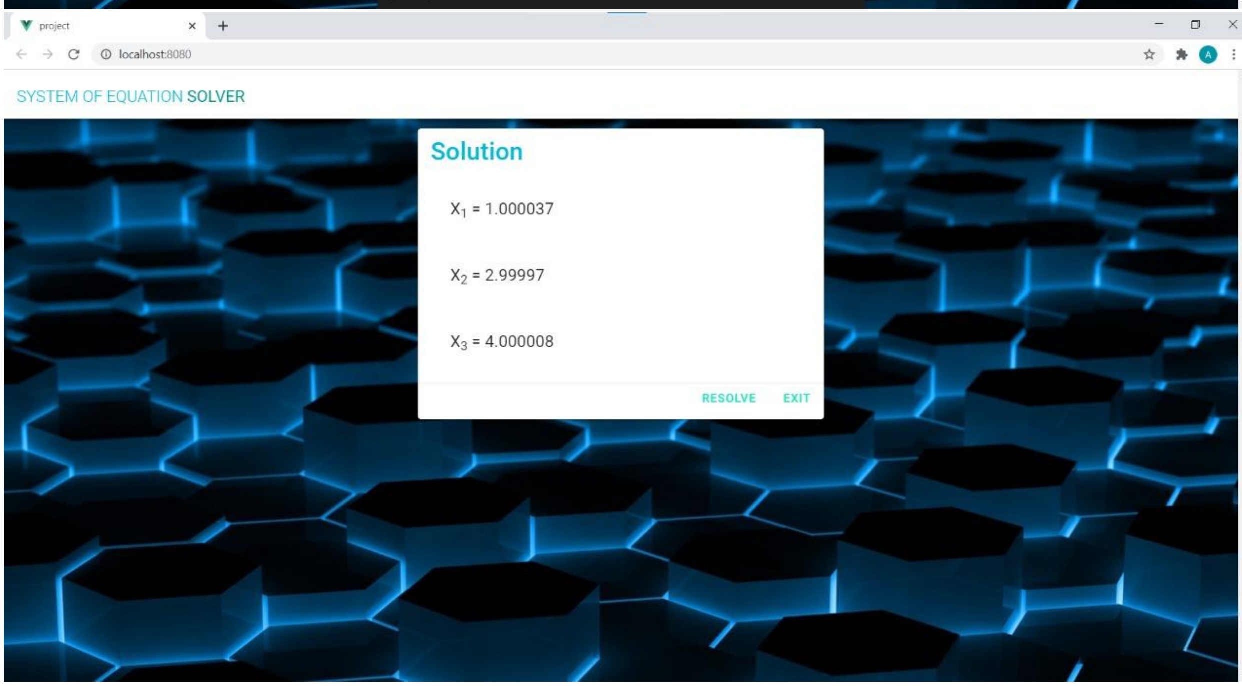

project
localhost:8080
SYSTEM OF EQUATION SOLVER
Write Coefficients
Equation1 25 5 1 106
Equation2 64 8 1 177
Equation3 144 12 1 279
SOLVE
Gauss Elimination
Gauss Elimination using pivoting
Gauss Jordan
LU Decomposition
Gauss Seidil
Jacobi Iteration
Downlittle Form
Crout Form
Cholesky Form
precision
6

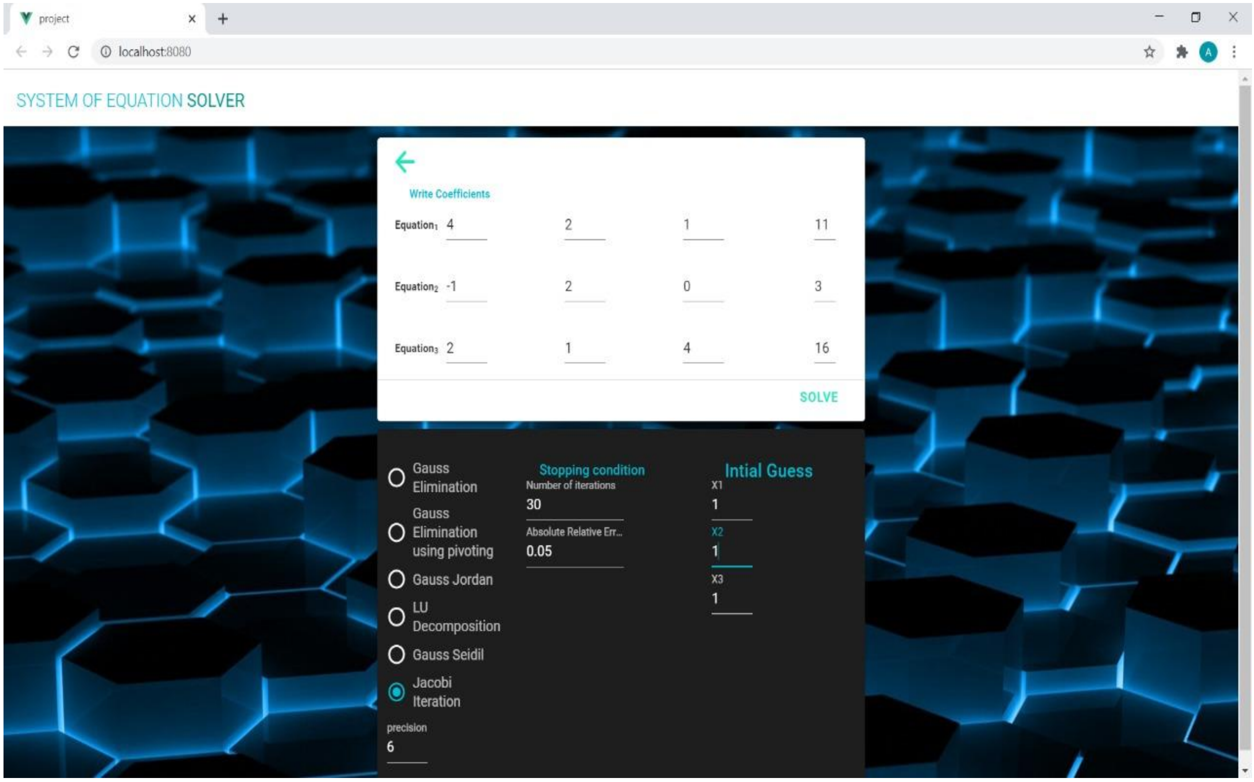

project
localhost:8080
SYSTEM OF EQUATION SOLVER
Solution
X1 = 0.290476
X2 = 19.690476
X3 = 1.085714
RESOLVE
EXIT

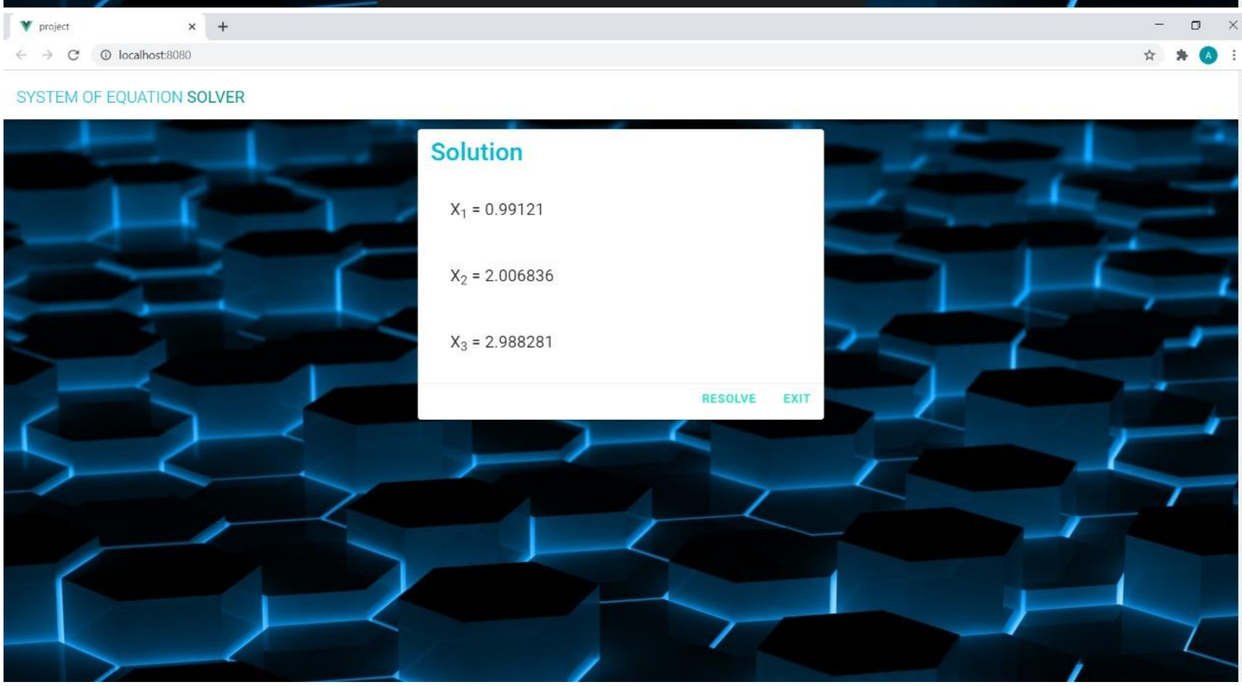

project
localhost:8080
SYSTEM OF EQUATION SOLVER
Write Coefficients
Equation1 6 15 55 55
Equation2 15 55 225 44
Equation3 55 225 979 33
SOLVE
Gauss Elimination
Gauss Elimination using pivoting
Gauss Jordan
LU Decomposition
Gauss Seidil
Jacobi Iteration
Downlittle Form
Crout Form
Cholesky Form
precision
6


project
localhost:8080
SYSTEM OF EQUATION SOLVER
Solution
X1 = 22.196408
X2 = -4.851769
X3 = -0.098217
RESOLVE
EXIT


project
localhost:8080
SYSTEM OF EQUATION SOLVER
Write Coefficients
Equation1 12 3 -5 1
Equation2 1 5 3 28
Equation3 3 7 13 76
SOLVE
Gauss Elimination
Gauss Elimination using pivoting
Gauss Jordan
LU Decomposition
Gauss Seidil
Jacobi Iteration
precision
6
Stopping condition
Number of iterations
30
Absolute Relative Err...
0.05
Intial Guess
X1
1
X2
0
X3
1


project
localhost:8080
SYSTEM OF EQUATION SOLVER
Solution
X1 = 1.000037
X2 = 2.99997
X3 = 4.000008
RESOLVE
EXIT


project
localhost:8080
SYSTEM OF EQUATION SOLVER
Write Coefficients
Equation1 4 2 1 11
Equation2 -1 2 0 3
Equation3 2 1 4 16
SOLVE
Gauss Elimination
Gauss Elimination using pivoting
Gauss Jordan
LU Decomposition
Gauss Seidil
Jacobi Iteration
precision
6
Stopping condition
Number of iterations
30
Absolute Relative Err...
0.05
Intial Guess
X1
1
X2
1
X3
1


project
localhost:8080
SYSTEM OF EQUATION SOLVER
Solution
X1 = 0.99121
X2 = 2.006836
X3 = 2.988281
RESOLVE
EXIT