CSE 4746 Numerical Methods Lab
Lab Assignment-1

MD Faisal Hoque Rifat
ID: C221076

March 14, 2025

1. Count Number of Significant Digits

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

int countSignificantDigits(string number)
{
    int count = 0;
    bool significantDigitEncountered = false;

    for (char ch : number)
    {
        if (isdigit(ch))
        {
            if (ch != '0' || significantDigitEncountered)
            {
                significantDigitEncountered = true;
                count++;
            }
        }
    }

    return count;
}

int main()
{
    string number;
    cout << "Enter a number: ";
    cin >> number;
    cout << "Number of significant digits: " <<
        countSignificantDigits(number) << endl;
    return 0;
}
```

GitHub Link: Click here to view on GitHub

2. Round Off a Number with n Significant Figures Using Banker's Rule

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

double roundToSignificantFigures(double num, int n)
{
    if (num == 0.0) return 0.0;

    double d = ceil(log10(num < 0 ? -num : num));
    int power = n - static_cast<int>(d);

    double magnitude = pow(10.0, power);
    long shifted = round(num * magnitude);

    return shifted / magnitude;
}

int main()
{
    double number;
    int n;
    cout << "Enter a number: ";
    cin >> number;
    cout << "Enter number of significant figures: ";
    cin >> n;
    cout << "Rounded number: " << roundToSignificantFigures(
        number, n) << endl;
    return 0;
}
```

GitHub Link: Click here to view on GitHub

3. Evaluate Polynomial Using Horner's Rule

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

double evaluatePolynomial(double x)
{
    return ((x * x - 2) * x + 5) * x + 10;
}

```

```
int main()
{
    double x = 5.0;
    cout << "f(5) = " << evaluatePolynomial(x) << endl;
    return 0;
}
```

GitHub Link: Click here to view on GitHub

4. Root of Equation Using Bisection Method

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

double f(double x)
{
    return x * x * x - 9 * x + 1;
}

double bisection(double a, double b, double tol)
{
    double c;
    while ((b - a) / 2.0 > tol)
    {
        c = (a + b) / 2.0;
        if (f(c) == 0.0) break;
        else if (f(c) * f(a) < 0) b = c;
        else a = c;
    }
    return c;
}

int main()
{
    double a = 0, b = 1, tol = 0.001;
    cout << "Root: " << bisection(a, b, tol) << endl;
    return 0;
}
```

GitHub Link: Click here to view on GitHub

5. All Roots Using Bisection Method

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

double f(double x)
```

```cpp
{
    return x * x * x - 6 * x + 4;
}

double bisection(double a, double b, double tol)
{
    double c;
    while ((b - a) / 2.0 > tol)
    {
        c = (a + b) / 2.0;
        if (f(c) == 0.0) break;
        else if (f(c) * f(a) < 0) b = c;
        else a = c;
    }
    return c;
}

int main()
{
    double tol = 0.001;
    cout << "Root 1: " << bisection(0, 1, tol) << endl;
    cout << "Root 2: " << bisection(1, 2, tol) << endl;
    cout << "Root 3: " << bisection(2, 3, tol) << endl;
    return 0;
}
```

GitHub Link: Click here to view on GitHub

6. Root Using Newton-Raphson Method

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

double f(double x)
{
    return x * x * x - 6 * x + 4;
}

double f_prime(double x)
{
    return 3 * x * x - 6;
}

double newtonRaphson(double x0, double tol)
{
    double x1;
    while (true)
    {
```

```
        x1 = x0 - f(x0) / f_prime(x0);
        if (fabs(x1 - x0) < tol) break;
        x0 = x1;
    }
    return x1;
}

int main()
{
    double x0 = 1.5, tol = 0.001;
    cout << "Root: " << newtonRaphson(x0, tol) << endl;
    return 0;
}
```

GitHub Link: Click here to view on GitHub

7. Root Using False Position Method

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

double f(double x)
{
    return x * x * x - x + 2;
}

double falsePosition(double a, double b, double tol)
{
    double c;
    while ((b - a) > tol)
    {
        c = (a * f(b) - b * f(a)) / (f(b) - f(a));
        if (f(c) == 0.0) break;
        else if (f(c) * f(a) < 0) b = c;
        else a = c;
    }
    return c;
}

int main()
{
    double a = -3, b = -2, tol = 0.001;
    cout << "Root: " << falsePosition(a, b, tol) << endl;
    return 0;
}
```

GitHub Link: Click here to view on GitHub

8. Root Using Secant Method

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

double f(double x)
{
    return x * x * x - 5 * x * x - 29;
}

double secant(double x0, double x1, double tol)
{
    double x2;
    while (true)
    {
        x2 = x1 - f(x1) * (x1 - x0) / (f(x1) - f(x0));
        if (fabs(x2 - x1) < tol) break;
        x0 = x1;
        x1 = x2;
    }
    return x2;
}

int main()
{
    double x0 = 3, x1 = 4, tol = 0.001;
    cout << "Root: " << secant(x0, x1, tol) << endl;
    return 0;
}
```

GitHub Link: Click here to view on GitHub

9. Quotient Polynomial

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

vector<double> syntheticDivision(vector<double>&
    coefficients, double root)
{
    vector<double> quotient(coefficients.size() - 1);
    quotient[0] = coefficients[0];
    for (auto i = 1; i < coefficients.size() - 1; i++)
    {
        quotient[i] = coefficients[i] + quotient[i - 1] *
            root;
    }
    return quotient;
```

```
}

int main()
{
    vector<double> coefficients = {1, -5, 10, -8}; // x^3 -
        5x^2 + 10x - 8
    double root = 2;
    vector<double> quotient = syntheticDivision(coefficients
        , root);

    cout << "Quotient polynomial: ";
    for (double coeff : quotient)
    {
        cout << coeff << " ";
    }
    cout << endl;
    return 0;
}
```

GitHub Link: Click here to view on GitHub

10. All Roots Using Newton-Raphson Method with Deflation

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

double f(double x)
{
    return x * x * x - 6 * x + 4;
}

double f_prime(double x)
{
    return 3 * x * x - 6;
}

double newtonRaphson(double x0, double tol)
{
    double x1;
    while (true)
    {
        x1 = x0 - f(x0) / f_prime(x0);
        if (fabs(x1 - x0) < tol) break;
        x0 = x1;
    }
    return x1;
}
```

```cpp

vector<double> deflatePolynomial(vector<double> coeffs,
    double root)
{
    vector<double> newCoeffs(coeffs.size() - 1);
    newCoeffs[0] = coeffs[0];
    for (auto i = 1; i < newCoeffs.size(); i++)
    {
        newCoeffs[i] = coeffs[i] + newCoeffs[i - 1] * root;
    }
    return newCoeffs;
}

int main()
{
    double tol = 0.001;
    vector<double> coeffs = {1, 0, -6, 4}; // x^3 - 6x + 4

    double root1 = newtonRaphson(1.5, tol);
    cout << "Root 1: " << root1 << endl;

    coeffs = deflatePolynomial(coeffs, root1);
    double root2 = newtonRaphson(1.5, tol);
    cout << "Root 2: " << root2 << endl;

    coeffs = deflatePolynomial(coeffs, root2);
    double root3 = newtonRaphson(1.5, tol);
    cout << "Root 3: " << root3 << endl;

    return 0;
}
```

GitHub Link: Click here to view on GitHub