13 Simulation

In a statistical setting there are a number of reasons we may wish to simulate from a distribution or a stochastic process. We may wish to get a feeling for how the process behaves or estimate some quantity that we cannot calculate analytically. An example of the latter case arises in Bayesian statistics, where the aim is to find the posterior distribution $f(\theta|x)$. This can be very difficult for two main reasons:

- The normalising constant ($P(D)$, above) involves a $p-$dimensional integral (where p is the number of parameters of the model, that is, $\theta = (\theta_1, \theta_2, \ldots, \theta_p)$) which is often impossible to calculate analytically.
- Even if we are able to find $f(\theta|x)$, if we want to find the marginal distribution for some part of θ this may again involve a high-dimensional integral.

Both of these reasons boil down to the fact that integration is hard.

To get around this problem, our approach will be to obtain a sample of values, $\{\theta^{(i)}\}$ for $i = 1, \ldots, n$, from the distribution of interest and use this sample to estimate properties of the distribution.

For example, the mean of the distribution is $E[\theta] = \sum_{\theta\in\Omega} \theta \Pr(\theta)$. We can estimate this from the sample set by $E[\theta] \approx \bar{\theta} = \frac{1}{n}\sum_1^n \theta^{(i)}$. This is called the sample mean of θ, and is indicated by the bar over the variable. The sample mean is an *estimator* of the mean. More generally, we estimate the mean of a function of θ by $E[g(\theta)] \approx \bar{g}(\theta) = \frac{1}{n}\sum_1^n g(\theta^{(i)})$.

How good are these estimates? Since each sample $\theta^{(i)}$ is a random variable, $\bar{g}$ is a random variable. That is, each time we obtain a different sample of values of θ, we will get a different value for $\bar{g}$. Clearly, as n, the size of the sample, increases our estimate will become more accurate but by how much?

It turns out that under quite general conditions, the main being that the samples, $\theta^{(i)}$ are independent of each other, $\bar{g}$ is normally distributed with mean $E[g]$ and variance $\mathrm{var}(\bar{g}) = \mathrm{var}(g)/n$. When stated formally, this is known as a central limit theorem.

Thus, if we have a method of simulating lots of independent samples, we can quickly get extremely accurate estimates of the quantities we are interested in.

Note that we can estimate more complex things than simple means using these methods. For example, we can estimate the shape of distributions by drawing a histogram of the sampled points.

So our attention turns to how we can generate this random sample. First we consider how we can generate or simulate randomness at all using (deterministic) computers.

13.1 Random number generation

All simulation relies on a ready supply of random numbers. There are currently no known methods to generate truly random numbers with a computer without measuring some

physical process. There are, however, many fast and efficient methods for generating pseudo-random numbers that, for most applications, are completely sufficient. The fact that these are based on algorithms that are repeatable makes them superior to physically based rngs for scientific simulation purposes.

We do not go into the mathematical details of pseudo-random number generators here as most major languages have libraries that implement perfectly adequate algorithms. It is worth considering briefly what we want in a RNG. The following quality criteria are taken from L'Ecuyer in the Handbook of Computational Statistics, 2004.

The RNG must:

- have a very long period so that no repetitions of the cycle occur;
- be efficient in memory and speed;
- repeatable so that simulations can be reproduced;
- portable between implementations;
- have efficient jump-ahead methods so that a number of virtual streams of random numbers can be created from a single stream for use in parallel computations; and,
- have good statistical properties approximating the uniform distribution on $[0, 1]$.

It is relatively simple to come up with rngs that satisfy the first of these criteria, yet the last is where the difficulties occur. The performance of rngs can be tested via the diehard test suite (or more recently, the dieharder suite). See `http://www.phy.duke.edu/~rgb/General/dieharder.php`.

13.1.1 Linear congruential generators

The most basic rngs are probably the linear congruential generators that have the form $X_{n+1} = (aX_n + b) \bmod m$ where the constants a, b and m need to be chosen. We divide the number by m to get it in the range $[0, 1]$, that is, set $U_i = \frac{X_i}{m}$.

These have poor statical properties, however, and should be avoided for simulations.

13.1.2 Shift register generators

All numbers in computers are stored as a group of bits (32 bits or 64 bits). Shift register generators work directly with this representation to produce a sequence of random numbers. Start with a seed $U_0 = 0.b_{01}b_{02}\ldots b_{0L}$ (where $L = 32$ or $L = 64$). Then get $U_i = 0.b_{i1}b_{i2}\ldots b_{iL}$ by $b_{ij} = f_j(b_{(i-1)1}, b_{(i-1)2}, \ldots, b_{(i-1)L})$ where f_j is some function $f : \{0, 1\}^L \to \{0, 1\}$.

Example: For $L = 4$, set $f_1 = b_{(i-1)3}\,\mathrm{XOR}\,b_{(i-1)4}$ and $f_j = b_{(i-1)(j-1)}$ otherwise. Starting with 0101 we get the following sequence:

```
0101
1010
1101
1110
1111
0111
0011
0001
1000
0100
0010
1001
1100
0110
1011
0101
```

□

An rng that extends this idea is the so-called Mersenne Twister which is the statisticians rng of choice for simulation. Most languages have an implementation of this algorithm. See the wikipedia page `http://en.wikipedia.org/wiki/Mersenne_twister` for more details.

The Mersenne Twister is implemented in the Colt library for Java (see `https://acs.lbl.gov/software/colt/`). It is the default rng in the Python random library.

13.2 Simulating from univariate distributions via Inversion sampling

Simulation from discrete or continuous distributions with cumulative density function $F(X)$ relies on the following result which tells us that all we need to simulate draws from an arbitrary univariate distribution is a draw from $U(0,1)$ and use the inverse of the cdf:

Result (Inversion method): If $U \sim U(0,1)$, then $X = F^{-1}(U)$ produces a draw from X where F^{-1} is the inverse of X.

Thus when the cdf is known and we can find the inverse, sampling from the distribution is easy, as the following example shows.

Example (simulating an exponential random variable): if $X \sim \text{Exp}(\lambda)$, then $F(x) = \int_{-\infty}^{x} f(t)dt = \int_{-\infty}^{x} \lambda e^{-\lambda t} dt = 1 - e^{-\lambda x}$.

It is simple to see (by setting $u = 1 - e^{-\lambda y}$ and solving for y) that $F^{-1}(u) = -\log(1-u)/\lambda$.

Since $1 - U \sim U(0,1)$ when $U \sim U(0,1)$, we can use this expression to generate exponential random variables by generating the uniform random variable u and setting $x = -\log(u)/\lambda$. □

Note that with discrete random variables, a inverse of F is ambiguously defined (since

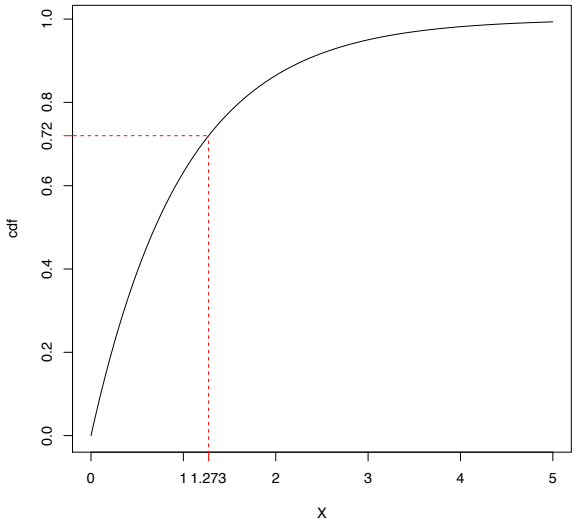

Figure 1: Sampling an $Exp(1)$ random variable using the inversion method. A uniform sample $u \sim U(0,1)$ is drawn. Here $u = 0.72$ shown on the vertical axis. This is mapped, via the cdf, to $x \sim Exp(1)$ to produce $x = -\log(1-u) = 1.272$ shown on the horizontal axis.

F is a step function). It is possible to extend the definition of an inverse to derive the following method for simulating discrete random variables.

Inversion sampling from a discrete distribution: If X is discrete with $P(X = x_i) = p_i$, we generate $U \sim U(0,1)$ and set $X = x_1$ if $u < p_1$ and $X = x_i$ if $\sum_{j=1}^{i-1} p_j < u < \sum_{j=1}^{i} p_j$.

Example: Use the inversion method to obtain samples from $X \sim Binomial$(n = 5, p = 0.3).

Solution: The possible values X can take are $(0, 1, 2, 3, 4, 5)$ with probabilities $f(x) = (0.168, 0.360, 0.309, 0.132, 0.028, 0.002)$, respectively (from Section 11.5.3). Obtain the cdf by taking the cumulative sum of these probabilities: $F(X) = (0.168, 0.528, 0.837, 0.969, 0.998, 1.000)$. Now obtain samples from X by sampling $u \sim U(0,1)$ and finding the index of the smallest value of the cdf which is larger than u. E.g. if $u = 0.439$, $x = 1$ since $F(0) = 0.168 < u < F(1) = 0.528$, Similarly, if $u = 0.972$, $x = 4$ since $F(3) < u < F(4)$. □