Home Work 02: EAS 520

Sayem Khan

Tuesday, October 22, 2019

Task 1

Problem a

Write a C, C++ or Fortran program that computes this Monte Carlo approximation to π for $N = 10^k$

Solution

The problem is given below:

Solution

```
#include <iostream>
#include <cstdlib>
#include <ctime>
#include <cmath>

using namespace std;

double mcpi(long int NumPoint);           // main calculation
double error(double itegral);             //Error calculation
int indicator_fqn(double x, double y);    // To check where is the point
double interval_map(double a, double b); // mapping the interval

int main(int argc, char **argv)
{
    double piN;
    srand(time(NULL));
    //Declearation and Initialization of the variables
    long int N = atol(argv[1]); // Number of Random point
    piN = mcpi(N);
    // cout << N << " " << piN << " " << error(piN) << endl;
    cout << N << " " << error(piN) << endl;
    return 0;
}

double mcpi(long int NumPoint)
{
    long int N_attemps = NumPoint; //Total Number of points
    double pi = 0.0;               //Approximate integration
    double a = -1.0;               //lower bound
    double b = 1.0;                // upper bound
    long int N_hits = 0.0;

    for (long int i = 0; i < N_attemps; i++)
    {
        double x = interval_map(a, b);
```

```
        double y = interval_map(a, b);
        N_hits = N_hits + indicator_fqn(x, y);
    }

    //cout << N_hits << endl;

    /******************************
     *      pi = 4 * (Nhits/N)
    ******************************/
    pi = 4 * (N_hits / (double)N_attemps);
    // cout << N_attemps << " " << pi << " " << error(pi) << endl;
    return pi;
}

double interval_map(double a, double b)
{
    /**************************************************
     * Mapping the [0,1]=>[lowerLim, upperLim]
     * f(x) = mx + c, f(0) = lowerLim and f(1) = upperLim
     * Solving this we get, f(x) = (upperLim-lowerLim)x + lowerLim
    **************************************************/
    double upperLim = b;
    double lowerLim = a;
    double randomNumber = ((double)rand()) / ((double)RAND_MAX);
    return ((upperLim - lowerLim) * randomNumber + lowerLim);
}

int indicator_fqn(double x, double y)
{
    /****************************
     * For MC pi, x^2 + y^2 <= 1
    ****************************/
    if (x * x + y * y <= 1.0)
    {
        return 1;
    }
    else
    {
        return 0;
    }
}

double error(double piN)
{
    const double pi = 3.141592653589793;
    double absolute_error = fabs(piN - pi);
    return absolute_error;
}
```

Listing 1: Code for Monte Carlo approximation to π

Problem b

Plot N vs absolute error $|\pi - \pi_N|$ on a log-log plot.

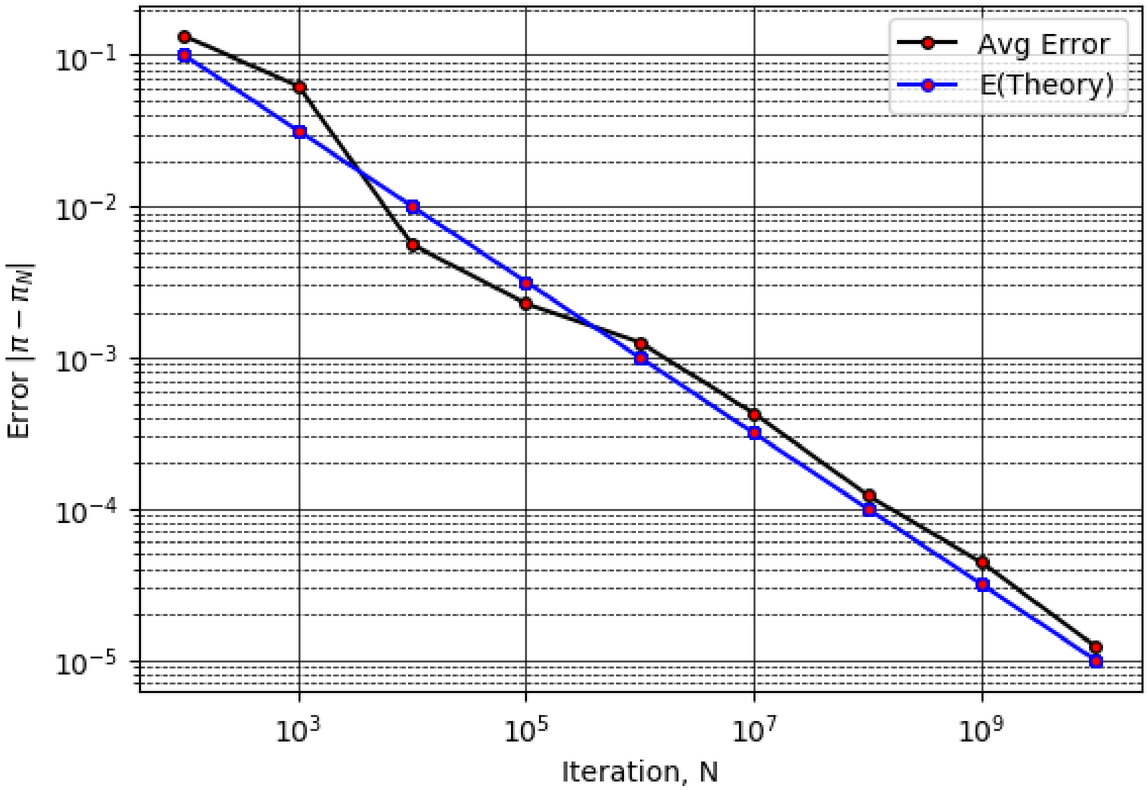

Figure 1: Numerical Integration Err vs Iteration: Problem a, Task 1

Solution

Here, The program run for 50 times for a each data and average error is taken. Also, the theoretical relation between error and iteration of MC method, i.e., $E \propto T^{-1/2}$ is also plotted. Comparing to the theoretical limit and calculated limit, it can be error is decreasing at the correct rate.

```
#!/bin/bash
g++ -Wall -o monteCarlopi monteCarlopi.cpp

file_1="avgErrdata.dat"

if [ -f $file_1 ] ; then
    rm $file_1
fi

i=2
while [[ i -le 10 ]]
do
echo "i = "$i "starts"
    for (( j=1; j<=50; j++ ))
     do
    ./monteCarlopi $((10**i)) >> avgErrdata.dat
     done
echo "i = "$i "is done"
((i = i + 1))
done

python3 plot.py
```

Listing 2: Bash script **avgData.sh** for Problem b, Task 1

Problem c

Error vs runtime calculation.

Solution

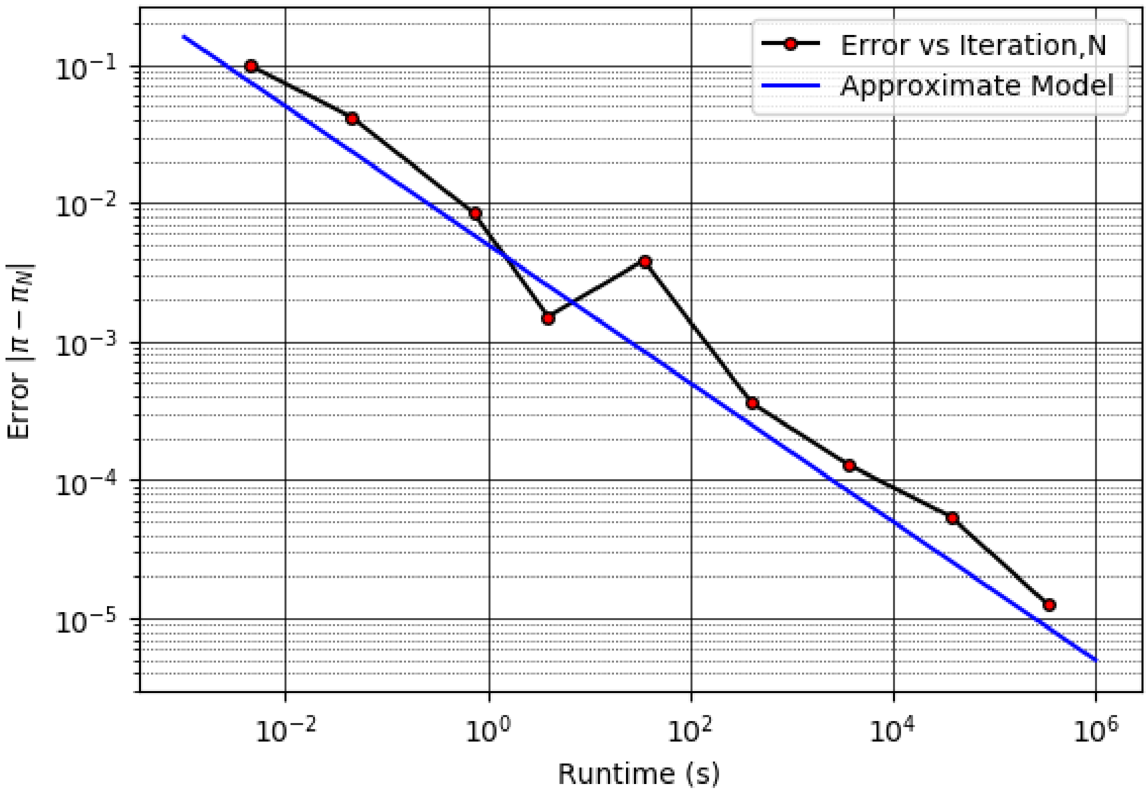

Figure 2: Numerical Integration Err vs Iteration: Problem c, Task 1

For the data suing linear regression, it is found that, for accuracy level $E = 10^{-16}$, the program will take 10^{27} second (only!) and for $E = 10^{-70030}$ it will take (only!) 10^{140026} seconds! Using theoretical calculation and trial-error, best model is $E \propto T^{-1/2}$.

For the calculating the required time, C++ library is used rather than Linux **time** command. The code is given below:

```
#include <chrono>
int main(int argc, char **argv)
{
    double piN;
    srand(time(NULL));
    //Declearation and Initialization of the variables
    long int N = atol(argv[1]); // Number of Random point
    auto t_start = std::chrono::high_resolution_clock::now();
    piN = mcpi(N);
    auto t_end = std::chrono::high_resolution_clock::now();
    std::chrono::duration<double, std::milli> duration = (t_end - t_start);
```

```
    cout << N << " " << duration.count() << " " << error(piN) << endl;
    return 0;
}
```

Listing 3: Code for Problem c, Task 1 (Only the focused part)

This program is run 10 times for each $N = 10^k$ and average value for runtime and error is taken. The driver program for this operation is given below:

```bash
#!/bin/bash
g++ -Wall -o monteCarlopi_time monteCarlopi_time.cpp

file_1="time.dat"

if [ -f $file_1 ] ; then
    rm $file_1
fi

# for (( i=2; i<=10; i++ ))
# do
#   #/usr/bin/time -f "%e"  ./monteCarlo_time $((2**i)) &>> time.dat
#     ./monteCarlopi_time $((10**i)) >> time.dat
#     echo "i = "$i "is done"
# done

i=2
while [[ i -le 10 ]]
do
echo "i = "$i "starts"
    for (( j=1; j<=10; j++ ))
     do
    ./monteCarlopi_time $((10**i)) >> time.dat
     done
echo "i = "$i "is done"
((i = i + 1))
done

python3 timeplot.py
```

Listing 4: Bash script **time.sh** for Problem c, Task 1

Task 2

Problem a

Plotting the likelihood function for model 1

$$L(x_1, x_2; M_1) = e^{-(1-x_1)^2 - 100(x_2 - x_1^2)^2} \tag{1}$$

Solve a

We can rewrite the equation as,

$$ln(L(x_1, x_2; M_1)) = -(1 - x_1)^2 - 100(x_2 - x_1^2)^2 \tag{2}$$

Contour plot for equation 2 is given below: The model's degeneracy will show up as lines in the $x_1 - x_2$ plane where the value of $L(x_1, x_2; M_1) = e^{-(1-x_1)^2 - 100(x_2 - x_1^2)^2}$ does not change.

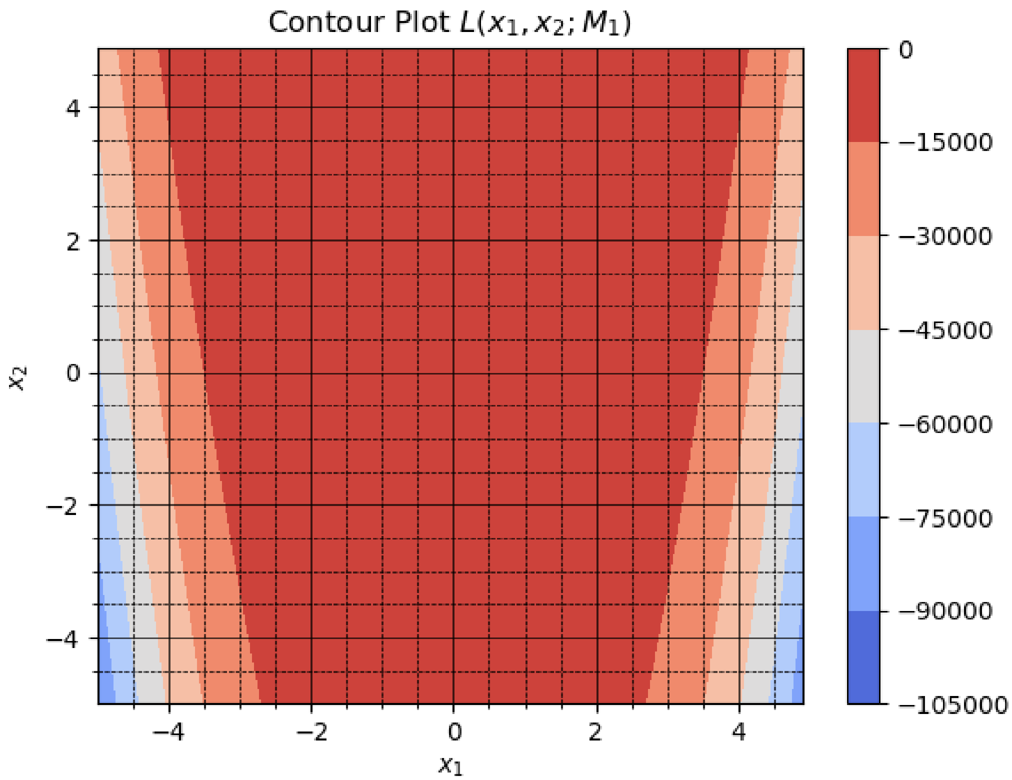

Figure 3: Contour plot for

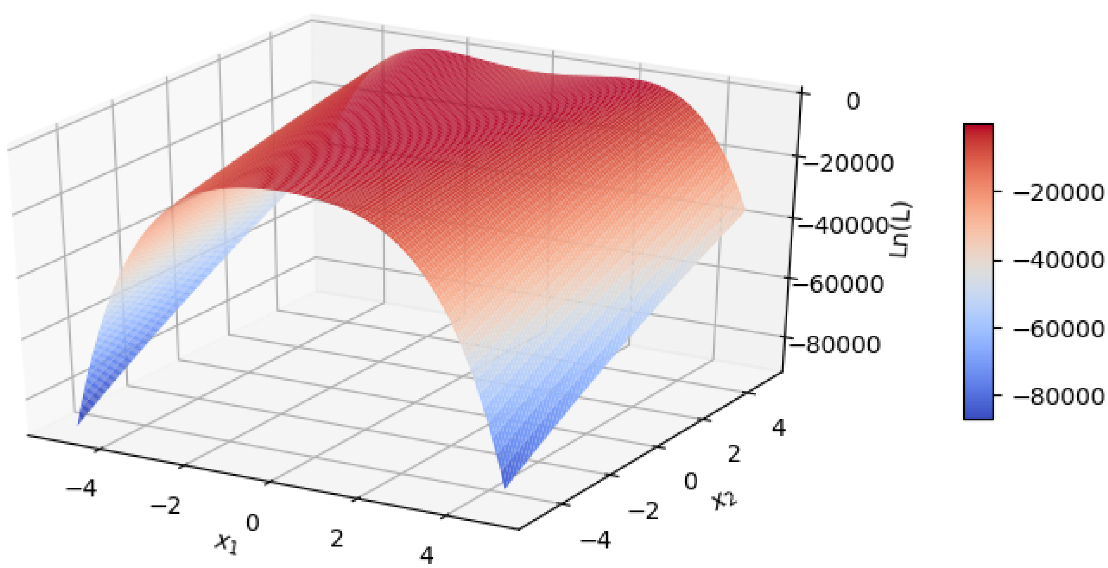

Figure 4: 3D plot for

Problem b & c

Using Monte Carlo compute,

$$Z_1 = \int_{-5}^{5} \int_{-5}^{5} L(x_1, x_2; M_1)$$

Solve b & c

C++ code is give below:

```
#include <iostream>
#include <cstdlib>
#include <ctime>
#include <cmath>

using namespace std;

double integration_fqn(long int NumPoint);                  // main calculation
double error(double itegral);                               //Error calculation
double likelihood_fqn(double x, double y);                  // To check where is the point
double interval_map(double a, double b);                    // mapping the interval
double volume(double a1, double b1, double a2, double b2); // function for 2D voulume calculation

int main(int argc, char **argv)
{
    double intigral;
    srand(time(NULL));
    //Declearation and Initialization of the variables
    long int N = atol(argv[1]); // Number of Random point
    intigral = integration_fqn(N);
    // cout << N << " " << piN << " " << error(piN) << endl;
    cout << N << " " << intigral << " " << log(intigral) << endl;
    return 0;
}

double integration_fqn(long int NumPoint)
{
    long int N_attemps = NumPoint; //Total Number of points
    double I = 0.0;                //Approximate integration
    double a = -5.0;               //lower bound
    double b = 5.0;                // upper bound
    double N_hits = 0.0;

    for (long int i = 0; i < N_attemps; i++)
    {
        double x = interval_map(a, b);
        double y = interval_map(a, b);
        N_hits = N_hits + likelihood_fqn(x, y);
    }

    double v = volume(-5.0, 5.0, -5.0, 5.0);

    I = v * (N_hits / (double)N_attemps);

    // cout << N_attemps << " " << pi << " " << error(pi) << endl;
    return I;
}

double interval_map(double a, double b)
{
    /***************************************************
     * Mapping the [0,1]=>[lowerLim, upperLim]
```

```
     * f(x) = mx + c, f(0) = lowerLim and f(1) = upperLim
     * Solving this we get, f(x) = (upperLim-lowerLim)x + lowerLim
    ******************************************************/
    double upperLim = b;
    double lowerLim = a;
    double randomNumber = ((double)rand()) / ((double)RAND_MAX);
    return ((upperLim - lowerLim) * randomNumber + lowerLim);
}

double likelihood_fqn(double x, double y)
{
    return exp(-(pow((1.00 - x), 2.00)) - 100.00 * (pow((y - pow(x, 2.00)), 2.00))
   );
}

double volume(double a1, double b1, double a2, double b2)
{
    return (b1 - a1) * (b2 - a2);
}

// double error(double piN)
// {
//     const double pi = 3.141592653589793;
//     double absolute_error = fabs(piN - pi);
//     return absolute_error;
// }
```

Listing 5: Code for Problem b & c, Task 2

The driver program for this operation is given below:

```
#!/bin/bash
g++ -Wall -o likelihood_fqn likelihood_fqn.cpp

file_1="likelihood.dat"

if [ -f $file_1 ] ; then
    rm $file_1
fi

i=2
while [[ i -le 10 ]]
do
  ./likelihood_fqn $((10**i)) >> likelihood.dat
echo "i = "$i "is done"
((i = i + 1))
done
```

Listing 6: Bash script **time.sh** for Problem a & c, Task 2

The values of Z_1 and $ln(Z_1)$ is given below:

N	Z_1	lnZ_1
100	0.517913	-0.657948
1,000	0.317282	-1.14796
10,000	0.270797	-1.30639
$1{\cdot}10^5$	0.29388	-1.22458
$1{\cdot}10^6$	0.297553	-1.21216
$1{\cdot}10^7$	0.300579	-1.20205
$1{\cdot}10^8$	0.301705	-1.19831
$1{\cdot}10^9$	0.301509	-1.19896
$1{\cdot}10^{10}$	0.301561	-1.19878

From the above table, for 2 digit precision the N value should be around 10^7.

Problem d

Compare the runtime between stampede2 and my Laptop.

Solve d

Runtime for $N = 10^9$ of the code:

1. My Laptop: $70.42s$
2. stampede2: $562.6s$

The stampede2 (Model name: Intel(R) Xeon(R) Gold 6132 CPU @ 2.60GHz, CPU(s) = 28) takes 7.9 times time than my Laptop (Intel(R) Core(TM) i5-6200U CPU @ 2.30GHz , CPU(s) = 4). The stampede2 is faster than my Laptop, but takes more time!

Problem e

Solve e

For 40 trials my Laptop will take $\frac{(70.42\times40)}{4} = 704s$ and stampede2 will take $\frac{(562.6\times40)}{28} = 803.71s$

Problem f

Solve f

For the 1st likelihood fqn,

$$\begin{aligned} Z_1 &= \int_{-5}^{5}\int_{-5}^{5} L(x_1, x_2; M_1)dx_1dx_2 \\ &= 3 : 013496893 \times 10^{-1}; \end{aligned}$$

Here,

$$L(x_1, x_2; M_1) = e^{-(1-x_1)^2-100(x_2-x_1^2)^2}$$

For the 2nd likelihood fqn,

$$\begin{aligned} Z_1 &= \int_{-5}^{5}\int_{-5}^{5} L(x_1, x_2; M_2)dx_1dx_2 \\ &= 100 \end{aligned}$$

$$L(x_1, x_2; M_2) = 1$$

Since, $Z_2 > Z_1$, so, Z_2 or model 2 fits data better.