University of Cape Town - Department of Physics
Honours Computational Physics

TOPIC 1 - BASIC MONTE CARLO : RANDOM NUMBER GENERATORS

This worksheet accompanies the EJS simulation `BasicMC_No1_RandomGeneratorTest.jar`

Computers generate 'pseudo-random' numbers by some deterministic process. One example is the linear congruential generator that generates a sequence of pseudo-random numbers r_1, r_2, ... over the interval $[0, M-1]$ according to:

$$r_{i+1} = (ar_i + c) \mod M,$$

with r_1 the seed (often supplied by the user) and a, c and M constants. Division of r_i by M then returns a result x_i in the interval $[0, 1)$. In addition, most programming languages have built-in functions that return pseudo-random numbers, generated by more elaborate means, in the interval $[0, 1)$ (e.g. `Math.random()` in Java).

Before using a random number generator, one should test its quality. Suppose that we have a sequence $x_1, x_2, \ldots, x_N$ thought to be random and uniformly distributed over the interval [0,1). The simplest test is simply to look for regularity in a scatter plot of (x_{2i}, x_{2i+1}). One can also easily calculate the k^{th} moment $\langle x^k \rangle$ of the sequence:

$$\langle x^k \rangle \equiv \frac{1}{N} \sum_{i=1}^{N} x_i^k,$$

as well as the near-neighbour correlations in the sequence:

$$C(l) \equiv \frac{1}{N-l} \sum_{i=1}^{N-l} x_i x_{i+l} \quad l = 1, 2, 3, \ldots.$$

Task 1: Assuming a uniform and random distribution for x over $[0, 1)$, show that:

$$\langle x^k \rangle \simeq \frac{1}{k+1} \pm \frac{1}{\sqrt{N}} \sqrt{\frac{1}{2k+1} - \left(\frac{1}{k+1}\right)^2}.$$

Task 2: Assuming a uniform and random distribution for x over $[0, 1)$, show that:

$$C(l) \simeq \frac{1}{4} \pm \frac{1}{\sqrt{N-l}} \sqrt{\frac{1}{24}}.$$

Questions:

1. Consider the linear congruential method using the unwise choice of $(a, c, M, r_1) = (57, 1, 256, 10)$.
   (a) Determine the period of the sequence.
   (b) Consider the scatter plot of successive pairs as well as the moments and near-neighbour correlations. Do these indicate a poor generator?
2. Test the randomness and uniformity of Java's built-in random number generator by focusing on the k^{th} moments for $k = 1, 3, 7$ and $N = 100, 10\,000, 100\,000$. Consider also the near-neighbour correlations with $l = 1, 5, 10$ and $N = 100, 10\,000, 100\,000$. Repeat this for the Mersenne Twister generator.