EE1103: Numerical Methods

Programming Assignment # 3

Amizhthni P R K, EE21B015
Collaborators:
Ankita H Murthy, EE21B020
Anirudh BS, EE21B019

March 12, 2022

Contents

List of Figures

1 Problem 1: Nodal analysis using Kirchhoff's Current Law (KCL)

The aim of this exercise is to model a system of linear equations and solve the same using **Gauss elimination**. Fig. 1 depicts an electrical circuit with *four* nodes, namely, one reference node and three nodes with unknown voltages marked v_1, v_2 and v_3. The voltage at the reference node is 0 V. The current sources and resistors have designated values and are marked on the schematic. The unknown voltages v_1, v_2 and v_3 can be determined using KCL.

[(a)]Construct KCL equations for each of the three nodes. Note that these equations are to be written in terms of the current sources and the current through each resistor.

[(b)]Write a routine in C/C++ that implements Gauss elimination to solve for v_1, v_2 and v_3. In your report, paste a screenshot of the output displayed in your console.

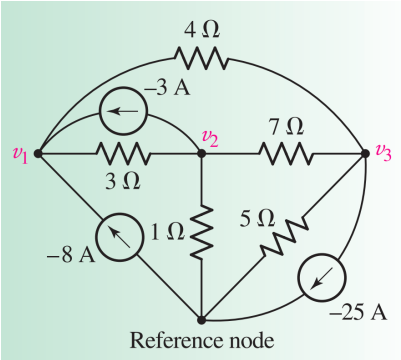

Figure 1: Circuit diagram

1.1 Approach

In this problem, we first use Kirchhoff's Current Law to obtain a system of 3 linear equations for node voltages v_1, v_2 and v_3. This involves equating the algebraic sum of currents at any junction to zero. We store the values of the coefficients of v_1, v_2 and v_3 and the constant values in an array which represents a 4X3 matrix.
We then operate on this matrix, carrying out Gauss elimination followed by back substitution, to obtain the values of the node voltages v_1, v_2 and v_3. This is done by using nested `for` loops and arrays.

1.1.1 Algorithm

The pseudocode for the problem is presented below.

Algorithm 1: Solving the set of three linear equations using Gauss elimination

Define Matrix=M
Define function swap rows which takes in values of rows i and j to be swapped:
for $k \leftarrow 0$ *to* 3 **do**
  $temp = M[i][k]$
  M[i][k]=M[j][k]
  M[j][k]=temp
end
Define function Forward Elimination that takes in matrix M:
for $k \leftarrow 0$ *to* 3 **do**
  $i_{max} = k$
  $\mathrm{v}_{max} = M[i_{max}][k]$
  for $I \leftarrow k+1$ *to* 3 **do**
    if $M[i][k] > v_{max}$ **then**
      $v_{max} = M[i][k], i_{max} = i$
    if i_{max} **then**
      $swap(M, k, i_{max})$
  end
  for $i \leftarrow k+1$ *to* 3 **do**
    $f = \frac{M[i][k]}{M[k][k]}$
    j $\leftarrow k+1, j \leq 3$
    $M[i][j] -= M[k][j] \times f$
  end
  M[i][k] $\leftarrow 0$
end
Define function Backward Substitution that takes in matrix M:
for $i \leftarrow 2$ *to* 0 *and* $i = i-1$ **do**
  $x[i] = M[i][3]$
  for $j \leftarrow i+1$ *to* 3 *and* $j = j+1$ **do**
    $x[i] -= M[i][j] + x[j]$
    end
    $x[i] = \frac{x[i]}{M[i][i]}$
  end

1.

1.2 Results

(a) The KCL equations obtained are as follows:

$$\mathbf{-7}v_1 \mathbf{-\ 4}v_2 \mathbf{-\ 3}v_3 = \mathbf{-\ 132}$$
$$\mathbf{-35}v_1 \mathbf{-\ 20}v_2 + \mathbf{83}v_3 = \mathbf{3500}$$
$$\mathbf{-7}v_1 + \mathbf{31}v_2 \mathbf{-\ 3}v_3 = \mathbf{63}$$

(b) The values of the voltages obtained are:

v_1= **5.413534 V**
v_2= **7.736842 V**
v_3= **46.315789 V**

(c) The code output for the values of Voltages at nodes as solved by Gauss elimination method is given under figure 2.

```
Solution for the system of equations involving V1 ,V2 and V3:
v1      =       5.412425
v2      =       7.737463
v3      =       46.312683
where v1, v2 and v3 are the potentials of junctions 1, 2 and 3 respectively
```

Figure 2: Code output for voltages at junctions

1.3 Inferences

We deduce the following inferences in this experiment:

- The node voltages are obtained using Gauss elimination with a high accuracy.
- Upon cancelling the integer denominator LCM values in Kirchhoff's equations and using the subsequent set of equations to solve, we obtain a slight error. The last three decimal places are off and a slight error is obtained when re-substituted into the equations. This stems from the fact that we arbitrarily cancel the denominator. To achieve a better degree of accuracy, we divide the integral numerators by the denominator, which in the first case is 12. This gives us a set of floating point numerals as shown in the code snippet.This error is not obtained when the integer LCM values are retained on both the LHS and RHS.
- Even though the solution obtained through the code is extremely close to the true answer, there is a slight discrepancy in the values as understood upon manually substituting the values into the KCL equations. This error is the round-off error obtained due to our use of six significant figures during the computation. It can be minimized by increasing the number of significant digits used in the computation. However, because computers carry only a limited number of significant figures, round-off errors can occur and must be considered when evaluating the results.
- In case pivoting is not done, there are chances of there occurring a division by zero in the steps. This would throw an error.
- In case any of the coefficients in the matrix are very small or close to zero, even the slightest deviation in calculations would give large errors. Therefore, in such cases, we must start with appropriate equations where the coefficients of the unknown values are sufficiently large.
- As the system gets larger, the computation time increases greatly.
- Most of the effort of the computation is incurred in the forward-elimination step rather than the back-substitution step. Thus, efforts to make the method more efficient should focus on this step.

1.4 Code

The code used for Gaussian elimination is given below in listing 1.

```
//GaussianElimination
#include <stdio.h>
#include <math.h>

#define N 3     // Number of unknowns

// function to reduce matrix to row reduced echelon form.
int ForwardElimination(double matrik[N][N+1]);

// function to calculate the values of the unknowns
void BackwardSubstitution(double matrik[N][N+1]);

// function to get matrix content
void GaussianElimination(double matrik[N][N+1])
{
    //reduction to row reduced echelon form
    int flag = ForwardElimination(matrik);

    //if matrix is singular
    if (flag != 1)
    {
        printf("Singular Matrix.\n");

        /* if the RHS of equation corresponding to
        zero row is 0, * system has infinitely
        many solutions, else inconsistent*/
        if (matrik[flag][N])
            printf("Inconsistent System."); //from cramer's rule
        else
            printf("May have infinitely many solutions.");

        return;
    }

    /* get solution to system and print it using
    backward substitution */
    BackwardSubstitution(matrik);
}

// function for elementary operation of swapping two rows
void swap_row(double matrik[N][N+1], int i, int j)
{
    //printf("Swapped rows %d and %d\n", i, j);

    for (int k=0; k<=N; k++)
    {
        double temp = matrik[i][k];
```

```
48          matrik[i][k] = matrik[j][k];
49          matrik[j][k] = temp;
50      }
51  }
52
53  // function to print matrix content at any stage
54  void print(double matrik[N][N+1])
55  {
56      for (int i=0; i<N; i++, printf("\n"))
57          for (int j=0; j<=N; j++)
58              printf("%lf ", matrik[i][j]);
59
60      printf("\n");
61  }
62
63  // function to reduce matrix to r.e.f.
64  int ForwardElimination(double matrik[N][N+1])
65  {
66      for (int k=0; k<N; k++)
67      {
68          // Initialize maximum value and index for pivot
69          int i_max = k;
70          int v_max = matrik[i_max][k];
71
72          /* find greater amplitude for pivot if any */
73          for (int i = k+1; i < N; i++)
74              if (fabs(matrik[i][k]) > v_max)
75                  v_max = matrik[i][k], i_max = i;
76
77          /* if a prinicipal diagonal element is zero,
78          * it denotes that matrix is singular, and
79          * will lead to a division-by-zero later. */
80          if (!matrik[k][i_max])
81              return k; // Matrix is singular
82
83          /* Swap the greatest value row with current row */
84          if (i_max != k)
85              swap_row(matrik, k, i_max);
86
87
88          for (int i=k+1; i<N; i++)
89          {
90              /* factor f to set current row kth element to 0,
91              * and subsequently remaining kth column to 0 */
92              double f = matrik[i][k]/matrik[k][k];
93
94              /* subtract fth multiple of corresponding kth
95              row element*/
96              for (int j=k+1; j<=N; j++)
97                  matrik[i][j] -= matrik[k][j]*f;
```

```
98
99              /* filling lower triangular matrix with zeros*/
100             matrik[i][k] = 0;
101         }
102
103     }
104
105     return 1;
106 }
107
108 // function to calculate the values of the unknowns
109 void BackwardSubstitution(double matrik[N][N+1])
110 {
111     double x[N]; // An array to store solution
112
113     /* Start calculating from last equation up to the
114     first */
115     for (int i = N-1; i >= 0; i--)
116     {
117         /* start with the RHS of the equation */
118         x[i] = matrik[i][N];
119
120         /* Initialize j to i+1 since matrix is upper
121         triangular*/
122         for (int j=i+1; j<N; j++)
123         {
124             /* subtract all the lhs values
125             * except the coefficient of the variable
126             * whose value is being calculated */
127             x[i] -= matrik[i][j]*x[j];
128         }
129
130         /* divide the RHS by the coefficient of the
131         unknown being calculated */
132         x[i] = x[i]/matrik[i][i];
133     }
134
135     printf("\nSolution for the system of equations involving V1 ,V2 and
    ↪  V3:\n");
136     for (int i=0; i<N; i++)
137         printf("v%d\t=\t%lf\n", i+1,x[i]);
138 }
139
140 // Driver program
141 int main()
142 {
143     /* input matrix */
144     double matrik[N][N+1] = {{0.5833, -0.3333,-0.25, -11}, //these values
    ↪  were obtained by solving the kirchhoff's equations manually
145                         {-0.3333, 1.4762, -0.1429, 3},
```

```
                        {-0.25, -0.1429, 0.5929, 25}
                        };

    GaussianElimination(matrik);
    //print(matrik); (can be used to print as and when required)
    printf("where v1, v2 and v3 are the potentials of junctions 1, 2 and 3
    ↪  respectively");
    return 0;
}
```

Listing 1: Code for Gaussian Elimination

1.5 Contributions

Complete code, Algorithm, Latex - Amizhthni

2 Problem 2

2.1 Using matrix operations for encoding/decoding messages

With modern networking environments becoming increasingly interconnected, the users' information is exposed to a lot of risks. Safely transmitting information, therefore becomes crucial. The two principles adopted for this purpose are called encoding (convert the original message to some secret message) and decoding (convert the secret message back to the original message). Some methods use matrices as part of the encoding and decoding processes ?.

A	B	C	D	E	F	G	H	I	J	K	L	M
1	2	3	4	5	6	7	8	9	10	11	12	13
N	O	P	Q	R	S	T	U	V	W	X	Y	Z
14	15	16	17	18	19	20	21	22	23	24	25	26

Figure 3: Mapping scheme for encoding messages

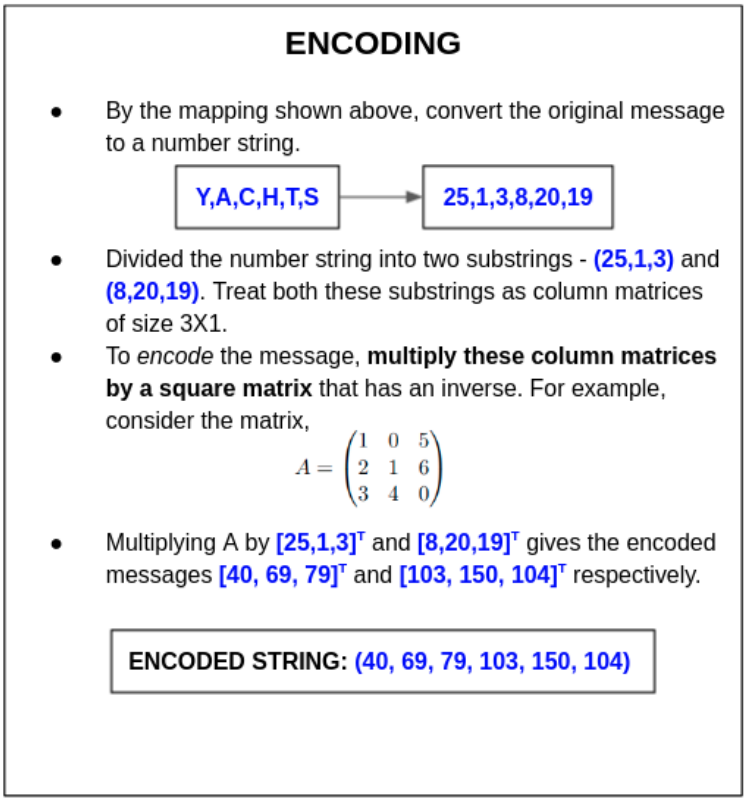

Figure 4: Encoding messages

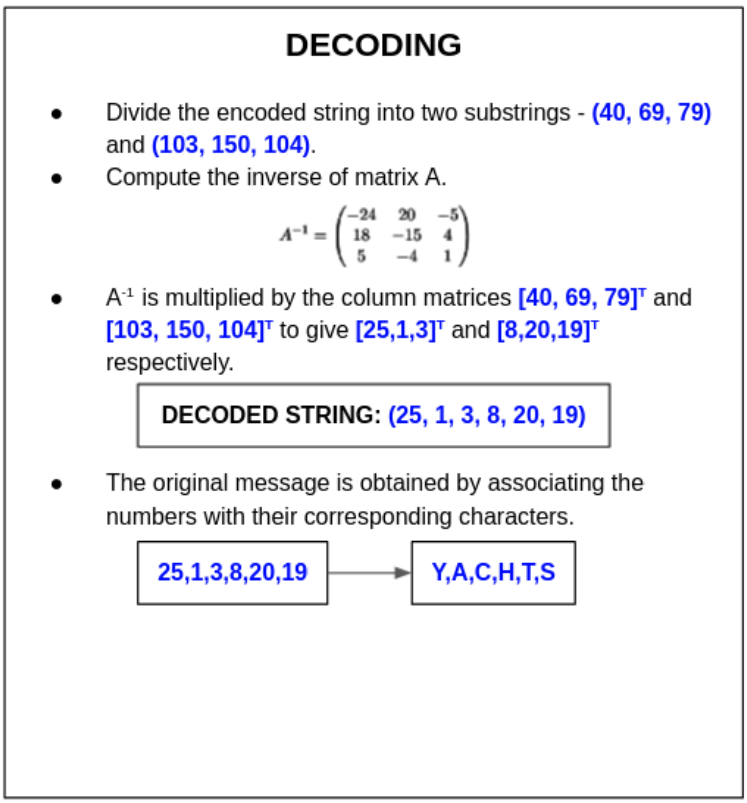

Figure 5: Decoding messages

For the remainder of this problem, we assume that the messages are character strings constructed from the set {A, B, C, ... Z}. The string of characters A to Z are mapped to integers 0 to 26 using the correspondence shown in Fig. 3. Note that the **blank space** character is mapped to 27. The procedure for encoding and decoding messages is outlined in Fig. 4 and Fig. 5. Use the same to solve the following parts:

[(a)]Find the inverse of the matrix M using LU decomposition. Follow Example 10.3 on Page 284 from Steven Chapra to solve this problem. Paste a screenshot of the matrices L, U and M^{-1}.

$$M = \begin{pmatrix} 1 & 4 & -3 \\ -2 & 8 & 5 \\ 3 & 4 & 7 \end{pmatrix}$$

For the given **encoded** strings, determine the original message that was sent. In other words, multiply each column matrix by M^{-1} and associate the numbers in the resulting column matrices with their respective characters. You may write a function to represent the character-number mapping and print the decoded message. In your report, paste a screenshot of the output displayed in your console.

2. (a) $[26, 115, 134]^T$, $[-16, 39, 88]^T$

   (b) $[7, 203, 269]^T$, $[-4, 269, 276]^T$, $[-20, 42, 156]^T$, $[-27, 116, 167]^T$

Coding remark: The ASCII equivalent of 'A' is 65. Add a normalisation factor of 64 to the decoded integer string to be able to implement the character-integer mapping routine. In this question, 27 maps to the blank space character. But 27 + 64 (91) stands for another

character in the ASCII system. Watch out for this and make sure to print a blank space if it turns up in your string!

2.2 Approach

In this problem, we first find the inverse of the matrix M using the LU decomposition method which gives us two matrices L and U satisfying the following equations:

$$UX=D$$
$$LU=A$$
$$LD=B$$

where A, B and D are column matrices.
This is done using arrays and nested `for` loops.
We then decode the given messages by multiplying each column matrix by M^{-1} to obtain a set of integer values.
Finally, we map these integers to their corresponding alphabet characters by adding 64 to each integer and converting it to the `char` data type. While doing so, we take blank spaces into account. The resulting sequence of letters gives us the decoded message.

2.3 Algorithm

The pseudocode for the inversion using LU Decomposition is given under 2.

Algorithm 2: Finding inverse using LU decomposition

```
Define Matrix=M
Define function swap rows which takes in values of rows i and j to be swapped:
for k ← 0 to 3 do
    temp = M[i][k]
    M[i][k]=M[j][k]
    M[j][k]=temp
end
Define function Forward Elimination that takes in matrix M:
for k ← 0 to 3 do
    i_max = k
    v_max = M[i_max][k]
    for I ← k + 1 to 3 do
        if M[i][k] > v_max then
            v_max = M[i][k], i_max = i
        if i_max then
            swap(M, k, i_max)
    end
    for i ← k + 1 to 3 do
        f = M[i][k] / M[k][k]
        j ← k + 1, j ≤ 3
        M[i][j]− = M[k][j] × f
    end
    M[i][k] ← 0
end
Define function Backward Substitution that takes in matrix M:
for i ← 2 to 0 and i = i − 1 do
    x[i] = M[i][3]
    for j ← i + 1 to 3 and j = j + 1 do
        x[i]− = M[i][j] + x[j]
        end
        x[i] = x[i] / M[i][i]
    end
```

The pseudocode for finding the decoded string is given under 3.

Algorithm 3: Finding the decoded strings

```
for i = k + 1 to 2 do
    for j = k + 1 to 2 do
        M_ij = M_ij − L_ik ∗ U_kj
    end
end
ReturnL, U
for i = 0 to < number − 1 do
    ans ← 0
    for j = 0 to j = 2 do
        ans = ans + ((M_inverse[(i%3)][j]) ∗ (message[i − (i%3) + j]));
    end
    decoded[i] = round(ans);

    for i = 0 to number−1 do
        if decoded[i]== 91 then
            text[i] =' ';
        else
            text[i] = 64 + decoded[i];
        end
    end
end
```

2.4 Results

- The inverse of the matrix M is displayed in the output in figure 6.
- The decoded messages are **IKIGAI** and **POWER RANGER**.

(a) The code output for the inverse of matrix M and the corresponding L and U matrices is given under figure 6.

```
The matrix LU decomposed which stores a_ij and f_ij values together i.e. L and U together
 1.000000  4.000000   -3.000000
-2.000000  16.000000   -1.000000
 3.000000  -0.500000   15.500000

The matrix U which is the upper triangular matrix which satisfies UX=D AND LU=A
 1.000000  4.000000   -3.000000
 0.000000  16.000000   -1.000000
 0.000000  0.000000   15.500000

The matrix L which is the lower triangular matrix which satisfies LD=B AND LU=A
 1.000000  0.000000   0.000000
-2.000000  1.000000   0.000000
 3.000000  -0.500000   1.000000

The Inverse Matrix
 0.145161 -0.161290 0.177419
 0.116935 0.064516 0.004032
 -0.129032 0.032258 0.064516
```

Figure 6: Finding the inverse of given matrix through LU Decomposition

(b) The code output for decoded message for problem **2B(1)** is given under figure 7.

```
The Inverse Matrix of M is as follows
 0.145161 -0.161290 0.177419
 0.116935 0.064516 0.004032
 -0.129032 0.032258 0.064516

Enter the number of letters in the supposed secret message
6
Enter the numbers which are to be encoded. Also hit enter after inserting each value
26
115
134
-16
39
88
THE SECRET MESSAGE IS:
IKIGAI
```

Figure 7: DECODING SECRET NUMBER 1

(c) The code output for the decoded message for problem **2B(2)** is given under figure 8.

```
The Inverse Matrix of M is as follows
 0.145161 -0.161290 0.177419
 0.116935 0.064516 0.004032
 -0.129032 0.032258 0.064516

Enter the number of letters in the supposed secret message
12
Enter the numbers which are to be encoded. Also hit enter after inserting each value
7
203
269
-4
269
276
-20
42
156
-27
116
167
THE SECRET MESSAGE IS:
POWER RANGER
```

Figure 8: DECODING SECRET NUMBER 2

2.5 Inferences

We deduce the following inferences in this question:

- The matrix M is invertible and LU decomposition gave its inverse quickly. Such inverse matrices are useful in encoding and decoding messages.
- The advantage of LU Decomposition over plain Gaussian Elimination is that LU Decomposition is efficient if one set of linear equations is repeatedly solved with different in-homogeneous terms (e.g., in the inverse matrix method). However, it is less efficient and more cumbersome than Gauss elimination if used only once.
- For our problem, the system is linear and of the form [A]X] = [B]. The matrix of coefficients [A] contains the parameters that express how the parts of the system interact or are coupled. Consequently, the equation [A]X] = [B] may be expressed as [Interactions][]response] = [stimuli].
  Therefore, the principles of superposition and proportionality hold. Superposition implies that if a system is subject to several different stimuli, the responses can be computed individually and the results summed to obtain a total response. Proportionality implies that that multiplying the stimuli by a quantity results in the response to those stimuli being multiplied by the same quantity.
- The LU decomposition algorithm requires the same total multiply/divide flops as for Gauss elimination. The only difference is that a little less effort is spent in the decomposition phase.
- Conversely, the substitution phase takes a little more effort. Thus, the number of flops for forward and back substitution is n^2, which implies that the total effort is the same as that in Gauss elimination.

2.6 Codes

(1) Code for **2(A)** is given under Listing 2

```
//To find the inverse of a matrix using LU decomposition
#include<math.h>
#include<stdio.h>

int main ()
{
  //Variable declarations
  int i, j, n, m;
  float M[3][3], d[3], C[3][3];
  //declaring two matrices, one 3X3 matrix for taking Matrix M and another column vector :M=MATRIX M of question,d= is the matrix D
  float x, s[3][3], y[3];

  //declaring two matrices, one 3X3 matrix for taking Matrix M and another column vector.s takes the inverse numbers, y is the intermediate d matrix in inverse calculation
  //Function def type
  void LU();

  n = 2;
  /* the matrix to be inverted is initialised here*/
  M[0][0] = 1.0; //initialised from 0 till N-1 since we use pointers
  M[0][1] = 4.0;
  M[0][2] = -3.0;
  M[1][0] = -2.0;
  M[1][1] = 8.0;
  M[1][2] = 5.0;
  M[2][0] = 3.0;
  M[2][1] = 4.0;
  M[2][2] = 7.0;


/* Call a sub-function to calculate the LU decomposed matrix. Note that
we pass the two dimensional array [D] to the function and get it back */
  LU (M, n);

  printf (" \n");
  printf
    ("The matrix LU decomposed which stores a_ij and f_ij values together i.e. L and U together \n");
  for (m = 0; m <= 2; m++)
    {
      printf (" %f  %f   %f \n", M[m][0], M[m][1], M[m][2]);
      //we reduce another loop by typing the column values
    }


```

```
  printf (" \n");
//Printing the U MATRIX from the LU matrix. we selectively choose the required elements. however this holds good if and only if N=3.
  printf
    ("The matrix U which is the upper triangular matrix which satisfies UX=D AND LU=A \n");
  printf (" %f  %f   %f \n", M[0][0], M[0][1], M[0][2]);
  printf (" %f  %f   %f \n", 0.00, M[1][1], M[1][2]);
  printf (" %f  %f   %f \n", 0.00, 0.00, M[2][2]);

  printf (" \n");
//Printing the L MATRIX from the LU matrix. we selectively choose the required elements. however this holds good if and only if N=3.
  printf
    ("The matrix L which is the lower triangular matrix which satisfies LD=B AND LU=A \n");
  printf (" %f  %f   %f \n", 1.0, 0.0, 0.0);
  printf (" %f  %f   %f \n", M[1][0], 1.0, 0.0);
  printf (" %f  %f   %f \n", M[2][0], M[2][1], 1.00);
printf (" \n");
//TO FIND THE INVERSE

/* to find the inverse we solve [D][y]=[d] with only one element in
the [d] array put equal to one at a time, for example:
in the first iteration: the first row of the column vector is 1 and rest are zeros. Similarly for the i th iteration,
the i th row of the column vector iz one while others are zeros*/

  for (m = 0; m <= 2; m++)
    {
      d[0] = 0.0; //all the non m th elements of the column vector are zeros
      d[1] = 0.0; //all the non m th elements of the column vector are zeros
      d[2] = 0.0; //all the non m th elements of the column vector are zeros
      d[m] = 1.0; //the m th iteration takes value 1
      for (i = 0; i <= n; i++)//row iteration
    {
      x = 0.0;
      for (j = 0; j <= i - 1; j++)//column iteration
        {
          x = x + M[i][j] * y[j];

        }
      y[i] = (d[i] - x);//intermediate d value
    }

      for (i = n; i >= 0; i--)
    {
```

```
      x = 0.0;
      for (j = i + 1; j <= n; j++)
        {
          x = x + M[i][j] * s[j][m];
        }
      s[i][m] = (y[i] - x) / M[i][i]; //whichever row is 1, the
      ↪  corresponding column gets the values. while rest are zeros.
    }
    }

//Print the inverse matrix
  printf ("The Inverse Matrix\n");
  for (m = 0; m <= 2; m++)
    {
      printf (" %f %f %f \n", s[m][0], s[m][1], s[m][2]);
    }
}

//The function that calcualtes the LU decomposed matrix.
//we use the concept of pointers here
void LU (float (*M)[3][3], int n) //we use pointers here so that the
↪  changes here are reflected in Original M matrix too and no need to
↪  return any value
{ //this function gives us the lu matrix with both upper and lower combined
↪  i.e. all aijs and fijs.
  int i, j, k, m;
  float multi;
  for (k = 0; k <= n - 1; k++) //fixes the column
    {
      for (j = k + 1; j <= n; j++)//fixes the row
    {
       multi= (*M)[j][k] / (*M)[k][k]; //the factor by which we need to
       ↪  multiply
      for (i = k; i <= n; i++)
        {
          (*M)[j][i] = (*M)[j][i] - multi * (*M)[k][i]; //this operation
          ↪  subtracts every element of the j th row from multi times
          ↪  corresponding element of preceding row
        }
      (*M)[j][k] = multi; //this is essentially because after all
      ↪  subtractions the leading term must remain intact and must not
      ↪  vanish
    }
    }
}
```

Listing 2: Code snippet used in the experiment 2A

(2) Code for **2(B)(1)** and **2(B)(2)** are given under Listing 3

```
//ENCRYPTION-DECRYPTION
#include<math.h>
#include<stdio.h>

double main ()
{

  int i, j, n, m; //variable declarations
  float M[3][3] = { {1, 4, -3}, {-2, 8, 5}, {3, 4, 7} }, d[3]; //matrix definition
  float x, s[3][3], y[3]; //extra matrix definitons, s takes the inverse numbers, y is the intermediate d matrix in inverse calculation
  void LU ();

  n = 2;

  LU (M, n);//function type def

  printf (" \n");


/*  TO FIND THE INVERSE */

  for (m = 0; m <= 2; m++)
    {
      d[0] = 0.0; //all the non m th elements of the column vector are zeros
      d[1] = 0.0;//all the non m th elements of the column vector are zeros
      d[2] = 0.0;//all the non m th elements of the column vector are zeros
      d[m] = 1.0;//the m th iteration takes value 1
      for (i = 0; i <= n; i++) //row ieration
    {
      x = 0.0;
      for (j = 0; j <= i - 1; j++) //column iteration
        {
          x = x + M[i][j] * y[j];
        }
      y[i] = (d[i] - x);//intermediate d value
    }

      for (i = n; i >= 0; i--)
    {
      x = 0.0;
      for (j = i + 1; j <= n; j++)
        {
          x = x + M[i][j] * s[j][m];
        }
      s[i][m] = (y[i] - x) / M[i][i];//whichever row is 1, the corresponding column gets the values. while rest are zeros.
```

```
46      }
47      }
48
49  /* Print the inverse matrix */
50    printf ("The Inverse Matrix of M is as follows\n");
51    for (m = 0; m <= 2; m++)
52      {
53        printf (" %f %f %f \n", s[m][0], s[m][1], s[m][2]);
54      }
55      printf("\n");
56
57  //decoding the hidden encrypted message
58    int number;
59    printf ("Enter the number of letters in the supposed secret message\n");
60    scanf ("%i", &number);
61    //we assign this length to the variable number
62    char ami[number];
63    printf ("Enter the numbers which are to be encoded. Also hit enter after
      ↪  inserting each value\n");
64    for (int t = 0; t <number / 3; t++)
65    //t is the number of column vectors each containing three rws
66      {
67      float a[3][1];
68      //initialise a matrix a with the encoding values i.e the column vector
69      for (int ank = 0; ank < 3; ank++)
70        {
71          for (int anj = 0; anj < 1; anj++)
72          scanf ("%f", &a[ank][anj]);
73          //each value entered by us is added to the matrix to give a column
            ↪  vector
74        }
75
76        for (int z = 0; z <= 2; z++)
77      {
78        double sum = s[z][0] * a[0][0] + s[z][1] * a[1][0] + s[z][2] *
          ↪  a[2][0];
79        //the elements of the column vector [3x1] are post multiplied with
          ↪  those of s [3x3] to get another colum vector [3x1]
80        int c = 64 + round (sum);
81        //we add 64 to the values of the column vector to get the ASCII
          ↪  Values of characters
82        if (c==91){
83            c=32;
84            //this addresses the discrepancy when a left bracket is typed
              ↪  instead of a space. space has ascii 32
85        }
86        ami[3*t+z]=c;
87  //this array takes up values of all characters as elements. 3*t+z is a
    ↪  special combination that yields 0,1,2...number methodically.
88  //this facilitates number and element assignment
```

```
89
90      }
91
92      }
93      printf("THE SECRET MESSAGE IS:\n"); //this prints values of the array a
        ↪  individually
94      for(i=0;i<number;i++){
95      printf("%c",ami[i]); }
96  }
97
98
99  //The function that calcualtes the LU decomposed matrix.
100 //we use the concept of pointers here
101 void LU (float (*M)[3][3], int n) //we use pointers here so that the
    ↪  changes here are reflected in Original M matrix too and no need to
    ↪  return any value
102 { //this function gives us the lu matrix with both upper and lower combined
    ↪  i.e. all aijs and fijs.
103   int i, j, k, m;
104   float multi;
105   for (k = 0; k <= n - 1; k++) //fixes the column
106     {
107       for (j = k + 1; j <= n; j++)//fixes the row
108     {
109        multi= (*M)[j][k] / (*M)[k][k]; //the factor by which we need to
          ↪  multiply
110       for (i = k; i <= n; i++)
111         {
112           (*M)[j][i] = (*M)[j][i] - multi * (*M)[k][i]; //this operation
              ↪  subtracts every element of the j th row from multi times
              ↪  corresponding element of preceding row
113         }
114       (*M)[j][k] = multi; //this is essentially because after all
          ↪  subtractions the leading term must remain intact and must not
          ↪  vanish
115     }
116     }
117 }
```

Listing 3: Code snippet used in the experiment 2B

2.7 Contributions

Code, Algorithm, Latex - Amizhthni