Jadavpur University

Numerical Analysis

Imran Alam

November 15, 2018

Contents

Numerical Method Assignment

Imran Alam, CSE
Roll No. 33

November 15, 2018

1 Bisection Method

Date: August 3, 2018

For the following Equation,

$$xsinx + cosx = 0 \tag{1}$$

1.1 Code

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

double fun(double x)
{
    return x*sin(x)+cos(x);
}
double bisection(double first, double second,double error)
{
    double mid_second,mid_first,f_mid,f_first,f_second,abserr=0,abserr0;
    double relerr=0,order=0;
    f_first=fun(first);
    f_second=fun(second);
    mid_first=(first+second)/2;
    int i=1;
    f_mid=fun(mid_first);
    printf("Itr  lower upper mid fun(midpoint) abserror relerror
        order\n");
    printf("%2d  %.6lf %.6lf %.6lf %.6lf %.6lf %.6lf
        %.6lf\n",i++,first,second,mid_first,f_mid,abserr,relerr,order);
    while(second-first > error)
    {
        if(f_mid*f_first>0)
        {
```

```
            first=mid_first;
            f_first=fun(a);
        }
        else if(f_mid*f_second>0)
        {
            second=mid_first;
            f_second=fun(second);
        }
        else
        {
            break;
        }
        mid_second=(first+second)/2;
        f_mid=fun(mid_second);
        abserr=mid_second-mid_first;
        relerr=abserr0/mid_second;
        order=log(fabs(abserr0))/log(fabs(abserr));
        abserr0=abserr;
        mid_first=mid_second;
        printf("%2d %.6lf %.6lf %.6lf %.6lf %.6lf %.6lf
            %.6lf\n",i++,first,second,mid_second,f_mid,abserr,relerr,order);
    }
    return mid_second;
}
int main(void)
{
    double first, second,f_first,f_second,root,error;
    printf("Enter the percision\n");
    scanf("%lf",&error);
    do
    {
        printf("Enter lower by upper bound");
        scanf("%lf%lf",&a,&b);
        f_first=fun(first);
        f_second=fun(second);
    } while(f_first*f_second>=0);
    root=bisection(first,second,error);
    printf("\nThe root is :%lf",root);
    return 0;
}
```

1.2 Output

Table:

Itr	lower	upper	mid	fun(midpoint)	abserror	relerror	order
1	0.000000	3.140000	1.570000	1.570796	0.000000	0.000000	0.000000
2	1.570000	3.140000	2.355000	0.960963	0.785000	0.000000	3028.978928
3	2.355000	3.140000	2.747500	0.131614	0.392500	0.285714	0.258840
4	2.747500	3.140000	2.943750	-0.401886	0.196250	0.133333	0.574330
5	2.747500	2.943750	2.845625	-0.126550	-0.098125	0.068966	0.701424
6	2.747500	2.845625	2.796563	0.004802	-0.049062	-0.035088	0.770075
7	2.796563	2.845625	2.821094	-0.060321	0.024531	-0.017391	0.813057
8	2.796563	2.821094	2.808828	-0.027619	-0.012266	0.008734	0.842501
9	2.796563	2.808828	2.802695	-0.011373	-0.006133	-0.004376	0.863931
10	2.796563	2.802695	2.799629	-0.003277	-0.003066	-0.002191	0.880229
11	2.796563	2.799629	2.798096	0.000765	-0.001533	-0.001096	0.893039
12	2.798096	2.799629	2.798862	-0.001255	0.000767	-0.000548	0.903375
13	2.798096	2.798862	2.798479	-0.000245	-0.000383	0.000274	0.911888
14	2.798096	2.798479	2.798287	0.000260	-0.000192	-0.000137	0.919023
15	2.798287	2.798479	2.798383	0.000008	0.000096	-0.000068	0.925089
16	2.798383	2.798479	2.798431	-0.000119	0.000048	0.000034	0.930310
17	2.798383	2.798431	2.798407	-0.000056	-0.000024	0.000017	0.934850
18	2.798383	2.798407	2.798395	-0.000024	-0.000012	-0.000009	0.938835
19	2.798383	2.798395	2.798389	-0.000008	-0.000006	-0.000004	0.942361
20	2.798383	2.798389	2.798386	-0.000000	-0.000003	-0.000002	0.945502
21	2.798383	2.798386	2.798385	0.000004	-0.000001	-0.000001	0.948318
22	2.798385	2.798386	2.798385	0.000002	0.000001	-0.000001	0.950858
23	2.798385	2.798386	2.798386	0.000001	0.000000	0.000000	0.953160

The root of equation (1) is 2.798386

2 Regula Falsi Method

Date: August 3, 2018

For the following Equation,

$$xsinx + cosx = 0 \tag{2}$$

2.1 Code

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

double fun(double x)
{
    return x*sin(x)+cos(x);
}
double regula(double first,double second,double error)
```

```
{
    double
        mid_second,mid_first,f_mid,f_first,f_second,abserr=0,abserr0,relerr=0,order=0;
    f_first=fun(first);
    f_second=fun(second);
    mid_first=(first*f_second-second*f_first)/(f_second-f_first);
    f_mid=fun(mid_first);
    printf("Itr  lower upper midpoint fun(midpoint) abserror relerror
        order\n");
    printf("%2d  %.6lf %.6lf %.6lf %.6lf %.6lf %.6lf
        %.6lf\n",i++,first,second,mid_first,f_mid,abserr,relerr,order);
    while(second-first > error)
    {
        if(f_first*f_mid<0)
            second=mid_first;
        else if(f_first*f_mid>0)
            first=mid_first;
        else
            break;
        mid_second=(first*f_second-second*f_first)/(f_second-f_first);
        f_mid=fun(mid_second);
        abserr=mid_second-mid_first;
        relerr=abserr0/mid_first;
        order=log(fabs(abserr0))/log(fabs(abserr));
        abserr0=abserr;
        mid_first=mid_second;
        printf("%2d %.6lf %.6lf %.6lf %.6lf %.6lf %.6lf
            %.6lf\n",i++,first,second,mid_first,f_mid,abserr,relerr,order);
    }
    return mid_second;
}
int main(void)
{
    double first,second,f_first,f_second,ans,error;
    printf("Enter the percision\n");
    scanf("%lf",&error);
    do {
        printf("Enter initial lower and upper bound for the root");
        scanf("%lf%lf",&first,&second);
        f_first=fun(first);
        f_second=fun(second);
    } while(f_first*f_second>=0);
    ans=regula(first,second,error);
    printf("\nThe root is :%lf",ans);
    return 0;
}
```

2.2 Output

Table:

Itr	lower	upper	midpoint	fun(midpoint)	abserror	relerror	order
1	0.000000	3.000000	1.914935	1.465269	0.000000	0.000000	0.000000
2	1.914935	3.000000	2.607545	0.466543	0.692610	0.000000	1996.329581
3	2.607545	3.000000	2.858054	-0.160516	0.250509	0.265617	0.265332
4	2.607545	2.858054	2.767448	0.080618	-0.090606	0.087650	0.576479
5	2.767448	2.858054	2.825283	-0.071552	0.057835	-0.032740	0.842490
6	2.767448	2.825283	2.804364	-0.015787	-0.020918	0.020470	0.737022
7	2.767448	2.804364	2.791012	0.019381	-0.013352	-0.007459	0.895987
8	2.791012	2.804364	2.799535	-0.003029	0.008523	-0.004784	0.905786
9	2.791012	2.799535	2.796452	0.005092	-0.003083	0.003044	0.824113
10	2.796452	2.799535	2.798420	-0.000089	0.001968	-0.001102	0.927951
11	2.796452	2.798420	2.797708	0.001786	-0.000712	0.000703	0.859687
12	2.797708	2.798420	2.798163	0.000589	0.000454	-0.000254	0.941673
13	2.798163	2.798420	2.798327	0.000156	0.000164	0.000162	0.883291
14	2.798327	2.798420	2.798386	-0.000001	0.000059	0.000059	0.895489
15	2.798327	2.798386	2.798365	0.000056	-0.000021	0.000021	0.905378
16	2.798365	2.798386	2.798379	0.000020	0.000014	-0.000008	0.959905
17	2.798379	2.798386	2.798383	0.000007	0.000005	0.000005	0.916734
18	2.798383	2.798386	2.798385	0.000002	0.000002	0.000002	0.923135

The root of equation (2) is 2.798385

3 Newton Raphson Method

Date: August 10, 2018

For the following Equation,

$$e^x = 2x + 1 \tag{3}$$

3.1 Code

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

double function(double x) //the given function
{
    return exp(x)-2*x-1;
}
double function1(double x) //differentiation of the given function
{
    return exp(x)-2;
```

```
}
void newton_raphson(double first,int num_iteration)
{
    double second,absolute_error0,absolute_error1 =0,order;
    int index=0;
    printf("Itr  xi f(xi) absolute_error order\n");
    while(index<num_iteration)
    {
        second=first-((function(first))/function1(first));
        absolute_error1=second-first;
        if(fabs(absolute_error1)<=0.00005)
            break;
        order=fabs(log(fabs(absolute_error1))/log(fabs(absolute_error0)));
            //to find the order of convergence
        absolute_error0=absolute_error1;
        printf("%2d %.4lf %.4lf  %.4lf
            %.4lf\n",index+1,first,function(first),fabs(absolute_error1),order);
        first=second;
        index++;
    }
}
int main()
{
    double first;
    int num_iteration;
    printf("Enter the first root: \n");
    scanf("%lf",&first);
    printf("Enter the number of iteration: \n");
    scanf("%d",&num_iteration);
    newton_raphson(first,num_iteration);
    return 0;
}
```

3.2 Output

Table:

Itr	xi	f(xi)	$absolute_error$	order
1	5.0000	137.4132	0.9385	0.0001
2	4.0615	48.9366	0.8729	2.1420
3	3.1885	16.8757	0.7584	2.0354
4	2.4302	5.5004	0.5876	1.9223
5	1.8426	1.6276	0.3774	1.8328
6	1.4652	0.3979	0.1709	1.8128
7	1.2943	0.0598	0.0363	1.8777
8	1.2580	0.0024	0.0016	1.9483

The root of equation (3) is 1.2580

4 Fixed Point Iteration Method

Date: August 3, 2018

For the following Equation,

$$e^x - 4x^2 = 0 \tag{4}$$

Initial Root Guess = 1
No. Of Iterations = 10

4.1 Code

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

#define E 0.000005 //Precision
double Gfunction(double x) //function in terms of x
{
    return sqrt(exp(x)/4);
}
double Gdiff(double x) //first derivative of Gfunction
{
    return (exp(x/2)/4);
}
double Ffunction(double x) //the given function
{
    return exp(x)-4*x*x;
}
double fixed_point(double first,int num_iteration)
{
    double second1,second2,absolute_error0,absolute_error1=0,order;
    int index=0;
    second1=Gfunction(first);
    printf("Itr \t xi \t |g'(xi)| \t f(xi) \t absolute_error \t
         order\n");
    while(index<num_iteration)
    {
        second2=Gfunction(second1);
        absolute_error1=second2-second1; //to find absolute error
        if(fabs(absolute_error1)<=E)
            break;
        order=log(fabs(absolute_error0))/log(fabs(absolute_error1));
        absolute_error0=absolute_error1; //to find the order of
             convergence
        printf("%2d %.6lf %.6lf %.6lf %.6lf
             %.6lf\n",index+1,second1,fabs(Gdiff(second1)),Ffunction(second1),fabs(absolute_error1),ord
        second1=second2;
        index++;
```

```
    }
}
int main()
{
    double first;
    int num_iteration;
    printf("Enter the first approximate root\n");
    scanf("%lf",&first);
    while(fabs(Gdiff(first))>1)
    {
      printf("Enter the first approximate root\n");
      scanf("%lf",&first);
    }
    printf("Enter the total no. of iteration\n");
    scanf("%d",&num_iteration);
    fixed_point(first,num_iteration);
    return 0;
}
```

4.2 Output

Table:

Itr	xi	abs(g'(xi))	f(xi)	absolute error	order
1	0.824361	0.377527	-0.437860	0.069307	270.570501
2	0.755053	0.364668	-0.152697	0.025717	0.729172
3	0.729336	0.360009	-0.054021	0.009318	0.782884
4	0.720018	0.358336	-0.019233	0.003347	0.820351
5	0.716671	0.357736	-0.006864	0.001198	0.847310
6	0.715473	0.357522	-0.002452	0.000429	0.867409
7	0.715044	0.357446	-0.000876	0.000153	0.882890
8	0.714891	0.357418	-0.000313	0.000055	0.895154
9	0.714836	0.357408	-0.000112	0.000020	0.905099
10	0.714817	0.357405	-0.000040	0.000007	0.913323

The root of equation (4) is 0.714817

5 Secant Method

Date: August 3, 2018

For the following Equation,

$$e^x - 2x - 1 = 0 \tag{5}$$

Intial Root Guess = 2, 3

5.1 Code

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

double function(double x) // the given function
{
   return exp(x)-2*x-1;
}
double function1(double x) //first derivative of given function
{
   return exp(x)-2;
}
double function2(double x) //second derivative of given function
{
   return exp(x);
}
double convergence_condition(double x) //conditon for the convergence
{
   return (fabs(function(x)*function2(x)))/(function1(x)*function1(x));
}
double secant(double x0,double x1) //finding root of function using
     secant method
{
   double x2,absolute_error0,absolute_error1=0,order;
   int index=1;
   printf("itr\t x0\t x1\t x2\t f(x2)\t abs_error\t order\n");
   do{
      x2= x1-(((x1-x0)*function(x1))/(function(x1)-function(x0)));
      printf("%d %.6f %.6f %.6f %.6f %.6f
          %.6f\n",index,x0,x1,x2,function(x2),absolute_error1,order);
      absolute_error0 = fabs(x1-x0);
      absolute_error1 = fabs(x2-x1);
      order=log(absolute_error1)/log(absolute_error0); //calculating the
          order of convergence
      x0=x1;
      x1=x2;
      index++;
   }while(fabs(absolute_error0)>0.000005);
   return x0; //returning the final root
}
int main()
{
   double first,second;
    do
    {
      printf("Enter the two approximate root\n");
      scanf("%lf %lf",&first,&second);
```

```
    }while(convergence_condition(first)>=1 ||
        convergence_condition(second)>=1); //condition for the
        convergence
   printf("Final root is %lf\n",secant(first,second));

}
```

5.2 Output

Table:

Itr	x0	x1	x2	f(x2)	abs error	order
1	2.000000	3.000000	1.776650	1.356726	0.000000	0.000000
2	3.000000	1.776650	1.635140	0.859895	1.223350	inf
3	1.776650	1.635140	1.390219	0.235291	0.141511	-9.699660
4	1.635140	1.390219	1.297956	0.065892	0.244921	0.719460
5	1.390219	1.297956	1.262068	0.008583	0.092263	1.693975
6	1.297956	1.262068	1.256693	0.000396	0.035888	1.396217
7	1.262068	1.256693	1.256433	0.000003	0.005375	1.570614
8	1.256693	1.256433	1.256431	0.000000	0.000260	1.579698
9	1.256433	1.256431	1.256431	0.000000	0.000002	1.609030

Final root is 1.256431

6 Bairstow Method

Date: August 25, 2018

For the following Equation,

$$x^4 - 5x^3 + 10x^2 - 10x + 4 = 0 \tag{6}$$

Parameters:

$$r = 0.5, s = -0.5(initialvalues) \tag{7}$$

$$e = 0.01(precisionvalue) \tag{8}$$

6.1 Code

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

int main()
{
    double r,s,e,er=99,es=99;
    printf("Enter initial values of r, s and e\n");
```

```c
    scanf("%lf",&r);
    scanf("%lf",&s);
    scanf("%lf",&e); //precision value
    int n;
    printf("Enter the highest power of the polynomial equation\n");
    scanf("%d",&n);
    double a[n+1];
    printf("Enter the coefficients of polynomial equation\n");
    for (int i = 0; i < n+1; ++i)
    {
        scanf("%lf",&a[i]);
    }
    double b[n+1],c[n+1];
    double delr,dels;
    int i=0,j=0;
    printf("Iter R S\n");
    while(fabs(er)>=e || fabs(es)>=e) //condition to be checked
    {
        b[n]=a[n];
        b[n-1]=a[n-1]+r*b[n];
        for(i=n-2; i>=0; i--)
        {
            b[i]=a[i]+r*b[i+1]+s*b[i+2]; //updating values of array b
        }
        c[n]=b[n];
        c[n-1]=b[n-1]+r*c[n];
        for(i=n-2; i>=1; i--)
        {
            c[i]=b[i]+r*c[i+1]+s*c[i+2]; //updating value of array c
        }
        delr=(-b[1]*c[2]+b[0]*c[3])/(c[2]*c[2]-c[1]*c[3]);
        dels=(b[1]*c[1]-b[0]*c[2])/(c[2]*c[2]-c[1]*c[3]);
        r=r+delr; //new r
        s=s+dels; //new s
        printf("%d  %lf %lf\n",++j,r,s);
        er=fabs(delr/r);
        es=fabs(dels/s);
    }
    printf("Two out of four roots are %lf and
        %lf\n",(r+sqrt(r*r+4*s))/2,(r-sqrt(r*r+4*s))/2);
    return 0;
}
```

6.2 Output

Table:

Iter	R	S
1	1.618037	-0.203581
2	3.898011	0.121352
3	2.936255	1.418393
4	2.787267	-0.204441
5	2.783626	-1.155663
6	2.883139	-1.711455
7	2.983940	-1.964652
8	2.999809	-1.999534
9	3.000000	-2.000000

The roots of equation (6) are 2, 1.

7 Gauss Elimination

Date: August 31, 2018

Solve for the following Equations,

$$x_1 + x_2 - x_3 + x_4 = 2 \tag{9}$$

$$2x_1 + x_2 + x_3 - 3x_4 = 1 \tag{10}$$

$$3x_1 - x_2 - x_3 + x_4 = 2 \tag{11}$$

$$5x_1 + x_2 + 3x_3 - 2x_4 = 7 \tag{12}$$

7.1 Code

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

void print_matrix(double **matrix, int size) //function to print the
    matrix in the form of equations
{
    int i,j;
    for(i=0; i<size; i++)
    {
        for(j=0; j<size; j++)
        {
            if(j!=size-1)
                printf("(%2.2lf) x%d + ",matrix[i][j],i+1);
            else
                printf("(%2.2lf) x%d",matrix[i][j],i+1);
        }
        printf(" = %2.2lf\n",matrix[i][size]); //printing the RHS of the
            equations
```

```
    }
}

int finding_pivot(double **matrix,int indx,int n) //function to get the
    partial pivot (the maximum value in each column)
{
    int row,pivot=indx;
    double max=fabs(matrix[indx][indx]);
    for(row=indx+1; row<n; row++)
    {
        if(fabs(matrix[row][indx])>max) { //row containing the largest
             element is selected as pivot
            max=matrix[row][indx];
            pivot=row;
        }
    }
    return pivot;
}

void swap_rows(double **p2Row1,double **p2Row2)
{   // function to swap pivotal row & current row
    double *temp=*p2Row1; // pointers pointing to respective rows are
         swapped
    *p2Row1=*p2Row2;
    *p2Row2=temp;
}

void gaussian_elimination(double **matrix, int n, int curIndx) //using
    gaussian elimination method, to find the upper triangular matrix
{
    if(curIndx==n-1)
    { // reached last row of augmented matrix so no more elimination is
         possible
        printf("The elimination is completed\n");
    }
    else
    {
        printf("\nSTEP %d:\n\n",curIndx+1);
        int row,col;
        int pivot=finding_pivot(matrix,curIndx,n); // finding the pivot
             row(row having maximum element in the current column)
        double multiplier;
        if(pivot!=curIndx)
        {
            printf("Swapping row %d and %d\n",curIndx,pivot);
            swap_rows(&matrix[pivot],&matrix[curIndx]); //we are swapping
                 two rows when pivot (max) is found
        }
        for(row=curIndx+1; row<n; row++)
        {
```

```
            multiplier=matrix[row][curIndx]/matrix[curIndx][curIndx];//generating the multiplier for each row below the current row
            printf("row %d --> row %d -(%lf)*row %d\n",row,row,multiplier,curIndx);
            for(col=curIndx; col<n+1; col++)
            {
                matrix[row][col]-=multiplier*matrix[curIndx][col];//eliminating the column elements with the transformation Rj=Rj-mj*Ri;i<j<n
            }
        }
        printf("Equations after Step %d :\n",curIndx+1);
        print_matrix(matrix,n); //printing the matrix in the form of equations

        gaussian_elimination(matrix,n,curIndx+1); //recursive call to the function to eliminate next column elements
    }

}

void backsubstitution(double **M, double *X, int n) //function for performing back substitution
{
    int i,j;
    double sum;
    X[n-1]=M[n-1][n]/M[n-1][n-1];
    for(i=n-2; i>=0; i--)
    {
        sum=0;
        for(j=i+1; j<n; j++) sum+=X[j]*M[i][j];
        X[i]=(M[i][n]-sum)/M[i][i];
    }
}

int main()
{
    int size,i,j;
    double **matrix,*roots;
    printf("Enter Order of matrix: ");
    scanf("%d",&size);
    matrix=(double **)(malloc(size*sizeof(int *)));
    for(i=0; i<size; i++)
        matrix[i]=(double *)(malloc((size+1)*sizeof(double)));
    printf("Enter the elements of augmented matrix row-wise:\n");
    for(i=0; i<size; i++)
    {
        for(j=0; j<size+1; j++)
            scanf("%lf",&matrix[i][j]);
```

```
    }
    printf("The given set of equations are :\n");
    print_matrix(matrix,size);
    gaussian_elimination(matrix,size,0);
    roots=(double *)(malloc(size*sizeof(double)));
    backsubstitution(matrix,roots,size);
    printf("The Roots Of the Given Set of Equation are :\n");
    for(i=0; i<size; i++)
        printf("x%d = %2.2lf\n",i+1,roots[i]);
    for(i=0; i<size; i++)
        free(matrix[i]);
    free(matrix); //deallocating memory
    free(roots);
    return 0;
}
```

7.2 Output

Augmented Matrix is:

$$A = \begin{bmatrix} 1 & 1 & -1 & 1 & 2 \\ 2 & 1 & 1 & -3 & 1 \\ 3 & -1 & -1 & 1 & 2 \\ 5 & 1 & 3 & -2 & 7 \end{bmatrix}$$

STEP 1:

Swapping R0 $\leftrightarrow$ R3
R1 $\rightarrow$ R1 -(0.400000)*R0
R2 $\rightarrow$ R2 -(0.600000)*R0
R3 $\rightarrow$ R3 -(0.200000)*R0

$$A = \begin{bmatrix} 5 & 1 & 3 & -2 & 7 \\ 0 & 0.6 & -0.2 & -2.2 & -1.8 \\ 0 & -1.6 & -2.8 & 2.2 & -2.2 \\ 0 & 0.8 & -1.6 & 1.4 & 0.6 \end{bmatrix}$$

STEP 2:

Swapping R1 $\leftrightarrow$ R3
R2 $\rightarrow$ R2 -(-2.000000)*R1
R3 $\rightarrow$ R3 -(0.750000)*R1

$$A = \begin{bmatrix} 5 & 1 & 3 & -2 & 7 \\ 0 & 0.8 & -1.6 & 1.4 & 0.6 \\ 0 & 0 & -6 & 5 & -1 \\ 0 & 0 & -1 & -3.25 & -2.25 \end{bmatrix}$$

STEP 3:

R3 $\rightarrow$ R3 -(-0.166667)*R2

$$A = \begin{bmatrix} 5 & 1 & 3 & -2 & 7 \\ 0 & 0.8 & -1.6 & 1.4 & 0.6 \\ 0 & 0 & -6 & 5 & -1 \\ 0 & 0 & 0 & -2.42 & -2.42 \end{bmatrix}$$

The elimination is completed.
Using Backsubstitution we get the Roots Of the Given Set of Equation :
$x_1 = 1.00$, $x_2 = 1.00$, $x_3 = 1.00$, $x_4 = 1.00$

8 Gauss Seidel

Date: August 31, 2018

Solve for the following Equations,

$$5x_1 - x_2 + x_3 = 10 \tag{13}$$

$$2x_1 + 8x_2 - x_3 = 11 \tag{14}$$

$$-x_1 + x_2 + 4x_3 = 3 \tag{15}$$

8.1 Code

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

float** interchangeRow(float** matrix,int range,int n,int
    m){//interchanging two given rows
   int i;
   float temp;
   if(n==m) return matrix;
   for(i=0;i<range;i++){
      temp=matrix[n][i];
      matrix[n][i]=matrix[m][i];
      matrix[m][i]=temp;
   }
   return matrix;
}
float** maxRowPivot(float** matrix,int range,int n){//pivoting the row
   int i,pos=n;
   float max=fabs(matrix[n][n]);
   for(i=n;i<(range-1);i++){
```

```
    if(max<fabs(matrix[i][n])){//max in a column
      max=matrix[i][n];
      pos=i;
    }
  }
  matrix=interchangeRow(matrix,range,n,pos);
  return matrix;
}
void display(float** matrix,int range){
  int i,j;
  for(i=0;i<(range-1);i++){
    for(j=0;j<(range);j++){
      printf("%6.3f\t",matrix[i][j]);
    }
    printf("\n");
  }
  return;
}
int main()
{
  float **matrix,term,*answers;
  int range,i,j;

  printf("Enter the number of variables\t");
  scanf("%d",&range);
  range++;
  matrix=(float**)malloc(sizeof(float*)*(range-1));
  answers=(float*)malloc(sizeof(float)*(range-1));
  for(i=0;i<range;i++){
    matrix[i]=(float*)malloc(sizeof(float)*range);
  }
  printf("Enter the coefficients separated with spaces\n");
  for(i=0;i<(range-1);i++){
    printf("Enter equation number %d\n",(i+1));
    for(j=0;j<range;j++){
      scanf("%f",&matrix[i][j]);
    }
  }
  float xprev[range],x[range];
  printf("Enter initial guess of variables satisfying the equations\n");
  for(i=0;i<range-1;i++)
  {
    scanf("%lf",&xprev[i]);
    x[i]=0;
  }
  xprev[range-1]=1;

  float max;
  float e[range-1];
  int iter=1;
```

```
  printf("i  x1 x2 x3 MaxError\n");
  do
  {
     for(i=0;i<range-1;i++)
        x[i]=xprev[i];
     for(i=0;i<range-1;i++)
     {
        xprev[i]=matrix[i][range-1]/matrix[i][i];
        for(j=0;j<range-1;j++)
        {
           if(i!=j)
              xprev[i]=xprev[i]-(matrix[i][j]*xprev[j]/matrix[i][i]);
        }
     }
     max=-1;
     printf("%d\t",iter);
     iter++;
     for(i=0;i<range-1;i++)
     {
        printf("%lf\t",x[i]);
     }
     for(i=0;i<range-1;i++)
     {
           e[i]=(-x[i]+xprev[i]);
           if(max<fabs(e[i]))
              max=fabs(e[i]);
     }
     printf("%lf\t",max);//maximum error
     printf("\n");
  }while(fabs(max)>=0.00001);
  for(i=0;i<range-1;i++)
     printf("%lf\n",xprev[i]);
  return 0;
}
```

8.2 Output

Augmented Matrix is:

$$A = \begin{bmatrix} 5 & -1 & 1 & 10 \\ 2 & 8 & -1 & 11 \\ -1 & 1 & 4 & 3 \end{bmatrix}$$

Table:

i	x1	x2	x3	MaxError
1	0.000000	0.000000	0.000000	2.000000
2	2.000000	0.875000	1.031250	0.136719
3	1.968750	1.011719	0.989258	0.035742
4	2.004492	0.997534	1.001739	0.005333
5	1.999159	1.000428	0.999683	0.000990
6	2.000149	0.999923	1.000056	0.000176
7	1.999973	1.000014	0.999990	0.000031
8	2.000005	0.999998	1.000002	0.000006

$x_1 = 2.000005$, $x_2 = 0.999998$, $x_3 = 1.000002$

9 Gauss Jordan Elimination

Date: September 14, 2018

Find Inverse of the Followong Matrix

$$A = \begin{bmatrix} 1 & 5 & 3 \\ 1 & 3 & 2 \\ 2 & 4 & -6 \end{bmatrix}$$

9.1 Code

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

void display(float** matrix,int range){
   int i,j;
   for(i=0;i<(range);i++){
      for(j=0;j<(range);j++){
         printf("%6.3f\t",matrix[i][j]);
      }
      printf("\n");
   }
   printf("\n");
   return;
}
float** interchangeRow(float** matrix,int range,int n,int m){
   int i;
   float temp;
   if(n==m) return matrix;
   for(i=0;i<range;i++){
      temp=matrix[n][i];
      matrix[n][i]=matrix[m][i];
      matrix[m][i]=temp;
```

```
   }
   return matrix;
}
void maxRowPivot(float** matrix,float** inverse,int range,int n){
   int i,pos=n;
   float max=fabs(matrix[n][n]);
   for(i=n;i<(range);i++){
      if(max<fabs(matrix[i][n])){
         max=matrix[i][n];
         pos=i;
      }
   }
   matrix=interchangeRow(matrix,range,n,pos);
   inverse=interchangeRow(inverse,range,n,pos);
   return ;
}
float** allocate(float** matrix,int range){//allocates matrix of n*n size
   int i;
   matrix=(float **)malloc(range*sizeof(float *));
   for(i=0;i<range;i++){
      matrix[i]=(float *)malloc(range*sizeof(float));
   }
   return matrix;
}
float** matrixMult(float** a,float** b,float** newm,int range){
   int i,j,k;
   for(i=0;i<range;i++){
      for(j=0;j<range;j++){
         newm[i][j]=0;
         for(k=0;k<range;k++){
            newm[i][j]+=(a[i][k]*b[k][j]);
         }
      }
   }
   return newm;
}
int main(){
   float **matrix,**inverse,**original,**newm,temp;
   int i,j,k,range;

   printf("Enter the size of the matrix(i.e. value of 'n' as size is
       nxn):\t");
   scanf("%d",&range);

   matrix=(float **)allocate(matrix,range);
   inverse=(float **)allocate(inverse,range);
   original=(float **)allocate(original,range);
   newm=(float **)allocate(newm,range);
   printf("Enter the matrix:\n");
   for(i=0;i<range;i++){
```

```
    printf("Row number %d\n",(i+1));
    for(j=0;j<range;j++){
      scanf("%f",&matrix[i][j]);
      original[i][j]=matrix[i][j];
    }
  }
  for(i=0;i<range;i++){
    for(j=0;j<range;j++){
      if(i==j) inverse[i][j]=1;
      else inverse[i][j]=0;
    }
  }
  for(k=0;k<range;k++){
    maxRowPivot(matrix,inverse,range,k);
    temp=matrix[k][k];
    for(j=0;j<range;j++){
      matrix[k][j]/=temp;
      inverse[k][j]/=temp;
    }
    for(i=0;i<range;i++){
      temp=matrix[i][k];
      for(j=0;j<range;j++){
        if(i==k) break;
        matrix[i][j]-=matrix[k][j]*temp;
        inverse[i][j]-=inverse[k][j]*temp;
      }
    }
    display(inverse,range);
  }
  printf("The inverse of the matrix is:\n");
  display(inverse,range);
  return 0;
}
```

9.2 Output

Matrix After Each Iteration:

$$A^{-1} = \begin{bmatrix} 0.000 & 0.000 & 0.500 \\ 0.000 & 1.000 & -0.500 \\ 1.000 & 0.000 & -0.500 \end{bmatrix}$$

$$= \begin{bmatrix} -0.667 & 0.000 & 0.833 \\ 0.333 & 0.000 & -0.167 \\ -0.333 & 1.000 & -0.333 \end{bmatrix}$$

$$= \begin{bmatrix} -1.444 & 2.333 & 0.056 \\ 0.556 & -0.667 & 0.056 \\ -0.111 & 0.333 & -0.111 \end{bmatrix}$$

Therefore, the inverse of the matrix A is:

$$A^{-1} = \begin{bmatrix} -1.444 & 2.333 & 0.056 \\ 0.556 & -0.667 & 0.056 \\ -0.111 & 0.333 & -0.111 \end{bmatrix}$$

10 Power Method

Date: September 14, 2018

Find Eigen Value and Vector of the Followong Matrix

$$A = \begin{bmatrix} 0 & 11 & -5 \\ -2 & 17 & -7 \\ 4 & 26 & -10 \end{bmatrix}$$

10.1 Code

```
#include <stdio.h>
#include <math.h>
void powerMethod(int n,float arr[n][n]); //function prototype
int main()
{
    int i,j,n;
    printf("\nEnter the order of matrix:");
    scanf("%d",&n);
    float arr[n][n];
    printf("Enter elements of matrix:-\n");
    for(i=0; i<n; i++)
        for(j=0; j<n; j++)
            scanf("%f",&arr[i][j]); //input the matrix
    powerMethod(n,arr);
    return 0;
}
void powerMethod(int n,float arr[n][n]) //function to calculate using
     power method
{
    int i,j,iteration=1;
    float x[n],z[n],error[n],zmax,emax;
    printf("Enter approximation:\n");
    for(i=0; i<n; i++)
        scanf("%f",&x[i]);
    printf("Iteration\tEigenvalue\t");
    for(i=1; i<=n; i++)
        printf("x%d\t\t",i);
    printf("error\n");
    do
```

```
    {
        for(i=0; i<n; i++)
        {
            z[i]=0;
            for(j=0; j<n; j++)
                z[i]=z[i]+arr[i][j]*x[j]; //putting the value of z[i]
        }
        zmax=fabs(z[0]);
        for(i=1; i<n; i++)
            if((fabs(z[i]))>zmax)
                zmax=fabs(z[i]); //calculating the zmax
        for(i=0; i<n; i++)
            z[i]=z[i]/zmax;
        for(i=0; i<n; i++)
        {
            error[i]=0;
            error[i]=fabs((fabs(z[i]))-(fabs(x[i]))); //calculating errors
        }
        emax=error[0];
        for(i=1; i<n; i++)
            if(error[i]>emax)
                emax=error[i];
        printf("%d\t\t%f\t",iteration,zmax);
        for(i=0; i<n; i++)
            printf("%f\t",x[i]);
        printf("%f\n",emax);
        for(i=0; i<n; i++)
            x[i]=z[i];
        iteration++;
    } while(emax>0.0001);
    printf("\nThe required eigen value is %f",zmax);
    printf("\n\nThe required eigen vector is :\n");
    for(i=0; i<n; i++)
        printf("%f\n",z[i]);
}
```

10.2 Output

Table:

Iteration	Eigenvalue	x1	x2	x3	error
1	12.000000	1.000000	1.00000	1.000000	0.500000
2	5.333334	0.500000	0.66666	1.000000	0.062500
3	4.500002	0.437500	0.62500	1.000000	0.020833
4	4.222221	0.416667	0.61111	1.000000	0.008772
5	4.105259	0.407895	0.60526	1.000000	0.004048
6	4.051280	0.403846	0.60256	1.000000	0.001947
7	4.025314	0.401899	0.60126	1.000000	0.000955
8	4.012577	0.400943	0.60062	1.000000	0.000473
9	4.006268	0.400470	0.60031	1.000000	0.000235
10	4.003129	0.400235	0.60015	1.000000	0.000118
11	4.001569	0.400117	0.60007	1.000000	0.000059

The required eigen value is 4.001569

The required eigen vector is :

$$\begin{pmatrix} 0.400059 \\ 0.600039 \\ 1.000000 \end{pmatrix}$$

11 LU Decomposition

Date: October 5, 2018

Decompose the following Matrix into Lower and Upper triangle Matrix:

$$A = \begin{bmatrix} 1 & 5 & 3 \\ 1 & 3 & 2 \\ 2 & 4 & -6 \end{bmatrix}$$

11.1 Code

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

int main()
{
    float **A,**L,**U,sum=0;
    int n,i,j,k,p;
    printf("\n\tPlease enter the size of the matrix: ");
    scanf("%d",&n);
    A=(float **)malloc(n*sizeof(float *));
    for(i=0; i<n; i++)
        A[i]=(float *)malloc(n*sizeof(float));
```

```
printf("\n\t Please entry the coefficient matrix: \n\n");
for(i=0; i<n; i++)
    for(j=0; j<n; j++)
        scanf("%f",&A[i][j]);

L=(float **)calloc(n,sizeof(float *));
for(i=0; i<n; i++)
    L[i]=(float *)calloc(n,sizeof(float));

U=(float **)calloc(n,sizeof(float *));
for(i=0; i<n; i++)
    U[i]=(float *)calloc(n,sizeof(float));

for(j=0; j<n; j++) //Decomposition
{
    for(i=0; i<n; i++)
    {
        if(i>=j)
        {
            L[i][j]=A[i][j];
            for(k=0; k<=j-1; k++)
                L[i][j]-=L[i][k]*U[k][j];
            if(i==j)
                U[i][j]=1;
        }
        else
        {

            U[i][j]=A[i][j];
            for(k=0; k<=i-1; k++)
                U[i][j]-=L[i][k]*U[k][j];
            U[i][j]/=L[i][i];
        }
    }
}
printf("\nL matrix:");
for(i=0; i<n; i++)
{
    printf("\n\t");
    for(j=0; j<n; j++)
        printf("%f ",L[i][j]);
}

printf("\n\n");

printf("\nU matrix:");
for(i=0; i<n; i++)
{
    printf("\n\t");
    for(j=0; j<n; j++)
```

```
        printf("%f ",U[i][j]);
    }
    return 0;
}
```

11.2 Output

Decomposing the Matrix A, such that A=L*U

$$A = \begin{bmatrix} 5 & -1 & 1 \\ 2 & 8 & -1 \\ -1 & 1 & 4 \end{bmatrix}$$

$$L = \begin{bmatrix} 5.000000 & 0.000000 & 0.000000 \\ 2.000000 & 8.400000 & 0.000000 \\ -1.000000 & 0.800000 & 4.333333 \end{bmatrix}$$

$$U = \begin{bmatrix} 1.000000 & -0.200000 & 0.200000 \\ 0.000000 & 1.000000 & -0.166667 \\ 0.000000 & 0.000000 & 1.000000 \end{bmatrix}$$

12 Jacobi

Date: November 2, 2018

Find Eigen Value and Vector of the following symmetric matrix:

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 4 & 5 \\ 3 & 5 & 6 \end{bmatrix}$$

12.1 Code

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

int main()
{

    printf("Jacobi eigen method\n");
   int n;
   printf("Enter order of symmetric matrix\n"); //scans matrix
   scanf("%d",&n);
   float **a=(float **)malloc(n*sizeof(float *));
   for (int i = 0; i < n; ++i)
   {
```

```
   a[i]=(float *)malloc(n*sizeof(float));
}
printf("Scan matrix rowwise\n");
int i,j,r,s;

float max=0.0,sum=0.0;
for (i=0;i<n;i++)
{
   for (j=0;j<n;j++)
   {
      scanf("%f",&a[i][j]);
   }
}
for (i=0;i<n;i++)
{
   for (j=0;j<n;j++)
   {
      if(j>i)
      {
         sum+=a[i][j];
         if(fabs(a[i][j])>=max)
         {
            max=a[i][j];
            r=i;
            s=j;
         }
      }
   }
}
while(sum>=0.0001)
{
   float ang=atan((2*a[r][s])/(a[r][r]-a[s][s]))/2;
   for (i=0;i<n;i++)
      for (j=0;j<n;j++)
      {
         if(j>i &&j!=r&&j!=s)
         {
            a[j][r]=a[j][r]*cos(ang)+a[j][s]*sin(ang);
            a[r][j]=a[j][r];
            a[j][s]=a[j][s]*cos(ang)-a[j][r]*sin(ang);
            a[s][j]=a[j][s];
         }
      }
   a[r][r]=a[r][r]*cos(ang)*cos(ang)+2*a[r][s]*cos(ang)*sin(ang)+a[s][s]*sin(ang)*sin(ang);
   a[s][s]=a[s][s]*cos(ang)*cos(ang)-2*a[r][s]*cos(ang)*sin(ang)+a[r][r]*sin(ang)*sin(ang);

   a[r][s]=0;a[s][r]=0;
   max=0,sum=0;
   for(i=0;i<n;i++)
      for(j=0;j<n;j++)
```

```
        {
            if(j>i)
            {
                sum+=a[i][j];
                if(max<fabs(a[i][j]))
                {
                    max=a[i][j];r=i;s=j;
                }
            }
        }
    }
    printf("The eigen values are:\n");
    for(i=0;i<n;i++)
    {
        printf("%f\n",a[i][i]);
    }
    return 0;
}
```

12.2 Output

Eigen Vector is: $\begin{pmatrix} 1.785970 \\ -1.042065 \\ 9.414134 \end{pmatrix}$

13 Interpolation

Date: October 5, 2018

Function Used:

$$f(x) = 3.14159x^2$$

Inputs Used
Enter number of records : 7
Enter x0 and interval for x :0 1
Enter x for finding f(x) : 2.5

13.1 Code

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

void input(int n,float x[n],float y[n][n],float* f);
void difference(int n,float y[n][n]);
```

```
float forwardInterpolation(int n,float x[n],float y[n][n],float f);
float backwardInterpolation(int n,float x[n],float y[n][n],float f);
float func(float n); //function prototypes
int main()
{
   int i,j,n;
   printf("Enter number of records : ");
   scanf("%d",&n);
   float x[n],y[n][n],f,forward,backward,fx;
   input(n,x,y,&f);
   difference(n,y);
   fx=func(f);
   printf("\nTable for Forward Interpolation Method :\n");
   for(i=0;i<n;i++)
    {
       printf("\t%.2f",x[i]);
       for(j=0;j<(n-i);j++)
           printf("\t%.2f",y[i][j]);
       printf("\n");
    }
   printf("Using Newton's Forward Interpolation :\n");
    printf("f(%f) = %f\nError = %f\n",f,forward,fabs(forward-fx));
   forward=forwardInterpolation(n,x,y,f);
   backward=backwardInterpolation(n,x,y,f);
   printf("\nTable for Forward Interpolation Method :\n");
   for(i=0;i<n;i++)
    {
       printf("\t%.2f",x[i]);
       for(j=0;j<=i;j++)
           printf("\t%.2f",y[i][j]);
       printf("\n");
    }
    printf("Using Newton's Backward Interpolation :\n");
    printf("f(%f) = %f\nError = %f\n",f,backward,fabs(backward-fx));
    return 0;
}
void input(int n,float x[n],float y[n][n],float* f) //function to input
    data
{
   int i;
   float interval;
   printf("Enter x0 and interval for x :");
   scanf("%f %f",&x[0],&interval);
   y[0][0]=func(x[0]);
   for(i=1;i<n;i++)
   {
      x[i]=x[i-1]+interval;
      y[i][0]=func(x[i]);
   }
   printf("Enter x for finding f(x) : ");
```

```
   scanf("%f",f);
}
void difference(int n,float y[n][n]) //function to calculate the
    difference
{
   int i,j;
   for(i=1;i<n;i++)
      for(j=0;j<n-i;j++)
         y[j][i]=y[j+1][i-1]-y[j][i-1];
}
float forwardInterpolation(int n,float x[n],float y[n][n],float f)
    //function to calculate using forward interpolation method
{
   int i,j;
   float h=x[1]-x[0],p=(f-x[0])/h,sum=y[0][0],term=1;
   for(i=1;i<n;i++)
   {
      term*=(p--)/i;
      sum+=term*y[0][i];
   }
   return sum;
}
float backwardInterpolation(int n,float x[n],float y[n][n],float f)
    //function to calculate using backward interpolation method
{
   int i,j;
   float h=x[1]-x[0],p=(f-x[n-1])/h,sum=y[n-1][0],term=1;
        for(i=1;i<n;i++)
        {
                term*=(p++)/i;
                sum+=term*y[n-i-1][i];
        }
    return sum;
}
float func(float n)//circle
{
   return 3.14159*n*n;
}
```

13.2 Output

Table for Forward Interpolation Method :

x(i)	y1(i)	y2(i)	y3(i)	y4(i)	y5(i)	y6(i)	y7(i)
0.00	0.00	3.14	6.28	-0.00	0.00	-0.00	0.00
1.00	3.14	9.42	6.28	0.00	-0.00	0.00	
2.00	12.57	15.71	6.28	-0.00	0.00		
3.00	28.27	21.99	6.28	0.00			
4.00	50.27	28.27	6.28				
5.00	78.54	34.56					
6.00	113.10						

Table for Backward Interpolation Method :

x(i)	y1(i)	y2(i)	y3(i)	y4(i)	y5(i)	y6(i)	y7(i)
0.00	0.00						
1.00	3.14	3.14					
2.00	12.57	9.42	6.28				
3.00	28.27	15.71	6.28	0.00			
4.00	50.27	21.99	6.28	0.00	-0.00		
5.00	78.54	28.27	6.28	0.00	-0.00	0.00	
6.00	113.10	34.56	6.28	0.00	0.00	0.00	0.00

Using Newton's Forward Interpolation :
f(2.500000) = 19.634937
Error = 0.000000
Using Newton's Backward Interpolation :
f(2.500000) = 19.634939
Error = 0.000002

14 Euler Method

Date: November 2, 2018

The given differential equation is

$$dy/dx = x + 2y$$

The actual equation is

$$y = 0.25e^{2x} - 0.5x - 0.25$$

Given:

$$y(0) = 0$$

$$h = 0.1$$

Find the value y'(1)

14.1 Code

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

float f(float a,float b)
{   //function for first order derivative at a and b
    return a+2*b;
}
float Y(float a)
{   //the given function
    return 0.25*exp(2*a)-0.5*a-0.25;
}
int main(){
    float x,y,h;
    x = 0; y = 0; h = 0.1;
    printf("\tEULER METHOD\n");
    printf("x\t\tycomputed\tyactual\t\tAbs.error\n");

    y += f(x,y)*h;                              //Euler iterative Formula
    printf("%f\t%f\t%f\t%f\n",x,y,Y(x),fabs(Y(x)-y));
    int i = 1;
    do{
        x += h;
        y += f(x,y)*h;
        printf("%f\t%f\t%f\t%f\n",x,y,Y(x),fabs(Y(x)-y));
        i++;
    }
    while(x <= 1);
    return 0;
}
```

14.2 Output

x	ycomputed	yactual	Abs.error
0.000000	0.000000	0.000000	0.000000
0.100000	0.010000	0.005351	0.004649
0.200000	0.032000	0.022956	0.009044
0.300000	0.068400	0.055530	0.012870
0.400000	0.122080	0.106385	0.015695
0.500000	0.196496	0.179570	0.016926
0.600000	0.295795	0.280029	0.015766
0.700000	0.424954	0.413800	0.011154
0.800000	0.589945	0.588258	0.001687
0.900000	0.797934	0.812412	0.014478
1.000000	1.057521	1.097264	0.039743

15 Modified Euler Method

Date: November 2, 2018

The given differential equation is

$$dy/dx = x + 2y$$

The actual equation is

$$y = 0.25e^{2x} - 0.5x - 0.25$$

Given:

$$y(0) = 0$$

$$h = 0.1$$

Find the value y'(1)

15.1 Code

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

float f(float a,float b)
{   //function for first order derivative at a and b
    return a+2*b;
}
float Y(float a)
{   //the given function
    return 0.25*exp(2*a)-0.5*a-0.25;
}
int main(){
    float x,y1,y,y00,h;
    x = 0; y = 0; h = 0.1;
    printf("\tMODIFIED EULER METHOD\n");
    printf("x\t\tycomputed\tyactual\t\tAbs.error\n");

    y00 = y + h*f(x,y);
    y1 = y + (f(x,y)+f(x+h,y00))/2*h; //formula for Modified Euler's
        Method.
    //In this case final slope will be the average of f(x,y) and f(x+h,y)
    x += h;
    printf("%f\t%f\t%f\t%f\n",x,y1,Y(x),fabs(Y(x)-y1));

    int i = 1;
    do{
        y = y1;
        y00 = y + h*f(x,y);
```

```
        y1 = y + (f(x,y)+f(x+h,y00))/2*h;
        x += h;
        printf("%f\t%f\t%f\t%f\n",x,y1,Y(x),fabs(Y(x)-y1));

        i++;
    }
    while(x <= 1);
    return 0;
}
```

15.2 Output

x	ycomputed	yactual	Abs.error
0.100000	0.005000	0.005351	0.000351
0.200000	0.022100	0.022956	0.000856
0.300000	0.053962	0.055530	0.001568
0.400000	0.103834	0.106385	0.002552
0.500000	0.175677	0.179570	0.003893
0.600000	0.274326	0.280029	0.005703
0.700000	0.405678	0.413800	0.008122
0.800000	0.576927	0.588258	0.011331
0.900000	0.796851	0.812412	0.015561
1.000000	1.076158	1.097264	0.021106

16 Trapezoidal Method

Date: November 2, 2018

Find the value of Definite Integral of the following equation using trapezoidal method

$$\int_0^1 \frac{1}{1+x}\,dx$$

16.1 Code

```
#include <stdio.h>
#include <stdlib.h>
#include<math.h>
float function(float x) //the given function
{
    return 1/(1+x);
}
float trapezoidal(float a,float b,float precision) //function to find
     the value of the integral using trapezoidal method
{
    int iteration=1; //to show the iteration
```

```
    float x,h=b-a,term=(function(a)+function(b))/2,sum0,sum1,err;
    sum1=h*term; //current sum
    printf("Iteration\tInterval\tPrevious_sum\tCurrent_sum\tError\n");
    do{
        sum0=sum1; // putting the value of current sum to previous sum
        sum1=0; //making current sum zero
        h/=2;
        for(x=a+h;x<b;x+=h)
            sum1+=function(x);
        sum1+=term;
        sum1*=h;
        err=fabs(sum1-sum0);
        printf("%d\t\t%ld\t\t%f\t%f\t%f\n",iteration++,(long)((b-a)/h),sum0,sum1,err);
    }while(err>precision);
    return sum1;
}
int main()
{
    float lower,upper,precision,sum;
    printf("Enter the lower bound, upper bound and precision:\n");
    scanf("%f%f%f",&lower,&upper,&precision);
    sum=trapezoidal(lower,upper,precision);
    printf("\nValue of integral= %f",sum);
    return 0;
}
```

16.2 Output

Iteration	Interval	Previous sum	Current sum	Error
1	2	0.750000	0.708333	0.041667
2	4	0.708333	0.697024	0.011310
3	8	0.697024	0.694122	0.002902
4	16	0.694122	0.693391	0.000731
5	32	0.693391	0.693208	0.000183
6	64	0.693208	0.693163	0.000046
7	128	0.693163	0.693151	0.000012
8	256	0.693151	0.693148	0.000003
9	512	0.693148	0.693147	0.000001

Value of integral = 0.693147

17 Simpson Method

Date: November 2, 2018

Find the value of Definite Integral of the following equation using Simpson method

$$\int_0^1 \frac{1}{1+x}\,dx$$

17.1 Code

```
#include <stdio.h>
#include <stdlib.h>
#include<math.h>
float function(float x) //the given function
{
    return 1/(1+x);
}
float simpsons(float a,float b,float precision) //function to find the
    value of the integral using simpson method
{

   int iteration=1; //to show the iteration
   float x,width,h=b-a,sum0,sum1,error;
   width=h;
   sum1=(h/6)*(function(a)+4*function((a+b)/2)+function(b)); //current
       sum
   printf("\n");
   printf("Iterations\tPartitions\tPrevious Sum\tCurrent Sum\tError\n");

   printf("\n");
   do
   {
      sum0=sum1; // putting the value of current sum to previous sum
      sum1=0; //making current sum zero
      width/=2;
      for(x=a;x<=b-width;x+=width)
         sum1+=function(x)+4*function(x+(width/2))+function(x+width);
      sum1=sum1*width/6;
      error=fabs(sum1-sum0);
      printf("%d\t\t%ld\t\t%f\t%f\t%f\t\n",iteration++,(long)((b-a)/width),sum0,sum1,error);
   }while(error>precision);
   printf("\n");
   return sum1;
}
int main()
{
    float lower,upper,precision,sum;
    printf("Enter the lower bound, upper bound and precision:\n");
    scanf("%f%f%f",&lower,&upper,&precision);
    sum=simpsons(lower,upper,precision);
    printf("\nValue of the integral =%f",sum);
    return 0;
}
```

17.2 Output

Iterations	Partitions	Previous Sum	Current Sum	Error
1	2	0.694445	0.693254	0.001191
2	4	0.693254	0.693155	0.000099
3	8	0.693155	0.693148	0.000007
4	16	0.693148	0.693147	0.000000

Value of the integral = 0.693147