Predictive accuracy

R.W. Oldford

Contents

---

1 Assessing accuracy

Response models separate variates into two groups, those that are explanatory and those that are response. The values of the explanatory variates are used to explain or predict the values of the response. To predict, we use our observed data to construct a function, $\widehat{\mu}(\mathbf{x})$, which can be used to predict y at any given value $\mathbf{x}$.

For any predictor $\widehat{\mu}(\mathbf{x})$ of y, we would like to know how well it does for values in some collection $\mathcal{P}$ of pairs $(\mathbf{x}, y)$. That is we would like to know how accurate it is in its predictions. Equivalently, we might ask instead how **inaccurate** are the predictions – this is usually easier to measure.

For example, we could measure the inaccuracy by the average squared error over $\mathcal{P}$, that is

$$Ave_{(\mathbf{x},y)\in\mathcal{P}}(y - \widehat{\mu}(\mathbf{x}))^2$$

which treats every point in $\mathcal{P}$ with equal weight. There are numerous possibilities for the set $\mathcal{P}$ – imagine for example, any study or target population.

So how might we determine this? A simple approach would be to take $\mathcal{P}$ to be the observed set for which we have measurements on both $\mathbf{x}$ and y. That is calculate

$$Ave_{i=1,\ldots,N}(y_i - \widehat{\mu}(\mathbf{x_i}))^2.$$

This leads to the familiar residual sum of squares over N or the "residual mean squared error".

The problem with this choice is that the estimate of the predictor function is based on the **same set** of observations and the data pairs $(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_N, y_N)$ as the set $\mathcal{P}$ over which the average is taken. This doesn't seem to be the most honest way to estimate a prediction's inaccuracy. After all, we are using the same x values **and the values** y **we want to predict** to construct the predictor function. Surely this will typically underestimate the average squared error for **prediction** of any other values of x.

The **average predicted squared error** of $\widehat{\mu}$ is written as

$$Ave_{(\mathbf{x},y)\in\mathcal{P}}(y - \widehat{\mu}_{\mathcal{S}}(\mathbf{x}))^2$$

to emphasize that the predictor function $\widehat{\mu}$ is based on a set of observed $(\mathbf{x}, y)$ pairs from some set $\mathcal{S}$. Typically $\mathcal{S}$ is a sample of size N and $\mathcal{P}$ is a study population. Or, $\mathcal{S}$ is an entire study population and $\mathcal{P}$ is some target population.

The problem with the simple residual mean squared error is that it is essentially the following estimate of average prediction squared error

$$\widehat{Ave}_{(\mathbf{x},y)\in\mathcal{P}}(y-\widehat{\mu}_{\mathcal{S}}(\mathbf{x}))^2 = Ave_{i=1,\ldots,N}(y_i-\widehat{\mu}(\mathbf{x_i}))^2 = Ave_{(\mathbf{x},y)\in\mathcal{S}}(y-\widehat{\mu}_{\mathcal{S}}(\mathbf{x}))^2$$

and this estimate uses $\mathcal{S}$ twice over – once as the population $\mathcal{P}$ and once as the sample used to construct $\widehat{\mu}$. The result is very likely an overly optimistic value for the inaccuracy of $\widehat{\mu}$ on $\mathcal{P}$.

To avoid this, we restrict the inaccuracy measurement to points in $\mathcal{P}$ but outside of the set $\mathcal{S}$ on which the predictor function was built. That is, letting $\mathcal{T} = \mathcal{P} - \mathcal{S}$ be the complement set in $\mathcal{P}$ of $\mathcal{S}$ we use the prediction inaccuracy

$$Ave_{(\mathbf{x},y)\in\mathcal{T}}(y-\widehat{\mu}_{\mathcal{S}}(\mathbf{x}))^2$$

instead. This makes it clear that the residual sum of squares is inappropriate as a measure of **prediction** inaccuracy.

1.1 Examples

Here we will look at some examples where we have complete knowledge of the joint distribution of X and Y which can be used to produce independent pairs of (x, y) values.

The generative model will be a response model for the conditional distribution of Y given $X = x$ together with a marginal model for X. The response model has the conditional mean of Y given X to be $\mu(x)$. The model is expressed mathematically as

$$\begin{aligned} X &\sim F_X(x) \quad \text{to be specified,} \\ Y &= \mu(x) + R \quad \text{this being conditional on the realization } X = x, \text{ and} \\ R &\sim N(0, \sigma^2). \end{aligned}$$

We will use this model to generate (x, y) pairs by first generating N independent realizations of X, $x_1, \ldots, x_N$, then N independent realizations of R, $r_1, \ldots, r_N$, and finally the N realizations of Y as $y_i = \mu(x_i) + r_i$, for $i = 1, \ldots, N$. In the end we will have pairs $(x_1, y_1), \ldots, (x_N, y_N)$ which can be used to produce a predictor function $\widehat{\mu}(x)$ for any other x of our choice.

Of course the best predictor function will be the $\mu(x)$ that actually generates the data and the lowest prediction inaccuracy (over all x) will simply be the value of σ^2.

In these experiments, then, the sample $\mathcal{S}$ will be some set representing the individuals corresponding to the realizations $(x_1, y_1), \ldots, (x_N, y_N)$ and $\mathcal{P}$ will be the same for the set of all possible realizations from this model. Because we will be doing simulation experiments, only a finite (though large) number of new values from $\mathcal{P}$ will be used – these will be generated according to the above model.

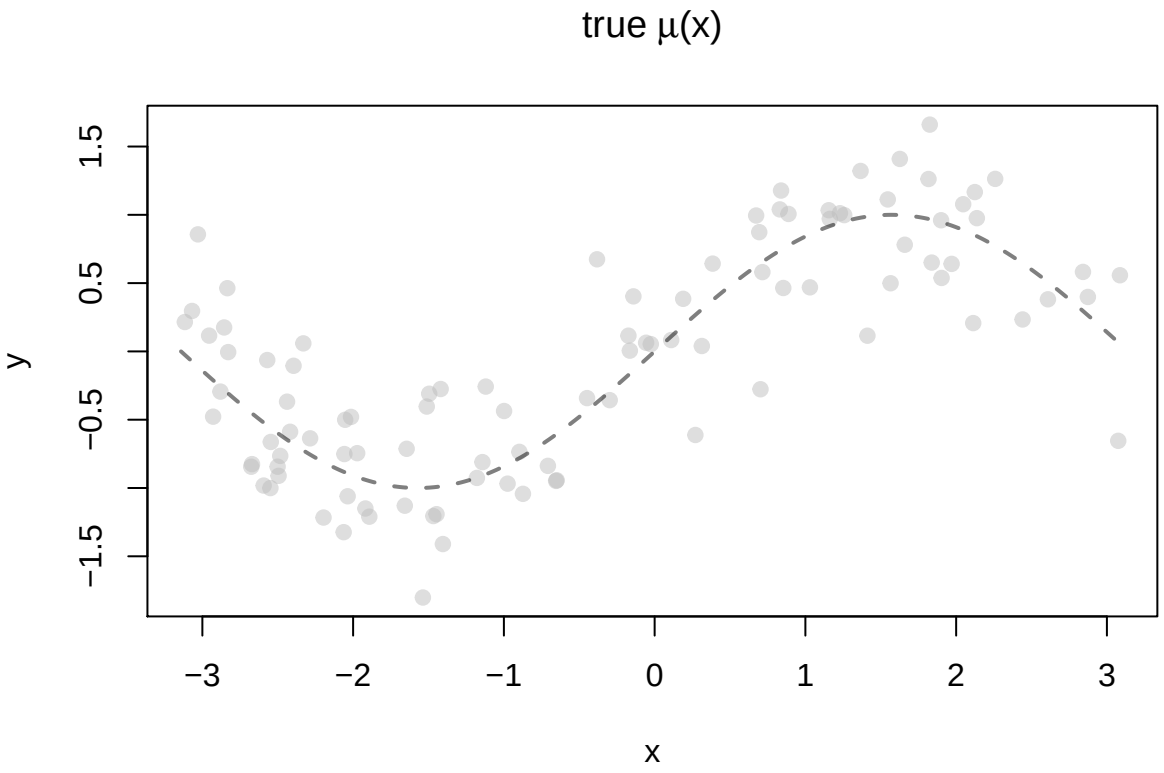
1.1.1 Example 1: A relatively simple predictor function

The generative model will be as above with

$$\begin{aligned} X &\sim U(-\pi, \pi) \\ \mu(x) &= \sin x \quad \text{and} \\ \sigma &= 0.4. \end{aligned}$$

We begin by getting a sample $\mathcal{S}$.

```
# generate some data; set the seed for reproducibility
set.seed(34243411)
n <- 100
from <- -pi
to <- pi
x <- runif(n, from, to)

mu <- function(x) {sin(x)}
resids <- function(n) {rnorm(n,0,0.4)}

y <- mu(x) + resids(n)

# plot the data in the set S
plot(x,y, pch=19, col=adjustcolor("grey", 0.5),
     main=expression(paste("true ", mu,"(x)")))
funx <-seq(from, to, length.out=1000)
lines(funx, mu(funx), col=adjustcolor("black", 0.5), lty=2, lwd=2)
```

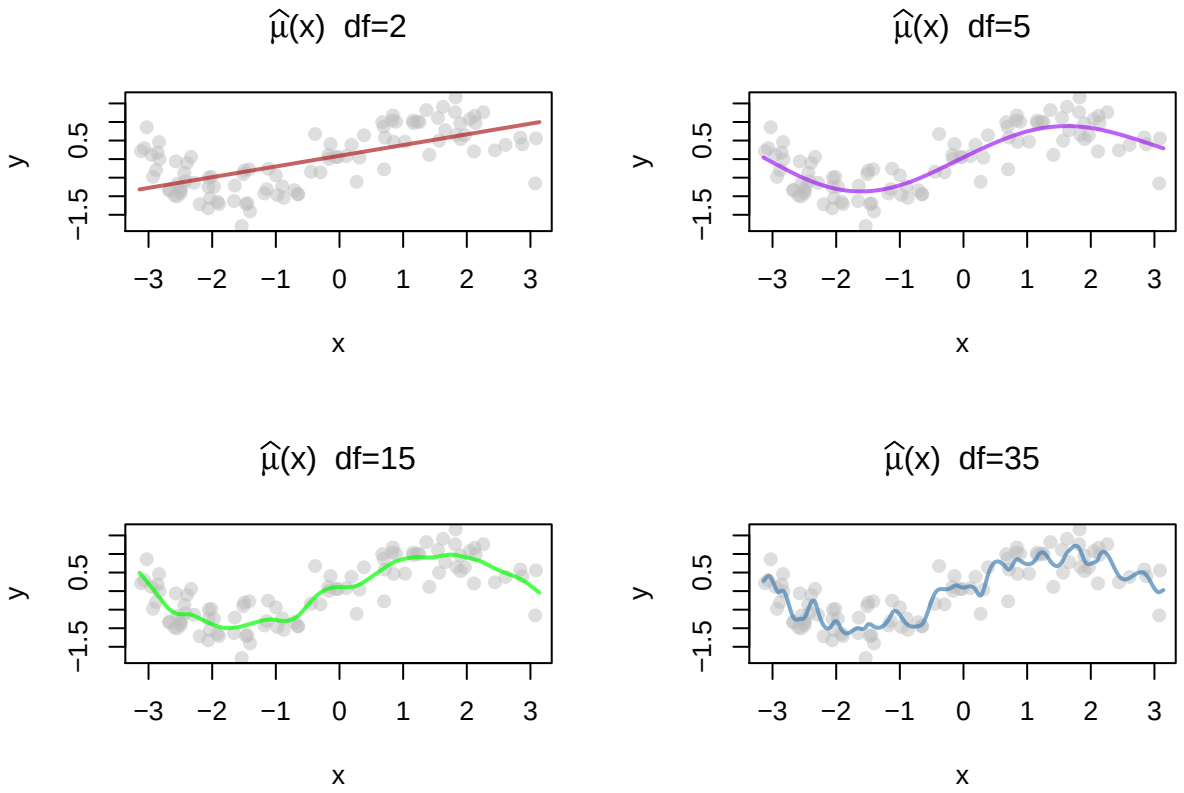


These data constitute our sample set $\mathcal{S}$; the curve is the $\mu(x)$ used to generate the data.

We can now find predictor functions $\widehat{\mu}(x)$ of differerent complexities.

```
# Now let's fit some predictor functions
#
# First a straight line
fit1 <- lm(y ~x)
mu1hat <- function(x){predict(fit1, newdata=data.frame(x=x))}

parOptions <- par(mfrow=c(2,2))
# The fitted values can be laid over the data:
```

```r
plot(x,y, pch=19, col=adjustcolor("grey", 0.5),
     main=expression(paste(widehat(mu),"(x)", "  df=2")))
lines(funx, mu1hat(funx), col=adjustcolor("firebrick",0.7), lwd=2)

# Now some smoothing splines
library(splines)
fit2 <-  smooth.spline(x, y, df = 5)
mu2hat <- function(x){predict(fit2, x=x)$y}
plot(x,y, pch=19, col=adjustcolor("grey", 0.5),
     main=expression(paste(widehat(mu),"(x)", "  df=5")))
lines(funx, mu2hat(funx), col=adjustcolor("purple",0.7), lwd=2)

# A smooth with 5 more degrees of freedom
fit3 <- smooth.spline(x, y, df = 10)
mu3hat <- function(x){predict(fit3,  x=x)$y}

# A smooth with 5 more degrees of freedom
fit4 <- smooth.spline(x, y, df = 15)
mu4hat <- function(x){predict(fit4,  x=x)$y}
plot(x,y, pch=19, col=adjustcolor("grey", 0.5),
     main=expression(paste(widehat(mu),"(x)", "  df=15")))
lines(funx, mu4hat(funx), col=adjustcolor("green",0.7), lwd=2)

# A smooth with 10 more degrees of freedom
fit5 <- smooth.spline(x, y, df = 25)
mu5hat <- function(x){predict(fit5,  x=x)$y}

# A smooth with 10 more degrees of freedom
fit6 <- smooth.spline(x, y, df = 35)
mu6hat <- function(x){predict(fit6,  x=x)$y}
plot(x,y, pch=19, col=adjustcolor("grey", 0.5),
     main=expression(paste(widehat(mu),"(x)", "  df=35")))
lines(funx, mu6hat(funx), col=adjustcolor("steelblue",0.7), lwd=2)
```




```
par(parOptions)
```

We have just calculated six different predictor functions $\widehat{\mu}_1(x)$, $\widehat{\mu}_2(x)$, $\widehat{\mu}_3(x)$, $\widehat{\mu}_4(x)$, $\widehat{\mu}_5(x)$, and $\widehat{\mu}_6(x)$ of increasing complexity (as measured by the effective degrees of freedom. Note that the above plots show only four of these, namely $\widehat{\mu}_1(x)$ in "firebrick" red, $\widehat{\mu}_2(x)$ in "purple", $\widehat{\mu}_4(x)$ in "green", and $\widehat{\mu}_6(x)$ in "steelblue". The true predictive function $\mu(x)$ is shown in dashed "black".

For each of these functions we can calculate the residual sum of squares (or predictive inaccuracy evaluated on the original sample $\mathcal{S}$).

To get the predictive inaccuracy evaluated on $\mathcal{T}$, we need to actually have the values of x and y for observations in $\mathcal{T}$. These we can get from our generative model. Knowing this model allows us to generate realizations y for x values of our choice.

We can use the data from $\mathcal{S}$ and from $\mathcal{T}$ to explore how the various estimates perform.

For $\mathcal{P}$ we'll generate a large number of new values from our model (or from $\mathcal{T}$ since these will not be the same ones we had before). For example, if we generate 1,000 pairs $(x_1^*, y_1^*), \ldots, (x_{1,000}^*, y_{1,000}^*)$ values from $\mathcal{T}$. we can then use these to estimate the average squared prediction error for each predictor function.

Here are the values for $\mathcal{T}$.

```
# Get our data for the set T
newn <- 1000
newx <- runif(newn, from, to)  # generate from the distribution we have for x
newy <- mu(newx) + resids(newn)
```

We can simply define a function to calculate the average prediction squared error for any given x, y, and predictor function $\widehat{\mu}_{\mathcal{S}}$.

```
# Average prediction squared error
apse <- function(x, y, predfun){mean((y - predfun(x))^2)}
```

This `apse` function can now be called on the apppropriate values of x and y (from either $\mathcal{S}$ or $\mathcal{P}$) using the predictor function as already constructed on x and y of $\mathcal{S}$.

When the data on which the prediction inaccuracy is to be assessed are from $\mathcal{S}$, the function is called on the x and y from the sample and the prediction accuracy is simply the average residual sum of squares for each predictor function $\widehat{\mu}_i(x)$ $(i = 1, \ldots, 6)$.

```
# Get the average RSS (apse on S) for each
rssmu <- apse(x, y, mu)
rssmu1hat <- apse(x, y, mu1hat)
rssmu2hat <- apse(x, y, mu2hat)
rssmu3hat <- apse(x, y, mu3hat)
rssmu4hat <- apse(x, y, mu4hat)
rssmu5hat <- apse(x, y, mu5hat)
rssmu6hat <- apse(x, y, mu6hat)

# the apse on S for the fitted predictors.
rssmuhats <- c(rssmu1hat, rssmu2hat, rssmu3hat, rssmu4hat, rssmu5hat, rssmu6hat)
```

Similarly, when the prediction data are from the out of sample $\mathcal{T}$, we call the prediction inaccuracy function on the x and y values we have selected from $\mathcal{T}$. Again, we can do this only because we are in the unique position of knowing the true $\mu(x)$ and how the generative model for the values y.

```
# Get the average apse on T for each
apsemu <- apse(newx, newy, mu)
apsemu1hat <- apse(newx, newy, mu1hat)
apsemu2hat <- apse(newx, newy, mu2hat)
apsemu3hat <- apse(newx, newy, mu3hat)
apsemu4hat <- apse(newx, newy, mu4hat)
apsemu5hat <- apse(newx, newy, mu5hat)
apsemu6hat <- apse(newx, newy, mu6hat)

# the apse on T for the fitted predictors.
apsemuhats <- c(apsemu1hat, apsemu2hat, apsemu3hat, apsemu4hat, apsemu5hat, apsemu6hat)
```

Note that each of the predictor functions have varying (effective) degrees of freedom associated with them. These are a measure of the model complexity.

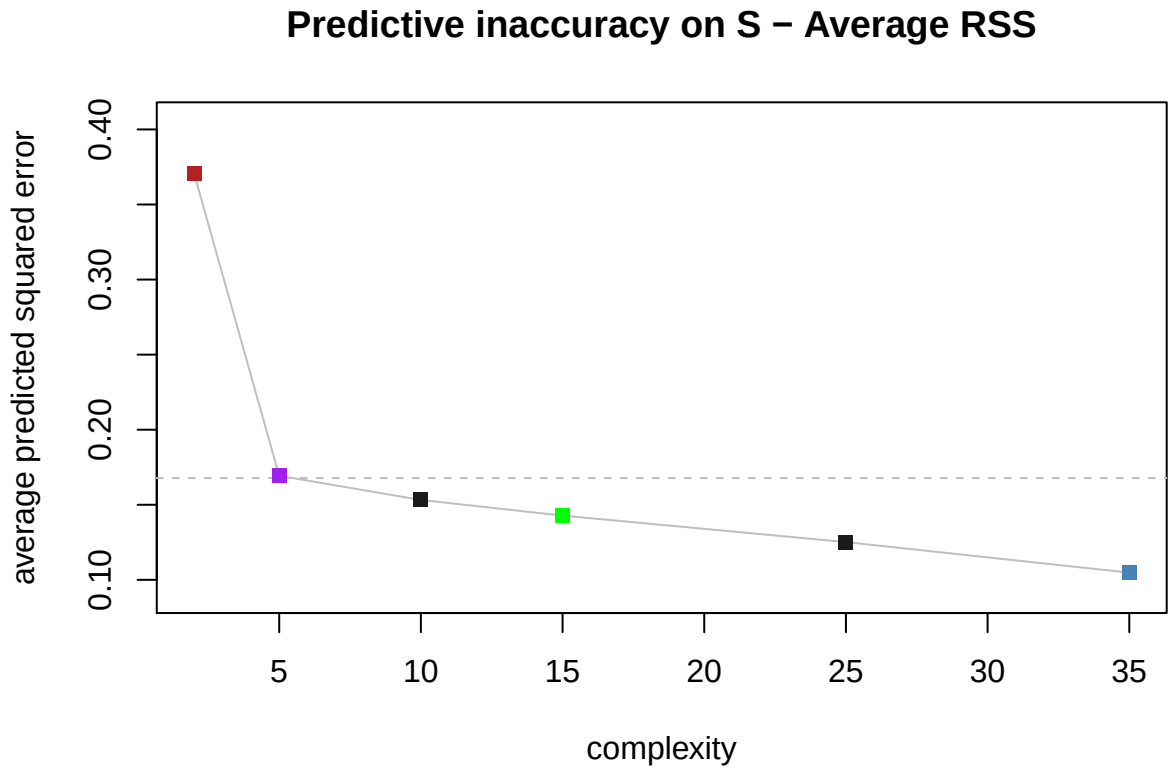
```
# the degrees of freedom associated with each
complexity <- c(2, 5, 10, 15, 25, 35)
```

We can now plot our results to see how well the various predictor functions perform with respect to the prediction set $\mathcal{S}$ or $\mathcal{T}$. We plot the average prediction squared error versus each predictor functions complexity (as given by their effective degrees of freedom).

First for the prediction set $\mathcal{S}$.

```
# Y limits to be matched on both plots
ylim <- extendrange(c(rssmuhats, apsemuhats))

# Plot the predictive inaccuracy on the sample S
plot(complexity, rssmuhats,
     ylab="average predicted squared error",
     main="Predictive inaccuracy on S - Average RSS",
     ylim=ylim, type="n")  # "n" suppresses the plot, add values later
abline(h=rssmu, lty=2, col="grey")
lines(complexity, rssmuhats, col="grey75")
points(complexity, rssmuhats, pch=15,
```

```
      col=c("firebrick", "purple",
            "grey10", "green", "grey10",
            "steelblue"))
```

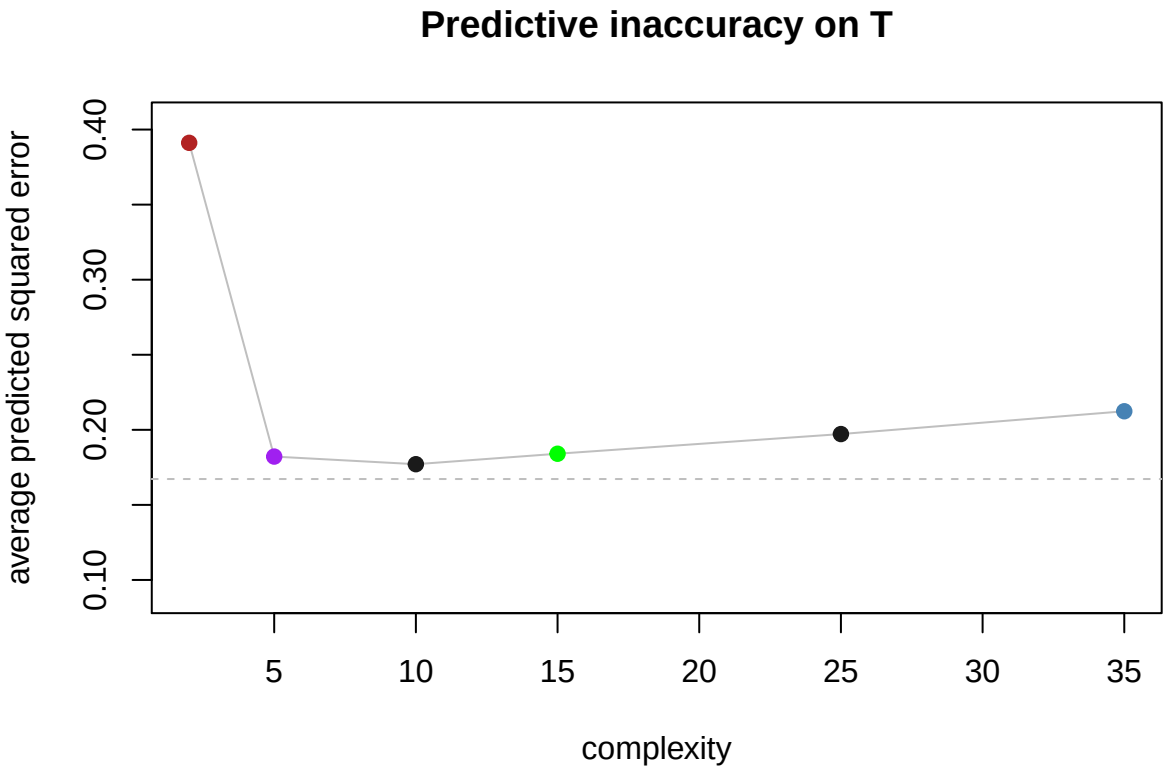


The four coloured points in this plot correspond to the four predictor functions shown in the previous plot; the remaining grey points are the other two predictor functions that were constructed but not shown. The horizontal dashed line is the prediction inaccuracy obtained on the points in $\mathcal{S}$ by the true $\mu(x)$.

As can be seen from the plot, the prediction error on the data gets smaller and smaller as the complexity of the predictor function increases. That is, as the predictor function gets more complex, it is able to better fit the data on which it is constructed and hence to predict that self same data. This should not be surprising given that these prediction inaccuracy calculations are proportional to the residual sum of squares of each model.

Now look at the prediction accuracies of the same predictor functions as before but now evaluated on data not in $\mathcal{S}$ but in $\mathcal{T}$.

```
# Plot the predictive inaccuracy on the out of sample T
plot(complexity, apsemuhats,
     ylab="average predicted squared error",
     main="Predictive inaccuracy on T",
     ylim=ylim, type="n")  # "n" suppresses the plot, add values later
abline(h=apsemu, lty=2, col="grey")
lines(complexity, apsemuhats, col="grey75")
points(complexity, apsemuhats, pch=19,
       col=c("firebrick", "purple",
             "grey10", "green", "grey10",
             "steelblue"))
```



As with the previous plot, the four coloured points here correspond to the four predictor functions and the remaining grey points the other two predictor functions that were constructed but not shown. Again, the solid horizontal line is the theoretical lower limit on the prediction inaccuracy using the true $\mu(x)$ over all x, namely σ^2 but now the dashed line slightly above it is the prediction inaccuracy obtained by the true $\mu(x)$ on the points in $\mathcal{T}$ as opposed to those in $\mathcal{S}$.

Note that this plot and the previous one have the same vertical scales so that the points of the second plot are easily seen to be above those of the first. This means that the average predictive squared error on $\mathcal{T}$ is greater than the same on $\mathcal{S}$. This reinforces the conclusion drawn earlier that the average squared residuals underestimate the inaccuracy of the prediction.

Note also that unlike the previous plot, the prediction inaccuracy does not continue to diminish as complexity increases. Rather there seems to be a local minimum around the predictor function having complexity of 10 (i.e. effective degrees of freedom of 10). Either side of this point the prediction inaccuracy increases again.

Based on these observations we would choose a smoothing spline with effective degrees of freedom of 10 over any of the others we considered.

1.1.2 Example 2: A more complex predictor function

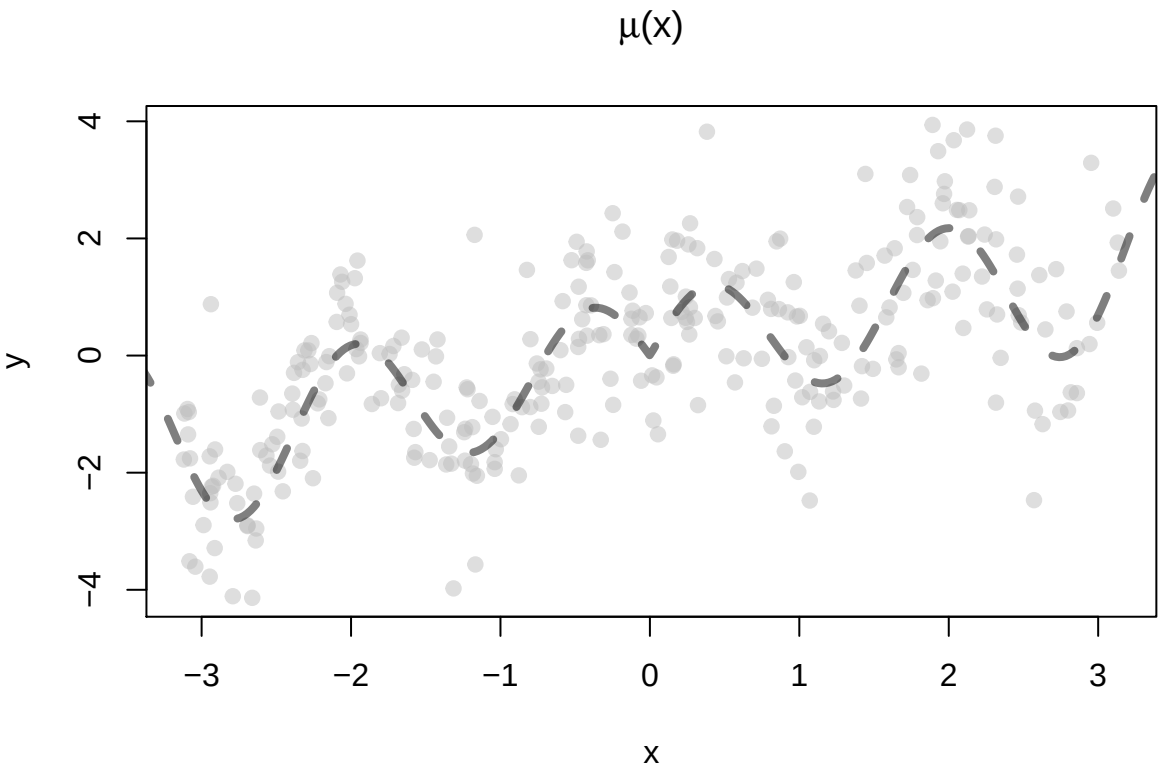
The generative model will be as above with

$$\begin{aligned} X &\sim U(-\pi, \pi) \\ \mu(x) &= \tfrac{1}{2}x + |x|(\cos(x/2) + \cos(3x/2) + \cos(5x/2) + \cos(7x/2)) \quad \text{and} \\ \sigma &= 1. \end{aligned}$$

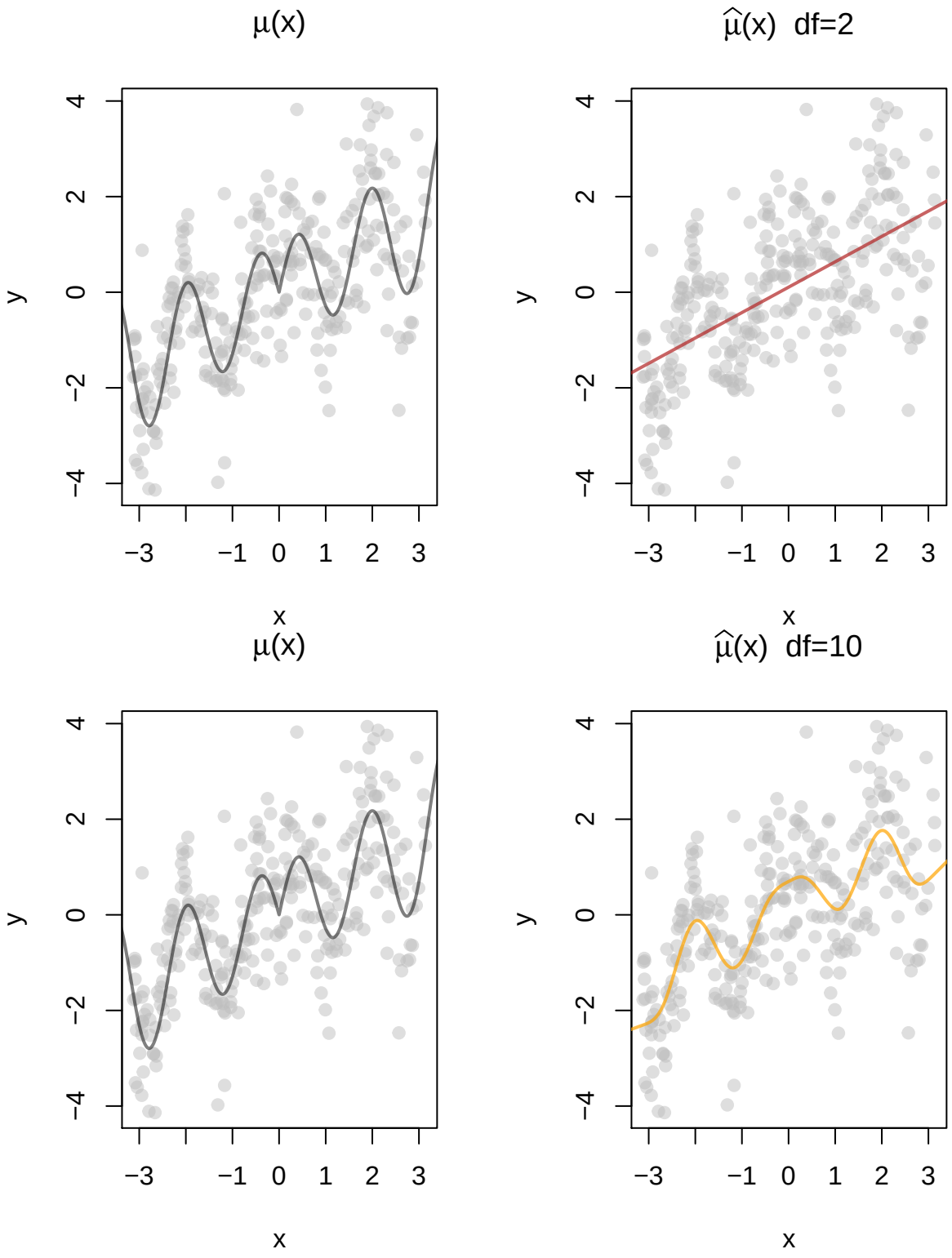
We begin by getting a sample $\mathcal{S}$.

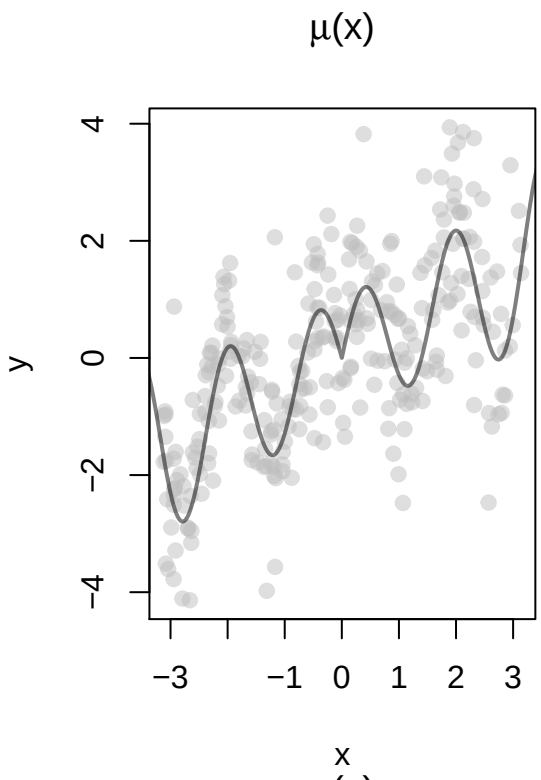
```
#
#
# Get some x's
set.seed(12032345)
N <- 300
x <- runif(N, -pi, pi)
#
#  A function for the mean of ys
#
mu <- function(x) {
       x/2 + abs(x) * (cos(x /2) +
                       cos(3 * x /2) +
                       cos(5 * x /2) +
                       cos(7 * x /2)
                      )
}

# generate some ys
#
sigma <- 1
y <- mu(x) + rnorm(N, 0, sigma)
```

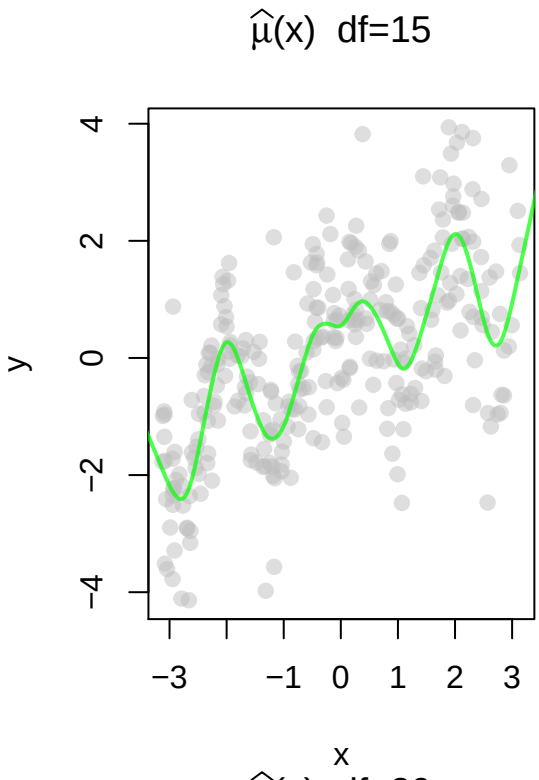
And plot them as below to show the sample set $\mathcal{S}$ and the curve $\mu(x)$ used to generate the ys.

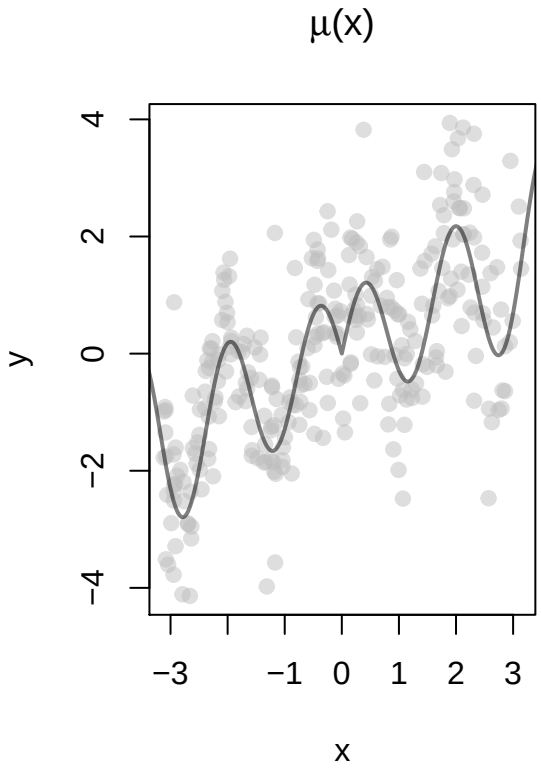


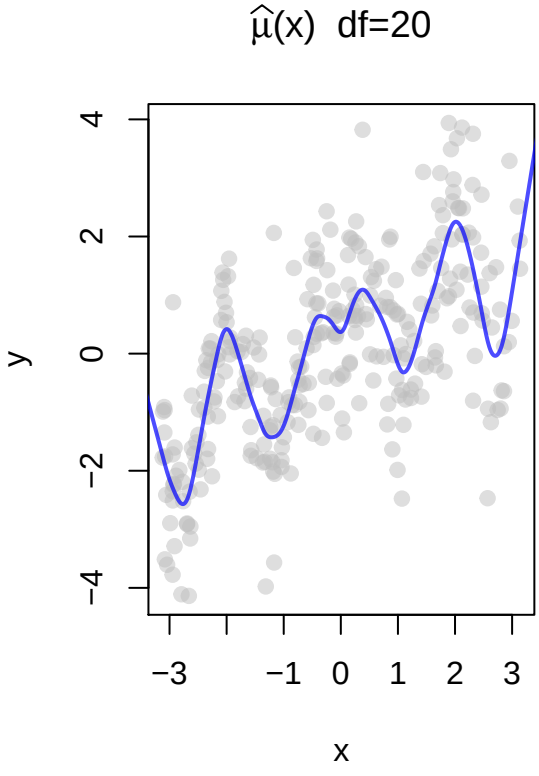
Predictor functions of various complexities can be estimated on $\mathcal{S}$ and plotted on top of the data.

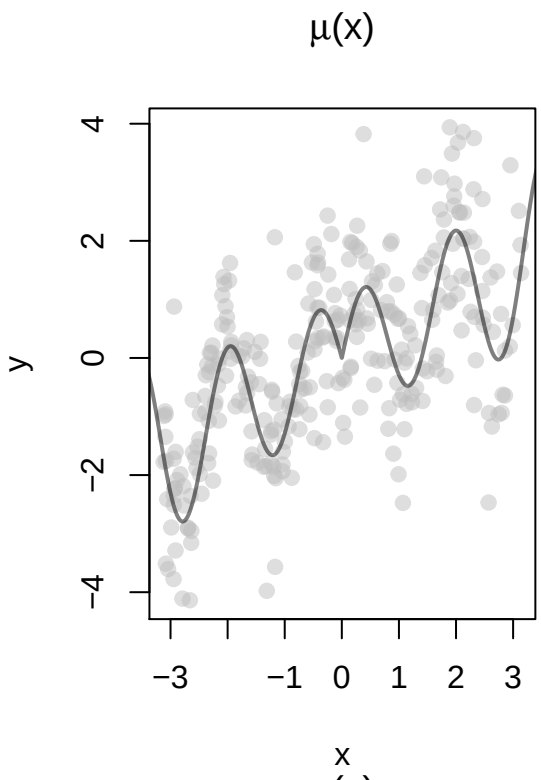

μ(x)
μ̂(x) df=2
μ(x)
μ̂(x) df=10
y
x
−4
−2
0
2
4
−3
−1
0
1
2
3

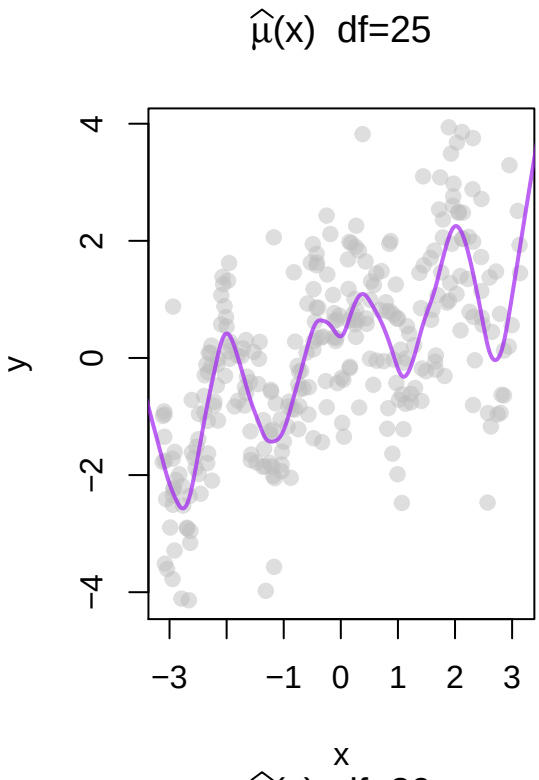

μ(x)
y
x
4
2
0
−2
−4
−3
−1
0
1
2
3

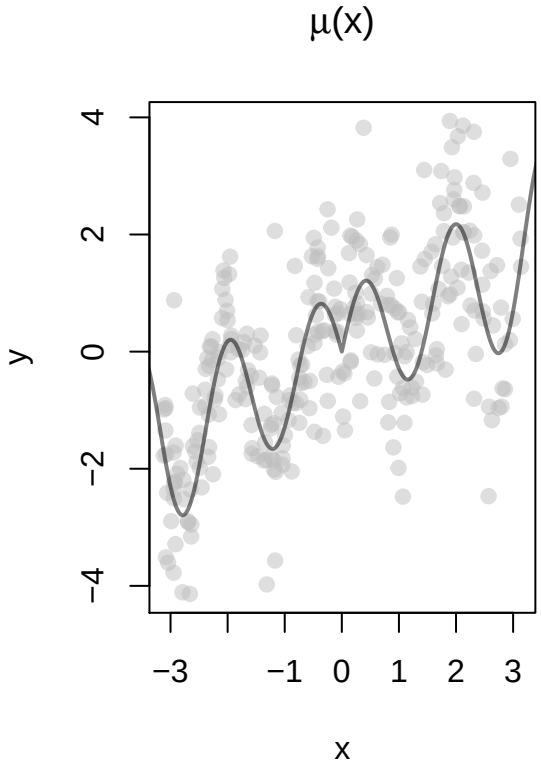

μ̂(x) df=15
y
x
4
2
0
−2
−4
−3
−1
0
1
2
3

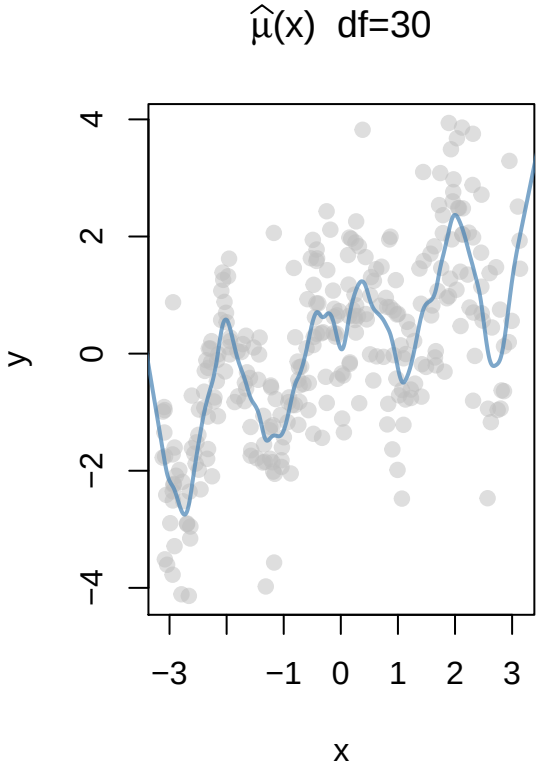

μ(x)
y
x
4
2
0
−2
−4
−3
−1
0
1
2
3

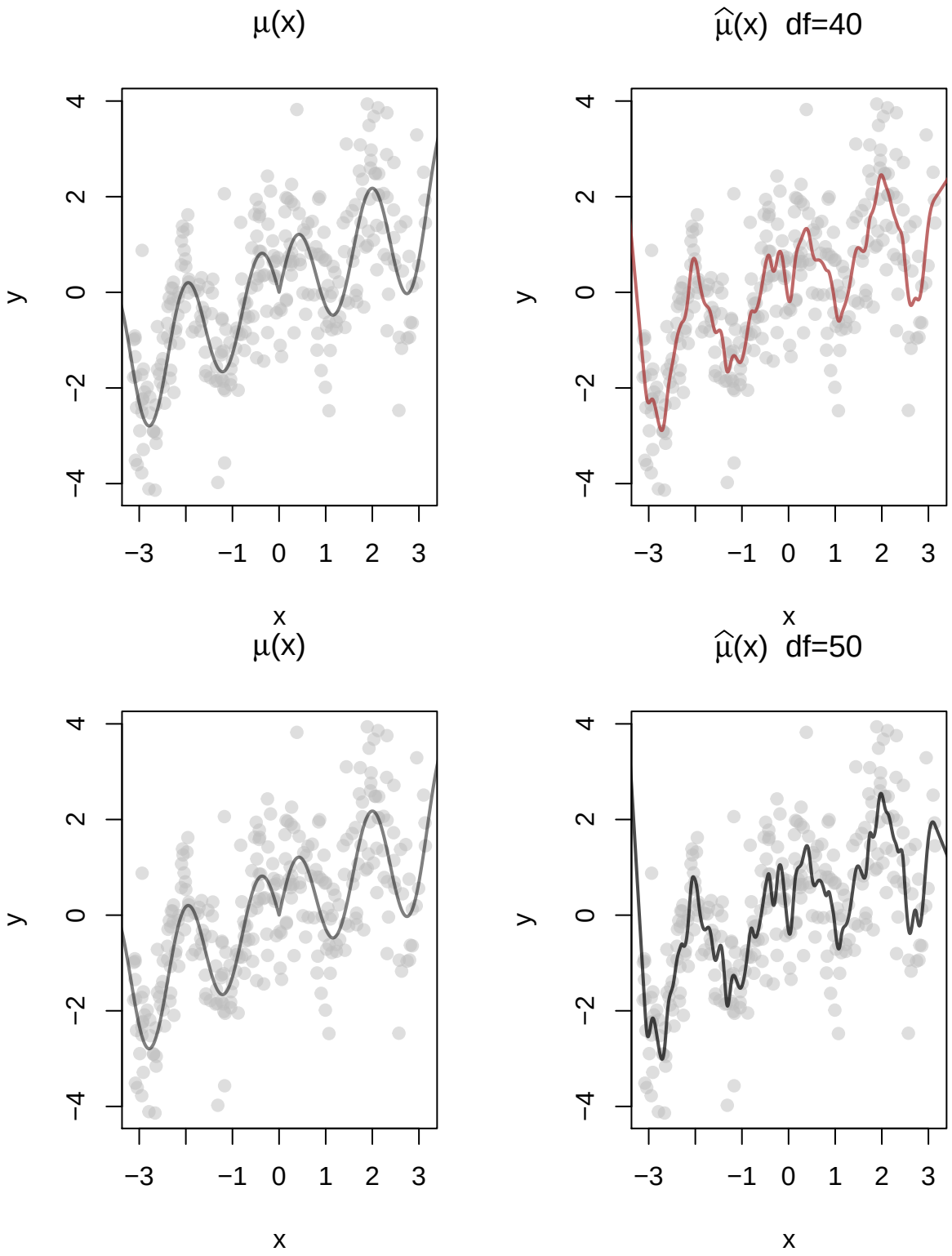

μ̂(x) df=20
y
x
4
2
0
−2
−4
−3
−1
0
1
2
3


μ(x)
y
x
−3
−1
0
1
2
3
−4
−2
0
2
4


μ̂(x) df=25
y
x
−3
−1
0
1
2
3
−4
−2
0
2
4


μ(x)
y
x
−3
−1
0
1
2
3
−4
−2
0
2
4

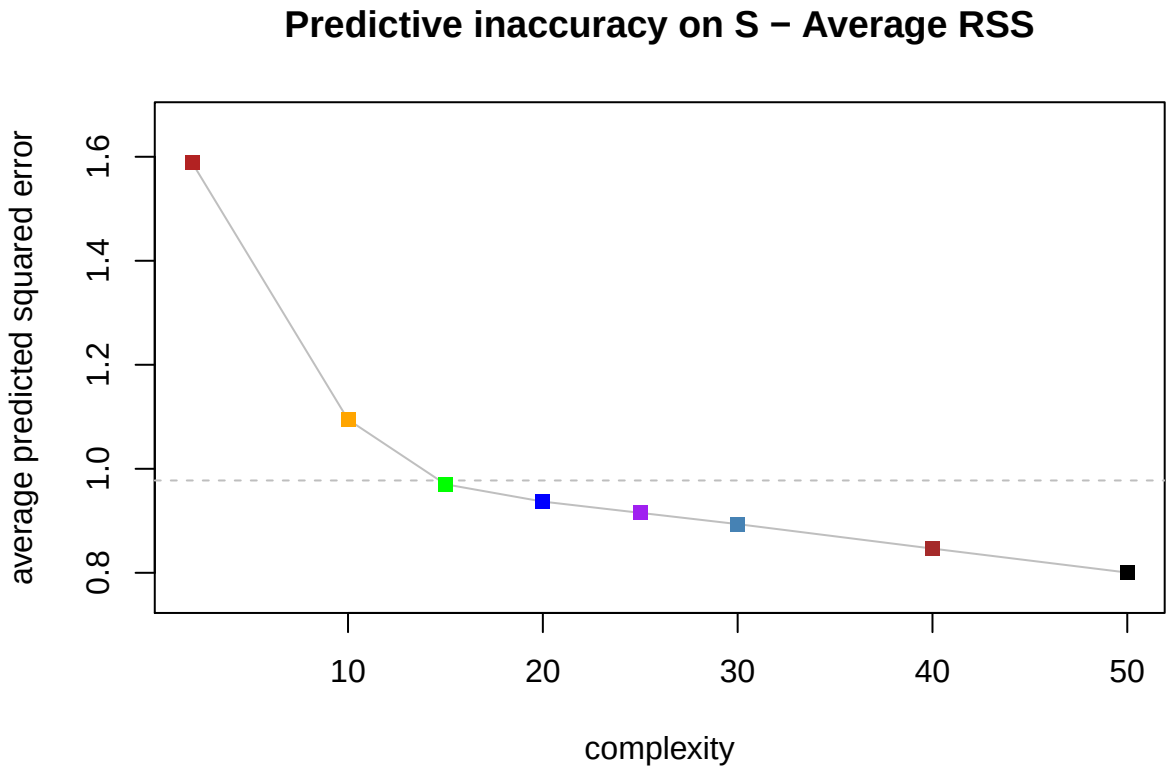

μ̂(x) df=30
y
x
−3
−1
0
1
2
3
−4
−2
0
2
4



We see that the functions become wigglier as the complexity (effective degrees of freedom) increase. We can now again plot the average predictive squared error of each function (colour coded) with its complexity.

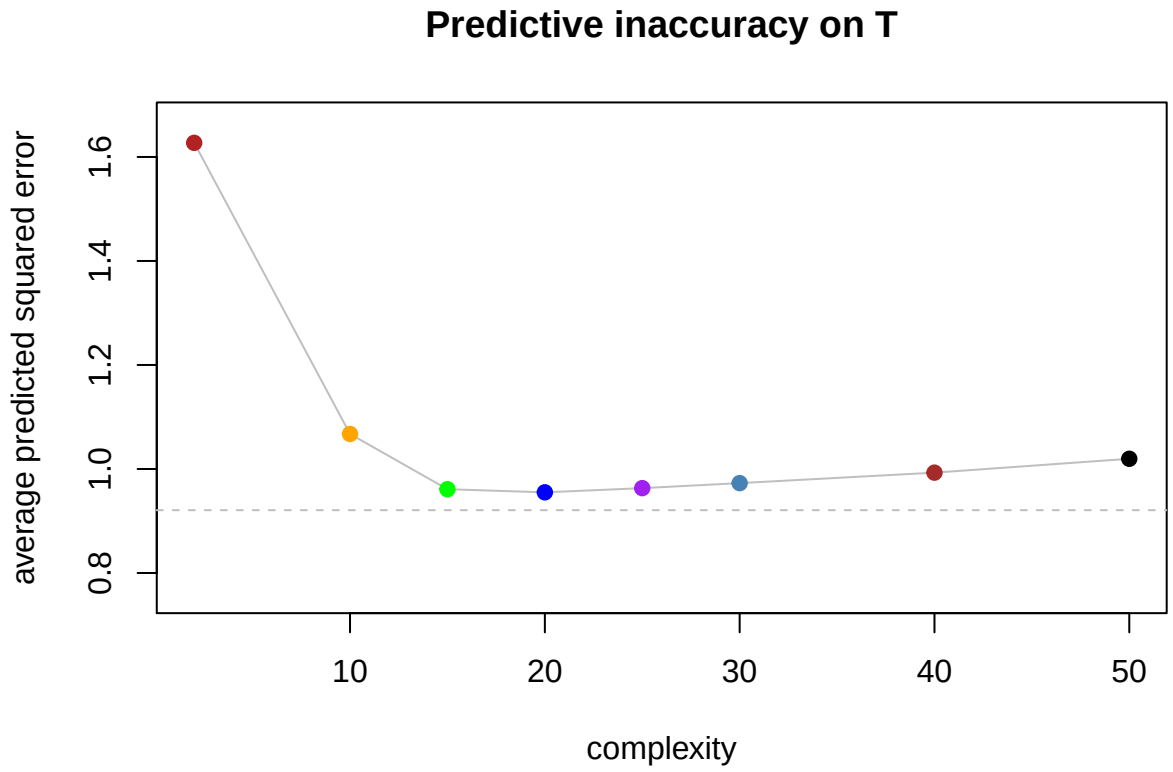
When the average is taken over the values in $\mathcal{S}$ that were also used to construct the predictor functions, we get unnaturally low values for the APSE, which decreases as the complexity for the predictor function increases. To construct a more honest assessment, we evaluate APSE at new data $\mathcal{T}$.

```
xrange <- extendrange(x)
newx <- runif(1000, -pi, pi)
newy <- mu(newx) + rnorm(length(newx), 0, sigma)
```



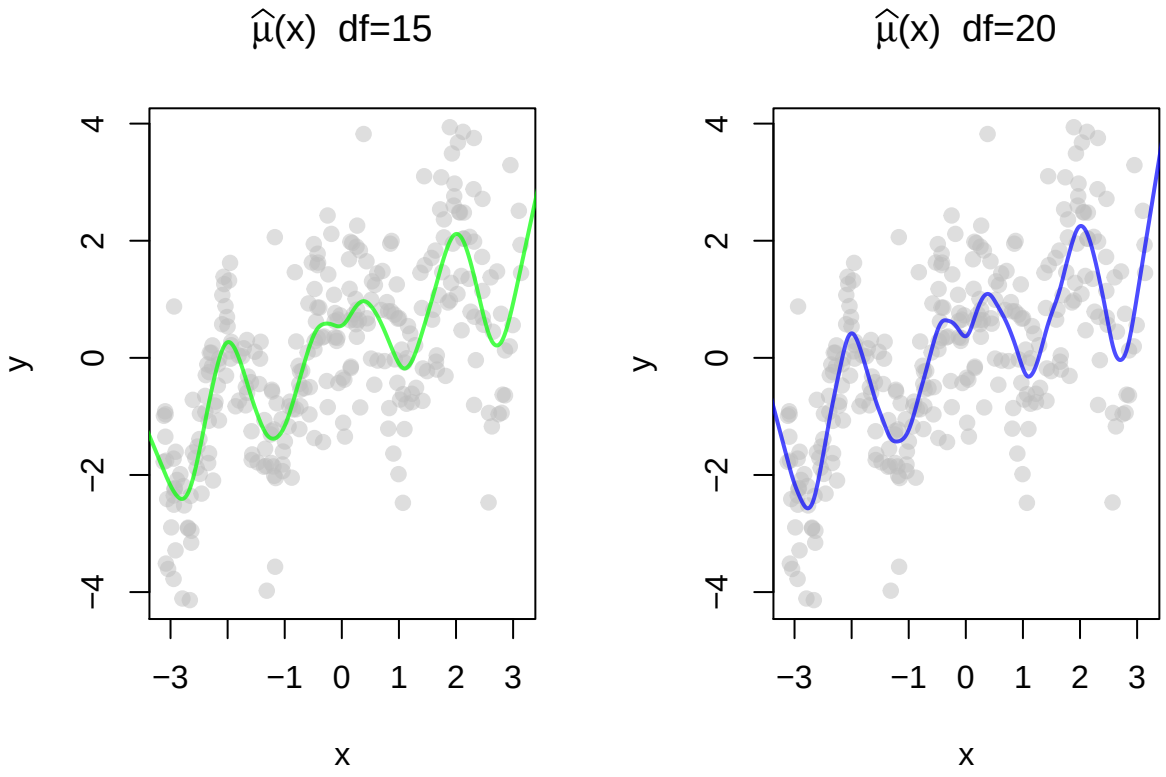
As in Example 1, the average RSS decreases with increasing model complexity, crossing over the theoretical and actual minima for the true predictor function $\mu(x)$ that was used to generate the data.

In contrast, when we look at the prediction accuracies of the same predictor functions but now evaluated on data not in $\mathcal{S}$ but in $\mathcal{T}$, we get the following.



The plot shows a minimum average predicted squared error at the blue point but is fairly flat from the green point to purple (values are 0.9610991, 0.9551148, 0.9631535) and beyond. The squared error rises clearly beyond 30 degrees of freedom.

Looking again at the plots of $\widehat{\mu}$ for 15 and 20, the main difference seems to be that extremes (particularly the cusp at $x = 0$) are slightly better fit for 20 than for 15.



Note that we could always do a finer grid search between 10 and 25, say, by evaluating more predictor functions. This might produce a different minimum.

We might also choose to not increase the complexity that much if the simpler model is easier to understand and predicts nearly as well (Ockham's razor).

2 Comparison over multiple samples.

So far we have been imagining determining the average predicted squared error of an estimate $\widehat{\mu}_{\mathcal{S}}$ on a given sample $\mathcal{S}$ and averaged over all individuals $i \in \mathcal{T} = \mathcal{P} - \mathcal{S}$. Denote this as

$$APSE(\mathcal{S}, \widehat{\mu}_{\mathcal{S}}) = Ave_{i\in\mathcal{T}}(y_i - \widehat{\mu}_{\mathcal{S}}(\mathbf{x}_i))^2$$

just to emphasize that this is being determined with respect to the particular sample $\mathcal{S}$ and the predictor function $\widehat{\mu}_{\mathcal{S}}(x)$ based on that sample. If there were $N_{\mathcal{T}}$ individuals in $\mathcal{T}$, then we could write

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

Note that we could also choose to define the assessment to be taken over the whole of $\mathcal{P}$ (of size N) rather than just $\mathcal{T} = \mathcal{P} - \mathcal{S}$. We could then define

$$\begin{aligned} APSE(\mathcal{S}, \widehat{\mu}_{\mathcal{S}}) &= Ave_{i\in\mathcal{P}}(y_i - \widehat{\mu}_{\mathcal{S}}(\mathbf{x}_i))^2 \\ &= \tfrac{1}{N} \textstyle\sum_{i\in\mathcal{P}} (y_i - \widehat{\mu}_{\mathcal{S}}(\mathbf{x}_i))^2. \end{aligned}$$

This turns out to be a more convenient measurement of inaccuracy for our purposes here and is not very whenever $N_{\mathcal{T}} \approx N$, that is when $\mathcal{T} \approx \mathcal{P}$.

Now suppose that we have $N_{\mathcal{S}}$ samples $\mathcal{S}_j$ for $j = 1, \ldots, N_{\mathcal{S}}$. Then we might look at the average $APSE$ over all of these as a means to assess the quality of a predictor function. With some abuse of notation, we will write this as

$$\begin{aligned} APSE(\mathcal{P}, \widetilde{\mu}) &= Ave_{j=1,\ldots,N_{\mathcal{S}}} \;\; APSE(\mathcal{S}_j, \widehat{\mu}_{\mathcal{S}_j}) \\ &= \tfrac{1}{N_{\mathcal{S}}} \textstyle\sum_{j=1}^{N_{\mathcal{S}}} \;\; APSE(\mathcal{S}_j, \widehat{\mu}_{\mathcal{S}_j}) \\ &= \tfrac{1}{N_{\mathcal{S}}} \textstyle\sum_{j=1}^{N_{\mathcal{S}}} \;\; \tfrac{1}{N} \textstyle\sum_{i\in\mathcal{P}} (y_i - \widehat{\mu}_{\mathcal{S}_j}(\mathbf{x_i}))^2. \end{aligned}$$

Note that now, in place of $\mathcal{T}_j = \mathcal{P} - \mathcal{S}_j$, we are averaging over the entire population $\mathcal{P}$.

Since this is no longer a function for a single sample, but of many samples from $\mathcal{P}$, the first argument is given as $\mathcal{P}$. Similarly, we now use the **estimator** notation $\widetilde{\mu}$ as the second argument in place of the **estimate** notation $\widehat{\mu}$ to emphasize that the function is looking at the values of $\widehat{\mu}$ for all possible samples $calS_j$ of $\mathcal{P}$ for $j = 1, \ldots, N_{\mathcal{S}}$.

2.1 Calculating the $APSE(\mathcal{P}, \widetilde{\mu})$

The average predicted squared error for the estimator $\widetilde{\mu}(\mathbf{x})$ can be shown to be composed of three separate and interpretable pieces when our generative model holds.

First, note that $\mu(\mathbf{x})$ is the conditional mean of y given $\mathbf{x}$ which, given that $\mathcal{P}$ is finite (of size N), is defined as the average of all of the y values in $\mathcal{P}$ that share that value of $\mathbf{x}$. To be precise, suppose that there are K different values of $\mathbf{x}$ in the population $\mathcal{P}$ so that $\mathcal{P}$ can be partitioned according to the different values of k as

$$\mathcal{P} = \bigcup_{k=1}^{K} \mathcal{A}_k$$

where the unique values of $\mathbf{x}$ are $\mathbf{x}_1, \ldots, \mathbf{x}_K$ and

$$\mathcal{A}_k = \{u \mid u \in \mathcal{P},\ \mathbf{x}_u = \mathbf{x}_k\}.$$

(Note that $\mathcal{A}_1 \ldots \mathcal{A}_K$ partition $\mathcal{P}$ since $\mathcal{A}_k \bigcap \mathcal{A}_j = \varnothing$ for all $k \neq j$.)
The conditional mean $\mu(\mathbf{x})$ can now be expressed for each distinct x_k as

$$\mu(\mathbf{x}_k) = Ave_{i \in \mathcal{A}_k}\ y_i.$$

Second, let $\overline{\widehat{\mu}}(\mathbf{x})$ denote the average of the estimated predictor function over all samples

$$\overline{\widehat{\mu}}(\mathbf{x}) = \frac{1}{N_{\mathcal{S}}} \sum_{j=1}^{N_{\mathcal{S}}} \widehat{\mu}_{\mathcal{S}_j}(\mathbf{x}).$$

With these two together, the average predicted squared error for the estimator $\widetilde{\mu}(\mathbf{x})$ can now be written as

$$\begin{aligned}
APSE(\mathcal{P}, \widetilde{\mu}) &= \tfrac{1}{N_{\mathcal{S}}} \textstyle\sum_{j=1}^{N_{\mathcal{S}}}\ \tfrac{1}{N} \sum_{i \in \mathcal{P}} (y_i - \widehat{\mu}_{\mathcal{S}_j}(\mathbf{x_i}))^2 \\
&= \tfrac{1}{N_{\mathcal{S}}} \textstyle\sum_{j=1}^{N_{\mathcal{S}}}\ \tfrac{1}{N} \sum_{i \in \mathcal{P}} (y_i - \mu(\mathbf{x_i}))^2 \\
&\qquad + \tfrac{1}{N_{\mathcal{S}}} \textstyle\sum_{j=1}^{N_{\mathcal{S}}}\ \tfrac{1}{N} \sum_{i \in \mathcal{P}} (\widehat{\mu}_{\mathcal{S}_j}(\mathbf{x_i}) - \mu(\mathbf{x_i}))^2 \\
&= \tfrac{1}{N_{\mathcal{S}}} \textstyle\sum_{j=1}^{N_{\mathcal{S}}}\ \tfrac{1}{N} \sum_{i \in \mathcal{P}} (y_i - \mu(\mathbf{x_i}))^2 \\
&\qquad + \tfrac{1}{N_{\mathcal{S}}} \textstyle\sum_{j=1}^{N_{\mathcal{S}}}\ \tfrac{1}{N} \sum_{i \in \mathcal{P}} (\widehat{\mu}_{\mathcal{S}_j}(\mathbf{x_i}) - \overline{\widehat{\mu}}(\mathbf{x_i}))^2 \\
&\qquad + \tfrac{1}{N_{\mathcal{S}}} \textstyle\sum_{j=1}^{N_{\mathcal{S}}}\ \tfrac{1}{N} \sum_{i \in \mathcal{P}} (\overline{\widehat{\mu}}(\mathbf{x_i}) - \mu(\mathbf{x_i}))^2 \\
&= \tfrac{1}{N} \textstyle\sum_{i \in \mathcal{P}} (y_i - \mu(\mathbf{x_i}))^2 \\
&\qquad + \tfrac{1}{N_{\mathcal{S}}} \textstyle\sum_{j=1}^{N_{\mathcal{S}}}\ \tfrac{1}{N} \sum_{i \in \mathcal{P}} (\widehat{\mu}_{\mathcal{S}_j}(\mathbf{x_i}) - \overline{\widehat{\mu}}(\mathbf{x_i}))^2 \\
&\qquad + \tfrac{1}{N} \textstyle\sum_{i \in \mathcal{P}} (\overline{\widehat{\mu}}(\mathbf{x_i}) - \mu(\mathbf{x_i}))^2 \\
&= Var(y|\mathbf{x}) + Var(\widetilde{\mu}) + Bias^2(\widetilde{\mu}).
\end{aligned}$$

Note that each term is an average over all $\mathbf{x}$ values **and** over all samples. The first is the average conditional variance, the second the average of the variance of the estimator and the last the squared bias. Different estimators $\widetilde{\mu}_{\mathcal{S}}$ will produce different values of each of the last two terms. So too will different sets of samples (so that the choice of sampling plan used to sample from the population will also affect the predictive accuracy). No matter what the estimator, the first term (the residual sampling variability) remains unchanged.

Note also that we could separate $\mathcal{P}$ into the parts that we are used to construct the estimates (viz. samples $\mathcal{S}$) and the parts which are not (viz. $\mathcal{T} - \mathcal{P} - \mathcal{S}$). The average predictive accuracy could then be written as

$$
\begin{aligned}
APSE(\mathcal{P},\widetilde{\mu}) &= \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 \\
&\qquad + \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})-\overline{\widehat{\mu}}(\mathbf{x_i}))^2 \\
&\qquad + \tfrac{1}{N_{\mathcal{S}}}\textstyle\sum_{j=1}^{N_{\mathcal{S}}}\ \tfrac{1}{N}\sum_{i\in\mathcal{P}}(\overline{\widehat{\mu}}(\mathbf{x_i})-\mu(\mathbf{x_i}))^2 \\
&= \tfrac{1}{N_{\mathcal{S}}}\textstyle\sum_{j=1}^{N_{\mathcal{S}}}\ \tfrac{1}{N}\left(\sum_{i\in\mathcal{S}_j}(y_i-\mu(\mathbf{x_i}))^2+\sum_{i\in\mathcal{T}_j}(y_i-\mu(\mathbf{x_i}))^2\right) \\
&\qquad + \tfrac{1}{N_{\mathcal{S}}}\textstyle\sum_{j=1}^{N_{\mathcal{S}}}\ \tfrac{1}{N}\left(\sum_{i\in\mathcal{S}_j}(\widehat{\mu}_{\mathcal{S}_j}(\mathbf{x_i})-\overline{\widehat{\mu}}(\mathbf{x_i}))^2+\sum_{i\in\mathcal{T}_j}(\widehat{\mu}_{\mathcal{S}_j}(\mathbf{x_i})-\overline{\widehat{\mu}}(\mathbf{x_i}))^2\right) \\
&\qquad + \tfrac{1}{N_{\mathcal{S}}}\textstyle\sum_{j=1}^{N_{\mathcal{S}}}\ \tfrac{1}{N}\left(\sum_{i\in\mathcal{S}_j}(\overline{\widehat{\mu}}(\mathbf{x_i})-\mu(\mathbf{x_i}))^2+\sum_{i\in\mathcal{T}_j}(\overline{\widehat{\mu}}(\mathbf{x_i})-\mu(\mathbf{x_i}))^2\right) \\
&= \left\{\tfrac{1}{N_{\mathcal{S}}}\textstyle\sum_{j=1}^{N_{\mathcal{S}}}\ \tfrac{N-N_{\mathcal{T}_j}}{N}\left(\tfrac{1}{N-N_{\mathcal{T}_j}}\sum_{i\in\mathcal{S}_j}(y_i-\mu(\mathbf{x_i}))^2\right)\right. \\
&\qquad + \tfrac{1}{N_{\mathcal{S}}}\textstyle\sum_{j=1}^{N_{\mathcal{S}}}\ \tfrac{N-N_{\mathcal{T}_j}}{N}\left(\tfrac{1}{N-N_{\mathcal{T}_j}}\sum_{i\in\mathcal{S}_j}(\widehat{\mu}_{\mathcal{S}_j}(\mathbf{x_i})-\overline{\widehat{\mu}}(\mathbf{x_i}))^2\right) \\
&\qquad \left. + \tfrac{1}{N_{\mathcal{S}}}\textstyle\sum_{j=1}^{N_{\mathcal{S}}}\ \tfrac{N-N_{\mathcal{T}_j}}{N}\left(\tfrac{1}{N-N_{\mathcal{T}_j}}\sum_{i\in\mathcal{S}_j}(\overline{\widehat{\mu}}(\mathbf{x_i})-\mu(\mathbf{x_i}))^2\right)\right\} \\
&\quad + \left\{\tfrac{1}{N_{\mathcal{S}}}\textstyle\sum_{j=1}^{N_{\mathcal{S}}}\ \tfrac{N_{\mathcal{T}_j}}{N}\left(\tfrac{1}{N_{\mathcal{T}_j}}\sum_{i\in\mathcal{T}_j}(y_i-\mu(\mathbf{x_i}))^2\right)\right. \\
&\qquad + \tfrac{1}{N_{\mathcal{S}}}\textstyle\sum_{j=1}^{N_{\mathcal{S}}}\ \tfrac{N_{\mathcal{T}_j}}{N}\left(\tfrac{1}{N_{\mathcal{T}_j}}\sum_{i\in\mathcal{T}_j}(\widehat{\mu}_{\mathcal{S}_j}(\mathbf{x_i})-\overline{\widehat{\mu}}(\mathbf{x_i}))^2\right) \\
&\qquad \left. + \tfrac{1}{N_{\mathcal{S}}}\textstyle\sum_{j=1}^{N_{\mathcal{S}}}\ \tfrac{N_{\mathcal{T}_j}}{N}\left(\tfrac{1}{N_{\mathcal{T}_j}}\sum_{i\in\mathcal{T}_j}(\overline{\widehat{\mu}}(\mathbf{x_i})-\mu(\mathbf{x_i}))^2\right)\right\} \\
&= \{\text{something based on the } \mathbf{same} \text{ samples used by } \widehat{\mu}\} \\
&\qquad + \{\text{something based on samples } \mathbf{not} \text{ used by } \widehat{\mu}\}\,.
\end{aligned}
$$

If, for example, all samples $\mathcal{S}_j$ were of the same size n and similarly for $\mathcal{T}_j$ (therefore giving $N_{\mathcal{T}_j} = N - n = N_{\mathcal{T}}$), then this could be written as

$$
\begin{aligned}
APSE(\mathcal{P},\widetilde{\mu}) &= \tfrac{n}{N}\Big\{\tfrac{1}{N_{\mathcal{S}}}\textstyle\sum_{j=1}^{N_{\mathcal{S}}}\Big(\tfrac{1}{n}\sum_{i\in\mathcal{S}_j}(y_i-\mu(\mathbf{x_i}))^2\Big) \\
&\qquad + \tfrac{1}{N_{\mathcal{S}}}\textstyle\sum_{j=1}^{N_{\mathcal{S}}}\Big(\tfrac{1}{n}\sum_{i\in\mathcal{S}_j}(\widehat{\mu}_{\mathcal{S}_j}(\mathbf{x_i})-\overline{\widehat{\mu}}(\mathbf{x_i}))^2\Big) \\
&\qquad + \tfrac{1}{N_{\mathcal{S}}}\textstyle\sum_{j=1}^{N_{\mathcal{S}}}\Big(\tfrac{1}{n}\sum_{i\in\mathcal{S}_j}(\overline{\widehat{\mu}}(\mathbf{x_i})-\mu(\mathbf{x_i}))^2\Big)\Big\} \\
&\quad + \left(1-\tfrac{n}{N}\right)\Big\{\tfrac{1}{N_{\mathcal{S}}}\textstyle\sum_{j=1}^{N_{\mathcal{S}}}\Big(\tfrac{1}{N-n}\sum_{i\in\mathcal{T}_j}(y_i-\mu(\mathbf{x_i}))^2\Big) \\
&\qquad + \tfrac{1}{N_{\mathcal{S}}}\textstyle\sum_{j=1}^{N_{\mathcal{S}}}\Big(\tfrac{1}{N-n}\sum_{i\in\mathcal{T}_j}(\widehat{\mu}_{\mathcal{S}_j}(\mathbf{x_i})-\overline{\widehat{\mu}}(\mathbf{x_i}))^2\Big) \\
&\qquad + \tfrac{1}{N_{\mathcal{S}}}\textstyle\sum_{j=1}^{N_{\mathcal{S}}}\Big(\tfrac{1}{N-n}\sum_{i\in\mathcal{T}_j}(\overline{\widehat{\mu}}(\mathbf{x_i})-\mu(\mathbf{x_i}))^2\Big)\Big\} \\
&= \left(\tfrac{n}{N}\right)\{APSE(\mathcal{P},\widetilde{\mu}) \text{ based on the } \textbf{same} \text{ samples used by } \widehat{\mu}\} \\
&\qquad + \left(1-\tfrac{n}{N}\right)\{APSE(\mathcal{P},\widetilde{\mu}) \text{ based on samples } \textbf{not} \text{ used by } \widehat{\mu}\}.
\end{aligned}
$$

Clearly, if $n << N$, then the second term dominates. Sometimes, even if $n \approx N$ we might also want to focus our evaluation only on the second term since this evaluation is based on values not used in the actual estimation process.

We can use these quantities to compare predictors as before. To do so will of course require simulation from our generative model.

We can set up the simulation generally by defining all of the pieces that we need as separate functions.

2.1.1 Generating samples

The generative model we are using requires two distributions, one for X and the other for R.

First for X.

```
#
# The generative model requires a means of generating the x
# values.
# The function will be called rdistx
# In our examples, we used
rdistx <- function(n=1) {runif(n, -pi, pi)}
```

Now for R.

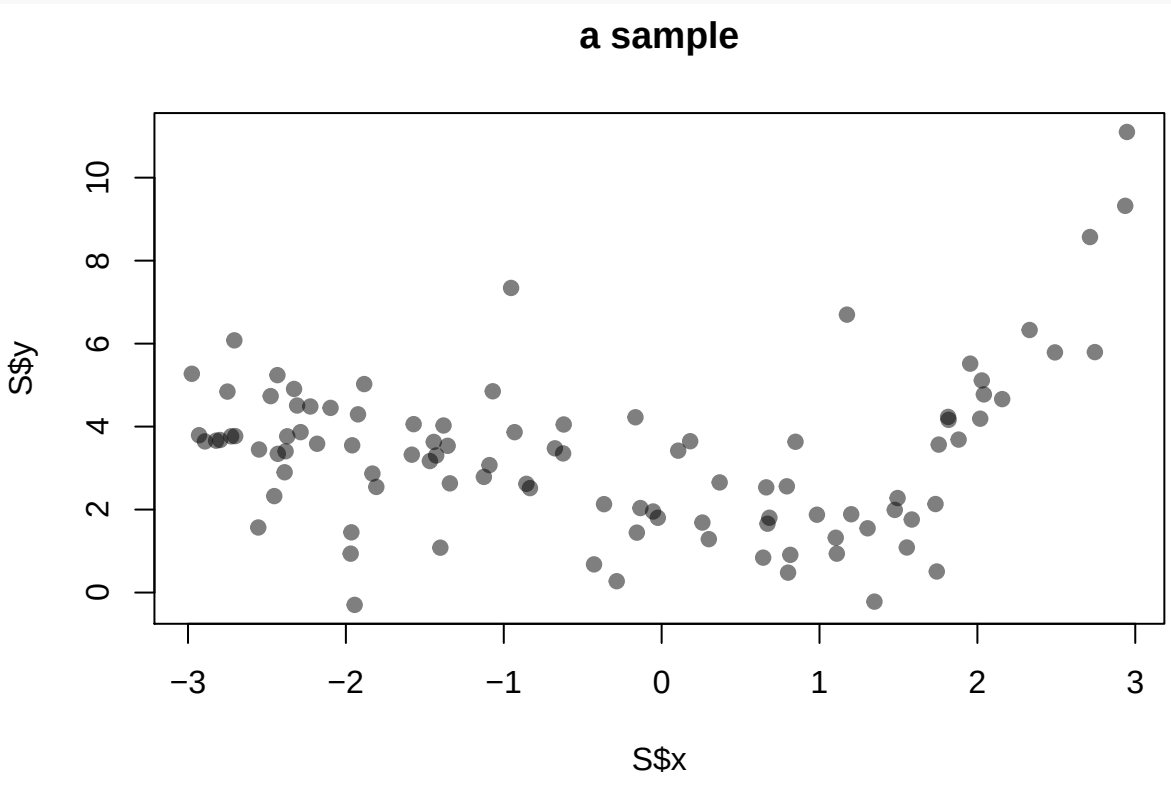
```
# The values of y are generated conditionally on x
# using a mean function, mu(x),
# to which we add a residual value generated independently
# from some distribution.
# In our generative model, we chose this to be N(0, sigma^2)
# and sigma=0.4 for the first example
rdistresid <- function(n=1) {rnorm(n, 0, sd=0.4)}
#
# The generative model therefore produces a sample of
# (x,y) pairs of size N from these component pieces
#
```

A sample of x and y values is generated by first generating x and then a y from the conditional distriution of $Y|X = x$.

```
#
# The function getSample puts it all together.
getSample <- function(N, mu, rdistx, rdistresid) {
  # Note that  mu, rdistx, and rdistresid must be functions
  x <- rdistx(N)
  r <- rdistresid(N)
  # Here we call the function mu
  y <- mu(x) + r
  list(x=x, y=y)
}
```

A sample can now be generated based entirely on $\mu(x)$, and the distributions of X and of R. For example,

```
# For example, you can use these to generate some data as follows:
#
# First set the seed so we are all looking at the same picture
set.seed(34243411)
# generate the sample using your favourite functions
S <- getSample(100,
               mu=function(x){2 - x + 0.5* x^2 + 0.25 * x^3},
               rdistx=function(n){runif(n, -3,3)},
               rdistresid=function(n){rt(n,df=3)})
# have a look
plot(S, main="a sample",
     pch=19, col= adjustcolor("black", 0.5))
```



So the above functions can be used very generally to construct a sample from any response model.

2.1.2 Predictor functions

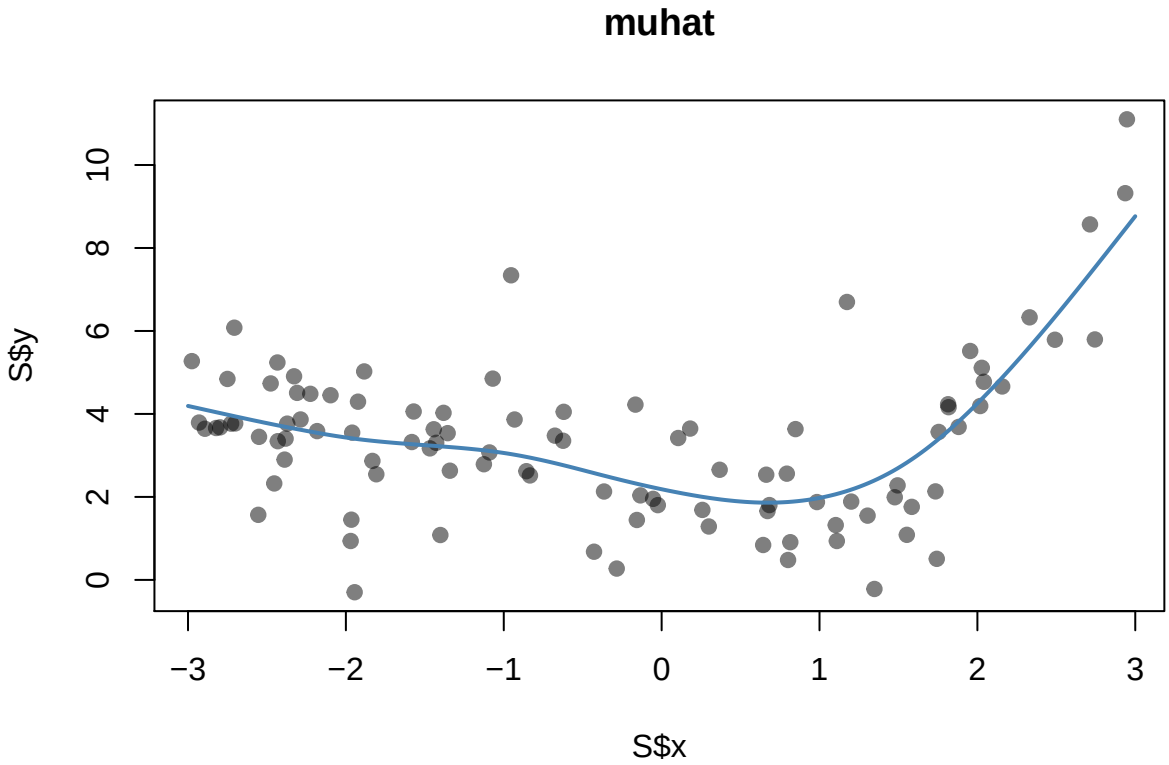
We need to be able to produce a predictor function $\widehat{\mu}_{\mathcal{S}_j}(\mathbf{x})$ that is estimated based on the sample $\mathcal{S}_j$ and which can be evaluated on any x. In the examples, the only predictor functions used were a straight line model and smoothing splines.

The first thing we define then is a function that takes a sample and some estimated degrees and which **returns the function** $\widehat{\mu}_{\mathcal{S}_j}(\mathbf{x})$ which can subsequently be called on any value x.

```
# The predictor functions are determined based on
# a given sample and a measure of complexity (here effective
# degrees of freedom).
# We want to be able to create the muhat **function**
# for any sample and complexity.
# Note that the following function does just that and
# returns the **function** muhat(x) that can be used later.
getmuhat <- function(sample, df) {
  if (df == 2) {
    fit <- lm(y ~ x, data=sample)
    muhat <- function(x){
      predict(fit,newdata=data.frame(x=x))
    }
  } else {
    fit <- smooth.spline(sample$x,
                         sample$y,
                         df = df)
    muhat <- function(x){predict(fit, x=x)$y}
  }
  # muhat is the function that we need to
  # calculate values at any x, so we return this function
  # as the value of getmuhat
  muhat
}
```

To illustrate the point, using the sample we previously generated, we can get a function $\widehat{\mu}_{\mathcal{S}_j}(\mathbf{x})$ and evaluate it on any x.

```
# Example
muhat <- getmuhat(sample=S,df=5)
xvals <- seq(-3,3,length.out=200)
# have a look
plot(S, main="muhat",
     pch=19, col= adjustcolor("black", 0.5))
lines(xvals, muhat(xvals), col="steelblue", lwd=2)
```



Given many samples $\mathcal{S}_j$, $j = 1, \ldots, N_S$, and hence many $\widehat{\mu}_{\mathcal{S}_j}(\mathbf{x})$, we also need the function

$$\overline{\widehat{\mu}}(\mathbf{x}) = \frac{1}{N_{\mathcal{S}}} \sum_{j=1}^{N_{\mathcal{S}}} \widehat{\mu}_{\mathcal{S}_j}(\mathbf{x}).$$

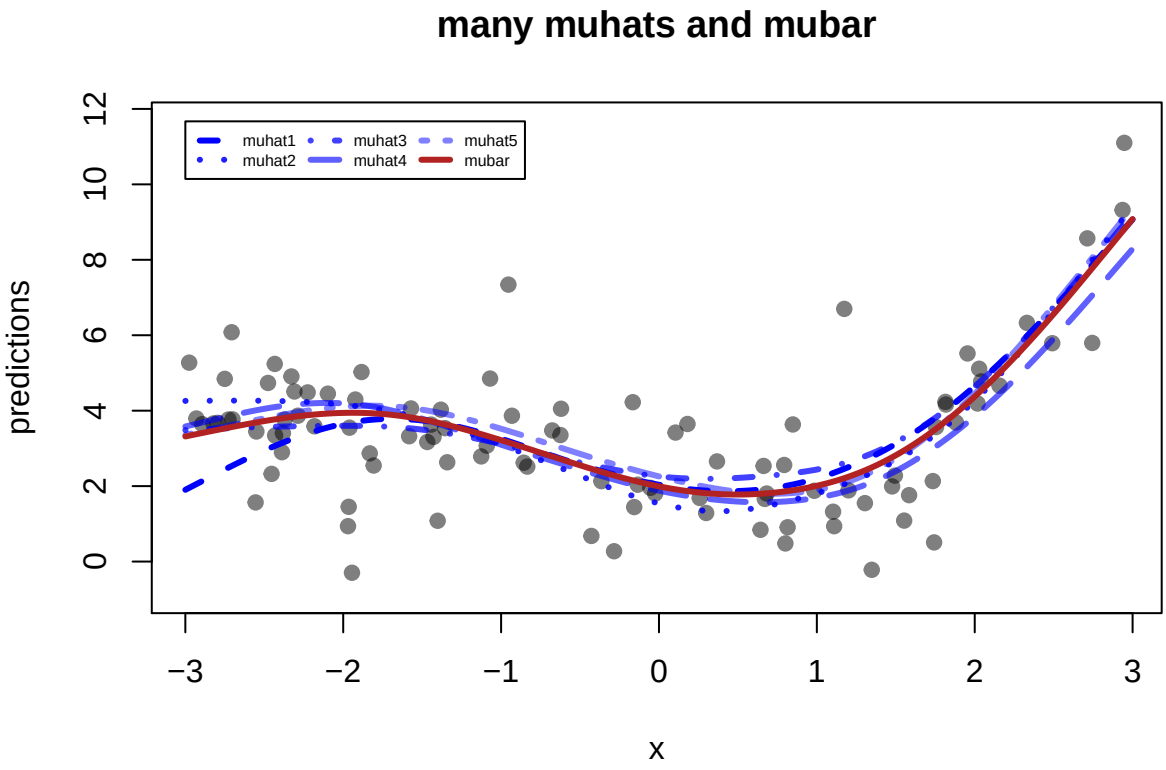
The following function takes a list of the functions $\widehat{\mu}_{\mathcal{S}_j}(\mathbf{x})$ and returns the function $\overline{\widehat{\mu}}(\mathbf{x})$.

```
# We also need a function that will calculate the average
# of the values over a set of functions ... mubar(x) = ave (muhats(x))
# The following function does that.
# It return a **function** mubar(x) that will calculate the average
# of the functions it is given at the value x.
getmubar <- function(muhats) {
  # the muhats must be a list of muhat functions
  # We build and return mubar, the function that
  # is the average of the functions in muhats
  # Here is mubar:
  function(x) {
    # x here is a vector of x values on which the
    # average of the muhats is to be determined.
    #
    # sapply applies the function given by FUN
    # to each muhat in the list muhats
    Ans <- sapply(muhats, FUN=function(muhat){muhat(x)})
    # FUN calculates muhat(x) for every muhat and
    # returns the answer Ans as a matrix having
    # as many rows as there are values of x and
    # as many columns as there are muhats.
    # We now just need to get the average
    # across rows (first dimension)
```

```
    # to find mubar(x) and return it
    apply(Ans, MARGIN=1, FUN=mean)
  }
}
```

To illustrate this, we need many samples $\mathcal{S}_j$ and the corresponding predictor functions $\widehat{\mu}_{\mathcal{S}_j}(\mathbf{x})$.

```
N_S <- 5
Ssamples <- lapply(1:N_S, FUN= function(i) {
  getSample(100,
            mu=function(x){2 - x + 0.5 * x^2 + 0.25 * x^3},
            rdistx=function(n){runif(n, -3,3)},
            rdistresid=function(n){rt(n,df=3)}
  )
}
)
muhats <- lapply(Ssamples, FUN=function(sample) getmuhat(sample, df=5))
mubar <- getmubar(muhats)
# have a look
ylim <- range(sapply(muhats,
                     FUN=function(muhat) {
                       range(muhat(xvals))
                     }
)
)
ylim <- extendrange(c(ylim, S$y))

cols <- colorRampPalette(c(rgb(0,0,1,1), rgb(0,0,1,0.5)),
                         alpha = TRUE)(N_S)

# plot the points from one sample
plot(S,
     pch=19, col= adjustcolor("black", 0.5),
     ylim=ylim, xlab="x", ylab="predictions",
     main="many muhats and mubar"
)

for (i in 1:N_S) {
  lines(xvals, muhats[[i]](xvals), col=cols[i], lwd=3, lty=(i+1))
}

lines(xvals, mubar(xvals), col="firebrick", lwd=3)
legend(-3, max(ylim),
       legend=c("muhat1","muhat2","muhat3",
                "muhat4","muhat5","mubar"),
       col=c(cols, "firebrick"),
       ncol=3, cex=0.5,
       lwd=3, lty=c(2:6,1))
```



2.1.3 Average squared differences

Finally, we need some functions to calculate the average squared differences over any given sample. One of these will be based on the difference between y and any predictor function, the other will be between any two predictor functions.

```
# We also need some average squared error functions.
#
# The first one we need is one that will return the
# average squared error of the predicted predfun(x)
# compared to the actual y values for those x values.
#
ave_y_mu_sq <- function(sample, predfun){
  mean(abs(sample$y - predfun(sample$x))^2)
}

# We will also need to calculate the average difference
# between two different predictor functions over some set
# of x values: Ave ( predfun1(x) - predfun2(x))^2
#
ave_mu_mu_sq <- function(predfun1, predfun2, x){
  mean((predfun1(x) - predfun2(x))^2)
}
```

These would be used as follows:

```
# For  y - muhat(x)
ave_y_mu_sq(S, muhat)
```

```
## [1] 1.697435
```

```
# For muhat - mubar
ave_mu_mu_sq(muhat, mubar, S$x)
```

```
## [1] 0.09387096
```

These functions are then used to determine the average predicted square errors and its three components.

The average predicted squared errors over many samples from $\mathcal{P}$, $APSE(\mathcal{P}, \widetilde{\mu})$ will be a function of the samples $\mathcal{S}_j$ and the samples $\mathcal{T}_j$, as well as the predictor function, which for our examples is determined entirely by the choice of the effective degrees of freedom.

```
# To determine APSE(P, mu_tilde) we need to average over all samples
# the average over all x and y in the test sample
# the squared error of the prediction by muhat based on each sample.
# Since each mutilde in our examples is determined by the
# effective degrees of freedom, apse will have df as its argument.

apse <- function(Ssamples, Tsamples, df){
  # average over the samples S
  #
  N_S <- length(Ssamples)
  mean(sapply(1:N_S,
              FUN=function(j){
                S_j <- Ssamples[[j]]
                # get the muhat function based on
                # the sample S_j
                muhat <- getmuhat(S_j, df=df)
                # average over (x_i,y_i) in a
                # single sample T_j the squares
                # (y - muhat(x))^2
                T_j <- Tsamples[[j]]
                ave_y_mu_sq(T_j,muhat)
              }
  )
  )
}
```

Similar functions can be defined for each of the components of $APSE(\mathcal{P}, \widetilde{\mu})$.

For $Ave_{\mathbf{x}}\{Var(y|\mathbf{x})\}$ we have

```
# To determine Var(y) we need to average over all samples
# the average over all x and y in the test sample
# the squared error of the prediction by mu.

var_y <- function(Ssamples, Tsamples, mu){
  # average over the samples S
  #
  N_S <- length(Ssamples)
  mean(sapply(1:N_S,
              FUN=function(j){
                # average over (x_i,y_i) in a
                # single sample T_j the squares
                # (y - muhat(x))^2
                T_j <- Tsamples[[j]]
                ave_y_mu_sq(T_j,mu)
              }
```

```
  )
  )
}
```

For $Var(\widetilde{\mu})$:

```
# To determine Var(mutilde) we need to average over all samples
# the average over all x and y in the test sample
# the squared difference of the muhat and mubar.
# Since each mutilde in our examples is determined by the
# effective degrees of freedom, apse will have df as its argument.

var_mutilde <- function(Ssamples, Tsamples, df){
  # get the predictor function for every sample S
  muhats <- lapply(Ssamples,
                   FUN=function(sample){
                     getmuhat(sample, df=df)
                   }
  )
  # get the average of these, mubar
  mubar <- getmubar(muhats)

  # average over all samples S
  N_S <- length(Ssamples)
  mean(sapply(1:N_S,
              FUN=function(j){
                # get muhat based on sample S_j
                muhat <- muhats[[j]]
                # average over (x_i,y_i) in a
                # single sample T_j the squares
                # (y - muhat(x))^2
                T_j <- Tsamples[[j]]
                ave_mu_mu_sq(muhat, mubar, T_j$x)
              }
  )
  )
}
```

And for $Bias^2(\widetilde{\mu})$:

```
# To determine bias(mutilde)^2 we need to average over all samples
# the average over all x and y in the test sample
# the squared difference of the muhat and mubar.
# Since each mutilde in our examples is determined by the
# effective degrees of freedom, apse will have df as its argument.

bias2_mutilde <- function(Ssamples, Tsamples, mu, df){
  # get the predictor function for every sample S
  muhats <- lapply(Ssamples,
                   FUN=function(sample) getmuhat(sample, df=df)
  )
  # get the average of these, mubar
  mubar <- getmubar(muhats)

  # average over all samples S
```

```
  N_S <- length(Ssamples)
  mean(sapply(1:N_S,
              FUN=function(j){
                # average over (x_i,y_i) in a
                # single sample T_j the squares
                # (y - muhat(x))^2
                T_j <- Tsamples[[j]]
                ave_mu_mu_sq(mubar, mu, T_j$x)
              }
  )
  )
}
```

If all three components and the total $APSE$ are wanted at once, then it would be best to try to minimize the number of loops. The following function will be more efficient than running each of the four functions separately.

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

  rowMeans(sapply(1:N_S,
                  FUN=function(j){
                    T_j <- Tsamples[[j]]
                    muhat <- muhats[[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
                    # (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)
                  }
  ))
}
```

2.1.4 Putting it all together

The above set of functions can now be put together to evaluate the $APSE(\mathcal{P}, \widetilde{\mu})$ and its components for any predictor function. We need to determine each of the following:

- Details of the generative model:
- N the sample size of interest, i.e. the number of observations in each sample S_j
- `rdistx`, a function that generates a value for X from $F_X(x)$
- `rdistresid`, a function that generates a value for R from $F_R(x)$
- `mu` the conditional mean of $Y|X = x$
- N_S, the number of samples $\mathcal{S}_j$ to generate
- N_T, the number of observations in each sample $\mathcal{T}_j$
- Details of the predictor function, i.e. the effective degrees of freedom, `df`

Given this information, we specify the values as follows:

```
# The generative model details
#
# N, the size of each sample S_j
N <- 100
# function to generate x
rdistx <- function(n){runif(n, -3,3)}
# function to generate r
rdistresid <- function(n){rt(n,df=3)}
# the conditional mean function of y
mu <- function(x){2 - x + 0.5 * x^2 + 0.25 * x^3}

# N_S, the number of samples of S_j
N_S <- 200
# N_T, the size of each test sample of T_j
N_T <- 500
```

With these values, we first generate the needed samples

```
set.seed(34243411)
# We get N_S samples S_j of size N,  j=1, ..., N_S
Ssamples <- lapply(1:N_S, FUN= function(i){
  getSample(N,
            mu= mu,
            rdistx = rdistx,
```

```
            rdistresid = rdistresid
  )
}
)

# And then matching N_S samples T_j, each of size N_T,  j=1, ..., N_S
Tsamples <- lapply(1:N_S, FUN= function(i){
  getSample(N_T,
            mu= mu,
            rdistx = rdistx,
            rdistresid = rdistresid
  )
}
)
```

And using these samples, we calculate $APSE(\mathcal{P}, \widetilde{\mu})$ and its components for any predictor function (by specifying `df`).

First $APSE(\mathcal{P}, \widetilde{\mu})$

```
df <- 5
apse(Ssamples, Tsamples, df=df)
```

```
## [1] 3.221617
```

The $Ave_{\mathbf{x}}\{Var(y|\mathbf{x})\}$

```
var_y(Ssamples, Tsamples, mu=mu)
```

```
## [1] 3.001818
```

The $Var(\widetilde{\mu})$

```
var_mutilde(Ssamples, Tsamples, df=df)
```

```
## [1] 0.1556922
```

and finally $Bias^2(\widetilde{\mu})$

```
bias2_mutilde(Ssamples, Tsamples, mu=mu, df=df)
```

```
## [1] 0.06404989
```

the last three of which sum to $APSE(\mathcal{P}, \widetilde{\mu})$.

Again, if we want all of these, it is more efficient to calculate them at all on the same pass through the data as in

```
apse_all(Ssamples, Tsamples, mu=mu, df=df)
```

```
##        apse var_mutilde       bias2       var_y
##  3.22161679  0.15569216  0.06404989  3.00181762
```

The result will be the same.

2.2 Examples revisited

We now have the tools to use $APSE(\mathcal{P}, \widetilde{\mu})$ as a means to compare different predictor functions for our two examples.

2.2.1 Example 1 revisited: A relatively simple predictor function

The generative model had

$$\begin{aligned} X &\sim U(-\pi, \pi) \\ \mu(x) &= \sin x \quad \text{and} \\ \sigma &= 0.4 \end{aligned}$$

with $N = 100$.

```
# First we set up the size of each collection
#
# N, the size of each sample S_j
N <- 100
# function to generate x
rdistx <- function(n){runif(n, -pi,pi)}
# function to generate r
rdistresid <- function(n){rnorm(N, sd=0.4)}
# the conditional mean function of y
mu <- function(x){sin(x)}

# N_S, the number of samples of S_j
N_S <- 200
# N_T, the size of each test sample of T_j
N_T <- 500

# generate some data; set the seed for reproducibility
set.seed(24553411)
# We get N_S samples S_j of size N,  j=1, ..., N_S
Ssamples <- lapply(1:N_S, FUN= function(i){
  getSample(N,
            mu= mu,
            rdistx = rdistx,
            rdistresid = rdistresid
  )
}
)

# And then matching N_S samples T_j, each of size N_T,  j=1, ..., N_S
Tsamples <- lapply(1:N_S, FUN= function(i){
  getSample(N_T,
            mu= mu,
            rdistx = rdistx,
            rdistresid = rdistresid
  )
}
)
```

With all of the samples in hand, we need now to just calculate the values of $APSE(\mathcal{P}, \widetilde{\mu})$ and its constituent components of which $Var(\widetilde{\mu})$ and $Bias^2(\widetilde{\mu})$ will be of most interest. We'll calculate these for the same predictors as before. These were determined by the degrees of freedom.

Since we want all of the components, we will calculate them using the function `apse_all` as follows. Similar calls to the individual functions could have been made had we been interested only in one of those.

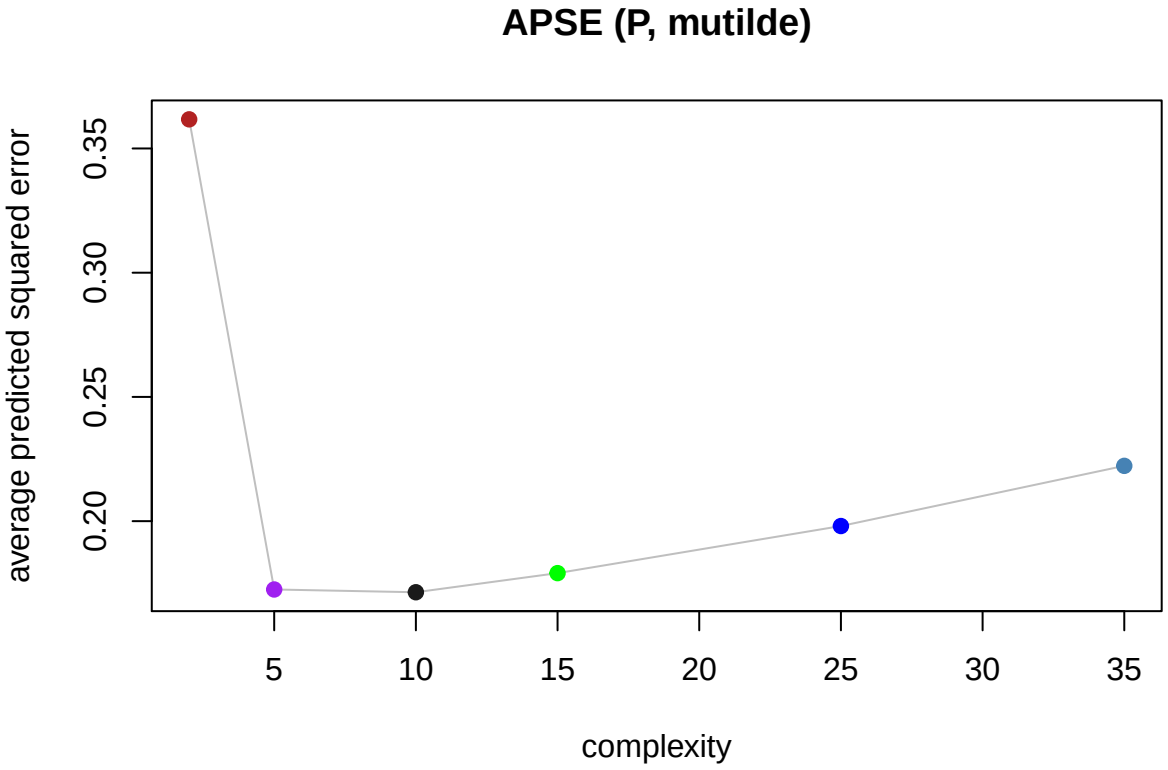
```
# the degrees of freedom associated with each
complexity <- c(2, 5, 10, 15, 25, 35)

apse_vals <-  sapply(complexity,
                     FUN = function(df){
                       apse_all(Ssamples, Tsamples, mu=mu, df=df)
                     }
)

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

```
##      complexity    apse var_mutilde   bias2   var_y
## [1,]          2 0.36171     0.00803 0.19574 0.15817
## [2,]          5 0.17250     0.00697 0.00751 0.15817
## [3,]         10 0.17139     0.01327 0.00012 0.15817
## [4,]         15 0.17908     0.02097 0.00013 0.15817
## [5,]         25 0.19804     0.03971 0.00019 0.15817
## [6,]         35 0.22226     0.06375 0.00030 0.15817
```

Note that `var_y` reproduces $\sigma^2 = (0.4)^2 = 0.16$.

Plotting the values of $APSE(\mathcal{P}, \widetilde{\mu})$ as a function of the complexity (as measured by `df`) will allow a visual comparison of the predictors.

```
# Plot the average predicted squared error of the predictors
plot(complexity, apse_vals["apse",],
     ylab="average predicted squared error",
     main="APSE (P, mutilde)",
     type="n")  # "n" suppresses the plot, add values later
lines(complexity, apse_vals["apse",], col="grey75")
points(complexity, apse_vals["apse",], pch=19,
       col=c("firebrick", "purple",
             "grey10", "green", "blue",
             "steelblue")
)
```



Of the predictor functions we considered, it would seem that the smoothing spline with ten effective degrees of freedom would achieve the lowest $APSE(\mathcal{P}, \widetilde{\mu})$. Again, we could consider looking more closely at smoothing splines having effective degrees of freedom between 5 and 15 to look at finer differences between predictors.

We might also choose 5 (or possibly fewer) degrees of freedom if we choose to trade simplicity (fewer degrees of freedom) for a slight increase in the average predicted error squares.

2.2.1.1 Bias variance tradeoff

The variance:

```
# This may take a little longer to compute
varmutildes <- apse_vals["var_mutilde",]

# Print out the results
cbind(complexity, variance=round(varmutildes,3))
```

```
##      complexity variance
## [1,]          2    0.008
## [2,]          5    0.007
## [3,]         10    0.013
## [4,]         15    0.021
## [5,]         25    0.040
## [6,]         35    0.064
```

We can see the variance growing with complexity beyond `df = 5`.

The squared bias:

```
# This may take a little longer to compute
bias2mutildes <-  apse_vals["bias2",]
```

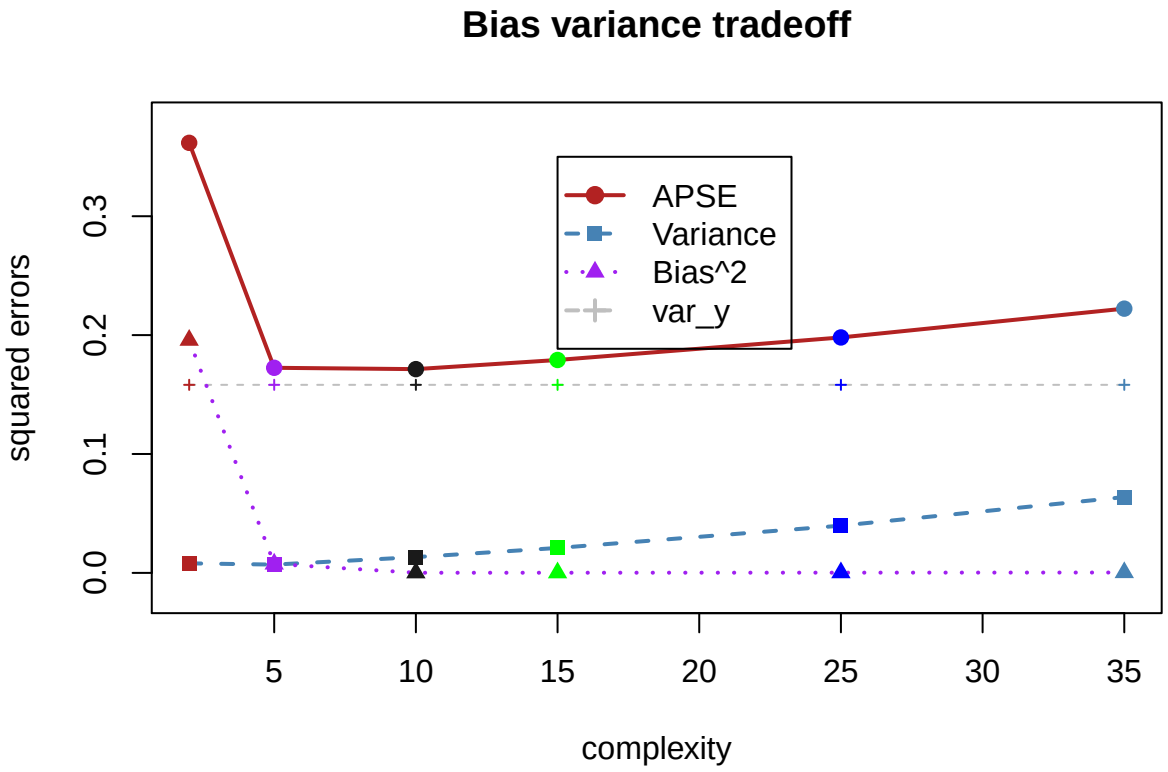
```
# Print out the results
cbind(complexity, bias2=round(bias2mutildes,5))
```

```
##      complexity   bias2
## [1,]          2 0.19574
## [2,]          5 0.00751
## [3,]         10 0.00012
## [4,]         15 0.00013
## [5,]         25 0.00019
## [6,]         35 0.00030
```

We see the bias diminsh vary quickly and then eventually slowly rise again.

Plot them all together

```
apsemutildes <-  apse_vals["apse",]
# All together now
ylim <- extendrange(apse_vals)
plot(complexity, varmutildes,
     ylab="squared errors", ylim=ylim,
     main="Bias variance tradeoff",
     type="n")  # "n" suppresses the plot, add values later

lines(complexity, apse_vals["var_y",], lwd=1, col="grey75", lty=2)
points(complexity, apse_vals["var_y",], pch=3, cex=0.5,
       col=c("firebrick", "purple",
             "grey10", "green", "blue",
             "steelblue"))
lines(complexity, apsemutildes, lwd=2, col="firebrick")
points(complexity, apsemutildes, pch=19,
       col=c("firebrick", "purple",
             "grey10", "green", "blue",
             "steelblue"))
lines(complexity, varmutildes, lwd=2, lty=2, col="steelblue")
points(complexity, varmutildes, pch=15,
       col=c("firebrick", "purple",
             "grey10", "green", "blue",
             "steelblue"))
lines(complexity, bias2mutildes, lwd=2, lty=3, col="purple")
points(complexity, bias2mutildes, pch=17,
       col=c("firebrick", "purple",
             "grey10", "green", "blue",
             "steelblue"))
legend(15, 0.35, legend=c("APSE", "Variance", "Bias^2", "var_y"),
       lty=c(1,2,3,2), lwd=2, pch=c(19,15,17,3),
       col=c("firebrick", "steelblue", "purple", "grey75"))
```



2.2.2 Example 2 revisited: A more complex predictor function

The generative model had

$$\begin{aligned} X &\sim U(-\pi, \pi) \\ \mu(x) &= x/2 + |x| \times (cos(x/2) + cos(3x/2) + cos(5x/2) + cos(7x/2)) \quad \text{and} \\ \sigma &= 1 \end{aligned}$$

with $N = 100$.

```
# First we set up the size of each collection
#
# N, the size of each sample S_j
N <- 300
# function to generate x
rdistx <- function(n){runif(n, -pi,pi)}
# function to generate r
rdistresid <- function(n){rnorm(n, sd=1)}
# the conditional mean function of y
mu <- function(x) {
x/2 + abs(x) * (cos(x /2) +
cos(3 * x /2) +
cos(5 * x /2) +
cos(7 * x /2)
)
}

# N_S, the number of samples of S_j
```

```
N_S <- 200
# N_T, the size of each test sample of T_j
N_T <- 500

# generate some data; set the seed for reproducibility
set.seed(12032345)
# We get N_S samples S_j of size N,  j=1, ..., N_S
Ssamples <- lapply(1:N_S, FUN= function(i){
getSample(N,
mu= mu,
rdistx = rdistx,
rdistresid = rdistresid
)
}
)

# And then matching N_S samples T_j, each of size N_T,  j=1, ..., N_S
Tsamples <- lapply(1:N_S, FUN= function(i){
getSample(N_T,
mu= mu,
rdistx = rdistx,
rdistresid = rdistresid
)
}
)
```

With all of the samples in hand, we need now to just calculate the values of $APSE(\mathcal{P}, \widetilde{\mu})$ and the two components of greatest interest, $Var(\widetilde{\mu})$ and $Bias^2(\widetilde{\mu})$.

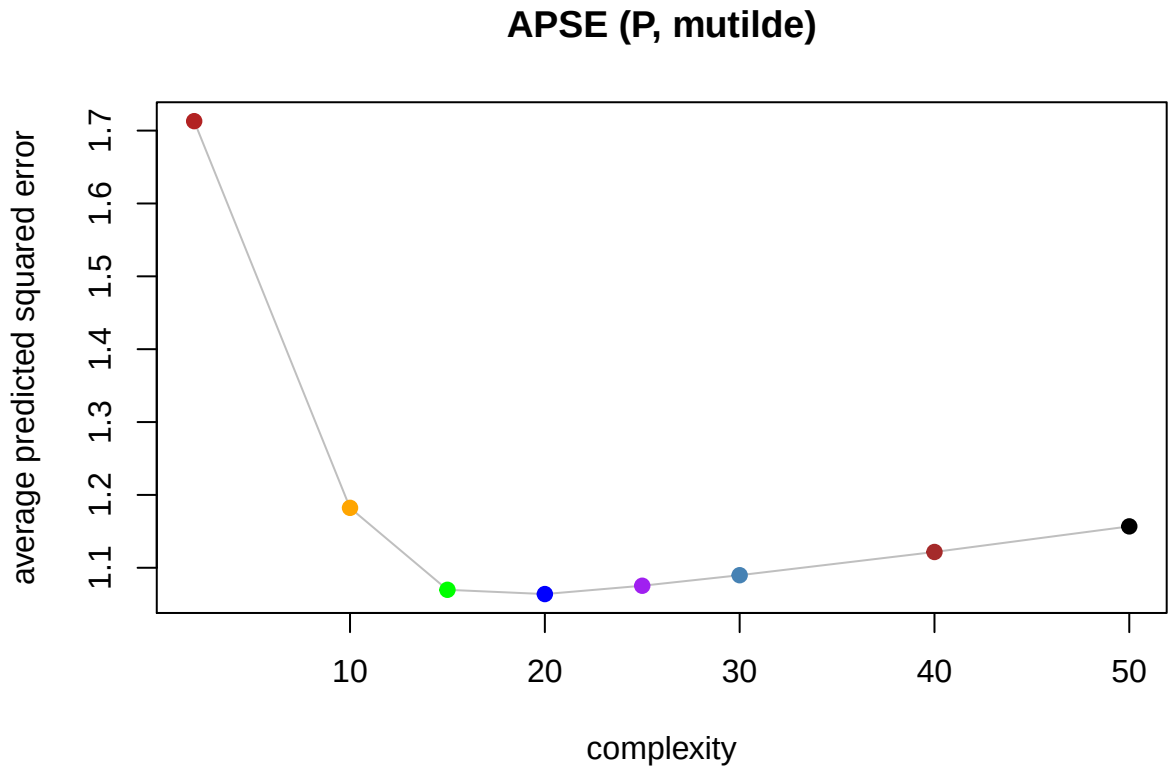
We'll calculate these for the same predictors as before. These were determined by the degrees of freedom.

```
# the degrees of freedom associated with each
complexity <- c(2, 10, 15, 20, 25, 30, 40, 50)

apse_vals <-  sapply(complexity,
                     FUN = function(df){
                       apse_all(Ssamples, Tsamples, mu=mu, df=df)
                     }
)

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

```
##      complexity    apse var_mutilde   bias2   var_y
## [1,]          2 1.71296     0.01184 0.69019 1.00672
## [2,]         10 1.18225     0.03132 0.14542 1.00672
## [3,]         15 1.06978     0.04076 0.02427 1.00672
## [4,]         20 1.06394     0.05370 0.00531 1.00672
## [5,]         25 1.07537     0.06803 0.00194 1.00672
## [6,]         30 1.08981     0.08283 0.00105 1.00672
## [7,]         40 1.12167     0.11410 0.00064 1.00672
## [8,]         50 1.15683     0.14853 0.00063 1.00672
```

Note that `var_y` reproduces $\sigma^2 = 1$.

```
apsemutildes <- apse_vals["apse",]
# Plot the average predicted squared error of the predictors
plot(complexity, apsemutildes,
     ylab="average predicted squared error",
     main="APSE (P, mutilde)",
     type="n")  # "n" suppresses the plot, add values later
lines(complexity, apsemutildes, col="grey75")
points(complexity, apsemutildes, pch=19,
       col=c("firebrick", "orange",
             "green", "blue", "purple", "steelblue",
             "brown", "black")
)
```



Of the predictor functions we considered, it would seem that the smoothing spline with 20 effective degrees of freedom would achieve the lowest $APSE(\mathcal{P}, \widetilde{\mu})$. Again, we could consider looking more closely at smoothing splines having effective degrees of freedom between 5 and 25 to look at finer differences between predictors.

We might also choose 10 (or possibly fewer) degrees of freedom if we choose to trade simplicity (fewer degrees of freedom) for a slight increase in the average predicted error squares.

2.2.2.1 Bias variance tradeoff

The variance:

```
# This may take a little longer to compute
varmutildes <- apse_vals["var_mutilde",]
# Print out the results
cbind(complexity, variance=round(varmutildes,3))
```

```
##      complexity variance
## [1,]          2    0.012
```

```
## [2,]         10    0.031
## [3,]         15    0.041
## [4,]         20    0.054
## [5,]         25    0.068
## [6,]         30    0.083
## [7,]         40    0.114
## [8,]         50    0.149
```

We can see the variance strictly growing with complexity.

The squared bias:

```
# This may take a little longer to compute
bias2mutildes <-  apse_vals["bias2",]
# Print out the results
cbind(complexity, bias2=round(bias2mutildes,5))
```

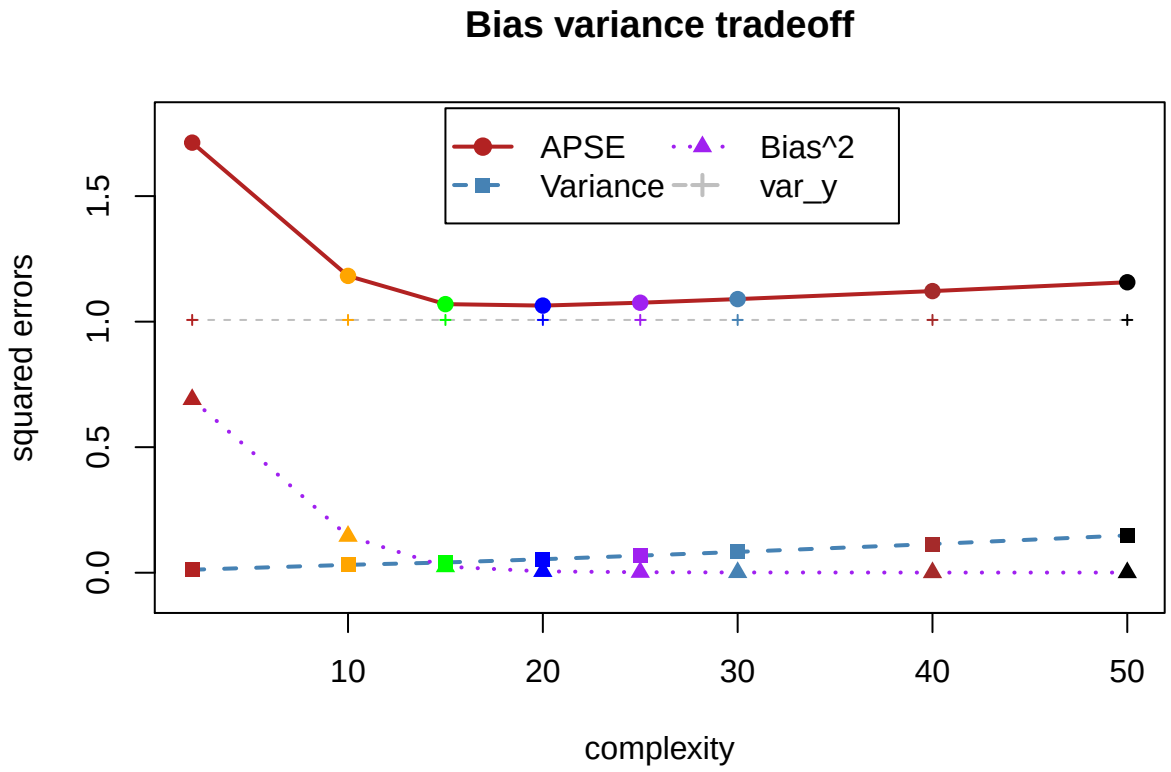
```
##      complexity   bias2
## [1,]          2 0.69019
## [2,]         10 0.14542
## [3,]         15 0.02427
## [4,]         20 0.00531
## [5,]         25 0.00194
## [6,]         30 0.00105
## [7,]         40 0.00064
## [8,]         50 0.00063
```

We see the bias d is strictly decreasing with increasing complexity (`df`).

Plot them all together

```
# All together now
ylim <- extendrange(apse_vals)
plot(complexity, varmutildes,
     ylab="squared errors", ylim=ylim,
     main="Bias variance tradeoff",
     type="n")  # "n" suppresses the plot, add values later

lines(complexity, apse_vals["var_y",], lwd=1, col="grey75", lty=2)
points(complexity, apse_vals["var_y",], pch=3, cex=0.5,
       col=c("firebrick", "orange",
             "green", "blue", "purple", "steelblue",
             "brown", "black"))
lines(complexity, apsemutildes, lwd=2, col="firebrick")
points(complexity, apsemutildes, pch=19,
       col=c("firebrick", "orange",
             "green", "blue", "purple", "steelblue",
             "brown", "black"))
lines(complexity, varmutildes, lwd=2, lty=2, col="steelblue")
points(complexity, varmutildes, pch=15,
       col=c("firebrick", "orange",
             "green", "blue", "purple", "steelblue",
             "brown", "black"))
lines(complexity, bias2mutildes, lwd=2, lty=3, col="purple")
points(complexity, bias2mutildes, pch=17,
       col=c("firebrick", "orange",
             "green", "blue", "purple", "steelblue",
             "brown", "black"))
```

```
legend(15, 1.85, legend=c("APSE", "Variance", "Bias^2", "var_y"),
       lty=c(1,2,3,2), lwd=2, pch=c(19,15,17,3), ncol=2,
       col=c("firebrick", "steelblue", "purple", "grey75"))
```



Again, the plot makes clear the tradeoff between bias and variance.

3 The harsh reality

As the examples show, predictive accuracy provides insight into the performance of a predictor and can be used to choose between competing ones. The key to this usefulness however is that the predictive accuracy can be measured on population $\mathcal{P}$ about which we we want to make inference.

Unfortunately, reality rarely provides us with data in $\mathcal{P}$ that is not in our sample $\mathcal{S}$. That is, we typically have no more than $\mathcal{S}$ and so have neither $\mathcal{P}$ nor $\mathcal{T} = \mathcal{P} - \mathcal{S}$.

The situation is not however unfamiliar – it is the basic problem of inductive inference. Experience there says that whenever interest lies in some attribute of the population, $a(\mathcal{P})$ say, we might use $a(\mathcal{S})$ as an estimate of that attribute. This can be fairly reliable for many attributes based on properly selected samples $\mathcal{S}$.

3.1 Predictive accuracy as a population attribute

Analogously, we cast predictive accuracy as an attribute of the $\mathcal{P}$ and then use the corresponding attribute evaluated on $\mathcal{S}$ as an its estimate.

We could, for example choose to make comparisons based on the population attribute:

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

which has, as part of its definition, the particular sample $\mathcal{S}$ that was chosen. Call this the **single subset version**.

Alternatively, we might choose to population attribute based on multiple (e.g. all possible) samples $\mathcal{S}$ from $\mathcal{P}$. This can be written as

$$APSE(\mathcal{P}, \widetilde{\mu}) = \frac{1}{N_{\mathcal{S}}} \sum_{j=1}^{N_{\mathcal{S}}} APSE(\mathcal{S}_j, \widehat{\mu}_{\mathcal{S}_j}).$$

This depends on all of the samples $\mathcal{S}_1, \ldots, \mathcal{S}_{N_{\mathcal{S}}}$ but remains a population attribute nevertheless. Call this the **multiple subset version**.

In either case, the attribute is a function of both the estimated predictor function $\widehat{\mu}(\mathbf{x})$ **and** of the samples used. These are two distinct population attributes, each a slightly different measure of an average prediction squared error.

Finally, since we are really more concerned with how well each estimator performs on that part of the population which was **not** used to construct the estimate, rather than average over all units in the population $\mathcal{P}$ we average over those units in $\mathcal{T} = \mathcal{P} - \mathcal{S}$. This change can be particularly valuable when $N_{\mathcal{T}} << N$.

Even with $\mathcal{P}$ replaced by $\mathcal{T}$ (or by $\mathcal{T}_j$) in the above, the two accuracy measures remain attributes of the population $\mathcal{P}$.

We now visit each of these in turn.

3.1.1 The single subset version

The first is based on a single subset $\mathcal{S}$ and a single predictor function $\widehat{\mu}_{\mathcal{S}}(\mathbf{x})$ constructed from that one sample $\mathcal{S}$. The prediction errors are evaluated on the single set $\mathcal{T} = \mathcal{P} - \mathcal{S}$.

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

is seen to be a population attribute like any other $a(\mathcal{P})$. It might seem unusual because it distinguishes subsets of $\mathcal{P}$ and uses them in different ways.

Following the same logic, we use the sample $\mathcal{S}$ in place of $\mathcal{P}$ to estimate the $APSE(\mathcal{S}, \widehat{\mu}_{\mathcal{S}})$. To emphasize that we now want to think of $\mathcal{S}$ as if it were the population $\mathcal{P}$, we let $\mathcal{P}_0$ denote $\mathcal{S}$.

Given our population substitute $\mathcal{P}_0$, we choose a subset $\mathcal{S}_0$ from $\mathcal{P}_0$ and denote its complement in $\mathcal{P}_0$ by $\mathcal{T}_0$. The attribute evaluated on $\mathcal{P}_0 = \mathcal{S}$ is then simply

$$APSE(\mathcal{S}_0, \widehat{\mu}_{\mathcal{S}_0}) = \frac{1}{N_{\mathcal{T}_0}} \sum_{i \in \mathcal{T}_0} (y_i - \widehat{\mu}_{\mathcal{S}_0}(\mathbf{x}_i))^2.$$

This could then serve as an estimate of $APSE(\mathcal{S}, \widehat{\mu}_{\mathcal{S}})$ and be used to choose between competing predictors $\widehat{\mu}$ as before. Given this notation, we could write

$$\widehat{APSE}(\mathcal{S}, \widehat{\mu}_{\mathcal{S}}) = APSE(\mathcal{S}_0, \widehat{\mu}_{\mathcal{S}_0}).$$

Because the estimate $\widehat{\mu}_{\mathcal{S}_0}(\mathbf{x})$ is determined only from observations in $\mathcal{S}_0$, this collection of observations is sometimes called the **training** set. The language is based on the metaphor that estimation of a prediction function is like **learning** the predictor function from the data (and we sometimes say $\mathcal{S}_0$ is used to "train" the predictorfunction).

Analogously, the set $\mathcal{T}_0$ is also often called the **test** set since it is used to asess the quality of the "learning". The test set is also more traditionally called a **hold-out sample** to not be used in estimation but to assess the quality of prediction. For the same reason it has also long been called a **validation** set.

The question of course arises as to how to pick $\mathcal{S}_0$ from $\mathcal{P}_0$.

3.1.2 The multiple subset version

The second predictive accuracy measure differs from the first principally in that it is based on many subsets, $\mathcal{S}_j$ of $\mathcal{P}$, for each of which a predictor $\widehat{\mu}_{\mathcal{S}_j}(\mathbf{x})$ is constructed and its performance evaluated on the complement set, $\mathcal{T}_j$. The average is taken of these performances over all $N_{\mathcal{S}}$ possible samples.

$$APSE(\mathcal{P}, \widetilde{\mu}) = \frac{1}{N_{\mathcal{S}}} \sum_{j=1}^{N_{\mathcal{S}}} APSE(\mathcal{S}_j, \widehat{\mu}_{\mathcal{S}_j})$$

is a population attribute that uses many subsets in its definition.

The sample version of this simply replaces $\mathcal{P}$ by the sample in hand, again now denoted as $\mathcal{P}_0 = \mathcal{S}$. That is we write it as

$$APSE(\mathcal{P}_0, \widetilde{\mu}) = \frac{1}{N_{\mathcal{S}}} \sum_{j=1}^{N_{\mathcal{S}}} APSE(\mathcal{S}_j, \widehat{\mu}_{\mathcal{S}_j}).$$

Notationally the only difference here is that the first argument of the $APSE$ is now $\mathcal{P}_0$ and **not** $\mathcal{P}$. The consequence is that each S_j in the definition is now a subset of $\mathcal{P}_0$ and $\mathcal{T}_j$ is its complement in $\mathcal{P}_0$ **not** in $\mathcal{P}$. Given this notation, we could write

$$\widehat{APSE}(\mathcal{P}, \widetilde{\mu}) = APSE(\mathcal{P}_0, \widetilde{\mu}).$$

As with a single subset, the question remains as to how to pick the subsets $\mathcal{S}_j$ from $\mathcal{P}_0$ for $j = 1, \ldots, N_{\mathcal{S}}$.

3.2 Choosing the subsets

It is not always obvious how one should choose $\mathcal{S}_j$ and $\mathcal{T}_j$ in a given situation.

One guide is that the method of selecting $\mathcal{S}_j$ from $\mathcal{P}_0$ should be as similar as possible to that of selecting the sample $\mathcal{S}$ from the study population $\mathcal{P}$. For example, if $\mathcal{S}$ is a sample chosen at random from $\mathcal{P}$, then so should $\mathcal{S}_j$ be one chosen at random from $\mathcal{P}_0 = \mathcal{S}$. Typically this is what is done. However in general there could be different choices made depending on other aspects of the scientific context.

Suppose each subset $\mathcal{S}_j$ is to be chosen at random from $\mathcal{P}_0$. There are still several questions to ask. First, should the sampling be done with, or without, replacement? Second, how large should each sample $\mathcal{S}_j$ be? Should $\mathcal{T}_j$ be the full complement of $\mathcal{S}_j$ or just a sample from the complement? If the latter, how large should that $\mathcal{T}_j$ be? Finally, how many samples S_j should we take? One? Many? How many?

To address the first, sampling with replacement would allow the possibility that the observations used in constructing the predictor function might also be used to assess its performance. Since the predictive accuracy is meant to be an "out-of-sample" assessment, it would seem more prudent to restrict ourselves to sampling without replacement. Sampling without replacement reduces the possibility of overestimating the predictor's accuracy.

As to how large the sample should be, we can get some insight from the fact that the predicted squared errors are averaged. For example, we know that if we have $N_{\mathcal{T}_j}$ observations in a test set $\mathcal{T}_j$, then the standard deviation of an average over the set will decrease as $N_{\mathcal{T}_j}$ increases proportionately to $1/\sqrt{N_{\mathcal{T}_j}}$. The larger is $N_{\mathcal{T}_j}$ the better (i.e. less variable) will be our estimate of the average squared error.

Conversely, the larger is $\mathcal{T}_j$ the smaller is $\mathcal{S}_j$, the size of the training set. The smaller the training set is, the lower is the quality of the predictor function $\widehat{\mu}_{\mathcal{S}_1}(\mathbf{x})$ on that training set. That could easily lead to systematically underestimating the predictor accuracy for the full population.

Choosing a sample size requires some tradeoff between the variability and the bias of the estimate. The sample size of $\mathcal{S}_j$ needs to be large enough to ensure that the predictor function will have stabilized and have low squared error, yet small enough so that its complement $\mathcal{T}_j$ is large enough to have small variability in estimating the prediction error over $\mathcal{T}_j$.

3.2.1 By partitioning $\mathcal{P}_0$

A simple way to create a sample $\mathcal{S}_j$ is to partition $\mathcal{P}_0$ into pieces, or groups, and then select some groups to form $\mathcal{S}_j$ and the remainder to form $\mathcal{T}_j$.

Typically, $\mathcal{P}_0$ is partitioned into k groups $G_1, G_2, \ldots, G_k$ of equal size (approximately equal in practice). We call this a k-**fold partition** of $\mathcal{P}_0$:

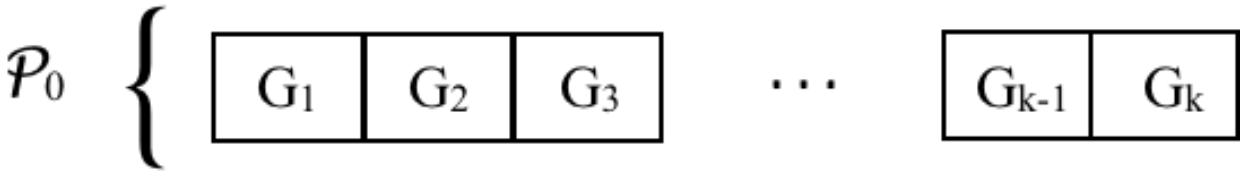


Figure 1: k-fold partition figure

Selecting any set of groups from the partition will define a sample $\mathcal{S}_j$ and the remaining groups will define its complement $\mathcal{T}_j$.

The most common (and simplest) means of selecting the groups would be to select $k-1$ groups to form $\mathcal{S}_j$ and the remaining groups to form $\mathcal{T}_j$. For example, when $k = 5$ we have the following partition of $\mathcal{P}_0$ with the green groups forming the sample and the red group forming the test.

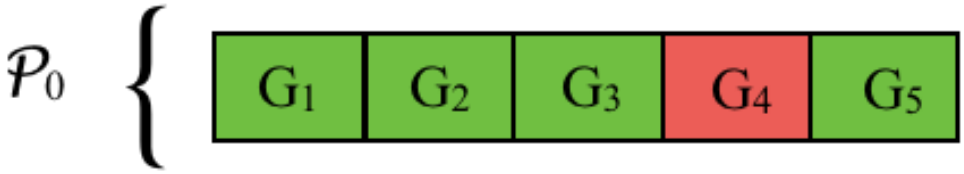


Figure 2: 5-fold partition figure

In this case, $\mathcal{S}_j = G_1 \cup G_2 \cup G_3 \cup G_5$ and $\mathcal{T}_j = G_4$.

Note that for a k-fold partition there can only be k different pairs of sample $\mathcal{S}_j$ and test set $\mathcal{T}_j$. That is $N_{\mathcal{S}} = k$.

Calculating $APSE(\mathcal{P}_0, \widetilde{\mu})$ using sampling that selects all $k-1$ groups from a k-fold partition is known as k-**fold cross-validation** in the literature.

Several questions remain. For example,

- What value should k take?
  Clearly a large value of k will produce a large sample $\mathcal{S}_j$ but a smaller test set $\mathcal{T}_j$. A predictor based on a larger $\mathcal{S}_j$ (i.e. larger k) should be closer to that based on all of $\mathcal{S}$; one based on smaller $\mathcal{S}_j$ (i.e. smaller k) should perform more poorly (being based on fewer observations) and so tend to systematically overestimate the prediction error.
- How should the partition be constructed?
  Simple random sampling is the obvious choice, but there may be contexts where other sampling protocols might also be considered.
- Should we consider only one such partition?
  If we have p partitions, then $N_{\mathcal{S}} = p \times k$ and our estimate $APSE(\mathcal{P}_0, \widetilde{\mu})$ of $APSE(\mathcal{P}, \widetilde{\mu})$ should be less variable. Note, however, the larger the overlap in the samples $\mathcal{S}_j$ (i.e. the larger the value of k) the greater will be the correlation between the estimated predictor functions in each element of the sum.

In determining the variance of the sum then, a $2 \times Cov(\widehat{\mu}_{\mathcal{S}_j}, \widehat{\mu}_{\mathcal{S}_k})$ term would be positive and inflating the variance of the average.

The first and third points suggest that even here a bias-variance trade-off must exist for selecting k.

3.2.2 Some k-fold partition functions

Let us assume that we are going to form a k-fold partition by selecting the groups as a random sample (without replacement) from $\mathcal{P}_0$. A function to do that in R can be written as follows:

```
kfold <- function(N, k=N, indices=NULL){
  # get the parameters right:
  if (is.null(indices)) {
    # Randomize if the index order is not supplied
    indices <- sample(1:N, N, replace=FALSE)
  } else {
    # else if supplied, force N to match its length
    N <- length(indices)
  }
  # Check that the k value makes sense.
  if (k > N) stop("k must not exceed N")
  #

  # How big is each group?
  gsize <- rep(round(N/k), k)

  # For how many groups do we need odjust the size?
  extra <- N - sum(gsize)

  # Do we have too few in some groups?
  if (extra > 0) {
    for (i in 1:extra) {
      gsize[i] <- gsize[i] +1
    }
  }
  # Or do we have too many in some groups?
  if (extra < 0) {
    for (i in 1:abs(extra)) {
      gsize[i] <- gsize[i] - 1
    }
  }

  running_total <- c(0,cumsum(gsize))

  # Return the list of k groups of indices
  lapply(1:k,
         FUN=function(i) {
           indices[seq(from = 1 + running_total[i],
                       to = running_total[i+1],
                       by = 1)
                   ]
         }
  )
}
```

The function `kfold` returns a list of the groups. We can use these to construct the sample pairs $(\mathcal{S}_j, \mathcal{T}_j)$ as follows.

```
getKfoldSamples <- function (x, y, k, indices=NULL){
  groups <- kfold(length(x), k, indices)
  Ssamples <- lapply(groups,
                     FUN=function(group) {
                       list(x=x[-group], y=y[-group])
                     })
  Tsamples <- lapply(groups,
                     FUN=function(group) {
                       list(x=x[group], y=y[group])
                     })
  list(Ssamples = Ssamples, Tsamples = Tsamples)
}
```

We can now have this simple function generate sets $\mathcal{S}_j$ and $\mathcal{T}_j$ from $\mathcal{P}_0$ for $j = 1, \ldots, k$.

The same `apse` and `var_mutilde` functions that we used before on $\mathcal{P}$ can now be used on $\mathcal{P}_0 = \mathcal{S}$. All that they require are the sets $\mathcal{S}_j$ and $\mathcal{T}_j$ and the degrees of freedom for the predictor function.

3.2.3 Leave one out

A common choice for the k-fold partition is $k = N$ where N is the size of $\mathcal{P}_0$. In this case each group G_i contains exactly one individual, say i, of $\mathcal{P}_0$.

A sample S_j then is always of size $N-1$ and the test set $\mathcal{T}_j$ is of size 1. That is $\mathcal{S}_j$ is formed from $\mathcal{P}_0$ by simply "leaving one out" of $\mathcal{P}_0$. This N-fold cross-validation is also known as **leave one out cross-validation**.

This answers the "number of partitions" question in as much as there is only one N-fold partition. It does mean that there will be at least as many subset pairs as there are observations, namely N. If N is large enough this could be computationally prohibitive.

For some predictors it is possible to write down the estimated average squared prediction error in closed form. For example, with linear models it is possible to get the value from a single model fit. For a linear model fitted by least-squares, it can be shown that

$$APSE(\mathcal{P}_0, \widetilde{\mu}) = \frac{1}{N} \sum_{i=1}^{N} \left(\frac{y_i - \widehat{\mu}(x_i)}{1 - h_{ii}} \right)^2$$

where $\widehat{\mu}(x_i)$ is the least-squares predictor and h_{ii} is the ith diagonal element of $\mathbf{H} = \mathbf{X}(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T$.

Aside: The elements of the sum, $\frac{y_i - \widehat{\mu}(x_i)}{1-h_{ii}}$ are often called the **prediction residuals** and their sum of squares the **prediction error sum of squares** or **PRESS**.

Note that the natural and basic splines we used as smoothers were in fact determined by least-squares, e.g. recall that $\widehat{\boldsymbol{\beta}} = (\mathbf{B}^T\mathbf{B})^{-1}\mathbf{B}^T\mathbf{y}$ for basic splines. Just as with any other linear model then, we therefore have

$$APSE(\mathcal{P}_0, \widetilde{\mu}) = \frac{1}{N} \sum_{i=1}^{N} \left(\frac{y_i - \widehat{\mu}(x_i)}{1 - s_{ii}} \right)^2$$

where $\widehat{\mu}(x_i)$ is now the spline predictor and s_{ii} is the ith diagonal element of its smoothing matrix $\mathbf{S}$. The result also holds for smoothing splines.

Aside: In cases where it might be difficult to compute the individual elements s_{ii} but relatively easy to find their sum $\sum_i s_{ii} = trace(\mathbf{S})$, replacing each s_{ii} by the average value $tr(\mathbf{S})/N$ can provide a reasonable

approximation as well. Motivated by adapting PRESS to choose the parameter λ of **ridge regression** (and hence also of of smoothing splines), this version of the leave-one-out cross-validation $APSE$, namely

$$APSE(\mathcal{P}_0, \widetilde{\mu}) = \frac{1}{N}\sum_{i=1}^{N}\left(\frac{y_i - \widehat{\mu}(x_i)}{1 - tr(\mathbf{S})/N}\right)^2$$

appears in the literature as **generalized cross validation**.

3.2.4 k-fold cross-validation

The larger k is the greater is the calculational effort – more data per predictor and more subset pairs to consider. So, we might like to have k be small.

Unfortunately, the smallest value of $k = 2$ seems much too small. In this case, the predictor functions will be estimated on only half of the observations we have in hand. Once we settle on a selected predictor function, we will estimate it based on all N cases in the sample, not just the $N/2$ on which our selection was made. Any estimate of error based on the $N/2$ seems likely to be biased to over-estimate the error of the same predictor estimated on N cases. Moreover, the variance of the estimated prediction error will be large since with $k = 2$ we are basing it on only one subset pair – no advantage is taken of averaging.

In selecting a value for k, we need to be guided as well by trade-offs between bias and variance.

Larger values of k make each subset $\mathcal{S}_j$ in a k-fold cross-validation larger. This in turn means that the estimated predictor on each $\mathcal{S}_j$ is closer to that on the whole sample $\mathcal{S}$ (which is the predictor that we will ultimately use). Hence, there should be less bias in the performance of these predictors as they represent the predictor on the whole sample.

Large k also means that we average over more subsets and so have lower variance than we would have over the individual subsets. Unfortunately, the larger is k, the larger is $\mathcal{S}_j$ and hence the greater the correlation between the elements of the sum. This in turn will inflate the variance of the average – an average based on positively correlated terms will will have a larger variance than the same average based on the same terms but which are uncorrelated. Decreasing k will reduce this correlation between the estimated predictor functions on different subets $\mathcal{S}_i$ and $\mathcal{S}_j$. The greater the intersection between $\mathcal{S}_i$ $\mathcal{S}_j$ the more correlated will be the estimated predictors.

Without knowing all of $\mathcal{P}$, the true $\mu(x)$, and the class of predictor functions used to estimate $\mu(x)$, it is difficult to choose an optimal value of k. **Experience related in the literature** seems to suggest that $k = 5$ or $k = 10$ often work well in balancing bias and variability. Certainly, these choices greatly reduce the computational burden over the $k = N$ choice of leave one out cross-validation.

3.3 Examples: revisited, again

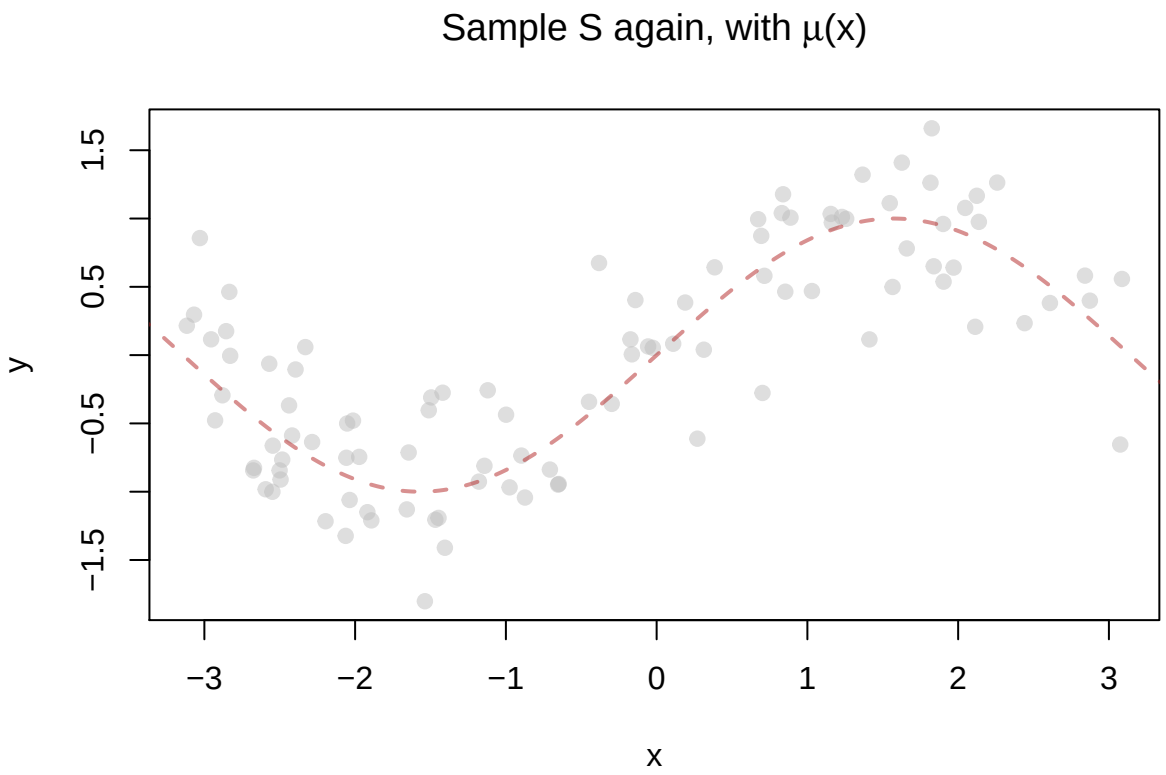
To see how to use k-fold cross validation in practice, and to compare choices of k, let us once more return to our artificial examples. As before, these have the enormous advantage in that we can calculate (or at least estimate exceptionally well) the true $APSE$ for these examples.

3.3.1 Example 1: revisited, again

We will first generate the same data we had before, stored now as `Sample_eg1` and plotted as below.

```
# plot the data in the set S
x <- Sample_eg1$x
y <- Sample_eg1$y
plot(x, y, pch=19, col=adjustcolor("grey", 0.5), xlab="x", ylab="y",
     main=expression(paste("Sample S again, with ", mu,"(x)")))
```

```
xrange <- extendrange(x)
funx <-seq(min(xrange), max(xrange), length.out=300)
lines(funx, mu(funx),
      col=adjustcolor("firebrick", 0.5), lty=2, lwd=2)
```



Based on this data only, we can now use k-fold cross-validation with various values of k to choose between the various predictors.

With the code already written, these are straight forward calculations in R. We first need to get the appropriate subsets

```
# For leave one out cross-validation
samples_loocv <-  getKfoldSamples(Sample_eg1$x, Sample_eg1$y, k=length(Sample_eg1$y))
# 10 fold cross-validation
samples_10fold <-  getKfoldSamples(Sample_eg1$x, Sample_eg1$y, k=10)
# 5 fold cross-validation
samples_5fold <-  getKfoldSamples(Sample_eg1$x, Sample_eg1$y, k=5)
```

To get the estimated average predicted squared errors for each, we simply call the appropriate functions on these samples. Here we will expand the complexity vector to include all values between 5 and 10 degrees of freedom and go no higher than 20.

```
# the degrees of freedom associated with each
complexity <- c(2, 5, 6, 7, 8, 9, 10, 20)

# leave one out
Ssamples <- samples_loocv$Ssamples
Tsamples <- samples_loocv$Tsamples
apsehat_loocv <-  sapply(complexity,
                         FUN = function(df){
                           apse(Ssamples, Tsamples, df=df)
```

```
                        }
)


# 10 fold cross-validation
Ssamples <- samples_10fold$Ssamples
Tsamples <- samples_10fold$Tsamples
apsehat_10fold <-  sapply(complexity,
                          FUN = function(df){
                            apse(Ssamples, Tsamples, df=df)
                          }
)

# 5 fold cross-validation
Ssamples <- samples_5fold$Ssamples
Tsamples <- samples_5fold$Tsamples
apsehat_5fold <-  sapply(complexity,
                         FUN = function(df){
                           apse(Ssamples, Tsamples, df=df)
                         }
)
```

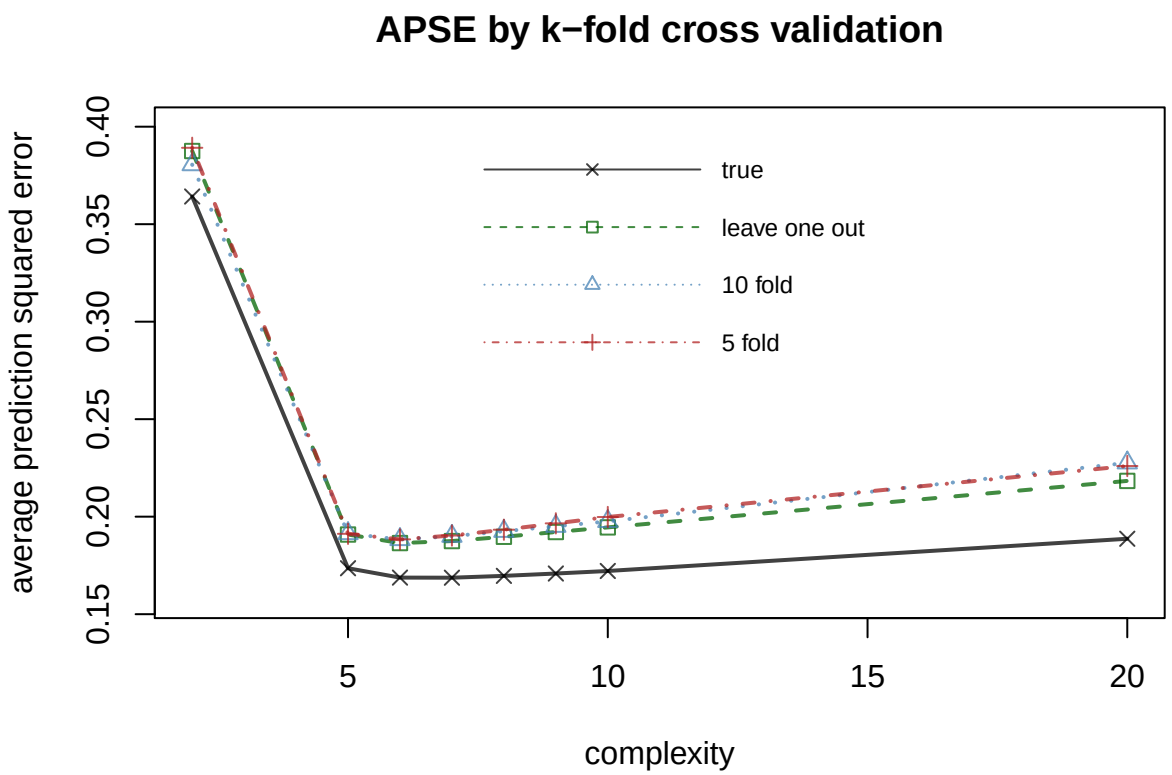
The "true" values can be determined as before by using the generative model to produce many new samples:

```
# The population based samples
Ssamples <- Ssamples_eg1_true
Tsamples <- Tsamples_eg1_true
apse_true <-  sapply(complexity,
                     FUN = function(df){
                       apse(Ssamples, Tsamples, df=df)
                     }
)
```

We plot these as follows:

```
ylim <- extendrange(c(apse_true, apsehat_loocv,
                      apsehat_10fold, apsehat_5fold))
# Plot the results
plot(complexity, apse_true, ylim=ylim,
     main="APSE by k-fold cross validation",
     ylab="average prediction squared error",
     pch=4, col= adjustcolor("black", 0.75))
lines(complexity, apse_true,
      col=adjustcolor("black", 0.75), lwd=2)

lines(complexity, apsehat_loocv,
      col=adjustcolor("darkgreen", 0.75), lwd=2, lty=2)
points(complexity, apsehat_loocv,
       col=adjustcolor("darkgreen", 0.75), pch=0)

lines(complexity, apsehat_10fold,
      col=adjustcolor("steelblue", 0.75), lwd=2, lty=3)
points(complexity, apsehat_10fold,
       col=adjustcolor("steelblue", 0.75), pch=2)
```

```
lines(complexity, apsehat_5fold,
      col=adjustcolor("firebrick", 0.75), lwd=2, lty=4)
points(complexity, apsehat_5fold,
       col=adjustcolor("firebrick", 0.75), pch=3)

legend(median(complexity), max(ylim),
       legend=c("true", "leave one out", "10 fold", "5 fold"),
       lty =1:4, pch=c(4,0,2,3),
       col=adjustcolor(c("black", "darkgreen", "steelblue","firebrick"), 0.75),
       cex=0.75, y.intersp=2, bty="n", seg.len=10)
```



As can be seen from the plot, for this example all of the cross-validated estimates overestimate the average prediction squared error. They are in near agreement with one another though, especially for 10 degrees of freedom or less.

If the only interest is in determining where the minimum of the curve is located, all methods are in close agreement. The actual values are

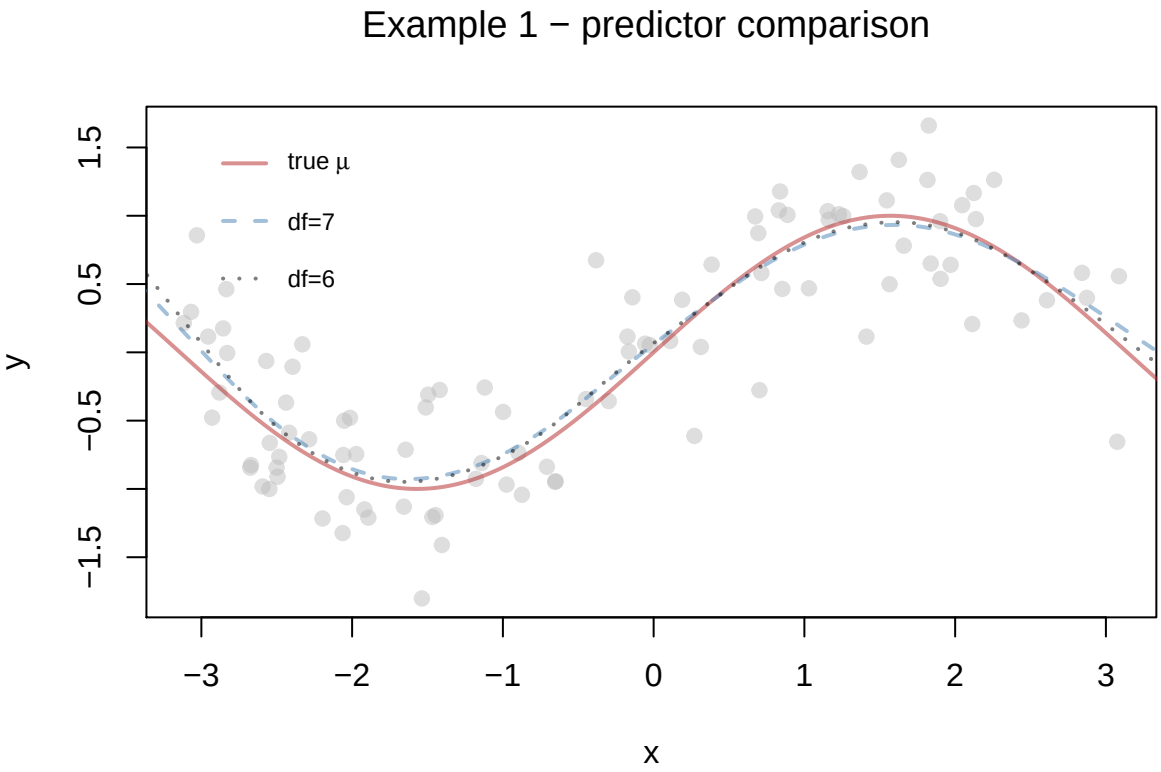
```
# Print out the results
t(rbind(complexity, loocv=round(apsehat_loocv,5),
        ten_fold=round(apsehat_10fold,5),
        five_fold=round(apsehat_5fold,5),
        true=round(apse_true, 5)))
```

```
##      complexity   loocv ten_fold five_fold    true
## [1,]          2 0.38753  0.38054   0.38918 0.36419
## [2,]          5 0.19084  0.19134   0.19121 0.17356
## [3,]          6 0.18647  0.18828   0.18837 0.16875
## [4,]          7 0.18749  0.18998   0.19034 0.16871
## [5,]          8 0.18970  0.19251   0.19332 0.16965
## [6,]          9 0.19213  0.19508   0.19657 0.17085
```

```
## [7,]         10 0.19451  0.19757   0.19983 0.17215
## [8,]         20 0.21835  0.22762   0.22597 0.18869
```

The true average prediction squared error suggests 7 degrees of freedom, while all three estimates suggest 6.

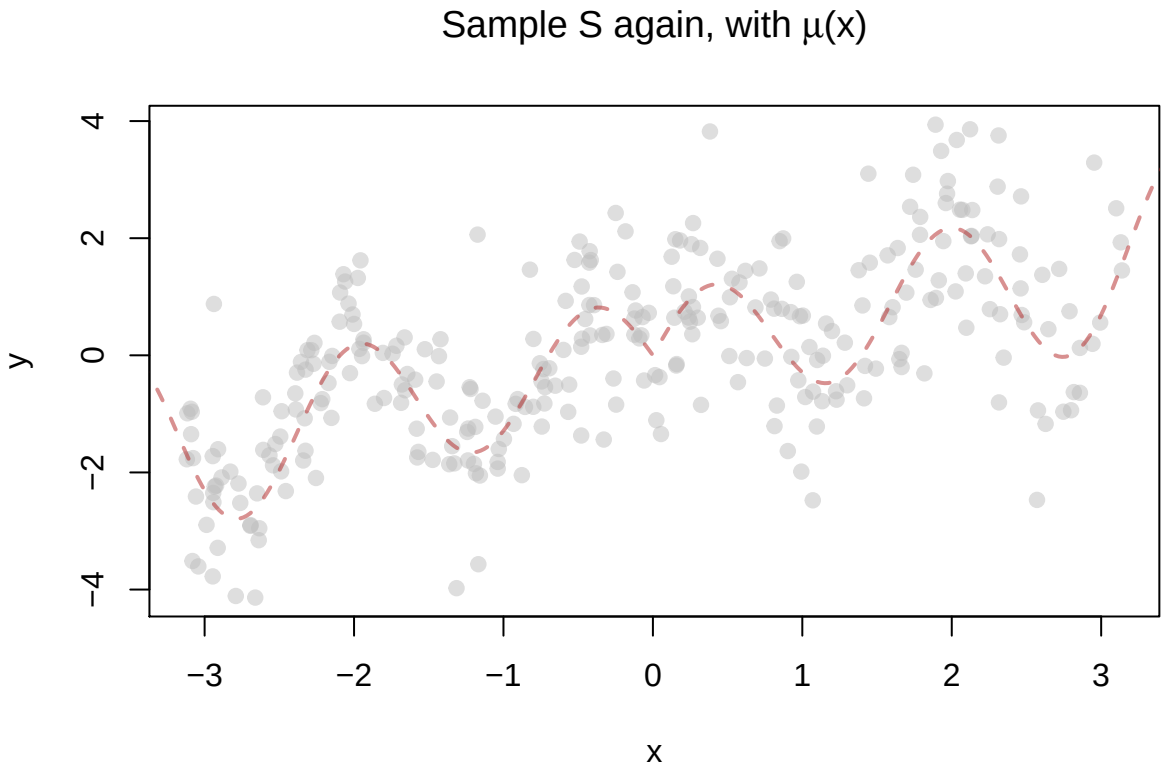
We can plot these two choices of predictor estimates on the data:



There does not seem a lot separating these estimated predictors.

3.3.2 Example 2: revisited, again

Again we generate the Example 2 data as before and now store the values as `Sample_eg2`. The data are plotted as below.



The k-fold cross-validation with various values of k are determined from samples.

```
# For leave one out cross-validation
samples_loocv <-  getKfoldSamples(Sample_eg2$x, Sample_eg2$y, k=length(Sample_eg2$y))
# 10 fold cross-validation
samples_10fold <-  getKfoldSamples(Sample_eg2$x, Sample_eg2$y, k=10)
# 5 fold cross-validation
samples_5fold <-  getKfoldSamples(Sample_eg2$x, Sample_eg2$y, k=5)
```

We get the estimated average predicted squared errors for eachon a complexity vector that now includes all values between 10 and 20 degrees of freedom but none higher than 30.

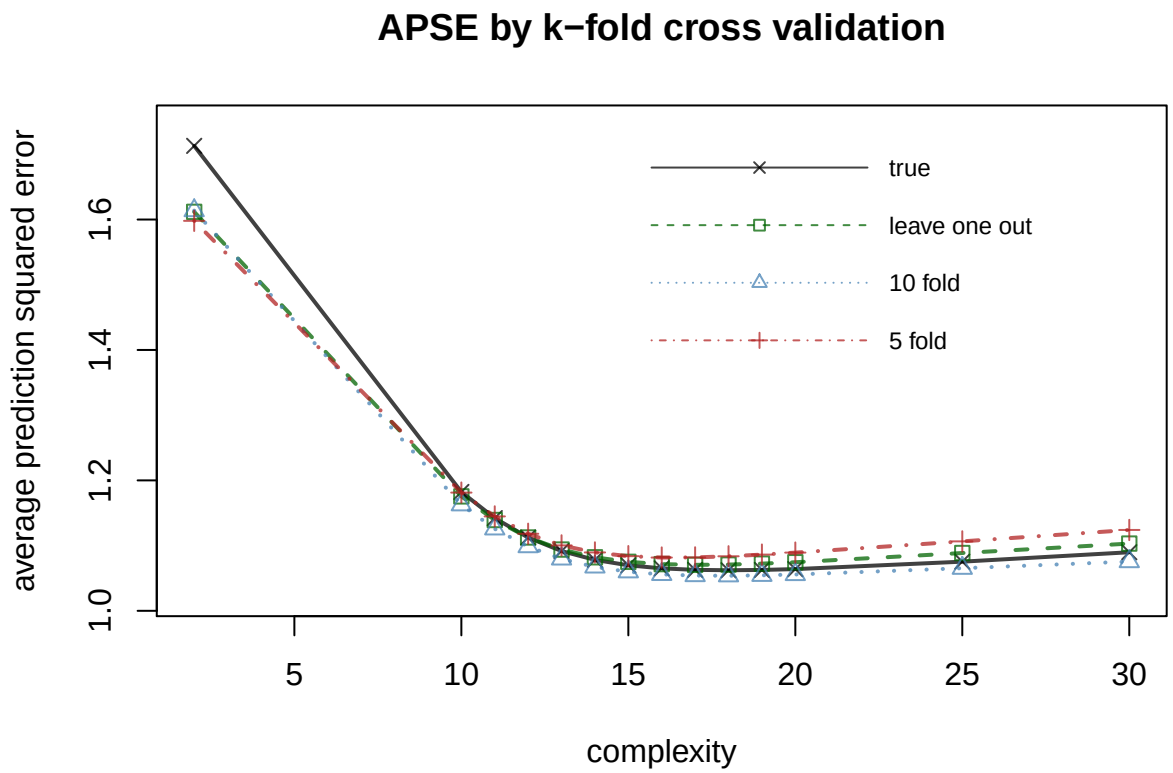
```
# the degrees of freedom associated with each
complexity <- c(2, 10:20, 25, 30)

# leave one out
Ssamples <- samples_loocv$Ssamples
Tsamples <- samples_loocv$Tsamples
apsehat_loocv <-  sapply(complexity,
                         FUN = function(df){
                           apse(Ssamples, Tsamples, df=df)
                         }
)


# 10 fold cross-validation
Ssamples <- samples_10fold$Ssamples
Tsamples <- samples_10fold$Tsamples
apsehat_10fold <-  sapply(complexity,
                          FUN = function(df){
                            apse(Ssamples, Tsamples, df=df)
```

```
                          }
)

# 5 fold cross-validation
Ssamples <- samples_5fold$Ssamples
Tsamples <- samples_5fold$Tsamples
apsehat_5fold <-  sapply(complexity,
                         FUN = function(df){
                           apse(Ssamples, Tsamples, df=df)
                         }
)
```

The "true" values are determined as before:

```
# The population based samples
Ssamples <- Ssamples_eg2_true
Tsamples <- Tsamples_eg2_true
apse_true <-  sapply(complexity,
                     FUN = function(df){
                       apse(Ssamples, Tsamples, df=df)
                     }
)
```

and plotted as below:



There is much agreement between the various cross-validated estimates and they are also in near agreement with true average prediction squared error, especially near 10 degrees of freedom.

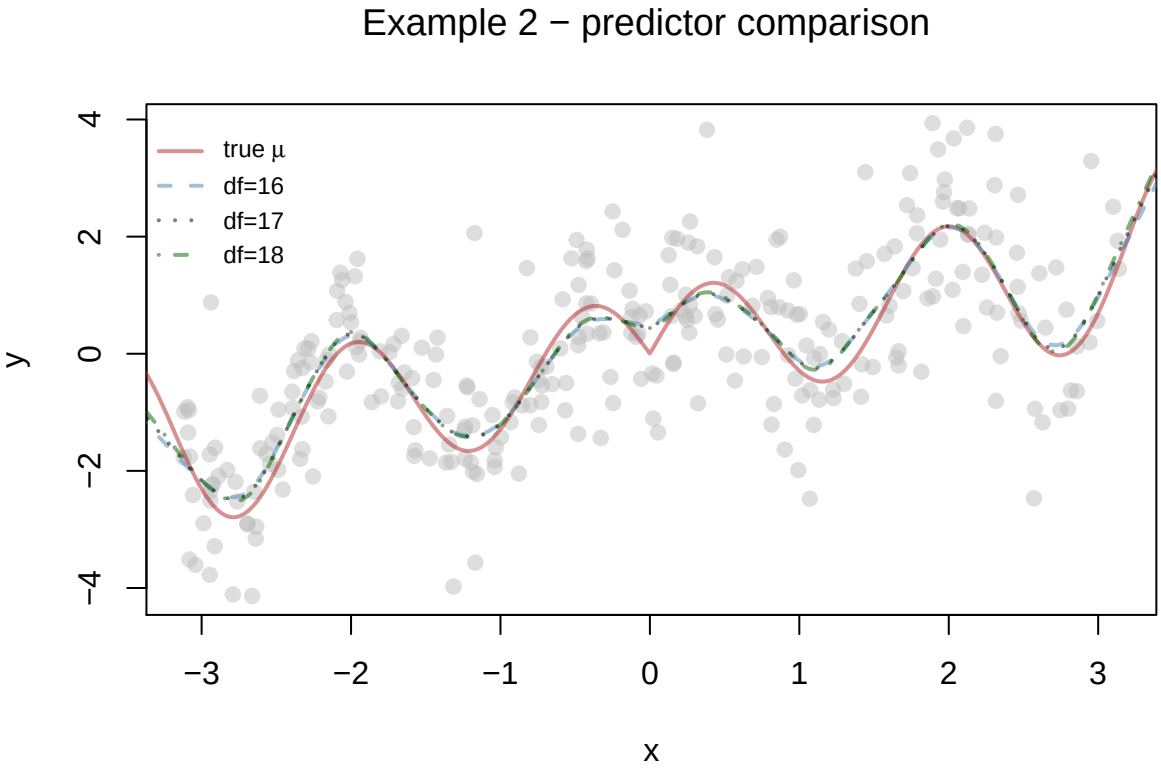
The actual values are

```
##       complexity   loocv ten_fold five_fold    true
##  [1,]          2 1.61178  1.61368   1.59799 1.71296
```

```
##  [2,]         10 1.17545  1.16259   1.18129 1.18225
##  [3,]         11 1.13924  1.12566   1.14475 1.14211
##  [4,]         12 1.11247  1.09841   1.11813 1.11237
##  [5,]         13 1.09393  1.07952   1.10030 1.09165
##  [6,]         14 1.08206  1.06729   1.08950 1.07812
##  [7,]         15 1.07510  1.05987   1.08380 1.06978
##  [8,]         16 1.07161  1.05582   1.08169 1.06508
##  [9,]         17 1.07048  1.05404   1.08190 1.06283
## [10,]         18 1.07093  1.05375   1.08360 1.06221
## [11,]         19 1.07237  1.05441   1.08619 1.06270
## [12,]         20 1.07448  1.05568   1.08926 1.06394
## [13,]         25 1.08862  1.06531   1.10647 1.07537
## [14,]         30 1.10307  1.07550   1.12400 1.08981
```

The true average prediction squared error suggests 18 degrees of freedom, as does the ten-fold. The five-fold suggests 16 and the leave one out 17.

We can plot these three choices of predictor estimates on the data:

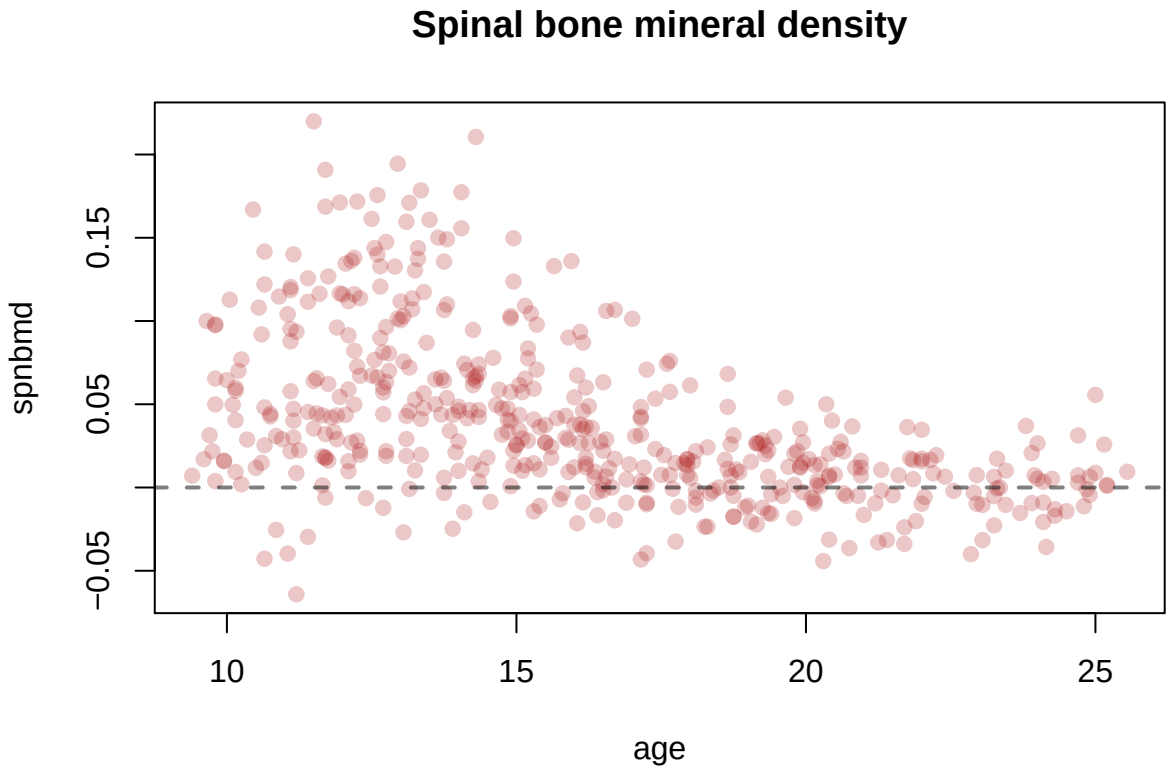


The three estimated predictors are visually indistinguishable from each other. All are different from the true $\mu(x)$, especially in the peaks and valleys.

3.4 Example: Spinal bone mineral density.

Here we will consider an example using real data

```
library(ElemStatLearn)
plot(bone$age, bone$spnbmd, col=adjustcolor("firebrick",0.25),
     pch=19, xlab="age", ylab="spnbmd",
     main = "Spinal bone mineral density")
abline(h=0, col=adjustcolor("black", 0.5), lty=2, lwd=2)
```




```
x <- bone$age
y <- bone$spnbmd
# Get the data by sex
female <- bone$gender == "female"
xfemale <- bone$age[female]
yfemale <- bone$spnbmd[female]
male <- bone$gender == "male"
xmale <- bone$age[male]
ymale <- bone$spnbmd[male]
```

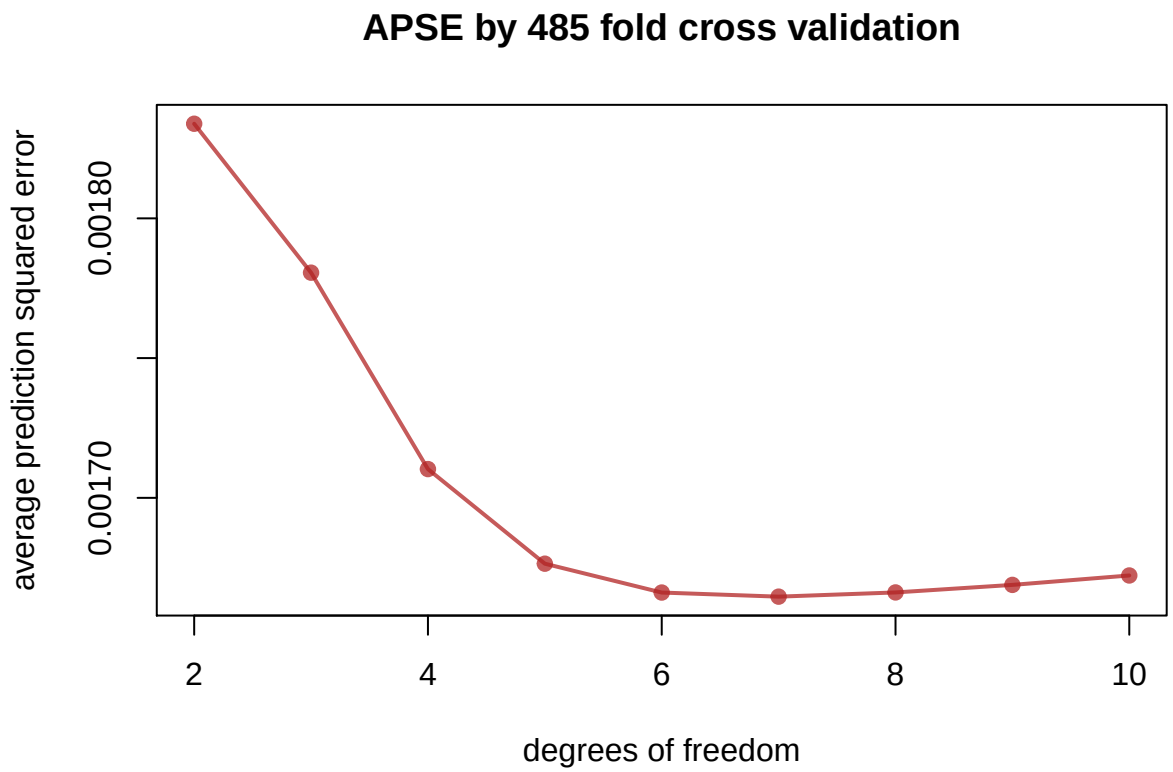
We will now use our functions to select a smoothing spline for the data all together as well as for each sex.

```
plot_apse <- function(x, y, df, k){
  samples <-  getKfoldSamples(x, y, k)
  apsehat <-  sapply(df,
                     FUN = function(df){
                       apse(samples$Ssamples,
                            samples$Tsamples,
                            df=df)
                     }
  )
  plot(df, apsehat,
       main=paste("APSE by", k, "fold cross validation"),
       ylab="average prediction squared error",
       xlab="degrees of freedom",
       pch=19, col= adjustcolor("firebrick", 0.75))
  lines(df, apsehat,
        col=adjustcolor("firebrick", 0.75), lwd=2)
  # return results
  apsehat
```

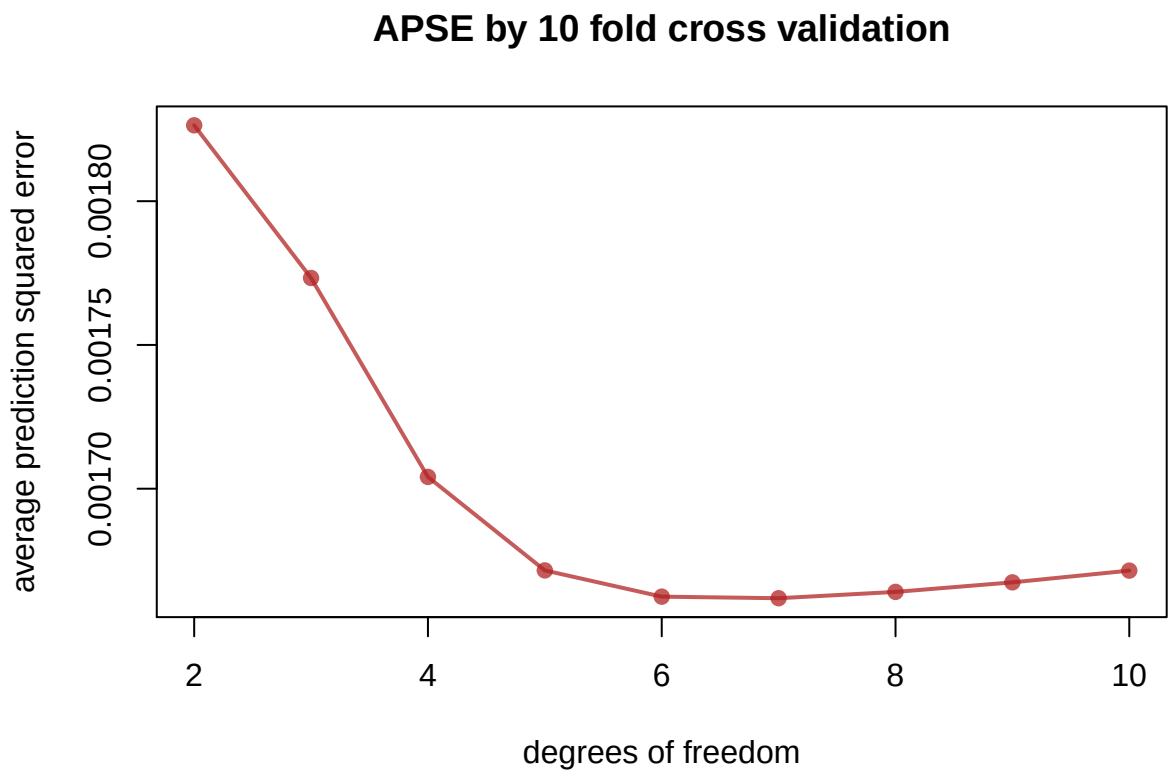
```
}
```

First all together

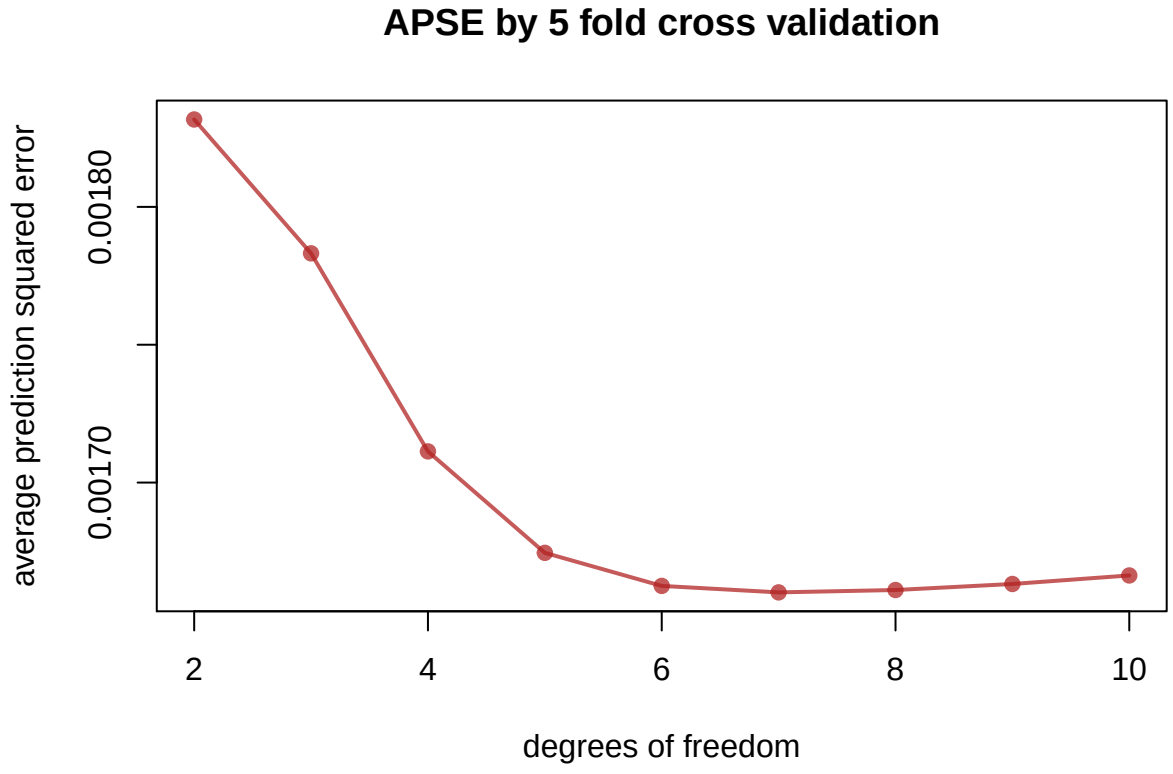
```
complexity <- 2:10

# leave one out cross-validation
apsehat_loocv <- plot_apse(x, y, df=complexity, k=length(y))
```



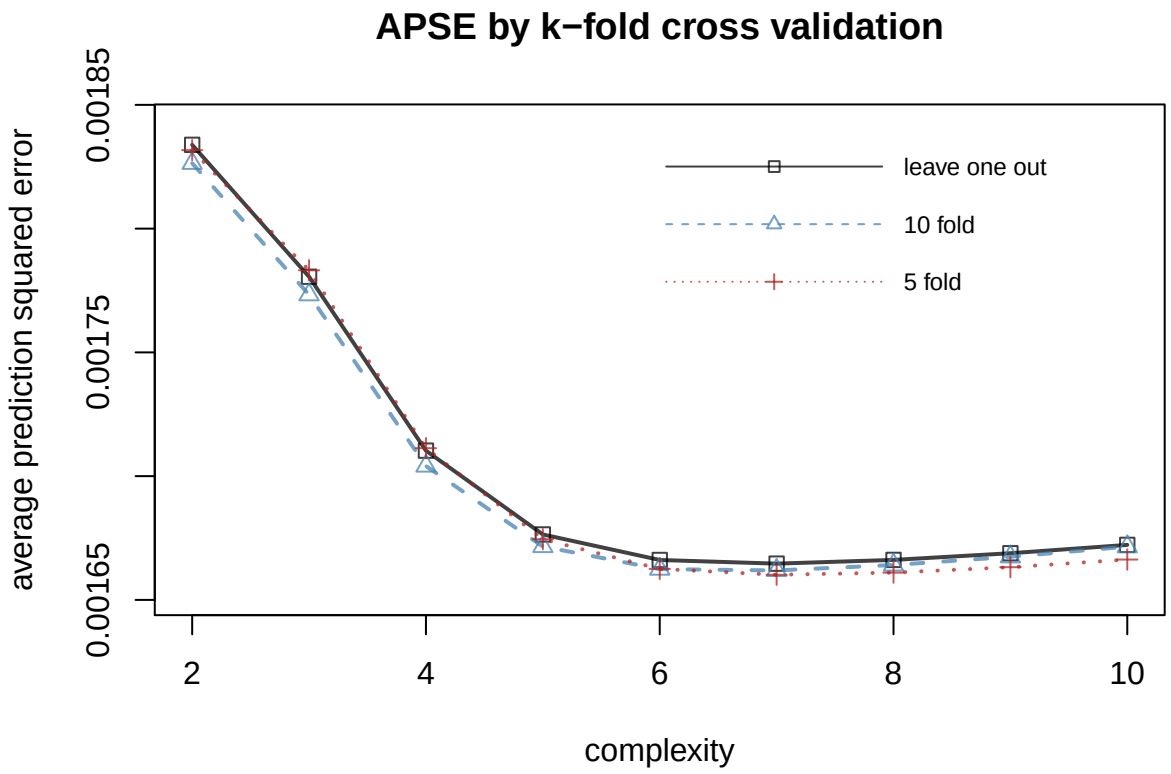

```
# 10 fold cross-
apsehat_10fold <- plot_apse(x, y, df=complexity, k=10)
```




```
# 5 fold cross-validation
apsehat_5fold <-  plot_apse(x, y, df=complexity, k=5)
```



All together on the same plot:

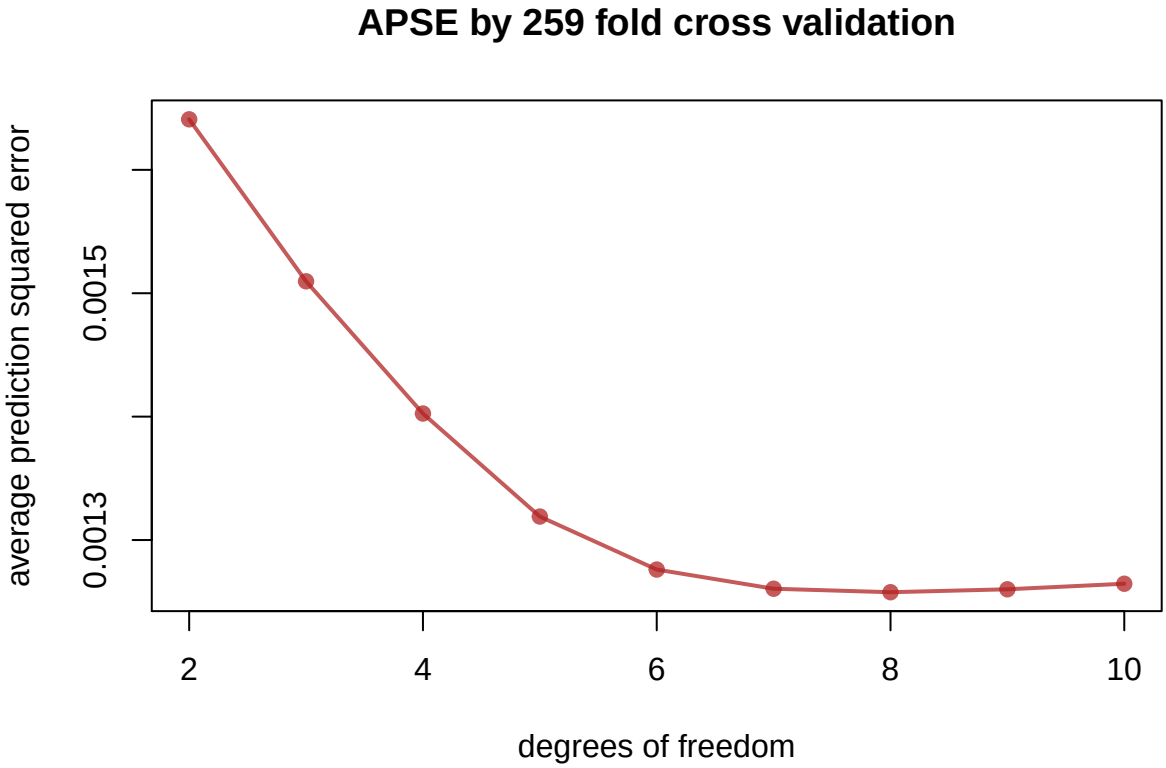


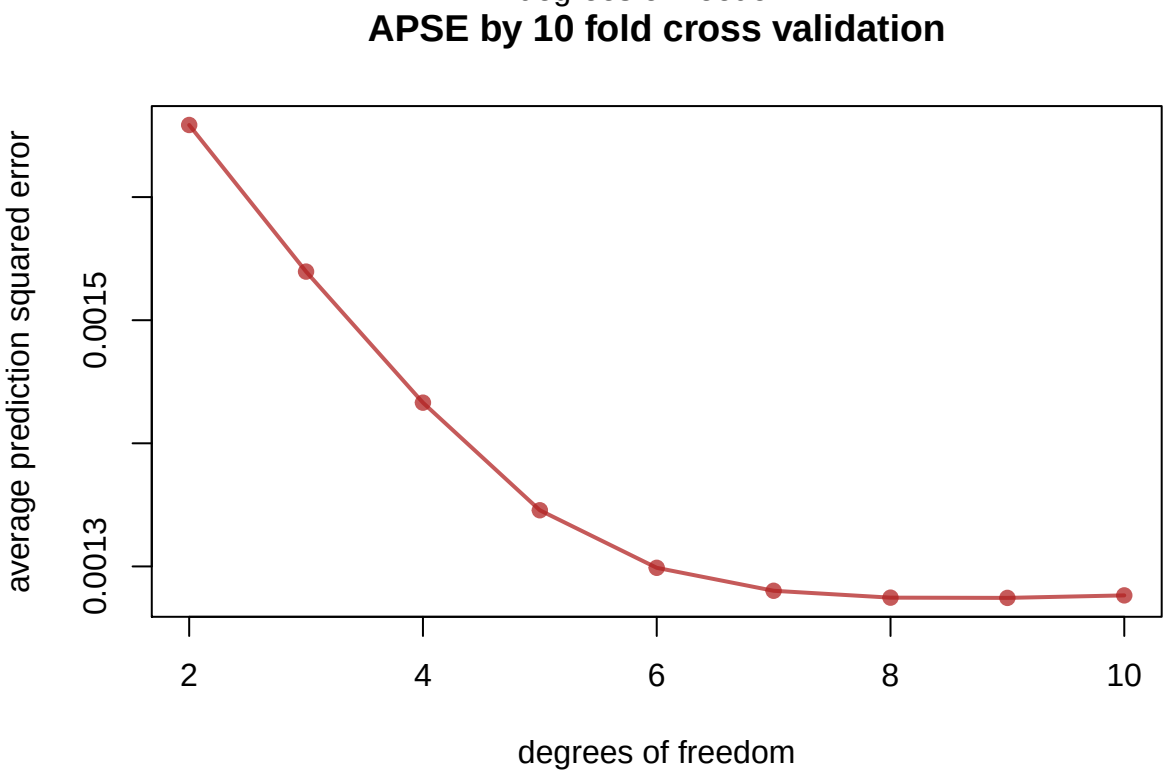
And print out the results

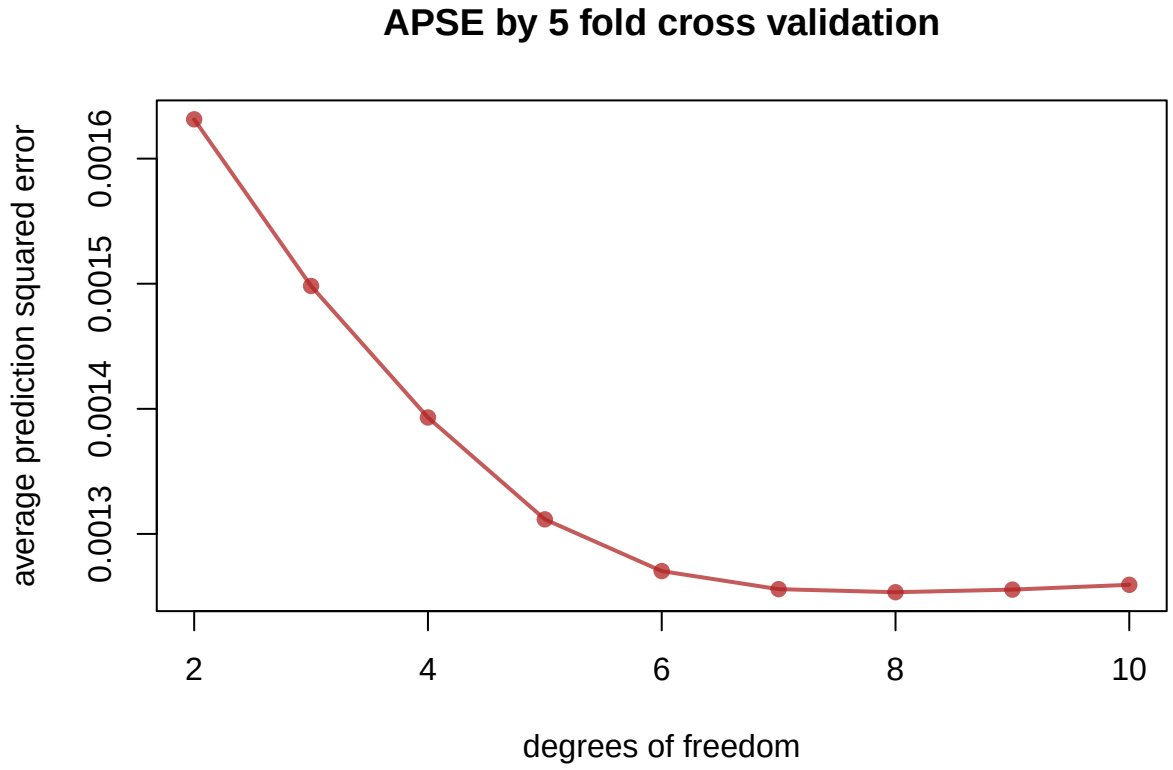
```
##       complexity     loocv  ten_fold five_fold
##  [1,]          2 0.0018338 0.0018264 0.0018317
##  [2,]          3 0.0017806 0.0017733 0.0017831
##  [3,]          4 0.0017103 0.0017041 0.0017113
##  [4,]          5 0.0016764 0.0016716 0.0016745
##  [5,]          6 0.0016661 0.0016624 0.0016625
##  [6,]          7 0.0016647 0.0016619 0.0016602
##  [7,]          8 0.0016662 0.0016641 0.0016610
##  [8,]          9 0.0016689 0.0016674 0.0016632
##  [9,]         10 0.0016722 0.0016715 0.0016663
```

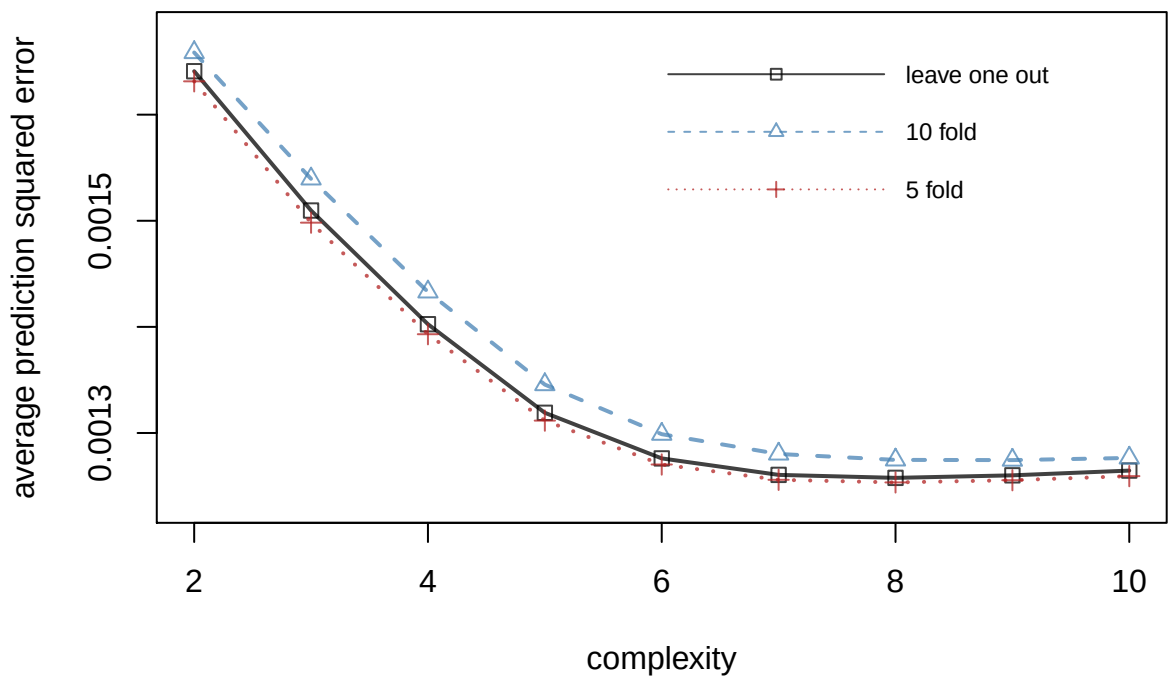
Leave one out, five fold, and ten fold measures all point to using 7 degrees of freedom for the smoothing spline.

Now the same for females:


APSE by 259 fold cross validation
average prediction squared error
0.0013
0.0015
2
4
6
8
10
degrees of freedom


APSE by 10 fold cross validation
average prediction squared error
0.0013
0.0015
2
4
6
8
10
degrees of freedom



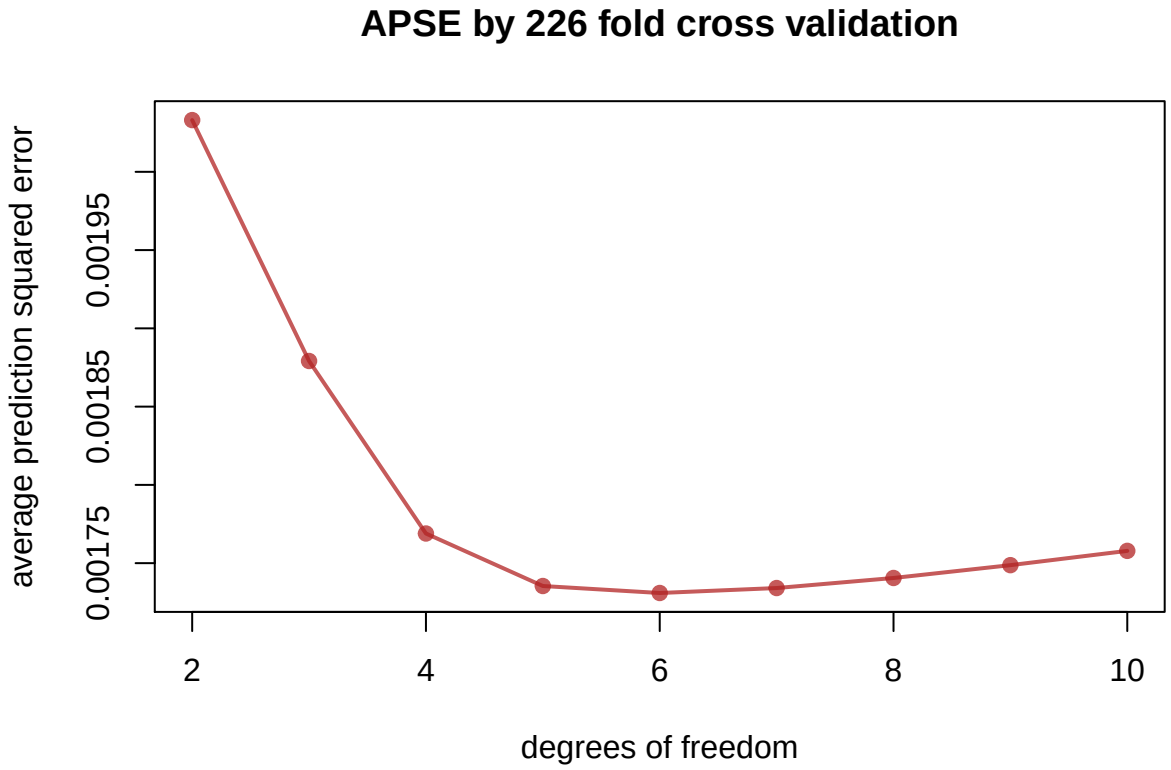


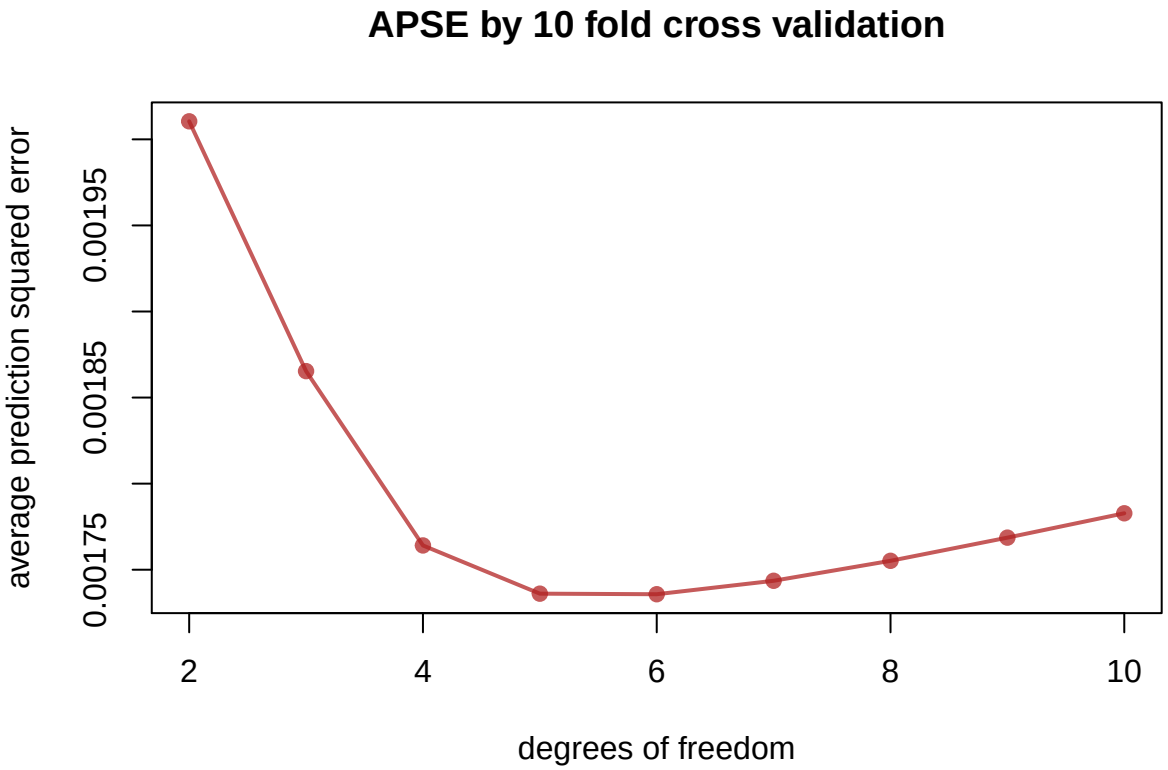

```
##       complexity    loocv ten_fold five_fold
##  [1,]          2 0.001641 0.001659  0.001631
##  [2,]          3 0.001510 0.001539  0.001498
##  [3,]          4 0.001403 0.001433  0.001393
##  [4,]          5 0.001319 0.001346  0.001312
##  [5,]          6 0.001276 0.001299  0.001270
```

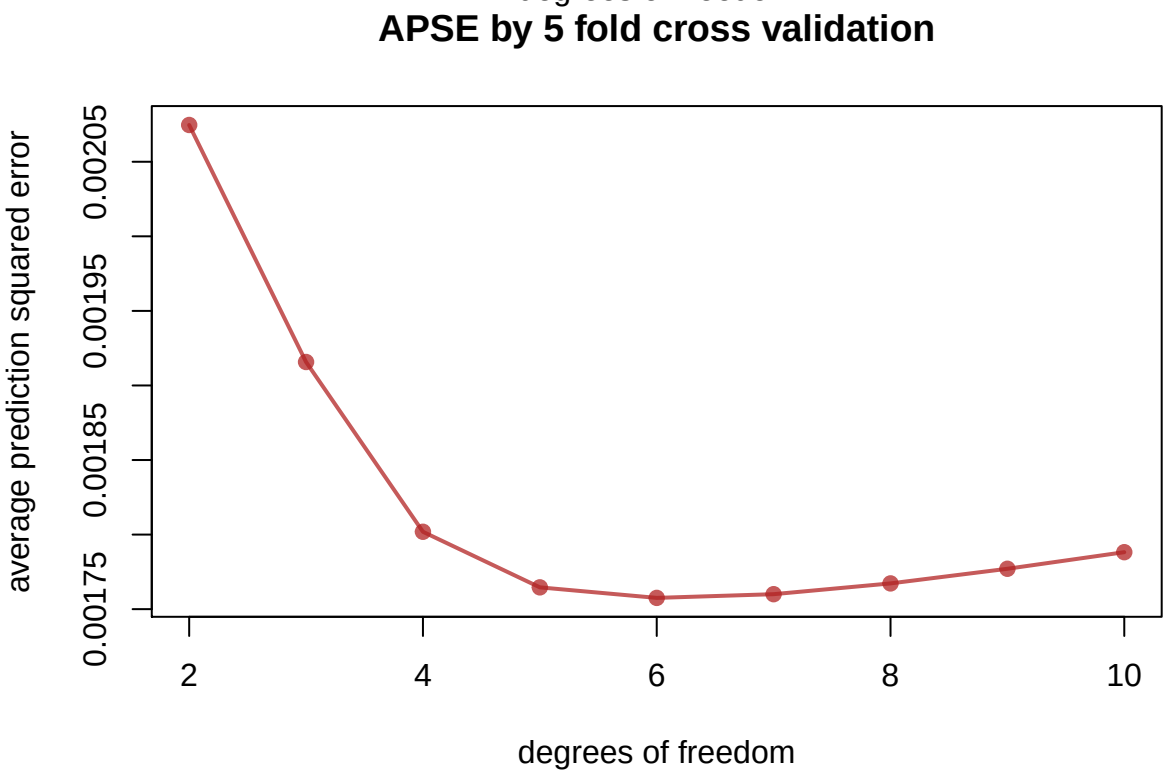
```
##  [6,]          7 0.001261 0.001280  0.001256
##  [7,]          8 0.001258 0.001275  0.001253
##  [8,]          9 0.001260 0.001274  0.001256
##  [9,]         10 0.001265 0.001276  0.001259
```

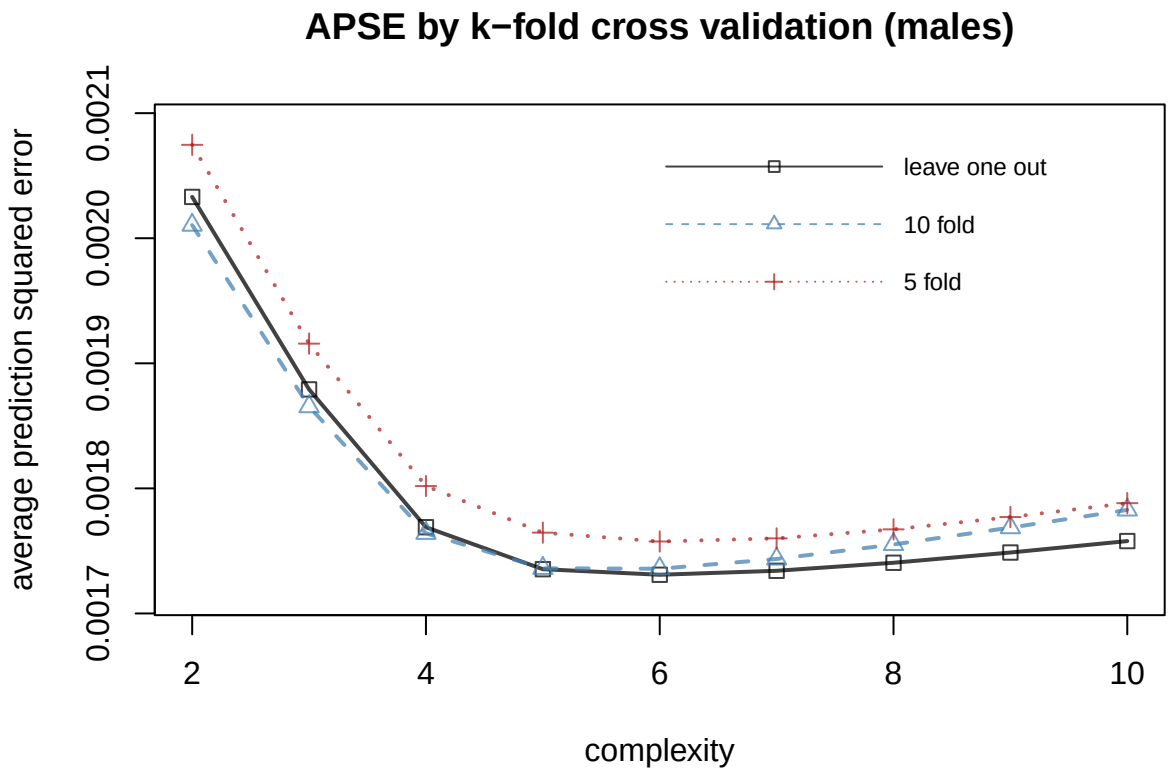
All measures suggest using 8 degrees of freedom for the females.

For the males:




APSE by 10 fold cross validation
average prediction squared error
0.00175
0.00185
0.00195
2
4
6
8
10
degrees of freedom


APSE by 5 fold cross validation
average prediction squared error
0.00175
0.00185
0.00195
0.00205
2
4
6
8
10
degrees of freedom




```
##       complexity    loocv ten_fold five_fold
##  [1,]          2 0.002033 0.002010  0.002075
##  [2,]          3 0.001879 0.001865  0.001916
##  [3,]          4 0.001769 0.001764  0.001802
##  [4,]          5 0.001735 0.001736  0.001765
##  [5,]          6 0.001731 0.001736  0.001758
##  [6,]          7 0.001734 0.001744  0.001760
##  [7,]          8 0.001741 0.001755  0.001767
##  [8,]          9 0.001749 0.001769  0.001777
##  [9,]         10 0.001758 0.001783  0.001788
```

All measures suggest 6 degrees of freedom for males.

The cross-validated selected smoothing splines are:

```
plot(x, y,  type="n",
     xlab="age", ylab="Change in density",
     main = "Spinal bone mineral density")
abline(h=0, col=adjustcolor("darkgrey", 0.85), lty=2, lwd=1)
points(xfemale, yfemale, col=adjustcolor("firebrick",0.25),
       pch=19)
points(xmale, ymale, col=adjustcolor("steelblue",0.25),
       pch=19)

xrange <- extendrange(x)
funx <-seq(min(xrange), max(xrange), length.out=300)
muhat1 <- getmuhat(list(x=x, y=y), df=7)
muhat2 <- getmuhat(list(x=xfemale, y=yfemale), df=8)
muhat3 <- getmuhat(list(x=xmale, y=ymale), df=6)
lines(funx, muhat1(funx),
      col=adjustcolor("black", 0.5), lty=4, lwd=2)
```

```
lines(funx, muhat2(funx),
      col=adjustcolor("firebrick", 0.5), lty=2, lwd=2)
lines(funx, muhat3(funx),
      col=adjustcolor("steelblue", 0.5), lty=3, lwd=2)
legend(median(xrange), max(y),
       legend=c("Female", "Male",
                "On both (df = 7)",
                "Female (df = 8)",
                "Male (df = 6)"),
       pch=c(19, 19, NA, NA, NA),
       lwd=c(NA, NA, 2, 2, 2),
       lty=c(NA, NA, 4, 3, 3),
       col=adjustcolor(c("firebrick", "steelblue",
                         "black","firebrick", "steelblue"), 0.5),
       bty="n", y.intersp=1.2, cex=0.75, seg.len=10)
```