ME766: Assignment1

Dikshant 180040033

Q1. Trapezoidal rule and montecarlo method

TRAPEZOIDAL RULE

No of Threads=8

No of Sampling Points	Integral Value
100	1.99827
10^3	1.999998
10^4	2.000000
10^6	2.000000
10^9	2.000000

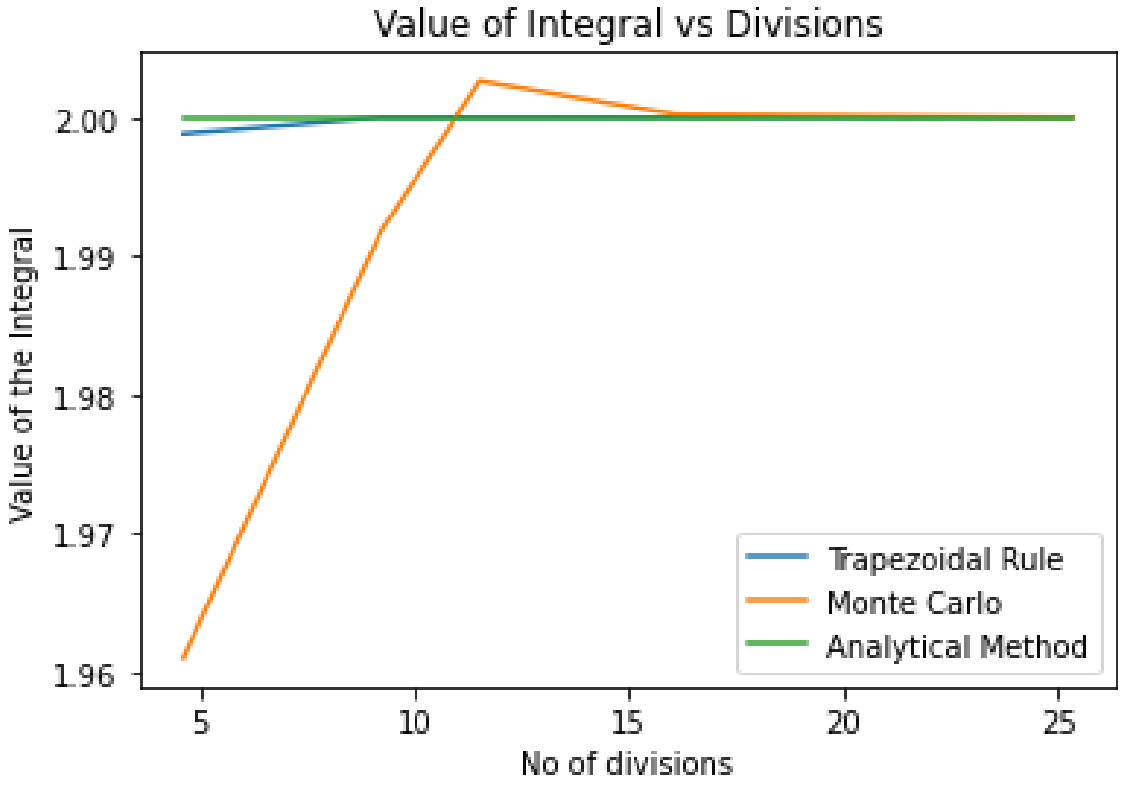
MONTE CARLO METHOD

No of Threads=8

No of Threads	Reading 1	Reading 2	Reading 3	Reading 4	Reading 5	Average Reading
10^2	1.8838 t=0.008s	1.88948 t=0.11s	1.97413 t=0.014s	2.03345 t=0.015s	2.02414 t=0.004s	1.961
10^3	2.05895 t=0.009s	2.04164 t=0.008s	2.01556 t=0.018s	2.05023 t=0.006s	2.04664 t=0.015s	2.042604

10^4	1.98422 t=0.018s	1.99932 t=0.014s	1.99584 t=0.013s	1.98946 t=0.009s	1.98984 t=0.013s	1.991736
10^6	2.00013 t=0.104s	2.00026 t=0.12s	2.00161 t=0.127s	1.99996 t=0.133s	1.99912 t=0.107s	2.000216
10^9	2.00005 t=48.238s	2.00002 t=52.47s	1.99996 t=46.635s	2.0001 t=48.130s	2.00001 t=52.787s	2.000028

2. convergence study



As the number of sampling points increased, the difference between the numerical integral value and the analytical value of the integral converged to zero. In the case of the Trapezoidal Rule, this was achieved at a lesser number of sampling points(10^4) than the Monte-Carlo Method(10^6).

The value of the integral remained constant when the no of samplings was made constant in the trapezoidal rule. This was not the case in the MonteCarlo method as the samplings were random and hence it gave a different value of integral in every execution with a fixed sampling number.

3. Timing study using 2,4,6 and 8 OpenMP threads.

TRAPEZOIDAL RULE

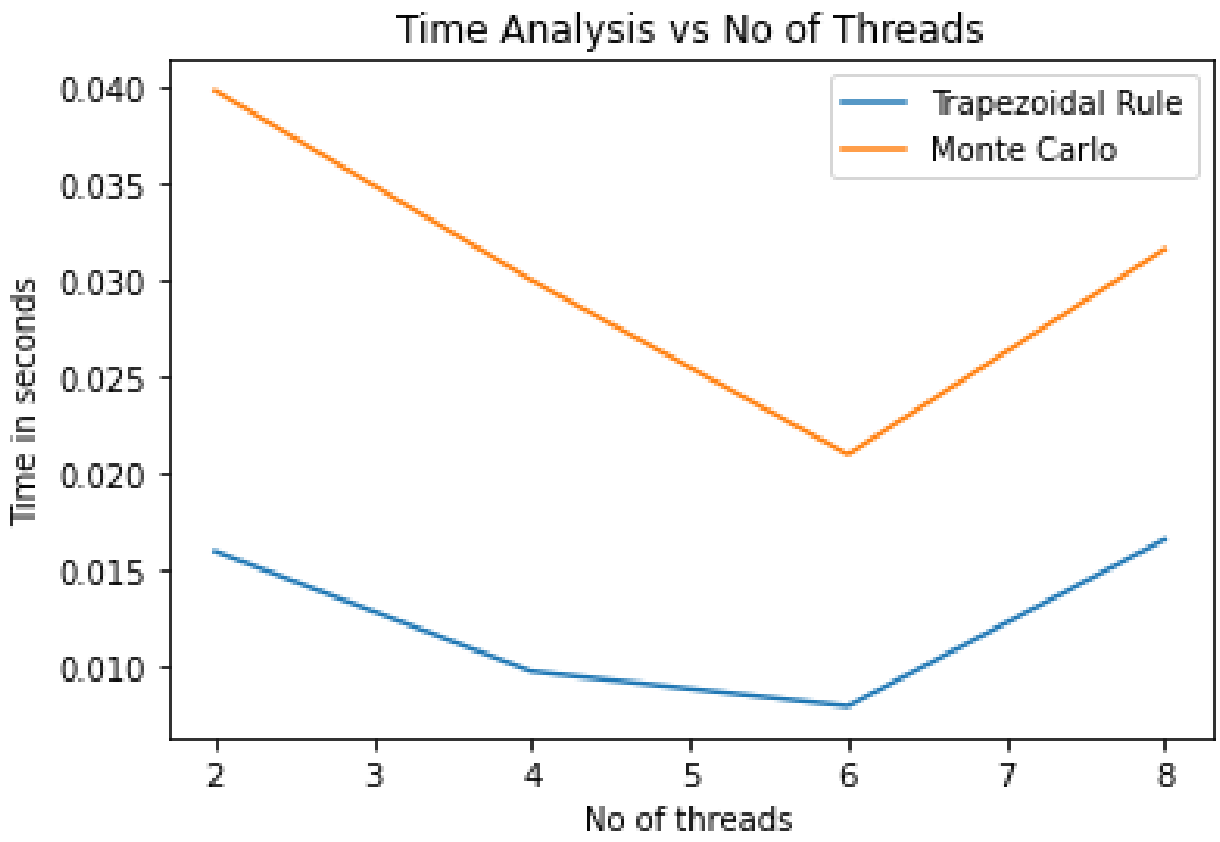
No of Sampling=100000

No of Threads	1	2	3	4	5	Avg
2	0.016s	0.014s	0.018s	0.017s	0.015s	**.01600**
4	0.011s	0.009s	0.008s	0.010s	0.011s	**0.0098**
6	0.007s	0.011s	0.006s	0.008s	0.008s	**0.0080**
8	0.018s	0.018s	0.017s	0.014s	0.016s	**0.0166**

MONTE CARLO METHOD

No of Sampling=100000

No of Threads	1	2	3	4	5	Avg
2	0.035s	0.044s	0.040s	0.038s	0.042s	**0.0398**
4	0.034s	0.022s	0.032s	0.027s	0.035s	**0.030**
6	0.012s	0.016s	0.013s	0.012s	0.010s	**0.021**
8	0.045s	0.030s	0.018s	0.045s	0.020s	**0.0316**



Ambiguity:
General Trend:
An increase in the number of threads decreased the computational time.
However, the trend showed a deviation with no_of_threads=8.
This was expected as my laptop has 4 core threads.
Explanation:

Let N be the number of CPU cores available.
If we utilize N threads: all the N threads will work on the CPU. However, the CPU usage will not be 100% optimized ((at some point there will always be I/O to wait for).
If we utilize N+1 threads in our code, Then N threads can work the CPU while 1 thread is waiting for disk I/O.
However, If we use many threads, that would cause threads to fight over the CPU resource.