Birla Institute of Technology
Mesra, Ranchi India

NUMERICAL METHOD LAB

NM204

Report Manual

Prepared By: Robin Roy
Roll No. BTECH/15114/19

Department of Electrical & Electronics Engineering

Patna Off-Campus

2021

EXP-1 GAUSS ELIMINATION METHOD

ALGORITHM:

1. Start the program
2. Read the order of the matrix n
3. Read the elements of the augmented matrix
4. For j=1 to j<=n and for i=1 to i<=n; if (i>j) then c=a[i][j]/a[j][j]
5. For k=1 to k<=n+1 ,a[i][k]=a[i][k]-c*a[j][k] ,k=k+1
6. Compute x[n]=a[n][n+1]/a[n][n]
7. For i=n-1 to i>=1 sum=0
8. For j=i+1 to j<=n, sum=sum+a[i][j]*x[j]
9. Compute x[i]=(a[i][n+1]-sum)/a[i][i]
10. For i=1 to i<=n
11. 11 display the result x[i];i=i+1
12. stop

CODE:

```
#include<stdio.h>
#include<conio.h>
void main()
{
int i,j,k,n;
float a[20][20],c,x[10],sum=0.0;
clrscr();
printf("\n enter the order of matrix:");
scanf("%d",&n);
printf("\n Enter the elememts of augmented matrix row-wise:\n\n");
for(i=0;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++)
{
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];
```

```
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("\n the solution is:\n");
for(i=1;i<=n;i++)
{
printf("\n x%d=%f\t",i,x[i]);
}
getch();
}
```

OUTPUT:

```
 enter the order of matrix:3

 Enter the elememts of augmented matrix row-wise:

a[1][1]:2
a[1][2]:2
a[1][3]:1
a[1][4]:6
a[2][1]:4
a[2][2]:2
a[2][3]:3
a[2][4]:4
a[3][1]:1
a[3][2]:-1
a[3][3]:1
a[3][4]:0

 the solution is:

 x1=9.000000
 x2=-1.000000
 x3=-10.000000
```

EXP-2 GAUSS JORDAN METHOD

ALGORITHM:

1. start
2. read the order pf matrix n
3. read the elements of augmented matrix
4. for j=1;j<=n and for i=1 to i<=n; repeat steps 5 to 10.
5. If(i!=j) then compute c=a[i][j]/a[i][i]
6. For k=1 to k<=n+1
7. Compute a[i][k]=a[i][k]-c*a[j][k]
8. K=k+1
9. I=i+1
10. J=j+1
11. For i=1 to i<=n repeat step 12 and 13
12. Compute x[i]=a[i][n+1]/a[i][i]
13. Print the solution is x[i], i=i+2
14. Stop

CODE:

```
#include<stdio.h>
#include<conio.h>
void main()
{
int i,j,k,n;
float a[20][20],c,x[10];
clrscr();
printf("\n enter the size of matrix:");
scanf("%d",&n);
printf("\n enter the elements of augmented matrix:\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++)
{
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];
```

```
}
}
}
printf("\n the solution is :\n");
for(i=1;i<=n;i++)
{
x[i]=a[i][n+1]/a[i][i];
printf("\n x%d=%f\n",i,x[i]);
}
}
getch();
}
```

OUTPUT:

```
 enter the size of matrix:3

 enter the elements of augmented matrix:
a[1][1]:10
a[1][2]:1
a[1][3]:1
a[1][4]:12
a[2][1]:2
a[2][2]:10
a[2][3]:1
a[2][4]:13
a[3][1]:1
a[3][2]:1
a[3][3]:5
a[3][4]:7_

 the solution is :

 x1=1.000000

 x2=1.000000

 x3=1.000000
```

Exp-3 Gauss Jacobi Method

Algorithm:

1. Start

2. Read Number of Unknowns: n

3. Read Augmented Matrix (A) of n by n+1 Size

4. Transform Augmented Matrix (A)
   to Diagonal Matrix by Row Operations.

5. Obtain Solution by Making All Diagonal Elements to 1.

6. Display Result.

7. Stop

CODE:

```
#include<stdio.h>
#include<conio.h>
#include<math.h>

#define SIZE 10

int main()
{
		float a[SIZE][SIZE], x[SIZE], ratio;
		int i,j,k,n;
		clrscr();

		printf("Enter number of unknowns: ");
		scanf("%d", &n);

		printf("Enter coefficients of Augmented Matrix:\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(i=1;i<=n;i++)
		{
			if(a[i][i] == 0.0)
			{
				printf("Mathematical Error!");
				break;
			}
```

```
			for(j=1;j<=n;j++)
			{
				if(i!=j)
				{
					ratio = a[j][i]/a[i][i];
					for(k=1;k<=n+1;k++)
					{
						a[j][k] = a[j][k] - ratio*a[i][k];
					}
				}
			}
		}

		for(i=1;i<=n;i++)
		{
			x[i] = a[i][n+1]/a[i][i];
		}

		printf("\nSolution:\n");
		for(i=1;i<=n;i++)
		{
			printf("x[%d] = %0.3f\n",i, x[i]);
		}
		getch();
		return(0);
}
```

OUTPUT:

```
Enter number of unknowns: 3
Enter coefficients of Augmented Matrix:
a[1][1] = 27
a[1][2] = 6
a[1][3] = -1
a[1][4] = 85
a[2][1] = 6
a[2][2] = 15
a[2][3] = 2
a[2][4] = 72
a[3][1] = 1
a[3][2] = 1
a[3][3] = 54
a[3][4] = 110

Solution:
x[1] = 2.425
x[2] = 3.573
x[3] = 1.926
_
```

EXP-4 Gauss Siedal Method

Algorithm:

1. Start
2. Arrange given system of linear equations in
   diagonally dominant form
3. Read tolerable error (e)
4. Convert the first equation in terms of first variable,
second equation in terms of second variable and so on.
5. Set initial guesses for x0, y0, z0 and so on
6. Substitute value of y0, z0 ... from step 5 in
   first equation obtained from step 4 to calculate
   new value of x1. Use x1, z0, u0 .... in second equation
   obtained from step 4 to caluclate new value of y1.
   Similarly, use x1, y1, u0... to find new z1 and so on.
7. If| x0 - x1| > e and | y0 - y1| > e and | z0 - z1| > e
   and so on then goto step 9
8. Set x0=x1, y0=y1, z0=z1 and so on and goto step 6
9. Print value of x1, y1, z1 and so on
10. Stop

CODE:

```
#include<stdio.h>
#include<math.h>
#include<conio.h>
int main()
{
    int count, t, limit;
    float temp, error, a, sum = 0;
    float matrix[10][10], y[10], allowed_error;
    clrscr();
    printf("\nEnter the Total Number of Equations:\t");
    scanf("%d", &limit);
    printf("Enter Allowed Error:\t");
    scanf("%f", &allowed_error);
    printf("\nEnter the Co-Efficients\n");
    for(count = 1; count <= limit; count++)
    {
        for(t = 1; t <= limit + 1; t++)
        {
            printf("Matrix[%d][%d] = ", count, t);
            scanf("%f", &matrix[count][t]);
        }
    }
```

```
    for(count = 1; count <= limit; count++)
    {
        y[count] = 0;
    }
    do
    {
        a = 0;
        for(count = 1; count <= limit; count++)
        {
            sum = 0;
            for(t = 1; t <= limit; t++)
            {
                if(t != count)
                {
                    sum = sum + matrix[count][t] * y[t];
                }
            }
            temp = (matrix[count][limit + 1] - sum) / matrix[count][count];
            error = fabs(y[count] - temp);
            if(error > a)
            {
                a = error;
            }
            y[count] = temp;
            printf("\nY[%d]=\t%f", count, y[count]);
        }
        printf("\n");
    }
    while(a >= allowed_error);
    printf("\n\nSolution\n\n");
    for(count = 1; count <= limit; count++)
    {
        printf("\nY[%d]:\t%f", count, y[count]);
    }
   getch();
   return 0;
}
```

OUTPUT:

```
Enter the Total Number of Equations:    3
Enter Allowed Error:    0.0001

Enter the Co-Efficients
Matrix[1][1] = 20
Matrix[1][2] = 1
Matrix[1][3] = -2
Matrix[1][4] = 17
Matrix[2][1] = 3
Matrix[2][2] = 20
Matrix[2][3] = -1
Matrix[2][4] = -18
Matrix[3][1] = 2
Matrix[3][2] = -3
Matrix[3][3] = 20
Matrix[3][4] = 25_
```

```
Matrix[3][4] = 25

Y[1]=   0.850000
Y[2]=   -1.027500
Y[3]=   1.010875

Y[1]=   1.002463
Y[2]=   -0.999826
Y[3]=   0.999780

Y[1]=   0.999969
Y[2]=   -1.000006
Y[3]=   1.000002

Y[1]=   1.000000
Y[2]=   -1.000000
Y[3]=   1.000000


Solution


Y[1]:   1.000000
Y[2]:   -1.000000
Y[3]:   1.000000_
```