Exercise 3: Random Numbers and Monte Carlo

Drew Silcock
Physics Department, University of Bristol
(Dated: March 31, 2014)

I. INTRODUCTION

This exercise is split into two parts. The former half deals with generating pseudo-random number datasets that follow particular distributions. This is implemented analytically for a sinusoidal distribution, and for a general distribution using the reject-accept method. The latter half implements a Monte Carlo simulation of an unstable nucleus decaying and emitting a γ-ray, taking advantage of the analytic sinusoidal method used in the first half.

The focus of this exercise is in random number generation, in particular with reference to Monte Carlo techniques. Monte Carlo techniques involve the generation of random or pseudo-random numbers over a certain domain with a given Probability Density Function (PDF), the classic example being calculating pi by calculating what ratio of evenly distributed random numbers fall in the area of a circle to a square[1]. The general random number distributor implemented in the first part of this exercise has the purpose of providing the random number distribution that follows the desired PDF. The latter part applies the Monte Carlo technique as specified here to model the decay of an unstable nucleus travelling towards a detector. The decay time (and therefore position) and γ-ray decay angles are modelled as randomly generated numbers and the resulting detector hits are calculated trigonometrically.

II. RANDOM NUMBERS IN A DISTRIBUTION

A. Pseudo-Random Number Generators (PRNGs)

In practice, PRNGs are used instead of real hardware RNGs. Sawilowsky gave the following characteristics of a correct and useful PRNG, for use in a Monte Carlo simulation[2]:

1. the PRNG has long period before the random values begin to repeat themselves, and
2. the PRNG produces values that pass tests for randomness.

For the purposes of this exercise, NumPy's `numpy.random` functions were used, which use the Mersenne Twister (MT) algorithm as developed by Matsumoto and Nishimura in 1998[3]. These are so-called because their period of repetition is a Mersenne prime; specifically NumPy uses the MT19937 version[4], which has period $2^{19937} - 1 \approx 4.3 \times 10^{6001}$[3]. Hence the period is sufficiently large as to be effectively non-periodic and Sawilowsky's first condition is satisfied. MT also passes the standard empirical tests of the randomness of a PRNG, namely the DIEHARD and TestU01 tests[5, 6]. The MT algorithm therefore satisfies all of Sawilowsky's requirements for a PRNG.

Throughout this exercise, the seed is set by default by accessing the system's `/dev/urandom` on *nix and `CryptGenRandom` on Windows NT, thereby ensuring that values are not repeated on each run.

B. Analytic Sinusoidal Distribution

Two methods are used to produce the desired random number distributions. The first produces a sinusoidal random number distribution analytically by translating evenly distributed random numbers to sinusoidally distributed ones. This is done as follows:

Let the desired distribution, in this case a sinusoid, be $P'(x')$. The problem then is to convert between an even distribution, $P(x)$, and $P'(x')$. If the assumption is made that $P(x)$ and $P'(x')$ are properly normalised, then the cumulative distribution must be the same, so for generated value x_{gen} corresponding to required value x'_{req}:

$$\int_{x_0}^{x_{\text{gen}}} P(x)\,\mathrm{d}x = \int_{x'_0}^{x'_{\text{req}}} P'(x')\,\mathrm{d}x'. \tag{1}$$

Letting $x_0 = 0$, then:

$$\int_{0}^{x_{\text{gen}}} P(x)\,\mathrm{d}x = x_{\text{gen}}. \tag{2}$$

Defining $Q(x'_{\text{req}}) = \int_{x_0}^{x_{\text{gen}}} P(x)\,\mathrm{d}x$, Equation 1 becomes:

$$Q(x'_{\text{req}}) = x_{\text{gen}}, \tag{3}$$

, so the solution for x'_{req} is found by inverting $Q(x'_{\text{req}})$ to obtain:

$$x'_{\text{req}} = Q^{-1}(x_{\text{gen}}). \tag{4}$$

This is applied to a sinusoid, $P'(x') = \sin(x')$ for $0 < x' < \pi$, to obtain:

$$\begin{aligned} Q(x'_{\text{req}}) &= \int_0^{x'_{\text{req}}} \sin(x')\,\mathrm{d}x' \\ &= \Big[\cos(x')\Big]_{x'=0}^{x'=x'_{\text{req}}} \\ &= \cos(x'_{\text{req}}) - 1 \end{aligned} \tag{5}$$

$$\implies x'_{\text{req}} = Q^{-1}(x_{\text{gen}}) = \arccos(1 - x_{\text{gen}}), \quad (6)$$

where $0 < x < 2$.

C. Random Number Distributor

III. SIMULATIONS

[1] J. H. Mathews and K. K. Fink, *Numerical Methods Using Matlab (4th Edition)* (Pearson, 2004), 4th ed., ISBN 0130652482, URL http://mathfaculty.fullerton.edu/mathews//n2003/montecarlopimod.html.

[2] S. S. Sawilowsky, Journal of Modern Applied Statistical Methods **2**, 218 (2003), URL http://digitalcommons.wayne.edu/coe_tbf/21.

[3] M. Matsumoto and T. Nishimura, ACM Trans. Model. Comput. Simul. **8**, 3 (1998), ISSN 1049-3301, URL http://doi.acm.org/10.1145/272991.272995.

[4] P. S. Foundation, Tech. Rep. (2014), URL https://docs.python.org/2/library/random.html.

[5] P. Ecuyer and A. Owen, *Monte Carlo and Quasi-Monte Carlo Methods 2008*, Mathematics and Statistics (Springer, 2010), ISBN 9783642041075, URL http://books.google.co.uk/books?id=5Y9cnmtLGQYC.

[6] A. Jagannatam, *Mersenne Twister - A Pseudo Random Number Generator and its Variants* (2009).