BINARY SOLUTIONS

CEE 394
Numerical Integration Using Simpson's Rules in C++

contact@thebinarysolutionsllc.com

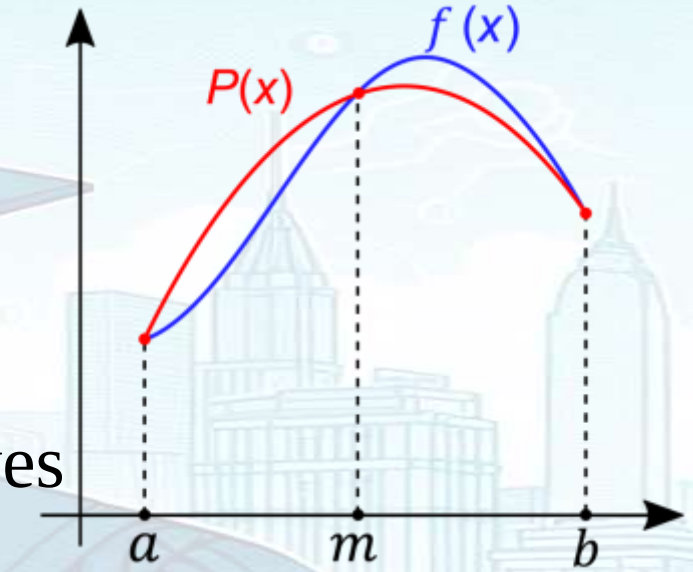
Introduction

This program numerically integrates a given function within a specified interval using Simpson's rules. It demonstrates the application of numerical methods for approximating definite integrals and assessing the accuracy of the result through error analysis.

Problem Statement

Given a function $f(x)$ and the interval $[a, b]$, the task is to approximate the integral of $f(x)$ over the interval $[a, b]$ using Simpson's rules and calculate the percentage error compared to the exact integral value.

Simpson’s Rule Theory



- Simpson’s Rule : Smarter area approximations for curves
- Steps to Simpson’s 1/3 Approximations:
  1. Divide the curve into even sections.
  2. In each section, draw a parabola using three points.
- Formula for Simpsons 1/3 rule

$$Integral = \frac{h}{3}[f(x_0) + 4f(x_1) + 2f(x_2) \ldots + 2f(x_{n-2}) + 4f(x_{n-1}) + f(x_n)$$

contact@thebinarysolutionsllc.com

Simpson's Rule Theory

- Simpson's 3/8 works with 4 points instead of 3 as in Simspon's 1/3
- Steps to Simpson's 3/8 Approximations:
  1. Divide the curve into multiple sections (multiples of 3).
  2. In each section, draw a parabola using four points.
- Formula for Simpsons 3/8 rule

$$Integral = \frac{3h}{8}[f(x_0) + 3f(x_1) + 3f(x_2) + 2f(x_3) \ldots + 2f(x_{n-2}) + 3f(x_{n-1}) + f(x_n)$$

Solution Steps

- Define the function $f(x)$ to be integrated.
- Specify the interval $[a, b]$ for integration.
- Calculate the step size hh using the formula:

$$h = \frac{b-a}{10}$$

- Apply Simpson's 1/3 rule for the first 4 intervals and Simpson's 3/8 rule for the last 6 intervals to compute the approximated integral.

Solution Steps

- Define the function $f(x)$ to be integrated.
- Compute the percentage error using the formula:

$$\frac{|Extract\ Integral - Approximated\ Integral|}{Extract\ Integral} \times 100$$

contact@thebinarysolutionsllc.com

Pseudo Code

1. Define the function f(x) that needs to be integrated.
2. Begin main function.
    2.1 Define the lower bound 'a' and upper bound 'b' of the integration interval.
    2.2 Calculate the step size 'h' as (b - a) / 10.
    2.3 Apply Simpson's 1/3 rule for the first 4 intervals:
        2.3.1 Calculate Simpson's 1/3 rule for the first 4 intervals using the formula:
            simp13 = (h / 3) * (f(a) + 4 * f(a + h) + 2 * f(a + 2 * h) + 4 * f(a + 3 * h) + f(a + 4 * h)).
    2.4 Apply Simpson's 3/8 rule for the last 6 intervals:
        2.4.1 Calculate Simpson's 3/8 rule for the last 6 intervals using the formula:
            simp38 = (3 * h / 8) * (f(a + 4 * h) + 3 * (f(a + 5 * h) + f(a + 6 * h) + f(a + 7 * h) + f(a + 8 * h)) + f(b)).
    2.5 Calculate the total integral 'int_f' as the sum of simp13 and simp38.
    2.6 Calculate the percentage error 'Et' using the formula:
        Et = ((exact_integral - int_f) / exact_integral) * 100.
    2.7 Output the calculated integral 'int_f' and the percentage error 'Et'.
    2.8 End main function.

contact@thebinarysolutionsllc.com

C++ Code

```
#include <iostream>
#include <cmath>
// Define the function
double f(double x) {
    return 0.2 + 25 * x - 200 * pow(x, 2) + 675 * pow(x, 3) - 900 * pow(x, 4) + 400 * pow(x, 5);}
int main() {
// Define interval [a, b] = [0.2, 0.8]
    double a = 0.2;
    double b = 0.8;
    // Calculate step size
    double h = (b - a) / 10;
    // Apply Simpson's 1/3 rule for the first 4 intervals
    double simp13 = (h / 3) * (f(a) + 4 * f(a + h) + 2 * f(a + 2 * h) + 4 * f(a + 3 * h) + f(a + 4 * h));
```

contact@thebinarysolutionsllc.com

C++ Code

```
// Apply Simpson's 3/8 rule for the last 6 intervals
    double simp38 = (3 * h / 8) * (f(a + 4 * h) + 3 * (f(a + 5 * h) + f(a + 6 * h) + f(a + 7 * h)
+ f(a + 8 * h)) + f(b));

    // Calculate the integral
    double int_f = simp13 + simp38;

    // Calculate percentage error
    double Et = ((1.640533 - int_f) / 1.640533) * 100;

    // Output results
    std::cout << "Integral of the function: " << int_f << std::endl;
    std::cout << "Percentage error: " << Et << "%" << std::endl;

    return 0;
}
```

Code Explanation

- **#include <iostream>#include <cmath>**

These lines include the necessary header files: ‘<iostream>’ for input/output stream functionality and ‘<cmath>’ for mathematical functions.

- **double f(double x) { return 0.2 + 25 * x - 200 * pow(x, 2) + 675 * pow(x, 3) - 900 * pow(x, 4) + 400 * pow(x, 5); }**

This block defines a function ‘f(x)’ that calculates the value of a given function at a given point ‘x’.

- **int main() {**

This line marks the beginning of the ‘main’ function, which serves as the entry point of the program.

- **double a = 0.2; double b = 0.8;**

These lines define the lower and upper bounds of integration.

Code Explanation

- **double h = (b - a) / 10;**

This line calculates the step size ‘h’ for the integration interval.

- **double simp13 = (h / 3) * (f(a) + 4 * f(a + h) + 2 * f(a + 2 * h) + 4 * f(a + 3 * h) + f(a + 4 * h));**

This line applies Simpson's 1/3 rule to the first 4 intervals of the integration interval.

- **double simp38 = (3 * h / 8) * (f(a + 4 * h) + 3 * (f(a + 5 * h) + f(a + 6 * h) + f(a + 7 * h) + f(a + 8 * h)) + f(b));**

This line applies Simpson's 3/8 rule to the last 6 intervals of the integration interval.

- **double int_f = simp13 + simp38;**

This line calculates the total integral by summing the results obtained from Simpson's 1/3 and 3/8 rules.

contact@thebinarysolutionsllc.com

Code Explanation

- **double Et = ((1.640533 - int_f) / 1.640533) * 100;**

This line calculates the percentage error of the calculated integral compared to the exact value.

- **std::cout << "Integral of the function: " << int_f << std::endl; std::cout << "Percentage error: " << Et << "%" << std::endl;**

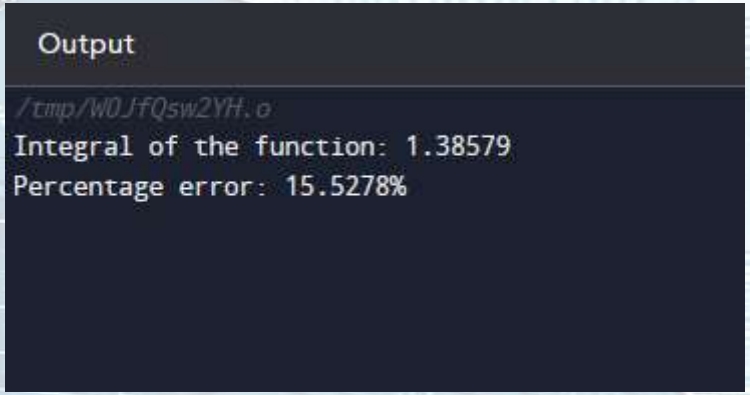
These lines output the calculated integral and the percentage error to the standard output (typically the console).

- **return 0;}**

This line indicates the end of the ‘main’ function and returns an integer value of ‘0’ to the operating system, typically indicating successful execution.

Final Answer

- The Resulting area under the curve is 1.38579 squared units.
- This is with an error of 15.5278%
- The accuracy of the resulting area under curve can be increase with an increase in number of intervals.



contact@thebinarysolutionsllc.com

Additional Comments/Tips

- Ensure the correctness of the function definition and interval specification for accurate integration.
- Experiment with different numbers of subintervals to observe how the accuracy of the approximation varies.

Conclusion

This program demonstrates the application of Simpson's rules for numerical integration in C++, providing an efficient and accurate method for approximating definite integrals.