**Faculty of Engineering**
**Alexandria University**
**CSED 2024**
**Numerical**
**CSE 211**

# Report
# Numerical Project
# Phase 1

Aliaa Fathy Elsayed
19016032
Aliaa Ibrahim Ahmed
19016031
Bassant Yasser Salah
19017262
Nada Mohamed Ibrahim
19016782
Toka Ashraf Abo elwafa
19015539

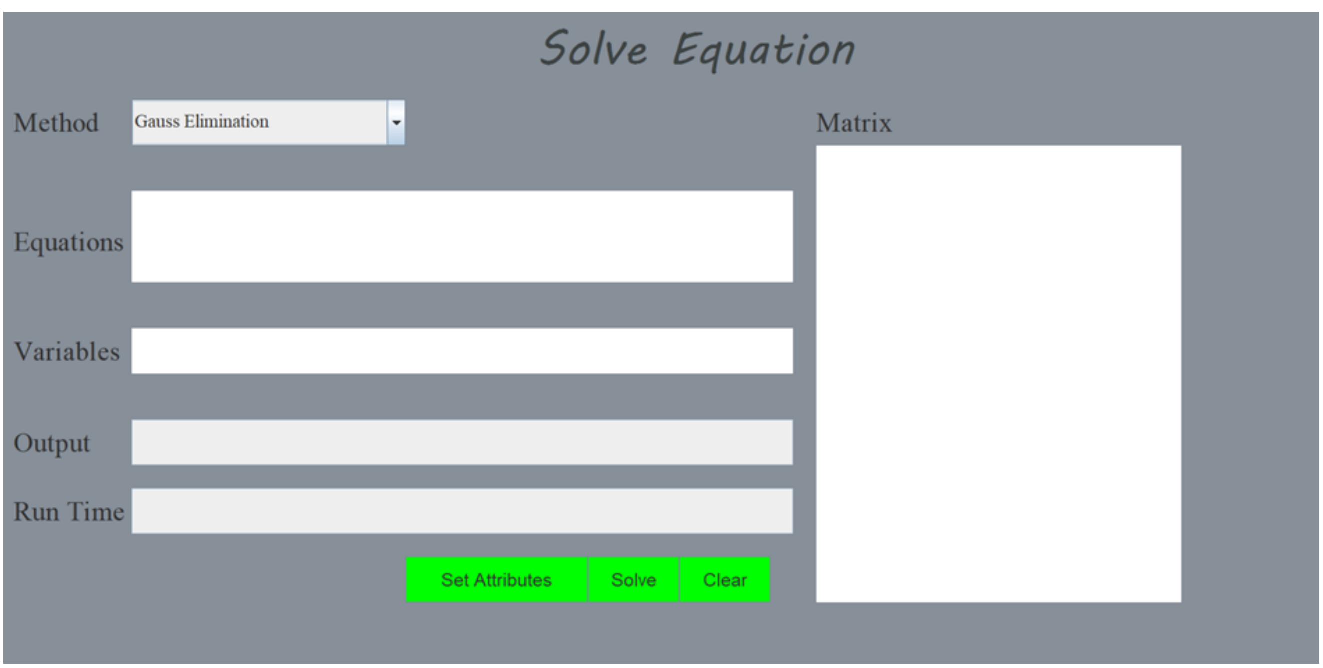


Table of contents:

## 1. Brief Description:

It is a calculator that helps the user to solve linear equations with 5 different methods which are(**Gauss Elimination- Gauss Jordan- Lu decomposition- Gauss Seidel- Jacobi Iteration**).

---

## 2. Flowchart or pseudo-code for each method:

### -pseudo-code

**a) Gauss-Elimination:**

```
/**A[][]: contain coefficient of input equations**/
/**B[]: contain the right hand side of each equation**/
/**scaling[]: contain the greatest number in every row**/
/**Answer[]: contain solution**/
/**length[]: number of unknowns**/
/**precision :the number the user select**/
 precision( num, precision) {
   BigDecimal round ⇐ new BigDecimal(Double.toString(num));

   round ⇐ round.setScale(precision, RoundingMode.HALF_UP);

   return round.doubleValue();

}

Gauss( A, B,precision) {
       length=B.length;
       i⇐0;
       j⇐1;
      /** scaling array to put the biggest value of every row to make scaling**/
      while (i less than length){
         Scaling[i] ⇐ Math.abs(A[i][0]);
         /**the initial value of scaling array is the first element in every row**/
         while (j less than length){
            if (Math.abs(A[i][j]) greater than scaling[i]) {
               scaling[i] ⇐ Math.abs(A[i][j]);
            }
            end if
            j++;
 }
         reset j ⇐ 1;
         increase i by 1;
      }
      ForwardElimination(A, B,scaling, precision);
      Sol⇐   BackSubstitute(A, B,precision);
     Return Sol;
```

```
}
Partial_Pivoting( A, B, scaling, k) {
    pivot ⇐ k;
    max ⇐ 0;
   temp ⇐ 0;
   length⇐ B.length;
    i⇐k+1;
   max ⇐ Math.abs(A[k][k] / scaliing[k]);
   while (i less than length){
      temp ⇐ Math.abs(A[i][k] / scaliing[i]);
      if (temp greater than max)
         max ⇐ temp;
         pivot ⇐ i;
      end if
     increase i by 1;
   }
   if (pivot not equal k)
       j⇐k;
      while (j less than length){
         temp ⇐ A[pivot][j];
         A[pivot][j] ⇐ A[k][j];
         A[k][j] ⇐ temp;
         increase j by 1;
      }
      temp ⇐ scaling[pivot];
      scaling[pivot] ⇐ scaliing[k];
      scaliing[k] ⇐ temp;
      temp ⇐ B[pivot];
      B[pivot] ⇐ B[k];
      B[k] ⇐ temp;
   end if
}

ForwardElimination( A,  B, scaling, precision) {
    length⇐ B.length;
   check⇐Math.pow(10,-precision);
   for (k⇐ 0 to length-1 increase by 1) {
      Partial_Pivoting(A, B, scaling, k);
      for ( i ⇐ k + 1 to length increase by 1) {
          factor ⇐ precision(A[i][k]/A[k][k],precision);
         for ( j ⇐ k to  length increase by 1) {
            A[i][j] ⇐ precision(A[i][j]-factor*A[k][j],precision);
         }
         B[i] ⇐precision(B[i] - factor * B[k],precision);
      }
          }
}
 BackSubstitute( A, B, precisionV) {
```

```
 length ⇐ B.length;
Answer[length - 1] ⇐  precision(B[length - 1] / A[length - 1][length - 1], precisionV);
for ( i ⇐ length - 2 to 0 decrease by 1) {
     sum ⇐ 0.0;
    for ( j ⇐ i + 1 to length increase by 1) {
       sum + ⇐ precision(A[i][j] * Answer[j], precisionV);
    }
    Answer[i] ⇐precision((B[i] - sum) / A[i][i], precisionV);
  }
Return Answer;
  }
```

## b)Gauss Jordan:

```
Gauus_jorrdan( A, B, precisionV) {

     length⇐B.length;

     I⇐0;
    j⇐1;

     /** scaling array to put the biggest value of every row to make scaling**/

     while (i less than length){

     Scaling[i] ⇐ Math.abs(A[i][0]);

     /**the initial value of scaling array is the first element in every row**/

     while (j less than length){

          if (Math.abs(A[i][j]) greater than scaling[i]) {

          scaling[i] ⇐ Math.abs(A[i][j]);

          }

          end if

          j++;

     }

     reset j ⇐1;

     increase i by 1;

     }

      ForwardElimination(A, B,scaling, precision);
```

```
        BackwardElimination(A,B,scaling,precisionV);
        Sol⇐BackSubstitute(A, B,precision);
   Return Sol; }
```

*BackwardElimination(A,B,scaling, precisionV){*

```
        length⇐B.length;
        check⇐Math.pow(10,-precisionV);
        for ( k ⇐ length-1 to 0 decrease by 1) {
        for ( i ⇐ k -1 to 0; decrease by 1) {
                factor ⇐ precision(A[i][k] / A[k][k],precisionV);
                for ( j ⇐ k+1 to length increase by 1) {
                A[i][j] ⇐precision( A[i][j] - factor * A[k][j],precisionV);
                }
                B[i] ⇐ precision(B[i] - factor * B[k],precisionV);
        }
        }
        }
        Jordan_sub(A,B, precisionV){
        length⇐B.length;
        for (i⇐0 to length increase by 1){
        Answer[i] ⇐B[i]/A[i][i];
        }
        Return Answer;
        }
```

c)Lu decomposition:

**Doolittle() {**

```
      z[] ⇐ {};
      fraction⇐ 0;
      luResult.U ⇐ matrix;
      for ( i = 0 to n, increase i by 1) {
         luResult.L[i][i] ⇐ 1;

         if (luResult.U[i][i] equal to 0) {
            /******NO LU Decomposition exist******/
            return null;
           End if;
         }
         luResult.L[i][i] ⇐ 1;
```

```
        for ( k⇐1; i + k to n; increase k by 1) {
            if (luResult.U[i + k][i] not equal to 0) {
                fraction ⇐precision(luResult.U[i + k][i] / luResult.U[i][i], prec);
                luResult.L[i + k][i] ⇐ fraction;
                luResult.U[i + k][i] ⇐ 0;
                for ( j ⇐ i + 1 to n, increase j by 1){
                    luResult.U[i + k][j] ⇐ precision((precision(fraction * luResult.U[i][j], prec) * -1) +
luResult.U[i + k][j], prec);
                    luResult.L[i][j] ⇐ 0;
                 End for;
                }
          End if;
            }
          End for;
        }
     End for;
      }
      for ( i⇐ 0 to n increase by 1) {
        System.out.println();
        for ( j ⇐0 to n; increase by 1) {
            System.out.print(luResult.L[i][j] + "   ");
           End for;
        }
    End for;
      }
      System.out.println("\n---------------------");
      for ( i ⇐ 0 to n; increase i by 1 ) {
        System.out.println();
        for ( j ⇐ 0 to n, increase j by 1) {
            System.out.print(luResult.U[i][j] + "   ");
       End for;
        }
 End for;
      }
      System.out.println("\n---------------------");

      z ⇐ forwardSub(luResult.L, results, prec);
      return BackSubstitute(luResult.U, z, prec);
    }

Crout() {
      Z[] ⇒{}
      for ( i ⇐ 0 to n increase by 1) {
        luResult.L[i][0] ⇐ matrix[i][0];
         End for
```

```
}
for ( j = 1 to n increase by 1) {
   luResult.U[0][j] = matrix[0][j] / luResult.L[0][0];
   End for
}
for ( i⇐ 0 to n increase by 1) {
   luResult.U[i][i] ⇐ 1;
  End for
}
for ( i⇐1 to n increase by 1) {
   for ( j ⇐ i to n increase j by 1) {
      luResult.L[j][i] ⇐ matrix[j][i];
      if (j not equal i) luResult.U[i][j] ⇐ matrix[i][j];  End if
      for ( k⇐ 0 to i increase by 1) {
         luResult.L[j][i] - ⇐ precision((luResult.L[j][k] * luResult.U[k][i]), prec);
         precision(luResult.L[j][i], prec);
         if (j not equal to i) {
            luResult.U[i][j] - ⇐ precision((luResult.L[i][k] * luResult.U[k][j]), prec);
            precision(luResult.U[i][j], prec);
             End if
         }
            End for
      }
      if (j not equal i) {
         luResult.U[i][j] /⇐ luResult.L[i][i];
         precision(luResult.U[i][j], prec);
              End if
      }
           End for
   }       End for
}

for ( i ⇐ 0 to n increase by 1) {
   System.out.println();
   for ( j ⇐ 0 to n increase by 1) {
      System.out.print(luResult.L[i][j] + "   ");
           End for
   }   End for
}
System.out.println("\n----------------------");
for ( i ⇐ 0 to n increase by 1) {
   System.out.println();
   for ( j ⇐  0 to n increase by 1) {
      System.out.print(luResult.U[i][j] + "   ");
           End for
   }
```

```
            End for
      }
      System.out.println("\n---------------------");

      z ⇐ forwardSub(luResult.L, results, prec);
      System.out.println("\n---------------------");

      return BackSubstitute(luResult.U, z, prec);
   }
```

## d)Gauss Seidel:

```
  seidelMethod(equ, initial, choice, num, precision) {
        ncol⇐equ[0].length; //no. of columns
        nrow⇐equ.length; //no. of rows
        iter⇐0;
        NoOfIterations⇐0;
        Es⇐0;
        flag=false;
        rounding⇐Math.pow(10, precision);
        switch(choice) {
        case"number of iterations":
                NoOfIterations⇐num;
                break;
        case"relative error":
                Es⇐num;
                      Break;
                      }
        while(flag equal to false) {
                for(i⇐0 to nrow increased by 1) {
                        sum⇐equ[i][ncol-1]; //last col
                        for(j⇐0 to nrow increased by 1) {
                          if(i not equal to j)
                                sum⇐(Math.round(sum*rounding)/rounding)-
((Math.round(initial[j]*rounding)/rounding) * (Math.round(equ[i][j]*rounding)/rounding));
                          end if
                        }
V[i] ⇐
(Math.round(((Math.round(sum*rounding)/rounding)/(Math.round(equ[i][i]*rounding)/roundin
g))*rounding)/rounding);

E[i]<==Math.abs(Math.round(((((Math.round(V[i]*rounding)/rounding)-(Math.round(initial[i]*rou
```

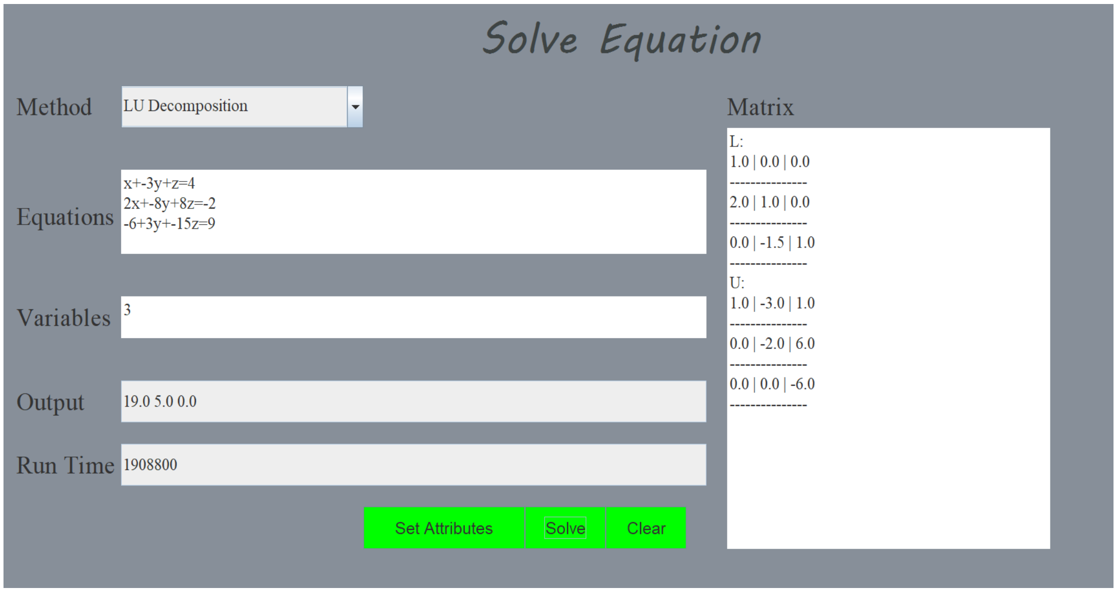
```
nding)/rounding))/(Math.round(V[i]*rounding)/rounding))*100)*rounding)/rounding);

                    Initial[i] ⇐ V[i];
                    }
          if(NoOfIterations equal to num)
                    ++iter;
                    if(iter equal to 1)
                    continue;
                    end if
            if ((iter equal to NoOfIterations) and
(Math.round(g.max_element(E)*rounding)/rounding bigger than 100))
                     System.out.println("diverge");
                     flag⇐true;
                     Break;
                 else if(iter equal to NoOfIterations &&
Math.round(g.max_element(E)*rounding)/rounding less than 50)
                                        flag⇐true;
                                        break;

                 end if
           else if(Es equal to num)
               if (Math.round(max_element(E)*rounding)/rounding  less than Es)
                              flag⇐true;
                              break;
               else
                     ++iter;
                     if(Math.round((max_element(E)*rounding)/rounding bigger than 100) and  (iter
                      bigger than 6))
                     System.out.print("diverage");
                     break;




            end if
      End while
                    return initial;
}
```

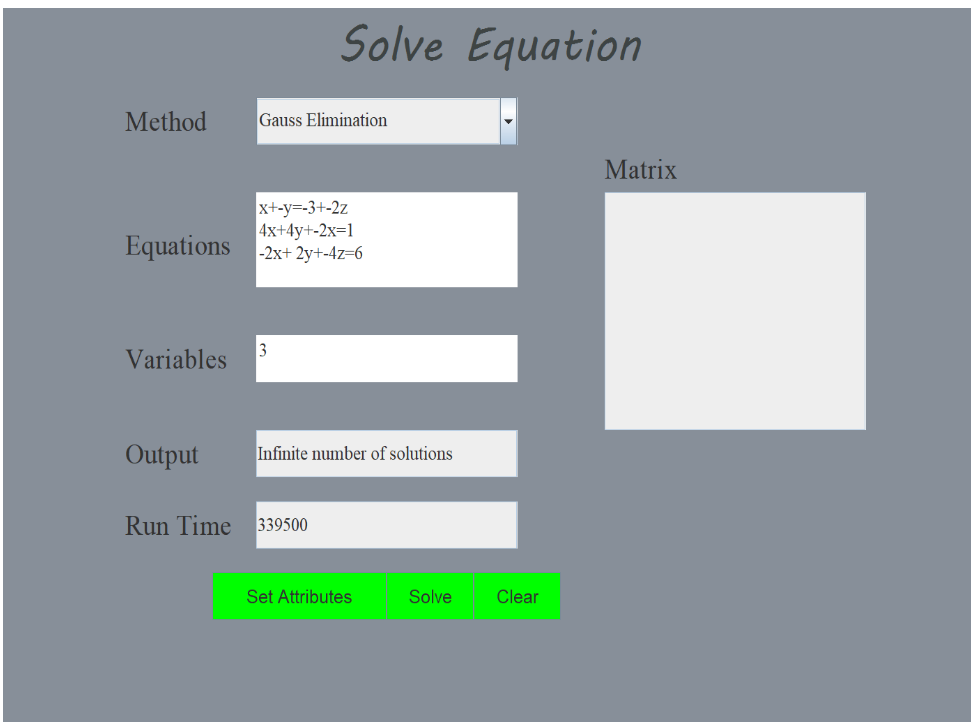
**e)Jacobi Iteration:**

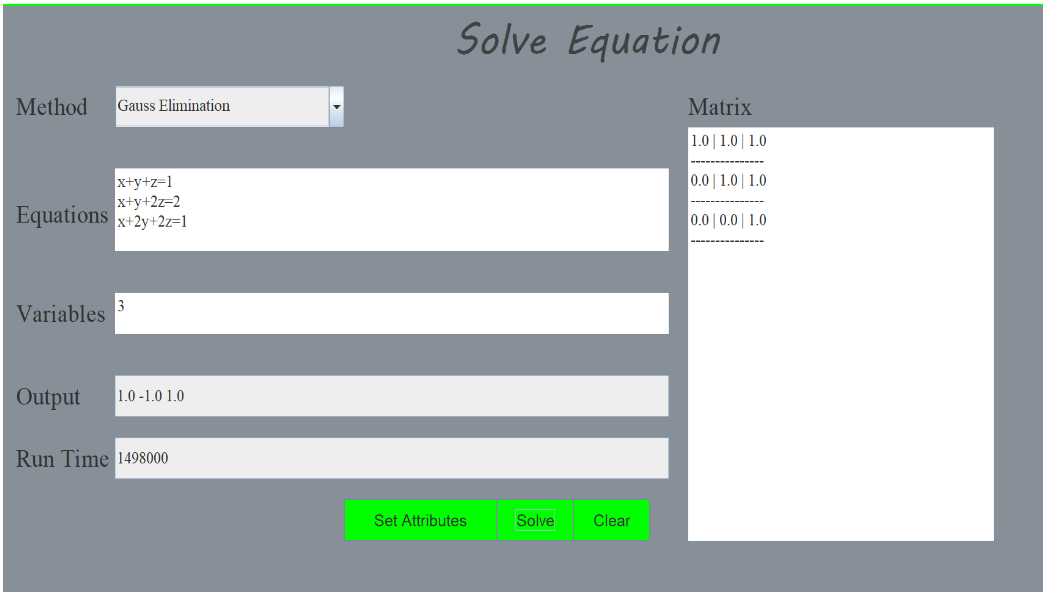
```
JacobiMethod(equ, initial, choice, num, precision) {
        ncol⇐equ[0].length; //no. of columns
        nrow⇐equ.length; //no. of rows
      seidel g⇐new seidel();
        iter⇐0;
        NoOfIterations⇐0;
        flag⇐false;
        rounding⇐Math.pow(10, precision);
        switch(choice) {
        case"number of iterations":
                NoOfIterations⇐num;
                break;
        case"relative error":
                Es⇐num;
                break;
        default:
                System.out.print("error..");
        }
        while(flag equal to false) {
                for(i⇐0 to nrow increased by 1) {
                        sum⇐equ[i][ncol-1]; //last column
                        for(j⇐0 to nrow increased by 1) {
                         if(i not equal to j)

sum⇐(Math.round(sum*rounding)/rounding)- ((Math.round(initial[j]*rounding)/rounding) *
(Math.round(equ[i][j]*rounding)/rounding));
                         else
                                continue;
                         end if
                        }
V[i]<==(Math.round(((Math.round(sum*rounding)/rounding)/(Math.round(equ[i][i]*rounding)/ro
unding))*rounding)/rounding);

E[i]<==Math.abs(Math.round(((((Math.round(V[i]*rounding)/rounding)-(Math.round(initial[i]*rou
nding)/rounding))/(Math.round(V[i]*rounding)/rounding))*100)*rounding)/rounding);

                }
           for(r⇐0 to nrow increased by 1) {
             initial[r]<== (Math.round(V[r]*rounding)/rounding); //to assign result from iteration
         }
```

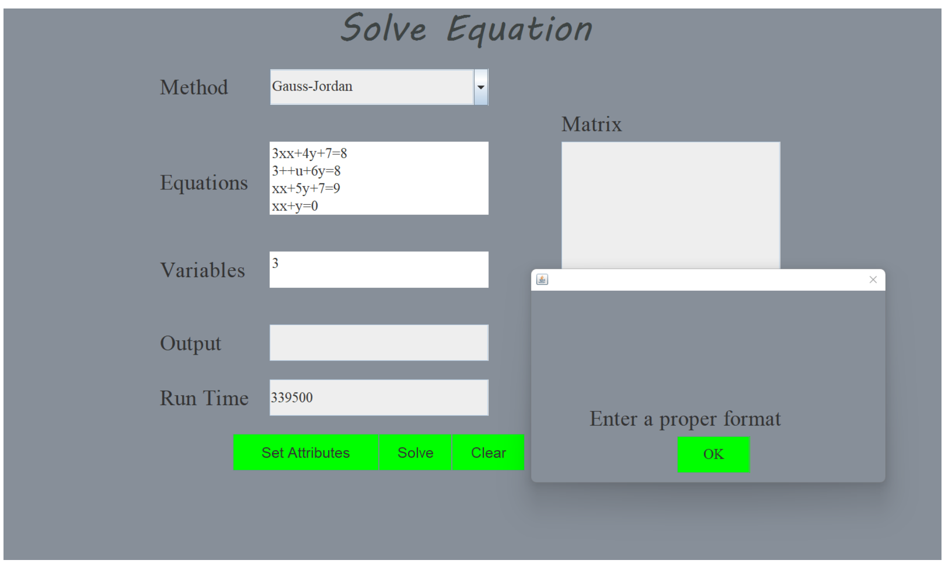
```
if(NoOfIterations equal to num)
                    ++iter;
                    if(iter equal to 1)
                    continue;
                    end if
            if ((iter equal to NoOfIterations) and
(Math.round(g.max_element(E)*rounding)/rounding bigger than 100))
                     System.out.println("diverge");
                     flag⇐true;
                     Break;
                 else if(iter equal to NoOfIterations &&
Math.round(g.max_element(E)*rounding)/rounding less than 50)
                                        flag⇐true;
                                        break;

                 end if
else if(Es equal to num)
               if (Math.round(max_element(E)*rounding)/rounding  less than Es)
                              flag⇐true;
                              break;
            else
                    ++iter;
                    if(Math.round((max_element(E)*rounding)/rounding bigger than 100) and  (iter
                     bigger than 6))
                    System.out.print("diverge");
                    break;

 end if
      End while
                    return initial;
          }
```

## 3. Sample runs for each method:

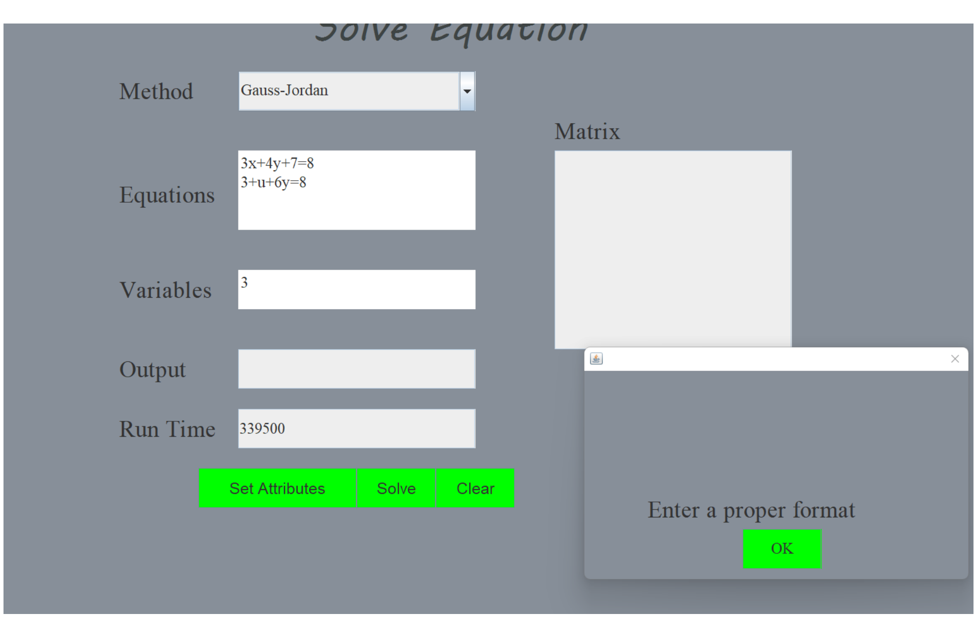




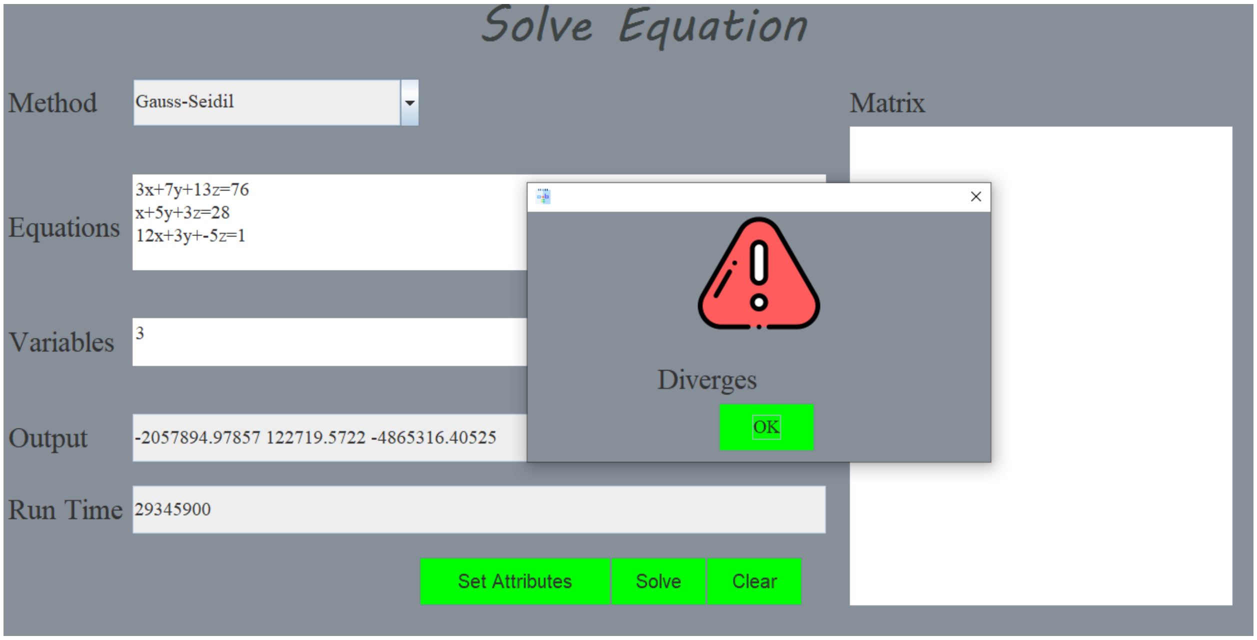




Error because of the name of the variable and many + signs

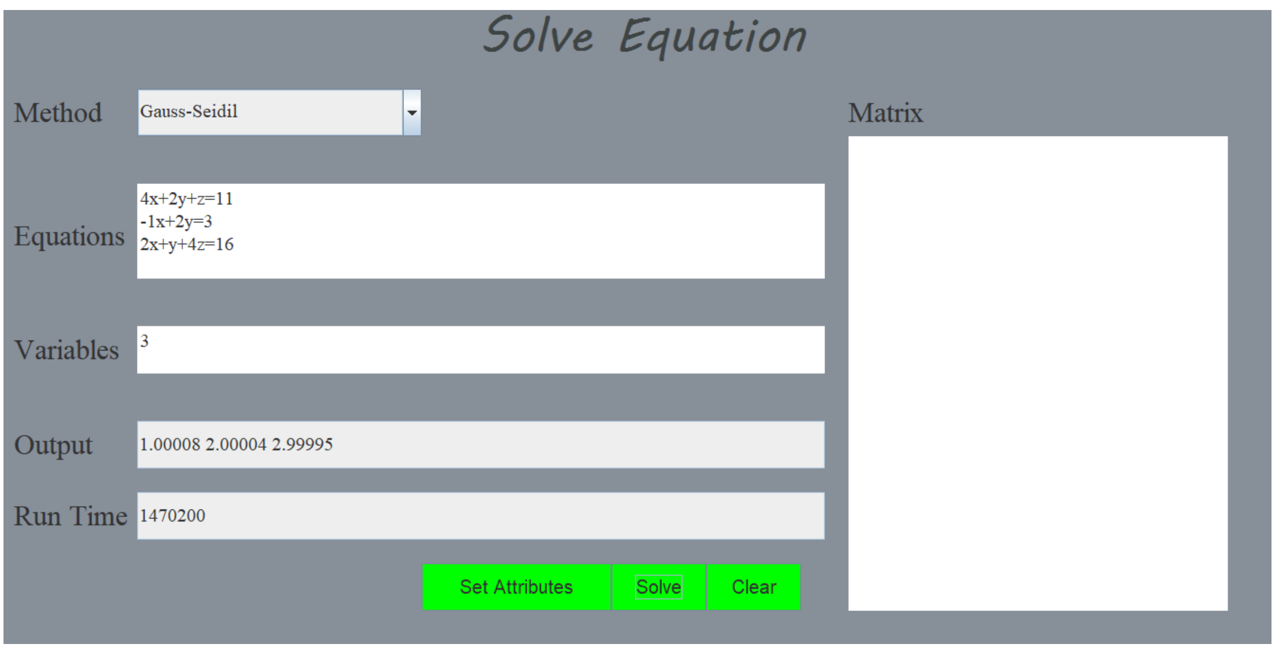



Number of equations less than number of variables

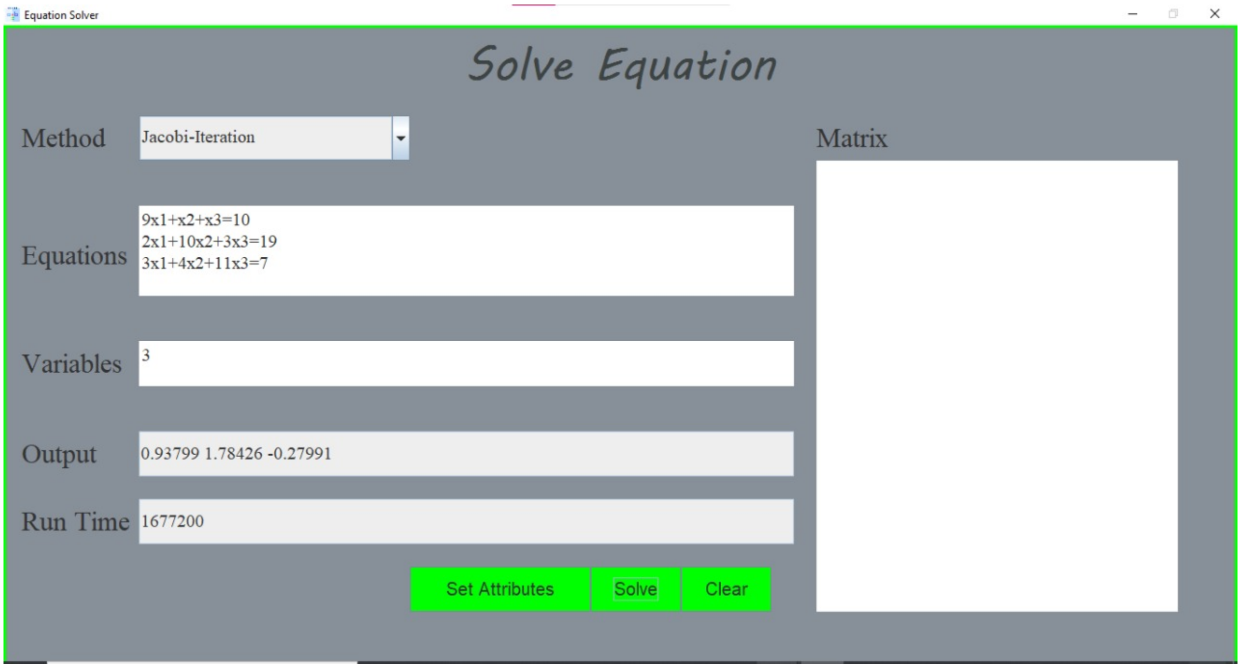
## Gauss Seidel:



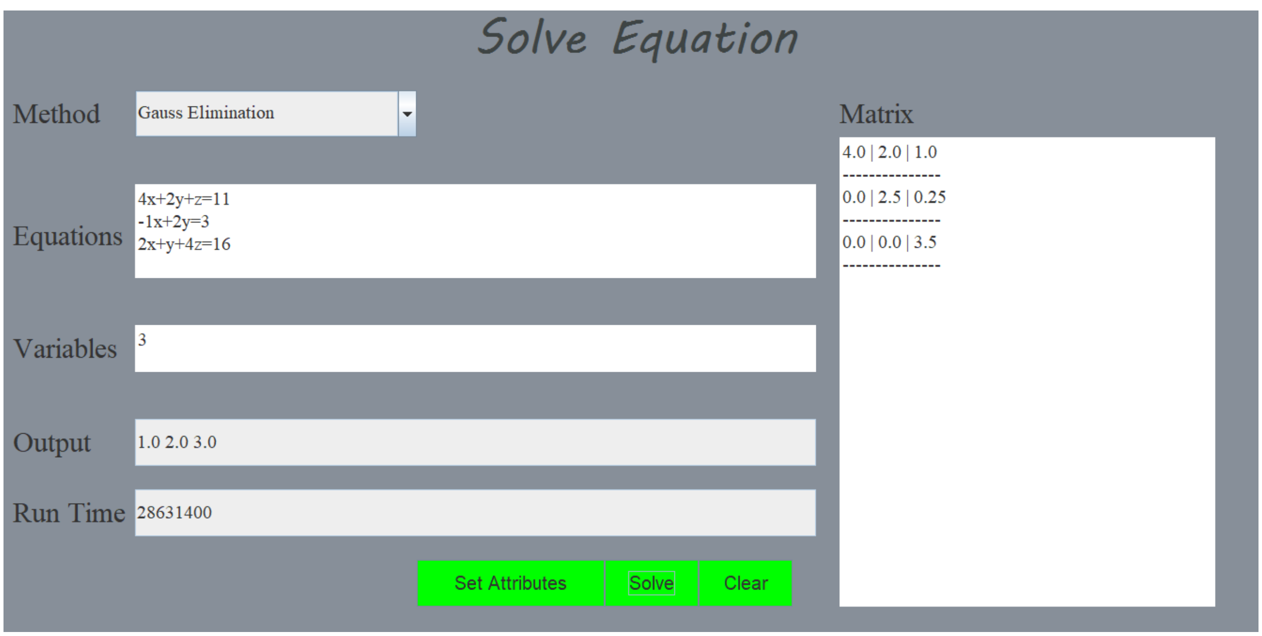

**With 6 iterations and 1 0 1 initial values.**



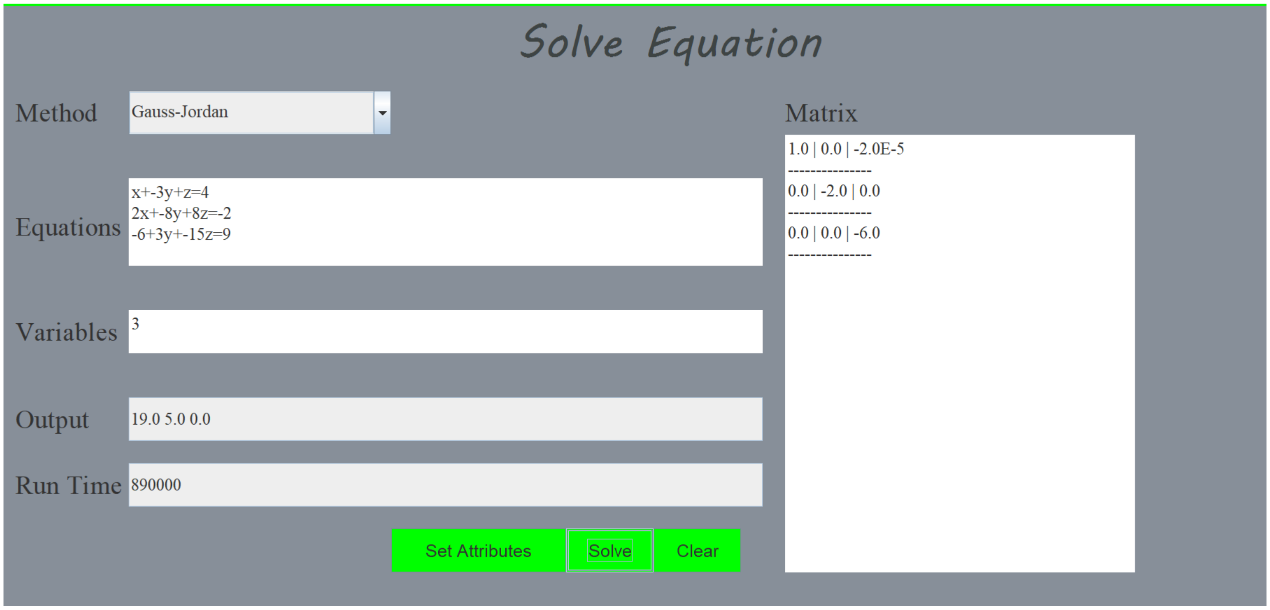

**Gauss Seidel with 0.1 absolute relative error and 1 1 1 initial values.**

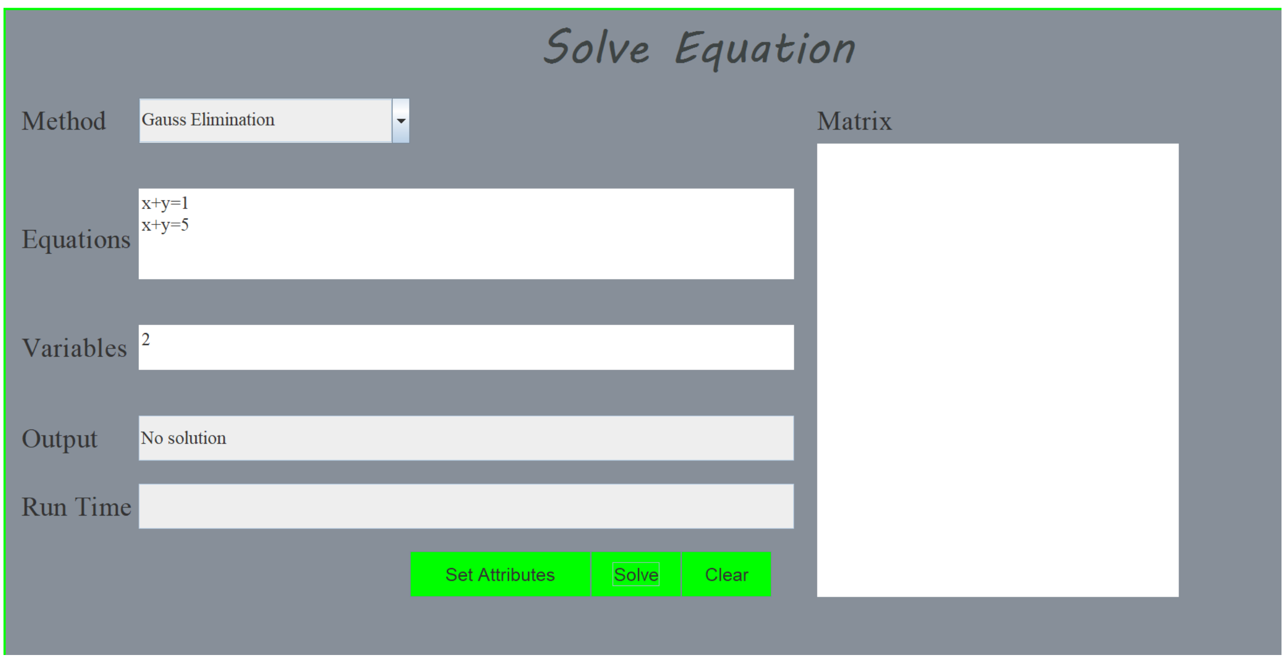



**Jacobi-6 iterations with initial 0,0,0**




**Gauss Elimination with the matrix after the forward elimination.**


Solve Equation
Method
Gauss-Jordan
Matrix
1.0 | 0.0 | -2.0E-5
---------------
0.0 | -2.0 | 0.0
---------------
0.0 | 0.0 | -6.0
---------------
Equations
x+-3y+z=4
2x+-8y+8z=-2
-6+3y+-15z=9
Variables
3
Output
19.0 5.0 0.0
Run Time
890000
Set Attributes
Solve
Clear


Solve Equation
Method
Gauss Elimination
Matrix
Equations
x+y=1
x+y=5
Variables
2
Output
No solution
Run Time
Set Attributes
Solve
Clear

## 4. Comparison between different methods (time complexity , convergence, best and approximate errors):

| | LU decomposition |
|---|---|
| Time complexity | • Any type of matrix (#N)<br>• **total= O(N^3)+cO(N^2)** |
| Best case | It is most suitable to solve the solutions that has many variables to be solved in the same matrix. |
| Worst Case | Number of equations are too large<br>When no partial pivoting applies |

| | Gauss Elimination with pivoting |
|---|---|
| Time complexity | • O(N)<br>• O(N^2) maximum vector |
| Best case | It is useful in minimizing the rounding off error because of the pivoting and the scaling. |
| Worst Case | Singular square Matrix<br>Round off error when the number of equations is very large. |
| Precision | It doesn't affect much from significant figures. |

| | Gauss Jordan with pivoting |
|---|---|
| Time complexity | • Total=(4n^3)/3<br>• It costs a lot when n is very large (cost=2((2N^3)/3) |
| Best case | It is useful in solving linear equations with a finite number of operations. |
| Worst Case | Singular square Matrix |

| | Jacobi iteration Method |
|---|---|
| Time complexity | O(N^2) |
| convergence | When the criteria that is determined before solving the equation is fulfilled the iteration will be stopped.<br>It is hard to determine the number of iterations through solving ,so it is better to determine them before solving. |
| Worst Case | Reaching the maximum number of solutions (divergence of the solution) so the program will be stopped. |
| pitfalls | Before solving the equation, we can't be sure if we will reach a solution or it will diverge. |
| Best Case | • Relative error of the value decreased.<br>• It converges before even reaching the determined number of iterations |
| Precisions | • Needing more correct significant figures means that the system will continue the alterations until fulfilling the determined condition. |

| | Gauss Seidel iteration method |
|---|---|
| Time complexity | O(N^2) |
| convergence | • It is better than the jacobi method because of the guarantee that convergence will happen before solving the equation.<br>• In case of diverging of the method then the number of iterations must be determined. |
| Worst Case | • Reaching the maximum number of solutions (divergence of the solution) so the program will be stopped. |

| | |
|---|---|
| | • Matrix isn't diagonally dominant |
| Best Case | • In the case of Diagonally dominant matrix, the system will converge for sure before reaching a determined number of iterations. |
| Precisions | • Needing more correct significant figures means that the system will continue the alterations until fulfilling the determined condition. |

## 5. Data structure used:

**1-Validation and evaluation of coefficients:**

1. **Arrays:** It was used in splitting the equation(String) that we took from the front (input box in gui). It was very useful as the number of the variables was. It can be multidimensional so it was really helpful to extract the coefficients from the equation to form the 2d matrix **(known size)** that will be solved using the methods.

2. **ArrayList:** it's size is really flexible, so it was used to store the coefficients at first before summing up the coefficients of the same variable. It was used also to store all the positions of the plus sign and variables to know the shape of the variable in the equation.

3. **Linked Hash Set:** it was the most helpful one as I used it to store variables of the equations and remove all the repetition of them because any kind of set has a property that stores the data without repetition. It is also a doubly linked list so iterating through it is very easy.

**2-methods:**

1. **Array:** because it can be multidimensional so it was the most suitable one to store the coefficients and work with them in scaling. The size of the matrix was also known..

[Link to the demo video.]