Numerical Methods Practicals

Submitted by → Rahul Agarwal, 2018UCO1665

NM Batch 1

# Contents

1. [**Bisection method**](#_t5edttgu1y7n)
2. [**Newton-Raphson method**](#_s10zfjg8twzy)
3. [**Gauss-Jacobi method**](#_3bunt4y9dr23)
4. [**Gauss-Seidel method**](#_au49zwuyf4tw)
5. [**Newton-forward interpolation method**](#_jybp8m25ac4t)
6. [**Newton-backward interpolation method**](#_dc8hzcvfqn4n)
7. [**Simpson’s ⅓ integration formula**](#_ynmu15y98lgb)
8. [**Trapezoidal Integration formula**](#_dar5yo8we48z)
9. [**Euler’s method**](#_hh99gcqqnxby)
10. [**Runge-Kutta method**](#_p6p0b2ql0cnm)

## 

# 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;*

*}*

*int main()*

*{*

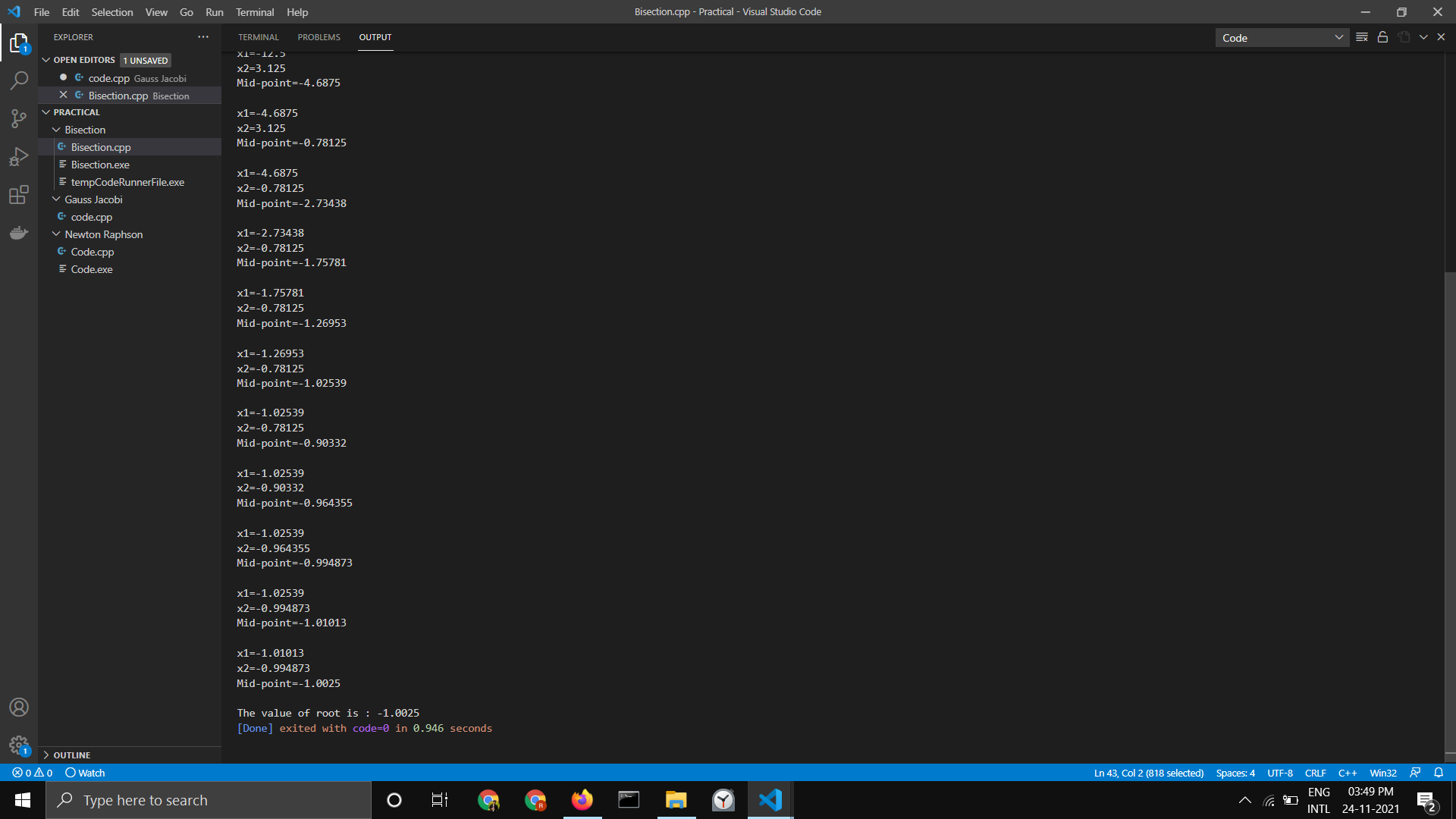
*double a = -200, b = 300;*

*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()*

*{*

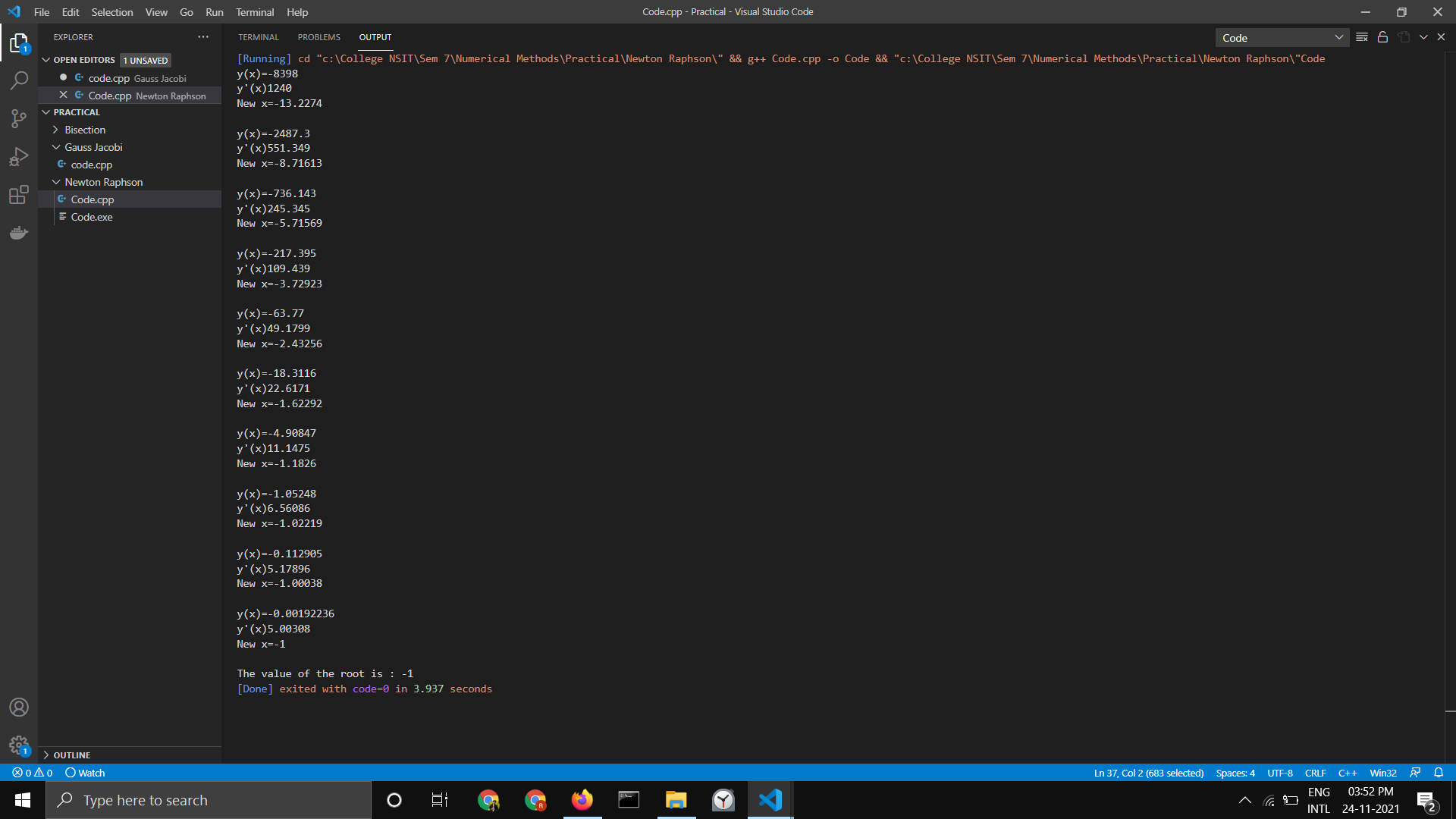
*double x0 = -20;*

*newtonRaphson(x0);*

*return 0;*

*}*

## Output



# 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;*

*/\* 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);*

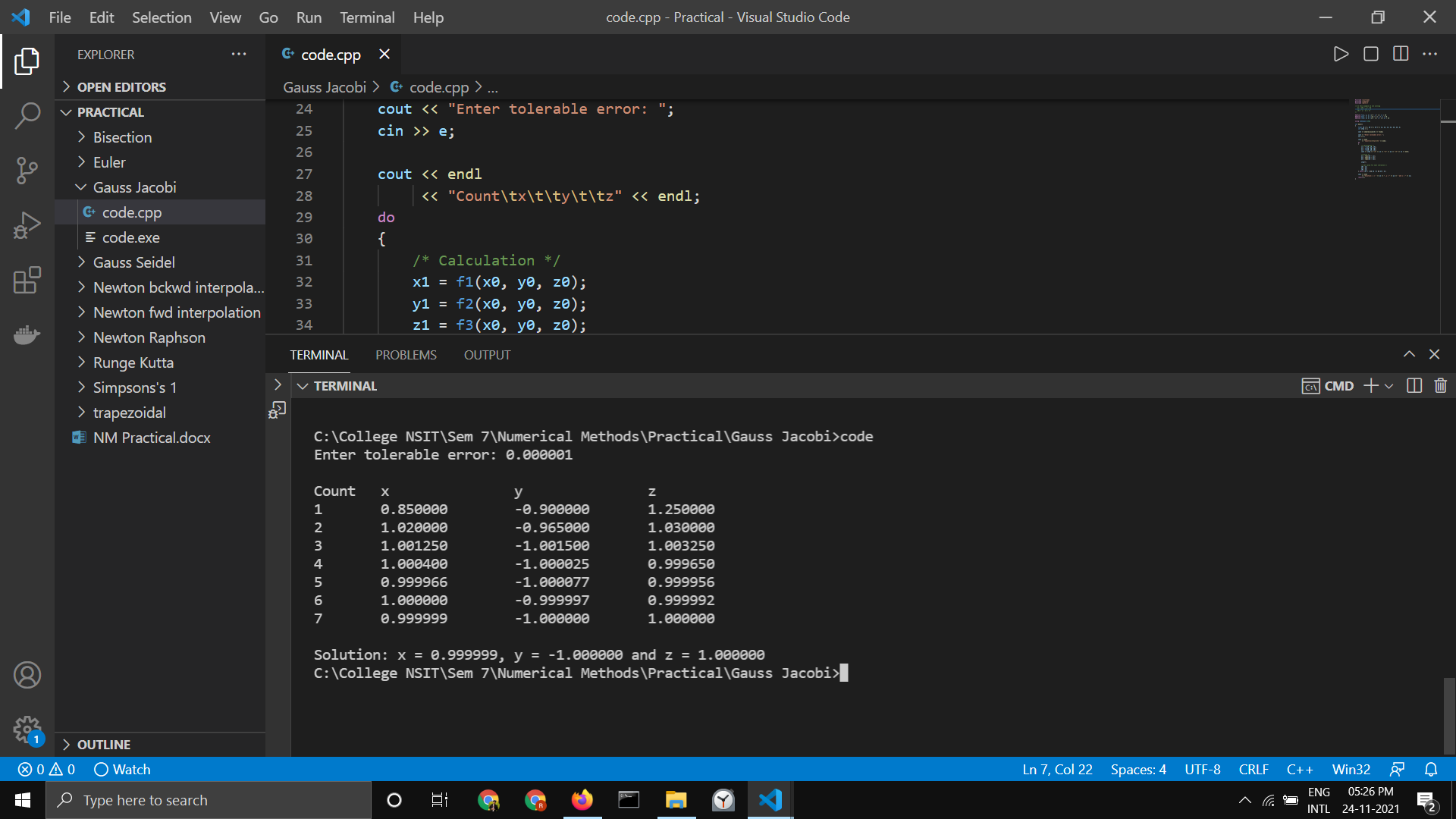
*cout << endl*

*<< "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 \*/*

*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);*

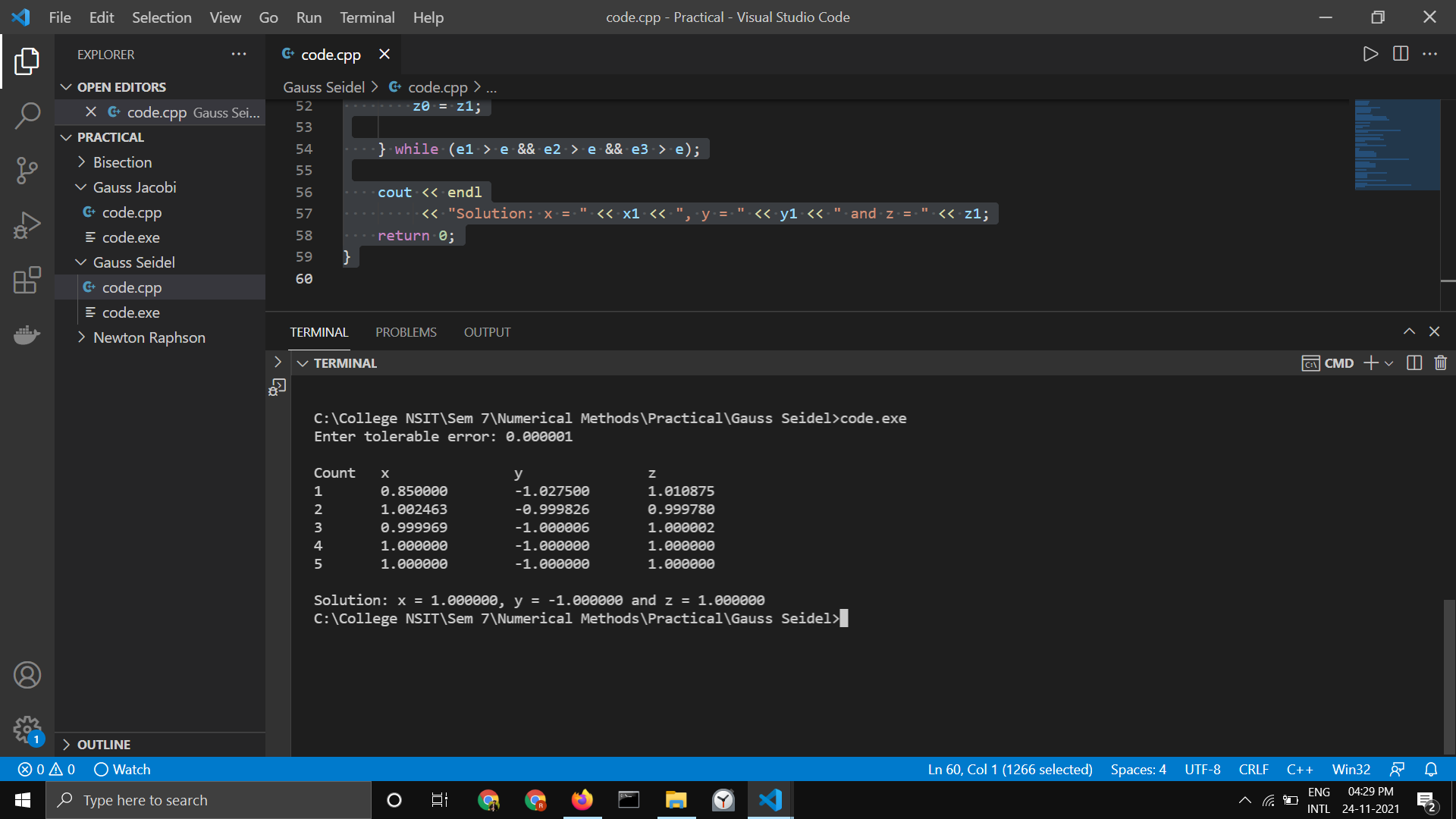
*cout << endl*

*<< "Solution: x = " << x1 << ", y = " << y1 << " and z = " << z1;*

*return 0;*

*}*

## Output



# 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);*

*}*

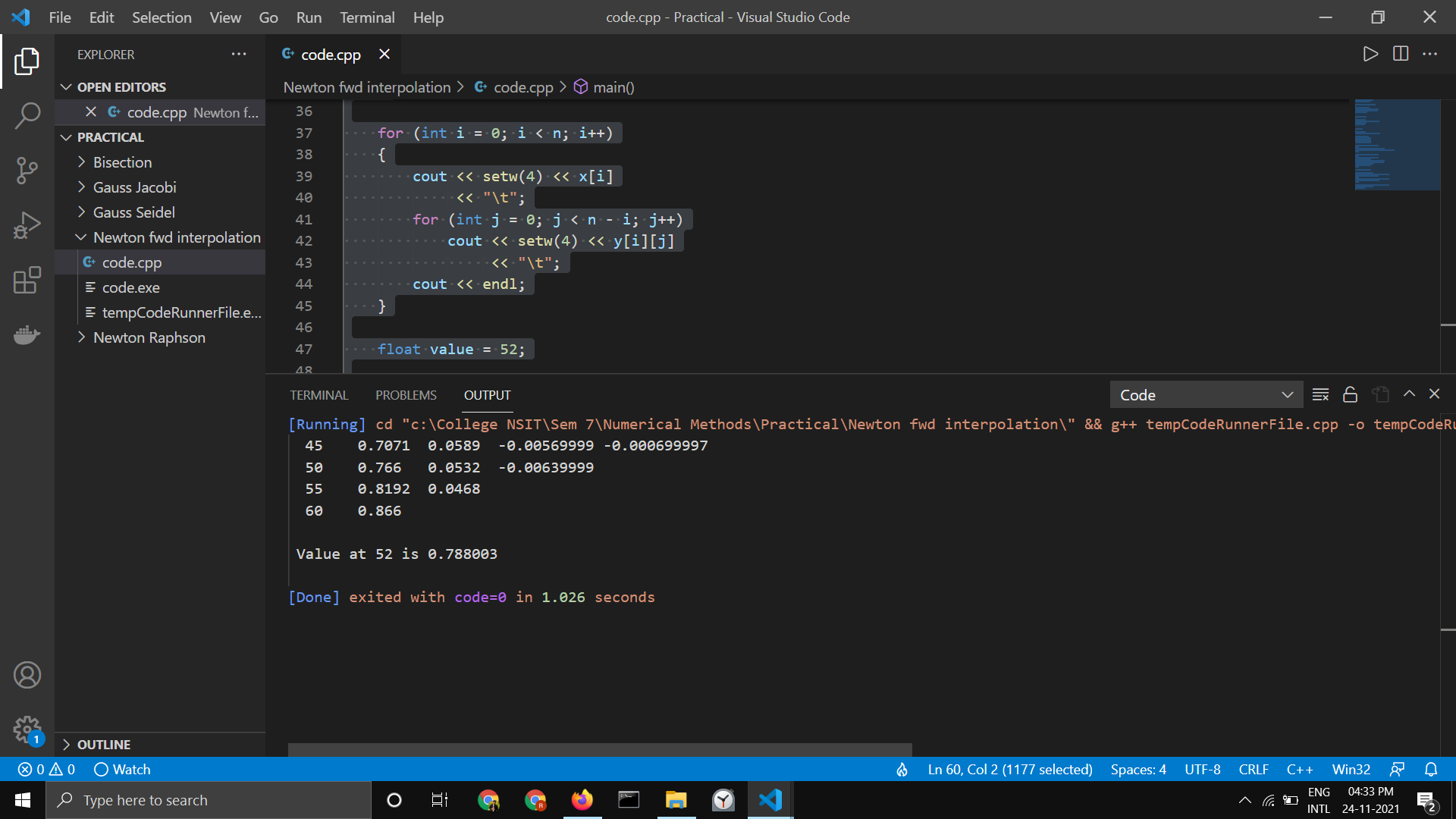
*cout << "\n Value at " << value << " is "*

*<< sum << endl;*

*return 0;*

*}*

## Output



# Newton 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);*

*}*

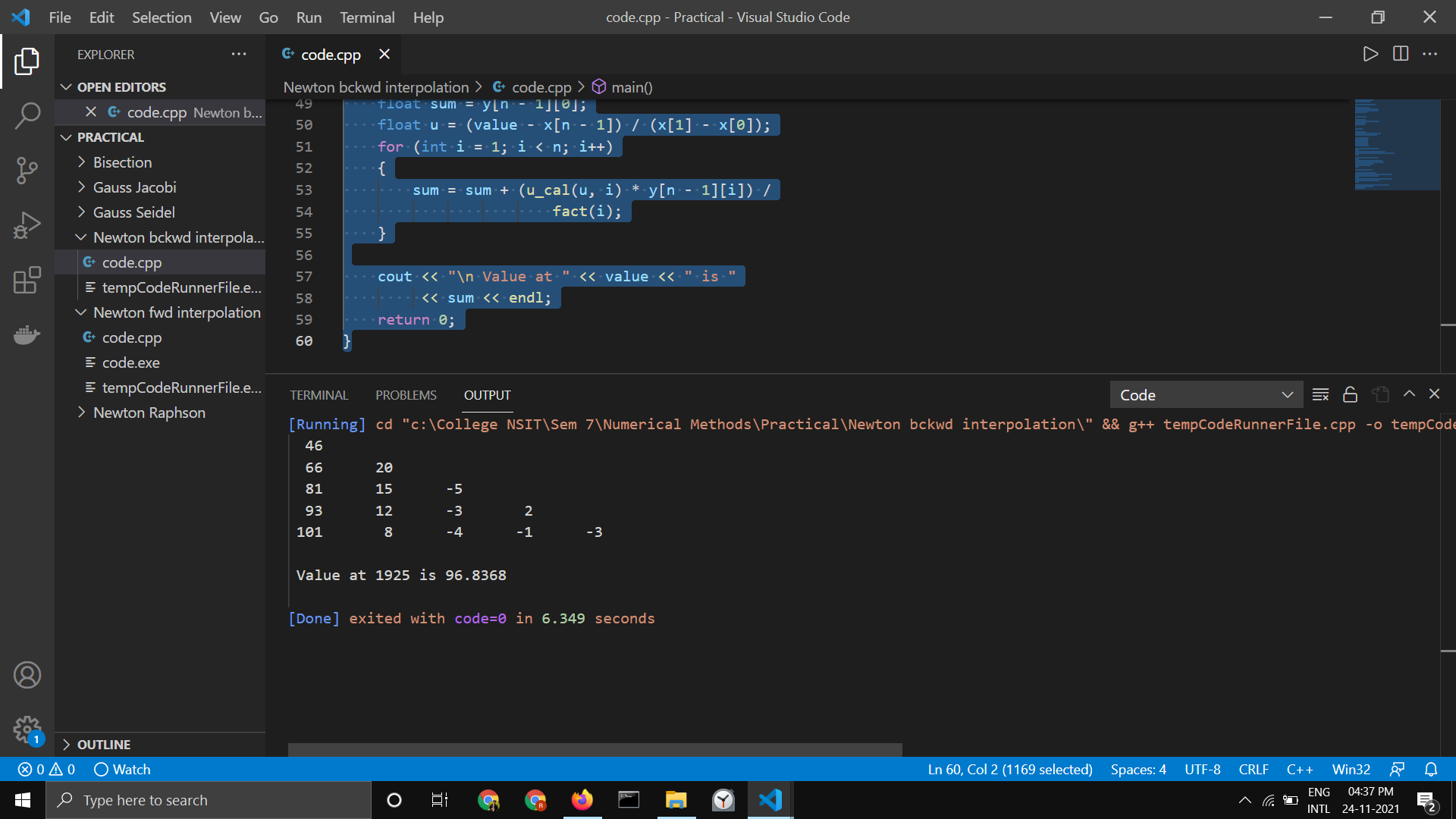
*cout << "\n Value at " << value << " is "*

*<< 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*

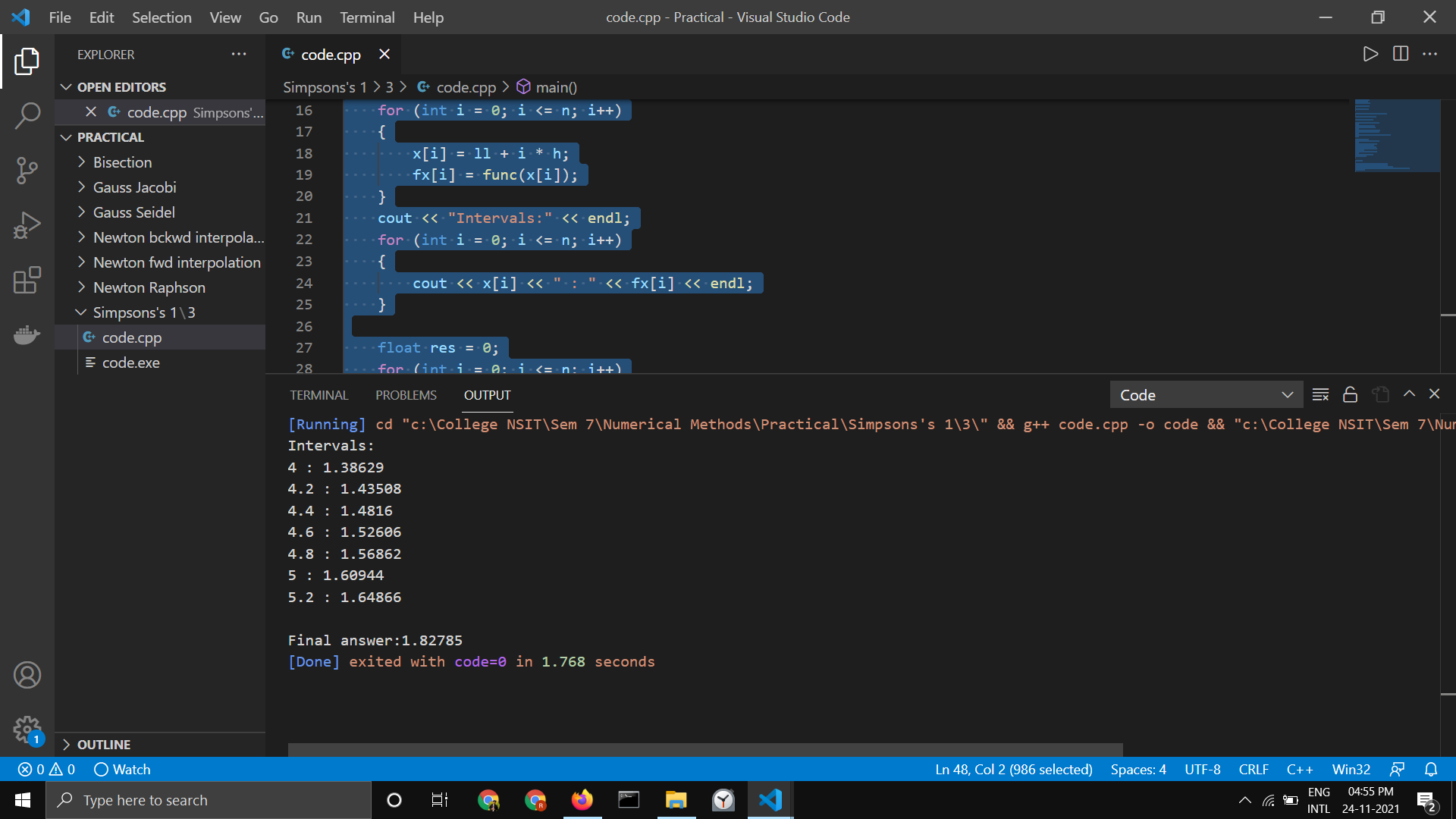
*int n = 6; // Number of interval*

*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;*

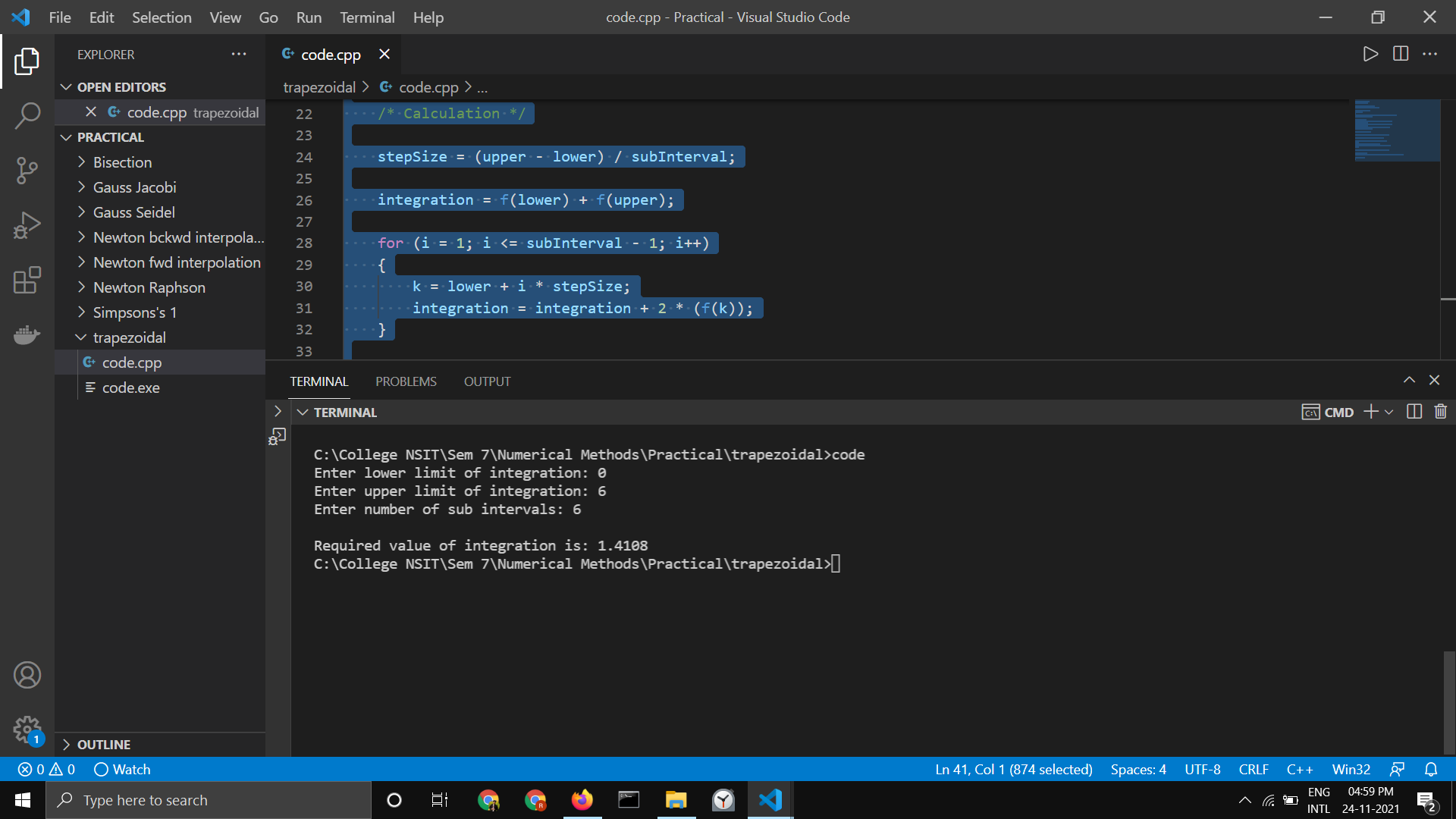
*cout << endl*

*<< "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;*

*y0 = yn;*

*x0 = x0 + h;*

*}*

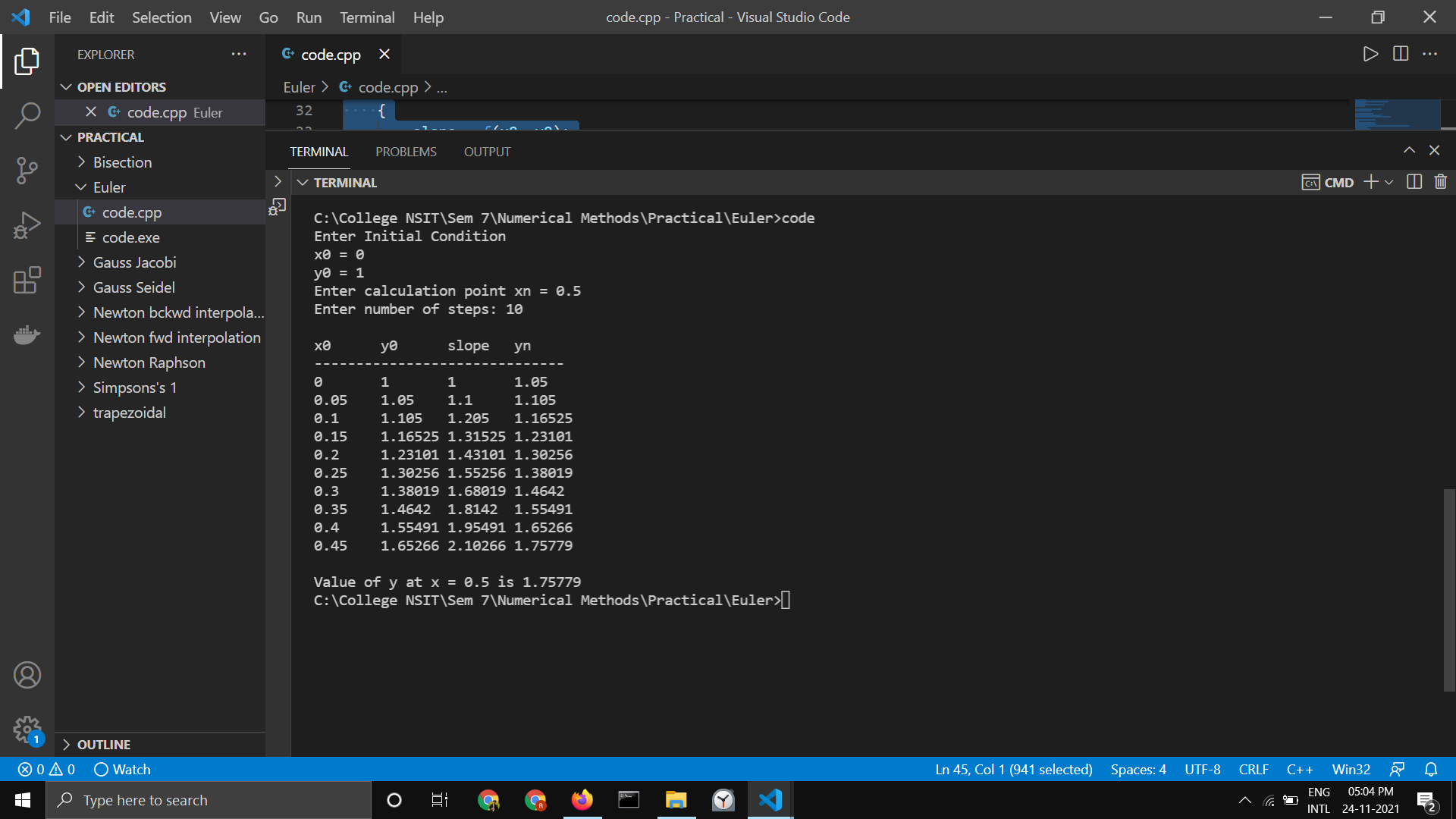
*/\* Displaying result \*/*

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

*return 0;*

*}*

## Output



# 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++)*

*{*

*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;*

*}*

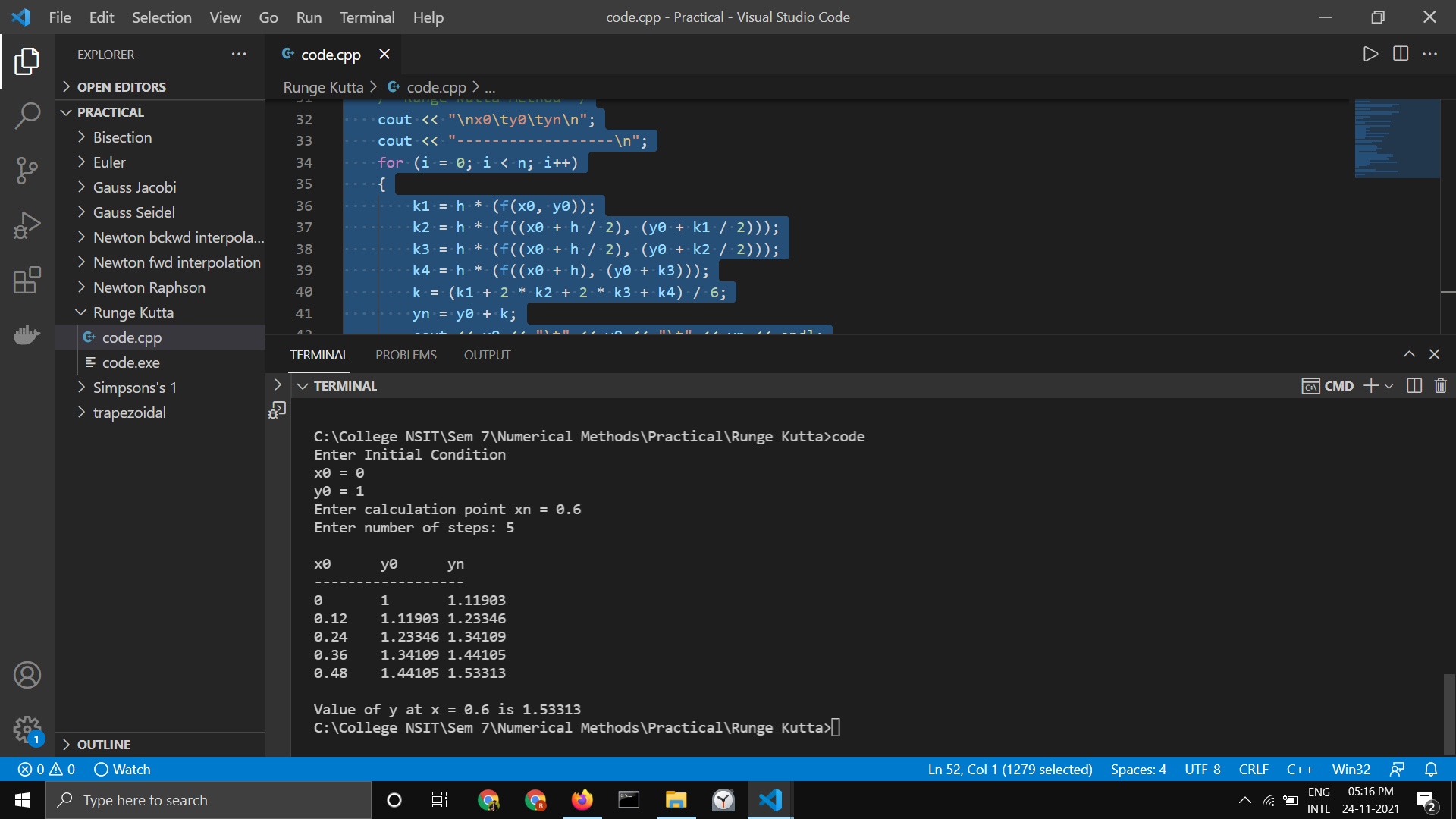
*/\* Displaying result \*/*

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

*return 0;*

*}*

## Output



## 