***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

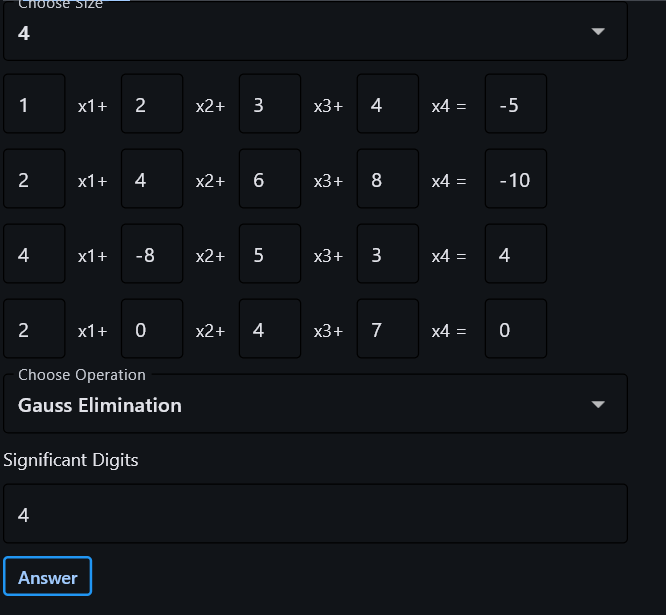
X ,augmented\_UY = backward\_substitution(augmented\_UY, n)

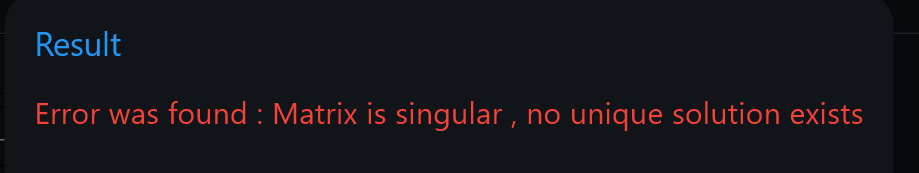
return X , L\_and\_U

1. ***Sample runs for each method:***

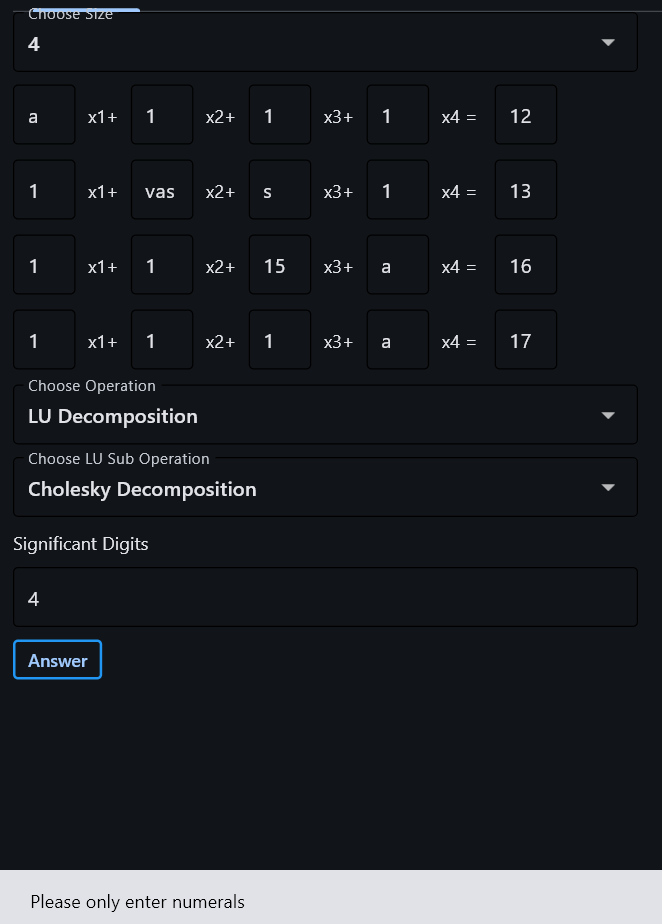
* General Cases Independent of Method:

1. A singular matrix will return an error

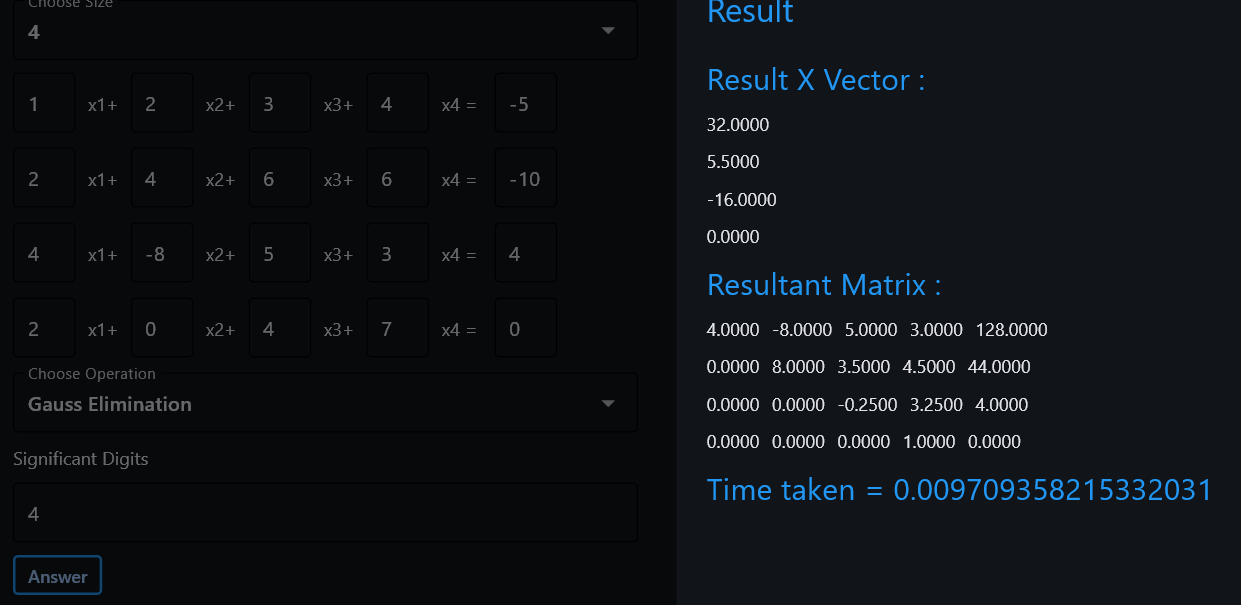


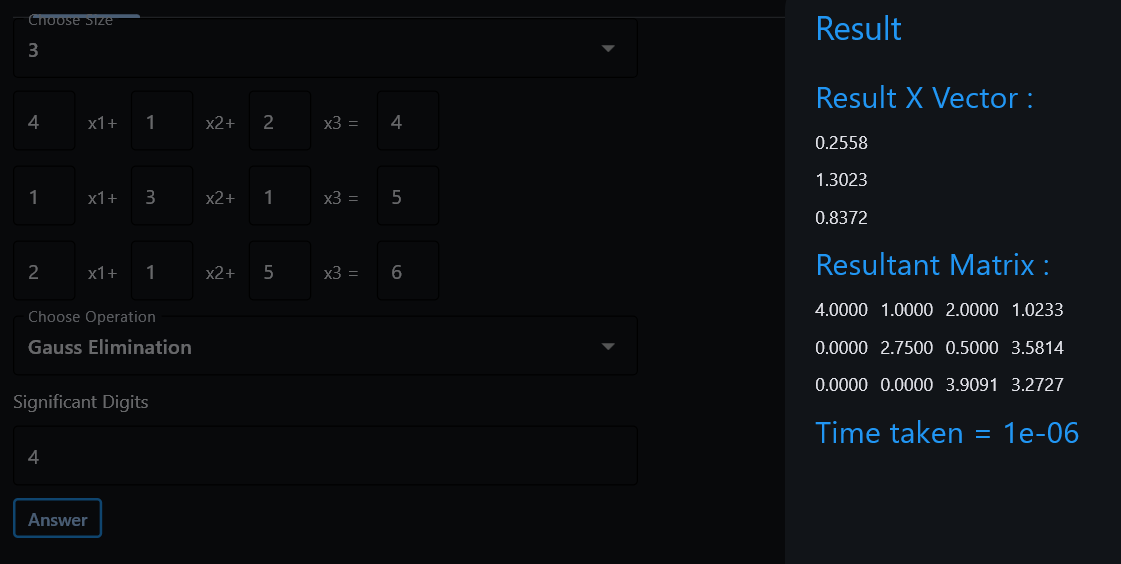


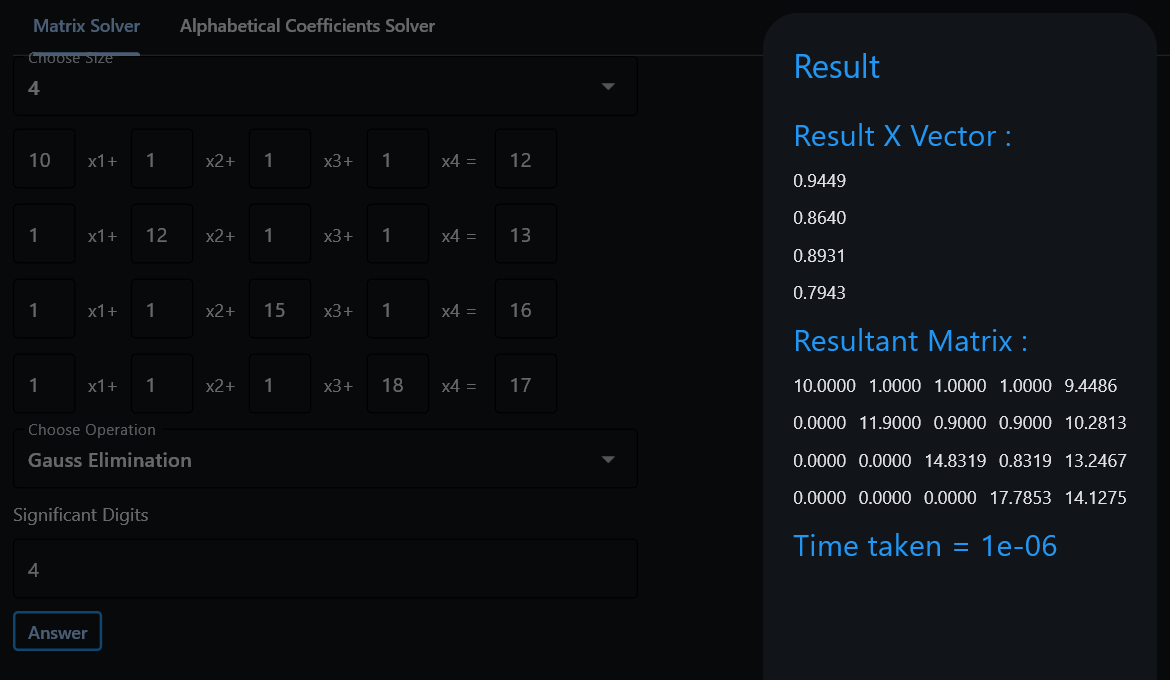
1. Entered a non-number or left some fields



* Gauss Elimination

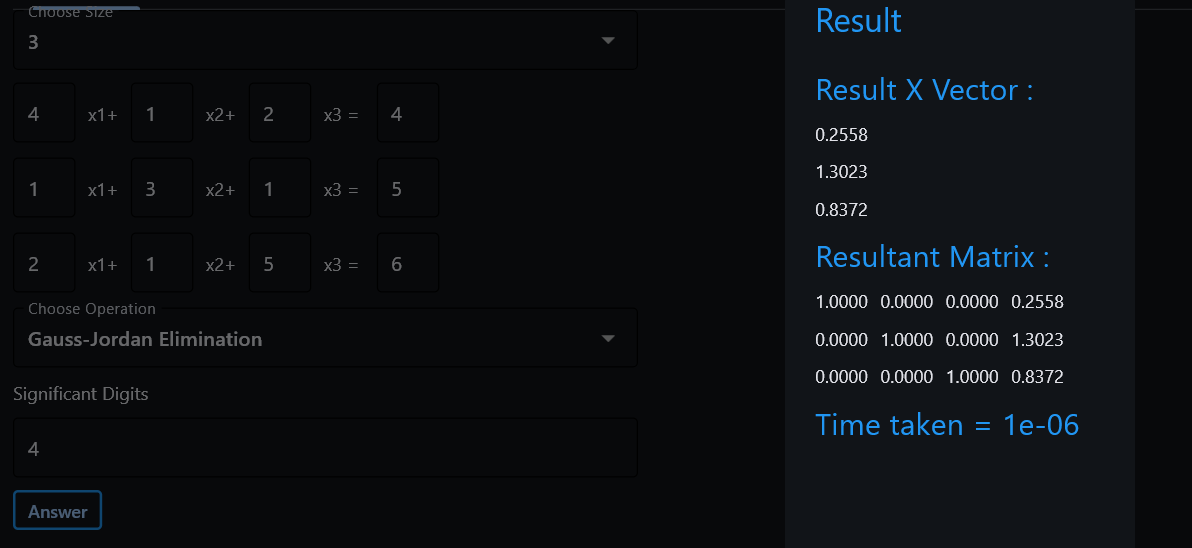


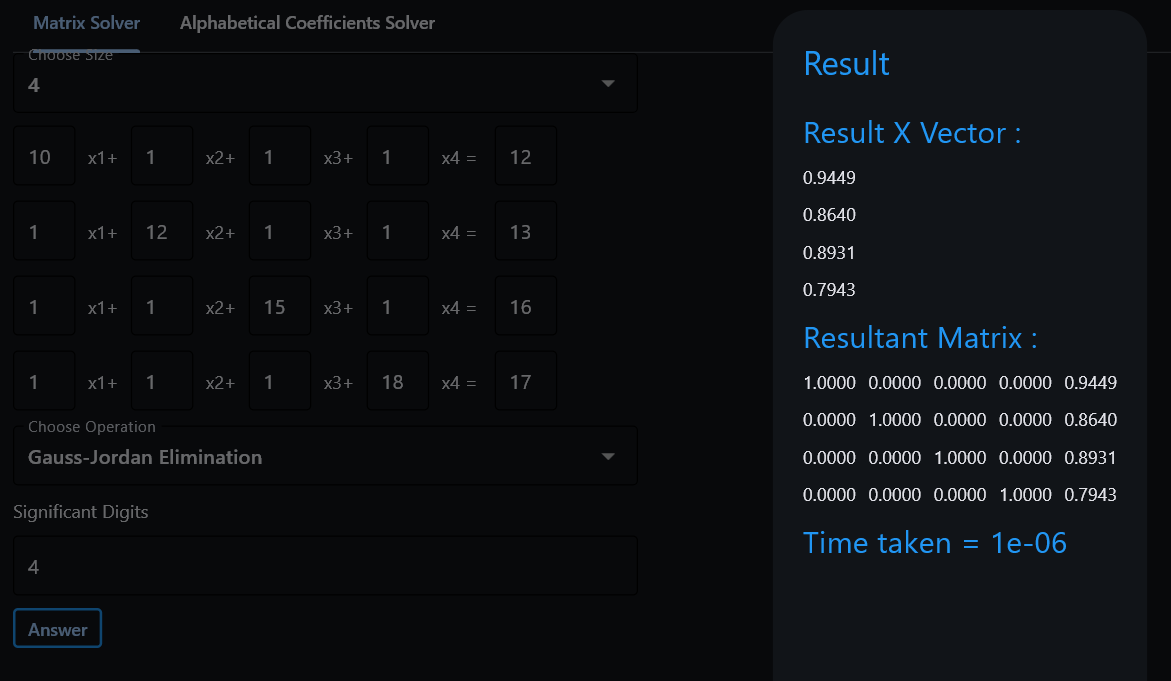




* Gauss-Jordan

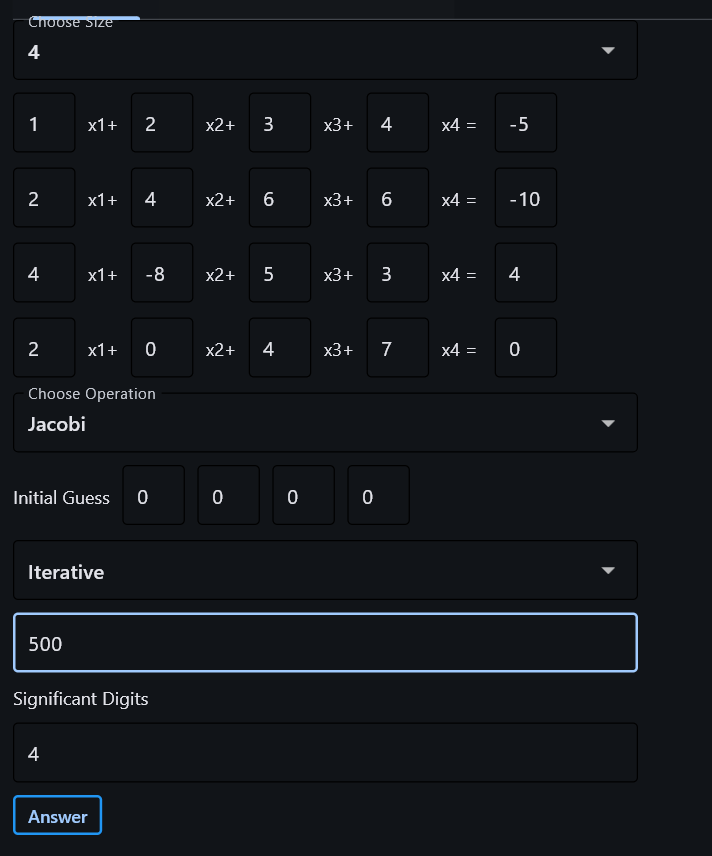


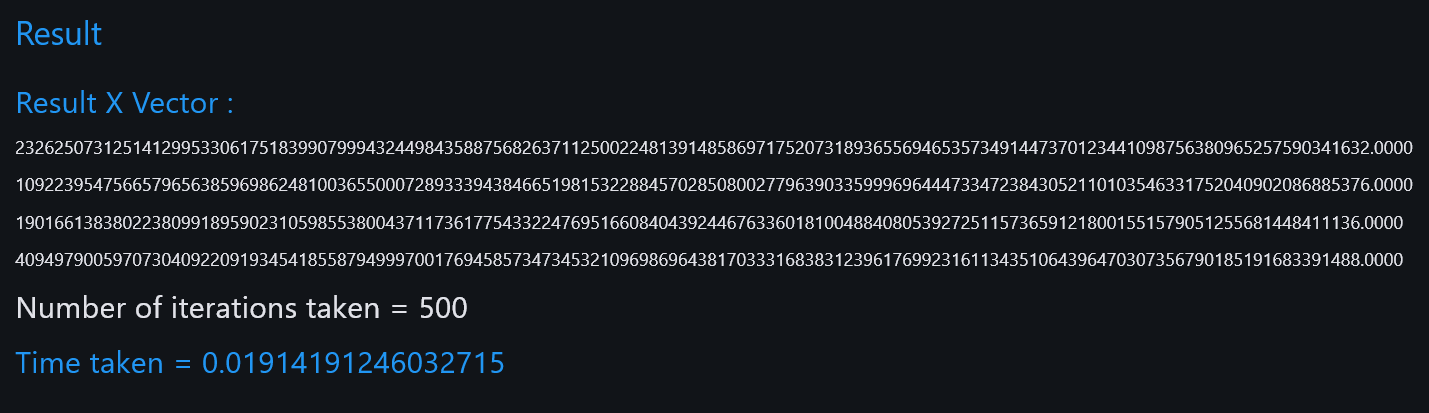




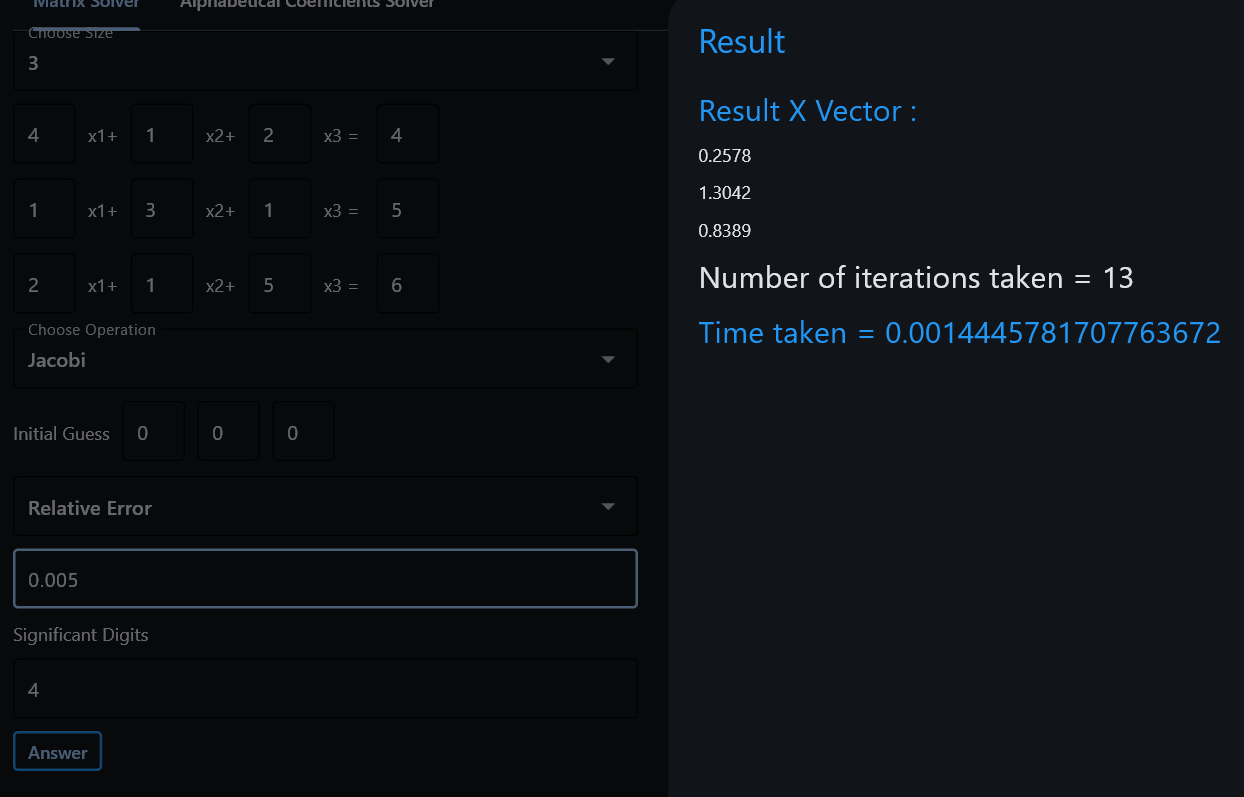
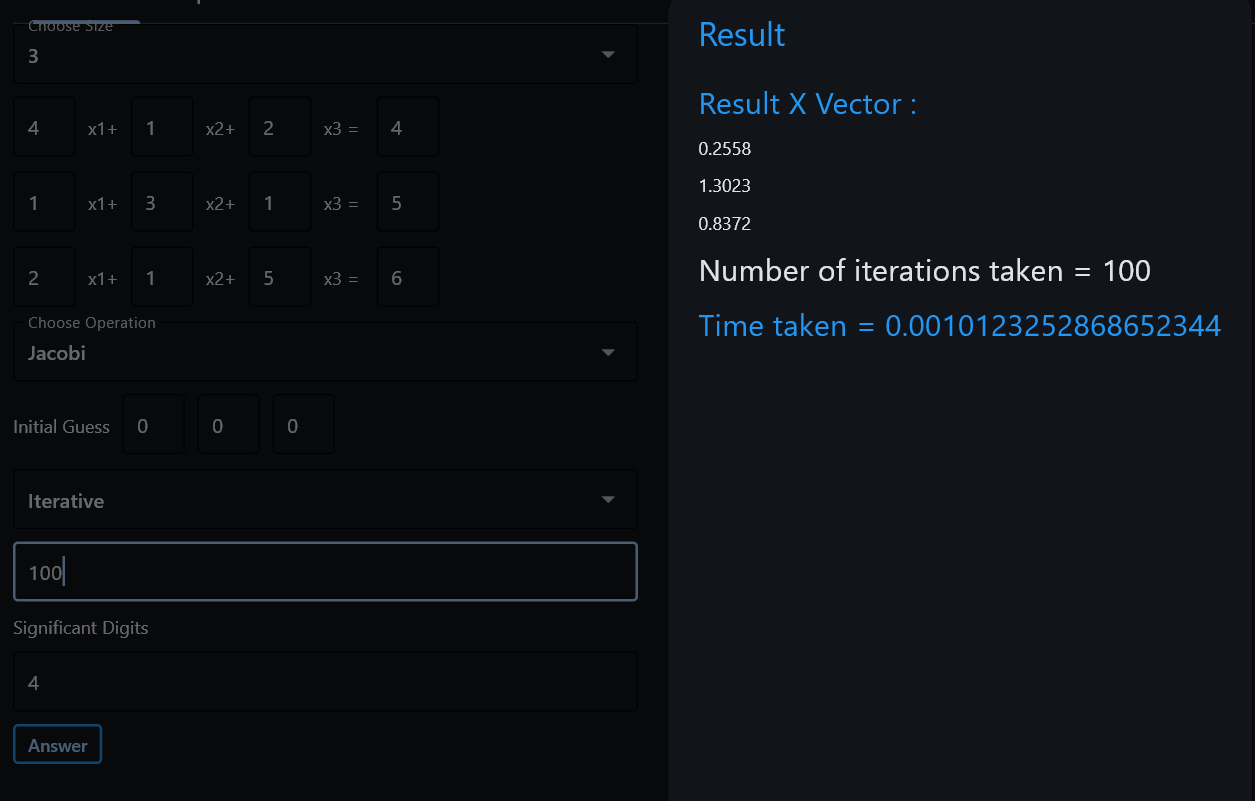
* Jacobi

1. Diverge

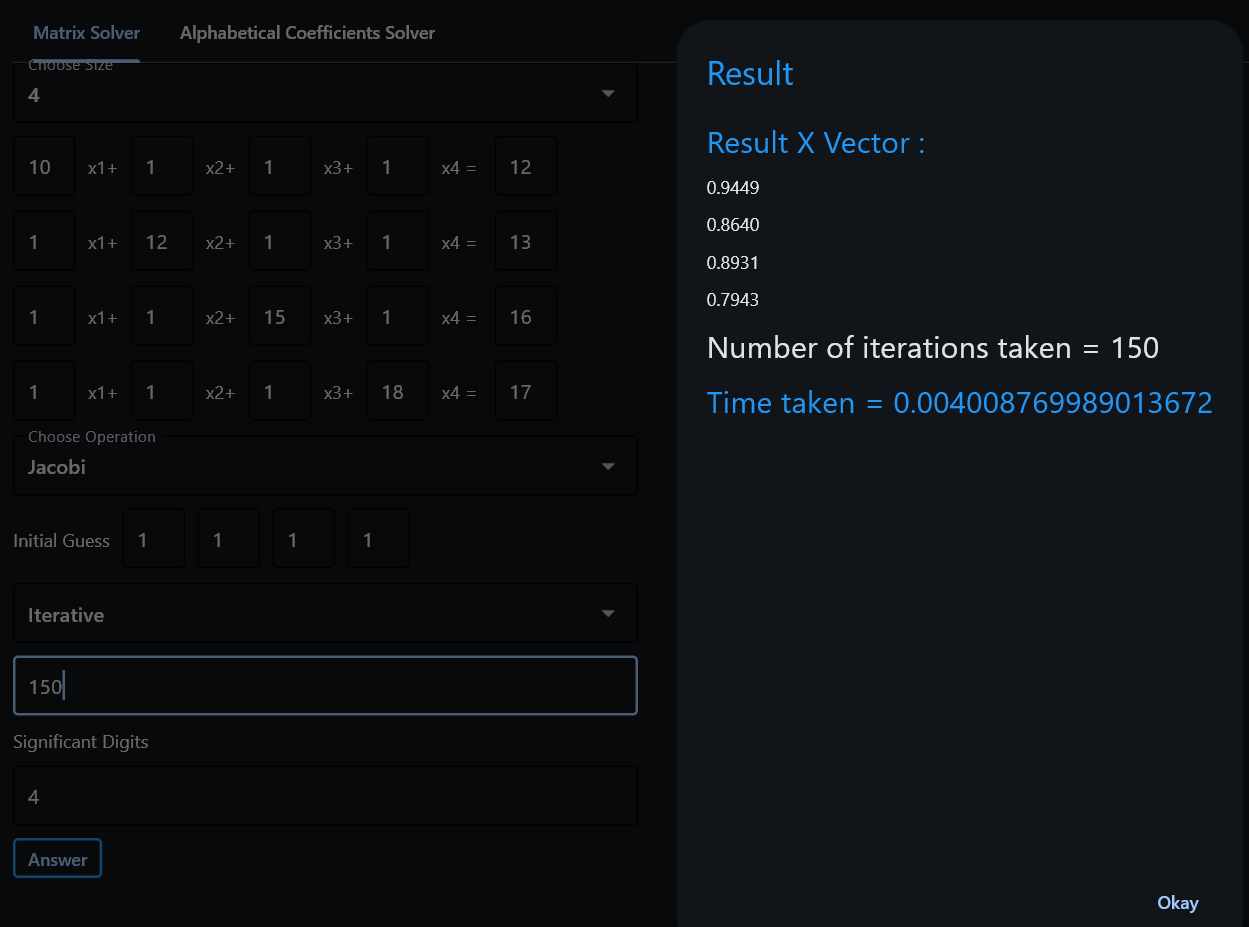


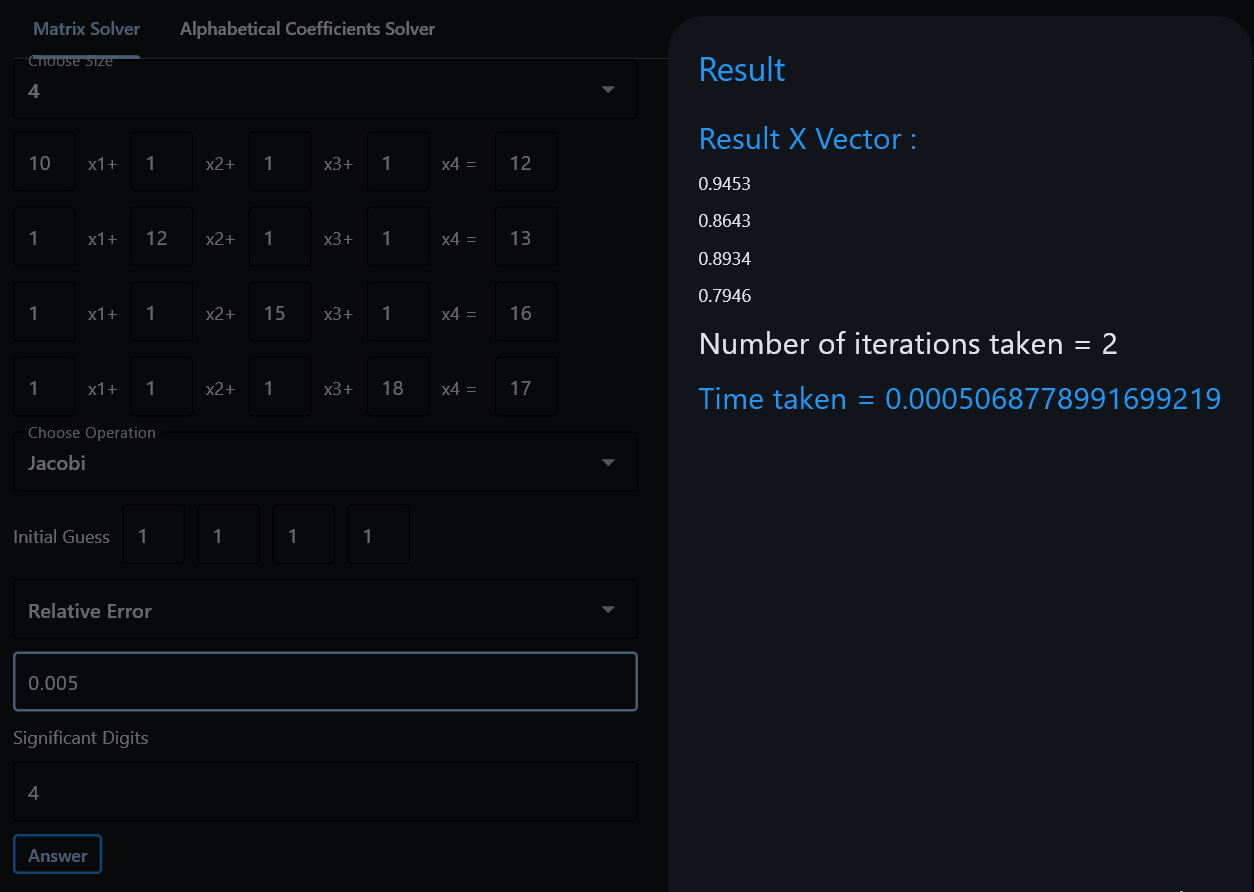


1. Converge



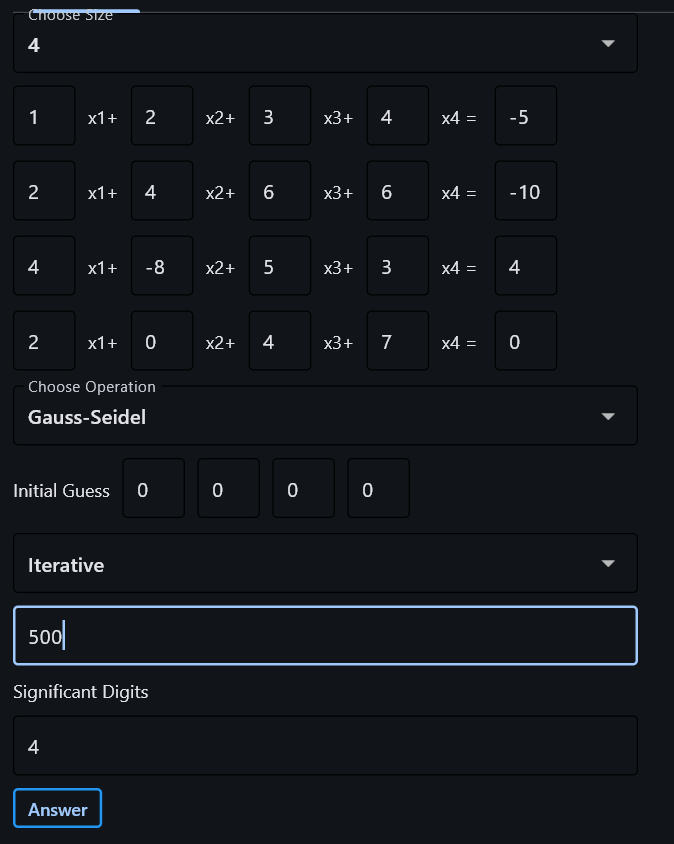
1. Converge

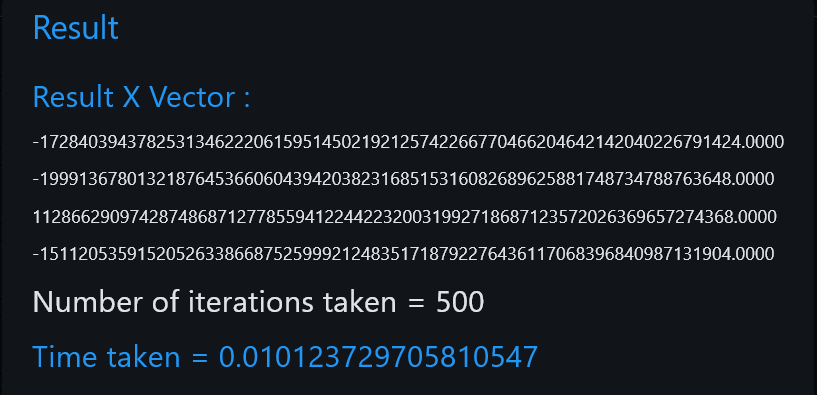




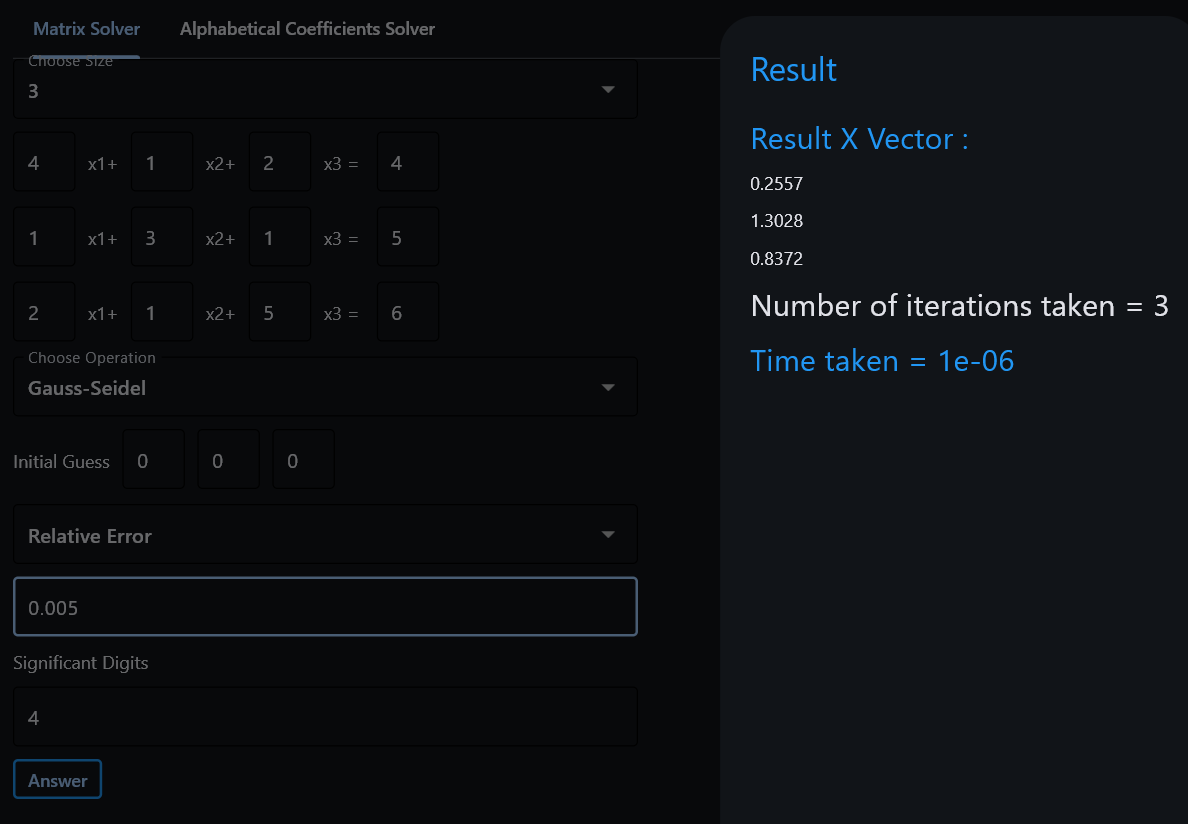
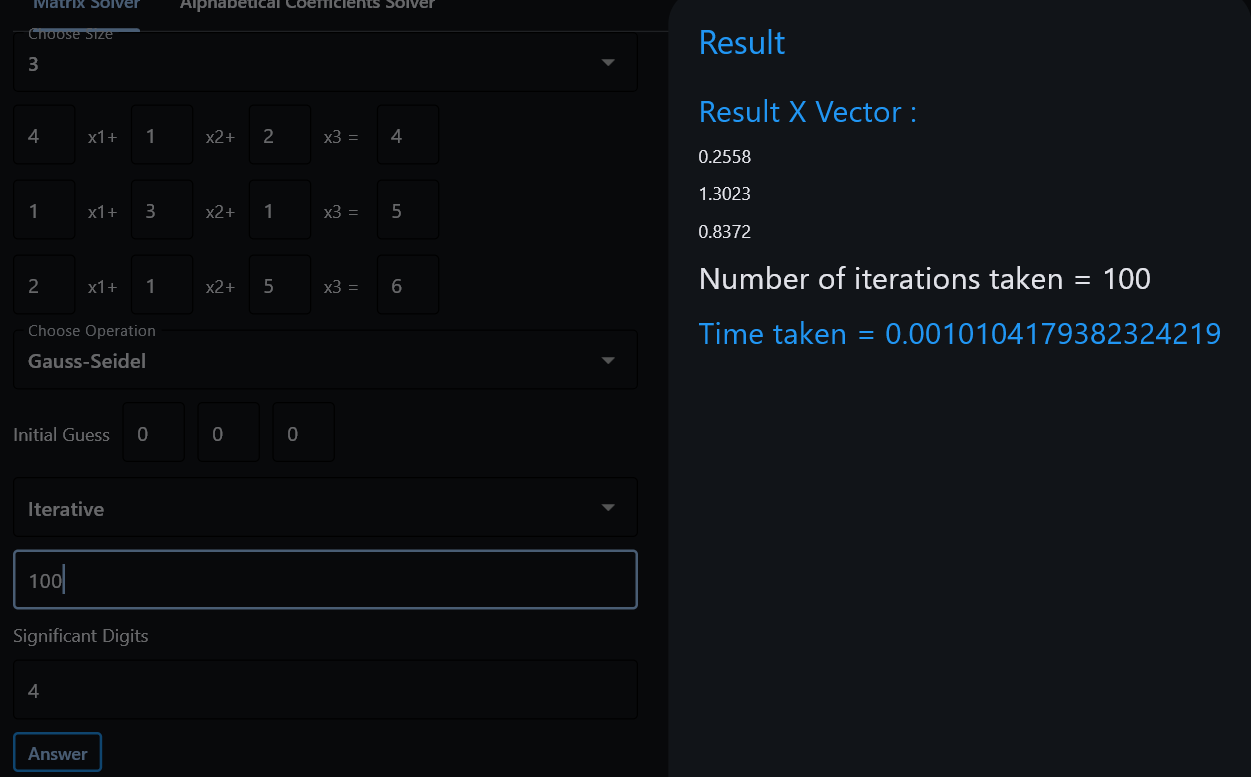
* Gauss-Seidel

1. Diverge

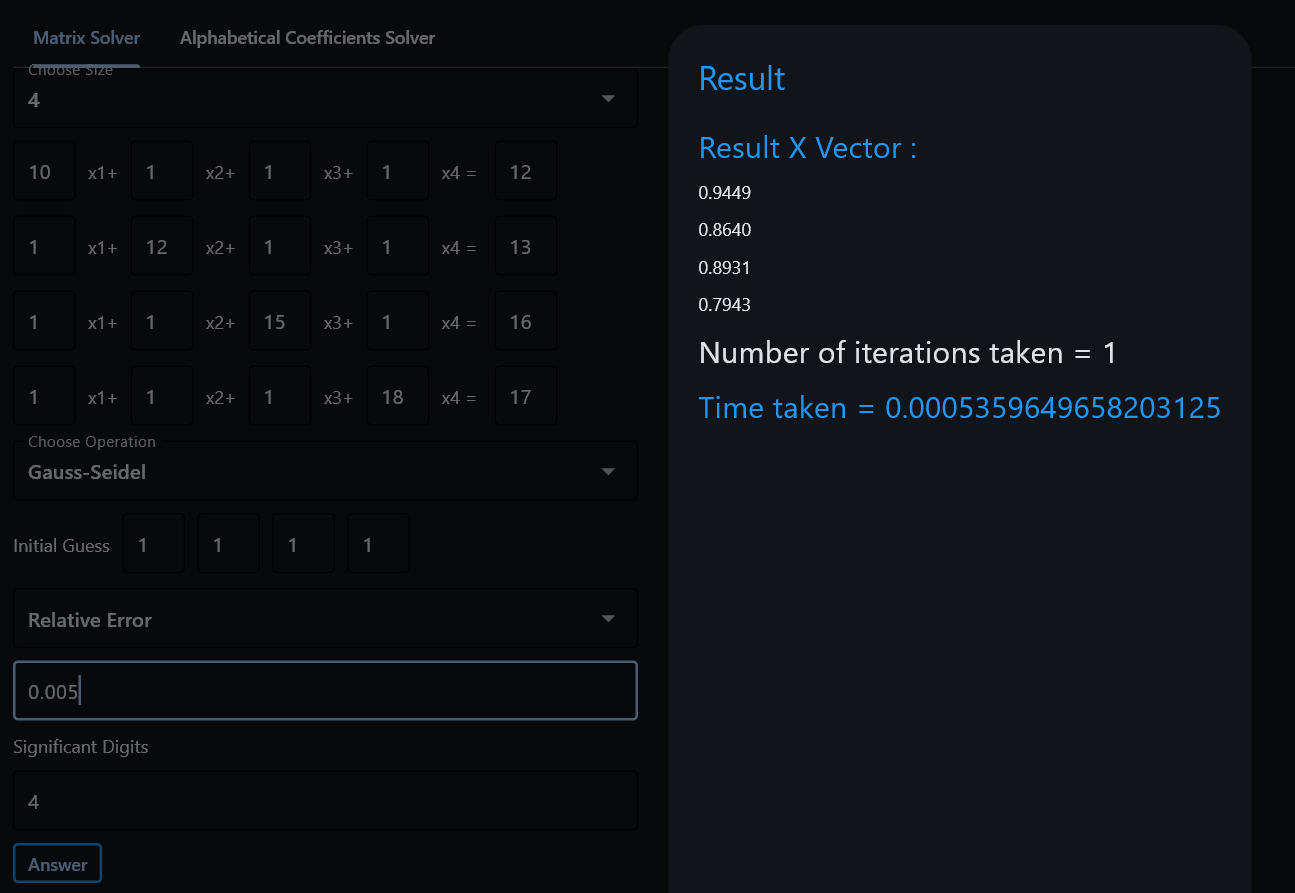
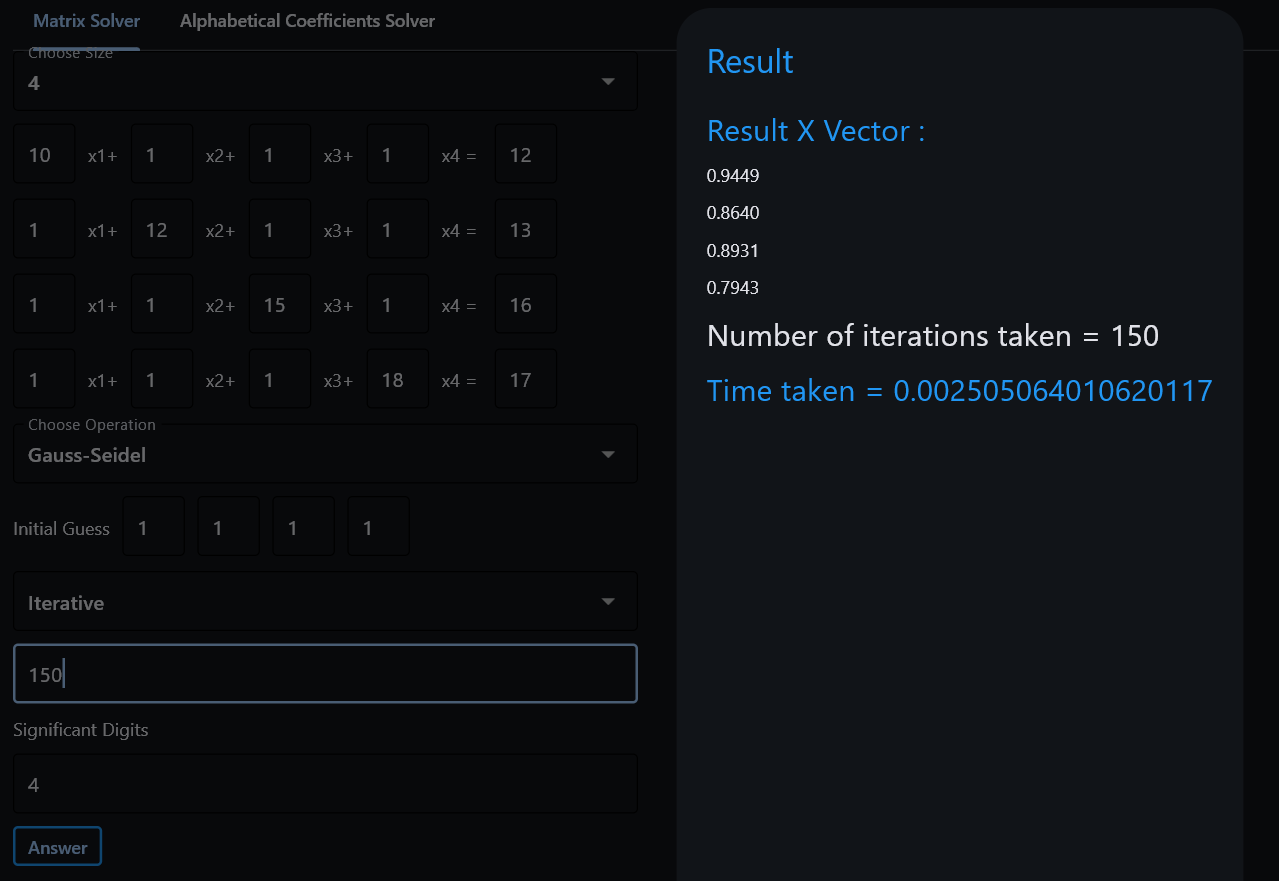




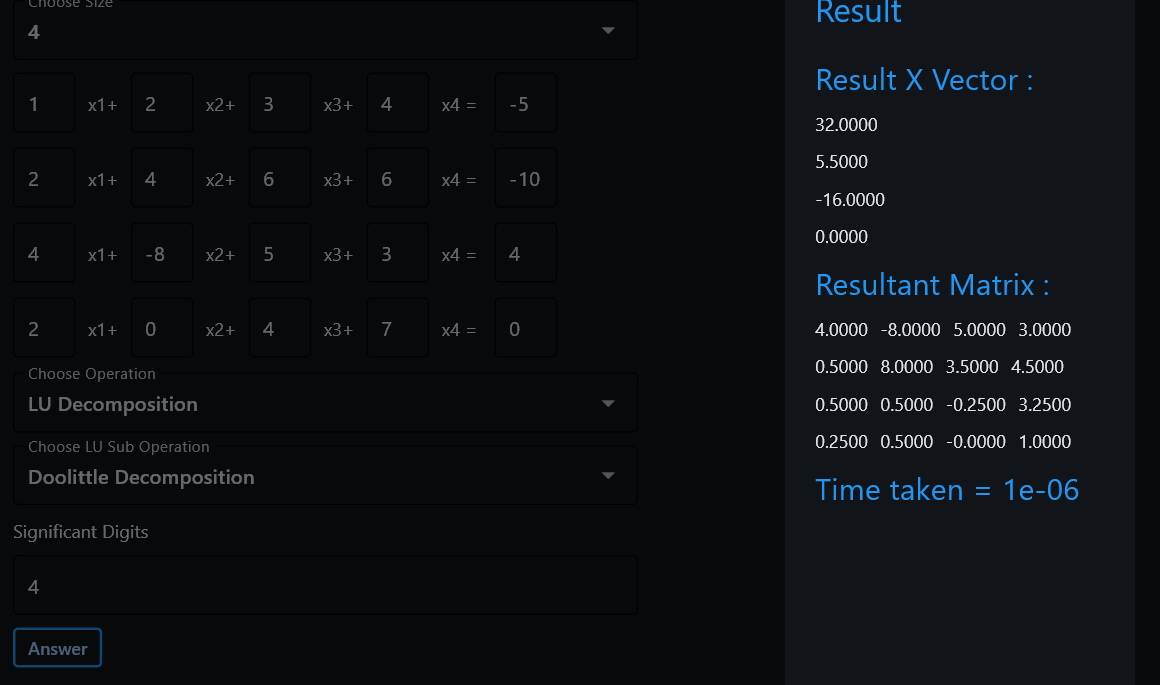
1. Converge

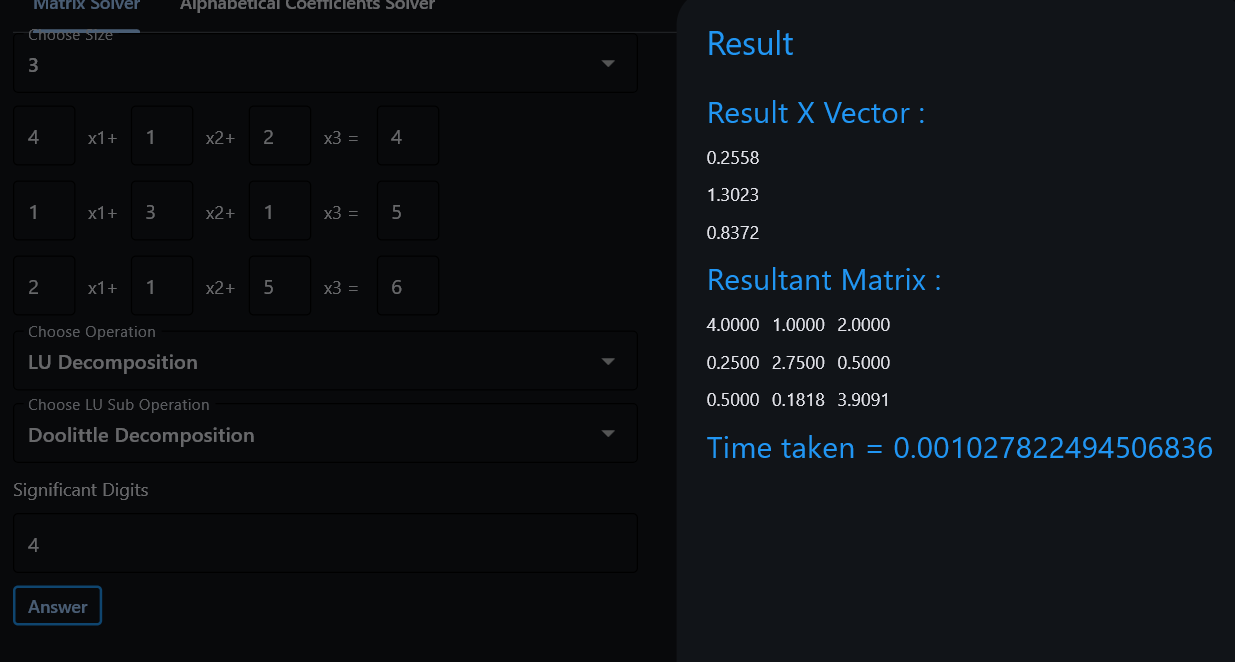


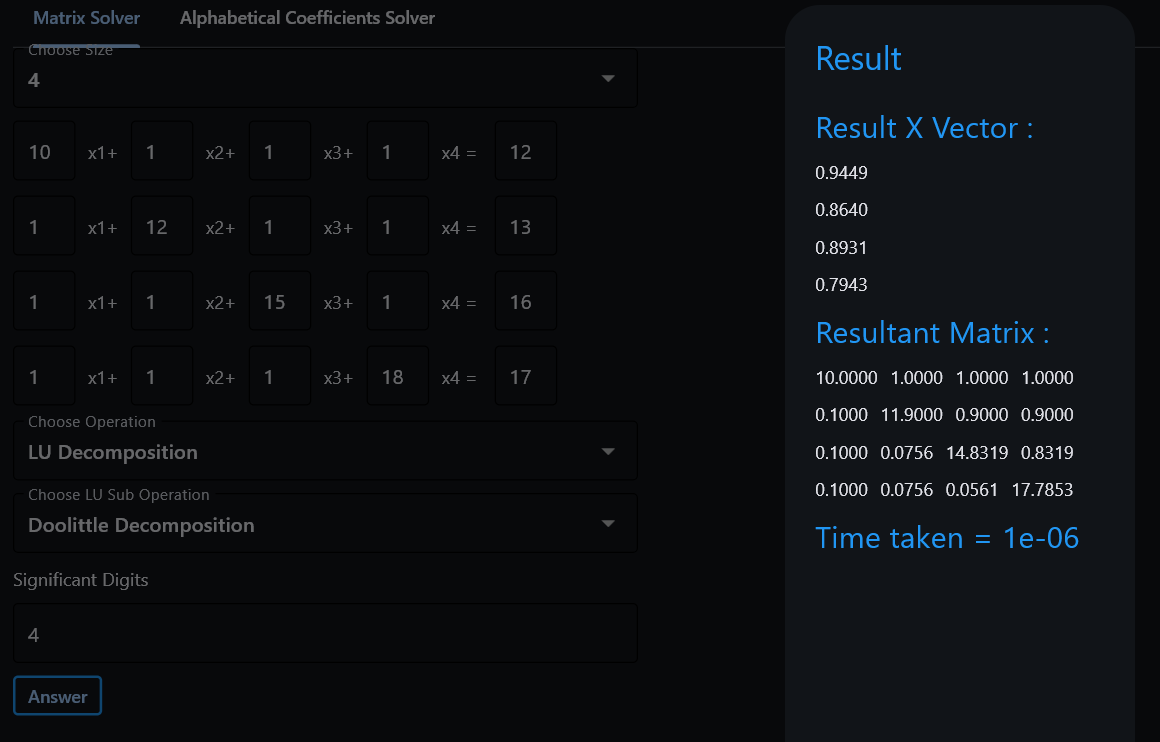
1. Converge



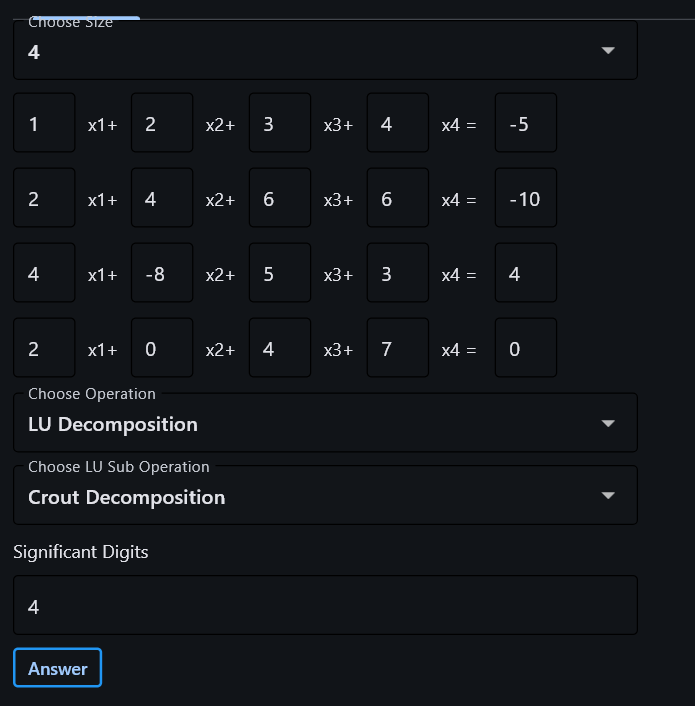
* LU Decomposition
  + Doolittle

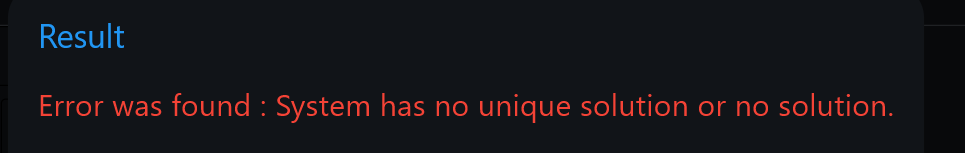


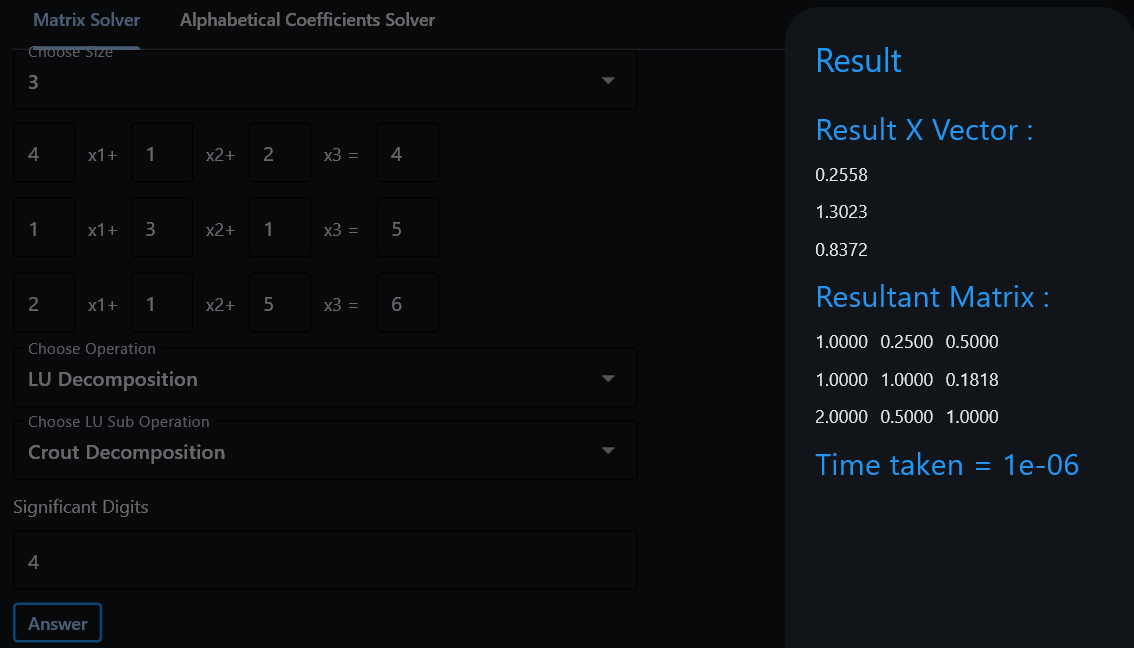


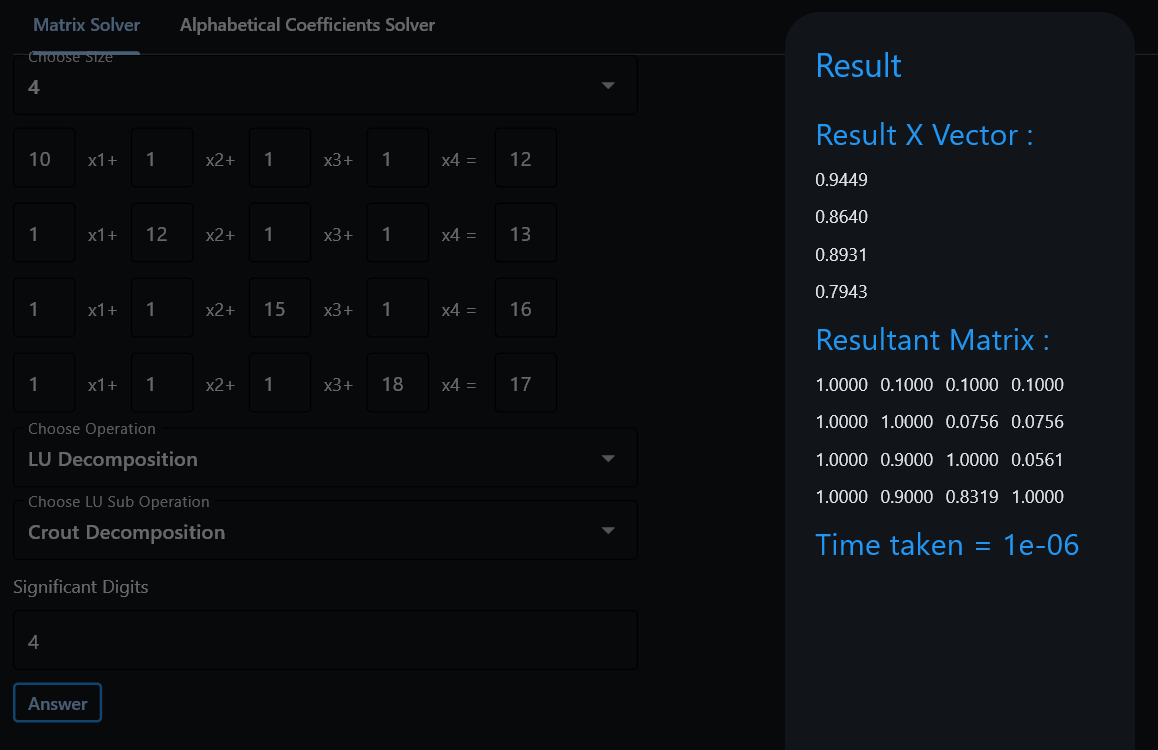


* + Crout

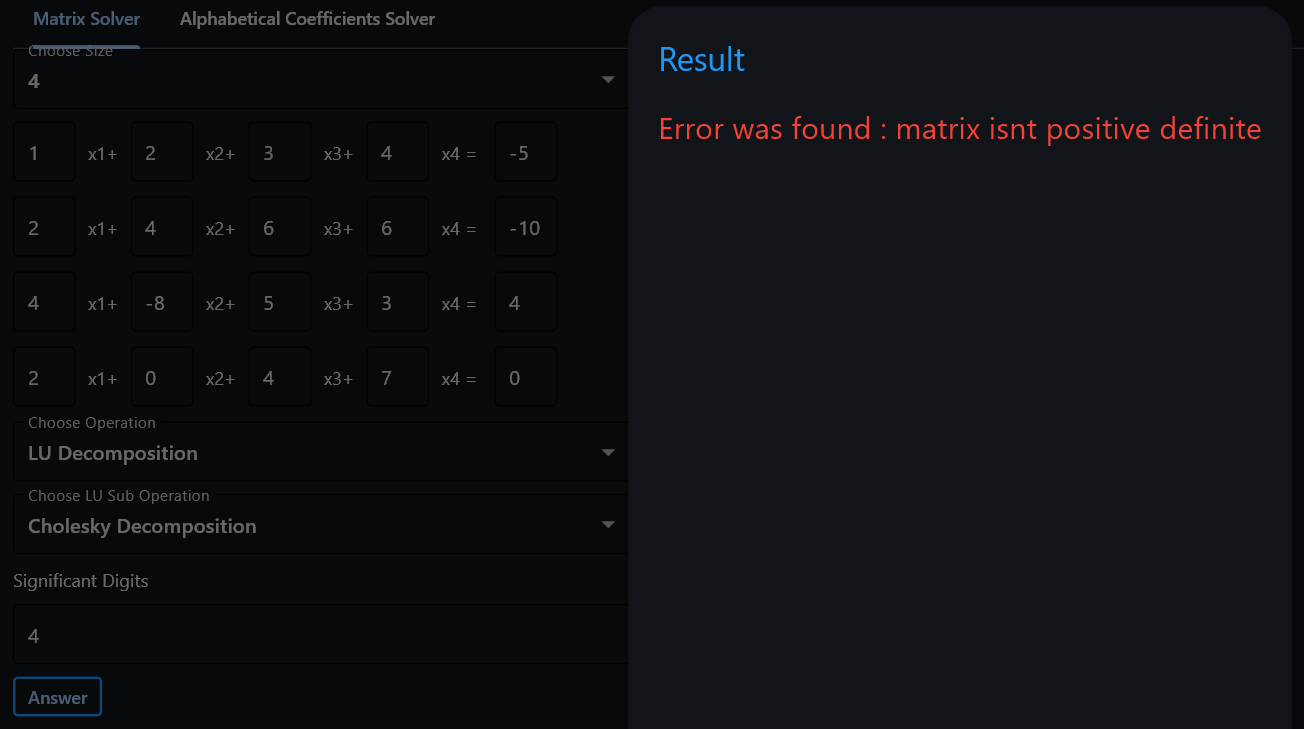


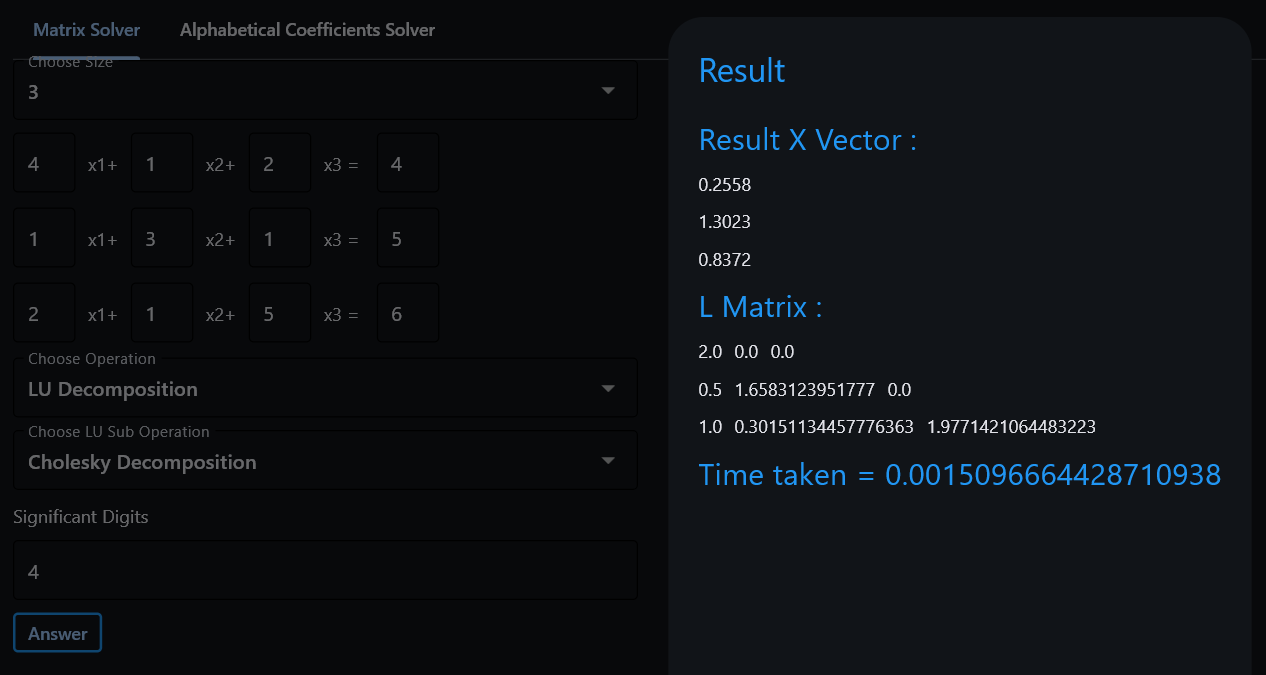


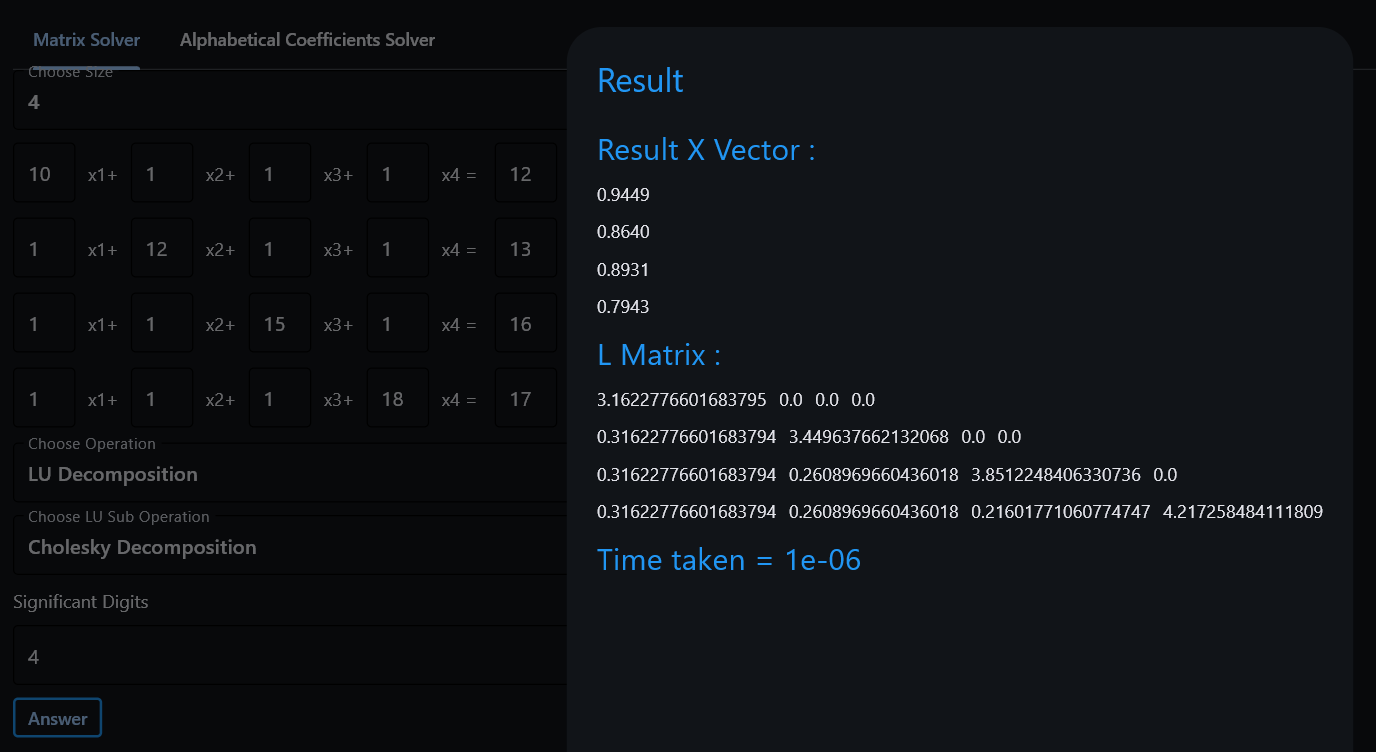




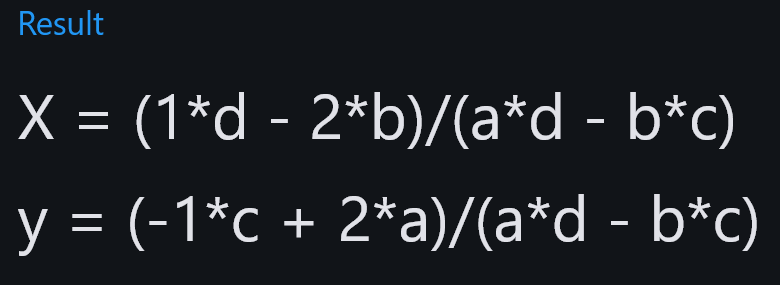
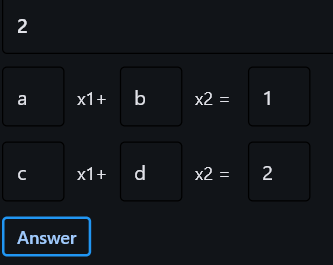
* + Cholesky

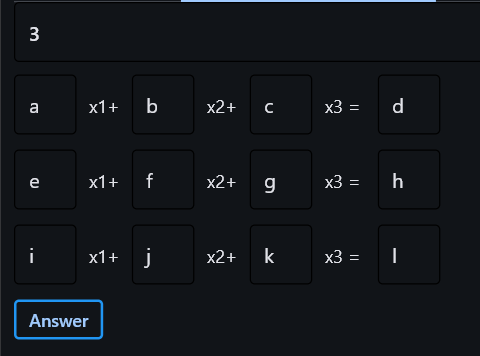


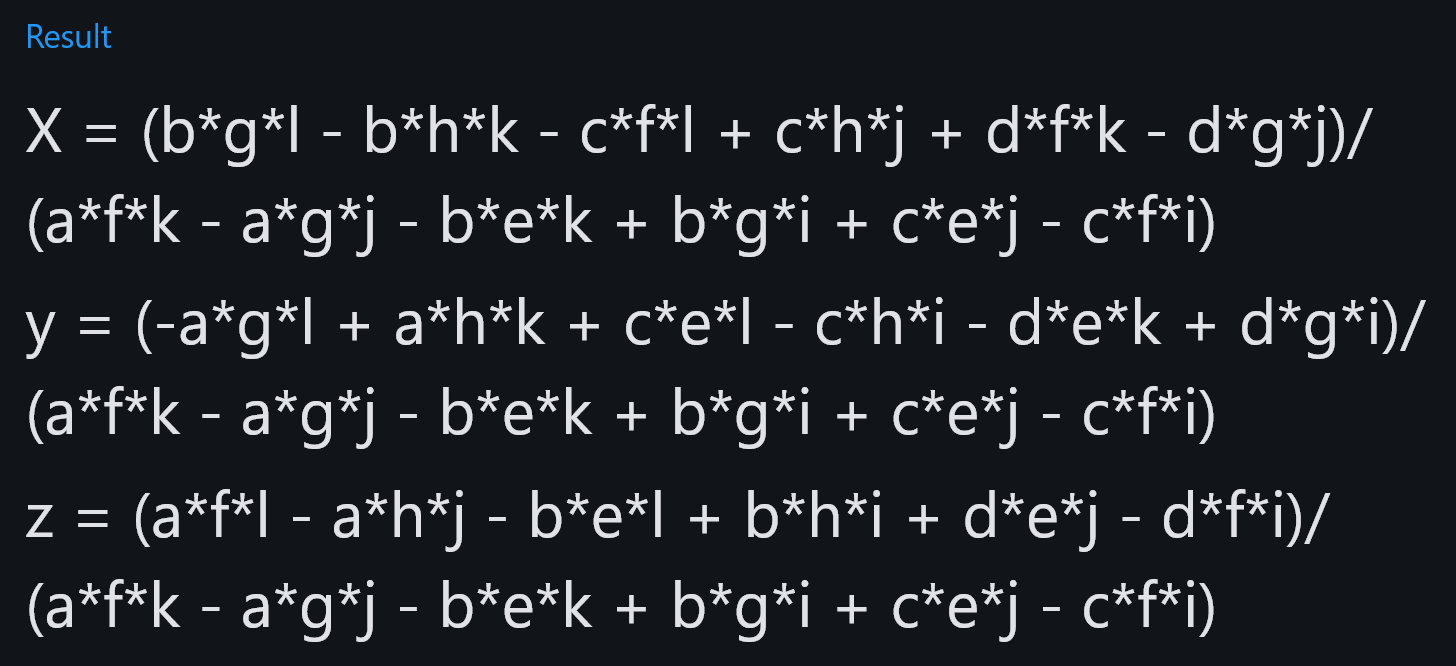




* Bonus: Coefficients can be letters







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

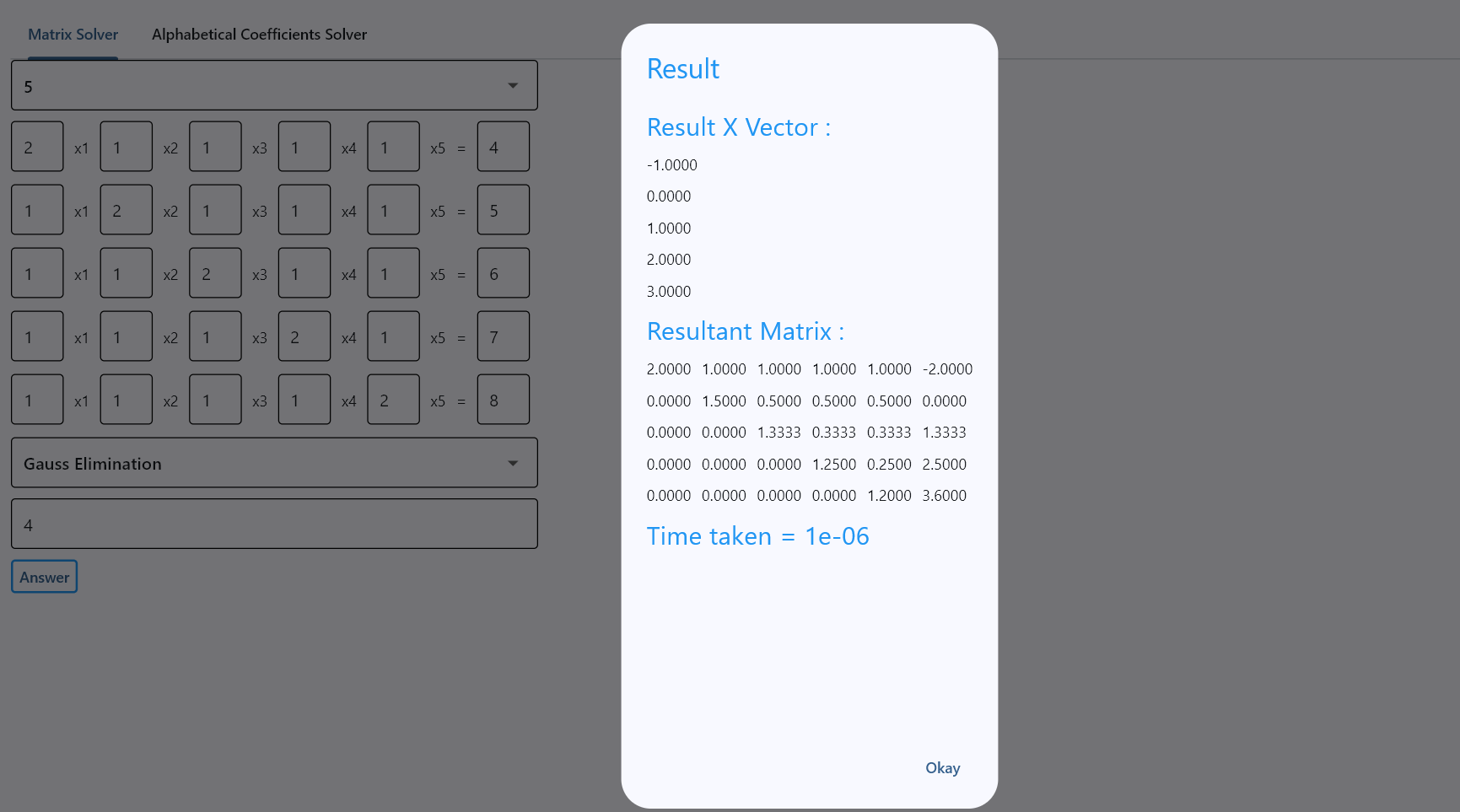
|  |  |  |  |
| --- | --- | --- | --- |
| *Method* | *Time Complexity* | *Convergence* | *Error* |
| Gauss Elimination | O(n³) | Direct method, no convergence issue | Numerical errors for ill-conditioned matrices |
| Gauss-Jordan | O(n³) | Direct method, no convergence issue | Numerical errors for ill-conditioned matrices |
| Jacobi | O(n²)  / Iteration | Converges for diagonally dominant or positive-definite matrices | Slower convergence than Gauss-Seidel |
| Gauss-Seidel | O(n²)  / Iteration | Converges for diagonally dominant or positive-definite matrices | Error decreases with each iteration |
| LU Decomposition | O(n³) | Converges for non-singular matrices | Sensitive to ill-conditioned matrices |

1. ***Data structures used:***

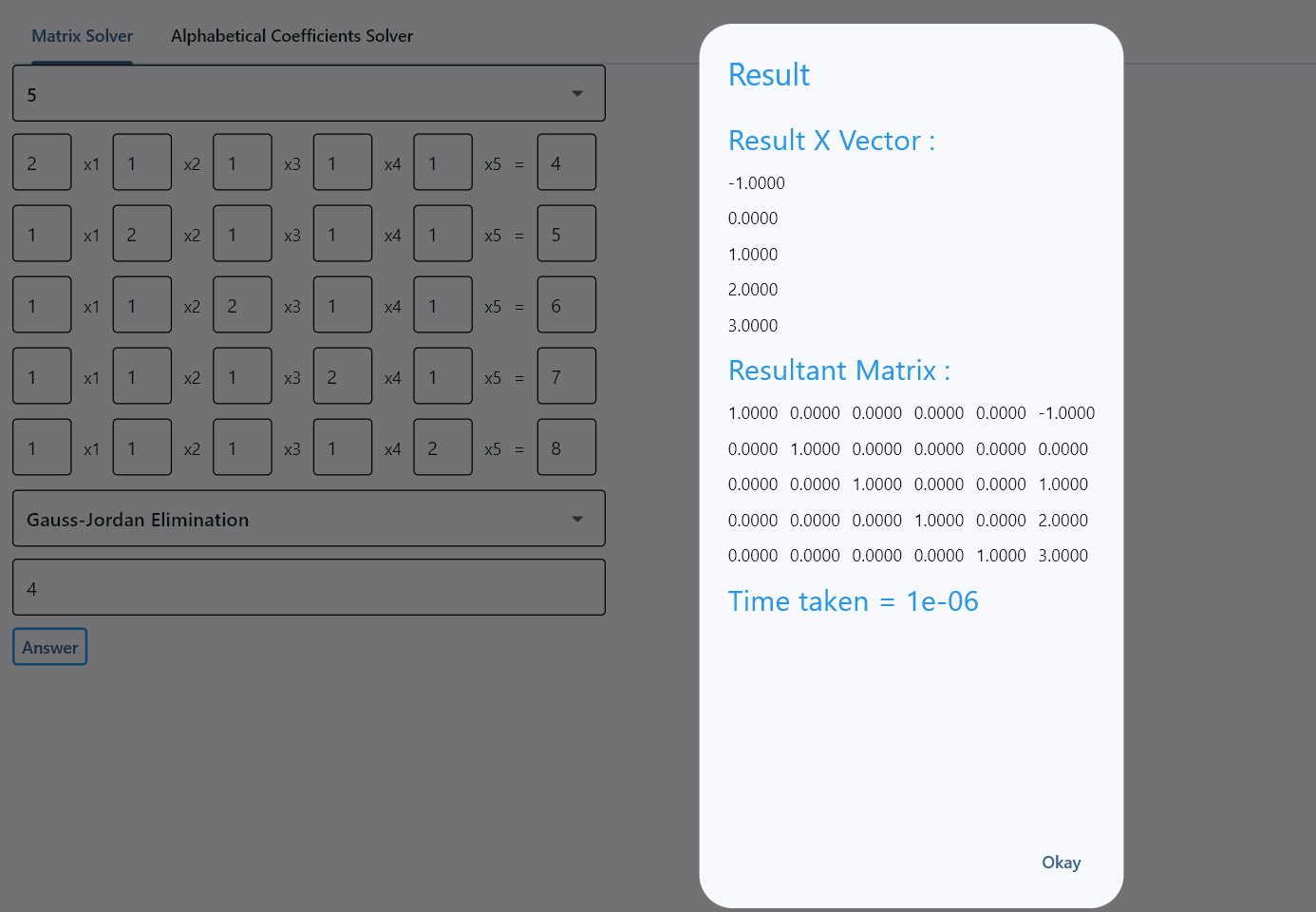
* 1D Lists (Arrays)
* 2D Lists (Matrices)
* Temporary (Scalar) variables
* Flags (Booleans)
* Indices

1. ***Test cases:***

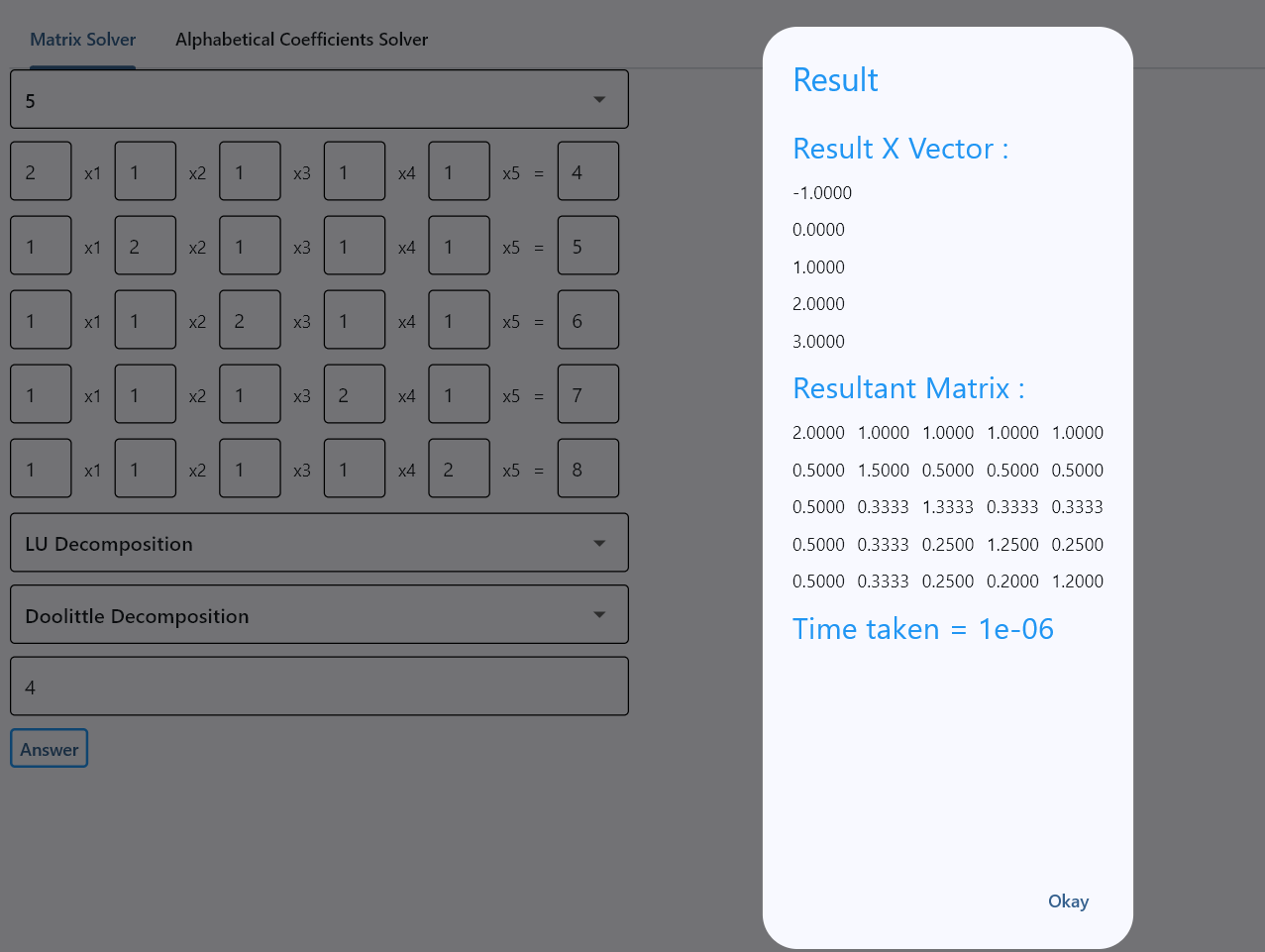
* 1st case
  + Gauss Elimination, precision = 4



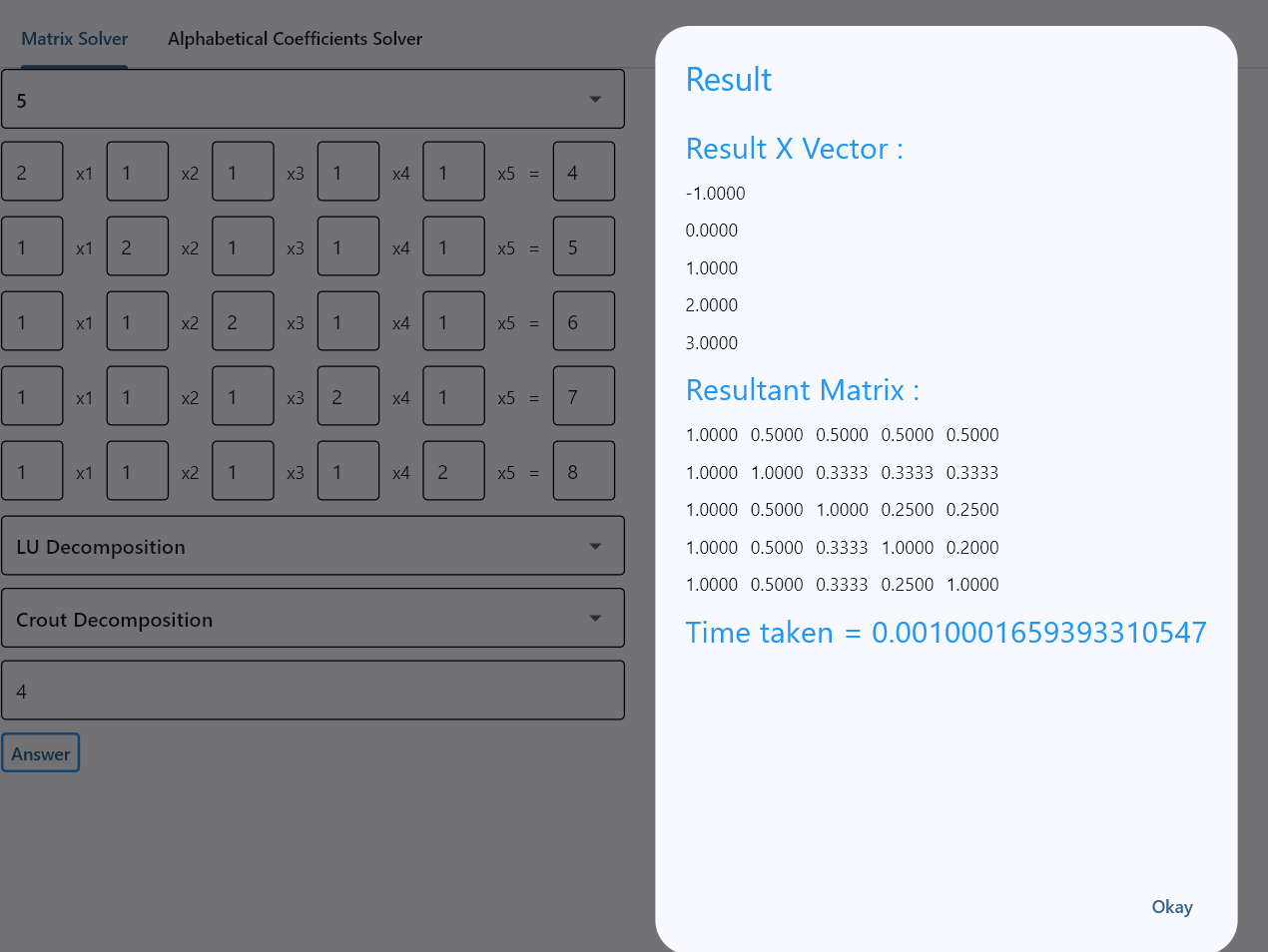
* + Gauss Jordan, precision = 4



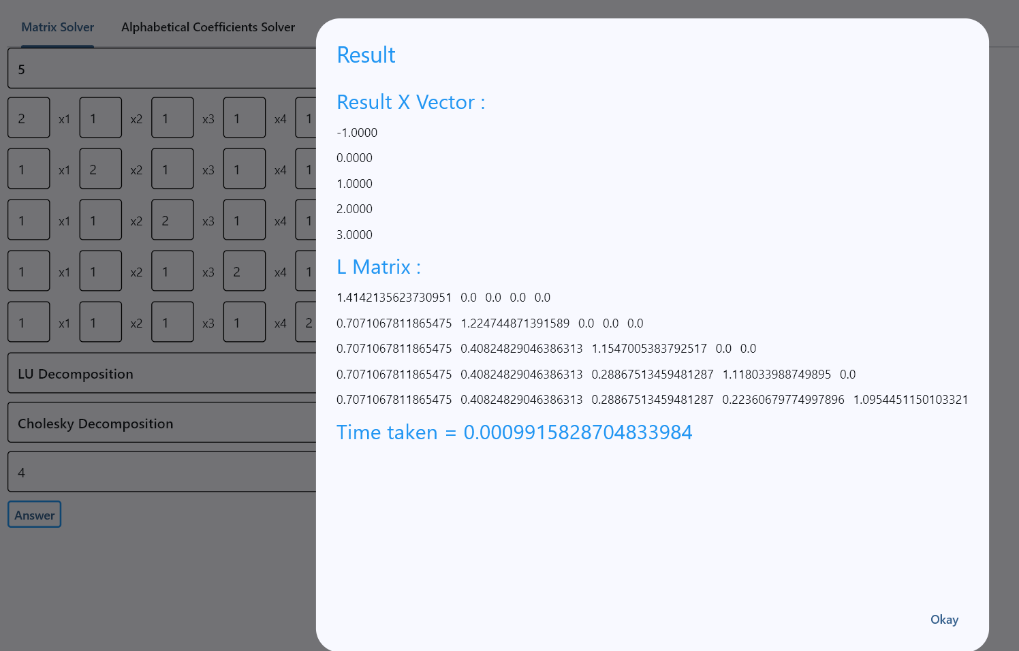
* + LU decomposition
    - Doolittle, precision = 4



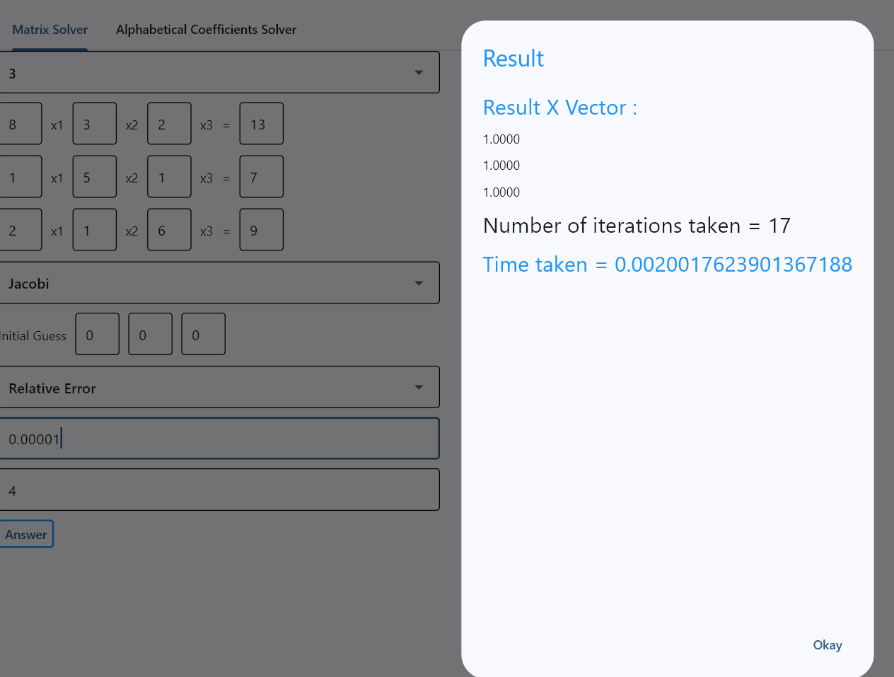
* + - Crout, precision = 4



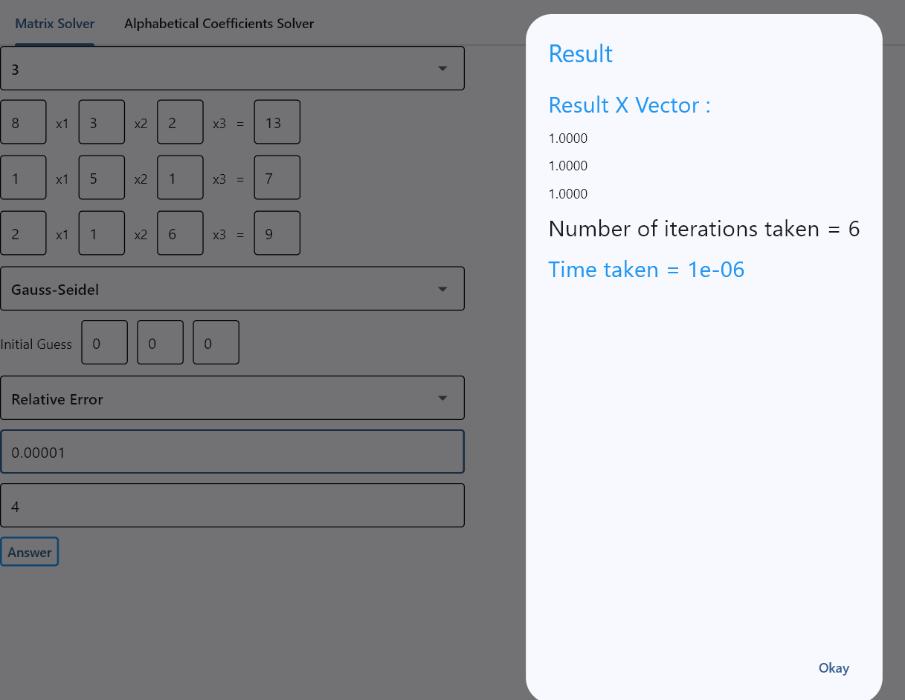
* + - Cholesky, precision = 4



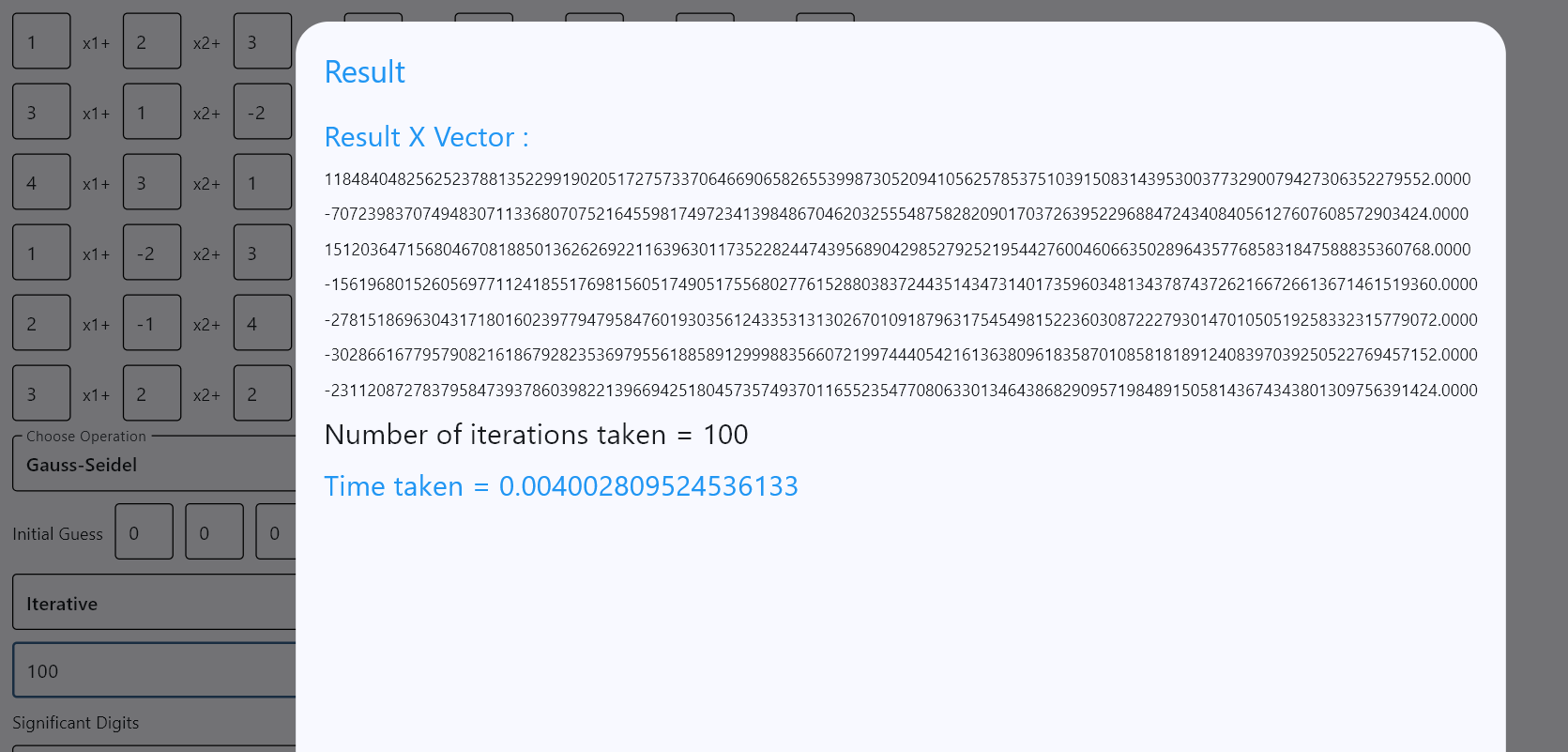
* 2nd case
  + Jacobi, absolute relative error = 0.00001



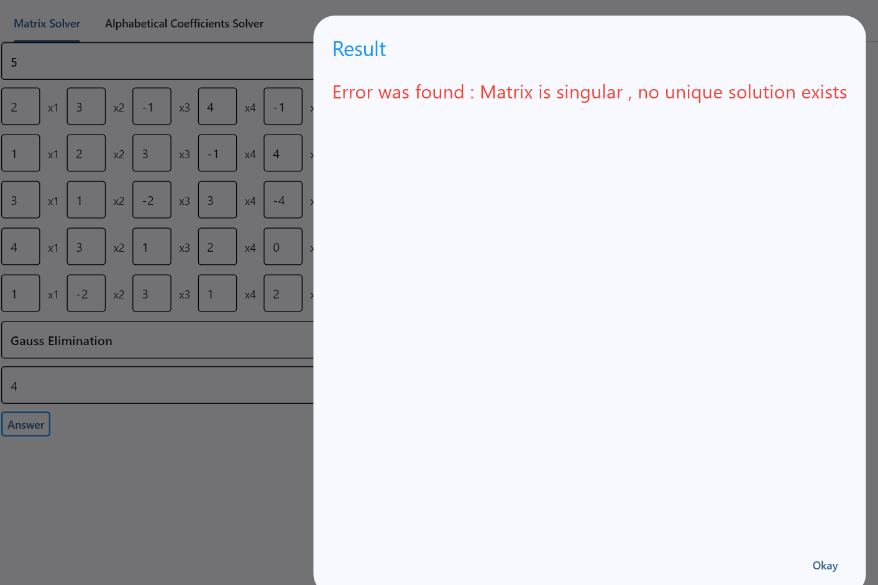
* + Gauss Seidel, absolute relative error = 0.00001



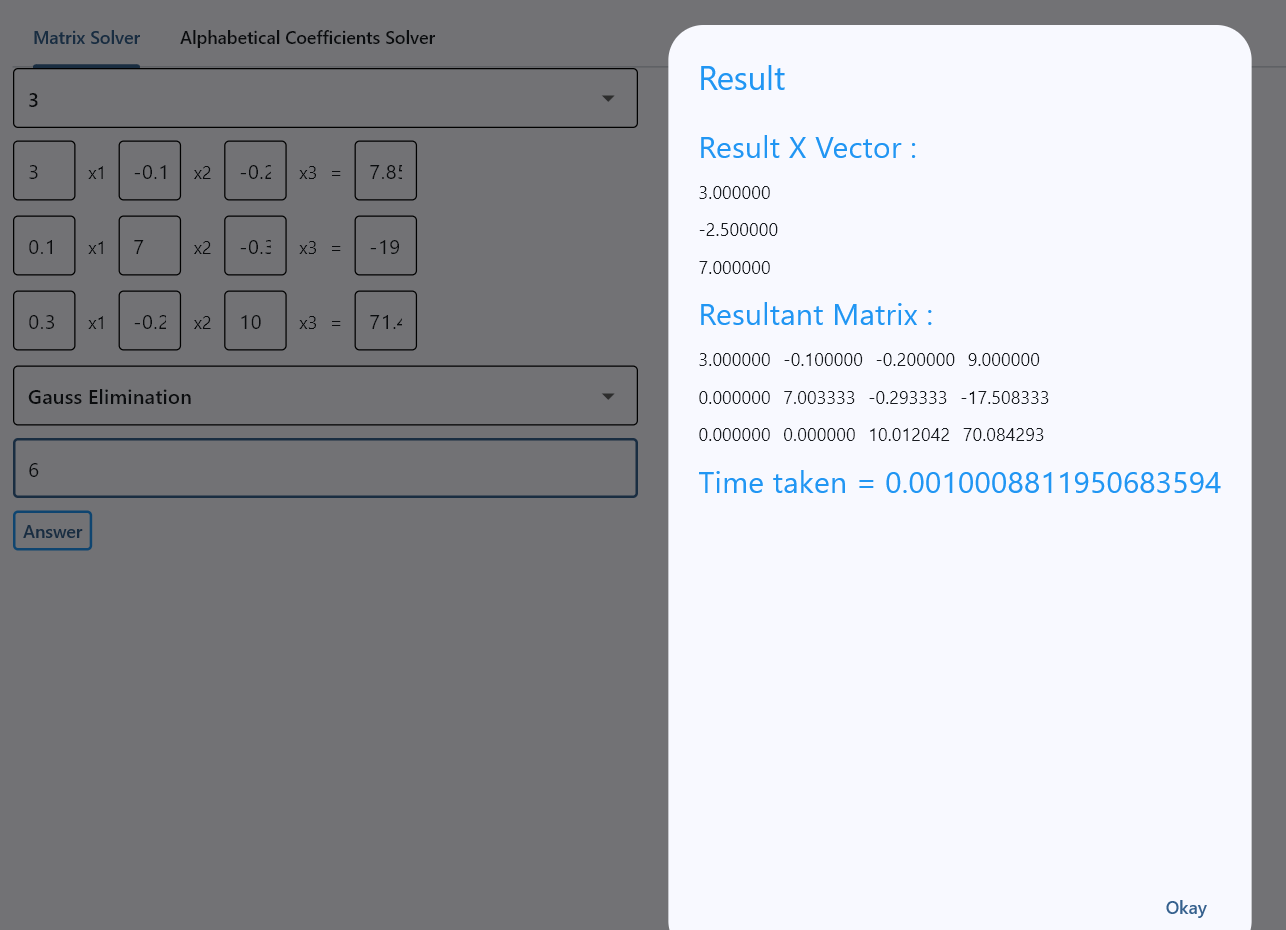
* 3rd case
  + Gauss Seidel, number of iterations = 100



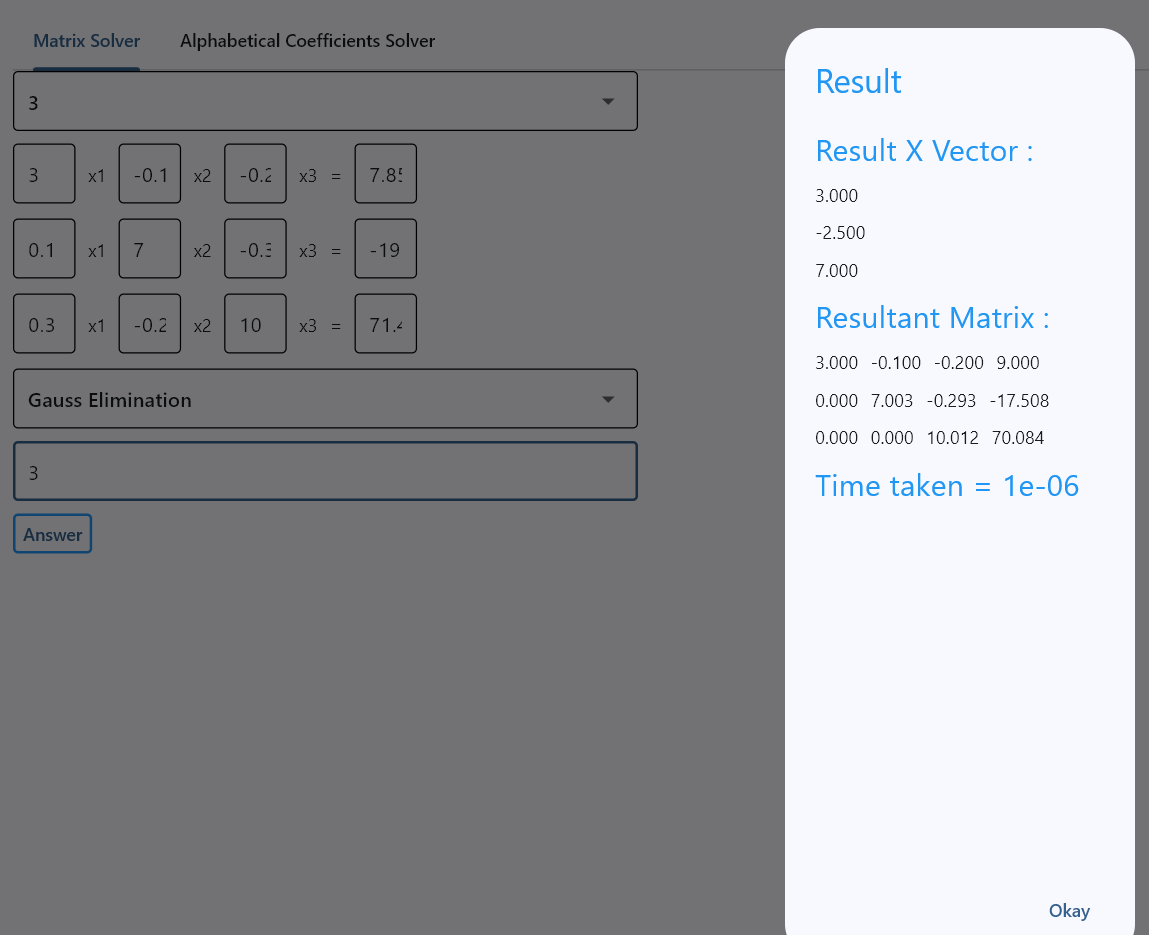
* 4th case
  + Gauss elimination



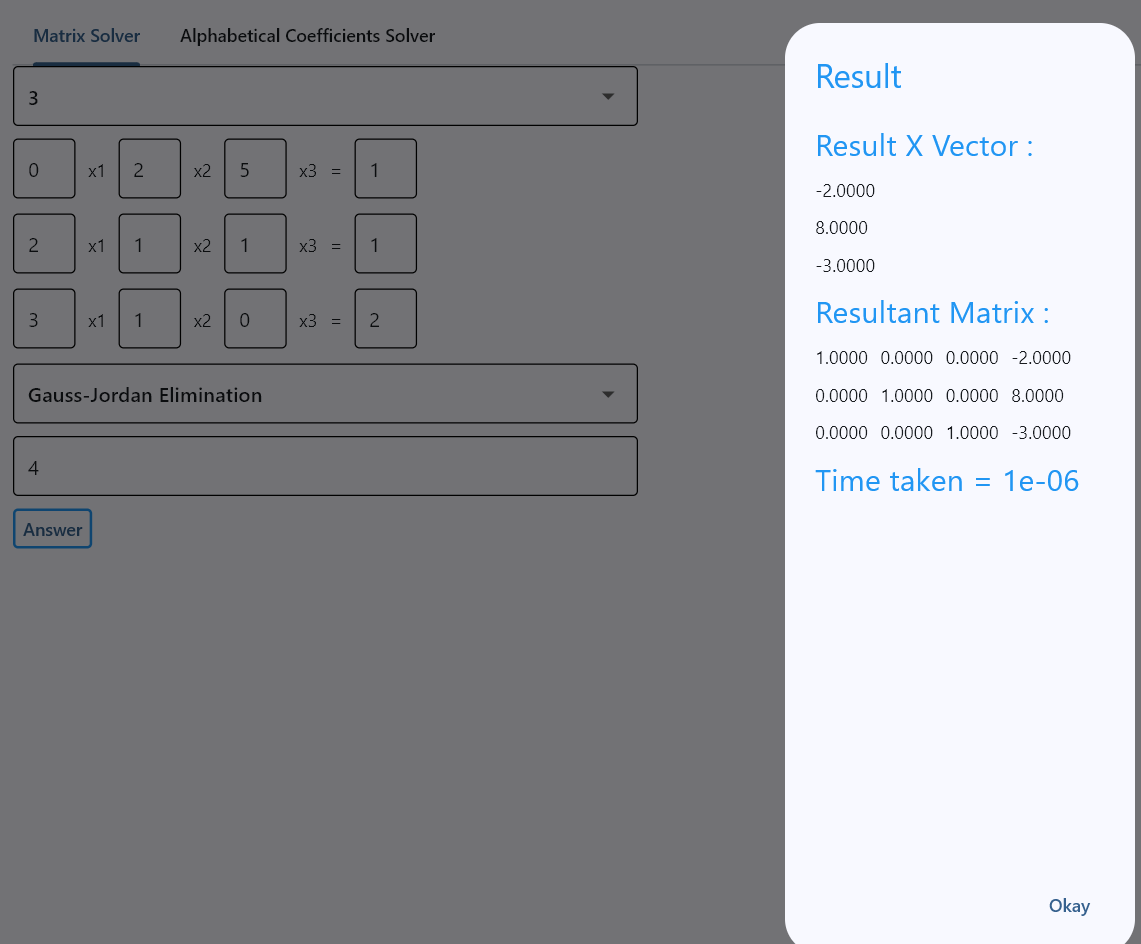
* 5th case
  + Gauss Elimination, precision = 6



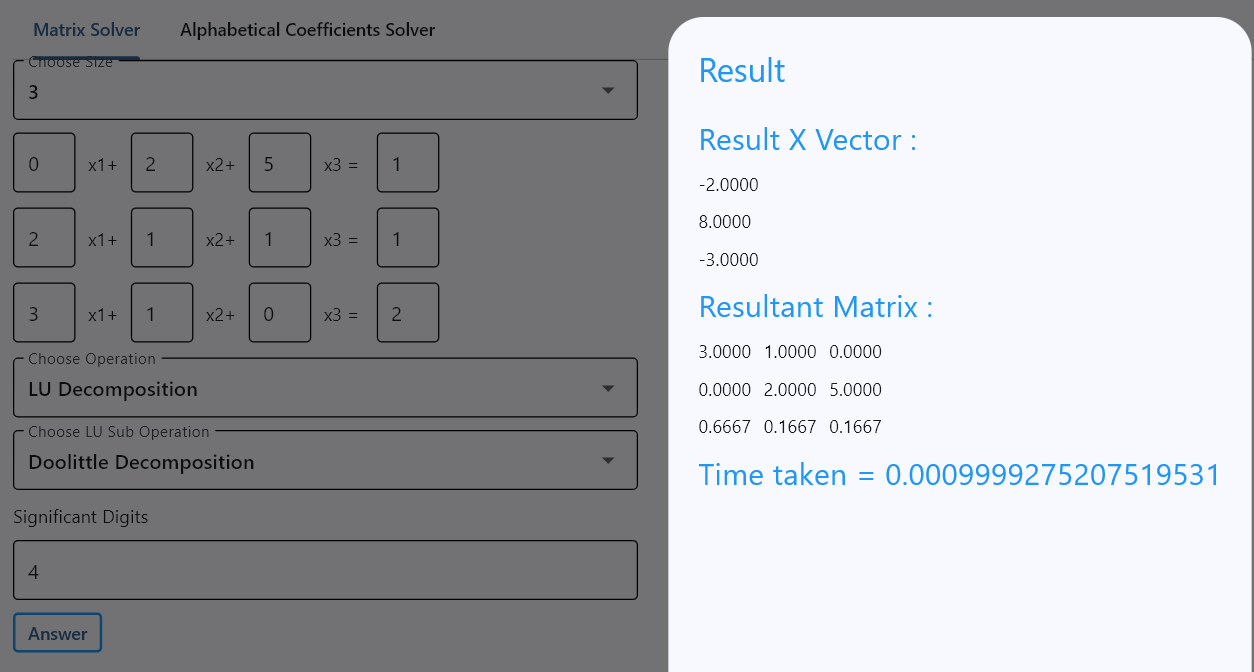
* + Gauss Elimination, precision = 3



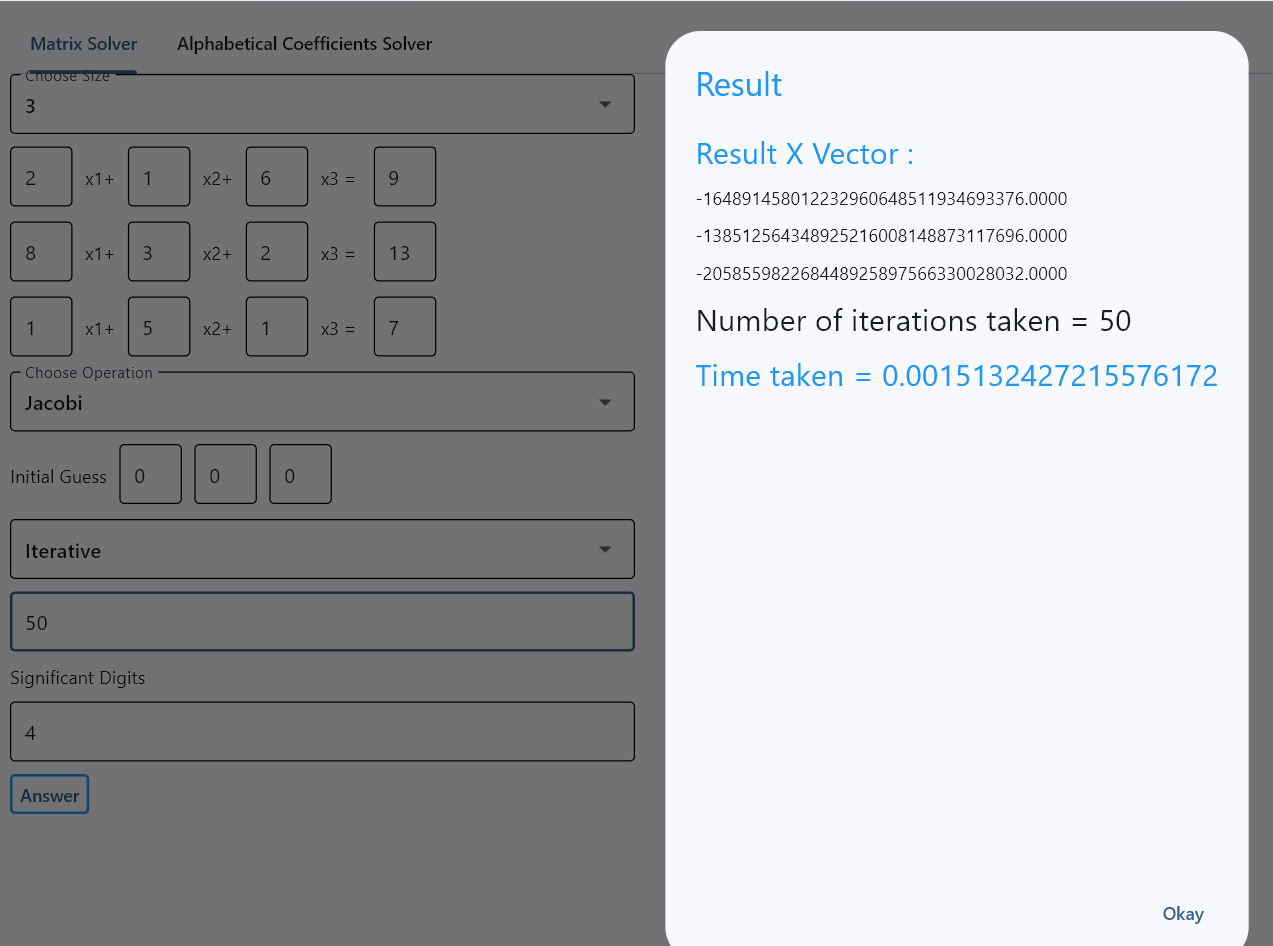
* 6th case
  + Gauss Jordan



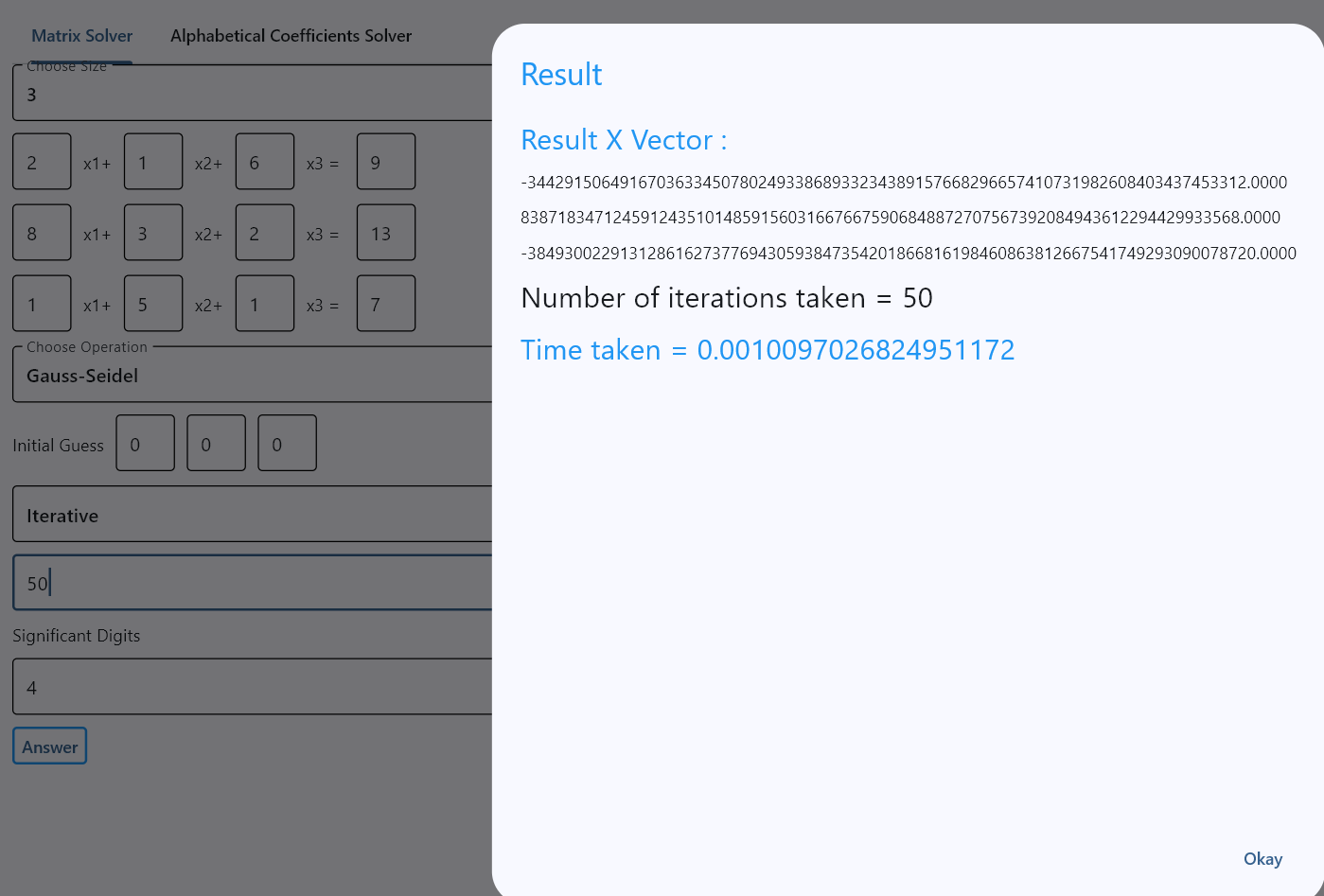
* + LU Decomposition
    - Doolittle



* 7th case
  + Jacobi, number of iterations = 50



* + Gauss Seidel, number of iterations = 50



1. ***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