Coursework 3: Pseudo-Random Numbers and Monte Carlo Methods

The aim of this project is to explore "random" number generation and its use in simulation. It is, of course, impossible to generate a truly random sequence of numbers using a computer. Instead, the idea is to produce very long (ultimately periodic) sequences of integers, $\{x_n\}$, that appear random. The numbers in such a sequence are known as pseudo-random numbers. If the sequence is sufficiently long and we start at a "random" initial position then the numbers generated by moving along the sequence should be suitable for use in the place of true random numbers. How can we tell whether a random number generator is "good"? The answer depends partly on the application. If we are simulating the behaviour of an asset over time and the period of the pseudo-random sequence is too short, the periodicity will be reflected in the simulated behaviour of the asset, biasing the results.

Pseudo-random number generation is a complex topic and there are many excellent sources of information on the web and in textbooks, *e.g.*

Knuth, D. *The Art of Computer Programming, Vol 2: Seminumerical Algorithms.*(1998).

`http://en.wikipedia.org/wiki/Pseudorandom_number_generator`
`http://www.mathcom.com/corpdir/techinfo.mdir/scifaq/q210.html#q210.1`

We shall consider a number of different Random Number Generators (RNGs) and compare the results produced in two different applications:

(A) Calculating the area of a circle using Monte Carlo Integration,

(B) Calculating the long-time behaviour of an asset using Monte Carlo simulation.

The project is divided into four sections. The second section depends upon the first and the third and fourth sections use the functions defined in the second section. Don't worry if you cannot implement all the different random number generators. It is more important that you can use one RNG to perform the simulations.

- Marks will be awarded for the structure and clarity of your code.
- You should submit your answers to all questions and hard-copies of all your C++ codes by 4:00pm on Monday 15th December 2008.
- You should be prepared to demonstrate working versions of your codes in the examples class on Monday 15th December 2008.

Random Number Generators (20 marks)

We shall consider three different random number generators

(a) The "default" C++ `rand()` function.

(b) The IBM `RANDU` generator, for which the sequence is described by

$$x_{n+1} = (65539 x_n) \mod 2^{31},$$

where mod a indicates the integer remainder after division by a. In C++, $a \mod b$ can be computed by the command `a%b`.

For example,

$$\begin{aligned} x_0 &= 1. \\ x_1 &= (65539 \times 1) \mod 2^{31} \\ &= 65539. \\ x_2 &= (65539 \times 65539) \mod 2^{31} \\ &= 4295360521 \mod 2^{31} \\ &= 393225. \\ x_3 &= \ldots \end{aligned}$$

(c) A multiply with carry generator

$$x_{n+1} = (36969 x_n + c_n) \mod 2^{16},$$

$$c_{n+1} = \left\lfloor (36969 x_n + c_n)/2^{16} \right\rfloor,$$

where $\lfloor x \rfloor$ denotes "the integer part of" x. The integer part of a number can be obtained in C++ by casting to an integer or by using the `floor()` function, *i.e.* $\lfloor x \rfloor$, is computed by `static_cast<int>(x)` or `floor(x)`.

For example,

$$\begin{array}{rcc rcc} x_0 &=& 1 & c_0 &=& 1 \\ x_1 &=& (36969 \times 1 + 1) \mod 2^{16} & c_1 &=& \lfloor (36969 \times 1 + 1)/2^{16} \rfloor = \lfloor 0.564 \rfloor \\ &=& 36970 & &=& 0 \\ x_2 &=& (36969 \times 36970 + 0) \mod 2^{16} & c_2 &=& \lfloor (36969 \times 36970 + 0)/2^{16} \rfloor \\ &=& (1366743930) \mod 2^{16} & &=& \lfloor 1366743930/2^{16} \rfloor = \lfloor 20854.857 \rfloor \\ &=& 56186 & &=& 20854 \\ x_3 &=& \ldots & c_3 &=& \ldots \end{array}$$

Each generator requires initial (seed) values, x_0, which are usually obtained from a call to the system time, see Question 2 below.

Questions:

1. What is maximum integer that can be generated by the three RNGs (a)–(c)? [3 marks]

2. Write a C++ program that implements the three RNGs (a)–(c) as three different C++ classes. Each class should inherit from the base class `RNG` given below:

```
#include <time.h>
#include <cstdlib>
#include <cmath>
class RNG
 {
  protected:
  //Maximum (positive) integer in the pseudo-random sequence
  unsigned long Max;

  public:
  //Pure virtual function that returns the next integer in the
  //pseudo-random sequence. Note the use of the long data type to
  //represent (at least) 32-bit integers (Range 0,(2^32)-1).
  virtual unsigned long X()=0;
 };
```

   A sample implementation for the "default" C++ `rand()` function is given below

```
//''Thin'' wrapper class to the built-in C rand() function
class C_Rand : public RNG
 {
  public:
  //Constructor
  C_Rand()
   {
    //Set the maximum value from a built-in variable.
    Max = RAND_MAX;
    //Set an initial seed from the system time
    srand(static_cast<unsigned>(time(NULL)));
   }

  //Return the next integer by calling the built-in rand() function.
  unsigned long X() {return rand();}
 };
```

   [14 marks]

3. Print 100 random numbers from each of the three RNGs. Do the random numbers all lie within the ranges reported in Question 1? [3 marks]

Probability Distributions and Interfacing to Excel (30 marks)

We can now generate random integers in the range $[0, \text{Max}]$, but for practical applications, we nearly always require samples from uniform or normal distributions. In this section, we shall transform our pseudo-random sequences into samples from probability distributions and write interface functions so that we can call these samples from Excel.

Questions:

4. Add a function to the **base** class `RNG` that transforms the numbers in the pseudo-random sequence into samples from a uniform distribution between 0 and 1. [3 marks]

5. The Box-Muller method can be used to generate random samples from a standard normal distribution by using two uniform distributions:

   If u_1 and u_2 are two independent samples from a uniform distribution on $(0, 1)$, then

   $$n_1 = \sqrt{-2\ln u_1}\cos 2\pi u_2 \quad \text{and} \quad n_2 = \sqrt{-2\ln u_1}\sin 2\pi u_2,$$

   are two independent normal variables.

   Add another member function to the class `RNG` that implements the Box-Muller method by calling your newly-written function that samples from a uniform distribution. You should only return one of the two possible normal samples. Make sure that you do not call the logarithm function when u_1 is nearly zero. Explain clearly what action you take when u_1 is nearly zero and the possible consequences.

   [5 marks]

6. By using cut and paste, create a C++ DLL that contains your RNG classes, but that does not contain the `main()` function. [2 marks]

7. Add a global "wrapper" function to your DLL

   ```
   double __stdcall normal_sample(short &i);
   ```

   that could be used in Excel to call the `normal()` member function of our different RNGs. The argument of the function should be a single (16-bit) integer (the `short` data type) that is used to determine which random number generator to call. For example,

   ```
   double __stdcall normal_sample(short &i)
   {
     //Create a single instance of our C_Rand class
     //The use of the static keyword means that the state of the
     //class will be saved between function calls
     static C_Rand rng1;
     //Create static instances of the other RNGs here

     switch(i)
   ```

```
    {
     case 1:
     //Return the random number from the first RNG
     return rng1.normal();
     //End of case 1
     break;

     //Fill in the other cases in a similar way here

     default:
     //Indicate an error by returning a large number
     //This is not a brilliant idea; we should do proper
     //error handling and throw an exception, but that’s
     //not necessary for this project.
     return 1000.0;
    }
  }
```

The VBA data type that corresponds to a C++ `short` is, confusingly, `Integer` and the VBA prototype for our function is

```
Declare Function normal_sample _
Lib "C:/Path to DLL/my.dll" (arg As Integer) As Double
```

A call to `=normal_sample(1)` in Excel should return a sample from a normal distribution using the underlying C++ `rand()` RNG.

[5 marks]

8. Check your functions by generating 1000 normal samples from each different RNG **within** Excel and plotting the results as histograms. You should use the Histogram utility in the Data Analysis section of the Tool menu and you should produce one histogram for each different RNG. [5 marks]

9. In addition implement the Box-Muller method in Excel directly using the built-in `RAND()` function — a function that returns a number from a uniform distribution between 0 and 1. Do you notice any difference in the speed of calculation compared to the C++ RNGS?

   [5 marks]

10. How might you check that your distribution is normal? [5 marks]

Monte Carlo Integration (15 marks)

Suppose that we wish to calculate the area of a circle with unit radius ($A = \pi$). Consider a box that contains the circle, *e.g.* $(x, y) \in ([-1, 1], [-1, 1])$. The idea underlying Monte Carlo integration is that we can approximate the area by generating random points in the box, *i.e.* choosing x and y coordinates from two independent *uniform* distributions. We count the number of points that lie within in the circle and our approximation to the area is then

$$A \approx \text{Area of Box}(= 4) \times \frac{\text{Number of points within circle}}{\text{Total number of points}}.$$

If we take a sufficiently large number of points, we should get an accurate estimate for the area.

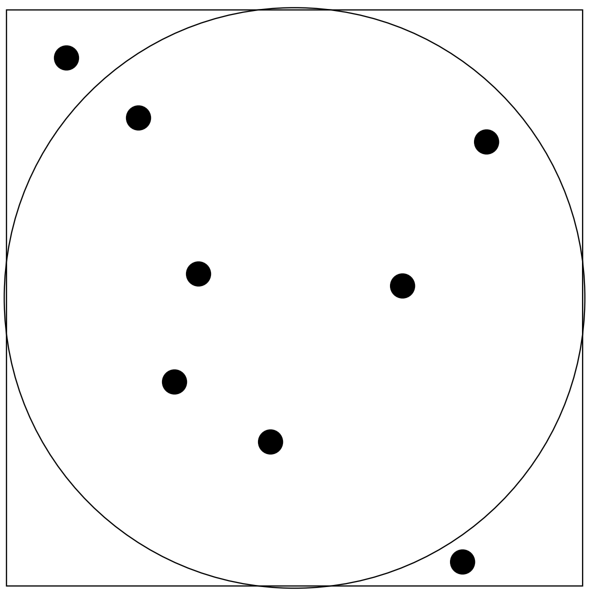

Figure 1: An illustration of Monte Carlo Integration. The points are uniformly distributed in the square $(x, y) \in ([-1, 1], [-1, 1])$. Six of the eight points lie within the circle and therefore our approximation to the area of the circle is $4 \times \frac{6}{8} = 3$.

Questions:

11. Write a C++ function that uses Monte Carlo integration to calculate the area of the unit circle using any (or all) of your RNGs. You need not call this function from Excel. You may, instead, write a standard C++ console application that contains your classes and the integration function. [10 marks]

12. Determine the approximate number of samples required to calculate the value of π to two decimal places. [5 marks]

Monte Carlo Simulation (25 marks)

We consider the simple stochastic simulation (arithmetic Brownian motion) of the price of an asset, $x(t)$, over time

$$x(t + \Delta t) = x(t) + \mu\Delta t + \sigma\sqrt{\Delta t}\mathcal{N},$$

where Δt is our chosen time increment, μ and σ are constants and $\mathcal{N}$ is a sample from a standard normal distribution.

Questions:

13. Inside the DLL write a C++ function that returns the final asset price at time $t = T$ by simulating a number of steps of the stochastic process:

    ```
    double __stdcall
    monte_carlo(double &x_init, double &T, double &delta_t,
                double &mu, double &sigma)
    ```

    The arguments to the function should be the initial price of the asset, the final time, T, the size of the time increment, Δt, and the constants μ and σ. You may choose to use any one of the three Random Number Generators, or, if you wish, specify an extra `short` argument that selects a particular RNG. [10 marks]

14. Use your function from within Excel to calculate the expected price of an asset at $T = 10$. The asset has an initial value $x(0) = 10$, with $\mu = 1$ and $\sigma = 0.001$. Compute results for the cases $\Delta t = 0.001, 0.01, 0.1, 0.5$, and 1.0. What do you notice about the results from different simulations as you vary Δt? [10 marks]

15. What is the price of the asset at $T = 10$ when $\sigma = 0$? [2 marks]

16. What happens to the expected price as σ increases? [3 marks]