STAT 206 Homework 4

**Due Monday, October 30, 5:00 PM**

***General instructions for homework***: Homework must be completed as an R Markdown file. Be sure to include your name in the file. Give the commands to answer each question in its own code block, which will also produce plots that will be automatically embedded in the output file. Each answer must be supported by written statements as well as any code used. (Examining your various objects in the "Environment" section of RStudio is insufficient -- you must use scripted commands.)

In lecture, we fit a gamma distribution to the weight of cats hearts. We did this by adjusting the parameters so that the theoretical values of the mean and variance matched the observed, sample mean and variance. Since the mean and variance are the first two moments of the distribution, this is an example of the method of moments for estimation.

The method of moments gives a point estimate of the parameters . To use a point estimate, we need to know how precise it is, i.e., how different it would be if we repeated the experiment with new data from the same population. We often measure imprecision by the standard error, which is the standard deviation of the point estimates . (You saw the standard error of the mean in your introductory statistics classes, but we are not computing the standard error of the mean here.)

If we actually did the experiment many times, getting many values of , we could take their standard deviation as the standard error. With only one data set, we need to do something else. There is usually no simple formula for standard errors of most estimates, the way there is for the standard error of the mean. Instead, we will see how to approximate the standard error of for our estimate of the gamma distribution computationally.

We can draw random values from a gamma distribution using the rgamma() function. For example, rgamma(n=35,shape=0.57,scale=15) would generate a vector of 35 random values, drawn from the gamma distribution with shape parameter a = 0.57 and scale s = 15. By applying the estimator to random samples drawn from the distribution, we can see how much the estimates will change purely due to noise.

# Part I - Estimates and standard errors

1. Write a function, gamma.est, which takes as input a vector of data values, and returns a vector containing the two estimated parameters of the gamma distribution, with components named shape and scale as appropriate.

library(MASS)  
data("cats", package="MASS")  
  
gamma.est = function(data) {  
 avg = mean(data)  
 var = var(data)  
 scale = var / avg  
 shape = avg / scale  
 return(c(shape = shape, scale = scale))  
   
}  
  
gamma.est(cats$Hwt)

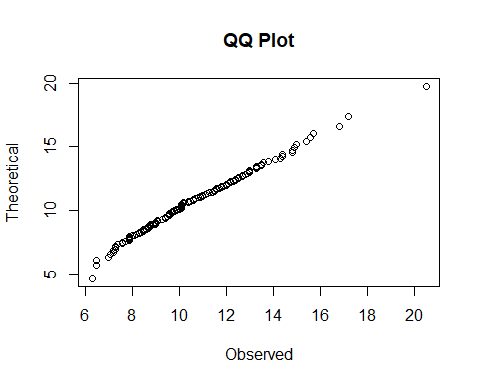
## shape scale   
## 19.0653121 0.5575862

1. Verify that your function implements the appropriate formulas by showing that it matches the results from lecture for the cat heart data.

#The following code was copied from the leecture notes. This shows that the values  
#are close to the above values. We can also generate a qqplot and look for a straight line  
  
fitdistr(cats$Hwt, densfun="gamma")

## shape rate   
## 20.2998092 1.9095724   
## ( 2.3729250) ( 0.2259942)

cats.gamma <- gamma.est(cats$Hwt)  
qqplot(cats$Hwt, qgamma(ppoints(500), shape=cats.gamma["shape"],  
 scale=cats.gamma["scale"]), xlab="Observed", ylab="Theoretical",  
 main="QQ Plot")



1. Generate a vector containing ten thousand random values from the gamma distribution with and . What are the theoretical values of the mean and of the variance? What are their sample values?

#to create a vector of 100000 random values from the gamma distribution we use rgamma  
#it is saved to a variable to we can use mean and var on that variable  
# avg = shape \* scale and var = shape \* scale^2  
#we can see the random sample values and derived values are very close  
  
rand\_gam = rgamma(seq(1,10000,1), shape = 19, scale =0.56)  
mean(rand\_gam)

## [1] 10.6569

var(rand\_gam)

## [1] 5.935505

derived\_avg = 19 \* .56  
derived\_avg

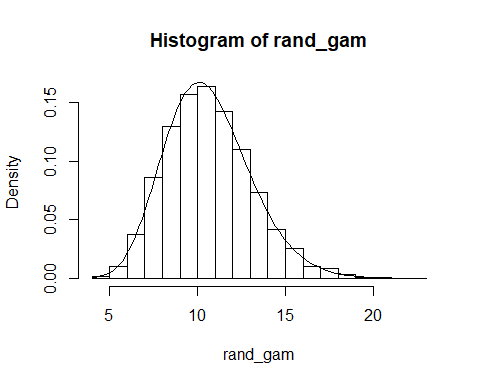
## [1] 10.64

derived\_var = 19\*(.56^2)  
derived\_var

## [1] 5.9584

1. Plot the histogram of the random values, and add the curve of the theoretical probability density function.

#To make this overlay work we need to pass the probability = TRUE argument to hist  
#if not, we will not have the correct scaling along the y axis   
#to make the curve go into the same windows as the hist we must pass the add = TRUE  
  
hist(rand\_gam, probability = TRUE)  
curve(dgamma(x, shape= 19, scale=.56), add = TRUE)



1. Apply your gamma.est function to your random sample. Report the estimated parameters and how far they are from the true values.

#the values are quite close to the actual values. First we take a look at the shape and scale   
#from the random values, then we subtract out the actual values to see how close they are  
  
gamma.est(rand\_gam)

## shape scale   
## 19.1339113 0.5569638

gamma.est(rand\_gam)[1] - gamma.est(cats$Hwt)[1]

## shape   
## 0.06859928

gamma.est(rand\_gam)[2] - gamma.est(cats$Hwt)[2]

## scale   
## -0.0006224296

1. Write a function, gamma.est.se, to calculate the standard error of your estimates of the gamma parameters, on simulated data drawn from the gamma distribution. It should take the following arguments: true shape parameter shape (or a), true scale parameter scale (or s), size of each sample n, and number of repetitions at that sample size B. It should return two standard errors, one for the shape parameter a and one for the scale parameter s. (These can be either in a vector or in a list, but should be named clearly.) It should call a function gamma.est.sim which takes the same arguments as gamma.est.se, and returns an array with two rows and B columns, one row holding shape estimates and the other row scale estimates. Your gamma.est.se function should not, itself, estimate any parameters or generate any random values

#first I want to write the gamma.est.sim function that will be called upon by the gamma.est.se function  
#then I can wrap this function with another function that will use those values to calculate   
#standard error of each row  
  
  
#set.seed(10)  
row\_1 = c()  
row\_2 = c()  
  
gamma.est.sim= function(shape, scale, n, B) {  
 raw\_data = rgamma(seq(1,B,1), shape = shape, scale = scale)  
 for(mean in raw\_data) {  
 value\_1 = mean / scale  
 row\_1 = c(row\_1, value\_1)  
 value\_2 = mean / shape  
 row\_2 = c(row\_2, value\_2)  
 }  
   
 return(rbind(row\_1,row\_2))  
   
}  
  
gamma.est.se = function(shape, scale, n, B) {  
 shape\_error = sd(gamma.est.sim(shape, scale, n, B)[1,])  
 scale\_error = sd(gamma.est.sim(shape, scale, n, B)[2,])  
 return(c(shape\_error, scale\_error))  
}  
  
gamma.est.se(19,.56,100,10000)

## [1] 4.3803847 0.1288364

# Part II - Testing with a *stub*

To check that gamma.est.se works properly, write a *stub* or *dummy* version of gamma.est.sim, which takes the correct arguments and returns an array of the proper size, but whose entries are fixed so that itâs easy for us to calculate what gamma.est.se ought to do.

1. Write gamma.est.sim so that the entries in the first row of the returned array alternate between shape and shape+1, and those in the second row alternate between scale and scale+n. For example gamma.est.sim(2,1,10,10) should return (row names are optional)

## [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]  
## shapes 2 3 2 3 2 3 2 3 2 3  
## scales 1 11 1 11 1 11 1 11 1 11

#we need to write some code that does different opetations for alternating columns  
  
gamma.est.sim = function(shape, scale, n, B) {  
 row\_1 = c()  
 row\_2 = c()  
   
 for(value in 1:B) {  
 value\_1 = value / scale  
 value\_2 = value / shape  
 if (value %% 2 == 0) {  
 row\_1 = c(row\_1, shape +1)  
 row\_2 = c(row\_2, scale + n)  
   
 }  
 else {  
 row\_1 = c(row\_1, shape)  
 row\_2 = c(row\_2, scale)   
 }  
 }  
   
return(rbind(row\_1, row\_2))  
   
}  
  
gamma.est.sim(2,1,10,10)

## [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]  
## row\_1 2 3 2 3 2 3 2 3 2 3  
## row\_2 1 11 1 11 1 11 1 11 1 11

gamma.est.sim(2,8,5,7)

## [,1] [,2] [,3] [,4] [,5] [,6] [,7]  
## row\_1 2 3 2 3 2 3 2  
## row\_2 8 13 8 13 8 13 8

and gamma.est.sim(2,8,5,7) should return

## [,1] [,2] [,3] [,4] [,5] [,6] [,7]  
## [1,] 2 3 2 3 2 3 2  
## [2,] 8 13 8 13 8 13 8

1. Calculate the standard deviations of each *row* in the two arrays above.

#I choose to use the built-in function of sd to accomplish this and using [] to extract the  
#desired rows  
  
sd(gamma.est.sim(2,1,10,10)[1,])

## [1] 0.5270463

sd(gamma.est.sim(2,1,10,10)[2,])

## [1] 5.270463

sd(gamma.est.sim(2,8,5,7)[1,])

## [1] 0.5345225

sd(gamma.est.sim(2,8,5,7)[2,])

## [1] 2.672612

1. Run your gamma.est.se, with this version of gamma.est.sim. Do its standard errors match the standard deviations you just calculated? Should they?

#The values from my function gamma.est.se match the standard deviations from the above  
  
gamma.est.se(2,1,10,10)

## [1] 0.5270463 5.2704628

gamma.est.se(2,8,5,7)

## [1] 0.5345225 2.6726124

# Part III - Replacing the stub

1. Write the actual gamma.est.sim. Each of the B columns in its output should be the result of applying gamma.est to a vector of n random numbers generated by a different call to rgamma, all with the same shape and scale parameters.

#The replicate function is ideal for this because its arguments, are the number of replications  
#which in this case we want to do this for the number of columns, then and expression  
#which we use our gamma estimate and finally we can use the rgamma to generate the random numbers  
  
gamma.est.sim = function(shape, scale, n, B) {  
 adjusted = replicate(n, gamma.est(rgamma(n, shape = shape, scale = scale)))  
 return(adjusted)  
}  
  
#gamma.est.sim(19,.56,100,10000)

1. Run gamma.est.se, calling your new gamma.est.sim, with shape=2, scale=1, n=10 and B=1e5. Check that the standard error for shape is approximately 1.6 and that for scale approximately 0.54. Explain why your answers are not exactly 1.6 and 0.54.

#Note: everytime we run this function the output is being generated off of "random" numbers generated   
#by the rgamma function. This means that each time we run this chunk we will get different numbers  
#sometimes that are close to 1.6 and .54 and sometimes not.   
  
gamma.est.se(shape=2, scale=1, n=10, B=1e5)

## [1] 1.0545956 0.5139997