Generalized Additive Models (GAMs) with R

Day 3: What are GAMs?

An Online Course
Presented by Geoffrey S. Hubona

Day 3 Agenda (1 of 2)

- What are GAMs?
- We Begin with Simple Univariate Additive Model.
- Representing Smooth Functions: Regression Splines
  - Polynomial Basis
- Representing Smooth Functions: Cubic Splines
  - Knot Locations
  - Cubic Spline Basis
  - **Example in R: Engine Wear as a Function of Capacity**
  - **Example Functions in R: Cubic Spline; Model Matrix; Select Knots and Fit Model**
- Controlling the Degree of Smoothing
  - Penalize "Wiggliness"
  - **Example Functions in R: Penalized Regression Spline Penalty Matrix; Fitting Penalized Regression Spline**

Day 3 Agenda (2 of 2)

- Cross Validation
  - Choosing a Smoothing Parameter
  - **Example Scripts in R: Search for GCV-optimal Smoothing Parameter; Plot Optimal Engine Wear Model**
- Two Term Additive Models
  - **Example Functions in R: Penalized Regression Spline Two Term Additive Model; Fit Model and Calculate Optimal GCV Score**
  - **New Example in R: Estimate 2-term Additive Model for 31 Felled Cherry Trees.**
- Generalized Additive Models (GAMs)
  - **Revised Example in R: Generalized Additive Model for 31 Felled Cherry Trees.**

What Are GAMs?

Simon N. Wood

Parametric Only: Row (x_i); Vector (β)

Smooth Functions

Isotropic Smooth

- A **Generalized Additive Model** (GAM) is a generalized linear model with a linear predictor involving a sum of smooth functions of covariates.
- General model structure of a GAM:

$$g(\mu_i) = X_i^* \Theta + f_1(x_{1i}) + f_2(x_{2i}) + f_3(x_{3i}, x_{4i}) + \dots$$

where $\mu_i \equiv E(Y_i)$, $Y_i \sim$ some exponential family distribution. Y_i is a response variable, X_i^* is a row of the model matrix for any strictly parametric model components, Θ is the corresponding parameter vector, and the f_j are smooth functions of the covariates, x_k.

Representing a Univariate Smooth Function using Regression Splines

- Here is an additive model containing one smooth function of one covariate:

$$y_i = f\left(x_i\right) + \varepsilon_i, \qquad (1)$$

  where y_i is a response variable, x_i a covariate, f is a smooth function and the ε_i are i.i.d. $N\left(0, \sigma^2\right)$ random variables. Also, we will assume that the x_i lie in the interval [0,1].
- **What is the function f ? How is f represented?**
- If we represent the smooth function f in a way that equation (1) becomes a linear model, we can estimate f .
- We choose a ***basis*** which defines a (limiting) space of functions ('***basis functions***') which can be treated as if completely known (even though f is unknown).

Representing a Univariate Smooth Function using Regression Splines

- Here is the original model containing one smooth function of one covariate: $y_i = f(x_i) + \varepsilon_i,$ (1)

Repeat equation (1)
From previous slide

- Have chosen some basis functions, let's say for example, a 4th order polynomial, which can be treated as completely known: if $b_j(x)$ is the j^{th} such basis function, then f is assumed to have this representation:

$$f(x) = \sum_{j=1}^{q} b_j(x)\beta_j \quad (2)$$

- For some unknown values of the parameters, β_j .

Representing a Univariate Smooth Function using Regression Splines

- Here is the original model containing one smooth function of one covariate: $y_i = f(x_i) + \varepsilon_i,$ (1)
- Function f now is: $f(x) = \sum_{j=1}^{q} b_j(x)\beta_j$ (2)

Repeat equations (1) and (2) from previous slide.

- If we substitute (2) into (1) we have $y_i = \sum_{j=1}^{q} b_j(x_i)\beta_j + \varepsilon_i$ which is a linear model.
- ***What does it look like?***
- If we believe f to be a 4th order polynomial, so that the space of polynomials of order 4 and below contains f, then a **basis** for this space is:

$$b_1(x) = 1, b_2(x) = x, b_3(x) = x^2, b_4(x) = x^3, b_5(x) = x^4$$

Representing a Univariate Smooth Function using Regression Splines

- So this is the ***basis*** of the space of polynomials of order 4 and below which contain f

$$b_1(x) = 1, b_2(x) = x, b_3(x) = x^2, b_4(x) = x^3, b_5(x) = x^4$$

- Then the original univariate additive model (equation (1)) becomes the simple linear model which we can estimate:

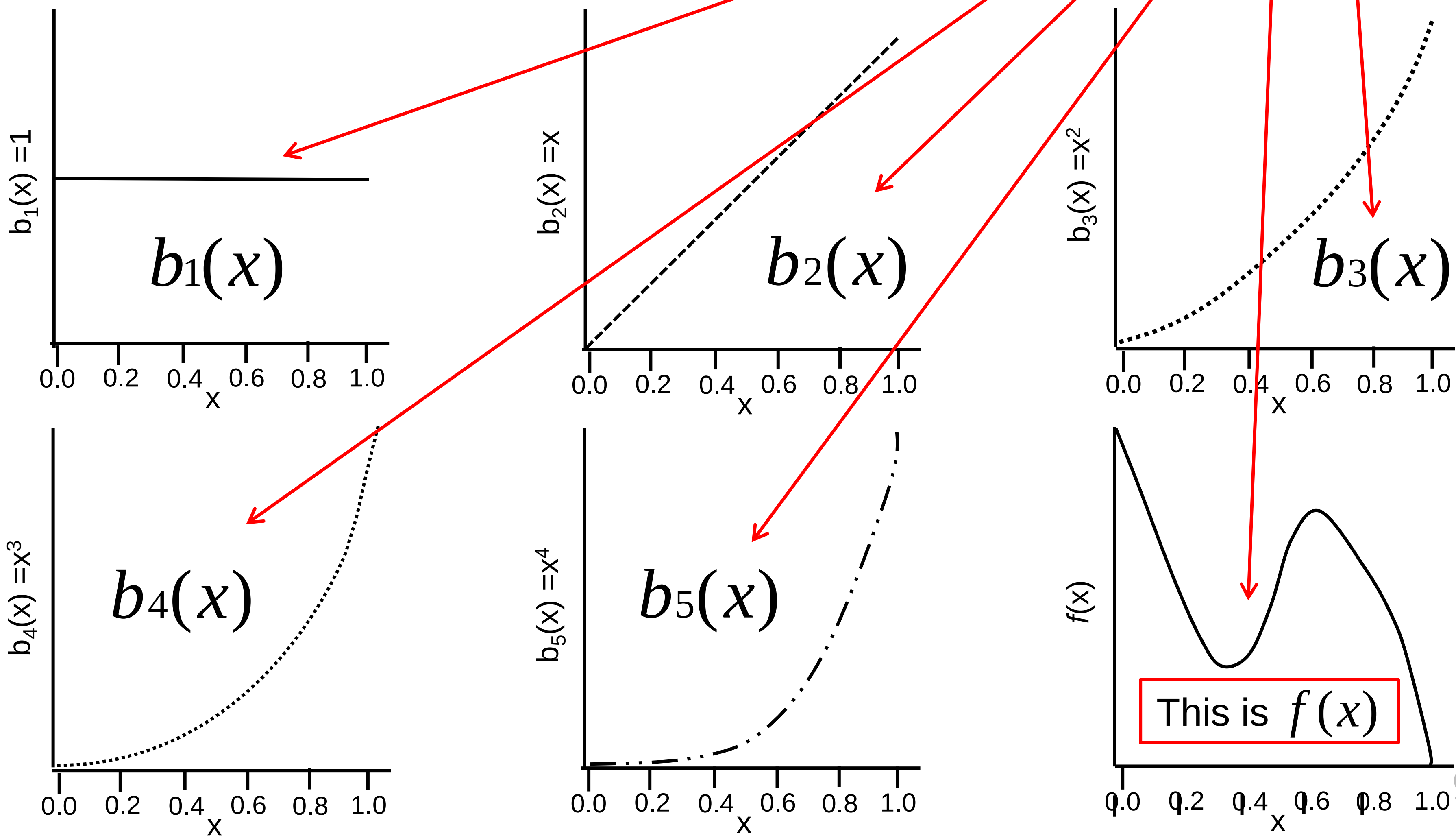
$$y_i = \beta_1 + x_i\beta_2 + x_i^2\beta_3 + x_i^3\beta_4 + x_i^4\beta_5 + \varepsilon_i$$

- So for this simple univariate additive (not generalized) model, we need to represent the red circled term in terms of a ***model matrix***:

$$y_i = \sum_{j=1}^{q} b_j(x_i)\beta_j + \varepsilon_i$$

Illustration of f as 4th Order Polynomial

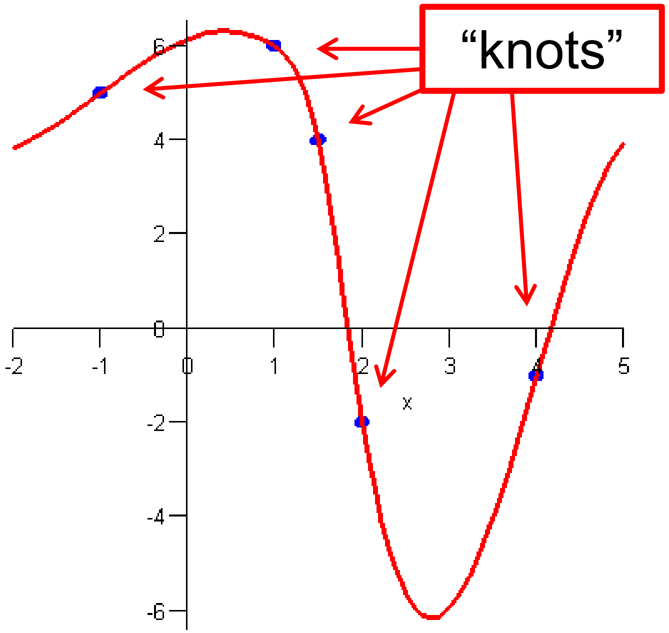
Multiply each basis function by real valued parameter, β_j and then sum to give final curve.



Note we have scaled x to [0,1].

Cubic Splines

- One can represent a univariate function (eq (1) on slides #4-6) with a ***cubic spline***:
  - Is a curve made up of sections of a cubic polynomial joined together so they are continuous in value.



Conventional cubic spline: knots occur where have data. **Regression splines**: must choose knot locations.

We denote **knot locations** by:

$$\{x_i^* : i = 1, \ldots, q-2\}$$

Question: Given knot locations, how do we establish the basis?

Knot Locations

- Given knot locations, there are alternative, but equivalent, ways of writing down a basis for cubic splines.
  - Knot locations are denoted by $\{x_i^* : i = 1, \cdots, q-2\}$
  - A simple, general basis function to use (Gu 2002) is:
    $b_1(x) = 1, b_2(x) = x$ and $b_{i+2} = R(x, x_i^*)$ for $i = 1 \ldots q-2$ where

$$\boxed{\begin{aligned} R(x,z) = &\left(\left(z - 1/2\right)^2 - 1/12\right)\left(\left(x - 1/2\right)^2 - 1/12\right)/4 \\ &- \left(\left(|x - z| - 1/2\right)^4 - 1/2\left(|x - z| - 1/2\right)^2 + 7/240\right)/24 \end{aligned}}$$

- Using this cubic spline basis for *f* means that eq (1) becomes a linear model and so the model can be estimated by least squares.
  - We illustrate a rank 5 example of this basis two slides back.

Cubic Spline Basis

- Using this cubic spline basis for f means that eq (1) becomes a linear model: $y = X\beta + \varepsilon$
  - Where the i^{th} row of the model matrix is:

$$X_i = [1, x_i, R(x_i, x_1^*), R(x_i, x_2^*), \ldots, R(x_i, x_{q-2}^*)]$$

- Steps: Cubic Spline Basis to Estimate a Model and Predict:
  1) Read in data, scale x axis [0,1], plot data for spline fitting.
  2) Write a basis function (**`rk()`**)
  3) Write a function (**`spl.X()`**) which will accept an array of x values and a sequence of knots to produce a model matrix for the spline.
  4) Select a set of knots.
  5) Fit the model.
  6) Generate x values for prediction.
  7) Create prediction matrix.
  8) Plot the fitted spline.

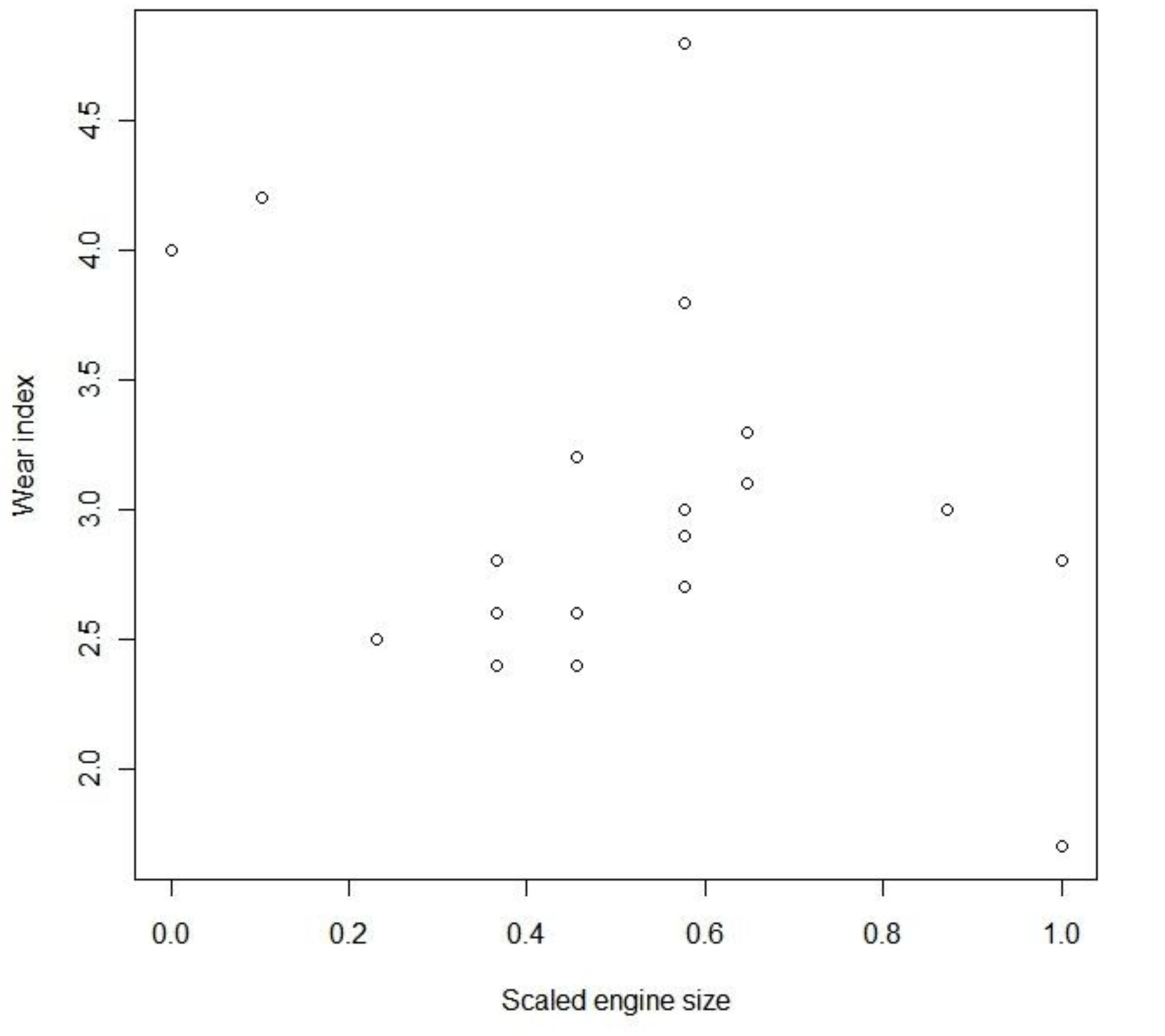
Example of Cubic Spline Basis in R

- People who do not know cars believe that a car engine with a larger cylinder capacity wears out less quickly than a car with a small engine capacity.
- We read the data into R and scale the engine capacity data to lie in [0, 1] interval:

```
#### Using the cubic spline basis
size <- c(1.42,1.58,1.78,1.99,1.99,1.99,2.13,2.13,2.13,
      2.32,2.32,2.32,2.32,2.32,2.43,2.43,2.78,2.98,2.98)
wear <- c(4.0,4.2,2.5,2.6,2.8,2.4,3.2,2.4,2.6,4.8,2.9,
          3.8,3.0,2.7,3.1,3.3,3.0,2.8,1.7)
x <- size-min(size); x <- x/max(x)
plot(x,wear,xlab="Scaled engine size",ylab="Wear index")
```

This line scales the engine data

Plot of Wear Index by Scaled Engine Size



Function Defining Basis $R(x,z)$

- So now we write a function **`rk()`** defining the cubic spline $R(x,z)$ on the interval [0,1].

```
rk<-function(x,z) # R(x,z) for cubic spline on [0,1]
{ ((z-0.5)^2-1/12)*((x-0.5)^2-1/12)/4-
  ((abs(x-z)-0.5)^4-(abs(x-z)-0.5)^2/2+7/240)/24
}
```

From Gu (2002):

$$R(x,z) = \left(\left(z - 1/2\right)^2 - 1/12\right)\left(\left(x - 1/2\right)^2 - 1/12\right)/4$$
$$- \left(\left(|x - z| - 1/2\right)^4 - 1/2\left(|x - z| - 1/2\right)^2 + 7/240\right)/24$$

Function Defining Basis $R(x,z)$

- For this basis (a rank 6 basis):
- $b_1(x)=1, b_2(x)=x$ and $b_{i+2}=R(x,x_i^*)$ for $i=1,\ldots,q-2$
- Where $R(x,z)=\left((z-1/2)^2-1/12\right)\left((x-1/2)^2-1/12\right)/4$

$$-\left(\left(|x-z|-1/2\right)^4-1/2\left(|x-z|-1/2\right)^2+7/240\right)/24$$

- Using this cubic spline basis for f means that the univariate smooth function $y_i=f(x_{1i})+\varepsilon_i$, becomes a linear model $y=X\beta+\varepsilon$, where X is the model matrix.
- The i^{th} row of the model matrix is

$$X_i=\left[1, x_i, R(x_i,x_1^*), R(x_i,x_2^*), \ldots, R(x_i,x_{q-2}^*)\right]$$

Function to Produce the Model Matrix for Spline

- We will use **`rk()`** function in the **`spl.X()`** function below to take a sequence of knots and an array of X values to produce a model matrix for the spline:

```
spl.X<-function(x,xk)
# set up model matrix for cubic penalized regression
# spline
{ q<-length(xk)+2   # number of parameters
  n<-length(x)      # number of data
  X<-matrix(1,n,q) # initialized model matrix
  X[,2]<-x          # set second column to x
  X[,3:q]<-outer(x,xk,FUN=rk) # and remaining to R(x,xk)
  X
}
```

This is the 'model matrix'

Select Knots and Fit Model

- Select a set of knots, x_i^*, and the model can be fitted.
- We use rank 6 basis, so that $q = 6$ and there are 4 knots evenly spread over [0,1]:

```
xk<-1:4/5        # choose some knots
X<-spl.X(x,xk) # generate model matrix
mod.1<-lm(wear~X-1) # fit model
xp<-0:100/100    # x values for prediction
Xp<-spl.X(xp,xk) # prediction matrix
lines(xp,Xp%*%coef(mod.1)) # plot fitted spline
```

- The model fit (next slide) seems plausible, but the degree of model smoothness, controlled with the basis dimension (i.e. q, the number of knots + 2), was essentially arbitrary.

Model Matrix X Looks Like

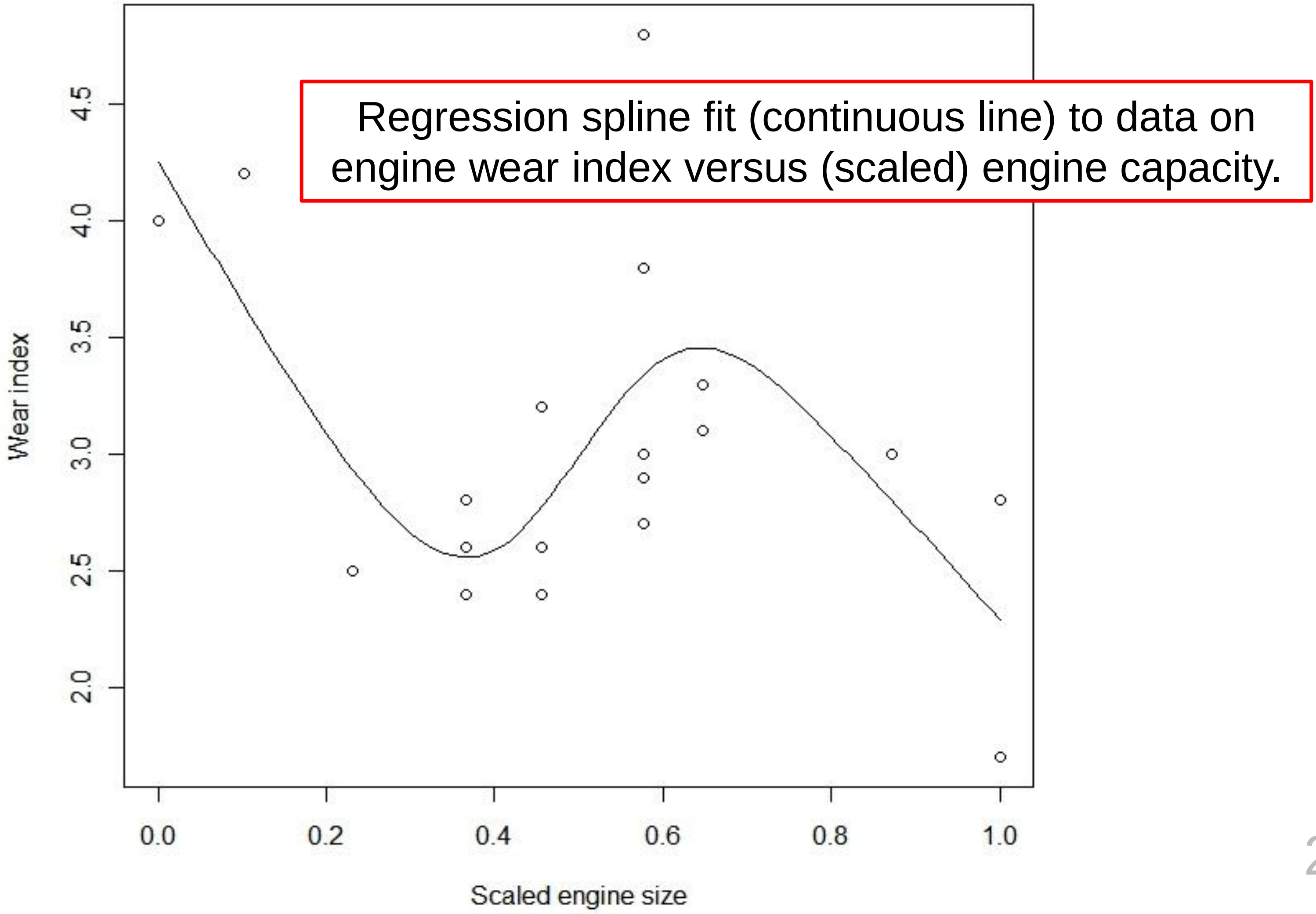
Is a rank 6 cubic spline

```
> X
      [,1]      [,2]         [,3]          [,4]          [,5]          [,6]
 [1,]    1 0.0000000  0.0006000000 -0.0040666667 -0.0040666667  6.000000e-04
 [2,]    1 0.1025641  0.0011910163 -0.0017986610 -0.0025832101 -3.421181e-04
 [3,]    1 0.2307692  0.0013337512  0.0007641866 -0.0006723177 -1.134463e-03
 [4,]    1 0.3653846  0.0004863279  0.0025379134  0.0012408700 -1.235660e-03
 [5,]    1 0.3653846  0.0004863279  0.0025379134  0.0012408700 -1.235660e-03
 [6,]    1 0.3653846  0.0004863279  0.0025379134  0.0012408700 -1.235660e-03
 [7,]    1 0.4551282 -0.0002514095  0.0027666999  0.0022402848 -8.735914e-04
 [8,]    1 0.4551282 -0.0002514095  0.0027666999  0.0022402848 -8.735914e-04
 [9,]    1 0.4551282 -0.0002514095  0.0027666999  0.0022402848 -8.735914e-04
[10,]    1 0.5769231 -0.0010382837  0.0019246203  0.0027870084  8.294965e-06
[11,]    1 0.5769231 -0.0010382837  0.0019246203  0.0027870084  8.294965e-06
[12,]    1 0.5769231 -0.0010382837  0.0019246203  0.0027870084  8.294965e-06
[13,]    1 0.5769231 -0.0010382837  0.0019246203  0.0027870084  8.294965e-06
[14,]    1 0.5769231 -0.0010382837  0.0019246203  0.0027870084  8.294965e-06
[15,]    1 0.6474359 -0.0012606937  0.0010733705  0.0024330758  5.897512e-04
[16,]    1 0.6474359 -0.0012606937  0.0010733705  0.0024330758  5.897512e-04
[17,]    1 0.8717949 -0.0005452104 -0.0022051957 -0.0012497963  1.295347e-03
[18,]    1 1.0000000  0.0006000000 -0.0040666667 -0.0040666667  6.000000e-04
[19,]    1 1.0000000  0.0006000000 -0.0040666667 -0.0040666667  6.000000e-04
```

Prediction Model Matrix Xp

```
> Xp
      [,1] [,2]          [,3]          [,4]          [,5]          [,6]
 [1,]    1 0.00  6.000000e-04 -4.066667e-03 -4.066667e-03  6.000000e-04
 [2,]    1 0.01  6.632829e-04 -3.843350e-03 -3.923317e-03  5.033829e-04
 [3,]    1 0.02  7.262600e-04 -3.620140e-03 -3.779873e-03  4.070600e-04
 [4,]    1 0.03  7.886162e-04 -3.397150e-03 -3.636250e-03  3.113163e-04
 [5,]    1 0.04  8.500267e-04 -3.174507e-03 -3.492373e-03  2.164267e-04
 [6,]    1 0.05  9.101562e-04 -2.952344e-03 -3.348177e-03  1.226563e-04
 [7,]    1 0.06  9.686600e-04 -2.730807e-03 -3.203607e-03  3.026000e-05
 [8,]    1 0.07  1.025183e-03 -2.510050e-03 -3.058617e-03 -6.051708e-05
 [9,]    1 0.08  1.079360e-03 -2.290240e-03 -2.913173e-03 -1.494400e-04
[10,]    1 0.09  1.130816e-03 -2.071550e-03 -2.767250e-03 -2.362837e-04
[11,]    1 0.10  1.179167e-03 -1.854167e-03 -2.620833e-03 -3.208333e-04
[12,]    1 0.11  1.224016e-03 -1.638284e-03 -2.473917e-03 -4.028837e-04
[13,]    1 0.12  1.264960e-03 -1.424107e-03 -2.326507e-03 -4.822400e-04
[14,]    1 0.13  1.301583e-03 -1.211850e-03 -2.178617e-03 -5.587171e-04
[15,]    1 0.14  1.333460e-03 -1.001740e-03 -2.030273e-03 -6.321400e-04
[16,]    1 0.15  1.360156e-03 -7.940104e-04 -1.881510e-03 -7.023437e-04
[17,]    1 0.16  1.381227e-03 -5.889067e-04 -1.732373e-03 -7.691733e-04
[18,]    1 0.17  1.396216e-03 -3.866837e-04 -1.582917e-03 -8.324837e-04
[19,]    1 0.18  1.404660e-03 -1.876067e-04 -1.433207e-03 -8.921400e-04
[20,]    1 0.19  1.406083e-03  8.049583e-06 -1.283317e-03 -9.480171e-04
[21,]    1 0.20  1.400000e-03  2.000000e-04 -1.133333e-03 -1.000000e-03
[22,]    1 0.21  1.386083e-03  3.879496e-04 -9.833504e-04 -1.047984e-03
[23,]    1 0.22  1.364660e-03  5.715933e-04 -8.334733e-04 -1.091873e-03
[24,]    1 0.23  1.336216e-03  7.506162e-04 -6.838171e-04 -1.131584e-03
```

Regression Spline Fit



Controlling the Degree of Smoothing

- So how do we ***control the degree of smoothing*** ?
- One obvious choice would be manipulate q via the number of knots (then backfit q and test fit hypotheses)
- Another: keep the basis dimension fixed, at a size slightly larger than believed necessary, but add a “wiggliness” penalty.
    - Instead of fitting the model by minimizing the sum of the least squares, minimize the sum of the least squares PLUS $\lambda\int_0^1 [f''(x)]^2 dx$, (**lambda** times **integrated square of second derivative**).
- ***Smoothing parameter*** λ controls trade off between model fit and model smoothness.
    - High λ leads to a straight-line estimate
    - $\lambda = 0$ results in an un-penalized regression spline estimate.

Penalized Regression Spline Penalty Matrix

- Need to derive the square root of the penalized regression spline matrix **S**:

```
spl.S<-function(xk)
# set up the penalized regression spline penalty matrix,
# given knot sequence xk
{ q<-length(xk)+2;S<-matrix(0,q,q) # initialize matrix to 0
  S[3:q,3:q]<-outer(xk,xk,FUN=rk)  # fill in non-zero part
  S
}

mat.sqrt<-function(S) # A simple matrix square root
{ d<-eigen(S,symmetric=TRUE)
  rS<-d$vectors%*%diag(d$values^0.5)%*%t(d$vectors)
}
```

Function for Fitting Penalized Regression Spline

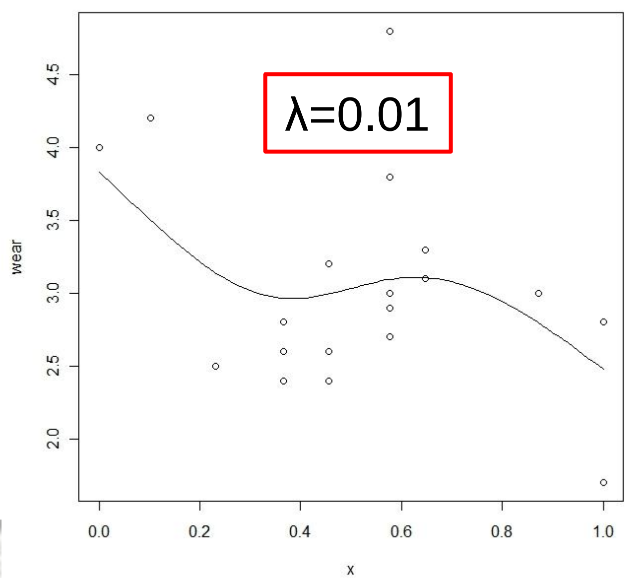
- To use this function, we need to choose the basis dimension q , the knot locations, x_j^*, and a value for the smoothing parameter, λ:

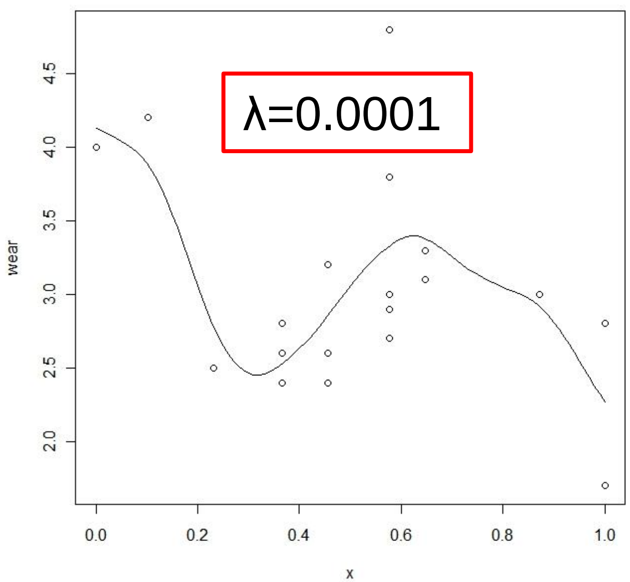
```
prs.fit<-function(y,x,xk,lambda)
# function to fit penalized regression spline to x,y data,
# with knots xk, given smoothing parameter, lambda.
 { q<-length(xk)+2                  # dimension of basis
   n<-length(x)                     # number of data
   # create augmented model matrix ....
   Xa <- rbind(spl.X(x,xk),mat.sqrt(spl.S(xk))*sqrt(lambda))
   y[(n+1):(n+q)]<-0  # augment the data vector
   lm(y~Xa-1) # fit and return penalized regression spline
 }
```

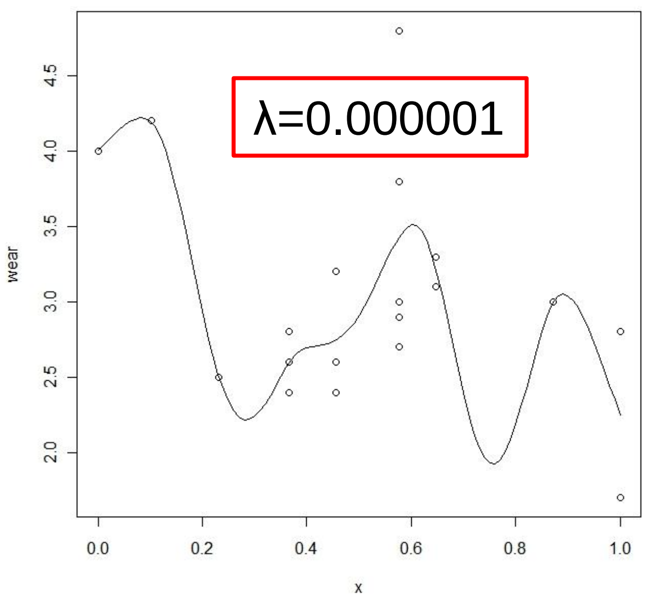
Smoothing Parameter Controls the Smooth

- In this example, $q = 9$ and the knots are evenly spread over [0,1]. The smoothing parameter, $\lambda = 10^{-4}$, really controls the behavior of the fitted model:

```
xk<-1:7/8  # choose some knots
mod.2<-prs.fit(wear,x,xk,0.0001) # fit pen. reg. spline
Xp<-spl.X(xp,xk) # matrix to map params to fitted values at xp
plot(x,wear);lines(xp,Xp%*%coef(mod.2)) # plot data & spl. fit
```







Choosing a Smoothing Parameter λ : Cross Validation

- If λ is too high, the data is over smoothed, if too low, the data is undersmoothed.
- In both cases, the spline estimate $\hat{f}$ will not be close to the true function f: This is our goal.
- Imagine the criterion ***choose λ to minimize***:

$$M = \frac{1}{n}\sum_{i=1}^{n}\left(\hat{fi} - fi\right)^2$$

- Since f is unknown, M cannot be used directly but you can derive an estimate of $\mathrm{E}\left(M\right) + \sigma^2$ which is the expected square error in predicting a new variable.

Choosing a Smoothing Parameter λ: Cross Validation

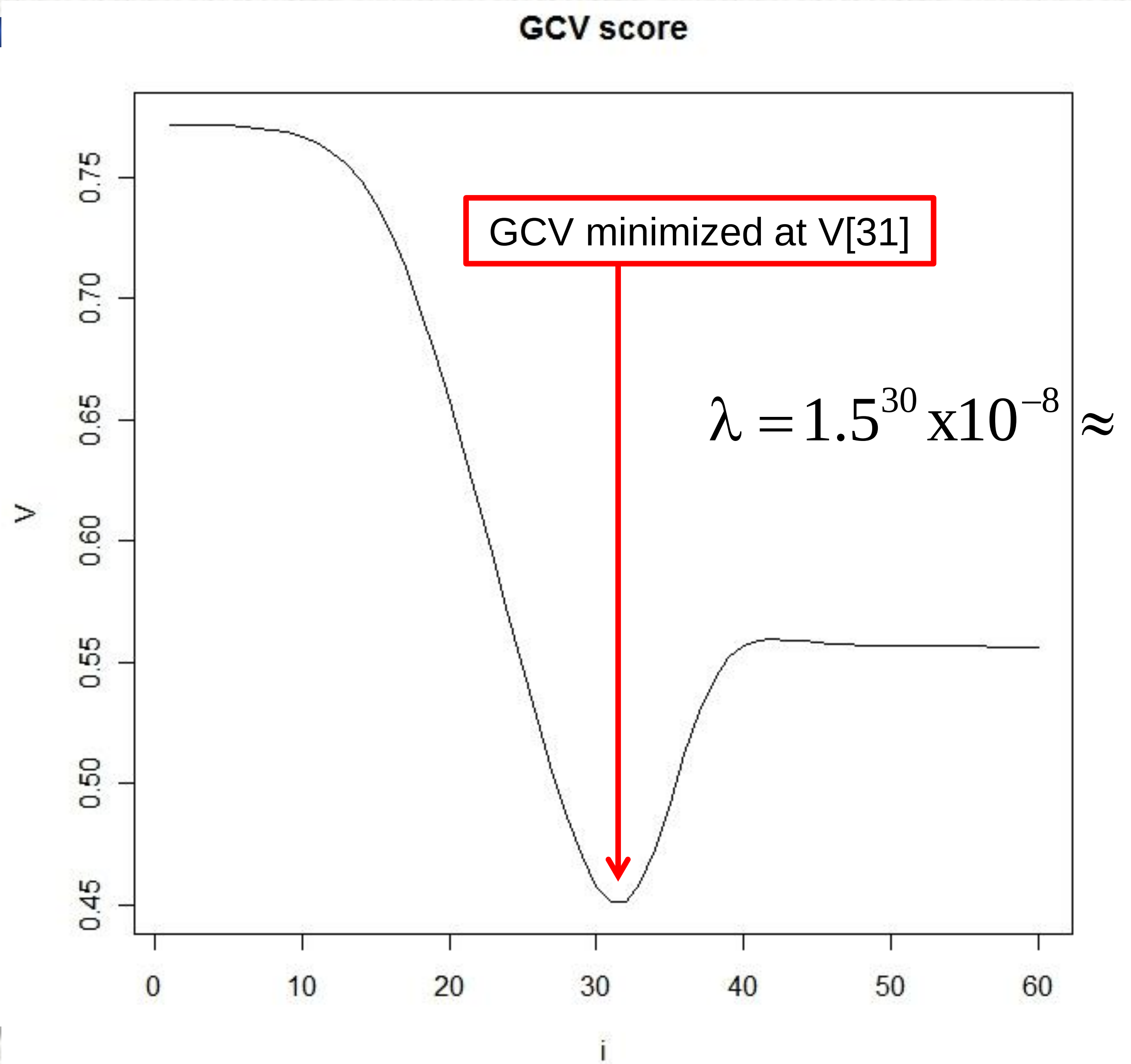
- Calculate ***ordinary cross validation (OCV) score***: leave out each datum, fit the model, and calculate squared difference between missing datum and predicted value.
- We derive a proxy for M, V_0, and choose λ to minimize V_0.
- Leaving out one datum at a time and fitting the model to the remaining data points is ***computationally inefficient***.
- It can be shown that you can use ***mean weights*** (instead of individual weights) to derive a similarly-purposed ***generalized cross validation*** (**GCV**) ***score***.
    - **GCV** has ***computational advantages*** over ***OCV***.
    - **GCV** also has advantages in terms of ***invariance***.

R Code to Search for GCV Optimal Smoothing Parameter

- **Engine Wear**: Simple direct search for GCV optimal smoothing parameter:

```
lambda <- 1e-8; n <- length(wear); V <- rep(0,60)
for (i in 1:60)            # loop through smoothing parameters
{ mod<-prs.fit(wear,x,xk,lambda)     # fit model, given lambda
  trA<-sum(influence(mod)$hat[1:n])  # find tr(A)
  rss<-sum((wear-fitted(mod)[1:n])^2)# residual sum of squares
  V[i]<-n*rss/(n-trA)^2              # obtain GCV score
  lambda<-lambda*1.5                 # increase lambda
}
plot(1:60,V,type="l",main="GCV score",xlab="i")  # plot score
```

GCV Function for Engine Wear



Lambda optimal to minimize GCV

$$\lambda = 1.5^{30} \mathrm{x} 10^{-8} \approx 0.002$$

X-axis reflects smoothing parameters on a log scale such that:

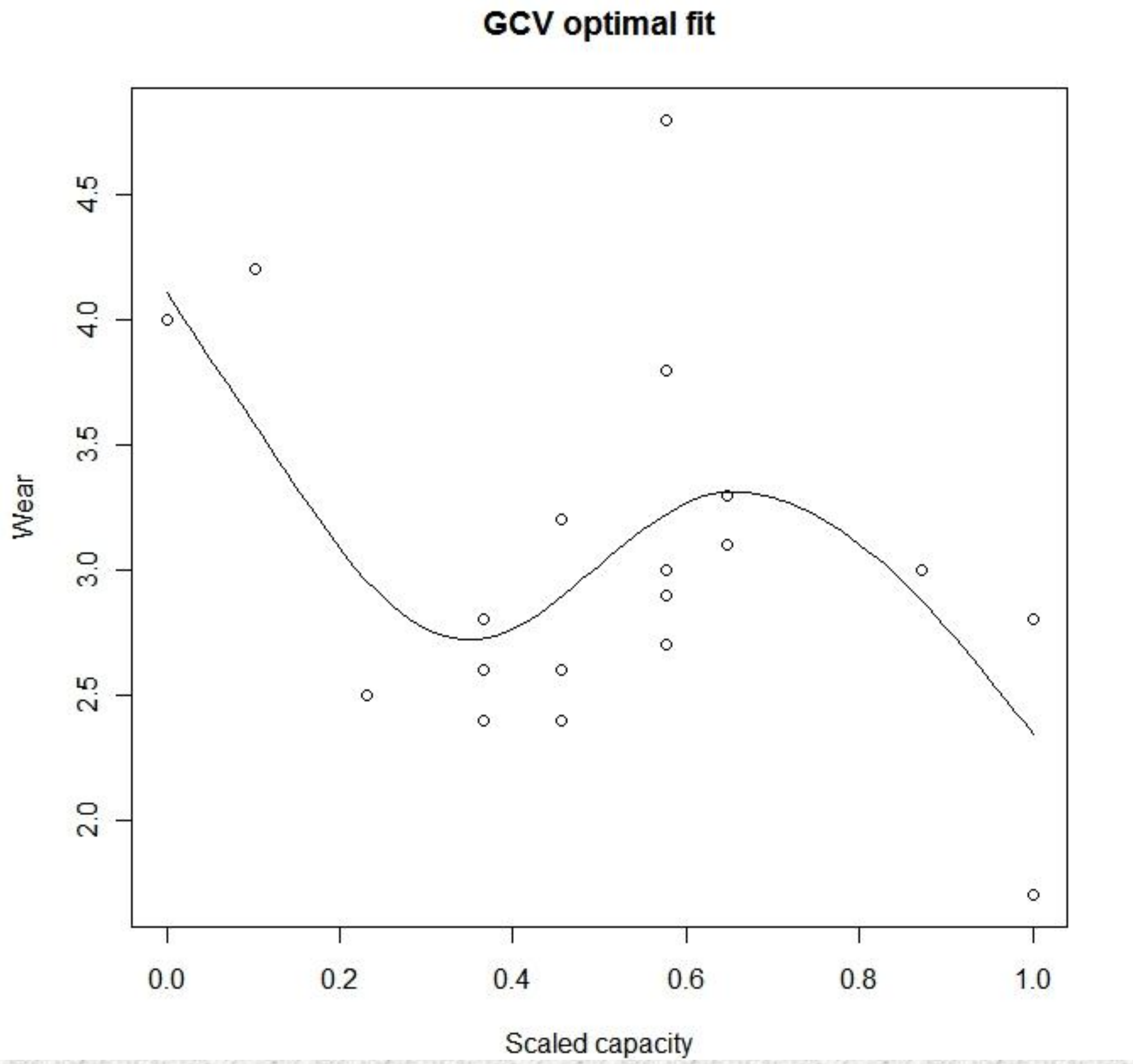
$$\lambda = 1.5^{i-1} \mathrm{x} 10^{-8}$$

R Code to Plot Optimal Model

- **Engine Wear**: Plot fitted model where GCV is minimized:

```
i<-(1:60)[V==min(V)]                          # extract index of min(V)
mod.3<-prs.fit(wear,x,xk,1.5^(i-1)*1e-8)   # fit optimal model
Xp<-spl.X(xp,xk)                              # .... and plot it
plot(x,wear);lines(xp,Xp%*%coef(mod.3))
```

Engine Wear: Plot GCV-optimal fit λ = 0.002



Additive Models: Two Explanatory Variables

Simon N. Wood

- Suppose there are two explanatory variables, x and z, for a response variable, y, and so we use this simple additive model structure:

$$y_i = f_1(x_i) + f_2(z_i) + \varepsilon_i$$

  Where f_1 and f_2 are smooth functions, and the ε_i are i.i.d. $N(0, \sigma^2)$ random normal variables and z_i and x_i lie in [0,1].
- First, the model above is a special restrictive case of $f(x, z)$
- The fact that the model contains more than one function introduces an identifiability problem:
  - For example, could simultaneously add any constant to f_1 and subtract it from f_2 without changing the model predictions.
  - Hence must impose identifiability constraints on the model before fitting.

Additive Models: Two Explanatory Variables

- Represent Each Smooth Function with a Penalized Regression Spline Basis:

$$f_1(x) = \delta_1 + x\delta_2 + \sum_{j=1}^{q_1-2} R\left(x, x_j^*\right)\delta_{j+2}$$

$$f_2(z) = \gamma_1 + z\gamma_2 + \sum_{j=1}^{q_2-2} R\left(z, z_j^*\right)\gamma_{j+2}$$

- Where δ_j and γ_j are the unknown parameters for f_1 and f_2 respectively, q_1 and q_2 are the number of unknown parameters for f_1 and f_2, while x_j^* and z_j^* are the knot locations for the two functions.

Function to Set Up Simple Two Term Additive Model

```
## 3.3.1 Penalized regression spline representation of an additive model
am.setup <- function(x,z,q=10)
# Get X, S_1 and S_2 for a simple 2 term AM
{ # choose knots ...
  xk <- quantile(unique(x),1:(q-2)/(q-1)) # knot locations x
  zk <- quantile(unique(z),1:(q-2)/(q-1)) # knot locations z
  # get penalty matrices ...
  S <- list() # initializes S to be a list
  S[[1]] <- S[[2]] <- matrix(0,2*q-1,2*q-1) # initialize two components
  S[[1]][2:q,2:q] <- spl.S(xk)[-1,-1]
  S[[2]][(q+1):(2*q-1),(q+1):(2*q-1)] <- spl.S(zk)[-1,-1]
  # get model matrix ...
  n <- length(x)
  X <- matrix(1,n,2*q-1)
  X[,2:q] <- spl.X(x,xk)[,-1]           # 1st smooth
  X[,(q+1):(2*q-1)] <- spl.X(z,zk)[,-1] # 2nd smooth
  list(X=X,S=S)
}
```

Function to Fit the Model and Calculate the GCV Score

```
fit.am <- function(y,X,S,sp)
# function to fit simple 2 term additive model
{ # get sqrt of total penalty matrix ...
  rS <- mat.sqrt(sp[1]*S[[1]]+sp[2]*S[[2]])
  q.tot <- ncol(X)                       # number of params
  n <- nrow(X)                           # number of data
  X1 <- rbind(X,rS)                      # augmented X
  y1 <- c(y,rep(0,q.tot))                # augment data
  b<-lm(y1~X1-1)                         # fit model
  trA<-sum(influence(b)$hat[1:n])        # tr(A)
  norm<-sum((y-fitted(b)[1:n])^2)        # RSS
  list(model=b,gcv=norm*n/(n-trA)^2,sp=sp)
}
```

Estimate an Additive Model to Fit Data for 31 Felled Cherry Trees

- We will use this routine to estimate an additive model for the data in data frame: **`trees`**.
  - **`Volume`**, **`Girth`** and **`Height`** for 31 felled cherry trees.
  - We want to predict **`Volume`** using the model:

$$\texttt{Volume} = f_1(\texttt{Girth}) + f_2(\texttt{Height}) + \varepsilon_i$$

- Given the simple smoothers we are using, we must rescale the predictors onto [0,1]:

```
data(trees) # get tree data
rg <- range(trees$Girth) # rg has low, high of girth
trees$Girth <- (trees$Girth - rg[1])/(rg[2]-rg[1])
rh <- range(trees$Height)
trees$Height <- (trees$Height - rh[1])/(rh[2]-rh[1])
```

- Then we obtain the model matrix and penalty matrices:

```
am0 <- am.setup(trees$Girth,trees$Height)
```

Grid Search to Find Model Fit Minimizing GCV Score

- Now we perform a grid search to find the model fit that approximately minimizes the GCV score:

```
sp <-c(0,0)  # initialize smoothing parameter (s.p.) array
for (i in 1:30) for (j in 1:30)   # loop over s.p. grid
{ sp[1]<-1e-5*2^(i-1);sp[2]<-1e-5*2^(j-1) # s.p.s
  b<-fit.am(trees$Volume,am0$X,am0$S,sp)  # fit using s.p.s
  if (i+j==2) best<-b else                # store 1st model
  if (b$gcv<best$gcv) best<-b             # store best model
}
best$sp   # GCV best smoothing parameters found
[1]   0.01024  5368.70912
```

`Height` has **high** smoothing parameter (straight line)

`Girth` has **low** smoothing parameter (allows curvature)

Grid Search to Find Model Fit Minimizing GCV Score

- So smooth of girth has a fairly low smoothing parameter (0.01), allowing f_1 some curvature, whereas f_2 has a very high smoothing parameter (5368.71).
- We obtain values of the smooths at predictor variable values by (1) zeroing all model coefficients (except those corresponding to the term of interest, and (2) using **`predict()`**: Next slide . . .

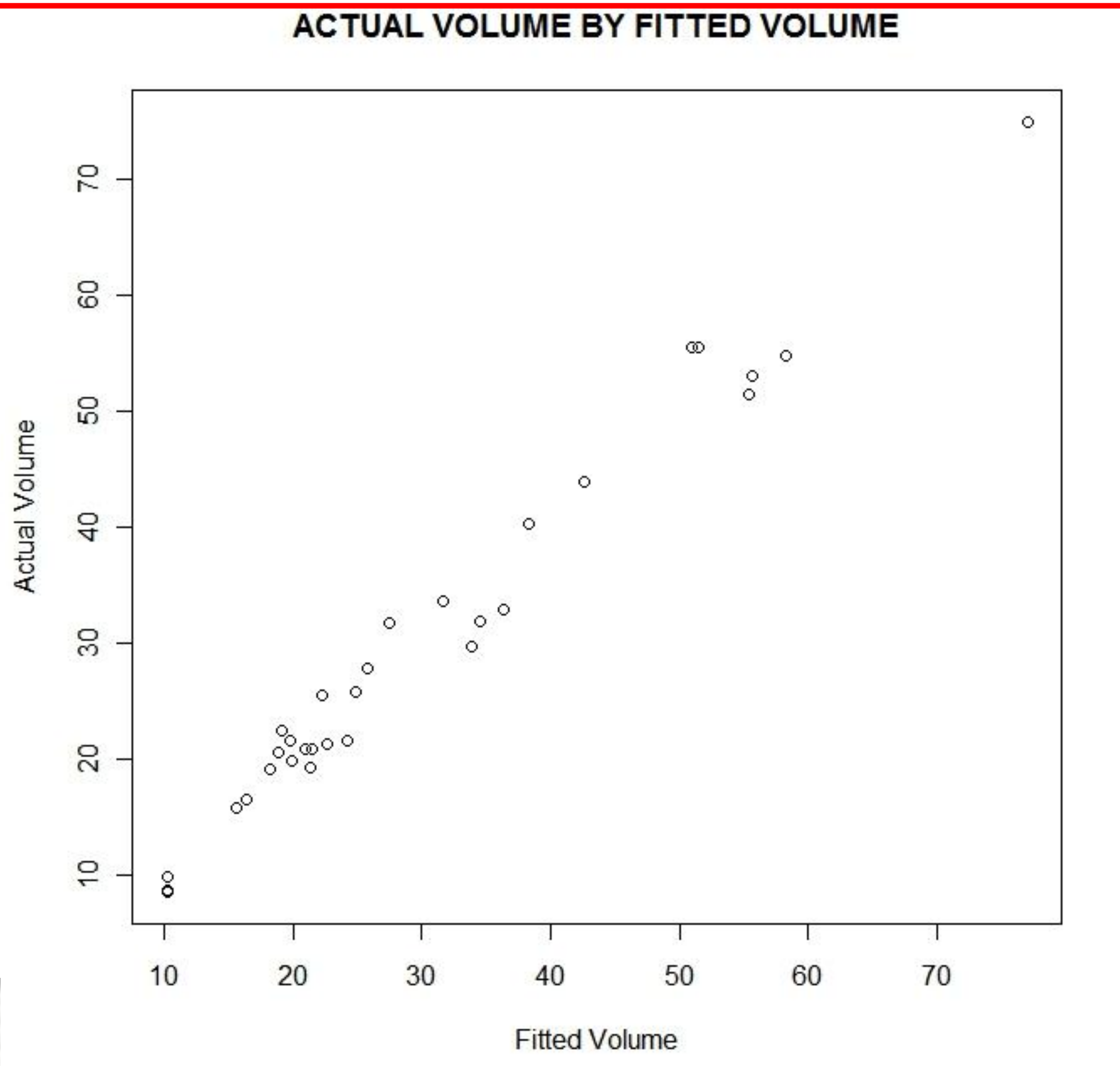
Plot: 1. Actual Volume versus Fitted; 2. f_1 Predicted by Scaled Girth

- Determine values of smooths at predictor variable values by zeroing all model coefficients (except predicted value):

```
# plot fitted volume against actual volume ...
plot(trees$Volume,fitted(best$model)[1:31],
   xlab="Fitted Volume",ylab="Actual Volume")
# evaluate and plot f_1 against Girth ...
b <-best$model
b$coefficients[1]<-0     # zero the intercept
b$coefficients[11:19]<-0 # zero the second smooth coefs
f0<-predict(b)           # predict f_1 only, at data values
plot(trees$Girth,f0[1:31],xlab="Scaled Girth",
   ylab=expression(hat(f[1])))
```

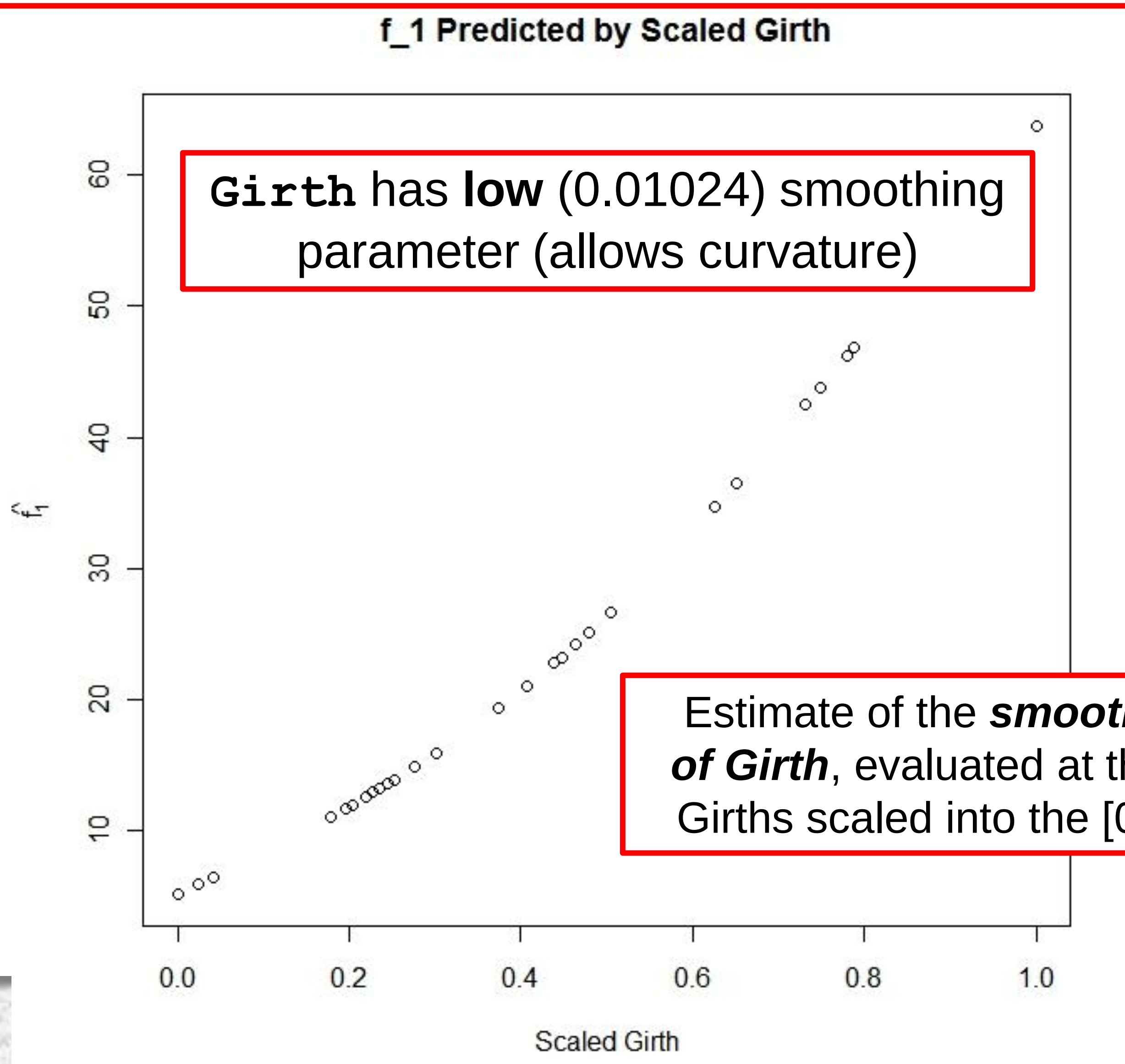
Plot Actual Volume versus Fitted

plot(trees$Volume,fitted(best$model)[1:31],main= “ACTUAL VOLUME BY FITTED VOLUME", xlab="Fitted Volume", ylab="Actual Volume")



Plot f_1 Predicted by Scaled Girth

```
plot(trees$Girth,f0[1:31],main= "f_1 Predicted by Scaled Girth",xlab="Scaled Girth", ylab=expression(hat(f[1])))
```



`Girth` has **low** (0.01024) smoothing parameter (allows curvature)

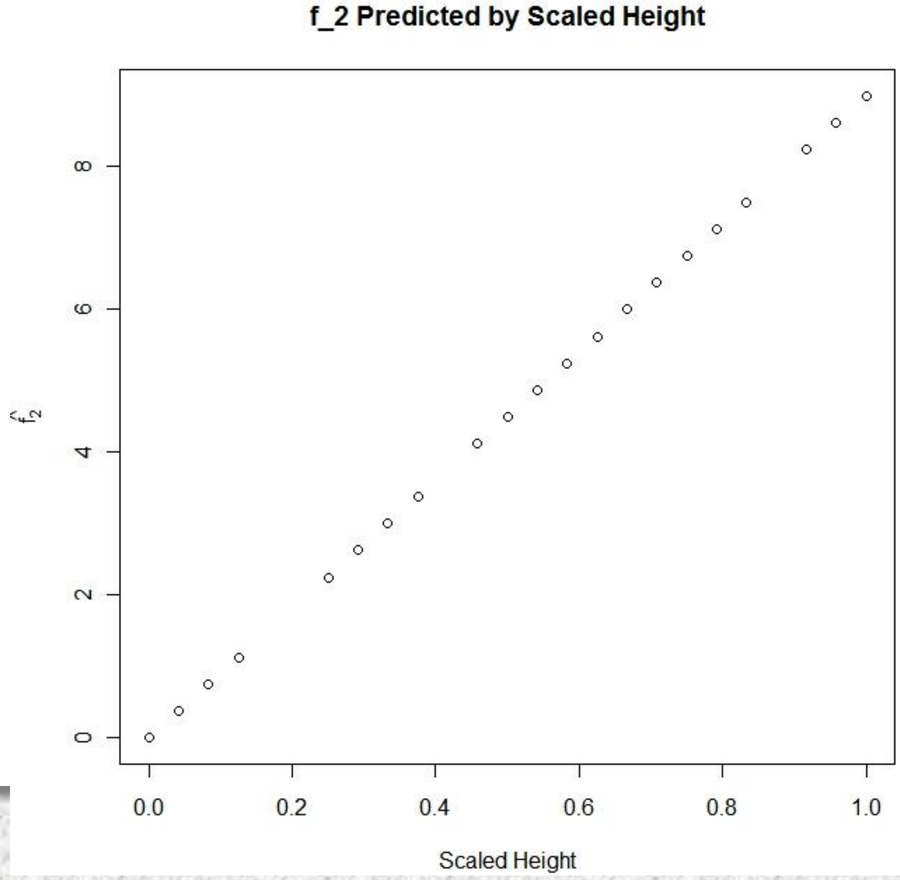
Estimate of the ***smooth function of Girth***, evaluated at the observed Girths scaled into the [0,1] interval.

Plot f_2 Predicted by Scaled Height

```
# evaluate and plot f_2 against Height ...
b <- best$model
b$coefficients[1] <- 0      # zero the intercept
b$coefficients[2:10] <- 0   # zero the first smooth coefs
f0<-predict(b)              # predict f_2 only, at data values
plot(trees$Height,f0[1:31],main= "f_2 Predicted by Scaled
      Height",xlab="Scaled Height",ylab=expression(hat(f[2])))
```

Estimate of the ***smooth function of Height***, evaluated at the observed Heights scaled into the [0,1] interval.

`Height` has **high** (5368) smoothing parameter (disallows curvature).



Generalized Additive Models

- **Generalized Additive Models** (GAMs) follow from **Additive Models**, as **GLMs** follow from **Linear Models**.
- **Linear Predictor** predicts some known ***smooth monotonic function*** of the **Expected Value** of the ***response***, which
  - Follows any exponential family distribution; or
  - Has a known mean variance relationship.
- Consider **`trees`** data again using GAM of form:

$$\texttt{log\{E(Volume}_i\texttt{)\}} = f_1\texttt{(Girth}_i\texttt{)} + f_2\texttt{(Height}_i\texttt{)}, \quad \texttt{Volume} \sim \texttt{Gamma}$$

`tree` GAM: What Needs To Be Done?

- Go back and look at **`fit.am()`** **`on`** slide 35:

```
fit.am <- function(y,X,S,sp)
# function to fit simple 2 term additive model
{ # get sqrt of total penalty matrix ...
  rS <- mat.sqrt(sp[1]*S[[1]]+sp[2]*S[[2]])
  q.tot <- ncol(X)                     # number of params
  n <- nrow(X)                         # number of data
  X1 <- rbind(X,rS)                    # augmented X
  y1 <- c(y,rep(0,q.tot))              # augment data
  b<-lm(y1~X1-1)    b<-glm(y1~X1-1)? NO!  # fit model
  trA<-sum(influence(b)$hat[1:n]) # tr(A)
  norm<-sum((y-fitted(b)[1:n])^2) # RSS
  list(model=b,gcv=norm*n/(n-trA)^2,sp=sp)
}
```

Additive model was estimated by **penalized least squares**.
GAM will be fitted by **penalized likelihood maximization**.
Using **penalized iteratively re-weighted least squares** (**P-IRLS**)

`tree` GAM: What Needs To Be Done?

- Modify **`fit.am()`** to **`fit.gamG()`** below:

```
fit.gamG <- function(y,X,S,sp)
# function to fit simple 2 term generalized additive model
# Gamma errors and log link
{ # get sqrt of combined penalty matrix ...
  rS <- mat.sqrt(sp[1]*S[[1]]+sp[2]*S[[2]])
  q <- ncol(X)    # number of params
  n <- nrow(X)    # number of data
  X1 <- rbind(X,rS)      # augmented model matrix
  eta <- log(y)          # initialize linear predictor
  norm <- 0;old.norm <- 1   # initialize convergence control
  while (abs(norm-old.norm)>1e-4*norm)  # repeat un-converged
  { mu <- exp(eta)             # fitted values
    z <- (y-mu)/mu + eta       # pseudodata (recall w_i=1, here)
    z[(n+1):(n+q)] <- 0        # augmented pseudodata
    m <- lm(z~X1-1)            # fit penalized working model
    b <- m$coefficients        # current parameter estimates
    eta <- (X1%*%b)[1:n]       # 'linear predictor'
    trA <- sum(influence(m)$hat[1:n]) # tr(A)
    old.norm <- norm           # store for convergence test
    norm <- sum((z-fitted(m))[1:n]^2) # RSS of working model
  }
  list(model=m,gcv=norm*n/(n-trA)^2,sp=sp)
}
```

`tree` GAM: What Needs To Be Done?

- To find GCV optimum fit, simply **replace `fit.am()` function with `fit.gamG()` function** in the smoothing parameter grid search loop (slide 37) and repeated below:

```
## to find model fit that approximately minimizes GCV score
sp <- c(0,0)    # initialize smoothing parameter (s.p.) array
for (i in 1:30) for (j in 1:30)      # loop over s.p. grid
{ sp[1]<-1e-5*2^(i-1);sp[2]<-1e-5*2^(j-1) # s.p.s
  b<-fit.am(trees$Volume,am0$X,am0$S,sp)  # fit using s.p.s
  if (i+j==2) best<-b else                # store 1st model
  if (b$gcv<best$gcv) best<-b             # store best model
}
best$sp   # GCV best smoothing parameters found
```

LET'S TRY IT ! ! !

tree GAM: *We Try It !*

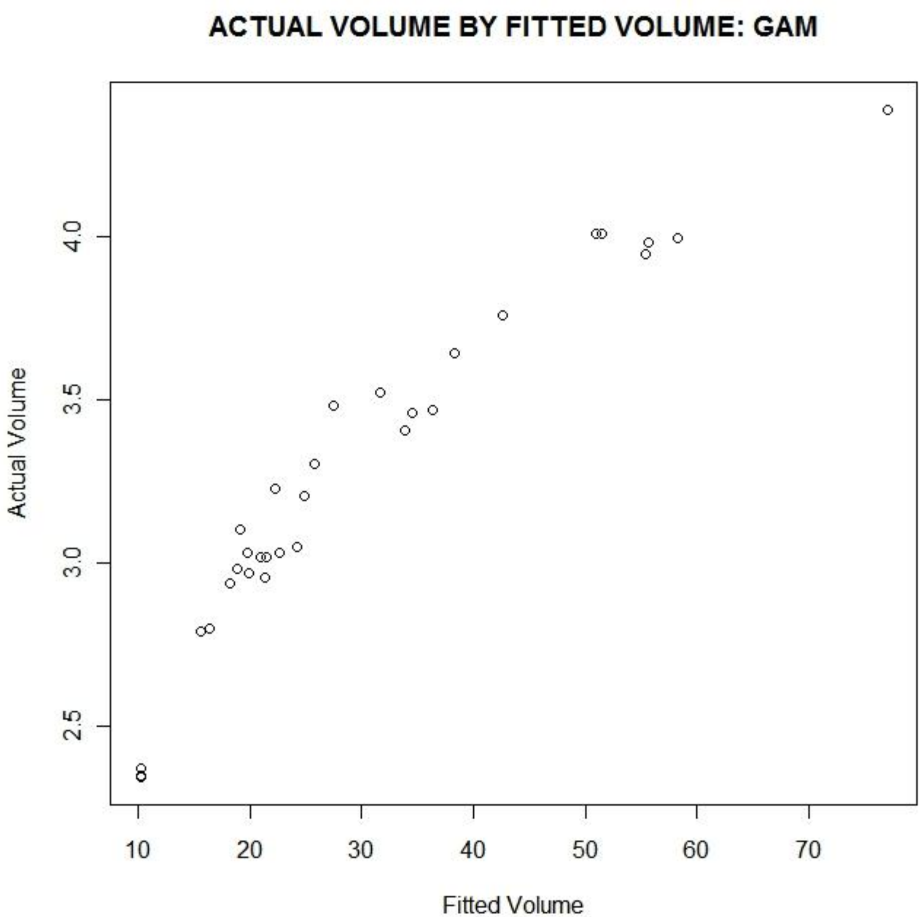
```
## to find model fit that approximately minimizes GCV score for
a GAM, we have
## replaced fit.am with fit.gamG in the smooting parameter grid
search loop below:
sp <- c(0,0)   # initialize smoothing parameter (s.p.) array
for (i in 1:30) for (j in 1:30)      # loop over s.p. grid
{ sp[1]<-1e-5*2^(i-1);sp[2]<-1e-5*2^(j-1) # s.p.s
  b<-fit.gamG(trees$Volume,am0$X,am0$S,sp)  # fit using s.p.s
  if (i+j==2) best<-b else                 # store 1st model
  if (b$gcv<best$gcv) best<-b              # store best model
}
best$sp   # GCV best smoothing parameters found
[1]     0.01024 5368.70912
```

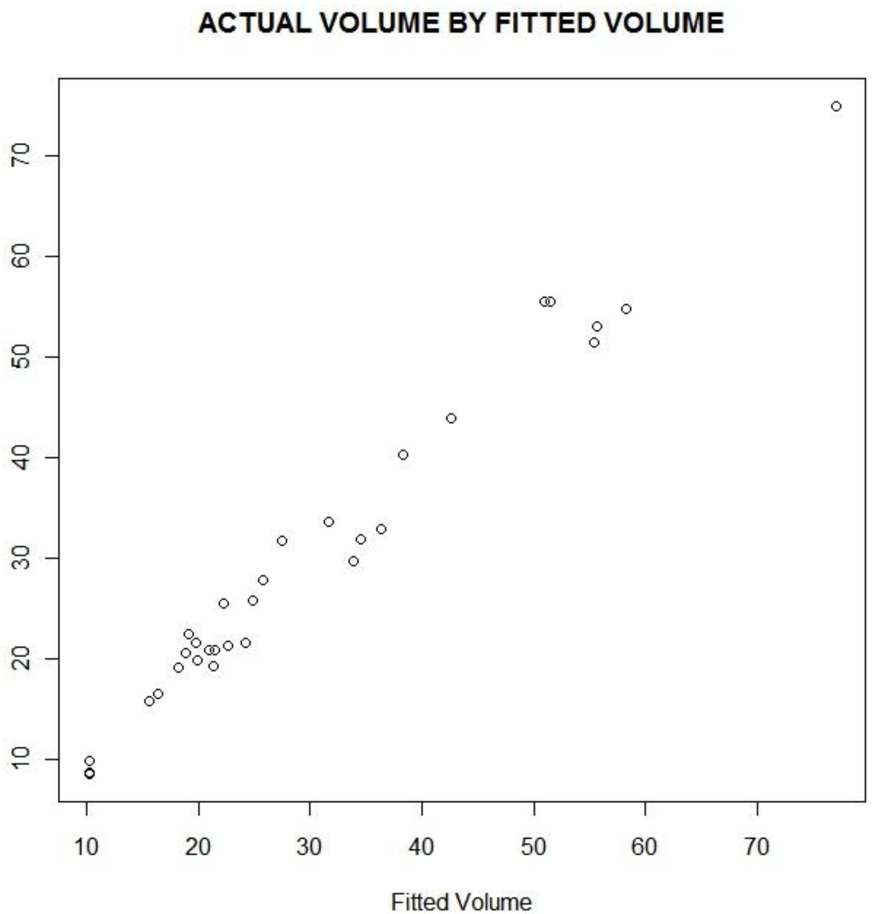
We get the same optimum smoothing GCV scores

Plot Actual Volume versus Fitted

```
plot(trees$Volume,fitted(best$model)[1:31],main= “ACTUAL VOLUME BY FITTED VOLUME", xlab="Fitted Volume", ylab="Actual Volume")
```

They are the same, but they should be !



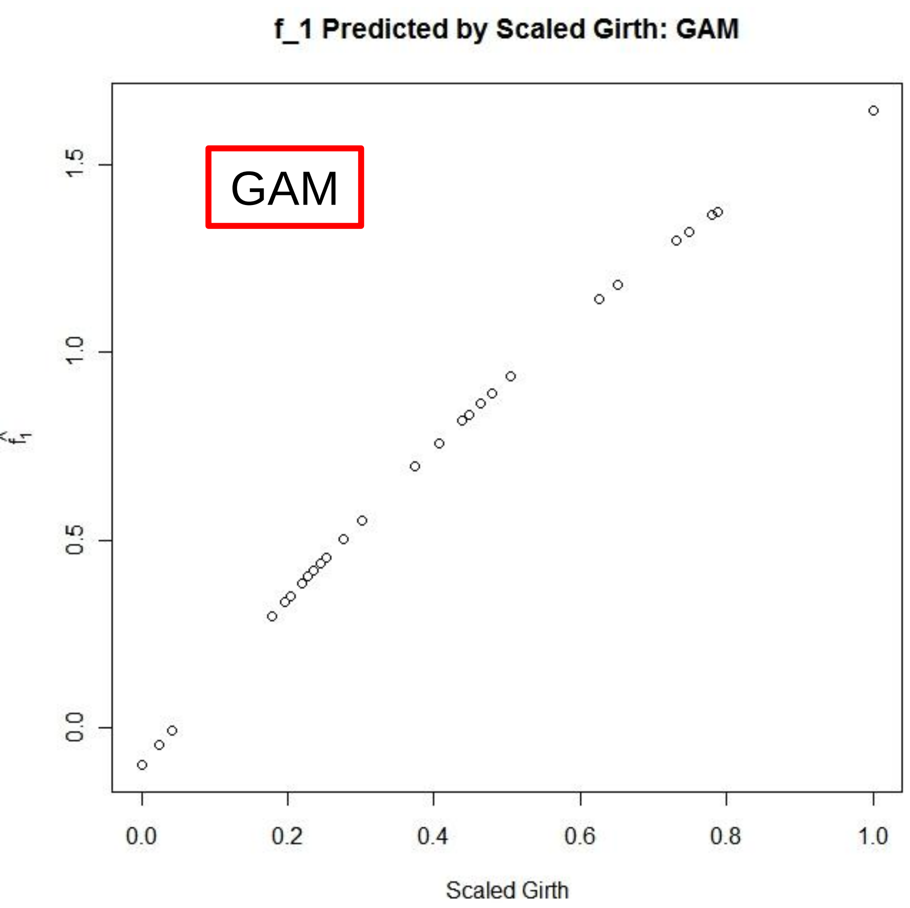


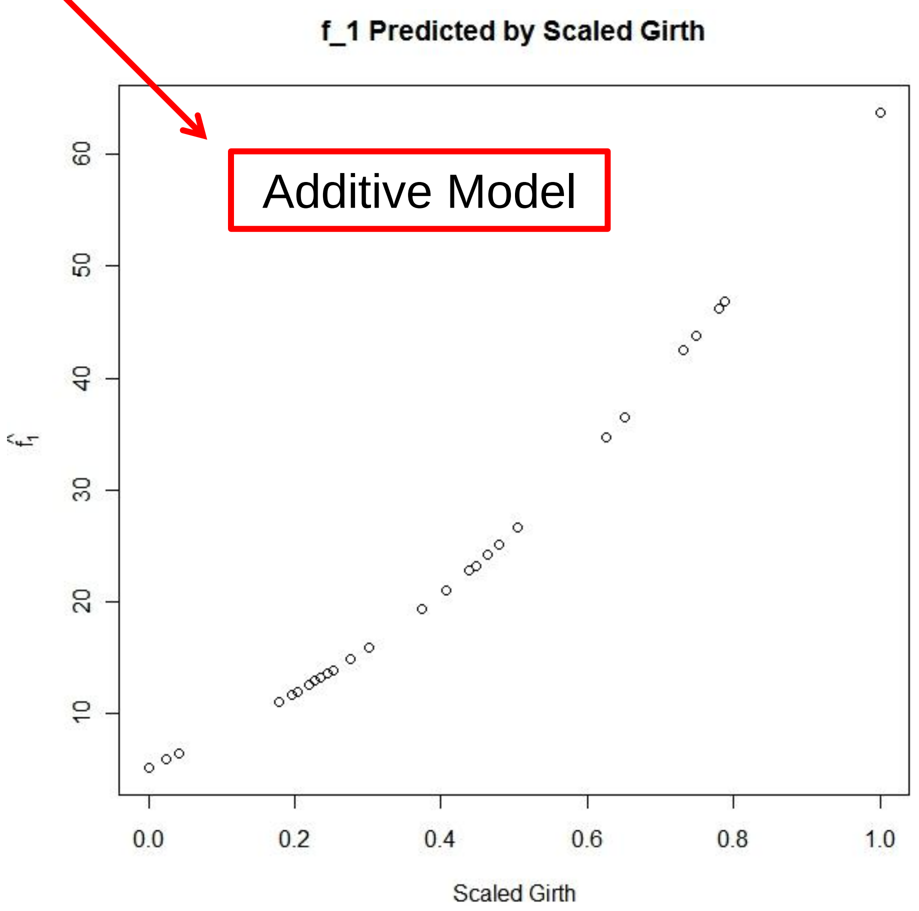
Plot f_1 Predicted by Scaled Girth

```
plot(trees$Girth,f0[1:31],main= "f_1 Predicted by Scaled Girth",xlab="Scaled Girth", ylab=expression(hat(f[1])))
```

`Girth` has **low** (0.01024) smoothing parameter (allows curvature)

Estimate of the ***smooth function of Girth***, evaluated at the observed Girths scaled into the [0,1] interval.





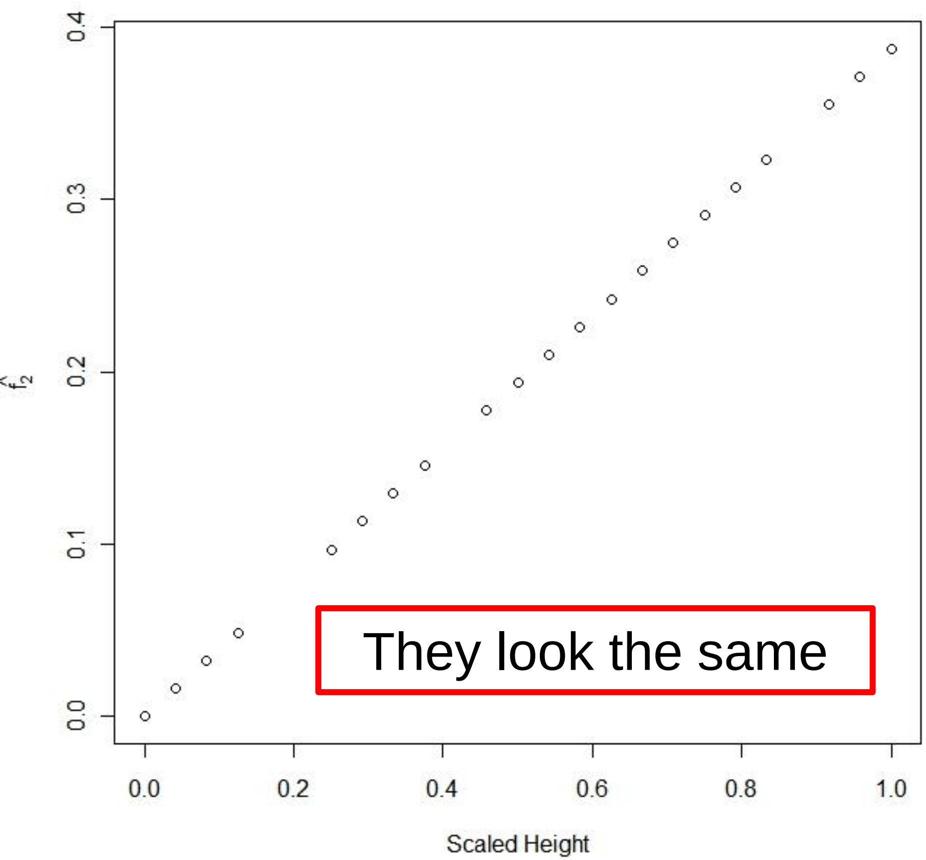
Plot f_2 Predicted by Scaled Height

plot(trees$Height,f0[1:31],main= "f_2 Predicted by Scaled Height: GAM",xlab="Scaled Height",ylab=expression(hat(f[2])))

`Height` has **high** (5368) smoothing parameter (disallows curvature).

Estimate of the ***smooth function of Height***, evaluated at the observed Heights scaled into the [0,1] interval.

f_2 Predicted by Scaled Height: GAM



They look the same

f_2 Predicted by Scaled Height