Accuracy of prediction

Contents

5.1 Accuracy of prediction

- Oftentimes interest lies in **predicting** the value of a variate (the **response** variate) given the value of some **explanatory** variates.

- We build a **response model** that encodes how that prediction is to be carried out.
  - The explanatory variates $\mathbf{x} = (x_1, \ldots, x_p)$ are used to explain or predict the values of the response.

- To predict, we use our observed data to construct a function, $\mu(\mathbf{x})$, which can be used to predict y at any given value $\mathbf{x}$.

$$y = \mu(\mathbf{x}) + \text{error}$$

  - e.g. for simple linear regression we have $\mu(x) = \alpha + \beta x$ and we might estimate the parameters of the function using LS.

$$\widehat{\mu}\,(x) = \widehat{\alpha} + \widehat{\beta} x$$

- How do we measure accuracy of a model?
  - It is usually easier to measure how **inaccurate** are the predictions

- One measure of inaccuracy over $\mathcal{P}$ is the **average prediction squared error** (APSE)

$$Ave_{u\in\mathcal{P}}\ (y_u - \widehat{\mu}(\mathbf{x}_u))^2$$

  - The quantity above is proportional to the more familiar residual sum of squares $\sum_{i=1}^{N} \widehat{r}_i^2$ with estimated residuals $\widehat{r}_i = y_i - \widehat{\mu}(\mathbf{x}_i)$.
  - This is sometimes called the estimated "residual mean squared error".

- We might use our accuracy (inaccuracy!) measure to choose between competing models.

- We consider two data-sets to demonstrate this concept:
  - Growth of Loblolly pine trees, and
  - Global Temperature Data.

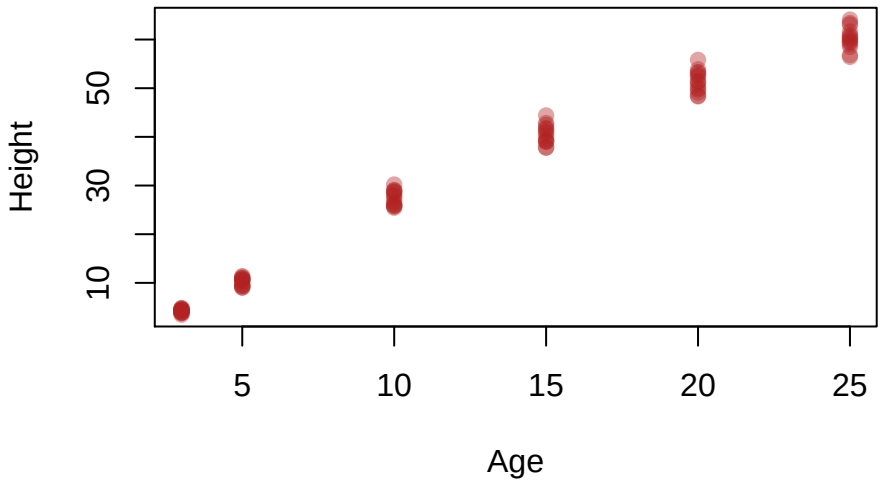
Growth of Loblolly pine trees

- The data-set has three variables;
  - the tree `height` in feet,
  - the `age` in years and
  - `Seed` the seed source for the tree.

```
data(Loblolly)
head(Loblolly)
```

```
##    height age Seed
## 1    4.51   3  301
## 15  10.89   5  301
## 29  28.72  10  301
## 43  41.74  15  301
## 57  52.70  20  301
## 71  60.92  25  301
```

```
plot(Loblolly$age, Loblolly$height, xlab="Age", ylab="Height",
     col=adjustcolor("firebrick", 0.4), pch=19)
```

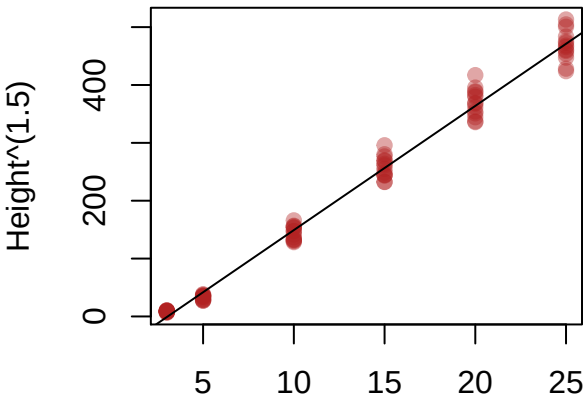


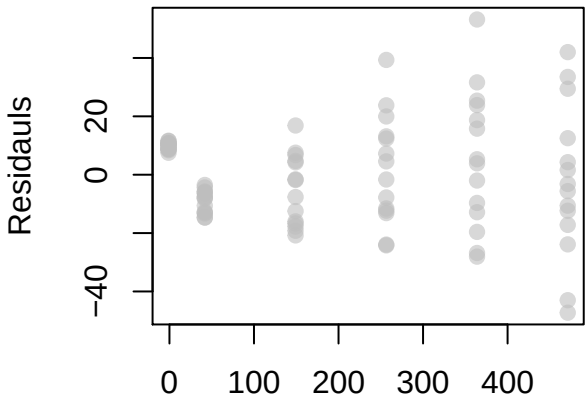
We might consider using a power transformation to make the data more linear.

Power Transformation

- The power transform, height$^{1.5}$ works well for ages 10, 15, 20 and 25 but not for 3, 5.

```
powerfun <- function(x, alpha) {
  if(sum(x <= 0) > 1) stop("x must be positive")
  if (alpha == 0)
    log(x)
  else if (alpha > 0) {
    x^alpha
  } else -x^alpha
}

par(mfrow=c(1,2) )

a0 = 1.5
plot(Loblolly$age, powerfun(Loblolly$height, a0), xlab="Age", ylab="Height^(1.5)",
     col=adjustcolor("firebrick", 0.4), pch=19)
abline(lm(I(powerfun(height, a0)) ~ age, data=Loblolly ))

model = lm(I(powerfun(height, a0)) ~ age, data=Loblolly)
plot( model$fitted.values, model$residuals,
      xlab="Fitted Values, i.e. mu.hat(x)",
      ylab="Residauls",
      pch=19, col=adjustcolor("Grey",  0.6))
```



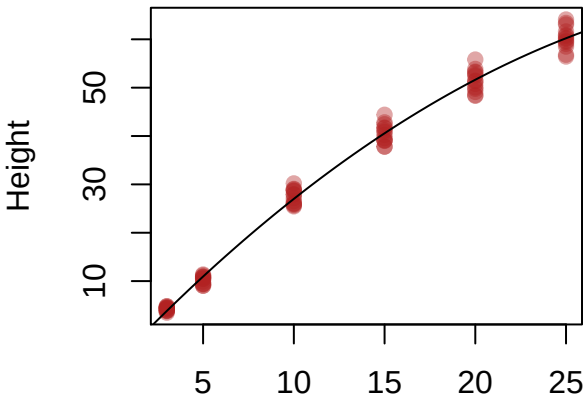


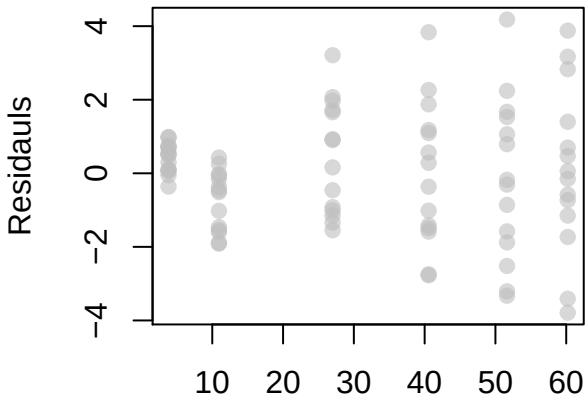
- What do you observe in these plots?
- How is the fit of the model?
- Note that the linear model has been fitted to the **transformed** data.

Fitting a Quadratic Model

- Instead of a transforming the data, we might fit a non-linear function such as a quadratic.

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

plot(Loblolly$age[order(Loblolly$age)],
     Loblolly$height[order(Loblolly$age)],
     xlab="Age", ylab="Height",
     col=adjustcolor("firebrick", 0.4), pch=19)

lm.age2 =  lm(height ~ poly(age, 2), data=Loblolly)
age.seq = seq(0, 30, by=0.1)
lines(age.seq,  predict(lm.age2,newdata=data.frame(age=age.seq)))

plot( lm.age2$fitted.values, lm.age2$residuals,
      xlab="Fitted Values, i.e. mu.hat(x)",
      ylab="Residauls",
      pch=19, col=adjustcolor("Grey",  0.6))
```





- What do you observe in these plots?
- How is the fit of the model?
- Note that the quadratic model has been fitted to the **original data**.

Fitting Different Polynomials

- If we start fitting varying degree polynomials,
  - we might want some functions to do this.
- A function that removes the variables of interest and returns a data frame containing only two variates: an x and a y:

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

- A function that uses least squares to fit a polynomial of `x` to `y`:

```
library(splines)

getmuhat <- function(sampleXY, complexity = 1) {
  formula <- paste0("y ~ ",
                    if (complexity==0) "1"
                    else {
                      if (complexity < 20 ) {
                        paste0("poly(x, ", complexity, ", raw = FALSE)")
                        ## due to Numerical overflow
                      } else {
                        ## if complexity >= 20 use a spline.
                        paste0("bs(x, ", complexity, ")")
                      }
                    }
```

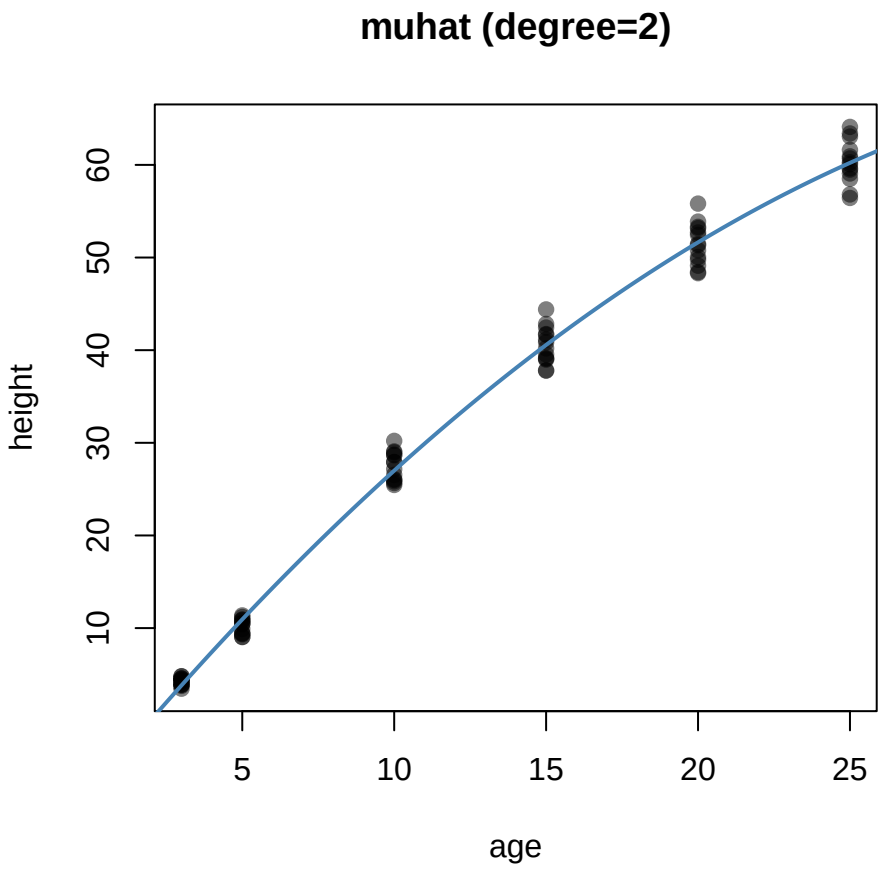
```
  )

  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 suppress warnings prediction
    suppressWarnings({
      ypred = predict(fit, newdata = newdata, silent = TRUE)    })

    ypred
  }
  ## muhat is the function that we need to calculate values
  ## at any x, so we return this function from getmuhat
  muhat
}
```

- A function to plot the data and add the fitted polynomial:

```
plotLoblollyfit <- function(muhat, complexity=NULL) {

  if ( is.null(complexity) ) title = ""
  else title = paste0("muhat (degree=", complexity,")")

plot(Loblolly[,c("age", "height")],
     main= title,
     xlab = "age", ylab = "height",
     pch=19, col= adjustcolor("black", 0.5))

xlim =  extendrange(Loblolly[, "age"])
curve(muhat, from = xlim[1], to = xlim[2],
      add = TRUE, col="steelblue", lwd=2, n=1000)

}
```

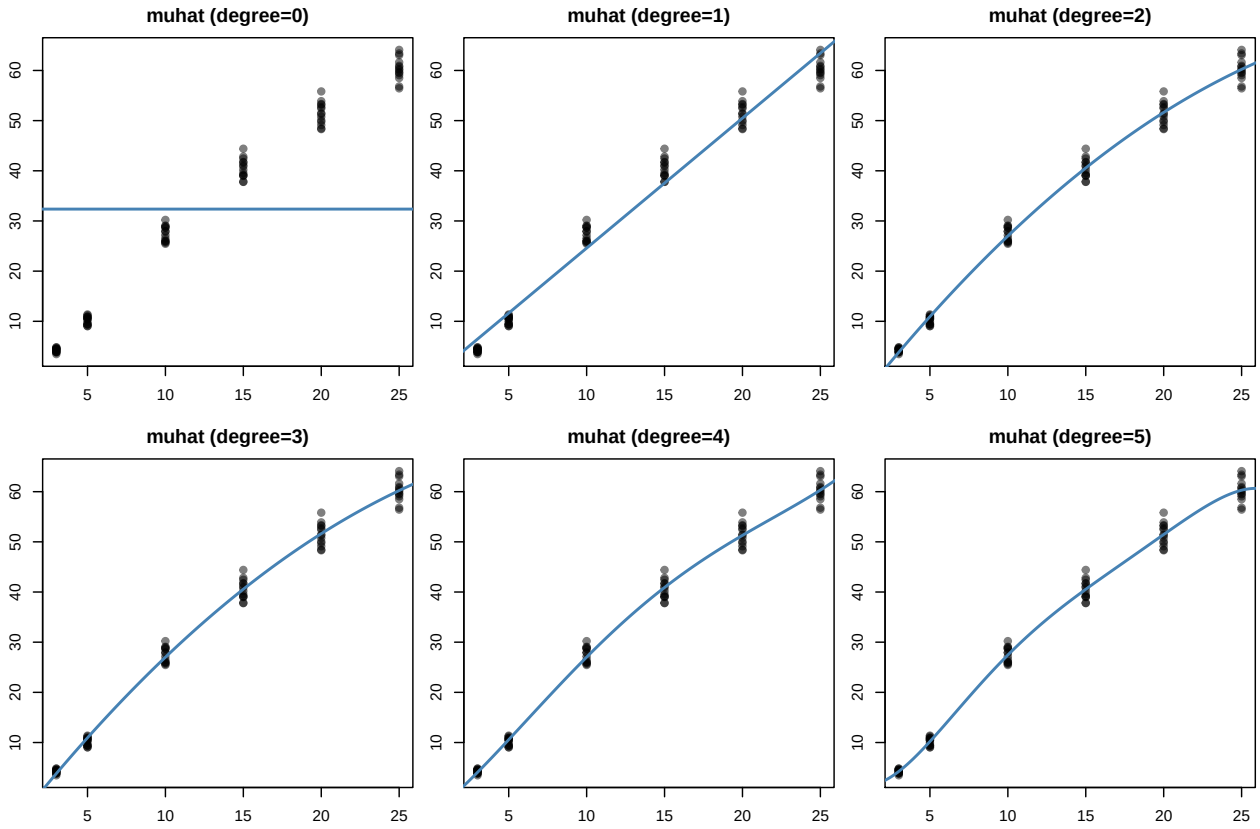
Fitting the quadratic model and plotting:

```
Lobpop= getXYpop(xvarname="age",
                 yvarname="height",
                 pop=Loblolly)

complexity = 2

muhat  =  getmuhat(Lobpop, complexity=complexity)
plotLoblollyfit(muhat, complexity)
```



Fitting Different Polynomials

```
par(mfrow=c(2,3), mar=2.5*c(1,1,1,0.1))

dset = 0:5
muhats = lapply(dset, getmuhat, sampleXY=Lobpop  )

for (deg in dset) plotLoblollyfit(muhats[[deg+1]], deg)
```



Using the APSE we obtain a numerical measure of accuracy

$$Ave_{u \in \mathcal{P}} \ (y_u - \widehat{\mu}(\mathbf{x}_u))^2$$

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

- we can calculate the APSE over the population.

```
dset = 0:5
Lob.apse = unlist( lapply(muhats, apse,
                      y= Lobpop$y, x =Lobpop$x ))


temp= rbind( 0:5, round(unlist(Lob.apse), 2) )
row.names(temp) = c("Polynomial Degree", "APSE")
dimnames(temp)[[2]] = rep("", 6)
temp
```

```
##
## Polynomial Degree   0.00 1.00 2.00 3.00 4.00 5.00
## APSE              422.31 8.48 2.79 2.79 2.72 2.64
```

- Which model is better?
- Do we gain much by making the model more complex (higher degree of the polynomial)?

Piecewise function

- To model the relationship between height and age,
  - we could fit a piecewise constant function where the height in each interval is the average value
  - This is because we only have 6 unique values for age:

```
age.unique = unique(Loblolly$age)
age.unique
```

```
## [1]  3  5 10 15 20 25
```

- A function called `getmuFun2` to construct the piecewise constant.

```
getmuFun2 <- 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]
```

```
          midx = (xlower + xhigher)/2
          if (xVal <= midx) result <- mulower
          else result <- muhigher
        }
        result
      }
    )
    ## Now return the predictions (including NAs)
    predictions
  }
  muFun
}
```

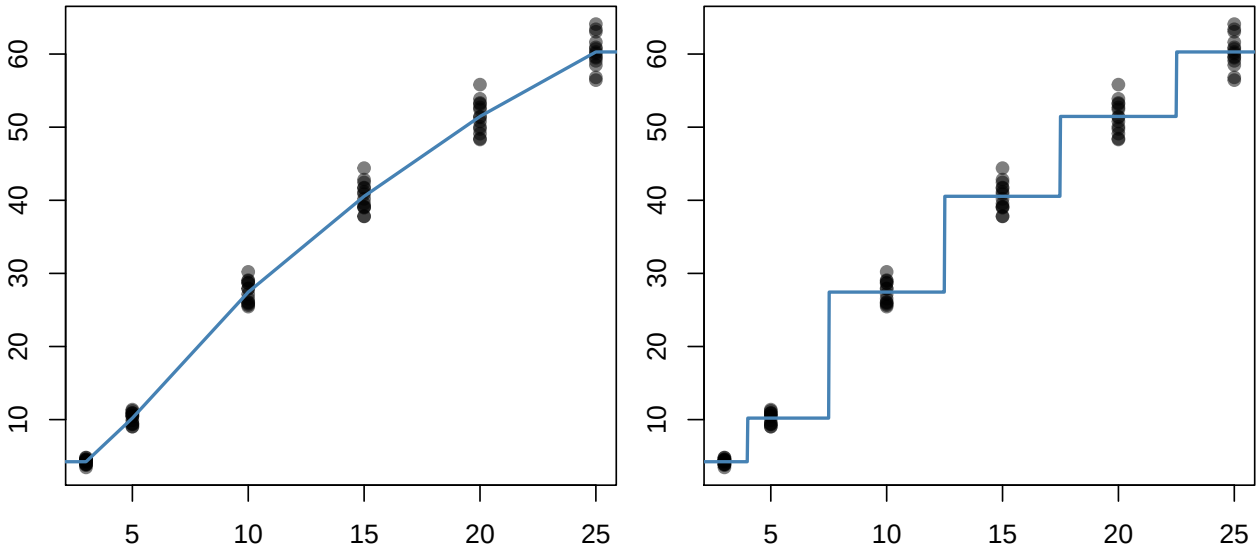
- a piecewise linear function using `getmuFun`.

```
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),
                                     method="linear")
                                     result <- interpolateFn(xVal)
        }
        result
      }
    )
    ## Now return the predictions (including NAs)
    predictions
  }
  muFun
}
```

- The two functions fitted to the population.

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

mu.age = getmuFun(pop=Loblolly,
                  xvarname="age", yvarname="height")
plotLoblollyfit(mu.age)

mu.age2 = getmuFun2(pop=Loblolly,
                  xvarname="age", yvarname="height")
plotLoblollyfit(mu.age2)
```



- Which function fits the population better?

Global Temperature Data

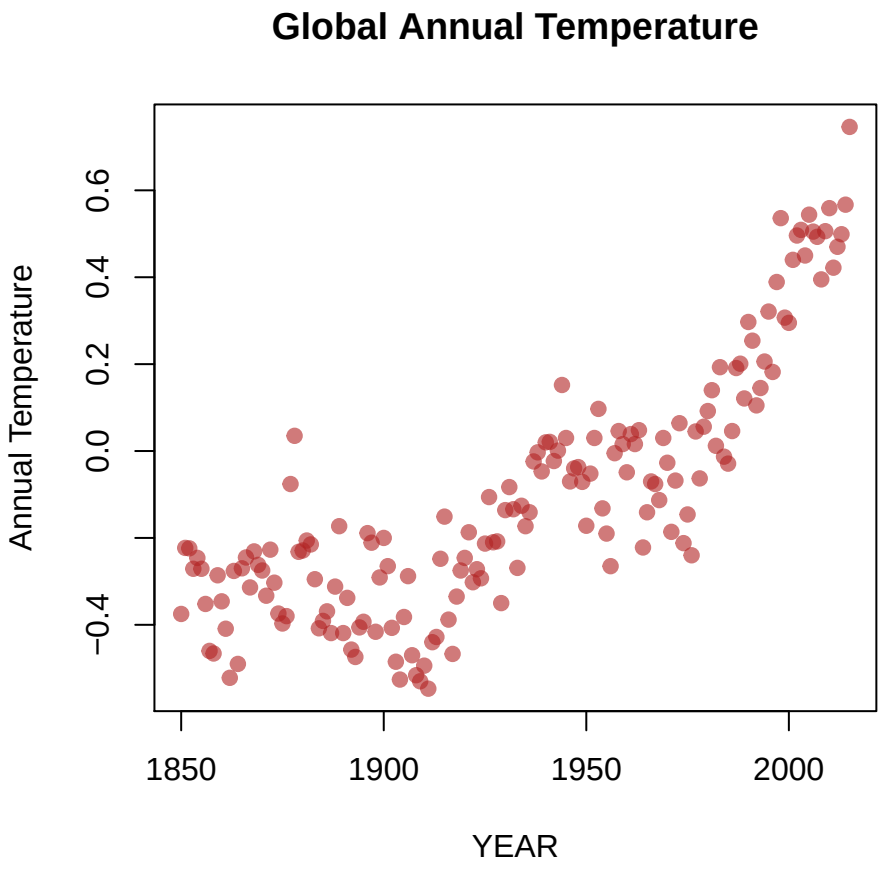
- The annual temperature over years.

```
temperature <- read.csv("../Data/Temperature.csv")
head(temperature)
```

```
##   YEAR    JAN    FEB    MAR    APR    MAY    JUN    JUL    AUG    SEP
## 1 1850 -0.702 -0.284 -0.732 -0.570 -0.325 -0.213 -0.128 -0.233 -0.444
## 2 1851 -0.303 -0.362 -0.485 -0.445 -0.302 -0.189 -0.215 -0.153 -0.108
## 3 1852 -0.308 -0.477 -0.505 -0.559 -0.209 -0.038 -0.016 -0.195 -0.125
## 4 1853 -0.177 -0.330 -0.318 -0.352 -0.268 -0.179 -0.059 -0.148 -0.409
## 5 1854 -0.360 -0.280 -0.284 -0.349 -0.230 -0.215 -0.228 -0.163 -0.115
## 6 1855 -0.176 -0.400 -0.303 -0.217 -0.336 -0.160 -0.268 -0.159 -0.339
##      OCT    NOV    DEC ANNUAL
## 1 -0.452 -0.190 -0.268 -0.375
## 2 -0.063 -0.030 -0.067 -0.223
## 3 -0.216 -0.187  0.083 -0.224
## 4 -0.359 -0.256 -0.444 -0.271
## 5 -0.188 -0.369 -0.232 -0.246
## 6 -0.211 -0.212 -0.510 -0.271
```

- Consider a scatter plot of the annual temperature versus year.
- We are interested in modelling temperature (y) as a function of year (x).

```
plot(temperature$YEAR, temperature$ANNUAL, xlab="YEAR", ylab="Annual Temperature",
     main="Global Annual Temperature", col=adjustcolor("firebrick", 0.6), pch=19 )
```

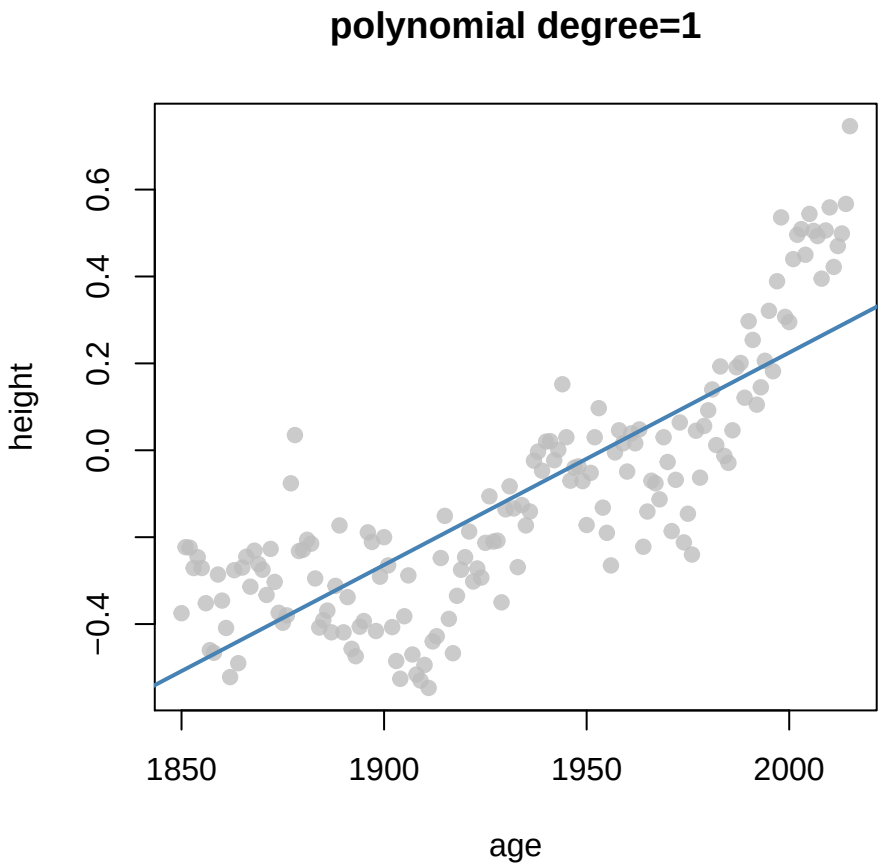


Linear regression

```
plotTemperaturefit <- function(muhat, complexity=NULL) {

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

plot(temperature[,c("YEAR", "ANNUAL")],
     main= title,
     xlab = "age", ylab = "height",
     pch=19, col= adjustcolor("Grey", 0.8))

xlim =  extendrange(temperature[, "YEAR"])
curve(muhat, from = xlim[1], to = xlim[2],
      add = TRUE, col="steelblue", lwd=2, n=1000)

}
```

```
temppop= getXYpop(xvarname="YEAR", yvarname="ANNUAL",
                  pop=temperature)
```

```
muhat  =  getmuhat(temppop, complexity=1)
plotTemperaturefit(muhat, 1)
```

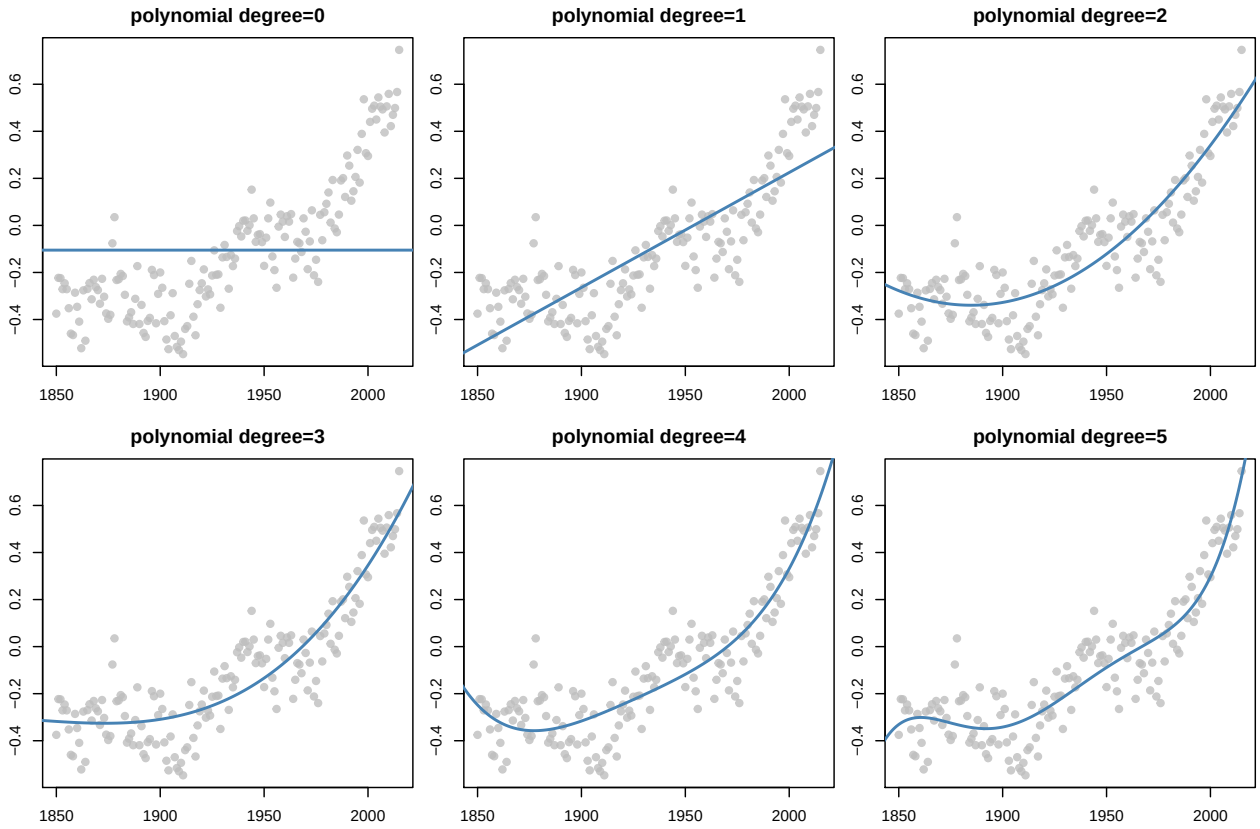


- How does the simple linear model perform?
- How does it fit the data?
- How does it perform in prediction?

Polynomial Regression (low degree)

- Consider fitting a polynomial of degree 0 to 5.

```
par(mfrow=c(2,3), mar=2.5*c(1,1,1,0.1))

dset = 0:5
muhats = lapply(dset, getmuhat, sampleXY=temppop  )

for (i in 1:length(dset) ) plotTemperaturefit(muhats[[i]], dset[i])
```



- we can calculate the APSE over the population.

```
dset = 0:5
temp.apse = unlist( lapply(muhats, apse,
                      y= temppop$y, x =temppop$x ))


tab = rbind( 0:5, round(unlist(temp.apse), 4) )
row.names(tab) = c("Polynomial Degree", "APSE")
dimnames(tab)[[2]] = rep("", 6)
tab
```

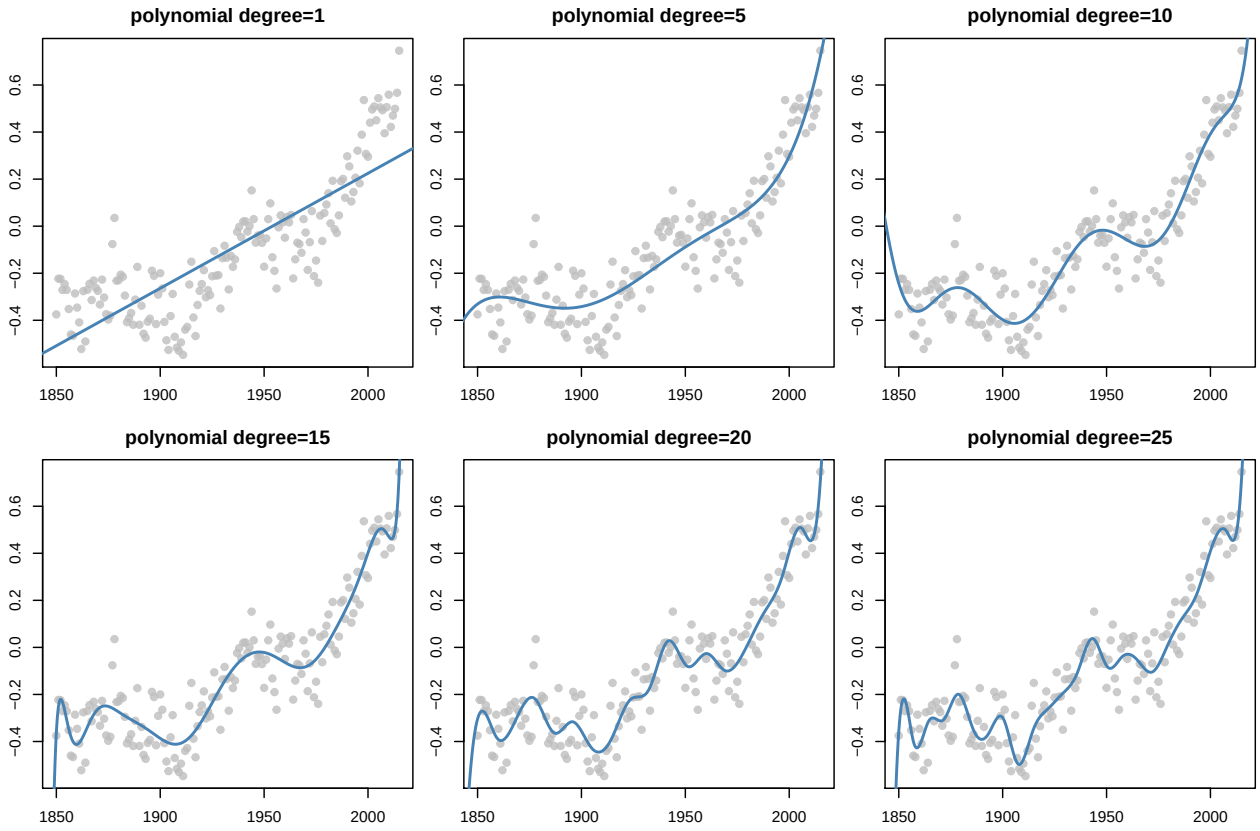
```
##
## Polynomial Degree 0.0000 1.0000 2.0000 3.0000 4.0000 5.0000
## APSE              0.0807 0.0258 0.0148 0.0145 0.0139 0.0132
```

- Which model is better?
- Do we gain much by making the model more complex (higher degree of the polynomial)?

Polynomial Regression (high degree)

- Consider fitting a polynomials of degree 1, 5,10,15,20 and 25 to Global Annual Temperature.

```
par(mfrow=c(2,3), mar=2.5*c(1,1,1,0.1))

dset = c(1, 5,10,15,20,25)

muhats = lapply(dset, getmuhat, sampleXY=temppop  )
for (i in 1:length(dset) ) plotTemperaturefit(muhats[[i]], dset[i])
```



The average prediction square error (APSE) from each model

```
temp.apse = unlist( lapply(muhats, apse,
                      y= temppop$y, x =temppop$x ))

dset = c(1, 5,10,15,20,25)

tab = rbind( dset, round(unlist(temp.apse), 4) )
row.names(tab) = c("Polynomial Degree", "APSE")
dimnames(tab)[[2]] = rep("", 6)
tab
```

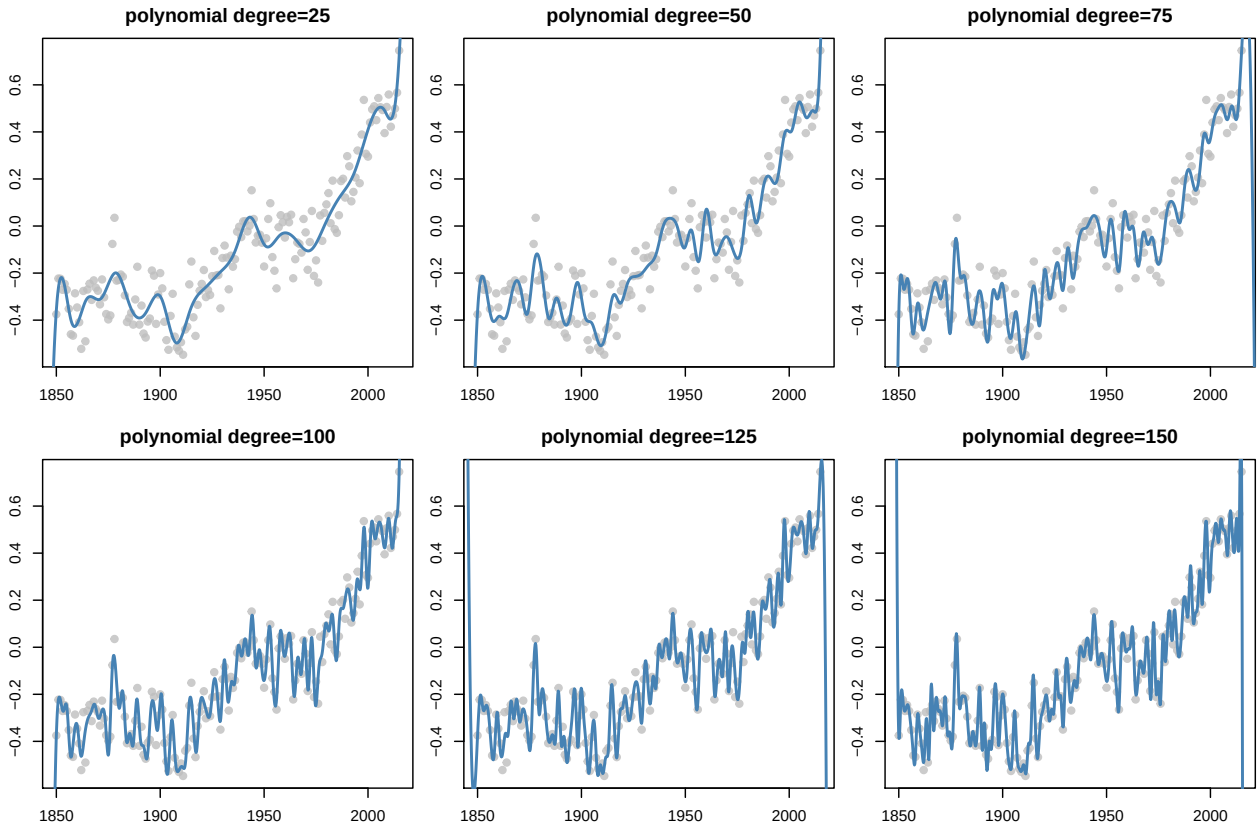
```
##
## Polynomial Degree 1.0000 5.0000 1e+01 15.0000 20.0000 25.0000
## APSE              0.0258 0.0132 9e-03  0.0084  0.0077  0.0071
```

- Do we gain much by making the model more complex (higher degree of the polynomial)?
- What do you learn from this?

Polynomial Regression (higher degree)

- Consider fitting a polynomials of degree 20, 50, 75, 100, 125 and 165 to Global Annual Temperature.

```
par(mfrow=c(2,3), mar=2.5*c(1,1,1,0.1))

dset = c(25, 50, 75, 100, 125, 150)
muhats = lapply(dset, getmuhat, sampleXY=temppop  )

for (i in 1:length(dset)) plotTemperaturefit(muhats[[i]], dset[i])
```



The average prediction square error (APSE) from each model

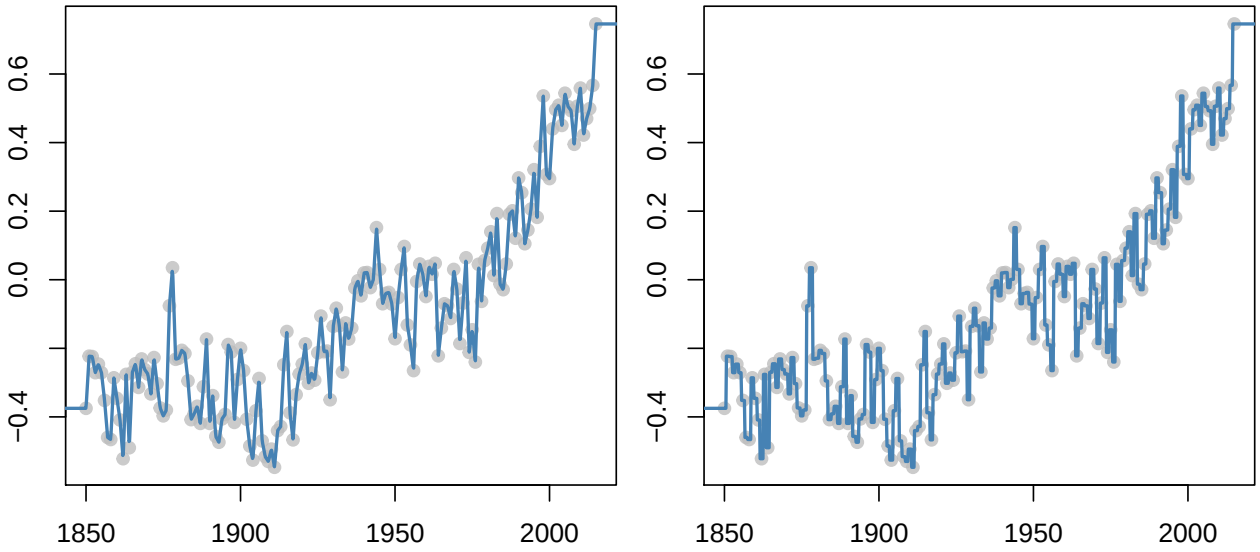
```
dset = c(25, 50, 75, 100, 125, 150)
temp.apse = unlist( lapply(muhats, apse,
                      y= temppop$y, x =temppop$x ))

tab = rbind( dset, round(unlist(temp.apse), 5) )
row.names(tab) = c("Polynomial Degree", "APSE")
dimnames(tab)[[2]] = rep("", 6)
tab
```

```
##
## Polynomial Degree 25.00000 50.00000 75.00000 1.00e+02 1.25e+02 1.5e+02
## APSE               0.00714  0.00544  0.00368 1.76e-03 1.11e-03 3.7e-04
```

Piecewise function

- To model this relationship we could fit
  - a piecewise constant function where the height in each interval is the average value or
  - using `getmuFun` piecewise linear function.

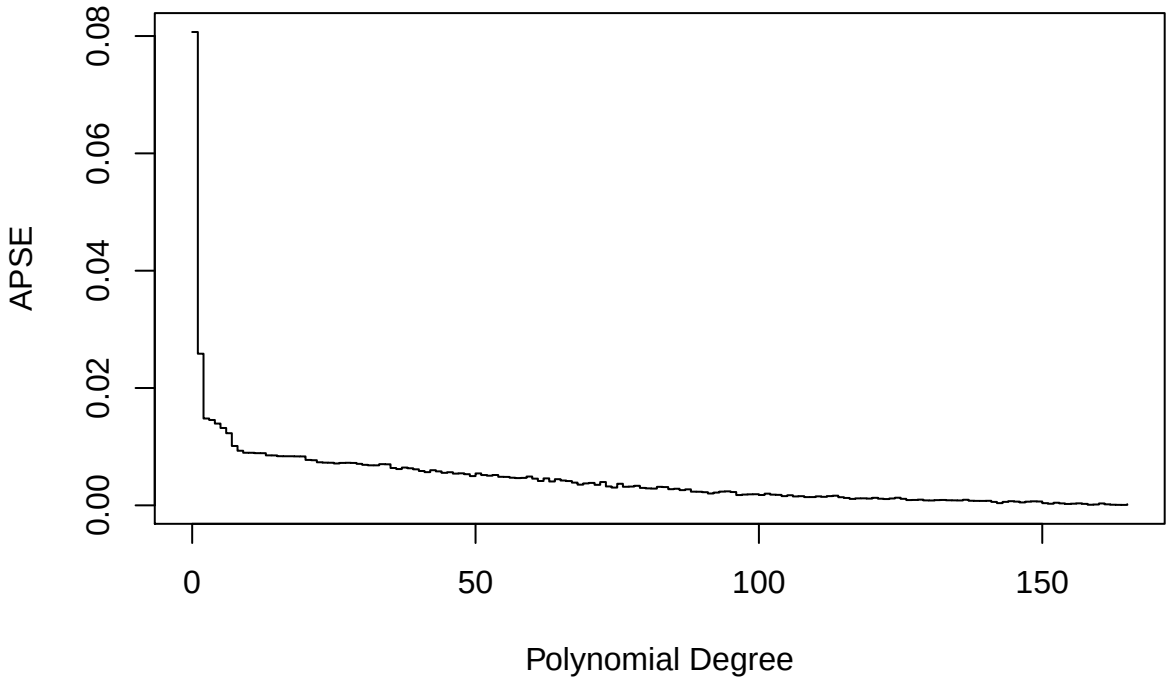
```
par(mfrow=c(1,2), mar=2.5*c(1,1,1,0.1))

mu.year = getmuFun(pop=temperature,
                   xvarname="YEAR", yvarname="ANNUAL")
plotTemperaturefit(mu.year)

mu.year2 = getmuFun2(pop=temperature,
                   xvarname="YEAR", yvarname="ANNUAL")
plotTemperaturefit(mu.year2)
```



- Comparing the two, what do you observe?
- Is this level of *complexity* appropriate?

The residuals/APSE versus Degree

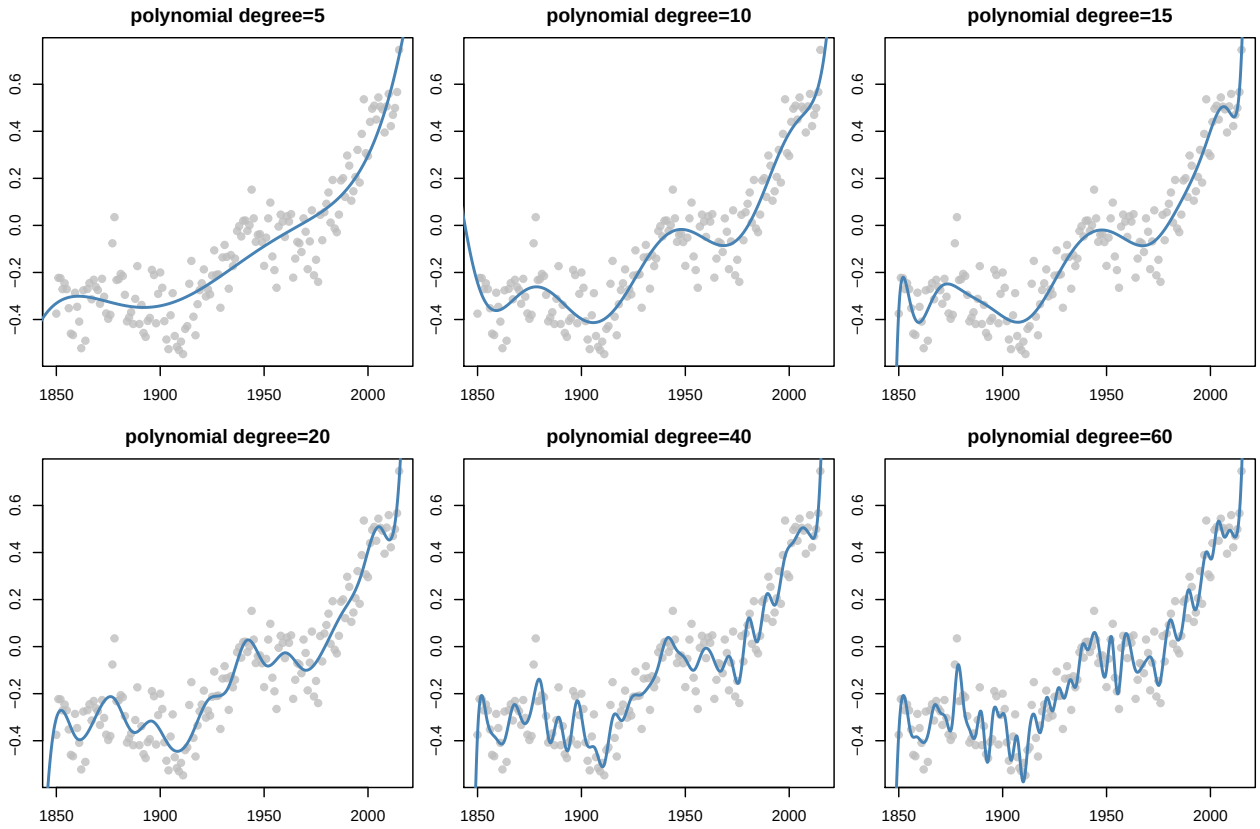
- What level of *complexity*, i.e. what degree of polynomial is appropriate?
  - To answer the question, let's look at the amount of improvement of the fit of the model as a function of the polynomial degree fitted to the data.
- What amount of residual variation, i,e, APSE is acceptable?



Polynomial Regression

- Consider fitting a polynomials of degree 5, 10, 15, 20, 40 and 60 to Global Annual Temperature.

```
par(mfrow=c(2,3), mar=2.5*c(1,1,1,0.1))

dset = c(5, 10, 15, 20, 40, 60)
muhats = lapply(dset, getmuhat, sampleXY=temppop  )

for (i in 1:length(dset)) plotTemperaturefit(muhats[[i]], dset[i])
```



The average prediction square error (APSE) from each model

```
dset = c(5, 10, 15, 20, 40, 60)
temp.apse = unlist( lapply(muhats, apse,
                      y= temppop$y, x =temppop$x ))

tab = rbind( dset, round(unlist(temp.apse), 5) )
row.names(tab) = c("Polynomial Degree", "APSE")
dimnames(tab)[[2]] = rep("", 6)
tab
```

```
##
## Polynomial Degree 5.00000 10.00000 15.00000 20.00000 40.00000 60.00000
## APSE              0.01319  0.00896  0.00838  0.00774  0.00586  0.00455
```

- Comment on the results.

Measuring Inaccuracy

- So far, we have been fitting polynomial models to data, and comparing the residuals sum of squares across different polynomial degrees.

$$Ave_{u\in\mathcal{P}}(y_u - \widehat{\mu}(\mathbf{x}_u))^2.$$

- In this approach we estimate the predictor function and measure inaccuracy based on the **same set** of observations.

- This doesn't seem to be the most honest way to estimate a prediction's inaccuracy.
  - This method will underestimate the average squared error for **prediction** of any other values of x not existing in the data (new values).
  - The data-set to fit the model and the test data-set are the same (bad idea).

- A predictor function $\mu(\mathbf{x})$ is intended to **predict** the response variate value y_u for any unit u selected from a population $\mathcal{P}$.
  - Our measure of inaccuracy needs to reflect this.

The average prediction squared error (APSE)

- Our current inaccuracy measure can be written as

$$Ave_{u\in\mathcal{P}}(y_u - \widehat{\mu}(\mathbf{x}_u))^2 = Ave_{u\in\mathcal{P}}(y_u - \widehat{\mu}_{\mathcal{P}}(\mathbf{x}_u))^2$$

- A better measure of inaccuracy measure should include some units that are not used to construct the predictor function.
  - So we estimate the predictor function using a sample $\mathcal{S}$, *and*
  - measure the inaccuracy over the population $\mathcal{P}$, or over the portion of the population not included in $\mathcal{S}$ so that it is a fair evaluation of prediction power.

- Then the **average prediction squared error** (APSE) of $\widehat{\mu}$ is

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

  - This notation emphasizes that the estimate of predictor function $\widehat{\mu}$ is based on a sample $\mathcal{S}$

The Test Set

- Since $\mathcal{S} \subset \mathcal{P}$, we can write the average prediction squared error as

$$\begin{aligned} APSE(\mathcal{P}, \widehat{\mu}_{\mathcal{S}}) &= Ave_{u\in\mathcal{P}}(y_u - \widehat{\mu}_{\mathcal{S}}(\mathbf{x}_u))^2 \\ &= \left(\tfrac{n}{N}\right) Ave_{u\in\mathcal{S}}(y_u - \widehat{\mu}_{\mathcal{S}}(\mathbf{x}_u))^2 \ + \ \left(\tfrac{N-n}{N}\right) Ave_{u\in\mathcal{T}}(y_u - \widehat{\mu}_{\mathcal{S}}(\mathbf{x}_u))^2 \\ &= \left(\tfrac{n}{N}\right) APSE(\mathcal{S}, \widehat{\mu}_{\mathcal{S}}) \ + \ \left(\tfrac{N-n}{N}\right) APSE(\mathcal{T}, \widehat{\mu}_{\mathcal{S}}) \end{aligned}$$

  where $\mathcal{T} = \mathcal{P} - \mathcal{S}$ is the complement set of the sample in the population.

- Given that interest often lies in the quality of the predictions **outside of the sample**
  - we might calculate average prediction squared error over $\mathcal{T}$

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

  - Clearly, if $n << N$ the value will not be that different from that averaged over the whole population $\mathcal{P}$.

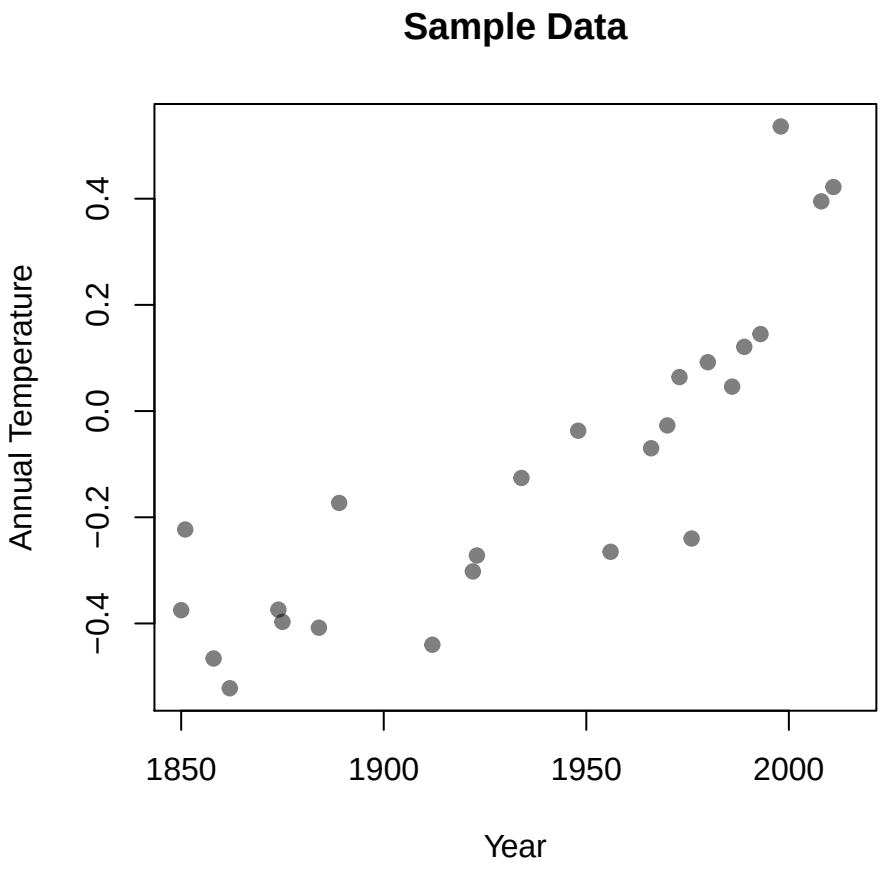
- We now need some functions:
  - The function `getXYSample` extracts a given sample of units from the population.
  - The function `getSampleComp` obtains the complement of the sample, i.e. the test set.
  - The functions `popSize` and `sampSize` calculate the population and sample sizes, respectively.

```
getXYSample <- function(xvarname, yvarname, samp, pop) {
  sampData <- pop[samp, c(xvarname, yvarname)]
  names(sampData) <- c("x", "y")
  sampData
}

getSampleComp <- function(pop, size, replace=FALSE) {
  N <- popSize(pop)
  samp <- rep(FALSE, N)
  samp[sample(1:N, size, replace = replace)] <- TRUE
  samp
}

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

Global Temperature Data

- Suppose we have the following sample of $n = 25$ observations.

```r
set.seed(341)
sample.temperature <- getSampleComp(temperature, 25)
sample.Data = getXYSample("YEAR", "ANNUAL",
                          samp=sample.temperature, pop=temperature)

plot(sample.Data,
     main= "Sample Data",
     xlab = "Year", ylab = "Annual Temperature",
     pch=19, col= adjustcolor("black", 0.5), xlim=range(temperature$YEAR))
```



Example: Global Temperature Data

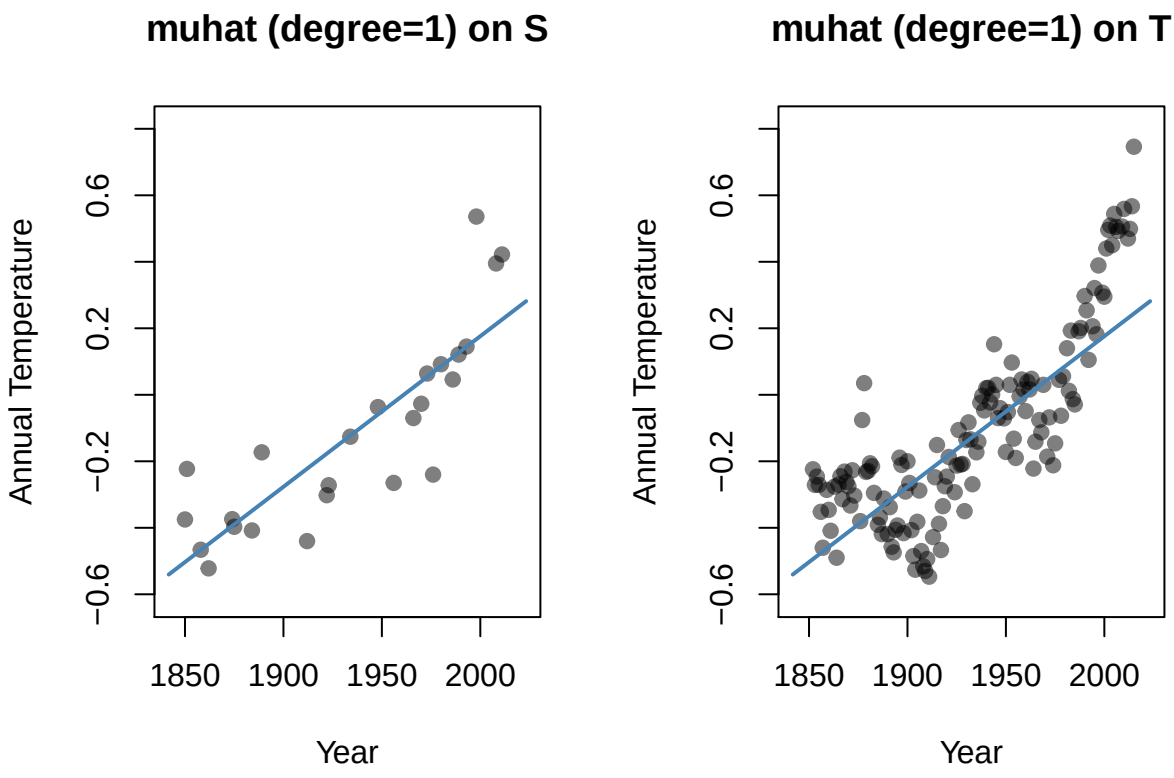
- A function to plot the fitted function on data on $\mathcal{S}$ and $\mathcal{T}$.

```r
plotTemperatureSTfit <- function(muhat=NULL, Sam=NULL, complexity=NULL) {

  if ( is.null(complexity) ) title = c("muhat on S", "muhat on T")
  else title = c(paste0("muhat (degree=", complexity,") on S"),
                 paste0("muhat (degree=", complexity,") on T"))

  xlim <- extendrange(temperature[, "YEAR"])
```

```
  ylim <- extendrange(temperature[, "ANNUAL"])

  plot(temperature[Sam,c("YEAR", "ANNUAL")],
     main= title[1],
     xlab = "Year", ylab = "Annual Temperature",
     pch=19, col= adjustcolor("black", 0.5),
     xlim=xlim, ylim=ylim)

  curve(muhat, from = xlim[1], to = xlim[2], add = TRUE,
      col="steelblue", lwd=2, n=1000)

  Tpop = !Sam

  plot(temperature[Tpop, c("YEAR", "ANNUAL")],
     main=title[2],
     xlab = "Year", ylab = "Annual Temperature",
     pch=19, col= adjustcolor("black", 0.5),
     xlim=xlim, ylim=ylim)

  curve(muhat, from = xlim[1], to = xlim[2], add = TRUE,
      col="steelblue", lwd=2, n=1000)

}
```

- Linear fit to the data (the line is fitted based on $\mathcal{S}$, and plotted on both $\mathcal{S}$ and $\mathcal{T}$).

```
muhat <- getmuhat(sample.Data, 1)

par(mfrow=c(1,2))
plotTemperatureSTfit(muhat, sample.temperature, 1)
```



- we can calculate the APSE over the sample

```
apse(y = temperature[sample.temperature,"ANNUAL"],
     x = temperature[sample.temperature,"YEAR"], predfun = muhat)
```

```
## [1] 0.02308721
```

- and the APSE over the complement set $\mathcal{T} = \mathcal{P} - \mathcal{S}$

```
Tpop = !sample.temperature
apse(y = temperature[Tpop,"ANNUAL"], x = temperature[Tpop,"YEAR"], predfun = muhat)
```

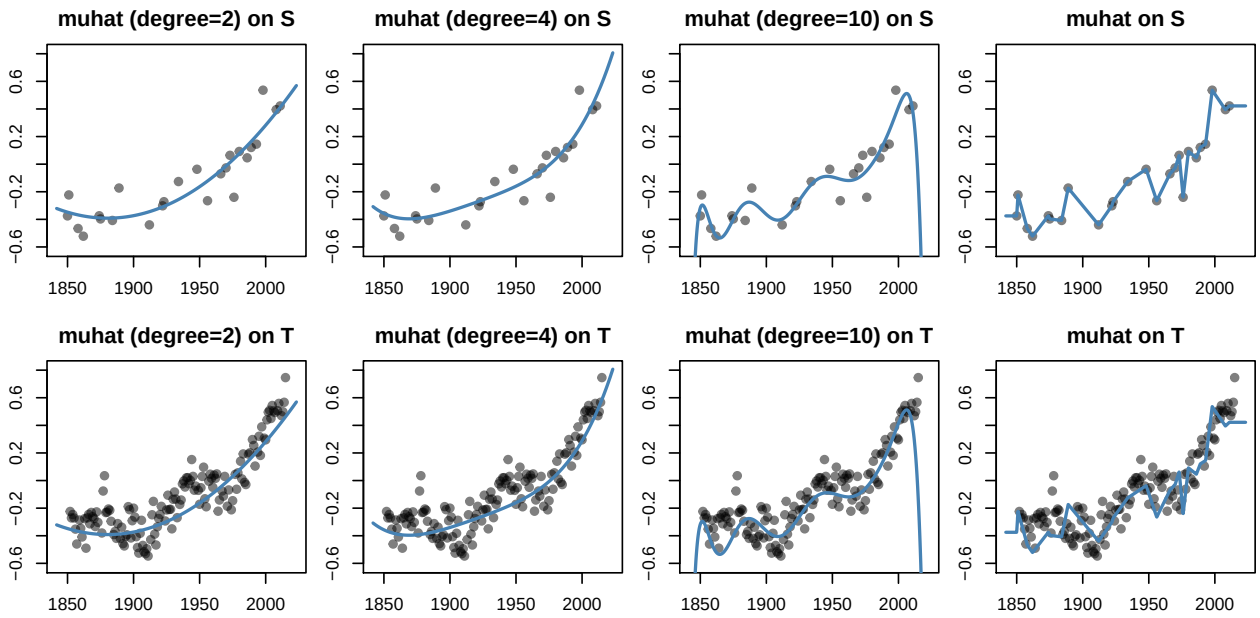
```
## [1] 0.0273872
```

- Note that the APSE over $\mathcal{S}$ is lower than APSE over $\mathcal{T}$, as expected.
  - This is because the information used to fit the model is also used to predict the values,
  - but when using $\mathcal{T}$, none of those observations were used to fit the model $y = \mu(x) + error$, hence true *prediction*.

Some polynomial fits to the sample

- Compare the fits between the sample and test data.

```
dset = c(2,4,10)
muhats    = lapply(dset, getmuhat, sampleXY=sample.Data  )
muhatFun  = getmuFun(sample.Data, "x", "y" )


par(mfcol=c(2,4), mar=2.5*c(1,1,1,0.1))
for (i in 1:length(dset)) plotTemperatureSTfit(muhats[[i]], sample.temperature, dset[i])
plotTemperatureSTfit(muhatFun, sample.temperature)
```



The APSE from each model

```
apseST <- function(y, x, sam, predfun) {
  apseS = apse(y[sam], x[sam], predfun)
  Tpop = !sam
  apseT = apse(y[Tpop], x[Tpop], predfun)

  c(apseS, apseT)
}
```

```
muhats[[4]] = muhatFun

temp.apse = sapply(muhats, apseST,
              y= temppop$y, x =temppop$x, sam=sample.temperature )

rownames(temp.apse) = c("APSE on S", "APSE on T")
colnames(temp.apse) = c(paste("deg=", dset), "piecewise linear ")
round(temp.apse,3)
```

```
##           deg= 2 deg= 4 deg= 10 piecewise linear
## APSE on S  0.013  0.012   0.008             0.000
## APSE on T  0.018  0.017   0.023             0.019
```

- Note the difference in APSEs between $\mathcal{S}$ and $\mathcal{T}$.

Extrapolation beyond the sample

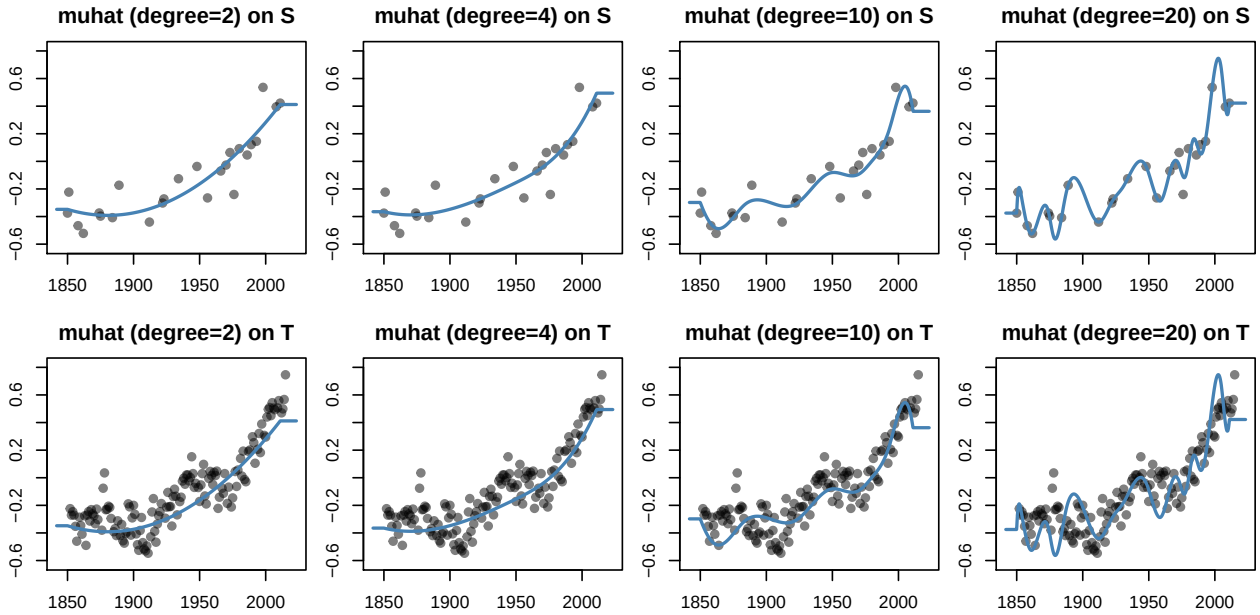
- We can modify the `getmuhat` function so that extrapolation beyond the min and max of the sample are set to constants.

```
library(splines)
getmuhat <- function(sampleXY, complexity = 1) {
  formula <- paste0("y ~ ",
                    if (complexity==0) "1"
                    else {
                      if (complexity < 3 ) {
                        paste0("poly(x, ", complexity, ", raw = FALSE)")
                        ## due to Numerical overflow
                      } else {
                        ## if complexity >= 20 use a spline.
                        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
    ##
    suppressWarnings({
      ypred = predict(fit, newdata = newdata, silent = TRUE)    })
    #val = predict(fit, newdata = newdata)
    ypred[newdata$x < range.X[1]] = val.rY[1]
    ypred[newdata$x > range.X[2]] = val.rY[2]
    ypred
  }
  ## muhat is the function that we need to calculate values
  ## at any x, so we return this function from getmuhat
  muhat
}
```

- Fit this new function on the sample and compare.

```
library(splines)
dset = c(2,4,10, 20)
muhats    = lapply(dset, getmuhat, sampleXY=sample.Data  )
muhatFun  = getmuFun(sample.Data, "x", "y" )


par(mfcol=c(2,4), mar=2.5*c(1,1,1,0.1))
for (i in 1:length(dset)) plotTemperatureSTfit(muhats[[i]], sample.temperature, dset[i])
```



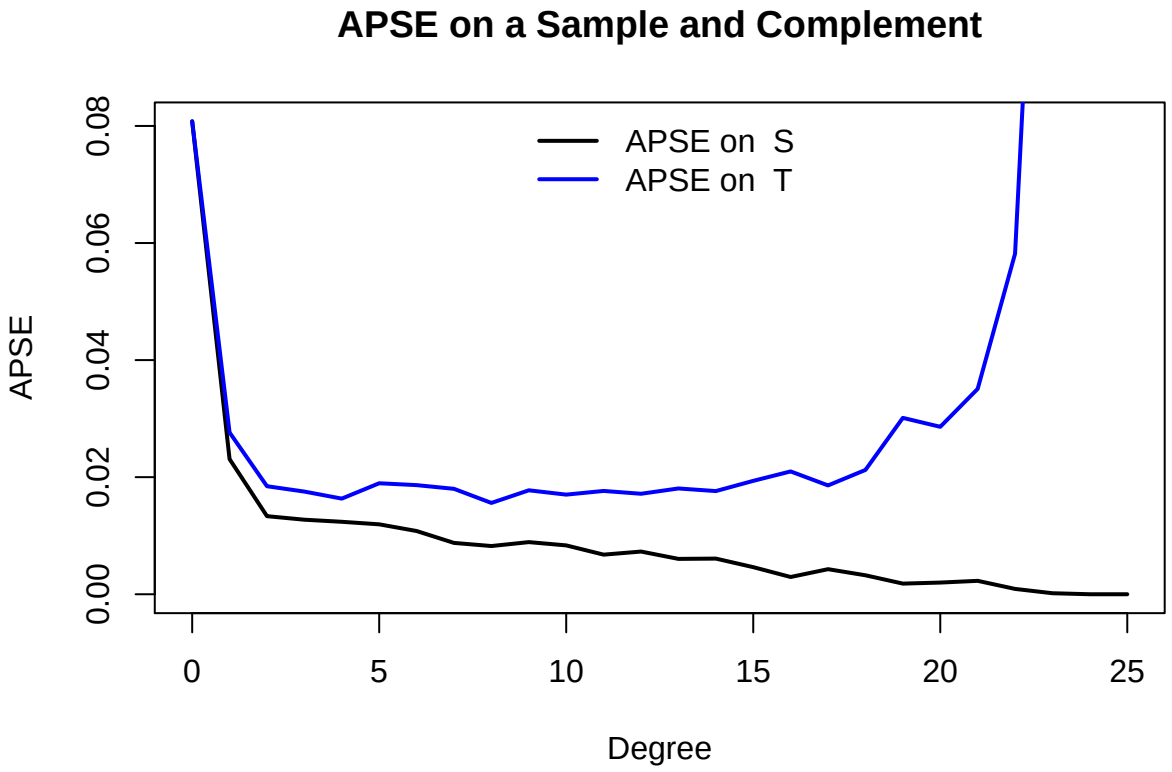

```
#plotTemperatureSTfit(muhatFun, sample.temperature)
```

- What else could we do instead of using constant values for extrapolation?

Example: Global Temperature Data

- Let us look at a larger collection of polynomials to pick an appropriate model.
- Comparing the APSE on the sample and complement set we have:

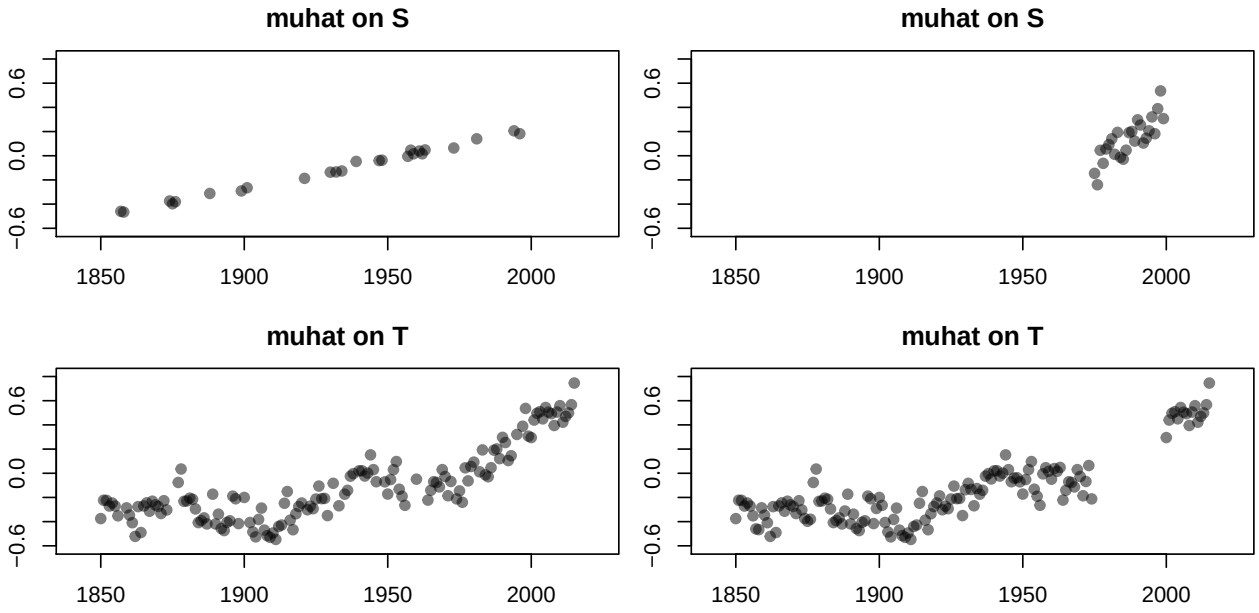
```
dset   = 0:25
muhats = lapply(dset, getmuhat, sampleXY=sample.Data  )

temp.apse = sapply(muhats, apseST,
              y= temppop$y, x =temppop$x, sam=sample.temperature )

rownames(temp.apse) = c("APSE on S", "APSE on T")
colnames(temp.apse) = paste("deg=", dset)


plot( dset, temp.apse[1,], type='l', col=1, lwd=2,
      xlab="Degree", ylab="APSE",
      main="APSE on a Sample and Complement" )
lines(dset, temp.apse[2,], type='l', col=4, lwd=2 )
legend( "top", legend= paste("APSE on ", c("S","T")),
        col=c(1,4), lwd=c(2,2), bty='n')
```



- At around degree equal to 4, increasing complexity yields no decrease in prediction accuracy.
  - Increasing complexity will continue to improve predictions on the sample but not on the rest of the population.
  - This effect is called **overfitting** in that the predictor has been too closely tailored to the peculiarities of the sample.
  - **overfitting:** increasing the complexity of the model to the level that the prediction power is compromised.

Just one sample?

- Of course, the quality of the predictor depends on the quality of the sample, i.e. if the sample is a good/fair representation of the population.
- 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 errors might be very different for another sample.
  - It is important then to choose a predictor function that performs well no matter which sample was used to construct the predictor.
    - though it is assumed that the sample is a representative of the population.

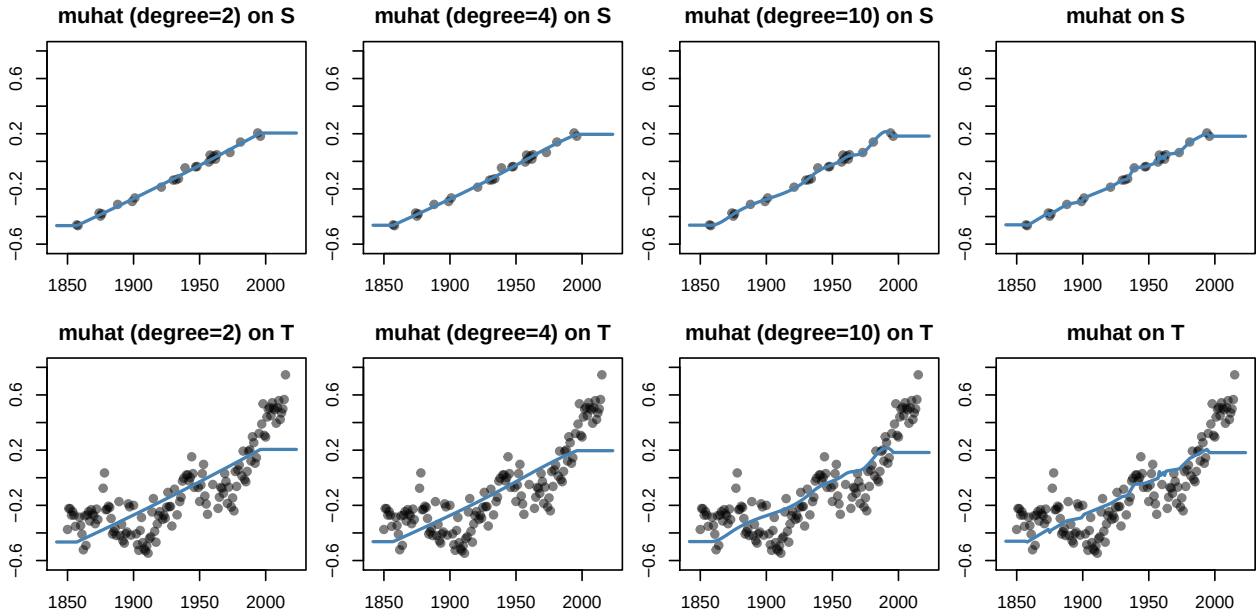


- What do you observe?
- Are both samples *good* representative of the population?

Example: Global Temperature Data (sample 1)

- Some polynomials fit to these samples.

```
sam1.Data = getXYSample("YEAR", "ANNUAL", samp=sam1, pop=temperature)

dset = c(2,4,10)
muhats    = lapply(dset, getmuhat, sampleXY=sam1.Data  )
muhatFun  = getmuFun(sam1.Data, "x", "y" )


par(mfcol=c(2,4), mar=2.5*c(1,1,1,0.1))
for (i in 1:length(dset)) plotTemperatureSTfit(muhats[[i]], sam1, dset[i])
plotTemperatureSTfit(muhatFun, sam1)
```

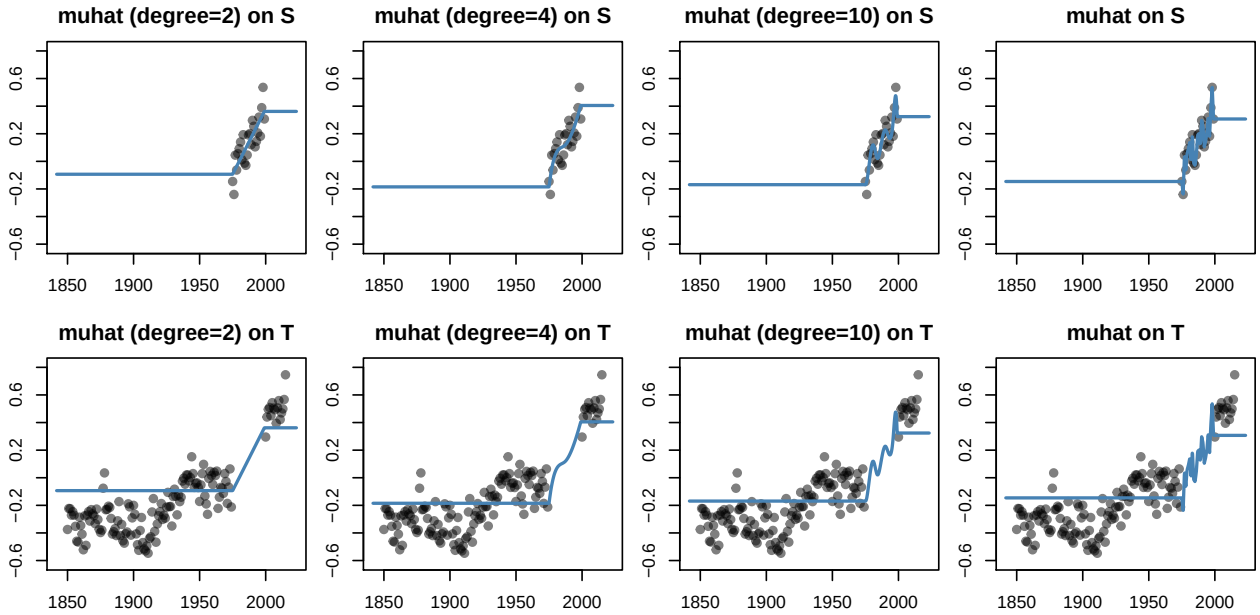


- Which predictor performs better on the test set?
  - Would that be what you would have chosen had you not seen the test set?

Example: Global Temperature Data (sample 2)

- Some polynomials fit to the sample.

```
sam2.Data = getXYSample("YEAR", "ANNUAL", samp=sam2, pop=temperature)

dset = c(2,4,10)
muhats    = lapply(dset, getmuhat, sampleXY=sam2.Data  )
muhatFun  = getmuFun(sam2.Data, "x", "y" )


par(mfcol=c(2,4), mar=2.5*c(1,1,1,0.1))
for (i in 1:length(dset)) plotTemperatureSTfit(muhats[[i]], sam2, dset[i])
plotTemperatureSTfit(muhatFun, sam2)
```



- Which predictor performs better on the test set?
  - Would that be what you would have chosen had you not seen the test set?