# Homework 1

Krut Patel

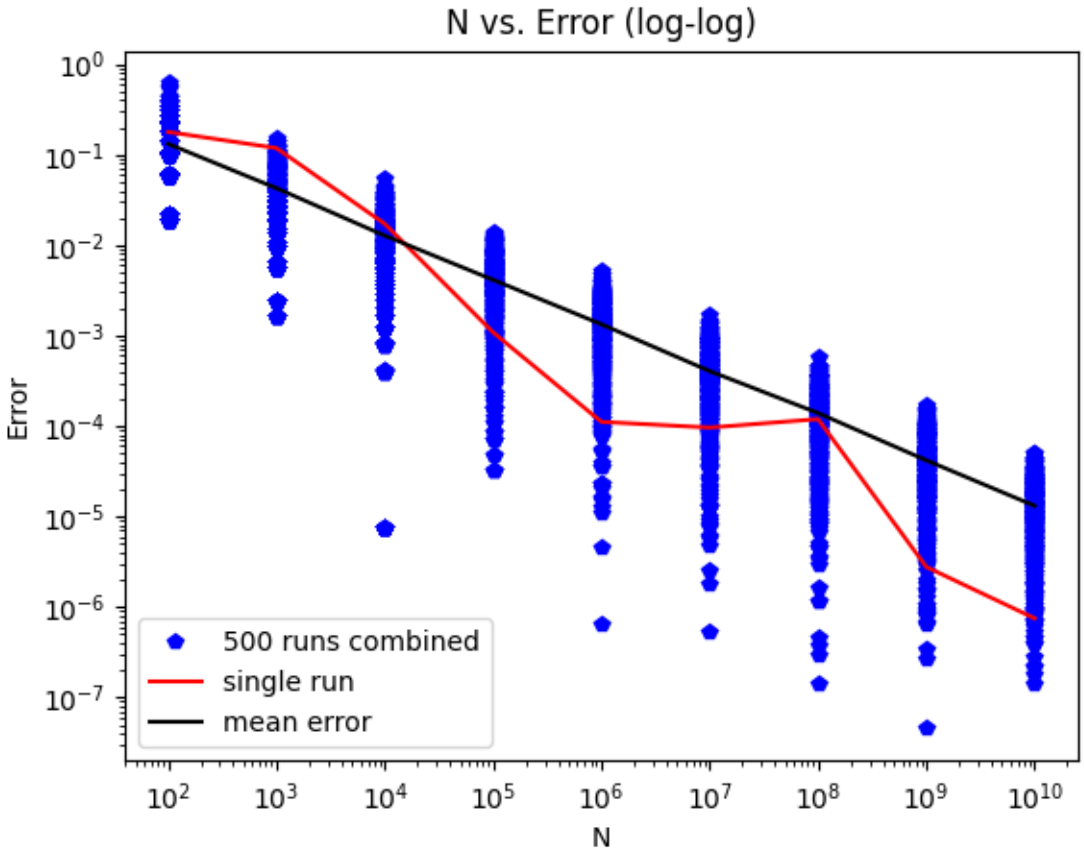
November 6,2023

## Problem 1

### Part A: C code

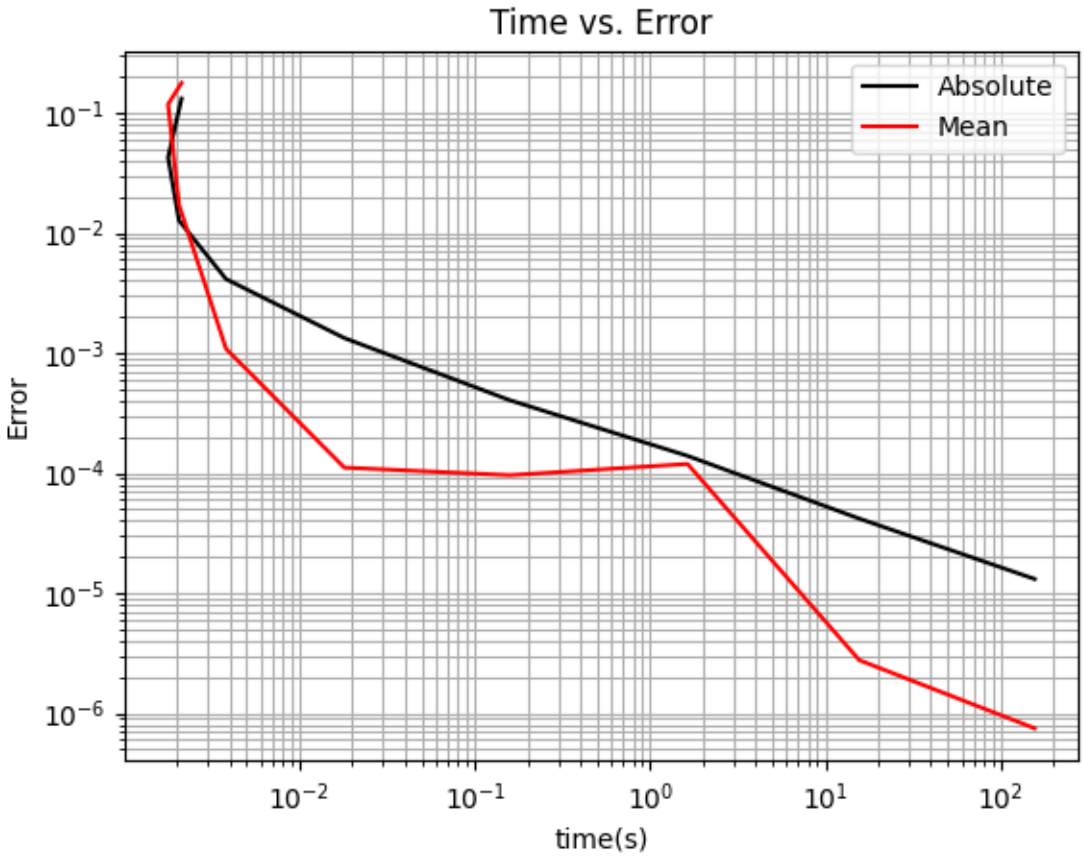
```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <math.h>

#define PI 3.14159265358979323846

int main(int argc, char *argv[]) {       //Checking if the given value of N is correct
    if (argc != 2) {
        printf("Usage: %s N\n", argv[0]);
        return 1;
    }

    unsigned long long N = strtoull(argv[1], NULL, 10);  //Unsigned long long is used
                                                         //as we're gonna take the N
                                                         //value up to 10^10

    srand((unsigned)time(NULL));     //set the random value generator according to time

    unsigned long long points_inside = 0;

    for (unsigned long long i = 0; i < N; i++) {         //Loop to generate N number of
                                                         //random x1 and x2 points

        double x1 = 2.0 * ((double)rand() / (double)RAND_MAX) - 1.0;
        double x2 = 2.0 * ((double)rand() / (double)RAND_MAX) - 1.0;

        if (x1 * x1 + x2 * x2 <= 1.0) {          //adding those points of the vector
                                                 //x=(x1, x2) which are inside the
                                                 //circle of radius 1

            points_inside++;
        }
    }

    double pi_approx = 4.0 * points_inside / N;          //Monte Carlo approximation
    double error = fabs(PI - pi_approx);
    printf("%llu %e\n", N, error);
    return 0;
}
```

## Part B: Error Analysis




This plot shows the N value vs errors of a single run, 500 combined runs, and the mean error of those 500 runs. We can compare the results of mean error with Homework 1. As we increased the run number from 30 to 500, we can now see that the graph of mean error vs N value is very close to a straight line. The slope of this line is exactly 0.5, which is the correct rate of decrement in error in the Monte Carlo simulations. These may runs were achieved using parallel program in the Unix environment, the scripts for which are uploaded on the bitbucket repo.

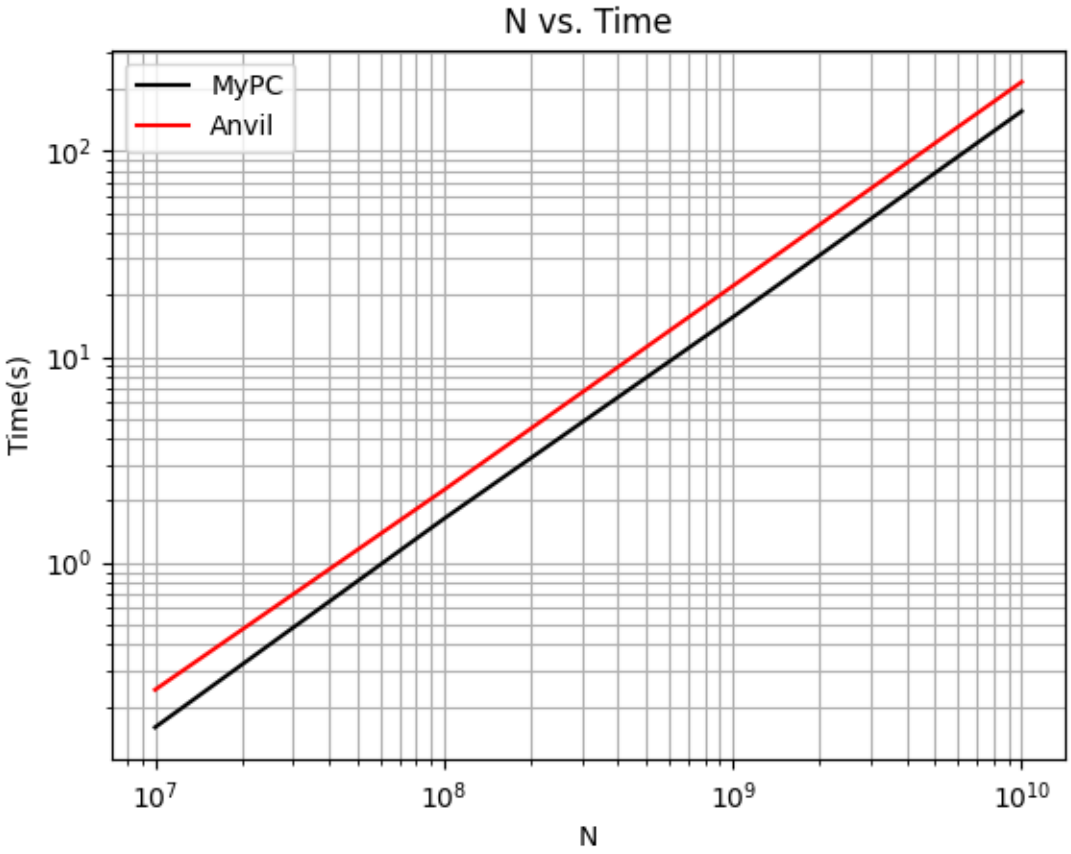
## Part C: Time Analysis




As suggested in the homework, I first plotted the absolute error with time, which turns out to be very uneven. This is due to the fact that the number generated for x1 and x2 are completely random. For better approximations of the accuracy, I also plotted time vs mean error. I could do that as the time values for different runs were similar to first decimal place. According to this second plot, the estimation of time for an accuracy of $10^{-16}$ was $1.62 * 10^{12}$, and for an accuracy of $10^{-70030}$, it was $1.62 * 10^{70026}$.

Just as the Homework 1, the Unix program 'time' was giving me just the total run-time of the whole program. I wrote a bash script which calculates the start time and end time at the beginning and ending of the Monte Carlo program execution. At the end of the loop, I've calculated 'end time - start time' to find the elapsed time for each iteration.

## Problem 2




This plot compares the runtime with the value of N in both my PC and Anvil(Shared Partition) on a single core. If we compare the times, we can see that Anvil's single core takes a little bit more time than my PC. My PC's processor has a maximum clock speed of 4.46 GHz. Whereas Anvil's shared partitions' processors have the maximum clock speed of 3.53 GHz. As the number of FLOPS solely depends on clock speeds, this variation in time does make sense.

Submitted to - Professor Scott Field