

Random Numbers

Observations and Problems

We all know from our everyday experiences that nature is full of *chance* occurrences. There are things that happen, but which we are not able to predict. We often presume that there are causes for all physical events, but, for whatever the reasons, we may not be able to discern them. Some people ascribe theological significance to some chance events, while others may believe that there are definite causes, but that the inherent complexity of the processes keeps us from deducing them. Our **problem** in this module is to explore whether a completely deterministic computer program can give output that seems to include elements of **chance**.

Background

This module addresses the **problem** of how computers generate numbers that appear random and how we can determine how random they are. Other modules use these random numbers to simulate physical processes like radioactive decays and random walks. In the module on “Determining Areas by Stone Throwing”, we see how to use these random numbers to evaluate integrals.

This is a fascinating subject in that we give precise definitions to words that are commonly used, but possibly with somewhat different meanings than the mathematical ones. It’s a free world and so you may continue to use these words as you like, but please remember that when we use them it is within the context of their mathematical definitions.

For example, the concept of *determinism* refers to events that are caused by prior events. This is related to the principle of *causality* which states that a physical event has its origin or cause in some other action at an earlier time (birth precedes death). The opposite of determinism is *chance*, which implies an event in which there is an element of unpredictability. Trying to understand these concepts in a philosophical way, or to understand how a world described by deterministic physical laws can exhibit chance, is deep question that has fascinated some brilliant minds [Chance]. For our purposes, we can think of *chance* as a useful, though approximate, way to describe very complex systems in which there are too many degrees of freedom for us to predict what is going on. For example, we probably all agree that chance enters when we throw a die. Yet if we knew exactly what the velocity and position of the die was when it was thrown, as well as a complete knowledge of the die’s geometry and the nature of the surface upon which it lands and the air through which it travels, then we should be able to calculate the die’s behavior. This being too complicated and requiring more information than possible to attain, all we can say what the *probability* of coming up with any number is $1/6$ th, but exactly which number comes up is a matter of chance.

As you will see, we use the word *random* often in this module. It too has many definitions, and we use it in the *statistical* sense to mean the absence of *correlation*. Here *correlation* means a connection

between two objects. So when an event is *random*, we mean that it is not connected to a previous event, and thus that it is not determined by a previous event.

Shodor Tutorial on Random Number Generators

<http://www.shodor.org/interactivate/discussions/RandomNumberGenerato/>

Deterministic Randomness

Some people are attracted to computing by its deterministic nature; it's nice to have a place in one's life where nothing is left to chance. Barring the unlikely chance of a computer error, a computer program always produces the same output when it is fed the same input. Nevertheless, much technical computing and most electronic game playing use what are called *Monte Carlo* techniques that at their very core include elements of *chance*. These are calculations in which random numbers generated by the computer are used to *simulate* naturally random processes, such as thermal motion or radioactive decay. The inclusion of the element of chance makes the calculations much like experiments done on a computer (this is the way in which the computer *simulates* natural processes). Indeed, many scientific and engineering advances have come about from the ability of computers to solve previously intractable problems using Monte Carlo techniques.

Random Sequences (Theory)

We define a sequence r_1, r_2, r_3, \dots as *random* if there are no correlations or relations among the numbers. This does *not* mean that all the numbers in the sequence are equally likely to occur, but rather that even if we know all of the numbers in the series up to some point, there is no way of knowing with certainty what the next number will be.

If the numbers in some series all have the same likelihood to occur, then the sequence is said to be *distributed uniformly*, or to be a *uniform sequence*. Note that a random sequence of numbers can be uniform or not; even if some numbers are more likely to occur than others, if we cannot be sure what the next number will be, then the series is still random.

To illustrate, the sequence

(1) $1, 2, 3, 4, \dots$

appears to be uniform but does not appear to be random. All integers are present and equally likely (uniform), but knowing any one integer in the sequence permits you to predict the next one (nonrandom).

In contrast, the sequence

(2) $0.84, 1.03, 2.5, 1.3, 0.74, 1.0, 1.6, 0.52, 1.8, 0.42,$

appears to be random, but does not appear to be uniform since number close to 1 appear more likely than those close to 0 or 2. So even though we can say that it is more likely for the next number in the

sequence to be close to 1 than to 0, the sequence is still random because we cannot predict what it is going to be.

By nature of their construction, computers are deterministic and so cannot generate truly random numbers. Consequently, the so called “random” numbers generated by computers are not truly random, and if you look hard enough you can verify that they are correlated to each other. Although it may be quite a bit of work, if we know one random number in the sequence as well as the preceding numbers, it is always possible to determine the next one. For this reason, computers are said to generate *pseudorandom numbers* (yet with our incurable laziness we won't bother saying “pseudo “ all the time).

A primitive method for generating truly random numbers is to read in a table of numbers determined by naturally random processes such as radioactive decay, or to get input from some physical device measuring such processes.

Random-Number Generation (Algorithm)

Most computer languages like Python and programs like Excel have a built-in function that supplies a different random number each time the function is called. These routines may have names like *rand*, *rn*, *random*, *srand*, *erand*, *drand*, or *drand48*. The sequences of numbers so produced are pseudorandom sequences, although they are usually called just “random numbers”. Sometimes you must, or at least have the option, to specify what the first number in the sequence (the “seed”) is. Otherwise, the computer makes its own pick and may use something like the local time as the seed in order to produce different sequences each time the program is run. For those who are interested in, or required to, understand the algorithm used to produce random numbers, we describe it in the optional subsection below. Even if you do not work through that section, you should go to the section describing how to test a random number generator.

*The Linear Congruent Method**

The *linear congruent* method is the most common way of generating a pseudorandom sequence of numbers. We denote the i^{th} random number in a sequence as r_i , and assume that we know the seed r_0 . Let us say that we want the random numbers to be generated so that they are all greater than 0, but less than $M - 1$, where we know the value of M . We express this mathematically as

$$(3) \quad 0 \leq r_i \leq M - 1.$$

To generate the entire random sequence we need a formula that takes r_i , and uses it to calculate the next random number r_{i+1} . The linear congruent method multiplies r_i by a constant a , adds a different constant c to the product, and then divides the sum by the range constant M :

$$(4) \quad r_{i+1} = \text{remainder} \left(\frac{a r_i + c}{M} \right).$$

The unusual aspect of Equation (4) is the “remainder” function, which may not be familiar to you. Do you recall the first time you learned division back in grade school, and learned that sometimes when you divide two integers there results another integer with a “remainder”? For example, $12/4 = 3$, yet $13/4 = 3 + \text{Remainder } 1$. Well the remainder function in (4) means just the same thing: divide $a r_i + c$ by M , throw away the integer part, and keep just the remainder. Yes, that is right, throw away the most important part ! For example

$$(5) \quad \text{remainder} \left(\frac{13}{4} \right) = \text{remainder} \left(3 \frac{1}{4} \right) = 1, \quad \text{remainder} \left(\frac{98}{3} \right) = \text{remainder} \left(32 \frac{2}{3} \right) = 2.$$

The reason Equation (4) works as an algorithm for generating random numbers, is that when very large values of a and M are used, that remainder, being the least significant part of the quotient, is essentially the result of round-off error or contains some “garbage” that was laying around in the computer’s memory. Furthermore, since the sequence repeats after M numbers, large M keeps the sequence from repeating often (repetition implies nonrandom).

Scaling and Uniformity of Random Numbers (Not Optional)

It is always a good idea, and sort of fun, to test a random number generator before using it.

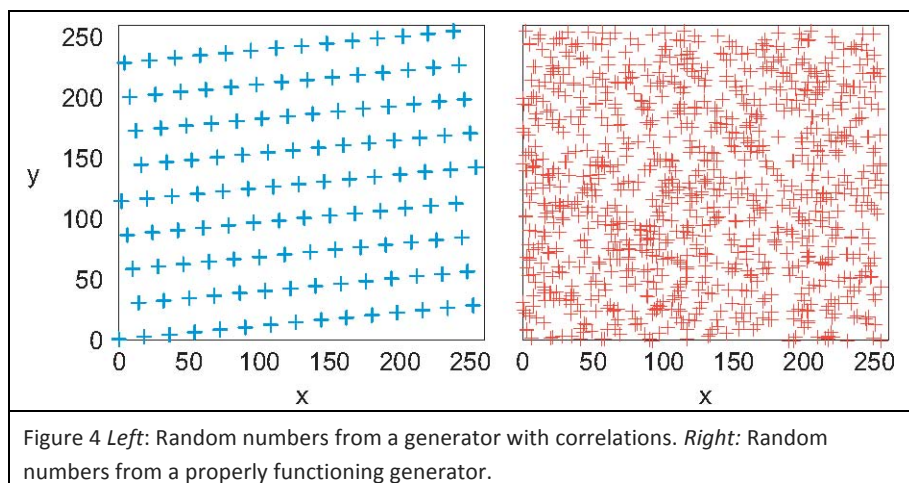
As seen from the above example, the linear congruent method generates random numbers in the range 0 -- M . If we want numbers in the more common range of 0 -- 1 , then we need only divide by the endpoint $M = 9$. By doing that, the sequence in Equations 5 -- 7 becomes

$$(10) \quad 0.333, 0.444, 0.889, 0.667, 0.778, 0.222, 0.000, 0.111, 0.555, 0.333$$

This is still a sequence of length 9, but is no longer a sequence of integers.

Uniform Distributions

Another concept, which is often confused with randomness, is that of *uniformity*. If the numbers in a sequence are generated with equal likelihood of being anywhere within the range of numbers $[0, 1]$ for Equation (10)], then those numbers are said to be distributed *evenly* or *uniformly*. For example, the sequence in equation (10) appears uniform, at least within the statistical variation that occurs when the sample size is small. Specifically, imagine making a histogram with bins, each of width 0.1. The sequence (10) will have one number in each bin, and so would result in a flat histogram. This all means that the sequence does look uniform.



On the right of Figure 3 you see $(x, y) = (r_i, r_{i+1})$ results from a random number generator that has made a “good” choice of internal parameters [$a = 25214903917$, $c = 11$, $M = 281,474,976,710,656$ in Equation (4)].

Make up your own plot and compare it to Figure 4 right. Be warned, your brain is very good at picking out patterns, and if you look at Figure 4 right long enough, you may well discern some patterns of points

clumped near each other or aligned in lines. Just as there variations in the uniformity of the distribution, a random distribution often shows some kind of clumping --- in contrast to a completely uniform “cloud” of numbers.

There are more sophisticated and more mathematical tests for randomness, but we will leave that for the references. We will mention however, that one of the best tests of a random number generator is how well computer simulations that use that generator are able to reproduce the nature of natural processes containing randomness (like spontaneous decay), or how well they can reproduce known mathematical results. The modules on *Random Walk* and *Spontaneous Decay* contain such simulations, and the module on *Stone Throwing* contains such a mathematical result.

Glossary References

[Chance] [Jacques Monod](#) (Nobel Prize 1965) essay "Chance and necessity". It is also asserted by [Werner Heisenberg](#) (Nobel Prize 1932), [Sir Arthur Eddington](#), [Max Born](#) (Nobel Prize 1954) and [Murray Gell-Mann](#) (Nobel Prize 1969). The physicist-chemist [Ilya Prigogine](#) (Nobel Prize 1977) argued for indeterminism in [complex](#) systems.

[CP] Landau, R.H., M.J. Paez and C.C. Bordeianu, (2008), *A Survey of Computational Physics*, Chapter 5, Princeton Univ. Press, Princeton.