Assignment Number

Problem Statement

Program in C to solve a system of linear equations using Gauss Elimination method.

Theory

In linear algebra, Gaussian elimination (also known as row reduction) is an algorithm for solving systems of linear equations. It is usually understood as a sequence of operations performed on the corresponding matrix of coefficients. This method can also be used to find the rank of a matrix, to calculate the determinant of a matrix, and to calculate the inverse of an invertible square matrix. The method is named after Carl Friedrich Gauss (1777–1855), although it was known to Chinese mathematicians as early as 179 CE.
To perform row reduction on a matrix, one uses a sequence of elementary row operations to modify the matrix until the lower left-hand corner of the matrix is filled with zeros, as much as possible. There are three types of elementary row operations:

1. Swapping two rows.
2. Multiplying a row by a non-zero number.
3. Adding a multiple of one row to another row.

Using these operations, a matrix can always be transformed into an upper triangular matrix, and in fact one that is in row echelon form. Once all of the leading coefficients (the left-most non-zero entry in each row) are 1, and every column containing a leading coefficient has zeros elsewhere, the matrix is said to be in reduced row echelon form. This final form is unique; in other words, it is independent of the sequence of row operations used.
For example, in the following sequence of row operations (where multiple

elementary operations might be done at each step), the third and fourth matrices are the ones in row echelon form, and the final matrix is the unique reduced row echelon form.

$$\left[\begin{array}{ccc|c} 1 & 3 & 1 & 9 \\ 1 & 1 & -1 & 1 \\ 3 & 11 & 5 & 35 \end{array}\right] \to \left[\begin{array}{ccc|c} 1 & 3 & 1 & 9 \\ 0 & -2 & -2 & -8 \\ 0 & 2 & 2 & 8 \end{array}\right] \to \left[\begin{array}{ccc|c} 1 & 3 & 1 & 9 \\ 0 & -2 & -2 & -8 \\ 0 & 0 & 0 & 0 \end{array}\right] \to \left[\begin{array}{ccc|c} 1 & 0 & -2 & -3 \\ 0 & 1 & 1 & 4 \\ 0 & 0 & 0 & 0 \end{array}\right]$$

Algorithm

Input : The coefficient matrix, say **a**

Output : The solution of the system of linear equations

Steps :

```
    Step 1 : For k=1 to n
    Step 2 : j = 1
    Step 3 : Input a[k][j]
    Step 4 : j = j + 1
    Step 5 : If(j < n + 1)
       Then
           Step 6 : Goto step 3
    Step 7 : k = k + 1
    Step 8 : If(k < n)
       Then
           Step 9 : Goto step 2
    Step 10 : k = 1
    Step 11 : i = k + 1
    Step 12 : If(i > j)
       Then
           Step 13 : c = a[i][k] / a[k][k]
           Step 14 : j = k + 1
           Step 15 : a[i][j] = a[i][j] – c * a[k][j]
```

```
        Step 16 : j = j + 1
        Step 17 : If(j < n + 1)
           Then
                Step 18 : Goto step 15
Step 19 : If(i < n)
   Then
        Step 20 : Goto step 12
Step 21 : If( k < n – 1)
   Then
        Step 22 : Goto step 11
Step 23 : x[n] = a[n][n+1]/a[n][n]
Step 24 : k = n - 1
Step 25 : sum = 0
Step 26 : j = k + 1
Step 27 : sum = sum + a[k][j] * x[j]
Step 28 : j = j + 1
Step 29 : If(j < n)
           Then
        Step 30 : Goto step 27
Step 31 : x[k] = 1/a[k][k] * (a[k][n+1] – sum)
Step 32 : k = k + 1
Step 33 : If( k > 1)
           Then
        Step 34 : Goto 25
Step 35 : Display the result x[1..n]
Step 36 : End
```

Source Code

```
#include <stdio.h>
int main() {
    int i, j, k, n;
    float A[20][20], c, x[10], sum = 0.0;
    printf("\nEnter the order of matrix: ");
    scanf("%d", &n);
    printf("\nEnter the elements of augmented matrix row-wise:\n\n");
    for (i = 1; i <= n; i++) {
        for (j = 1; j <= (n + 1); j++) {
            printf("A[%d][%d] : ", i, j);
            scanf("%f", &A[i][j]);
        }
    }
    for (j = 1; j <= n;
         j++) /* loop for the generation of upper triangular matrix*/
    {
        for (i = 1; i <= n; i++) {
            if (i > j) {
                c = A[i][j] / A[j][j];
                for (k = 1; k <= n + 1; k++) {
                    A[i][k] = A[i][k] - c * A[j][k];
                }
            }
        }
    }
    x[n] = A[n][n + 1] / A[n][n];
    /* this loop is for backward substitution*/
    for (i = n - 1; i >= 1; i--) {
        sum = 0;
        for (j = i + 1; j <= n; j++) {
            sum = sum + A[i][j] * x[j];
        }
        x[i] = (A[i][n + 1] - sum) / A[i][i];
    }
    printf("\nThe solution is: \n");
    for (i = 1; i <= n; i++) {
        printf("\nx%d=%f\t", i,
               x[i]); /* x1, x2, x3 are the required solutions*/
    }
    return (0);
}
```

Input and Output

Set 1:

```
Enter the order of matrix: 3

Enter the elements of augmented matrix row-wise:

A[1][1] : 3
A[1][2] : 5
A[1][3] : 6
A[1][4] : 3
A[2][1] : 8
A[2][2] : 2
A[2][3] : 3
A[2][4] : 5
A[3][1] : 7
A[3][2] : 3
A[3][3] : 4
A[3][4] : 6

The solution is:

x1=2.499964
x2=25.499563
```

```
x3=-21.999619
```

Set 2 :

```
Enter the order of matrix: 3

Enter the elements of augmented matrix row-wise:

A[1][1] : 1
A[1][2] : 2
A[1][3] : 3
A[1][4] : 43
A[2][1] : 21
A[2][2] : 23
A[2][3] : 82
A[2][4] : 37
A[3][1] : 27
A[3][2] : 12
A[3][3] : 1
A[3][4] : 2

The solution is:

x1=-17.202339
x2=39.387836
x3=-6.191112
```

Discussion

1. One more way of solving this would be computing the inverse and multiplying with that. The inverse can be computed in (at least) two ways: with Gaussian elimination, or Kramer’s rule. The latter is very expensive and probably unstable. But since we already LU factorization from Gaussian elimination, we might as well use that, rather than first computing the inverse.
2. More interesting approach towards the solution is that one can use an iterative method for solving the linear system. In that case Gauss Elimination has the pro of being guaranteed to work (up to roundoff), while iterative methods can fail, or use an unpredictable amount of time. Gauss Elimination has the disadvantage in the practical case of sparse matrices that it needs way more memory, and potentially more time.