BMS INSTITUTE OF TECHNOLOGY & MANAGEMENT

BENGALURU-560064

DEPARTMENT OF MATHEMATICS

ADVANCED CALCULUS AND NUMERICAL METHODS (18MAT21)

Activity Based Learning

GAUSS SEIDEL METHOD OF SOLVING SIMULTANEOUS LINEAR EQUATIONS

AND

NEWTON'S FORWARD INTERPOLATION FORMULA

1BY20ET048_S VARSHA

Course coordinator

Dr.Chetan A S

Professor

Department of Mathematics

BMSIT & M

BMS INSTITUTE OF TECHNOLOGY AND MANAGEMENT

YELAHANKA,BENGALURU-560064

DEPARTMENT OF MATHEMATICS

CERTIFICATE

This is to certify that the Mathematics activity based learning on “Gauss Seidel Method of solving linear equations” and “Newton’s Forward Interpolation Formula” has been carried out by **1BY20ET048_S VARSHA,** in association with the team members

1BY20ET053_SHARMILA S

1BY20ET035_MEGHANA A

during the academic year 2020-21.

Course coordinator

Dr.Chetan A S

TABLE OF CONTENTS

GAUSS SEIDEL METHOD

ABSTRACT

Gauss Seidel Method of solving simutaneous linear equations :

- *In numerical linear algebra, the Gauss–Seidel method, also known as the Liebmann method or the method of successive displacement, is an iterative method used* ***to solve a system of linear equations.***

- *It is named after the German mathematicians Carl Friedrich Gauss and Philipp Ludwig von Seidel, and it is similar to the Jacobi method.*

- *Though it can be applied to any matrix with non-zero elements on the diagonals, convergence is only guaranteed if the matrix is either strictly diagonally dominant, or symmetric and positive definite.*

- *Diagonally dominant matrix : A square matrix is said to be diagonally dominant if, for every row of matrix,* ***the magnitude of the diagonal entry in a row is larger than or equal to the sum of the magnitudes*** *of all other(non diagonal) entries of that row.*

- *The element-wise formula for the Gauss–Seidel method is extremely similar to that of the Jacobi method. The difference between the Gauss–Seidel and Jacobi methods is that the Jacobi method uses the values obtained from the previous step while the Gauss–Seidel method always applies the latest updated values during the iterative procedures.*

Advantages of Gauss–Seidel Method:

- *The method is very simple in calculations and thus programming is easier.*
- *The storage needed in the computer memory is relatively less .*

APPLICATION OF GAUSS SEIDEL METHOD

- *Ground water flow analysis*
- *Making climate models*
- *Used to calculate the value of unknown voltages*
- *Used to study load flow or power flow*
- *To obtain solution of the system of thermal radiation transfer equations for absorbing, radiating, and scattering media*

SOFTWARE/LANGUAGE USED

- *We have used C language to illustrate the Gauss Seidel Method to solve system of linear equations.*

- *C computer programming language was developed in the early 1970s by American computer scientist Dennis M. Ritchie.*

- *C is a general purpose, imperative procedural computer programming language supporting structured programming.*

- *It allows complex program to be broken into simpler program called functions. It allows free moment of data across these functions.*

- *Many of the ideas of C language were derived from B language, hence the name C .*

- *It has a powerful library that provides several built-in functions and it also gives dynamic memory allocation.*

- *It can efficiently work on enterprise applications, games, graphics and application requiring calculations etc.*

ANALYTICAL METHOD

Consider a system of equations :

$$a_{11}x + a_{12}y + a_{13}z = b_1$$

$$a_{21}x + a_{22}y + a_{23}z = b_2$$

$$a_{31}x + a_{32}y + a_{33}z = b_3$$

step 1 *: Rewrite the given system as*

$$x = (b_1 - a_{12}y - a_{13}z)/a_{11}$$

$$y = (b_2 - a_{21}x - a_{23}z)/a_{22}$$

$$z = (b_3 - a_{31}x - a_{32}z)/a_{33}$$

step 2 *: Take the initial approximation as*

$$x^{(0)} = y^{(0)} = z^{(0)} = 0$$

step 3 *: Use the initial approximation to obtain the next iteration.*

NOTE*: Use the latest updated values during the iterative procedures.*

Continue this process till any two consecutive approximations are almost same.

EXAMPLE 01: *Solve* $10x + y + z = 12$

$$2x + 10y + z = 13$$

$$2x + 2y + 10z = 14$$

Solution : $x = (12-y-z)/10$

$$y = (13-2x-z)/10$$

$$z = (14-2x-2y)/10$$

Let $x^{(0)} = y^{(0)} = z^{(0)}=0$

The 1st approximation is

$$x^{(1)} = (12-y^{(0)}-z^{(0)})/10 = 1.2000$$

$$y^{(1)}= (13-2x^{(1)}-z^{(0)})/10 = 1.0600$$

$$z^{(1)} = (14-2x^{(1)}-2y^{(1)})/10=0.9480$$

The 2nd approximation is

$$x^{(2)} = (12-y^{(1)}-z^{(1)})/10 =0.9992$$

$$y^{(2)}= (13-2x^{(2)}-z^{(1)})/10 =1.0054$$

$$z^{(2)}= (14-2x^{(2)}-2y^{(2)})/10 =0.9991$$

The 3rd approximation is

$$x^{(3)} = (12 -y^{(2)}-z^{(2)})/10 =0.9996$$

$$y^{(3)}= (13-2x^{(3)}-z^{(2)})/10 =1.0002$$

$$z^{(3)} = (14-2x^{(3)}-2y^{(3)})/10 =1.0000$$

The 4th approximation is

$$x^{(4)} = (12 -y^{(3)}-z^{(3)})/10 =1.0000$$

$$y^{(4)} = (13-2x^{(4)}-z^{(3)})/10 =1.0000$$

$$z^{(4)} = (14-2x^{(4)}-2y^{(4)})/10 =1.0000$$

The 5th approximation is

$$x^{(5)} = (12 -y^{(4)}-z^{(4)})/10 =1.0000$$

$$y^{(5)} = (13-2x^{(5)}-z^{(4)})/10 =1.0000$$

$$z^{(5)}= (14-2x^{(5)}-2y^{(5)})/10 =1.0000$$

Therefore, $\underline{\underline{x = 1, y = 1, z = 1}}$

EXAMPLE 02 *: Solve* $x + 2y + 5z = 20$

$$5x + 2y + z = 12$$

$$x + 4y + 2z = 15$$

perform 4 iterations by taking initial approximation as (1, 0, 3)

Solution : *given :* $5x + 2y + z = 12$

$$x + 4y + 2z = 15$$

$$x + 2y + 5z = 20$$

$x^0 = 1, y^0 = 0, z^0 = 3$

The 1st approximation is

$$x^{(1)} = (12-2y^{(0)}-z^{(0)})/5=1.8000$$

$$y^{(1)}=(15-x^{(1)}-2z^{(0)})/4=1.8000$$

$$z^{(1)}=(20-x^{(1)}-2y^{(1)})/5=2.9200$$

The 2^nd^ approximation is

$$x^{(2)} = (12-2y^{(1)}-z^{(1)})/5=1.0960$$

$$y^{(2)}=(15-x^{(2)}-2z^{(1)})/4=2.0160$$

$$z^{(2)}=(20-x^{(2)}-2y^{(2)})/5=2.9744$$

The 3^rd^ approximation is

$$x^{(3)} = (12-2y^{(2)}-z^{(2)})/5=0.9987$$

$$y^{(3)}=(15-x^{(3)}-2z^{(2)})/4=2.0131$$

$$z^{(3)}=(20-x^{(3)}-2y^{(3)})/5=2.9950$$

The 4^th^ approximation is

$$x^{(4)} = (12-2y^{(3)}-z^{(3)})/5=0.9958$$

$$y^{(4)}=(15-x^{(4)}-2z^{(3)})/4=2.0036$$

$$z^{(4)}=(20-x^{(4)}-2y^{(4)})/5=2.9999$$

SOURCE CODE

```
#include <stdio.h>
int main()
{
   int i,j,m=3;
   float a[100][100],x[100],y[100],z[100],b[100];

   printf("\n\n\tGAUSS SEIDEL METHOD OF SOLVING LINEAR SYSTEM OF 3 EQUATIONS\n\n");
   start:
   printf("   Enter coefficients of x,y,z of a diagonally dominant system\n");
   for(i=1;i<=m;i++)
  {
      for(j=1;j<=m;j++)
     {
        printf("\ta[%d][%d] = ",i,j);
        scanf("%f",&a[i][j]);
     }
   }

   for(i=1;i<=m;i++)
   {
    if(a[i][i]!=0)
     {
        if(a[i][i] > a[i][i+2] && a[i][i+2])
         {
           printf("   Enter constants of each equation\n");
           for(i=1;i<=m;i++)
            {
               printf("\tb[%d] = ",i);
               scanf("%f",&b[i]);
            }
           printf("   Enter initial approximations\n");
           printf("\tx[0] = ");
           scanf("%f",&x[0]);
           printf("\ty[0] = ");
```

```
        scanf("%f",&y[0]);
        printf("\tz[0] = ");
        scanf("%f",&z[0]);
        for(i=1;i<100;i++)
         {
           printf("   The %d approximation is\n",i);

           x[i]=((b[1]-a[1][2]*y[i-1]-a[1][3]*z[i-1])/a[1][1]);
           printf("\tx[%d] = %.10f\n",i,x[i]);

           y[i]=((b[2]-a[2][1]*x[i]-a[2][3]*z[i-1])/a[2][2]);
           printf("\ty[%d] = %.10f\n",i,y[i]);

           z[i]=((b[3]-a[3][1]*x[i]-a[3][2]*y[i])/a[3][3]);
           printf("\tz[%d] = %.10f\n",i,z[i]);

           if(x[i-1]==x[i] && y[i-1]==y[i] && z[i-1]==z[i])
            {
              printf("   The solutions of the equations are\n");
              printf("\tx = %.4f\n",x[i]);
              printf("\ty = %.4f\n",y[i]);
              printf("\tz = %.4f\n",z[i]);
              break;
            }  }  }
      else
       {
        printf("\n   Coefficients are not diagonally dominant\n\tRENTER\n\n");
        goto start;
        }
     }
     else
    {
       printf("\n   Coefficients are not diagonally dominant\n\tRENTER\n\n");
       goto start;
     }
  }
}
```

EXECUTION

Output of Example 01 :

```
      GAUSS SEIDEL METHOD OF SOLVING LINEAR SYSTEM OF 3 EQUATIONS

 Enter coefficients of x,y,z of a diagonally dominant system
      a[1][1] = 10
      a[1][2] = 1
      a[1][3] = 1
      a[2][1] = 2
      a[2][2] = 10
      a[2][3] = 1
      a[3][1] = 2
      a[3][2] = 2
      a[3][3] = 10
 Enter constants of each equation
      b[1] = 12
      b[2] = 13
      b[3] = 14
 Enter initial approximations
      x[0] = 0
      y[0] = 0
      z[0] = 0
 The 1 approximation is
      x[1] = 1.2000000477
      y[1] = 1.0600000620
      z[1] = 0.9480000734
 The 2 approximation is
      x[2] = 0.9991999865
      y[2] = 1.0053600073
      z[2] = 0.9990879893
 The 3 approximation is
      x[3] = 0.9995552301
      y[3] = 1.0001801252
      z[3] = 1.0000529289
 The 4 approximation is
      x[4] = 0.9999767542
      y[4] = 0.9999994040
      z[4] = 1.0000047684
 The 5 approximation is
      x[5] = 0.9999996424
      y[5] = 0.9999996424
      z[5] = 1.0000002384
 The 6 approximation is
      x[6] = 1.0000000000
      y[6] = 1.0000000000
      z[6] = 1.0000000000
 The 7 approximation is
      x[7] = 1.0000000000
      y[7] = 1.0000000000
      z[7] = 1.0000000000
```

```
 The solutions of the equations are
      x = 1.0000
      y = 1.0000
      z = 1.0000


...Program finished with exit code 0
Press ENTER to exit console.
```

Output of Example 02 :

```
     GAUSS SEIDEL METHOD OF SOLVING LINEAR SYSTEM OF 3 EQUATIONS

 Enter coefficients of x,y,z of a diagonally dominant system
     a[1][1] = 5
     a[1][2] = 2
     a[1][3] = 1
     a[2][1] = 1
     a[2][2] = 4
     a[2][3] = 2
     a[3][1] = 1
     a[3][2] = 2
     a[3][3] = 5
 Enter constants of each equation
     b[1] = 12
     b[2] = 15
     b[3] = 20
 Enter initial approximations
     x[0] = 1
     y[0] = 0
     z[0] = 3
 The 1 approximation is
     x[1] = 1.7999999523
     y[1] = 1.7999999523
     z[1] = 2.9200000763
```

```
 The 2 approximation is
     x[2] = 1.0959999561
     y[2] = 2.0160000324
     z[2] = 2.9744000435
 The 3 approximation is
     x[3] = 0.9987199903
     y[3] = 2.0131199360
     z[3] = 2.9950079918
 The 4 approximation is
     x[4] = 0.9957504272
     y[4] = 2.0035583973
     z[4] = 2.9994266033
 The 5 approximation is
     x[5] = 0.9986913800
     y[5] = 2.0006136894
     z[5] = 3.0000162125
 The 6 approximation is
     x[6] = 0.9997512698
     y[6] = 2.0000541210
     z[6] = 3.0000278950
 The 7 approximation is
     x[7] = 0.9999727011
     y[7] = 1.9999929667
     z[7] = 3.0000081062
```

```
   The 8 approximation is
        x[8] = 1.0000011921
        y[8] = 1.9999957085
        z[8] = 3.0000014305
   The 9 approximation is
        x[9] = 1.0000014305
        y[9] = 1.9999988079
        z[9] = 3.0000000000
   The 10 approximation is
        x[10] = 1.0000003576
        y[10] = 2.0000000000
        z[10] = 3.0000000000
   The 11 approximation is
        x[11] = 1.0000000000
        y[11] = 2.0000000000
        z[11] = 3.0000000000
   The 12 approximation is
        x[12] = 1.0000000000
        y[12] = 2.0000000000
        z[12] = 3.0000000000
   The solutions of the equations are
        x = 1.0000
        y = 2.0000
        z = 3.0000


...Program finished with exit code 0
Press ENTER to exit console.
```

ABSTRACT

***INTERPOLATION* :**

- *Interpolation is a method of constructing new data points within the range of a discrete set of known data points.*

Newton's Forward Interpolation :

- *Newton's forward difference interpolation is used when the function is tabulated at equal intervals.*

- *If the data point to be interpolated lies in the upper half or in the beginning of the table then Newton's Forward difference interpolation is used because it gives better approximation.*

- ***Formula for NFIF is given by :***

$y = y_0 + p\Delta y_0 + ((p(p-1))/2!)*\Delta^2 y_0 + ((p(p-1)(p-2))/3!)*\Delta^3 y_0 + \ldots\ldots\ldots$

where $p = (x-x_0)/h$

h is the interval between data points

Δ is the forward difference symbol

- ***Formation of Forward difference table** :*

Let us consider set of data points,

X	x_0	x_1	x_2	x_3	x_4	x_5
Y	y_0	y_1	y_2	y_3	y_4	y_5

The forward difference table for the above set of point is given by

x	y	Δy	$\Delta^2 y$	$\Delta^3 y$	$\Delta^4 y$	$\Delta^5 y$
x0	y0	$\Delta y0 = y1 - y0$	$\Delta^2 y0 = \Delta y1 - \Delta y0$	$\Delta^3 y0 = \Delta^2 y1 - \Delta^2 y0$	$\Delta^4 y0 = \Delta^3 y1 - \Delta^3 y0$	$\Delta^5 y0 = \Delta^4 y1 - \Delta^4 y0$
x1	y1	$\Delta y1 = y2 - y1$	$\Delta^2 y1 = \Delta y2 - \Delta y1$	$\Delta^3 y1 = \Delta^2 y2 - \Delta^2 y1$	$\Delta^4 y1 = \Delta^3 y2 - \Delta^3 y1$	
x2	y2	$\Delta y2 = y3 - y2$	$\Delta^2 y2 = \Delta y3 - \Delta y2$	$\Delta^3 y2 = \Delta^2 y3 - \Delta^2 y2$		
x3	y3	$\Delta y3 = y4 - y3$	$\Delta^2 y3 = \Delta y4 - \Delta y3$			
x4	y4	$\Delta y4 = y5 - y4$				
x5	y5					

APPLICATIONS

- *To add new data to the data set, to create a new interpolating polynomial without recalculating the old coefficients.*

- *Data Science application: In spatial and time series data it can be used for missing value treatments.*

ANALYTICAL METHOD

Let y_0, y_1, y_2….yn be the values of a function y=f(x) corresponding to the equidistant values $x_0, x_1, \ldots, x_n$ of x with step length h respectively

Then the value of y at any point is given by

$y = y_0 + p\Delta y_0 + ((p(p-1))/2!)*\Delta^2 y_0 + ((p(p-1)(p-2))/3!)*\Delta^3 y_0 + \ldots\ldots\ldots$

where $p=(x-x_0)/h$

$\Delta y = y_1 - y_0$

$\Delta_2 y = y_2 - y_1$

$\vdots$

$\Delta_n y = y_{(n+1)} - y_{(n)}$

Δ is called forward difference operator

$\Delta_n y$ are called forward differences

EXAMPLE 01:

Compute f(x) = e^X for x=0.02 from table

x	0	0.1	0.2	0.3	0.4
y	1	1.1052	1.2214	1.3499	1.4918

x	y	Δy	$\Delta^2 y$	$\Delta^3 y$	$\Delta^4 y$
0	1	0.1052	0.0110	0.0013	-0.0002
0.1	1.1052	0.1162	0.0123	0.0011	
0.2	1.2214	0.1285	0.0134		
0.3	1.3499	0.1419			
0.4	1.4918				

$p = (x-x_0)/h = (0.02–0) / (0.2-0.1)=0.2$

By NFIF,

$y = y_0+ p\Delta y_0 + ((p(p-1))/2!)*\Delta^2 y_0 +((p(p-1)(p-2))/3!)*\Delta^3 y_0+$………..

$y = 1 + (0.2*0.1052)$

$+$

$(0.2*(0.2–1)*(0.0110))/2$

$+$

$(0.2*(0.2–1)*(0.2–2)*(0.0013))/6$

$+$

$(0.2*(0.2–1)*(0.2–2)*(0.2–3)*(-0.0002))/24$

$y = 1+0.0210-0.0009+0.0001+0$

$\underline{\underline{y = 1.0202}}$

EXAMPLE 02 :

x	0.1	0.2	0.3	0.4
y	2.68	3.04	3.38	3.68

find f(0.15).

x	y	Δy	$\Delta^2 y$	$\Delta^3 y$
0.1	2.68	0.36	-0.02	-0.02
0.2	3.04	0.34	-0.04	
0.3	3.38	0.3		
0.4	3.68			

$p = (x-x_0)/h = (0.15 - 1) /0.1 = 0.5$

By NFIF,

$y = y_0 + p\Delta y_0 + ((p(p-1))/2!)*\Delta^2 y_0 + ((p(p-1)(p-2))/3!)*\Delta^3 y_0 + \ldots\ldots\ldots$

$y = 2.68$

$+$

$(0.5*0.36)$

$+$

$((0.5*(0.5-1))/2)*(-0.02)$

$+$

$((0.5*(0.5 - 1)*(0.5-2))/6)*(-0.02)$

$y = 2.68 +0.18 +0.0025 - 0.00125$

<u>$y = 2.8613$</u>

SOURCE CODE

```
#include <stdio.h>
int main()
{
  int n,i,j,rows,o;
  float yin,Y,p,z,b[100],a[100][100];

  printf("\n\n\tNEWTON GREGORY FORWARD INTERPOLATION FORMULA\n\n");
  printf("   Enter number of values ");
  scanf("%d",&n);

  printf("\n   enter x values\n");
  for(i=1;i<=n;i++)
  {
      printf("\tx[%d][1] = ",i);
      scanf("%f",&a[i][1]);
   }

  printf("\n   enter y values\n");
  for(i=1;i<=n;i++)
  {
      printf("\ty[%d][2] = ",i);
      scanf("%f",&a[i][2]);
   }

  printf("\n   Enter x value for which value of y is to be found  ");
  scanf("%f",&z);

  rows=n-1;

  for(j=3;j<=n+1;j++)
  {
    for(i=1;i<=rows;i++)
    {
       a[i][j]=a[i+1][j-1]-a[i][j-1];
```

```
    }
   rows=rows-1;
 }

 printf("\n");

 printf("\tx\t\ty");

 for(i=1;i<n;i++)
 {
    printf("\t\tD%dy",i);
 }

 printf("\n");
 printf("\n");
 rows=n+1;

 for(i=1;i<=n;i++)
 {
    for(j=1;j<=rows;j++)
    printf("\t%0.4f\t",a[i][j]);
    printf("\n");
    rows=rows-1;

 }

 p=((z-a[1][1])/(a[2][1]-a[1][1]));
 printf("\n   p = %f\n\n",p);

 yin=1;

 for(j=0;j<n-1;j++)
 {
   yin=(((p-j)/(j+1))*yin);
   b[j]=yin;
   printf("   p term %d = %0.4f\n",j+1,b[j]);
 }
```

```
  Y=a[1][2];

  for(j=0;j<n-1;j++)
 {
   b[j]=b[j]*a[1][j+3];
   Y=Y+b[j];
   printf("\n   sum %d = %0.4f",j+1,Y);
  }

printf("\n\n\ty for value of x = %0.2f is %0.4f\n",z,Y);

}
```

EXECUTION

Output of Example 01 :

```
     NEWTON GREGORY FORWARD INTERPOLATION FORMULA

 Enter number of values 5

 enter x values
     x[1][1] = 0
     x[2][1] = 0.1
     x[3][1] = 0.2
     x[4][1] = 0.3
     x[5][1] = 0.4

 enter y values
     y[1][2] = 1
     y[2][2] = 1.1052
     y[3][2] = 1.2214
     y[4][2] = 1.3499
     y[5][2] = 1.4918

 Enter x value for which value of y is to be found  0.02

     x               y               D1y             D2y             D3y             D4y

     0.0000          1.0000          0.1052          0.0110          0.0013          -0.0002
     0.1000          1.1052          0.1162          0.0123          0.0011
     0.2000          1.2214          0.1285          0.0134
     0.3000          1.3499          0.1419
     0.4000          1.4918
```

```
  p = 0.200000

  p term 1 = 0.2000
  p term 2 = -0.0800
  p term 3 = 0.0480
  p term 4 = -0.0336

  sum 1 = 1.0210
  sum 2 = 1.0202
  sum 3 = 1.0202
  sum 4 = 1.0202

       y for value of x = 0.02 is 1.0202


...Program finished with exit code 0
Press ENTER to exit console.
```

Output of Example 02:

```
     NEWTON GREGORY FORWARD INTERPOLATION FORMULA

 Enter number of values 4

 enter x values
     x[1][1] = 0.1
     x[2][1] = 0.2
     x[3][1] = 0.3
     x[4][1] = 0.4

 enter y values
     y[1][2] = 2.68
     y[2][2] = 3.04
     y[3][2] = 3.38
     y[4][2] = 3.68

 Enter x value for which value of y is to be found  0.15

     x               y               D1y             D2y             D3y

     0.1000          2.6800          0.3600          -0.0200         -0.0200
     0.2000          3.0400          0.3400          -0.0400
     0.3000          3.3800          0.3000
     0.4000          3.6800
```

```
  p = 0.500000

  p term 1 = 0.5000
  p term 2 = -0.1250
  p term 3 = 0.0625

  sum 1 = 2.8600
  sum 2 = 2.8625
  sum 3 = 2.8613

       y for value of x = 0.15 is 2.8613


...Program finished with exit code 0
Press ENTER to exit console.
```