Lecture 1: Introduction

Anyone who considers arithmetical methods of producing random digits is, of course, in a state of sin.

-- John von Neumann

1. Theme and Variations

You are likely coming to this course with some idea of what Monte Carlo simulation involves. In the next few lectures, we will cover the elements of mathematical probability theory that are most relevant to Monte Carlo applications. In this lecture, since it is the first, I will begin with a few examples as a way of introducing the main themes of the course. In doing so I'll omit some technical details that we'll return to in later lectures.

Example 1: *Continuous random variables*. Imagine the following *simple response time* experiment: on each trial the observer waits, finger poised over a button, until a small light abruptly illuminates. As rapidly as she can, the subject presses the button, relaxes a moment, and prepares for the next trial. That's all. The dependent measure of interest to us is the time difference D between the onset of the light and the button press.

This sort of experiment is called a *simple* response time experiment since the subject has only one possible response (Luce, 1986). In a *choice* response time experiment the subject might be asked to make a decision about the visual stimulus (*Did the light turn red? Or green?*) and then choose one of two actions according to the outcome of the decision (*Press the left button if the light turned red, else press the right button*). But here we are interested only in the speed with which a human subject can generate a motor response to a visual stimulus.

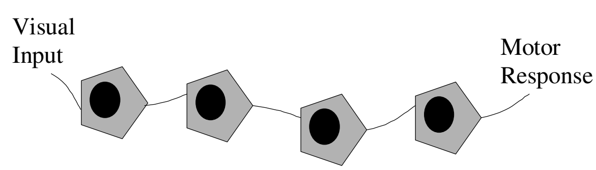

Fig. 1

Evidently, the ideal subject for this experiment would be little more than a string of neurons leading from the eye to the arm muscles that control the finger and this is the model of the process we will develop here (Figure 1). In our model, there are n neurons linked in series and each transmits a single action potential to the next neuron in the series. The first neuron initiates the process when the light is turned on and the last triggers the motor response. The overall latency D is the sum of the latencies $D_1, \cdots, D_n$ of the n neurons 'in series':

$$D = \sum_{i=1}^{n} D_j \tag{1.1}$$

Since the subject's response time D varies from trial to trial even though the stimulus is always the same, it is plausible to assume that the time required for each of the n processes to respond and pass on the signal can be modeled as a *real-valued random variable*. For our purposes, a random variable can be thought of as a box with a button on the top. Every time you press the button, you get a *realization* of the random variable. The realization of a real-valued random variable is, as you might expect, a real number. The odd thing about a random variable is that, if you press the button repeatedly, it may realize something different every time. It's random!

The one piece of information we need to characterize completely a real-valued random variable T is its *probability density function,* $f(t)$, abbreviated *pdf.* This function summarizes the probability that the realization T will be in any interval *(a,b)* of the real line as follows:

$$P[a < T < b] = \int_a^b f(t)dt \tag{1.2}$$

The pdf is all the information we can ever know concerning the random variable before it is realized.

Unfortunately, it is customary to use the same symbol, *T,* for the random variable and its realization. You just have to know from context whether T refers to the random variable (the process) or the outcome (some number).

Note: In a former course, you may have learned to distinguish between continuous and discrete random variables and also learned that only continuous random variables have pdfs. In the next lecture, though, we are going to eliminate the distinction between the two in order to simplify notation and also in order to allow us to talk about variables that are partly continuous and discrete. Every random variable will have a pdf. These turn out to be very useful and we'll encounter our first *mixed* variable in the third example below. In this response time example, though, we will confine ourselves to continuous random variables.

The successive realizations of a random variable are *independent* of one another. We'll define independence more precisely next time, but, for now it means simply that no matter what has happened on previous button presses, the pdf of a random variable never changes, and, as noted above, you can never know anything more about what the realization of a random variable might be than its pdf.

A first example of a pdf for a continuous random variable is,

$$
\begin{aligned}
f(t) &= \alpha e^{-\alpha t} \quad , \quad t \geq 0 \\
&= 0 \quad , \quad t < 0
\end{aligned}
\tag{1.3}
$$

the pdf of an *exponential random variable* with rate parameter $\alpha > 0$. To be precise, the previous equation specifies a pdf only after we have chosen a value for $\alpha > 0$. As it stands, it specifies a *parametric family* of pdfs with rate parameter $\alpha > 0$.

Let's assume that the latencies $D_1, \cdots, D_n$ can be modeled by successive realizations of the same exponential random variable all sharing the same parameter $\alpha > 0$. We still have to decide what the rate parameter is, but we have at least decided that it is the same for all of the 'neurons'. Then D is also a random variable, the sum of n independent, identically distributed exponential random variables. Once we choose $\alpha > 0$ and n, our model of the subject's observed response times is completely specified.

The term 'independent, identically-distributed' occurs often enough that we abbreviate it to *iid.* Be warned: statisticians love *tlm*'s (three-letter mnemonics) such as pdf, iid. We'll see several more.

Some of the things we might want to do with this model are, first of all, to simulate it and see how it behaves for various choices of n and $\alpha > 0$. We could carry out such Monte Carlo simulations if we had the equivalent of a 'box' that generated an exponential random variable every time we pressed the button.

Second, we might want to work out the pdf of the overall response time D. This is fairly easy to do with a little bit of calculus. If n is greater than 1, D is not an exponential random variable, but rather a *gamma random variable* (a variable with a gamma distribution). It's also continous. We'll meet the gamma family in a later lecture. For now we need only note that the sums of iid exponential random variables are gamma random variables, something we can take as simply the definition of a gamma random variable.

Third, we might want to *estimate* the values of n and $\alpha > 0$ that best account for observed experimental data. Now we are into statistics proper: this is a *statistical estimation problem* and we postpone consideration of such problems to the second half the course.

One last thing we might want to do is to decide whether the model we have developed is consistent with the data we observe for any choice of n and $\alpha > 0$. We would like *to test the hypothesis* that simple reaction times can be modeled as the sum of exponential variables. We'll return to consider hypothesis testing, an important topic in statistics, in the second half of the course.

For this simple model, we could readily estimate parameters or test the simple response time model we have developed using standard 'off-the-shelf' methods found in statistics books.

This sort of response time model was taken seriously in the 1950's and researchers tried to figure out how many neurons (n) linked visual input to motor response (Luce, 1986). The more you know about either response times or neural sciecne, the more likely you are eager to change some of the assumptions we've made in order to make the model more plausible. For example, it is typical to model the uncertainty in the motor response (the last stage) by a Normal (Gaussian) random variable, not an exponential (Luce, 1986). You might want to consider the possibility that there are multiple possible connections (a *network*) of neurons between eye and finger and that the response is triggered by the *first* signal to reach the motor stage along any path. In such a model, we have a *race* between signals flowing in parallel. How could that be modeled? Further, how could we model a *choice response time* experiment? With two possible actions and a decision to be made?

Once we've changed our model to make it more plausible or tried to model a more complex experiment, will we still be able to find solutions to our problems among standard 'off-the-shelf' statistical methods? The brief answer: very likely no.

2. *Example: A Discrete Random Variable*. In class I'll describe a hypothetical experiment where we would like to studying learning in a neural network and compare it to human learning. The problem to be solved is one of abduction: reasoning from symptoms (*sore throat, fallen arches, fever*) to a reasonable diagnosis of a disease state (e.g. *Ebola NYU).* We want to model how learning is affected in the neural network by errors in symptom clasifications (the input) or in the feedback given to the network while it is learning. We want to compare human learning to network learning as a function of the amount of corruption ('noise') in the input and feedback.

The error process we consider makes use of a *Bernoulli(p)* random variable, an example of a discrete random variable. The realization of a Bernoulli random variable is restricted to the discrete set of values $\{0,1\}$ and it takes on the value 1 with probability p, the value 0 with probability $1-p$. I'll introduce Dirac notation and briefly discuss how we might assign a pdf to a Bernoulli random variable. I'll change the definition of a pdf to the following (compare Equ. 1.2):

$$P\left[a < T \le b\right] = \int_a^b f\left(t\right)dt \tag{1.4}$$

In introducing Dirac notation we will effectively change how we compute integrals. The change is intuitively appealing and will simplify our lives immensely.

3. *Example: A Mixed Random Variable*. The third example is drawn from the literature on Bayesian vision and concerns how we might model the prior distribution of the velocities of objects in an environment. We note, first of all, that most things don't move around and so we want to assign a rather high probability p to the velocity 0. We might also decide that the distribution of the speeds of the things that are moving is exponential with rate parameter α. The resulting random variable is an example of a mixed random variable. It will turn out to be easy to do so using Dirac notation.

This course is about methods for approaching problems of estimation and hypothesis testing that are computationally intensive but very general. These methods are based on the same statistical principles as 'off the shelf' methods but computing the answer we want typically involves extensive Monte Carlo simulations on a computer. To do that, we need a source of realization of random variables that can be incorporated into a computer program.

2. Pseudo-random Number Generators (PRNGs).

Matlab includes a function ***rand(n,m)*** that returns a matrix with *n* rows and *m* columns filled with pseudo-random numbers between 0 and 1. If you start ***matlab*** and type in ***rand(1,5),*** you might get back,

ans =
0.4447 0.6154 0.799 0.9218 0.7383

a matrix with one row and five columns. There's no obvious pattern in the numbers returned and there shouldn't be: the output of ***rand(n,m)*** is intended to mimic in certain important respect a particular kind of random variable, a *uniform(0,1) random variable*, often denoted *U(0,1)*. If you type the above command again, you get a different set of five numbers.

The pdf of a uniform(0,1) random variable is,

$$\begin{aligned} f(x) &= 1 \quad , \quad 0 < x < 1 \\ &= 0 \quad , \quad \textit{otherwise} \end{aligned} \tag{1.5}$$

plotted in Figure 2. The output of ***rand(m,n)*** is intended to mimic a true uniform random variable in important respects, but in one fundamental respect it cannot. The output of ***rand(m,n)*** is computed by an algorithmic process that, if we were to start it over again, would produce precisely the same sequence of numbers.

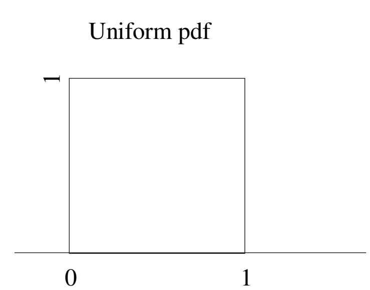

Fig. 2

In order to better understand what the advantages and limitations of pseudo-random numbers are, it helps to dissect a very simple example, *a linear congruential generator* (Knuth, 1971).

A linear congruential generator ***lnc( )*** generates a sequences of integers $U_0, U_1, \cdots, U_k, \cdots$ that are restricted to the range *0* to *m*. On each call to ***lnc( )***, you must give it as argument the previous number in the sequence. It returns the following by

$$U_{k+1} = \left(aU_k + b\right) \bmod (m+1) \tag{1.6}$$

where *a*, *b,* and m can be chosen to be any positive integers. (The operation $n \bmod p$ is the integer remainder when the integer n is divided by the integer *p.* This remainder must be between 0 and *p-1,* inclusive.) Once you've chosen these three constants and the starting value U_0 you've completely defined the sequence that ***lnc( )*** will produce. If you set *a*, *b,* and m to reasonably large values (there are rules of thumb to follow in choosing these values so as the maximize the apparent disorder in the sequence: see Knuth, 1971), the resulting sequence looks satisfyingly random. Of course, you want numbers between 0 and 1, not integers between 0 and *m.* But you need only divide $U_k + 1$ by *m+2* to map the output of ***lnc( )*** into the desired range.

It's easier to see what ***lnc( )*** is doing if we pick small values *a=5*, *b=1,* and *m=7.* If U_0 is set to 2, the resulting sequence is

2, 3, 0, 1, 6, 7, 4, 5, 2, 3, 0, 1 ….

and we see that it repeats after the eighth term in the sequence. Of course it *must* repeat after the eighth term or earlier since there are only eight possible values in the series (0,1,2,…,7). Once the series repeats it must continue to cycle endlessly since the next and later terms in the series are completely determined by the current term. Note also that the sequence produced is a little too uniform. There is exactly one occurrence of each possible value in each sequence of 8 values and so, after 16, 24, 32, etc draws we will discover that the number of 0's, 1's, etc. are all the same. Morever, notice that the sequence 2, 2 will never occur (and the sequence 2, 3 will occur in every cycle of 8). If we choose poor values of *a, b,* and *m,* then the sequence may cycle much more frequently than every *m+1* terms. If we choose, *a=2, b=2, m=7*, for example, and start with 2, we get

2, 6, 6, 6, 6, …..

a most boring outcome.

3. Tests of Randomness

Linear congruential RNGs were once considered to be a good method of generating pseudo-random numbers (1960's) but are now obsolete. As the example above suggests, they did tend to produce (after division by m+2) a series of numbers that took on values in the unit interval that were remarkably uniformly-distributed. They tended to fail in other important respects, notably in the distribution of sequences of numbers. Modern pseudo-random number generators pass tests of sequential randomness that any linear congruential generator would flunk. They will also cycle if you wait long enough but the length of a cycle is so great that you need never worry about repeating (given the expected life span of the universe).

The point of this example is that we can characterize any pseudo-random number generator in terms of the tests of randomness that it can pass. The linear congruential generator did produce a very uniform distribution but it flunked a simple sequential test.

Note that in the sequence we computed, that ODD and EVEN alternate!

The tests that a pseudo-random generator can pass determine what it is good for. If, for example, your application requires only a source of uniformly-distributed number, a linear-congruential generator wouldn't be so bad. But if you were simulating a coin toss sequence by generating random integers and choosing H (heads) when the integer is odd, T (tails) when it is even, you might be very surprised to get HTHTHTHTHTH…..

The pseudo-random number generators available in modern computer languages like ***matlab*** have passed very many tests of sequential randomness and it is very likely you'll never be able to tell then from a truly random sequence. But beware. *Never* use a home-grown random number generator. It's like doing your own dental work.

From this point on in the course, I'll drop the 'pseudo' in 'pseudo-random generator and, for example, refer to the output of ***rand(n,m)*** as realizations of a uniform(0,1) random variable.

4. Generating Non-Uniform Random Variables

Standard ***matlab*** (and the student edition) provide only two functions that generate random numbers. We've see the first, ***rand(n,m)***, and the other, ***randn(n,m)*** generates Normally-distributed (Gaussian-distributed) random variables. We'll postpone discussing them until the next lecture. The question that confronts us at the moment is, how can we generate realizations of other kinds of random variables such as the exponential distribution we encountered in the first example? The key concept we need is the definition of the *cumulative distribution function (cdf)* of a random variable *X* with pdf $f(x)$:

$$F(x) = \int_{-\infty}^{x} f(t)\,dt \tag{1.7}$$

This is just the probability that the realization of *X* ends up in the interval $(-\infty, x]$. Note the asymmetric brackets here. They're intended to remind you of Eq. 1.4. Some basic facts about probability guarantee that *F(x)* is non-decreasing and that it goes from 0 to 1.

With a little bit of geometric reasoning we can show that the cdf of a *U(0,1)* variable is just,

$$\begin{aligned} F_U(x) &= 0 \quad , \quad x \le 0 \\ &= x \quad , \quad 0 < x \le 1 \\ &= 1 \quad , \quad x > 1 \end{aligned} \tag{1.8}$$

which is plotted in Figure 3.

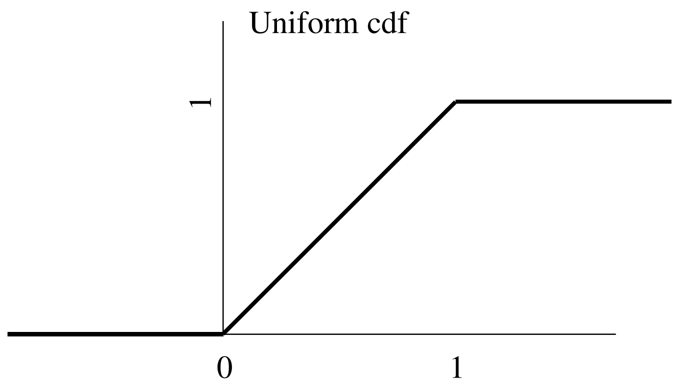

Fig. 3

A little bit of calculus gives us the cdf of the exponential distribution,

$$F_E(x) = \begin{cases} 1 - e^{-\alpha x} & , \quad x \geq 0 \\ 0 & , \quad x < 0 \end{cases} \tag{1.9}$$

It turns out that knowledge of the cdf completely determines a random variable just as knowledge of the pdf did. We need to go over some parts of the definitions carefully before we can be certain about that, but it will turn out to be true.

But, back to the problem at hand. We want to generate an exponentially-distributed variable given a U(0,1) variable and the key to that will be the inverse function of the cdf over the interval (0,1), which I'll denote as the *icdf*. It's also known as the quantile function for reasons that will become obvious later on. The icdf for the exponential is,

$$F_E^{-1}(u) = -\alpha^{-1} \ln(1-u) \quad , \quad 0 < u < 1 \tag{1.10}$$

Note the suggestive choice of dummy variable, u. *What would happen if we applied the icdf of the exponential to the realization of the uniform(0,1) random variable, U?*

Let $Y = F_E^{-1}(U) = -\alpha^{-1} \ln(1-U)$. Then we want to compute the cdf of *Y:*

$$\begin{aligned} P[Y \le x] &= P[-\alpha^{-1}\ln(1-U) \le x] \\ &= P\left[U \le 1 - e^{-\alpha x}\right] = 1 - e^{-\alpha x} \end{aligned} \tag{1.11}$$

when $x \ge 0$ and it is 0 otherwise. But this is precisely the cdf of the exponential random variable. So we have the following recipe for generating exponential random variables with rate parameter α: generate a U(0,1) random variable and transform it by the icdf of the exponential.

This recipe works for *any* random variable if we substitute the icdf of the random variable in question.

So we have a general recipe for computing random variables with any distribution whose cdf can be written down in a simple closed form.

References

Knuth, D. E. (1971), *The Art of Computer Programming: Volume 2, Seminumerical Algorithms.* Reading, MA: Addison-Wesley.

Luce, R. D. (1986), *Response Times.* New York: Oxford.

The material in this lecture and the next corresponds to Chapters 2 and 3 in Wackerly. Please go through the beginnings of both of these chapters this week.