ASSIGNMENT 04 SOLUTIONS (ALL IN ONE)

Suppose we have a linear system of equations as following and we want to solve for x.

$$\begin{aligned}
a_{11}x_1 + a_{12}x_2 + a_{13}x_2 + \cdots + a_{1n}x_n &= b_1 \\
a_{21}x_1 + a_{22}x_2 + a_{23}x_2 + \cdots + a_{2n}x_n &= b_2 \\
a_{31}x_1 + a_{32}x_2 + a_{33}x_2 + \cdots + a_{3n}x_n &= b_3 \\
&\cdots \\
a_{n1}x_1 + a_{n2}x_2 + a_{n3}x_2 + \cdots + a_{nn}x_n &= b_n
\end{aligned}$$

First we'll write in matrix form.

$$\underbrace{\begin{bmatrix} a_{11} & a_{12} & a_{13} & \dots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \dots & a_{2n} \\ a_{31} & a_{32} & a_{33} & \dots & a_{3n} \\ \dots & \dots & \dots & \ddots & \dots \\ a_{n1} & a_{n2} & a_{n3} & \dots & a_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \dots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ \dots \\ b_n \end{bmatrix}}_{\mathbf{AX=B}} \qquad \underbrace{\begin{bmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} & b_1 \\ a_{21} & a_{22} & a_{23} & \cdots & a_{2n} & b_2 \\ a_{31} & a_{32} & a_{33} & \cdots & a_{3n} & b_3 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{n1} & a_{n2} & a_{n3} & \cdots & a_{nn} & b_n \end{bmatrix}}_{\mathbf{AUGMENTED\ MATRIX}}$$

We'll use matrix **Gauss Elimination Method** and **Gauss Jordan Method** to solve such system. Below are the graphical representations of both the methods. For detailed methodology please look the book Numerical Analysis by Richard L. Burden and J. Douglas Faires.

Gauss Elimination Method

In this method our goal is to write the augmented matrix in upper triangular form and using the back substitution we can get the values of x.

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} & b_1 \\ a_{21} & a_{22} & a_{23} & \cdots & a_{2n} & b_2 \\ a_{31} & a_{32} & a_{33} & \cdots & a_{3n} & b_3 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{n1} & a_{n2} & a_{n3} & \cdots & a_{nn} & b_n \end{bmatrix} \implies \begin{bmatrix} x & x & x & \cdots & x & x \\ 0 & x & x & \cdots & x & x \\ 0 & 0 & x & \cdots & x & x \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & x & x \end{bmatrix}$$

Gauss Jordan Method

In this method our goal is to write the augmented matrix in such a way that all the elments above and below the diagonal become zero.(Now we don't need to back substitute in the equations.)

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} & b_1 \\ a_{21} & a_{22} & a_{23} & \cdots & a_{2n} & b_2 \\ a_{31} & a_{32} & a_{33} & \cdots & a_{3n} & b_3 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{n1} & a_{n2} & a_{n3} & \cdots & a_{nn} & b_n \end{bmatrix} \quad \Longrightarrow \quad \begin{bmatrix} x & 0 & 0 & \cdots & 0 & x \\ 0 & x & 0 & \cdots & 0 & x \\ 0 & 0 & x & \cdots & 0 & x \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & x & x \end{bmatrix}$$

Two achieve this we can perform following elementary row operations:

1. Swapping two rows,
2. Multiplying a row by a nonzero number,
3. Adding a multiple of one row to another row.

It's not neccessory that every system has an unique solution there can be infinity solutions or no solution at all and we need to incorparate that message in our program as well.

Like everytime I'm going to write a c file **`linear.c`** which will contain all the necessary functions such as Gauss Elimination and Gauss Jordan method with and without pivoting.

In [ ]:
```c
// it contains the linear equations solving functions
// (gauss elimination, gauss elimination with partial pivoting and gauss jordan)
// use #include"linear.c" in the program you wished to use this

//DISPLAYING THE MATRIX
//Parameters: (n) no of eq or rows,mat[n][n+1]
void printmat(int n,double mat[n][n+1]){
    int i,j;
    for(i=0;i<n;i++){
        for(j=0;j<n+1;j++){
            printf("%.2lf\t",mat[i][j]);
        }
        printf("\n");
    }
}
//Gaussian Elimination
//Parameters: (n) no of eq/rows,mat[n][n+1],x[n]
void gausselim(int n,double A[n][n+1],double x[]){
    int i,j,k;
    for(i=0;i<n-1;i++){
        // swapping rows if the diagonal is zero
        if (A[i][i]==0){
            for(k=i+1;k<n;k++)
            {
                if (A[k][i]!=0){
                    for(j=0;j<=n;j++){
                        double temp;
                        temp=A[i][j];
                        A[i][j]=A[k][j];
                        A[k][j]=temp;
                    }
                }
            }
        }
        //Begin the Gauss Elimination
        for(k=i+1;k<n;k++)
        {
            double term;
```

```
            term=A[k][i]/A[i][i];
            for(j=0;j<=n;j++){
                A[k][j]=A[k][j]-term*A[i][j];
            }
        }
    }
    printf("\nThe upper triangular matrix:\n");
    printmat(n,A);
    if (A[n-1][n-1]==0){
        printf("\nSorry! No unique solution exists.");
    }
    else{
        //Start with the back-substitution
        for(i=n-1;i>=0;i--){
            x[i]=A[i][n];
            for(j=i+1;j<n;j++){
                x[i]=x[i]-A[i][j]*x[j];
            }
            x[i]=x[i]/A[i][i];
        }
        printf("\nThe solution of linear equations:\n");
        for(i=0;i<n;i++){
            printf(" x[%d]= %.2lf\n",i+1,x[i]);
        }
    }
}
//Gaussian Elimination with Partial Pivoting
//Parameters: (n) no of eq/rows,mat[n][n+1],x[n]
void gausspivot(int n,double A[n][n+1],double x[]){
    int i,j,k;
    for(i=0;i<n-1;i++){
        //Partial Pivoting
        for(k=i+1;k<n;k++){
            //If the diagonal element (absolute value) is smaller than any of th
            if(fabs(A[i][i])<fabs(A[k][i])){
                //Swap the rows in the matrix
                for(j=0;j<=n;j++){
                    double temp;
                    temp=A[i][j];
                    A[i][j]=A[k][j];
                    A[k][j]=temp;
                }
            }
        }
        //Begin the Gauss Elimination
        for(k=i+1;k<n;k++)
        {
            double term;
            term=A[k][i]/A[i][i];
            for(j=0;j<=n;j++){
                A[k][j]=A[k][j]-term*A[i][j];
            }
        }
    }
    printf("\nThe upper triangular matrix:\n");
    printmat(n,A);
    if (A[n-1][n-1]==0){
        printf("\nSorry! No unique solution exists.");
    }
    else{
        //Start with the back-substitution
```

```
        for(i=n-1;i>=0;i--){
            x[i]=A[i][n];
            for(j=i+1;j<n;j++){
                x[i]=x[i]-A[i][j]*x[j];
            }
            x[i]=x[i]/A[i][i];
        }
        printf("\nThe solution of linear equations:\n");
        for(i=0;i<n;i++){
            printf(" x[%d]= %.2lf\n",i+1,x[i]);
        }
    }
}
//Gauss Jordan method with Partial Pivoting
//Parameters: (n) no of eq/rows,mat[n][n+1],x[n]
void gaussjordan(int n,double A[n][n+1],double x[]){
    int i,j,k;
    for(i=0;i<n;i++){
        //Partial Pivoting
        for(k=i+1;k<n;k++){
            //If the diagonal element (absolute value) is smaller than any of th
            if(fabs(A[i][i])<fabs(A[k][i])){
                //Swap the rows in the matrix
                for(j=0;j<=n;j++){
                    double temp;
                    temp=A[i][j];
                    A[i][j]=A[k][j];
                    A[k][j]=temp;
                }
            }
        }
        //Begin the Gauss Jordan
        for (k=0;k<n;k++)
        {
            if (k!=i){
                double term;
                term=A[k][i]/A[i][i];
                for (j=0;j<=n;j++)
                {
                    A[k][j]=A[k][j]-term*A[i][j];
                }
            }
        }
    }

    if (A[n-1][n-1]==0){
        printf("\nSorry! No unique solution exists.");
    }
    else{
        printf("\nThe diagonal augmented matrix:\n");
        printmat(n,A);
        // getting the x array
        for (i=0;i<n;i++){
            x[i]=A[i][n]/A[i][i];
        }
        printf("\nThe solution of linear equations:\n");
        for(i=0;i<n;i++){
            printf(" x[%d]= %.2lf\n",i+1,x[i]);
        }
    }
}
```

Now since I have written all the required functions in the `"linear.c"`, we'll use `#include"linear.c"` as a library.

How to use the functions?

- Gauss Elimination Method: `gausselim(n,a,x)`
- Gauss Elimination with partial pivoting: `gausspivot(n,a,x)`
- Gauss Jordan with partial pivoting: `gaussjordan(n,a,x)`

Where **n** is the no of variables (or no of equations) defined as integer, **a** is two dimensional array with n rows and n+1 coulumn defined as double and **x** is the unknown variable arrays which we want to calculate defined as double.

PROBLEM 1 (a) : Gauss Elimination Method

In [ ]:
```c
// problem 1a
// make sure "linear.c" file is in the same directory
#include<stdio.h>
#include<math.h>
#include"linear.c"

int main()
{
    int i,n=4;  // no of equations
    //Declare a matrix to store the user given matrix
    double a[4][5]={
                        {1,-1,2,-1,-8},
                        {2,-2,3,-3,-20},
                        {1,1,1,0,-2},
                        {1,-1,4,3,4}
                    };
    //Declare an array to store the solution of equations
    double x[n];
    printf("The augmented matrix:\n");
    printmat(n,a);  // printing the augmented matrix
    gausselim(n,a,x);   // Perform Gauss Elimination
}
```

OUTPUT:

The augmented matrix:

1.00 -1.00 2.00 -1.00 -8.00

2.00 -2.00 3.00 -3.00 -20.00

1.00 1.00 1.00 0.00 -2.00

1.00 -1.00 4.00 3.00 4.00

The upper triangular matrix:

1.00 -1.00 2.00 -1.00 -8.00

0.00 2.00 -1.00 1.00 6.00

0.00 0.00 -1.00 -1.00 -4.00

0.00 0.00 0.00 2.00 4.00

The solution of linear equations:

x[1]= -7.00

x[2]= 3.00

x[3]= 2.00

x[4]= 2.00

PROBLEM 1 (b) : Gauss Elimination Method

In [ ]:

```c
// problem 1b
// make sure "linear.c" file is in the same directory
#include<stdio.h>
#include<math.h>
#include"linear.c"

int main()
{
    int i,n=5;  // no of equations
    //Declare a matrix to store the user given matrix
    double a[5][6]={
                        {2,1,-1,1,-3,7},
                        {1,0,2,-1,1,2},
                        {0,-2,-1,1,-1,-5},
                        {3,1,-4,0,5,6},
                        {1,-1,-1,-1,1,3}
                    };
    //Declare an array to store the solution of equations
    double x[n];
    printf("The augmented matrix:\n");
    printmat(n,a);  // printing the augmented matrix
    gausselim(n,a,x);   // Perform Gauss Elimination
}
```

OUTPUT:

The augmented matrix:

2.00 1.00 -1.00 1.00 -3.00 7.00

1.00 0.00 2.00 -1.00 1.00 2.00

0.00 -2.00 -1.00 1.00 -1.00 -5.00

3.00 1.00 -4.00 0.00 5.00 6.00

1.00 -1.00 -1.00 -1.00 1.00 3.00

The upper triangular matrix:

2.00 1.00 -1.00 1.00 -3.00 7.00

0.00 -0.50 2.50 -1.50 2.50 -1.50

0.00 0.00 -11.00 7.00 -11.00 1.00

0.00 0.00 0.00 -3.18 12.00 -3.45

0.00 0.00 0.00 0.00 -4.89 5.54

The solution of linear equations:

x[1]= 1.92

x[2]= 1.96

x[3]= -0.99

x[4]= -3.19

x[5]= -1.13

PROBLEM 2 : Unique solution - No solution - Infinity solution

Linear system : $2x_1 - 6\alpha x_2 = 3$

$3\alpha x_1 - x_2 = \frac{3}{2}$

in matrix form

$$\begin{bmatrix} 2 & -6\alpha \\ 3\alpha & -1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 3 \\ 3/2 \end{bmatrix}$$

now writing in augmented matrix

$$\left[\begin{array}{cc|c} 2 & -6\alpha & 3 \\ 3\alpha & -1 & 3/2 \end{array}\right]$$

converting in upper triangular matrix using gauss elim.

$R_1 \rightarrow R_1 \times \frac{3}{2}\alpha$

$$\left[\begin{array}{cc|c} 3\alpha & -9\alpha^2 & 9/2\,\alpha \\ 3\alpha & -1 & 3/2 \end{array}\right]$$

$R_2 \rightarrow R_2 - R_1$

$$\left[\begin{array}{cc|c} 3\alpha & -9\alpha^2 & 9/2\,\alpha \\ 0 & 9\alpha^2 - 1 & 3/2 - 9/2\,\alpha \end{array}\right] \Rightarrow \left[\begin{array}{cc|c} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{array}\right]$$

this element will decide the nature of sol^{n}.

[a] for no solution

$a_{22} = 0$ AND $a_{23} \neq 0$

$9\alpha^2 - 1 = 0$ $\qquad$ $3/2 - 9/2\,\alpha \neq 0$

$\alpha = \pm 1/3$ AND $\alpha \neq 1/3$

hence for $\alpha = -1/3$ the system has no solution.

[b] for infinte solution

$a_{22} = 0$ AND $a_{23} = 0 \Rightarrow 3/2 - 9/2\alpha = 0$

$\alpha = \pm 1/3$ AND $\alpha = 1/3$

since for $\alpha = 1/3$ satisfy both the equation hence $\alpha = 1/3$, system will have infinete no of sol

[c] for unique solution

from part (a) the system has no sol^{n} for $\alpha = -1/3$ and from part (b) system has infinite no of sol^{n} for $\alpha = 1/3$

so, to get unique solution $\boxed{\alpha \neq \pm 1/3}$

<u>suppose that $\alpha = 1$</u>

now the augmented matrix

$$\begin{bmatrix} 3\alpha & -9\alpha^2 & | & 9/2\alpha \\ 0 & 9\alpha^2 - 1 & | & 3/2 - 9/2\alpha \end{bmatrix} = \begin{bmatrix} 3 & -9 & | & 9/2 \\ 0 & 8 & | & -3 \end{bmatrix}$$

unique sol^{n} will exist.

now calculating the unique sol^{n}

$3x_1 - 9x_2 = 9/2$

$8x_2 = -3$

$\boxed{x_2 = -\frac{3}{8}}$

$3x_1 + 9 \times \frac{3}{8} = \frac{9}{2}$

$x_1 + \frac{9}{8} = \frac{3}{2} \times \frac{4}{4}$

$\boxed{x_1 = \frac{3}{8}}$

for $\alpha = 1$ unique sol^{n} $(x_1, x_2) = \left(\frac{3}{8}, -\frac{3}{8}\right)$

Round-off error: Round-off error occurs because computers use fixed number of bits and hence fixed number of binary digits to represent numbers. In a numerical computation round-off errors are introduced at every stage of computation. Hence though an individual round-off error due to a given number at a given numerical step may be small but the cumulative effect can be significant.

When the number of bits required for representing a number are less then the number is usually rounded to fit the available number of bits. This is done either by chopping or by symmetric rounding.

Chopping: Chopping a number by chopping amounts to dropping the extra digits. Here the given number is truncated. Suppose that we are using a computer with a fixed word length of four digits. Then the truncated representation of the number 72.32451 will be 72.32. The digits 451 will be dropped.

Rounding Rounding a number means replacing that number with an approximate value that has a shorter, simpler, or more explicit representation. For example, replacing 23.4476 with 23.45, the fraction 312/937 with 1/3, or the expression √2 with 1.414.

Source:https://nptel.ac.in/content/storage2/courses/122104019/numerical-analysis/Rathish-kumar/numerr/new4.htm

For the problem 3 I'm rewriting the `"linear.c"` in a new file **`"arithmetic.c"`** which will contain the functions for gauss elimination with three digit chopping, three digit rounding and partial pivoting.

In [ ]:
```c
// Rewriting the <linear.c> to add 3-digit chopping and rounding arithmetic
// use #include"arithmetic.c" in the program you wished to use this

// displaying the matrix
void printmat(int n,double mat[n][n+1]){
    int i,j;
    for(i=0;i<n;i++){
        for(j=0;j<n+1;j++){
            printf("%lf\t",mat[i][j]);
        }
        printf("\n");
    }
}
// three digit chopping or truncating
double truncating(double x)
{
    double y,z;
    y=x*1000.0;
    z=floor(y);
    return z/1000.0;
}
// three digit rounding arithmetic
double rounding(double x)
{
    double y,z;
    y=(x*1000.0)+.5;
    z=floor(y);
    return z/1000.0;
}
```

```c
// Gaussian elimination with three digit chopping
void gausschopping(int n,double A[n][n+1],double x[]){
    int i,j,k;
    for(i=0;i<n-1;i++){
        // swapping rows if the diagonal is zero
        if (A[i][i]==0){
            for(k=i+1;k<n;k++)
            {
                if (A[k][i]!=0){
                    for(j=0;j<=n;j++){
                        double temp;
                        temp=A[i][j];
                        A[i][j]=A[k][j];
                        A[k][j]=temp;
                    }
                }
            }
        }
        //Begin the Gauss Elimination
        for(k=i+1;k<n;k++)
        {
            double term;
            term=truncating(A[k][i]/A[i][i]);
            for(j=0;j<=n;j++){
                A[k][j]=truncating(A[k][j]-term*A[i][j]);
            }
        }
    }
    printf("\nThe upper triangular matrix:\n");
    printmat(n,A);
    if (A[n-1][n-1]==0){
        printf("\nSorry! No unique solution exists.");
    }
    else{
        //Start with the back-substitution
        for(i=n-1;i>=0;i--){
            x[i]=A[i][n];
            for(j=i+1;j<n;j++){
                x[i]=truncating(x[i]-A[i][j]*x[j]);
            }
            x[i]=truncating(x[i]/A[i][i]);
        }
        printf("\nThe solution of linear equations:\n");
        for(i=0;i<n;i++){
            printf(" x[%d]= %lf\n",i+1,x[i]);
        }
    }
}
// Gaussian elimination with three digit rounding
void gaussround(int n,double A[n][n+1],double x[]){
    int i,j,k;
    for(i=0;i<n-1;i++){
        // swapping rows if the diagonal is zero
        if (A[i][i]==0){
            for(k=i+1;k<n;k++)
            {
                if (A[k][i]!=0){
                    for(j=0;j<=n;j++){
                        double temp;
                        temp=A[i][j];
                        A[i][j]=A[k][j];
```

```
                        A[k][j]=temp;
                    }
                }
            }
        }
        //Begin the Gauss Elimination
        for(k=i+1;k<n;k++)
        {
            double term;
            term=rounding(A[k][i]/A[i][i]);
            for(j=0;j<=n;j++){
                A[k][j]=rounding(A[k][j]-term*A[i][j]);
            }
        }
    }
    printf("\nThe upper triangular matrix:\n");
    printmat(n,A);
    if (A[n-1][n-1]==0){
        printf("\nSorry! No unique solution exists.");
    }
    else{
        //Start with the back-substitution
        for(i=n-1;i>=0;i--){
            x[i]=A[i][n];
            for(j=i+1;j<n;j++){
                x[i]=rounding(x[i]-A[i][j]*x[j]);
            }
            x[i]=rounding(x[i]/A[i][i]);
        }
        printf("\nThe solution of linear equations:\n");
        for(i=0;i<n;i++){
            printf(" x[%d]= %lf\n",i+1,x[i]);
        }
    }
}
// Gaussian elimination with partial pivoting
void gausspivot(int n,double A[n][n+1],double x[]){
    int i,j,k;
    for(i=0;i<n-1;i++){
        //Partial Pivoting
        for(k=i+1;k<n;k++){
            //If the diagonal element (absolute value) is smaller than any of th
            if(fabs(A[i][i])<fabs(A[k][i])){
                //Swap the rows in the matrix
                for(j=0;j<=n;j++){
                    double temp;
                    temp=A[i][j];
                    A[i][j]=A[k][j];
                    A[k][j]=temp;
                }
            }
        }
        //Begin the Gauss Elimination
        for(k=i+1;k<n;k++)
        {
            double term;
            term=A[k][i]/A[i][i];
            for(j=0;j<=n;j++){
                A[k][j]=A[k][j]-term*A[i][j];
            }
        }
```

```
    }
    printf("\nThe upper triangular matrix:\n");
    printmat(n,A);
    if (A[n-1][n-1]==0){
        printf("\nSorry! No unique solution exists.");
    }
    else{
        //Start with the back-substitution
        for(i=n-1;i>=0;i--){
            x[i]=A[i][n];
            for(j=i+1;j<n;j++){
                x[i]=x[i]-A[i][j]*x[j];
            }
            x[i]=x[i]/A[i][i];
        }
        printf("\nThe solution of linear equations:\n");
        for(i=0;i<n;i++){
            printf(" x[%d]= %.2lf\n",i+1,x[i]);
        }
    }
}
```

Now since I have written all the required functions in the `"arithmetic.c"`, we'll use `#include"arithmetic.c"` as a library.

How to use these functions?

- With three digit chopping: `gausschopping(n,a,x)`
- With three digit rounding: `gaussround(n,a,x)`
- With partial pivoting: `gausspivot(n,a,x)`

Where **n** is the no of variables (or no of equations) defined as integer, **a** is two dimensional array with n rows and n+1 coulumn defined as double and **x** is the unknown variable arrays which we want to calculate defined as double.

PROBLEM 3 (a) : Chopping - Rounding - Partial pivoting

In [ ]:

```
// problem 3a
// make sure "arithmetic.c" file is in the same directory
#include<stdio.h>
#include<math.h>
#include"arithmetic.c"

int main()
{
    int i,n=2;  // no of equations
    //Declare a matrix to store the user given matrix
    double a[2][3]={
                        {58.9,0.03,59.2},
                        {-6.10,5.31,47.0}
                    };
    //Declare an array to store the solution of equations
    double x[n];
    printf("The augmented matrix for part a:\n");
    printmat(n,a);  // augmented matrix
    printf("\n(i) Gauss Elimination with three digit chopping:\n");
    gausschopping(n,a,x);   // for three digit chopping
```

```
    printf("\n\n");
    printf("(ii) Gauss Elimination with three digit rounding:\n");
    gaussround(n,a,x);   // for three digit rounding
    printf("\n\n");
    printf("(iii) Gauss Elimination with partial pivoting:\n");
    gausspivot(n,a,x);   // for partial pivoting
}
```

OUTPUT:

The augmented matrix for part a:

58.900000 0.030000 59.200000

-6.100000 5.310000 47.000000

(i) Gauss Elimination with three digit chopping:

The upper triangular matrix:

58.900000 0.030000 59.200000

0.025000 5.313000 53.156000

The solution of linear equations:

x[1]= 0.999000

x[2]= 10.004000

(ii) Gauss Elimination with three digit rounding:

The upper triangular matrix:

58.900000 0.030000 59.200000

0.025000 5.313000 53.156000

The solution of linear equations:

x[1]= 1.000000

x[2]= 10.005000

(iii) Gauss Elimination with partial pivoting:

The upper triangular matrix:

58.900000 0.030000 59.200000

0.000000 5.312987 53.130873

The solution of linear equations:

x[1]= 1.00

x[2]= 10.00

PROBLEM 3 (b) : Chopping - Rounding - Partial pivoting

In [ ]:

```
// problem 3b
// make sure "arithmetic.c" file is in the same directory
#include<stdio.h>
#include<math.h>
#include"arithmetic.c"

int main()
```

```
{
    int i,n=3;  // no of equations
    //Declare a matrix to store the user given matrix
    double a[3][4]={
                        {3.3330,15920,10.333,7953},
                        {2.2220,16.710,9.6120,0.965},
                        {-1.5611,5.1792,-1.6855,2.714}
                    };
    //Declare an array to store the solution of equations
    double x[n];
    printf("The augmented matrix for part a:\n");
    printmat(n,a);  // augmented matrix
    printf("\n(i) Gauss Elimination with three digit chopping:\n");
    gausschopping(n,a,x);   // for three digit chopping
    printf("\n\n");
    printf("(ii) Gauss Elimination with three digit rounding:\n");
    gaussround(n,a,x);   // for three digit rounding
    printf("\n\n");
    printf("(iii) Gauss Elimination with partial pivoting:\n");
    gausspivot(n,a,x);   // for partial pivoting
}
```

OUTPUT:

The augmented matrix for part a:

3.333000 15920.000000 10.333000 7953.000000

2.222000 16.710000 9.612000 0.965000

-1.561100 5.179200 -1.685500 2.714000

(i) Gauss Elimination with three digit chopping:

The upper triangular matrix:

3.333000 15920.000000 10.333000 7953.000000

0.002000 -10586.011000 2.730000 -5295.733000

0.003000 -2.065000 5.087000 -6.117000

The solution of linear equations:

x[1]= 6.405000

x[2]= 0.499000

x[3]= -1.203000

(ii) Gauss Elimination with three digit rounding:

The upper triangular matrix:

3.333000 15920.000000 10.333000 7953.000000

-0.001000 -10601.931000 2.720000 -5303.686000

0.000000 3.219000 5.072000 -3.463000

The solution of linear equations:

x[1]= 0.017000

x[2]= 0.500000

x[3]= -0.683000

(iii) Gauss Elimination with partial pivoting:

The upper triangular matrix:

3.333000 15920.000000 10.333000 7953.000000

0.000000 -10597.154522 2.723100 -5301.299861

0.000000 0.000000 5.072827 -5.073327

The solution of linear equations:

x[1]= 1.00

x[2]= 0.50

x[3]= -1.00

PROBLEM 4 : A biological system having n species of animals and m sources of food.

In [ ]:

```
// problem 4
// biological system there are n species of animals and m sources of food
#include <stdio.h>
#include <math.h>

// product of matrix Cmn = Amo X bom
void PROD(int m,int o,int n,int a[m][o],int b[o][n],int c[m][n])
{
    for(int i=0;i<m;++i)
    {
        for(int j=0;j<n;++j)
        {
            c[i][j]=0;
            for(int k=0;k<o;++k)
            {
                c[i][j]=c[i][j]+a[i][k]*b[k][j];
            }
        }
    }
}
void species(int A[3][4],int X[4][1], int Bi[3][1], int index)
{
    int Xj[4][1];
    // copying X elements into Xj
    for (int x=0;x<4;++x)
    {
        for (int y=0;y<4;++y)
        {
            Xj[x][y]=X[x][y];
        }
    }
    int i,j,k,n=0,flag=0;
    int AX[3][1];
    do
    {
        Xj[index][0]++;
        n++;
        PROD(3,4,1,A,Xj,AX);
        for(i=0;i<3;++i)
        {
            if(AX[i][0]>Bi[i][0]){
            flag=1;
            n--;
            break;
```

```
            }
        }
    }while(flag==0);
    printf("For species %d maximum can be added: %d\n",(index+1), n);
}

int main()
{
    // given initial conditions for our system
    int A[3][4]={{1,2,0,3},{1,0,2,2},{0,0,1,1}};
    int B[3][1]={{3500},{2700},{900}};
    int X[4][1]={{1000},{500},{350},{400}};

    printf("(a) Is there sufficient food to satisfy the average daily consumptio
    int ax[3][1];
    PROD(3,4,1,A,X,ax);
    if (ax<B){printf("Yes. There is sufficient food.\n");}
        else{printf("No. There isn't sufficient food.\n");}

    printf("\n");

    printf("(b) Individual increase of each species satisfying food supply:\n");
    species(A,X,B,0); //for species 1
    species(A,X,B,1); //for species 2
    species(A,X,B,2); //for species 3
    species(A,X,B,3); //for species 4

    printf("\n");

    printf("(c) Individual increase of each species when 1 is extinct:\n");
    int Xc[4][1]={{0},{500},{350},{400}};
    species(A,Xc,B,1); //for species 2
    species(A,Xc,B,2); //for species 3
    species(A,Xc,B,3); //for species 4

    printf("\n");

    printf("(d) Individual increase of each species when 2 is extinct:\n");
    int Xd[4][1]={{1000},{0},{350},{400}};
    species(A,Xd,B,0); //for species 1
    species(A,Xd,B,2); //for species 3
    species(A,Xd,B,3); //for species 4
}
```

OUTPUT:

(a) Is there sufficient food to satisfy the average daily consumption?

Yes. There is sufficient food.

(b) Individual increase of each species satisfying food supply:

For species 1 maximum can be added: 200

For species 2 maximum can be added: 150

For species 3 maximum can be added: 100

For species 4 maximum can be added: 100

(c) Individual increase of each species when 1 is extinct:

For species 2 maximum can be added: 650

For species 3 maximum can be added: 150

For species 4 maximum can be added: 150

(d) Individual increase of each species when 2 is extinct:

For species 1 maximum can be added: 200

For species 3 maximum can be added: 100

For species 4 maximum can be added: 100

PROBLEM 5 : Gauss Jordan Method

In [ ]:

```c
// problem 5
// make sure "linear.c" file is in the same directory
#include<stdio.h>
#include<math.h>
#include"linear.c"
int main()
{
    int i,n=4;  // no of equations
    //Declare a matrix to store the user given matrix
    double a[4][5]={
                        {1,-1,2,-1,-8},
                        {2,-2,3,-3,-20},
                        {1,1,1,0,-2},
                        {1,-1,4,3,4}
                    };
    //Declare an array to store the solution of equations
    double x[n];
    printf("The augmented matrix:\n");
    printmat(n,a);  // printing the augmented matrix
    gaussjordan(n,a,x);   // Perform Gauss jordan
}
```

OUTPUT:

The augmented matrix:

1.00 -1.00 2.00 -1.00 -8.00

2.00 -2.00 3.00 -3.00 -20.00

1.00 1.00 1.00 0.00 -2.00

1.00 -1.00 4.00 3.00 4.00

The diagonal augmented matrix:

2.00 0.00 0.00 0.00 -14.00

0.00 2.00 0.00 0.00 6.00

0.00 0.00 2.50 0.00 5.00

0.00 0.00 0.00 -0.40 -0.80

The solution of linear equations:

x[1]= -7.00

x[2]= 3.00

x[3]= 2.00

x[4]= 2.00

Inverse of a matrix:

Say we have a square matrix A which is non-singular (means the Determinant of A is nonzero); Then there exists a matrix A^{-1} which is called inverse of matrix A. The inverse of a matrix is only possible when it holds the following arguments:

1. The matrix must be a square matrix.
2. The matrix must be a non-singular matrix.
3. There exist an Identity matrix I for which $AA^{-1} = A^{-1}A = I$

We are doing almost the same thing in the gauss jordan method and below is the program to calculate the inverse of a matrix.

PROBLEM 6 (a) and 6 (b) : Inverse of a matrix

In [ ]:

```c
// program for inverse of a matrix using gauss jordan method.
#include<stdio.h>
#include<math.h>

int main()
{
    int i,j,k,n;
    float a[10][10]={0},temp;
    printf("Enter order of matrix:");
    scanf("%d",&n);
    printf("Enter the elements of Matrix:\n");
    for(i=1;i<=n;i++)
    {
        for(j=1;j<=n;j++)
            scanf("%f",&a[i][j]);
    }
    //displaying the augmented Matrix
    printf("\nThe augmented matrix:\n");
    for (i=1;i<=n;i++){
        for (j=1;j<=2*n;j++){
            if (j==(i+n))
                a[i][j]=1;
            printf("%.2lf\t",a[i][j]);
        }
        printf("\n");
    }
    //partial pivoting
    for (i=n;i>1;i--)
    {
        if (a[i-1][1]<a[i][1])
            //Swap the rows in the matrix
            for (j=1;j<=2*n;j++)
            {
                temp=a[i][j];
                a[i][j]=a[i-1][j];
                a[i-1][j]=temp;
            }
    }
    //Begin the Gauss Jordan
    for (i=1;i<=n;i++)
    {
        for (j=1;j<=2*n;j++)
            if (j!=i)
            {
```

```
                temp=a[j][i]/a[i][i];
                for (k=1;k<=2*n;k++)
                    a[j][k]=a[j][k]-temp*a[i][k];
            }
    }
    // checking if the Ann value is 0 or nan(not a number)
    if (a[n][n]==0 || isnan(a[n][n])){
        printf("\nSorry! Inverse matrix doesn't exist.");
    }
    else{
        //reducing to unit matrix
        for (i=1;i<=n;i++)
        {
            temp=a[i][i];
            for (j=1;j<=2*n;j++)
                a[i][j]=a[i][j]/temp;
        }
        //displaying Inverse Matrix
        printf("\nUsing Gauss Jordan Inverse of matrix:\n");
        for(i=1;i<=n;i++)
        {
            for(j=n+1;j<=2*n;j++)
            {
                printf("%0.3f\t",a[i][j]);
            }
        printf("\n");
        }
    }
}
```

OUTPUT:

./a.out

Enter order of matrix:3

Enter the elements of Matrix:

1 2 -1

2 1 0

-1 1 2

The augmented matrix:

1.00 2.00 -1.00 1.00 0.00 0.00

2.00 1.00 0.00 0.00 1.00 0.00

-1.00 1.00 2.00 0.00 0.00 1.00

Using Gauss Jordan Inverse of matrix:

-0.222 0.556 -0.111

0.444 -0.111 0.222

-0.333 0.333 0.333

./a.out

Enter order of matrix:4

Enter the elements of Matrix:

1 1 -1 1

1 2 -4 -2

2 1 1 5
-1 0 -2 -4

The augmented matrix:
1.00 1.00 -1.00 1.00 1.00 0.00 0.00 0.00
1.00 2.00 -4.00 -2.00 0.00 1.00 0.00 0.00
2.00 1.00 1.00 5.00 0.00 0.00 1.00 0.00
-1.00 0.00 -2.00 -4.00 0.00 0.00 0.00 1.00

Sorry! Inverse matrix doesn't exist.

Comment on Inverse matrix: In the Gauss Jordan method we make all elements to be zero except the diagonal and this really helped to easily add the condition if the inverse matrix doesn't exists. For non singular matrix the determinant is zero which implies one of row must be zero. Since we already made the elements zero in the last row except the last element and that element has the power to decide the fate of matrix. And luckily it worked for every test matrix except *in our problem 2nd matrix which doesn't have the inverse and challenged this argument. To troubleshoot this issue when I printed the reduced matrix using gauss elimination the third row had already become zero consequently fourth row got nan values (not a number). So I had to add* `isnan(a[n][n]` *too in the condition.*