

High Performance Scientific Computing

Homework 1

Report

Ayan Tanwar, 22b0931

Contents

1	Introduction	2
2	Convergence Study	2
3	Timing Study	3

1 Introduction

In this report, we studied the influence of the number of iterations and number of threads on the runtime of the program. Using OpenMP, we parallelized the serial version of the code, making it more robust, and applied some optimizations to reduce the time even further.

2 Convergence Study

Given function: $\cos(x)$

Integral limits: $(-\frac{\pi}{2}, \frac{\pi}{2})$

Analytical integral value: 2

I plotted the errors vs. number of iterations in log scale (to capture even minute errors) and observed that, in the case of Trapezoidal Integration, the $\log(\text{error})$ followed a decreasing linear trend. On the other hand, the $\log(\text{error})$ for Monte Carlo integration was decreasing but not smooth, instead quite erratic, mainly due to the randomness inherent in the method. I performed this analysis for 10^6 iterations, as beyond that the error became too insignificant to count.

Below are the plots for the same:

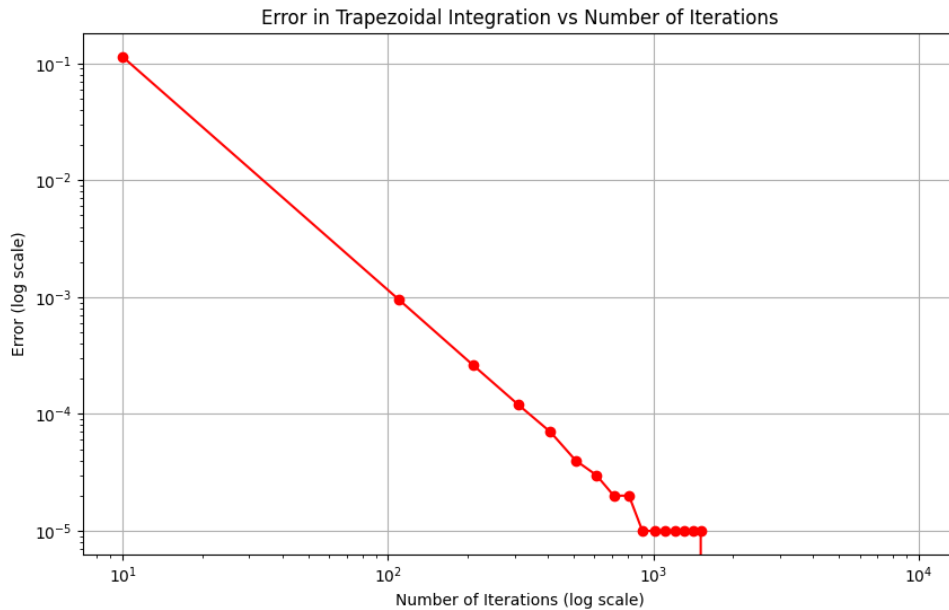


Figure 1: Error vs. Number of Iterations (Trapezoidal Integration)

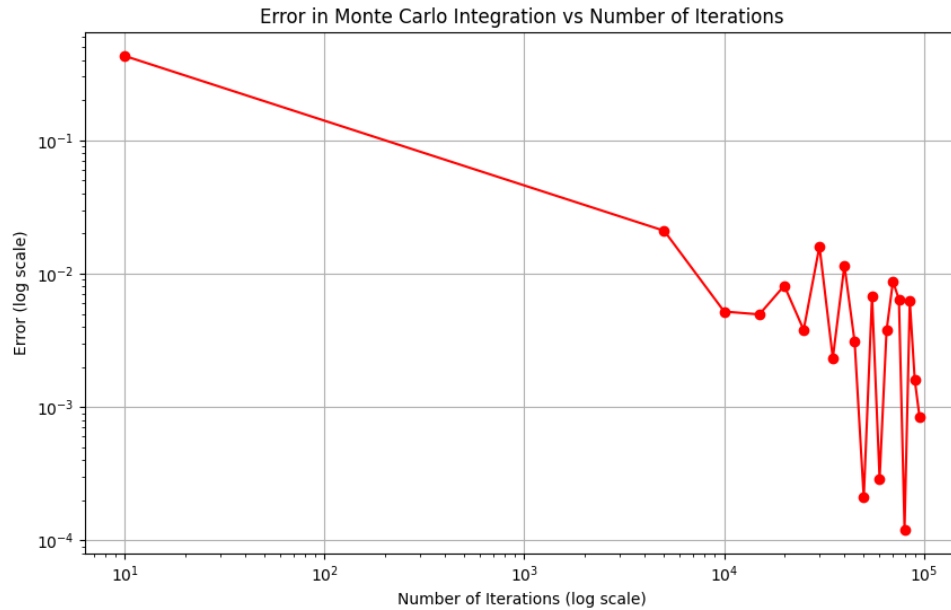


Figure 2: Error vs. Number of Iterations (Monte Carlo Integration)

3 Timing Study

I ran the code with each thread configuration 5 times on large number of iterations to obtain some distinguishable results and averaged the runtimes to get the average runtime for each thread count, for both the Monte Carlo method and the Trapezoidal method of integration. The Trapezoidal method took less time on average compared to the Monte Carlo method, mainly due to the overhead of generating random numbers repeatedly. I also observed that as the number of threads increased, the runtime decreased for both methods. Below are the runtimes for both methods.

```
(PyEnv) ayan@ayan-HP-Pavilion-Laptop-15-eh2xxx:~/Desktop/Work/Parallel_Computing/HW_1$ bash timing_study.sh
Running with 2 threads
Run 1: 4.10109 seconds
Run 2: 4.10611 seconds
Run 3: 4.06415 seconds
Run 4: 4.10184 seconds
Run 5: 4.07664 seconds
Average Time for 2 threads: 4.0899 seconds

Running with 4 threads
Run 1: 2.50065 seconds
Run 2: 2.50965 seconds
Run 3: 2.50581 seconds
Run 4: 2.51689 seconds
Run 5: 2.50868 seconds
Average Time for 4 threads: 2.5083 seconds

Running with 6 threads
Run 1: 1.91393 seconds
Run 2: 1.90401 seconds
Run 3: 1.9015 seconds
Run 4: 1.91084 seconds
Run 5: 1.90174 seconds
Average Time for 6 threads: 1.9064 seconds

Running with 8 threads
Run 1: 1.90712 seconds
Run 2: 1.90175 seconds
Run 3: 1.90278 seconds
Run 4: 1.9029 seconds
Run 5: 1.89503 seconds
Average Time for 8 threads: 1.9019 seconds
```

Figure 3: Timing Study (iterations : 10^9) (Trapezoidal Integration)

```
(PyEnv) ayan@ayan-HP-Pavilion-Laptop-15-eh2xxx:~/Desktop/Work/Parallel_Computing/HW_1$ bash timing_study.sh
Running with 2 threads
Run 1: 20.4295 seconds
Run 2: 16.1283 seconds
Run 3: 16.2883 seconds
Run 4: 16.2406 seconds
Run 5: 16.2705 seconds
Average Time for 2 threads: 17.0714 seconds

Running with 4 threads
Run 1: 9.35007 seconds
Run 2: 9.34913 seconds
Run 3: 9.40084 seconds
Run 4: 9.35475 seconds
Run 5: 9.41093 seconds
Average Time for 4 threads: 9.3731 seconds

Running with 6 threads
Run 1: 6.99614 seconds
Run 2: 6.99965 seconds
Run 3: 7.0034 seconds
Run 4: 7.41463 seconds
Run 5: 6.99513 seconds
Average Time for 6 threads: 7.0017 seconds

Running with 8 threads
Run 1: 5.88048 seconds
Run 2: 5.89701 seconds
Run 3: 5.9233 seconds
Run 4: 5.89377 seconds
Run 5: 5.88702 seconds
Average Time for 8 threads: 5.8963 seconds
```

Figure 4: Timing Study (iterations : 10^8) (Monte Carlo Integration)