ECL 290
Statistical Models in Ecology using R

Problem set for Week 6
Monte Carlo simulation, power analysis, bootstrapping

1. Monte Carlo simulations - Central Limit Theorem example

To start getting a handle on Monte Carlo simulations, we will test the accuracy of the Central Limit Theorem. Let's hope it works, otherwise we all have to stop using ANOVAs.

The logic behind Monte Carlo is very appealing to frequentists. Most frequentist stats are based on the idea that we are estimating the probability that a certain thing would happen if we repeated the same experiment very many times. Usually we never actually repeat the experiment because the grant has run out and we have to publish the manuscript. However, we can use Monte Carlo techniques to simulate repeating the experiment (assuming the data are described by a particular statistical distribution) and thus obtain a frequentist estimate of a particular event happening.

To see how this works, we will perform Monte Carlo simulations and compare the results to a theoretical expectation, the Central Limit Theorem (CLT). The CLT says that when one takes random samples from any distribution with true mean μ and standard deviation σ, the distribution of the sample means will follow a normal distribution with mean μ and standard deviation $\sigma/n^{0.5}$ (the standard devation of the disribution of means is usually called the *standard error*). The CLT is the reason that we calculate the standard error of our sample data and use the SE to construct 95% confidence intervals around our mean. If the 95% CIs don't overlap zero, then the mean is significantly different from zero.

Anyway, the CLT is *asymptotically* true - it is guaranteed to be correct for large sample sizes *n*, but not so much for smaller *n*. It is always a good idea to calibrate our intuition about how large *n* must be before we start assuming that the CLT holds. This is where Monte Carlo comes in.

First we will establish a simple procedure for generating large numbers of random samples from a particular distribution. Let's say we need to simulate taking 10,000 samples of size *n* = 5 from a distribution with mean = 1, sd = 2. We could write a for-loop that would loop 10,000 times and simulate rnorm(n=5,mean=1,sd=2) each time, but that is long and inefficient. Instead, we will take a shortcut:

```
X = rnorm(5e4,mean=1,sd=2) # we need a total of 5 x 10,000 = 5e4 samples
X = matrix(data=X,nrow=1e4,ncol=5)
```

This rearranges X into a matrix with 10,000 rows (one for each sample) and 5 columns (the number of observations in each sample). Now let's find the distribution of sample means:

```
Mean.dist = apply(X, MARGIN = 1, FUN = "mean")
```

`apply` performs an operation on each row (`MARGIN = 1`) or column (`MARGIN = 2`) of a matrix. In this case we take the mean of each 'sample' (each row). Now look at the mean and standard deviation of this distribution. Do they match the CLT prediction?

```
mean(Mean.dist)
sd(Mean.dist)
```

Now, what about our estimate of the standard error from a single sample? In other words, when we take the sample standard deviation from each sample and use that to estimate the standard error of the mean, how well does the result match up to the CLT prediction? We have to use `apply` again, this time to take the standard deviation, not the mean, of each sample. Then we divide by the square root of 5 to estimate the SE.

```
Mean.sd = apply(X, MARGIN = 1, FUN = "sd")
Mean.se = Mean.sd/sqrt(5)
```

Now calculate the mean of the distribution of standard errors (`Mean.se`). This is the estimate of the SE you will get on average with $n = 5$. How well does it compare to the CLT prediction? Pay attention to how this last step (taking the mean of the sample standard deviations for 10,000 samples of size $n = 5$) is different from what we did at first (take the standard deviation of the distribution of sample means from 10,000 samples of size $n = 5$). The latter tells us how accurate the CLT is (how well does the actual distribution of sample means match the theoretical prediction) and the former tells us how well our sample estimate of the SE matches up to the theoretical value (how much bias is there in the estimate of SE for that *n*?).

Now we should calibrate our intuition. How does the comparison between predicted and actual SEs vary with sample size? We will need to use a for loop. To make things somewhat faster, we will generate one large 10,000 x 50 matrix, and then just take an increasing number of columns out of that matrix to simulate sample size of $n = 2, 3, 4...50$. The loop will just repeat the steps we did above.

```
X2=rnorm(50e4,mean=1,sd=2) # up to n = 50
X2 = matrix(data=X2,nrow=1e4,ncol=50)

# pre-allocate some variables for the for loop
Means = rep(NA,ncol(X2))
SDs = Means
SEs = Means
n = 0

# now loop over each possible sample size
for(n in 2:ncol(X2)){
     Mean.dist = apply(X2[,1:n],MARGIN=1,FUN="mean")
     Means[n]=mean(Mean.dist)
     SDs[n]=sd(Mean.dist) # this is the actual SD of the distribution of means
     SE.dist = apply(X2[,1:n],MARGIN=1,FUN='sd')
     SEs[n]=mean(SE.dist)/sqrt(n) # average estimate of the SE for each n
```

```
    }
```

Now calculate the expected values from the CLT:

```
n = seq(1,ncol(X2))
Mean.pred = rep(1,ncol(X2))
SD.pred = rep(2,ncol(X2))/sqrt(n)
```

Plot the results: expected mean (from CLT) and actual mean (from Monte Carlo) vs *n*, expected SE and actual standard deviation of distribution of means vs *n*, and expected SE & SE calculated from the sample standard deviation vs. *n*. How large must *n* be before the actual values start to meet the CLT prediction?

Hopefully you found that the CLT was pretty accurate. But - people always want to know if it still works when the underlying distribution of the data is non-normal. So try that again, but this time assume the data come from a very non-normal gamma distribution. Do all the same steps, but start with

```
X2=rgamma(50e4,shape=1,scale=1)
```

How well does the CLT work now?

2. Power analysis - Normal distribution

Now that we have the basics of Monte Carlo down, we will start applying it to different problems. The most common use is in power analysis. Here we are asking about the probability of making a type II error: failing to reject a false null hypothesis. Often this problem is posed as follows: we are sampling some quantity that follows a normal distribution (a sample mean, for example). The null hypothesis is that the mean = 0. If the null hypothesis is false (mean $\neq$ 0), how often will we actually *fail* to reject the null hypothesis? Let's assume the way we are performing our statistical test is to examine the 95% CI of the mean to discover whether the CI overlaps zero. This is using an alpha (probability of type I error) of 0.05.

Power analysis can be used to find out how many samples you need, or how small the variance must be, or how large the effect size must be, to reject the null hypothesis with a certain probability. For this example, let's say that logistics constrain sample size to $n = 5$, and that from prior sampling efforts we know that the standard deviation of the underlying distribution = 1. Now our question is "how does the probability of rejecting the null hypothesis vary with the actual effect size?" In other words, as the true mean gets further and further from zero (larger "effect size"), how frequently do we fail to reject the (false) null hypothesis that mean = 0?

First, create a vector of possible true means, and pre-allocate a variable we will need for our for loop:

```
MU = seq(0,5,0.01)
Test = rep(NA,length(MU))
```

Now execute our loop. At each step we will create a large vector of 'samples' drawn from a distribution with that true mean and the standard deviation we already knew. Then for each sample we will calculate the sample mean and sample standard error. Then we can determine what fraction of the samples would have led to rejecting the null hypothesis.

```
for(n in 1:length(MU)){
# Create 'samples' drawn from distribution with that value of MU
Data = matrix(rnorm(5e4,MU[n],1),nrow=1e4,ncol=5)
# Calculate mean, SE
Means = apply(Data,MARGIN=1,FUN="mean")
SEs = apply(Data,MARGIN=1,FUN="sd")/sqrt(5)

# Now do the test.  We will assume that we are testing at alpha=0.05
# So if the lower CI does not overlap zero, we reject the null hypothesis
Lower.CIs = Means-1.96*SEs
#How many times is null hypothesis rejected?
Test[n] = mean(Lower.CIs > 0)
# The proportion of cases in which we reject the (false) null hypo. = Power.
}
```

The vector `Test` holds the results: the power of our test for a range of effect sizes. Plot `Test` vs. `MU` and see how power increases with effect size.

Often we want to know a slightly different question: how large must the sample size be in order to detect a a particular effect size with a particular power? Re-write the code we just created to calculate power for a range of *n* but assuming an effect size (mu) of 1. How large does *n* have to be to detect that difference in means with power = 0.8?

3. Power analysis - Binomial distribution

Power analysis for normal distribution problems like the last is pretty straightforward. Sometimes we need to answer a slightly more difficult question. Consider this example: you are sampling fish from a location that may be exposed to endocrine disrupting compounds (EDCs). The symptom of endocrine disruption is male fish with malformed testes. You take a sample of male fish, and none of them show malformation. How confident can you be that there is no endocrine disruption at the site?

To rephrase that question in a more statistical way, let's assume that if EDCs are present, they produce some probability *T* of malformed testes. If EDCs are not present, $T = 0$. The actual number of fish with malformed testes in a sample of size *n* will follow a binomial distribution with probability *T*. For any sample size, there will be some nonzero values of *T* that will produce observations of all zeros some fraction of the time. The idea is to figure out what is the smallest value of *T* that is likely to give that result - you can rule out higher values of *T* as unlikely. Given a particular value of *n*, what is the minimum value of *T* that would produce a sample of all zeros < 5% of the time (assuming you want to be 95% confident)?

First, setup a vector of sample sizes and values of *T*, and pre-allocate some variables for a for loop.

```
Ns = seq(1,100,1)
Ts = seq(0,1,0.01)
Frac = Ts*NA
MinFrac = Ns*NA
```

Now run the loop. For each value of *n*, find the minimum value of *T* that would produce a sample of entirely zeros less than 5% of the time.

```
for(N in 1:length(Ns)){ # loop over Ns
for(T in 1:length(Ts)){ # loop over Ts for each value of N
Data = rbinom(n=1e4,size=Ns[N],prob=Ts[T]) # simulate the data
Frac[T] = mean(Data==0) # fraction of observations with no malformations
} # end the loop over Ts
is5pct = Frac<0.05 # the values of T for which <5% samples are all zero
whichT = which(is5pct) # indices of T corresponding to TRUE values in is5pct
MinFrac[N]=Ts[whichT[1]] # the lowest value of T that can be rejected at 95%
} # end the loop over Ns
```

Now plot the results (MinFrac vs. Ns). How large must *n* be to determine whether $T = 0$?

4. Bootstrapping

Monte Carlo techniques are useful when we know what the underlying distribution is and want to simulate a large number of repeated samples to estimate p-values in the frequentist paradigm. Bootstrapping is useful in a similar way, especially when the underlying data distribution is not known.

To get the hang of bootstrapping, we will work an example where we should know the correct answer. In the first exercise we found that when we take many Monte Carlo samples from the same distribution, the SD of the distribution of the sample means is equal to the true standard deviation divided by the square root of the sample size. Now let's see whether bootstrapping the data is equivalent to taking those Monte Carlo samples.

First, pre-allocate a variable
```
M = rep(NA,1e4)
```

Now, obtain one sample from a normal distribution
```
N = 100
Data = rnorm(N,mean=0,sd=1)
```

Then resample the data (with replacement!). This is bootstrapping.
```
for(b in 1:1e4){
     Bsamp = resample(x = Data,size = N,replace=TRUE)
     M[b] = mean(Bsamp) # find the mean of the resampled dataset
```

```
    }
```

There may be a way to do this without a for loop. In Matlab there definitely is, but I haven't figured out how in R. In any case, this works fine if a bit slowly.

Now compare the SD of the distribution of bootstrapped means (`M`) to the predicted SE of the mean (`1/sqrt(N)`). Try a range of values of N (say 10, 100, 1000) and a range of number of bootstraps (say 10, 100, 1000, 10,000) and see how the accuracy of the result is affected.

5. Bootstrapping - second example

We would almost never bootstrap in an example like #4, because it is so easy to use the CLT to find our answer. So here is a different example in which bootstrapping could be more useful. Consider a case like the bluehead wrasse data: a sample that could be drawn from a negative binomial or a Poisson distribution, but we are not sure which it is. One easy test for determining whether data are 'clumped' (as they would be in the negative binomial case) is to compare the mean and variance. In a Poisson distribution, mean = variance, but variance > mean for a negative binomial distribution. So, given a single sample we could compare the variance to the mean, and if sample variance > sample mean, then that is evidence for the negative binomial instead of the Poisson. Of course, even if the data are Poisson, the sample mean will almost never be exactly equal to the sample variance. So it would be nice to be able to compare the distribution of the difference between the sample mean and sample variance so that we can have more information about the weight of evidence for the negative binomial distribution. We can't use a Monte Carlo techniques to simulate the negative binomial distribution, because we don't know the parameters (if we did, we'd have our answer already!). So we can bootstrap instead.

First, obtain some data from a distribution with known parameters.

```
Data = rnbinom(10,size=1,mu=0.5)
```

Now pretend that we don't know what those parameters are and want to examine the evidence for the hypothesis that variance = mean.

First, pre-allocate a variable, and load in a package we will need.

```
Btest = rep(NA,1e4)
library(gdata)
```

Now bootstrap. This time our "test statistic" is not the sample mean (as in the last example) but is the difference between the variance and mean.

```
for(b in 1:1e4){
     Bsamp = resample(Data,length(Data),replace=TRUE) # resample Data
     Btest[b] = var(Bsamp)-mean(Bsamp) # calculate the 'test statistic'
      }
```

Now examine the distribution of the test statistic. If the distribution overlaps zero, that is evidence that the true mean = true variance and the underlying distribution of the data is Poisson.

For example, look at the lower 5% quantile of the test statistic distribution. If this value is less than zero, then 5% of the distribution is less than zero. This is roughly equivalent to seeing whether the 95% CI of the difference between the mean and variance overlaps zero. In other words, we are testing the hypothesis that the difference equals zero with alpha = 0.05.

```
quantile(Btest,0.05)
```

Try this again with a larger sample size in the original dataset and see if the results improve.

At the beginning of this question, I suggested that we couldn't use Monte Carlo techniques in this case. That's not quite right - we could have simulated a Poisson distribution with lambda = sample mean of the data, and the used those Monte Carlo samples to find the distribution of the difference between sample mean and sample variance. Then we could compare our observed difference to that distribution to see how extreme our observation was under the null hypothesis that the data are Poisson. As I've said before, there are usually several ways to do anything.