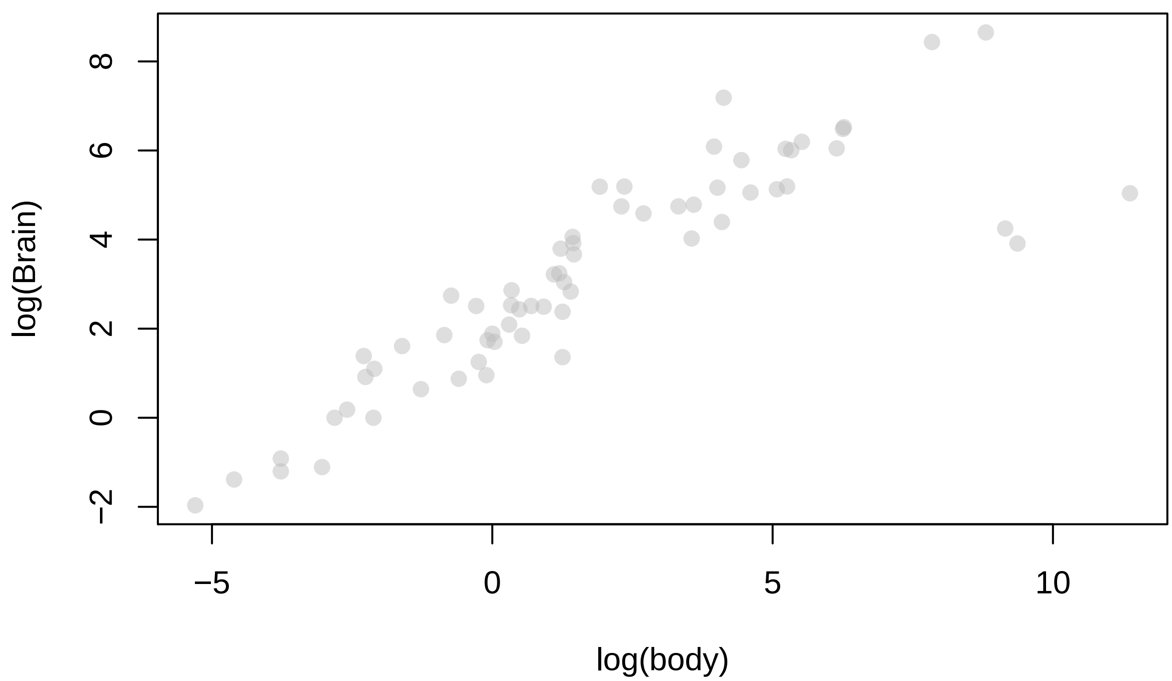
STAT 341 - Assignment 4

Due Tuesday Dec 3 at 9am - to be submitted through crowdmark

Bootstrap for Robust Regression

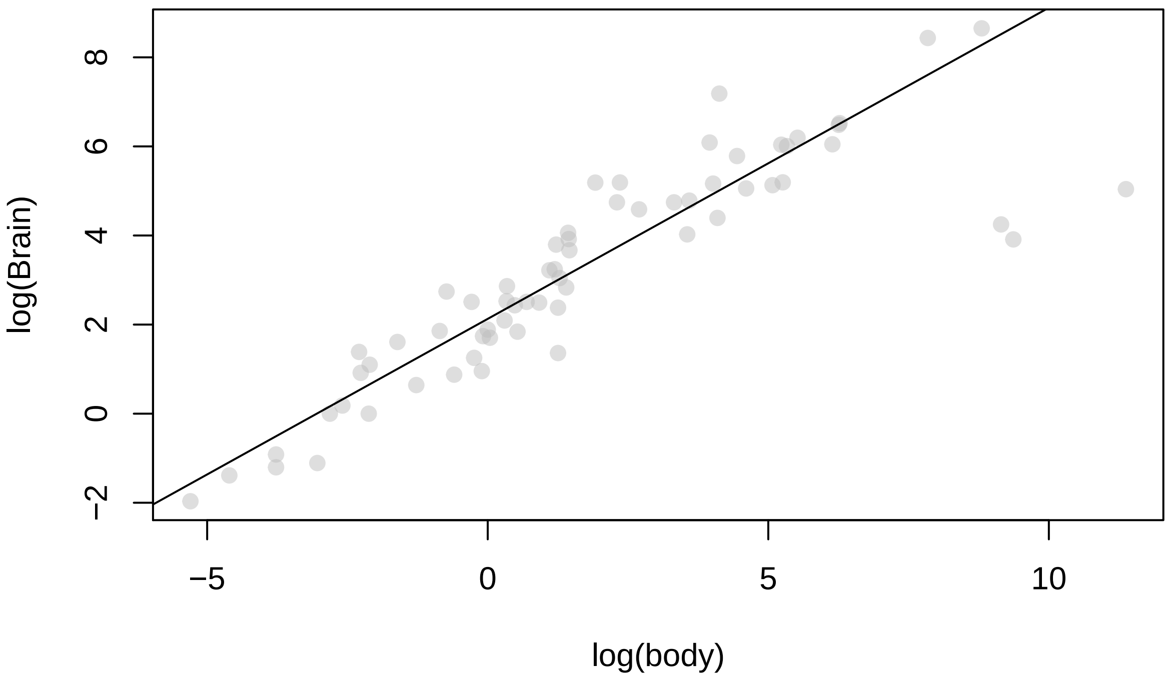
Suppose that the Animals Data is a sample and we are interested in a linear relationship between log(Brain) and log(Body).

```
library(MASS)
data(Animals2, package="robustbase")

plot(log(Animals2$body), log(Animals2$brain),
     pch=19, col=adjustcolor("grey", .5),
     xlab="log(body)", ylab="log(Brain)")
```



a) [**3 Marks**] Fit a linear function to the given sample using robust regression and the huber function.

```
fit = rlm(log(Animals2$brain) ~ log(Animals2$body))

plot(log(Animals2$body), log(Animals2$brain),
     pch=19, col=adjustcolor("grey", .5),
     xlab="log(body)", ylab="log(Brain)")
abline(fit$coef)
```

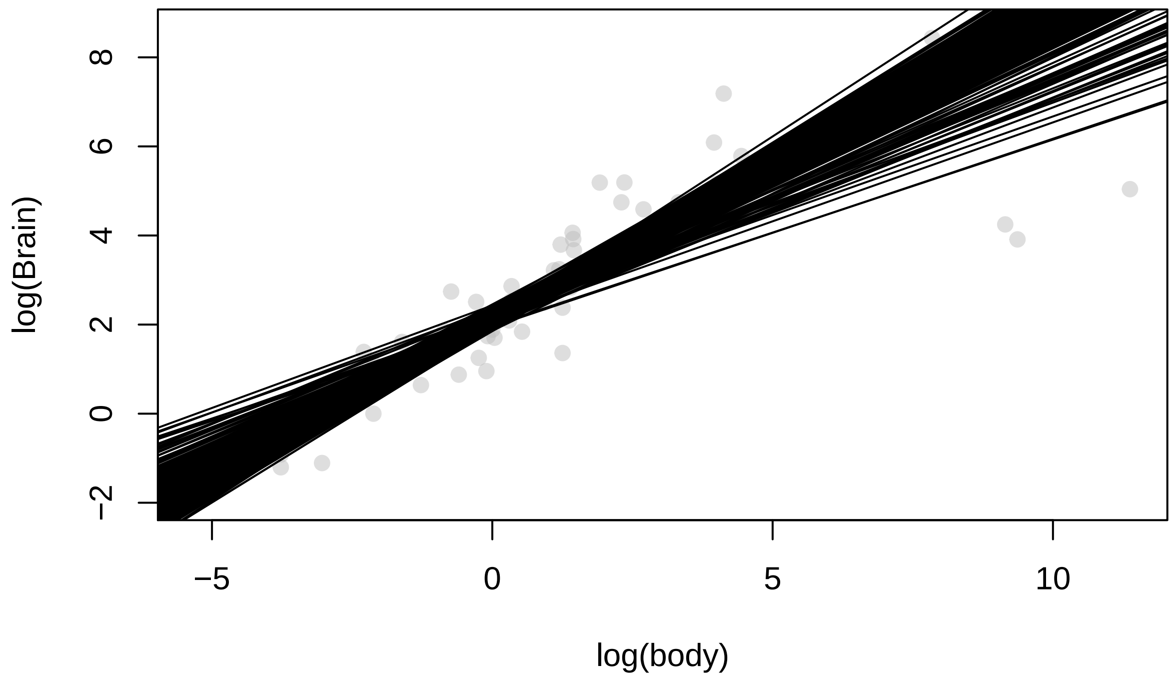


b) Using $B = 1000$ bootstrap samples by sampling the pairs and fit a linear function to each sample using robust regression and the huber function.

```
x = log(Animals2$body)
y = log(Animals2$brain)
B = 1000;
n = nrow(Animals2)
beta.boot = t(sapply(1:B, FUN =function(b)
  rlm(y~x, subset=sample(n,n, replace=TRUE), maxit = 600, psi = "psi.huber")$coef))
```

```
i) **[3 Marks]** Generate a scatterplot of the data along with all the robust regression lines using sha
```

```
plot(log(Animals2$body), log(Animals2$brain),
     pch=19, col=adjustcolor("grey", .5),
     xlab="log(body)", ylab="log(Brain)")

for(i in 1:B) {
  abline(beta.boot[i,])
}
```




```
ii) **[2 Marks]** Obtain a 99\% confidence interval (using the percentile method) for the robust regres
```

```
x0.1 = -1
mu0.star.hat = apply(beta.boot, 1, function(z,a) { sum(z*a) }, a=c( 1, x0.1))
boot.ci.1 = quantile(mu0.star.hat, prob= c(0.005, 0.995))
boot.ci.1
```

```
##     0.5%    99.5%
## 1.141385 1.786146
```

A 99% ci is (1.186826, 1.830306). iii) [**2 Marks**] Obtain a 99% confidence interval (using the percentile method) for the robust regression line when $x = 10$.

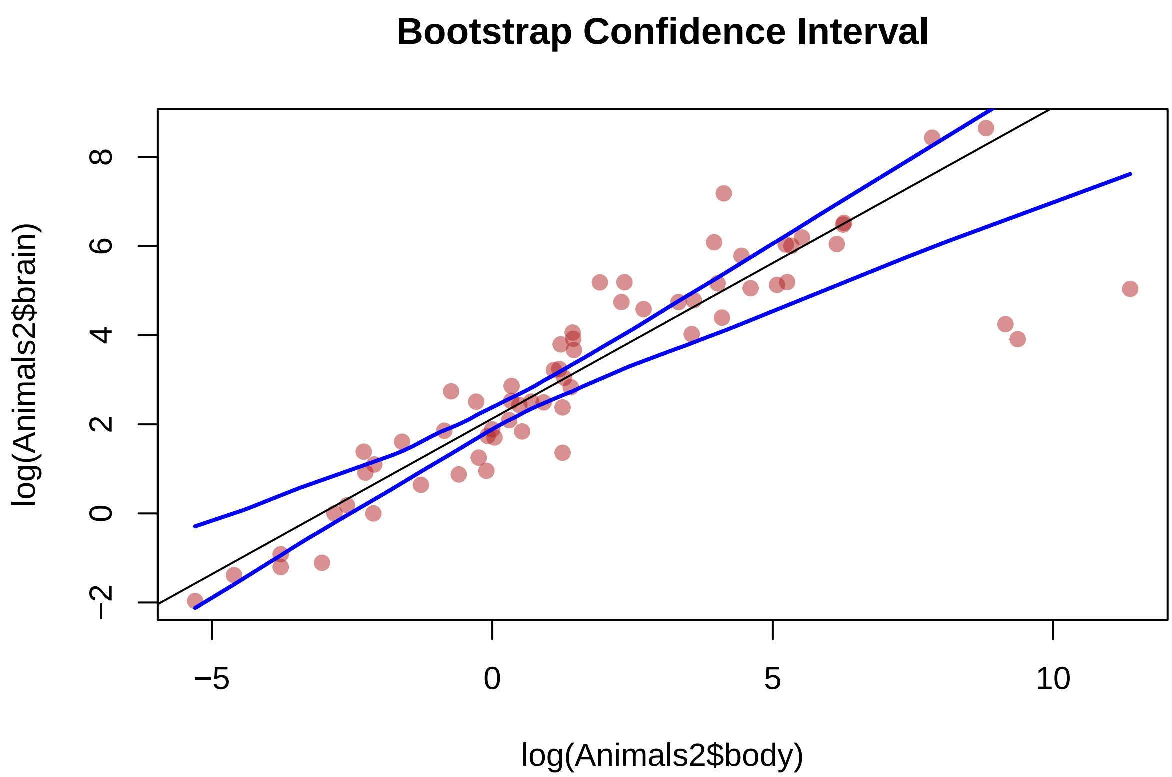
```
x0.2 = 10
mu0.star.hat2 = apply(beta.boot, 1, function(z,a) { sum(z*a) }, a=c( 1, x0.2))
boot.ci.2 = quantile(mu0.star.hat2, prob= c(0.005, 0.995))
boot.ci.2
```

```
##     0.5%    99.5%
## 6.981901 9.920813
```

A 99% ci is (6.791317, 9.837687) iv) [**3 Marks**] Obtain a 99% confidence interval (using the percentile method) for the robust regression line.

```
x.seq = seq(min(log(Animals2$body)), max(log(Animals2$body)), length.out=100)
boot.ci.3 = matrix(0, nrow=length(x.seq), 2)
for (i in 1:length(x.seq)) {
y.hat = apply(beta.boot, 1, function(z,a) { sum(z*a) }, a=c( 1, x.seq[i]))
boot.ci.3[i,] = quantile( y.hat, prob= c(.005, .995))
}
```

```r
plot(log(Animals2$body), log(Animals2$brain), pch=19, col=adjustcolor("firebrick", 0.5),
main="Bootstrap Confidence Interval")
abline(fit$coef)
lines(x.seq, boot.ci.3[,1], col=4, lwd=2)
lines(x.seq, boot.ci.3[,2], col=4, lwd=2)
```



c) **[10 Marks]** Repeat part b) by resampling the errors to generate the bootstrap samples.

```r
summary(fit)
```

```
##
## Call: rlm(formula = log(Animals2$brain) ~ log(Animals2$body))
## Residuals:
##      Min       1Q   Median       3Q      Max
## -5.03298 -0.54348 -0.01343  0.49407  2.17426
##
## Coefficients:
##                      Value   Std. Error t value
## (Intercept)           2.1281  0.1064    20.0080
## log(Animals2$body)   0.6985  0.0270    25.8464
##
## Residual standard error: 0.7763 on 63 degrees of freedom
```

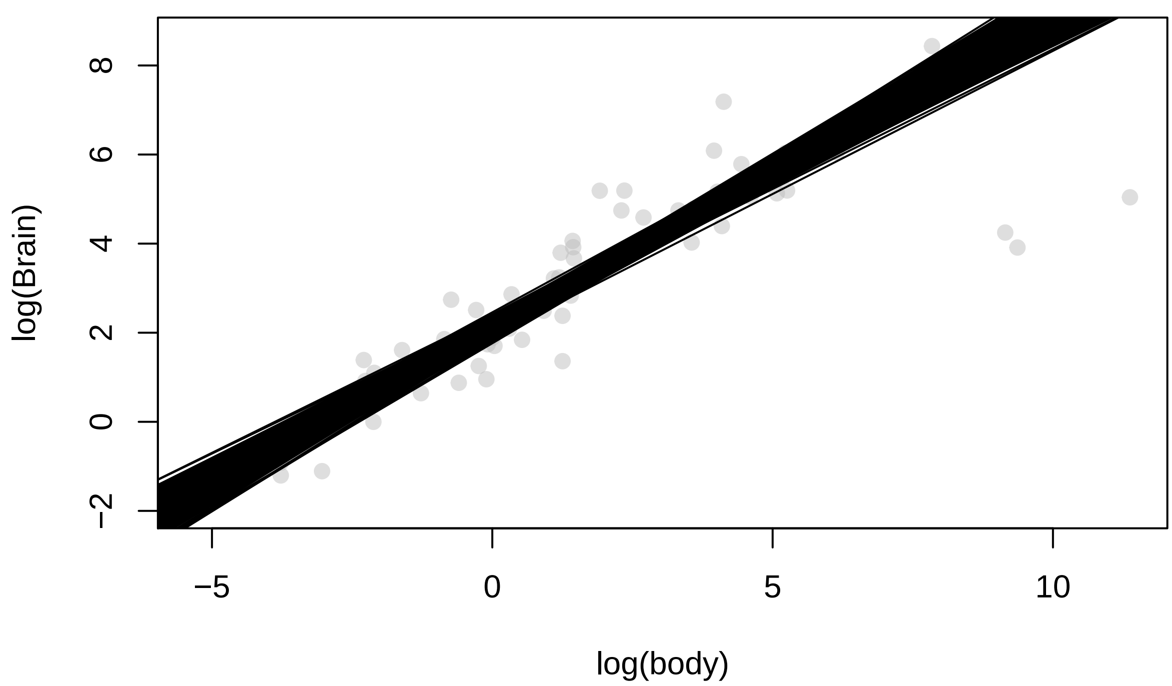
From the summary, $\hat{\alpha} = 2.1281$, $\hat{\beta} = 0.6985$ and $\hat{\sigma} = 0.7763$

```r
par.boot.sam = Map(function(b)
{ Rstar = rnorm(n, mean=0, sd= 0.7763)
```

```
y = 2.1281 + 0.6985*x + Rstar
data.frame( x = x, y= y ) } , 1:B)
par.boot.coef.l = Map(function(sam)
lm(y~x, data=sam)$coef, par.boot.sam)
```

```
par.boot.coef = matrix(1:2000, nrow = 1000, ncol = 2)
for(i in 1:1000){
  par.boot.coef[i,] = par.boot.coef.l[[i]]
}
```

```
plot(log(Animals2$body), log(Animals2$brain),
     pch=19, col=adjustcolor("grey", .5),
     xlab="log(body)", ylab="log(Brain)")

for(i in 1:B) {
  abline(par.boot.coef[i,])
}
```




```
par.mu0.star.hat = apply(par.boot.coef, 1, function(z,a) { sum(z*a) }, a=c( 1, x0.1))
par.boot.ci.1 = quantile(par.mu0.star.hat, prob= c(0.005, 0.995))
par.boot.ci.1
```

```
##     0.5%    99.5%
## 1.120433 1.735597
```

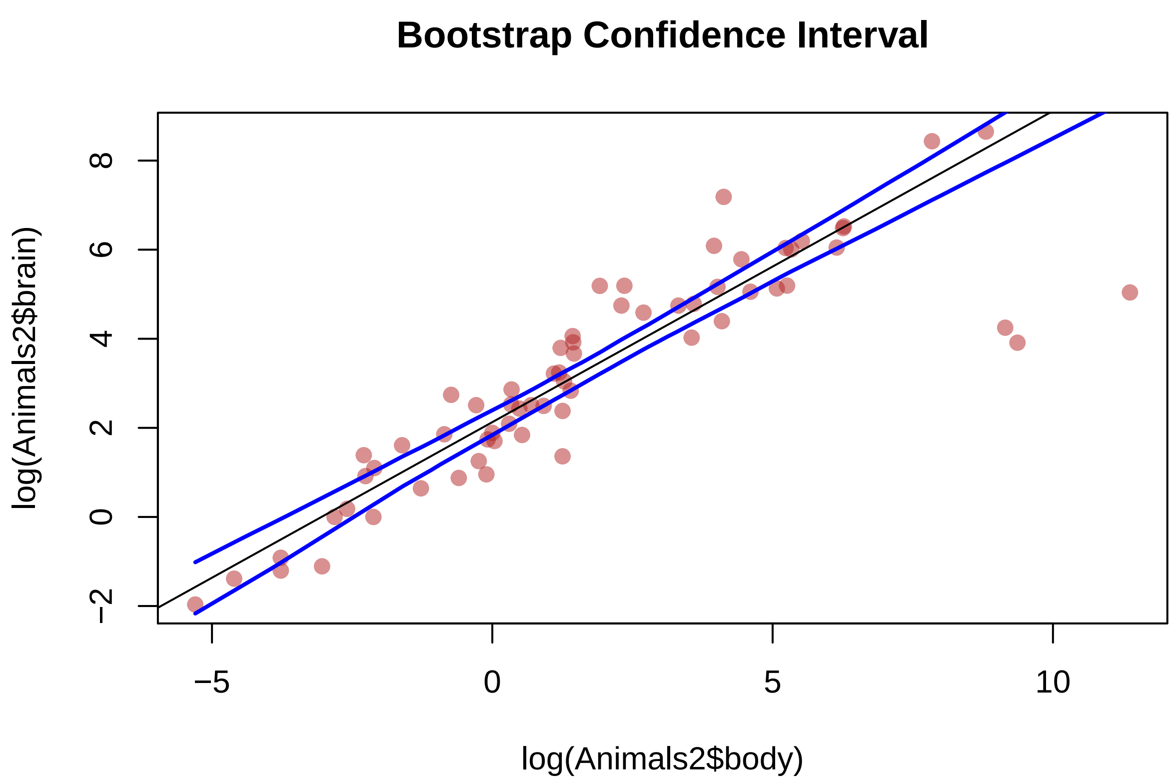
A 99% ci is (1.152408, 1.763074).

```r
x0.2 = 10
par.mu0.star.hat2 = apply(par.boot.coef, 1, function(z,a) { sum(z*a) }, a=c( 1, x0.2))
boot.ci.2 = quantile(par.mu0.star.hat2, prob= c(0.005, 0.995))
boot.ci.2
```

```
##     0.5%    99.5%
## 8.499413 9.736608
```

A 99% ci is (8.425071, 9.768483).

```r
par.boot.ci.3 = matrix(0, nrow=length(x.seq), 2)
for (i in 1:length(x.seq)) {
y.hat3 = apply(par.boot.coef, 1, function(z,a) { sum(z*a) }, a=c( 1, x.seq[i]))
par.boot.ci.3[i,] = quantile( y.hat3, prob= c(.005, .995))
}

plot(log(Animals2$body), log(Animals2$brain), pch=19, col=adjustcolor("firebrick", 0.5),
main="Bootstrap Confidence Interval")
abline(fit$coef)
lines(x.seq, par.boot.ci.3[,1], col=4, lwd=2)
lines(x.seq, par.boot.ci.3[,2], col=4, lwd=2)
```



Average prediction squared errors

The calculation of the average prediction squared error ($APSE$)

$$APSE(\mathcal{P}, \widetilde{\mu}) = \frac{1}{N_{\mathcal{S}}} \sum_{j=1}^{N_{\mathcal{S}}} \frac{1}{N} \sum_{i \in \mathcal{P}} (y_i - \widehat{\mu}_{\mathcal{S}_j}(\mathbf{x}_i))^2$$

was broken into more interpretable components. In this question, you are going to prove each step. Your notation should follow that of the notes and each simplification must be justified mathematically.

a. **[5 Marks]** Prove that

$$\begin{aligned} \tfrac{1}{N_{\mathcal{S}}} \textstyle\sum_{j=1}^{N_{\mathcal{S}}} \tfrac{1}{N} \sum_{i \in \mathcal{P}} (y_i - \widehat{\mu}_{\mathcal{S}_j}(\mathbf{x}_i))^2 \quad = \quad & \tfrac{1}{N_{\mathcal{S}}} \textstyle\sum_{j=1}^{N_{\mathcal{S}}} \tfrac{1}{N} \sum_{i \in \mathcal{P}} (y_i - \mu(\mathbf{x}_i))^2 \\ & + \tfrac{1}{N_{\mathcal{S}}} \textstyle\sum_{j=1}^{N_{\mathcal{S}}} \tfrac{1}{N} \sum_{i \in \mathcal{P}} (\widehat{\mu}_{\mathcal{S}_j}(\mathbf{x}_i) - \mu(\mathbf{x}_i))^2 \end{aligned} \tag{1}$$

Proof. We first prove $\frac{1}{N_{\mathcal{S}}} \sum_{j=1}^{N_{\mathcal{S}}} \frac{1}{N} \sum_{i \in \mathcal{P}} (y_i - \mu(\mathbf{x}))(\mu(\mathbf{x}) - \widehat{\mu}_{\mathbf{S}_j}(\mathbf{x}_i)) = 0$
Let $P = \cup_{i=1}^{k} A_i$ where $A_i = \{u : u \in P, x_u = x_i\}$ and the unique value of $\mathbf{x}$ is $\mathbf{x}_1, \ldots, \mathbf{x}_k$.
$|A_i| = n_i$ for all $1 \leq i \leq k$

$$\begin{aligned} & \frac{1}{N_{\mathcal{S}}} \sum_{j=1}^{N_{\mathcal{S}}} \frac{1}{N} \sum_{i \in \mathcal{P}} (y_i - \mu(\mathbf{x}_i))(\mu(\mathbf{x}_i) - \widehat{\mu}_{\mathcal{S}_j}(\mathbf{x}_i)) \\ = & \frac{1}{N_{\mathcal{S}}} \sum_{j=1}^{N_{\mathcal{S}}} \frac{1}{N} [\sum_{i \in \mathcal{A}_1} (y_i - \mu(\mathbf{x}_i))(\mu(\mathbf{x}_i) - \widehat{\mu}_{\mathcal{S}_j}(\mathbf{x}_i)) + \cdots + \sum_{i \in \mathcal{A}_k} (y_i - \mu(\mathbf{x}_i))(\mu(\mathbf{x}_i) - \widehat{\mu}_{\mathcal{S}_j}(\mathbf{x}_i))] \end{aligned}$$

We define $Ave_{i \in A_k} y_i = \bar{y}_{A_k}$
For $i \in A_m$ where $1 \leq m \leq k$, we have,

$$\begin{aligned} & \frac{1}{N_{\mathcal{S}}} \sum_{j=1}^{N_{\mathcal{S}}} \frac{1}{N} [\sum_{i \in A_m} y_i \bar{y}_{A_m} - y_i \widehat{\mu}_{\mathcal{S}_j}(\mathbf{x}_i) - \bar{y}_{A_m}^2 + \bar{y}_{A_m} \widehat{\mu}_{\mathcal{S}_j}(\mathbf{x}_i)] \\ = & \frac{1}{N_{\mathcal{S}}} \sum_{j=1}^{N_{\mathcal{S}}} \frac{1}{N} n_m \bar{y}_{A_m}^2 - \frac{1}{N_{\mathcal{S}} N} \sum_{i \in A_m} y_i \sum_{j=1}^{N_{\mathcal{S}}} \widehat{\mu}_{\mathcal{S}_j}(\mathbf{x}_i) - \frac{n_m}{N} \bar{y}_{A_m}^2 + \frac{1}{N_{\mathcal{S}} N} \sum_{j=1}^{N_{\mathcal{S}}} \bar{y}_{A_m} \sum_{i \in A_m} \widehat{\mu}_{\mathcal{S}_j}(\mathbf{x}_i) \\ = & \frac{n_m}{N} \bar{y}_{A_m}^2 - \frac{n_m}{N} \bar{y}_{A_m}^2 + \frac{1}{N} [\sum_{i \in A_m} \bar{y}_{A_m} \frac{1}{N_{\mathcal{S}}} \sum_{j=1}^{N_{\mathcal{S}}} \widehat{\mu}_{\mathcal{S}_j}(\mathbf{x}_i) - \sum_{i \in A_m} \bar{y}_{A_m} y_i \frac{1}{N_{\mathcal{S}}} \sum_{j=1}^{N_{\mathcal{S}}} \widehat{\mu}_{\mathcal{S}_j}(\mathbf{x}_i)] \\ = & \frac{1}{N} [\sum_{i \in A_m} \bar{y}_{A_m} \bar{\widehat{\mu}}(\mathbf{x}_i) - \sum_{i \in A_m} y_i \bar{\widehat{\mu}}(\mathbf{x}_i)] \\ & \text{for all i} \in A_m, \bar{\widehat{\mu}}(\mathbf{x}_i) \text{ is a constant say c} \\ = & \frac{\bar{y}_{A_m} \times n_m \times c}{N} - \frac{c \times n_m \times \bar{y}_{A_m}}{N} \\ = & 0 \end{aligned}$$

$\frac{1}{N_{\mathcal{S}}}\sum_{j=1}^{N_{\mathcal{S}}}\frac{1}{N}[\sum_{i\in\mathcal{A}_1}(y_i-\mu(\mathbf{x}))(\mu(\mathbf{x})-\widehat{\mu}_{\mathcal{S}_j}(\mathbf{x}_i))+\cdots+\sum_{i\in\mathcal{A}_k}(y_i-\mu(\mathbf{x}))(\mu(\mathbf{x})-\widehat{\mu}_{\mathcal{S}_j}(\mathbf{x}_i))]=0$ Hence,

$$
\begin{aligned}
&\frac{1}{N_{\mathcal{S}}}\sum_{j=1}^{N_{\mathcal{S}}}\frac{1}{N}\sum_{i\in\mathcal{P}}(y_i-\widehat{\mu}_{\mathcal{S}_j}(\mathbf{x}_i))^2\\
=&\frac{1}{N_{\mathcal{S}}}\sum_{j=1}^{N_{\mathcal{S}}}\frac{1}{N}\sum_{i\in\mathcal{P}}[(y_i-\mu(\mathbf{x}_i))-(\widehat{\mu}_{\mathcal{S}_j}(\mathbf{x}_i)-\mu(\mathbf{x}_i))]^2\\
=&\frac{1}{N_{\mathcal{S}}}\sum_{j=1}^{N_{\mathcal{S}}}\frac{1}{N}\sum_{i\in\mathcal{P}}(y_i-\mu(\mathbf{x}_i))^2-2(y_i-\mu(\mathbf{x}))(\mu(\mathbf{x})-\widehat{\mu}_{\mathcal{S}_j}(\mathbf{x}_i))+(\widehat{\mu}_{\mathcal{S}_j}(\mathbf{x}_i)-\mu(\mathbf{x}_i))^2\\
=&\frac{1}{N_{\mathcal{S}}}\sum_{j=1}^{N_{\mathcal{S}}}\frac{1}{N}\sum_{i\in\mathcal{P}}(y_i-\mu(\mathbf{x}_i))^2+\frac{1}{N_{\mathcal{S}}}\sum_{j=1}^{N_{\mathcal{S}}}\frac{1}{N}\sum_{i\in\mathcal{P}}(\widehat{\mu}_{\mathcal{S}_j}(\mathbf{x}_i)-\mu(\mathbf{x}_i))^2
\end{aligned}
$$

$\square$

b. **[5 Marks]** Prove that

$$
\begin{aligned}
\frac{1}{N_{\mathcal{S}}}\sum_{j=1}^{N_{\mathcal{S}}}\frac{1}{N}\sum_{i\in\mathcal{P}}(\widehat{\mu}_{\mathcal{S}_j}(\mathbf{x}_i)-\mu(\mathbf{x}_i))^2 \quad = \quad &\frac{1}{N_{\mathcal{S}}}\sum_{j=1}^{N_{\mathcal{S}}}\frac{1}{N}\sum_{i\in\mathcal{P}}(\widehat{\mu}_{\mathcal{S}_j}(\mathbf{x}_i)-\overline{\widehat{\mu}}(\mathbf{x}_i))^2\\
&+\frac{1}{N_{\mathcal{S}}}\sum_{j=1}^{N_{\mathcal{S}}}\frac{1}{N}\sum_{i\in\mathcal{P}}(\overline{\widehat{\mu}}(\mathbf{x}_i)-\mu(\mathbf{x}_i))^2
\end{aligned}
\tag{2}
$$

Proof. We first prove $\frac{1}{N_{\mathcal{S}}}\sum_{j=1}^{N_{\mathcal{S}}}\frac{1}{N}\sum_{i\in\mathcal{P}}(\widehat{\mu}_{\mathcal{S}_j}(\mathbf{x}_i)-\overline{\widehat{\mu}}(\mathbf{x}_i))(\overline{\widehat{\mu}}(\mathbf{x}_i)-\mu(\mathbf{x}_i))=0$.
Let $P=\cup_{i=1}^{k}A_i$ where $A_i=\{u:u\in P,x_u=x_i\}$ and the unique value of $\mathbf{x}$ is $\mathbf{x}_1,\ldots,\mathbf{x}_k$.
$|A_i|=n_i$ for all $1\leq i\leq k$

$$
\begin{aligned}
&\frac{1}{N_{\mathcal{S}}}\sum_{j=1}^{N_{\mathcal{S}}}\frac{1}{N}\sum_{i\in\mathcal{P}}(\widehat{\mu}_{\mathcal{S}_j}(\mathbf{x}_i)-\overline{\widehat{\mu}}(\mathbf{x}_i))(\overline{\widehat{\mu}}(\mathbf{x}_i)-\mu(\mathbf{x}_i))\\
=&\frac{1}{N_{\mathcal{S}}N}\sum_{j=1}^{N_{\mathcal{S}}}[\sum_{i\in A_1}(\widehat{\mu}_{\mathcal{S}_j}(\mathbf{x}_i)-\overline{\widehat{\mu}}(\mathbf{x}_i))(\overline{\widehat{\mu}}(\mathbf{x}_i)-\mu(\mathbf{x}_i))+\cdots+\sum_{i\in A_k}(\widehat{\mu}_{\mathcal{S}_j}(\mathbf{x}_i)-\overline{\widehat{\mu}}(\mathbf{x}_i))(\overline{\widehat{\mu}}(\mathbf{x}_i)-\mu(\mathbf{x}_i))]
\end{aligned}
$$

For $i\in A_m$ where $1\leq m\leq k$, we have,

$$
\begin{aligned}
&\frac{1}{N_{\mathcal{S}}N}\sum_{j=1}^{N_{\mathcal{S}}}\sum_{i\in A_m}(\widehat{\mu}_{\mathcal{S}_j}(\mathbf{x}_i)-\overline{\widehat{\mu}}(\mathbf{x}_i))(\overline{\widehat{\mu}}(\mathbf{x}_i)-\mu(\mathbf{x}_i))\\
=&\frac{1}{N_{\mathcal{S}}N}\sum_{j=1}^{N_{\mathcal{S}}}\sum_{i\in A_m}\widehat{\mu}_{\mathcal{S}_j}(\mathbf{x}_i)\overline{\widehat{\mu}}(\mathbf{x}_i)-\frac{1}{N_{\mathcal{S}}N}\sum_{j=1}^{N_{\mathcal{S}}}\sum_{i\in A_m}\widehat{\mu}_{\mathcal{S}_j}(\mathbf{x}_i)\mu(\mathbf{x}_i)\\
&-\frac{1}{N_{\mathcal{S}}N}\sum_{j=1}^{N_{\mathcal{S}}}\sum_{i\in A_m}\overline{\widehat{\mu}}(\mathbf{x}_i)^2+\frac{1}{N_{\mathcal{S}}N}\sum_{j=1}^{N_{\mathcal{S}}}\overline{\widehat{\mu}}(\mathbf{x}_i)\mu(\mathbf{x}_i)\\
=&\frac{1}{N}\sum_{i\in A_m}\frac{\overline{\widehat{\mu}}(\mathbf{x}_i)}{N_{\mathcal{S}}}\sum_{j=1}^{N_{\mathcal{S}}}\widehat{\mu}_{\mathcal{S}_j}(\mathbf{x}_i)-\frac{1}{N}\sum_{i\in A_m}\frac{\mu(\mathbf{x}_i)}{N_{\mathcal{S}}}\sum_{j=1}^{N_{\mathcal{S}}}\widehat{\mu}_{\mathcal{S}_j}(\mathbf{x}_i)-\frac{1}{N}\sum_{i\in A_m}\overline{\widehat{\mu}}(\mathbf{x}_i)^2+\frac{1}{N}\sum_{i\in A_m}\overline{\widehat{\mu}}(\mathbf{x}_i)\mu(\mathbf{x}_i)\\
=&\frac{1}{N}\sum_{i\in A_m}\overline{\widehat{\mu}}(\mathbf{x}_i)^2-\frac{1}{N}\sum_{i\in A_m}\overline{\widehat{\mu}}(\mathbf{x}_i)^2+\frac{1}{N}\sum_{i\in A_m}\overline{\widehat{\mu}}(\mathbf{x}_i)\mu(\mathbf{x}_i)-\frac{1}{N}\sum_{i\in A_m}\overline{\widehat{\mu}}(\mathbf{x}_i)\mu(\mathbf{x}_i)\\
=&0
\end{aligned}
$$

So, $\frac{1}{N_{\mathcal{S}}}\sum_{j=1}^{N_{\mathcal{S}}}\frac{1}{N}\sum_{i\in\mathcal{P}}(\widehat{\mu}_{\mathcal{S}_j}(\mathbf{x}_i)-\overline{\widehat{\mu}}(\mathbf{x}_i))(\overline{\widehat{\mu}}(\mathbf{x}_i)-\mu(\mathbf{x}_i))=0$
Hence, we have,

$$
\begin{aligned}
&\frac{1}{N_{\mathcal{S}}}\sum_{j=1}^{N_{\mathcal{S}}}\frac{1}{N}\sum_{i\in\mathcal{P}}(\widehat{\mu}_{\mathcal{S}_j}(\mathbf{x}_i)-\mu(\mathbf{x}_i))^2\\
=&\frac{1}{N_{\mathcal{S}}}\sum_{j=1}^{N_{\mathcal{S}}}\frac{1}{N}\sum_{i\in\mathcal{P}}(\widehat{\mu}_{\mathcal{S}_j}(\mathbf{x}_i)-\overline{\widehat{\mu}}(\mathbf{x}_i)+\overline{\widehat{\mu}}(\mathbf{x}_i)-\mu(\mathbf{x}_i))^2\\
=&\frac{1}{N_{\mathcal{S}}}\sum_{j=1}^{N_{\mathcal{S}}}\frac{1}{N}\sum_{i\in\mathcal{P}}(\widehat{\mu}_{\mathcal{S}_j}(\mathbf{x}_i)-\overline{\widehat{\mu}}(\mathbf{x}_i))^2+(\overline{\widehat{\mu}}(\mathbf{x}_i)-\mu(\mathbf{x}_i))^2+2(\widehat{\mu}_{\mathcal{S}_j}(\mathbf{x}_i)-\overline{\widehat{\mu}}(\mathbf{x}_i))(\overline{\widehat{\mu}}(\mathbf{x}_i)-\mu(\mathbf{x}_i))\\
=&\frac{1}{N_{\mathcal{S}}}\sum_{j=1}^{N_{\mathcal{S}}}\frac{1}{N}\sum_{i\in\mathcal{P}}(\widehat{\mu}_{\mathcal{S}_j}(\mathbf{x}_i)-\overline{\widehat{\mu}}(\mathbf{x}_i))^2+\frac{1}{N_{\mathcal{S}}}\sum_{j=1}^{N_{\mathcal{S}}}\frac{1}{N}\sum_{i\in\mathcal{P}}(\overline{\widehat{\mu}}(\mathbf{x}_i)-\mu(\mathbf{x}_i))^2
\end{aligned}
$$

□

Weekly temperature data for Cairo

To explore the nature of the predictive accuracy of various polynomials, we will use some weekly average air temperatures in Cairo. Load the file `cairo_temp.csv` from LEARN.

```
cairo.temp = read.csv("cairo_temp.csv")
head(cairo.temp)
```

```
##   week temp
## 1    1 59.2
## 2    2 56.5
## 3    3 55.7
## 4    4 56.1
## 5    5 58.4
## 6    6 58.5
```

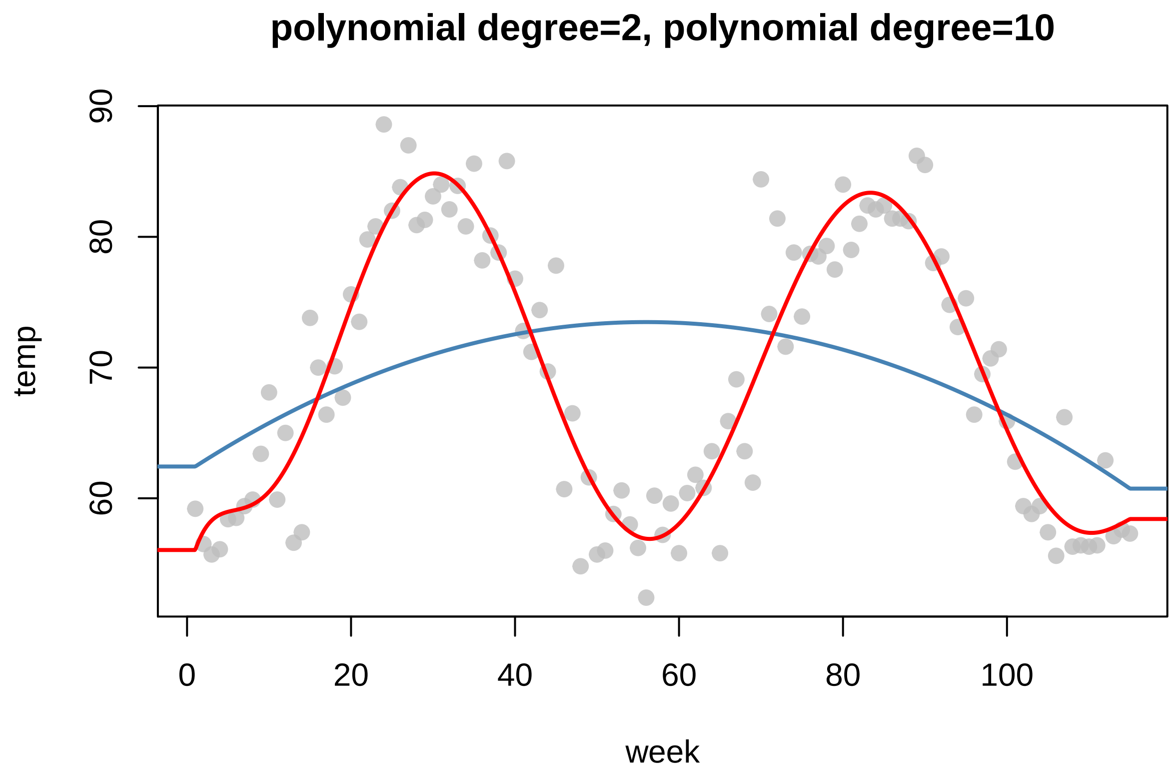
```
names(cairo.temp) = c('x', 'y')
```

```
library(splines)

getmuhat = function(sampleXY, complexity = 1) {
  # If complexity = 0, fit only intercept
  if (complexity == 0) fit = lm(y ~ 1, data = sampleXY)
    else fit = lm(y ~ poly(x, complexity, raw = TRUE), data = sampleXY)
  xmin = min(sampleXY$x, na.rm = TRUE)
  xmax = max(sampleXY$x, na.rm = TRUE)

  # From this we construct the predictor function
  muhat = function(x) {
```

```r
    x    = as.data.frame(x)          # Convert to data frame, needed by predict
    x$x  = pmax(x$x, xmin)           # *Replace values below xmin with xmin
    x$x  = pmin(x$x, xmax)           # *Replace values above xmax with xmax
    pred = predict(fit, newdata = x) # Get yhat values from fitted lm model
    return(pred)
  }
  return(muhat)
}

plotTemperaturefit <- function(muhat1, complexity1=NULL, muhat2, complexity2=NULL) {

  if ( is.null(complexity1) ) title = ""
  else title = paste0("polynomial degree=", complexity1,", ", "polynomial degree=", complexity2,"")

plot(cairo.temp,
     main= title,
     xlab = "week", ylab = "temp",
     pch=19, col= adjustcolor("Grey", 0.8))

xlim =  extendrange(cairo.temp[, 'x'])
curve(muhat1, from = xlim[1], to = xlim[2],
      add = TRUE, col="steelblue", lwd=2, n=1000)
curve(muhat2, from = xlim[1], to = xlim[2],
      add = TRUE, col="red", lwd=2, n=1000)

}
getmuFun = function(pop, xvarname, yvarname){
  pop   = na.omit(pop[, c(xvarname, yvarname)])

  # rule = 2 means return the nearest y-value when extrapolating, same as above.
  # ties = mean means that repeated x-values have their y-values averaged, as above.
  muFun = approxfun(pop[,xvarname], pop[,yvarname], rule = 2, ties = mean)
  return(muFun)
}
```

a) [**4 Marks**] Generate the scatter plot of the data and overlay fitted polynomials with degrees 2 and 10 to the data.

```r
muhat = getmuFun(cairo.temp, "x", 'y')
muhat1 = getmuhat(cairo.temp, complexity=2)
muhat2 = getmuhat(cairo.temp, complexity=10)
plotTemperaturefit(muhat1, 2, muhat2, 10)
```

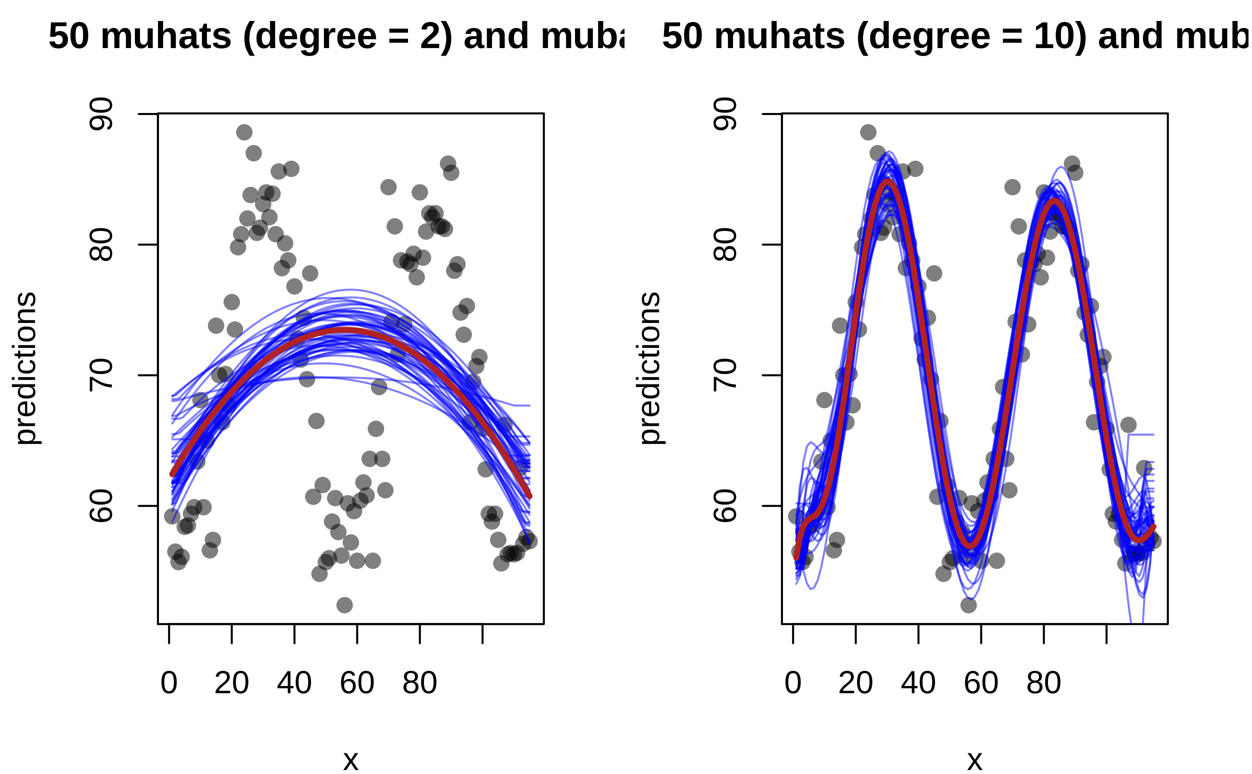


b) [**4 Marks**] Generate $m = 50$ samples of size $n = 50$. Fit polynomials of degree 2 and 10 to every sample.

```
# This function returns a boolean (TRUE/FALSE) vector representing the
#   inclusion indicators. (This way the complement is also recorded.)
getSampleComp = function(pop, size, replace = FALSE) {
  N    = nrow(pop)
  samp = rep(FALSE, N)
  samp[sample(1:N, size, replace = replace)] = TRUE
  return(samp)
}

# This function returns a data frame with only two variates, relabelled as
#   x and y explicitly.
getXYSample = function(xvarname, yvarname, samp, pop) {
  sampData         = pop[samp, c(xvarname, yvarname)]
  names(sampData) = c("x", "y")
  return(sampData)
}
N_S      = 50
n        = 50
samps    = replicate(N_S, getSampleComp(cairo.temp, n), simplify = FALSE)
Ssamples = lapply(samps, function(Si) {getXYSample("x", "y", Si, cairo.temp)})
Tsamples = lapply(samps, function(Si) {getXYSample("x", "y", !Si, cairo.temp)})
muhats2  = lapply(Ssamples, getmuhat, complexity = 2)
muhats10 = lapply(Ssamples, getmuhat, complexity = 10)
```

c) [**5 Marks**] Using `par(mfrow=c(1,2))` plot all the fitted polynomials with degree 2 and 10 on two different figures. Overlay the two fitted polynomials of degree 2 and 10 based on the whole population (make the colour of the population curves different from the others to make them stand out).

```
par(mfrow = c(1, 2))
plot(cairo.temp[,c('x', 'y')],
     pch  = 19,
     col  = adjustcolor("black", 0.5),
     xlab = "x",
     ylab = "predictions",
     main = paste0(N_S, " muhats (degree = 2) and mubar")
)

for (f in muhats2) curve(f(x), add = TRUE, col = adjustcolor("blue", 0.5))
curve(muhat1, add = TRUE, col = "firebrick", lwd = 3)

plot(cairo.temp[,c('x', 'y')],
     pch  = 19,
     col  = adjustcolor("black", 0.5),
     xlab = "x",
     ylab = "predictions",
     main = paste0(N_S, " muhats (degree = 10) and mubar")
)

for (f in muhats10) curve(f(x), add = TRUE, col = adjustcolor("blue", 0.5))
curve(muhat2, add = TRUE, col = "firebrick", lwd = 3)
```



d) [**2 Marks**] Using `var_mutilde` function, calculate the sampling variability of the function of the polynomials with degree equal to 2 and 10

```
ave_y_mu_sq <- function(sample, predfun, na.rm = TRUE){
  mean((sample$y - predfun(sample$x))^2, na.rm = na.rm)
}

ave_mu_mu_sq <- function(predfun1, predfun2, x, na.rm = TRUE){
  mean((predfun1(x) - predfun2(x))^2, na.rm = na.rm)
}
getmubar <- function(muhats) {
  function(x) {
    Ans <- sapply(muhats, function(muhat) {muhat(x)})
    apply(Ans, MARGIN = 1, FUN = mean) # Equivalently, rowMeans(A)
  }
}

var_mutilde <- function(Ssamples, Tsamples, complexity){
  # Evaluate predictor function on each sample S in Ssamples
  muhats = lapply(Ssamples, getmuhat, complexity = complexity)

  # Get the average of these, name it mubar
  mubar  = getmubar(muhats)

  # Average over all samples S
  N_S    = length(Ssamples)
  mean(sapply(1:N_S, function(j) {
    # Use muhat function from sample S_j in Ssamples
    muhat = muhats[[j]]
    ## Average over (x_i, y_i) of sample T_j the squares (y_i - muhat(x_i))^2
    T_j   = Tsamples[[j]]
    return(ave_mu_mu_sq(muhat, mubar, T_j$x))
  }))
}
var_mutilde(Ssamples, Tsamples, complexity = 2)
```

```
## [1] 2.410097
```

```
var_mutilde(Ssamples, Tsamples, complexity = 10)
```

```
## [1] 2.75328
```

The sampling variability of the functionof the polynomials with degree 2 is 3.929952. The sampling variability of the functionof the polynomials with degree 10 is 2.713136. e) [**2 Marks**] Using `bias2_mutilde` function, calculate the squared bias of the polynomials with degree equal to 2 and 10.

```
bias2_mutilde <- function(Ssamples, Tsamples, mu, complexity){
  # Evaluate predictor function on each sample S in Ssamples
  muhats = lapply(Ssamples, getmuhat, complexity = complexity)

  # Get the average of these, name it mubar
  mubar  = getmubar(muhats)

  # Average over all samples S
  N_S    = length(Ssamples)
  mean(sapply(1:N_S, function(j) {
```

```
    ## Average over (x_i, y_i) of sample T_j the squares (y_i - muhat(x_i))^2
    T_j = Tsamples[[j]]
    return(ave_mu_mu_sq(mubar, mu, T_j$x))
  }))
}
bias2_mutilde(Ssamples, Tsamples, muhat, complexity = 2)
```

```
## [1] 95.47591
```

```
bias2_mutilde(Ssamples, Tsamples, muhat, complexity = 10)
```

```
## [1] 13.37413
```

f) [**2 Marks**] Generate $m = 50$ samples of size $n = 50$, and using `apse_all` function, calculate the APSE for complexities equal to `0:10`.
   - Summarize the results with a table and a graphical display.
   - Give a conclusion.

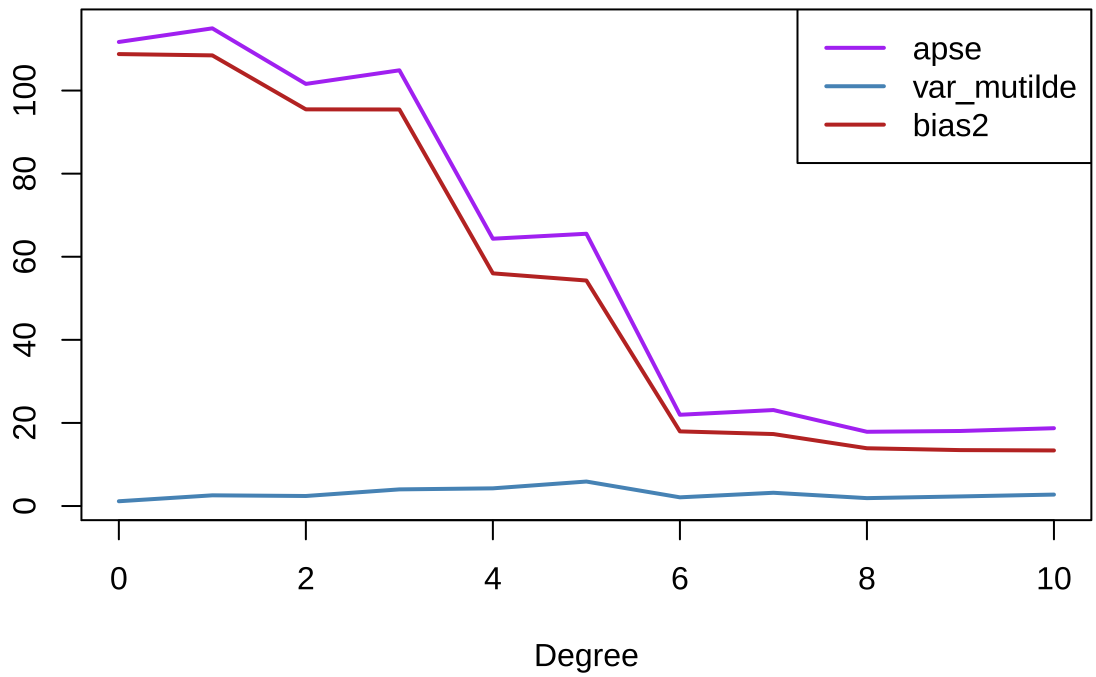
```
apse_all <- function(Ssamples, Tsamples, complexity, mu){
  ## average over the samples S
  ##
  N_S <- length(Ssamples)
  muhats <- lapply(Ssamples,
                   FUN=function(sample) getmuhat(sample, complexity)
  )
  ## get the average of these, mubar
  mubar <- getmubar(muhats)

  rowMeans(sapply(1:N_S,
                  FUN=function(j){
                    T_j         <- Tsamples[[j]]
                    muhat       <- muhats[[j]]
                    ## Take care of any NAs
                    T_j         <- na.omit(T_j)
                    y           <- T_j$y
                    x           <- T_j$x
                    mu_x        <- mu(x)
                    muhat_x     <- muhat(x)
                    mubar_x     <- mubar(x)

                    ## apse
                    ## average over (x_i,y_i) in a
                    ## single sample T_j the squares
                    ## (y - muhat(x))^2
                    apse        <- (y - muhat_x)

                    ## bias2:
                    ## average over (x_i,y_i) in a
                    ## single sample T_j the squares
                    ## (y - muhat(x))^2
                    bias2       <- (mubar_x -mu_x)

                    ## var_mutilde
                    ## average over (x_i,y_i) in a
                    ## single sample T_j the squares
```

```r
                     ## (y - muhat(x))^2
                     var_mutilde <- (muhat_x - mubar_x)

                     ## var_y :
                     ## average over (x_i,y_i) in a
                     ## single sample T_j the squares
                     ## (y - muhat(x))^2
                     var_y       <- (y - mu_x)

                     ## Put them together and square them
                     squares     <- rbind(apse, var_mutilde, bias2, var_y)^2

                     ## return means
                     rowMeans(squares)
                   }
  ))
}
```

```r
complexities = 0:10
apse_vals    = sapply(complexities, function(complexity) {
                       apse_all(Ssamples, Tsamples, complexity = complexity, mu = muhat)
                      })

# Print out the results
t(rbind(complexities, apse=round(apse_vals, 5)))
```

```
##       complexities      apse var_mutilde     bias2 var_y
##  [1,]            0 111.69561     1.14910 108.77866     0
##  [2,]            1 114.97276     2.56982 108.46380     0
##  [3,]            2 101.59985     2.41010  95.47591     0
##  [4,]            3 104.86989     4.00427  95.44454     0
##  [5,]            4  64.33962     4.25348  56.00062     0
##  [6,]            5  65.53230     5.88942  54.27277     0
##  [7,]            6  21.96313     2.09015  17.96315     0
##  [8,]            7  23.09589     3.18524  17.31981     0
##  [9,]            8  17.86756     1.89512  13.90055     0
## [10,]            9  18.05951     2.31098  13.45582     0
## [11,]           10  18.73031     2.75328  13.37413     0
```

```r
matplot(complexities, t(apse_vals[1:3,]),
        type = 'l',
        lty = 1,
        lwd = 2,
        col = c("purple", "steelblue", "firebrick"),
        xlab = "Degree",
        ylab = "")
legend('topright',
       legend = rownames(apse_vals)[1:3],
       lty = 1,
       lwd = 2,
       col = c("purple", "steelblue", "firebrick"))
```



APSE is decreasing at first and increases a little bit at the end. The smallest APSE occurs when degree = 8.

g) Instead of randomly constructing sample and test sets we can use k-fold cross-validation.

i) [**4 Marks**] Create a function creates the k-fold samples from a given population. i.e.

- The function has arguements `k` the number of k-fold, `pop` a population, `xvarname` the name of the x variable, and `yvarname` the of the y variable.
- The function outputs a list containing the k-fold samples and test samples labelled as `Ssamples` and `Tsamples`.
- The function `rep_len` might be helpful.

```
sample.kfold <- function(k = NULL, pop = NULL, xvarname = NULL, yvarname = NULL) {
  N        = nrow(pop)
  kset     = rep_len(1:k, N)
  kset     = sample(kset)
  samps    = lapply(1:k, function(k) kset != k)
  Ssamples = lapply(samps, function(Si) getXYSample(xvarname, yvarname, Si, pop))
  Tsamples = lapply(samps, function(Si) getXYSample(xvarname, yvarname, !Si, pop))
  return(list(Ssamples = Ssamples, Tsamples = Tsamples))
}
```

```
ii) **[3 Marks]** Use the function from part i) and the `apse` function to find an estimate of the APSE
```

```
kfold.samples = sample.kfold(k = 5, pop = cairo.temp, "x", "y")
sample.muFun = getmuFun(cairo.temp, "x", 'y')

apse_all(kfold.samples$Ssamples, kfold.samples$Tsamples, complexity = 2, mu = sample.muFun)
```

```
##        apse var_mutilde       bias2       var_y
```

```
## 99.4080292  0.4870992  95.2897975  0.0000000
```

iii) **[4 Marks]** Perform $k=10$-fold cross-validation to estimate the complexity parameter from the s

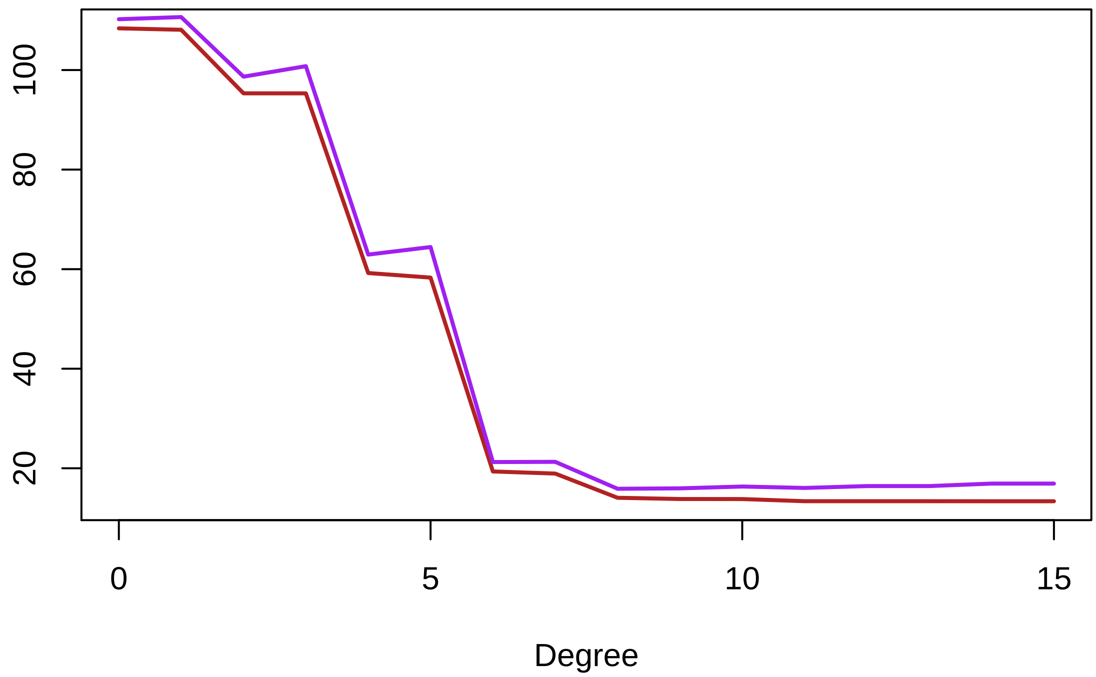
```r
kfold.samples2 = sample.kfold(k = 10, pop = cairo.temp, "x", "y")
complexities = 0:15
apse_vals    = sapply(complexities, function(complexity) {
                        apse_all(kfold.samples2$Ssamples, kfold.samples2$Tsamples,
                                 complexity = complexity, mu = sample.muFun)
                     })

# Print out the results
t(rbind(complexities, apse=round(apse_vals,5)))
```

```
##       complexities      apse var_mutilde     bias2 var_y
##  [1,]            0 110.21183     0.09441 108.37676     0
##  [2,]            1 110.64424     0.13419 108.09399     0
##  [3,]            2  98.66934     0.20450  95.32572     0
##  [4,]            3 100.78210     0.40264  95.32043     0
##  [5,]            4  62.92634     0.41023  59.22090     0
##  [6,]            5  64.44216     0.51740  58.30753     0
##  [7,]            6  21.24790     0.21867  19.36001     0
##  [8,]            7  21.29529     0.25713  18.93406     0
##  [9,]            8  15.87987     0.15545  14.07435     0
## [10,]            9  15.97016     0.20162  13.82792     0
## [11,]           10  16.33281     0.27447  13.81735     0
## [12,]           11  16.05049     0.33007  13.38727     0
## [13,]           12  16.42397     0.37618  13.38357     0
## [14,]           13  16.42397     0.37618  13.38357     0
## [15,]           14  16.92164     0.48811  13.37719     0
## [16,]           15  16.92164     0.48811  13.37719     0
```

```r
complexities[apse_vals[2, ] == min(apse_vals[2, ])]
```

```
## [1] 0
```

```r
plot(complexities, apse_vals[3,], xlab="Degree", ylab="", type='l', col="firebrick", lwd=2)
lines(complexities, apse_vals[2,], xlab="Degree", ylab="", col="steelblue", lwd=2 )
lines(complexities, apse_vals[1,], col="purple", lwd=2)
```



We choose the polynomial with degree 8