Numerical Methods Practicals

Submitted by → Rahul Agarwal, 2018UCO1665

NM Batch 1

Contents

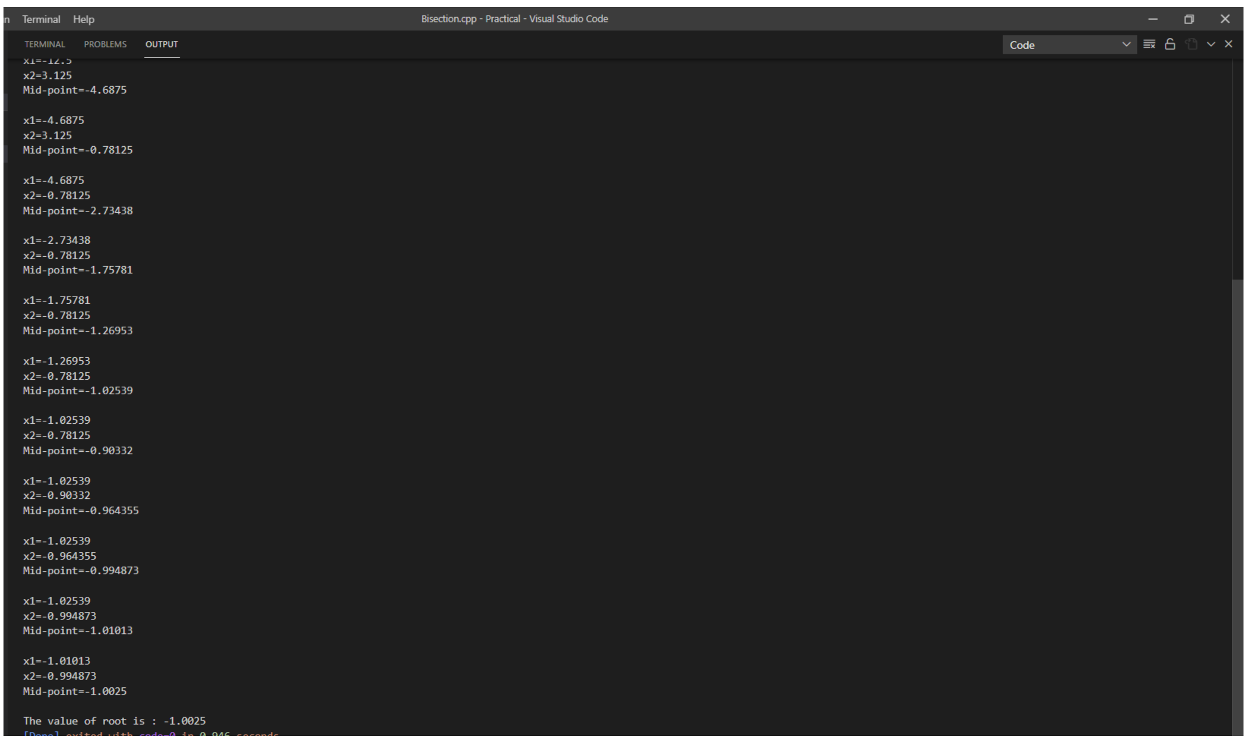
Bisection Method

Code

```
#include <bits/stdc++.h>
using namespace std;
#define EPSILON 0.01

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

void bisection(double a, double b)
{
    if (func(a) * func(b) >= 0)
    {
        cout << "You have not assumed right a and b\n";
        return;
    }

    double c = a;
    while ((b - a) >= EPSILON)
    {
        cout << "x1=" << a << endl;
        cout << "x2=" << b << endl;
        c = (a + b) / 2;
        cout << "Mid-point=" << c << endl;

        if (func(c) == 0.0)
            break;

        else if (func(c) * func(a) < 0)
            b = c;
        else
            a = c;
        cout << endl;
    }
    cout << "The value of root is : " << c;
```

```cpp
}

int main()
{
    double a = -200, b = 300;
    bisection(a, b);
    return 0;
}
```

Output



Newton Raphson Method

Code

```
#include <bits/stdc++.h>
#define EPSILON 0.001
using namespace std;

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

double derivFunc(double x)
{
    return (3 * x * x) - (2 * x);
}

void newtonRaphson(double x)
{
    double h = func(x) / derivFunc(x);
    while (abs(h) >= EPSILON)
    {
        cout << "y(x)=" << func(x) << endl;
        cout << "y\'(x)" << derivFunc(x) << endl;
        h = func(x) / derivFunc(x);

        x = x - h;
        cout << "New x=" << x << endl;
        cout << endl;
    }

    cout << "The value of the root is : " << x;
}

int main()
{
    double x0 = -20;
    newtonRaphson(x0);
```

```
    return 0;
}
```

Output

```
Terminal  Help                Code.cpp - Practical - Visual Studio Code
TERMINAL  PROBLEMS  OUTPUT                                          Code
[Running] cd "c:\College NSIT\Sem 7\Numerical Methods\Practical\Newton Raphson\" && g++ Code.cpp -o Code && "c:\College NSIT\Sem 7\Numerical Methods\Practical\Newton Raphson\"Code
y(x)=-8398
y'(x)1240
New x=-13.2274

y(x)=-2487.3
y'(x)551.349
New x=-8.71613

y(x)=-736.143
y'(x)245.345
New x=-5.71569

y(x)=-217.395
y'(x)109.439
New x=-3.72923

y(x)=-63.77
y'(x)49.1799
New x=-2.43256

y(x)=-18.3116
y'(x)22.6171
New x=-1.62292

y(x)=-4.90847
y'(x)11.1475
New x=-1.1826

y(x)=-1.05248
y'(x)6.56086
New x=-1.02219

y(x)=-0.112905
y'(x)5.17896
New x=-1.00038

y(x)=-0.00192236
y'(x)5.00308
New x=-1

The value of the root is : -1
```

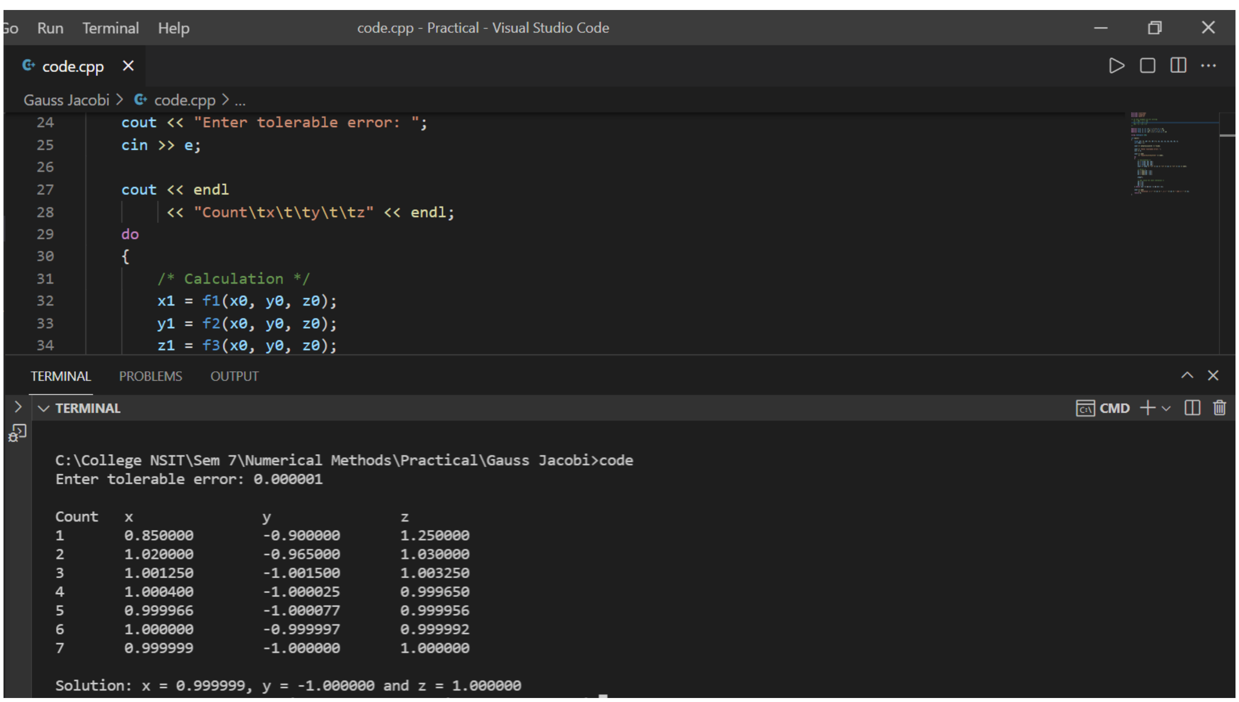
Gauss-Jacobi Method

Code

```
#include <iostream>
#include <iomanip>
#include <math.h>

/* In this example we are solving
   3x + 20y - z = -18
   2x - 3y + 20z = 25
   20x + y - 2z = 17
*/

#define f1(x, y, z) (17 - y + 2 * z) / 20
#define f2(x, y, z) (-18 - 3 * x + z) / 20
#define f3(x, y, z) (25 - 2 * x + 3 * y) / 20

using namespace std;

int main()
{
    float x0 = 0, y0 = 0, z0 = 0, x1, y1, z1, e1, e2, e3, e;
    int step = 1;

    cout << setprecision(6) << fixed;

    cout << "Enter tolerable error: ";
    cin >> e;

    cout << endl
         << "Count\tx\t\ty\t\tz" << endl;
    do
    {
        /* Calculation */
        x1 = f1(x0, y0, z0);
        y1 = f2(x0, y0, z0);
        z1 = f3(x0, y0, z0);
        cout << step << "\t" << x1 << "\t" << y1 << "\t" << z1 << endl;
```

```cpp
        /* Error */
        e1 = fabs(x0 - x1);
        e2 = fabs(y0 - y1);
        e3 = fabs(z0 - z1);

        step++;

        /* Set value for next iteration */
        x0 = x1;
        y0 = y1;
        z0 = z1;
    } while (e1 > e && e2 > e && e3 > e);

    cout << endl
         << "Solution: x = " << x1 << ", y = " << y1 << " and z = " << z1;
    return 0;
}
```

Output



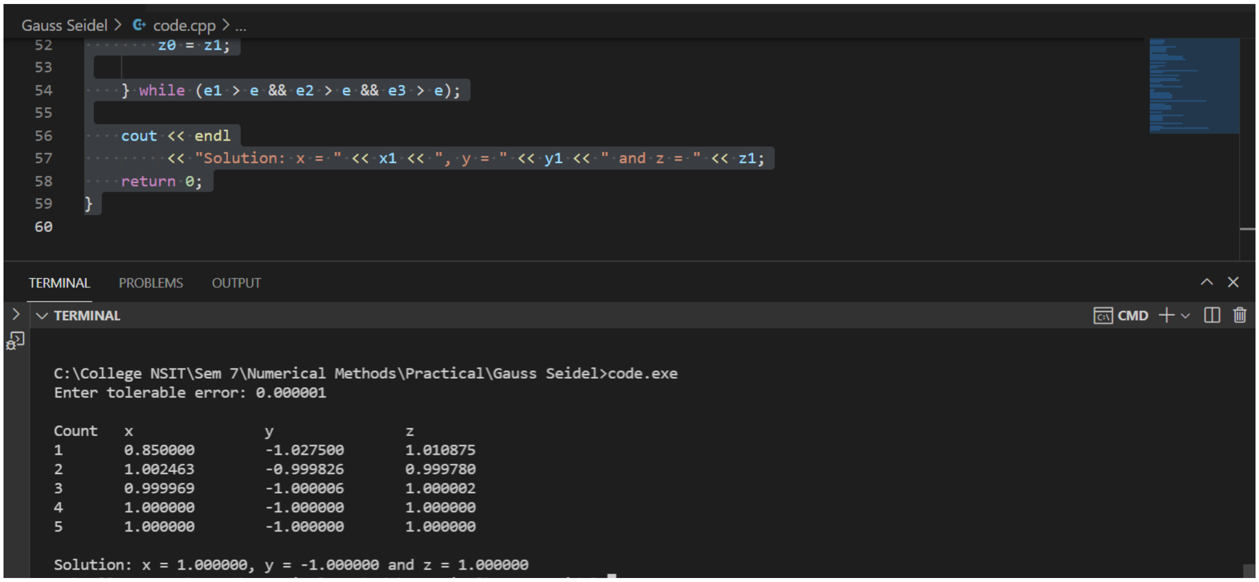
Gauss Seidel Method

Code

```
#include <iostream>
#include <iomanip>
#include <math.h>

/* In this example we are solving
   3x + 20y - z = -18
   2x - 3y + 20z = 25
   20x + y - 2z = 17
*/
/* Defining function */
#define f1(x, y, z) (17 - y + 2 * z) / 20
#define f2(x, y, z) (-18 - 3 * x + z) / 20
#define f3(x, y, z) (25 - 2 * x + 3 * y) / 20

using namespace std;

/* Main function */
int main()
{
    float x0 = 0, y0 = 0, z0 = 0, x1, y1, z1, e1, e2, e3, e;
    int step = 1;

    cout << setprecision(6) << fixed;

    /* Reading tolerable error */
    cout << "Enter tolerable error: ";
    cin >> e;

    cout << endl
         << "Count\tx\t\ty\t\tz" << endl;

    do
    {
        /* Calculation */
```

```cpp
        x1 = f1(x0, y0, z0);
        y1 = f2(x1, y0, z0);
        z1 = f3(x1, y1, z0);

        cout << step << "\t" << x1 << "\t" << y1 << "\t" << z1 << endl;

        /* Error */
        e1 = fabs(x0 - x1);
        e2 = fabs(y0 - y1);
        e3 = fabs(z0 - z1);

        step++;

        /* Set value for next iteration */
        x0 = x1;
        y0 = y1;
        z0 = z1;

    } while (e1 > e && e2 > e && e3 > e);

    cout << endl
         << "Solution: x = " << x1 << ", y = " << y1 << " and z = " << z1;
    return 0;
}
```

Output


Gauss Seidel > code.cpp > ...
z0 = z1;
} while (e1 > e && e2 > e && e3 > e);
cout << endl
<< "Solution: x = " << x1 << ", y = " << y1 << " and z = " << z1;
return 0;
}
TERMINAL
PROBLEMS
OUTPUT
TERMINAL
CMD
C:\College NSIT\Sem 7\Numerical Methods\Practical\Gauss Seidel>code.exe
Enter tolerable error: 0.000001
Count x y z
1 0.850000 -1.027500 1.010875
2 1.002463 -0.999826 0.999780
3 0.999969 -1.000006 1.000002
4 1.000000 -1.000000 1.000000
5 1.000000 -1.000000 1.000000
Solution: x = 1.000000, y = -1.000000 and z = 1.000000

Newton Forward Interpolation Method

Code

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

float u_cal(float u, int n)
{
    float temp = u;
    for (int i = 1; i < n; i++)
        temp = temp * (u - i);
    return temp;
}

int fact(int n)
{
    int f = 1;
    for (int i = 2; i <= n; i++)
        f *= i;
    return f;
}

int main()
{
    int n = 4;
    float x[] = {45, 50, 55, 60};

    float y[n][n];
    y[0][0] = 0.7071;
    y[1][0] = 0.7660;
    y[2][0] = 0.8192;
    y[3][0] = 0.8660;

    for (int i = 1; i < n; i++)
    {
        for (int j = 0; j < n - i; j++)
            y[j][i] = y[j + 1][i - 1] - y[j][i - 1];
    }
```

```
    for (int i = 0; i < n; i++)
    {
        cout << setw(4) << x[i]
             << "\t";
        for (int j = 0; j < n - i; j++)
            cout << setw(4) << y[i][j]
                 << "\t";
        cout << endl;
    }

    float value = 52;

    float sum = y[0][0];
    float u = (value - x[0]) / (x[1] - x[0]);
    for (int i = 1; i < n; i++)
    {
        sum = sum + (u_cal(u, i) * y[0][i]) /
                                fact(i);
    }

    cout << "\n Value at " << value << " is "
         << sum << endl;
    return 0;
}
```

Output

Newton fwd interpolation > code.cpp > main()

```cpp

    for (int i = 0; i < n; i++)
    {
        cout << setw(4) << x[i]
             << "\t";
        for (int j = 0; j < n - i; j++)
            cout << setw(4) << y[i][j]
                 << "\t";
        cout << endl;
    }

    float value = 52;
```

TERMINAL PROBLEMS OUTPUT Code

```
[Running] cd "c:\College NSIT\Sem 7\Numerical Methods\Practical\Newton fwd interpolation\" && g++ tempCodeRunnerFile.cpp -o tempCodeRu
  45    0.7071  0.0589  -0.00569999 -0.000699997
  50    0.766   0.0532  -0.00639999
  55    0.8192  0.0468
  60    0.866

 Value at 52 is 0.788003

[Done] exited with code=0 in 1.026 seconds
```

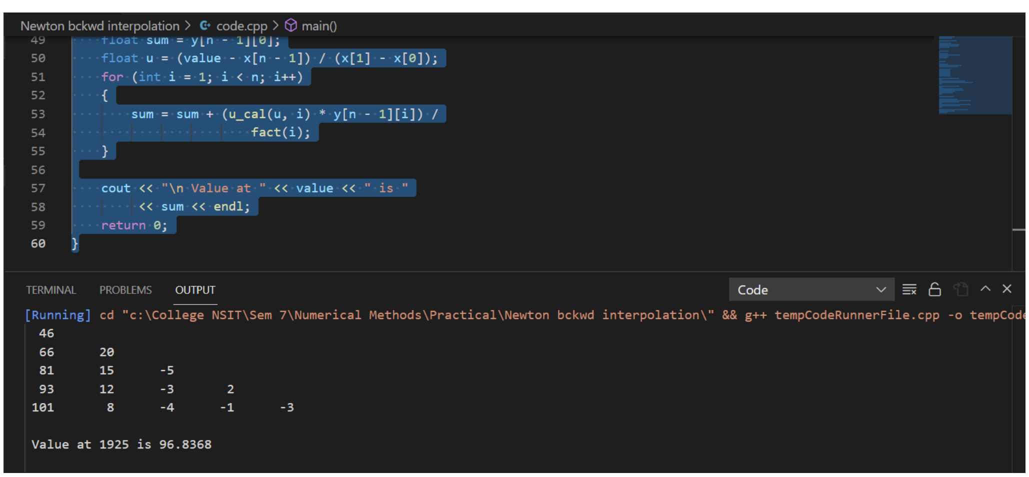
Newton Backward Interpolation Method

Code

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

float u_cal(float u, int n)
{
    float temp = u;
    for (int i = 1; i < n; i++)
        temp = temp * (u + i);
    return temp;
}

int fact(int n)
{
    int f = 1;
    for (int i = 2; i <= n; i++)
        f *= i;
    return f;
}

int main()
{
    int n = 5;
    float x[] = {1891, 1901, 1911,
                 1921, 1931};

    float y[n][n];
    y[0][0] = 46;
    y[1][0] = 66;
    y[2][0] = 81;
    y[3][0] = 93;
    y[4][0] = 101;

    for (int i = 1; i < n; i++)
    {
        for (int j = n - 1; j >= i; j--)
```

```
            y[j][i] = y[j][i - 1] - y[j - 1][i - 1];
    }

    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j <= i; j++)
            cout << setw(4) << y[i][j]
                 << "\t";
        cout << endl;
    }

    float value = 1925;

    float sum = y[n - 1][0];
    float u = (value - x[n - 1]) / (x[1] - x[0]);
    for (int i = 1; i < n; i++)
    {
        sum = sum + (u_cal(u, i) * y[n - 1][i]) /
                        fact(i);
    }

    cout << "\n Value at " << value << " is "
         << sum << endl;
    return 0;
}
```

Output



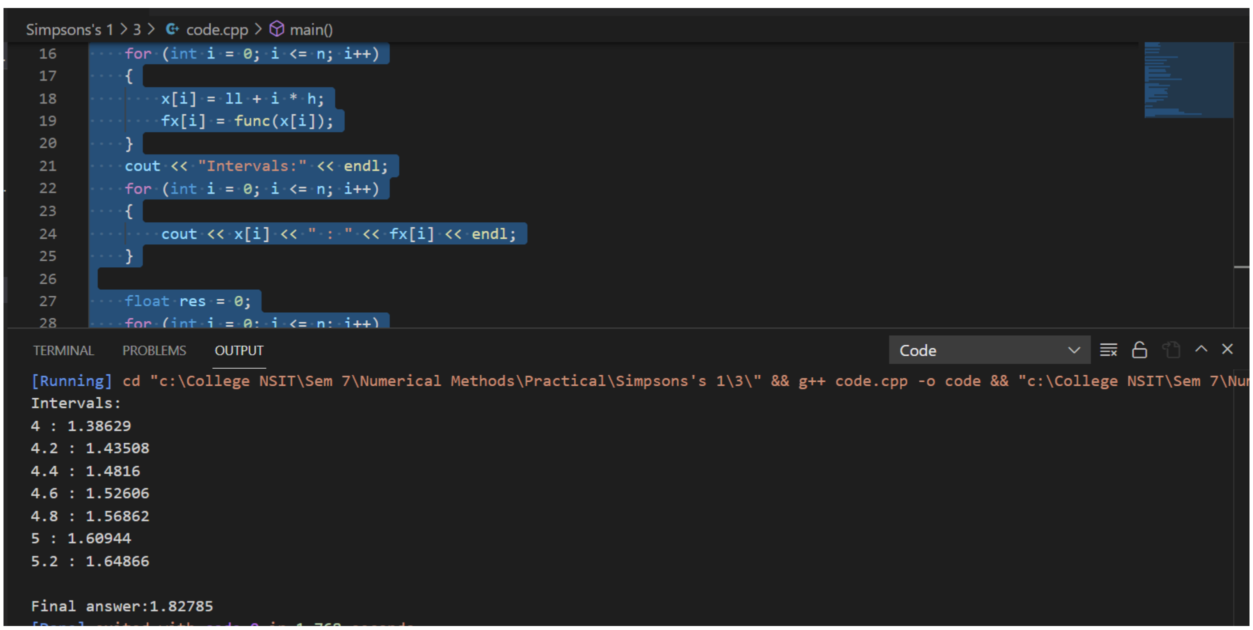
Simpson’s ⅓ Integration Formula

Code

```
#include <iostream>
#include <math.h>
using namespace std;

float func(float x)
{
    return log(x);
}

float simpsons_(float ll, float ul, int n)
{
    float h = (ul - ll) / n;

    float x[10], fx[10];

    for (int i = 0; i <= n; i++)
    {
        x[i] = ll + i * h;
        fx[i] = func(x[i]);
    }
    cout << "Intervals:" << endl;
    for (int i = 0; i <= n; i++)
    {
        cout << x[i] << " : " << fx[i] << endl;
    }

    float res = 0;
    for (int i = 0; i <= n; i++)
    {
        if (i == 0 || i == n)
            res += fx[i];
        else if (i % 2 != 0)
            res += 4 * fx[i];
        else
            res += 2 * fx[i];
```

```
    }
    res = res * (h / 3);
    return res;
}

int main()
{
    float lower_limit = 4;    // Lower limit
    float upper_limit = 5.2; // Upper limit
    int n = 6;                // Number of interval
    cout << "\nFinal answer:" << simpsons_(lower_limit, upper_limit, n);
    return 0;
}
```

Output



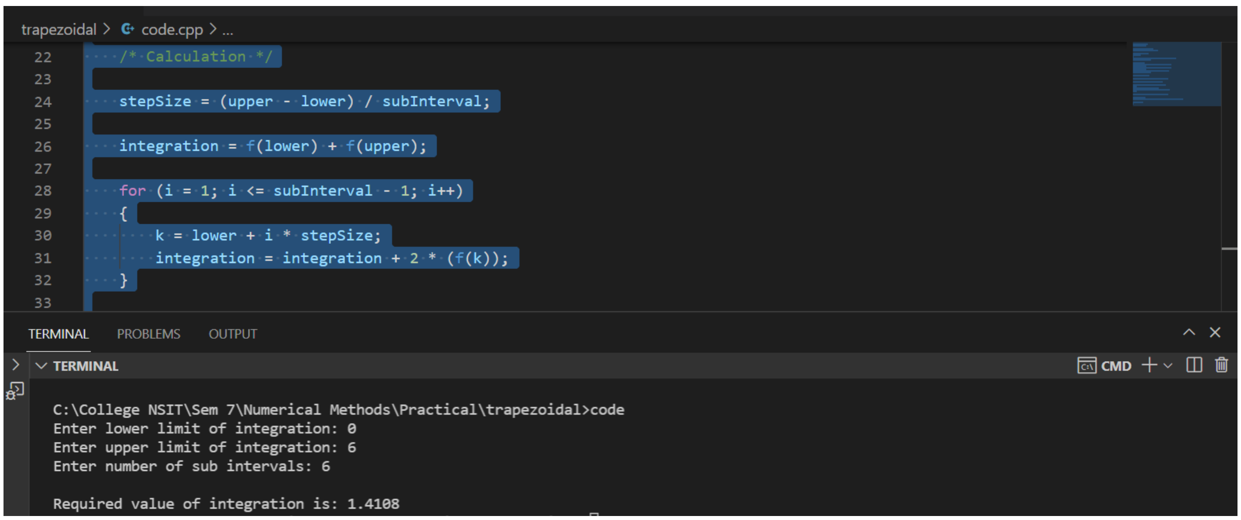
Trapezoidal Integration Formula

Code

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

/* Define function here */
#define f(x) 1 / (1 + pow(x, 2))

using namespace std;
int main()
{
    float lower, upper, integration = 0.0, stepSize, k;
    int i, subInterval;

    /* Input */
    cout << "Enter lower limit of integration: ";
    cin >> lower;
    cout << "Enter upper limit of integration: ";
    cin >> upper;
    cout << "Enter number of sub intervals: ";
    cin >> subInterval;

    /* Calculation */

    stepSize = (upper - lower) / subInterval;

    integration = f(lower) + f(upper);

    for (i = 1; i <= subInterval - 1; i++)
    {
        k = lower + i * stepSize;
        integration = integration + 2 * (f(k));
    }

    integration = integration * stepSize / 2;
```

```
    cout << endl
         << "Required value of integration is: " << integration;

    return 0;
}
```

Output



Euler’s Method

Code

```
#include <iostream>

/* In this example we are solving dy/dx = x + y */
#define f(x, y) x + y

using namespace std;

int main()
{
    float x0, y0, xn, h, yn, slope;
    int i, n;

    cout << "Enter Initial Condition" << endl;
    cout << "x0 = ";
    cin >> x0;
    cout << "y0 = ";
    cin >> y0;
    cout << "Enter calculation point xn = ";
    cin >> xn;
    cout << "Enter number of steps: ";
    cin >> n;

    /* Calculating step size (h) */
    h = (xn - x0) / n;

    /* Euler's Method */
    cout << "\nx0\ty0\tslope\tyn\n";
    cout << "------------------------------\n";

    for (i = 0; i < n; i++)
    {
        slope = f(x0, y0);
        yn = y0 + h * slope;
        cout << x0 << "\t" << y0 << "\t" << slope << "\t" << yn << endl;
```

```cpp
        y0 = yn;
        x0 = x0 + h;
    }

    /* Displaying result */
    cout << "\nValue of y at x = " << xn << " is " << yn;

    return 0;
}
```

Output

```
C:\College NSIT\Sem 7\Numerical Methods\Practical\Euler>code
Enter Initial Condition
x0 = 0
y0 = 1
Enter calculation point xn = 0.5
Enter number of steps: 10

x0      y0      slope   yn
------------------------------
0       1       1       1.05
0.05    1.05    1.1     1.105
0.1     1.105   1.205   1.16525
0.15    1.16525 1.31525 1.23101
0.2     1.23101 1.43101 1.30256
0.25    1.30256 1.55256 1.38019
0.3     1.38019 1.68019 1.4642
0.35    1.4642  1.8142  1.55491
0.4     1.55491 1.95491 1.65266
0.45    1.65266 2.10266 1.75779

Value of y at x = 0.5 is 1.75779
```

Runge-Kutta Method

Code

```
#include <iostream>

/* Defining ordinary differential equation to be solved */
#define f(x, y) (y * y - x * x) / (y * y + x * x)

using namespace std;

/* defining ordinary differential equation to be solved */
#define f(x, y) (y * y - x * x) / (y * y + x * x)

using namespace std;
int main()
{
    float x0, y0, xn, h, yn, k1, k2, k3, k4, k;
    int i, n;

    cout << "Enter Initial Condition" << endl;
    cout << "x0 = ";
    cin >> x0;
    cout << "y0 = ";
    cin >> y0;
    cout << "Enter calculation point xn = ";
    cin >> xn;
    cout << "Enter number of steps: ";
    cin >> n;

    /* Calculating step size (h) */
    h = (xn - x0) / n;

    /* Runge Kutta Method */
    cout << "\nx0\ty0\tyn\n";
    cout << "------------------\n";
    for (i = 0; i < n; i++)
    {
```

```cpp
        k1 = h * (f(x0, y0));
        k2 = h * (f((x0 + h / 2), (y0 + k1 / 2)));
        k3 = h * (f((x0 + h / 2), (y0 + k2 / 2)));
        k4 = h * (f((x0 + h), (y0 + k3)));
        k = (k1 + 2 * k2 + 2 * k3 + k4) / 6;
        yn = y0 + k;
        cout << x0 << "\t" << y0 << "\t" << yn << endl;
        x0 = x0 + h;
        y0 = yn;
    }

    /* Displaying result */
    cout << "\nValue of y at x = " << xn << " is " << yn;

    return 0;
}
```

Output

Runge Kutta > code.cpp > ...

```cpp
    cout << "\nx0\ty0\tyn\n";
    cout << "------------------\n";
    for (i = 0; i < n; i++)
    {
        k1 = h * (f(x0, y0));
        k2 = h * (f((x0 + h / 2), (y0 + k1 / 2)));
        k3 = h * (f((x0 + h / 2), (y0 + k2 / 2)));
        k4 = h * (f((x0 + h), (y0 + k3)));
        k = (k1 + 2 * k2 + 2 * k3 + k4) / 6;
        yn = y0 + k;
```

TERMINAL PROBLEMS OUTPUT

TERMINAL

```
C:\College NSIT\Sem 7\Numerical Methods\Practical\Runge Kutta>code
Enter Initial Condition
x0 = 0
y0 = 1
Enter calculation point xn = 0.6
Enter number of steps: 5

x0      y0      yn
------------------
0       1       1.11903
0.12    1.11903 1.23346
0.24    1.23346 1.34109
0.36    1.34109 1.44105
0.48    1.44105 1.53313

Value of y at x = 0.6 is 1.53313
```