Name	Mamun Mahmud
ID	C221046
Course Code	CSE-4746
Course Title	Numerical Methods Lab

Problems

1. Write a program to count number of significant digits in a given number.
2. Write a program to round off a number with n significant figures using banker's rule.
3. Write a program to evaluate a polynomial $f(x) = x^3 - 2x^2 + 5x + 10$ by using Horner's rule x = 5.
4. Write a program to find the root of the equation $x^3 – 9x + 1 = 0$, correct to 3 decimal places, by using the bisection method.
5. Write a program to find all the roots of the equation $x^3 - 6x + 4 = 0$, correct to 3 decimal places. [Use bisection method].
6. Write a program to find the root of the equation $x^3 - 6x + 4 = 0$, correct to 3 decimal places, by using Newton-Raphson method.
7. Write a program to find the root of the equation $x^3 - x + 2 = 0$, correct to 3 decimal places, by using false position method.
8. Write a program to find the root of the equation $x^3 – 5x^2 –29 = 0$, correct to 3 decimal places, by using secant method.
9. Write a program to find the *quotient polynomial* q(x) such that p(x) = (x - 2) q(x) where the polynomial $p(x) = x^3 - 5x^2 + 10x - 8 = 0$ has a root at x = 2.
10. Write a program to find all the roots of the equation $x^3 - 6x + 4 = 0$, correct to 3 decimal places. [Use Newton-Raphson method with deflation].

Solution 1:

```
#include <bits/stdc++.h>
using namespace std;
int cal()
{
    int i, f = 0, p;
    char ch;
    string sf = "", si = "", s;
    cout << "Enter the number (x.y type): ";
    cin >> s;
    for (auto a : s)
    {
        if (a == '.')
        {
            f = 1;
            continue;
        }
        if (!f)
            si += a;
        else
            sf += a;
    }
    if (!f)
    {
        while (!si.empty() and si.back() == '0')
            si.pop_back();
        cout << "Number of Significant digits is " << si.size() + sf.size() <<
endl;
        return 0;
    }
    while (!si.empty() and si[0] == '0')
        si.erase(si.begin());
    if (si.empty())
```

```
        while (!sf.empty() and sf[0] == '0')
            sf.erase(sf.begin());
    cout << "Number of Significant digits is " << si.size() + sf.size() << endl;
}
int32_t main()
{
    int t;
    cout << "Enter the number of test case:";
    cin >> t;
    while (t--)
    {
        cal();
    }
}
/*
TestCase:
4
45.003
00.0033
00330.003300
1200
*/
```

Solution 2:

```
/// You are given a number (eg: 2.995) and a point that indicates the digit after decimal point. Round to that point
#include <bits/stdc++.h>


using namespace std;


void print_final(string si, string sf, int p)
{
    /// remove leading zeros
    while (sf.size() > p)
        sf.pop_back();


    while (!si.empty() and si[0] == '0')
        si.erase(si.begin());
    if (sf.empty())
        sf = "0";
    cout << si << '.' << sf << endl;
}
int cal()
{
    int i, f = 0, p;
    char ch;
    cout << "Enter the point: ";
    cin >> p;
    cout << "Enter the number (x.y type): ";
    string sf = "", si = "", s;
    cin >> s;
    for (auto a : s)
    {
        if (a == '.')
        {
            f = 1;
```

```
            continue;
        }
        if (f)
            sf += a;
        else
            si += a;
    }

    if (sf.size() <= p)
    {
        cout << si << '.' << sf << endl;
        return 0;
    }
    /// Adding a leading '0' for easier calculation
    si = '0' + si;

    int t = sf[p] - '0';

    /// flag to control how far the rounding should traverse
    int fl = 0;

    if (t < 5 or t == 5 and (sf[p - 1] - '0') % 2 == 0)
    {
        print_final(si, sf, p);
        return 0;
    }
    if (t == 5 and (sf[p - 1] - '0') % 2 or t > 5)
    {
        int ptr = p - 1;
        while (ptr >= 0)
        {
            if (sf[ptr] == '9')
            {
```

```
                sf[ptr] = '0';
                --ptr;
                fl = 1;
                continue;
            }
            else
            {
                sf[ptr] += 1;
                fl = 0;
                break;
            }
        }
        //    ///when rounding comes to integer part
        if (fl)
        {
            int ptr = si.size() - 1;
            while (si[ptr] == '9')
            {
                si[ptr] = '0';
                --ptr;
            }
            ++si[ptr];
        }
        print_final(si, sf, p);
    }
}
int32_t main()
{
    int t;
    cout << "Enter the number of test case:";
    cin >> t;
    while (t--)
    {
```

```
        cal();
    }
}

/*
TC:
6
2
2.995
2
2.985
2
2.987
2
2.997
2
2.983
2
2.993
*/
```

Solution 3:

```
#include <bits/stdc++.h>

using namespace std;
#define ll long long

int main()
{
    string str1, str2;
    while (getline(cin, str1))
    {
        stringstream ss(str1);
        vector<ll> c, x;
        ll t;
        while (ss >> t)
            c.push_back(t);

        getline(cin, str2);
        stringstream ss2(str2);
        while (ss2 >> t)
            x.push_back(t);
        int i = 0;
        for (auto xx : x)
        {
            int n = c.size() - 1;
            ll sum = 0, p = c[0];
            for (int j = 1; j < c.size(); j++)
            {
                p = (p * xx) + c[j];
            }
            if (i)
                cout << ' ';
            cout << p;
```

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

Solution 4:

```
#include <bits/stdc++.h>
using namespace std;

double eps = 0.005;

double f_x(double x,vector<double>v)
{
    double ans = 0;
    for(int i = v.size() - 1;i >=0;i--)
    {
        if(v[i])
        {
            double t = 1;
            for( int p = 1 ;p <= i;p++)
                t *= x;
            ans += v[i] * t;
        }
    }
    return ans;
}
int main()
{
    int n ;
    cout<<"Enter the degree of the polynomial:";
    cin>>n;
    vector<double>c(n + 1);
    for( int  i = n ; i >= 0; i--)
    {
        cout<<"Enter the coefficient of x^"<<i<<" :";
        cin>>c[i];
    }
```

```
    double largest = 3.0;
    double smallest = 2.0;
    while (true)
    {
        double diff = fabs(largest - smallest);
        if (diff <= eps)
            break;
        double mid = (largest + smallest) / 2.00;
        if (f_x(mid,c) < 0.0)
            smallest = mid;
        else
            largest = mid;
    }
    cout << fixed << setprecision(9) << smallest << endl;
}
/*
Input:
3
1
0
-9
1
Output:
2.941406250
*/
```

Solution 5:

```
#include <bits/stdc++.h>
using namespace std;
double eps = 0.001;

double f_x(double x,vector<double>v)
{
    double ans = 0;
    for(int i = v.size() - 1;i >=0;i--)
    {
        if(v[i])
        {
            double t = 1;
            for( int p = 1 ;p <= i;p++)
                t *= x;
            ans += v[i] * t;
        }

    }
    return ans;
}

double bisection_root(double x1, double x2,vector<double>v)
{
    while(fabs(x1 - x2) > eps)
    {
        double mid = ( x1 + x2)/2.00;
        if(f_x(mid,v) * f_x(x1,v) < 0.0)
            x2 = mid;
        else
            x1 = mid;
    }
    return x2;
```

```
}

int main()
{
    int n ;
    cout<<"Enter the degree of the polynomial:";
    cin>>n;
    vector<double>c(n + 1);
    for( int  i = n ; i >= 0; i--)
    {
        cout<<"Enter the coefficient of x^"<<i<<" :";
        cin>>c[i];
    }
    double lower = -100, upper = 100, x = 1.0;///boundary and increment

    double x2 = lower, x1 = lower;

    while(x2 < upper)
    {
        x1 = lower, x2 = lower + x;
        double f1 = f_x(x1,c),f2 =f_x(x2,c);
        lower = x2 + 0.1;
        if((f1 * f2) > 0)
        {
            continue;
        }
        cout<<bisection_root(x1, x2,c)<<endl;
    }
}
/*
Input:
3
1
```

```
0
-6
4
Output:
-2.73125
0.732812
2.00078
*/
```

Solution 6:

```
#include<bits/stdc++.h>

using namespace std;
    /*Newton-Raphson Method

    ex: given f(x)= x^2 - 6x + 4
    thus, f'(x) = 2x - 6 (first derivative)
    fix two point , x1, and x2;
    asume x1 initially and find x2 by,
    x2 = x1 - f(x1) / f'(x1)

    replace x1 by x2 and find x2 again,
    repeat this process untill abs(x2 -x1 ) < E

    */

double E = .0005;

///find f(x1)
double f_x(double x1)
{
    return (x1 * x1) - (6 * x1) + 4;
}
///find f'(x1)
double FD_x(double x1)
{
    return (2 * x1) - 6.0;
}

int main()
{
    double x1 = 0;///assumption
```

```
    double x2 = x1 - f_x(x1)/ FD_x(x1);
    while(abs(x2 - x1) > E )
    {
       x1 = x2;
       x2 = x1 - f_x(x1)/ FD_x(x1);
    }
    cout<<fixed<<setprecision(4)<<x1<<endl;
}
```

Solution 7:

```
#include<bits/stdc++.h>


using namespace std;

/*

False Position method.

for root point(x,y), x = x0 and y = 0;

by placing root point to line equation((f (x2) − f (x1)) / (x2 − x1) = (y− f (x1)) / (x − x1))

we find,

x0 = x1− (f(x1) (x2−x1) ) / (f (x2) − f (x1))

repeat untill the absolute difference between two successive x0 is less then E

*/


double E = .00005;


double f_x(double x,vector<double>v)

{

    double ans = 0;

    for(int i = v.size() − 1;i >=0;i−−)

    {

        if(v[i])

        {

            double t = 1;

            for( int p = 1 ;p <= i;p++)

                t *= x;

            ans += v[i] * t;

        }


    }

    return ans;

}
```

```
int main()
{
    int n ;
    cout<<"Enter the degree of the polynomial:";
    cin>>n;
    vector<double>c(n + 1);
    for( int  i = n ; i >= 0; i--)
    {
        cout<<"Enter the coefficient of x^"<<i<<" :";
        cin>>c[i];
    }
    double x1 = -2.0, x2 = 1.0;

    double x0 = x1 - (f_x(x1,c) * (x2- x1))/ (f_x(x2,c) - f_x(x1,c));
    if(f_x(x1,c) * f_x(x0,c) < 0.0)
        x2 = x0;
    else x1 = x0;

    double x0_prev = x0;
    int cnt = 1;
    x0 = x1 - (f_x(x1,c) * (x2- x1))/ (f_x(x2,c) - f_x(x1,c));
    while(abs(x0_prev - x0) > E)
    {
        if(f_x(x1,c) * f_x(x0,c) < 0)
            x2 = x0;
        else x1 = x0;
        x0_prev = x0;
        x0 = x1 - (f_x(x1,c) * (x2- x1))/ (f_x(x2,c) - f_x(x1,c));
    }
    cout<<fixed<<setprecision(4)<<x0<<endl;
}
/*
```

```
Input:
3
1
0
-1
2
Output:
-1.5214
*/
```

Solution 8:

```
/*
Secant Method
1. Decide two initial points x1 and x2 and required accuracy level E.
2. Compute f1 = f (x1) and f2 = f (x2)
3. Compute x3 = x2 – (f2 * (x2 – x1)) / (f2 – f1);
4. If |x3– x2| > E, then
set x1 = x2 and f1 = f2
set x2 = x3 and f2 = f(x3)
go to step 3
Else
set root = x3
print results
5. Stop.
*/

#include <bits/stdc++.h>
using namespace std;

double eps = 0.001;
double f_x(double x,vector<double>v)
{
    double ans = 0;
    for(int i = v.size() – 1;i >=0;i--)
    {
        if(v[i])
        {
            double t = 1;
            for( int p = 1 ;p <= i;p++)
                t *= x;
            ans += v[i] * t;
        }
```

```
    }
    return ans;
}

int main()
{
int n ;
cout<<"Enter the degree of the polynomial:";
cin>>n;
vector<double>c(n + 1);
for( int  i = n ; i >= 0;i--)
{
    cout<<"Enter the coefficient of x^"<<i<<" :";
    cin>>c[i];
}
double x1 = 4.0, x2 = 2.0;//Initial estimate
double fx1 = f_x(x1,c), fx2 = f_x(x2,c);
double x3 = x2 - (fx2 * (x2 - x1)) / (fx2 - fx1);
while(abs(x3 - x2) > eps)
{
    x1 = x2;
    x2 = x3;
    fx1 = fx2;
    fx2 = f_x(x2,c);
    x3 = x2 - (fx2 * (x2 - x1)) / (fx2 - fx1);
}
cout<<fixed<<setprecision(3)<<x3<<endl;
}
/*
Input:
3
1
-5
```

```
0
-29
Output: 5.848
*/
```

Solution 9:

```
#include <bits/stdc++.h>
using namespace std;
/*
p(x) = (x - xr ) q(x),
bi-1 = ai + xr*bi where, bn = 0 and i = n, n-1, ……..1
*/

int main()
{
    int n;
    cout<<"Enter the degree of polynomial p(x):";
    cin>>n;
    vector<double>p(n + 1),q(n + 1);
    double xr;/// Has a root on this point
    for(int i = n ; i >= 0 ; i--)
    {
        cout<<"Enter the coefficients of x^"<<i<<" :";
        cin>>p[i];
    }
    cout<<"Enter Xr :";
    cin>>xr;
    q[n] = 0;
    for( int i = n - 1; i >= 0; i--)
    {
        q[i] = p[i + 1] + (xr * q[i + 1]);
    }
    cout<<"Quotient Polynomial is : ";
```

```
    for( int i = n ; i >= 0; i--)
    {
        if(q[i])
        {
            if(q[i] < 0 and i < (n - 1))
                cout<<"-";
            else if (i < (n - 1))
                cout<<"+";
            if(abs(q[i]) > 1)
                cout<<abs(q[i]);
            if(i)
                cout<<"x";
            if(i > 1)
            {
                cout<<"^"<<i;
            }
        }
    }
    cout<<endl;
}
/*
input:
3
1
-5
10
-8
2
x^2-3x+4
Output:
*/
```

Solution 10:

```
///Copy
///Copy

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

// Evaluate the polynomial a[0]*x^n + a[1]*x^(n-1) + ... + a[n] via Horner's method
double f(double x, const double a[], int n) {
    double val = a[0];
    for(int i = 1; i <= n; i++) {
        val = val * x + a[i];
    }
    return val;
}

// Evaluate the derivative f'(x) of the polynomial above
// f'(x) = n*a[0]*x^(n-1) + (n-1)*a[1]*x^(n-2) + ... + a[n-1]
double fprime(double x, const double a[], int n) {
    double val = n * a[0];
    for(int i = 1; i < n; i++) {
        val = val * x + (n - i) * a[i];
    }
    return val;
}

// Synthetic division to deflate the polynomial by (x - r)
// After division, the degree is reduced by 1 and the new coefficients overwrite
a[]
void syntheticDivision(double a[], int &n, double r) {
    // b[] will hold the result of synthetic division
```

```
    double b[20];  // Assume n <= 20 for simplicity
    b[0] = a[0];
    for(int i = 1; i <= n; i++) {
        b[i] = a[i] + r * b[i - 1];
    }
    // b[n] is the remainder (should be close to zero if r is a true root)
    // The quotient coefficients are b[0..n-1]
    n = n - 1; // One degree lower
    for(int i = 0; i <= n; i++) {
        a[i] = b[i]; // Overwrite original coefficients
    }
}

int main() {
    int n = 3; // Current polynomial degree
    double a[4] = {1.0, 0.0, -6.0, 4.0};
    double tol = 1e-7;
    int maxIter = 1000;
    double roots[3];
    int rootCount = 0;

    // Initial guess for the first root
    double x0 = 1.0;

    // Step through polynomial degrees until we reach a linear polynomial
    while(n > 1) {
        // Newton-Raphson to find one root of the current polynomial
        double x = x0;
        for(int i = 0; i < maxIter; i++) {
            double fx  = f(x, a, n);
            double dfx = fprime(x, a, n);
            if(fabs(dfx) < 1e-14) {
```

```
                cerr << "Derivative is too small; try another guess or check
polynomial." << endl;
                break;
            }
            double xNew = x - fx / dfx;
            if(fabs(xNew - x) < tol) {
                x = xNew;
                break;
            }
            x = xNew;
        }
        // Save this root
        roots[rootCount++] = x;

        // Deflate the polynomial by dividing out (x - root)
        syntheticDivision(a, n, x);

        // Use this root as initial guess for the next root
        x0 = x;
    }
    roots[rootCount++] = -a[1] / a[0];

    // Print all roots, to 3 decimal places
    cout << fixed << setprecision(3);
    for(int i = 0; i < 3; i++) {
        cout << "Root \" << (i + 1) << \" = " << roots[i] << endl;
    }

    return 0;
}
```