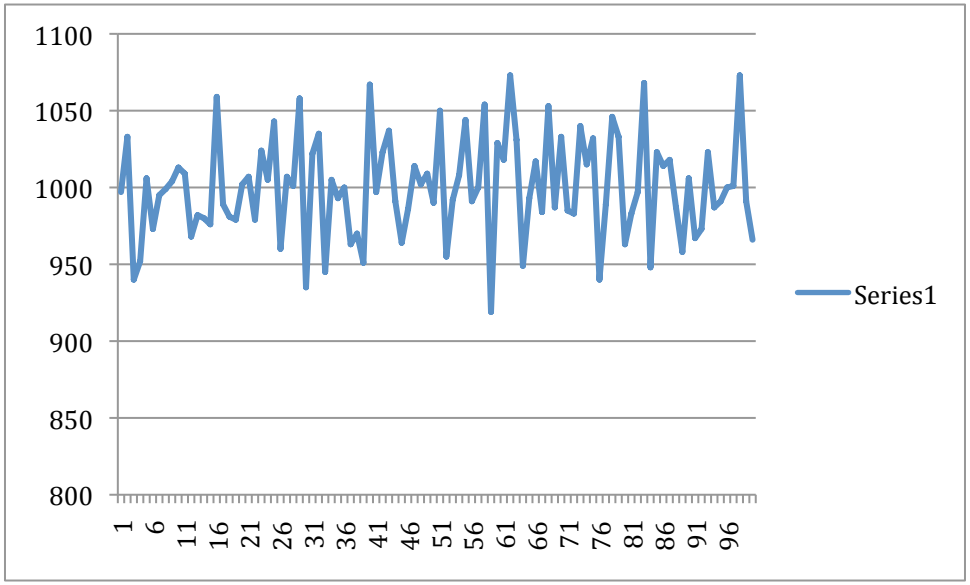
CSC 343 Operating Systems, Dr. Dale Parson, Fall 2013: **PSEUDO-RANDOM DISTRIBUTIONS**

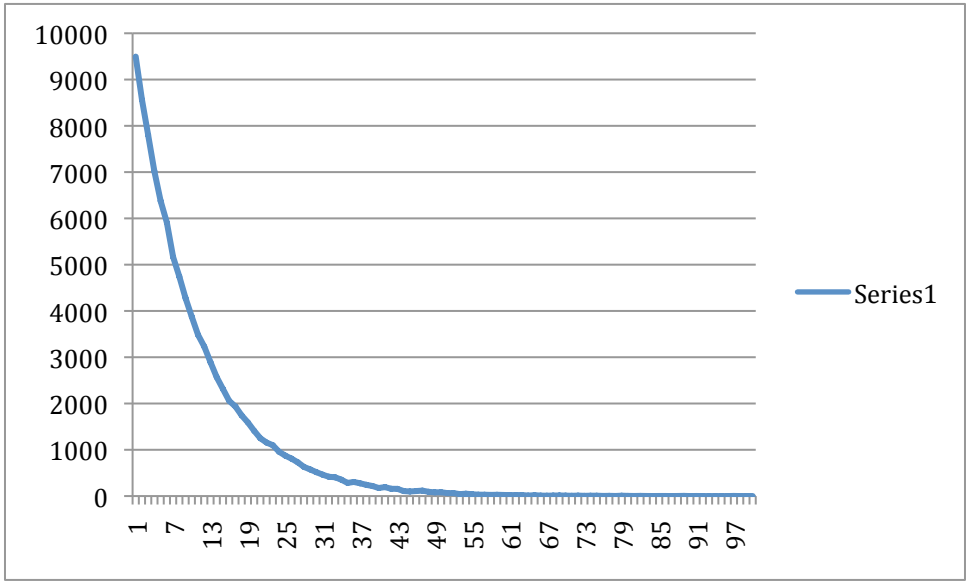
Students will be writing operating system simulations as state machines. Some threads of execution will be I/O bound, meaning that they spend most of their time waiting for input-output to complete (or waiting for other events such as timers or synchronization events from other threads). Some threads will be compute bound, meaning that they spend most of their time using CPU resources to compute. All threads are really a mix of I/O and compute activity, but many tend towards one or the other end of this I/O – CPU continuum. In using pseudo-random number generators to simulate I/O and similar wait times, or CPU occupation times, we may use four possible distributions of generated numbers this semester.

Uniform distribution, shown here from integer time values 1 through 100 (in abstract *ticks*), is the result of what many people think of as random distribution. It does not capture the I/O versus CPU bound continuum.



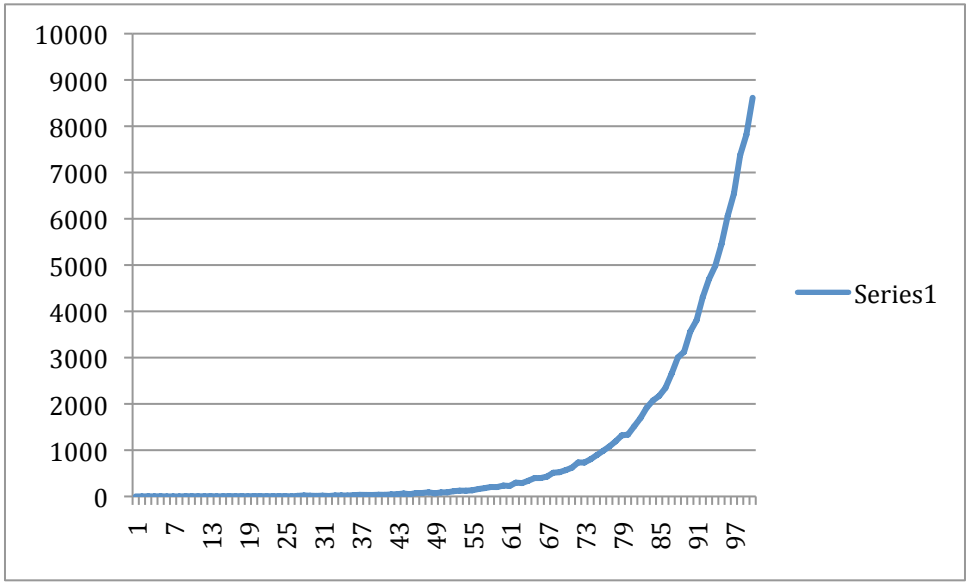
sample(1, 100, 'uniform') X 100,000 samples, X axis is return value, Y axis is histogram count

Exponential distribution starts high and then tails off according to application-specific parameters supplied to the number generator. Below is (1, 100, 'exponential', 10.0), which is an exponential decay rate starting at value 1 and generating approximately half of the 100,000 values by value 10. We can use this generator to simulate a minimum application value (such as wait time) of 1, with half the sample requests taking [1, 10] ticks. Some values fall outside the [1, 100] range – the Python library function extends to infinity – but our simulation library discards these values. The notation [1, 100] means 1 through 100 inclusive; the notation [1, 101) means 1 inclusive up to but not including 101 (exclusive). We will simulate using abstract integer time increments – ticks – that we supply when running a simulation.



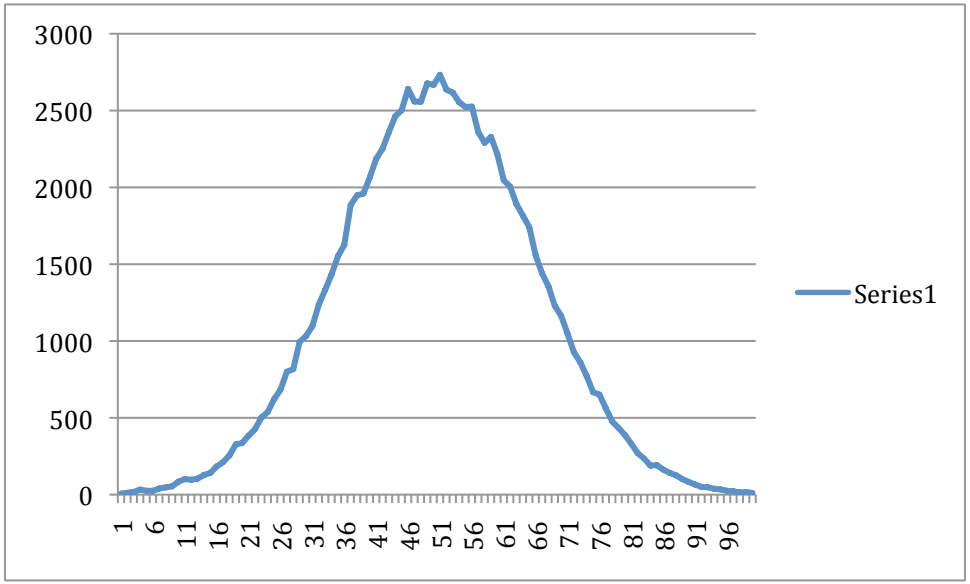
sample(1, 100, 'exponential', 10.0) X 100,000 samples, X axis is return value, Y axis is histogram count

The next graph is simply the above graph in reverse order. If we want to model CPU-bound threads, we would want most of them to have high CPU times. Note the use of 90.0 instead of 10.0, giving half of the samples in the final 10-tick range.



sample(1, 100, 'revexponential', 90.0) X 100,000 samples, X axis is return value, Y axis is histogram count

Finally comes **Gaussian** or **normal distribution**, the so-called *bell shaped curve*. Generator parameters are the *mean* and the *standard deviation*. The mean in the curve below is 50.5, halfway through the range [1, 100], and the standard deviation is 15. One standard deviation on each side (15+15 here) symmetrically from the center accounts for somewhere around 68% of the samples. Two standard deviations away from the mean account for roughly 95% of the samples, and three standard deviations account for about 99%. Again some values fall outside the [1, 100] range; the simulation library again discards these values.



sample(1, 100, 'gaussian, (1.0 + 100.0)/2.0, 15.0) X 100,000 samples, X axis is return value, Y axis is histogram count

For our purposes **exponential** with varying degrees of steepness will be useful for modeling CPU time requirements of I/O bound threads, **revexponential** with varying degrees of steepness will be useful for modeling CPU time requirements of CPU bound threads, and **uniform** and **gauss** will be useful for modeling I/O devices and networked interactions with remote processors for reasons that we will discuss.