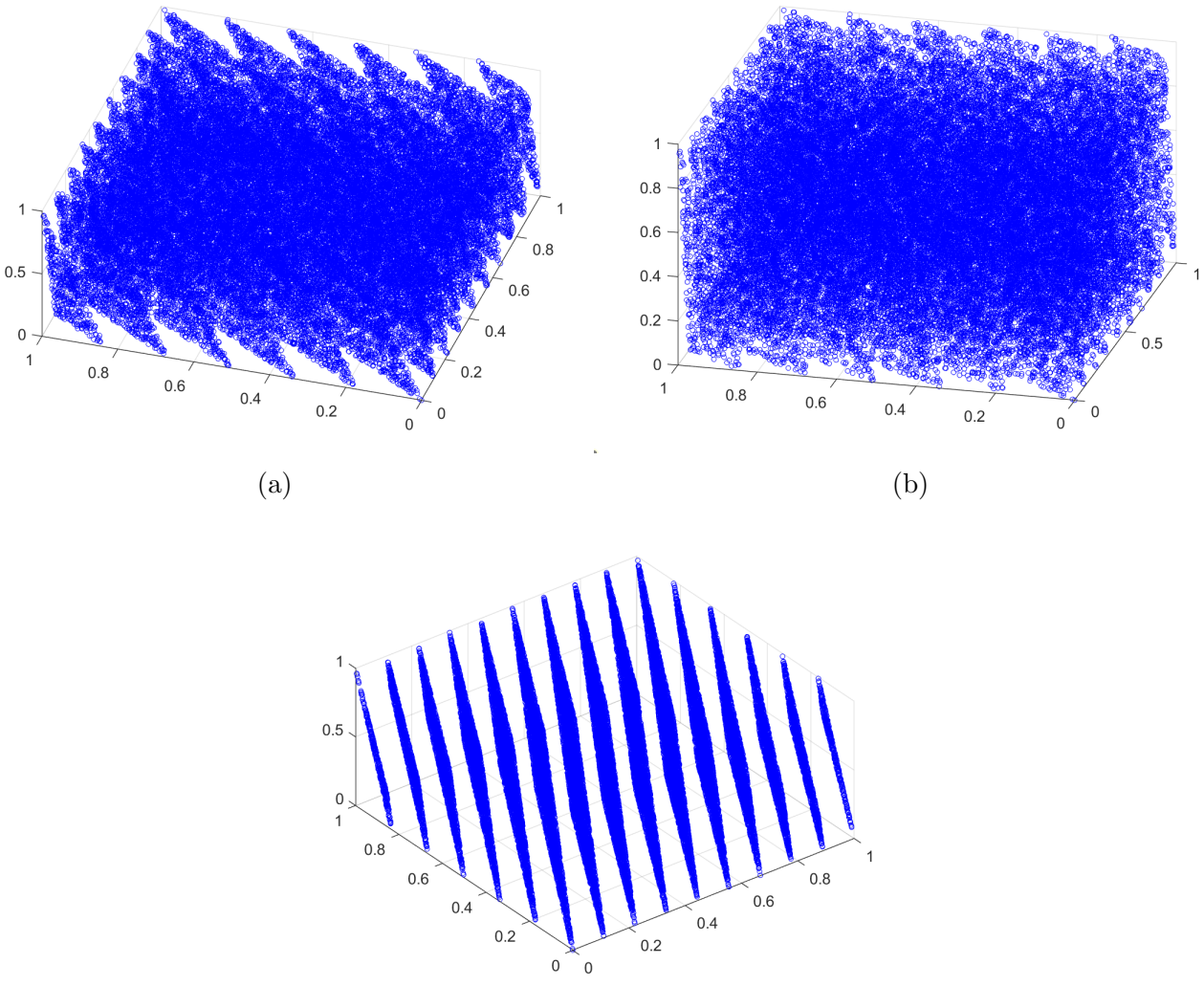
AM3911G: Modelling and Simulation **(Due: 02/24/2020)**

Assignment #2

Instructor: Allan MacIsaac *Name:* Peter Akioyamen, *StudentID:* 250949002

Problem 1 – Points



(a) (b)

(c)

Figure 1: Views of the planes created by RANDU using IBM's parameters

Problem 2 – Points

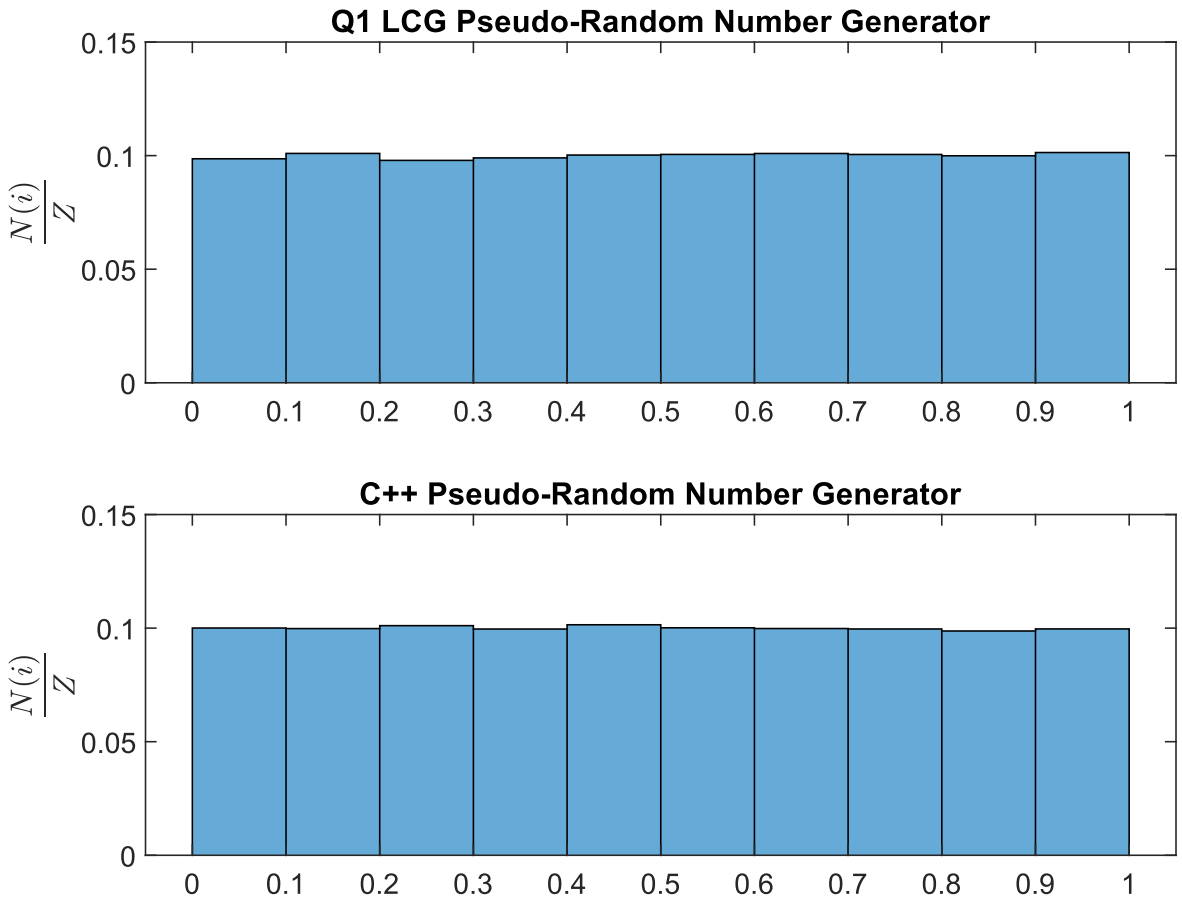
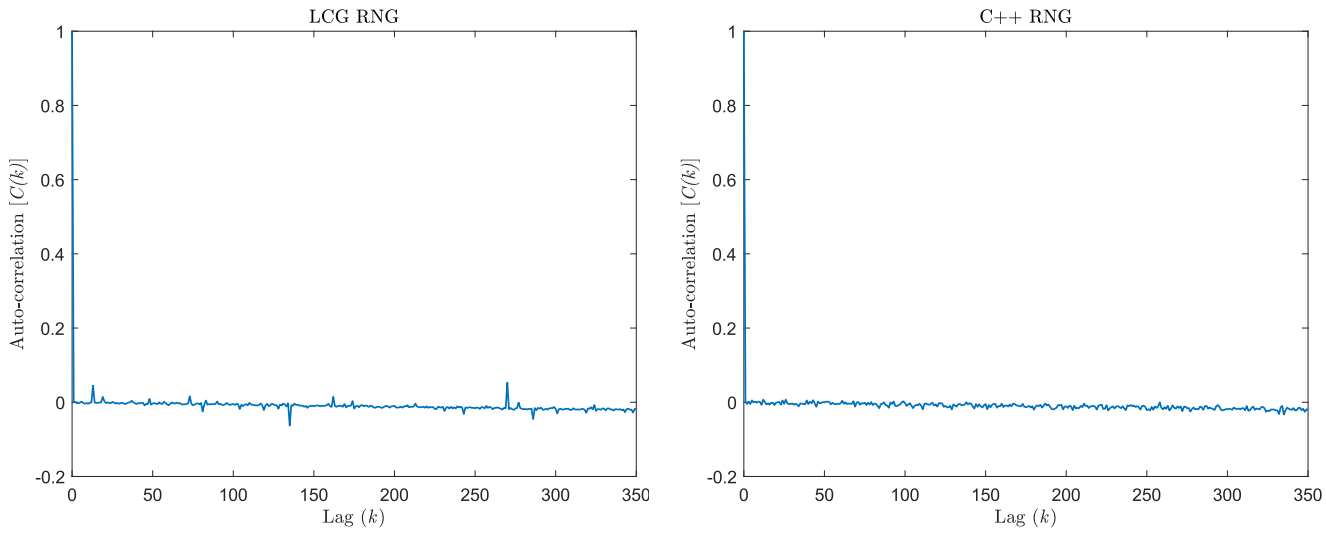


Figure 2: The distribution of pseudo-random numbers generated

10^5 numbers were generated by the LCG random number generator from question one (with identical parameters as IBM) and the native C++ uniform random number generator, respectively. The generated pseudo-random numbers were placed in one of 10 bins with $\Delta h = \frac{1}{10} = 0.1$ for a uniform distribution on $[0, 1]$. Both seem to produce sequences of numbers whose distribution appears uniform – since uniformity is a desired property of both of these random number generators, this result intuitively makes sense. Based solely on inspection, there seem to be no major differences in the distribution of random numbers generated by either generator. Though, noticeable differences could be present if the number of bins is increased.

Problem 3 – Points



(a) Autocorrelation of LCG RNG (b) Autocorrelation of C++ RNG

Figure 3: Autocorrelation plotted against the lag value used for each RNG

For both random number generators the autocorrelation seems to exhibit subtle fluctuations above and below zero. Interestingly, the LCG random number generator whose autocorrelation can be seen in Figure: 3a displays more drastic fluctuations in $C(k)$ in comparison to the C++ RNG (Figure: 3b). Both random number generators seem to have a slight downtrend in $C(k)$ as the number of lags is increased from 0 to 350, which was unexpected. Since it is known that the numbers generated by these two RNGs are not random, it is understandable that $C(k) \neq 0$ for $k = 0, \ldots, 350$.

Problem 4 – Points

(a) Below is a code snippet of my nuclear decay Monte Carlo simulation, written in C++. *Note: Data was saved in .csv and plotted in MATLAB.*

Listing 1: Simulation of Nuclear Decay using Monte Carlo.

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

int main() {
    // Define monte carlo parameters
    const int ntrials = 20;
    const int tmax = 200;
    const int N_0 = 200;
    const double p = 0.01;
    int t, unstable_nuclei;

    // Define array to store monte carlo ensemble average
    double nuclei[tmax + 1] = { 0.0 };

    // Define data files
    fstream sim1_file, simN_file;
    sim1_file.open("sim1_4.csv", ios::out | ios::app);
    simN_file.open("simN_4.csv", ios::out | ios::app);


    // Define the system uniform random number generator
    default_random_engine generator;
    uniform_real_distribution<double> distribution(0.0, 1.0);

    // Simulate nuclear decay for ntrials (20)
    for (int iter = 0; iter < ntrials; iter++) {
        unstable_nuclei = N_0;
        nuclei[0] += unstable_nuclei;

        // Run computations at each time step
        for (t = 1; t <= tmax; t++) {
            // Compute probability of decay for each unstable nuclei
            for (int i = 0; i < unstable_nuclei; i++) {
                if (distribution(generator) <= p) {
                    unstable_nuclei -= 1;
                }
            }
```

```
40
41              // The number of unstable nuclei remaining at the ↩
                   beginning of the t+1 time step
42              nuclei[t] += unstable_nuclei;
43
44              // Save the nuclear decay data from the first simulation
45              if (iter == 0) {
46                  if (t == 1) {
47                      sim1_file << (t - 1) << "," << nuclei[t - 1] << "\↩
                           n";
48                  }
49                  sim1_file << t << "," << nuclei[t] << "\n";
50              }
51          }
52      }
53
54      // Compute ensemble average for monte carlo
55      // Save nuclear decay data from monte carlo
56      for (t = 0; t <= tmax; t++) {
57          nuclei[t] /= ntrials;
58          simN_file << t << "," << nuclei[t] << "\n";
59      }
60
61      // Close files
62      sim1_file.close();
63      simN_file.close();
64      return 0;
65  }
```

(b – c)

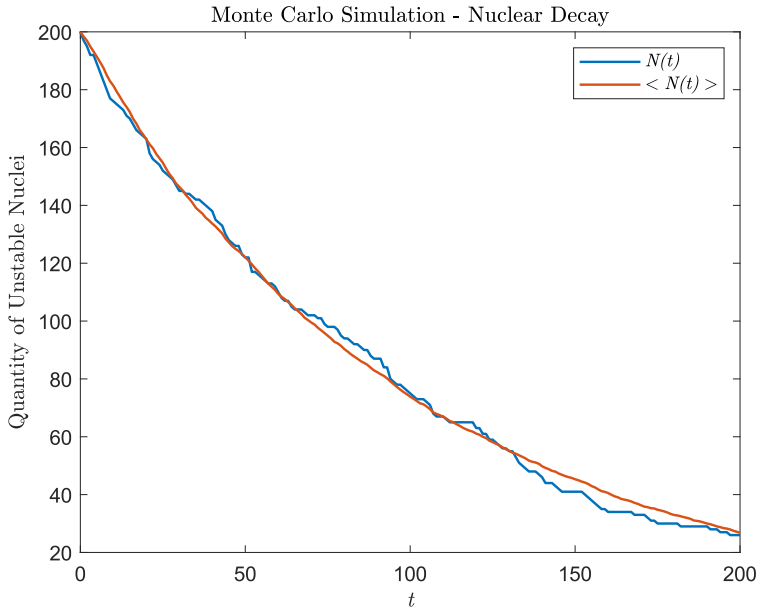


Figure 4: One simulation and ensemble average of nuclear decay Monte Carlo

(d)

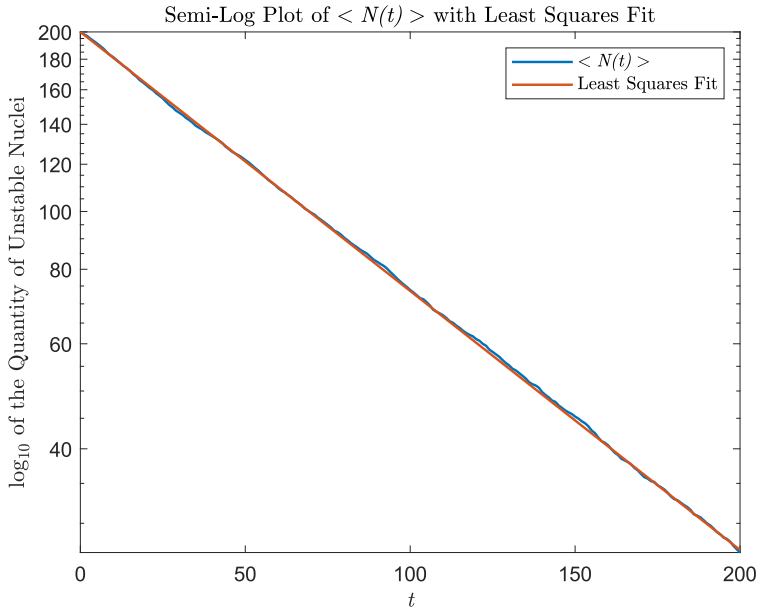


Figure 5: Semi-log plot of nuclear decay Monte Marlo simulation data

The value of λ which was computed for the exact solution using least squares was about 0.01.