Alexandria University

Faculty of Engineering

Computer and Systems Engineering Dept.

Numerical Analysis

Project Phase 1

نور الدين حسن محمد قباري حسن ID: 19016802
عمر مصطفى حسن متولى صفا ID: 19016099
مصطفى عبد الحليم عبد المحسن عبد الحليم الطوى ID: 19016662
مصطفى فراج مصطفى محمد فراج ID: 19016666
جمال عبد الحميد ناصف نويصر ID: 19015550

➢ Pseudo code

```
Begin union (list_1, list_2):
  copy_1 ← copy list_1
  FOR each element in list_2:
    IF element not in list_1:
      add element to copy_1
    END IF
  END FOR
  RETURN copy_1
END union ()

Begin scale (coefficient_matrix, constants_vector, start_index):
  copy_coeff ← coefficient_matrix
  IF constants_vector provided:
    copy_const ← constants_vector
  END IF
  FOR each row in copy_coeff starting from start_index:
    row_max ← absolute max value in row
    FOR each column in copy_coeff starting from start_index:
      divide coeff in copy row, column by row_max
    END FOR
    IF constants_vector provided:
      divide constant in copy row by row_max
    END IF
  END FOR
  IF constants_vector provided:
    RETURN copy_coeff, copy_const
  END IF
  RETURN copy_coeff
END scale()
```

```
Begin partial_pivot (coefficient_matrix, column_index):
  positions ← list of numbers [0, number of rows of matrix[
  max_val ← 0
  max_index ← 0
  FOR each rows starting from column_index:
    IF absolute value of element in row, column_index greater than absolute value of max_val:
      max_val ← element
      max_index ← row index of element
    END IF
  END FOR
  swap positions[column_index] and positions[max_index]
  RETURN positions
```

```
END partial_pivot()
```

```
Begin normalize (number):
  IF number is zero:
    return 0.0
  END IF
  shifts ← 0
  IF absolute value of number > 1:
    WHILE absolute value of number > 1:
      divide number by 10
      add 1 to shifts
    END WHILE
  ELSE IF absolute value of number < 0.1:
    WHILE absolute value of number < 0.1:
      multiply number by 10
      subtract 1 from shifts
    END WHILE
  END IF
  RETURN number, shifts
END normalize()
```

Begin chop_to_n_digits (number, precision)
  number, shifts ← normalize(number)
  IF number is positive:
    RETURN $round\left(number - 0.5 \times 10^{-1 \times precision}, precision\right) \times 10^{shifts}$
  IF number is negative:
    RETURN $round\left(number + 0.5 \times 10^{-1 \times precision}, precision\right) \times 10^{shifts}$
  IF number is zero:
    RETURN 0.0
END chop_to_n_digits()

Begin round_to_n_digits (number, precision):
  number, shifts ← normalize(number)
  RETURN $round\left(number \times 10^{-1 \times precision}, precision\right) \times 10^{shifts}$
    END round_to_n_digits()

```
Begin update_positions():
  FOR each rows in coefficient_matrix; index i:
    IF row not equal matrix[position_list[i]]:
      swap row with matrix[position_list[i]]
      swap constant[i] with constant_vector[position_list[i]]
      position_list ← list of numbers [0, rows of matrix[
    END IF
  END FOR
```

END update_positions()

Begin check_empty_rows():
  FOR each rows in coefficient_matrix:
    IF absolute row max is zero:
      THROW exception
    END IF
  END FOR
END check_empty_rows()

Begin eliminate():
  FOR each diagonal_elements in coefficient_matrix:
    positions_list ← partial_pivot(scale(coefficient_matrix, start_index = index of diagonal_element))
    update_positions()
    FOR each rows below diagonal_element:
      IF element in diagonal_element column is zero:
        skip row
      END IF
      multiplier ← $\frac{element\ in\ row,\ \ column\ of\ diagonal\ element}{diagonal\ element}$
      FOR each columns in coefficient_matrix:
        element in row, column ← element in row, column – multiplier x element in diagonal_element row, column
      END FOR
      constant in row ← constant in row – constant in diagonal_element row x multiplier
    END FOR
    TRY check_empty_rows()
    CATCH exception:
      THROW exception
    END CATCH
  END FOR
END eliminate()

Begin substitute():
  variable_list[last] ← $\frac{constant[last]}{coefficient\_matrix[last\ row][last\ column]}$
  FOR each row in coefficient_matrix except last; index i:
    sum ← constant_list[i]
    FOR each column in coefficient_matrix > i; index j:
      sum ← sum – coefficient_matrix[i][j] x variable_list[j]
    END FOR
    variable_list[i] ← $\frac{sum}{coefficient\_matrix[i][i]}$
  END FOR
END substitute()

Begin solve():
  TRY eliminate()
  CATCH exception:
    THROW exception
  END CATCH
  substitute()
  RETURN variable_list

```
END solve()
```

LU Controller:

```
Constructor (method, A [] [], B [], converter:FloatConverter):
def solve(self):
  solver=LUDecomposerService(A,B,converter)
 Begin Solve ():
  Begin if (if method entered is Doolittle):
            execute Doolittle algorithm
     elseif (if method entered is  crout):
            execute crout algorithm
  End if
End solve
```

LUDecomposerService:

```
Constructor (a [] [], b [], converter:FloatConverter):
Begin findScalers():
  n=len(a)
  o=[0]*n
  s=[0]*n
  for i in range (0,n):
    o[i]=i
    s[i]=abs(a[i][0])
    for j in range(1,n):
      Begin if(abs(self.__a[i][j])>s[i]):
        s[i]=abs(self.__a[i][j])
  return o,s
End findScalers ()
```

```
Begin pivoting (scalers, o, k):
    pivot = k
    N=len(a)
```

```
        biggestPivot = converter.convert(abs(a[o[k]][k]) / float(scalers[k]))
        Begin FOR:
          temp = converter.convert(abs(self.__a[o[i]][k]) / float(scalers[o[i]]))
         Begin IF:
                If (temp > biggestPivot):
                pivot = i
                 biggestPivot = temp
         End IF
        End FOR
         temp=o[pivot]
         o[pivot]=o[k]
         o[k]=temp
End pivoting
  Begin forward_eliminate():
    n=len(a)
    o, s = findScalers()
    Begin For:
      Getting biggest pivot if exist and swapping rows of array o
      Begin For:
        mult = converter.convert(float(a[o[i]][k])/float(a[o[k]][k]))
        a[o[i]][k]=mult
        Begin For:
          a[o[i]][j]=converter.convert(a[o[i]][j]-converter.convert(mult*a[o[k]][j]))
        End For
       End For
    End For
    return o
  End forward_eliminate()
Begin forwardSubstitution(o):
```

```
    n=len(a)
    y=[0]*n
    assigning first element of vector b to first element of vector of y
    Begin For
      value= b[o[i]]
      Begin For
        Evaluating element number i in vector y
      End For
      y[o[i]] = value
    End For
    return y
End forwardSubstitution
```

```
Begin backSubstitution(y,o):
    n=len(a)
    x = [0.0] * n
    assign last element of vector x to last of vector y
    Begin For
      sum=0
      Begin For
        Evaluating part of equation in every row
      End For
      Getting elements of vector x
      End For
    End For
   return x
End backSubstitution
```

```
Begin croutFormation():
```

```
    n = len(a)
    Begin For:
      Getting first row (u12,u13,u14,…)
    End For
    Begin For:
      Begin For:
        sum=0
        k=0
        m=0
        Begin While:
          Evaluating part of the expression
          k=k+1
          m=m+1
        if(i<j):
          getting Uij
        else:
          get Lij
        End While
      End For
    End For
  End croutFormation ()
```

```
Begin croutForwardSubstitution ():
      n = len(a)
      y = [0.0] * n
      getting first element of vector y
      Begin For:
        sum = 0
        Begin For:
          Evaluating part of the expression
```

```
          End For
           storing value of y
       End For
         return y
   End croutForwardSubstitution()
```

```
Begin croutBackSubtitution(y):
      n = len(a)
      x=[0.0]*n
      assign last value of x to last value of y
      Begin For
         value = y[i]
         Begin For
            Evaluating value of expression
         End For
         Storing the value of x
      End For
      return x
    End croutBackSubtitution
```

```
Begin DooLittle_Decomposition():
     o = forward_eliminate()
     y = forwardSubstitution(o)
     x = backSubstitution(y, o)
   End DooLittle_Decompostion
```

```
Begin croutDecomposition():
         croutFormation()
         y = croutForwardSubstitution()
         x = croutBackSubtitution(y)
           End croutDecompostion
```

```
Gauss_Jordan:
  Gauss_Jordan_Constructor(A, b, float_converter: FloatConverter):
  solve():
        try:
      elimination()
      throw error if there is
    normalize()
    return answers
```

```
  update_positions():
    for i = 0 to the number of coeffeicient matrix rows:
      if i != positions[i]:
        swap A[i] and A[positions[i]]
        swap b[i] and b[positions[i]]
        positions[positions[i]] = positions[i]
        positions[i] = i
        break
  checkEmptyRow(rowIndex):
        sum = 0
        for i = 0 to the number of coeffeicients in A[rowIndex]:
        sum = sum + absolute(A[rowIndex][i])
    if sum == 0:
      raise ValueError("Error, empty row exists!")
  elimination():
     for i = 0 to the number of coeffeicient matrix rows:
      positions = partial_pivot(scale(A, converter, start_index=i), i)
      update_positions()
      for j = 0 to the number of coeffeicient matrix rows:
        if i == j:
          continue
        factor =  converter.convert(A[j][i] / A[i][i])
        for k = i to the number of coeffeicient matrix rows:
          A[j][k] = converter.convert(A[j][k] - converter.convert(factor * A[i][k]))
        try:
          checkEmptyRow(j)
          throw error if there is
          b[j] = converter.convert(b[j] - factor * b[i])
  normalize():
    for i = 0 to the number of coeffeicient matrix rows:
      b[i] = converter.convert(b[i] / A[i][i])
```

Begin class GaussSeidil:

Begin function getAbsoluteRelativeError(self, newValue, oldValue):

```
        Begin if (newValue == oldValue):
            return 0
        End if
        Begin if (newValue == 0):
            return 100
        End if
        ans ← self.converter.convert( abs(self.converter.convert(self.converter.convert(
        newValue – oldValue ) / newValue ) ) )
        return self.converter.convert( ans )
    End function getAbsoluteRelativeError
```

```
    Begin function Solve (self):
        finished ← false
        Begin for iteration = 0 to (self.iterations):
            maxRelativeError ← 0
            oldX ← self.newX
            Begin for i = 0 to len(self.newX):
                tempSum ← 0
                Begin for j = 0 to len(self.newX):
                    Begin if (i = j):
                        continue
                    End if
                    tempSum ← self. converter.convert ( tempSum +
                    self.converter.convert( self.A[i][j] * self.newX[j] ) )
                End for
                self.newX[i] ← self.converter.convert( self.converter.convert(self.B[i] -
                tempSum) / self.A[i][i] )
            End for
            Begin for i = 0 to len(self.newX):
                relativeError ← self.getAbsoluteRelativeError(self.newX[i], oldX [i])
```

```
                    Begin if ( maxRelativeError < relativeError ):
                        maxRelativeError ← relativeError
                    End if
                End for
                Begin if ( maxRelativeError < self.eps ):
                    finished ← True
                    break
                End if
            End for
            Begin for i = 0 to len(self.newX):
                Begin if(abs(self.newX[i] - oldX[i]) >= 10 to power 10):
                    MessageError("Error! Diverge!!")
                End if
            End for
            return self.newX
        End solve
```

End class GaussSeidil

Begin class Jacobi:

```
        Begin function getAbsoluteRelativeError(self, newValue, oldValue):
            Begin if (newValue == oldValue):
                return 0
            End if
            Begin if (newValue == 0):
                return 100
            End if
            ans ← self.converter.convert( abs(self.converter.convert(self.converter.convert(
            newValue – oldValue ) / newValue ) ) )
            return self.converter.convert( ans )
        End function getAbsoluteRelativeError
```

```
Begin function Solve (self):

    finished ← false

    Begin for iteration = 0 to (self.iterations):

        maxRelativeError ← 0

        oldX ← self.newX

        Begin for i = 0 to len(self.newX):

            tempSum ← 0

            Begin for j = 0 to len(self.newX):

                Begin if (i = j):

                    continue

                End if

                tempSum ← self. converter.convert ( tempSum +
                self.converter.convert( self.A[i][j] * oldX[j] ) )

            End for

            self.newX[i] ← self.converter.convert( self.converter.convert(self.B[i] -
            tempSum) / self.A[i][i] )

        End for

        Begin for i = 0 to len(self.newX):

            relativeError ← self.getAbsoluteRelativeError(self.newX[i], oldX [i])

            Begin if ( maxRelativeError < relativeError ):

                maxRelativeError ← relativeError

            End if

        End for

        Begin if ( maxRelativeError < self.eps ):

            finished ← True

            break

        End if

    End for

    Begin for i = 0 to len(self.newX):
```

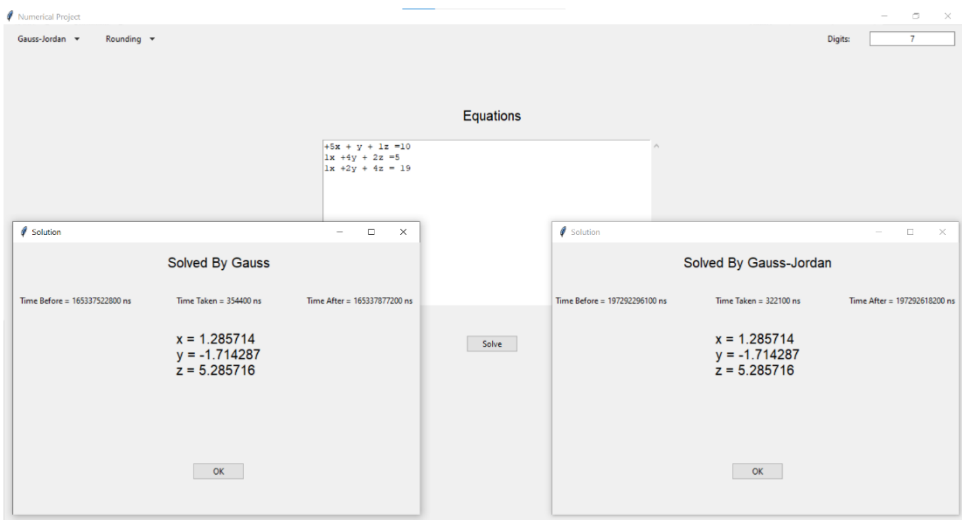
```
        Begin if(abs(self.newX[i] - oldX[i]) >= 10 to power 10):
            MessageError("Error! Diverge!!")
        End if
    End for
    return self.newX
End solve
```

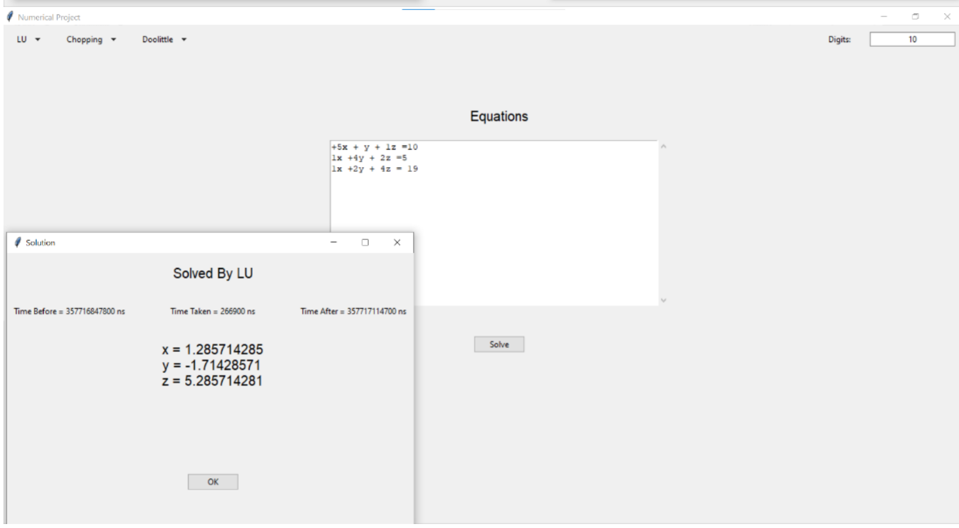
End class Jacobi

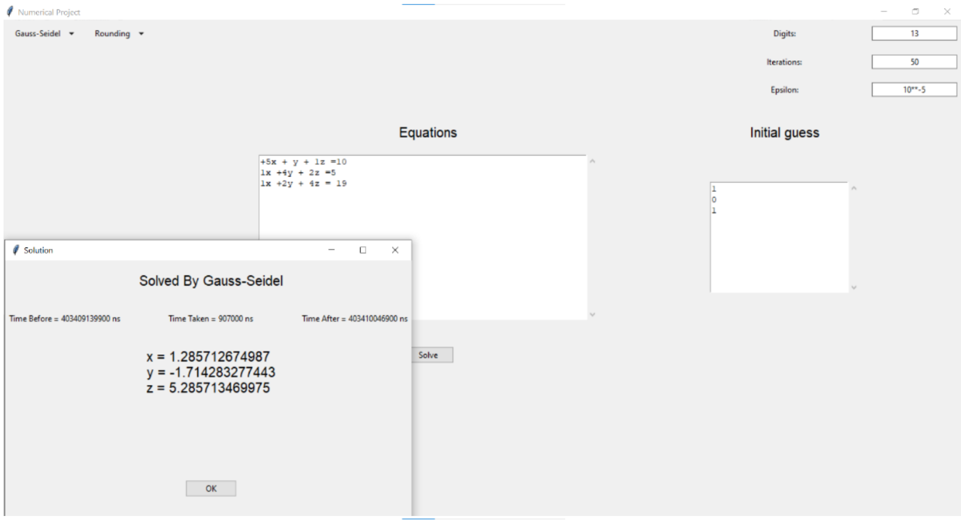
➢ Assumptions:

1. We do not sort the equations in Gauss Seidel nor Jacobi
2. Coefficients must be on the left of the variables
3. The initial guess must be entered in the same order of the unknowns and each number in a separate line.
4. We solve only unique solution equations

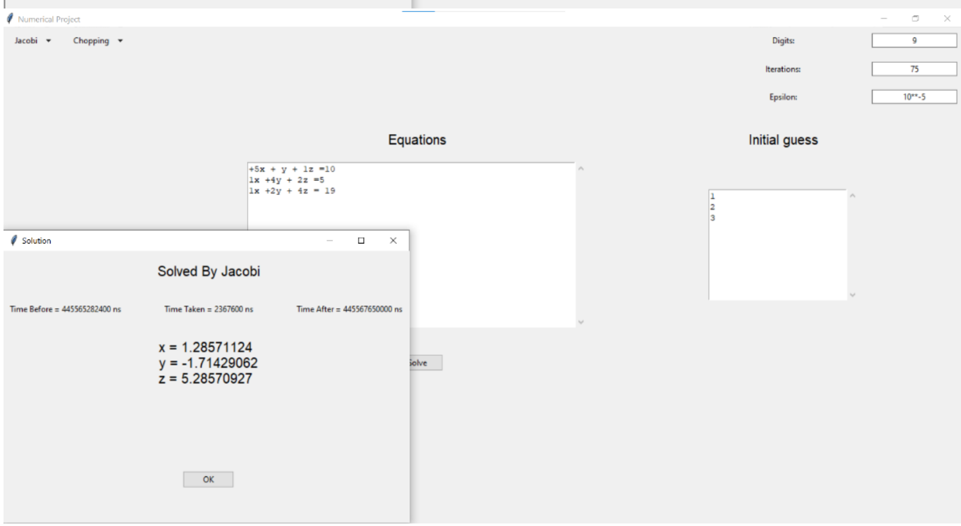
➢ Sample runs


Numerical Project
Gauss-Jordan
Rounding
Digits:
7
Equations
+5x + y + 1z =10
1x +4y + 2z =5
1x +2y + 4z = 19
Solve
Solution
Solved By Gauss
Time Before = 165337522800 ns
Time Taken = 354400 ns
Time After = 165337877200 ns
x = 1.285714
y = -1.714287
z = 5.285716
OK
Solution
Solved By Gauss-Jordan
Time Before = 197292296100 ns
Time Taken = 322100 ns
Time After = 197292618200 ns
x = 1.285714
y = -1.714287
z = 5.285716
OK

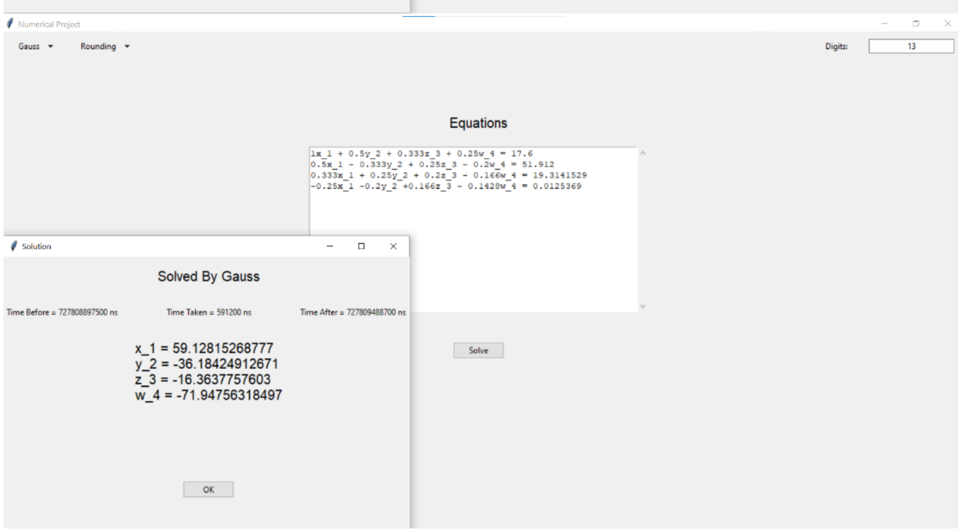

Numerical Project
LU
Chopping
Doolittle
Digits:
10
Equations
+5x + y + 1z =10
1x +4y + 2z =5
1x +2y + 4z = 19
Solve
Solution
Solved By LU
Time Before = 357716847800 ns
Time Taken = 266900 ns
Time After = 357717114700 ns
x = 1.285714285
y = -1.71428571
z = 5.285714281
OK



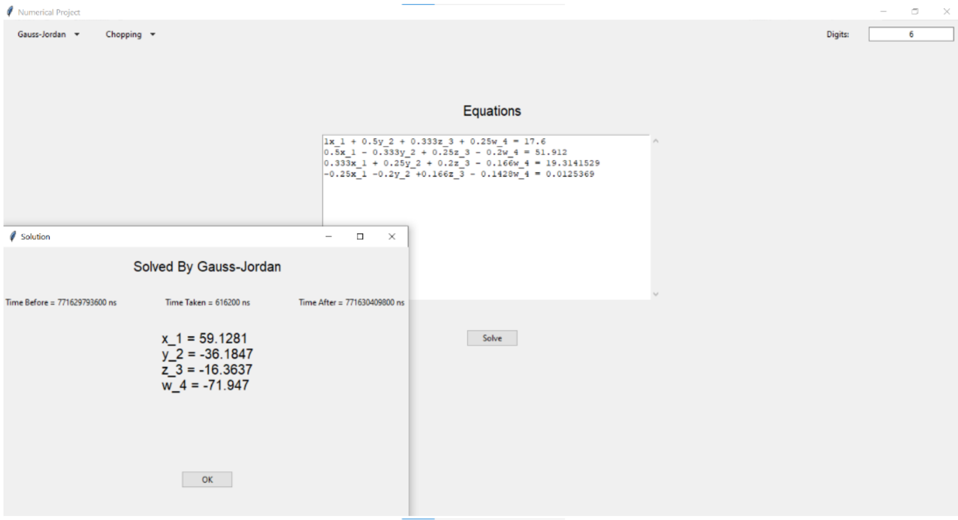
Numerical Project
Gauss-Seidel
Rounding
Digits:
13
Iterations:
50
Epsilon:
10**-5
Equations
+5x + y + 1z =10
1x +4y + 2z =5
1x +2y + 4z = 19
Initial guess
1
0
1
Solution
Solved By Gauss-Seidel
Time Before = 403409139900 ns
Time Taken = 907000 ns
Time After = 403410046900 ns
x = 1.285712674987
y = -1.714283277443
z = 5.285713469975
OK
Solve

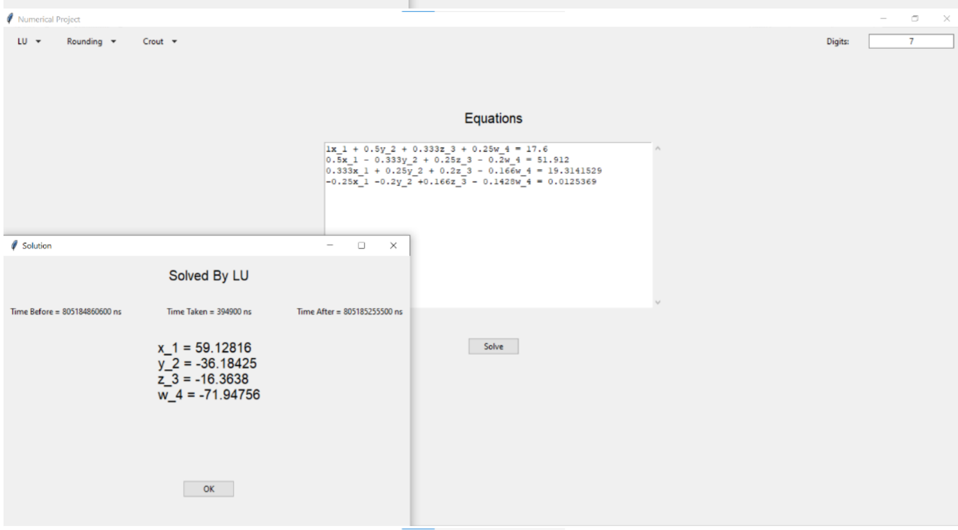


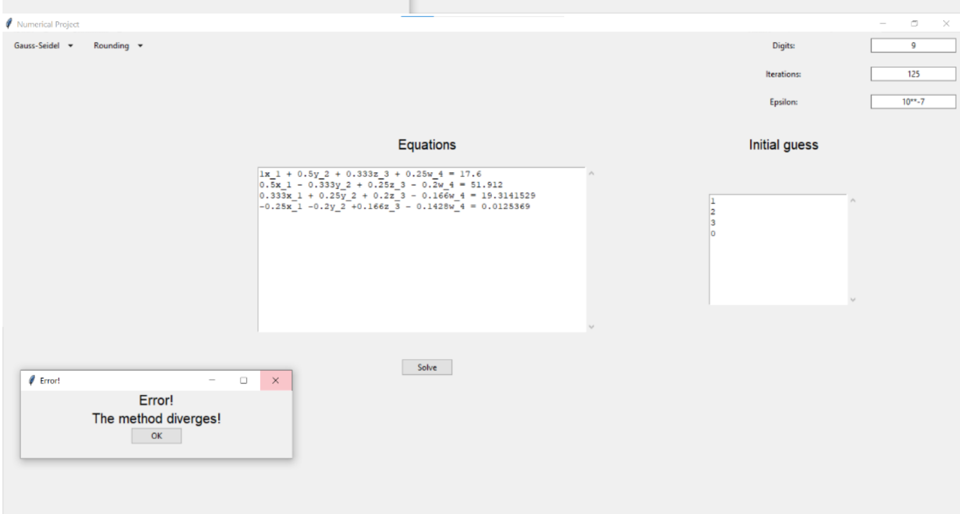
Numerical Project
Jacobi
Chopping
Digits:
9
Iterations:
75
Epsilon:
10**-5
Equations
+5x + y + 1z =10
1x +4y + 2z =5
1x +2y + 4z = 19
Initial guess
1
2
3
Solution
Solved By Jacobi
Time Before = 445565282400 ns
Time Taken = 2367600 ns
Time After = 445567650000 ns
x = 1.28571124
y = -1.71429062
z = 5.28570927
OK

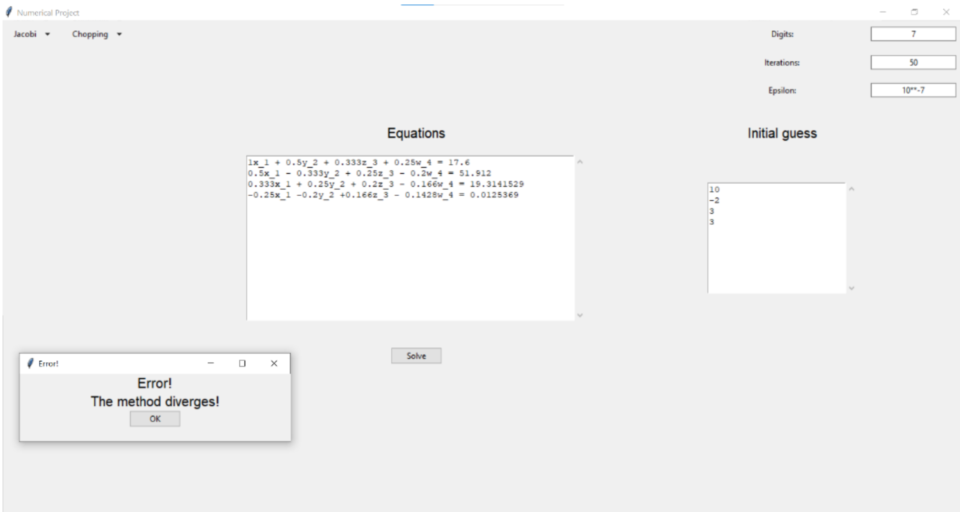


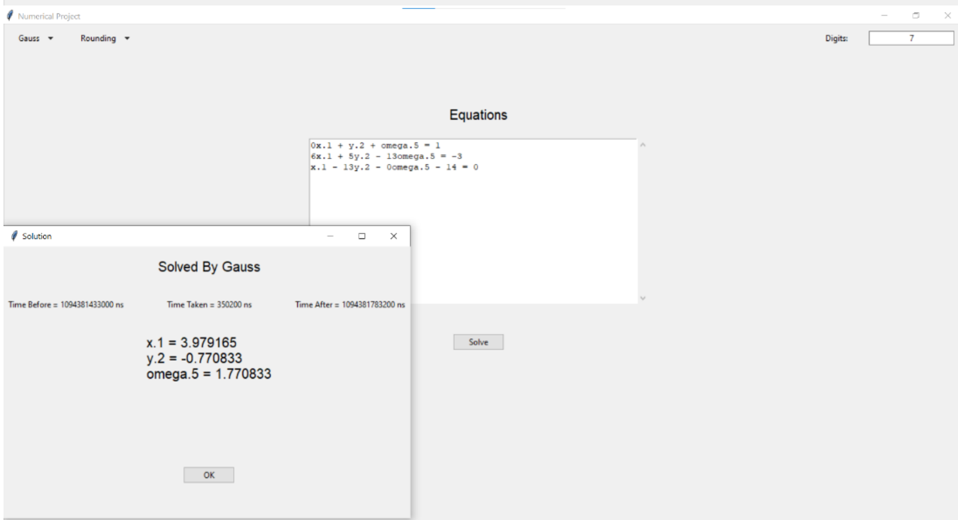
Numerical Project
Gauss
Rounding
Digits:
13
Equations
1x_1 + 0.5y_2 + 0.333z_3 + 0.25w_4 = 17.6
0.5x_1 - 0.333y_2 + 0.25z_3 - 0.2w_4 = 51.912
0.333x_1 + 0.25y_2 + 0.2z_3 - 0.166w_4 = 19.3141529
-0.25x_1 -0.2y_2 +0.166z_3 - 0.1428w_4 = 0.0125369
Solution
Solved By Gauss
Time Before = 727808897500 ns
Time Taken = 591200 ns
Time After = 727809488700 ns
x_1 = 59.12815268777
y_2 = -36.18424912671
z_3 = -16.3637757603
w_4 = -71.94756318497
OK
Solve

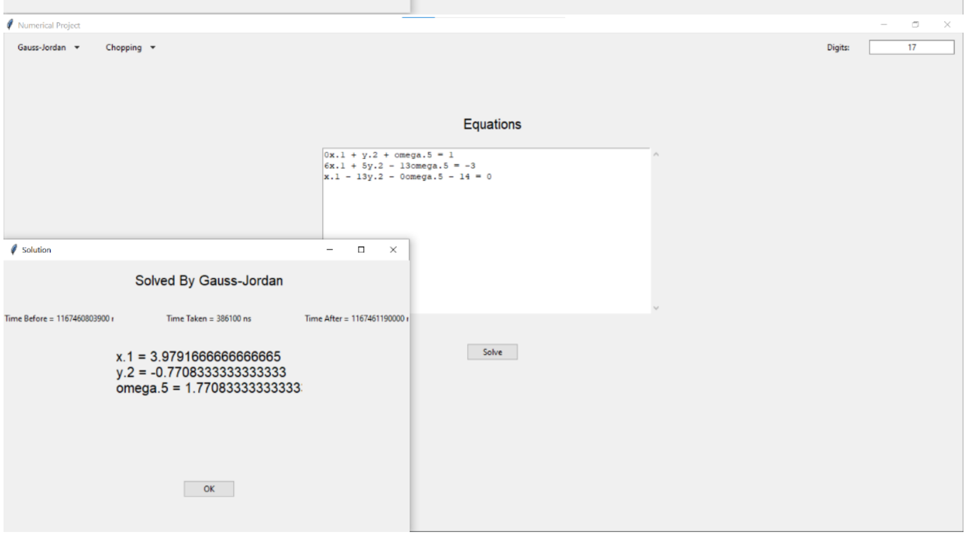

Numerical Project
Gauss-Jordan
Chopping
Digits:
6
Equations
1x_1 + 0.5y_2 + 0.333z_3 + 0.25w_4 = 17.6
0.5x_1 - 0.333y_2 + 0.25z_3 - 0.2w_4 = 51.912
0.333x_1 + 0.25y_2 + 0.2z_3 - 0.166w_4 = 19.3141529
-0.25x_1 -0.2y_2 +0.166z_3 - 0.1428w_4 = 0.0125369
Solve
Solution
Solved By Gauss-Jordan
Time Before = 771629793600 ns
Time Taken = 616200 ns
Time After = 771630409800 ns
x_1 = 59.1281
y_2 = -36.1847
z_3 = -16.3637
w_4 = -71.947
OK

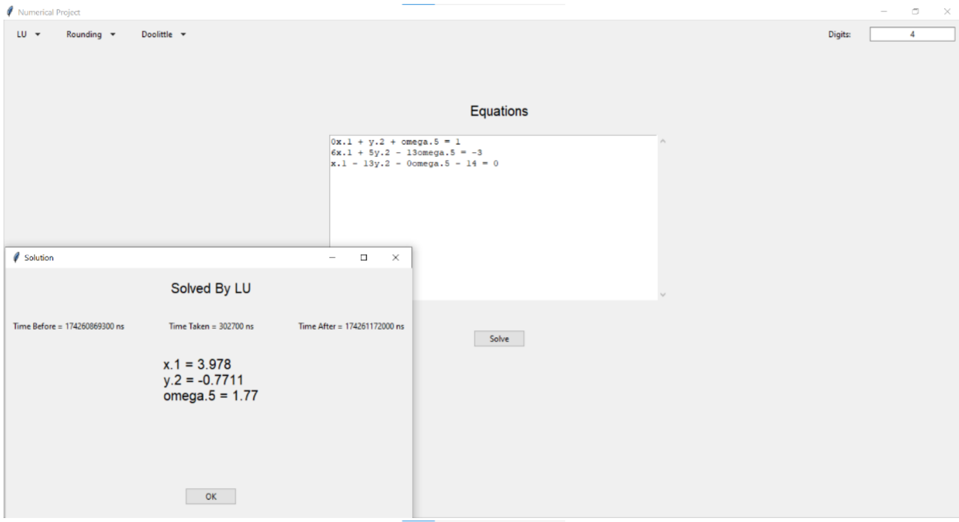

Numerical Project
LU
Rounding
Crout
Digits:
7
Equations
1x_1 + 0.5y_2 + 0.333z_3 + 0.25w_4 = 17.6
0.5x_1 - 0.333y_2 + 0.25z_3 - 0.2w_4 = 51.912
0.333x_1 + 0.25y_2 + 0.2z_3 - 0.166w_4 = 19.3141529
-0.25x_1 -0.2y_2 +0.166z_3 - 0.1428w_4 = 0.0125369
Solve
Solution
Solved By LU
Time Before = 805184860600 ns
Time Taken = 394900 ns
Time After = 805185255500 ns
x_1 = 59.12816
y_2 = -36.18425
z_3 = -16.3638
w_4 = -71.94756
OK

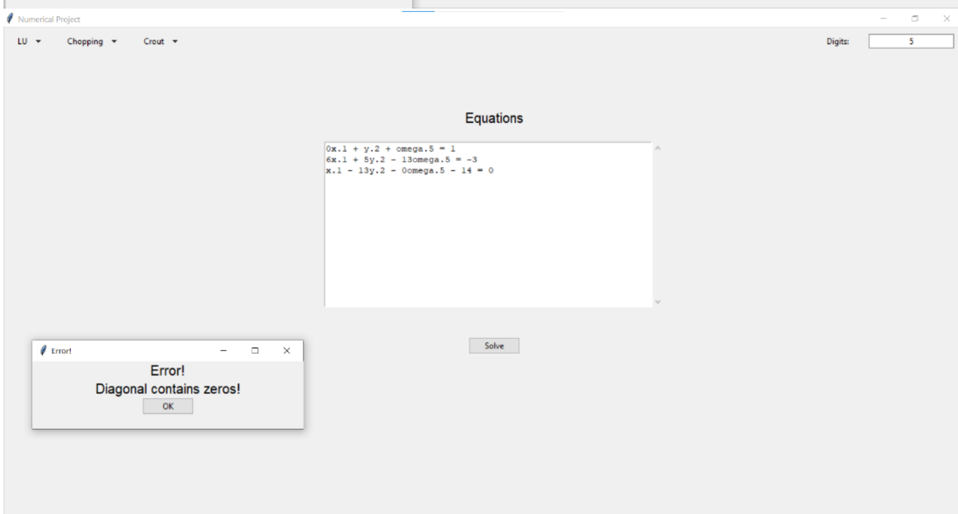

Numerical Project
Gauss-Seidel
Rounding
Digits:
9
Iterations:
125
Epsilon:
10**-7
Equations
1x_1 + 0.5y_2 + 0.333z_3 + 0.25w_4 = 17.6
0.5x_1 - 0.333y_2 + 0.25z_3 - 0.2w_4 = 51.912
0.333x_1 + 0.25y_2 + 0.2z_3 - 0.166w_4 = 19.3141529
-0.25x_1 -0.2y_2 +0.166z_3 - 0.1428w_4 = 0.0125369
Initial guess
1
2
3
0
Solve
Error!
Error!
The method diverges!
OK

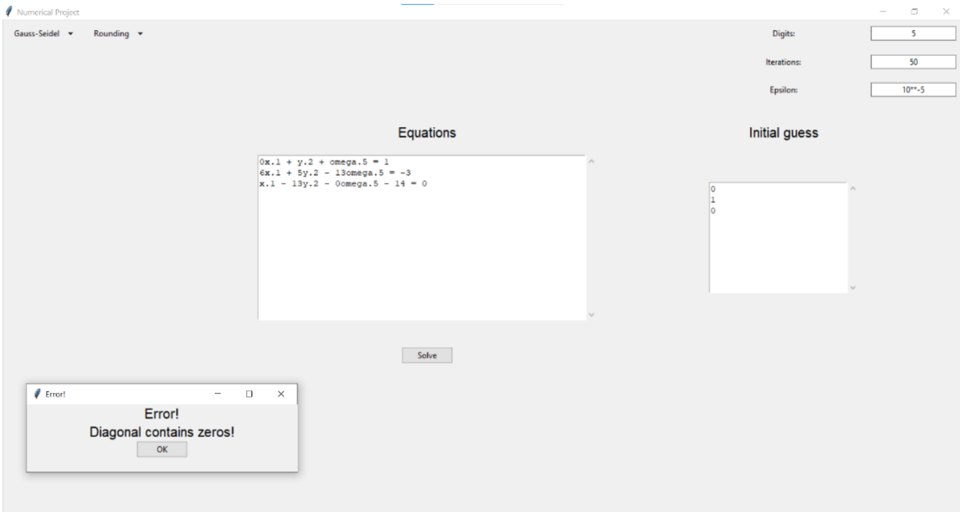

Numerical Project
Jacobi
Chopping
Digits:
7
Iterations:
50
Epsilon:
10**-7
Equations
1x_1 + 0.5y_2 + 0.333z_3 + 0.25w_4 = 17.6
0.5x_1 - 0.333y_2 + 0.25z_3 - 0.2w_4 = 51.912
0.333x_1 + 0.25y_2 + 0.2z_3 - 0.166w_4 = 19.3141529
-0.25x_1 -0.2y_2 +0.166z_3 - 0.1428w_4 = 0.0125369
Initial guess
10
-2
3
3
Solve
Error!
Error!
The method diverges!
OK

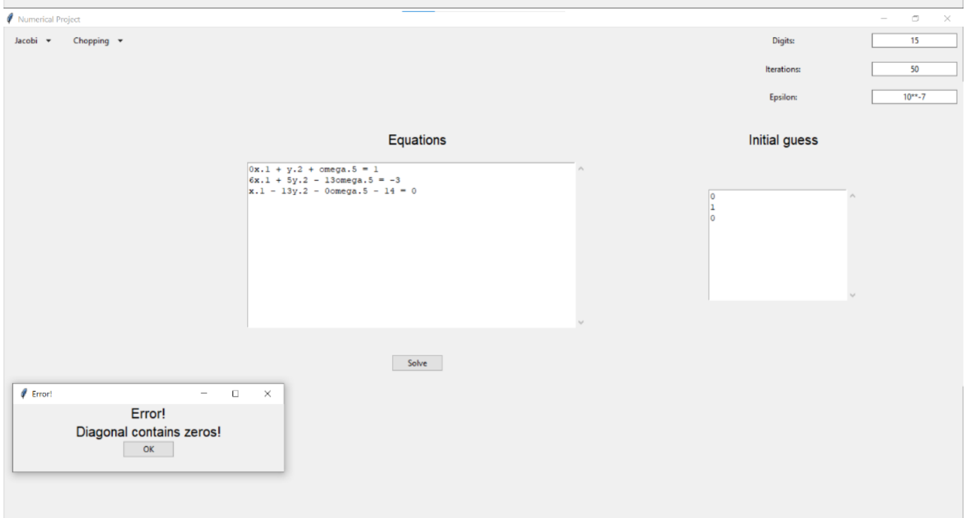

Numerical Project
Gauss
Rounding
Digits:
7
Equations
0x.1 + y.2 + omega.5 = 1
6x.1 + 5y.2 - 13omega.5 = -3
x.1 - 13y.2 - 0omega.5 - 14 = 0
Solve
Solution
Solved By Gauss
Time Before = 1094381433000 ns
Time Taken = 350200 ns
Time After = 1094381783200 ns
x.1 = 3.979165
y.2 = -0.770833
omega.5 = 1.770833
OK


Numerical Project
Gauss-Jordan
Chopping
Digits:
17
Equations
0x.1 + y.2 + omega.5 = 1
6x.1 + 5y.2 - 13omega.5 = -3
x.1 - 13y.2 - 0omega.5 - 14 = 0
Solve
Solution
Solved By Gauss-Jordan
Time Before = 1167460803900
Time Taken = 386100 ns
Time After = 1167461190000
x.1 = 3.9791666666666665
y.2 = -0.7708333333333333
omega.5 = 1.77083333333333
OK


Numerical Project
LU
Rounding
Doolittle
Digits:
4
Equations
0x.1 + y.2 + omega.5 = 1
6x.1 + 5y.2 - 13omega.5 = -3
x.1 - 13y.2 - 0omega.5 - 14 = 0
Solve
Solution
Solved By LU
Time Before = 174260869300 ns
Time Taken = 302700 ns
Time After = 174261172000 ns
x.1 = 3.978
y.2 = -0.7711
omega.5 = 1.77
OK


Numerical Project
LU
Chopping
Crout
Digits:
5
Equations
0x.1 + y.2 + omega.5 = 1
6x.1 + 5y.2 - 13omega.5 = -3
x.1 - 13y.2 - 0omega.5 - 14 = 0
Solve
Error!
Error!
Diagonal contains zeros!
OK


Numerical Project
Gauss-Seidel
Rounding
Digits:
5
Iterations:
50
Epsilon:
10**-5
Equations
0x.1 + y.2 + omega.5 = 1
6x.1 + 5y.2 - 13omega.5 = -3
x.1 - 13y.2 - 0omega.5 - 14 = 0
Initial guess
0
1
0
Solve
Error!
Error!
Diagonal contains zeros!
OK


Numerical Project
Jacobi
Chopping
Digits:
15
Iterations:
50
Epsilon:
10**-7
Equations
0x.1 + y.2 + omega.5 = 1
6x.1 + 5y.2 - 13omega.5 = -3
x.1 - 13y.2 - 0omega.5 - 14 = 0
Initial guess
0
1
0
Solve
Error!
Error!
Diagonal contains zeros!
OK

- Comparisons

Comparison	Gauss Elimination	Gauss Jordan
Time Complexity	Pivoting and scaling O(n^2) Check Empty rows O(n) Update position O(n) Elimination Process O(n^3) Substitution O(n^2) Total O(n^3)	Pivoting and scaling O(n^2) Check Empty rows O(n) Update position O(n) Elimination Process O(n^3) Normalization O(n) Total O(n^3)
Convergence	Always Converge	Always Converge

Comparison	LU Decomposition Doolittle	LU Decomposition Croute
Time Complexity	Find Scalars O(n^2) Forward Elimination O(n^3) Forward Elimination O(n^3) Forward Substitution O(n^2) Backward Substitution O(n^2) Total O(n^3)	Croute Formation O(n^3) Forward Substitution O(n^2) Backward Substitution O(n^2) Total O(n^3)
Convergence	Always Converge	Always Converge

Comparison	Gauss Seidel	Jacobi
Time Complexity	Each iteration O(n^2)	Each iteration O(n^2)
Convergence	A sufficient condition for convergence is Diagonally dominant but not necessary	A sufficient condition for convergence is Diagonally dominant but not necessary
Best Error	0	0
Approximate Error	Less Approximate error as it mostly converge faster	Larger Approximate Error

➢ Data structure used

- List of Lists: to store the coefficient matrix. It helps us to get any element directly instead of getting it from two lists.

Help us to remove elements at any position unlike stacks for example

Provide random access to any element

- List: to store the vector of constants and the vector of Unknowns. It helps us to directly get any element in the list instead of getting it sequentially.