ITCS 6145

Parallel Computing
Professor : Jinzhen Wang

Name : Aravinda Reddy Gangalakunta
Student Id : 801393674

Module 1: Homework 1 : Numerical Integration

Numerical Integration:

Numerical integration is a way to find the area under a curve. In complicated graphs or curves we can only approximate the area.
It can be done by

1) Create small rectangles or trapezoids along the curve
2) Calculate areas of those shapes
3) Add them together

Common Numerical Integration Methods:

1. Rectangular Rule: Approximates the function with rectangles.
2. Trapezoidal Rule: Approximates the function with trapezoids.
3. Midpoint Rule: Approximates the function at the midpoint of each subinterval (used in your code).
4. Simpson's Rule: Approximates the function with parabolic segments.
5. Gaussian Quadrature: Approximates the function using weighted sum of function values.

We are going to use midpoint rule
Given Question

$$\int_a^b f(x)\,dx \approx \Delta x \sum_{n=0}^{n-1} f(a + (i + .5) * \Delta x, intensity)$$

Where

- $\Delta x = \frac{b-a}{n}$
- $a + (i + 0.5) * \Delta x$ represents the midpoint of the i-th subinterval
- a is the lower bound of integration.
- b is the upper bound of integration.
- n is the number of subintervals.
- i is the index for each subinterval.

Approach

- From interval [a,b] is divided into n equal subintervals, each of width Δx.
- In that each sub interval we are finding out midpoint using $a + (i + 0.5) * \Delta x$
- The sum of f(midpoint,intensity), is multiplied by dx to give the final approximation for the integral.

Code Walkthrough

```
// Function to perform numerical integration using the midpoint rule
// Takes in:
//    f: the function to integrate (pointer to function f1, f2, etc.)
//    a: lower bound of integration
//    b: upper bound of integration
//    n: number of points to use for approximation
//    intensity: a parameter to pass to the function (controls computation
intensity)
float numericalIntegration(FuncPtr f, float a, float b, int n, int intensity) {
    float sum = 0.0f; // Variable to accumulate the result of the integration
    float dx = (b - a) / n;// Step size (difference between consecutive points)

    // Loop through n points and apply the midpoint rule
    for (int i = 0; i < n; ++i) {
        float x = a + (i + 0.5f) * dx;  // Midpoint of the current interval
        sum += f(x, intensity);  // Add the value of the function at this
midpoint to the sum
 // this will call functions f1, f2, f3, or f4 in libfunctions.a
    }

    return sum * dx;  // Multiply by dx to get the final integral approximation
}
```

```
// Start measuring the time taken for numerical integration
    auto start = std::chrono::high_resolution_clock::now();  // Get the start
time
    // Perform the numerical integration using the selected function
    float result = numericalIntegration(selectedFunc, a, b, n, intensity);
    // Stop measuring the time after the integration is complete
    auto end = std::chrono::high_resolution_clock::now();    // Get the end
time
    std::chrono::duration<double> elapsed = end - start;     // Calculate the
elapsed time
    // Output both the result and time to stdout, separated by a space
    std::cout << std::setprecision(15) << result << " "
              << std::fixed << std::setprecision(6) << elapsed.count() <<
std::endl;
```

Output Generated:

```
1 0.000001 50.000000000000000
2 0.000001 50.000000000000000
3 0.000001 50.000000000000000
4 0.000001 50.000000000000000
5 0.000001 50.000000000000000
6 0.000001 49.999996185302734
7 0.000001 50.000000000000000
8 0.000001 50.000000000000000
9 0.000001 50.000003814697266
10 0.000001 50.000000000000000
20 0.000001 50.000000000000000
30 0.000001 50.000000000000000
40 0.000002 50.000000000000000
50 0.000002 50.000000000000000
60 0.000002 49.999996185302734
70 0.000002 50.000003814697266
```