A Sampling of Ideas

Bill DeRose
Advisor: Gabe Chandler

September 19, 2014

1 Introduction

The difficulties we face in computational statistics are a consequence of modern technology. We should be so lucky to face the problem of designing algorithms to generate truly random numbers. And yet, we delude ourselves into believing there is randomness where there is none.

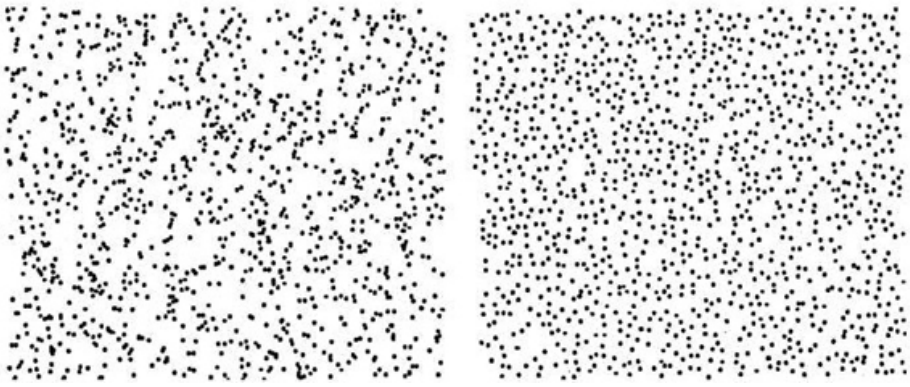

Figure 1: Which is random?

The image on the left is genuine randomness, while the image on the right is too evenly spaced for it to be *truly* random. In actuality, the left plots star locations while the right depicts the positions of glowworms on the ceiling of a cave in New Zealand. The glowworms spread themselves out evenly to reduce competition for food amongst themselves; the even distribution is the result of a non-random force. The right image suggests true randomness appears in clusters.

In our study of sampling methods, we assume the existence of a random number generator that allows us to sample $U \sim \text{Unif}(0, 1)$. From this, we explore the techniques and methods that allow us to sample from more complex distributions. We begin with a

review of classical sampling (simple sampling, rejection sampling, importance sampling) before directing the majority of our attention to Monte Carlo based methods. In specific, we explore the advantages of Hamiltonian and gradient based Monte Carlo over traditional Markov Chain Monte Carlo simulations. By treating the state space as a dynamical system, we may apply Hamiltonian dynamics to explore the target distribution more efficiently. However, before we embark on the discussion of gradient methods, we discuss the classical Markov Chain Monte Carlo algorithm and its application to sampling from high-dimensional distributions.

1.1 Markov Chain Monte Carlo

Although the sampling techniques we have discussed worked well, they required us to have a closed form cdf of the target distribution. As we will see, the previous approaches also fail as we move to higher dimensional space and the curse of dimensionality sets in. We turn to Markov Chain Monte Carlo simulations because they ameliorate many of the problems faced by classical sampling.

A Monte Carlo simulation allows us to approximate the probability of certain outcomes by running a large number of trials to obtain an empirical distribution of possible events. A Markov chain is a series of random variables $\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(m)}$ such that

$$\mathrm{P}(\mathbf{x}^{(m+1)}|\mathbf{x}^{(m)}, \mathbf{x}^{(m-1)}, \ldots, \mathbf{x}^{(1)}) = \mathrm{P}(\mathbf{x}^{(m+1)}|\mathbf{x}^{(m)}) \tag{1}$$

Markov Chain Monte Carlo simulations use Markov chains whose stationary (equilibrium) distribution we wish to sample from. By initializing the Markov chain with an initial state and transition probabilities, we may run the chain for a "sufficient" number of steps to draw samples from the desired distribution. It is common to ignore some number of samples at the beginning (burn-in), and then consider only every n^{th} sample when computing an expectation.

Below is an implementation of such a Markov Chain Monte Carlo sampler, a Gibbs Sampler which will sample from a 2D exponential.

```
> Exp.Bounded <- function(rate, B) {
+   # rexpT samples from the exponential distribution
+   # until a value less than B is observed
+   x <- rexp(1, rate)
+   while (x > B) {
+     x <- rexp(1, rate)
+   }
+   return(x)
+ }
> Gibbs.Sampler <- function(M, B) {
+   # Gibbs.Sampler uses the Gibbs sampling method
```

```
+   # to sample from a joint distribution given our
+   # knowledge of the condition distributions.
+   mat <- matrix(ncol=2, nrow = M)
+   x <- 1
+   y <- 1
+   mat[1, ] <- c(x, y)
+   for (i in 2:M) {
+       x <- Exp.Bounded(y, B)
+       y <- Exp.Bounded(x, B)
+       mat[i,] <- c(x, y)
+   }
+   layout(matrix(c(1,1,2,3), 2, 2, byrow = TRUE))
+   plot(mat, main="Joint Distribution", xlab="X",ylab="Y")
+   hist(mat[ , 1], main="Marginal dist. of X", xlab="X")
+   hist(mat[ , 2], main="Marginal dist. of Y", xlab="Y")
+ }

> Gibbs.Sampler(1000, 10)
```