Luke Beebe
Assignment 7

From the assignment 3 bootstrap, I created a **func** parameter that passes a statistical function into the bootstrap. Every place where I took a standard deviation, I called a different bootstrap program to calculate standard error of my statistic. I thought about recursively using the function it was already inside, but I'd have to complicate the code and add a confidence interval parameter so it wouldn't calculate the confidence interval for each standard error. Thus, I created a strict bootstrapping program outside the function to call, based on **bootstrap.exp** in **xy.pck**. Below is the code for both functions.

```
my.bootstrap<-function(vec0,statfunc,nboot=100){
  n0<-length(vec0)
  stat0<-statfunc(vec0)
  bootvec<-NULL
  for( i in 1:nboot){
    vecb<-sample(vec0,replace=T)
    statb<-statfunc(vecb)
    bootvec<-c(bootvec,statb)
  }
  list(stat0=stat0,bootmean=mean(bootvec),bootvar=var(bootvec))
}

my.bootstrapci <- function(func,vec0,nboot=1000,alpha){
  # creates function, taking parameters vec0, nboot, alpha
  n0<-length(vec0) #calculates n0, length of vec0
  stat0<-func(vec0) #calculates mean of vec0
  sd0<-sqrt(my.bootstrap(vec0,func)$bootvar) #caclulates standard deviation of vec0
  bootvec<-NULL #creates empty vector bootvec
  for(i in 1:nboot){ #loops amount of nboot times
    vecb<-sample(vec0,replace=T) #samples vec0, creating vecb
    statb<-func(vecb) #takes func of sample data
    sdb<-sqrt(my.bootstrap(vecb,func)$bootvar) #takes standard deviation of sample data
    while(sdb==0){ #resamples data if sd=0 to avoid dividing by 0
      vecb<-sample(vec0,replace=T)
      statb<-func(vecb)
      sdb<-sqrt(my.bootstrap(vecb,func)$bootvar)
    }
    bootvec<-c(bootvec,(statb-stat0)/(sdb/sqrt(n0))) #saves (bootvec, t-stat)
  }
  lq<-quantile(bootvec,alpha/2) #finds value of lower quantile from bootvec
  uq<-quantile(bootvec,1-alpha/2) #finds value of upper quantile from bootvec
  LB<-stat0-(sd0/sqrt(n0))*uq #creates lower bound of bootvec
  UB<-stat0-(sd0/sqrt(n0))*lq #creates upper bound of bootvec
  NLB<-stat0-(sd0/sqrt(n0))*qt(1-alpha/2,n0-1) #creates lower bound of vec0
  NUB<-stat0+(sd0/sqrt(n0))*qt(1-alpha/2,n0-1) #creates upper bound of vec0
list(bootstrap.ci=c(LB,UB),bootstrap.ci.range=UB-LB,bootstrap.var=var(bootvec),normal.ci=c(NLB,NUB))
}
```

From here, I used **my.bootstrapci** twice to compare the ranges of the confidence intervals from using the median and mean functions. While comparing them I saw the median function provided a smaller range, but I wasn't completely convinced off of one trial. So, I ran a loop comparing the range of each of them 100 times and tallied **median.count** or **mean.count** depending on which range was smaller. Below is the code and results of the n=1 trial and the n=100 trial.

```
> my.bootstrapci(median, special.sample, alpha=0.05)
$bootstrap.ci
   97.5%     2.5%
1.934706 5.743800

$bootstrap.ci.range
    2.5%
3.809094

$bootstrap.var
[1] 139.7383

$normal.ci
[1] 3.390798 3.716006

> my.bootstrapci(mean, special.sample, alpha=0.05)
$bootstrap.ci
    97.5%      2.5%
0.3974764 5.3724844

$bootstrap.ci.range
    2.5%
4.975008

$bootstrap.var
[1] 108.9485

$normal.ci
[1] 2.635480 3.129183
```

```
for(i in 1:100){
  if(my.bootstrapci(median,special.sample,alpha=0.05)$bootstrap.ci.range<my.bootstrapci(mean,special.sample,alpha=0.05)$bootstrap.ci.range){
    median.count=median.count+1
  }else(mean.count=mean.count+1)
}
```

```
> print(median.count)
[1] 75
> print(mean.count)
[1] 25
```

From our n=100 trial, bootstrapping while using the median function provided slimmer confidence interval margins than bootstrapping while using the mean function 75% of the time.