CS3811 - High Performance Computing and Big Data Lab

Lab 1

> Name: M K Lokesh Kumar

> Registration No.: 2201113026

> Class: Cyber Security(Semester 5)

Experiment 1

Objective

To estimate the value of π using a randomised algorithm technique(Monte Carlo), and plot 2 performance graphs:

- Graph 1 - Time taken by the code vs Number of samples
- Graph 2 - Error vs number of samples

Code

Written in C++.

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

double estimatePi(int numSamples) {
    int pointsInsideCircle = 0;

    for (int i = 0; i < numSamples; i++) {
        double x = static_cast<double>(rand()) / RAND_MAX;
        double y = static_cast<double>(rand()) / RAND_MAX;

        if (x * x + y * y <= 1) {
            pointsInsideCircle++;
        }
    }

    return 4.0 * pointsInsideCircle / numSamples;
}
```

```
int main() {
    srand(static_cast<unsigned int>(time(0)));
    const int maxSamples = 1000000;
    const int step = 10000;

    std::vector<int> sampleSizes;
    std::vector<double> times;
    std::vector<double> errors;
    std::vector<double> pi;

    for (int numSamples = step; numSamples <= maxSamples; numSamples +=
step) {
        auto start = std::chrono::high_resolution_clock::now();
        double piEstimate = estimatePi(numSamples);
        auto end = std::chrono::high_resolution_clock::now();

        std::chrono::duration<double> duration = end - start;
        double timeTaken = duration.count();
        double error = std::abs(M_PI - piEstimate);

        sampleSizes.push_back(numSamples);
        times.push_back(timeTaken);
        errors.push_back(error);
        pi.push_back(piEstimate);
    }

    std::cout << "Samples\tTime\tError\tPi Estimate" << std::endl;
    for (size_t i = 0; i < sampleSizes.size(); ++i) {
        std::cout << sampleSizes[i] << "\t" << times[i] << "\t" <<
errors[i] << "\t" << pi[i] << std::endl;
    }

    return 0;
}
```

Gnuplot script for plotting performance graphs 1 and 2

```
set terminal pngcairo enhanced font 'Verdana,12'

set output 'ex_1_time_vs_samples.png'
set title 'Time Taken vs Number of Samples'
set xlabel 'Number of Samples'
set ylabel 'Time Taken (seconds)'
plot 'data1.dat' using 1:2 with linespoints title 'Time Taken'

set output 'ex_1_error_vs_samples.png'
set title 'Error in Estimation of π vs Number of Samples'
set xlabel 'Number of Samples'
set ylabel 'Error'
```

```
plot 'data1.dat' using 1:3 with linespoints title 'Error'
```

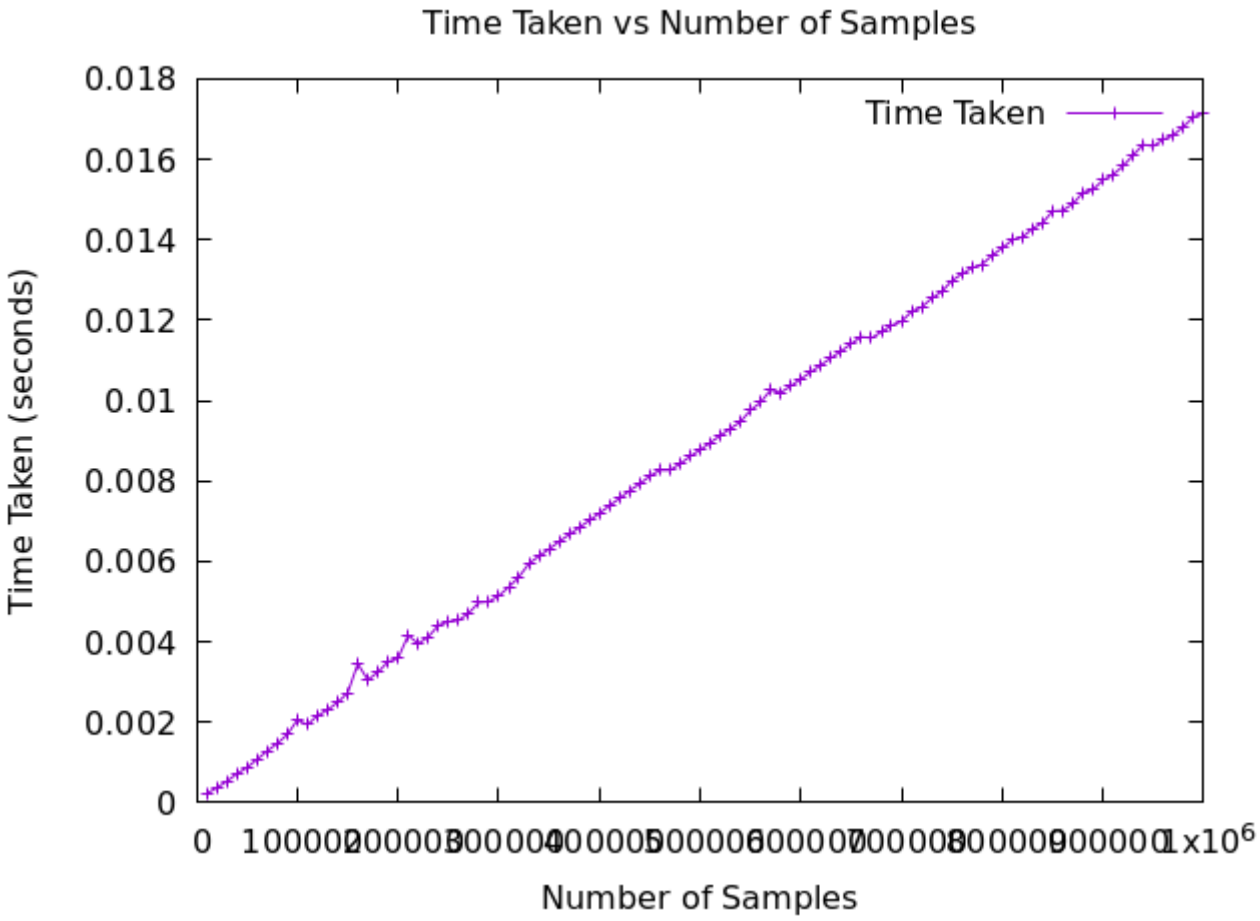
Output

Samples	Time	Error	Pi Estimate	
10000	0.000278414	0.000807346	3.1424	
20000	0.000610885	0.0120073	3.1536	
30000	0.000860183	0.00239265	3.1392	
40000	0.00113147	0.00409265	3.1375	
50000	0.00138829	0.00952735	3.15112	
60000	0.00167266	0.00754068	3.14913	
70000	0.001954	0.00416408	3.13743	
80000	0.00227491	0.00649265	3.1351	
90000	0.00258685	0.00227401	3.14387	
100000	0.00278274	0.00795265	3.13364	
110000	0.00321252	0.00355629	3.13804	
120000	0.003246	0.000592654	3.141	
130000	0.00361894	0.00728496	3.13431	
140000	0.00403375	0.0053502	3.14694	
150000	0.0042086	0.00200735	3.1436	
160000	0.0045163	0.000967654	3.14062	
170000	0.00475619	0.00530146	3.14689	
180000	0.00502096	0.0019629	3.14356	
190000	0.00533114	0.0064284	3.14802	
200000	0.00536131	0.0100127	3.13158	
210000	0.00588245	0.00242639	3.14402	
220000	0.00611366	0.000992654	3.1406	
230000	0.00612452	0.001697	3.1399	
240000	0.00639209	0.00817401	3.14977	
250000	0.0069648	0.000952654	3.14064	
260000	0.00720612	0.00582342	3.13577	
270000	0.00737869	7.4684e-06	3.14159	
280000	0.00773827	0.00129306	3.14289	
290000	0.00797314	0.000986657	3.14258	
300000	0.00817643	0.00497932	3.13661	
310000	0.00839234	0.000147492	3.14145	
320000	0.00892622	0.000344846	3.14194	
330000	0.0090751	0.00348613	3.14508	
340000	0.00930424	0.000651477	3.14094	
350000	0.0091494	0.0031502	3.14474	
360000	0.0104005	0.000851791	3.14244	
370000	0.0108487	0.00149924	3.14309	
380000	0.0107987	0.00294002	3.13865	
390000	0.0111005	0.000109911	3.1417	
400000	0.0113232	0.00199265	3.1396	
410000	0.0115427	0.000529239	3.14106	
420000	0.0117598	0.00260735	3.1442	
430000	0.0123287	0.00251897	3.14411	
440000	0.0124723	0.00229826	3.14389	
450000	0.0127661	0.00323401	3.14483	
460000	0.0129028	0.00307961	3.13851	
470000	0.0132835	0.0012267	3.14037	
480000	0.0136069	0.00117599	3.14042	
490000	0.0136849	0.00134776	3.14024	
500000	0.013914	0.00436735	3.14596	
510000	0.0142011	0.000736758	3.14233	
520000	0.0145825	0.000338116	3.14193	
530000	0.0145695	0.00238511	3.13921	
540000	0.0151355	0.00126661	3.14286	
550000	0.0151262	0.00174538	3.13985	

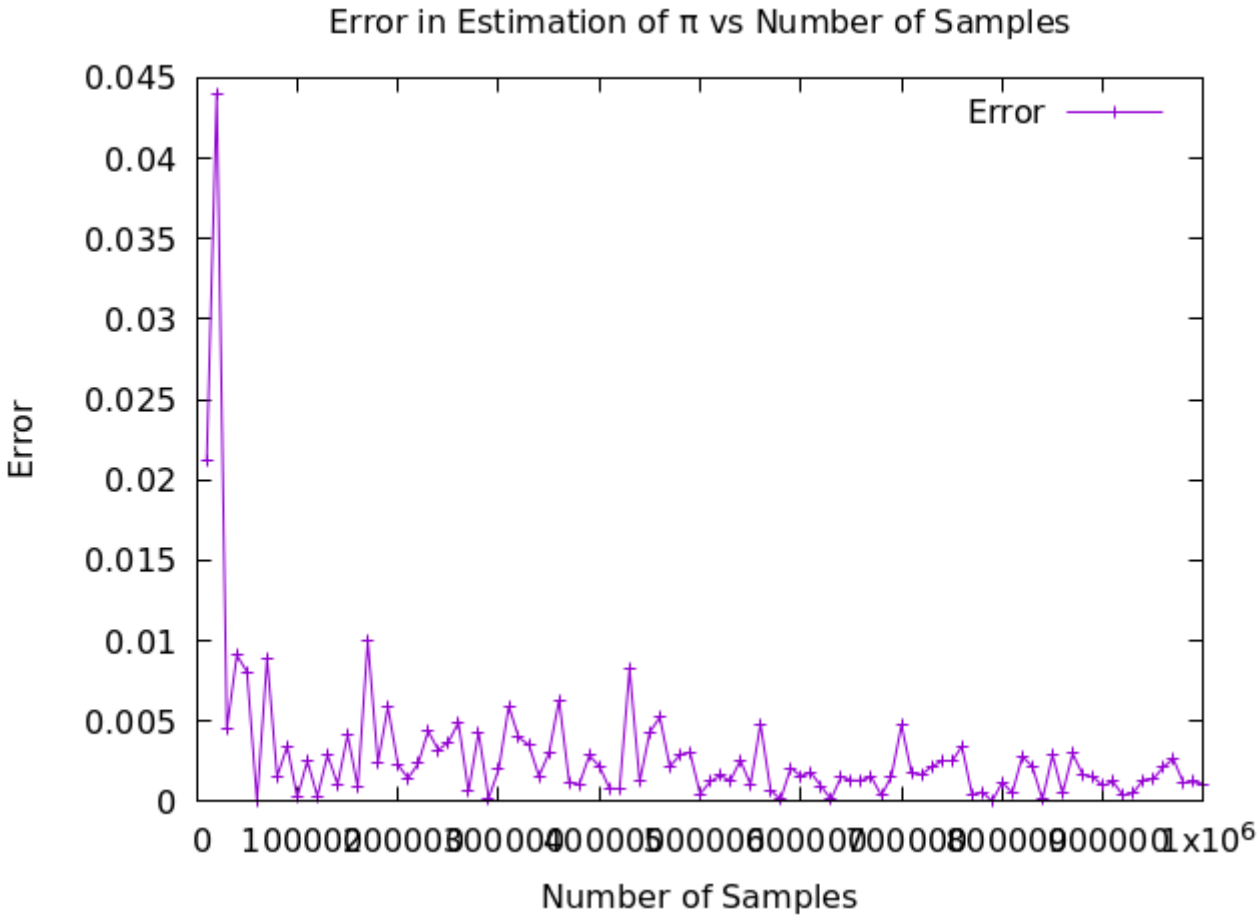
```
560000  0.0147679       7.83679e-05     3.14151
570000  0.0152571       0.00068739      3.14091
580000  0.0153061       0.000992654     3.1406
590000  0.0151568       0.00258362      3.14418
600000  0.0159686       0.00288735      3.14448
610000  0.016713        0.000792654     3.1408
620000  0.0169281       0.00396219      3.14555
630000  0.01734 0.000172426     3.14177
640000  0.0171854       3.23464e-05     3.14162
650000  0.0170866       0.000552654     3.14104
660000  0.017753        0.00178659      3.13981
670000  0.0176369       0.00332078      3.14491
680000  0.0187227       0.000260288     3.14185
690000  0.0196701       0.00139285      3.14299
700000  0.0187286       0.000478368     3.14111
710000  0.0186236       0.00233974      3.14393
720000  0.0192235       0.00249624      3.14409
730000  0.0188429       0.00303375      3.13856
740000  0.0195392       0.00281428      3.13878
750000  0.0194673       0.000167346     3.14176
760000  0.0196735       0.00276524      3.14436
770000  0.0208195       0.000693061     3.14229
780000  0.0208039       0.000920167     3.14251
790000  0.0215895       0.000534426     3.14106
800000  0.0221681       0.000697346     3.14229
810000  0.0222894       0.000891419     3.1407
820000  0.022644        0.00327558      3.13832
830000  0.0226114       0.00129386      3.1403
840000  0.0228508       0.000745035     3.14085
850000  0.0235773       0.000717359     3.14088
860000  0.0240288       0.000998044     3.14259
870000  0.0243733       0.000406447     3.14119
880000  0.0245022       0.00233007      3.14392
890000  0.0246128       0.000325238     3.14127
900000  0.0250051       0.00139265      3.1402
910000  0.0258083       0.00307397      3.13852
920000  0.025845        0.00165352      3.13994
930000  0.0259343       0.000861471     3.14073
940000  0.0261734       0.000190325     3.14178
950000  0.0268085       0.00119686      3.1404
960000  0.0274866       0.00050932      3.14108
970000  0.0267407       0.000363788     3.14123
980000  0.0265272       4.57148e-05     3.14155
990000  0.0265179       0.000512397     3.14211
1000000 0.0281924       0.000848654     3.14074
```

Performance Graphs

- Time taken by the graph vs Number of samples



- Error vs Number of samples



Experiment 2

Objective

Write a C/C++ program to perform matrix-vector multiplication using 2 different methods.

$$b := Ax$$

$$\text{In the above, } b = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_N \end{bmatrix}, A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1N} \\ a_{21} & a_{22} & \cdots & a_{2N} \\ \vdots & & & \vdots \\ a_{N1} & a_{N2} & \cdots & a_{NN} \end{bmatrix} \text{ and } x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{bmatrix}$$

Method 1

$$b_i := \sum_{j=1}^{N} a_{ij}x_j, \forall i \in \{1, 2, \cdots, N\}$$

Method 2

$$\begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_N \end{bmatrix} = x_1 \times \begin{bmatrix} a_{11} \\ a_{21} \\ \vdots \\ a_{N1} \end{bmatrix} + x_2 \times \begin{bmatrix} a_{12} \\ a_{22} \\ \vdots \\ a_{N2} \end{bmatrix} + \cdots + x_N \times \begin{bmatrix} a_{1N} \\ a_{2N} \\ \vdots \\ a_{NN} \end{bmatrix}$$

Also, plot 2 performance graphs:

- Graph 1 - Time taken by Method 1 and Method 2 vs N
- Graph 2 - FLOPs vs N

Code

Writen in C++.

```cpp
#include <iostream>
#include <vector>
#include <chrono>

void method1(const std::vector<std::vector<double>>& A, const
std::vector<double>& x, std::vector<double>& b) {
    int N = A.size();
    for (int i = 0; i < N; i++) {
        b[i] = 0;
        for (int j = 0; j < N; j++) {
            b[i] += A[i][j] * x[j];
        }
    }
```

```
}

void method2(const std::vector<std::vector<double>>& A, const
std::vector<double>& x, std::vector<double>& b) {
    int N = A.size();
    for (int i = 0; i < N; i++) {
        b[i] = 0;
        for (int j = 0; j < N; j++) {
            b[i] += A[j][i] * x[i];
        }
    }
}

int main() {
    const std::vector<int> sizes = {128, 512, 1024, 2048, 4096, 8192};

    std::cout << "N" << "\t" << "Time taken by method 1" << "\t" << "Time
taken by method 2" << "\t" << "FLOPs" << std::endl;

    for (int N : sizes) {
        std::vector<std::vector<double>> A(N, std::vector<double>(N));
        std::vector<double> x(N);
        std::vector<double> b1(N, 0);
        std::vector<double> b2(N, 0);

        for (int i = 0; i < N; i++) {
            for (int j = 0; j < N; j++) {
                A[i][j] = static_cast<double>(rand()) / RAND_MAX;
            }
            x[i] = static_cast<double>(rand()) / RAND_MAX;
        }

        auto start1 = std::chrono::high_resolution_clock::now();
        method1(A, x, b1);
        auto end1 = std::chrono::high_resolution_clock::now();
        std::chrono::duration<double> duration1 = end1 - start1;
        double timeMethod1 = duration1.count();

        auto start2 = std::chrono::high_resolution_clock::now();
        method2(A, x, b2);
        auto end2 = std::chrono::high_resolution_clock::now();
        std::chrono::duration<double> duration2 = end2 - start2;
        double timeMethod2 = duration2.count();

        double flops = 2.0 * N * N;

        std::cout << N << "\t" << timeMethod1 << "\t" << timeMethod2 <<
"\t" << flops << std::endl;
    }

    return 0;
}
```

Gnuplot script for plotting performance graphs 1 and 2

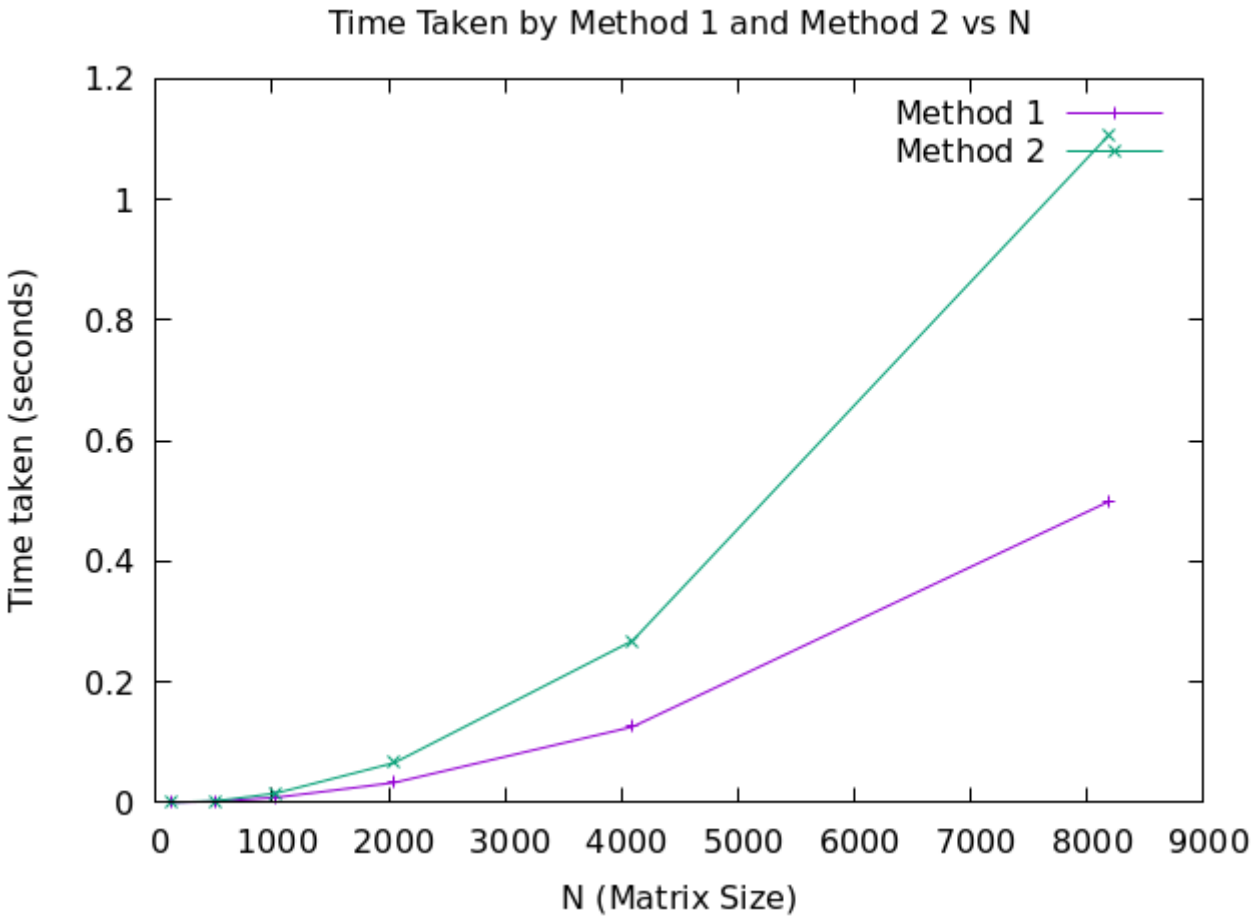
```
set terminal pngcairo enhanced font 'Verdana,12'

set output 'ex_2_time_vs_N.png'
set title 'Time Taken by Method 1 and Method 2 vs N'
set xlabel 'N (Matrix Size)'
set ylabel 'Time taken (seconds)'
plot 'data2.dat' using 1:2 with linespoints title 'Method 1', \
     'data2.dat' using 1:3 with linespoints title 'Method 2'

set output 'ex_2_flops_vs_N.png'
set title 'FLOPs vs N'
set xlabel 'N (Matrix Size)'
set ylabel 'FLOPs'
plot 'data2.dat' using 1:4 with linespoints title 'FLOPs'
```

Output

```
N       Time taken by method 1  Time taken by method 2  FLOPs
128     8.4685e-05      8.7317e-05      32768
512     0.0015207       0.00206153      524288
1024    0.00668042      0.0124618       2.09715e+06
2048    0.0248162       0.0464221       8.38861e+06
4096    0.0923269       0.178096        3.35544e+07
8192    0.402208        0.961143        1.34218e+08
```

Performance Graphs

- Graph 1 - Time taken by Method 1 and Method 2 vs N



- Graph 2 - FLOPs vs N