STAT 621 Lecture Notes
Monte Carlo Simulation: Overview and Examples

Broadly speaking, Monte Carlo simulation methods use random sampling to evaluate approximately the solution to a computational question. Some examples:

In this procedure, random numbers generated on a computer take the place of actual observed data in the computation of a quantity of interest. Unlike with real data, the properties of those random numbers can be controlled completely, and the process can be repeated many, many times with little or no expense. Examining the set of results we get from this repeated sampling allows us to evaluate how well our computation works, under the conditions with which the data were generated.

Thus we can use Monte Carlo simulation to test out ideas or compare competing methods without having to actually gather data, and without having to work through potentially complex theoretical derivations. One caveat about that: It is always better to derive properties analytically if possible. Monte Carlo methods are always approximate.

Let's write out the general steps for Monte Carlo simulation. After that we will look at some examples (there are many more on the web. See for example Count Bayesie's blog post *6 Neat Tricks With Monte Carlo Simulations*).

Example 1: If we flip a fair coin 10 times, what is the probability that we get three or fewer heads? Well this is a pretty easy question to answer exactly, and there is no reason to use simulation other than to illustrate the process.

If $X \sim$Bin(10,.5), we want $P(X \leq 3)$ which we find in R is about 0.1719.

```
> pbinom(3,10,.5)
  [1] 0.171875
```

Now let's use Monte Carlo simulation to approximate an answer to that question. What do we need to do?

Here is my R code and result. Note there are many equivalent ways to write code for this.

```
# first try the process one time
flips=rbinom(10, 1, .5)
flips
  [1] 0 1 1 0 1 1 0 1 1 1
no.heads=sum(flips)
no.heads
  [1] 7
(no.heads<4)
  [1] FALSE


# Now wrap a loop around these commnads and repeat M times
M=100
out=rep(NA, M)
for (i in 1:M)
  {
  flips=rbinom(10, 1, .5)
  no.heads=sum(flips)
  out[i]=(no.heads < 4)
  }

out
  [1] FALSE FALSE  TRUE  TRUE FALSE FALSE FALSE FALSE FALSE .... ETC.....

sum(out)/M     # our MC estimate
  [1] 0.16
```

Example 2: Now let's think about something more complicated. Again imagine flipping a coin 10 times. Suppose that the probability of a head on the 1st flip is $p_1 = 1/2$, but for subsequent flips, the probability depends on whether or not a head was observed on the previous flip. Specifically, for flip $i = 2, \ldots, 10$,

$$p_i = \begin{cases} p_{i-1} + .05 & \text{Head on flip } i-1 \\ p_{i-1} & \text{Tail on flip } i-1 \end{cases} \tag{1}$$

Here the success probability is not constant and trials are not independent. Suppose you just ignored this and computed $P(X \leq 3)$ as in the previous example. Would that be ok? What would you actually be estimating here? These are just some of the questions we might investigate in a situation like this. Others?

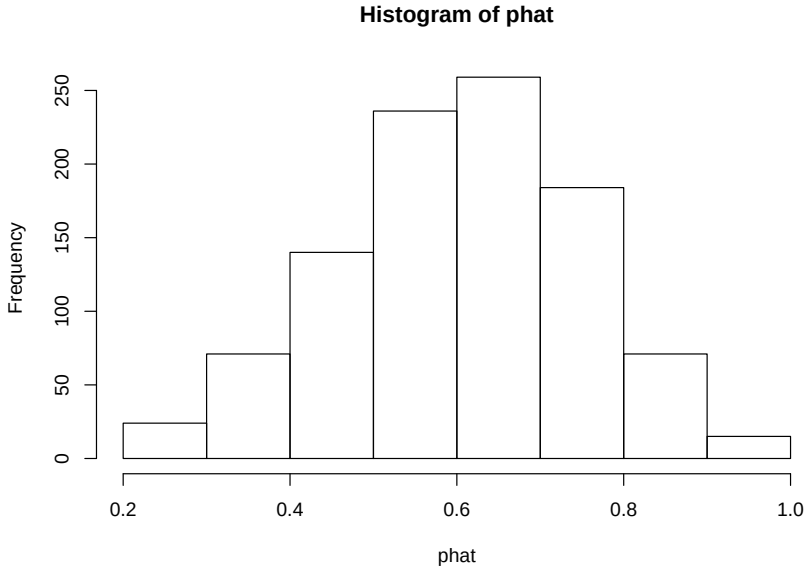
Here's some R code that simulates sets of $n = 10$ flips with the probability structure described above. I repeat this M times, and here I save the number of heads in each set of 10 flips.

```
  # first work out the code to get one sequence of 10 flips\

p1=.5
flips=rbinom(1,1,p1)
for (i in 2:10)
  {
  oldflip=flips[i-1]
  newprob=p1[i-1]+.05*(oldflip==1)
  flips=c(flips, rbinom(1,1,newprob))
  p1=c(p1,newprob)
  }


flips
   [1] 1 0 1 1 0 1 1 1 1 1


p1
   [1] 0.50 0.55 0.55 0.60 0.65 0.65 0.70 0.75 0.80 0.85


#---------------------------------------------------------------------
```

```
  # now wrap a loop around this and repeat many times; count # heads

M=1000
out=rep(NA, M)
for (j in 1:M)
  {
  flips=rbinom(1,1,.5)
    for (i in 2:10)
    {
    oldflip=flips[i-1]
    newprob=p1[i-1]+.05*(oldflip==1)
    flips=c(flips, rbinom(1,1,newprob))
    p1=c(p1,newprob)
    }
  out[j]=sum(flips)
  }

out
  [1]  8  7  7  7  8  6  9  6  8  4  4  5  4  7  8  3  ....ETC ....
```

Now let's estimate some stuff: $P(X \leq 3)$, $\widehat{p} = X/10$, sampling distribution.....

```
sum(out<4)/M
  [1] 0.024

phat=out/10
mean(phat)
  0.654

hist(phat)
```



Example 3: Now let's imagine using Monte Carlo simulation to compare the power of the Wilcoxon signed rank test and the sign test under some different conditions. Here are the steps in my simulation:

(1) Simulate n differences $Z_i = Y_i - X_i$ from distributions with median θ.

(2) For each set of differences test $H_0 : \theta = 0$ against $H_1 : \theta > 0$.

(3) Record whether the null is rejected or not.

(4) Repeat steps 1-3 M times. Estimate of power for each procedure as the proportion of times the null is rejected.

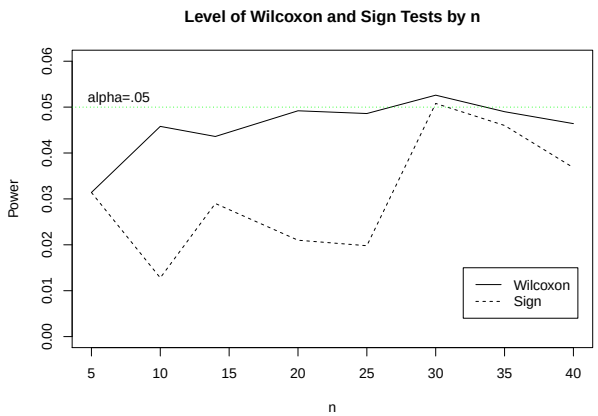
Here's some R code.

```
library(BSDA)
M=500
n=20
theta=0
pvalue.Wilcoxon=rep(NA, M)
pvalue.Sign=rep(NA, M)

for (i in 1:M)
  {
  Z=rnorm(n)+theta              #assuming Z is Normal(theta, 1)
  temp1=wilcox.test(Z, alt="greater")
  temp2=SIGN.test(Z, alt="greater")
  pvalue.Wilcoxon[i]=temp1$p.value
  pvalue.Sign[i]=temp2$p.value
  }
Reject.Wilcoxon=(pvalue.Wilcoxon<.05)
Reject.Sign=(pvalue.Sign<.05)
Power.Wilcoxon=sum(Reject.Wilcoxon)/M
Power.Sign=sum(Reject.Sign)/M
cbind(n, theta, Power.Wilcoxon, Power.Sign)
          n   theta   Power.Wilcoxon   Power.Sign
    [1,] 20      0        0.04           0.02
```

Interesting! What are those estimates of? Comments?

I repeated this for several different n. Just run the code above multiple times, each time changing n. (Actually I just wrapped another loop around the whole thing as shown below.) The results are shown in a table as well as a plot.

```
M=5000
n=c(5, 10, 14, 20, 25, 30, 35, 40)
theta=0
Results=matrix(NA,nrow=length(n), ncol=4)

for (j in 1:length(n))
{
pvalue.Wilcoxon=rep(NA, M)
pvalue.Sign=rep(NA, M)

for (i in 1:M)
  {
  Z=rnorm(n[j])+theta              #assuming Z is Normal(theta, 1)
  temp1=wilcox.test(Z, alt="greater")
  temp2=SIGN.test(Z, alt="greater")
  pvalue.Wilcoxon[i]=temp1$p.value
  pvalue.Sign[i]=temp2$p.value
  }
Reject.Wilcoxon=(pvalue.Wilcoxon<.05)
Reject.Sign=(pvalue.Sign<.05)
Power.Wilcoxon=sum(Reject.Wilcoxon)/M
Power.Sign=sum(Reject.Sign)/M
Results[j,]=cbind(n[j], theta, Power.Wilcoxon, Power.Sign)
}
Results=as.data.frame(Results)
names(Results)=c("n", "theta", "Power.Wilcoxon", "Power.Sign")
Results
   n theta Power.Wilcoxon Power.Sign
   5     0         0.0314     0.0314
  10     0         0.0458     0.0128
  14     0         0.0436     0.0290
  20     0         0.0492     0.0210
  25     0         0.0486     0.0198
  30     0         0.0526     0.0508
  35     0         0.0490     0.0460
  40     0         0.0464     0.0368

plot(Results$n, Results$Power.Wilcoxon, ylim=c(0, .06), type='l', xlab="n", ylab="Power")
lines(Results$n, Results$Power.Sign, lty=2)
abline(h=.05, lty=3, col="green")
title("Level of Wilcoxon and Sign Tests by n")
legend(32,.015,c("Wilcoxon", "Sign"),lty=c(1,2))
text(7, .052, "alpha=.05")
```



Example 4: The One-Sample Kolmogorov-Smirnov Test Revisited. Recall that the one-smaple KS test is often used for testing whether a sample is Normally distributed. Recall the test statistic:

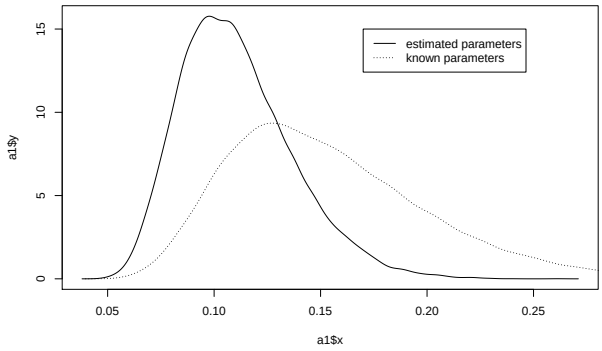
Typically, the actual mean and variance of that Normal distribution is unknown. A common practice is to substitute the estimated mean and variance in place of the unknown parameters. You examined this in a previous homework problem – what did you find?

I wondered if we could approximate the null distribution of the KS test statistic that is computed with sample estimates of population mean and variance. There may very well be an analytical solution for this, but since I was in a hurry I decided to use Monte Carlo simulation. Let's sketch a quick outline of a simulation program:

My R code for simulating the null distribution is on the following page.

First I let $\mu = 0$ and $\sigma = 1$ and $n = 30$. I get M=20,000 realizations of the KS test statistic, both as if these parameters were known, and as if they were unknown but replaced with sample estimates. And I make a plot.

```
# ---------------------------------------------------------------------
# mc example:
# null dist of ks normality stat -- one sample, estimate parameters

# H0:  data are normal vs. Ha: not normal
# generate normal data
# calc sample mean and var
# get value of test stat
# save M times, and summarize distribution

# first -- use same mean and var  mu=0 and sd=1, n=30
M=20000
n=30
mu=0
sigma=1
KSknown=rep(NA, M)
KSest=rep(NA, M)
for (i in 1:M)
{
  data=rnorm(n, mean=mu, sd=sigma)
  KSknown[i]=ks.test(data, pnorm, mu, sigma)$statistic
  KSest[i]=ks.test(data, pnorm, mean(data), sd(data))$statistic
}
a1=density(KSest)
plot(a1$x, a1$y, type='l')
b=density(KSknown)
lines(b$x, b$y, lty=3)
legend(.2, 15, c("estimated parameters", "known parameters"), lty=c(1,3))
```

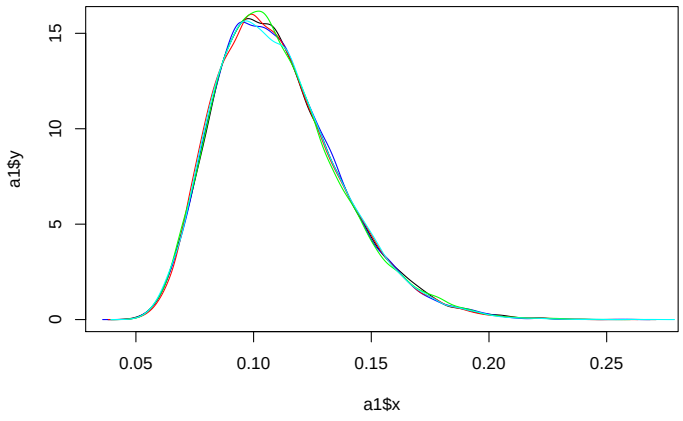



```
# ---------------------------------------------------------------------
```

As we saw in the homework, there is a big difference between these two distributions. The solid line in the above plot is the null distribution of the KS statistic using estimated parameters. But as far as I know, this is specific to the choices of μ, σ^2 and n. Maybe the null distribution will look different if I let $\mu = 10$, etc. Let's use Monte Carlo simulation to find out.

```
# ------------------
# mu=10, sd=1
M=20000
n=30
mu=10
sigma=1
KSknown=rep(NA, M)
KSest=rep(NA, M)
for (i in 1:M)
{
  data=rnorm(n, mean=mu, sd=sigma)
  KSknown[i]=ks.test(data, pnorm, mu, sigma)$statistic
  KSest[i]=ks.test(data, pnorm, mean(data), sd(data))$statistic
}

# ------------------------------
```

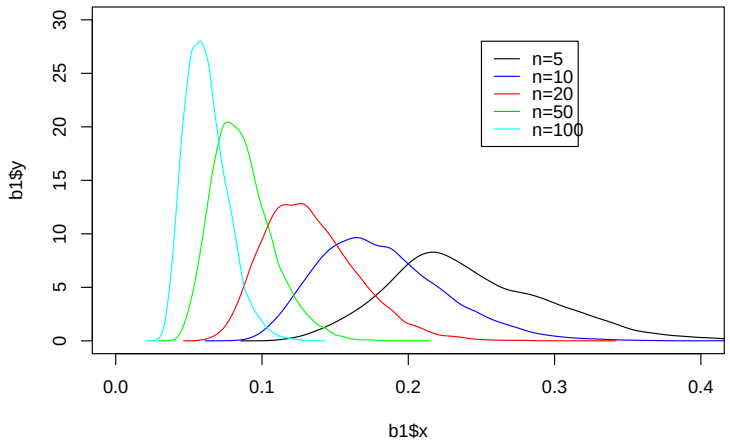
```
# mu=0, sd=10
M=20000
n=30
mu=0
sigma=10
KSknown=rep(NA, M)
KSest=rep(NA, M)
for (i in 1:M)
{
  data=rnorm(n, mean=mu, sd=sigma)
  KSknown[i]=ks.test(data, pnorm, mu, sigma)$statistic
  KSest[i]=ks.test(data, pnorm, mean(data), sd(data))$statistic
}

# --------------------
# mu=10 sd=10
M=20000
n=30
mu=10
sigma=10
KSknown=rep(NA, M)
KSest=rep(NA, M)
for (i in 1:M)
{
  data=rnorm(n, mean=mu, sd=sigma)
  KSknown[i]=ks.test(data, pnorm, mu, sigma)$statistic
  KSest[i]=ks.test(data, pnorm, mean(data), sd(data))$statistic
}

# -----------------
# mu=-145 sd=39

M=20000
n=30
mu=-145
sigma=39
KSknown=rep(NA, M)
KSest=rep(NA, M)
for (i in 1:M)
{
  data=rnorm(n, mean=mu, sd=sigma)
  KSknown[i]=ks.test(data, pnorm, mu, sigma)$statisti
  KSest[i]=ks.test(data, pnorm, mean(data), sd(data))$
}
#======================

# plot all normals
plot(a1$x, a1$y, type='l')
lines(a2$x, a2$y, type='l', col='blue')
lines(a3$x, a3$y, type='l', col='red')
lines(a4$x, a4$y, type='l', col='green')
lines(a5$x, a5$y, type='l', col='cyan')

#=================
```



Doesn't look like the actual value of mu or σ^2 matters too much. I suspect that if we used a larger M, those density estimates would line up almost exactly.

What about n? I repeated the basic process keeping $\mu = 0$ and $\sigma = 1$, but letting n vary, $n = 5, 10, 20, 50, 100$. The plot of the null distributions is below

.



So the null distribution does depend on n. In practice, we'd have to approximate the distribution separately for various n. But we wouldn't have to worry about the true value of μ and σ.

Here is an example of how we might use this information. Suppose I have the following random sample, and I want to know if it came from a normal distribution (they didn't by the way).

```
> X

 [1]  2.70  6.42  4.95  6.96  2.82 17.28  7.69  3.23  7.15  1.42  9.99  4.27
```

Running the R `ks.test` function gives the following:

```
> ks.test(X, pnorm, mean(X), sd(X))

One-sample Kolmogorov-Smirnov test

data:  X
D = 0.20072, p-value = 0.6489
alternative hypothesis: two-sided
```

The observed test statistic is about 0.2. But we can't trust that p-value. R is assuming we know the true mean and standard deviation. Here's what we can do. For the sample size here $n = 12$, I will approximate the null distribution of the KS statistic using Monte Carlo simulation. I'll use that distribution to find the p-value for the observation $D = 0.20072$.

```
M=20000
n=12
mu=0
sigma=1
KSest=rep(NA, M)
for (i in 1:M)
{
  data=rnorm(n, mean=mu, sd=sigma)
  KSest[i]=ks.test(data, pnorm, mean(data), sd(data))$statistic
}
f12=density(KSest)
plot(f12$x, f12$y, type='l')
```

The plot really says it all. The p-value is $P(D > 0.2) \approx 0$. Very different than the conclusion we would reach if we just trusted R!