Simulations

MA386 - Statistical Programming

Summary: Monte Carlo (or numerical simulation) techniques can be used to investigate complex processes as well as evaluate the performance of various statistical methods. These are particularly useful when we have a firm model for the underlying components of the process and how these components fit together.

Simulations depend upon being able to draw a random sample from a known distribution within the computer. For example, can we select a number at random from the interval (0, 1)? The theory behind such computation is beyond the scope of this course. For our purposes, the following is sufficient:

- `R` (as well as most modern computer programs) has the ability to generate a random sample from common probability models (Uniform, Normal, Gamma, etc.).
- All random number generators within a computer are actually pseudo-random number generators. That is, they approximate true randomness to the degree that we cannot distinguish the difference. However, these tend to retain the two key components of a random sample: the values are independent of one another, and each value is representative of the same underlying population from which it was drawn.

A list of of common distributions available in the standard distribution of `R` can be obtained with `help('Distributions')`. Every distribution has four flavors of the associated function. For example, consider the Uniform distribution. The four functions are

- `dunif(x)`: Returns the value of the density function at the value `x`.
- `punif(q)`: Returns the value of the cumulative distribution function at the value of `q`.
- `qunif(p)`: Returns the quantile associated with probability `p` (inverse cumulative distribution function).
- `runif(n)`: Generates a sample of size `n` from the distribution.

The first letter of the function denotes the operation, and the remainder of the function name indicates the distribution being used.

1 Monte Carlo Integration

Summary: A large fraction of questions in statistics focus on an "expectation." Mathematically, and expectation (like the population mean) is an integral. Instead of using numerical methods to attack the problem of integration, we want to use simulation techniques, as they generalize to higher dimensional problems easily.

When modeling time-to-event data, a common probability distribution is the Gamma distribution with shape parameter α and rate parameter β:

$$f(x) = \frac{\beta^\alpha}{\Gamma(\alpha)} x^{\alpha-1} e^{-x\beta} \quad x > 0$$

where $\alpha, \beta > 0$. It is a straight-forward exercise to show that $E(X) = \alpha/\beta$. But, suppose we are interested in

$$E(\log(X)) = \int_0^\infty \log(x) f(x) dx$$

.

This integral does not provide an analytical solution. However, for specific values of the parameters (for example, $\alpha = 5$ and $\beta = 3$), we could approximate the integral numerically. Instead, let's consider the problem from a statistical perspective. Suppose X represents the lifetime of a circuit operated under normal conditions. If we wanted to estimate (approximate) the mean lifetime of circuits, we would perform the following steps:

1. Take a random sample of n circuits from the population.
2. For each of the n circuits, we would record the lifetime of the circuit under normal conditions.
3. We would compute the average in the sample, $\bar{x}$ as our point estimate; or, we could compute a confidence interval if we desire an interval estimate.
4. And if we wanted to estimate the mean of the log-lifetime, we would form a new variable $y_i = \log(x_i)$, computed for each subject in the sample and then compute $\bar{y}$.

What makes this process work, the focus of an introductory course, is that a random sample is representative of the population. Therefore, the numerical summaries computed in the sample (the statistics) should approximate the corresponding value in the population (the parameters). The key is the random sampling. The idea of a statistical simulation is to follow the above process, but within a computer instead of in the physical world. The only difference is that it *requires* us to formulate the underlying model for the *population*. With this in place, we can use the computer to take a random sample from that population (satisfying the key component of the statistical process). In this case, if we *assume* $X \sim Gamma(5, 3)$, then, we can perform a similar process:

1. *Generate* a random sample of n variates from a $Gamma(5, 3)$ distribution.
2. To estimate $E(X)$, compute $n^{-1} \sum_{i=1}^{n} X_i$.
3. To estimate $E(\log(X))$, compute $n^{-1} \sum_{i=1}^{n} \log(X_i)$.

This process is known as *Monte Carlo Integration*. We can formalize it as follows: > Let X be a random variable with a known distribution with density function $f(x)$, and let g be a known real-valued function. Then, we can estimate

$$E[g(X)] = \int g(x) f(x) dx$$

by first taking a random sample of size n such that $X_i \sim f(x)$, all independent, and then computing $n^{-1} \sum_{i=1}^{n} g(X_i)$.

Example: *(Simple Integration)* Use the `rgamma` function to approximate the mean of X and the mean of $\log(X)$ when $X \sim Gamma(5, 3)$ where 5 is the shape parameter and 3 is the rate parameter.

```
# Specify Seed
#  Controls random number generation so that the results are repeatable.
set.seed(20151023)

# Set Parameters
#  These parameters govern the simulation so that small changes can be made
#  locally instead of throughout the code.
#
#  m: number of replications
#  a: shape parameter for Gamma distribution
#  b: rate parameter for Gamma distribution
m <- 10000
a <- 5
```

```r
b <- 3


# Generate Random Variables
#  Use R to generate a random sample from population.
X <- rgamma(m, shape=a, rate=b)


# Compute Summaries
#  Estimate the mean and average log-lifetime.
#  Notice that E(log(X)) =\= log(E(X))
mean(X)
mean(log(X))
```

This code begins by specifying a *seed* value. Suppose you were to construct an elaborate simulation which had striking results, and then someone asked you to replicate your work. Because the simulation relies on random generation, the results would never be able to be replicated precisely (though we imagine the results would be similar). We would like the ability to replicate even with randomness. If the values generated were truly random, this would be impossible. However, recall that the sequences are actually pseudo-random numbers. That is, in reality there is a deterministic process underlying the generation. Therefore, we can recreate the *exact* sequence of numbers generated if we repeat the deterministic process. The seed determines where that process begins. If a seed is not specified, the internal clock of the computer is used as the seed.

Why does the above method work? That is, what makes Monte Carlo integration successful? There are a set of probabilistic theorems that govern how this average behaves to varying degrees. One of these should be familiar to each of us:

> (Central Limit Theorem) If $X_1, X_2, \ldots, X_n$ are IID random variables from an underlying distribution such that $E(X_i) = \mu$ and $Var(X_i) = \sigma^2 < \infty$, then as n increases, the distribution of the ratio
>
> $$\frac{\sqrt{n}\left(\bar{X} - \mu\right)}{\sigma}$$
>
> converges to that of a standard Normal random variable.

The CLT allows us to say that, roughly, $\bar{X} \sim N\left(\mu, \sigma^2/n\right)$. Therefore, as the sample size increases, we know that $\bar{X}$ becomes a very good approximation of μ. Further, note that since X_i is an arbitrary random variable in the definition of the CLT with only mild regularity conditions (finite mean and variance), the theorem applies to a multitude of applications. Specifically, suppose we are interested in estimating $E(\log(X))$. Then, for a random sample such that X_i has finite mean and variance, then $Y_i = \log(X_i)$ is also a random variable with finite mean and variance (typically), and further

$$E(Y_i) = E(\log(X_i))$$

So, we can apply the CLT to the Y_i's, and we have that

$$n^{-1}\sum_{i=1}^{n}\log(X_i)$$

is a good estimate of $E(\log(X))$.

Exercise: *(Numerical Integration)* Write a script that uses Monte Carlo integration to show that

$$\int_0^{\infty} e^{-x^2/2}dx = \frac{\sqrt{2\pi}}{2}$$

The key to completing this exercise is to recognize that any integrand can be re-written as an expectation with respect to some distribution. Let $f(x)$ represent the density of a distribution with support on the positive reals (like the Exponential distribution). Then, we have that

$$\int_0^\infty e^{-x^2/2} dx = \int_0^\infty \frac{e^{-x^2/2} f(x)}{f(x)} dx = E\left[\frac{e^{-x^2/2}}{f(x)}\right]$$

Now that this integral has been written as an expectation, we can use the process of Monte Carlo integration to approximate it.

The beauty of Monte Carlo integration is that we now have a justification for examining a host of problems. Expected values, standard deviations, probabilities; these are all really integrals and therefore can be estimated by their sample counter parts. We can essentially view any parameter as an integral, and it can therefore be estimated by a sample counterpart by means of Monte Carlo integration.

2 Simulating a Process

Summary: Simulations are powerful in situations in which we can describe the process, and we can model the individual components involved, but the model for the overall process is complex.

In the previous section, we introduced the key concept behind any simulation: replicate in a computer the process we would undertake if conducting a scientific study in practice. The difficult part is generally not in the implementation but in the modeling. In order for simulation to work, we stated that we needed a model for the underlying population. What if this model is complex? A good strategy is to map the process, dividing it into smaller components which can be modeled. If we can replicate the process, we can construct a simulation.

Example: *(Free Throws)* Statistics on free throw shots are available for a specific basketball player:

- She makes her first free throw of the game 20% of the time.
- She makes a free throw 30% of the time if she missed the previous shot.
- She makes a free throw 60% of the time if she made the previous shot.

Using these statistics, we are interested in determining the average number of points she will earn for the team (from free throw shots) if she shoots 10 free throws in a game. Similarly, we are interested in the probability that she will earn at least 8 points from free throws (if she shoots 10).

Notice that this is not an "easy" task analytically. With 10 free throws, there are many potential combinations. And, since the likelihood of each shot is related to the result of the previous shot, this is not an easy model to develop. However, each shot individually is simple. It is fully characterized by the above three rules. We walk through a simulation to address these questions, breaking the code into small chunks.

First, we begin by setting the seed and the parameters. Again, this ensures the simulation is reproducible and that if we would like to make any changes to the model, those changes are localized instead of searching for them throughout the code.

```
# Set Seed
set.seed(20151023)

# Specify Parameters
#  m: Number of replications
#  p0: Probability of a free throw if cold (no previous shots).
#  p1: Probability of a free throw if miss prior.
```

```
#  p2: Probability of a free throw if made prior.
#  n: Number of free throws attempted in the game.
m <- 10000
p0 <- 0.2
p1 <- 0.3
p2 <- 0.6
n <- 10
```

The first shot is unique and is therefore handled separately. There are only two outcomes with respect to this first shot: either she makes it (success) or misses it (failure). Further, the success occurs with probability 0.20. Binary decisions are modeled probabilistically using a Bernoulli distribution (a Binomial distribution with only one event).

```
# Generate the First Shot
first.shot <- rbinom(m, size=1, prob=p0)
```

The above code says to generate `m` "first shots", each made with probability `p0`. The `size=1` tells the program that she only attempts the first shot 1 time.

Each shot following the first is governed by the last two rules in the process; the likelihood of making a shot depends on the result of the previous shot. As we progress through the remainder of this example, keep in mind that there are several ways to generate the remaining 9 shots she will take in a game. We try to consider all m replicates simultaneously in order to avoid a large for-loop and therefore improve efficiency by taking advantage of the vectorized nature of the `R` language. Notice that the model is essentially the same regardless of whether the previous shot was made or missed — it is a Bernoulli model. The difference is the parameter. If the previous shot is missed, we use the probability `p1` and `p2` if the previous shot was made. What needs to be known is whether the previous shot was missed or made.

We cycle through each shot and draw the next shot from the appropriate distribution. With each step of the iteration, we update the previous shot.

```
# Determine Appropriate Sequence
#  For each simulation, loop through the 10 shots and determine the appropriate
#  action to take.

# After first shot, number made is dependent only on that first shot.
# And, the previous shot then becomes the first shot.
number.made <- first.shot
prev.shot <- first.shot

# For remainder of shots, consider the previous shot when determining the next.
for(i in 2:n){
  # Determine probability required for next shot, dependent on previous shot.
  prob.next <- ifelse(prev.shot==1, p2, p1)

  # Determine next shot, which will become the previous one.
  prev.shot <- rbinom(m, size=1, prob=prob.next)

  # Update shot count
  number.made <- number.made + prev.shot
}
```

Notice that we do not store the entire sequence of shots. We only store the information required to address the question of interest. In this case, we store the number of shots. Now that we have the number of shots made, we can *estimate* the quantities of interest.

```r
# Compute Summaries
mean(number.made)
mean(number.made >= 8)
```

The average number of shots made is estimated by taking the average observed in our study; again, this is equivalent to taking the mean of an observed sample in an experiment. The probability of making at least 8 shots is equalivalent to taking the proportion made in a sample within an experiment. We emphasize that these are estimates, not the precise analytical results. However, given the large number of simulations performed, these estimates are extremely accurate. We can also visualize the entire distribution of the number of shots made graphically.

```r
# Construct Plot Data
plot.dat <- data.frame(number.made = factor(number.made))

# Graphical Summary
#  Ensure y-axis is proportion, not frequency
ggplot(data=plot.dat, mapping=aes(x=number.made)) +
  geom_bar(aes(y=(..count..)/sum(..count..))) +
  labs(x="Number of Shots Made", y="Probability") +
  theme_bw(16)
```

The key idea in this example is that we have knowledge of how to model each component (each shot in this case). It is a Binomial random variable; that is, it takes on one of two values with a known probability (she either makes the shot or she does not). While the variable of interest (total number of shots made) is difficult to model directly, the process is not. We can easily sketch out the rules (or a flow chart) describing how the process proceeds. We simply need to translate that process to the computer.

Since we are able to repeat the process several times in the computer, it is similar to conducting a study with several replications (or samples). Therefore, we can summarize it in much the same way. The beautiful part is that our cost is minimal (some computational cost, but in this case it is minimal).

Exercise: *(Business Simulation)* Suppose that the demand for a particular Christmas card is governed by the following distribution:

Demand	Probability
10000	0.10
20000	0.35
40000	0.30
60000	0.25

We know that it costs \$1.50 to produce each card, and the card sells for \$4.00. Any cards not sold must be disposed of at a cost of \$0.20 per card. If we are primarily interested in maximizing profit, how many cards should be printed? Tip: in addition to the known distributions, `sample()` allows you to sample from specified discrete distributions.

3 Investigating Statistical Properties

Summary: Simulating a process is very important in business and scientific studies. However, in statistical studies, we are generally interested in examining the properties of a statistical method. Often, simulation studies are useful when the analytical approach is intractible (or to get a sense of what will happen before trying to develop the theory in full).

Consider a simple communication system which transmits a binary signal (either a 0 or a 1) to some receiver. However, due to interference/noise in the medium, what is received is some real number. The receiver must then interpret this value and make a decision regarding the content of the original message. For example, suppose a 0 is sent, but due to noise, the receiver sees 0.346. The receiver must then classify this 0.346 as either a 0 or a 1. If the receiver classifies it as a 0, it makes the correct decision in this case; if the receiver classifies it as a 1, it makes the incorrect decision. Suppose further that we can characterize how the noise interferes with the original signal. Specifically,

- If a 0 is sent, the noise added to the signal follows a Normal distribution with a mean of 0 and a standard deviation of 0.4.
- If a 1 is sent, the noise added to the signal follows a Normal distribution with a mean of 0 and a standard deviation of 0.5.

This scenario is illustrated in the figure below:

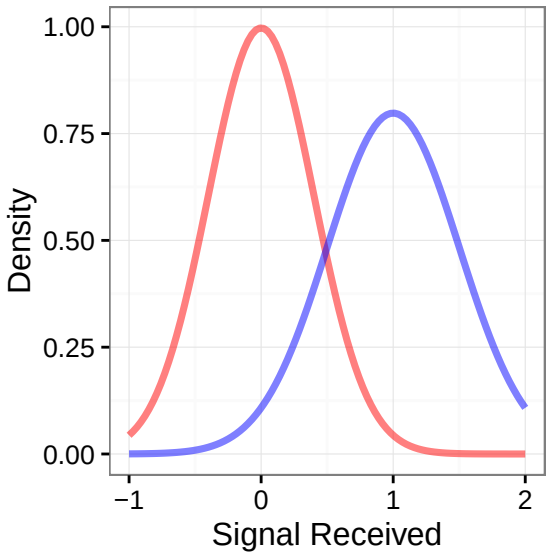

Figure 1: Illustration of a noisy communication system.

The simplest rule for classifying the received signals is to say "if the received signal is larger than q, we classify the original signal as a 1; otherwise, we classify the orignal signal as a 0." A naive rule may be to take $q = 0.5$. Essentially, this says that if the signal is closer to 1 than it is to 0, classify the original signal as a 1; otherwise, classify it as a 0. Can we do better with a simple adjustment? The popular Bayes Classifier results in choosing the value q where the two densities intersect (since each of the original signals is equally likely). In this case, the Bayes Classifier yields $q = 0.4886$. The naive approach and that of the Bayes Classifier look so similar. How great is the advantage from using this improved rule?

With some straight-forward analytical work, we could obtain the results. But, it may be just as fast to simulate this as a way of confirming our analytical results.

```
# Set Seed
set.seed(20160812)

# Specify Parameters
#  m: number of simulations
#  mu0: mean noise when signal is 0
#  mu1: mean noise when signal is 1
```

```r
#  s0: standard deviation of noise when signal is 0
#  s1: standard deviation of noise when signal is 1
#  qn: naive cut rule
#  qb: bayes classifier rule
m <- 10000
mu0 <- 0
mu1 <- 0
s0 <- 0.4
s1 <- 0.5
qn <- 0.5
qb <- 0.4886


## Run Simulation

# Generate Binary Signal
orig.signal <- rbinom(m, size=1, prob=0.5)

# Add Noise
rec.signal <- rnorm(m, mean=orig.signal,
                    sd=ifelse(orig.signal==1, s1, s0))

# Classify Using Naive Rule
naive.rule <- ifelse(rec.signal>qn, 1, 0)

# Classify Using Bayes Rule
bayes.rule <- ifelse(rec.signal>qb, 1, 0)


# Compare Accuracy
#  Compare original signal to the classification made.
table(naive.rule, orig.signal)
mean(naive.rule==orig.signal)

table(bayes.rule, orig.signal)
mean(bayes.rule==orig.signal)
```

The results suggest that the gain from the improved rule is somewhat minimal, but it does exist. We misclassify the signal a little less often. It also reveals what we might expect from examining the previous graphic, we are more likely to misclassify the signal if the original signal was a 1.

This simulation is not simulating a process, we are conducting a study. However, instead of actually sending signals and recording what the receiver sees, we perform this operation inside the computer. This allows us to evaluate the accuracy/effectiveness/properties of the classification method. We can do this with any statistical method. The downside of this approach is that simulations are limited by their scope. This simulation, for example, only considered a specific model for the noise; if the noise should change, would the difference between the classifiers still be this minimal? We would need another simulation to establish these results.

3.1 Statistical Power

As a more extended example, let's consider the case of using a simulation to investigate the power of a statistical method. Recall from your introductory course that *power* is the probability of finding evidence of an effect (rejecting the null hypothesis) when the effect actually exists (when the null hypothesis is false).

There are several "power calculators" online and in statistical programs. However, for more complex problems, the power must be estimated via simulation. We illustrate the process in this section.

Consider the production of a residential weedkiller. The chemical is sold in containers available for purchase in local home improvement stores. The company knows that after the manufacture date, there is an optimum period of time in which the weedkiller should be used for maximum effect, and that applications of the product after this time are not as effective. Products with an age beyond this time period are considered "overage" products. The company is interested in determining if more than 2% of their product on store shelves is overage. That is, they are interested in testing

$$H_0 : p \leq 0.02 \qquad \text{vs.} \qquad H_1 : p > 0.02$$

where p represents the proportion of product on store shelves which is overage. They intend to use a significance level of 0.05 for their test. As they are in the planning phase, the company would like to know how many containers need to be sampled?

This question is too broad; so, we narrow it by providing additional constraints. What level of overage is practically relevant? That is, if the actual value of p is 2.5%, is that close enough to 2% that they would not care, or is that relevant? And, if the overage were at this level, how often would you want your study to detect this difference? That is, what should the power of your study be?

Suppose that the company would like to detect if the proportion is really 3% with 90% power (80-90% power tends to be industry standards). With this goal, what sample size should be considered? This can be addressed with a simulation study.

The idea is to generate data we would likely see *if the true proportion is 3%* (the alternative hypothesis is true). Then, conduct the statistical test of interest to determine how often we would reject the null hypothesis; that will be our estimate of the power. We do this for several sample sizes.

The first step is to specify what the parameters of interest are. So, we set the value under the alternative as well as the null value and the sample sizes we would like to consider.

```
# Set Seed
set.seed(20160813)

# Specify Parameters
#  m: Number of replications
#  p0: Null value
#  p1: Alternative value to detect
#  n: Sample size
m <- 10000
p0 <- 0.02
p1 <- 0.03
samp.size <- seq(from=10, to=10000, by=10)
```

Now, for each of the sample sizes, we want to generate a sample of size n *assuming the alternative is true* (this is the key conceptual step). As each of the n containers is overage or not, we generate n containers as a Binomial distribution. We then record the number of overages.

```
# Generate Samples of Size n
#  Each sample as Binomial of size n. Needs to be repeated m times.
overages <- samp.size %>%
  lapply(function(u) rbinom(m, size=u, prob=p1))
```

Each element of the list `overages` corresponds to a different choice of the sample size. Each element is a vector of length m, corresponding to the different replications within the simulation. Now, for each sample

size choice and each replicate, we then conduct our statistical test to determine if we would reject the null hypothesis

$$H_0 : p \leq 0.02$$

(which we know to be false since we generated the data). The exact p-value can be determined by taking

$$Pr(Y \geq \text{num overages obs})$$

where $Y \sim Bin(n, 0.02)$ since we are forcing the null hypothesis. Even if you have not seen this particular test, the idea should be familiar. We enforced the null hypothesis and then determine the probability of a statistic being observed as large or larger than that observed. Since we need to change the sample size with each iteration, we use the `mapply()` function.

```
# Compute P-values
p.vals <- mapply(function(u,v){
  1 - pbinom(u - 1, size=v, prob=p0)
}, overages, samp.size)
```

The object `p.vals` is now a matrix in which each column represents a different sample size and each row a different replication of the simulation. We will "reject the null hypothesis" whenever the p-value is lower than 0.05 (the chosen significance level).

```
# Determine Decision
#  Reject if p-value less than significance level
reject <- p.vals < 0.05
```

The power, for a particular sample size, is then how often we reject H_0 for that sample size (since it was generated when the alternative was true).

```
# Compute Power
power <- apply(reject, 2, mean)
```

We now visualize what is known as the "power curve" in which we show the relationship between the power and the sample size.

```
# Compute Power Curve
ggplot(data.frame(Power=power, N=samp.size),
       mapping=aes(x=N, y=Power)) +
  geom_line() +
  geom_hline(yintercept = 0.90, colour="red", linetype=2) +
  labs(x="Sample Size", y="Power to Detect p = 0.03") +
  theme_bw()
```

From the plot, we see that by around a sample size of $n = 2000$, we have a power near 90%. Certainly, by a sample of size $n = 2500$, we have exceeded the power requirements. This lets us know the sample size to use when we conduct our study.

Again, this simulation works for specific conditions: want to detect $p = 0.03$ and we intend to test the hypothesis with an $\alpha = 0.05$ significance level. If these components change, the simulation would need to be rerun.

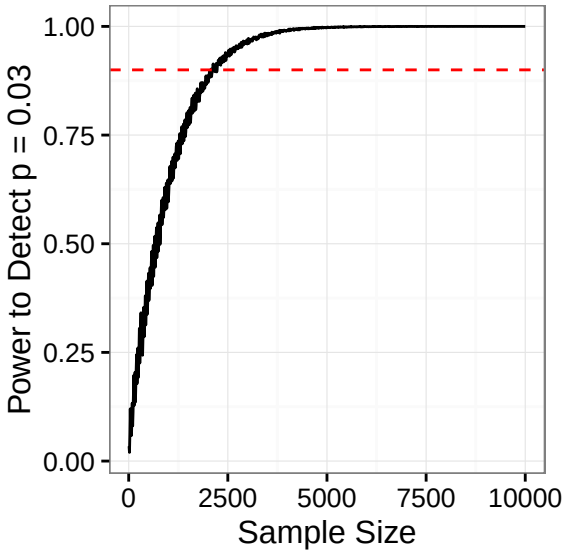

Figure 2: Power curve corresponding to Weedkiller example.