850 Homework 4

Yifang Zhang

Q1. Consider the Boston housing data set, from the MASS library.

(a). Provide an estimate for the population mean of medv, $\hat{\mu}$

```
library(MASS)
data("Boston")
#summary(Boston)
muhat=mean(Boston$medv)
muhat
```

```
## [1] 22.53281
```

We can use the sample mean as an estimate for the population mean of medv, and according to the result, we get $\hat{\mu}$ = 22.5328.

(b). Provide an estimate of the standard error of $\hat{\mu}$

```
se_est=sd(Boston$medv)/sqrt(nrow(Boston))
se_est
```

```
## [1] 0.4088611
```

Usually, we can the following formula to provide an estimate of the standard error of the sample mean:

$$SE_{\hat{\mu}} = \sqrt{\frac{s^2}{n}}$$

where s^2 is the sample variance and n is the sample size. And according to the result, we get $SE_{\hat{\mu}}$ = 0.4089.

(c). Estimate the standard error of $\hat{\mu}$ using the bootstrap

```
set.seed(1202)
n=length(Boston$medv)
B=5000
mu_boot=numeric(B)
for (i in 1:B){
  s=sample(Boston$medv,size=n,replace = TRUE)
  mu_boot[i]=mean(s)
}
sd(mu_boot)
```

```
## [1] 0.4120066
```

* Here in Rmarkdown, the output is 0.4063132.

By using the bootstrap, we get $SE^*_{\hat{\mu}}$ = 0.4063, which is quite close to the estimated standard error we obtained in part (b).

(d). Provide a 95% CI for the mean of medv based on bootstrap estimate

```
lower_boot=muhat-2*sd(mu_boot)
upper_boot=muhat+2*sd(mu_boot)
print(paste("95% CI for the mean of medv by bootstrap is (",
            round(lower_boot,4),",",round(upper_boot,4),")"))
```

```
## [1] "95% CI for the mean of medv by bootstrap is ( 21.7088 , 23.3568 )"
```

```
t.test(Boston$medv)                    # Here original output is (21.7202, 23.3454).
```

```
##
##  One Sample t-test
##
## data:  Boston$medv
## t = 55.111, df = 505, p-value < 2.2e-16
## alternative hypothesis: true mean is not equal to 0
## 95 percent confidence interval:
##  21.72953 23.33608
## sample estimates:
## mean of x
##  22.53281
```

Based on our bootstrap estimate, the 95% confidence interval for the mean of medv is $(21.7202, 23.3454)$. And the result for t-test is $(21.7295, 23.3361)$. So we could say that these two results are almost the same.

(e). Provide an estimate, $\hat{\mu_{med}}$

```
med_est=median(Boston$medv)
med_est
```

```
## [1] 21.2
```

Again, we use the sample median to estimate the median value of medv in the population, and we get $\hat{\mu}_{med} = 21.2$.

(f). Estimate the standard error of the median using bootstrap

```
set.seed(1202)
n=length(Boston$medv)
B=5000
median_boot=numeric(B)
for (j in 1:B){
  s=sample(Boston$medv,size=n,replace = TRUE)
  median_boot[j]=median(s)
}
sd(median_boot)
```

```
## [1] 0.3782927
```

By using the bootstrap, we find the standard error of $\hat{\mu}_{med}$ is 0.3799, which is smaller comparing to the standard error of the estimated mean $\hat{\mu}$.

(g). Provide an estimate for the tenth percentile of medv

```
tenth_est=quantile(Boston$medv,p=0.1)
tenth_est
```

```
##   10%
## 12.75
```

The estimated for the tenth percentile of medv is $\hat{\mu}_{0.1} = 12.75$.

(h). Estimate the standard error of $\hat{\mu_{0.1}}$ using bootstrap

```
set.seed(1202)
n=length(Boston$medv)
B=5000
tenth_boot=numeric(B)
for (k in 1:B){
  s=sample(Boston$medv,size = n, replace = TRUE)
  tenth_boot[k]=quantile(s,p=0.1)
}
sd(tenth_boot)
```

```
## [1] 0.496037   # Original output in Rmarkdown is 0.5038881
```

By using the bootstrap, we get the standard error of $\hat{\mu_{0.1}}$ is 0.5037. And if we compare this standard error with other two, we can find it has the largest value among all three standard errors. But we can also notice that all standard errors are quite small.

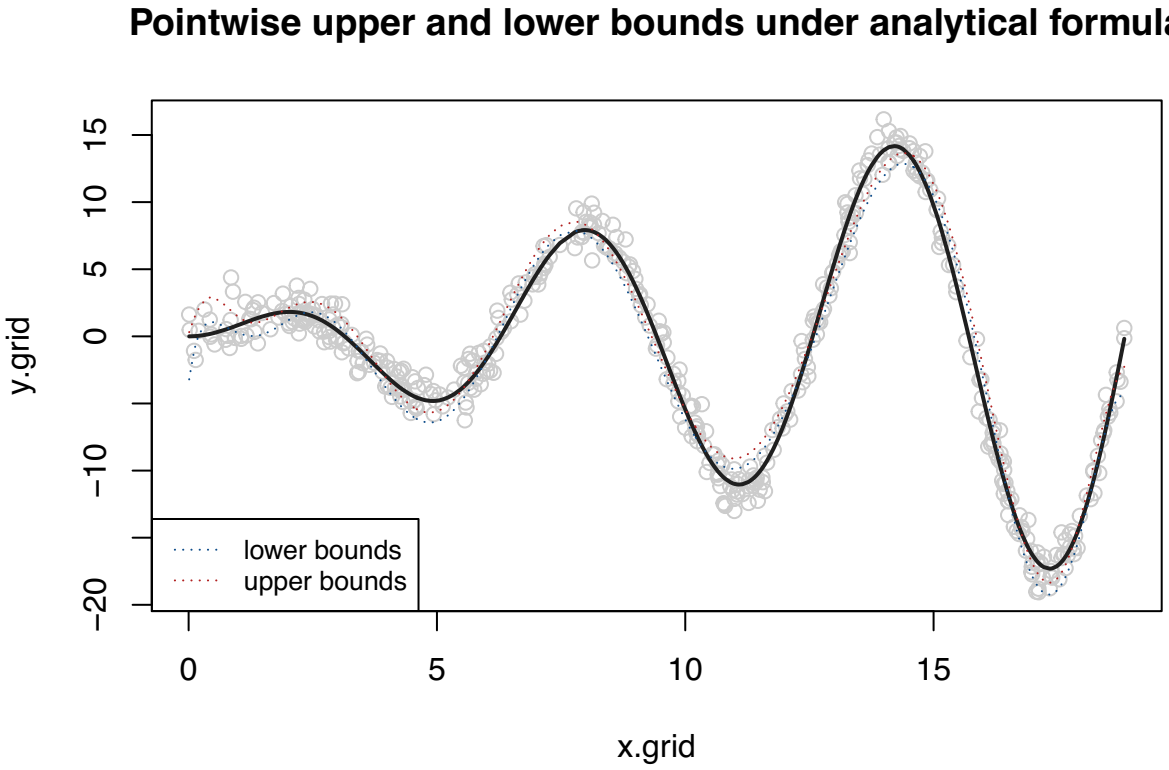
Q2. Investigate the performance of different bootstrap confidence bands.

```
### Create a simple nonlinear example
set.seed(0)
n=500
x=sort(runif(n,0,6*pi))
r=x*sin(x)
y=r+rnorm(n)

### Regression splines
library(splines)
knots=seq(2,16,length=6)
G=bs(x,knots=knots,degree=5,intercept = TRUE)
fit.bs=lm(y~G-1)
summary(fit.bs)
beta.bs=as.matrix(fit.bs$coefficients)

set.seed(1007)
x.grid=sort(runif(n,0,6*pi))
r.grid=x.grid*sin(x.grid)
y.grid=x.grid*sin(x.grid)+rnorm(n)
G.grid=bs(x.grid,knots=knots,degree=5, intercept = TRUE)
ypred.grid=G.grid%*%beta.bs
```

First of all, above is a grid of x values I generated which will be used in the following questions. And since we have setted intercept is true when we constrcut basis functions for x, in the lm() code, I asked no more intercepte.

(a). Pointwise upper and lower bounds

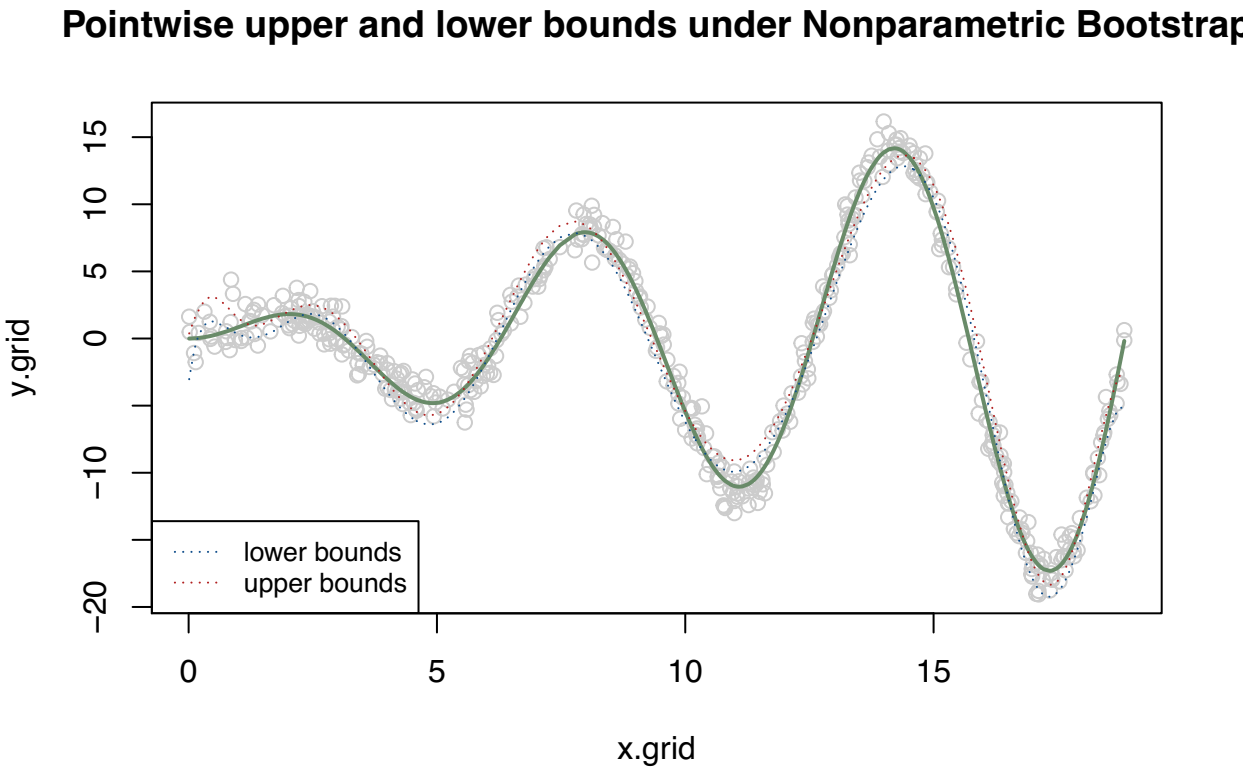
```
sigma_xy=summary(fit.bs)$sigma
M=G.grid%*%solve(t(G)%*%G)%*%t(G.grid)
diagM=diag(M)
pointse.grid=sqrt(diagM)*sigma_xy
lowerbound.bs=ypred.grid-1.96*pointse.grid
upperbound.bs=ypred.grid+1.96*pointse.grid
plot(x.grid,y.grid,col="gray80",
     main = "Pointwise upper and lower bounds under analytical formula")
lines(x.grid,r.grid,col="gray12",lwd=2)
lines(x.grid,lowerbound.bs,lty=3,col="dodgerblue4")
lines(x.grid,upperbound.bs,lty=3,col="firebrick")
legend("bottomleft",col=c("dodgerblue4","firebrick"), lty = c(3,3),
       cex=0.8, legend = c("lower bounds","upper bounds"))
```



(b). Pointwise upper and lower bounds under nonparametric bootstrap

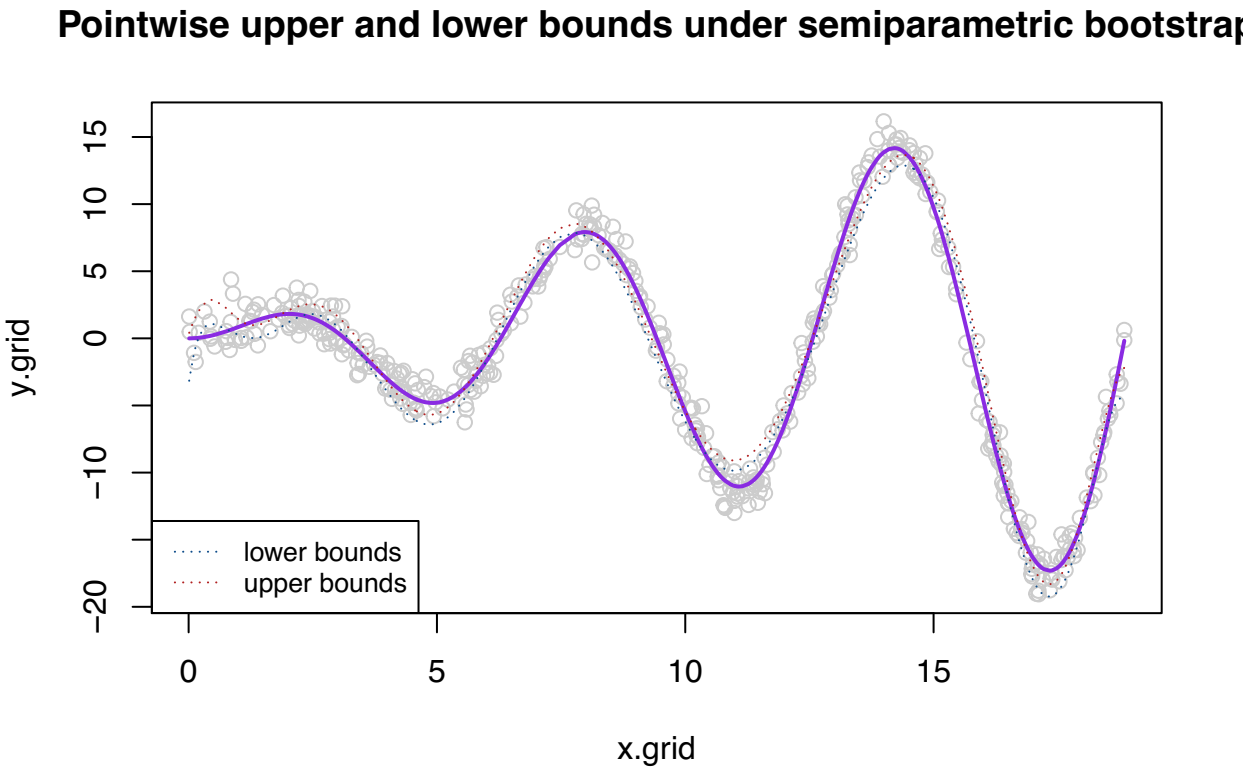
```
Dnonpara=data.frame(x,y)
resample.non=function(dataframe){
  n=nrow(dataframe)
  resamplerow=sample(1:n,size = n,replace = TRUE)
  return(dataframe[resamplerow,])
}
bs.nonpara.est=function(data,grid){
  y=data[,2]
  x=data[,1]
  fit=lm(y~bs(x,knots = knots,degree=5,intercept = TRUE)-1)
  beta=as.matrix(fit$coefficients)
  G.grid=bs(grid,knots = knots,degree=5,intercept = TRUE)
  return(G.grid%*%beta)
}

B=1000
bootResult=matrix(0,nrow=n,ncol=B)
for (i in 1:B){
  data=resample.non(Dnonpara)
  bootResult[,i]=as.vector(bs.nonpara.est(data,x.grid))
}
ypred.nonpara=as.vector(bs.nonpara.est(Dnonpara,x.grid))
lowerbound.nonpara=apply(bootResult,1,quantile,0.025)
upperbound.nonpara=apply(bootResult,1,quantile,0.975)
plot(x.grid,y.grid,col="gray80",
     main = "Pointwise upper and lower bounds under Nonparametric Bootstrap")
lines(x.grid,r.grid,col="darkseagreen4",lwd=2)
```

```
lines(x.grid,lowerbound.nonpara,lty=3,col="dodgerblue4")
lines(x.grid,upperbound.nonpara,lty=3,col="firebrick")
legend("bottomleft",col=c("dodgerblue4","firebrick"), lty = c(3,3),
       cex=0.8, legend = c("lower bounds","upper bounds"))
```



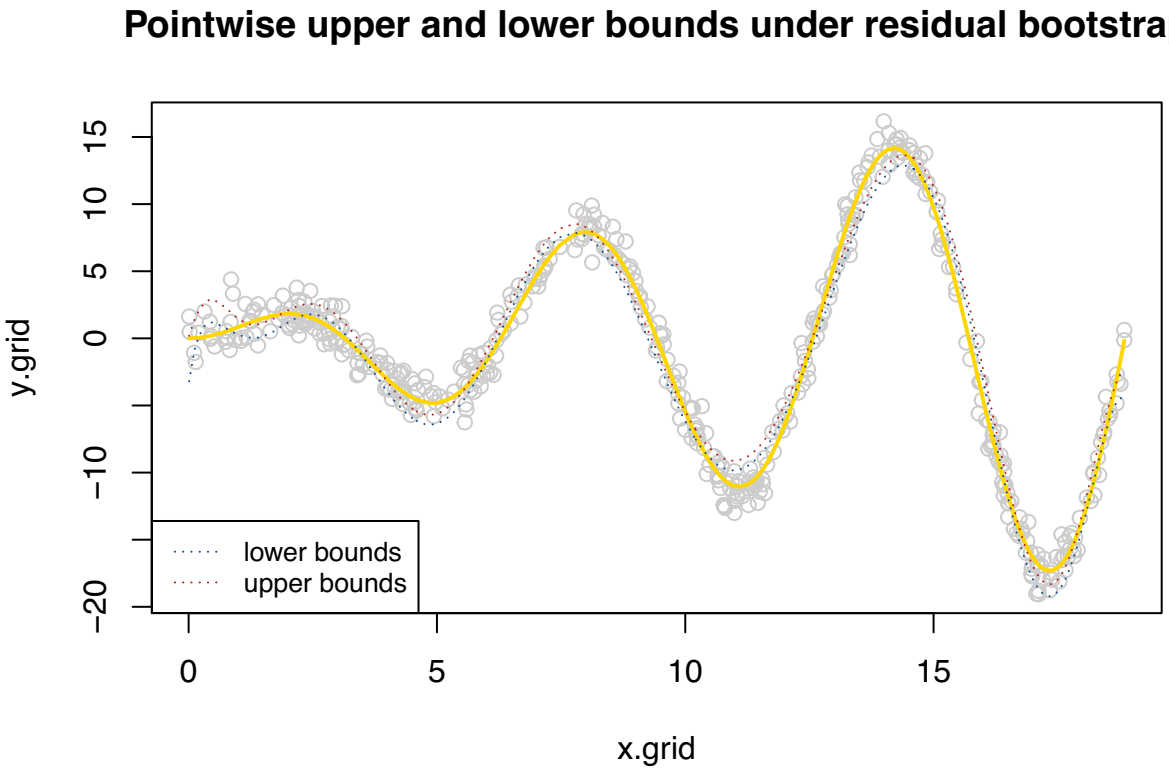
(c). Pointwise upper and lower bounds under semiparametric bootstrap

```
B=1000
bootResult.semi=matrix(0,nrow=n,ncol=B)
for (j in 1:B){
  ynew=predict(fit.bs)+rnorm(n,0,sigma_xy)
  refit=lm(ynew~bs(x,knots = knots,degree=5,intercept = TRUE)-1)
  betanew=as.matrix(refit$coefficients)
  G.grid=bs(x.grid,knots = knots,degree=5,intercept = TRUE)
  bootResult.semi[,j]=as.vector(G.grid%*%betanew)
}
lowerbound.semi=apply(bootResult.semi,1,quantile,probs=0.025)
upperbound.semi=apply(bootResult.semi,1,quantile,probs=0.975)
plot(x.grid,y.grid,col="gray80",
     main = "Pointwise upper and lower bounds under semiparametric bootstrap")
lines(x.grid,r.grid,col="blueviolet",lwd=2)
lines(x.grid,lowerbound.semi,lty=3,col="dodgerblue4")
lines(x.grid,upperbound.semi,lty=3,col="firebrick")
legend("bottomleft",col=c("dodgerblue4","firebrick"), lty = c(3,3),
       cex=0.8, legend = c("lower bounds","upper bounds"))
```



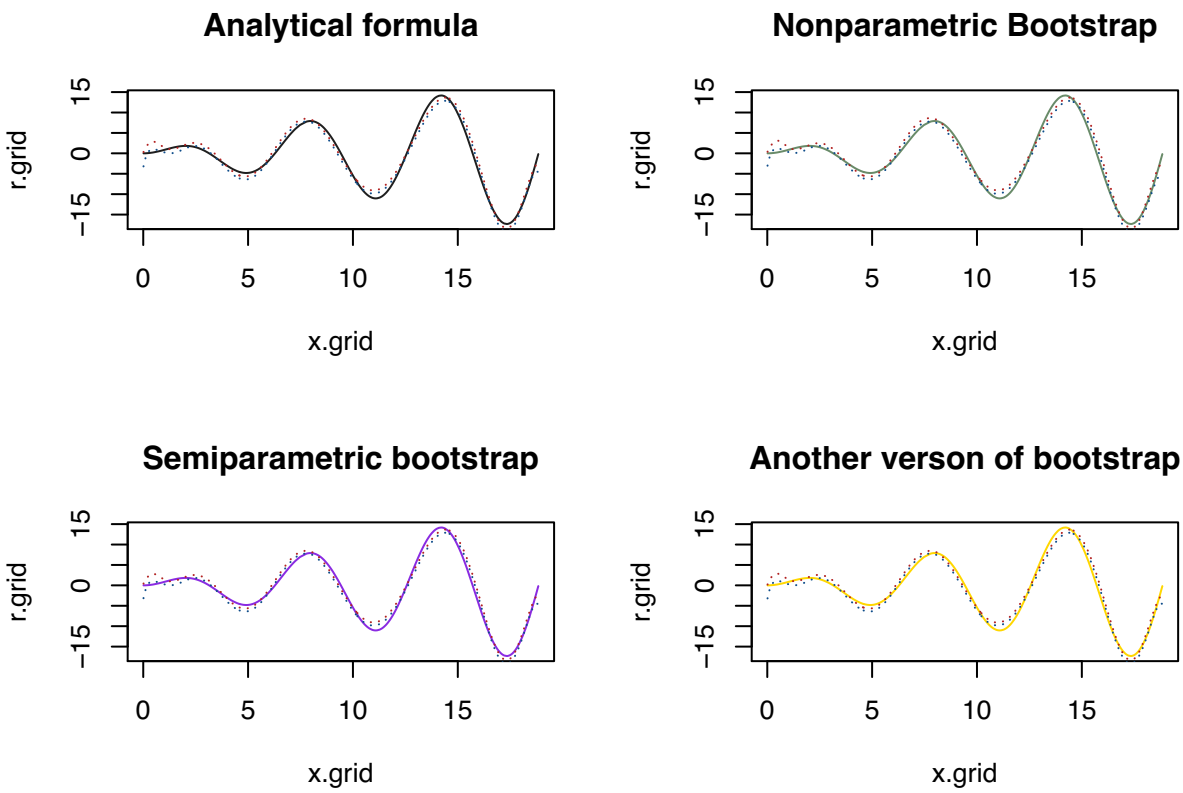
(d). Pointwise upper and lower bounds under another verson of bootstrap

```
res.boot=fit.bs$residuals
center.res=scale(res.boot,center = TRUE,scale = FALSE)
B=1000
bootResult.res=matrix(0,nrow=n,ncol=B)
for (j in 1:B){
  ynew=predict(fit.bs)+rnorm(n,0,sd(center.res))
  refit=lm(ynew~bs(x,knots = knots,degree=5,intercept = TRUE)-1)
  betanew=as.matrix(refit$coefficients)
  G.grid=bs(x.grid,knots = knots,degree=5,intercept = TRUE)
  bootResult.res[,j]=as.vector(G.grid%*%betanew)
}
lowerbound.res=apply(bootResult.res,1,quantile,probs=0.025)
upperbound.res=apply(bootResult.res,1,quantile,probs=0.975)
plot(x.grid,y.grid,col="gray80",
     main = "Pointwise upper and lower bounds under residual bootstrap")
lines(x.grid,r.grid,col="gold",lwd=2)
lines(x.grid,lowerbound.res,lty=3,col="dodgerblue4")
lines(x.grid,upperbound.res,lty=3,col="firebrick")
legend("bottomleft",col=c("dodgerblue4","firebrick"), lty = c(3,3),
       cex=0.8, legend = c("lower bounds","upper bounds"))
```



(e). Comparision

```
par(mfrow=c(2,2))
plot(x.grid,r.grid,col="gray12", type="l",
     main = "Analytical formula")
lines(x.grid,lowerbound.bs,lty=3,col="dodgerblue4")
lines(x.grid,upperbound.bs,lty=3,col="firebrick")

plot(x.grid,r.grid,col="darkseagreen4",type="l",
     main = "Nonparametric Bootstrap")
lines(x.grid,lowerbound.nonpara,lty=3,col="dodgerblue4")
lines(x.grid,upperbound.nonpara,lty=3,col="firebrick")

plot(x.grid,r.grid,col="blueviolet",type="l",
     main = "Semiparametric bootstrap")
lines(x.grid,lowerbound.semi,lty=3,col="dodgerblue4")
lines(x.grid,upperbound.semi,lty=3,col="firebrick")

plot(x.grid,r.grid,col="gold",type="l",
     main = "Another verson of bootstrap")
lines(x.grid,lowerbound.res,lty=3,col="dodgerblue4")
lines(x.grid,upperbound.res,lty=3,col="firebrick")
```



First, we can find there is no huge difference between analytical confidence band and other three bootstrap confidence bands.

Second, by simply using the absoult difference between the analytical confidence band and each bootstrap confidence band we can find that, semiparametric bootstrap and another version bootstrap which changes the Gaussian noise in semiparametric by the empirical distribution of the centered residuals are more closest to the analytical confidence band comparing to nonparametric bootstrap confidence band.

Finally, I consider that the bootstrap method which used the empirical distribution of the centered residuals is the closest to the analytical confidence band by checking the width of the 95% confidence interval of each value.

Q3. ESL Ex.8.4

First note that:

$$\hat{f(x)} = h(x)^T\hat{\beta} = h(x)^T(H^TH)^{-1}H^Ty$$

where $h(x) = (h_1(x), \cdots, h_M(x))$.

Consider the k-th bootstrap sample:

$$(x_1, \hat{f}(x_1) + \epsilon_1^{*k}), \cdots, (x_n, \hat{f}(x_n) + \epsilon_n^{*k})$$

Then we have:

$$\hat{f}^{*k}(x) = h(x)^T(H^TH)^{-1}H^T\begin{bmatrix}\hat{f}(x_1) + \epsilon_1^{*k}\\ \vdots \\ \hat{f}(x_n) + \epsilon_n^{*k}\end{bmatrix}$$

where

$$\begin{bmatrix}\hat{f}(x_1)\\ \vdots \\ \hat{f}(x_n)\end{bmatrix} = \hat{y} = H\hat{\beta} = H(H^TH)^{-1}H^Ty$$

So we have:

$$\begin{aligned}\hat{f}^{*k}(x) &= h(x)^T(H^TH)^{-1}H^T(H(H^TH)^{-1}H^Ty + \epsilon^{*k})\\ &= h(x)^T(H^TH)^{-1}H^Ty + h(x)^T(H^TH)^{-1}H^T\epsilon^{*k}\\ &= \hat{f}(x) + h(x)^T(H^TH)^{-1}H^T\epsilon^{*k}\end{aligned}$$

Follow bagging PPT slides p.6, we get:

$$\begin{aligned}\hat{f}_{bag}(x) &= \frac{1}{B}\sum_{k=1}^{B}\hat{f}^{*k}\\ &= \hat{f}(x) + h(x)^T(H^TH)^{-1}H^T(\frac{1}{B}\sum_{k=1}^{B}\epsilon^{*k})\end{aligned}$$

where $\epsilon^{*k} \sim N(0, \hat{\sigma}^2 I)$. So when $B \to \infty$, we have $\frac{1}{B}\sum_{k=1}^{B}\epsilon^{*k} \to 0$.

Hence we get:

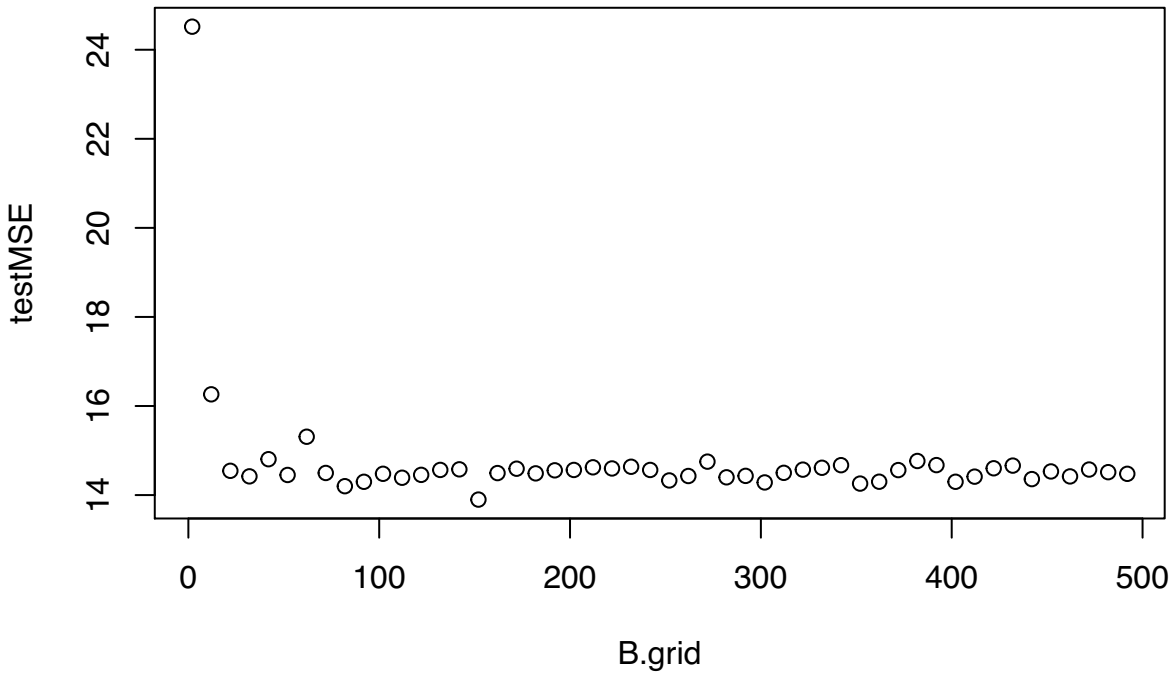
$$\hat{f}_{bag}(x) \to \hat{f}(x) \text{ as } B \to \infty$$

Q4. Investigate the effect of B on the test error of regression problems.

```
library(MASS)
library(tree)
data(Boston)
set.seed(10)
train=sample(1:nrow(Boston),0.5*nrow(Boston))
Boston.train=Boston[train,]
Boston.test=Boston[-train,]

baggingfunction=function(grid,train,test){
  t=length(grid)
  MSE=numeric(t)
  for (i in 1:t){
    fhat=matrix(NA,nrow=nrow(test),ncol=grid[i])
    for (j in 1:grid[i]){
      boot.sample=sample(1:nrow(train),nrow(train),replace = TRUE)
      Boston.boot=train[boot.sample,]
      tree.boston=tree(medv~.,Boston.boot)
      fhat[,j]=predict(tree.boston, newdata=test)
    }
    MSE[i]=mean((test$medv-apply(fhat,1,mean))^2)
  }
  return(MSE)
}

testMSE.100=baggingfunction(c(100),Boston.train,Boston.test)
testMSE.100
```

```
## [1] 14.53424
```

```
B.grid=seq(2,500,by=10)
testMSE=baggingfunction(B.grid,Boston.train,Boston.test)
plot(B.grid,testMSE,type="p")
```



When B=100: When we set the number of trees B = 100, the test MSE is about 14.53;

Various different B values: Then we created a grid of B values from 2 to 500 and plot the test MSE. We can find that as B increasing, the test MSE basically fluctuated between 14 and 15. So we can say that after some large number ($B \geq 300$), the test MSE basically stablizes.

Note:
When I used set.seed(1) in R, the testMSE.100 is about 16.54, and after using the same B.grid, we can find the test MSE stabilizes between 16 and 16.5 when B is larger then 300. But when I put the code in Rmarkdown and knit in to pdf, the result changed to 27. After several trails, finally I used set.seed(10) in Rmarkdown, then output in Rmarkdown is the same when it knits to pdf.