19.3: Monte Carlo methods

edits: — under construction —

Introduction

Statistical methods that employ Monte Carlo methods use repeated random sampling to estimate properties of a frequency distribution. These distributions may be well-known, e.g., gamma-distribution, normal distribution, or t-distribution. The simulation is based on generation of a set of random numbers on the **open interval** $(0, 1)$ — the set of real numbers between zero and one (all numbers greater than 0 and less than 1).

> **Note:**
>
> If the set included 0 and 1, then it would be called a **closed set**, i.e., the set includes the boundary points zero and one.

The Markov chain Monte Carlo (MCMC) sampling approach can be used to solve large scale problems. The Markov chain refers to how the sample is drawn from a specified probability distribution. It can be drawn by discrete time steps (DTMC) or by a continuous process (CTMC). The Markov process is "memoryless:" predictions of future events are derived solely from their present state — the future and past states are independent.

Gibbs sampling is a common MCMC algorithm.

R code

R's uniform generator is `runif` function. Examples of the samples generated over different values (100, 1000, 10000, 100000) with output displayed as histograms (Fig. 1). Note that as sample size increases, the simulated distributions resemble more and more the uniform distribution. Use `set.seed()` to reproduce the same set and sequence of numbers

```
require(RcmdrMisc)
par(mfrow = c(2, 2))
myUniformH <- data.frame(runif(100))
with(myUniformH, Hist(runif.100., scale="frequency", ylim=c(0,20), breaks="Sturges", (
myUniform1K <- data.frame(runif(1000))
with(myUniform1K, Hist(runif.1000., scale="frequency", ylim=c(0,150), breaks="Sturges
myUniform10K <- data.frame(runif(10000))
with(myUniform10K, Hist(runif.10000., scale="frequency", ylim=c(0,600), breaks="Sturg(
myUniform100K <- data.frame(runif(100000))
with(myUniform100K, Hist(runif.100000., scale="frequency", ylim=c(0,5000),breaks="Stu
#reset par()
dev.off()
```

> **Note:**
>
> Yes, a nice repeating function would be more elegant code, but we move on. As a suggestion, you should create one! Use `sapply()` or a basic for loop.

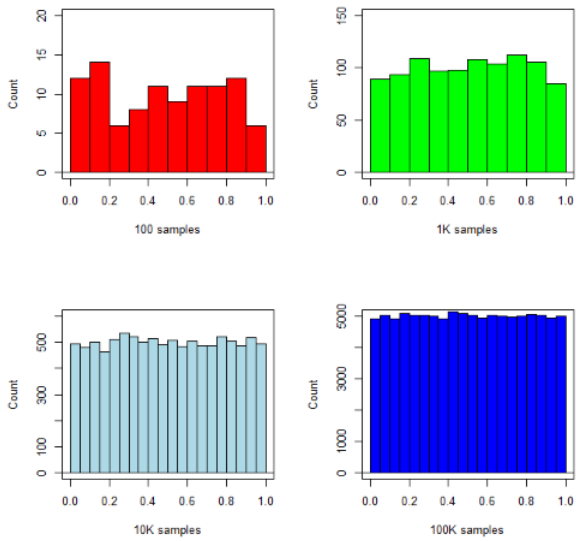

Figure 19.3.1: Histograms of runif results with 100, 1K, 10K, and 100K numbers of values to be generated.

Looks pretty uniform. A property of random numbers is that history should not influence the future, i.e., no **autocorrelation**. We can check using the `acf()` function (Fig. 19.3.2).

```
par(mfrow = c(2, 2))
acf(myUniformH, main="100")
acf(myUniform1K, main="1K")
acf(myUniform10K, main="10K")
acf(myUniform100K, main="100K"
dev.off()
```

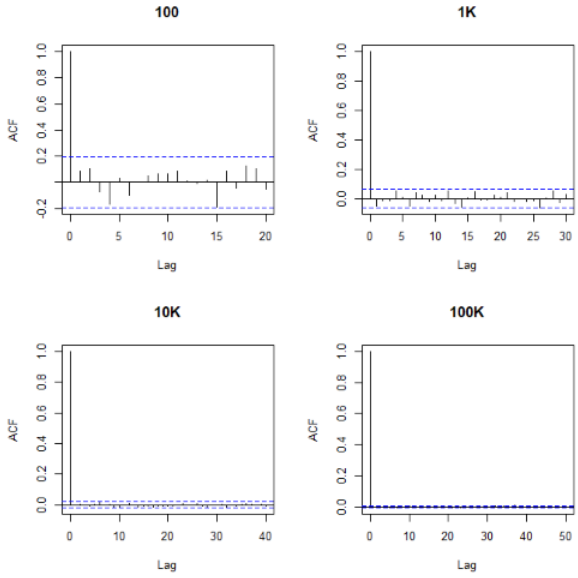

Figure 19.3.2: Autocorrelation plots of runif results with 100, 1K, 10K, and 100K numbers of values.

Correlations among points are plotted versus lag, where **lag** refers to the number of points between adjacent points, e.g., lag = 10 reflects the correlation among points 1 and 11, 2 and 12, and so forth. The band defined by two parallel blue dashed lines

Questions

1. Use `set.seed(123)` and repeat `runif(10)` twice. Confirm that the two sets are different (do not set seed) or the same when `set.seed` is used. R hint: use function `identical(x,y)` , where x and y are the two generated samples. This function tests whether the values and sequence of elements are the same between the two vectors.
