EE1103: Numerical Methods

Programming Assignment # 4

Nachiket Dighe, EE21B093

March 12, 2022

Contents

1 Problem 1 **1**
- 1.1 Approach . . . 1
- 1.2 Algorithm . . . 1
- 1.3 Results . . . 1
- 1.4 Inferences . . . 2
- 1.5 Code . . . 3

2 Problem 2 **4**
- 2.1 Approach . . . 4
- 2.2 Algorithm . . . 4
- 2.3 Results . . . 5
- 2.4 Inference . . . 7
- 2.5 Code . . . 8

List of Figures

1 Screenshot of V1,V2,V3 . . . 2
2 Inverse of Matrix using LU decomposition . . . 6
3 Decoded Message for Part 1 . . . 7
4 Decoded Message for Part 2 . . . 7

List of Tables

1 Problem 1

1.1 Approach

In this problem, we use the *Forward elimination* and *Back substitution* methods to calculate the unknown variables. We make use of the *For* loop and *Arrays* to find the solutions.

1.2 Algorithm

The Pseudo-Code for Finding the solution for equations of N-variables using *Gauss elimination* is given in Algorithm 1

Algorithm 1: Pseudo code for Gauss Elimination.

```
Inputs : a_ij, b_i, N
Initialize : i, j, k, x_i, sum, factor
(a) Forward Elimination
for k = 1 to N − 1 do
    for i = k + 1 to N do
        factor = a_ik / a_kk;
        for j = k + 1 to N do
            a_ij = a_ij − factor · a_kj;
        end
        b_i = b_i − factor · b_k;
    end
end
(b) Back Substitution
x_n = b_n / a_nn;
for i = N − 1 to 1 do
    sum = b_i;
    for j = i + 1to N do
        sum = sum + a_ij · x_j;
    end
    x_i = sum / a_ii;
end
```

1.3 Results

The Equations formed by using KCL are as follows:

$$0.5833v_1 - 0.3333v_2 - 0.25v_3 = -11$$

$$-0.3333v_1 + 1.4762v_2 - 0.1429v_3 = 3$$

$$-0.25v_1 - 0.1429v_2 + 0.5929v_3 = 25$$

The Output Displayed by using C code is shown in Fig. 1.

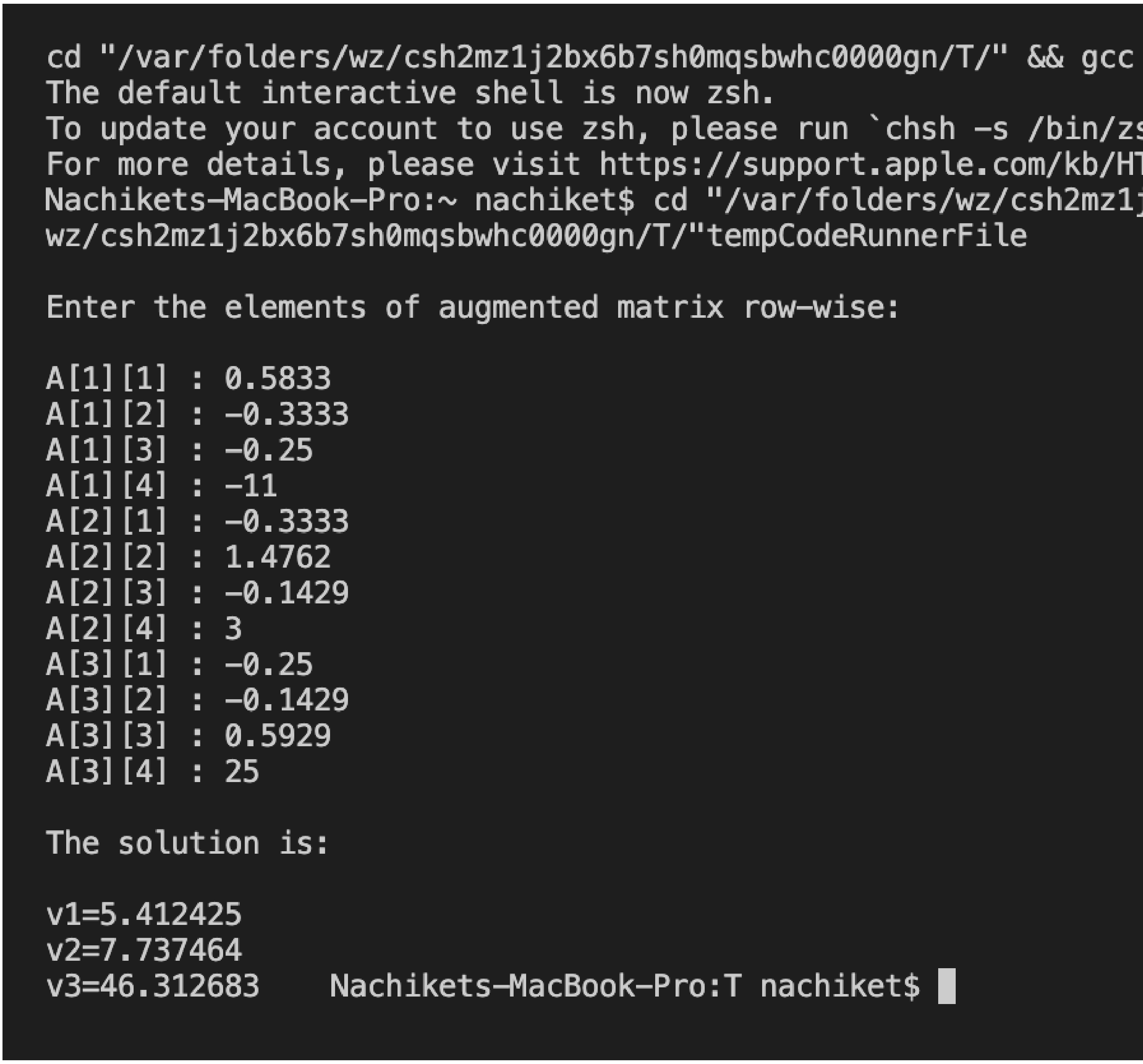

Figure 1: Screenshot of V1,V2,V3

1.4 Inferences

We deduce the following inferences in this experiment:

- In Gauss Elimination, as the system gets larger, the computation time increases greatly. The amount of flops increases nearly three orders of magnitude for every order of magnitude increase in the dimension.
- Most of the effort is incurred in the elimination step. Thus, efforts to make the method more efficient should probably focus on this step.
- In the Gauss Elimination, we have to be careful when the Pivot coefficient becomes 0. In such case ,we can use **Partial Pivoting**.Also another thing to

note is the *determinant* of Matrix A in $AX = B$. If this is equal to **0**, then we get **no** solutions .

- The Rank of the Matrix A in this particular problem is **3**.

1.5 Code

The code used for the experiments is mentioned in Listing 1

```
//Naive Gauss Elimination for n linear equations
#include<stdio.h>
int main()
{
    int i,j,k,n=3;
    float A[20][20],c,v[10],sum=0.0;
    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];
                }
            }
        }
    }
    v[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]*v[j];
        }
```

```
        v[i]=(A[i][n+1]-sum)/A[i][i];
    }
    printf("\nThe solution is: \n");
    for(i=1; i<=n; i++)
    {
        printf("\nv%d=%f\t",i,v[i]); /* v1, v2, v3 are the required
         ↪  solutions*/
    }
    return(0);
}
```

Listing 1: Code to calculate Linear equations using Gauss Elimination.

2 Problem 2

2.1 Approach

In this problem, we use Matrix Multiplication to find the inverse of a Matrix using LU decomposition. We also make use of Arrays and For loops.

2.2 Algorithm

The Pseudo-Code for Finding the Inverse of Matrix M using *LU decomposition* is given in Algorithm 2

Algorithm 2: Pseudo code for LU decomposition.

```
Inputs : a_ij, N
Initialize : i, j, k, sum
for j = 2 to N do
    a_1j = a1j/a11;
end
for j = 2 to N − 1 do
    for i = j to N do
        sum = 0;
        for k = 1 to j − 1 do
            sum = sum + a_ik · a_kj;
        end
        a_ij = a_ij − sum;
    end
    for k = j + 1 to N do
        sum = 0;
        for i = 1 to j − 1 do
            sum = sum + a_ji · a_ik;
        end
        a_jk = (a_jk − sum)/a_jj;
    end
end
sum = 0;
for k = 1 to N − 1 do
    sum = sum + a_nk ∗ a_kn;
end
a_nn = a_nn − sum;
```

2.3 Results

The Inverse of Matrix M in Part(a) is shown in Fig. 2. The Decoded message using Matrix Multiplication of Part(b) are shown in Fig. 3 and Fig. 4.

```
cd "/Users/nachiket/Desktop/" && gcc assignment4-prob2a.c -o assignment4-p

The default interactive shell is now zsh.
To update your account to use zsh, please run `chsh -s /bin/zsh`.
For more details, please visit https://support.apple.com/kb/HT208050.
Nachikets-MacBook-Pro:~ nachiket$ cd "/Users/nachiket/Desktop/" && gcc ass
rob2a
Enter the elements of matrix M:
 M[0][0]: 1
M[0][1]: 4
M[0][2]: -3
M[1][0]: -2
M[1][1]: 8
M[1][2]: 5
M[2][0]: 3
M[2][1]: 4
M[2][2]: 7
The Matrix M is:
1.000000        4.000000        -3.000000
-2.000000       8.000000        5.000000
3.000000        4.000000        7.000000

The Matrix L is:
1.000000        0.000000        0.000000
-2.000000       1.000000        0.000000
3.000000        -1.000000       1.000000

The Matrix U is:
1.000000        4.000000        -3.000000
0.000000        8.000000        5.000000
0.000000        0.000000        16.000000

The Matrix LU decomposed
1.000000        4.000000        -3.000000
-2.000000       16.000000       -1.000000
3.000000        -0.500000       15.500000

The Inverse Matrix M is:
0.145161        -0.161290       0.177419
0.116935        0.064516        0.004032
-0.129032       0.032258        0.064516
Nachikets-MacBook-Pro:Desktop nachiket$
```

Figure 2: Inverse of Matrix using LU decomposition

```
cd "/var/folders/wz/csh2mz1j2bx6b7sh0mqsbwhc0000gn/T/" && gcc tempCodeRunnerFile.c -o tempCode
The default interactive shell is now zsh.
To update your account to use zsh, please run `chsh -s /bin/zsh`.
For more details, please visit https://support.apple.com/kb/HT208050.
Nachikets-MacBook-Pro:~ nachiket$ cd "/var/folders/wz/csh2mz1j2bx6b7sh0mqsbwhc0000gn/T/" && gc
wz/csh2mz1j2bx6b7sh0mqsbwhc0000gn/T/"tempCodeRunnerFile
The decoded Strings for part A are:

9.000000        11.000000       9.000000
7.000000        1.000000        9.000000

The Decoded Message of the column matrices is:
IKIGAI
Nachikets-MacBook-Pro:T nachiket$
```

Figure 3: Decoded Message for Part 1

```
cd "/var/folders/wz/csh2mz1j2bx6b7sh0mqsbwhc0000gn/T/" && gcc tempCodeRunnerFile.c -o tempCode
The default interactive shell is now zsh.
To update your account to use zsh, please run `chsh -s /bin/zsh`.
For more details, please visit https://support.apple.com/kb/HT208050.
Nachikets-MacBook-Pro:~ nachiket$ cd "/var/folders/wz/csh2mz1j2bx6b7sh0mqsbwhc0000gn/T/" && gc
wz/csh2mz1j2bx6b7sh0mqsbwhc0000gn/T/"tempCodeRunnerFile
The decoded Strings for part B are:

16.000000       15.000000       23.000000
5.000000        18.000000       27.000000
18.000000       1.000000        14.000000
7.000000        5.000000        18.000000

The Decoded Message of the column matrices is:
POWER RANGER
Nachikets-MacBook-Pro:T nachiket$
```

Figure 4: Decoded Message for Part 2

2.4 Inference

We deduce the following inferences in this experiment:

- The rank of the Matrix M is **3**.
- 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 expended in the decomposition phase since the operations are not applied to the right-hand side
- Conversely, the substitution phase takes a little more effort. Thus, the number of flops for forward and back substitution is n2. The total effort is therefore identical to Gauss elimination.
- As can be seen from Fig. 3 and Fig. 4 the decoded message for 1st part is **IKIGAI** and the decoded message for 2nd part is **POWER RANGER**.

2.5 Code

The code used for the experiments are mentioned in Listing 2, Listing 3, Listing 4.

```
/* To find the inverse of a matrix using LU decomposition */
#include<math.h>
#include<stdio.h>
int         main()
{
        int i,j,n=2,m;
    float D[3][3],d[3];
    float x,I[3][3],y[3];
        void LU();
//Initialize the Matrix D
printf("Enter the elements of matrix M:\n ");
for(i=0;i<3;i++)
{
    for(j=0;j<3;j++)
    {
        printf("M[%d][%d]: ",i,j);
        scanf("%f",&D[i][j]);
    }
}

//Print the Actual matrix M
    printf("The Matrix M is: \n");
        for(m=0;m<=2;m++)
          printf("%f \t %f \t %f \n", D[m][0],D[m][1],D[m][2]);

// Call a sub-function to calculate the LU decomposed matrix.
    LU(D,n);
        printf("\nThe Matrix LU decomposed \n");
        for(m=0;m<=2;m++)
        {
        printf("%f \t %f \t %f \n",D[m][0],D[m][1],D[m][2]);
        }

/*  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(m=0;m<=2;m++)
    {
        d[0]=0.0;d[1]=0.0;d[2]=0.0;
            d[m]=1.0;
            for(i=0;i<=2;i++)
        {
```

```
            x=0.0;
                for(j=0;j<=i-1;j++)
                 x=x+D[i][j]*y[j];
                 y[i]=(d[i]-x);
            }

            for(i=2;i>=0;i--)
        {
            x=0.0;
                for(j=i+1;j<=2;j++)
                x=x+D[i][j]*I[j][m];
                 I[i][m]=(y[i]-x)/D[i][i];
            }
        }

/* Print the inverse matrix */
        printf("\nThe Inverse Matrix M is: \n");
        for(m=0;m<=2;m++)
          printf("%f \t %f \t %f \n", I[m][0],I[m][1],I[m][2]);

}

// The function that calcualtes the LU deomposed matrix
void LU(float(*D)[3][3],int n)
{
        int i,j,k,m;
        float x,L[3][3],U[3][3];
    for(j=0; j<3; j++)
    {
        for(i=0; i<3; i++)
        {
            if(i<=j)
            {
                U[i][j]=(*D)[i][j];
                for(k=0; k<i-1; k++)
                    U[i][j]-=L[i][k]*U[k][j];
                if(i==j)
                    L[i][j]=1;
                else
                    L[i][j]=0;
            }
            else
            {
                L[i][j]=(*D)[i][j];
                for(k=0; k<=j-1; k++)
                    L[i][j]-=L[i][k]*U[k][j];
                L[i][j]/=U[j][j];
```

```
                U[i][j]=0;
            }
        }
    }
   //To print the Matrix L
    printf("\nThe Matrix L is: \n");
    for(m=0;m<=2;m++)
          printf("%f \t %f \t %f \n", L[m][0],L[m][1],L[m][2]);

    //To print the Matrix U
    printf("\nThe Matrix U is: \n");
    for(m=0;m<=2;m++)
          printf("%f \t %f \t %f \n", U[m][0],U[m][1],U[m][2]);

     for(k=0;k<=n-1;k++)
    {
          for(j=k+1;j<=n;j++)
          {
            x=(*D)[j][k]/(*D)[k][k];
            for(i=k;i<=n;i++)
               (*D)[j][i]=(*D)[j][i]-x*(*D)[k][i];

            (*D)[j][k]=x;
          }
    }
}
```

Listing 2: Code To Calculate Inverse of Matrix.

```
#include<stdio.h>
#include<math.h>
int main()
{
    int i,j,k;
    float M_inv[3][3]={{0.1451,-0.1612,0.1774},
                       {0.1169,0.0645,0.0040},
                       {-0.1290,0.0322,0.0645}};
    //This is 3x3 Inverse matrix of M which was calculated in problem
    ↪ 1
    float col[2][3]={{26,115,134},{-16,39,88}};
    //These are the String Matrices Given in problem
    float res[2][3]={{0,0,0},{0,0,0}};
    char decode[2][3];
    printf("The decoded Strings for part A are: \n\n");
    for(k=0;k<2;k++)
    {
        for(i=0;i<3;i++)
```

```
        {
            for(j=0;j<3;j++)
               res[k][i]+=M_inv[i][j]*col[k][j];
            printf("%f\t ",round(res[k][i]));

        }
        printf("\n");

    }
    printf("\nThe Decoded Message of the column matrices is:\n");
    //This Loop decodes the given matrix into its ASCII equivalent
    //and then finds the letters
    for(i=0;i<2;i++)
    {
        for(j=0;j<3;j++)
        {
            decode[i][j]=(int)(round(res[i][j]))%26+64;
            if((int)decode[i][j]==91)
            {
                decode[i][j]=' ';
            }
            printf("%c",decode[i][j]);
        }
    }
    printf("\n");
}
```

Listing 3: Decoding a Message Part A.

```
#include<stdio.h>
#include<math.h>
int main()
{
    int i,j,k;
    float M_inv[3][3]={{0.1451,-0.1612,0.1774},
                       {0.1169,0.0645,0.0040},
                       {-0.1290,0.0322,0.0645}};
    //This is 3x3 Inverse matrix of M which was calculated in problem
    ↪ 1
    float col[4][3]={{7,203,269},
                     {-4,269,276},
                     {-20,42,156},
                     {-27,116,167} };
    //These are the String Matrices Given in problem
    float res[4][3]={{0,0,0},{0,0,0},{0,0,0},{0,0,0}};
    char decode[4][3];
    printf("The decoded Strings for part B are: \n\n");
```

```
    for(k=0;k<4;k++)
    {
        for(i=0;i<3;i++)
        {
            for(j=0;j<3;j++)
               res[k][i]+=M_inv[i][j]*col[k][j];
            printf("%f \t ",round(res[k][i]));

        }
        printf("\n");

    }
    printf("\nThe Decoded Message of the column matrices is:\n");
    //This Loop decodes the given matrix into its ASCII equivalent
    //and then finds the letters
    for(i=0;i<4;i++)
    {

        for(j=0;j<3;j++)
        {
            k=(int)(round(res[i][j]));
            if(k==27)
            {
                decode[i][j]=' ';
            }
            decode[i][j]=(int)(round(res[i][j]))%26+64;
            if(k==27)
            {
                decode[i][j]=' ';
            }
            printf("%c",decode[i][j]);
        }
    }
    printf("\n");
}
```

Listing 4: Decoding a Message part B.