Practical file submitted in partial fulfillment for the evaluation of

"Computational Methods Lab"

Submitted By:

Student Name:

Enrolment no:

Branch & Section: Computer Science and Engineering 'A'

Submitted To:

- Name of Faculty: Dr. Bharti Seth

Index

S.No	Experiment Title (GGSIPU)	Page No.	Date	Sign
5	Write a program to calculate a point of minima for a given function and an interval. The search should continue until a pre-assigned error bound is reached but not beyond 100 in each case. Run the program for f(x) = |ln x| on [0.5, 4].			
6	Write a program to predict this year's rainfall using the Lagrange Interpolation method. Use the annual rainfall data for the last 10 years in your city. Is the answer obtained reasonable?			
7	Write a program to verify that 36 subintervals are required to evaluate $\int_0^1 e^{-x^2} dx$ with atmost error of $0.5*10^{-4}$ for the composite Trapezoidal Rule.			
8	Write a program to solve *x = 100 (sin t - x)* with initial value x(0)=0 using fourth order Runge-Kutta method on the interval [0 3] with step sizes 0.015, 0.020, 0.025, 0.030. Observe the numerical instability.			
9	Write a program for Fibonacci Search Algorithm. Run the code for $f(x) = x^2 + sin(53x)$ to find the minima of the function near the origin.			

10	Write a program to find the largest and smallest eigenvalue of the matrix [− 57 192 148 20 −53 −44 −48 144 115] using the Power method.			

EXERCISE 1

Q1. Write a C++ program which prints all odd positive integers less than 100, omitting those integers divisible by 7.

Programming Code:

```
#include<iostream>
using namespace std;
int main()
{
   int i,c=0;
   cout<<"Odd numbers less than 100 not divisble by 7 are:"<<endl;
   for(i=1;i<100;i=i+2)
   {
      if(i%7!=0)
      {
         cout<<i<<"\t";
         c++;
      }
      if(c==7) //print in tabular format
      {
         cout<<endl;
         c=0;
      }
   }
   cout<<endl;
   return 0;
}
```

Output:

```
Odd numbers less than 100 not divisble by 7 are:
1   3   5   9   11  13  15
17  19  23  25  27  29  31
33  37  39  41  43  45  47
51  53  55  57  59  61  65
67  69  71  73  75  79  81
83  85  87  89  93  95  97
99
Program ended with exit code: 0
```

Q2. Determine the number of integers between 1 and 2000, that are not divisible by 2, 3 or 5 but are divisible by 7.

Programming Code:

```
#include<iostream>
using namespace std;
int main()
{
   int i,c=0;
   cout<<"Positive integers less than 2000 that are not divisible by 2, 3 or 5 but
are divisble by 7 are: "<<endl;

   for(i=7;i<2000;i=i+7)
   {
      if(i%2!=0 && i%3!=0 && i%5!=0)
      {
         cout<<i<<"\t";
         c++;
      }
      if(c%8==0) //print in tabular format
         cout<<endl;
   }
   cout<<"\n\nTotal such numbers are: "<<c<<endl;
   return 0;
}
```

Output:

```
Positive integers less than 2000 that are not divisible by 2, 3 or 5 but are divisble by 7 are:
7       49      77      91      119     133     161     203

217     259     287     301     329     343     371     413

427     469     497     511     539     553     581     623

637     679     707     721     749     763     791     833

847     889     917     931     959     973     1001    1043

1057    1099    1127    1141    1169    1183    1211    1253

1267    1309    1337    1351    1379    1393    1421    1463

1477    1519    1547    1561    1589    1603    1631    1673

1687    1729    1757    1771    1799    1813    1841    1883

1897    1939    1967    1981

Total such numbers are: 76
```

Q3. Write a complete C++ program to (i) Add two matrices (ii) Multiply two matrices.

Programming Code:

```
#include<iostream>
using namespace std;
int main()
{
   int m,n,x,y,i,j,k;
   cout<<"Enter number of rows and columns for matrix 1: ";
   cin>>m>>n;
   int a[m][n];
   cout<<"Enter elements of matrix 1:"<<endl;
   for(i=0;i<m;i++)
      for(j=0;j<n;j++)
         cin>>a[i][j];
   cout<<"Enter number of rows and columns for matrix 2: ";
   cin>>x>>y;
   int b[x][y];
   cout<<"Enter elements of matrix 2:"<<endl;
   for(i=0;i<x;i++)
      for(j=0;j<y;j++)
         cin>>b[i][j];

   //addition of matrices
   if(m==x && n==y)
   {
      int sum[m][n];
      for(i=0;i<m;i++)
         for(j=0;j<n;j++)
            sum[i][j]=a[i][j]+b[i][j];
      cout<<"\nAddition of matrices is:"<<endl;
      for(i=0;i<m;i++)
      {
         for(j=0;j<n;j++)
            cout<<sum[i][j]<<"\t";
         cout<<endl;
      }
   }
   else
      cout<<"The number of rows and columns of matrices 1 and 2 must match
for addition"<<endl;
```

```
//multiplication
if(n==x)
{
   int pro[m][y];
   for(i=0;i<m;i++)
      for(j=0;j<y;j++)
      {
         pro[i][j]=0;
         for(k=0;k<n;k++)
            pro[i][j]+=a[i][k]+b[k][j];
      }
   cout<<"\nProduct of matrices is:"<<endl;
   for(i=0;i<m;i++)
   {
      for(j=0;j<y;j++)
         cout<<pro[i][j]<<"\t";
      cout<<endl;
   }
}
else
   cout<<"Number of columns of matrix 1 must be equal to number of rows of
matrix 2 for multiplication"<<endl;

return 0;
}
```

Output:

```
Enter number of rows and columns for matrix 1: 3 3
Enter elemts of matrix 1:
1 2 3
4 5 6
7 8 9
Enter number of rows and columns for matrix 2: 3 3
Enter elemts of matrix 2:
2 3 4
5 6 7
8 9 1

Addition of matrices is:
3   5   7
9   11  13
15  17  10

Product of matrices is:
21  24  18
30  33  27
39  42  36
Program ended with exit code: 0
```

EXERCISE 2

Problem statement: Write a program to find a root of the equation x^3 -8x -3=0 using bisection method with 6 decimal digits

Programming code:

```
#include<iostream>
#include<iomanip>
using namespace std;

#define tol 0.000001

int ite1=0;
double lastMid=0;

double fx(double x) {
   return x*x*x-8*x-3;
}


double bsct(double a, double b)
{
   double mid=(a+b)/2;

cout<<fixed<<setprecision(6)<<++ite1<<"\t"<<a<<"\t"<<b<<"\t"<<mid<<"\t"<<fx(mid)<<"\
t"<<fabs(lastMid-mid)<<endl;
   if(fx(mid)==0 || fabs(fx(mid))<=tol)
   {
      return mid;
   }
   else if(fabs(lastMid-mid)<=tol)
   {
      return mid;
   }
   else if(fabs(fx(mid))<=tol)
   {
      return mid;
   }
   else if((fx(a)*fx(mid))<0)
   {
      lastMid=mid;
      return bsct(a,mid);
   }
   else if((fx(mid)*fx(b))<0)
   {
      lastMid=mid;
      return bsct(mid,b);
   }
}
```

```
int main()
{
   double m,n,ia,root,root1;

   cout<<"For BISECTION Method \nEnter initial interval a b ";
   cin>>m>>n;
   if(fx(m)==0)
   cout<<"Root is "<<m<<endl;
   else if(fx(n)==0)
   cout<<"Root is "<<n<<endl;
   else if((fx(m)*fx(n))<0)
   {
      cout<<"\nite\ta\t\tb\t\tmid\t\tfx(mid))\terror"<<endl;
      root=bsct(m,n);
      cout<<"\nRoot by BISECTION Method is: "<<root<<endl<<endl;
   }
   else
   cout<<"Sign is same at both values, try another interval";

   return 0;
}
```

Output:

```
For BISECTION Method
Enter initial interval a b 2 5

ite     a               b               mid             fx(mid))        error
1       2.000000        5.000000        3.500000        11.875000       3.500000
2       2.000000        3.500000        2.750000        -4.203125       0.750000
3       2.750000        3.500000        3.125000        2.517578        0.375000
4       2.750000        3.125000        2.937500        -1.152588       0.187500
5       2.937500        3.125000        3.031250        0.602570        0.093750
6       2.937500        3.031250        2.984375        -0.294682       0.046875
7       2.984375        3.031250        3.007812        0.148987        0.023438
8       2.984375        3.007812        2.996094        -0.074081       0.011719
9       2.996094        3.007812        3.001953        0.037144        0.005859
10      2.996094        3.001953        2.999023        -0.018546       0.002930
11      2.999023        3.001953        3.000488        0.009279        0.001465
12      2.999023        3.000488        2.999756        -0.004638       0.000732
13      2.999756        3.000488        3.000122        0.002319        0.000366
14      2.999756        3.000122        2.999939        -0.001160       0.000183
15      2.999939        3.000122        3.000031        0.000580        0.000092
16      2.999939        3.000031        2.999985        -0.000290       0.000046
17      2.999985        3.000031        3.000008        0.000145        0.000023
18      2.999985        3.000008        2.999996        -0.000072       0.000011
19      2.999996        3.000008        3.000002        0.000036        0.000006
20      2.999996        3.000002        2.999999        -0.000018       0.000003
21      2.999999        3.000002        3.000000        0.000009        0.000001
22      2.999999        3.000000        3.000000        -0.000005       0.000001

Root by BISECTION Method is: 3.000000
```

EXERCISE 3

Problem statement: Write a program to solve for a root of the equation e^(-x^2) = cosx + 1 on [0,4]. Run the program thrice for
x0 = 0
x0 = 1
0 < x0 < 4
Error=0.00001

Programming code:

```
#include<iostream>
#include<iomanip>
#include<cmath>
using namespace std;

#define tol 0.00001

int ite1=1;

double fx(double x) {
   return exp(-x*x)-cos(x)-1;
}

double dfx(double x) {
   return (-2*x*exp(-(x*x))+sin(x));
}

double newt(double x)
{
   double rt=x-(fx(x)/dfx(x));
   cout<<fixed<<setprecision(5)<<ite1++<<" \t "<<x<<" \t "<<fx(x)<<" \t "<<dfx(x)<<" \t
"<<rt<<" \t "<<fabs(x-rt)<<" \t "<<fx(rt)<<endl;
   if(fx(rt)==0 || fabs(fx(rt))<=tol)
   return rt;
   else if(fabs(x-rt)<=tol)
   return rt;
   else if(dfx(rt)==0)
   return x;
   else
   return newt(rt);
}

int main()
{
   int i;
   double ia,root;
   cout<<"For NEWTON-RAPHSON Method"<<endl;
   for(i=1;i<=3;i++)
   {
```

```
    ite1=1;
    cout<<"Enter initial approximation ";
    cin>>ia;
  if(dfx(ia)==0)
  {
    cout<<"Not defined"<<endl<<endl;
  }
  else
  {
    cout<<"\nite \t x \t\t f(x) \t\t f'(x) \t\t root \t\t error \t\t fx(rt)"<<endl;
    root=newt(ia);
    cout<<"\nRoot by NEWTON RAPHSON Method is: "<<root<<endl<<endl;
  }
  }
  return 0;
}
```

Output:

```
For NEWTON-RAPHSON Method
Enter initial approximation 0
Not defined

Enter initial approximation 1

ite     x          f(x)       f'(x)      root       error      fx(rt)
1       1.00000    -1.17242   0.10571    12.09072   11.09072   -1.88899
2       12.09072   -1.88899   -0.45792   7.96556    4.12515    -0.88865
3       7.96556    -0.88865   0.99378    8.85977    0.89421    -0.15541
4       8.85977    -0.15541   0.53542    9.15004    0.29027    -0.03750
5       9.15004    -0.03750   0.27130    9.28828    0.13824    -0.00930
6       9.28828    -0.00930   0.13608    9.35663    0.06836    -0.00232
7       9.35663    -0.00232   0.06809    9.39072    0.03408    -0.00058
8       9.39072    -0.00058   0.03405    9.40775    0.01703    -0.00014
9       9.40775    -0.00014   0.01703    9.41626    0.00851    -0.00004
10      9.41626    -0.00004   0.00851    9.42052    0.00426    -0.00001

Root by NEWTON RAPHSON Method is: 9.42052

Enter initial approximation 2

ite     x          f(x)       f'(x)      root       error      fx(rt)
1       2.00000    -0.56554   0.83603    2.67645    0.67645    -0.10547
2       2.67645    -0.10547   0.44440    2.91378    0.23732    -0.02563
3       2.91378    -0.02563   0.22465    3.02787    0.11410    -0.00635
4       3.02787    -0.00635   0.11284    3.08419    0.05631    -0.00157
5       3.08419    -0.00157   0.05692    3.11183    0.02764    -0.00038
6       3.11183    -0.00038   0.02937    3.12479    0.01296    -0.00008
7       3.12479    -0.00008   0.01645    3.12988    0.00509    -0.00001
8       3.12988    -0.00001   0.01136    3.13102    0.00114    -0.00000

Root by NEWTON RAPHSON Method is: 3.13102
```

EXERCISE 4

Problem statement: Write a program to find the root of x^3 + 10x -20 using secant method with starting values x0 = 2 and x1= 1.
Compare the number of iteration required for given f(x) using secant & Newton's Method
Error=0.00001

Programming code:

```
#include<iostream>
#include<iomanip>
using namespace std;

#define tol 0.00001

int ite1=0,ite2=0;
double lastMid=0;

double fx(double x) {
   return x*x*x + 2*x*x + 10*x - 20;
}

double dfx(double x)
{
   return 3*x*x + 4*x + 10;
}

double secant(double x0, double x1)
{
   double x2=(x0*fx(x1)-x1*fx(x0))/(fx(x1)-fx(x0));
   cout<<fixed<<setprecision(5)<<++ite1<<" \t "<<x0<<" \t "<<x1<<" \t "<<x2<<" \t
"<<fx(x2)<<" \t "<<fabs(x1-x2)<<endl;
   if(fabs(x1-x2)<=tol || fabs(fx(x2))<=tol)
   {
      return x2;
   }
   else if((fx(x1)-fx(x2)==0))
   cout<<"\nMethod not defined for "<<x1<<" & "<<x2<<" values";
   else
   {
      return secant(x1,x2);
   }
}

double newt(double x)
{
   double rt=x-(fx(x)/dfx(x));
   cout<<fixed<<setprecision(5)<<++ite2<<" \t "<<x<<" \t "<<fx(x)<<" \t "<<dfx(x)<<" \t
"<<rt<<" \t "<<fabs(x-rt)<<" \t "<<fx(rt)<<endl;
   if(fx(rt)==0 || fabs(fx(rt))<=tol)
```

```
  return rt;
  else if(fabs(x-rt)<=tol)
  return rt;
  else if(dfx(rt)==0)
  return x;
  else
  return newt(rt);
}

int main()
{
  int m,n;
  cout<<"For SECANT Method, \nEnter values of x0 and x1 "<<endl;
  cin>>m>>n;
  if(fx(m)==0)
  cout<<"Root is "<<m<<endl;
  else if(fx(n)==0)
  cout<<"Root is "<<n<<endl;
  else if((fx(n)-fx(m))==0)
  cout<<"Method not defined for these values";
  else
  {
    cout<<"\nite \t x(n-1) \t x(n) \t\t x(n+1) \t f(x(n+1)) \t error\n";
    double sroot=secant(m,n);
    cout<<"\nRoot by SECANT Method is: "<<sroot<<endl<<endl;
  }

  cout<<"\nBy NEWTON RAPHSON\nite \t x \t\t f(x) \t\t f'(x) \t\t root \t\t error \t\t
fx(rt)"<<endl;
  double nroot=newt(m);
  cout<<"Root by NEWTON RAPHSON Method is: "<<nroot<<endl<<endl;

  cout<<"Number of iterations for SECANT Method is: "<<ite1<<endl;
  cout<<"Number of iterations for NEWTON RAPHSON Method is: "<<ite2<<endl;
  cout<<"Difference in iterations= "<<abs(ite1-ite2)<<endl<<endl;

  return 0;
}
```

Output:

```
For SECANT Method,
Enter values of x0 and x1
2 1

ite     x(n-1)          x(n)            x(n+1)          f(x(n+1))       error
1       2.00000         1.00000         1.30435         -1.33476        0.30435
2       1.00000         1.30435         1.37605         0.15317         0.07171
3       1.30435         1.37605         1.36867         -0.00287        0.00738
4       1.37605         1.36867         1.36881         -0.00001        0.00014

Root by SECANT Method is: 1.36881


By NEWTON RAPHSON
ite     x               f(x)            f'(x)           root            error           fx(rt)
1       2.00000         16.00000        30.00000        1.46667         0.53333         2.12385
2       1.46667         2.12385         22.32000        1.37151         0.09515         0.05709
3       1.37151         0.05709         21.12918        1.36881         0.00270         0.00004
4       1.36881         0.00004         21.09617        1.36881         0.00000         0.00000
Root by NEWTON RAPHSON Method is: 1.36881

Number of iterations for SECANT Method is: 4
Number of iterations for NEWTON RAPHSON Method is: 4
Difference in iterations= 0
```

EXERCISE 5

Problem Statement:

Write a program to calculate a point of minima for a given function and an interval. The search should continue until a pre-assigned error bound is reached but not beyond 100 in each case. Run the program for f(x) = |ln x| on [0.5, 4].

Programming Code:

```
#include<iostream>
#include<cmath>
#include<iomanip>
using namespace std;

#define r 0.618034
#define r2 0.381966
#define error 0.0001

int ite=0;
double mid=0,prevmid=0;

double f(double x)
{
  return (fabs(log(x)));
}

double golden(double a, double b)
{
double x,y;
mid=(a-b)/2;
if((fabs(prevmid-mid)<error) || (ite==100))
return a;
x=a+r*(b-a);
y=a+r2*(b-a);
cout<<fixed<<setprecision(3)<<++ite<<" \t "<<a<<" \t "<<b<<" \t "<<x<<" \t "<<y<<" \t
"<<f(x)<<" \t "<<f(y)<<" \t ";
cout<<fixed<<setprecision(4)<<fabs(prevmid-mid)<<endl;
prevmid=mid;
if(f(x)>f(y))
return golden(a,x);
else
return golden(y,b);
}

int main()
{
double a,b;
cout<<"\nFor GOLDEN SECTION SEARCH Method:";
cout<<"\nEnter initial values of a and b: ";
```

```
cin>>a>>b;
cout<<"ite\t a\tb\tx\ty\tf(x)\tf(y)\terror\n";
double pmin=golden(a,b);
cout<<"\nNumber of iterations= "<<ite<<endl;
cout<<fixed<<setprecision(3)<<"Point of Minima= "<<pmin<<endl;
cout<<"Minima= "<<f(pmin)<<endl;

cout<<endl;
return 0;
}
```

Output:

```
For GOLDEN SECTION SEARCH Method:
Enter initial values of a and b: 0.5
4
ite     a       b       x       y       f(x)    f(y)    error
1       0.500   4.000   2.663   1.837   0.979   0.608   1.7500
2       0.500   2.663   1.837   1.326   0.608   0.282   0.6684
3       0.500   1.837   1.326   1.011   0.282   0.011   0.4131
4       0.500   1.326   1.011   0.816   0.011   0.204   0.2553
5       0.816   1.326   1.131   1.011   0.123   0.011   0.1578
6       0.816   1.131   1.011   0.936   0.011   0.066   0.0975
7       0.936   1.131   1.057   1.011   0.055   0.011   0.0603
8       0.936   1.057   1.011   0.982   0.011   0.018   0.0373
9       0.982   1.057   1.028   1.011   0.028   0.011   0.0230
10      0.982   1.028   1.011   1.000   0.011   0.000   0.0142
11      0.982   1.011   1.000   0.993   0.000   0.007   0.0088
12      0.993   1.011   1.004   1.000   0.004   0.000   0.0054
13      0.993   1.004   1.000   0.997   0.000   0.003   0.0034
14      0.997   1.004   1.001   1.000   0.001   0.000   0.0021
15      0.997   1.001   1.000   0.999   0.000   0.001   0.0013
16      0.999   1.001   1.000   1.000   0.000   0.000   0.0008
17      0.999   1.000   1.000   0.999   0.000   0.001   0.0005
18      0.999   1.000   1.000   1.000   0.000   0.000   0.0003
19      1.000   1.000   1.000   1.000   0.000   0.000   0.0002
20      1.000   1.000   1.000   1.000   0.000   0.000   0.0001

Number of iterations= 20
Point of Minima= 1.000
Minima= 0.000
```

Learning Outcomes:

EXERCISE 6

Problem Statement:

Write a program to predict this year's rainfall using the Lagrange Interpolation method. Use the annual rainfall data for the last 10 years in your city. Is the answer obtained reasonable?

Programming Code:

```
#include<iostream>
#include<cmath>
using namespace std;

int main()
{
   cout<<"Calculation of data by LEGRANGE INTERPOLATION:\n\n";
   int n;
   cout<<"Enter number of data points: ";
   cin>>n;
   double x[n];
   double y[n];
   double l[n];
   double X,P=0;
   cout<<"Enter value of:\n";
   for(int i=0;i<n;i++)
   {
      cout<<"x"<<i<<": ";
      cin>>x[i];
      cout<<"y"<<i<<": ";
      cin>>y[i];
      cout<<endl;
   }
   cout<<"Enter value of x at which data is to be assumed: ";
   cin>>X;

   cout<<"x \t y \t L"<<endl;

   for(int i=0;i<n;i++)
   {
      l[i]=1;
      for(int j=0;j<n;j++)
         if(i!=j)
            l[i] = l[i] * ((X-x[j]) / (x[i]-x[j]));

      P=P + (l[i]*y[i]);

      cout<<x[i]<<" \t "<<y[i]<<" \t "<<l[i]<<endl;
   }

   cout<<"\nThe interpolated value at "<<X<<" is: "<<P<<endl;
```

```
    cout<<endl;
    return 0;
}
```

Output:

```
Calculation of data by LEGRANGE INTERPOLATION:

Enter number of data points: 10
Enter value of:
x0: 2013
y0: 46.32

x1: 2014
y1: 53.79

x2: 2015
y2: 40.97

x3: 2016
y3: 42.17

x4: 2017
y4: 45.04

x5: 2018
y5: 65.55

x6: 2019
y6: 53.03

x7: 2020
y7: 45.35

x8: 2021
y8: 59.73

x9: 2022
y9: 46.30

Enter value of x at which data is to be assumed: 2023
x         y        L
2013      46.32    -1
2014      53.79    10
2015      40.97    -45
2016      42.17    120
2017      45.04    -210
2018      65.55    252
2019      53.03    -210
2020      45.35    120
2021      59.73    -45
2022      46.3     10

The interpolated value at 2023 is: 2849.38
```

Learning Outcomes:

EXERCISE 7

Problem Statement:

Write a program to verify that 36 subintervals are required to evaluate $\int_0^1 e^{-x^2} dx$ with atmost error of $0.5*10^{-4}$ for the composite Trapezoidal Rule.

Programming Code:

```
#include<iostream>
#include<cmath>
#include<iomanip>
using namespace std;
double error=0.00005;
double f(double x)
{
   return exp(-x*x);
}

double I(double a, double b, double h, int n)
{

   double val=0;
   for(int i=1;i<n;i++)
   {
      val += f(a+i*h);
   }
   return h/2*(f(a)+f(b)+2*(val));
}

int main()
{
   cout<<"Using COMPOSITE TRAPEZOIDAL RULE:"<<endl;
   int n=0;
   double a=0,b=1,Integral=0;
   double h=(b-a)/n;
   cout<<"interval \t h \t\t Integrated value \t error"<<endl;
   while ((fabs(Integral-0.746824132812))>error)
   {
      n++;
      h=(b-a)/n;
      Integral=I(a,b,h,n);
      cout<<fixed<<setprecision(5)<<n<<" \t\t "<<h<<" \t "<<Integral<<" \t\t
"<<fabs(Integral-0.746824132812)<<endl;
   }
   cout<<"\nNumber of intervals required to evaluate is : "<<n<<endl<<endl;
   return 0;
}
```

Output:

```
Using COMPOSITE TRAPEZOIDAL RULE:
interval          h                Integrated value        error
1                 1.00000          0.68394                 0.06288
2                 0.50000          0.73137                 0.01545
3                 0.33333          0.73999                 0.00684
4                 0.25000          0.74298                 0.00384
5                 0.20000          0.74437                 0.00246
6                 0.16667          0.74512                 0.00170
7                 0.14286          0.74557                 0.00125
8                 0.12500          0.74587                 0.00096
9                 0.11111          0.74607                 0.00076
10                0.10000          0.74621                 0.00061
11                0.09091          0.74632                 0.00051
12                0.08333          0.74640                 0.00043
13                0.07692          0.74646                 0.00036
14                0.07143          0.74651                 0.00031
15                0.06667          0.74655                 0.00027
16                0.06250          0.74658                 0.00024
17                0.05882          0.74661                 0.00021
18                0.05556          0.74663                 0.00019
19                0.05263          0.74665                 0.00017
20                0.05000          0.74667                 0.00015
21                0.04762          0.74669                 0.00014
22                0.04545          0.74670                 0.00013
23                0.04348          0.74671                 0.00012
24                0.04167          0.74672                 0.00011
25                0.04000          0.74673                 0.00010
26                0.03846          0.74673                 0.00009
27                0.03704          0.74674                 0.00008
28                0.03571          0.74675                 0.00008
29                0.03448          0.74675                 0.00007
30                0.03333          0.74676                 0.00007
31                0.03226          0.74676                 0.00006
32                0.03125          0.74676                 0.00006
33                0.03030          0.74677                 0.00006
34                0.02941          0.74677                 0.00005
35                0.02857          0.74677                 0.00005
36                0.02778          0.74678                 0.00005

Number of intervals required to evaluate is : 36
```

Learning Outcomes:

EXERCISE 8

Problem Statement:

Write a program to solve $x = 100\ (\sin t - x\ )$ with initial value x(0)=0 using fourth order Runge-Kutta method on the interval [0 3] with step sizes 0.015, 0.020, 0.025, 0.030. Observe the numerical instability.

Programming Code:

```
#include<iostream>
#include<math.h>
# define PI 3.14159265358979323846 /* pi */
using namespace std;


long double func(long double t, long double x){
    return 100*(sin(t) - x);
}
// long double func(long double t, long double x){
//     return t*t - x;
// }


long double Archae(long double h){
    long double end = 3.0;
    long double x = 0.0;
    long double t0 = 0.0;
    int n = ((end - x)/h);
    // cout<<"Total iterations are:"<<n<<endl;
    int i = 0;
    for(; i < n; i++){
        long double k1 = h*func(t0, x);
        long double k2 = h*func((t0 + h/2), (x + k1/2));
        long double k3 = h*func((t0 + h/2), (x + k2/2));
        long double k4 = h*func((t0 + h), (x + k3));
        x = x + ((1.0/6.0)*(k1 + 2.0*(k2 + k3) + k4));
        t0 = t0+h;
        // cout<<"For h = "<<h<<" Value of x"<<counter<<" is = "<<x<<endl;
    }
    cout<<"For h = "<<h<<" Value of x"<<i<<" is = "<<x<<endl;
}

int main(){
    long double h = 0;
    for(int i = 0; i < 4; i++){
        cout<<"Enter h : ";
        cin>>h;
        Archae(h);
    }
```

```
    return 0;
}
```

Output:

```
PS C:\Users\btech\OneDrive\Desktop\AA> cd "c:\Users\btech\OneDrive\Des
Enter h : 0.015
For h = 0.015 Value of x200 is = 0.151003
Enter h : 0.020
For h = 0.02 Value of x150 is = 0.150996
Enter h : 0.025
For h = 0.025 Value of x120 is = 0.150943
Enter h : 0.030
For h = 0.03 Value of x100 is = 6.72891e+011
PS C:\Users\btech\OneDrive\Desktop\AA>
```

Learning Outcomes:

EXERCISE 9

Problem Statement:

Write a program for Fibonacci Search Algorithm. Run the code for $f(x) = x^2 + sin(53x)$ to find the minima of the function near the origin.

Programming Code:

```
#include<iostream>
#include<cmath>
#include<iomanip>
using namespace std;

double f(double x){
   return (x*x + sin(53*x));
}

int main(){
   cout<<"\nMinima of Function using FIBONACCI SEARCH ALGORITHM METHOD : \n";
   double a=-0.2, b=0.2;
   double x,y;
   double D,d;
   int N=ceil(log((b-a)/(0.0001*1.618))/log(1.618))+1;
   cout<<"\nNumber of steps (N) = "<<N-1<<endl;

   int fib[N];
   fib[0]=0;
   fib[1]=1;
   for(int i=2;i<N;i++){
      fib[i]=fib[i-1]+fib[i-2];
   }
   cout<<"\nFibonacci Numbers used are : "<<endl;
   for(int i=0;i<N;i++){
      cout<<fib[i]<<" ";
   }
   cout<<endl<<endl;

   cout<<"k \t a \t\t b \t\t x \t\t y\n";
   int k;
   for(k=N-1;k>=2;k--){
      D=(1.0*fib[k-2]/fib[k])*(b-a);
      x=a+D;
      y=b-D;
      cout<<fixed<<setprecision(5)<<k<<" \t "<<a<<" \t "<<b<<" \t "<<x<<" \t "<<y<<endl;
      if(f(x)>=f(y))
      a=x;
      else
      b=y;
```

```
  }
  d=(y-x)/4;
  x=(a+b)/2-2*d;
  y=(a+b)/2+2*d;
  cout<<fixed<<setprecision(5)<<k--<<" \t "<<a<<" \t "<<b<<" \t "<<x<<" \t "<<y<<endl;
  if(f(x)>=f(y))
    a=x;
  else
    b=y;

  double xmin=(a+b)/2;
  double minima=f(xmin);

  cout<<"Minima is at (a+b)/2 = "<<xmin<<endl;
  cout<<"\nMinima of the function  near the origin at "<<xmin<<" is "<<minima;
  cout<<endl<<endl;
  return 0;
}
```

Output:

```
Minima of Function using FIBONACCI SEARCH ALGORITHM METHOD :

Number of steps (N) = 17

Fibonacci Numbers used are :
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597

k        a              b              x              y
17       -0.20000       0.20000        -0.04721       0.04721
16       -0.20000       0.04721        -0.10557       -0.04721
15       -0.10557       0.04721        -0.04721       -0.01115
14       -0.10557       -0.01115       -0.06951       -0.04721
13       -0.06951       -0.01115       -0.04721       -0.03344
12       -0.04721       -0.01115       -0.03344       -0.02492
11       -0.04721       -0.02492       -0.03870       -0.03344
10       -0.03870       -0.02492       -0.03344       -0.03018
9        -0.03344       -0.02492       -0.03018       -0.02818
8        -0.03344       -0.02818       -0.03143       -0.03018
7        -0.03143       -0.02818       -0.03018       -0.02943
6        -0.03018       -0.02818       -0.02943       -0.02893
5        -0.03018       -0.02893       -0.02968       -0.02943
4        -0.03018       -0.02943       -0.02993       -0.02968
3        -0.02993       -0.02943       -0.02968       -0.02968
2        -0.02968       -0.02943       -0.02968       -0.02943
1        -0.02968       -0.02943       -0.02968       -0.02943
Minima is at (a+b)/2 = -0.02956

Minima of the function  near the origin at -0.02956 is -0.99912
```

Learning Outcomes:

EXERCISE 10

Problem Statement:

Write a program to find the largest and smallest eigenvalue of the matrix

[− 57 192 148 20 −53 −44 −48 144 115] using the Power method.

Programming Code:

```
#include<iostream>
#include<cmath>
#include<vector>

using namespace std;

double largest_Eigen(double A[3][3])
{
   double x[3] = {1, 1, 1};
   double tol = 0.00001;
   int max_iter = 100;

   double lambda_old,lambda_new;
   lambda_old = x[0];
   int n=0;

   for (int i = 0; i < max_iter; i++)
   {
      double y[3] = {0, 0, 0};
      // Calculate the next approximation of the eigenvector
      for (int j = 0; j < 3; j++)
      {
         for (int k = 0; k < 3; k++)
         {
            y[j] += A[j][k] * x[k];
         }
      }

      // Calculate the largest eigenvalue
      lambda_new = y[0];
      for (int j = 1; j < 3; j++)
      {
         if (fabs(y[j]) > fabs(lambda_new))
         {
            lambda_new = y[j];
         }
      }

      // Normalize the eigenvector
      for (int j = 0; j < 3; j++)
```

```
        {
          x[j] = y[j] / lambda_new;
        }

        // Check for convergence
        if (fabs(lambda_new - lambda_old) < tol)
        {
          break;
        }
        lambda_old=lambda_new;
        n++;
    }
    cout<<"\nNumber of iterations required = "<<n<<endl;
    return lambda_new;
}

double determinant(double A[3][3]) {
    double result = 0;
    result += A[0][0] * (A[1][1] * A[2][2] - A[2][1] * A[1][2]);
    result -= A[0][1] * (A[1][0] * A[2][2] - A[2][0] * A[1][2]);
    result += A[0][2] * (A[1][0] * A[2][1] - A[2][0] * A[1][1]);
    return result;
}

void adjugate(double A[3][3], double adj[3][3]) {
    adj[0][0] = A[1][1] * A[2][2] - A[2][1] * A[1][2];
    adj[0][1] = A[0][2] * A[2][1] - A[0][1] * A[2][2];
    adj[0][2] = A[0][1] * A[1][2] - A[0][2] * A[1][1];
    adj[1][0] = A[1][2] * A[2][0] - A[1][0] * A[2][2];
    adj[1][1] = A[0][0] * A[2][2] - A[0][2] * A[2][0];
    adj[1][2] = A[1][0] * A[0][2] - A[0][0] * A[1][2];
    adj[2][0] = A[1][0] * A[2][1] - A[2][0] * A[1][1];
    adj[2][1] = A[2][0] * A[0][1] - A[0][0] * A[2][1];
    adj[2][2] = A[0][0] * A[1][1] - A[1][0] * A[0][1];
}

void inverse_matrix(double A[3][3], double inv_A[3][3]) {
    double det = determinant(A);
    if (det == 0) {
        cout<<"\nDeterminant is 0. Inverse can't be found\n";
        exit(0);
    }
    double adj[3][3];
    adjugate(A, adj);
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
            inv_A[i][j] = adj[i][j] / det;
        }
    }
}
```

```
int main()
{
   cout<<"\nUsing POWER METHOD : \n";
   double A[3][3] = {{-57, 192, 148}, {20, -53, -44}, {-48, 144, 115}};
   cout<<"\nGiven Matrix : \n";
   for(int i=0;i<3;i++)
   {
      for(int j=0;j<3;j++)
      cout<<A[i][j]<<" ";
      cout<<endl;
   }

   cout<<"\nFinding Largest Eigenvalue : ";
   double largestEigen=largest_Eigen(A);
   cout << "The largest eigenvalue of the matrix is : " << largestEigen <<endl;

   cout<<"\nFinding Smallest Eigenvalue : ";
   double inv_A[3][3];
   inverse_matrix(A,inv_A);
   double smallestEigen=1/largest_Eigen(inv_A);
   cout <<"The smallest eigenvalue of the matrix is : " << smallestEigen <<endl;

   cout<<endl;
   return 0;
}
```

Output:

```
Using POWER METHOD :

Given Matrix :
-57 192 148
20 -53 -44
-48 144 115

Finding Largest Eigenvalue :
Number of iterations required = 46
The largest eigenvalue of the matrix is : 7

Finding Smallest Eigenvalue :
Number of iterations required = 24
The smallest eigenvalue of the matrix is : 2.99998
```

Learning Outcomes: