Simulation and Modeling

Lab Sheet 1: Pseudo-random Numbers

Nischal Regmi
Everest Engineering College, Sanepa

2022

Simulation is an important technique that allows researchers to formulate and test hypotheses when actual data is scarce. The process of simulation mimics actual process in some phenomena of interest, and thus allows to understand how different conditions would lead to different output of the phenomena. We will discuss these details in the theory part.

Here, I would like to mention about the practical part of the course. There are some dedicated programming languages and tools used in simulation based studies. Examples are GPSS, Simscript, NetLogo, etc. General purpose languages like C/C++ and Java are also equally popular in simulation based studies. Nonetheless, as an undergraduate level student, you need to understand the theory behind the algorithms and the implementation details. So, you have to do the lab assignments using a general purpose language. My suggestion is either among C++ or Java will be better for this course. I will be teaching using the C++ language. But you are free to choose any other programming language of your choice.

Note: The syllabus mentions that students are required to develop a simulation model in this course. However, without being familiar with the basic simulation techniques, you would find it difficult to develop any meaningful simulation model. I will thus provide you four lab sheets teaching various concepts, so that you can 'join' these concepts to make a larger program, which will be your project work for the course. The fifth lab sheet will describe you about the 'small project' you will be doing in this course.

1 Pseudo-random Numbers

A linear congruential pseudo-random number generator is defined by the following recursive formula

$$Z_i = (aZ_{i-1} + c)(\text{mod } m) \tag{1}$$

where m is the modulus, a the multiplier, c the increment, and Z_0 is the seed. It is obvious that the maximum value of the random number is $m - 1$.

A pseudo-random generator may not have full period. That is, the generator may produce repeated cycle of same sequence of values, and the length of such sequence is lesser than m. For example, if we take $a = 2$, $b = 2$, $m = 10$, and $Z_0 = 1234$, the linear congruential pseudo-random number generator will repeatedly produce the sequence 2,6,4,0 (check this). In this case, we say the generator has period 4.

For a linear congruential generator to have a full period, we need to assure that

(a) The parameters m and c are relatively prime, i.e. the only positive integer that divides both m and c is 1.

(b) If q is a prime number that divides m, then q divides $a - 1$

(c) If 4 divides m, then 4 divides $a - 1$.

So, you should think what combination of a, c and m satisfy the above three properties.

2 Generating Random Variates

Simulation models may require random numbers whose distribution are other than the uniform distribution. I will not cover the question of generating non-uniform random numbers in the lab. Rather, you will be taught to use random variates from different distributions using standard library functions.

Consider the exponential distribution whose PDF is defined as following.

$$f(x) = \lambda e^{-\lambda x} \qquad x > 0$$

Here, $\lambda > 0$ is called the *rate parameter*. The mean of the exponential distribution is $\beta = 1/\lambda$, which is also called the *scale* of the distribution. The variance of the exponential distribution is $1/\lambda^2$.

Following C++ program generates 10 random numbers from the exponential distribution with $\lambda = 0.2$ and calculates the sample mean.

```
#include <iostream>
#include <random>
using namespace std;

int main()
{
        default_random_engine generator;
```

```
        exponential_distribution<double> distribution(0.2
        generator.seed(2341);//set the seed

        double mean=0;
        int nSample = 10;

        for(int i=0;i<nSample;i++){
                double r = distribution(generator);
                cout<<r<<endl;
                mean = mean+r;
        }
        mean = mean/nSample;
        cout<<endl<<"Mean of the sample: "<<mean;
        return 0;
}
```

3 Working With CSV Files

Comma-Separated Value (CSV) files are widely used to store tabular data. To solve realistic problems in simulation and other scientific studies, you must be able to read and write CSV files. In this study, we will use CSV files to save simulation output. The data saved in the CSV file can be later opened in data visualizing tools like Microsoft Excel, Matlab, Python or R.

I am listing a C++ program that saves some output in the CSV format. You can use this program as an template in your work. This program calculates the values of the function $y = e^{-x/10}$ in the interval $[0, 9]$ and saves the pairs (x, y) in a CSV file.

```
#include <iostream>
#include <cmath>
#include <fstream>

using namespace std;


int main(){
        ofstream fout("curve.csv");
        fout<<"x-value,y-value\n";
        for(int x=0; x<10; x++){
                double y = exp(-x/10.0);
                fout<<x<<","<<y<<"\n";
        }
        fout.close();
```

```
        return 0;
}
```

Following C++ program reads the CSV file generated by the above program and displays its content in the screen.

```
#include <iostream>
#include <cmath>
#include <fstream>

using namespace std;

int main(){
        ifstream fin("curve.csv");
        int i;
        double x;
        char ch;
        string header;
        fin>>header;
        cout<<header<<endl;
        while(fin){
                fin>>i;
                fin>>ch; //to read the comma
                fin>>x;
                cout<<i<<", "<<x<<endl;
        }
        fin.close();
        return 0;
}
```

4 Problems

1. Implement the linear congruential random number generator and answer the following questions.
   (a) Take different values of the multiplier a, increment c, modulus m, and identify the period of the generator. Tabulate your results.
   (b) How would you ensure that the generator has full period? Demonstrate by selecting appropriate values of a, c, and m.
   (c) How would you generate random numbers in the range $[0, 1]$?
2. Answer graphically whether your random numbers are uniform or not. For this, generate 10,000 (or more if you wish) random numbers and save it in a CSV file. Open the file using Excel and observe the histogram.

3. For what combination of a, m and c will your program produce nearly uniformly distributed random numbers? Find out at least two combinations and justify your results graphically.

4. Answer the following questions making necessary changes in the example of exponential distribution.

   (a) What are the mean and variance of 10,000 exponential random number generated using the C++ library? Taking rate $\lambda = 0.5$, how much do the sample mean and variance differ with the actual?

   (b) Generate 10,000 random numbers, save them in a CSV file, plot the histogram, and describe the shape and other noticeable features in the plot.

5 Programming Challenge – Testing the Quality of a Random Number Generator[1]

In the above given problems 2 and 3, you examined the uniformity of random numbers graphically. Graphical examination is not a good idea. There are formal statistical tests for testing whether a sequence of numbers is uniformly distributed or not. One such test is the chi-squared test.

The chi-squared test is done in the following way. Let U_i denote the i'th random number in your output. Make necessary changes in your program so that U_i's have values in $[0, 1]$. Divide the interval $[0, 1]$ in k equal sized subintervals (we usually take $k \geq 100$). Now count the random numbers U_i that fall in each of the equally spaced subinterval. Let f_j denote the number of the U_i's that are in the j'th subinterval. The chi-squared statistic is defined as

$$\chi^2 = \frac{k}{n} \sum_{j=1}^{k} \left(f_j - \frac{n}{k} \right)^2$$

You need to compute χ^2 for the random numbers you generated. Then we can test the following null hypothesis.

H_0: Given set of numbers are uniformly distributed in the interval $[0, 1]$.

For testing the null hypothesis, select the level of significance α for testing the null hypothesis. Then identify, $\chi^2_{k-1,1-\alpha}$ the critical point of the chi-square

[1] In my lab sheets, you will find a section for 'Programming Challenge'. These problems are not mandatory, but they are necessary to make you a competent engineer. You are highly encouraged to complete them. Students able to complete challenging problems will be rewarded appropriately (may be you will be given a chocolate with a big applause).

distribution with $k-1$ degree of freedom. We can approximate the critical point as

$$\chi^2_{k-1,1-\alpha} \approx (k-1)\left(1 - \frac{2}{9(k-1)} + z_{1-\alpha}\sqrt{\frac{2}{9(k-1)}}\right)$$

where $z_{1=\alpha}$ is the upper $1-\alpha$ critical point of the standard normal distribution. If you want 95% confidence (i.e. $\alpha = 0.05$), take $z_{0.95} = 1.959$. If you want 90% confidence (i.e. $\alpha = 0.9$), take $z_{0.9} = 1.645$.

So, the overall procedure is

1. Generate a large number of random numbers (≥ 10000) denoted as U_i
2. Divide the interval $[0, 1]$ into k subintervals, $k \geq 100$
3. For each subinterval j, calculate f_j, the number of random numbers in the considered subinterval
4. Calculate χ^2 and compare with the critical value:
   (a) If χ^2 is greater than the critical value, then reject the null hypothesis H_0. This means than the numbers cannot be considered uniformly distributed in the interval $[0, 1]$
   (b) If χ^2 is less or equal to the critical value, then do not reject the null hypothesis. This means than we can consider the numbers are uniformly distributed up to the specified degree of confidence.

Programming Task: (a) Write a program to test whether a given set of numbers is uniformly distributed in the interval $[0, 1]$. Your program should read the numbers from a file. (b) Using your program, test the uniformity of the random numbers you implemented in problem 1, and the random number generator *rand*() provided in the standard C library "stdlib.h"