Phase 1 Report

Team Members:

Badr Elsayed – 22010664

Abdel Rahman Nasr – 22010887

Lojine Sameh – 22011054

Nour Khaled – 22011319

Adham Anas – 22010601

1. *Pseudo-Code:*

 # For solving system of linear equations with alphabetical coefficients
Function alphabeticalSolution(matrix):

```
If number of rows of matrix != number of unique rows:
return "No Solution For Matrix"

    # Define symbolic variables for each of the unknowns based on the number of
rows
    variables = generate symbolic variables ('x', 'y', ...,)

    # Convert string coefficients to symbolic variables
    symbolic_matrix = Empty list
    For each row i in matrix:
        symbolic_row = Empty list
        For each element j in row i:
            If element is a string: Convert to symbolic variable and add to
        symbolic_row
            Else: Add the numeric value to symbolic_row

        Add symbolic_row to symbolic_matrix

    # Define the system of equations
    equations = Empty list



# Iterate through the matrix to create the equations
```

```
For each row i in symbolic_matrix:
    equation = 0
    For each element j in row i (except the last one):
        equation = equation + (element at i,j * corresponding variable in
    variables)
    Add (equation, last element in row i) to equations

# Solve the system of equations

solutions = Solve equations for variables using "sympy.solve()"

return solutions
```

Helper functions for different methods

Function forward_Elimination(augmented_matrix,n):

```
for i=0 to n-1:
    # Find the row with the maximum absolute value in column i
    max_row = row with maximum value in column i
Swap rows i and max_row

    # Check if the pivot element is zero or very close to zero
    if pivot element is too small (close to zero):
        return "Pivot element is zero or very small, cannot proceed."

    # Perform the row reduction
    for j =i + 1to n-1:
        factor = augmented_matrix[j, i] / augmented_matrix[i, i]
        augmented_matrix[j, i:] -= factor * augmented_matrix[i, i:]

return augmented_matrix
```

Function forward_substitution(augmented_matrix, n):

```
x = array of zeroes with length n
for i=0 to n-1:
        if augmented_matrix[i, i] == 0:
            return "System has no unique solution."
        x[i] = augmented_matrix[i, -1] / augmented_matrix[i, i]
        for j=i + 1 to n-1:
            augmented_matrix[j, -1] -= augmented_matrix[j, i] * x[i]

return x, augmented_matrix
```

Function backward_Elimination(augmented_matrix,n):
    for i = n-1 to 0:
        Normalize the pivot row (make the pivot element equal to 1
        # Eliminate the elements above the pivot
        for j=i-1 to 0: # Go through rows above the pivot
            factor = augmented_matrix[j, i]
            each element in row j from i to n-1 -= factor * each element in row i from
i to n-1
    return augmented_matrix

Function backward_substitution(augmented_matrix,n):
    x = array of zeroes with length n
    for i = n-1 to 0:
        if augmented_matrix[i, i] == 0:
            return "System has no unique solution."
        x[i] = augmented_matrix[i, n] / augmented_matrix[i, i]
        Update augmented_matrix for rows above the pivot

    return x , augmented_matrix

 # The methods

Function Gauss_Elimination(A, B):
  n = number of rows of B
  augmented_matrix = append B as last column to A
  augmented_matrix = forward_Elimination(augmented_matrix, n)
  x, augmented_matrix = backward_substitution(augmented_matrix, n)
  return x, augmented_matrix

Function Gauss_Jordan_Elimination(A, B):
  augmented_matrix = append B as last column to A
  augmented_matrix = forward_Elimination(augmented_matrix, n)
  augmented_matrix = backward_Elimination(augmented_matrix, n)
  x = last column of augmented_matrix
  return x, augmented_matrix

Function Jacobi(self, A: np.ndarray, b: np.ndarray, epsilon=1e-9, iterations=50, x=None, mode=2):
  # 'mode' is what determines wether to use iterations or epsilon for solving
  # 'mode' = 1 is iterations, 'mode' = 2 is epsilon (absolute relative error)
  # if 'mode' is negative it means Gauess Seidel is applied and not Jacobi

```
max_its = 2000
GaussSeidel = False
if (mode < 0):
  GaussSeidel = True
  mode = abs(mode)

if (mode != 1 and mode != 2):
  print("Unknown Mode choosen in Jacobi.")

if (x is None):
  x = zeroes with length rows of A

D = diagonal of A
if an element in D is 0
  print("Matrix contains zero diagonal elements, Jacobi method cannot
proceed.")

curr_it = 0
while (True):
  x_new = x.copy()
  for each row i A:
    x_new[i] = b[i]
    for each column j in A:
      if (i == j):
        continue
      if (GaussSeidel):
        x_new[i] -= A[i][j] * x_new[j]
      else:
        x_new[i] -= A[i][j] * x[j]
    x_new[i] /= A[i][i]

  if (mode == 1):
    if (curr_it > iterations):
      return x_new, curr_it-1

    if (curr_it > max_its or there's an overflow):
      if (GaussSeidel): print("Divergence occured in Gauss Seidel")
      else: print("Divergence occured in Jacobi")
      return x_new, curr_it-1

  else if (mode == 2):
    condition = True
```

```
        for each row i in A:
            if (not (absolute(x[i] - x_new[i]) < epsilon)):
                condition = False
        if (condition):
            return x_new, curr_it-1

        if (curr_it > max_its or there's an overflow):
            if (GaussSeidel): print("Divergence occured in Gauss Seidel")
            else: print("Divergence occured in Jacobi")
            return x_new, curr_it-1

    x = x_new
    curr_it += 1
```

Function GaussSeidelA, b, epsilon=1e-9, iterations=50, x=None, mode=2):

```
    return Jacobi(A, b, epsilon, iterations, x, mode = -mode)
```

Function LUCroutsForm(A, B):

```
    L = zeroes matrix with same size of A
    U = zeroes matrix with same size of A
    L_and_U = zeroes matrix with same size of A
    n = number of rows of B

        for i=0 to n-1:
            for j=0 to n-1:
                L[i][j] = A[i][j]
                A[i] -= L[i][j] * U[j]

            L[i][i] = A[i][i] #i==j (diagonal)

            if L[i][i] != 0:
                U[i] = A[i] / L[i][i]
            else:  # infinite number of solution or no solution
                return "System has no unique solution or no solution."

        #solve the equation
        augmented_L = add column B to L
        Y, augmented_L = forward_substitution(augmented_L,n)
        augmented_U = add column Y to U
        X, augmented_U = backward_substitution(augmented_U,n)
        for i=0 to n-1:
```

```
            for j=0 to n-1:
                if i <= j:
                  L_and_U[i][j] = U[i][j]
                else:
                  L_and_U[i][j] = L[i][j]

        return X, L_and_U
```

Function LUCholeskyForm(A, B):

```
      L = zeroes matrix with same size of A
      If matrix A isn't positive definite:
Print("Error")
          return

        for i=0 to n-1:
          Update L matrix using Cholesky method
        U= Transpose of L   #U = L transpose
        n=number of rows of B
        #solve the equation
        augmented_L = add column b to L
        Y, augmented_L = forward_substitution(augmented_L,n)
        augmented_U = add column Y to U
        X, augmented_U = backward_substitution(augmented_U,n)

        return X,L
```

Function LUDoolittlesForm(A, B):

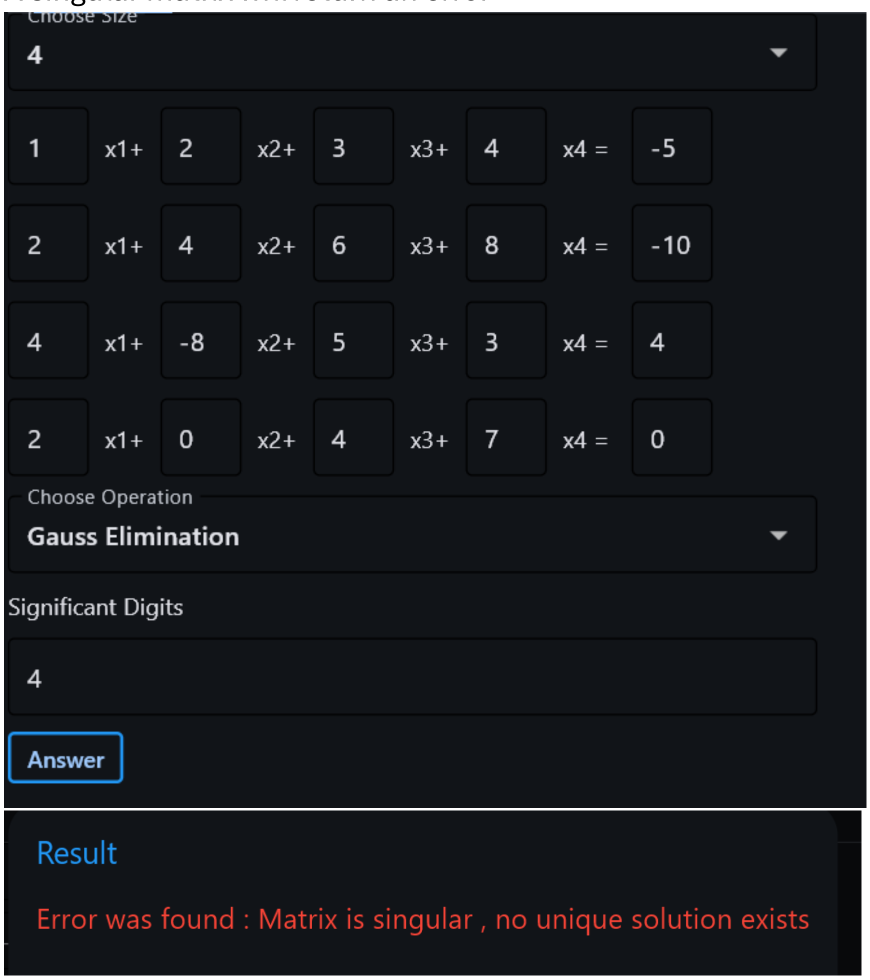
```
      L = zeroes matrix with same size of A
      U = zeroes matrix with same size of A
      L_and_U = zeroes matrix with same size of A
      n = number of rows of B

        for i=0 to n-1:
          L[i][i] = 1
          # Compute Upper Triangular Matrix
          for j=i to n-1:
            sum = 0
            for k=0 to i-1:
sum += L[i][k] * U[k][j]
            U[i][j] = A[i][j] - sum

          if U[i, i] == 0:
```

```
        return "Matrix is singular, no unique solution."

    # Compute Lower Triangular Matrix
    for j=i+1 to n-1:
        sum = 0
        for k=0 to i-1:
   sum += L[j][k] * U[k][i]
        L[j][i] = (A[j][i] - sum) / U[i][i]

 # Storing L and U in one matrix
for i=0 to n-1:
    for j=0 to n-1:
        if i <= j:
            L_and_U[i][j] = U[i][j]
        else:
            L_and_U[i][j] = L[i][j]

 #solve the equation
 augmented_LB = add colomn B to L
 Y , augmented_LB = forward_substitution(augmented_LB, n)
 if isinstance(Y, str):
    return Y
 else:
    augmented_UY = add coloumn Y to U
    X ,augmented_UY = backward_substitution(augmented_UY, n)
 return X , L_and_U
```

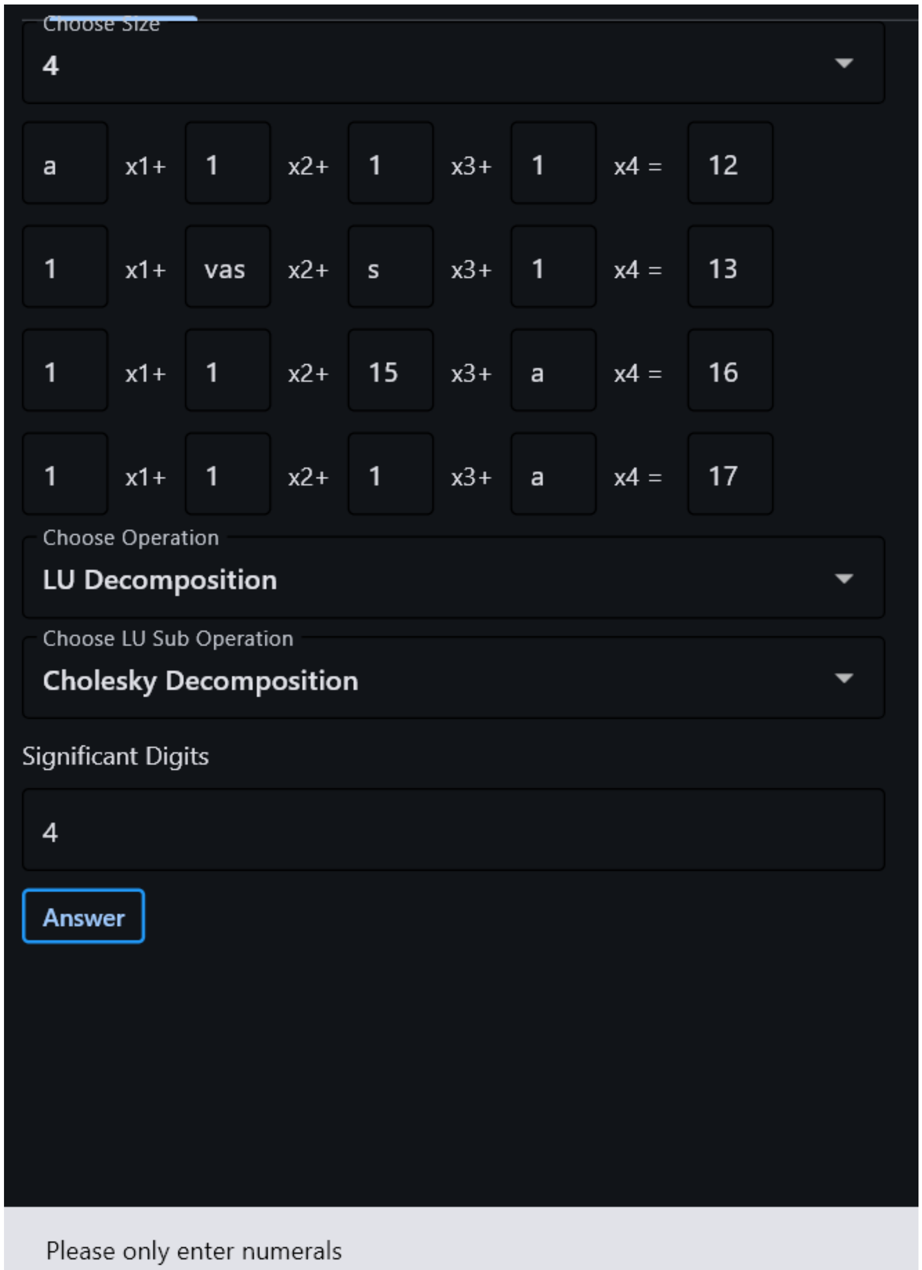
2. *Sample runs for each method:*

- General Cases Independent of Method:

1- A singular matrix will return an error



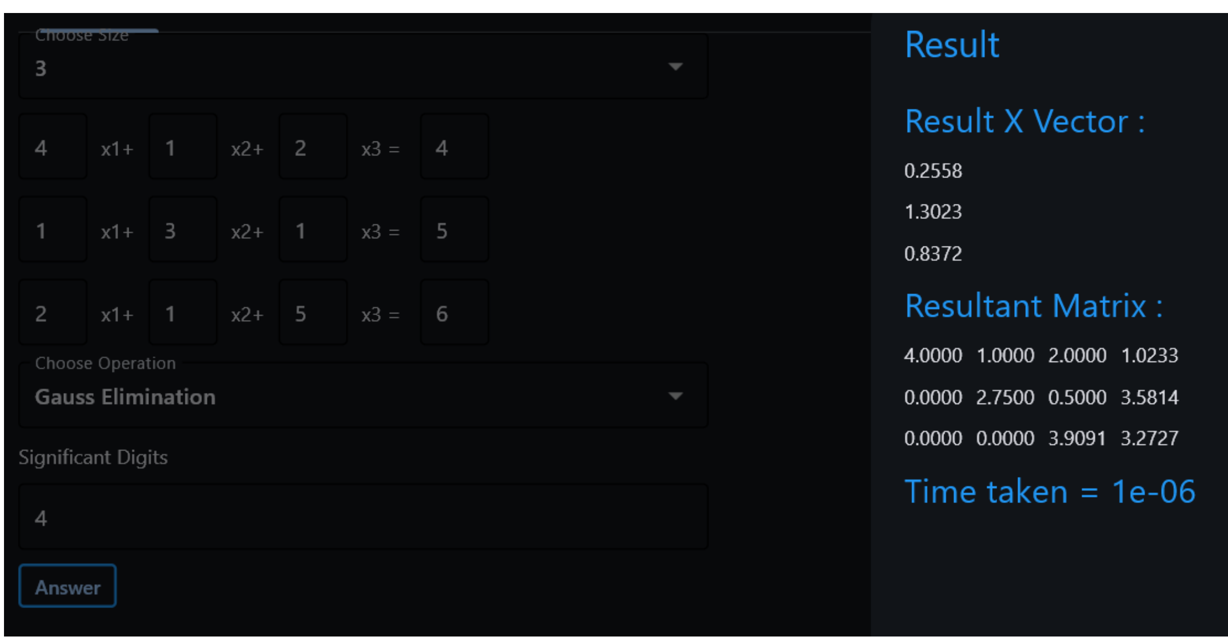
2- Entered a non-number or left some fields



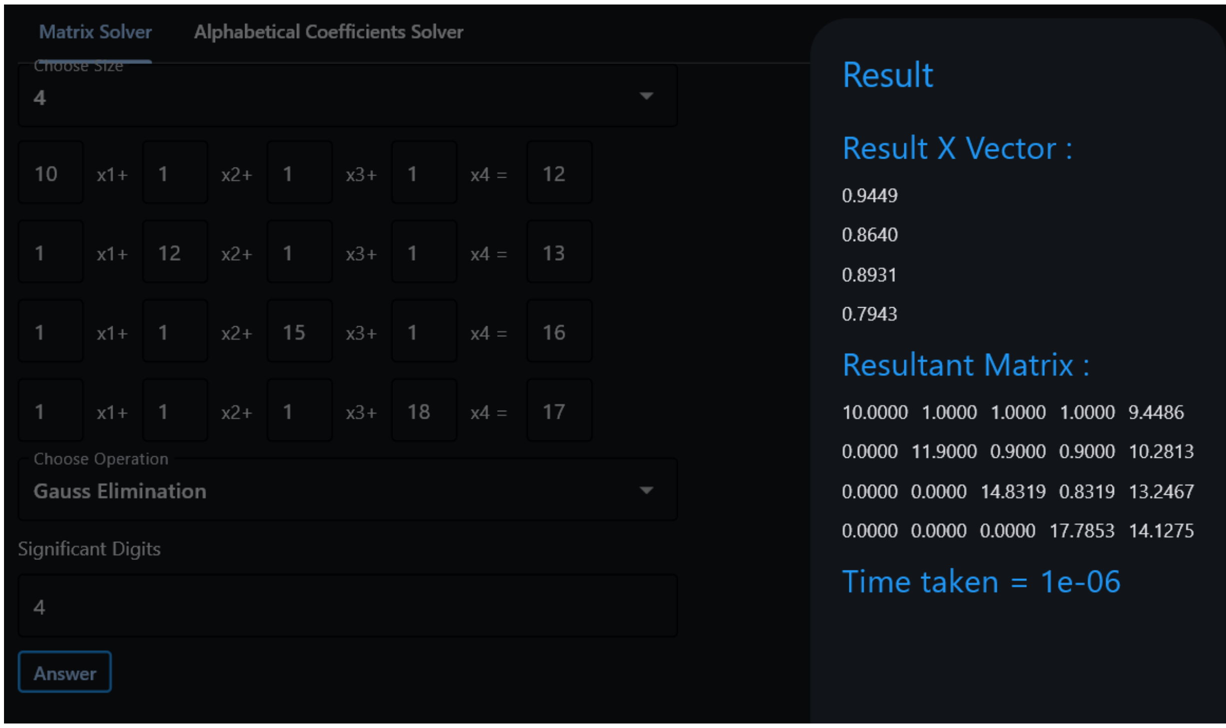
❖ Gauss Elimination

1-

2-



3-



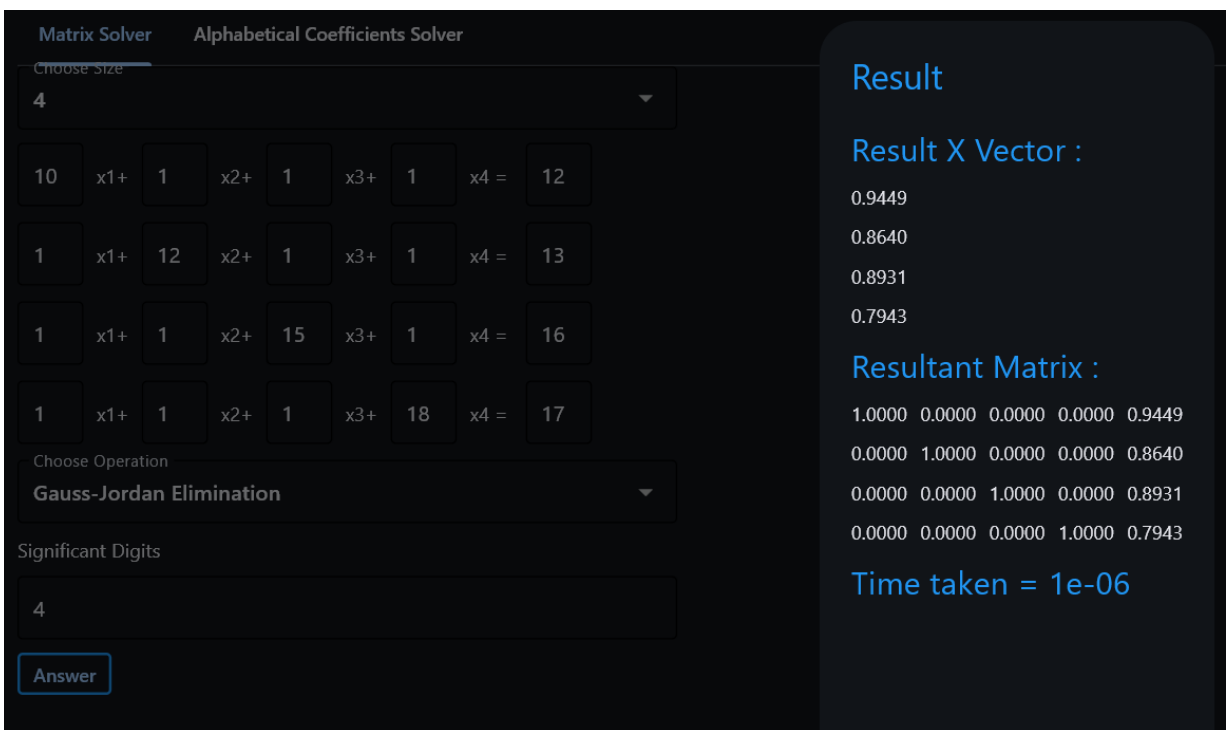
- Gauss-Jordan

1-

2-

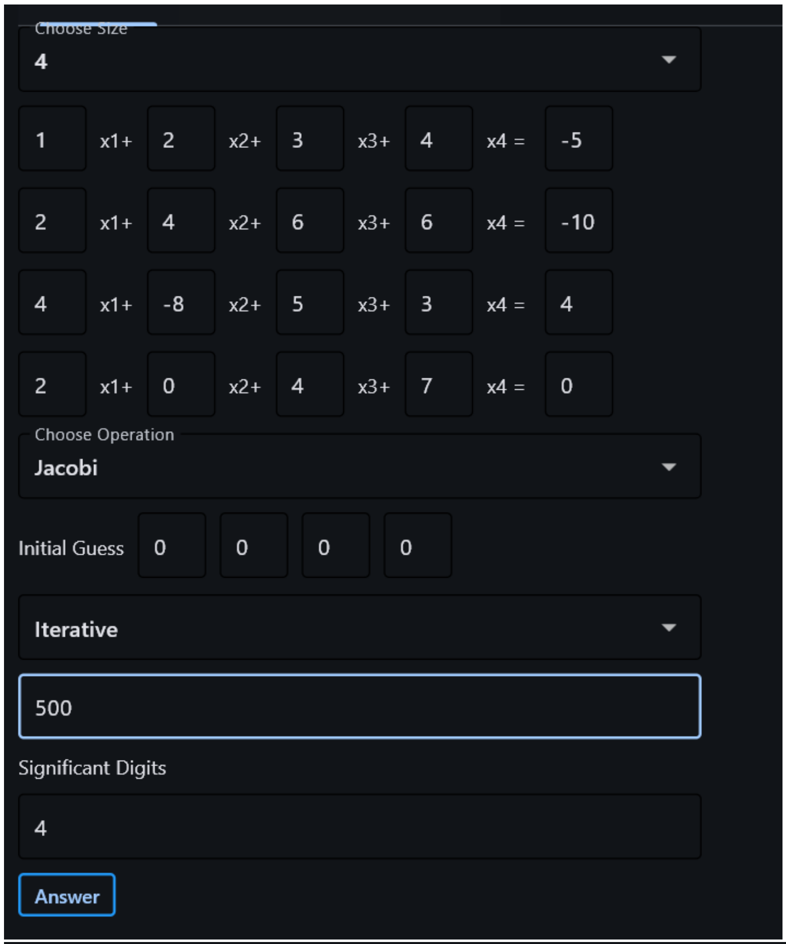

Choose Size
3
4 x1+ 1 x2+ 2 x3 = 4
1 x1+ 3 x2+ 1 x3 = 5
2 x1+ 1 x2+ 5 x3 = 6
Choose Operation
Gauss-Jordan Elimination
Significant Digits
4
Answer
Result
Result X Vector :
0.2558
1.3023
0.8372
Resultant Matrix :
1.0000 0.0000 0.0000 0.2558
0.0000 1.0000 0.0000 1.3023
0.0000 0.0000 1.0000 0.8372
Time taken = 1e-06

3-

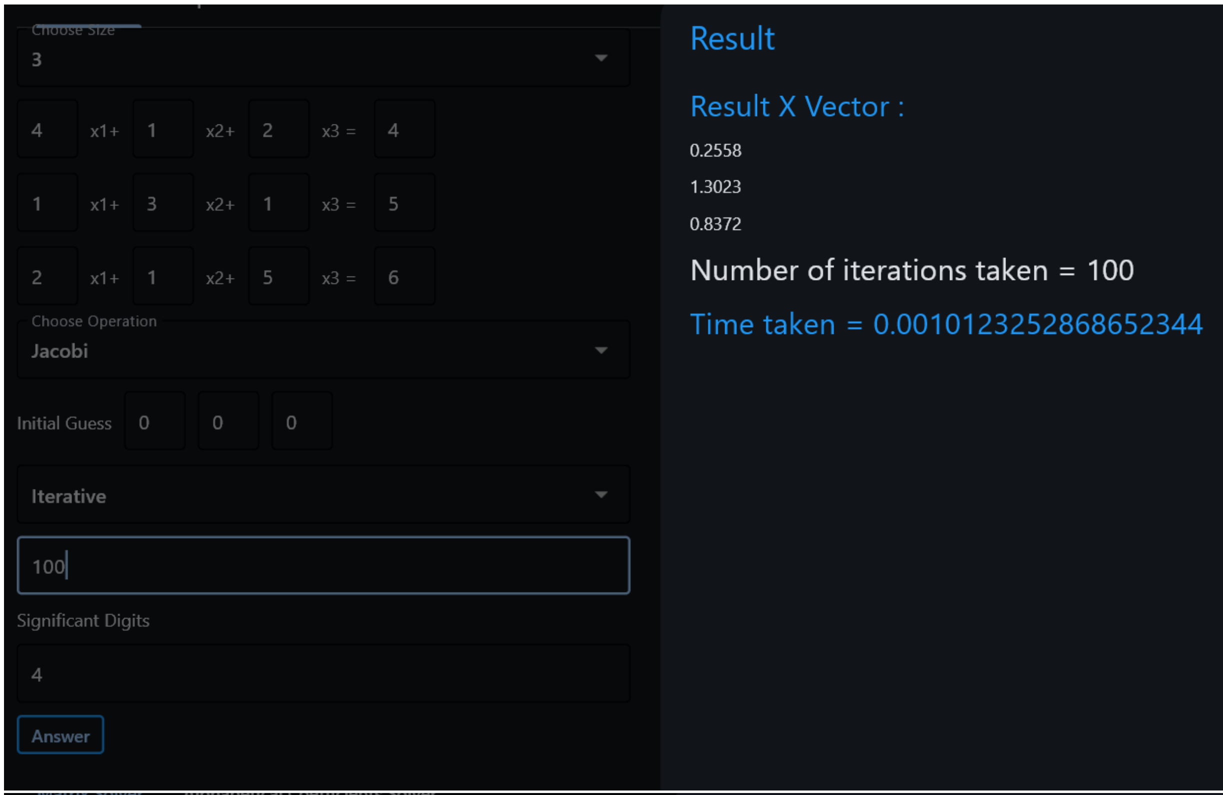

Matrix Solver
Alphabetical Coefficients Solver
Choose Size
4
10 x1+ 1 x2+ 1 x3+ 1 x4 = 12
1 x1+ 12 x2+ 1 x3+ 1 x4 = 13
1 x1+ 1 x2+ 15 x3+ 1 x4 = 16
1 x1+ 1 x2+ 1 x3+ 18 x4 = 17
Choose Operation
Gauss-Jordan Elimination
Significant Digits
4
Answer
Result
Result X Vector :
0.9449
0.8640
0.8931
0.7943
Resultant Matrix :
1.0000 0.0000 0.0000 0.0000 0.9449
0.0000 1.0000 0.0000 0.0000 0.8640
0.0000 0.0000 1.0000 0.0000 0.8931
0.0000 0.0000 0.0000 1.0000 0.7943
Time taken = 1e-06

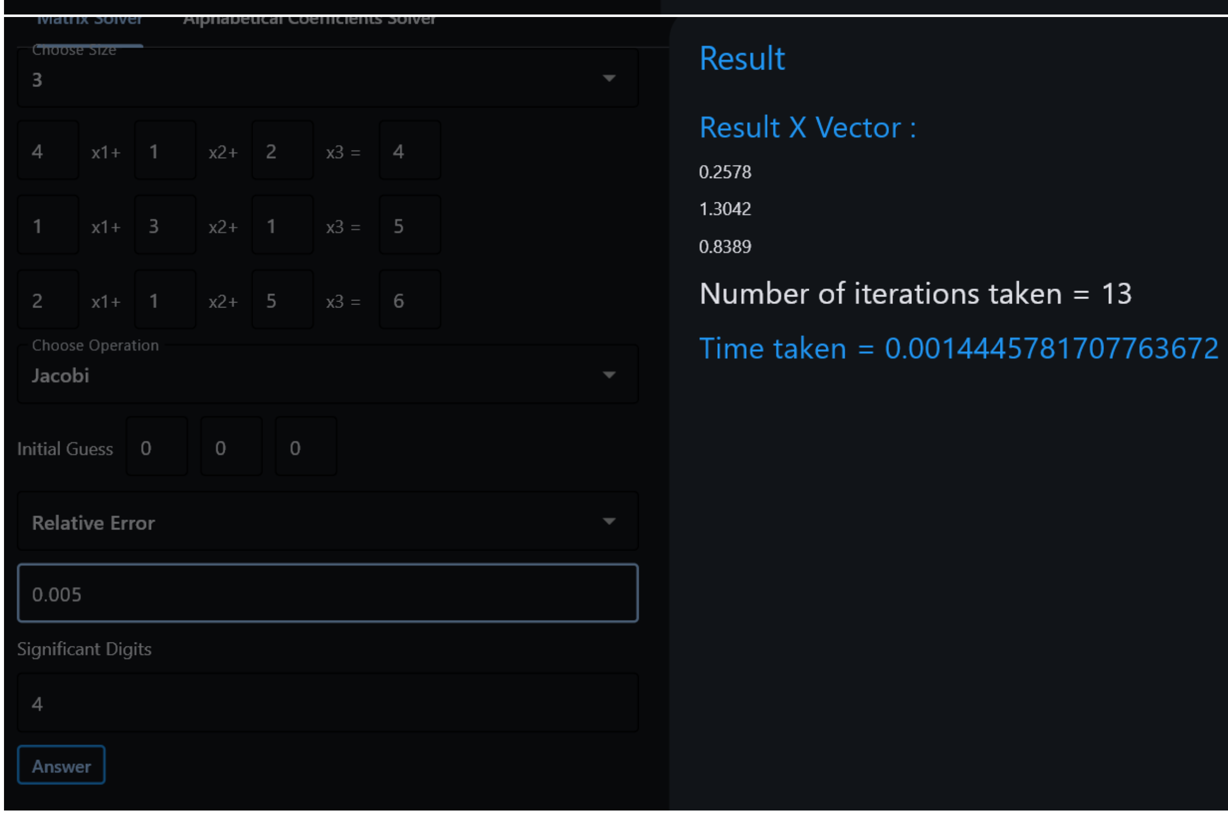
- ❖ Jacobi

1- Diverge

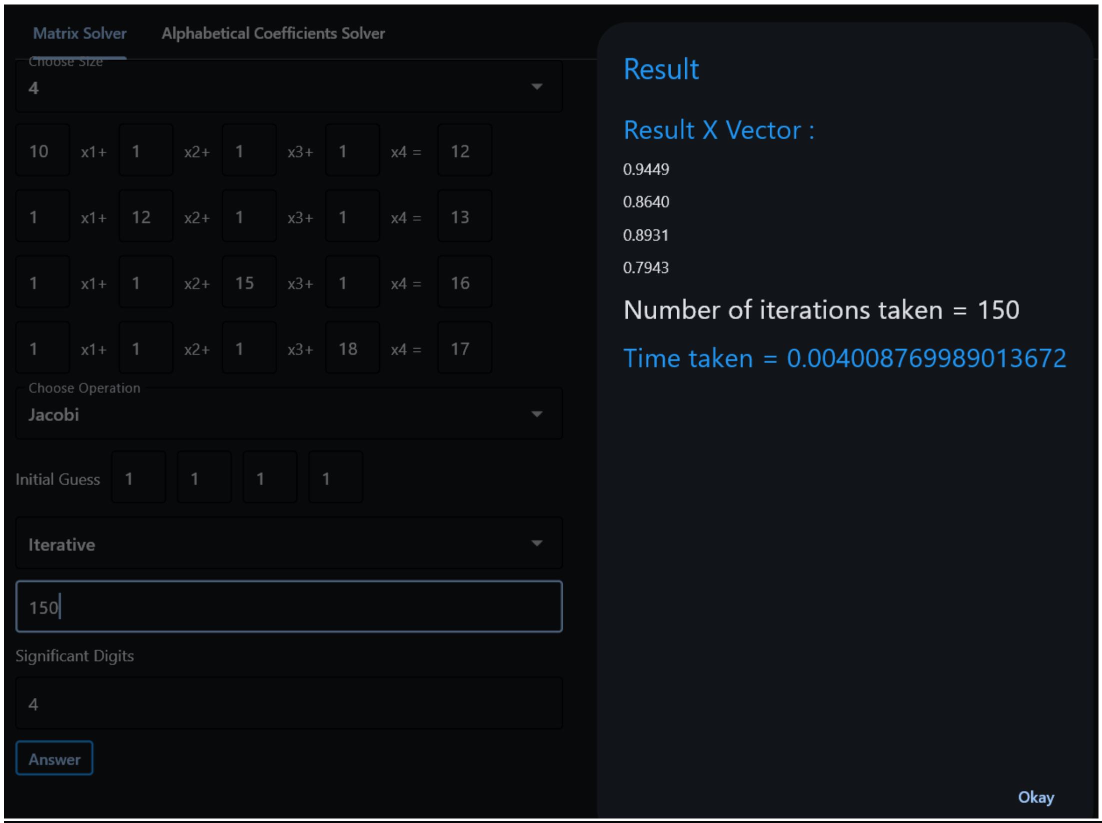


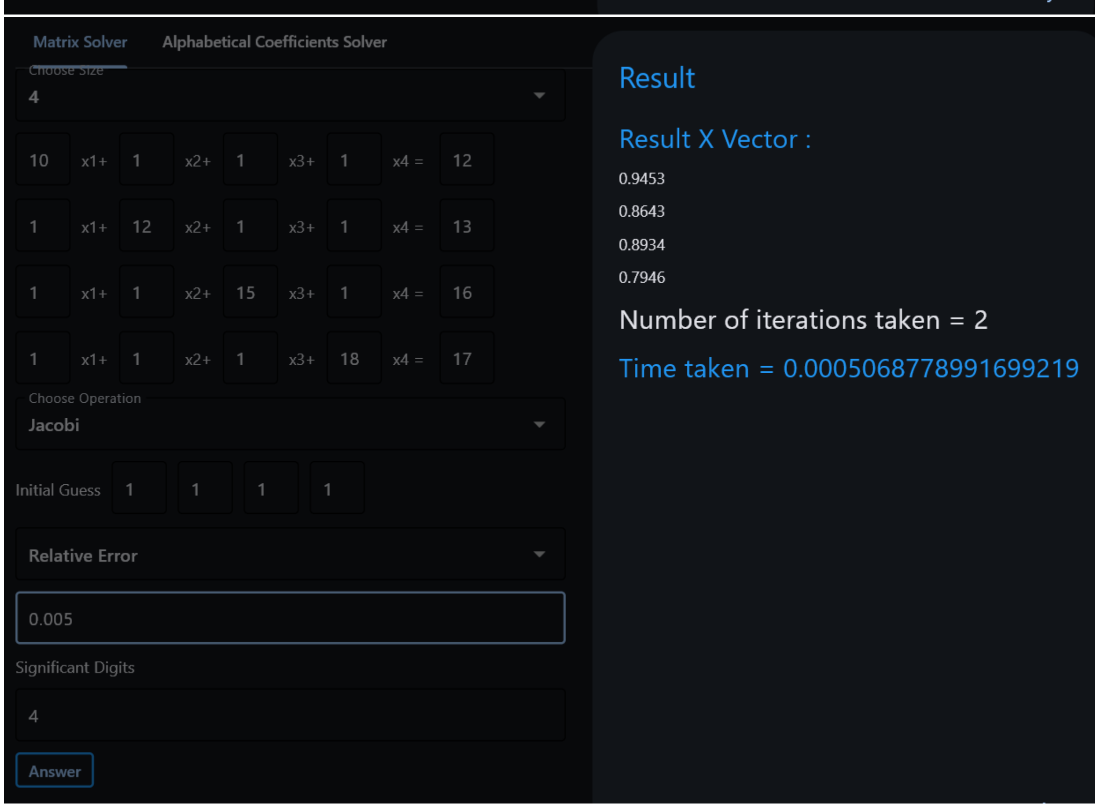
2- Converge


Choose Size
3
4 x1+ 1 x2+ 2 x3 = 4
1 x1+ 3 x2+ 1 x3 = 5
2 x1+ 1 x2+ 5 x3 = 6
Choose Operation
Jacobi
Initial Guess 0 0 0
Iterative
100
Significant Digits
4
Answer
Result
Result X Vector :
0.2558
1.3023
0.8372
Number of iterations taken = 100
Time taken = 0.0010123252868652344


Choose Size
3
4 x1+ 1 x2+ 2 x3 = 4
1 x1+ 3 x2+ 1 x3 = 5
2 x1+ 1 x2+ 5 x3 = 6
Choose Operation
Jacobi
Initial Guess 0 0 0
Relative Error
0.005
Significant Digits
4
Answer
Result
Result X Vector :
0.2578
1.3042
0.8389
Number of iterations taken = 13
Time taken = 0.0014445781707763672

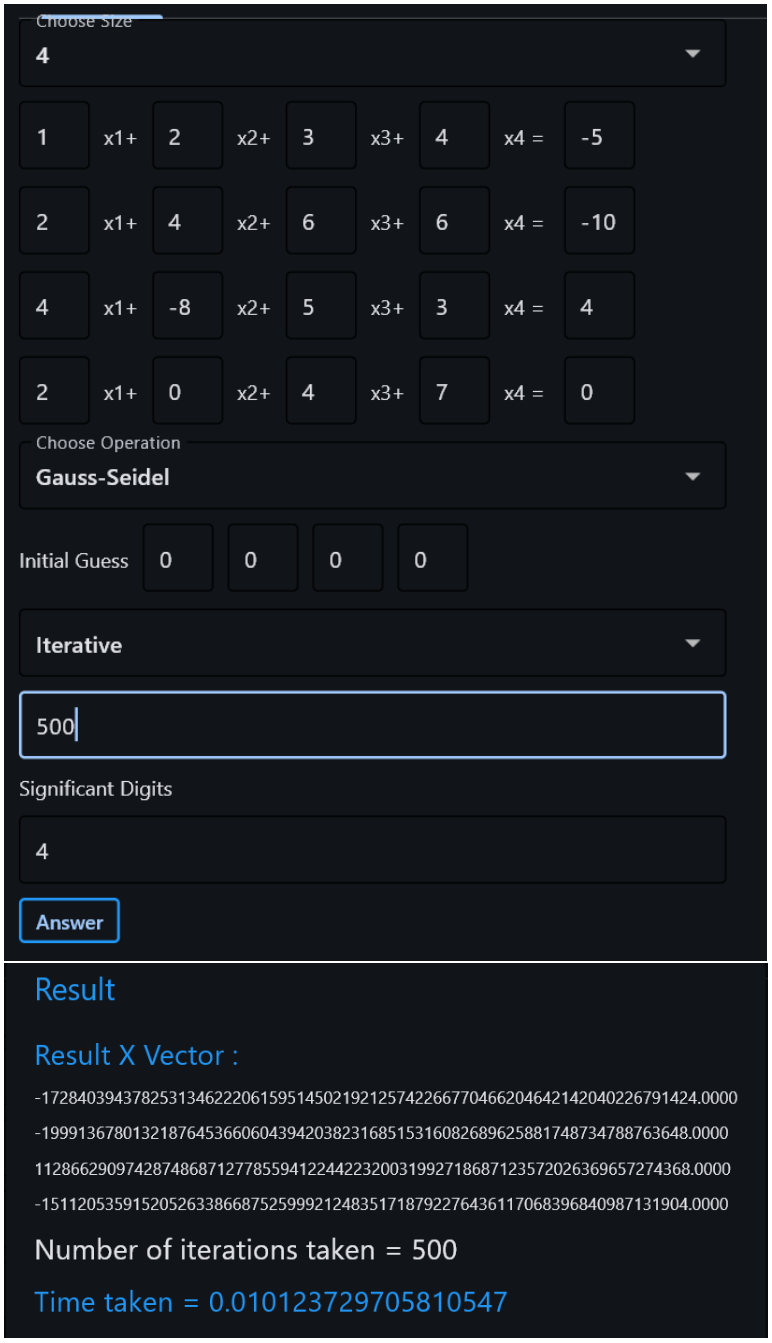
3- Converge


Matrix Solver
Alphabetical Coefficients Solver
Choose Size
4
10 x1+ 1 x2+ 1 x3+ 1 x4 = 12
1 x1+ 12 x2+ 1 x3+ 1 x4 = 13
1 x1+ 1 x2+ 15 x3+ 1 x4 = 16
1 x1+ 1 x2+ 1 x3+ 18 x4 = 17
Choose Operation
Jacobi
Initial Guess 1 1 1 1
Iterative
150
Significant Digits
4
Answer
Result
Result X Vector :
0.9449
0.8640
0.8931
0.7943
Number of iterations taken = 150
Time taken = 0.004008769989013672
Okay

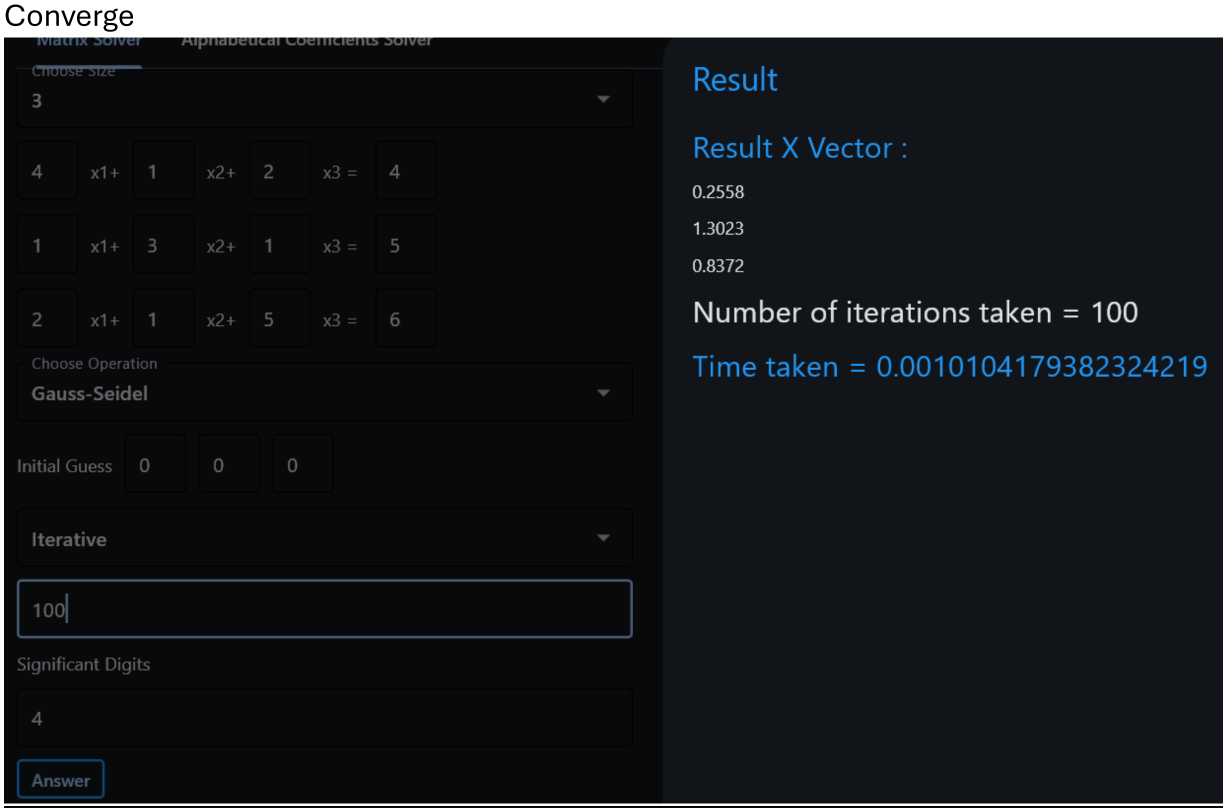

Matrix Solver
Alphabetical Coefficients Solver
Choose Size
4
10 x1+ 1 x2+ 1 x3+ 1 x4 = 12
1 x1+ 12 x2+ 1 x3+ 1 x4 = 13
1 x1+ 1 x2+ 15 x3+ 1 x4 = 16
1 x1+ 1 x2+ 1 x3+ 18 x4 = 17
Choose Operation
Jacobi
Initial Guess 1 1 1 1
Relative Error
0.005
Significant Digits
4
Answer
Result
Result X Vector :
0.9453
0.8643
0.8934
0.7946
Number of iterations taken = 2
Time taken = 0.0005068778991699219

- Gauss-Seidel

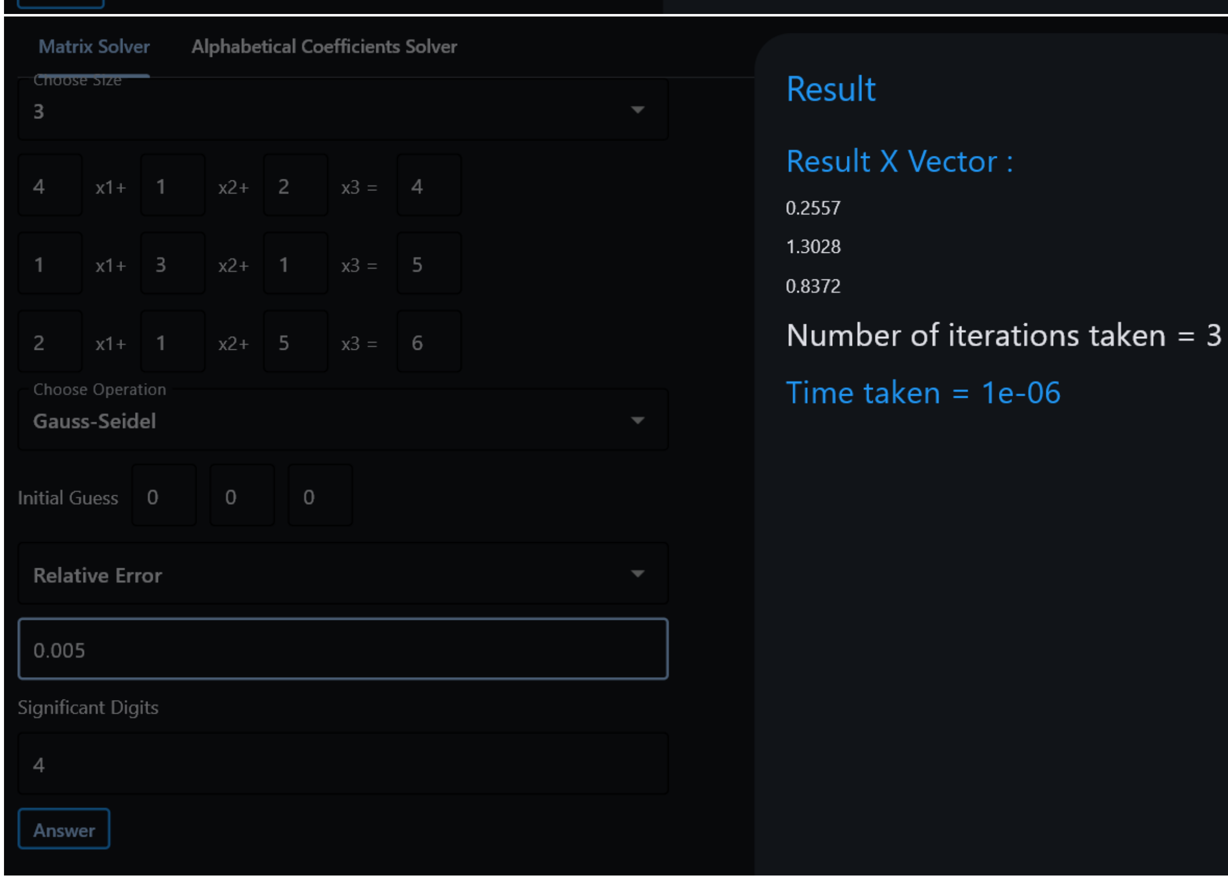
1- Diverge



2- Converge

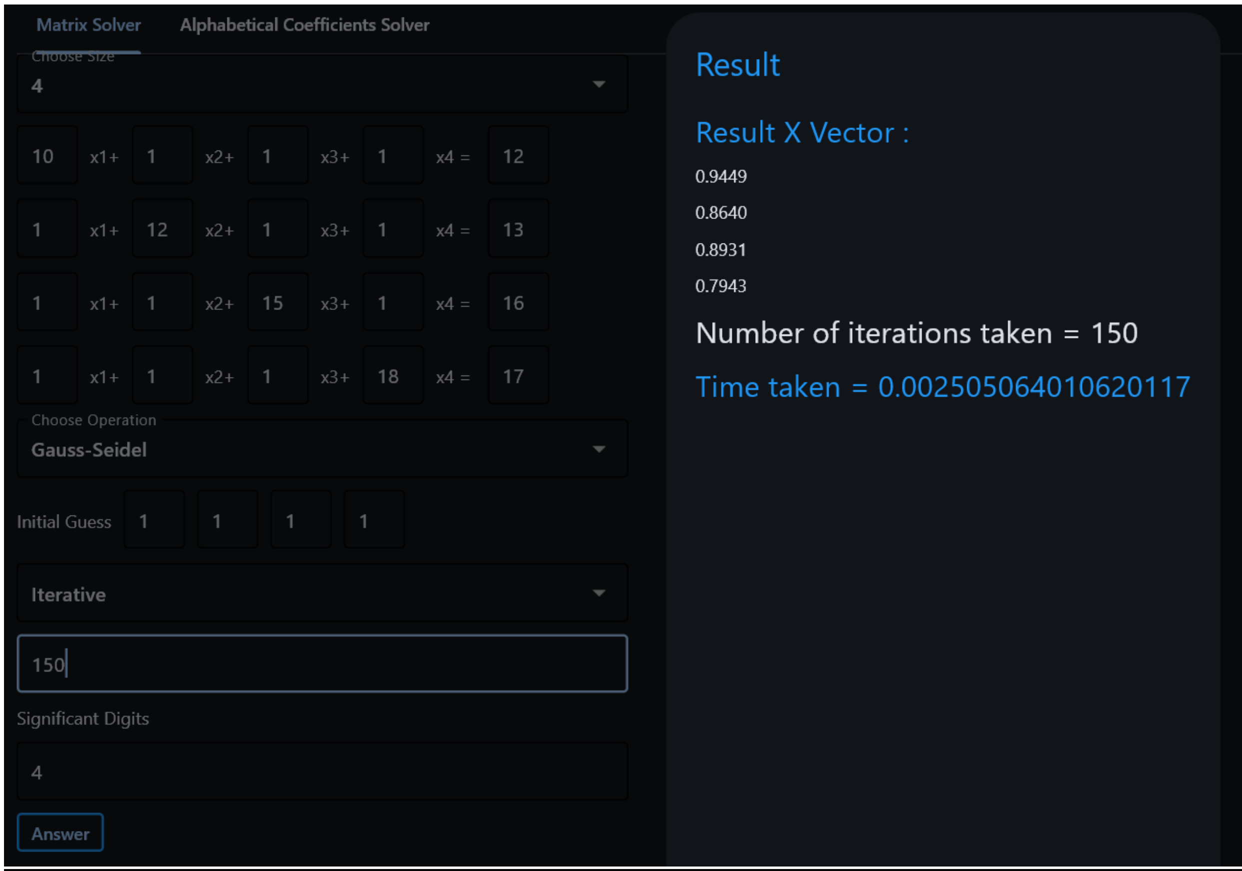


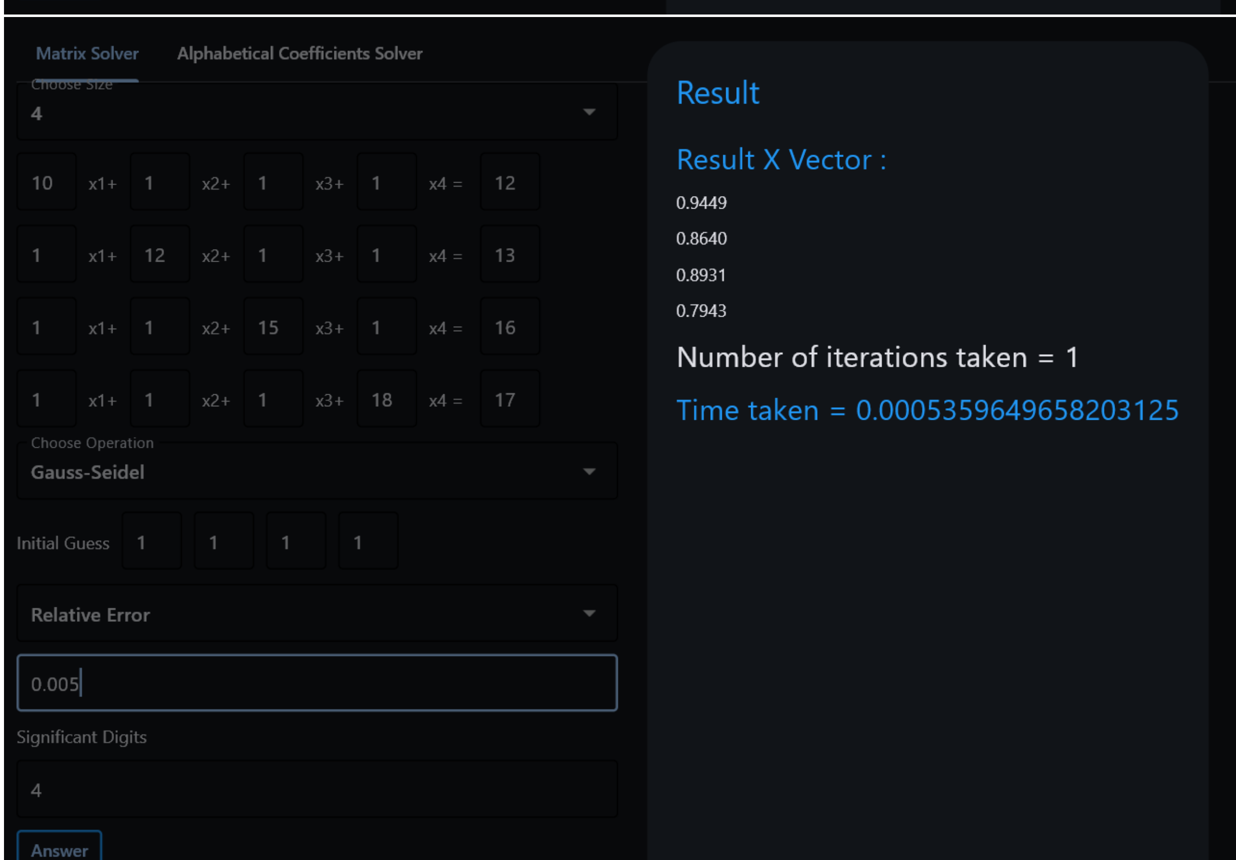
Matrix Solver
Alphabetical Coefficients Solver
Choose Size
3
4 x1+ 1 x2+ 2 x3 = 4
1 x1+ 3 x2+ 1 x3 = 5
2 x1+ 1 x2+ 5 x3 = 6
Choose Operation
Gauss-Seidel
Initial Guess 0 0 0
Iterative
100
Significant Digits
4
Answer
Result
Result X Vector :
0.2558
1.3023
0.8372
Number of iterations taken = 100
Time taken = 0.0010104179382324219



Matrix Solver
Alphabetical Coefficients Solver
Choose Size
3
4 x1+ 1 x2+ 2 x3 = 4
1 x1+ 3 x2+ 1 x3 = 5
2 x1+ 1 x2+ 5 x3 = 6
Choose Operation
Gauss-Seidel
Initial Guess 0 0 0
Relative Error
0.005
Significant Digits
4
Answer
Result
Result X Vector :
0.2557
1.3028
0.8372
Number of iterations taken = 3
Time taken = 1e-06

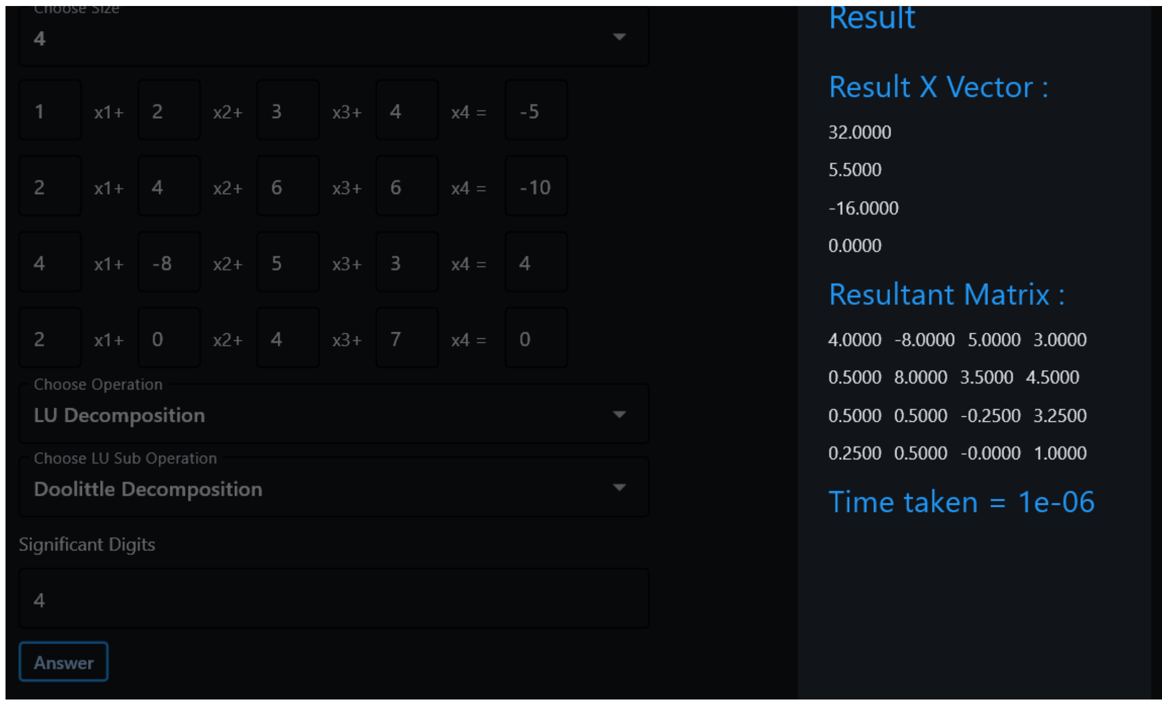
3- Converge


Matrix Solver
Alphabetical Coefficients Solver
Choose Size
4
10 x1+ 1 x2+ 1 x3+ 1 x4 = 12
1 x1+ 12 x2+ 1 x3+ 1 x4 = 13
1 x1+ 1 x2+ 15 x3+ 1 x4 = 16
1 x1+ 1 x2+ 1 x3+ 18 x4 = 17
Choose Operation
Gauss-Seidel
Initial Guess 1 1 1 1
Iterative
150
Significant Digits
4
Answer
Result
Result X Vector :
0.9449
0.8640
0.8931
0.7943
Number of iterations taken = 150
Time taken = 0.002505064010620117

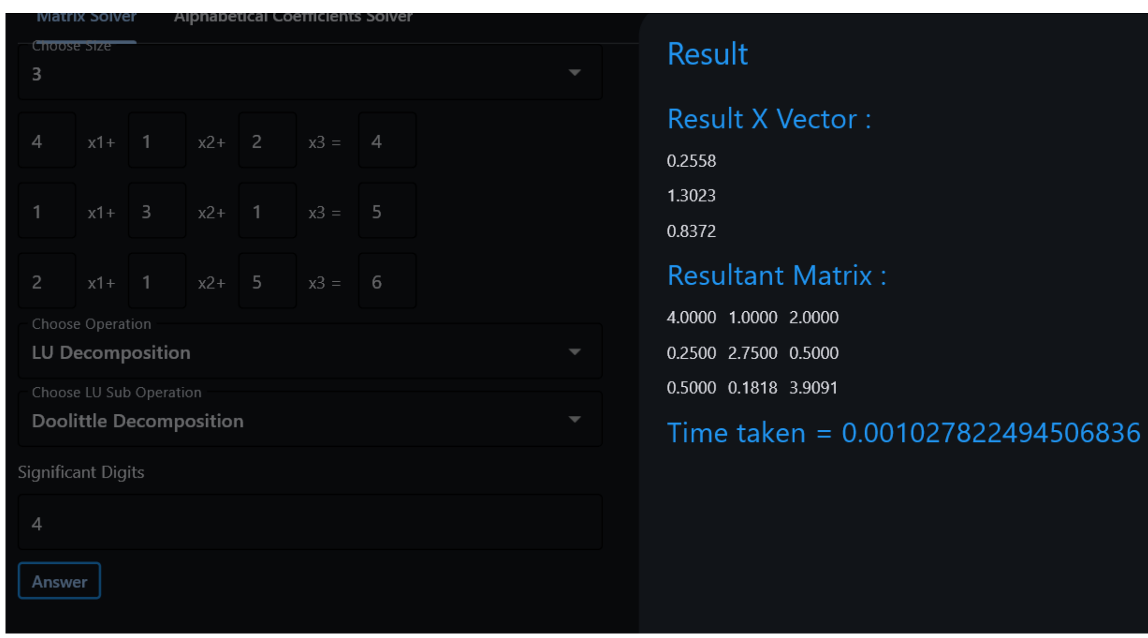

Matrix Solver
Alphabetical Coefficients Solver
Choose Size
4
10 x1+ 1 x2+ 1 x3+ 1 x4 = 12
1 x1+ 12 x2+ 1 x3+ 1 x4 = 13
1 x1+ 1 x2+ 15 x3+ 1 x4 = 16
1 x1+ 1 x2+ 1 x3+ 18 x4 = 17
Choose Operation
Gauss-Seidel
Initial Guess 1 1 1 1
Relative Error
0.005
Significant Digits
4
Answer
Result
Result X Vector :
0.9449
0.8640
0.8931
0.7943
Number of iterations taken = 1
Time taken = 0.0005359649658203125

- LU Decomposition
  - Doolittle

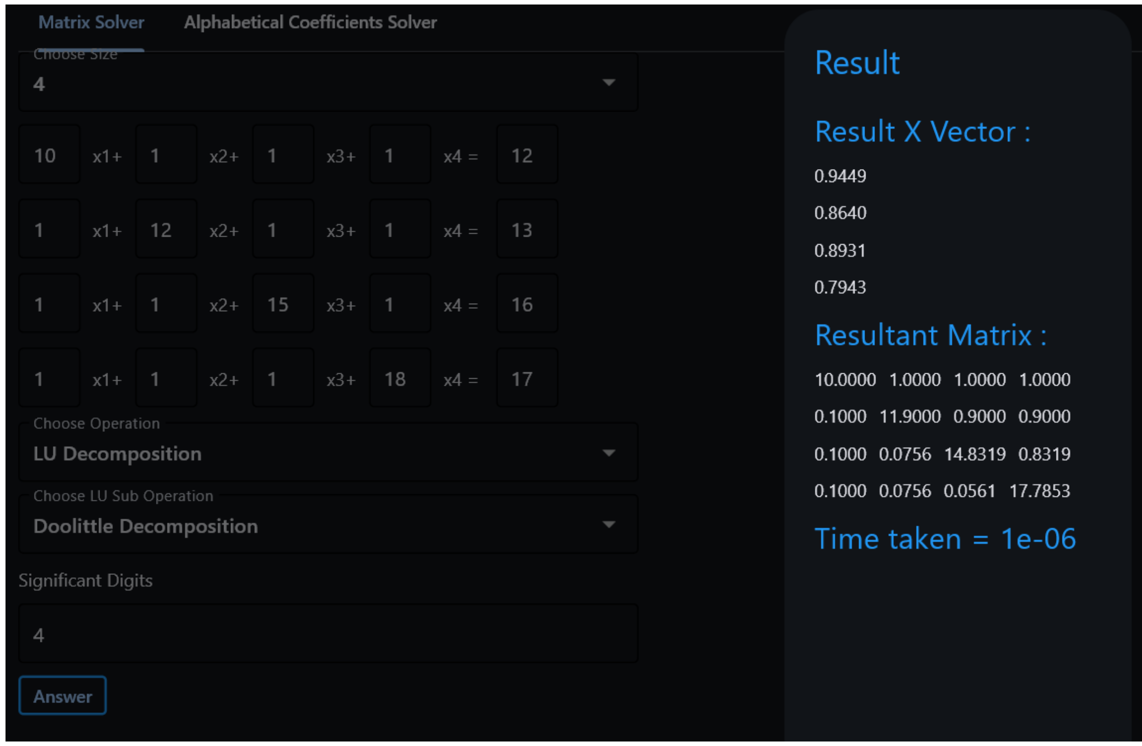
1-



2-



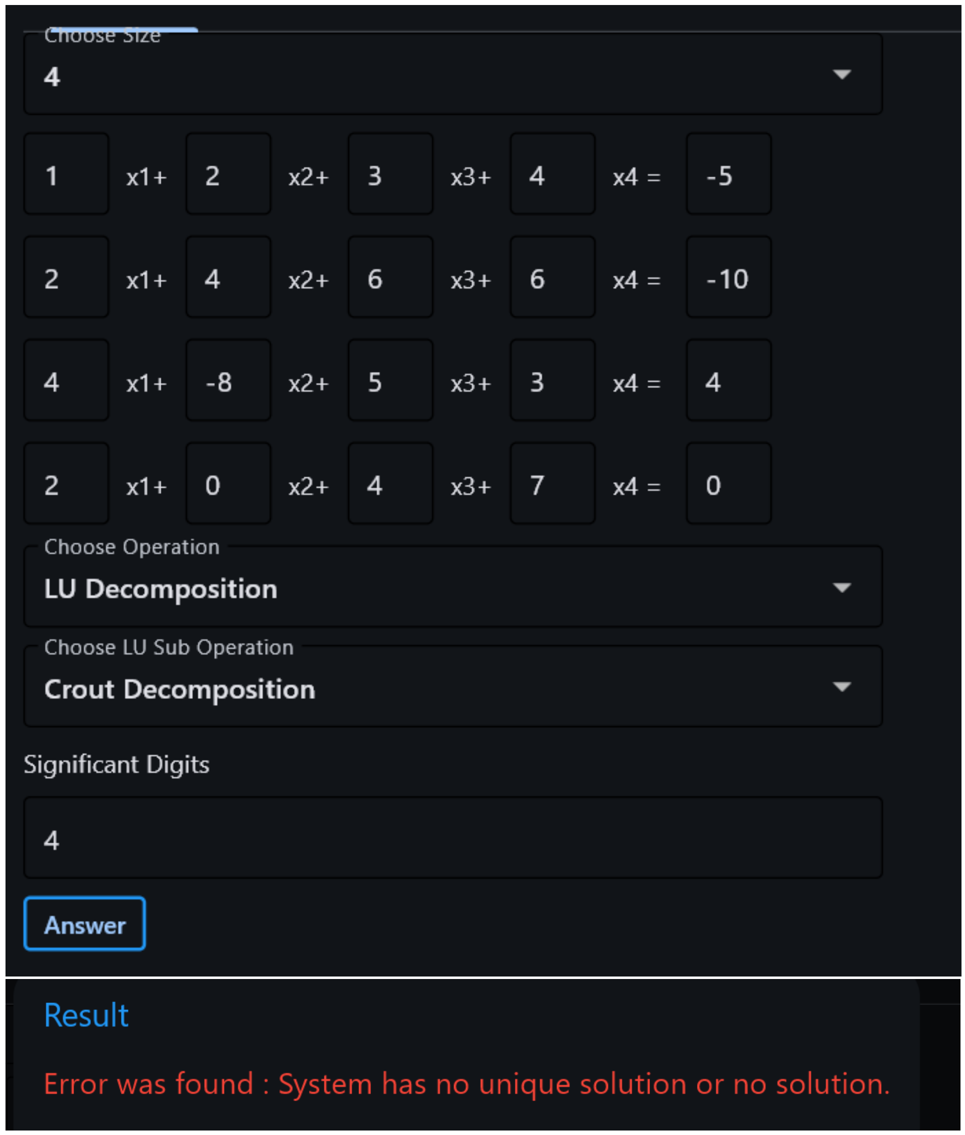
3-



Matrix Solver
Alphabetical Coefficients Solver
Choose Size
4
10 x1+ 1 x2+ 1 x3+ 1 x4 = 12
1 x1+ 12 x2+ 1 x3+ 1 x4 = 13
1 x1+ 1 x2+ 15 x3+ 1 x4 = 16
1 x1+ 1 x2+ 1 x3+ 18 x4 = 17
Choose Operation
LU Decomposition
Choose LU Sub Operation
Doolittle Decomposition
Significant Digits
4
Answer
Result
Result X Vector :
0.9449
0.8640
0.8931
0.7943
Resultant Matrix :
10.0000 1.0000 1.0000 1.0000
0.1000 11.9000 0.9000 0.9000
0.1000 0.0756 14.8319 0.8319
0.1000 0.0756 0.0561 17.7853
Time taken = 1e-06

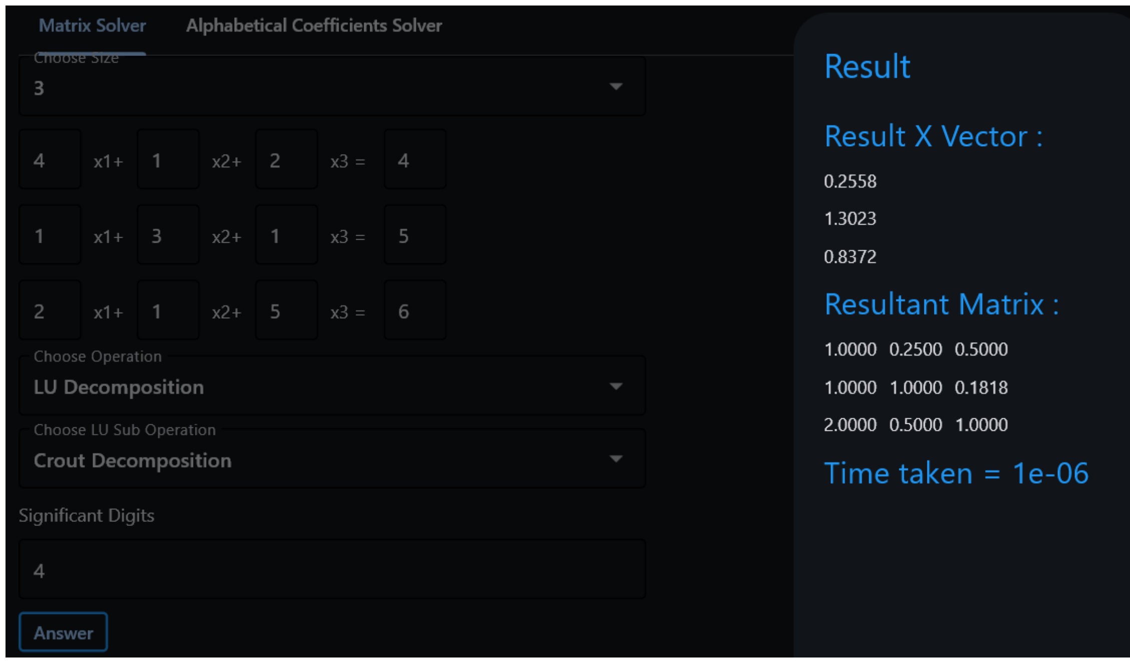
- Crout

  1-

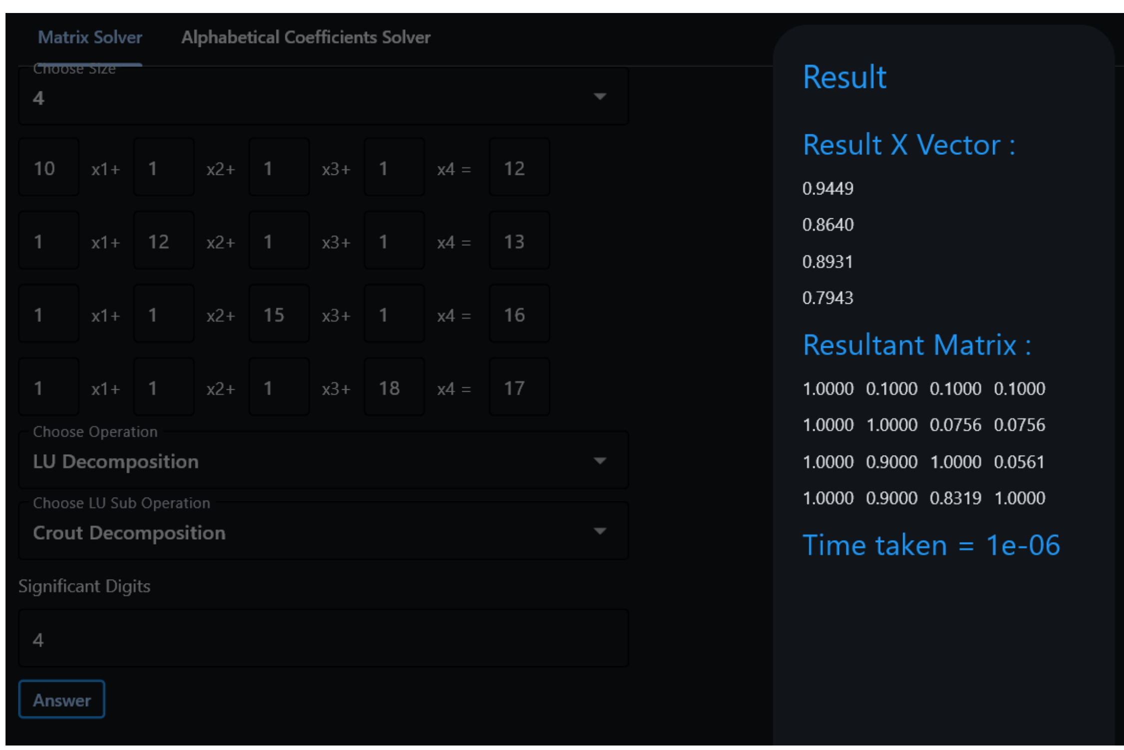


Choose Size
4
1 x1+ 2 x2+ 3 x3+ 4 x4 = -5
2 x1+ 4 x2+ 6 x3+ 6 x4 = -10
4 x1+ -8 x2+ 5 x3+ 3 x4 = 4
2 x1+ 0 x2+ 4 x3+ 7 x4 = 0
Choose Operation
LU Decomposition
Choose LU Sub Operation
Crout Decomposition
Significant Digits
4
Answer
Result
Error was found : System has no unique solution or no solution.

2-

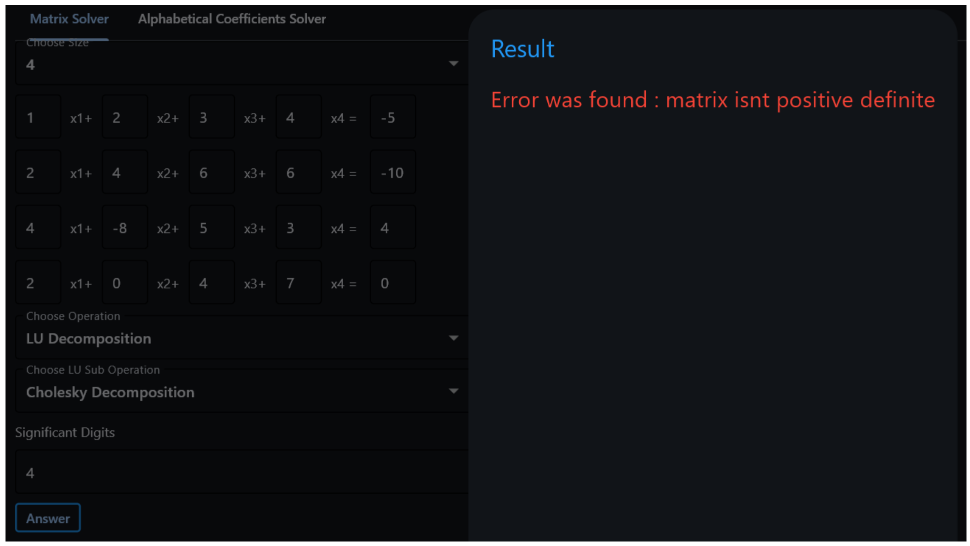

Matrix Solver
Alphabetical Coefficients Solver
Choose Size
3
4 x1+ 1 x2+ 2 x3 = 4
1 x1+ 3 x2+ 1 x3 = 5
2 x1+ 1 x2+ 5 x3 = 6
Choose Operation
LU Decomposition
Choose LU Sub Operation
Crout Decomposition
Significant Digits
4
Answer
Result
Result X Vector :
0.2558
1.3023
0.8372
Resultant Matrix :
1.0000 0.2500 0.5000
1.0000 1.0000 0.1818
2.0000 0.5000 1.0000
Time taken = 1e-06

3-

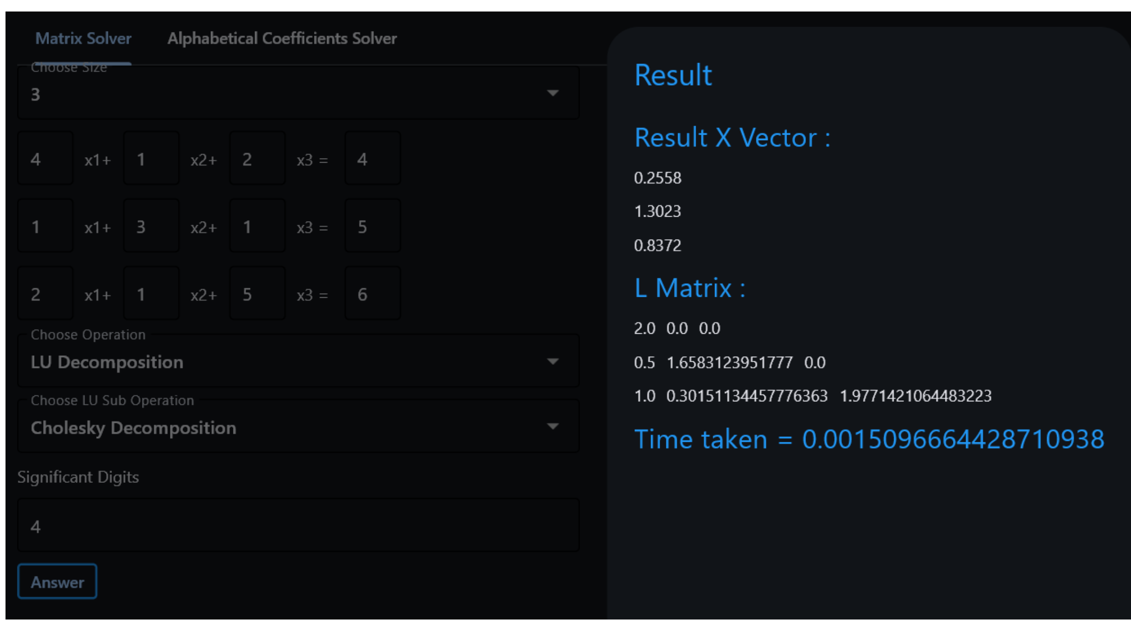

Matrix Solver
Alphabetical Coefficients Solver
Choose Size
4
10 x1+ 1 x2+ 1 x3+ 1 x4 = 12
1 x1+ 12 x2+ 1 x3+ 1 x4 = 13
1 x1+ 1 x2+ 15 x3+ 1 x4 = 16
1 x1+ 1 x2+ 1 x3+ 18 x4 = 17
Choose Operation
LU Decomposition
Choose LU Sub Operation
Crout Decomposition
Significant Digits
4
Answer
Result
Result X Vector :
0.9449
0.8640
0.8931
0.7943
Resultant Matrix :
1.0000 0.1000 0.1000 0.1000
1.0000 1.0000 0.0756 0.0756
1.0000 0.9000 1.0000 0.0561
1.0000 0.9000 0.8319 1.0000
Time taken = 1e-06

- Cholesky

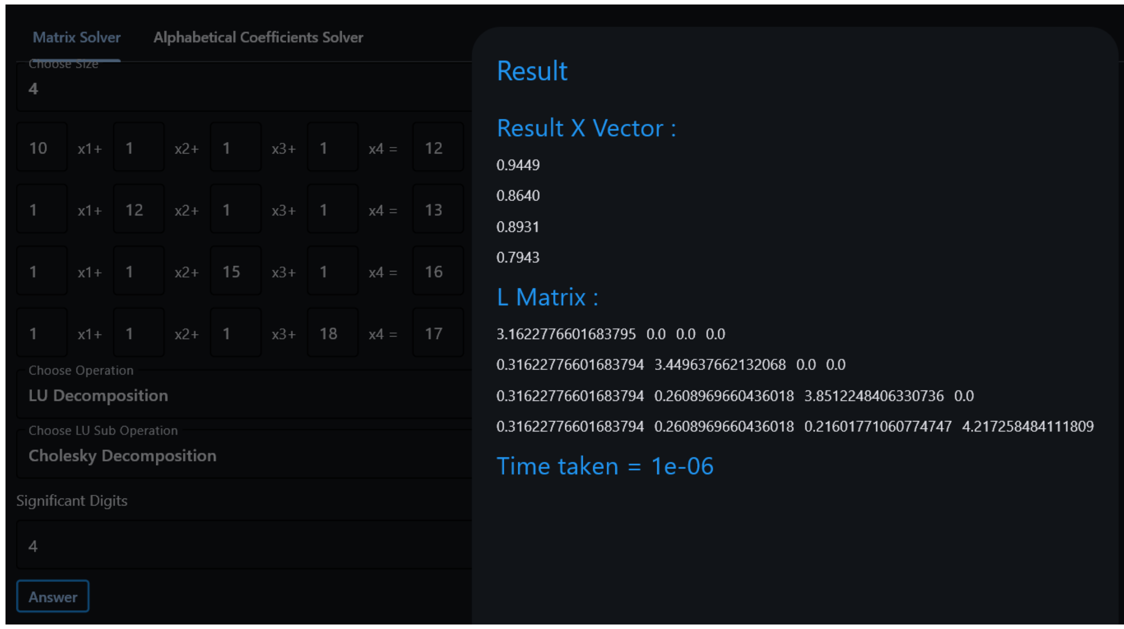
1-


Matrix Solver
Alphabetical Coefficients Solver
Choose Size
4
1 x1+ 2 x2+ 3 x3+ 4 x4 = -5
2 x1+ 4 x2+ 6 x3+ 6 x4 = -10
4 x1+ -8 x2+ 5 x3+ 3 x4 = 4
2 x1+ 0 x2+ 4 x3+ 7 x4 = 0
Choose Operation
LU Decomposition
Choose LU Sub Operation
Cholesky Decomposition
Significant Digits
4
Answer
Result
Error was found : matrix isnt positive definite

2-

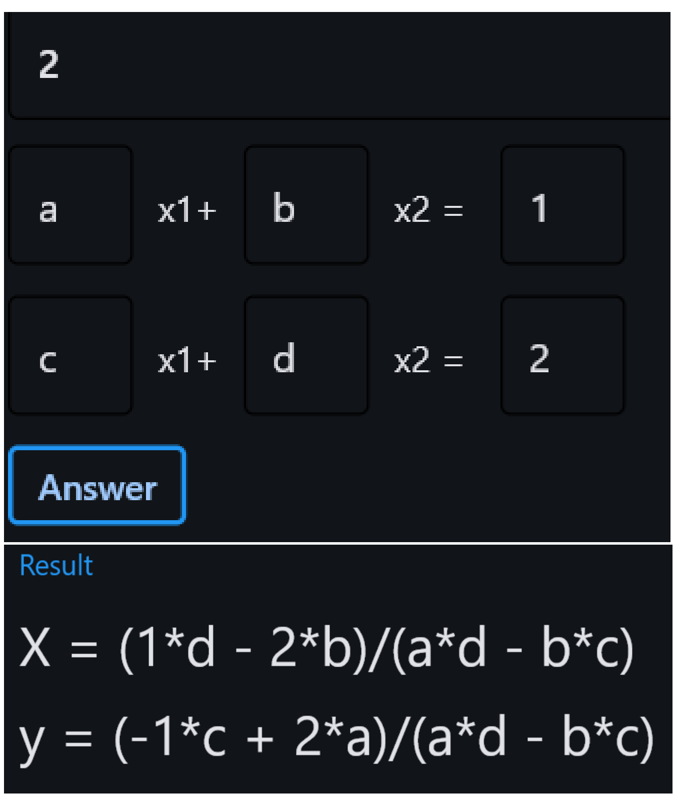

Matrix Solver
Alphabetical Coefficients Solver
Choose Size
3
4 x1+ 1 x2+ 2 x3 = 4
1 x1+ 3 x2+ 1 x3 = 5
2 x1+ 1 x2+ 5 x3 = 6
Choose Operation
LU Decomposition
Choose LU Sub Operation
Cholesky Decomposition
Significant Digits
4
Answer
Result
Result X Vector :
0.2558
1.3023
0.8372
L Matrix :
2.0 0.0 0.0
0.5 1.6583123951777 0.0
1.0 0.30151134457776363 1.9771421064483223
Time taken = 0.0015096664428710938

3-

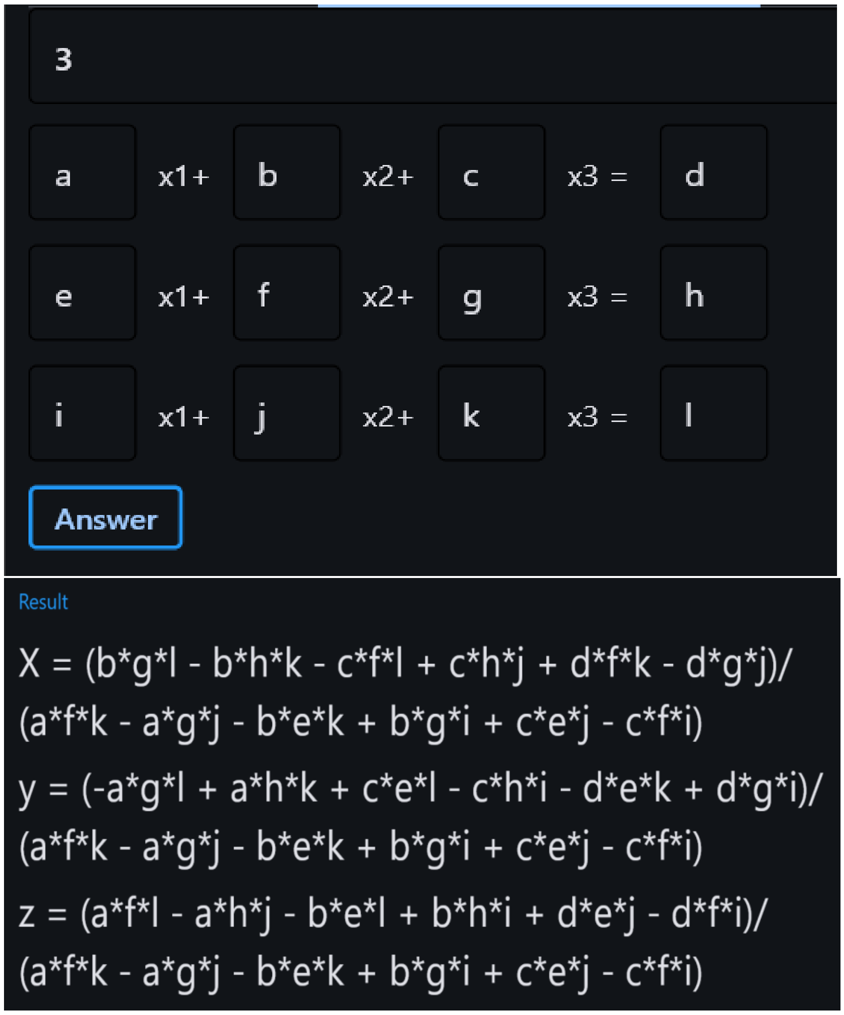


- Bonus: Coefficients can be letters

1-



2-



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

Method	*Time Complexity*	*Convergence*	*Error*
Gauss Elimination	$O(n^3)$	Direct method, no convergence issue	Numerical errors for ill-conditioned matrices
Gauss-Jordan	$O(n^3)$	Direct method, no convergence issue	Numerical errors for ill-conditioned matrices
Jacobi	$O(n^2)$ / Iteration	Converges for diagonally dominant or	Slower convergence than Gauss-Seidel

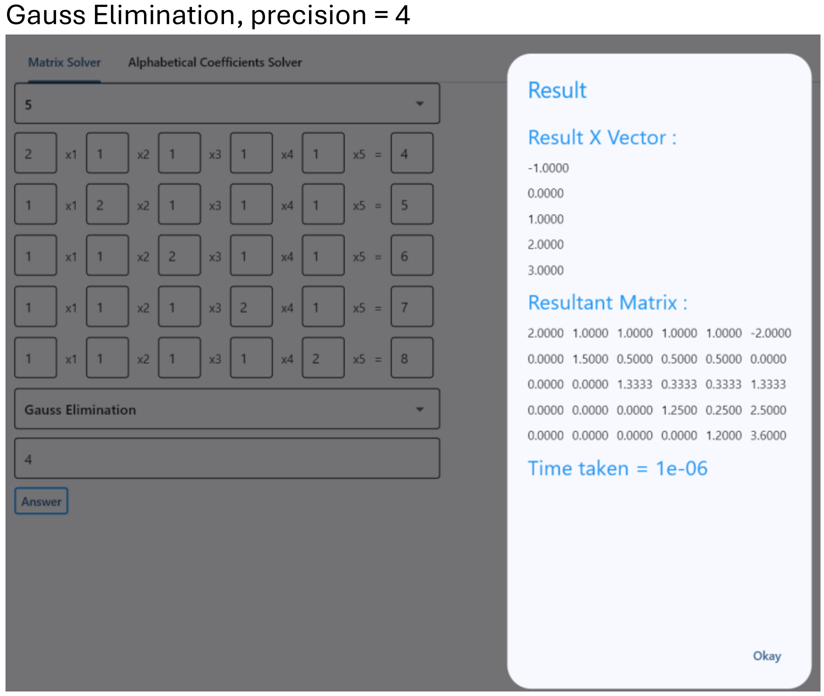
		positive-definite matrices	
Gauss-Seidel	$O(n^2)$ / Iteration	Converges for diagonally dominant or positive-definite matrices	Error decreases with each iteration
LU Decomposition	$O(n^3)$	Converges for non-singular matrices	Sensitive to ill-conditioned matrices

4. *Data structures used:*

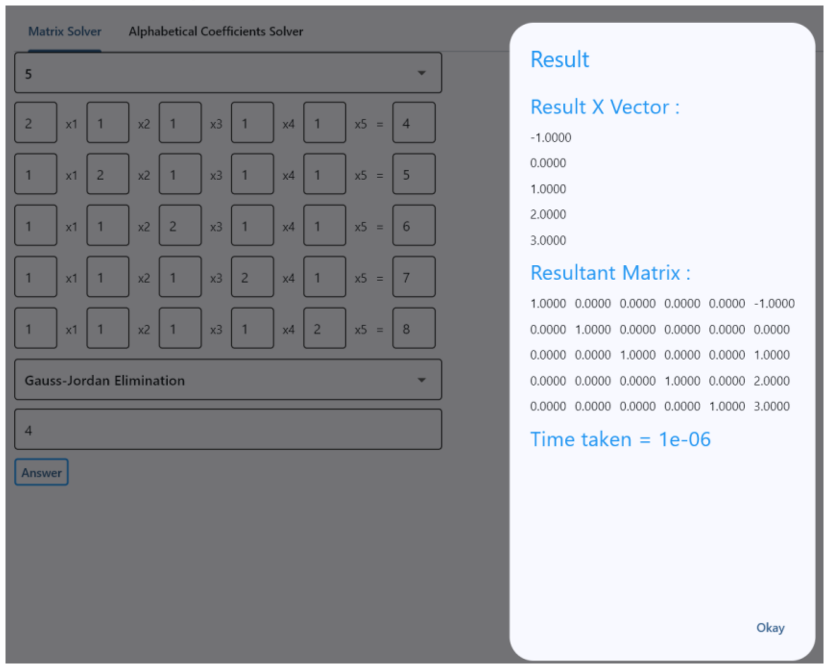
- 1D Lists (Arrays)
- 2D Lists (Matrices)
- Temporary (Scalar) variables
- Flags (Booleans)
- Indices

5. *Test cases:*

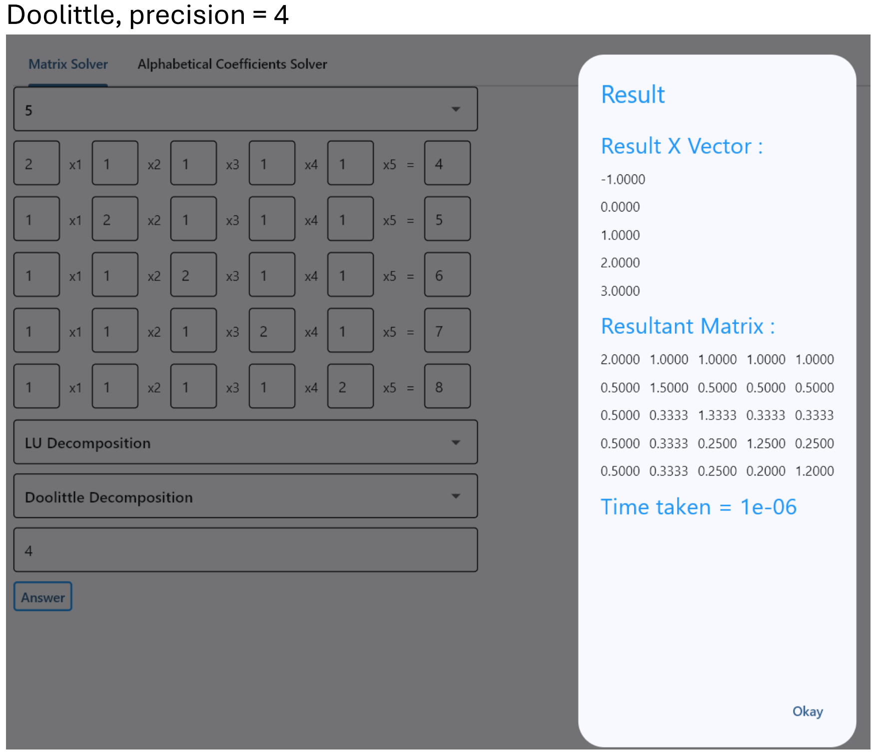
- 1st case
  - Gauss Elimination, precision = 4



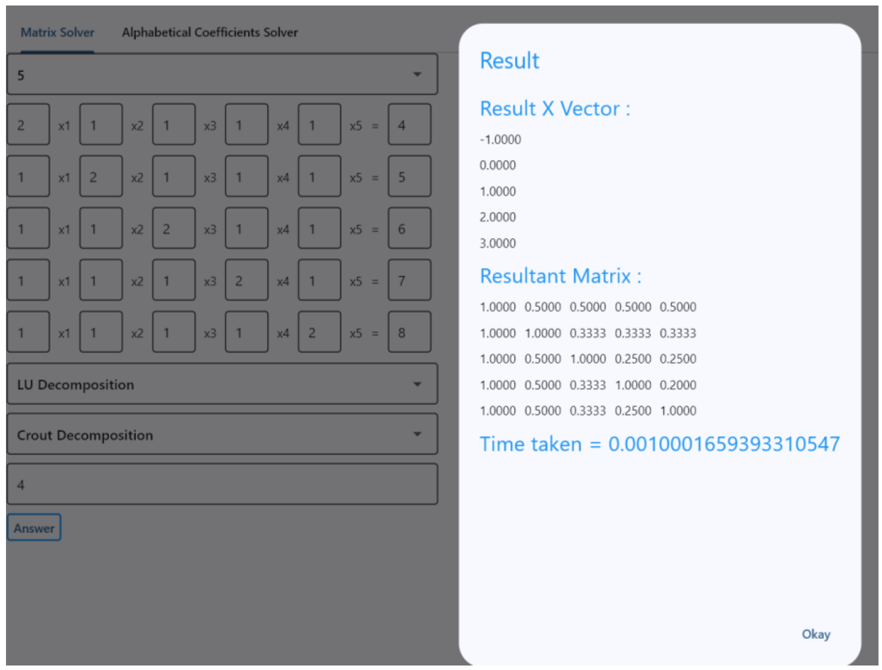
- Gauss Jordan, precision = 4



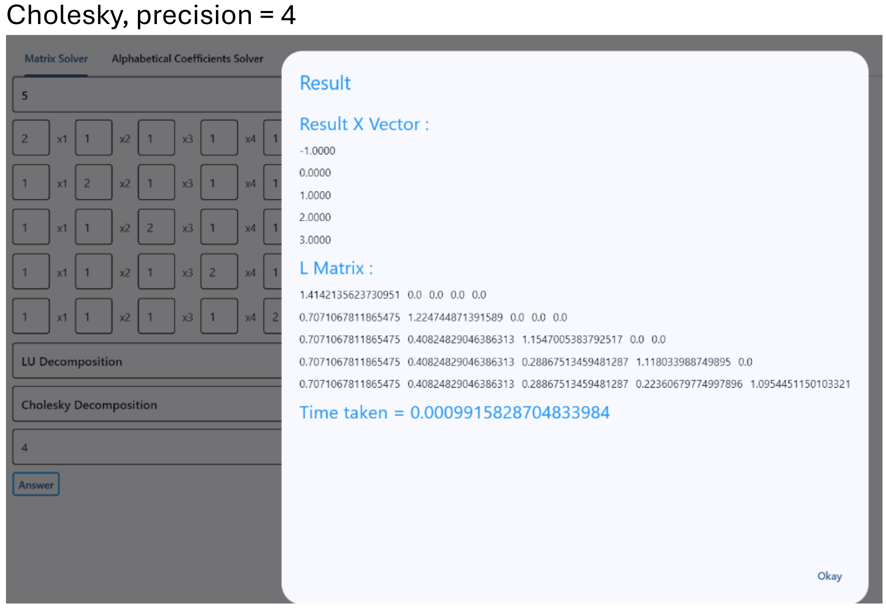
- LU decomposition
  - Doolittle, precision = 4



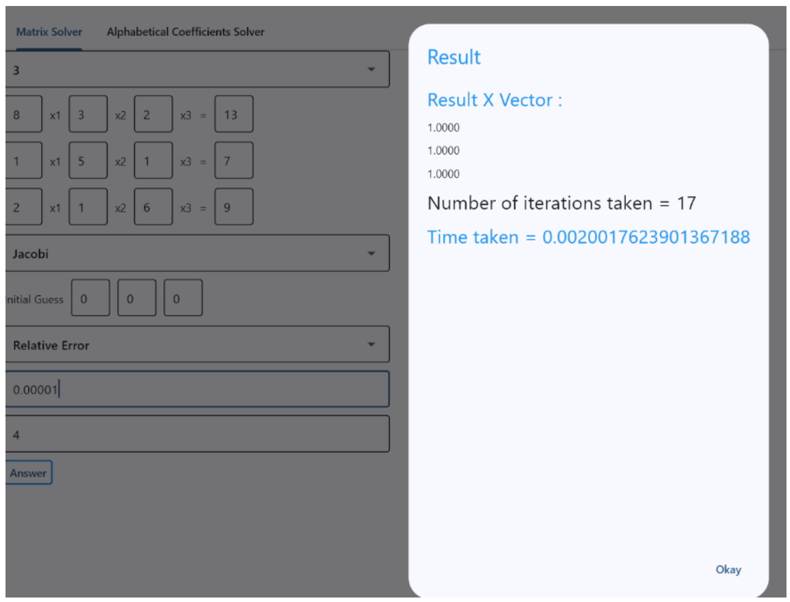
- Crout, precision = 4



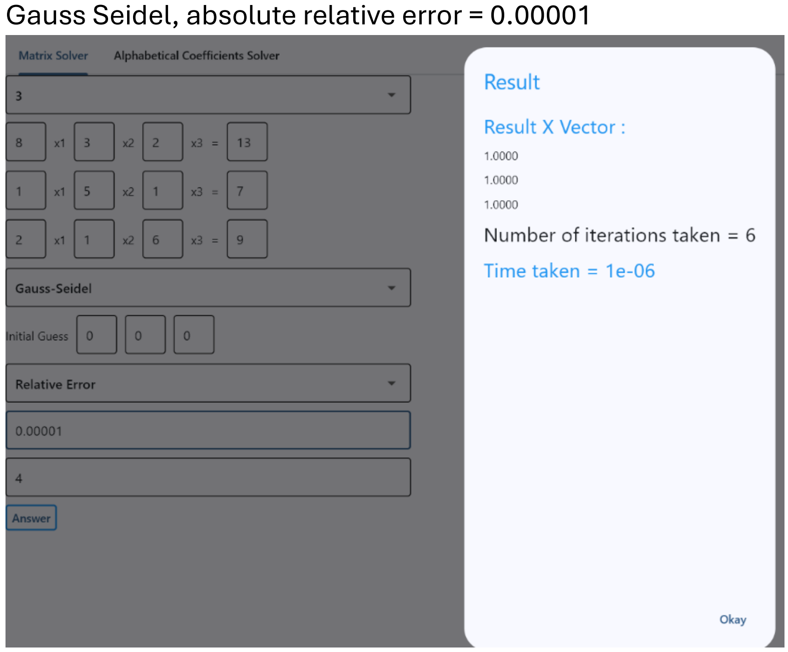
- Cholesky, precision = 4



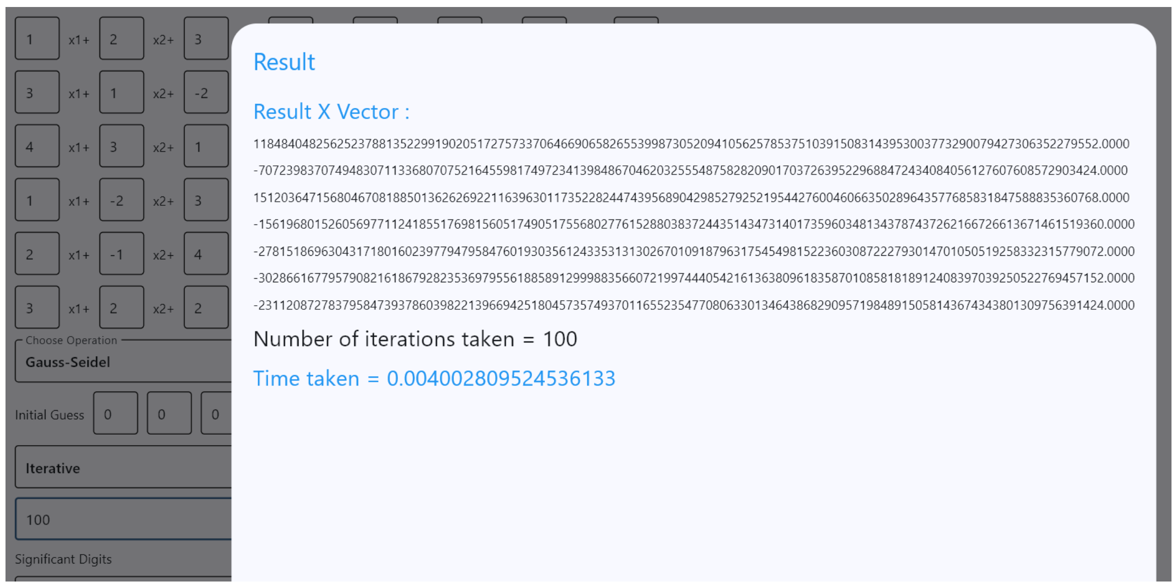
- 2nd case
  - Jacobi, absolute relative error = 0.00001



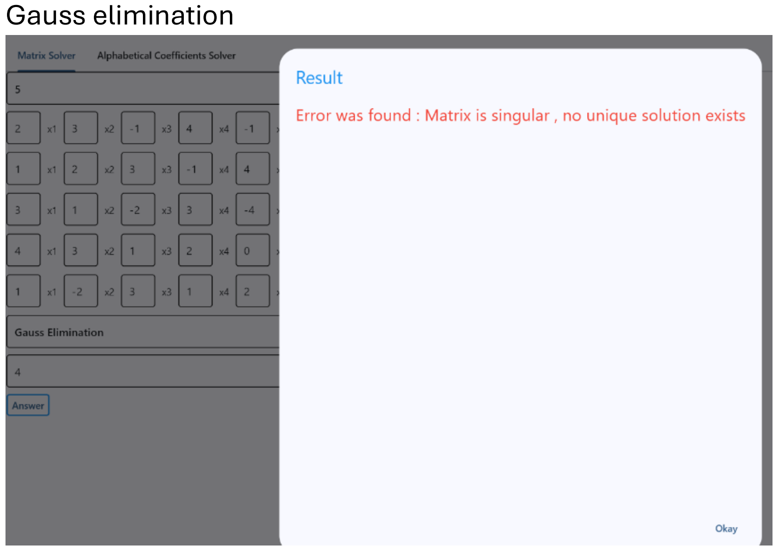
  - Gauss Seidel, absolute relative error = 0.00001



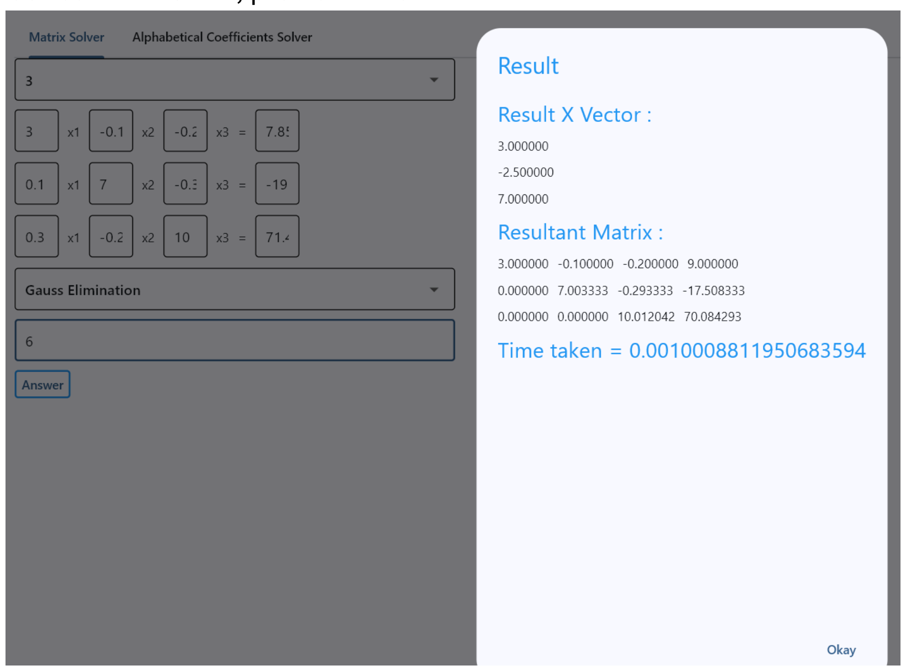
- ❖ 3rd case
  - ➢ Gauss Seidel, number of iterations = 100



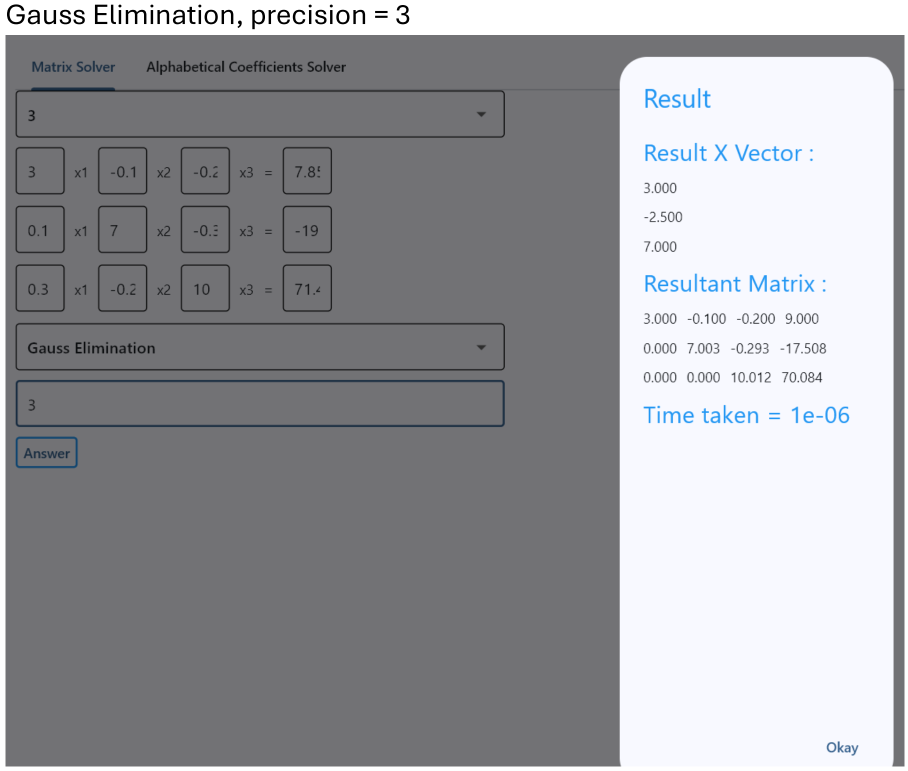
- ❖ 4th case
  - ➢ Gauss elimination



- 5th case
  - Gauss Elimination, precision = 6

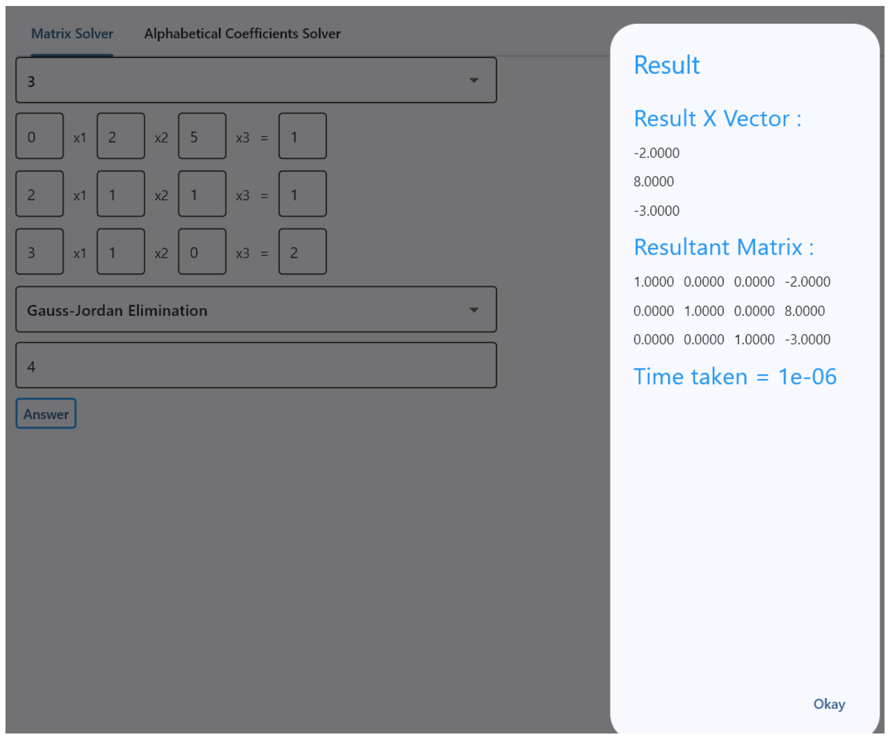


  - Gauss Elimination, precision = 3



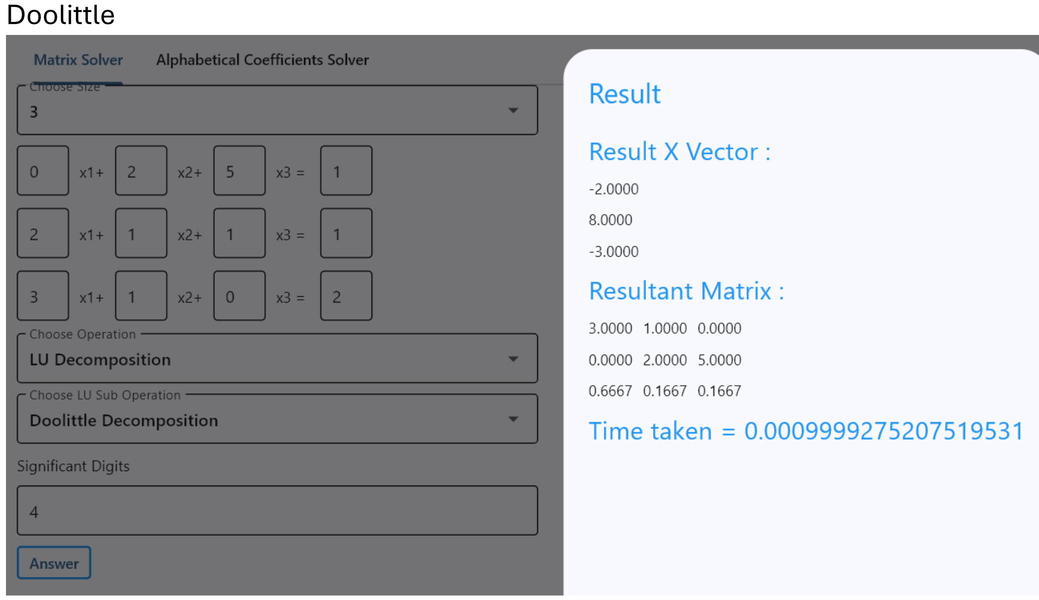
❖ 6th case

➢ Gauss Jordan

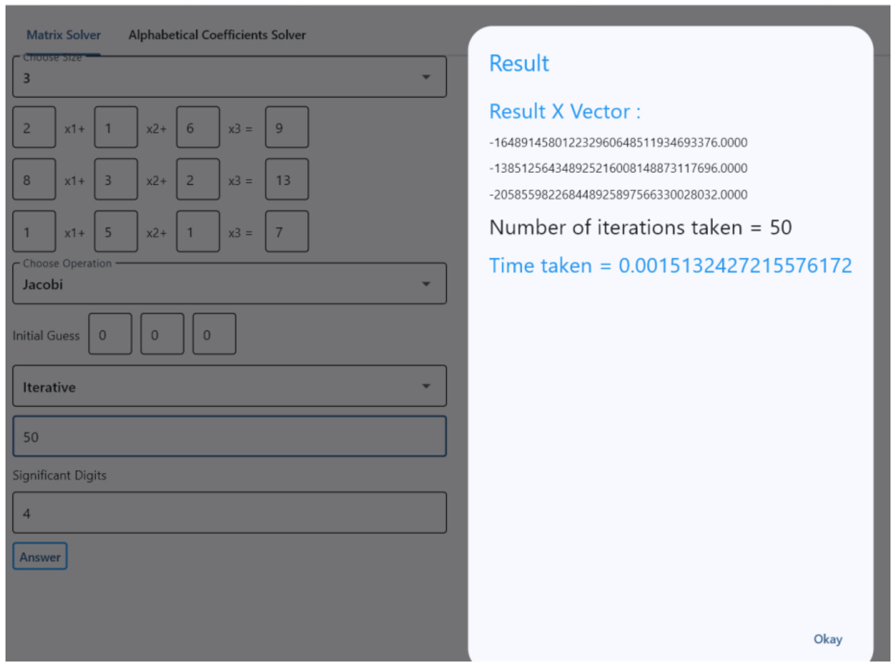


➢ LU Decomposition

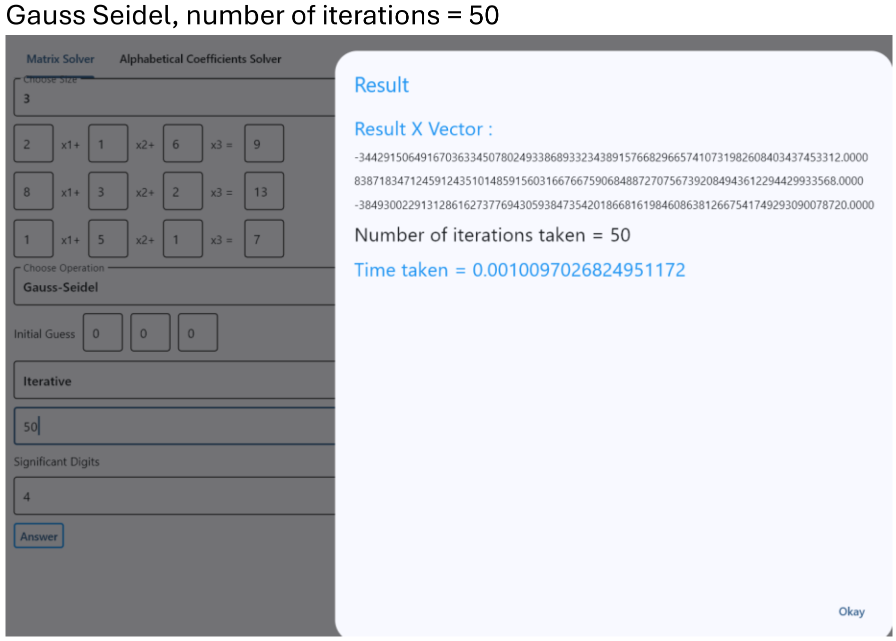
- Doolittle



- 7th case
  - Jacobi, number of iterations = 50



  - Gauss Seidel, number of iterations = 50



6. *Bonus:*

Coefficients can be letters and the output is expressed in term of the letters:

Suitable for system of equations of 2 or 3 variables. Any number of variables bigger than that is too much load for the computer.

Notes:

1- To run the app, open “RUN ME FOR PROJECT.exe”
2- You may need to install some python libraries

   Run the following in a terminal:

   pip install flask flask-cors numpy sympy