Numerical Tools for Physical Scientists: Random numbers and Monte Carlo

Erik Spence

SciNet HPC Consortium

6 February 2014

Today's class

Today we will discuss the following topics:

- Random numbers, what are they?
- Random number tests.
- Examples of Monte Carlo simulations.
- Assignment 5.

Random numbers

What is a random number?

- A random number is a number drawn from a set of possible values, each of which is equally probable (for uniform distributions).
- The numbers must be statistically independent of each other, meaning that drawing one value doesn't change the probability of drawing it again.
- Which means that the sequence 9, 9, 9, 9, 9, 9 could be random.

What properties do we expect from a random number generator?

- We would like them from a given distribution (uniform, Gaussian).
- We would like them to be unpredictable.
- We would like them to be reproducible.
- We need them to be generated quickly.
- We need to have a long period (before they start repeating).

Physically-generated random numbers

One option for generating random numbers is to generate them using a physical process. These are called *true* random number generators (TRNGs), because they are not generated deterministically.

How might such a sequence be generated?

- Lava lamps.
- Radioactive decay.
- Various quantum processes.
- Atmospheric noise.
- Random computer hardware noise signals (thermals noise).

However, there are shortcomings to this approach:

- Generally slow, expensive, impossible to reproduce for debugging.
- It's hard to characterize the underlying distribution.

Software-generated random numbers

Generators of random numbers that use software are called pseudo-random numbers generators (PRNGs). This is because the numbers are not truly random, since they are generated deterministically, using an algorithm.

- These sequences of numbers based on some starting 'seed' (a number).
- The sequences seem random, but are reproducible using the same seed, since they are deterministic.
- The sequences can usually be generated very quickly.
- The sequences usually have a uniform distribution on [0,1). Given this, they can be used to create other distributions.

But how will we decide if it's random? To test for randomness our best bet is to perform statistical tests on the generated sequence.

Tests for randomness: correlation

One way to test the randomness of a sequence of numbers is to test for correlation within the sequence. Given the expectation value of the sequence:

$$E(x) = \frac{1}{N}\sum_{i=1}^{N} x_i$$

we want to avoid pairwise correlations, on average:

$$\epsilon(N,n) = \frac{1}{N}\sum_{i=1}^{N} x_i x_{i+n} - E(x)^2$$

The quantity $\epsilon \longrightarrow 0$ as $N \longrightarrow \infty$, with an error of

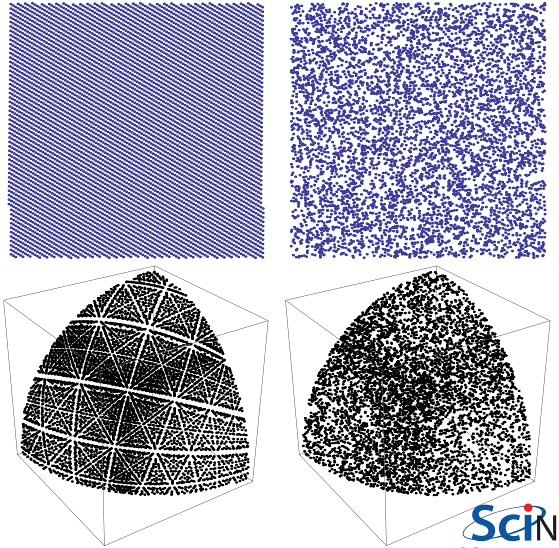
$$\epsilon(N,n) \sim \mathcal{O}(N^{-\frac{1}{2}}) \quad \forall n$$

Tests for randomness: graphical correlations

The sequence, $x_1, x_2, ...$, can be plotted as (x_i, x_j) in 2D, or as normalized unit vectors $\vec{x}/\left|\vec{x}\right|$ in 3D.

left: Bad Linear Congruential Generator.
right: Mersenne Twister.

Stolen from Katzgraber.



Tests for randomness: moments

A uniform distribution of pseudo-random numbers, in the interval [0,1], should have a kth moment of $1/(k+1)$. Thus, we can test the value of the function

$$\mu(N,k) = \left| \frac{1}{N} \sum_{i=1}^{N} x_i^k - \frac{1}{k+1} \right|$$

Again, the quantity $\mu \to 0$ as $N \to \infty$, with an error of

$$\epsilon(N,n) \sim \mathcal{O}(N^{-\frac{1}{2}}) \quad \forall k$$

Tests for randomness: other tests

Other tests that could be performed on a sequence to test for randomness:

- Spacings between random points should follow a Poisson integral if uniformly distributed.
- Examine sequences of 5 numbers. There are 120 ways to sort 5 numbers. The 120 ways should occur with equal probability.
- Parking circle test: randomly place unit circles in a 100 x 100 square. If the circle overlaps an existing one, try again. After 12,000 tries, the number of successfully "parked" circles should follow a certain normal distribution.
- Craps test: play 200,000 games of craps, counting the wins and number of throws per game. Each count follow a certain distribution.
- And many others. See, for example, the NIST test suite: `http://csrc.nist.gov/groups/ST/toolkit/rng`.

Existing PRNGs

Some good PRNGs:

- r1279 (good lagged-Fibonacci generator).
- Mersenne twister (mt19937).
- WELL generator (Well Equidistributed Long-period Linear, developed at U. Montréal).

Some not-good PRNGs:

- r250 (bad lagged-Fibonacci generator).
- Anything from Numerical Recipes - short periods, slow, ran0 and ran1 spectacularly fail statistical tests.
- Standard Unix generators, rand(), drand48() - not a disaster, but contain short periods, correlations.

Monte Carlo

Monte Carlo analyses are a collection of techniques whose unifying feature is the use of randomness. These analyses generally fall into one of three categories:

- Adding randomness to otherwise-deterministic dynamics, and studying how the dynamics are changed.
- Generating samples from a given probability distribution, $\boldsymbol{P}(\mathbf{x})$, usually a distribution that is complicated and can't be dealt with nicely in closed form.
- Estimating expectations of functions under this distribution, for example:

$$\boldsymbol{\Phi} = \langle \boldsymbol{\phi}(\mathbf{x}) \rangle = \int \boldsymbol{d}^{\boldsymbol{N}} \mathbf{x} \boldsymbol{P}(\mathbf{x}) \boldsymbol{\phi}(\mathbf{x})$$

Monte Carlo example: investment returns

A crude assessment of risk is routinely used in the financial industry: use Monte Carlo to assess the 'likelihood' of losing money.

Here's an algorithm:

- The S&P 500 has an average annualized return of 10.6% with a standard deviation of 18.1% (2/1994 - 1/2004).
- Assume a normal distribution of returns, with the characteristics described above. Randomly draw from this distribution to get your annual return.
- Start with $100.0. Calculate how much money you have after 10 years (meaning get your annual return 10 times from the distribution).
- Repeat 500 times.

What is the average return? What is the resulting distribution?

Monte Carlo example: investment returns

```
#ifndef MYRANDOM_H     // myrandom.h
#define MYRANDOM_H     // Uses the BOOST package: boost.org
#include <boost/random/linear_congruential.hpp>
#include <boost/random/mersenne_twister.hpp>
#include <boost/random/normal_distribution.hpp>
#include <boost/random/uniform_01.hpp>
#include <boost/random/variate_generator.hpp>

// Bad LCG: a = 106, c = 1283, m = 6075.
typedef boost::random::linear_congruential<int, 106, 1283, 6075> B_ENG;
typedef boost::mt19937 ENG; // The Mersenne Twister algorithm.

typedef boost::normal_distribution<double> N_DIST; // Gaussian distribution.
typedef boost::uniform_01<double> U_DIST; // Uniform distribution from 0 to 1.

// The random number generators themselves.
typedef boost::variate_generator<ENG, N_DIST> N_GEN;
typedef boost::variate_generator<ENG, U_DIST> U_GEN;
typedef boost::variate_generator<B_ENG, U_DIST> B_U_GEN;
#endif
```

Monte Carlo example: investment returns

```cpp
// MyReturns.cpp
#include <fstream>
#include "myrandom.h"

int main() {
  const int numtests = 500;
  const int numyears = 10;
  const float mean = 0.106;
  const float std = 0.181;

  float *results =
    new float[numtests];

  for(int i = 0; i < numtests; i++)
    results[i] = 100.0;

  ENG eng;      // Mersenne twister.
  N_DIST dist(mean, std);
  GEN gen(eng, dist);
```

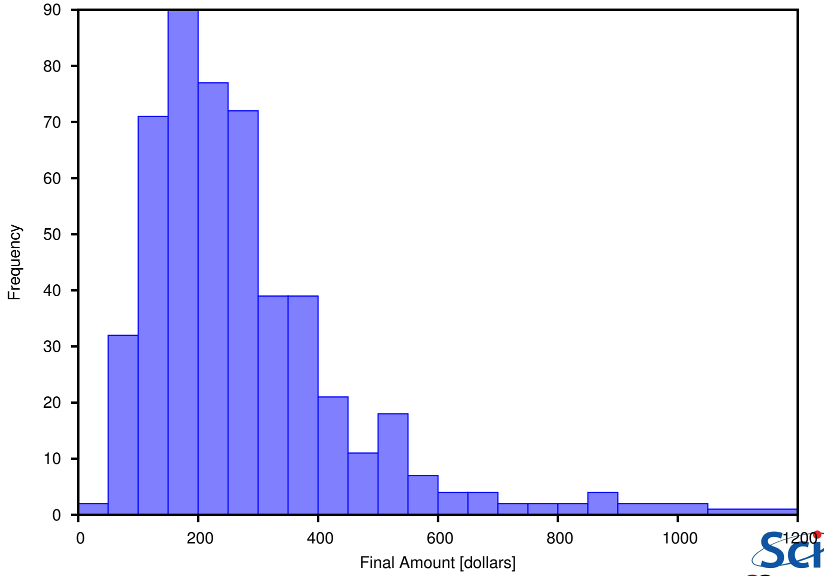
```cpp
  // Repeat for the 500 trials.
  for(int i = 0; i < numtests; i++) {
    // Calc the return for each year.
    for(int j = 0; j < numyears; j++)
      results[i] *= (1.0 + gen());
  }

  std::ofstream
    dataFile("returns.dat");

  for(int i = 0; i < numtests; i++)
    dataFile << results[i] <<
    std::endl;

  dataFile.close();
  delete [] results;
  return 0;
}
```

Monte Carlo example: investment returns



Monte Carlo example: traffic flow

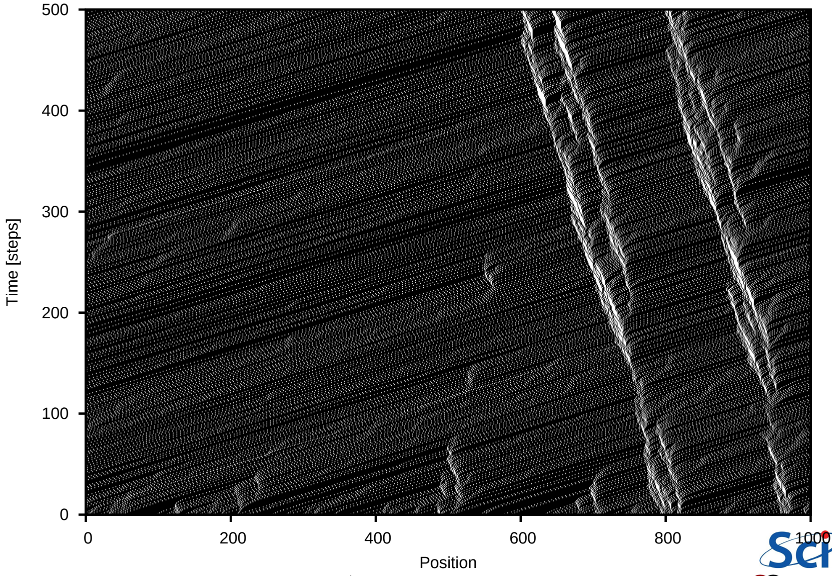
Nagel-Schreckenberg traffic is a 1D toy model used to generate traffic-like behaviour. At each time step in the model, the following rules are applied to each car in the simulation:

1. If the velocity is below $v_{\max}$, then increase v by 1 (try to speed up).
2. If the car in front of the given car is a distance d away, and $v \geq d$, then reduce v to $d - 1$ (don't want to hit the car).
3. Add randomness: if $v > 0$ then with probability p the car reduces its speed by 1.
4. The car moves ahead by v steps (on a circular track).

The four rules boil down to

1. $v \leftarrow \min(v + 1, v_{\max})$
2. $v \leftarrow \min(v, d - 1)$
3. $v \leftarrow \max(0, v - 1)$ with probability p
4. $x \leftarrow x + v$.

Monte Carlo example: traffic flow



numcars $= 200$, gridsize $= 1000$, $p = 1/3$, maxv $= 5$

Monte Carlo example: molecular dynamics

Consider a simple molecular dynamics model, which consists of a collection of molecules. For each timestep:

1. Randomly perturb the 2D position of a given molecule.
2. Calculate the new total energy of the system, $E = \sum_j A/(\mathbf{r}_i - \mathbf{r}_j)^2$, where A is a 'strength' parameter:
    1. If the energy of the system goes down, keep the new position.
    2. If the energy of the system goes up, keep the position if $r < \exp(-\Delta E/T)$, where r is a random number between 0 and 1, and T is the system temperature.
3. Repeat for all molecules.
4. Repeat for all timesteps.

Monte Carlo example: molecular dynamics

$T = 0.3$, $A = 5$, $N = 200$.

Assignment 5

Consider a random walk in 2D:

- Iterate the position of a walker, randomly moving $\pm$ 0.1 in the x and y directions at each iteration, starting from the origin.
- When the radius of the walker's position, with respect to the origin, exceeds 2, stop iterating.
- Record the angle of the walker's position, with respect to the x axis.
- Repeat for a total of 10,000 walkers.
- Plot a histogram of the angles, using whatever program you wish.
- Perform this operation twice: once using the bad linear congruential generator given above $(\boldsymbol{a = 106, c = 1283, m = 6075})$ and once with the Mersenne Twister algorithm.

Please submit:

- all source, header and make files for the new program.
- the output of 'git log' for your code development.
- the two resulting histograms.