Random Numbers

Learning objective

Science and Math content objective:

- Students will better understand the meaning of *chance* and *randomness* as they occur in nature.
- Students will learn that a sequence of numbers has properties such as *uniformity* and *randomness,* and that they can learn to recognize these properties.
- Students will learn to use some tools for assessing whether a distribution is uniform and whether it is random.
- In an optional section, students will explore an actual algorithm that generates pseudorandom numbers.

Science model/computation objectives:

- Students will learn that computers, being deterministic by their very nature, cannot produce truly random numbers.
- Students will understand that the distribution of random numbers produced by a computer are really *pseudorandom* distributions, and so not truly random.
- As part of computational thinking, students will gain some understanding of how a computer algorithm can produce numbers that appear to contain an element of chance.
- In a latter module, students will gain some understanding of how a computer can *simulate* natural processes that contain elements of *chance.*
- As part of computational thinking, students will know to question the reliability of computer models that simulate random events.

Scientific skills objective:

- Students will practice the following scientific skills:
  - Graphing (visualizing) data as histograms and *(x,y)* plots.
  - Describing correlations and randomness displayed by data.
  - Interpreting the contextual meaning of graphs.

Activities

In this lesson, students will:

- Use a computer program to generate random numbers.
- Test a sequence of random numbers for uniformity.
- Test a sequence of random numbers for randomness in multiple ways.

- *Optionally*, students will work with an actual computer algorithm used to generate a series of pseudorandom numbers.

Related Video Lectures (Upper-Division Undergraduate)

1. Random Numbers for Monte Carlo

   (science.oregonstate.edu/~rubin/Books/eBookWorking/VideoLecs/MonteCarlo/MonteCarlo.html)
2. Monte Carlo Simulations

   (http://science.oregonstate.edu/~rubin/Books/eBookWorking/VideoLecs/MonteCarloApps/MonteCarloApps.html)

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 show 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 be 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 that possible to attain, all we can say what the *probability* of coming up with any number is 1/6th, 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.

Exercise: Describe in your own words the meaning of "chance", "deterministic", "correlation", and "random". Give an example of the proper use of each word.

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 $\boldsymbol{r_1}, \boldsymbol{r_2}, \boldsymbol{r_3}, \ldots$ 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, …

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 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 ith random number in a sequence as $\boldsymbol{r_i}$, and assume that we know the seed $\boldsymbol{r_0}$. Let us say that we want the random numbers to be generated so that they are all greater than 0, but less than $\boldsymbol{M-1}$, where we know the value of *M*. We express this mathematically as

(3) $$\boldsymbol{0 \le r_i \le M-1}.$$

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

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

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 + Remainder 1. Well the remainder function in (4) means just the same thing: divide $\boldsymbol{a\,r_i + c}$ 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) $$\text{remainder}\left(\tfrac{13}{4}\right) = \text{remainder}\left(3\tfrac{1}{4}\right) = 1, \quad \text{remainder}\left(\tfrac{98}{3}\right) = \text{remainder}\left(32\tfrac{2}{3}\right) = 2.$$

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).

Exercise: Use $\boldsymbol{r_1}$ = 3, *c* = 1, *a* = 4, *M* = 9 in Equation (4) to generate a sequence of pseudorandom numbers. We will do this for the first three numbers in the sequence and then let you have some fun and finish the job:

(6) $$r_2 = \text{remainder}\left(\frac{4\times3+1}{9}\right) = \text{remainder}\left(\frac{13}{9}\right) = \text{remainder}\left(1\,\frac{4}{9}\right) = 4,$$

(7) $$r_3 = \text{remainder}\left(\frac{4\times4+1}{9}\right) = \text{remainder}\left(\frac{17}{9}\right) = \text{remainder}\left(1\,\frac{8}{9}\right) = 8,$$

(8) $$r_4 = \text{remainder}\left(\frac{4\times8+1}{9}\right) = \text{remainder}\left(\frac{33}{9}\right) = \text{remainder}\left(3\,\frac{6}{9}\right) = 6,$$

(9) $$r_{5-10} = 7, 2, 0, 1, 5, 3.$$

We see from this example that that we get a sequence of length *M* = 9, after which the entire sequence repeats in exactly the same order. Since repetition is anything but random, using a very large value of *M* helps hide the lack of randomness, and is one of the tricks used in pseudorandom number generators.

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 see 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) 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.

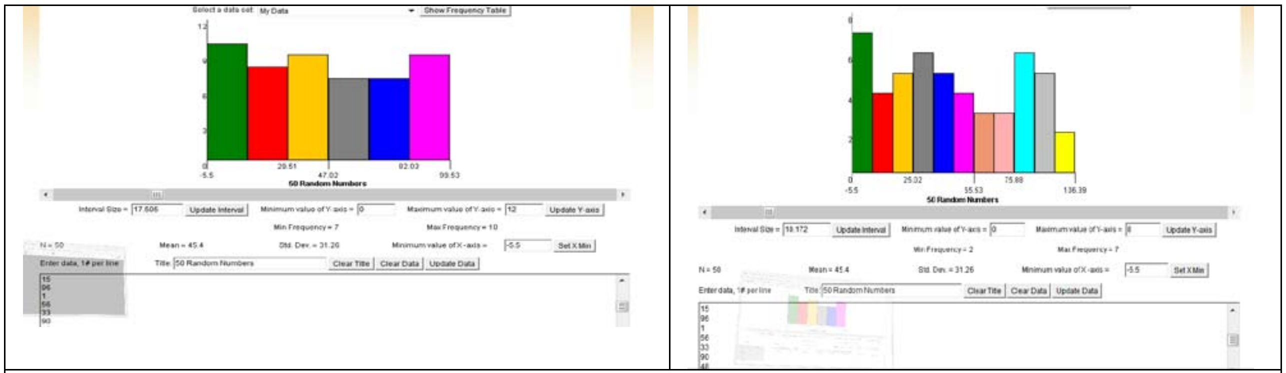
Exercise: Use your favorite random number generator to generate a sequence of 50 random numbers. (You can do this in Excel by using the *rand* function under *Formulas*, in Python using *random* function, and in Vensim using the *RANDOM UNIFORM* function; see examples in the Spontaneous Decay Module.) In case you are not able to generate those numbers at this instant, here are 50 random numbers that we have generated between 0 and 100:

(11) 21 24 43 88 78 35 38 0 13 50 66 45 6 98 99 95 4 76 89 2 34 86 85 15 96
1 86 33 90 48 44 9 88 27 8 91 17 37 4 3 79 79 31 48 72 26 64 24 71 4

Plot a histogram with 10 columns and decide if this distribution looks uniform. You can use Excell or some other program to plot a histogram of these numbers. We have gone to a Web-based interactive histogram plotter from the Shodor Foundation at

http://www.shodor.org/interactivate/activities/Histogram/

and used it to make the two histograms in Figure 1.



Interval Size = 17.606 · Minimum value of Y-axis = 0 · Maximum value of Y-axis = 12
Min Frequency = 7 · Max Frequency = 10
N = 50 · Mean = 45.4 · Std. Dev. = 31.26 · Minimum value of X-axis = -5.5
Title: 50 Random Numbers
15 96 1 56 33 90

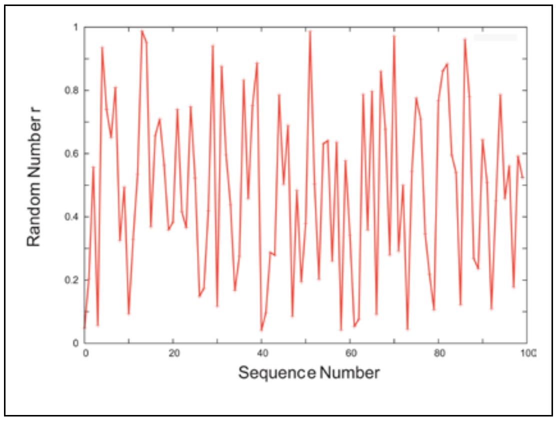
Interval Size = 10.172 · Minimum value of Y-axis = 0 · Maximum value of Y-axis = 8
Min Frequency = 2 · Max Frequency = 7
N = 50 · Mean = 45.4 · Std. Dev. = 31.26 · Minimum value of X-axis = -5.5
Title: 50 Random Numbers
15 96 1 56 33 90 48

Figure 1: *Left:* A histogram of 50 random number using bin size of 17.5. *Right:* The same numbers using a bin size of 10.2.

While these histograms do *not* tell us anything about the randomness of the distribution, they do tell us about its uniformity. The large bin sizes on the left show a fairly uniform distribution. Since the number are random, some statistical fluctuation is to be expected. Note that is we plot these same numbers using a smaller bin size, as we do on the right, then there is more fluctuation and the uniformity appears less evident. In general, statistical fluctuations are less evident as the sample size increases.

Exercise: Testing a Sequence for Randomness

Figure 2. A sequence plotted as $(x_i, y_i) = (r_i, i)$.



OK, now that we have a way of testing a distribution for uniformity, let's get on to testing for randomness. You do not have to understand how the numbers are being generated or do anything involving math, but rather just plot some graphs. Your visual cortex is quite refined at recognizing patterns, and will tell you immediately if there is a pattern in your random numbers. While you cannot "prove" that the computer is generating random numbers in this way, you might be able to prove that the numbers are not random. In any case, it is an excellent exercise to help you "see" and learn what randomness looks like.

Probably the first test you should make of a random number generator, is to see what the numbers look like. You can do this by printing them out, as we did in Equation (11), which tells you immediately the range of values they span and if they seem uniform. Another test is to plot the random number versus its position in the sequence, that is, to plot $(x_i, y_i) = (r_i, i)$ with the points connected. We do this in Figure 2 for another sequence we have generated, and you should do it now for your sequence. Here we see that the numbers lie between 0 and 1 and jump around a lot. This is what random numbers tend to look like!

A better test of the randomness of a sequence is one in which your brain can help you discern patterns in the numbers that may indicate correlations (nonrandomness). This is essentially a scatter plot as shown in Figure 3. Here we plot successive pairs of random numbers as the *x* and *y* values of data points, that is, plot $(x, y) = (r_i, r_{i+1})$, without connecting the points. For example, (r_1, r_2), (r_3, r_4), (r_5, r_6). On the left of Figure 3 you see results from a random number generator that has made a "bad" choice of internal parameters [(*a, c, M,* r_1) =(57, 1, 256, 10) in Equation (4)]. It should be clear that the sequence represented on the left is not random.

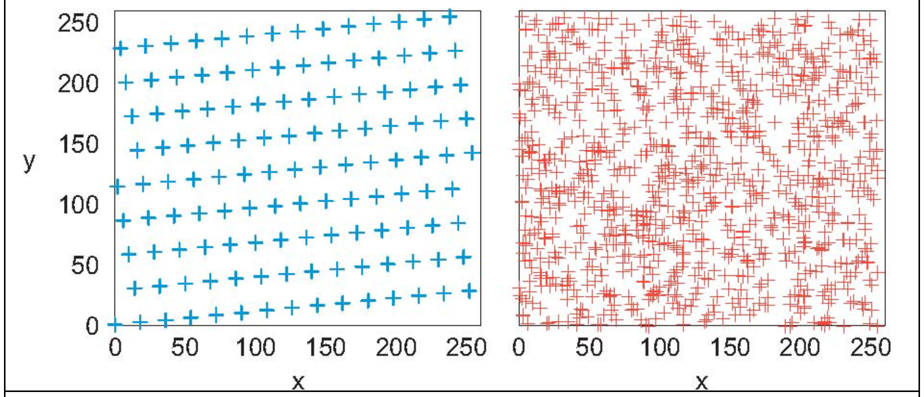


Figure 4 *Left*: Random numbers from a generator with correlations. *Right:* Random numbers from a properly functioning generator.

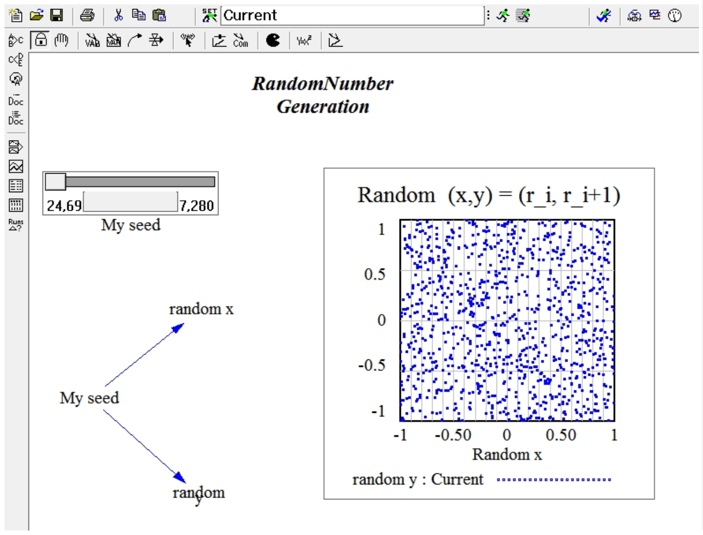
On the right of Figure 3 you see $(\boldsymbol{x}, \boldsymbol{y}) = (\boldsymbol{r_i}, \boldsymbol{r_{i+1}})$ results from a random number generator that has made a “good” choice of internal parameters [*a* = 25214903917, c = 11, *M* = 281,474,976,710,656 in Equation (4)].

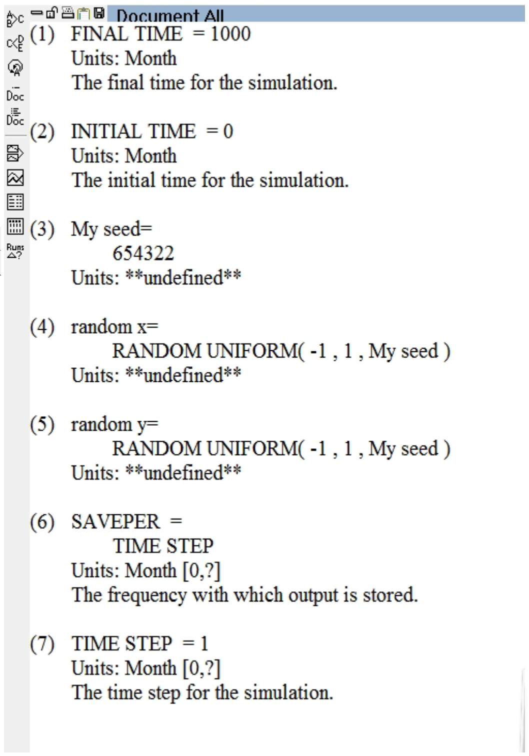
Make up your own plot and compare it to Figure 4 *right*. Be warned, your brain is very good at picking out patterns, and if you look at Figure 4 *right* long enough, you may well discern some patterns of points clumped near each other or aligned in lines. Just as there variations in the uniformity of 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 will 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.

Vensim Implementation (RandomNumbers.mdl)

Below we show the desktop and documentation window for a Vensim simulation that produces a random scatterplot using the Vensim RANDOM UNIFORM function. You can vary the seed via a slider, and vary the number of points generated by varying the FINAL TIME.





```
(1)  FINAL TIME  = 1000
     Units: Month
     The final time for the simulation.

(2)  INITIAL TIME  = 0
     Units: Month
     The initial time for the simulation.

(3)  My seed=
          654322
     Units: **undefined**

(4)  random x=
          RANDOM UNIFORM( -1 , 1 , My seed )
     Units: **undefined**

(5)  random y=
          RANDOM UNIFORM( -1 , 1 , My seed )
     Units: **undefined**

(6)  SAVEPER  =
          TIME STEP
     Units: Month [0,?]
     The frequency with which output is stored.

(7)  TIME STEP  = 1
     Units: Month [0,?]
     The time step for the simulation.
```

Assessment

1. Use a random number generator with your favorite computer program to generate a series of 1000 random numbers.
2. Test your sequence for *uniformity* by making histogram similar to Figure 1.
3. Try bin sizes corresponding to 1/5th , 1/10th, and 1/50th of the number of data points, and comment on the apparent changes in uniformity. Which size do you think represents the best test?
4. Test your distribution for randomness by making a plot of $(x_i, y_i) \;= (r_i, i)$ similar to Figure 2. What is your conclusion?
5. Test your distribution for randomness by making a plot of $(\boldsymbol{x}, \boldsymbol{y}) \;= (\boldsymbol{r}_i, \boldsymbol{r}_{i+1})$ similar to Figure 3. What is your conclusion?

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 incomplex systems.

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