CLARK UNIVERSITY 1887

Computer simulations and random walks
Math 217 Probability and Statistics

Prof. D. Joyce, Fall 2014

Today we'll look at some computer simulations on line at `http://www.dartmouth.edu/~chance/teaching_aids/books_articles/probability_book/bookapplets/index.html` specifically, the applets on RandomNumbers, CoinTosses, HTSimulation, HTWin, and HTLead.

The meaning of probability. Let's look at the case where we toss a fair coin, and by 'fair' in this case we mean the two outcomes H and T have the same probability, namely $\frac{1}{2}$. What does that mean in practice? If we toss the coin 10 times, does it mean we'll get 5 heads and 5 tails? Let's each do it, record our results, and talk about it. This will give us some feeling for what it means for an event to have probability $\frac{1}{2}$. We'll get a better feeling with the various computer simulations coming up next.

Computer simulations. One of the applets, CoinTosses, simulates tossing a fair coin. We'll look at it.

The advantage of a computer simulation is that you can simulate tossing the coin thousands of times, something that would take forever by hand. You might ask: how does a computer, a deterministic machine, simulate random things? For tossing a fair coin, first it generates a random number uniformly distributed between 0 and 1, then if that number turns out to be less than $\frac{1}{2}$, it calls the coin toss a head.

But now you ask, how does a computer generate a random number uniformly distributed between 0 and 1? The first applet, RandomNumbers, does that, and we'll look at the output of that applet.

These applets are written in Java, and, like most programming languages, Java has a built-in random number generator. The RandomNumbers applet creates a new random number generator, then uses a method `nextFloat` repeatedly to generate the random numbers. Documentation for Java's pseudorandom number generator is at Oracle's (once Sun's) webpage `http://docs.oracle.com/javase/8/docs/api/index.html` under Random.

It is explained there that

> An instance of this class is used to generate a stream of pseudorandom numbers. The class uses a 48-bit seed, which is modified using a linear congruential formula. (See Donald Knuth, *The Art of Computer Programming*, Volume 2, Section 3.2.1.)

So these aren't random numbers being generated, but pseudorandom numbers. That means they look like they're random and they pass many random number tests. For many purposes a linear congruential pseudorandom number generator is just fine, but it doesn't pass some random number tests, and there are better, but slightly slower, pseudorandom number generators.

The way a linear congruential pseudorandom number generator works it this. First a large number n, called the *modulus* is chosen, and for Java, it's about 2^{48}. Also, two integers a and b are selected between 0 and $n-1$. These three numbers n, a, and b are fixed once and for all.

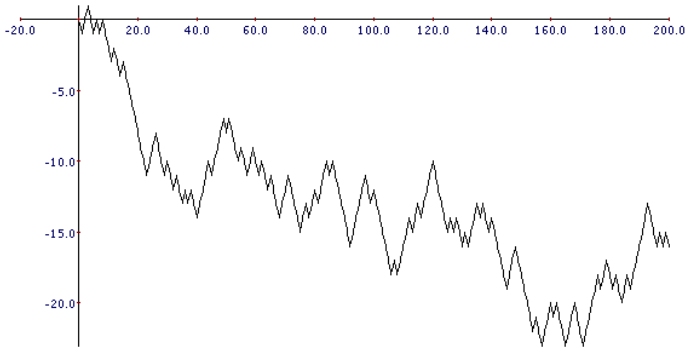
Next, when a new random number generator is created, a seed, m_0, an integer between 0 and $n-1$ is selected, usually depending on the time of day or some other variable quantity that a computer has access to. Then $x_0 = m_0/n$ is the first pseudorandom number generated. Then a new number m_1 is found from m_0 according to the formula

$$m_1 = (am_0 + b) \bmod n$$

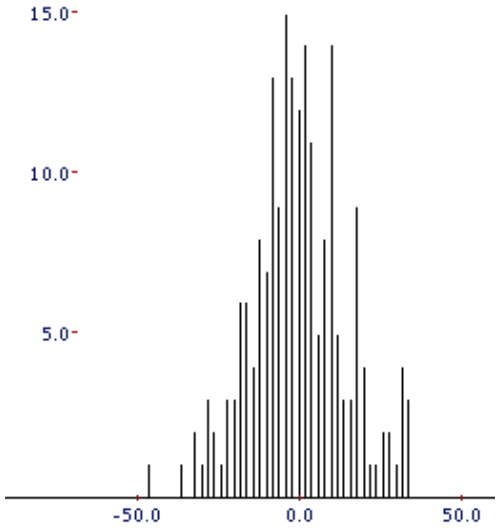
which means multiply m_0 by a and add b (that's the linear part), then take the remainder of that when you divide by n (that's the congruential

part). Then the next pseudorandom number is $x_1 = m_1/n$. And so on.

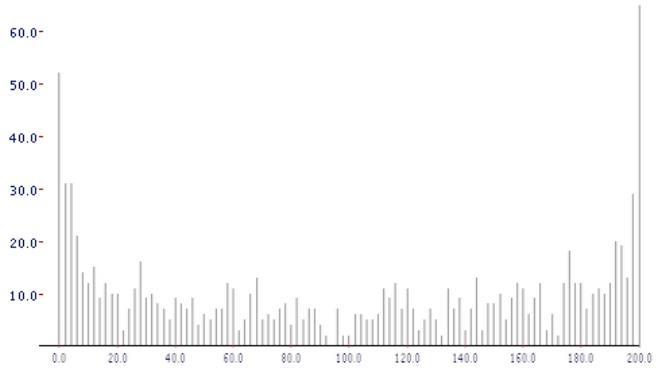
Random walks. Consider a game in which Peter and Paul flip a fair coin. Peter gets a penny from Paul each time H comes up, and loses a penny to Paul each time T comes up. They flip the penny a fixed number of times, like 200. The applet HTSimulation simulates this game and displays how how much money Peter has won over time. If he's losing it will be negative. What you see is a random walk. Here's a screenshot of 200 tosses. Peter is losing in it, but sometimes he wins.



The applet HTWin records the final state of the HTSimulation if many games are played. Note the shape of the graph, especially when the number of games is really large, like 10000. Here's a screenshot of the final outcome of 200 games, each consisting of 200 coin flips. It's high in the middle and low at both ends meaning that it usually comes out fairly even, but sometimes either Peter either wins or loses big. We'll see shapes like this throughout the course.



The applet HTLead looks at the random walk in a different way. It tells you how many of the tosses Peter is ahead during the game. The shape of this graph is very different. This screenshot shows the statistics for 1000 games, each consisting of 200 coin flips. It's low in the middle and high at both ends. That suggests that it's fairly likely that one of the players will be ahead most or all of the time.



Math 217 Home Page at
`http://math.clarku.edu/~djoyce/ma217/`