Comparison over Multiple Samples

Contents

Some R functions we use in this section:

```
# This function calculates the average sum of squares
# for prediction error, i.e. mean((y-yhat)^2)
ave_y_mu_sq <- function(sample, predfun, na.rm = TRUE){
  mean((sample$y - predfun(sample$x))^2, na.rm = na.rm)
}


# Given two prediction functions predfun1 and predfun2,
#this function calculates the mean of the difference squared
ave_mu_mu_sq <- function(predfun1, predfun2, x, na.rm = TRUE){
  mean((predfun1(x) - predfun2(x))^2, na.rm = na.rm)
}

### The function getSampleComp will return a logical vector
### (that way the complement is also recorded in the sample)
getSampleComp <- function(pop, size, replace=FALSE) {
  N <- popSize(pop)
  samp <- rep(FALSE, N)
  samp[sample(1:N, size, replace = replace)] <- TRUE
  samp
```

```r
}

### This function will return a data frame containing
### only two variates, an x and a y
getXYSample <- function(xvarname, yvarname, samp, pop) {
  sampData <- pop[samp, c(xvarname, yvarname)]
  names(sampData) <- c("x", "y")
  sampData
}



popSize <- function(pop) {nrow(as.data.frame(pop))}
sampSize <- function(samp) {popSize(samp)}


age.unique = unique(Loblolly$age)


getmuFun <- function(pop, xvarname, yvarname){
  ## First remove NAs
  pop <- na.omit(pop[, c(xvarname, yvarname)])
  x <- pop[, xvarname]
  y <- pop[, yvarname]
  xks <- unique(x)
  muVals <- sapply(xks,
                   FUN = function(xk) {
                     mean(y[x==xk])
                   })
  ## Put the values in the order of xks
  ord <- order(xks)
  xks <- xks[ord]
  xkRange <-xks[c(1,length(xks))]
  minxk <- min(xkRange)
  maxxk <- max(xkRange)
  ## mu values
  muVals <- muVals[ord]
  muRange <- muVals[c(1, length(muVals))]
  muFun <- function(xVals){
    ## vector of predictions
    ## same size as xVals and NA in same locations
    predictions <- xVals
    ## Take care of NAs
    xValsLocs <- !is.na(xVals)
    ## Just predict non-NA xVals
    predictions[xValsLocs] <- sapply(xVals[xValsLocs],
                              FUN = function(xVal) {
                                 if (xVal < minxk) {
                                     result <- muRange[1]
                                 } else if (xVal > maxxk) {
                                     result <- muRange[2]
                                 } else if ( any(xVal == xks) ) {
                                   result <- muVals[xks == xVal]
```

```
                                  } else {
                                      xlower <- max(c(minxk, xks[xks < xVal]))
                                      xhigher <- min(c(maxxk, xks[xks >= xVal]))
                                      mulower <- muVals[xks == xlower]
                                      muhigher <- muVals[xks == xhigher]
                                      interpolateFn <- approxfun(x=c(xlower, xhigher),
                                                                 y=c(mulower, muhigher))
                                      result <- interpolateFn(xVal)
                                   }
                                      result
                                      }
    )
    ## Now return the predictions (including NAs)
    predictions
  }
  muFun
}
```

Note we are using the **getmuhat** function that does extrapolation using a constant.

```
getmuhat <- function(sampleXY, complexity = 1) {
  formula <- paste0("y ~ ",
                    if (complexity==0) {
                      "1"
                      } else
                      paste0("poly(x, ", complexity, ", raw = FALSE)")
                      #paste0("bs(x, ", complexity, ")")
  )

  fit <- lm(as.formula(formula), data = sampleXY)
  tx = sampleXY$x
  ty = fit$fitted.values

  range.X = range(tx)
  val.rY  = c( mean(ty[tx == range.X[1]]),
               mean(ty[tx == range.X[2]]) )

  ## From this we construct the predictor function
  muhat <- function(x){
    if ("x" %in% names(x)) {
      ## x is a dataframe containing the variate named
      ## by xvarname
      newdata <- x
    } else
      ## x is a vector of values that needs to be a data.frame
    { newdata <- data.frame(x = x) }
    ## The prediction
    ##
    val = predict(fit, newdata = newdata)
    val[newdata$x < range.X[1]] = val.rY[1]
    val[newdata$x > range.X[2]] = val.rY[2]
    val
  }
  ## muhat is the function that we need to calculate values
```

```
  ## at any x, so we return this function from getmuhat
  muhat
}
```

5.3 Comparison Over Multiple Samples

- The inaccuracy of a predictor is measured by

$$APSE(\mathcal{P}, \widehat{\mu}_{\mathcal{S}}) = Ave_{u \in \mathcal{P}} \left(y_u - \widehat{\mu}_{\mathcal{S}}(\mathbf{x}_u) \right)^2.$$

  - where $\mathcal{S}$ is the sample we used to fit the function, and
  - $\mathcal{T} := \mathcal{P} - \mathcal{S}$ is the complement set of $\mathcal{S}$ such that the population $\mathcal{P} = \{\mathcal{S}, \mathcal{T}\}$

- The function $\widehat{\mu}_{\mathcal{S}}(\mathbf{x})$ is based on a single sample $\mathcal{S}$ and
  - its performance might be peculiar to the particular choice of sample.
  - the average prediction error might be very different from one sample to another sample (changing $\mathcal{S}$).
  - It is important then to choose a predictor function that performs well no matter which sample was used to construct the predictor.

- Suppose that we have many samples, say $N_{\mathcal{S}}$ samples $\mathcal{S}_j$ for $j = 1, \ldots, N_{\mathcal{S}}$.
  - For each sample, we can calculate $\widehat{\mu}_{\mathcal{S}_j}(\mathbf{x})$ and

$$APSE(\mathcal{P}, \widehat{\mu}_{\mathcal{S}_j})$$

- The average $APSE$ over all $N_{\mathcal{S}}$ samples should be a better measure of the quality of a predictor function.

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

- Note that in $APSE(\mathcal{P}, \widetilde{\mu})$, the **estimator** notation $\widetilde{\mu}$ is used to emphasize that the function is looking at the values of $\widehat{\mu}$ for all possible samples $\mathcal{S}_j$.

Notation

- It can be shown that the average predicted squared error, $APSE$, for the estimator $\widetilde{\mu}(\mathbf{x})$ is composed of three separate and interpretable pieces.

- But two bits of notation are needed before we discuss this decomposition.

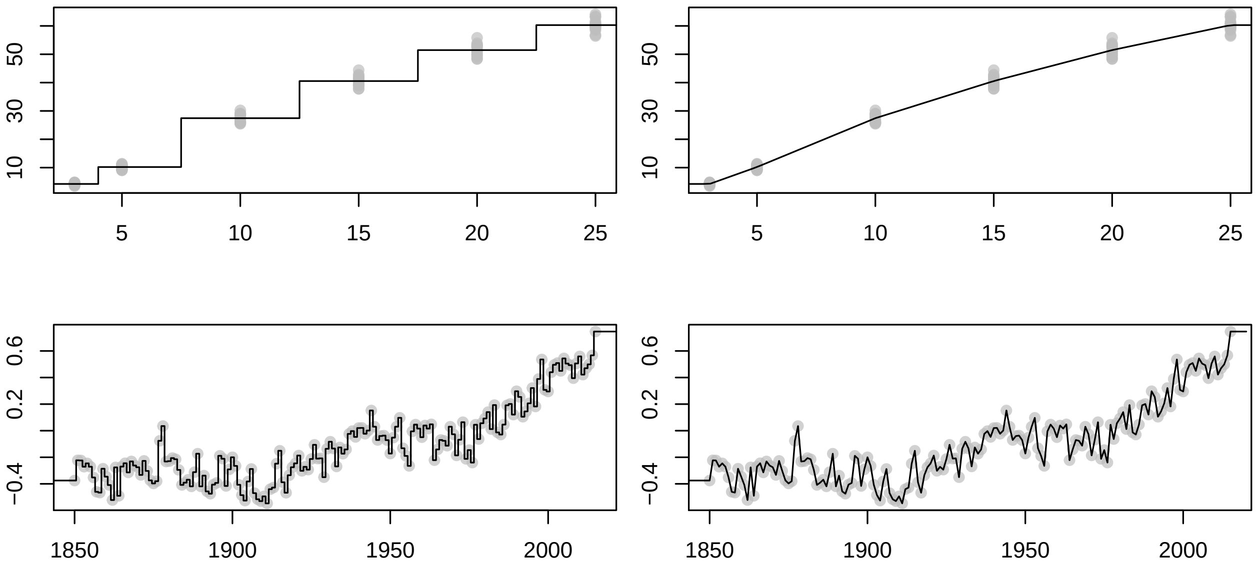
1. We observe that $\mu(\mathbf{x})$ denotes a **conditional** average of y **given** $\mathbf{x}$, defined as the average of all y values in $\mathcal{P}$ that share that value of $\mathbf{x}$. In other words, $\mu(\mathbf{x}) = E(Y|X = \mathbf{x})$.
   - Suppose that there are K different values of $\mathbf{x}$ 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
   $$\mathcal{A}_k = \{u \ : \ u \in \mathcal{P}, \ \mathbf{x}_u = \mathbf{x}_k\}$$
   and the unique values of $\mathbf{x}$ are $\mathbf{x}_1, \ldots, \mathbf{x}_K$
   - (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 average $\mu(\mathbf{x})$ can now be expressed for each distinct x_k as
   $$\mu(\mathbf{x}_k) = Ave_{i \in \mathcal{A}_k} \ y_i.$$

2. Let $\overline{\widehat{\mu}}(\mathbf{x})$ denote the average of the estimated predictor function over all samples
$$\overline{\widehat{\mu}}(\mathbf{x}) = \frac{1}{N_{\mathcal{S}}} \sum_{j=1}^{N_{\mathcal{S}}} \widehat{\mu}_{\mathcal{S}_j}(\mathbf{x}).$$

Example

- Growth of Loblolly pine trees and temperature data
  - This is **conditional** average of y **given** $\mathbf{x}$

$$\mu(\mathbf{x}_k) = Ave_{i \in \mathcal{A}_k} \ y_i. \quad \text{where} \quad \mathcal{A}_k = \{u \ : \ u \in \mathcal{P}, \ \mathbf{x}_u = \mathbf{x}_k\}$$



5.4 Decomposing the $APSE(\mathcal{P}, \widetilde{\mu})$

- As mentioned above, $APSE(\mathcal{P}, \widetilde{\mu})$ can be decomposed into three meaningful terms.
  - this is from the structure of the estimator $\widetilde{\mu}$ point of view.

$$
\begin{array}{llll}
APSE(\mathcal{P}, \widetilde{\mu}) & & & \\
= \frac{1}{N_{\mathcal{S}}} \sum_{j=1}^{N_{\mathcal{S}}} \; \frac{1}{N} \sum_{i \in \mathcal{P}} (y_i - \widehat{\mu}_{\mathcal{S}_j}(\mathbf{x}_i))^2 & & \\
= \frac{1}{N_{\mathcal{S}}} \sum_{j=1}^{N_{\mathcal{S}}} \; \frac{1}{N} \sum_{i \in \mathcal{P}} (y_i - \mu(\mathbf{x}_i))^2 & + \frac{1}{N_{\mathcal{S}}} \sum_{j=1}^{N_{\mathcal{S}}} \; \frac{1}{N} \sum_{i \in \mathcal{P}} (\widehat{\mu}_{\mathcal{S}_j}(\mathbf{x}_i) - \mu(\mathbf{x}_i))^2 & \\
= \frac{1}{N_{\mathcal{S}}} \sum_{j=1}^{N_{\mathcal{S}}} \; \frac{1}{N} \sum_{i \in \mathcal{P}} (y_i - \mu(\mathbf{x}_i))^2 & + \frac{1}{N_{\mathcal{S}}} \sum_{j=1}^{N_{\mathcal{S}}} \; \frac{1}{N} \sum_{i \in \mathcal{P}} (\widehat{\mu}_{\mathcal{S}_j}(\mathbf{x}_i) - \overline{\widehat{\mu}}(\mathbf{x}_i))^2 & + \frac{1}{N_{\mathcal{S}}} \sum_{j=1}^{N_{\mathcal{S}}} \; \frac{1}{N} \sum_{i \in \mathcal{P}} (\overline{\widehat{\mu}}(\mathbf{x}_i) - \mu(\mathbf{x}_i))^2 \\
= \frac{1}{N} \sum_{i \in \mathcal{P}} (y_i - \mu(\mathbf{x}_i))^2 & + \frac{1}{N_{\mathcal{S}}} \sum_{j=1}^{N_{\mathcal{S}}} \; \frac{1}{N} \sum_{i \in \mathcal{P}} (\widehat{\mu}_{\mathcal{S}_j}(\mathbf{x}_i) - \overline{\widehat{\mu}}(\mathbf{x}_i))^2 & + \frac{1}{N} \sum_{i \in \mathcal{P}} (\overline{\widehat{\mu}}(\mathbf{x}_i) - \mu(\mathbf{x}_i))^2 \\
= \sum_{k=1}^{K} \frac{n_k}{N} \sum_{i \in \mathcal{A}_k} \frac{1}{n_k} (y_i - \mu(\mathbf{x}_k))^2 & + \; \sum_{k=1}^{K} \frac{n_k}{N} \; \sum_{j=1}^{N_{\mathcal{S}}} \frac{1}{N_{\mathcal{S}}} (\widehat{\mu}_{\mathcal{S}_j}(\mathbf{x}_k) - \overline{\widehat{\mu}}(\mathbf{x}_k))^2 & + \sum_{k=1}^{K} \frac{n_k}{N} (\overline{\widehat{\mu}}(\mathbf{x}_k) - \mu(\mathbf{x}_k))^2 \\
= Ave_{\mathbf{x}} \left(Var(Y|\mathbf{x}) \right) & + Var(\widetilde{\mu}) & + Bias^2(\widetilde{\mu}).
\end{array}
$$

- Each term is an average over all $\mathbf{x}$ values **and** over all samples.
  - The first term is the average of the conditional variance of the response y,
  - the second term is the average of the variance of the estimator and
  - the last is term the squared bias.

- Note: The second last line accounts for units in the population with the same $\mathbf{x}$ values. There are K unique values of $\mathbf{x}$ in the population.

Another Decomposition of the APSE

- $APSE(\mathcal{P}, \widetilde{\mu})$ can be decomposed from the population's structure point of view as well.

$$
\begin{aligned}
& APSE(\mathcal{P}, \widetilde{\mu}) \\
&= \tfrac{1}{N_{\mathcal{S}}} \textstyle\sum_{j=1}^{N_{\mathcal{S}}} \ \tfrac{1}{N} \sum_{i \in \mathcal{P}} (y_i - \mu(\mathbf{x}_i))^2 \quad + \tfrac{1}{N_{\mathcal{S}}} \sum_{j=1}^{N_{\mathcal{S}}} \ \tfrac{1}{N} \sum_{i \in \mathcal{P}} (\widehat{\mu}_{\mathcal{S}_j}(\mathbf{x}_i) - \overline{\widehat{\mu}}(\mathbf{x}_i))^2 \quad + \tfrac{1}{N_{\mathcal{S}}} \sum_{j=1}^{N_{\mathcal{S}}} \ \tfrac{1}{N} \sum_{i \in \mathcal{P}} (\overline{\widehat{\mu}}(\mathbf{x}_i) - \mu \\
&= \tfrac{n}{N} \Big[\tfrac{1}{N_{\mathcal{S}}} \textstyle\sum_{j=1}^{N_{\mathcal{S}}} \ \tfrac{1}{n} \sum_{i \in \mathcal{S}} (y_i - \mu(\mathbf{x}_i))^2 \quad + \tfrac{1}{N_{\mathcal{S}}} \sum_{j=1}^{N_{\mathcal{S}}} \ \tfrac{1}{n} \sum_{i \in \mathcal{S}} (\widehat{\mu}_{\mathcal{S}_j}(\mathbf{x}_i) - \overline{\widehat{\mu}}(\mathbf{x}_i))^2 \quad + \tfrac{1}{N_{\mathcal{S}}} \sum_{j=1}^{N_{\mathcal{S}}} \ \tfrac{1}{n} \sum_{i \in \mathcal{S}} (\overline{\widehat{\mu}}(\mathbf{x}_i) - \mu(\\
&+ \left(1 - \tfrac{n}{N}\right) \Big[\tfrac{1}{N_{\mathcal{S}}} \textstyle\sum_{j=1}^{N_{\mathcal{S}}} \ \tfrac{1}{N-n} \sum_{i \notin \mathcal{S}} (y_i - \mu(\mathbf{x}_i))^2 \quad + \tfrac{1}{N_{\mathcal{S}}} \sum_{j=1}^{N_{\mathcal{S}}} \ \tfrac{1}{N-n} \sum_{i \notin \mathcal{S}} (\widehat{\mu}_{\mathcal{S}_j}(\mathbf{x}_i) - \overline{\widehat{\mu}}(\mathbf{x}_i))^2 \quad + \tfrac{1}{N_{\mathcal{S}}} \sum_{j=1}^{N_{\mathcal{S}}} \ \tfrac{1}{N-n} \sum_{i \notin \mathcal{S}} (\overline{\widehat{\mu}}(\mathbf{x}_i) -
\end{aligned}
$$

- We might write

$$
\begin{aligned}
APSE(\mathcal{P}, \widetilde{\mu}) = \left(\frac{n}{N}\right) & \left\{ \widehat{APSE}(\mathcal{P}, \widetilde{\mu}) \ \text{ based on the } \textbf{same} \text{ samples used by } \ \widehat{\mu} \right\} \\
+ \left(1 - \frac{n}{N}\right) & \left\{ \widehat{APSE}(\mathcal{P}, \widetilde{\mu}) \quad \text{based on samples } \textbf{not} \text{ used by } \ \widehat{\mu} \right\}
\end{aligned}
$$

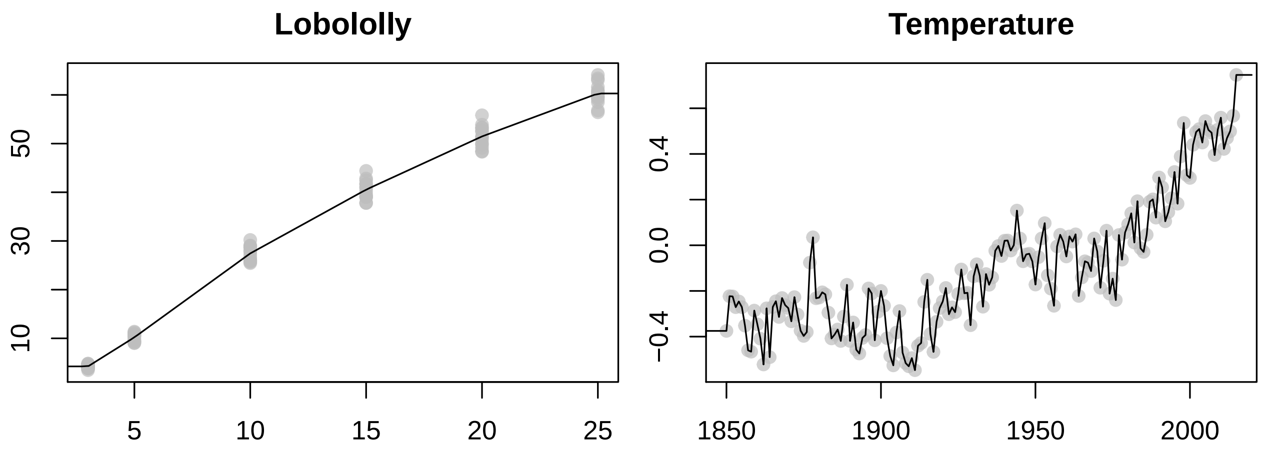
- If $n << N$, then the second term dominates.
  - Sometimes, even if $n \approx N$ we might also want to focus our evaluation only on the second term since this evaluation is based on values not used in the actual estimation process.

Example of $\mu(x)$

- Growth of Loblolly pine trees and temperature data
  - One difference between the two data-sets is that in the Loblolly data, we have multiple y measurements for any given $\mathbf{x}$.
  - This facilitates computation of **conditional** average of $Var(Y|\mathbf{x})$.

$$
\mu(\mathbf{x}_k) = Ave_{i \in \mathcal{A}_k} \ y_i. \quad \text{where} \quad \mathcal{A}_k = \{u \ : \ u \in \mathcal{P}, \ \mathbf{x}_u = \mathbf{x}_k\}
$$

```
par(mfrow=c(1,2), mar=2.5*c(1,1,1,0.1))

plot(Loblolly$age[order(Loblolly$age)], Loblolly$height[order(Loblolly$age)], xlab="Age",
     ylab="Height", col=adjustcolor("grey", 0.7),
     pch=19, main="Lobololly")
mu.age = getmuFun(pop=Loblolly, xvarname="age", yvarname="height")
curve(mu.age, 0, 30, add=TRUE, n=100)

plot(temperature$YEAR, temperature$ANNUAL, xlab="Year", ylab="Annual Temperature",
     col=adjustcolor("grey", 0.7), pch=19, main="Temperature")
mu.annual = getmuFun(pop=temperature, xvarname="YEAR", yvarname="ANNUAL")
curve(mu.annual, 1840, 2020, add=TRUE, 181)
```



- $Ave_x(Var(y|x)) = \frac{1}{N}\sum_{i\in\mathcal{P}}\left[y_i - \mu(x)\right]^2$ is

```
mat = matrix(0, nrow=2,ncol=1)
mat[,1] = c(mean( (Loblolly$height - mu.age(Loblolly$age))^2  ),
mean( (temperature$ANNUAL - mu.annual(temperature$YEAR))^2  ) )
rownames(mat) = c("Lobololly", "Temperature")
colnames(mat) = "Ave_x Var(y|x)"
round(mat,2)
```

```
##             Ave_x Var(y|x)
## Lobololly             2.64
## Temperature           0.00
```

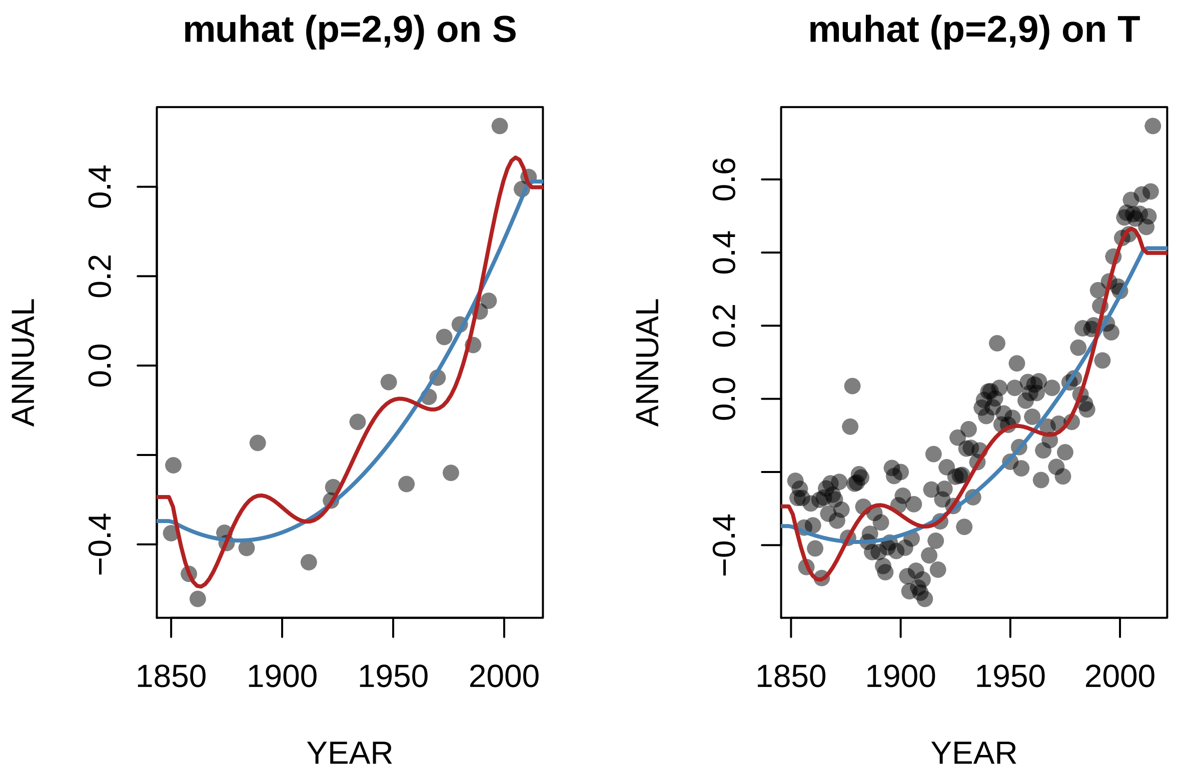
- What does $Var(Y|\mathbf{x}) = 0$ mean?

One sample ($\mathcal{S}$) and the complement set ($\mathcal{T}$)

- Consider the global temperature data.
- Generate one sample of size $n = 25$ and its complement set.

```
xnam <- "YEAR"
ynam <- "ANNUAL"
pop  <- temperature
n    <- 25

set.seed(341)  # for reproducibility
samp     <- getSampleComp(pop, n)
Ssamp <- getXYSample(xnam, ynam, samp, pop)
Tsamp <- getXYSample(xnam, ynam, !samp, pop)

muhat2 <- getmuhat(Ssamp, 2)
muhat9 <- getmuhat(Ssamp, 9)

xlim  <- extendrange(pop[, xnam])

par(mfrow=c(1,2))
```

```
plot(Ssamp,
     main=paste0("muhat (p=2,9) on S"),
     xlab = xnam, ylab = ynam,
     pch=19, col= adjustcolor("black", 0.5))

curve(muhat2, from = xlim[1], to = xlim[2],
      add = TRUE, col="steelblue", lwd=2)
curve(muhat9, from = xlim[1], to = xlim[2],
      add = TRUE, col="firebrick", lwd=2)

plot(Tsamp,
     main=paste0("muhat (p=2,9) on T"),
     xlab = xnam, ylab = ynam,
     pch=19, col= adjustcolor("black", 0.5))

curve(muhat2, from = xlim[1], to = xlim[2],
      add = TRUE, col="steelblue", lwd=2)
curve(muhat9, from = xlim[1], to = xlim[2],
      add = TRUE, col="firebrick", lwd=2)
```



Nine samples and complements

- Set some variables and generate $N_{\mathcal{S}} = 9$ samples of size $n = 25$

```r
xnam <- "YEAR"
ynam <- "ANNUAL"
pop  <- temperature
n    <- 25
N_S  <- 9

set.seed(1)  # for reproducibility
samples <- lapply(1:N_S, FUN= function(i){getSampleComp(pop, n)})
Ssam    <- lapply(samples, FUN= function(Si){getXYSample(xnam, ynam, Si, pop)})
Tsam    <- lapply(samples, FUN= function(Si){getXYSample(xnam, ynam, !Si, pop)})
```

- The function to construct $\overline{\widehat{\mu}}(x)$

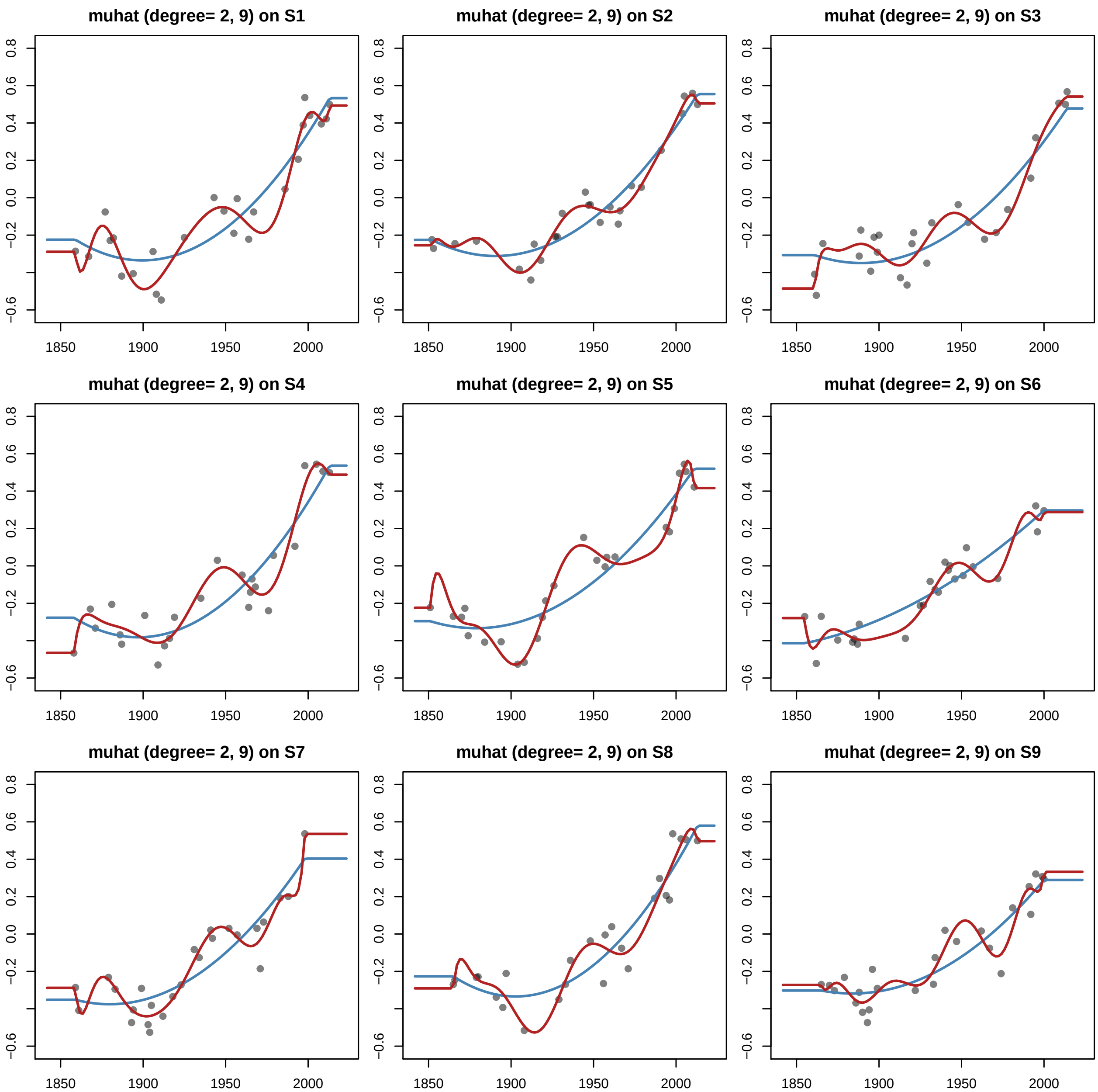
```r
getmubar <- function(muhats) {
  function(x) {
    Ans <- sapply(muhats, FUN=function(muhat){muhat(x)})
    apply(Ans, MARGIN=1, FUN=mean)
  }
}
```

- Fit polynomials of degree 2 and 9 on all samples and get the average function.

```r
muhats2 <- lapply(Ssam, getmuhat, complexity = 2)
mubar2  <- getmubar(muhats2)

muhats9 <- lapply(Ssam, getmuhat, complexity = 10)
mubar9  <- getmubar(muhats10)
```

The fitted polynomials on $\mathcal{S}$

```r
par(mfrow=c(3,3), mar=2.5*c(1,1,1,0.1))

xlim <- extendrange(temperature[, xnam])
ylim <- extendrange(temperature[, ynam])

for (i in 1:N_S) {
    plot(Ssam[[i]],
     main=paste("muhat (degree= 2, 9) on S", i, sep=""),
     xlab = xnam, ylab = ynam,
     pch=19, col= adjustcolor("black", 0.5), ylim=ylim, xlim=xlim)

   tempfn =  muhats2[[i]]
   curve(tempfn, from = xlim[1], to = xlim[2], add = TRUE, col="steelblue", lwd=2)
   tempfn =  muhats9[[i]]
   curve(tempfn, from = xlim[1], to = xlim[2], add = TRUE, col="firebrick", lwd=2)
}
```



- The residuals sum of squares from each $\mathcal{S}_i$

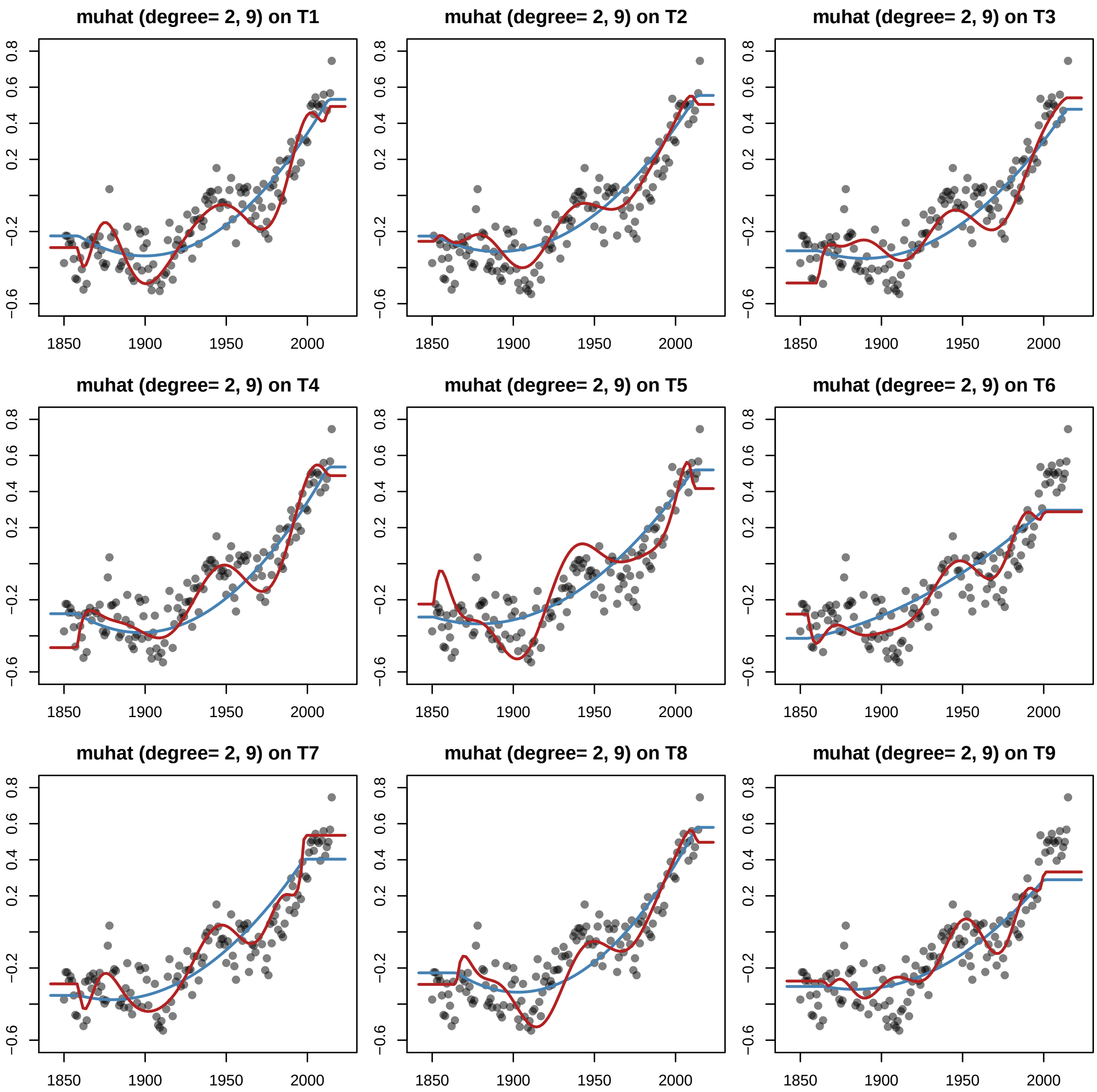
```
Smat = matrix(0, nrow=2,ncol= N_S+1 )

for (j in 1:N_S) {
    tempfn =  muhats2[[j]]
    Smat[1,j] =  mean( (Ssam[[j]]$y - tempfn(Ssam[[j]]$x))^2 )

    tempfn =  muhats9[[j]]
    Smat[2,j] = mean( (Ssam[[j]]$y - tempfn(Ssam[[j]]$x))^2 )
}

rownames(Smat) = c("Degree = 2", "Degree = 9")
colnames(Smat) = c(paste("S", 1:N_S, sep=""), "Avg.")
Smat[,N_S+1] = apply(Smat[,1:N_S],1,mean)
```

```r
round(Smat,3)
```

```
##                S1    S2    S3    S4    S5    S6    S7    S8    S9  Avg.
## Degree = 2 0.015 0.007 0.014 0.016 0.013 0.007 0.013 0.011 0.009 0.012
## Degree = 9 0.005 0.002 0.006 0.006 0.002 0.003 0.003 0.007 0.005 0.004
```

Fitted polynomials on $\mathcal{T}$

```r
par(mfrow=c(3,3), mar=2.5*c(1,1,1,0.1))

dset = c(2,4,10)
xlim <- extendrange(temperature[, xnam])
ylim <- extendrange(temperature[, ynam])

for (i in 1:N_S) {
    plot(Tsam[[i]],
     main=paste("muhat (degree= 2, 9) on T", i, sep=""),
     xlab = xnam, ylab = ynam,
     pch=19, col= adjustcolor("black", 0.5), ylim=ylim, xlim=xlim)

   tempfn =  muhats2[[i]]
   curve(tempfn, from = xlim[1], to = xlim[2], add = TRUE, col="steelblue", lwd=2)
   tempfn =  muhats9[[i]]
   curve(tempfn, from = xlim[1], to = xlim[2], add = TRUE, col="firebrick", lwd=2)
}
```



- The residuals sum of squares from each $\mathcal{T}_i$

```
Tmat = matrix(0, nrow=2,ncol= N_S+1 )

for (j in 1:N_S) {
    tempfn =  muhats2[[j]]
    Tmat[1,j] =  mean( (Tsam[[j]]$y - tempfn(Tsam[[j]]$x))^2 )

    tempfn =  muhats9[[j]]
    Tmat[2,j] = mean( (Tsam[[j]]$y - tempfn(Tsam[[j]]$x))^2 )
}

rownames(Tmat) = c("Degree = 2", "Degree = 9")
colnames(Tmat) = c(paste("T", 1:N_S, sep=""), "Avg.")
Tmat[,N_S+1] = apply(Tmat[,1:N_S],1,mean)
```

```r
round(Tmat,3)
```

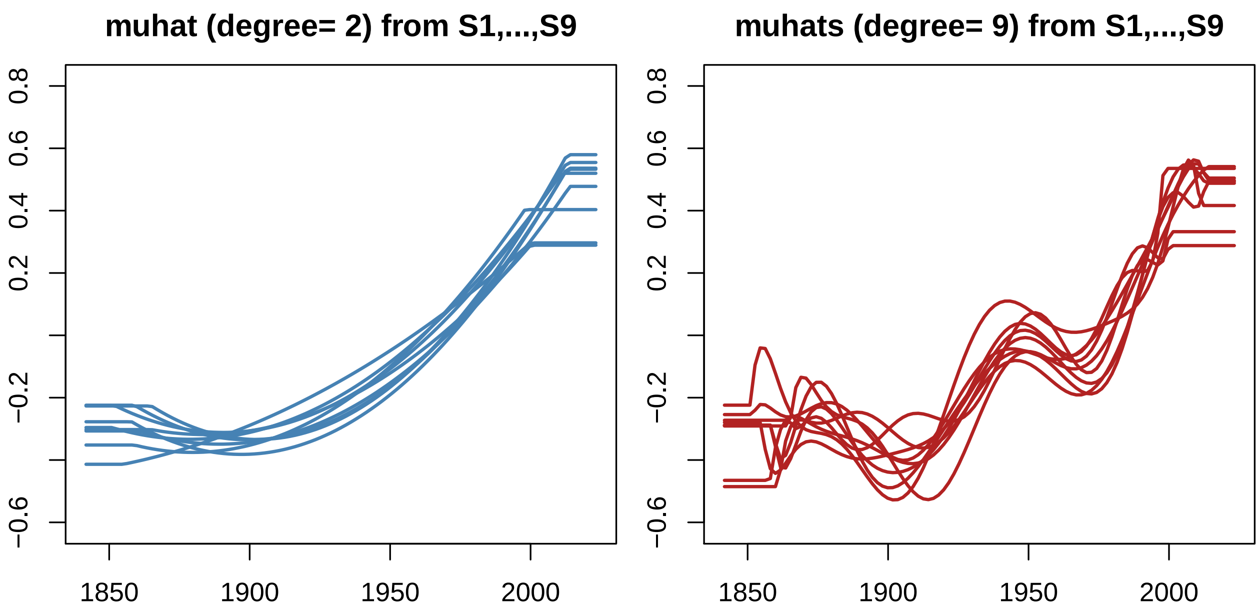
```
##                T1    T2    T3    T4    T5    T6    T7    T8    T9  Avg.
## Degree = 2 0.016 0.017 0.016 0.017 0.016 0.023 0.019 0.018 0.020 0.018
## Degree = 9 0.014 0.013 0.016 0.013 0.025 0.019 0.013 0.017 0.018 0.016
```

- Comparing this slide with the previous one what do you learn?

Sampling Variability

- Let us compare the fits across the 9 samples from above.

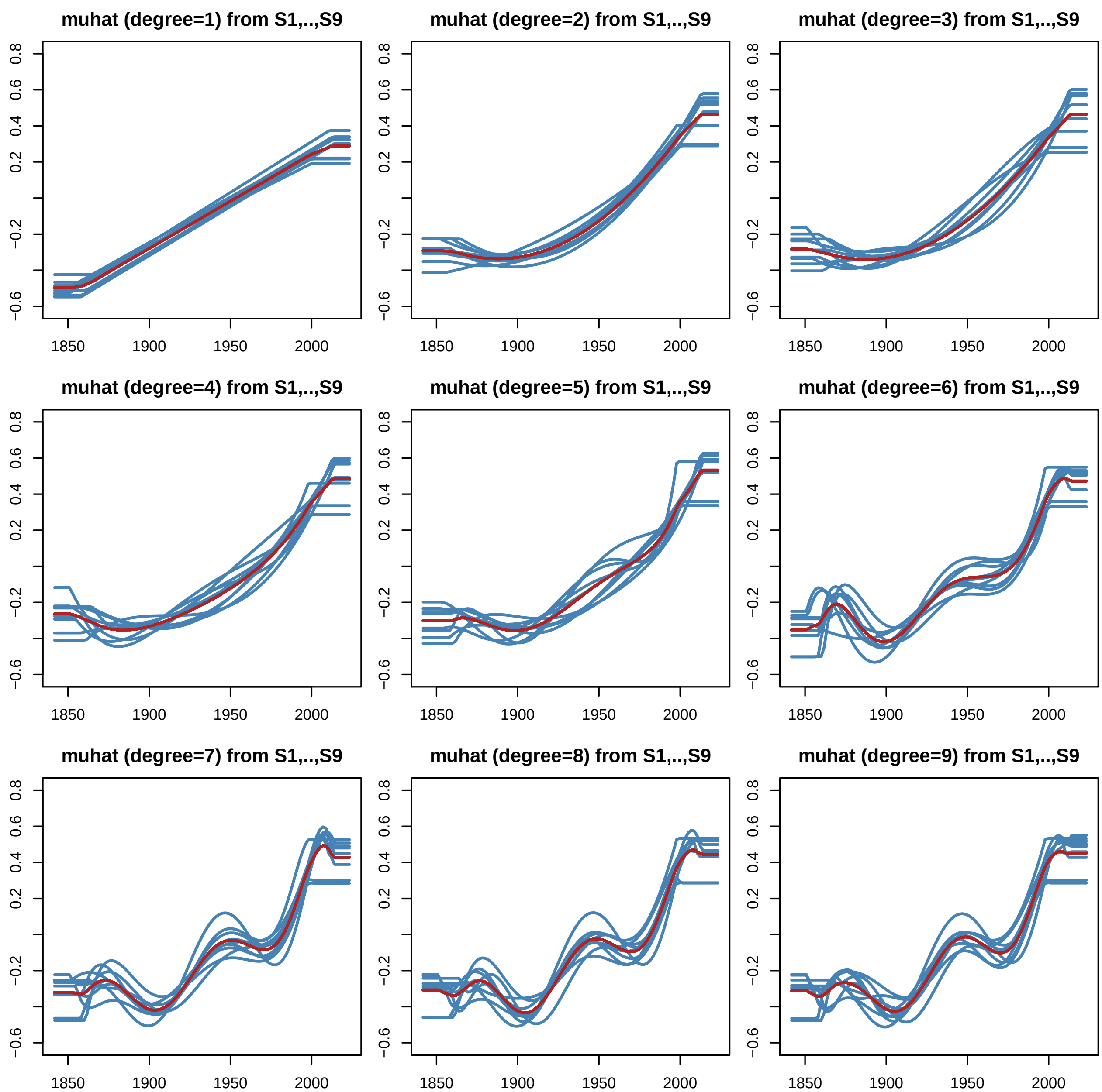
```r
par(mfrow=c(1,2), mar=2.5*c(1,1,1,0.1))

plot(Ssam[[i]], main="muhat (degree= 2) from S1,...,S9", xlab = xnam,
     ylab = ynam, pch=19, col=0, ylim=ylim, xlim=xlim)

for (i in 1:N_S) {
   tempfn =  muhats2[[i]]
   curve(tempfn, from = xlim[1], to = xlim[2], add = TRUE, col="steelblue", lwd=2)
}

plot(Ssam[[i]], main="muhats (degree= 9) from S1,...,S9", xlab = xnam,
     ylab = ynam, pch=19, col=0, ylim=ylim, xlim=xlim)
for (i in 1:N_S) {
   tempfn =  muhats9[[i]]
   curve(tempfn, from = xlim[1], to = xlim[2], add = TRUE, col="firebrick", lwd=2)
}
```



- Generate the polynomials and the averages for degrees 1 to 9 over all the samples.

```r
cset = 1:9
muhats <- Map( function(i) {lapply(Ssam, getmuhat, complexity = i)}, cset)
mubars <- Map( function(muhati) { getmubar(muhati) } , muhats)
```

- The average function is different from the function fitted on the whole population.
  - Blue lines are $\widehat{\mu}_{\mathcal{S}_i}(x)$ for $i = 1, \ldots, N_{\mathcal{S}}$
  - Red line is $\overline{\widehat{\mu}}(x) = \frac{1}{N_{\mathcal{S}}} \sum_{i=1}^{N_{\mathcal{S}}} \widehat{\mu}_{\mathcal{S}_i}(x)$

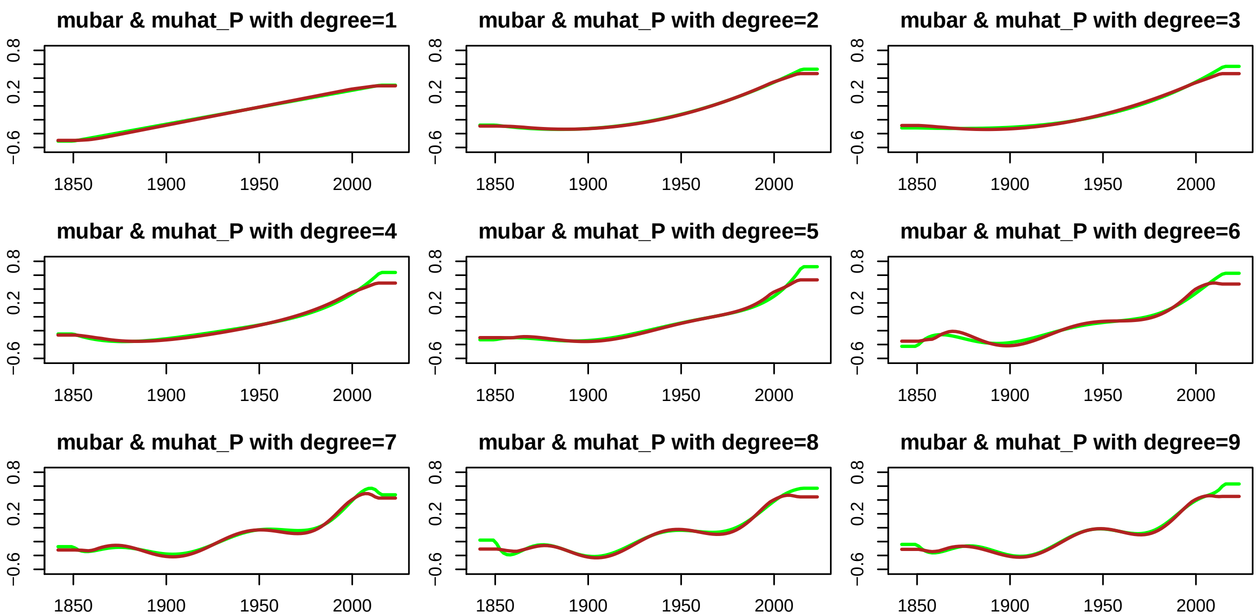


- What do you observe in these plots?

Aside

- The average function is different from the function fitted on the whole population.
- For the same degree, each plot shows a
  - Red line, representing the average fitted function $\overline{\widehat{\mu}}(x) = \frac{1}{N_{\mathcal{S}}} \sum_{i=1}^{N_{\mathcal{S}}} \widehat{\mu}_{\mathcal{S}_i}(x)$ and
  - Green line, representing the fitted function using the population denoted by $\widehat{\mu}_{\mathcal{P}}(x)$.

```
par(mfrow=c(3,3), mar=2.5*c(1,1,1,0.1))

for (i in 1:length(cset) ) {
   plot(Ssam[[i]],  main=paste0("mubar & muhat_P with degree=", cset[i],""),
        xlab = xnam, ylab = ynam, pch=19, col=0, ylim=ylim, xlim=xlim)
   tempfn = getmuhat(getXYSample(xnam, ynam, rep(TRUE,166), pop), complexity = cset[i])
   curve(tempfn, from = xlim[1], to = xlim[2], add = TRUE, col="green", lwd=2)
   tempfn =  mubars[[i]]
   curve(tempfn, from = xlim[1], to = xlim[2], add = TRUE, col="firebrick", lwd=2)
}
```



- But what is $\mu(x)$?
  - it is NOT the green line!
  - it is the piecewise function (at least, that is our best guess of it).

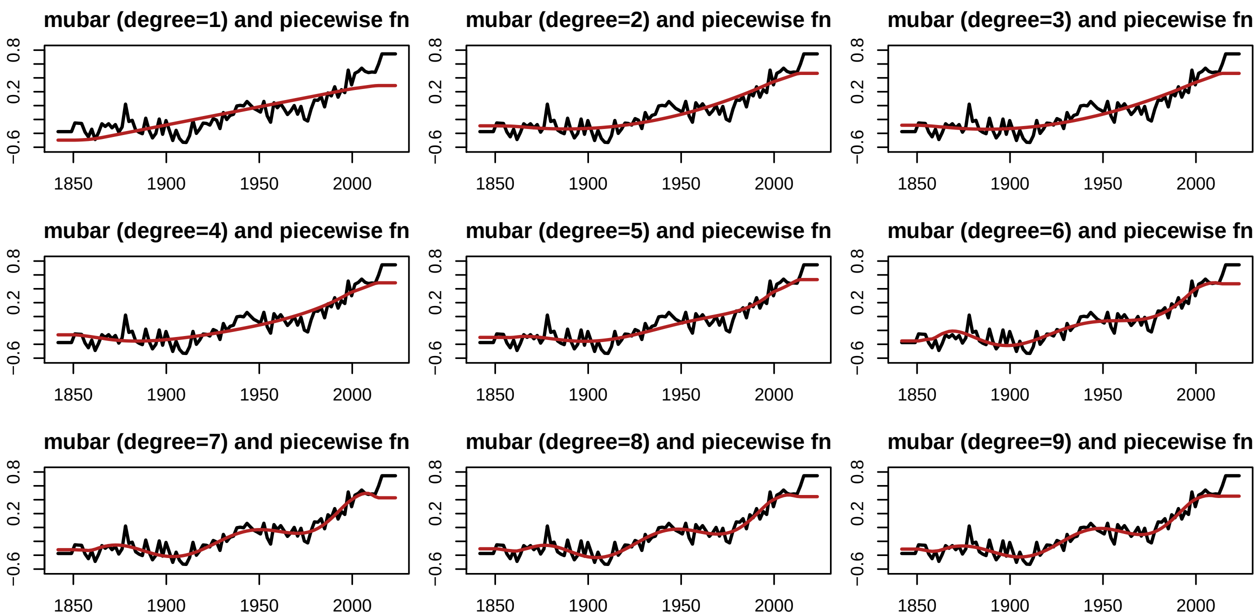
Bias

The bias concerns the difference

$$\overline{\widehat{\mu}}(\mathbf{x}_k) - \mu(\mathbf{x}_k)$$

- Let's compare the average function over the samples, i.e. $\overline{\widehat{\mu}}\,(x)$, to the piecewise function defined on the whole population. Each plot shows a
  - Red line representing $\overline{\widehat{\mu}}\,(x) = \frac{1}{N_{\mathcal{S}}}\sum_{i=1}^{N_{\mathcal{S}}} \widehat{\mu}_{\mathcal{S}_i}\,(x)$
  - Black line representing $\mu\,(x)$: the piecewise function defined on $\mathcal{P}$

```
par(mfrow=c(3,3), mar=2.5*c(1,1,1,0.1))


for (i in 1:length(cset) ) {
   plot(Ssam[[i]],  main=paste0("mubar (degree=", cset[i],") and piecewise fn"),
        xlab = xnam, ylab = ynam, pch=19, col=0, ylim=ylim, xlim=xlim)
   tempfn = getmuFun(temperature, xnam, ynam)
   curve(tempfn, from = xlim[1], to = xlim[2], add = TRUE, col="black", lwd=2)
   tempfn =  mubars[[i]]
   curve(tempfn, from = xlim[1], to = xlim[2], add = TRUE, col="firebrick", lwd=2)
}
```

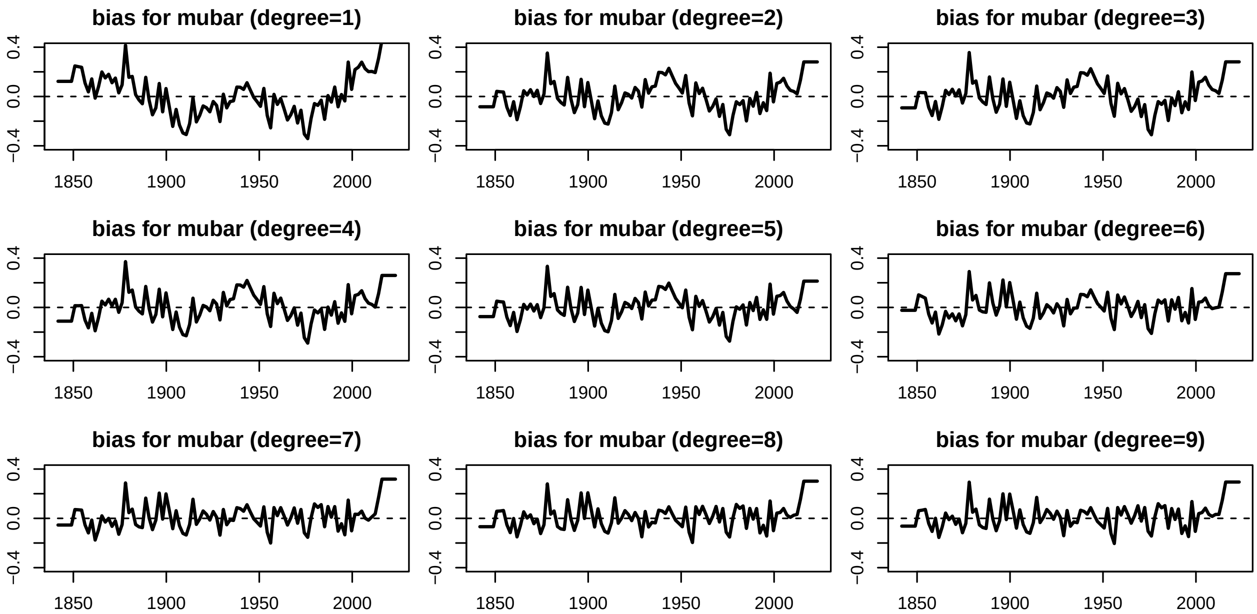


- It might be easier to view the bias by plotting the difference

$$\overline{\widehat{\mu}}\,(x) - \mu\,(x)$$

```
par(mfrow=c(3,3), mar=2.5*c(1,1,1,0.1))

for (i in 1:length(cset) ) {
   plot(Ssam[[i]],  main=paste0("bias for mubar (degree=", cset[i],")"),
```

```
        xlab = xnam, ylab = ynam, pch=19, col=0, ylim=c(-0.4, 0.4), xlim=xlim)
    tempfn1 = getmuFun(temperature, xnam, ynam)
    tempfn2 =  mubars[[i]]
    tempnew = function(z) { tempfn1(z) - tempfn2(z)}
    curve(tempnew, from = xlim[1], to = xlim[2], add = TRUE, col="black", lwd=2)
    abline(h=0, lty=2)
}
```



- To calculate the bias numerically, we need the following function.

```
bias2_mutilde <- function(Ssamples, Tsamples, mu, complexity) {
  ## get the predictor function for every sample S
  muhats <- lapply(Ssamples,
                   FUN=function(sample) getmuhat(sample, complexity)
  )
  ## 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)
              }
  )
  )
}
```

- Then we can also calculate the bias for the global temperature data

```
degrees <- 1:9
muhat = getmuFun(pop, xnam, ynam)

bias2 <-  sapply(degrees,
     FUN = function(deg){ bias2_mutilde(Ssam, Tsam, complexity = deg, mu = muhat)} )

Bmat = matrix(bias2, nrow=1, dimnames=list("[Bias(mu)]^2", paste("deg=", degrees, sep="") ) )
round(Bmat,4)
```

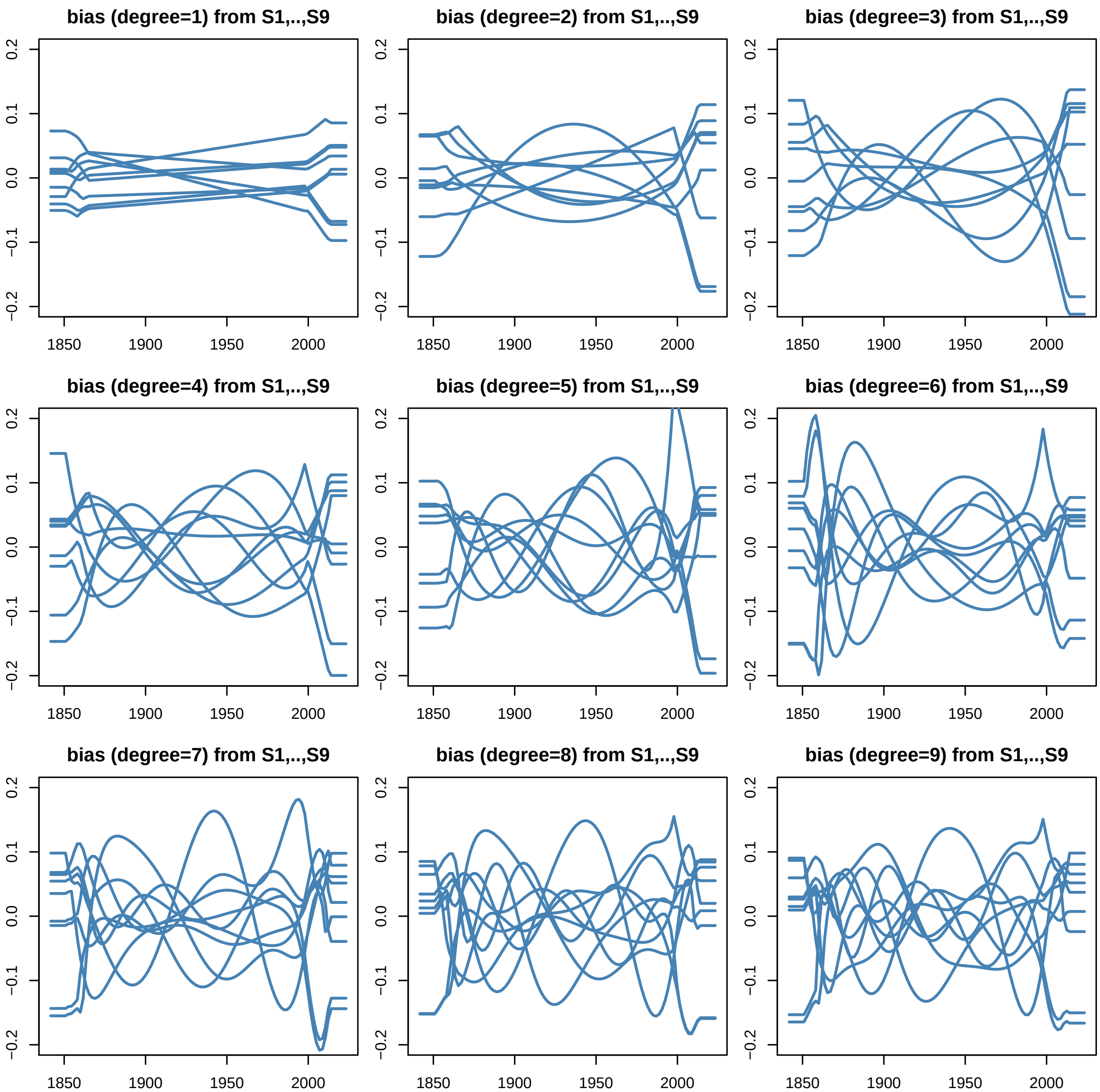
```
##               deg=1  deg=2  deg=3  deg=4  deg=5  deg=6  deg=7  deg=8 deg=9
## [Bias(mu)]^2 0.0265 0.0156 0.0156 0.0148 0.0127 0.0112 0.0102 0.0099  0.01
```

- Which estimate has the smallest bias?

Variance of the $\widetilde{\mu}_{\mathcal{S}}(x)$

- Here we examine the variability of
$$\widehat{\mu}_{\mathcal{S}_i}(x) - \overline{\widehat{\mu}}_{\mathcal{S}}(x)$$
for each $\mathcal{S}_i$, where $i = 1, \ldots, N_{\mathcal{S}}$ and varying degree of the polynomial.

```
par(mfrow=c(3,3), mar=2.5*c(1,1,1,0.1))



for (i in 1:length(cset) ) {
  plot(Ssam[[i]], main=paste0("bias (degree=", cset[i],") from S1,..,S9"),
       xlab = xnam, ylab = ynam, pch=19, col=0, ylim=c(-0.2,0.2), xlim=xlim)
    for (j in 1:N_S) {
   tempfn1 =  muhats[[i]][[j]]
   tempfn2 =  mubars[[i]]
   tempnew = function(z) { tempfn1(z) - tempfn2(z)}
   curve(tempnew, from = xlim[1], to = xlim[2], add = TRUE, col="steelblue", lwd=2)
}}
```



We can numerically calculate the variance using

```
var_mutilde <- function(Ssamples, Tsamples, complexity){
  ## get the predictor function for every sample S
  muhats <- lapply(Ssamples,
                   FUN=function(sample){
                     getmuhat(sample, complexity)
                   }
  )
  ## 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)
               }
  )
  )
}
```

Applying the function to our example.

```
degrees <- 1:9

varmu <-  sapply(degrees,
     FUN = function(deg){ var_mutilde(Ssam, Tsam, complexity = deg)} )

Vmat = matrix(varmu, nrow=1, dimnames=list("Var(mu)", paste("deg=", degrees, sep="") ) )
round(Vmat,4)
```

```
##         deg=1 deg=2  deg=3  deg=4  deg=5  deg=6  deg=7  deg=8  deg=9
## Var(mu) 0.001 0.002 0.0034 0.0034 0.0039 0.0045 0.0044 0.0045 0.0042
```

Putting it all together

- We can combine the three measurements
  - Sampling Variability
  - Bias
  - Variance of the $\widetilde{\mu}_{\mathcal{S}}(x)$

into a single measure to obtain the APSE.

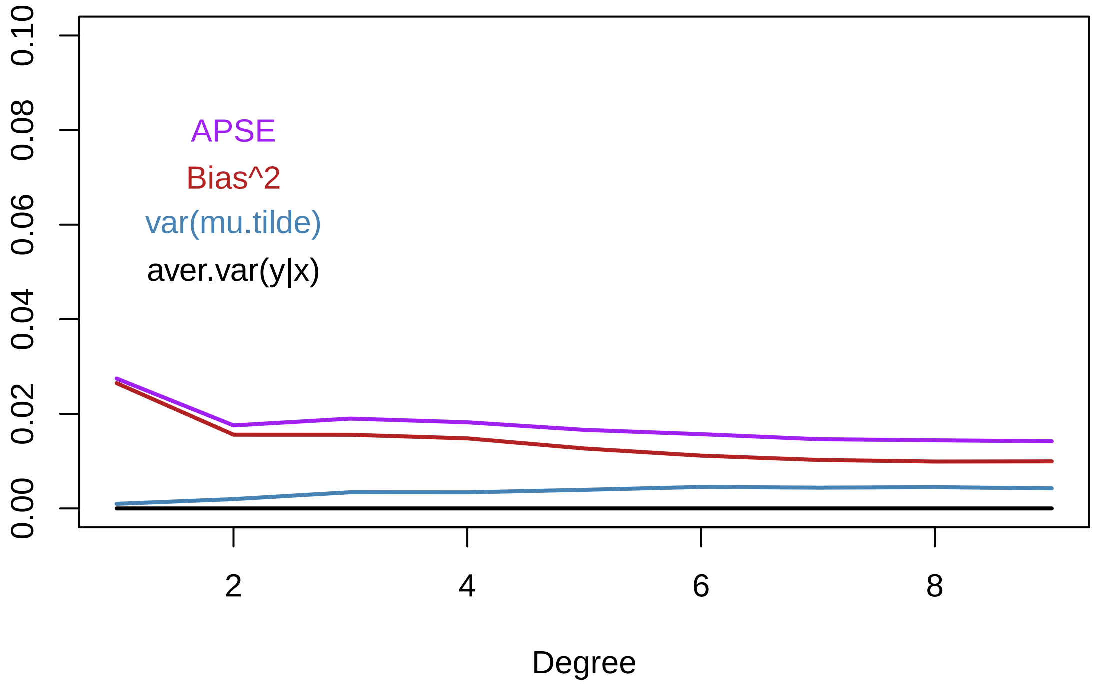
```
BVmat = rbind(0,Bmat, Vmat, Bmat+Vmat)
rownames(BVmat)[1] = "Var(y|x)"
rownames(BVmat)[4] = "APSE"
round(BVmat,3)
```

```
##              deg=1 deg=2 deg=3 deg=4 deg=5 deg=6 deg=7 deg=8 deg=9
## Var(y|x)     0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
## [Bias(mu)]^2 0.026 0.016 0.016 0.015 0.013 0.011 0.010 0.010 0.010
## Var(mu)      0.001 0.002 0.003 0.003 0.004 0.005 0.004 0.004 0.004
## APSE         0.027 0.018 0.019 0.018 0.017 0.016 0.015 0.014 0.014
```

- Note that $Var(y|x) = 0$. Why?

```
plot( 1:9, BVmat[2,], xlab="Degree", ylab="", type='l', ylim=c(0, 0.1), col="firebrick", lwd=2 )
lines( 1:9, (BVmat[1,]^2), xlab="Degree", ylab="", col="black", lwd=2 )
lines( 1:9, BVmat[3,], xlab="Degree", ylab="", col="steelblue", lwd=2 )
```

```r
lines( 1:9, BVmat[4,], col="purple", lwd=2)
text(2,0.08,'APSE',col="purple")
text(2,0.07,'Bias^2',col="firebrick")
text(2,0.06,'var(mu.tilde)',col="steelblue")
text(2,0.05,'aver.var(y|x)',col="black")
```



APSE

- We can calculate all components of APSE at once with the following code:

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

  rowMeans(sapply(1:N_S,
                  FUN=function(j){
                    T_j <- Tsamples[[j]]
                    muhat <- muhats[[j]]
                    ## Take care of any NAs
```

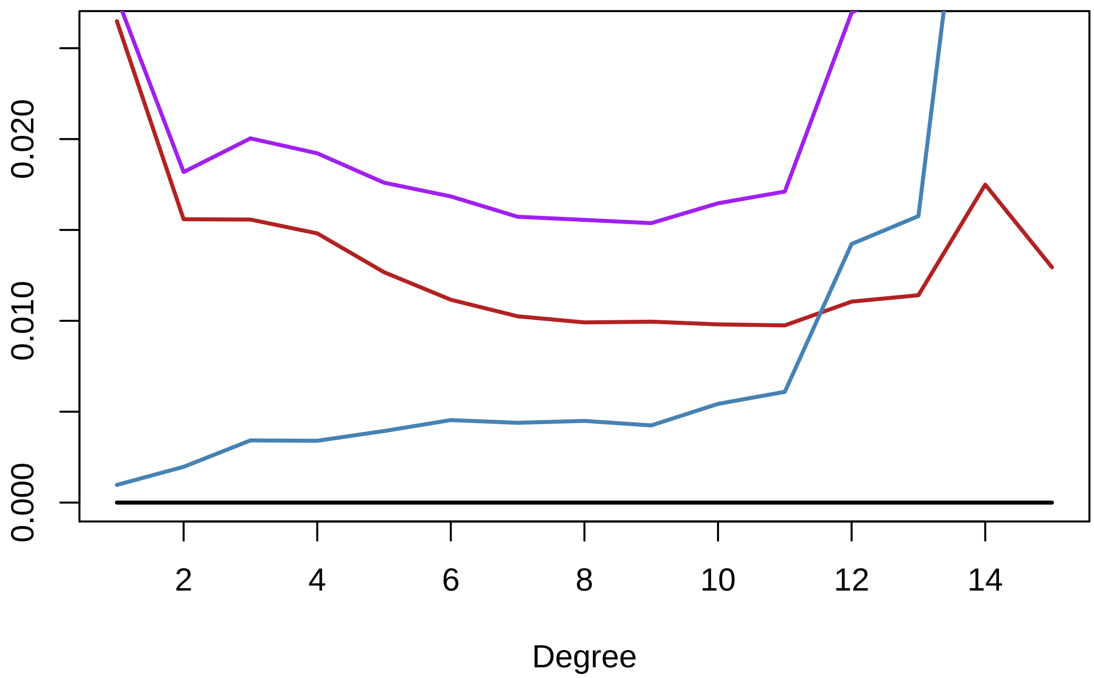
```
                    T_j <- na.omit(T_j)
                    y <- T_j$y
                    x <- T_j$x
                    mu_x <- mu(x)
                    muhat_x <- muhat(x)
                    mubar_x <- mubar(x)

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

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

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

- We extend the polynomial range as well.

```
degrees <- 1:15
muhat = getmuFun(pop, xnam, ynam)

apse_vals <-  sapply(degrees,
                     FUN = function(deg){
                       apse_all(Ssam, Tsam,
                                complexity = deg, mu = muhat)
                     }
)

colnames(apse_vals) =  paste("deg=", degrees, sep="")
round(apse_vals,3)
```

```
##              deg=1 deg=2 deg=3 deg=4 deg=5 deg=6 deg=7 deg=8 deg=9 deg=10
## apse         0.028 0.018 0.020 0.019 0.018 0.017 0.016 0.016 0.015  0.016
## var_mutilde  0.001 0.002 0.003 0.003 0.004 0.005 0.004 0.004 0.004  0.005
## bias2        0.026 0.016 0.016 0.015 0.013 0.011 0.010 0.010 0.010  0.010
## var_y        0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000  0.000
##              deg=11 deg=12 deg=13 deg=14 deg=15
## apse          0.017  0.027  0.029  0.066  0.063
## var_mutilde   0.006  0.014  0.016  0.045  0.047
## bias2         0.010  0.011  0.011  0.017  0.013
## var_y         0.000  0.000  0.000  0.000  0.000
```

```
plot( degrees, apse_vals[3,], xlab="Degree", ylab="", type='l',
      ylim=c(0, 0.026), col="firebrick", lwd=2 )
lines(degrees, apse_vals[2,], xlab="Degree", ylab="", col="steelblue", lwd=2 )
lines(degrees, apse_vals[1,], col="purple", lwd=2)
lines(degrees, apse_vals[4,], col="black", lwd=2)
```

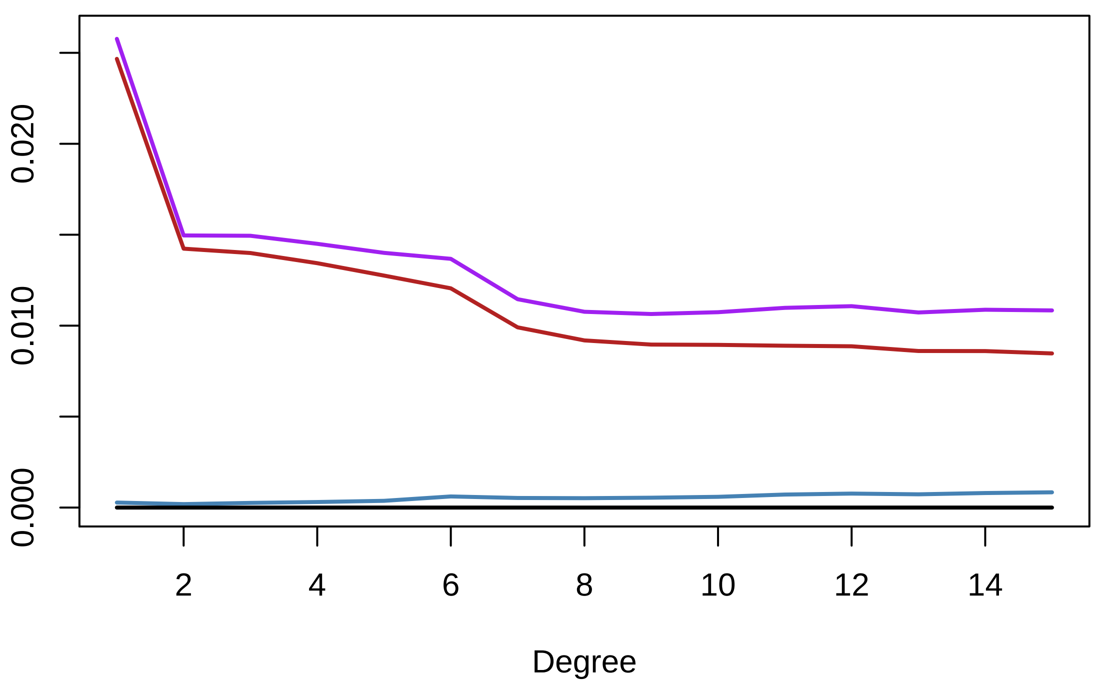


Changing n and N_S

- We might want to vary N_S and n to see our conclusion is sensitive to these inputs.
  - Here we set $N_S = 25$ and $n = 100$.

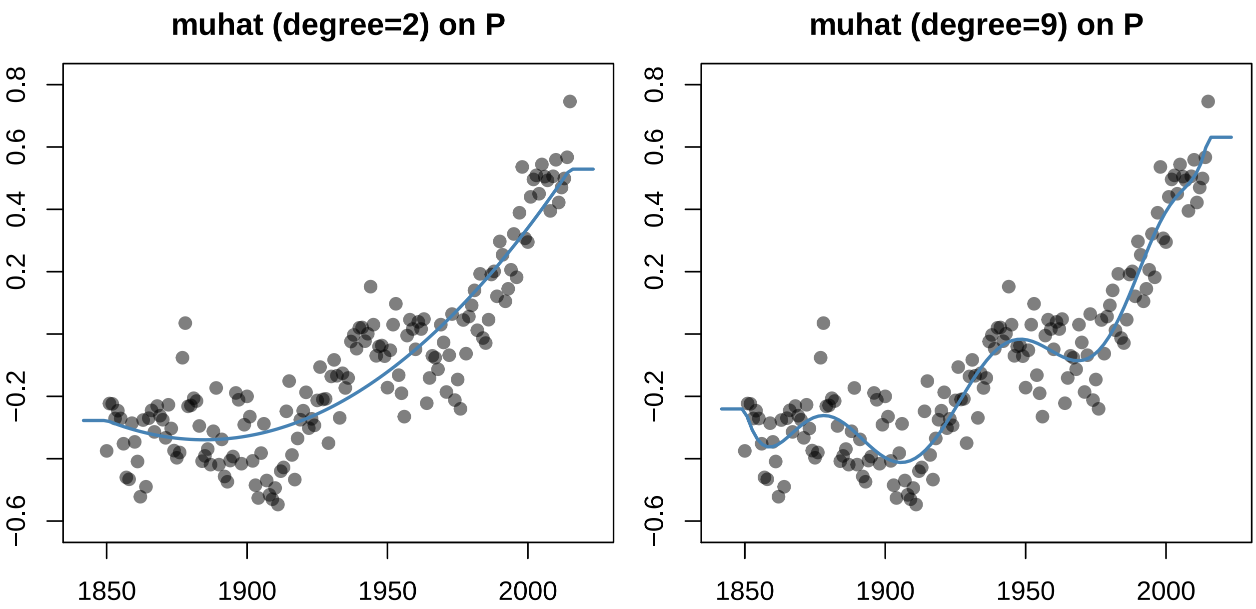
```
N_S <- 25
xnam <- "YEAR"
ynam <- "ANNUAL"
pop  <- temperature
n    <- 100
```

```r
set.seed(341)  # for reproducibility
samples <- lapply(1:N_S, FUN= function(i){getSampleComp(pop, n)})
Ssam    <- lapply(samples, FUN= function(Si){getXYSample(xnam, ynam, Si, pop)})
Tsam    <- lapply(samples, FUN= function(Si){getXYSample(xnam, ynam, !Si, pop)})


degrees <- 1:15
muhat = getmuFun(pop, xnam, ynam)

apse_vals <-  sapply(degrees,
                     FUN = function(deg){
                       apse_all(Ssam, Tsam,
                                complexity = deg, mu = muhat)
                     }
)

colnames(apse_vals) =  paste("deg=", degrees, sep="")
#round(apse_vals,5)


plot( degrees, apse_vals[3,], xlab="Degree", ylab="", type='l',
      ylim=c(0, 0.026), col="firebrick", lwd=2 )
lines(degrees, apse_vals[2,], xlab="Degree", ylab="", col="steelblue", lwd=2 )
lines(degrees, apse_vals[1,], col="purple", lwd=2)
lines(degrees, apse_vals[4,], col="black", lwd=2)
```



- Can you guess which colour is which by just looking at the trends?
  - black: $ave._x[var(y|x)]$
  - blue: $var(\widetilde{\mu}(x))$
  - red: $Bias^2$
  - purple: APSE

Plot of Temperature



Examples

Set `getmuhat` back to the original function.

```
getmuhat <- function(sampleXY, complexity = 1) {
  formula <- paste0("y ~ ",
                    if (complexity==0) {
                      "1"
                      } else
                      paste0("poly(x, ", complexity, ", raw = FALSE)")
                      #paste0("bs(x, ", complexity, ")")
  )

  fit <- lm(as.formula(formula), data = sampleXY)

  ## From this we construct the predictor function
  muhat <- function(x){
    if ("x" %in% names(x)) {
      ## x is a dataframe containing the variate named
      ## by xvarname
      newdata <- x
```

```
    } else
      ## x is a vector of values that needs to be a data.frame
    {newdata <- data.frame(x = x) }
    ## The prediction
    predict(fit, newdata = newdata)
  }
  ## muhat is the function that we need to calculate values
  ## at any x, so we return this function from getmuhat
  muhat
}
```

Growth of Loblolly pine trees.

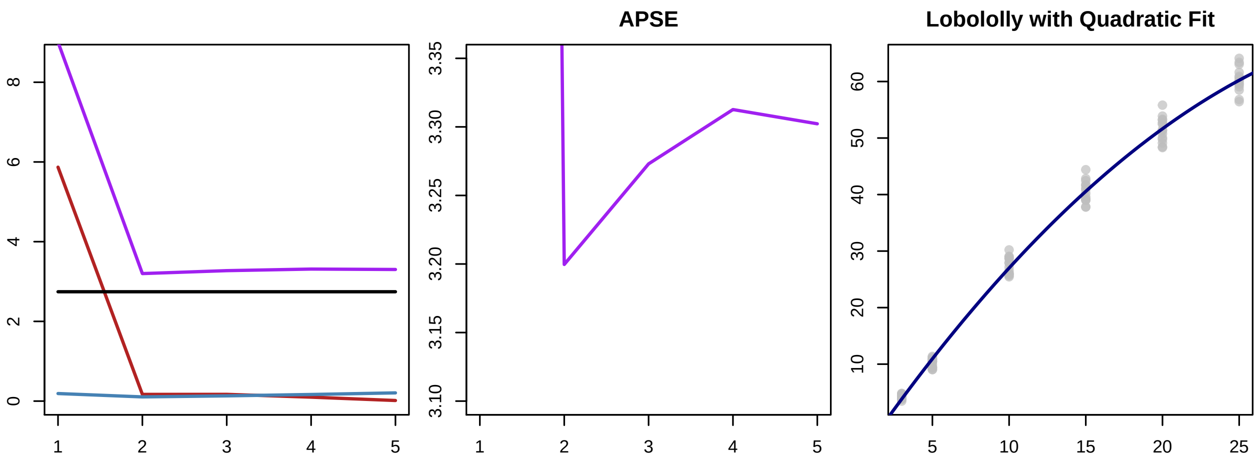
- Here we set $N_S = 25$ and $n = 40$.
- With $n = 40$ means we are using 0.6 of the population in each sample

```
N_S <- 25
xnam <- "age"
ynam <- "height"
pop  <- Loblolly
n    <- 40

set.seed(341)  # for reproducibility
samples <- lapply(1:N_S, FUN= function(i){getSampleComp(pop, n)})
Ssam    <- lapply(samples, FUN= function(Si){getXYSample(xnam, ynam, Si, pop)})
Tsam    <- lapply(samples, FUN= function(Si){getXYSample(xnam, ynam, !Si, pop)})

degrees <- 1:5
```

Then we fit polynomials of different degrees to all the samples and calculate the APSE.

```
muhat = getmuFun(pop, xnam, ynam)
set.seed(341)
apse_vals <-  sapply(degrees,
                     FUN = function(deg){
                       apse_all(Ssam, Tsam,
                                complexity = deg, mu = muhat)
                     }
)

colnames(apse_vals) =  paste("deg=", degrees, sep="")
round(apse_vals,4)
```

```
##              deg=1  deg=2  deg=3  deg=4  deg=5
## apse        9.0085 3.1996 3.2730 3.3126 3.3023
## var_mutilde 0.1897 0.1055 0.1310 0.1662 0.2065
## bias2       5.8694 0.1684 0.1692 0.0986 0.0150
## var_y       2.7440 2.7440 2.7440 2.7440 2.7440
```



- Here we form samples by selecting units without replacement.
    - What other way could we be selecting units?

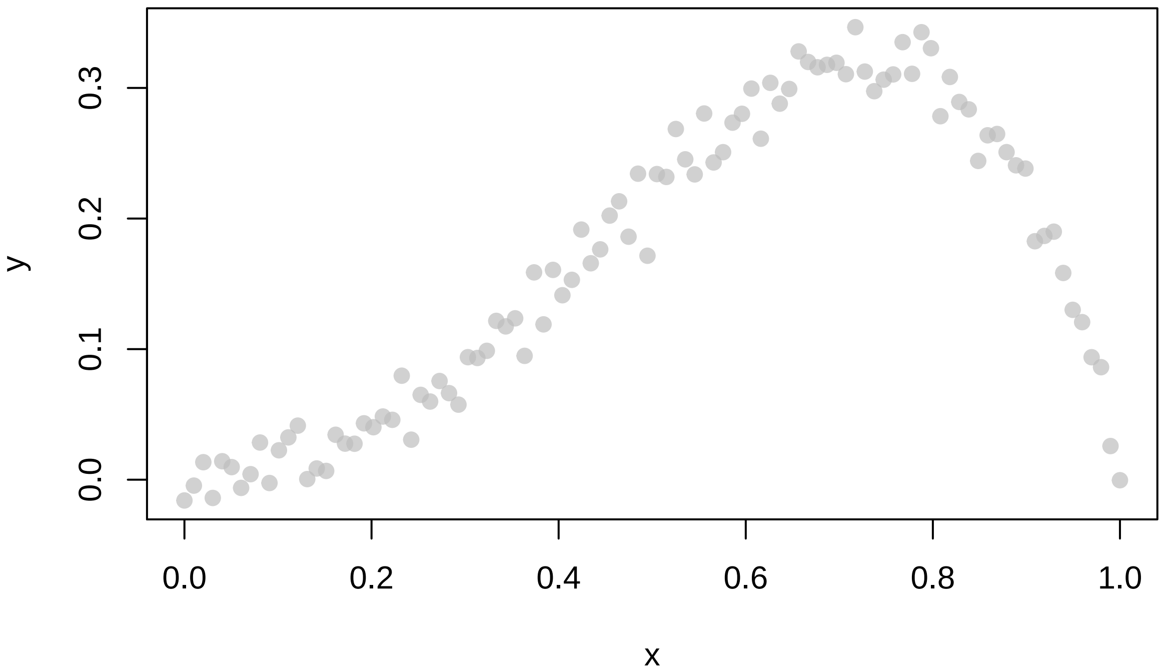
Polynomial with degree = 5

- Here we demonstrate that the chosen/fitted function is an approximation to the underlying true function (if a true function exists).
    - Recall that, in real world, “all models are wrong, but some are useful.”
    - In this particular example though, we will work with simulated data, so “the true model” does actually exist.

- Suppose that $\mu(x) = x^2 - x^5$ (true model) and we generate apopulation with $N = 100$

$$y_i = \mu(x_i) + r_i = x_i^2 - x_i^5 + r_i$$

```r
set.seed(341)
x = seq(0,1, length.out=100)
y = x^2 - x^5 + rnorm(100, sd=.015)
plot(x, y, col=adjustcolor("grey", 0.7), pch=19)
```



- We will start with $N_S = 25$ samples each of size $n = 25$.

```
datax = data.frame(x=x,y=y)
N_S  <- 25
xnam <- "x"
ynam <- "y"
pop  <- datax
n    <- 25

set.seed(341)  # for reproducibility
samples <- lapply(1:N_S, FUN= function(i){getSampleComp(pop, n)})
Ssam    <- lapply(samples, FUN= function(Si){getXYSample(xnam, ynam, Si, pop)})
Tsam    <- lapply(samples, FUN= function(Si){getXYSample(xnam, ynam, !Si, pop)})
```

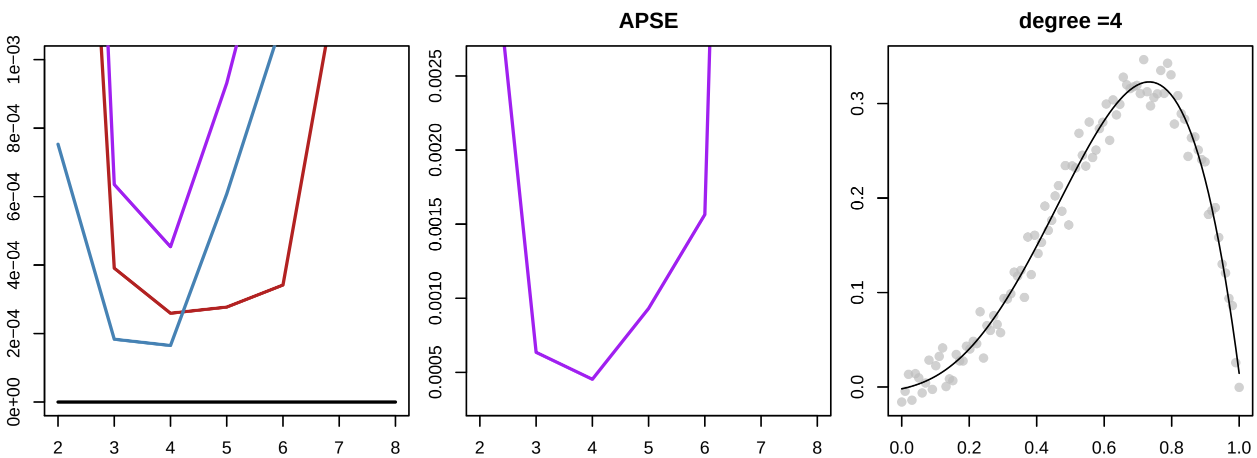
- Calculate the APSE

```
degrees <- 2:8
muhat = getmuFun(pop, xnam, ynam)

apse_vals <-  sapply(degrees,
                     FUN = function(deg){
                       apse_all(Ssam, Tsam,
                                complexity = deg, mu = muhat)
                     }
)

colnames(apse_vals) =  paste("deg=", degrees, sep="")
round(apse_vals,5)
```

```
##                deg=2   deg=3   deg=4   deg=5   deg=6   deg=7   deg=8
## apse        0.00429 0.00064 0.00045 0.00093 0.00156 0.01459 0.07619
## var_mutilde 0.00075 0.00018 0.00016 0.00061 0.00112 0.01255 0.07149
## bias2       0.00318 0.00039 0.00026 0.00028 0.00034 0.00126 0.00275
```

```
## var_y       0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
```

```
par(mfrow=c(1,3), mar=2.5*c(1,1,1,0.1))

plot( degrees, apse_vals[3,], xlab="Degree", ylab="", type='l',
      ylim=c(0, 0.001),
      col="firebrick", lwd=2 )
lines(degrees, apse_vals[2,], xlab="Degree", ylab="", col="steelblue", lwd=2 )
lines(degrees, apse_vals[1,], col="purple", lwd=2)
lines(degrees, apse_vals[4,], col="black", lwd=2)

plot( degrees, apse_vals[1,], xlab="Degree", ylab="", type='l',
      col="purple", lwd=2, ylim=c(0.00030, 0.00261), main="APSE" )

plot(datax$x, datax$y, xlab="x", ylab="y", col=adjustcolor("grey", 0.7), pch=19, main="degree =4")

samL = rep(TRUE, nrow(datax))
sample.Data <- getXYSample(xnam, ynam, samL, datax)
mu.age = getmuhat(sample.Data, complexity = 4)
curve(mu.age, 0, 1, add=TRUE, n=100)
```



$N_S = 25$ **and** $n = 75$

- Let us keep $N_S = 25$ but increase the sample size from $n = 25$ to $n = 75$.

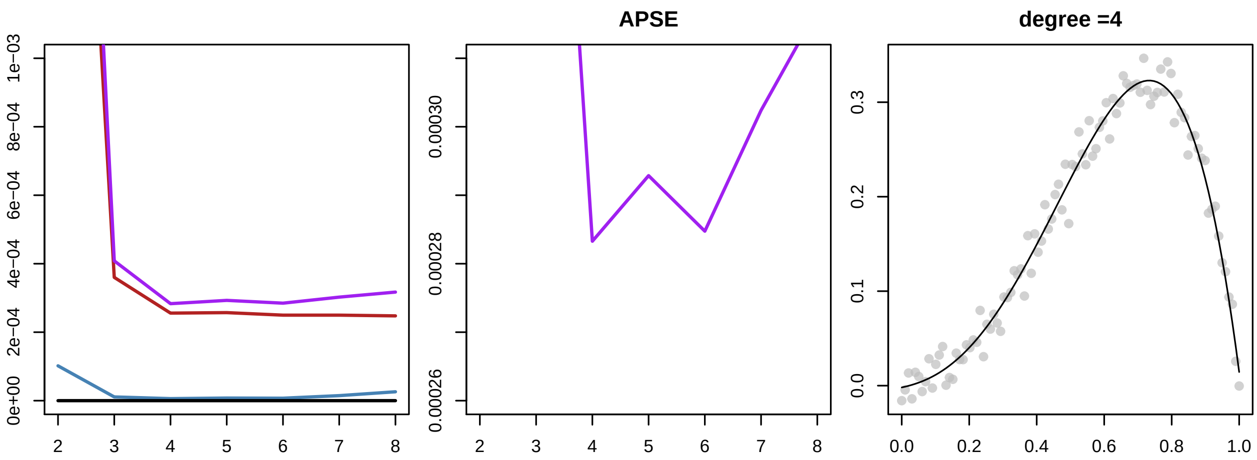
```
datax = data.frame(x=x,y=y)
N_S  <- 25
xnam <- "x"
ynam <- "y"
pop  <- datax
n    <- 75
```

```
set.seed(341)  # for reproducibility
samples <- lapply(1:N_S, FUN= function(i){getSampleComp(pop, n)})
Ssam    <- lapply(samples, FUN= function(Si){getXYSample(xnam, ynam, Si, pop)})
```

```r
Tsam    <- lapply(samples, FUN= function(Si){getXYSample(xnam, ynam, !Si, pop)})

degrees <- 2:8
muhat = getmuFun(pop, xnam, ynam)

apse_vals <-  sapply(degrees,
                     FUN = function(deg){
                       apse_all(Ssam, Tsam,
                                complexity = deg, mu = muhat)
                     }
)

colnames(apse_vals) =  paste("deg=", degrees, sep="")
round(apse_vals,5)
```

```
##                deg=2   deg=3   deg=4   deg=5   deg=6   deg=7   deg=8
## apse        0.00369 0.00041 0.00028 0.00029 0.00028 0.00030 0.00032
## var_mutilde 0.00010 0.00001 0.00001 0.00001 0.00001 0.00001 0.00003
## bias2       0.00322 0.00036 0.00026 0.00026 0.00025 0.00025 0.00025
## var_y       0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
```

```r
par(mfrow=c(1,3), mar=2.5*c(1,1,1,0.1))

plot( degrees, apse_vals[3,], xlab="Degree", ylab="", type='l',
      ylim=c(0, 0.001), col="firebrick", lwd=2 )
lines(degrees, apse_vals[2,], xlab="Degree", ylab="", col="steelblue", lwd=2 )
lines(degrees, apse_vals[1,], col="purple", lwd=2)
lines(degrees, apse_vals[4,], col="black", lwd=2)


plot( degrees, apse_vals[1,], xlab="Degree", ylab="", type='l',
      col="purple", lwd=2, ylim=c(0.00026, 0.00031), main="APSE" )


plot(datax$x, datax$y, xlab="x", ylab="y",
     col=adjustcolor("grey", 0.7), pch=19, main="degree =4")


samL = rep(TRUE, nrow(datax))
sample.Data <- getXYSample(xnam, ynam, samL, datax)
mu.age = getmuhat(sample.Data, complexity = 4)
curve(mu.age, 0, 1, add=TRUE, n=100)
```



$N_S = 25$ **and** $n = 95$

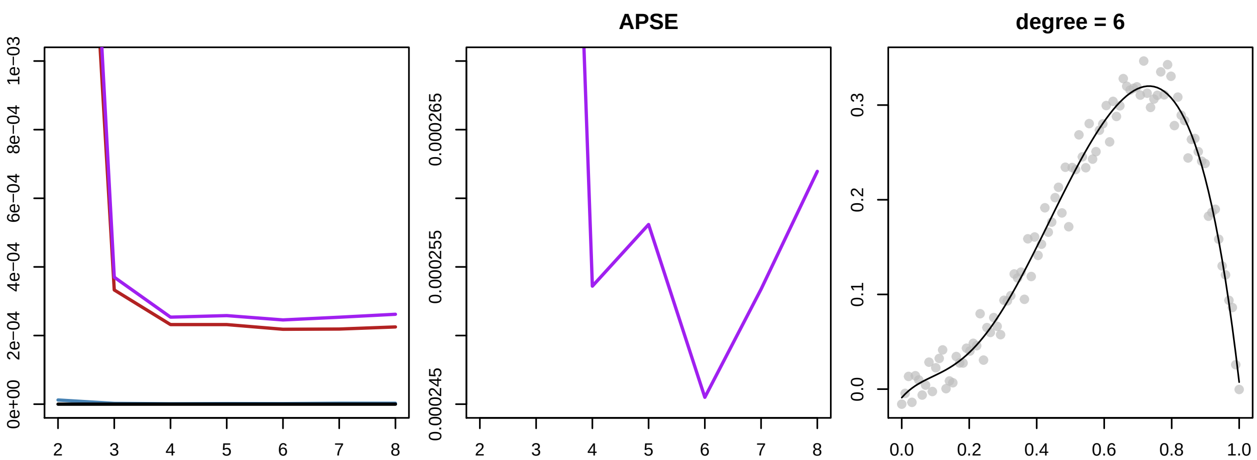
- Again, we'll keep $N_S = 25$ but increase the sample size from $n = 75$ to $n = 95$.

```
datax = data.frame(x=x,y=y)
N_S  <- 25
xnam <- "x"
ynam <- "y"
pop  <- datax
n    <- 95
```

```
set.seed(341)  # for reproducibility
samples <- lapply(1:N_S, FUN= function(i){getSampleComp(pop, n)})
Ssam    <- lapply(samples, FUN= function(Si){getXYSample(xnam, ynam, Si, pop)})
Tsam    <- lapply(samples, FUN= function(Si){getXYSample(xnam, ynam, !Si, pop)})

degrees <- 2:8
muhat = getmuFun(pop, xnam, ynam)

apse_vals <-  sapply(degrees,
                     FUN = function(deg){
                       apse_all(Ssam, Tsam,
                                complexity = deg, mu = muhat)
                     }
)

colnames(apse_vals) =  paste("deg=", degrees, sep="")
round(apse_vals,5)
```

```
##                 deg=2   deg=3   deg=4   deg=5   deg=6   deg=7   deg=8
## apse          0.00340 0.00037 0.00025 0.00026 0.00025 0.00025 0.00026
## var_mutilde   0.00001 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
## bias2         0.00314 0.00033 0.00023 0.00023 0.00022 0.00022 0.00023
## var_y         0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
```

```r
par(mfrow=c(1,3), mar=2.5*c(1,1,1,0.1))

plot( degrees, apse_vals[3,], xlab="Degree", ylab="", type='l',
      ylim=c(0, 0.001), col="firebrick", lwd=2 )
lines(degrees, apse_vals[2,], xlab="Degree", ylab="", col="steelblue", lwd=2 )
lines(degrees, apse_vals[1,], col="purple", lwd=2)
lines(degrees, apse_vals[4,], col="black", lwd=2)


plot( degrees, apse_vals[1,], xlab="Degree", ylab="", type='l',
      col="purple", lwd=2, ylim=c(0.000245, 0.00027), main="APSE" )


plot(datax$x, datax$y, xlab="x", ylab="y",
     col=adjustcolor("grey", 0.7), pch=19, main="degree = 6")


samL = rep(TRUE, nrow(datax))
sample.Data <- getXYSample(xnam, ynam, samL, datax)
mu.age = getmuhat(sample.Data, complexity = 6)
curve(mu.age, 0, 1, add=TRUE, n=100)
```



- Note that this time the APSE has chosen polynomial degree 6 for the data.
  - However, the mount of decrease in the APSE from degree 4 to degree 6 is not really significant: notice the vertical axis scale.