INTRODUCTION:

In numerical analysis, solving a system of linear equations is one of the most essential computational tasks. Such systems appear in engineering, physics, computer science, economics, and applied mathematics. A linear system can be written in the form:

a11x1 + a12x2 + …+ a1nxn = b1
a21x1 + a22x2 + … + a2nxn = b2
…………………………………..
an1x1 + an2x2 + … + annxn = bn

Matrix representation: AX = B
Where,
A = coefficient matrix,
X = vector of unknowns,
B = constant vector.

To solve such systems, direct methods,specifically Gauss Elimination and Gauss–Jordan Elimination are among the most widely used.

GAUSS ELIMINATION METHOD:

Gauss Elimination is a direct method that converts a system of linear equations into an upper triangular system using forward elimination. After the matrix becomes upper triangular, back substitution is applied to determine the values of the unknowns.Gauss Elimination produces one of the following outcomes:

1. Unique Solution
2. No Solution (Inconsistent System)
3. Infinitely Many Solutions (Dependent System)

The process of finding the solution includes writing the augmented matrix [A|B].Applying forward elimination to make all elements below the diagonal equal to zero.After forward elimination, the matrix becomes upper triangular.Applying back substitution starting from the last equation to find the solutions.
Detecting:
• If a row becomes [0 0 0 | c] with c != 0 => No Solution
• If a row becomes [0 0 0 | 0] => Infinite Solutions

Algorithm:

Input:
    Number of equations n
    Augmented matrix A[n][n+1]

Steps:
1. Forward Elimination:
  For each column k = 1 to n−1:
    a. Pivoting:
      Finding row with maximum absolute value in column k

Swapping the pivot row with row k
b. Elimination:
For each row i = k+1 to n:
m = A[i][k] / A[k][k]
For each column j = k to n+1:
A[i][j] = A[i][j] − m * A[k][j]

2. Checking for No Solution:
If a row is [0 0 0 ... 0 | c] where c != 0 => No solution

3. Checking for Infinite Solutions:
If a row is [0 0 0 ... 0 | 0] and rank(A) = rank(A|b) < n => Infinite solutions

4. Back Substitution:
For i = n down to 1:
sum = Σ (A[i][j] * x[j]) for j = i+1 to n
x[i] = (A[i][n+1] − sum) / A[i][i]

Pseudocode:

```
START
Input n
Input augmented matrix A[n][n+1]

# FORWARD ELIMINATION
for k = 1 to n-1:
   # Pivoting
   pivot_row = k
   for i = k+1 to n:
      if abs(A[i][k]) > abs(A[pivot_row][k]):
         pivot_row = i
   swap rows A[k] and A[pivot_row]

   if A[k][k] == 0:
      continue

   # Eliminate rows below pivot
   for i = k+1 to n:
      m = A[i][k] / A[k][k]
      for j = k to n+1:
         A[i][j] = A[i][j] - m * A[k][j]

# CHECK SOLUTION TYPE
infinite_flag = false
no_solution_flag = false

for i = 1 to n:
   if all A[i][1..n] == 0 AND A[i][n+1] != 0:
      no_solution_flag = true
```

```
    if all A[i][1..n] == 0 AND A[i][n+1] == 0:
        infinite_flag = true

if no_solution_flag:
    PRINT "No Solution"
    STOP

if infinite_flag:
    PRINT "Infinite Solutions"
    STOP

# BACK SUBSTITUTION
Create vector x[n]

for i = n to 1 step -1:
    sum = 0
    for j = i+1 to n:
        sum = sum + A[i][j] * x[j]
    x[i] = (A[i][n+1] - sum) / A[i][i]

PRINT "Unique Solution: ", x
STOP
```

Worked Example 1 (unique solution case):

Solve:
$3x + 2y - z = 1$
$2x - 2y + 4z = -2$
$-x + \frac{1}{2}y - z = 0$

Augmented matrix:
[3 2 –1 | 1]
[2 –2 4 | –2]
[–1 ½ –1 | 0]

After forward elimination:
[3 2 –1 | 1]
[0 –8/3 10/3 | –4/3]
[0 0 –4/3 | –2/3]

Back substitution:
$z = 0.5$
$y = -1$
$x = 1$

Unique Solution: $(x, y, z) = (1, -1, 0.5)$

Worked Example 2 (infinite solution case):
Solve:
x + y = 2
2x + 2y = 5

Augmented matrix:
[1 1 | 2]
[2 2 | 5]
After elimination:
[1 1 | 2]
[0 0 | 1]
Row: [0 0 | 1] => contradiction => NO SOLUTION

Worked Example 3 (infinite solutions)
Solve :
x + y + z = 3
2x + 2y + 2z = 6
3x + 3y + 3z = 9

After elimination:
[1 1 1 | 3]
[0 0 0 | 0]
[0 0 0 | 0]

Two zero rows => Infinitely many solutions

General solution: infinitely many.

ADVANTAGES:

Always gives exact solution for non-singular systems.Simple and systematic procedure.Works for any size of linear system.Efficient for computer implementation.

LIMITATIONS:

Back substitution needed.Fails if pivot element becomes zero.Can be unstable without pivoting.Large computations for very big matrices.

GAUSS-JORDAN METHOD:

Gauss-Jordan Elimination is an extended form of Gauss Elimination. Instead of producing an upper triangular matrix, it reduces the augmented matrix directly to reduced row echelon form (RREF).This eliminates the need for back substitution.

The method for solving system of linear equations include writing the augmented matrix [A|B].Converting the matrix to upper triangular form (like Gauss

elimination).Continuing eliminating values above the diagonal to create a diagonal matrix.Converting diagonal elements to 1 by dividing the row.
The matrix becomes:

[I | X]

Where I = identity matrix, X = solutions.

Algorithm:

Input:
  Number of equations n
  Augmented matrix A[n][n+1]

Steps:
1. For each pivot column k = 1 to n:
    a. Pivoting:
        Find row with maximum absolute value in column k
        Swap it with row k
    b. Normalize Pivot:
        pivot = A[k][k]
        Divide entire row k by pivot to make pivot = 1
    c. Eliminate All Other Rows:
        For each row i != k:
            m = A[i][k]
            For each column j = k to n+1:
                A[i][j] = A[i][j] − m * A[k][j]

2. Detect Solution Type:
    No solution if row is [0 0 0 | c], c != 0
    Infinite solutions if row is [0 0 0 | 0] and rank < n

3. Extract solution:
    If unique, solution x[i] = A[i][n+1]

Pseudocode:

```
Algorithm GaussJordan(AugmentedMatrix M):
   Input : M is an m x (n+1) augmented matrix [A | b]
   Output: RREF of M and type of solution (unique / none / infinite)

   # Step 1 - Initialization
   pivot_row <= 0
   pivot_col <= 0

   # Step 2 - Looping over all columns except the last one (which is b)
   while pivot_row < m AND pivot_col < n:

      # Step 2.1 - Finding a row with a non-zero entry in pivot_col
      row_with_pivot <= -1
      for r from pivot_row to m-1:
```

```
        if M[r][pivot_col] !=0:
            row_with_pivot <= r
            break

    # If no pivot found in this column => move to next column
    if row_with_pivot = -1:
        pivot_col <= pivot_col + 1
        continue

    # Step 2.2 - Swap pivot row into correct position
    SwapRows(M, pivot_row, row_with_pivot)

    # Step 2.3 - Normalize pivot row (make pivot = 1)
    pivot_value <= M[pivot_row][pivot_col]
    for c from pivot_col to n:
        M[pivot_row][c] <= M[pivot_row][c] / pivot_value

    # Step 2.4 - Eliminate all other rows
    for r from 0 to m-1:
        if r != pivot_row:
            factor <= M[r][pivot_col]
            for c from pivot_col to n:
                M[r][c] <= M[r][c] - factor * M[pivot_row][c]

    # Move to next pivot location
    pivot_row <= pivot_row + 1
    pivot_col <= pivot_col + 1

# Step 3 - Analyze RREF for solutions
contradiction <= false
for r from 0 to m-1:
    if (all M[r][0..n-1] = 0) AND (M[r][n] != 0):
        contradiction <= true

if contradiction:
    return ("No solution", M)

# Count pivot rows
rank <= number of rows that contain a pivot (leading 1)

if rank = n:
    return ("Unique solution", M)
else:
    return ("Infinite solutions", M)
```

Worked Example 1 (unique solution)
Solve:
x + y + z = 6
2x + 3y + 7z = 20
x – y + z = 2

Final RREF:
[1 0 0 | 1]
[0 1 0 | 2]
[0 0 1 | 3]
Solution: x = 1, y = 2, z = 3

Worked Example 2 (no solution)
x + y = 2
2x + 2y = 5
RREF becomes:
[1   1 | 2]
[0   0 | 1]
Contradiction => No Solution

Worked Example 2 (infinite solutions)
x + y + z = 3
2x + 2y + 2z = 6
3x + 3y + 3z = 9
RREF becomes:
[1   1   1 | 3]
[0   0   0 | 0]
[0   0   0 | 0]
Infinite solutions => Parametric form.

ADVANTAGES:

No back substitution needed.Produces solution directly.Gives reduced row echelon form (RREF).Helpful for matrix inversion.

LIMITATIONS:

More computation than Gauss elimination.Not suitable for large matrices.Prone to rounding errors.Sensitive to pivot choice.