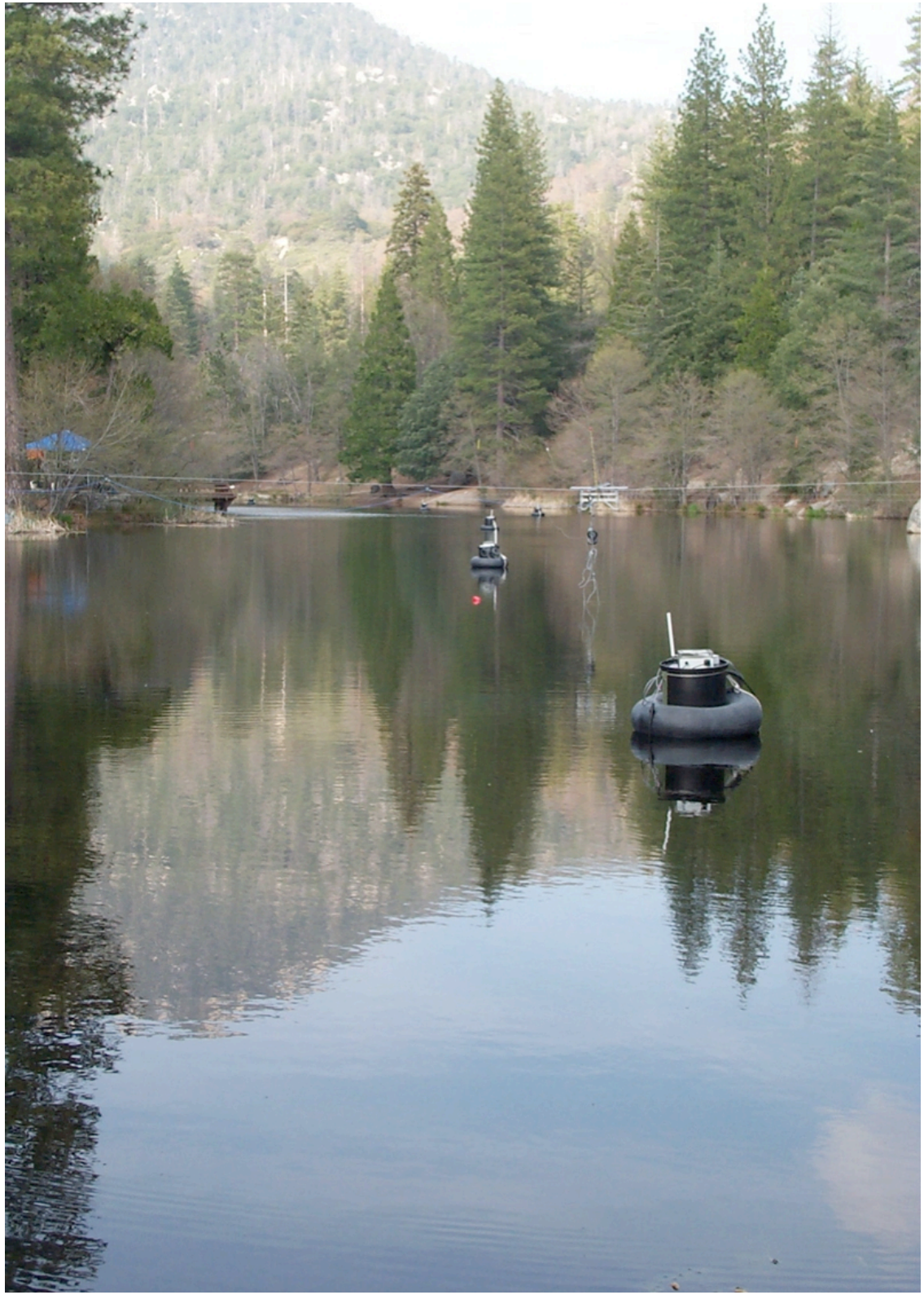
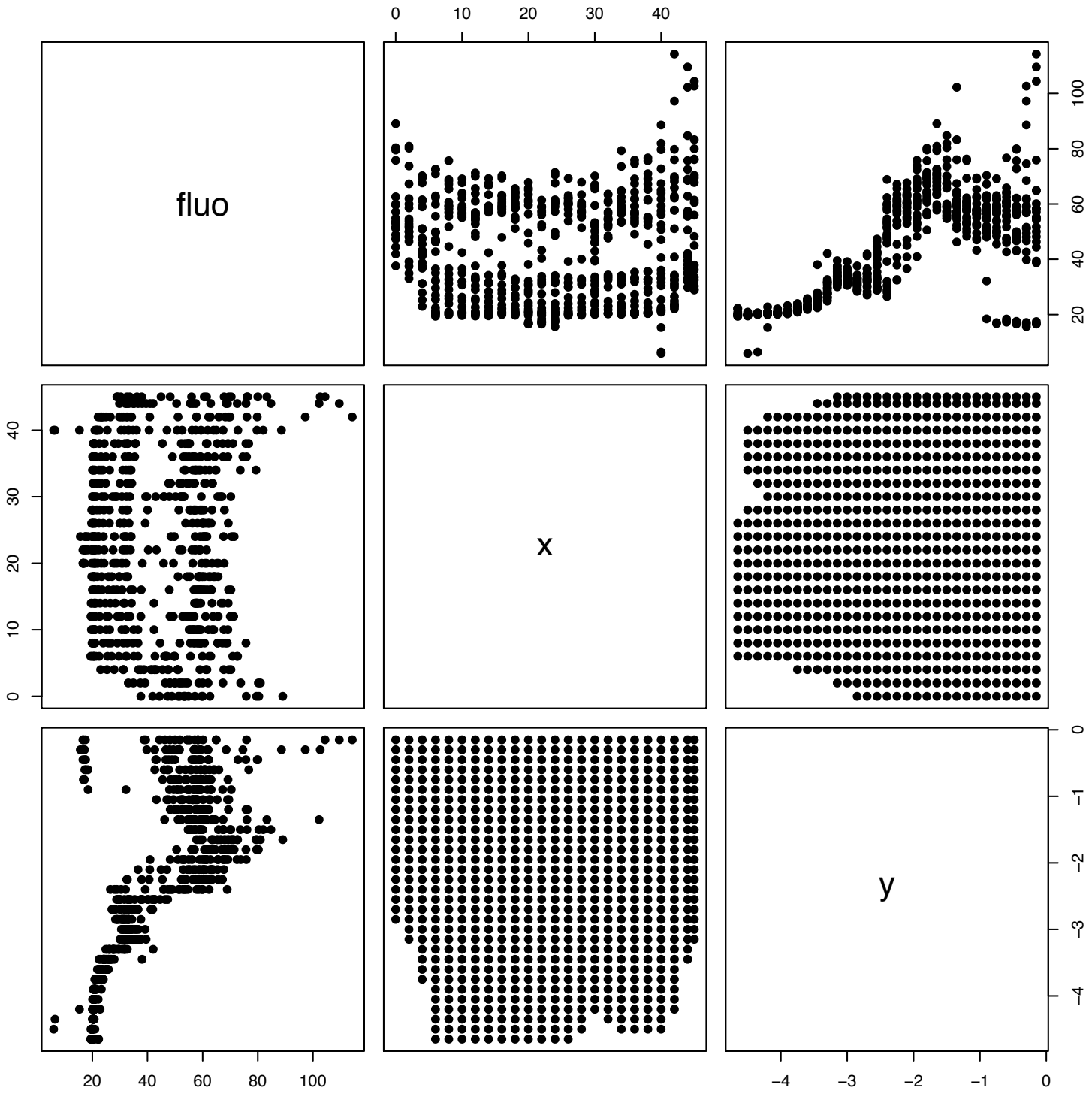
Lecture 9



```
# load the data
> lake = read.csv(url("http://www.stat.ucla.edu/~cocteau/lake.csv"),head=T)
> names(lake)
[1] "fluo" "x"    "y"

# our friend the scatterplot matrix
> pairs(lake,pch=16)

# something new
> library(lattice)
> cloud(fluo~x+y,data=lake)
> wireframe(fluo~x+y,data=lake,drape=T)
```

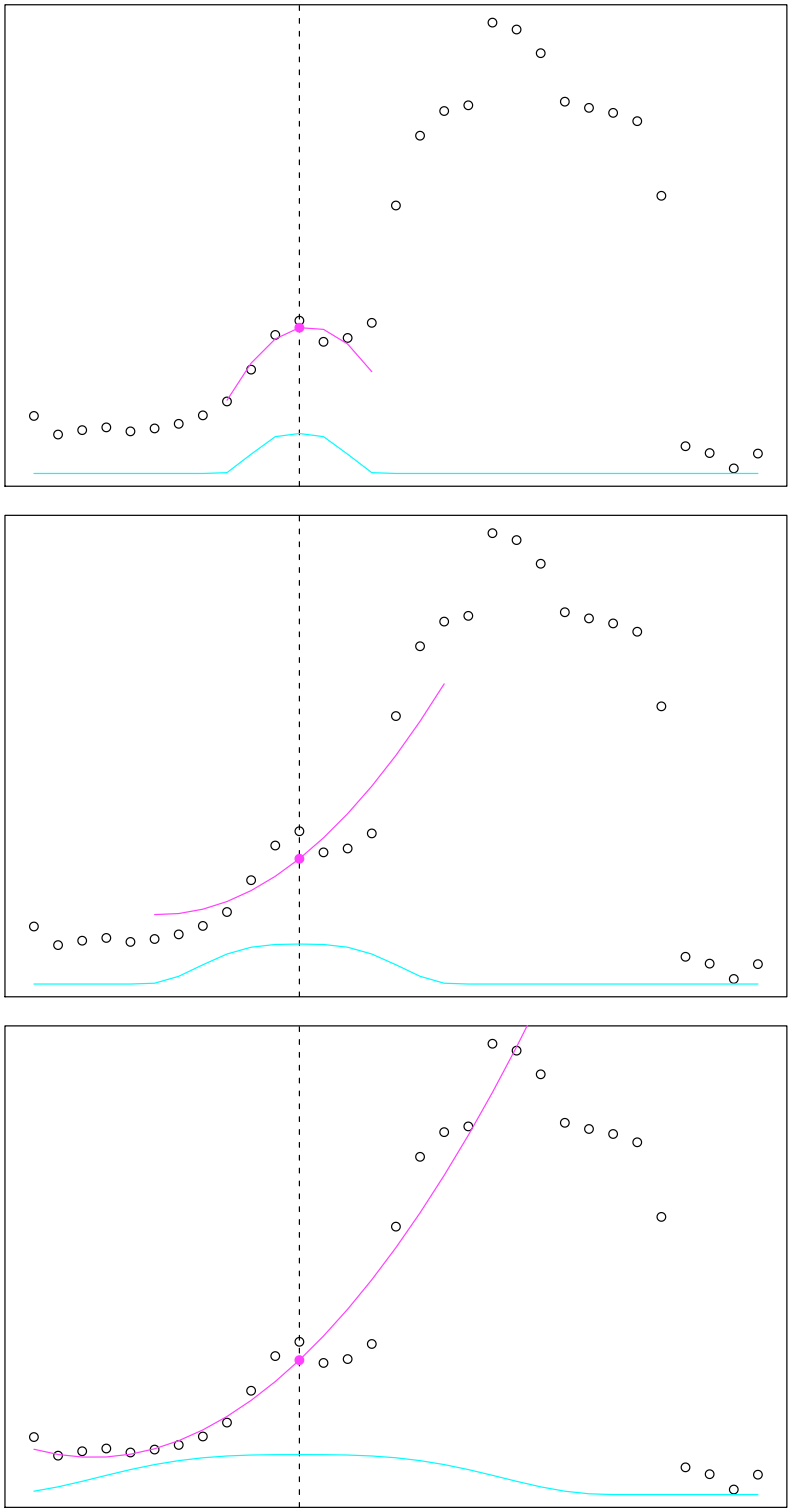

fluo
x
y
0
10
20
30
40
20
40
60
80
100
−4
−3
−2
−1
0

Local polynomials

To accomplish this “local” fit, we have to introduce the notion of a weighted regression; simply, instead of solving the OLS criterion, we are given weights for each data point

In the case of a local polynomial, these weights are biggest near the point we’d like to predict and then tend to zero as you move farther away; at the right we plot a few sets of weights and the associated local fits

In general, we have defined these weights using a kernel function; the wider the kernel’s “bandwidth,” the more points are included in the regression locally



Weighted regressions

To create these fits, we need the notion of a weighted regression; let's again consider the case of fitting a quadratic in the neighborhood of $x_0 = -3$ meters; for each $i = 1, \ldots, n$, let $w_i \geq 0$ be a weight (say, the cyan colored lines from the previous slide)

Then, we seek to solve not the OLS criterion but instead the weighted criterion

$$\sum_{i=1}^{n} w_i \left[y_i - \beta_0 - \beta_1 (x - x_0) - \beta_2 (x - x_0)^2 \right]^2$$

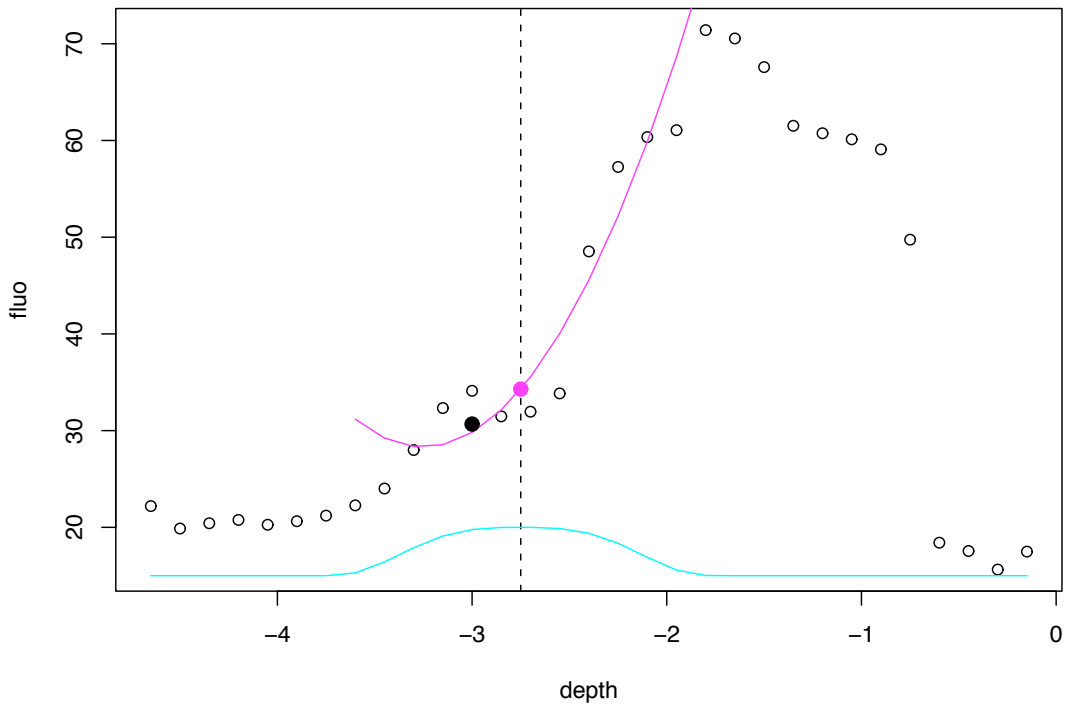
which gives $\widehat{\beta}_0, \widehat{\beta}_1, \widehat{\beta}_2$; we then take our estimate of $f(x_0)$ to be $\widehat{\beta}_0$

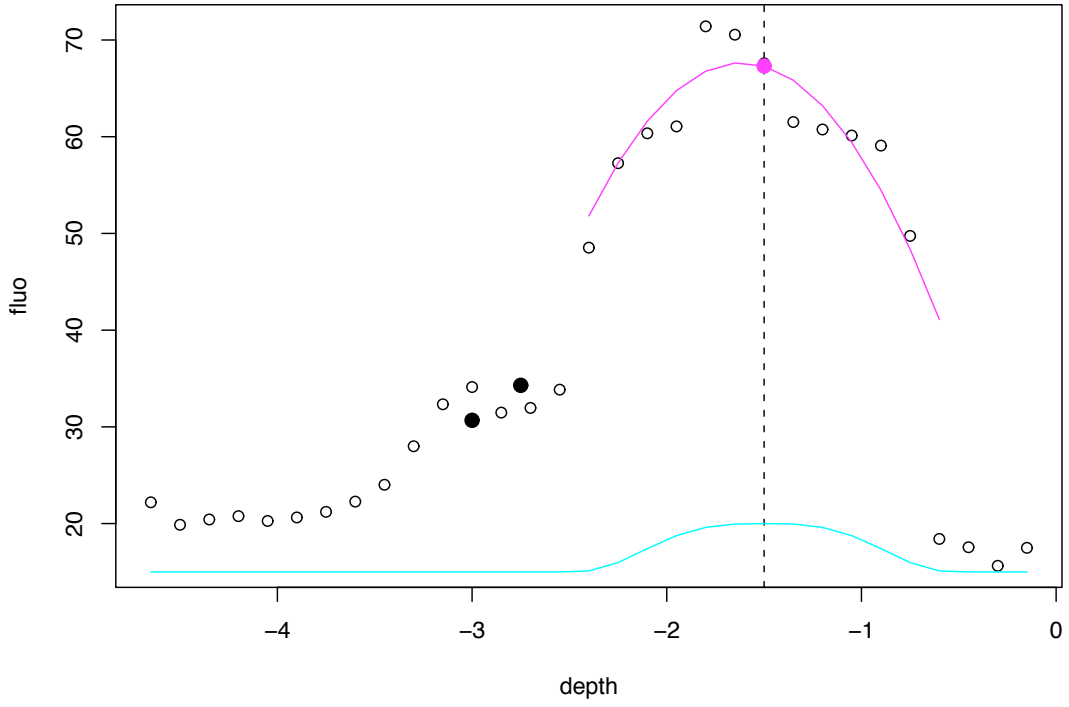
An example

We can continue the process and make predictions for any depth; here's -2.75 and -1.5, keeping the previous predictions marked in black

Again, keep in mind the basic character of the fit; a polynomial is being fit locally using the weights in cyan

Notice also that we're making predictions at any point; this procedure works whether or not we have an observation at x_0



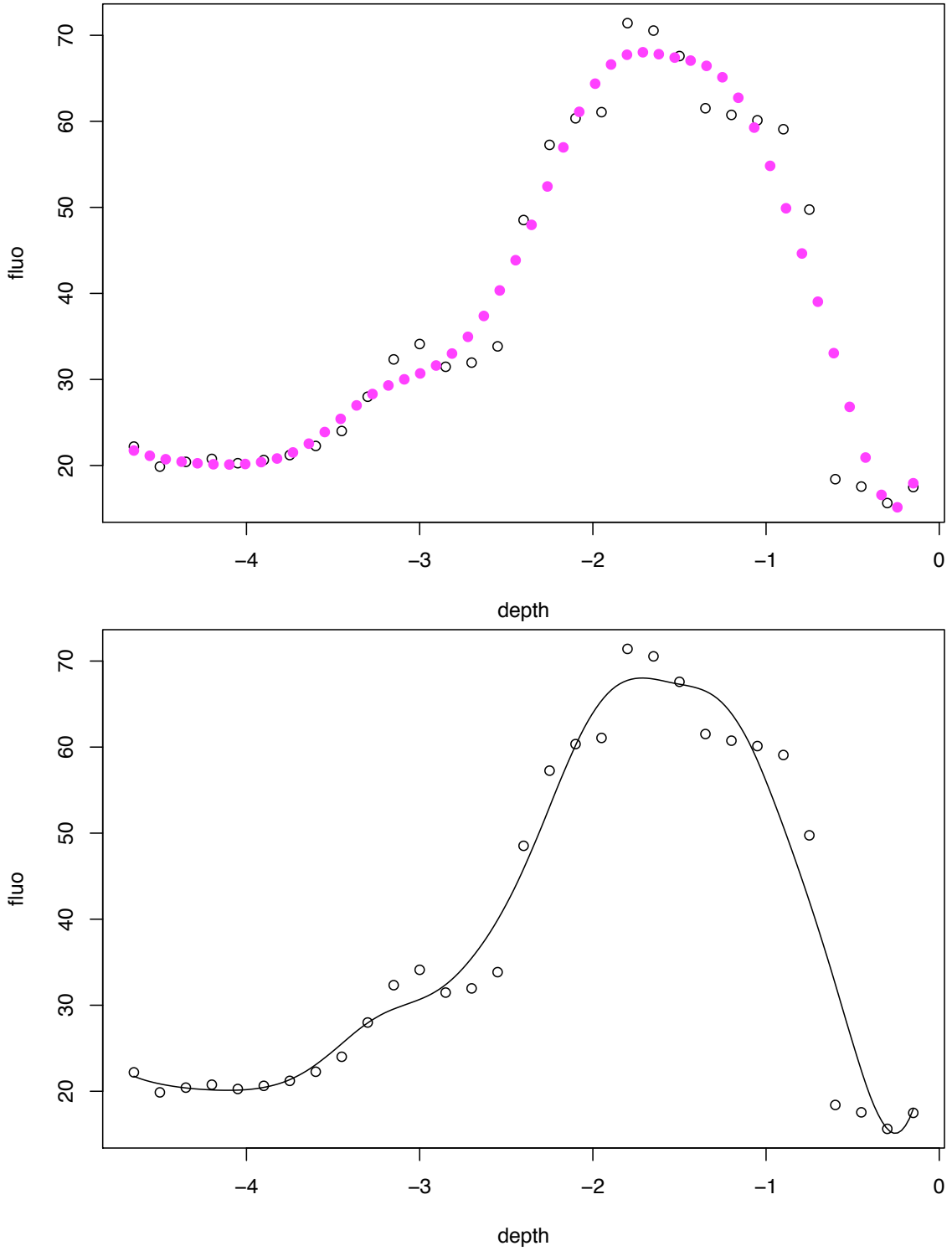


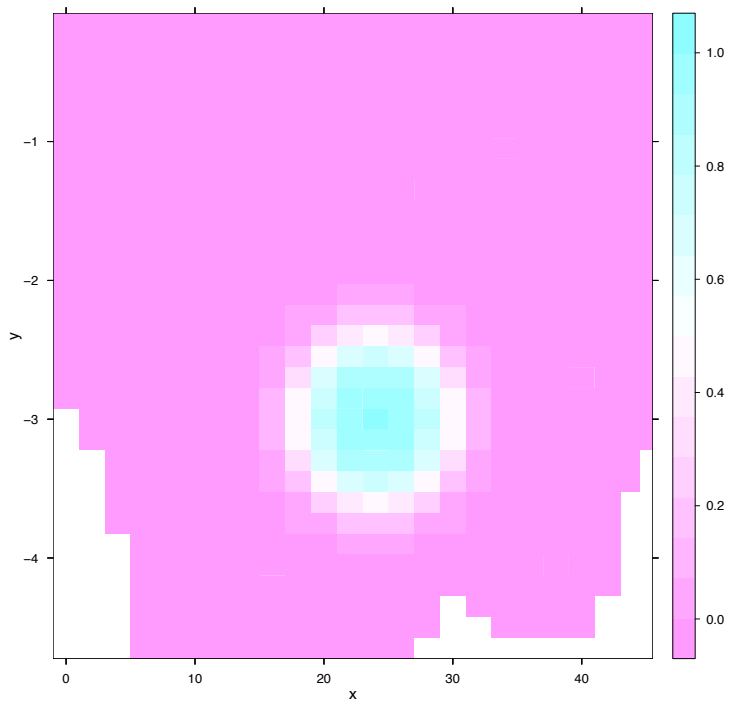
Local polynomials

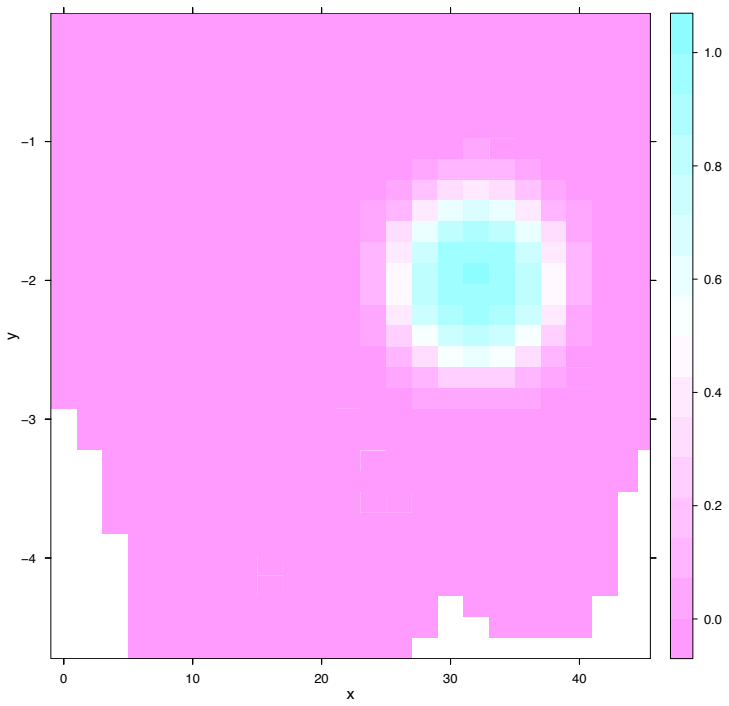
And voila! If we continue this process at a number of points we create a smooth curve (why does the curve have to be smooth?)

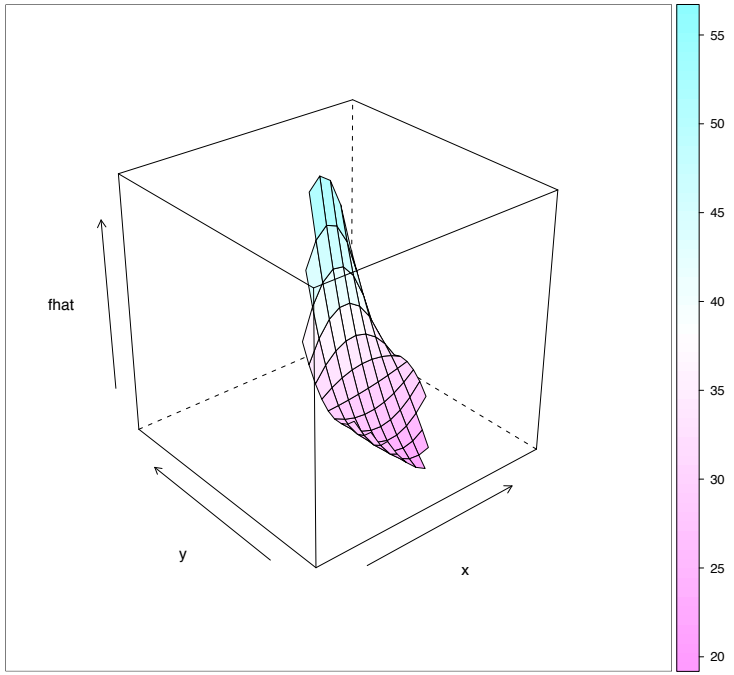
The fit at the right uses a “bandwidth” of 1; as we vary the bandwidth, we get smoother or wigglier fits

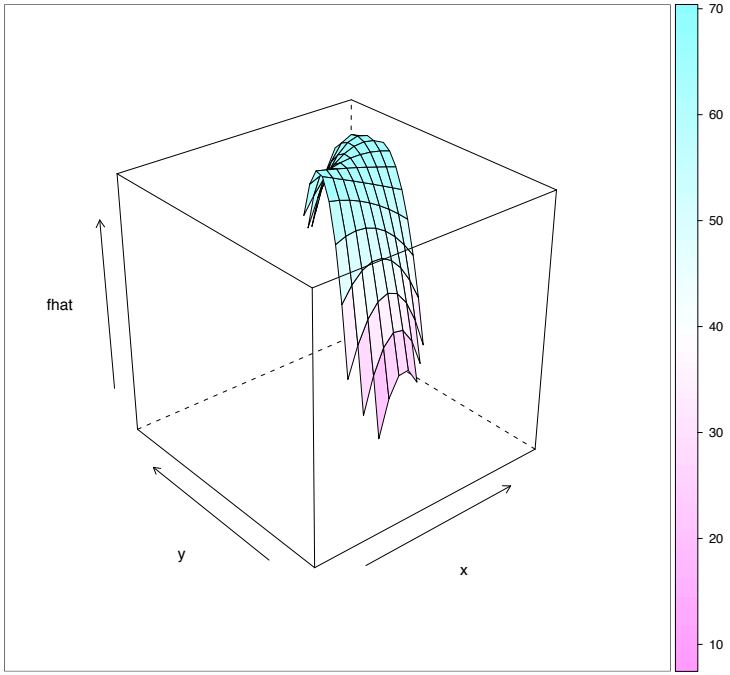
If the bandwidth is chosen very large, then we are essentially fitting a quadratic polynomial; if it is very small, we “interpolate the data”

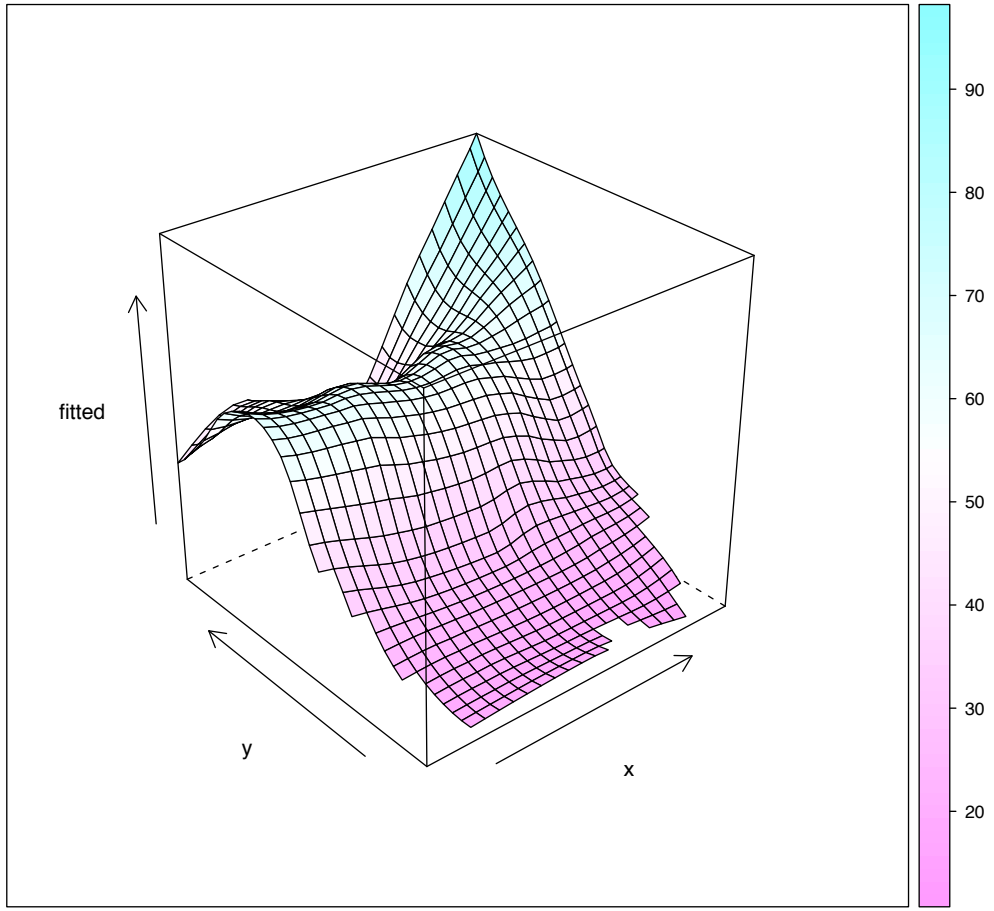



y
x
−1
−2
−3
−4
0
10
20
30
40
1.0
0.8
0.6
0.4
0.2
0.0


y
x
−1
−2
−3
−4
0
10
20
30
40
1.0
0.8
0.6
0.4
0.2
0.0


fhat
y
x
55
50
45
40
35
30
25
20


fhat
y
x
70
60
50
40
30
20
10



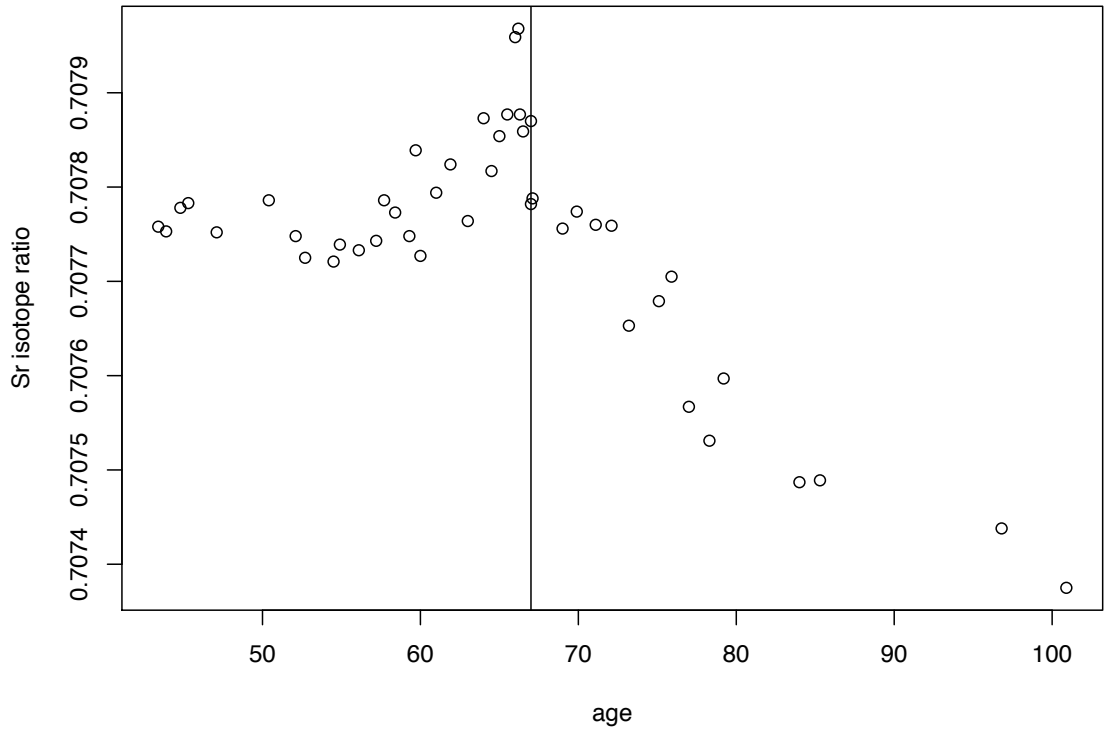

```
# load up graphics library to make lovely 3d plots

library(lattice)

# fit a local polynomial in two dimensions using the full
# lake data set

fit = loess(fluo~x+y,data=lake,enp.target=25)

# create a new data frame to plot with and plot the fit

newlake = data.frame(x=lake$x,y=lake$y,fitted=predict(fit))
wireframe(fitted~x+y,data=newlake,drape=T)
```

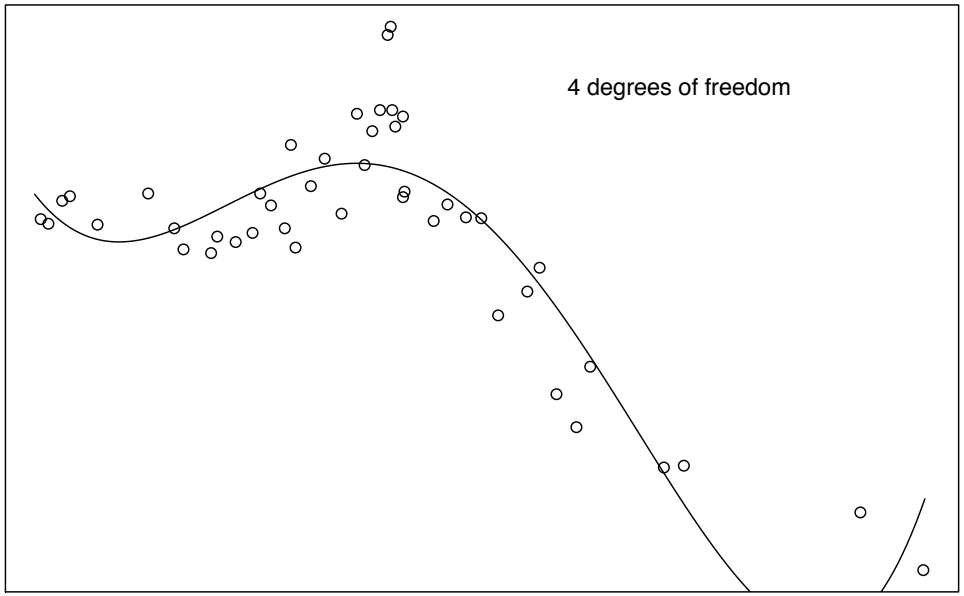
Sr isotopes

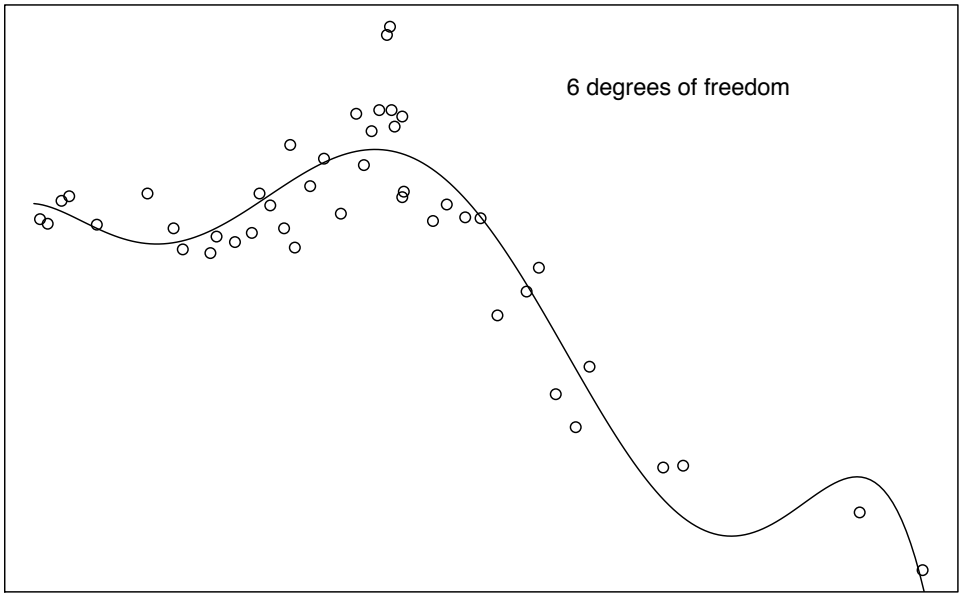
We're going to start by looking at what is now a fairly old data set but it illustrates some of
the reasons why one might want to appeal to constructions like piecewise polynomials

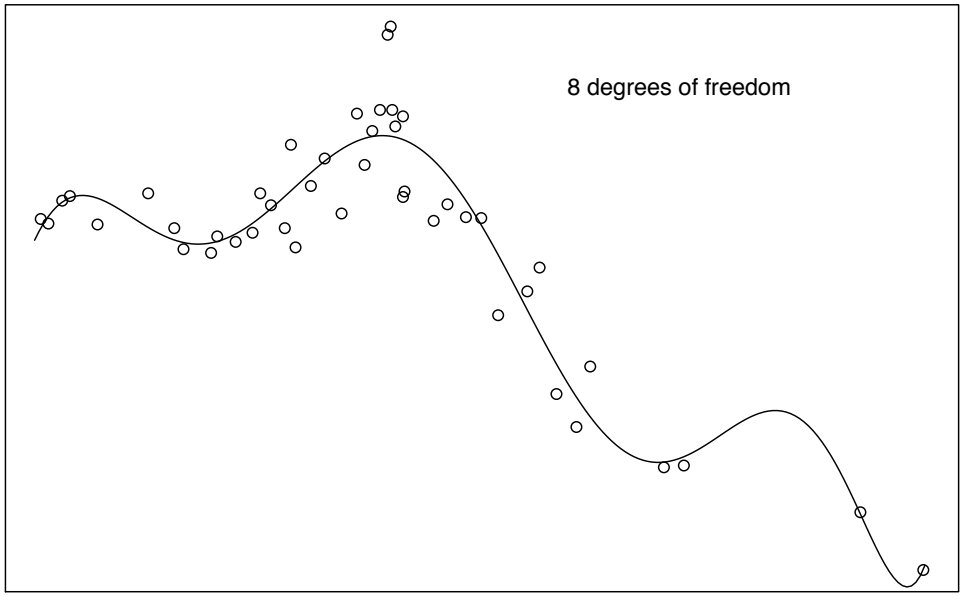
1. The "predictors" are unequally spaced (notice the data points trail off after about 80 million years ago)

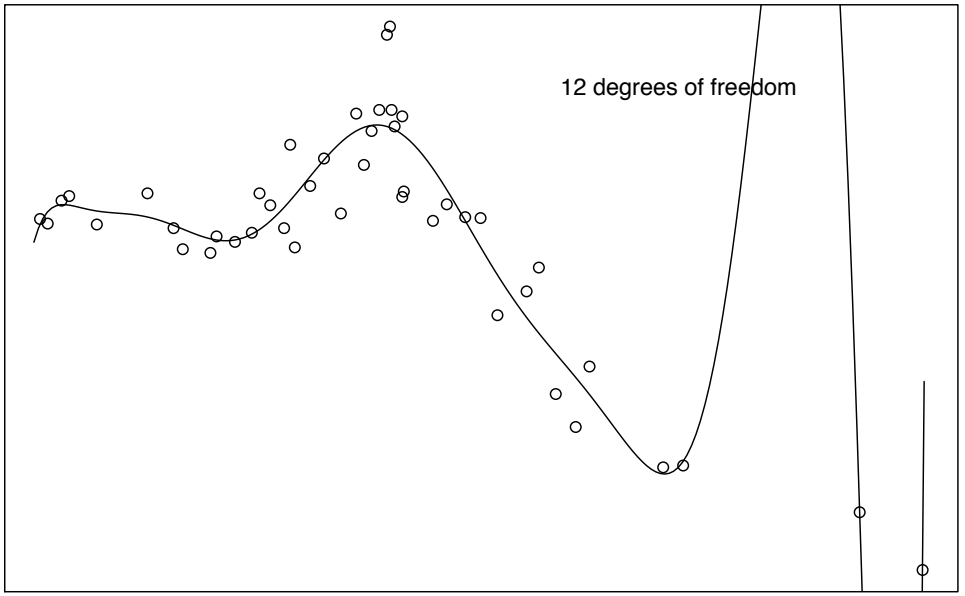
2. There is scientific interest in a feature of the estimated curve (the jump at 67 million years ago tells us something about the catastrophe that resulted in a mass extinction)



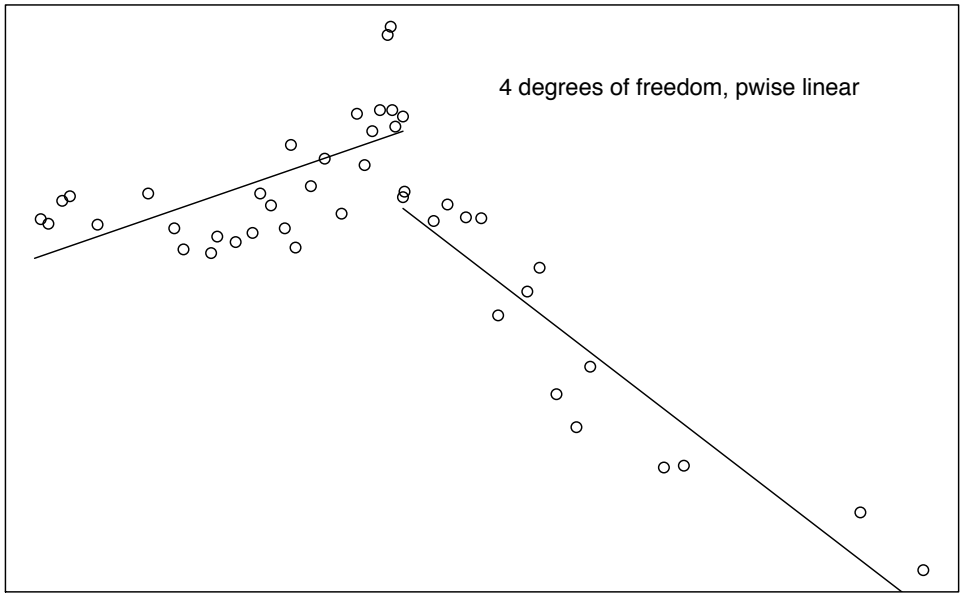


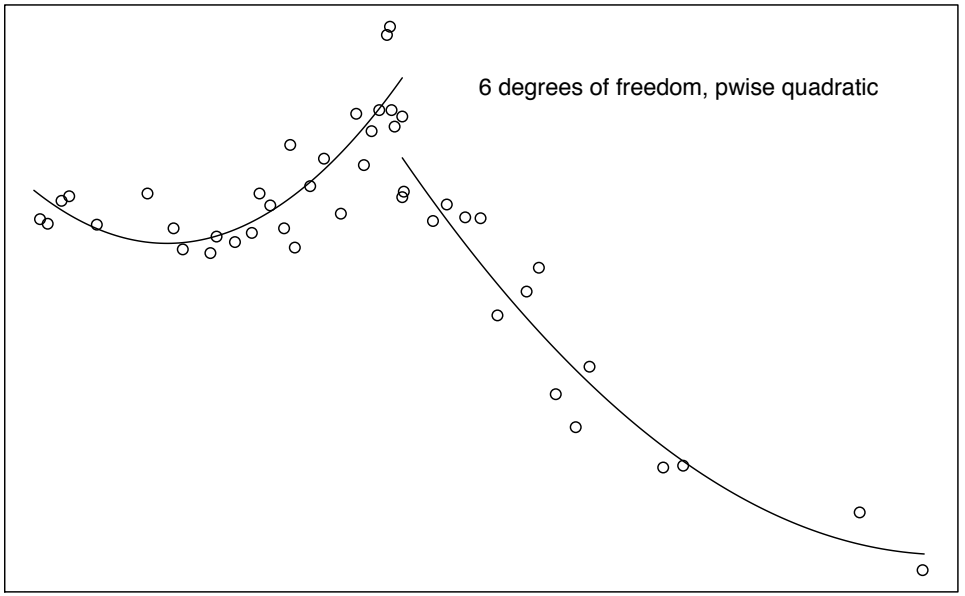




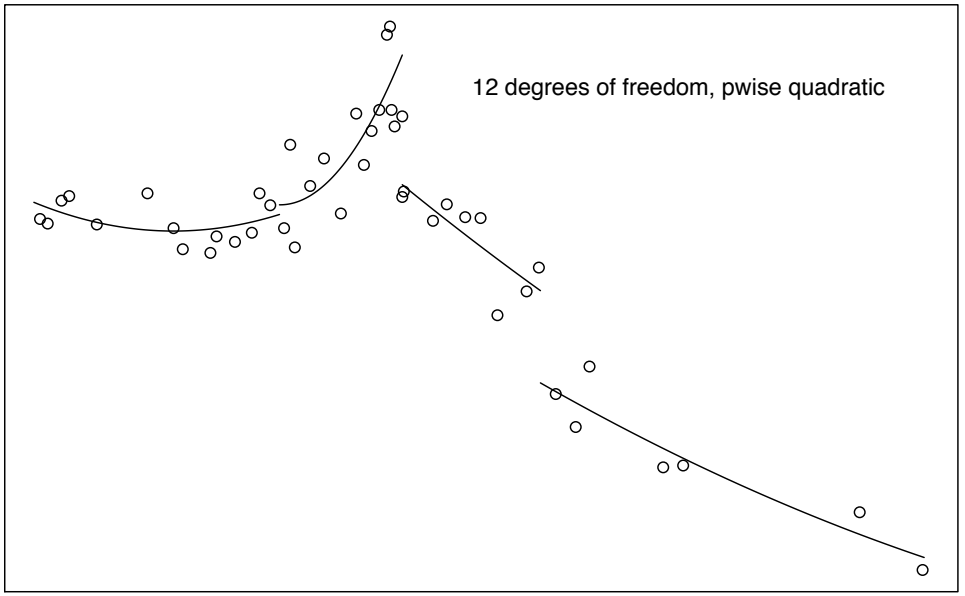


*This is why Tukey referred to high-degree polynomials as “sharp” tools


4 degrees of freedom, pwise linear


6 degrees of freedom, pwise quadratic


8 degrees of freedom, pwise linear


12 degrees of freedom, pwise quadratic

Smooth, piecewise linear

At the right, we have a piecewise linear fit, but we have constrained the different pieces to join, making a kind of broken stick; in technical terms, we've enforced a continuity condition on the model

The "breakpoints" or "knots" are located at 59, 67 and 76 (the positions used in the previous slides)

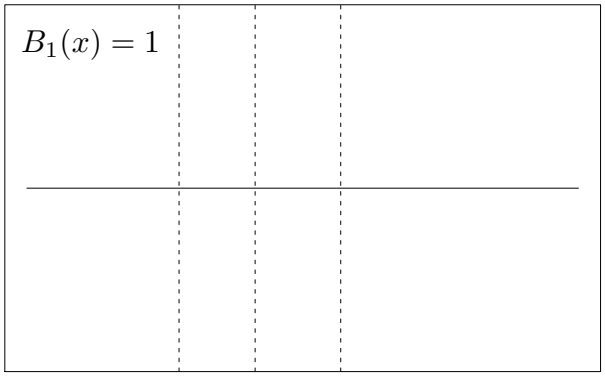
How many degrees of freedom do we have now? What "basis" functions do we use here?

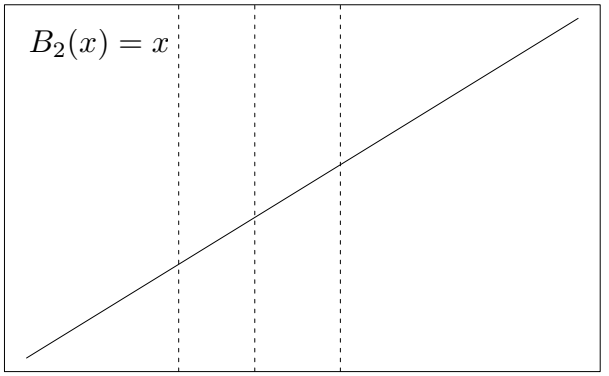
Basis choice

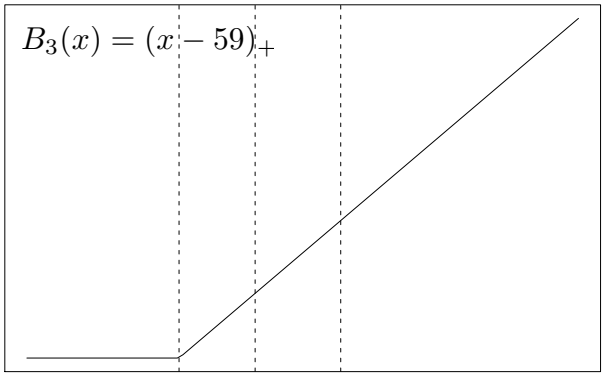
Rather than try to solve a "constrained" least squares problem in which we force the ends of the lines to join, it's easier to work with basis functions that have the conditions we're after

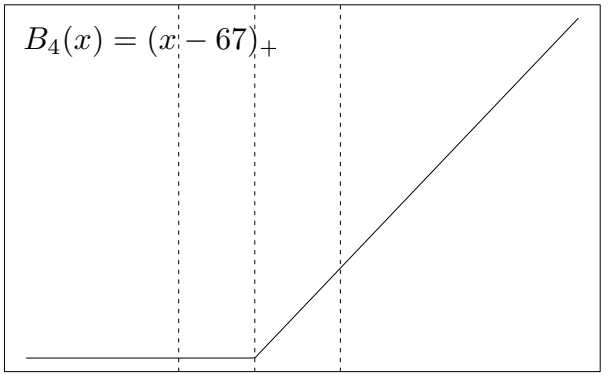
At the right we show five basis functions that can be used in a fitting routine to create the broken-stick model on the previous slide

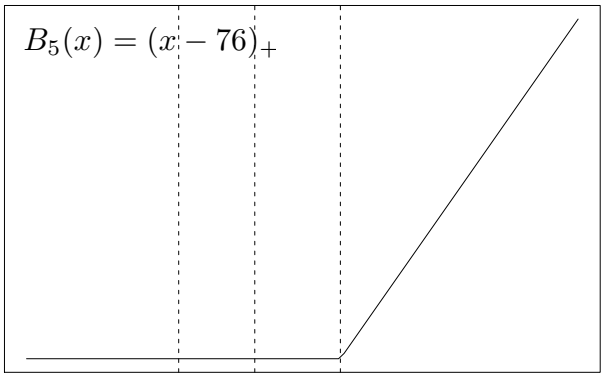
This is called the **truncated power basis**; anything interesting here?







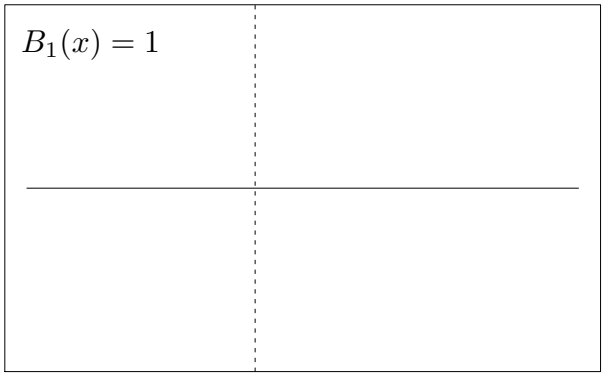


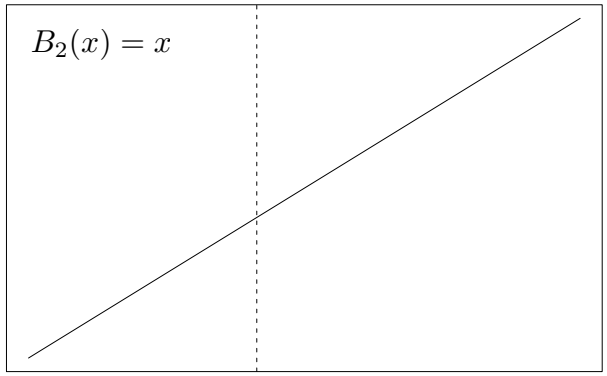


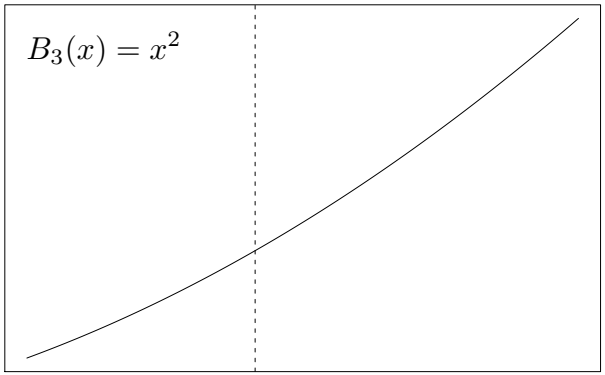
Truncated power basis

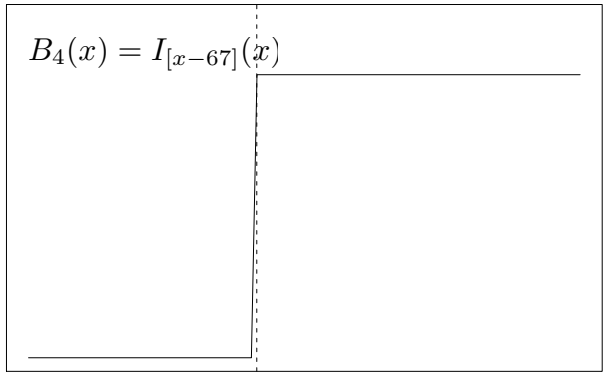
The truncation idea generalizes pretty easily; here is the basis associated with the full set of piecewise quadratics

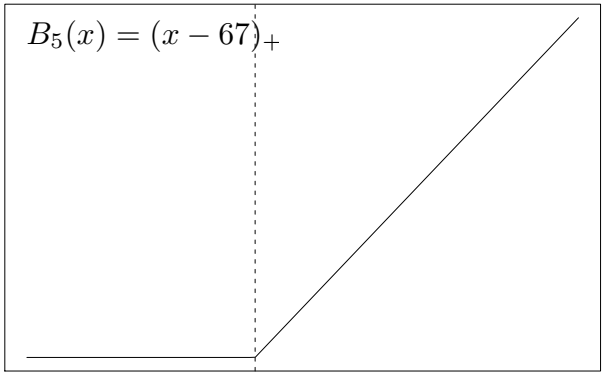
If we wanted to enforce continuity across $x = 67$ what would we do? What about creating one continuous derivative? Two continuous derivatives?

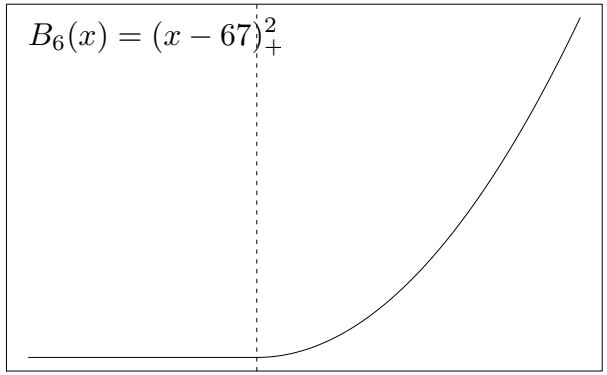


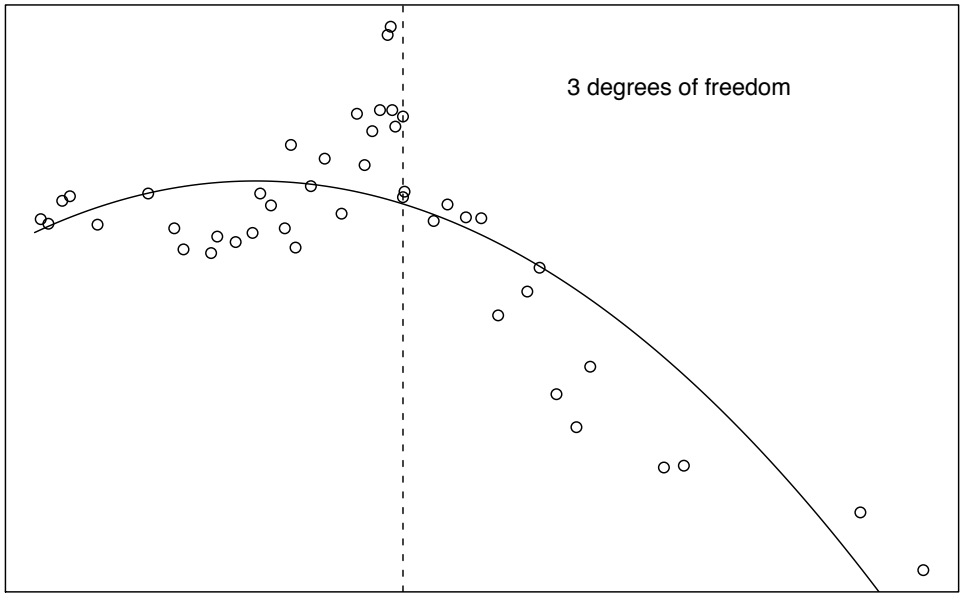


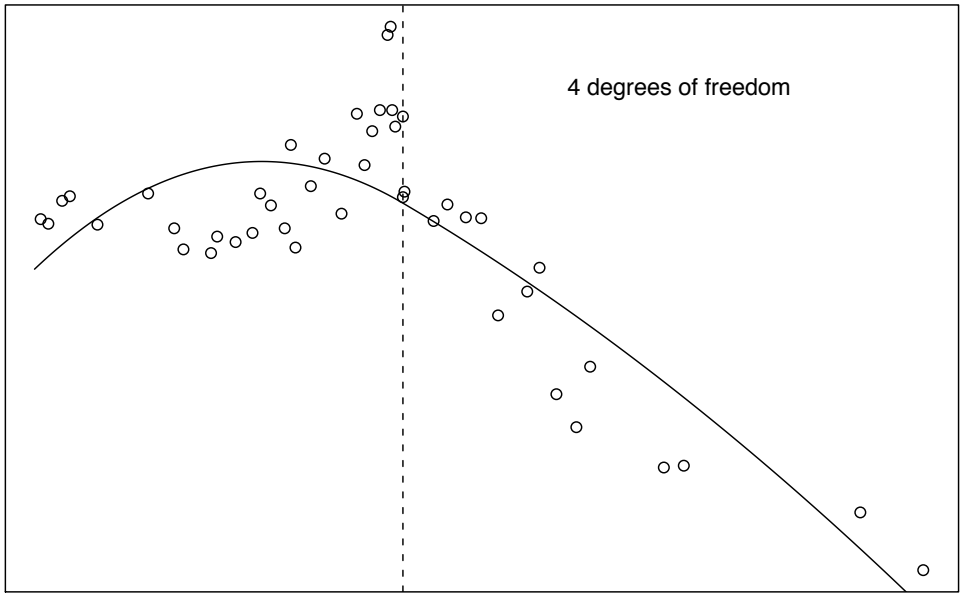


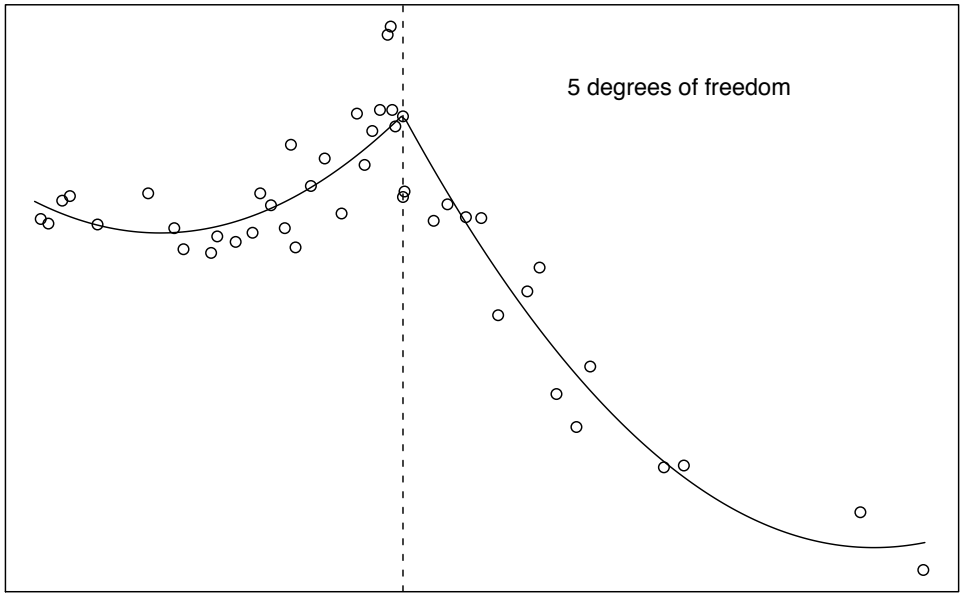


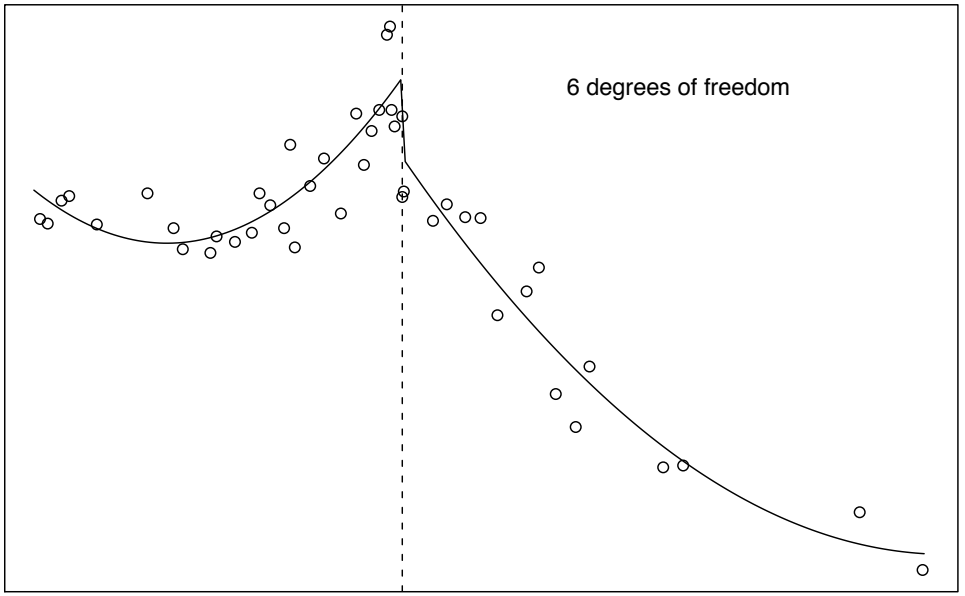





3 degrees of freedom


4 degrees of freedom


5 degrees of freedom


6 degrees of freedom

An interesting byproduct

Using the truncated power basis, we can add flexibility to regions of the curve that seem to need it

We can also “test” to see if that structure is needed; here, for example, the coefficient on `B4` tells us whether or not there is a discontinuity across the point at 67 million years ago

Note that we would typically not remove `B5` or `B6` if `B4` is important; the same is true for leaving `B6` in place if `B5` is important

```
Coefficients:
                      Estimate Std. Error  t value Pr(>|t|)
(Intercept)           7.094e-01  4.710e-04 1506.270  < 2e-16 ***
`B2: linear`         -6.423e-05  1.710e-05   -3.756 0.000564 ***
`B3: quadratic`       6.243e-07  1.529e-07    4.084 0.000213 ***
`B4: step`           -6.417e-05  2.952e-05   -2.173 0.035881 *
`B5: step-linear`    -4.377e-05  5.395e-06   -8.113  6.7e-10 ***
`B6: step-quadratic` -2.822e-07  1.918e-07   -1.471 0.149326
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 4.201e-05 on 39 degrees of freedom
Multiple R-Squared: 0.9017,   Adjusted R-squared: 0.889
F-statistic: 71.51 on 5 and 39 DF,  p-value: < 2.2e-16
```

The big question

With the continuity conditions set, we have limited our choices somewhat; the obvious remaining question is where on earth are we going to place the breakpoints?

An interesting answer

In 1979, Patricia Smith, a researcher at Old Dominion University (with funding from NASA) came up with a clever idea that, frankly, you are probably best positioned to find amusing

Suppose we are working with cubic splines: Starting with a cubic polynomial model (4 degrees of freedom) $1, x, x^2, x^3$ we want to add a single breakpoint; that is, we want to add a new basis function of the form $(x-t)^3_+$

An interesting answer

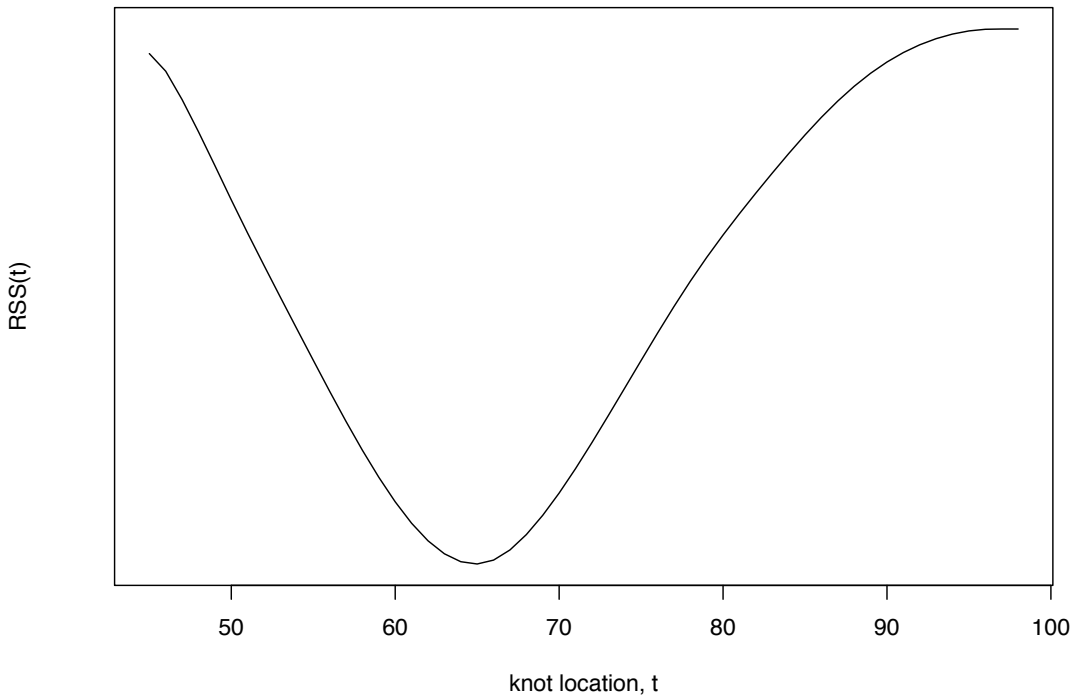
For each value of t, we can fit an ordinary regression; we can then associate with each value of t the error $RSS(t)$

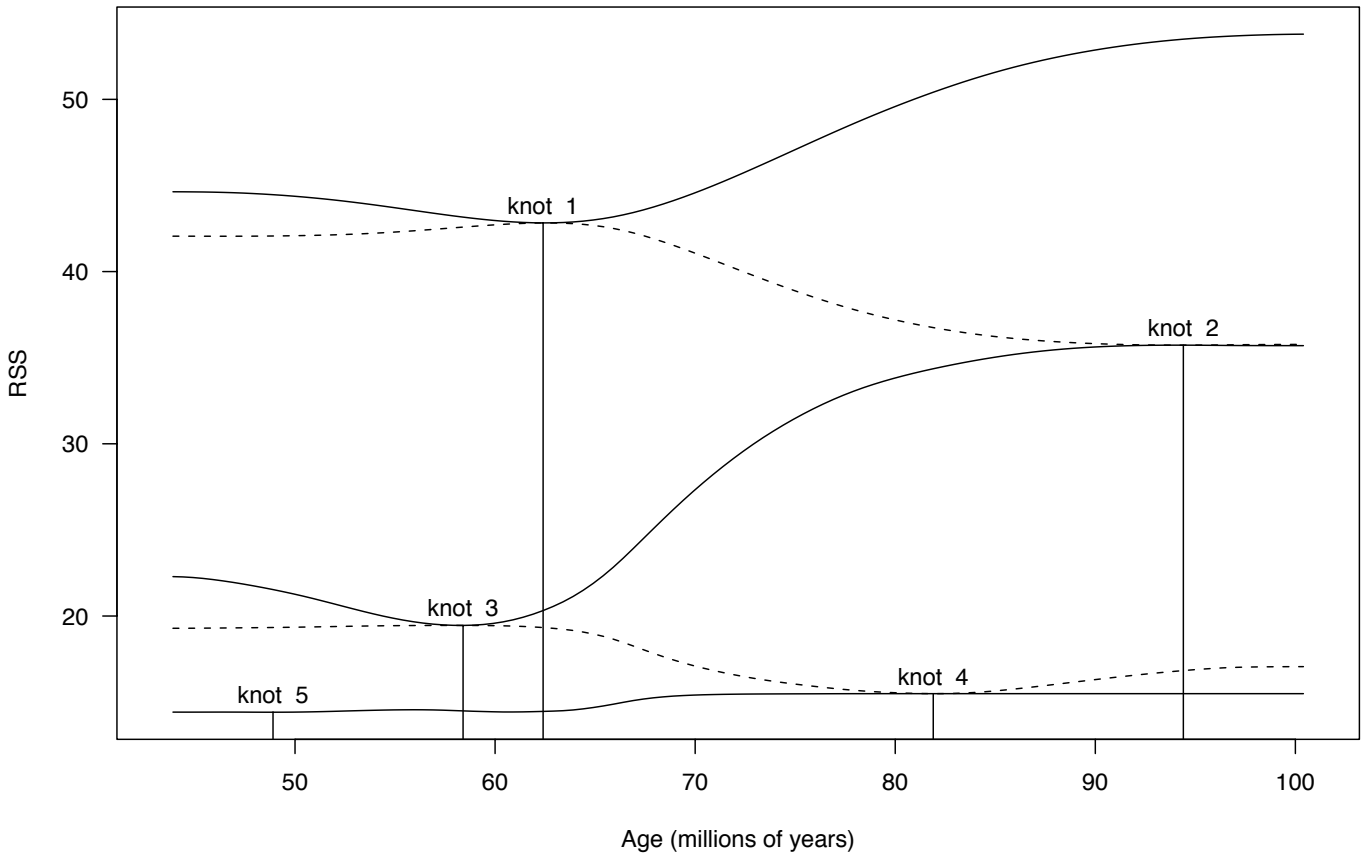
We could then find the location that minimizes $RSS(t)$; in this case it occurs at $t = 65$

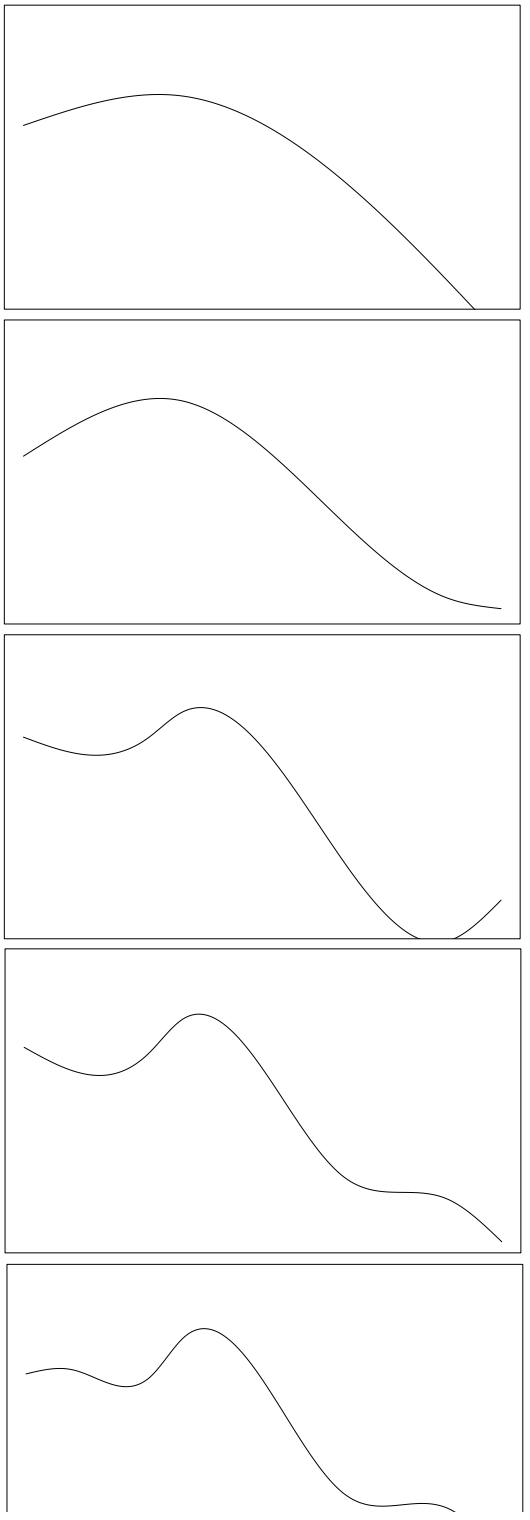
Having picked the first knot, we can then start with the basis

$$1, x, x^2, x^3, (x-65)^3_+$$

and find the best basis to add in the form $(x-t)^3_+$, again using $RSS(t)$ as a guide




RSS
50
40
30
20
knot 1
knot 2
knot 3
knot 4
knot 5
50
60
70
80
90
100
Age (millions of years)



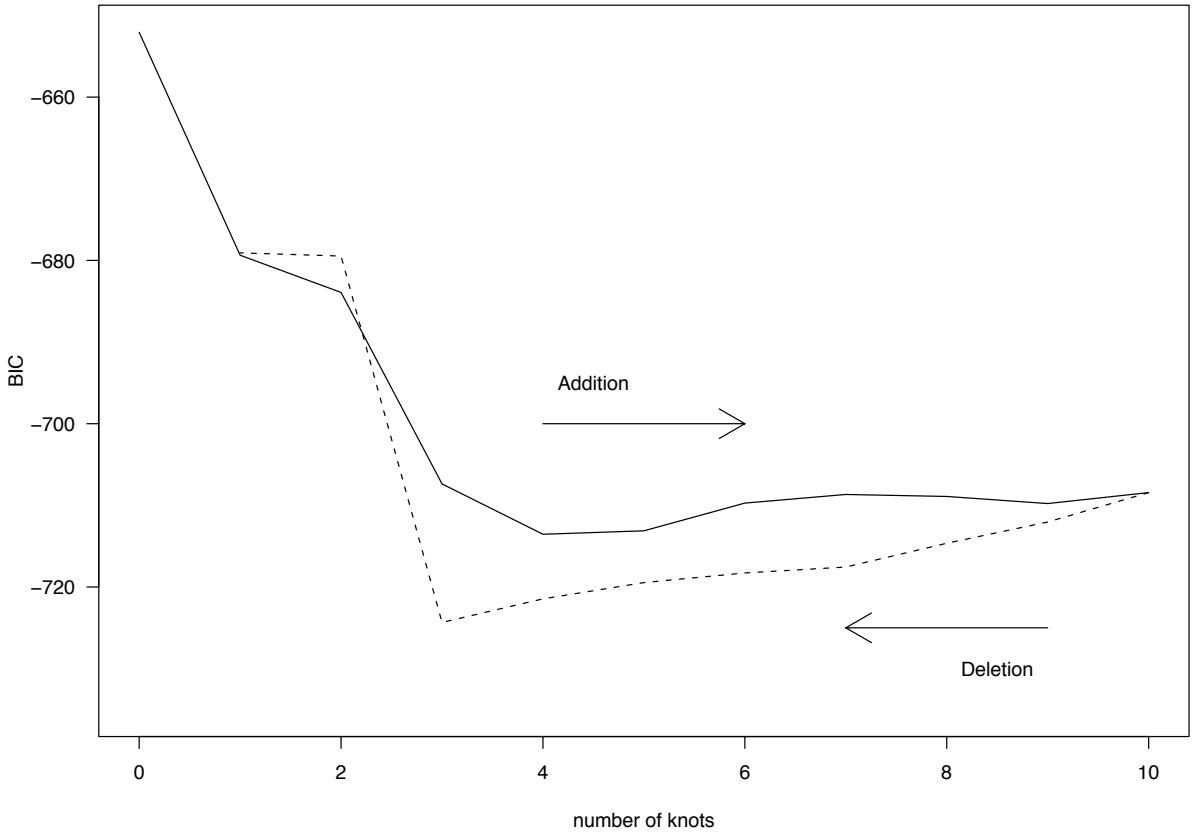
What does this remind you of?

The big idea

Right. Variable selection, only now our “variables” map directly to features of the regression curve

Patricia Smith suggested you perform stepwise addition of knots, followed by backward deletion and then use a selection rule like AIC or BIC to pick the model

Here, we decide that three knots is about right



Adaptive regression splines

The procedure I've just outlined was studied extensively in the 80s and 90s, with different strategies for candidate selection (all subsets versus forward/backward) and different evaluation criterion

But morally, the approach involves Smith's great idea that you can tie the behavior of a smoother to well-known regression tools; that the truncated power basis has a very special role in spline modeling

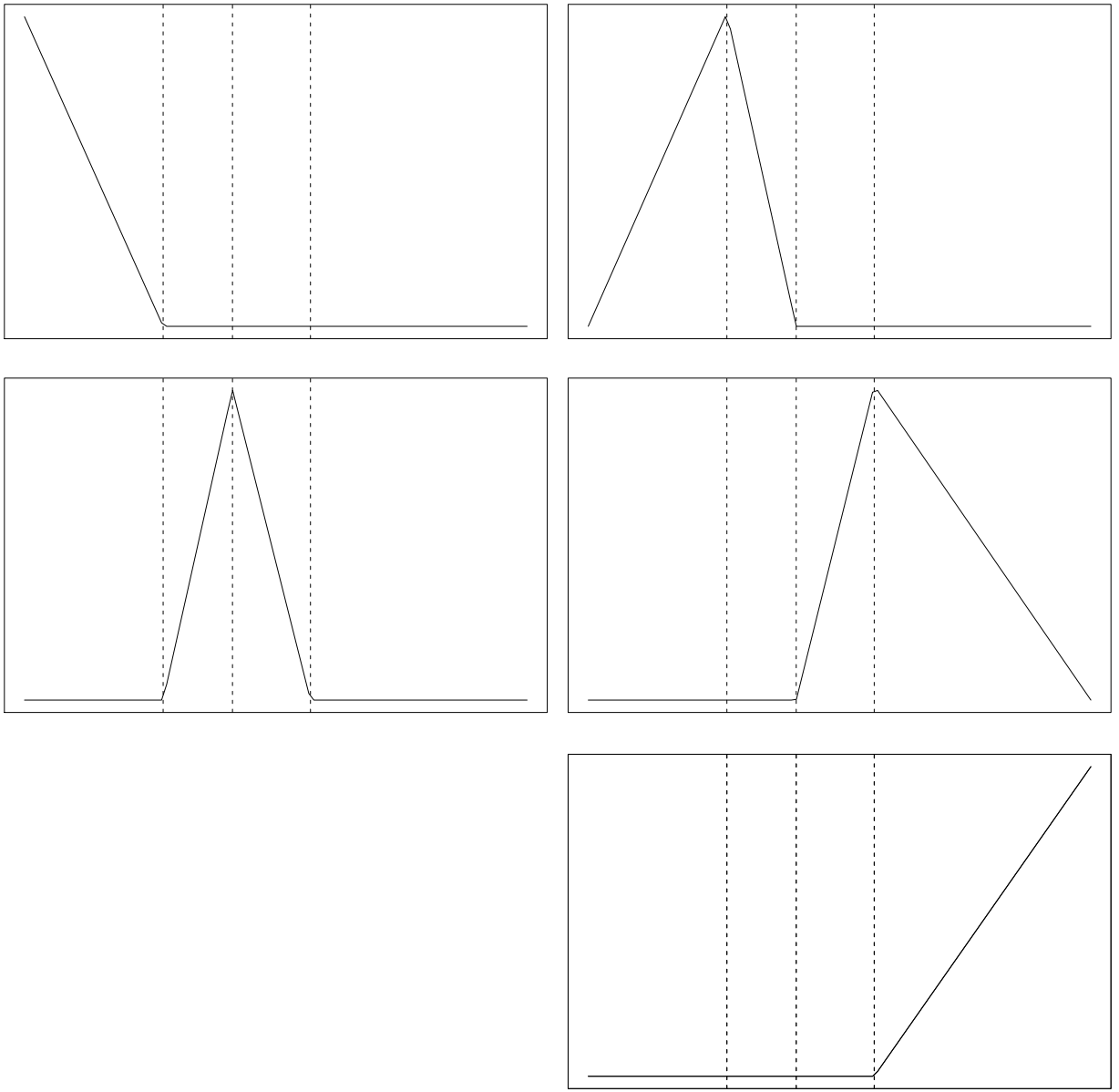
That said, it is not the only basis representation...

Basis choice

Remember, rather than trying to solve a "constrained" least squares problem in which we force smoothness constraints it's easier to work with basis functions that satisfy the conditions we're after

At the right we show five basis functions that can be used in a fitting routine to create the broken-stick model on slide 25

This is called the **B-spline basis**



B-splines

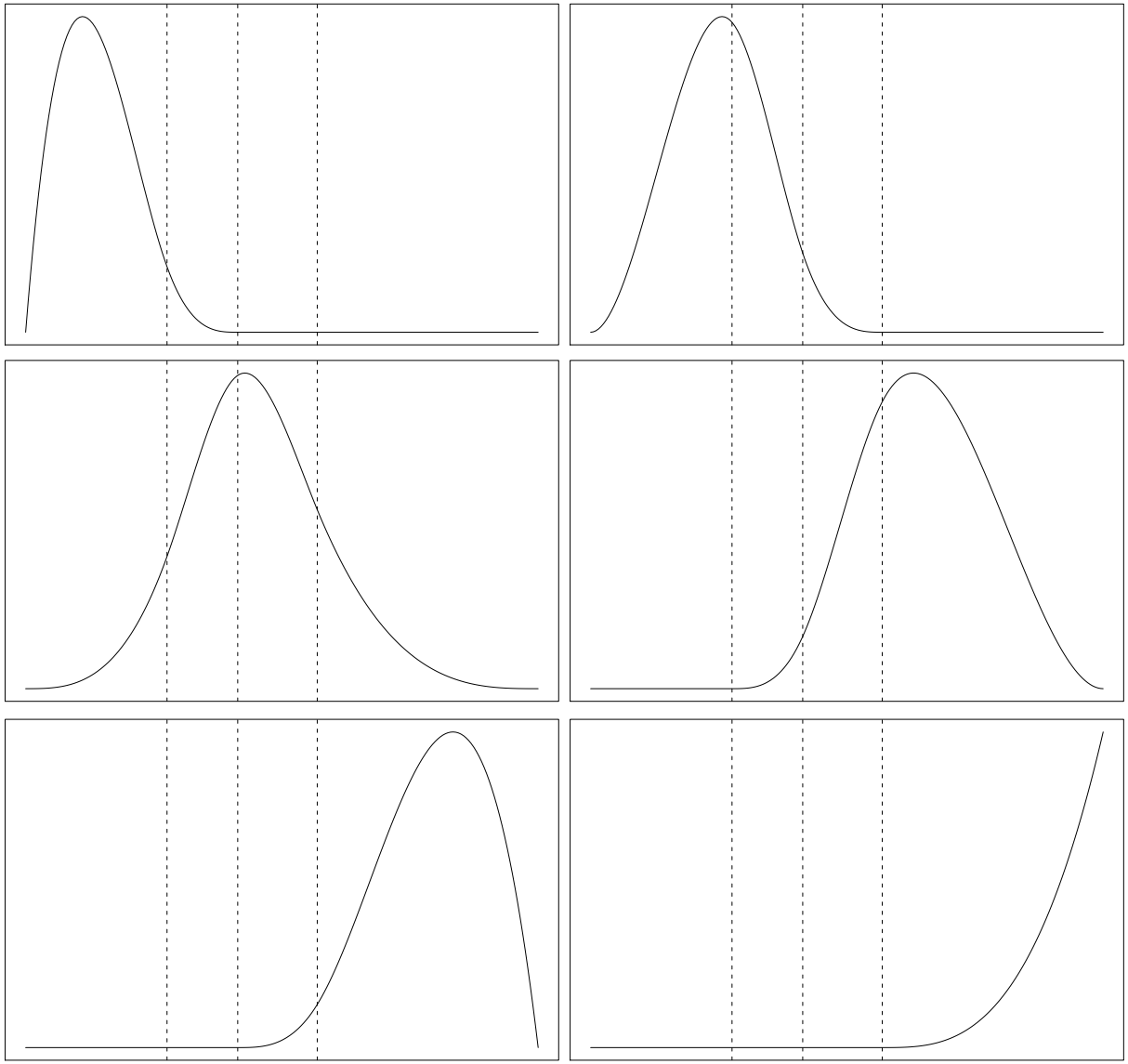
An obvious down-side to the truncated power basis is that it is just that, a series of truncated powers; all of the instability we associated with polynomials (that led us to orthogonal polynomials) applies for this basis

As an alternative, B-splines emerged in the late 60s and early 70s as a computationally convenient basis; it has good numerical properties, stemming in part from the fact that each basis function is “localized”

```
# load up a new library devoted entirely to splines (!)

library(splines)

# then create the b-spline basis functions for a space of
# quadratics with knots at 59, 67 and 76

x = seq(43,101,len=200)
b = bs(x,deg=3,knots=c(59,67,76))

# plot em!

plot(x,b[,1],type="l")
abline(v=c(59,67,76),lty=2)

plot(x,b[,2],type="l")
abline(v=c(59,67,76),lty=2)

plot(x,b[,3],type="l")
abline(v=c(59,67,76),lty=2)

plot(x,b[,4],type="l",axes=F,xlab="",ylab="")
abline(v=c(59,67,76),lty=2)

plot(x,b[,5],type="l",axes=F,xlab="",ylab="")
abline(v=c(59,67,76),lty=2)

plot(x,b[,6],type="l",axes=F,xlab="",ylab="")
abline(v=c(59,67,76),lty=2)
```

B-splines

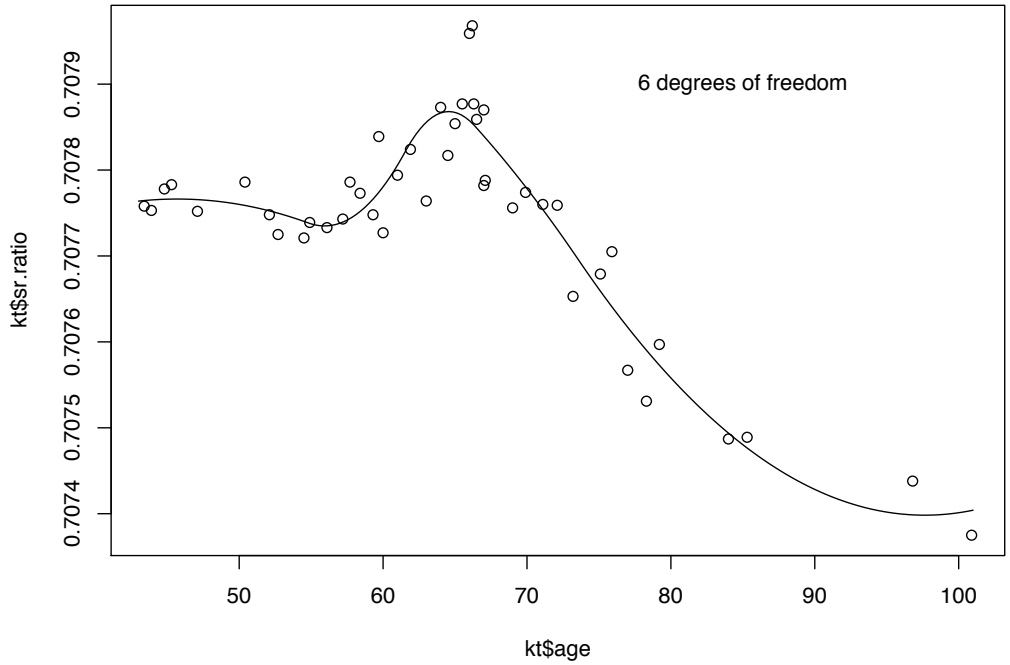
Here we present the basis for the quadratic B-splines (piecewise quadratic functions that are continuously differentiable); what does their shape remind you of?

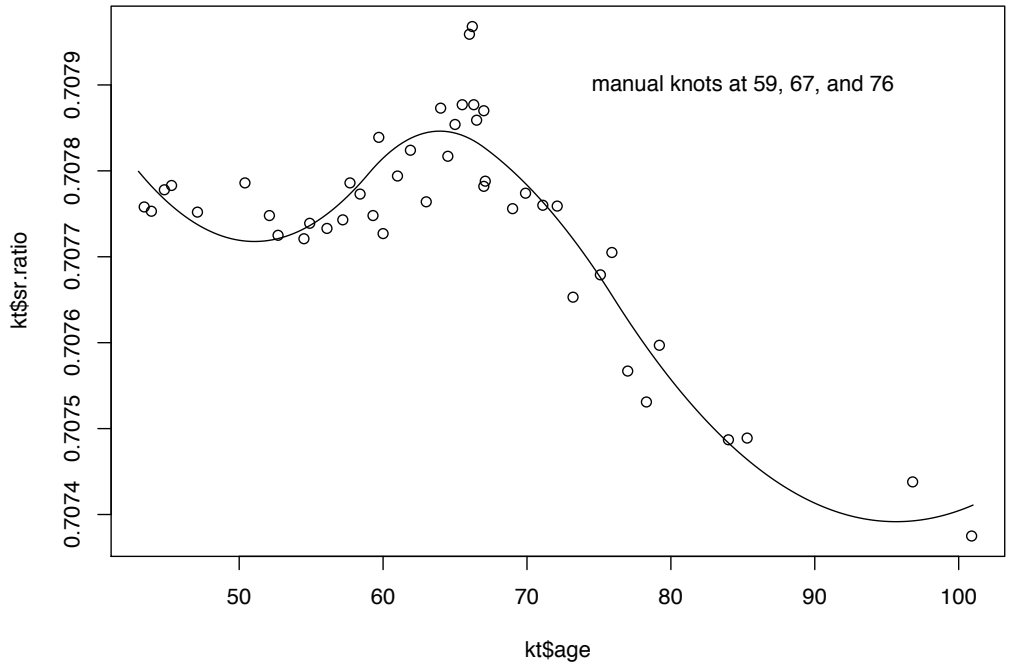


Modeling with B-splines

R has a number of routines for working with splines that are as easy as using polynomials

You can either specify the number of degrees of freedom you would like (and R places knots at the right number of quantiles of the data - i.e. “evenly” spaced knots); or you can place them manually



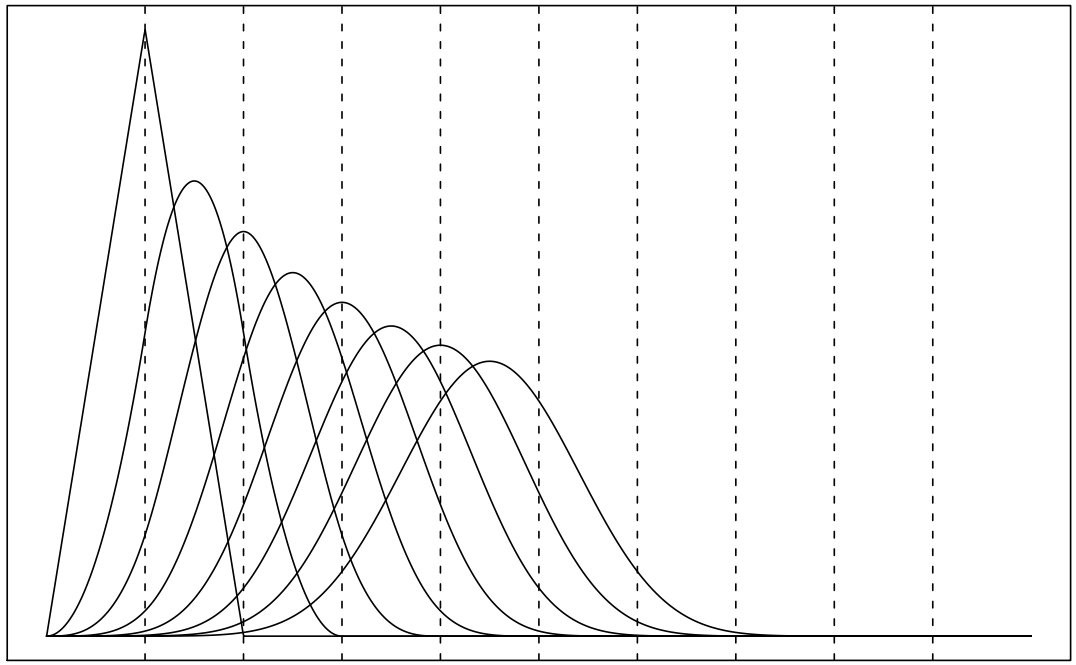


```
library(splines)

newkt = data.frame(age=seq(43,101,len=200))

# set degrees of freedom for evenly spaced knots

plot(kt$age,kt$sr.ratio)

fit = lm(sr.ratio~bs(age,deg=2,df=6),data=kt)
lines(newkt$age,predict(fit,newdata=newkt))
text(85,0.7079,"6 degrees of freedom")

# or place them yourself

plot(kt$age,kt$sr.ratio)

fit = lm(sr.ratio~bs(age,deg=2,knots=c(59,67,76)),data=kt)
lines(newkt$age,predict(fit,newdata=newkt))
text(85,0.7079,"manual knots at 59, 67, and 76")
```

B-splines

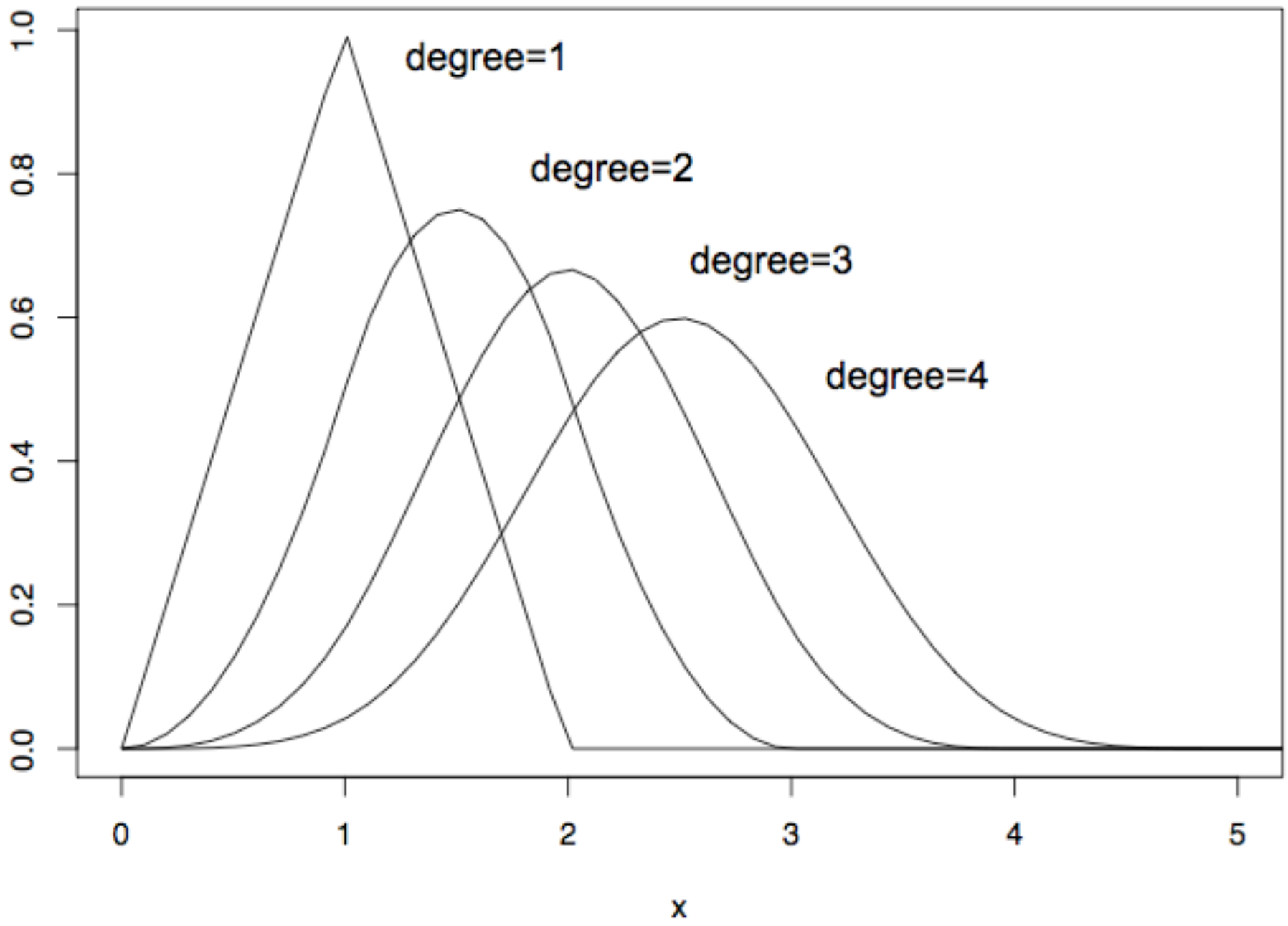
I leave you with a question...

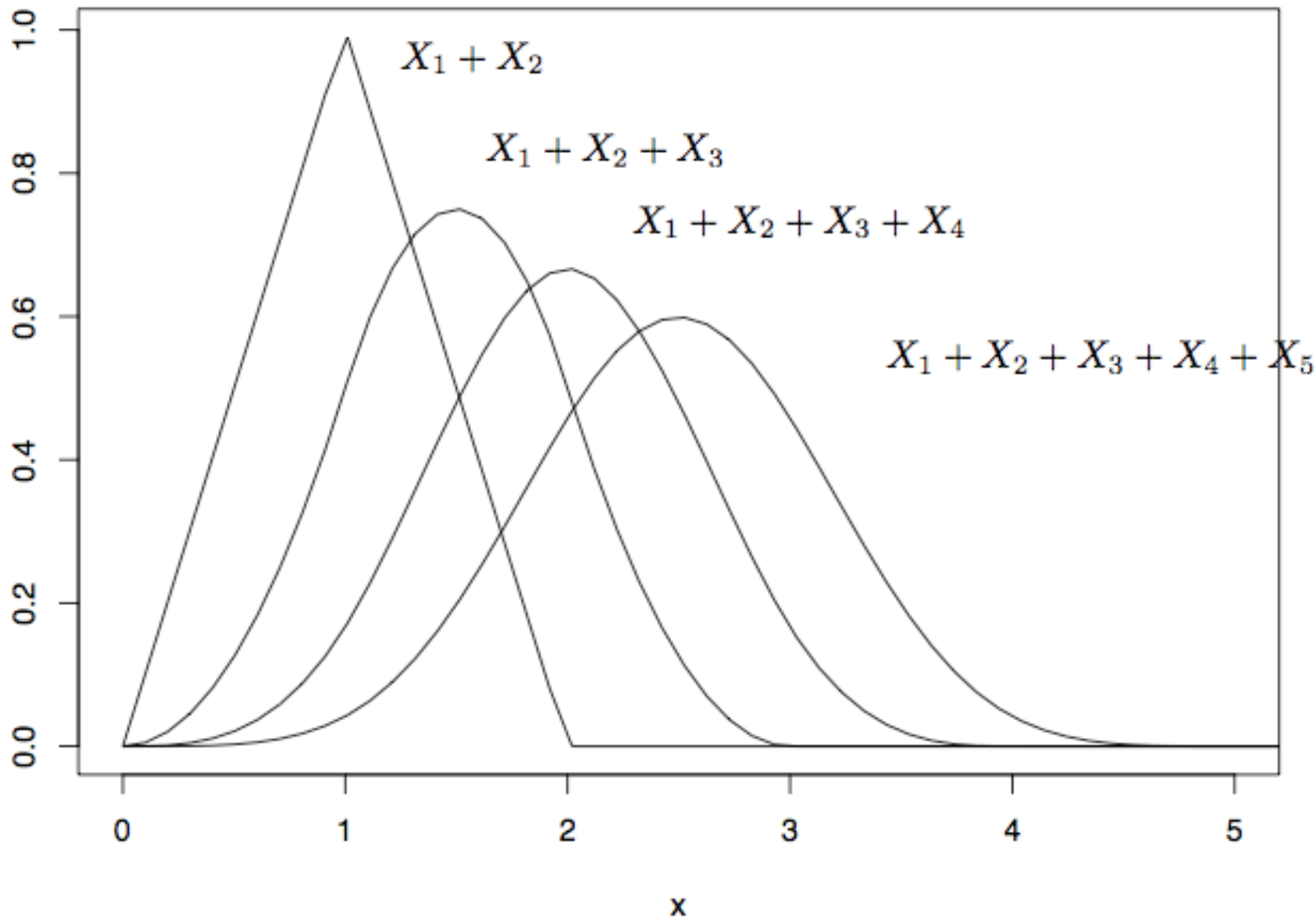
At the right we have a series of spline basis functions, each associated with a higher and higher degree space; the first corresponds to linear splines, then quadratic, the cubic and so on



Each space has equally spaced knots as indicated by the dashed vertical lines; what can you tell me about these B-spline “bumps” as the degree of the space gets larger?

Have we seen this before?


degree=1
degree=2
degree=3
degree=4
0.0
0.2
0.4
0.6
0.8
1.0
0
1
2
3
4
5
x


$X_1 + X_2$
$X_1 + X_2 + X_3$
$X_1 + X_2 + X_3 + X_4$
$X_1 + X_2 + X_3 + X_4 + X_5$
0.0
0.2
0.4
0.6
0.8
1.0
0
1
2
3
4
5
x

Model selection

No matter which basis function set we are using, the essential character of the fit proposed by Smith involved adding knots one at a time, presumably introducing flexibility in regions where we need it as we go

There is, of course, a question of when we stop this addition (or addition and deletion or deletion) process -- We could appeal to the typical selection criterion but there are complications

Despite our ability to make it look like variable selection (with special basis functions), when is it different? Here is a simple simulation...

Simulation

Let $x_i = i/100$, i=1,..., 100 and set $y_i = x_i + \epsilon_i$ where ϵ_i has a standard normal distribution -- Let RSS_0 denote the residual sum of squares fitting with the variables 1 and x and let RSS(t) denote the residual sum of squares after fitting with the variables 1, x and $(x - t)_+$

What can you say about $RSS_0 - RSS(t)$ for a fixed knot location t?

Simulation

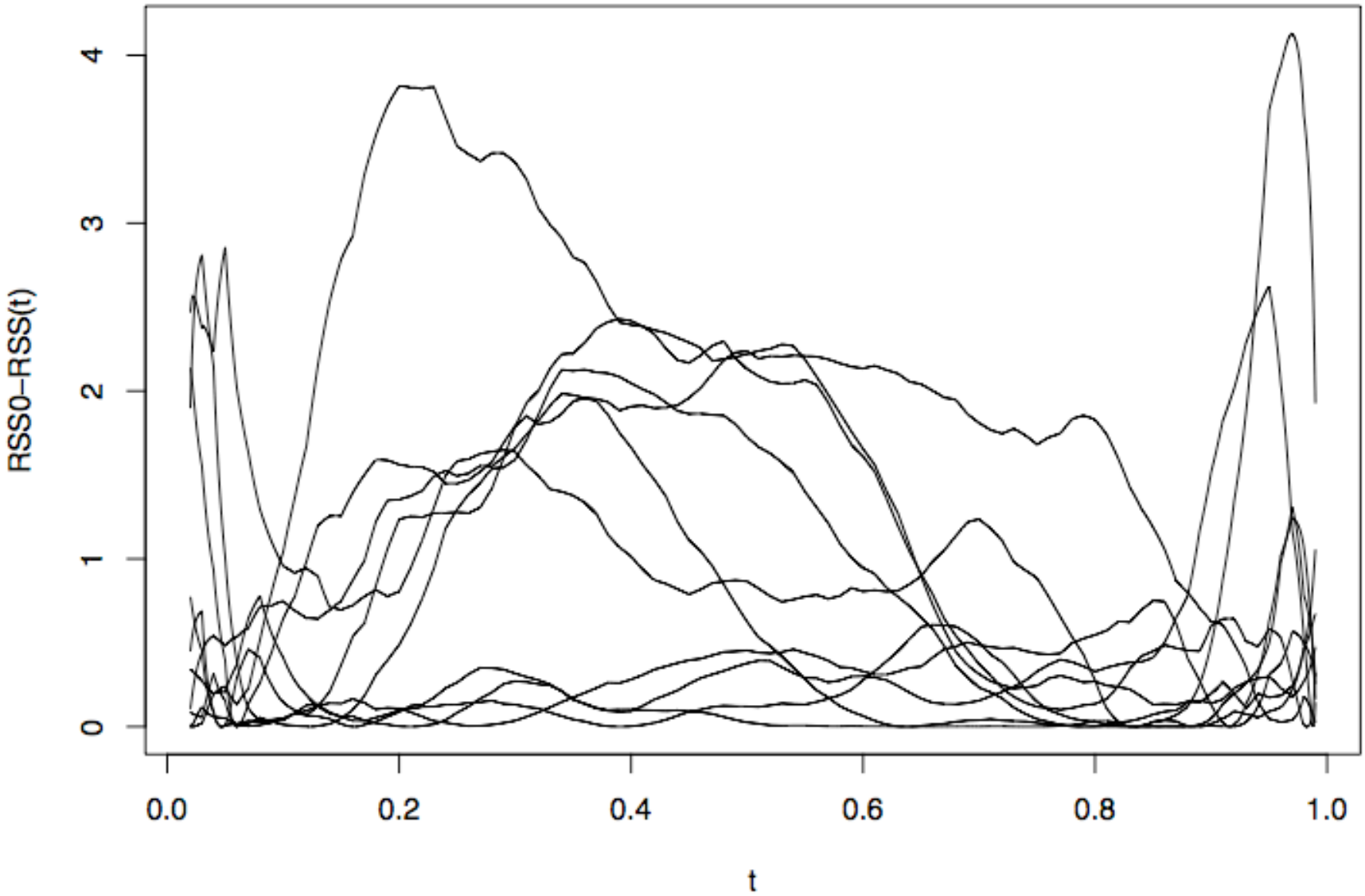
Technically, $\mathrm{RSS}_0 - \mathrm{RSS}(t)$ should have a chi-square distribution with one degree of freedom -- Now, when we search for a knot, we are considering a number of t's and not just one

That is, our interest is in $V = \max_t[\mathrm{RSS}_0 - \mathrm{RSS}(t)]$

What has changed? To get a sense of it, let's simulate! We constructed 5,000 data sets and for each computed a value of V

Simulation

Traces of $\mathrm{RSS}_0 - \mathrm{RSS}(t)$ as a function of t for 10 of the 5,000 simulations

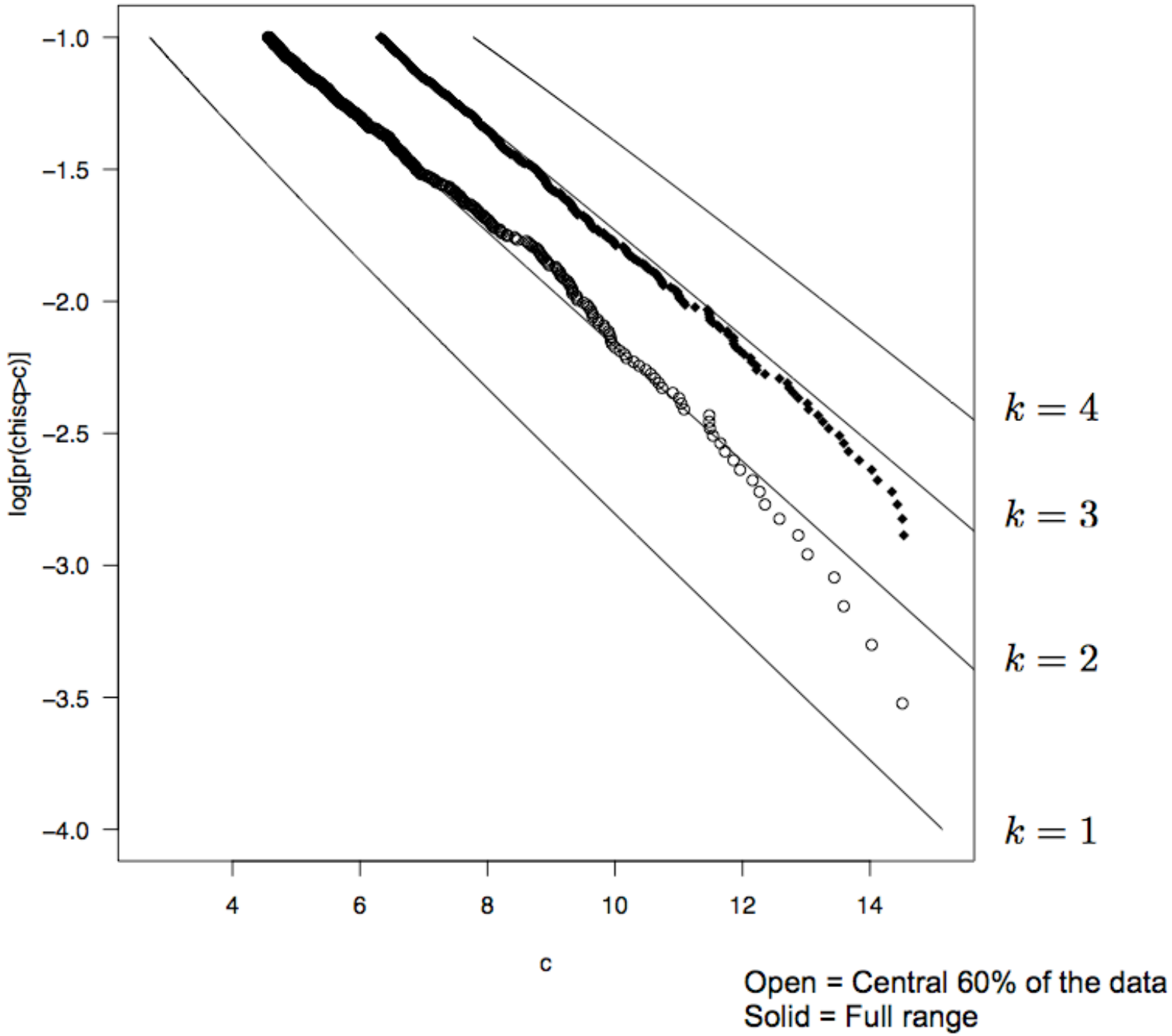


Simulation

To get a sense of what's changed, we now compare the 5,000 values of V to the quantiles of various chi-square distributions

We've actually computed two values of V -- One where t is allowed to range over the entire input domain, [0,1], and the other for which we are searching just the central 60% of the data, [0.2,0.8]

What do you see?


−1.0
−1.5
−2.0
−2.5
−3.0
−3.5
−4.0
log[pr(chisq>c)]
4
6
8
10
12
14
c
$k = 4$
$k = 3$
$k = 2$
$k = 1$
Open = Central 60% of the data
Solid = Full range

The upshot

Because of their roots in model selection, many of the rules for determining how many knots to include depend on the quantity V -- If we assume that the error variance is known, for example, then AIC is

Therefore, we want to keep a single knot using AIC providing

$$\frac{\mathrm{RSS}_0}{n} + \frac{2 \times 2\sigma^2}{n} - \left(\frac{\mathrm{RSS}(t)}{n} + \frac{3 \times 2\sigma^2}{n} \right) = \frac{V}{n} - \frac{2\sigma^2}{n} > 0$$

or V > 2 (because we assumed $\sigma^2 = 1$)

The upshot

When V is roughly chi-square with one degree of freedom, this has a Type I error of 16% -- For two degrees of freedom, this is 37% and for three it's 60%

For these reasons, people tend to employ larger penalties for these applications with BIC being typical -- For more on this, see Guo and Wahba (1997) and Owen (1990)

Univariate smoothing splines

Suppose we are given n pairs (x_i, y_i) , where the x_i are sorted in increasing order so that $x_1 < x_2 < \ldots < x_n$

Wahba and Silverman each consider the solution to a so-called variational problem; that is, within some class of "smooth" functions (more later), find the function f that minimizes

$$\frac{1}{n}\sum_{i=1}^{n}(y_i - g(x_i))^2 + \lambda \int_{-\infty}^{\infty} (g''(u))^2 du$$

We recognize this as a penalized regression problem, where the penalty is of a form you're probably not very familiar with

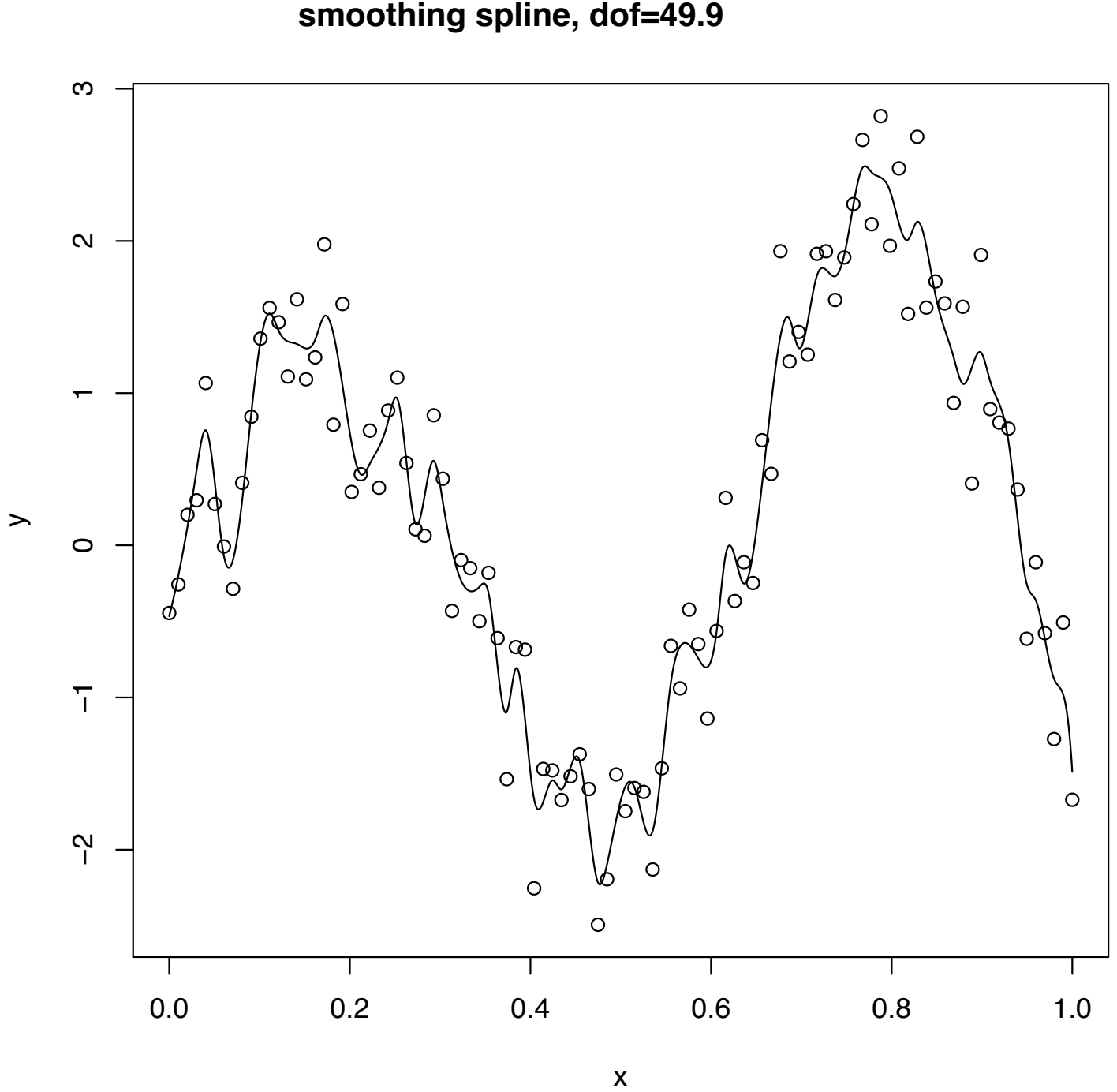
Univariate smoothing splines

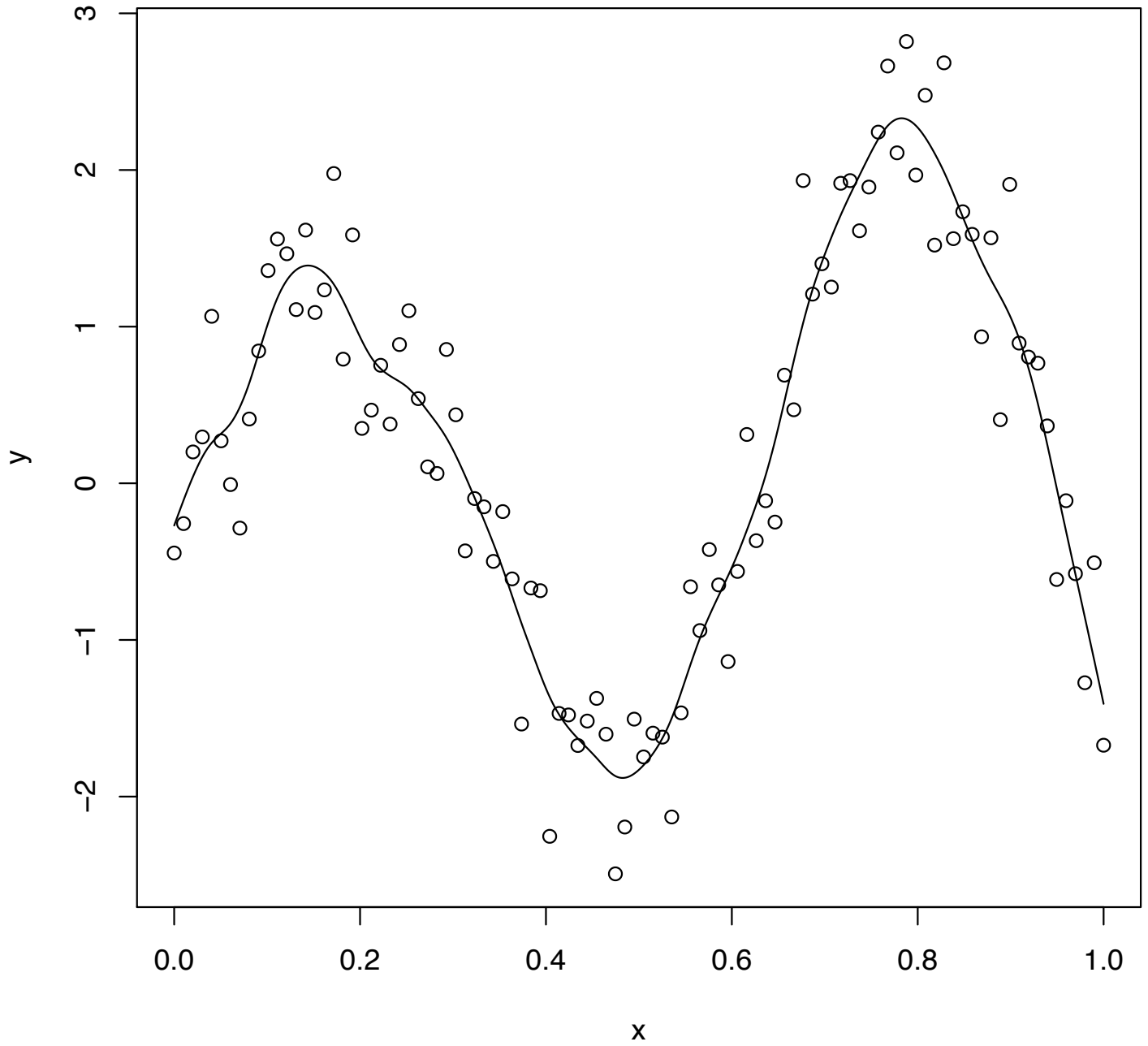
Remarkably, this problem has a closed-form solution; assuming that the ordinary regression problem (linear) has a unique solution, then the penalized criterion has a minimizer of the form

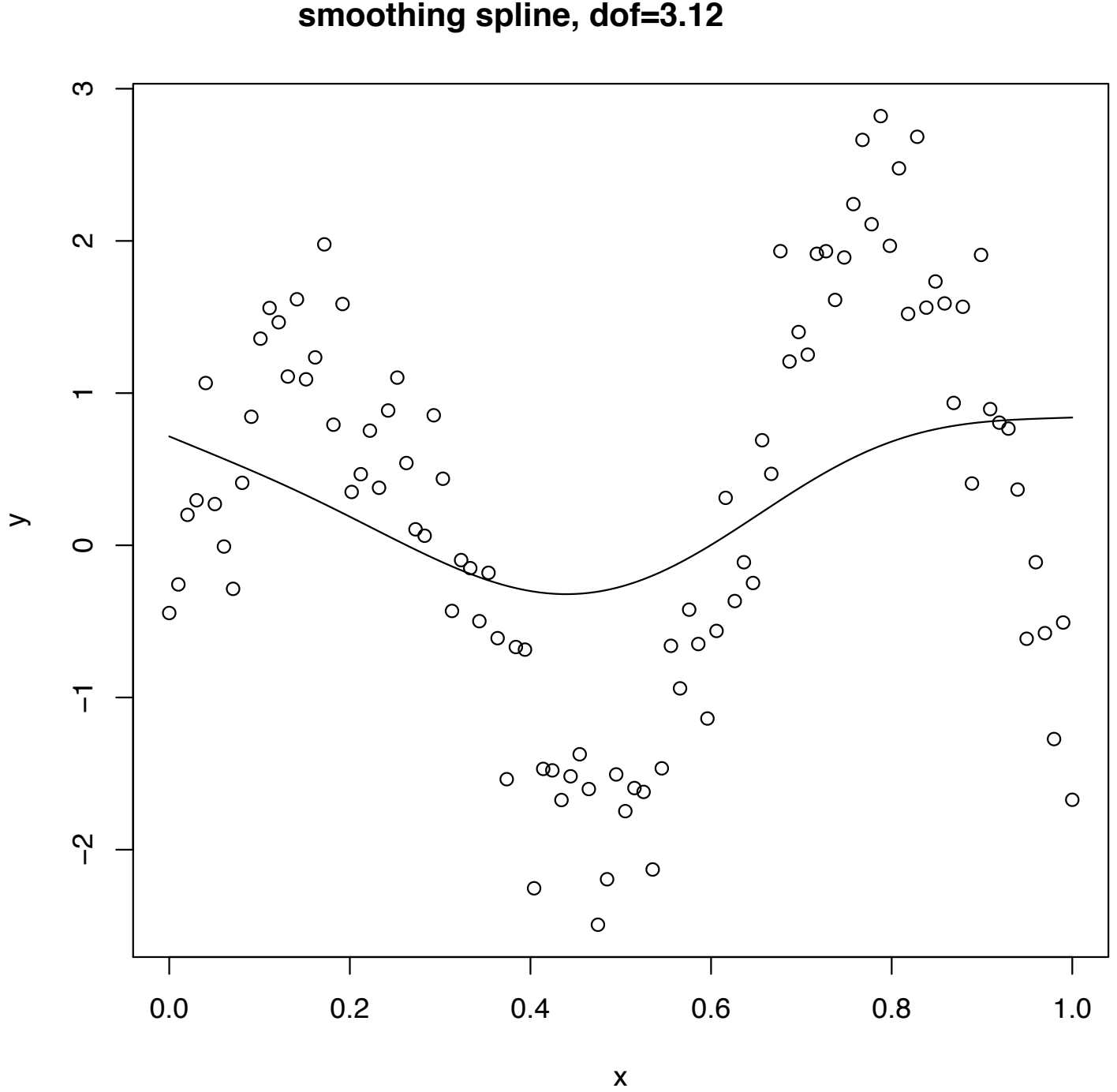
$$\widehat{f}(x) = \begin{cases} \text{linear polynomial} & x \in (-\infty, x_1] \\ \text{cubic polynomial} & x \in [x_j, x_{j+1}] \\ \text{linear polynomial} & x \in [x_n, \infty) \end{cases}$$

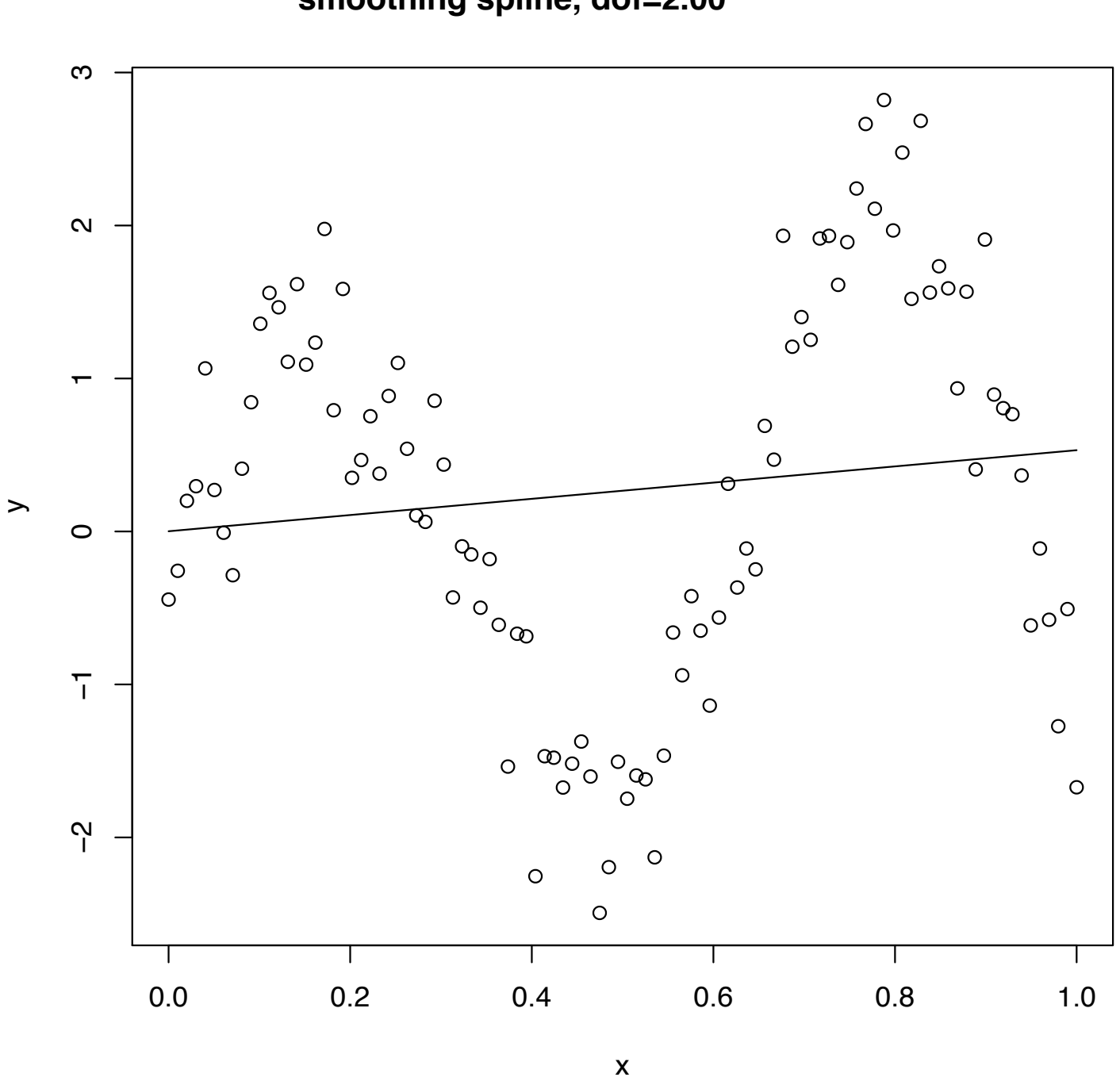
Further, $\widehat{f}(x)$ has two continuous derivatives; so, piecewise cubic polynomial, smooth across the knots -- the solution is a cubic spline with breaks at each of the data points!

```
n = 100

x = seq(0,1,len=n)
f = sin(10*x)*exp(x)

y = f+rnorm(length(x),sd=0.5)

xnew = seq(0,1,len=1000)

fit = smooth.spline(x,y,spar=0.2)
plot(x,y,main=paste("smoothing spline, dof=",fit$df,sep=""))
lines(predict(fit,xnew))

fit = smooth.spline(x,y,spar=0.5)
plot(x,y,main=paste("smoothing spline, dof=",fit$df,sep=""))
lines(predict(fit,xnew))

fit = smooth.spline(x,y,spar=1)
plot(x,y,main=paste("smoothing spline, dof=",fit$df,sep=""))
lines(predict(fit,xnew))

fit = smooth.spline(x,y,spar=2)
plot(x,y,main=paste("smoothing spline, dof=",fit$df,sep=""))
lines(predict(fit,xnew))
```


smoothing spline, dof=49.9
y
x
3
2
1
0
−1
−2
0.0
0.2
0.4
0.6
0.8
1.0


smoothing spline, dof=17.8
y
x
3
2
1
0
−1
−2
0.0
0.2
0.4
0.6
0.8
1.0


smoothing spline, dof=3.12
y
x
3
2
1
0
−1
−2
0.0
0.2
0.4
0.6
0.8
1.0


smoothing spline, dof=2.00
y
x
3
2
1
0
−1
−2
0.0
0.2
0.4
0.6
0.8
1.0

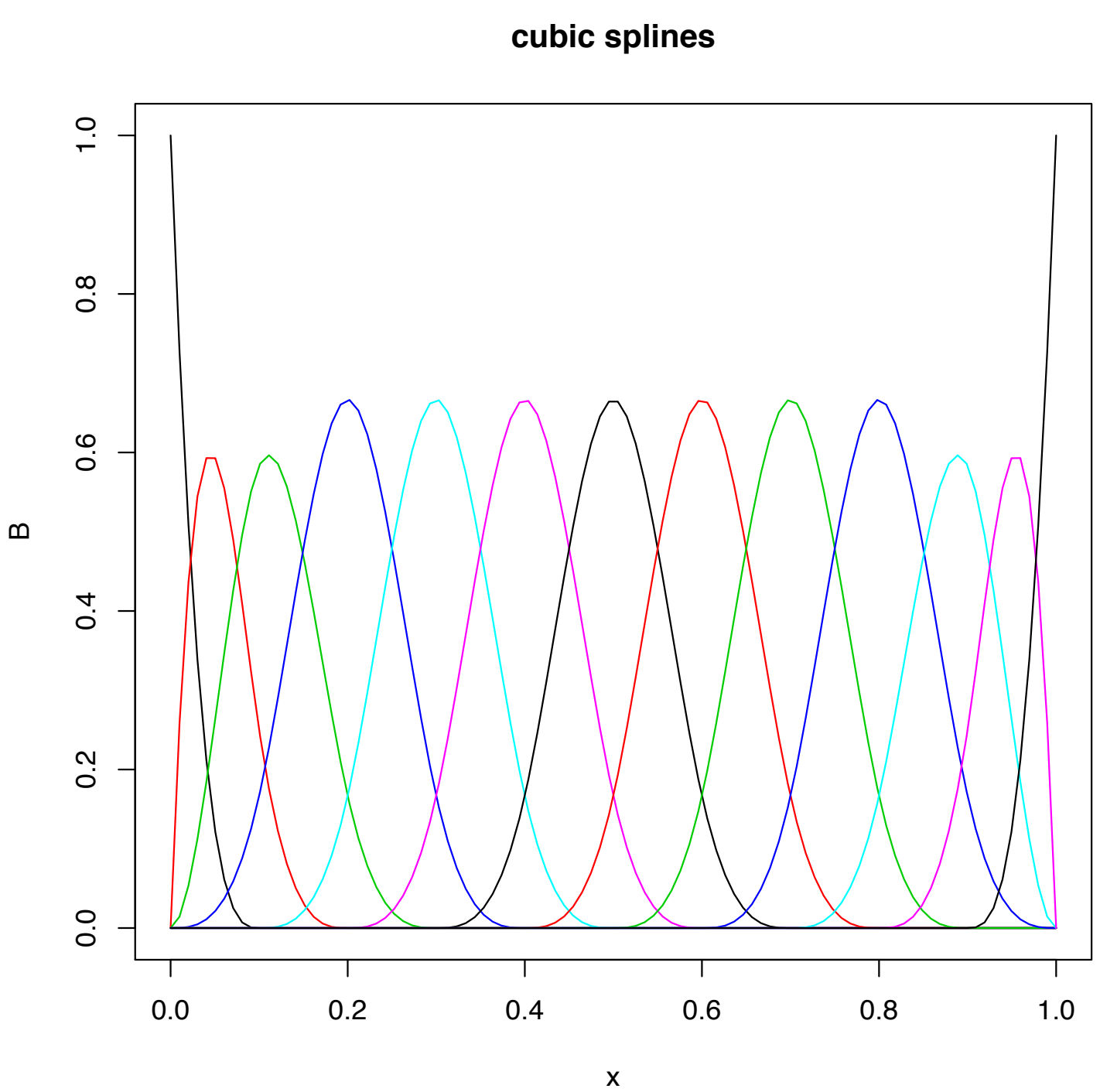
Basis set

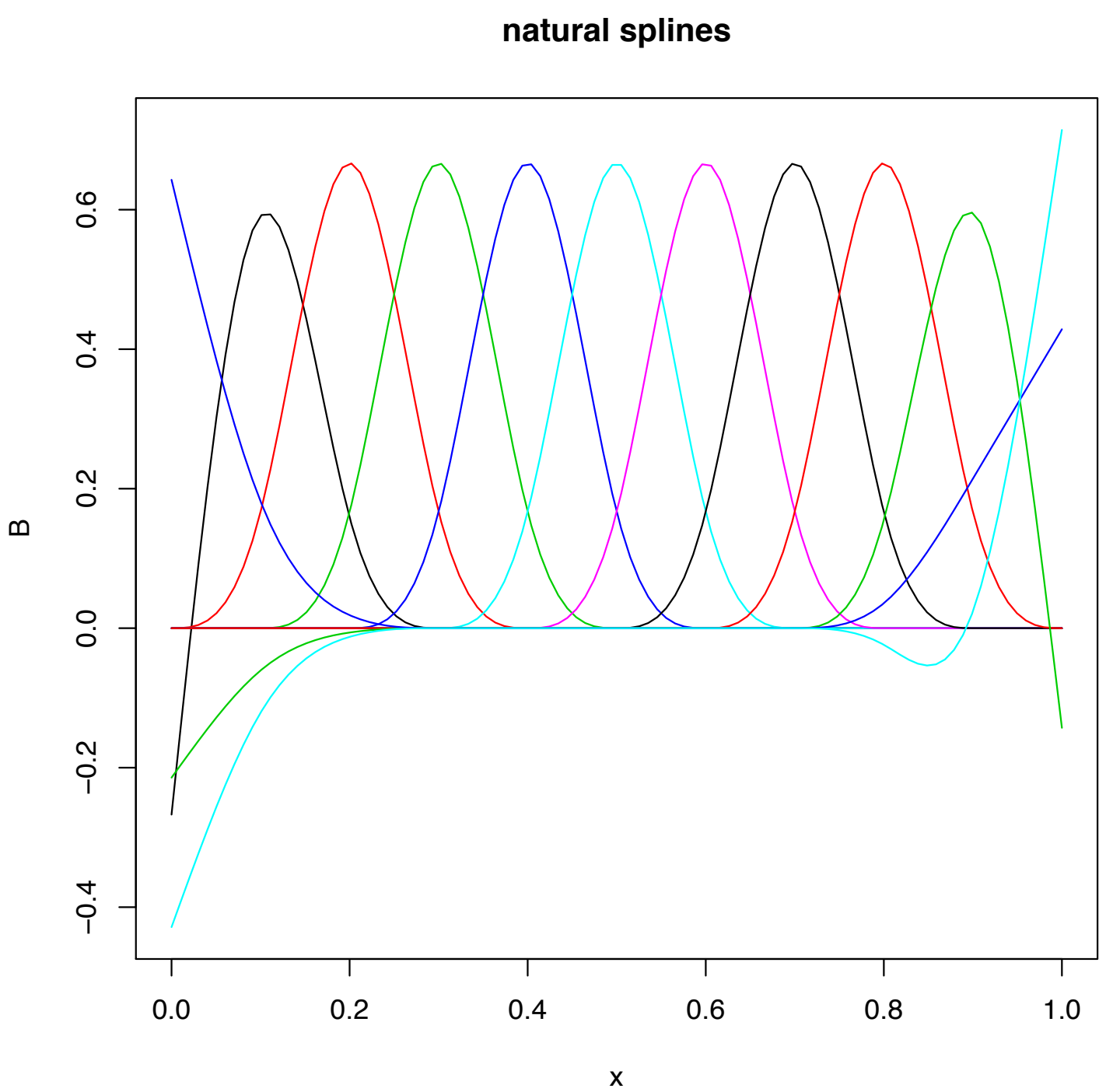
The solution to the smoothing spline problem is a little different than a simple spline space; beyond the first and last knot, the function is a line -- this means that at the first and last knots we have

$$\widehat{f}''(x_1) = \widehat{f}''(x_n) = 0$$

This is often referred to as **a tail-linear constraint**; the resulting collection of functions (twice continuously differentiable, cubic in each interval defined by adjacent knots and linear beyond the first and last knot) is again a linear space known as **the natural splines**

```
library(splines)

x = seq(0,1,len=100)

B = ns(x,knots=seq(0.1,0.9,by=0.1),intercept=T)   # include the intercept
matplot(x,B,type="l",lty=1)
dim(B)  #  100x11

B = bs(x,knots=seq(0.1,0.9,by=0.1),intercept=T)   # include the intercept
matplot(x,B,type="l",lty=1)
dim(B)  #  100x13
```


cubic splines
B
x
0.0
0.2
0.4
0.6
0.8
1.0


natural splines
B
x
0.0
0.2
0.4
0.6
0.8
1.0
−0.4
−0.2
0.0
0.2
0.4
0.6

Some linear algebra

Let $B_1, \ldots, B_n$ denote the basis for the associated space of natural splines so that we can write $g(x) = \beta_1 B_1 + \cdots + \beta_n B_n$

Next, we define two matrices $\mathbf{B}$ and Λ such that

$$[\mathbf{B}]_{ij} = B_j(x_i) \quad \text{and} \quad [\Lambda]_{ij} = \int_0^1 B_i''(x) B_j''(x) dx$$

Now, we can rewrite our original penalized expression in matrix form as

$$(\boldsymbol{y} - \mathbf{B}\boldsymbol{\beta})^t (\boldsymbol{y} - \mathbf{B}\boldsymbol{\beta}) + \lambda \boldsymbol{\beta}^t \Lambda \boldsymbol{\beta}$$

where we have also set $\boldsymbol{y} = (y_1, \ldots, y_n)$ and $\boldsymbol{\beta} = (\beta_1, \ldots, \beta_n)$

The mechanics of a smoothing spline

We'll use this setup to consider what happens when we increase the penalty in a smoothing spline fit; there's a little linear algebra along the way, but it's not too bad

Most of this was worked out by Reinsch in the late 60s and early 70s; this construction, however, will be useful in a variety of linear smoothing problems

Some linear algebra

Solving this system (differentiating with respect to the elements in the coefficient vector $\boldsymbol{\beta}$) we find the solution

$$\left(\mathbf{B}^t\mathbf{B} + \lambda\Lambda\right)\widehat{\boldsymbol{\beta}} = \mathbf{B}^t\boldsymbol{y} \quad \text{or} \quad \widehat{\boldsymbol{y}} = \mathbf{B}\left(\mathbf{B}^t\mathbf{B} + \lambda\Lambda\right)^{-1}\mathbf{B}^t\boldsymbol{y}$$

We can work with this form slightly; recalling that $\mathbf{B}$ is a square, invertible matrix we have

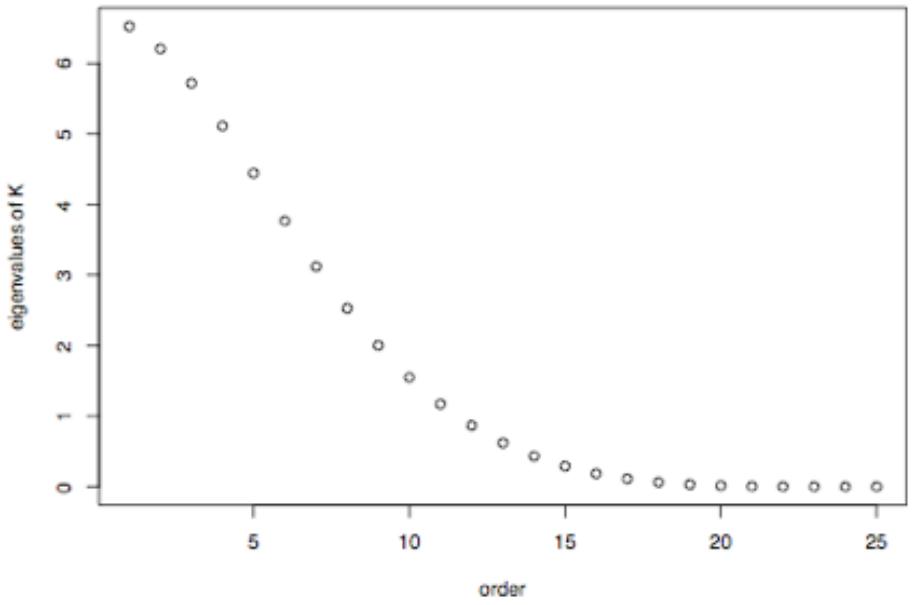
$$\begin{aligned}
\widehat{\boldsymbol{y}} &= \mathbf{B}\left(\mathbf{B}^t[\boldsymbol{I} + \lambda\mathbf{B}^{-t}\Lambda\mathbf{B}^{-1}]\mathbf{B}\right)^{-1}\mathbf{B}^t\boldsymbol{y} \\
&= \left(\boldsymbol{I} + \lambda\mathbf{B}^{-t}\Lambda\mathbf{B}^{-1}\right)^{-1}\boldsymbol{y} \\
&= (\boldsymbol{I} + \lambda\boldsymbol{K})^{-1}\boldsymbol{y}
\end{aligned}$$

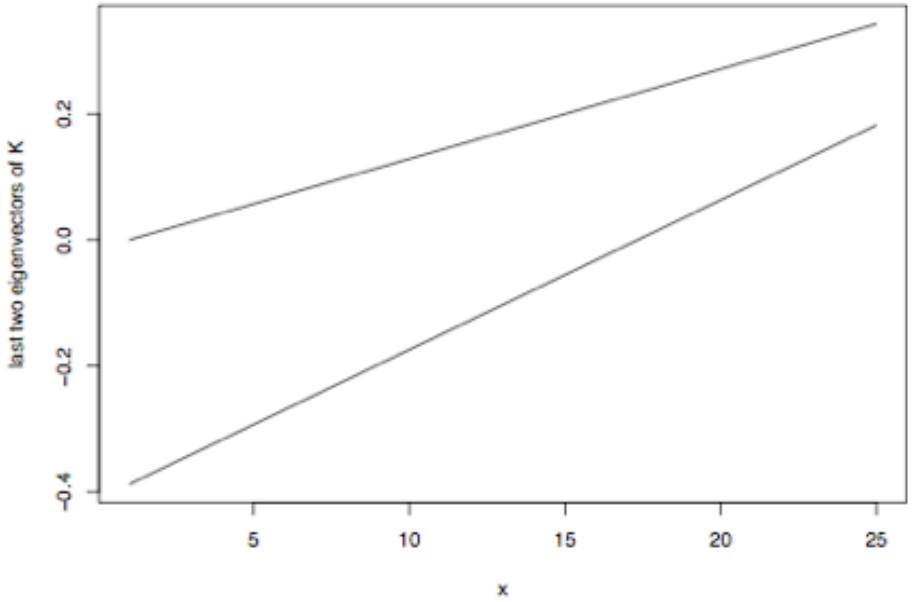
A little more linear algebra

Let's consider the eigendecomposition of the matrix $\boldsymbol{K}$; recall that the penalty Λ sandwiched in between comes from a penalty and hence has two zero eigenvalues

At the right we show the eigenvalues for a matrix derived from 25 equally spaced data points in the interval 0 to 1

We also show eigenvectors corresponding to the two smallest eigenvalues (pick two)





And a little more

If we let GDG^t be the eigendecomposition of the matrix K, then we can know that the eigenvalues of $(I + \lambda K)^{-1}$ are

$$\frac{1}{1 + \lambda D_i}$$

Therefore, the eigenvalues of $(I + \lambda K)^{-1}$ are between 0 and 1; and our two zero eigenvalues of K correspond to the largest

Demmler-Reinsch

The columns of the matrix G in constitute a new basis for the natural splines; it is an orthogonal basis (by design)

If we were to solve a least squares problem using this basis, we would have

$(\boldsymbol{y} - \boldsymbol{G\gamma})^t(\boldsymbol{y} - \boldsymbol{G\gamma})$ leading to $\widehat{\boldsymbol{\gamma}} = \boldsymbol{G}^t\boldsymbol{y}$ or $\widehat{y} = \boldsymbol{G}\boldsymbol{G}^t\boldsymbol{y} = \boldsymbol{y}$

The corresponding penalized problem has a solution

$$\widetilde{\boldsymbol{\gamma}} = \boldsymbol{D}^*\boldsymbol{G}^t\boldsymbol{y}$$

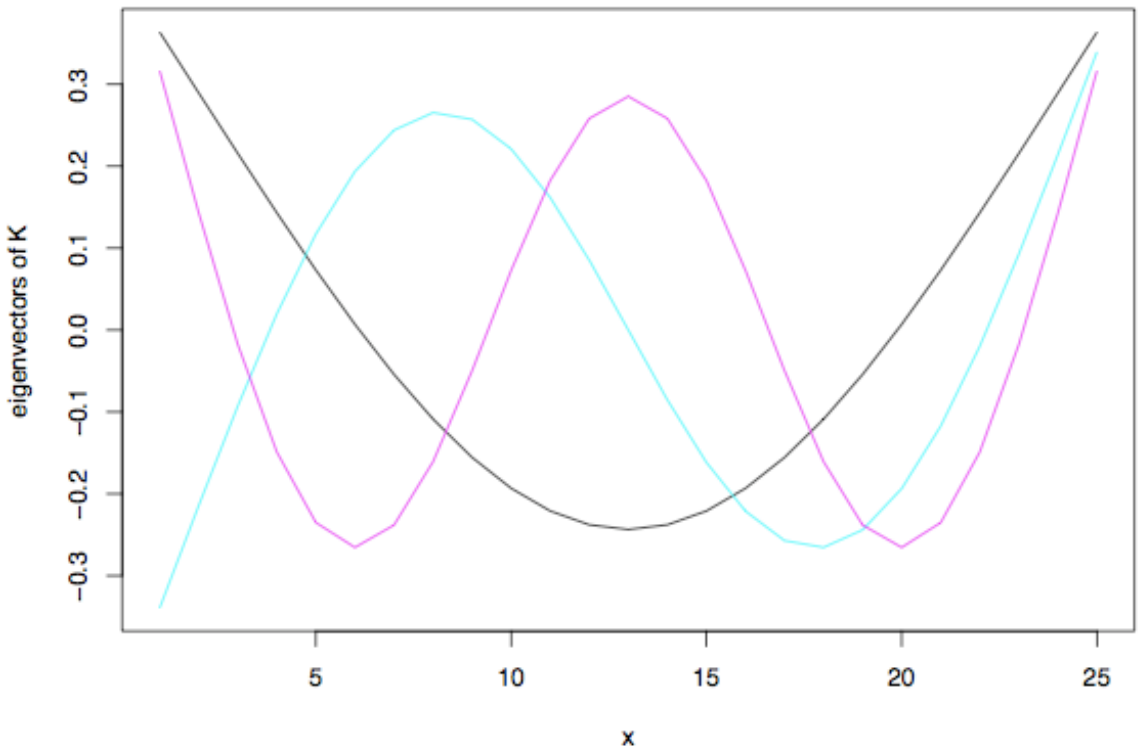
In effect, we are shrinking these coefficients by

$$[\boldsymbol{D}^*]_i = \frac{1}{1 + \lambda D_i}$$

And finally...

So we see that the eigenvectors are increasingly oscillatory; Reinsch worked out the number of zero-crossings for these basis functions

What does this tell us about the nature of the smoothing spline estimate?

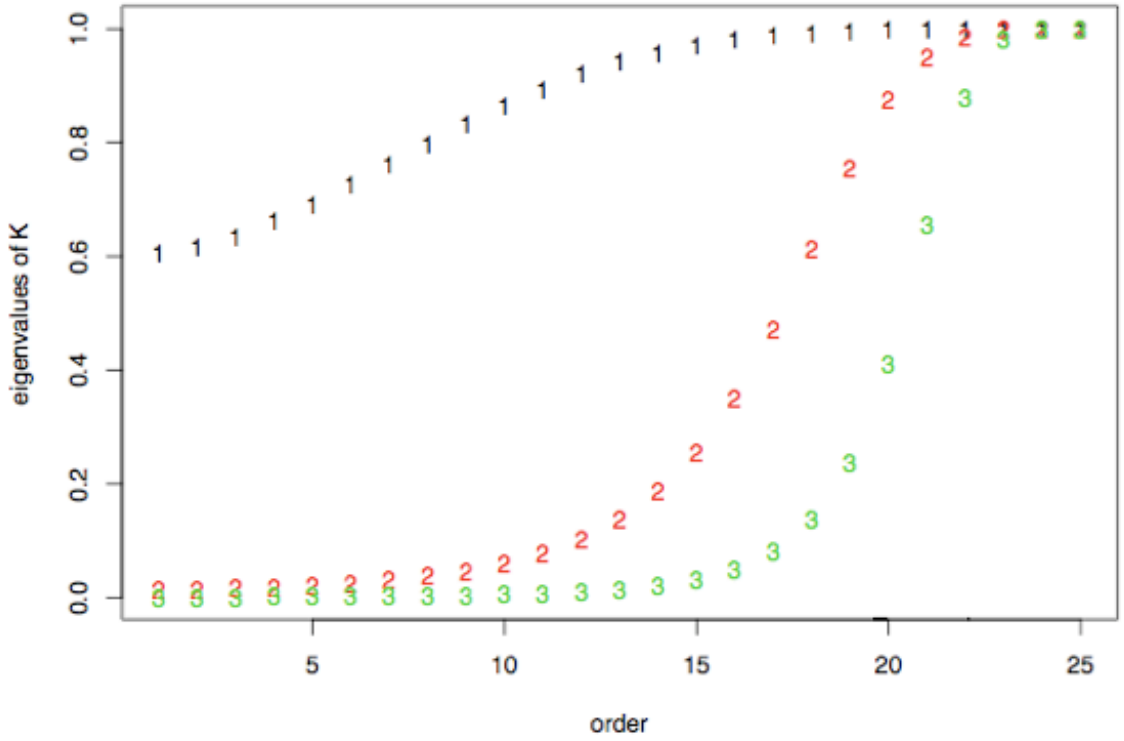


The upshot

This means that the more wiggly basis functions (those associated with larger eigenvalues of G) get hit more in the shrinkage

So by increasing the penalty, we are tuning out the higher frequency components

Note that we have kept the ordering in K so the last two correspond to linear terms



Smoothing operators

Having written the smoothing spline in the form $\mathrm{A}(\lambda)\mathrm{y}$, we can apply some of the tools we've developed already -- For example, we might want to select the smoothing parameter based on a cross validation score

$$CV(\lambda) = \sum_{i=1}^{n} (y_i - \widehat{g_\lambda}^{-i}(x_i))^2$$

CV for smoothing splines

We can use some facts about the smoothing spline estimate so that we don't literally have to drop points and resmooth

The first is a linear relation that connects the value of the estimate at the deleted point

$$\widehat{g}_\lambda^{-i}(x_i) = \sum_{j=1, j\neq i}^{n} A_{ij}(\lambda) y_j + A_{ii}(\lambda)\widehat{g}_\lambda^{-i}(x_i)$$

CV for smoothing splines

From here, we can add and subtract elements to derive the following equality

$$y_i - \widehat{g_\lambda}^{-i}(x_i) = \frac{y_i - \widehat{g}_\lambda(x_i)}{1 - A_{ii}(\lambda)}$$

With this expression, we can rewrite the CV score as

$$CV(\lambda) = \frac{1}{n} \sum_{i=1}^{n} \left(\frac{y_i - \widehat{g}_\lambda(x_i)}{1 - A_{ii}(\lambda)} \right)^2$$

GCV

Until about just over a decade ago, it was not known how to compute the diagonal elements of the smoother matrix efficiently; this led to so-called generalized cross validation

$$GCV(\lambda) = \frac{1}{n}\sum_{i=1}^{n}\left(\frac{y_i - \widehat{g}_\lambda(x_i)}{1 - \text{Trace } A_{ii}(\lambda)/n}\right)^2$$

Technically, the move to the trace was both mathematical as well as computational; it is easier to work with the trace

Let's see what that means...

A little justification

While we motivated GCV for computational reasons, Craven and Wahba showed that GCV can be used to approximate the model error

$$\frac{1}{n}\sum_{i=1}^{n} E[f(m_i) - g_\lambda(m_i)]^2$$

in the sense that $E\,GCV(\lambda) \approx ME + \sigma^2$

On computation

For those more interested in the practical aspects of this methodology...

Hutchinson and deHoog proposed an O(n) algorithm for finding the diagonal elements as well as the trace of the smoother matrix

Numer. Math. 47, 99–106 (1985)

Smoothing Noisy Data with Spline Functions

M.F. Hutchinson and F.R. de Hoog

CSIRO Division of Mathematics and Statistics, GPO Box 1965, Canberra, ACT 2601, A

Summary. A procedure for calculating the trace of the influenc associated with a polynomial smoothing spline of degree $2m-1$ fi distinct, not necessarily equally spaced or uniformly weighted, dat is presented. The procedure requires order m^2n operations and t permits efficient order m^2n calculation of statistics associated with nomial smoothing spline, including the generalized cross validati method is a significant improvement over an existing method w quires order n^3 operations.

Subject Classifications: AMS(MOS): 65D, 65K; CR: G.1.2, G.1.1.

1. Introduction

Since its introduction by Schoenberg [13] and Reinsch [11], the pol smoothing spline has provided an attractive way of smoothing no values observed at n distinct points on a finite interval. Craven and Wa

Comparison

There's a shrinkage (smoothing splines) versus selection (adaptive splines) story to be told here, but one that you're familiar with in some sense...

An interesting extension

In some cases, we are not interested in a conditional mean but a quantile; recall that the mean minimizes

$$\sum_{i=1}^{n}(y_i - \mu)^2$$

whereas the median, say, minimizes

$$\sum_{i=1}^{n}\rho_{0.5}(y_i - \mu)$$

where $\rho_\tau(u) = u\{\tau - I(u < 0)\}$ or for the median $\rho_{0.5}(u) = 0.5|u|$

Quantile smoothing splines

Koenker, Ng and Portnoy studied minimizing the penalized quantile regression problem

$$\sum_{i=1}^{n} \rho_\tau(y_i - g(x_i)) + \lambda V(g')$$

where $V(g')$ is the total variation norm of g'; for any function h, we define this quantity to be

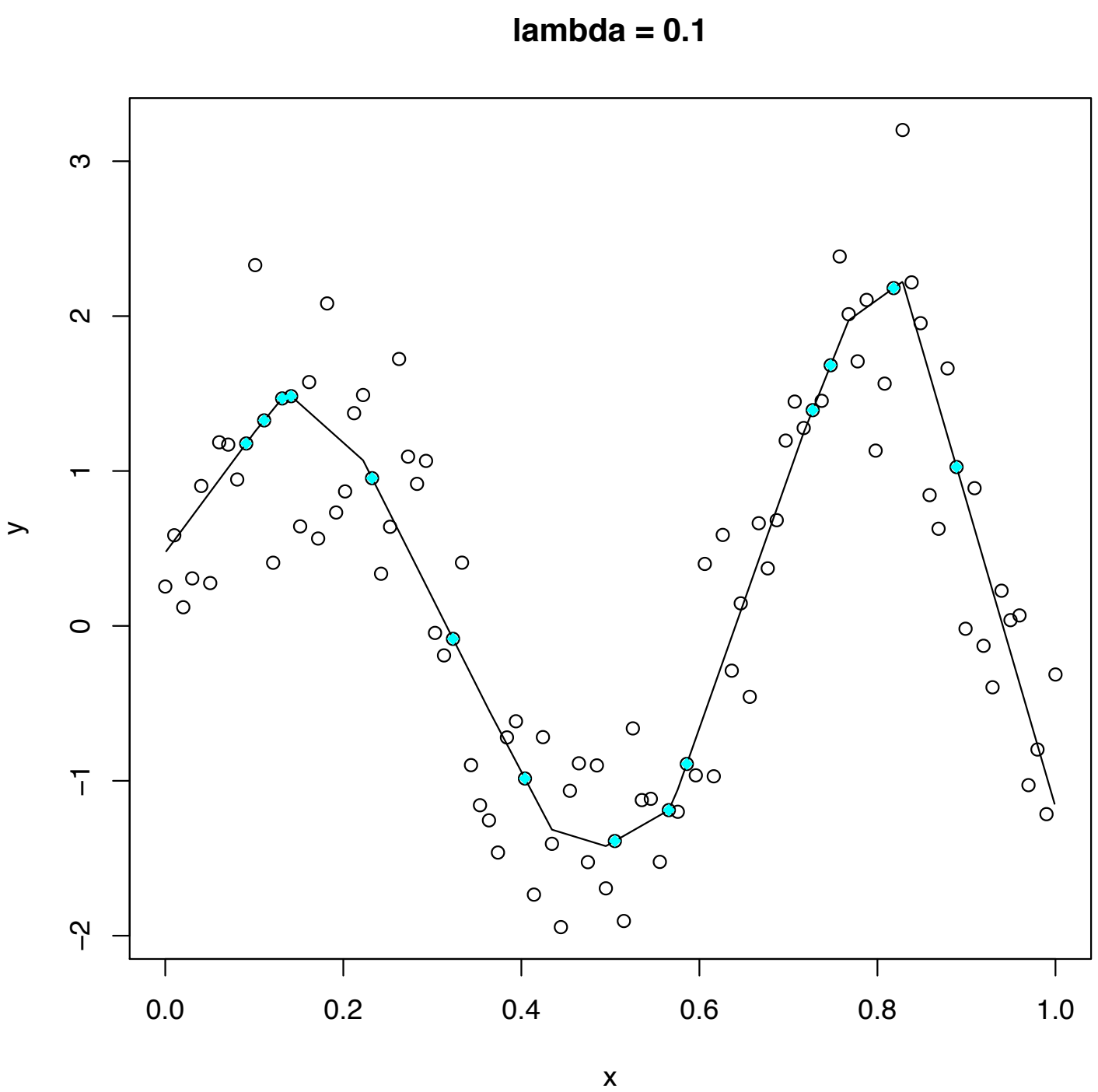
$$V(h) = \int |h'(u)| du$$

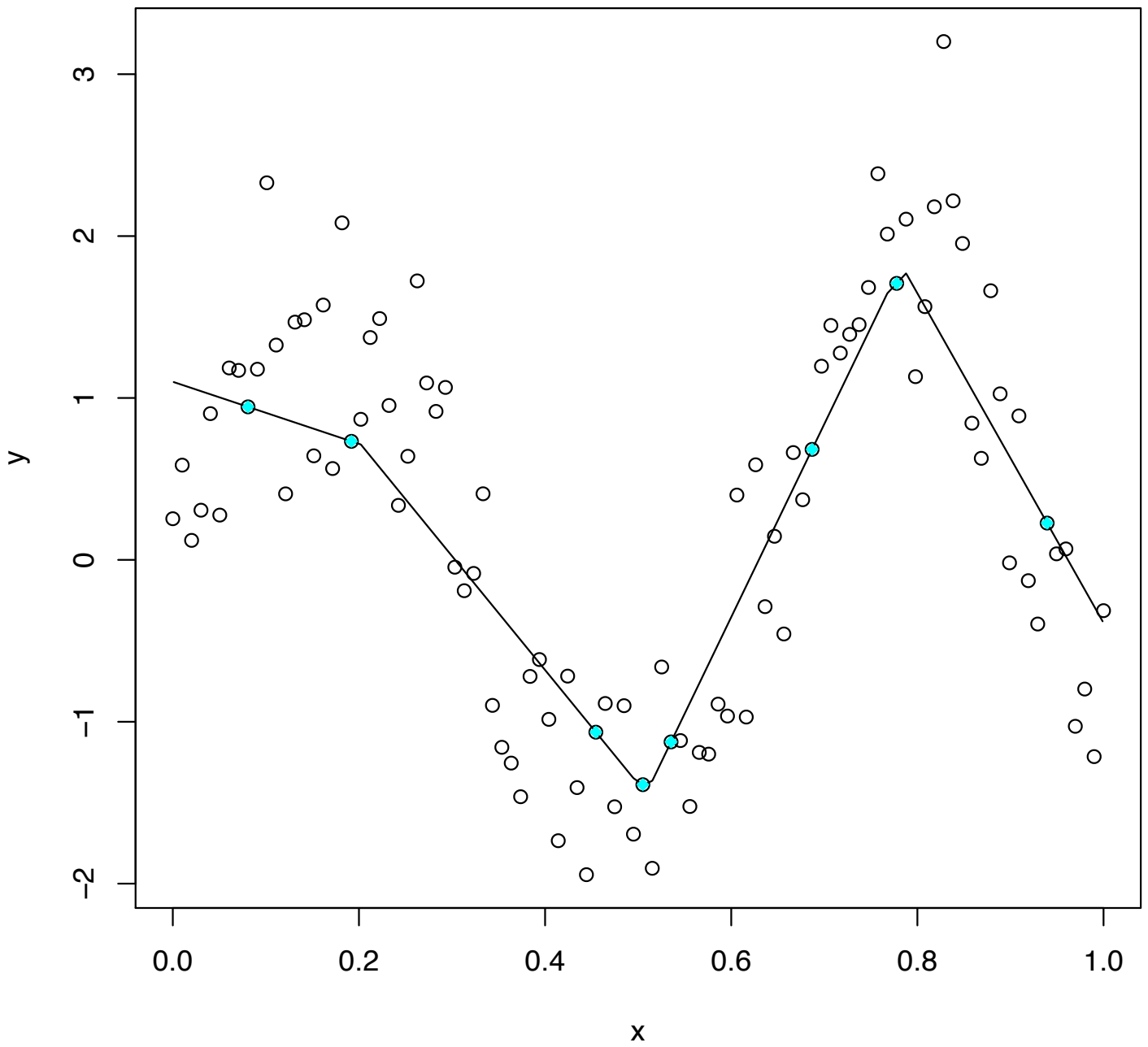
assuming certain smoothness conditions on h

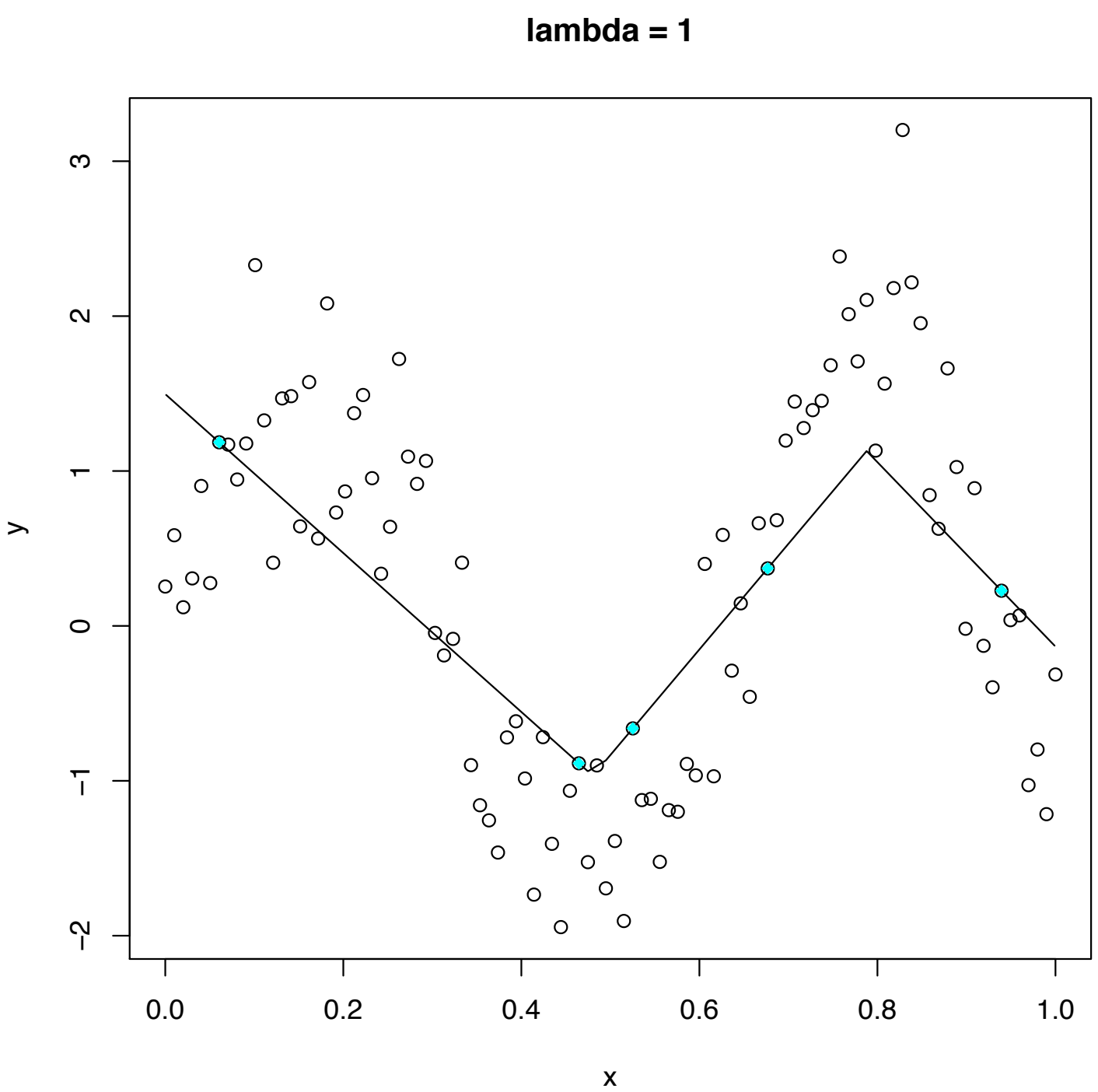
Quantile smoothing splines

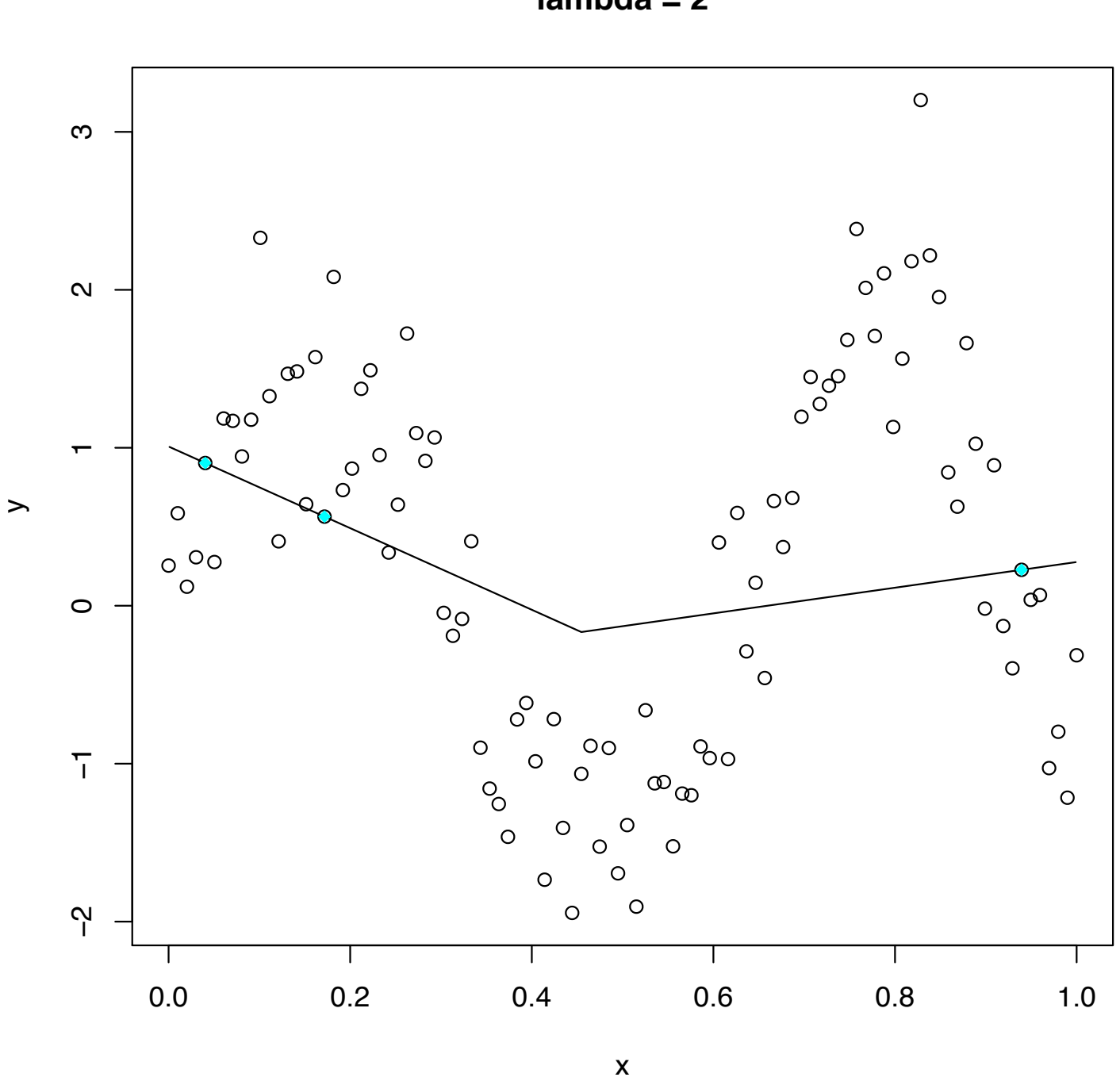
The solution to this problem can be shown to be a linear spline with knots at the input points $x_1, \ldots, x_n$

The degrees of freedom of the fit is approximated by the number of points interpolated...


lambda = 0.1
y
x
3
2
1
0
−1
−2
0.0
0.2
0.4
0.6
0.8
1.0


lambda = 0.5
y
x
3
2
1
0
−1
−2
0.0
0.2
0.4
0.6
0.8
1.0


lambda = 1
y
x
3
2
1
0
−1
−2
0.0
0.2
0.4
0.6
0.8
1.0


lambda = 2
y
x
3
2
1
0
−1
−2
0.0
0.2
0.4
0.6
0.8
1.0

Function estimation

Last time we considered ways to relax the assumption that our unknown regression function (conditional expectation) is simply a plane -- That is, a linear combination of the predictor variables

Today we'll spell out a slightly more formal treatment of the basic ideas and motivate how they translate to multivariate problems -- To start, let's be a little explicit about our setup and assume that our input space is d-dimensional, meaning that $x = (x_1, \ldots, x_d)$, and assume that for some unknown function f our response variable Y satisfies

$$f(x) = E(Y|X = x) \text{ and } \text{var}(Y|X = x) = \sigma^2 \text{ for all } x \in \mathcal{X} \subset \mathbb{R}^d$$

In the special case of a normal model we have

$$Y = f(X) + \epsilon, \ \epsilon \sim N(0, \sigma^2)$$

Function estimation

Now, instead of thinking of our predictors as vectors and our model as a member of the column space of the model matrix, let's take a step back and introduce a p-dimensional linear space of functions $g \in \mathbb{G}$ defined on $\mathcal{X}$ consisting of all

$$g(x) = g(x; \beta) = \beta_1 B_1(x) + \cdots + \beta_p B_p(x)$$

where $B_1(x), \ldots, B_p(x)$ are basis functions of $\mathbb{G}$

In this expanded treatment, we think of the elements of $\mathbb{G}$ as being candidate descriptions for the regression function f

Function estimation

For most of this quarter we have been assuming that p = d+1 and that our basis functions are

$$B_1(x) = 1 \text{ and } B_{j+1}(x) = x_j \text{ for } j = 1, \ldots, d$$

Ironically, we've spent big part of this class questioning this model and examining diagnostics to reveal its shortcomings -- Let's look at other examples

Polynomials

If we let d=1, then $\mathbb{G}$ might be the space of all polynomials of **order k** -- That is, we consider all combinations of the basis elements

$$B_j(x) = x^{j-1} \text{ for } j = 1, \ldots, k \text{ and } x \in \mathbb{R}$$

and the space has dimension p=k

If d > 1, we might consider the space of all polynomials of **(coordinate) order k** -- That is, we consider all combinations of the basis elements

$$x_1^{k_1} x_2^{k_2} \cdots x_d^{k_d} \text{ for } k_j \leq k$$

giving a space of dimension p= k^d

Splines

For d=1 again, the space of cubic splines with m knots has dimension m+4 and in general, the spline space of order k has dimension p = m+k (cubics are polynomials with order 4)

What about d>1? How do we generalize this? Let's look at a general construction that works for any set of function spaces and not just splines...

Tensor products

If we let $\mathbb{G}_1, \ldots, \mathbb{G}_d$ be d spaces with dimensions $p_1, \ldots, p_d$, respectively, where $\mathbb{G}_j$ has basis set

$$B_{j,1}(x_j), \ldots, B_{j,p_j}(x_j)$$

then we can define the tensor-product space $\mathbb{G}_1 \otimes \mathbb{G}_2 \otimes \cdots \otimes \mathbb{G}_d$ (you were dying to use that symbol, right?!) to be the space of all functions of the form

$$g(x; \beta) = \sum_{j_1=1}^{p_1} \cdots \sum_{j_d=1}^{p_d} \beta_{j_1,\ldots,j_d} B_{1,j_1}(x_1) \cdots B_{d,j_d}(x_d)$$

which has dimension $p_1 \cdots p_d$

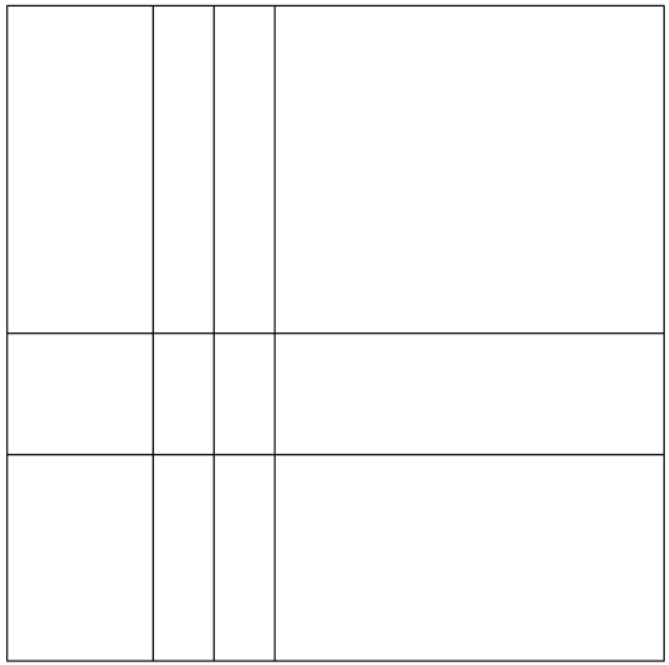
Multivariate splines

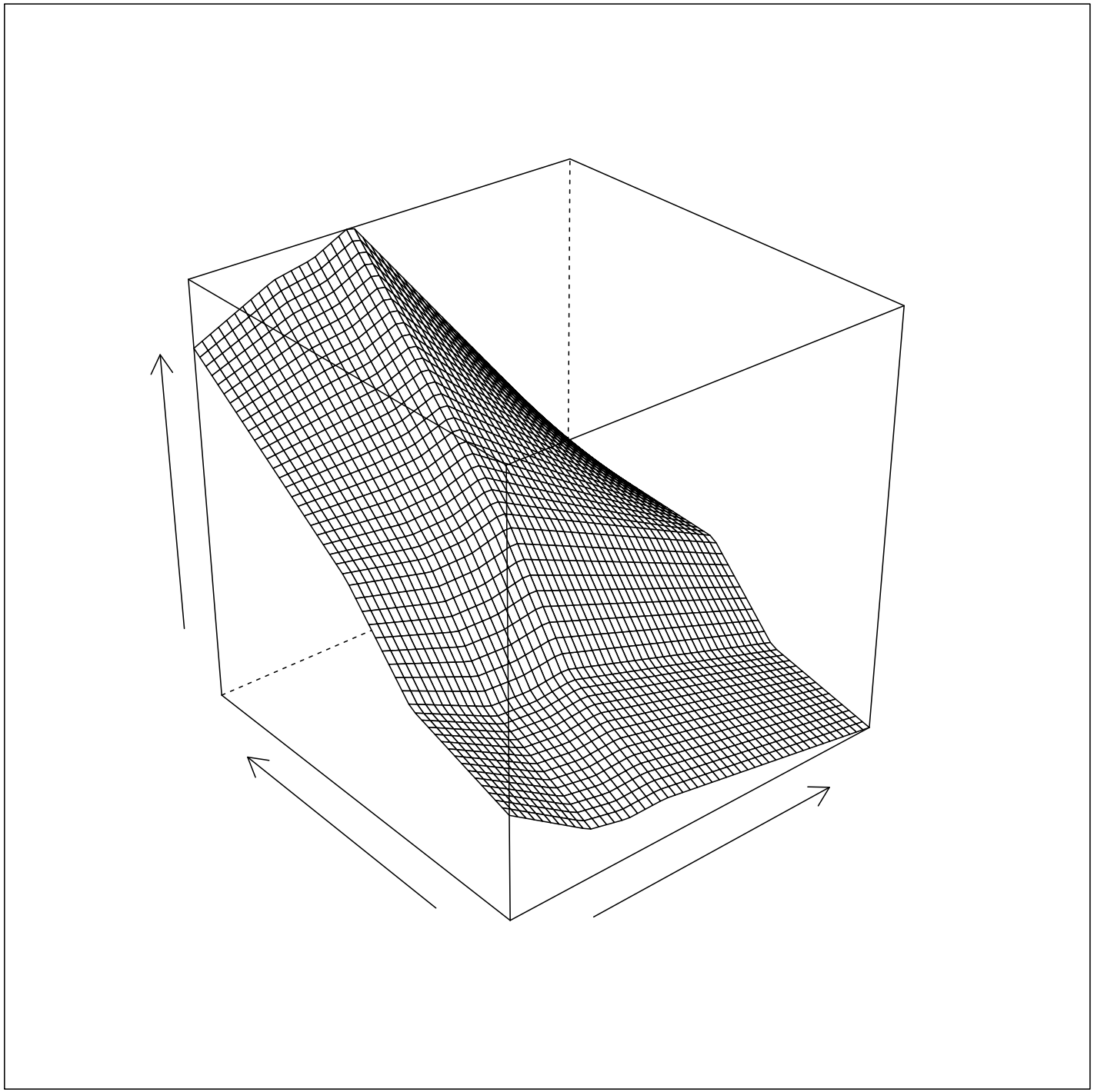
Given this simple construction, let's now take each $\mathbb{G}_j, j = 1, \ldots, d$ to be a spline space with possibly varying order and separate knot sets

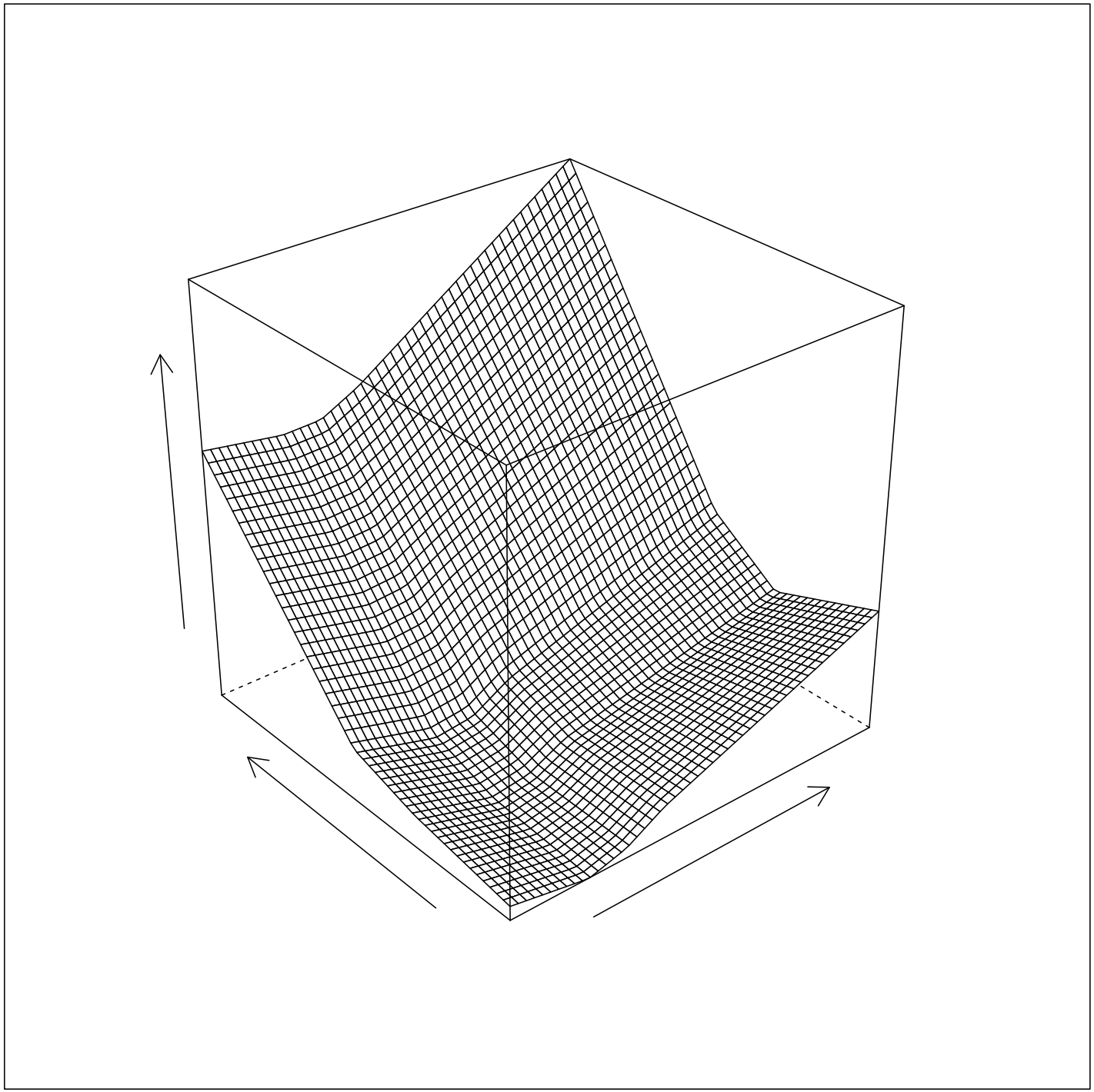
After a little thought, it should be clear that the resulting space is a polynomial inside cubes defined by the locations of the knots on each variable

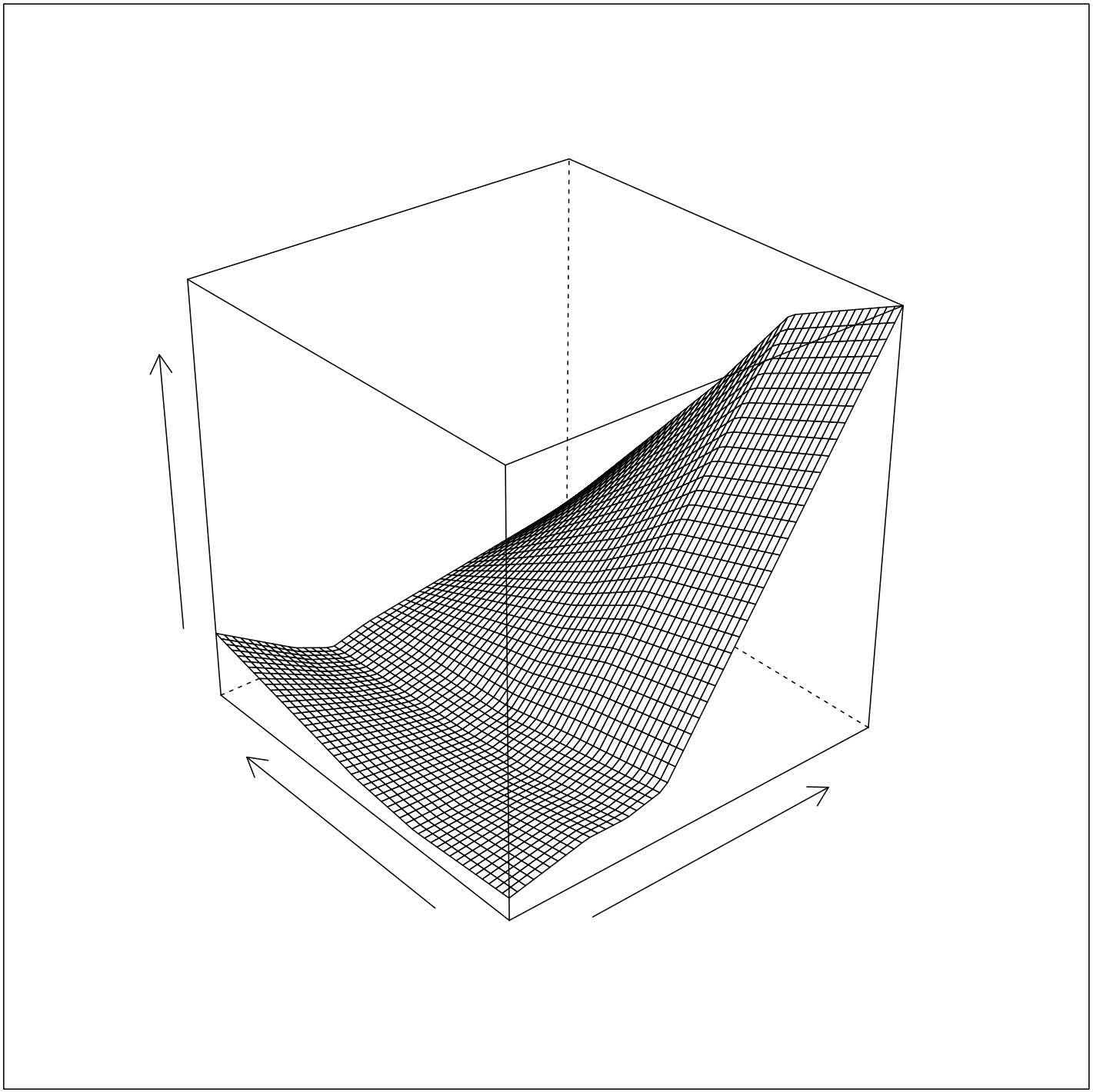
Here we have d=2 and $\mathcal{X}$ is the unit square -- We have three knots in the x-direction (0.2, 0.3 and 0.4) and two knots on the y (at 0.3 and 0.5)

The tensor product of two spline spaces defined with these knots has single polynomial pieces in each box and breaks in the derivatives as you cross the knot lines -- The degrees and smoothness depending on your original choices in the two spaces









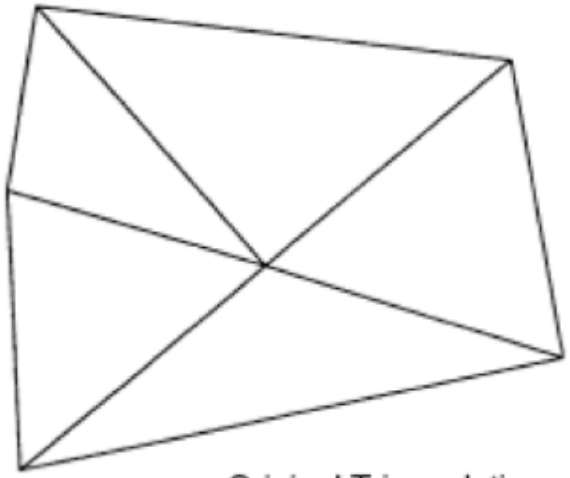
Multivariate splines

So far we've seen only splines used in the context of univariate or additive problems; what do we do to describe interactions?

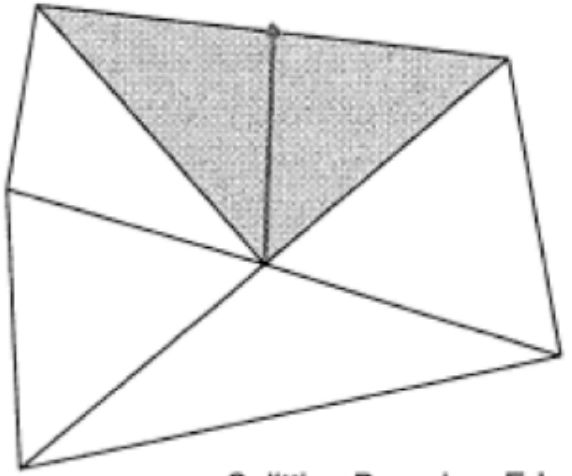
There is one line of reasoning that tries to build genuinely multivariate spline spaces; starting with an extension of "piecewise" polynomials, we could define regions in the plane, say, to be our pieces

We would then introduce constraints to smooth things out; in the case of univariate splines, we end up with compactly supported basis functions (bumps) -- for a multivariate problem this becomes harder

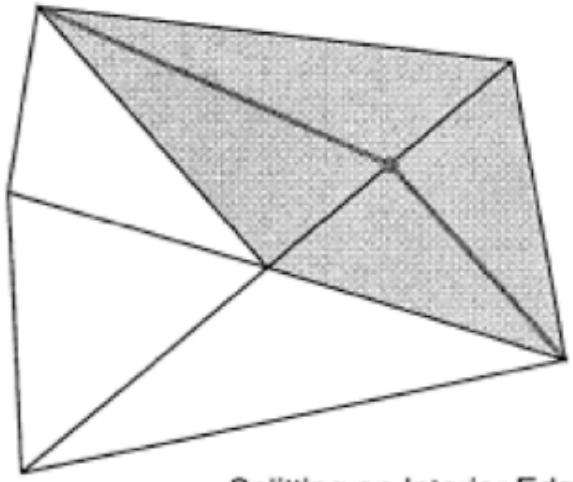
Here's one simple example...



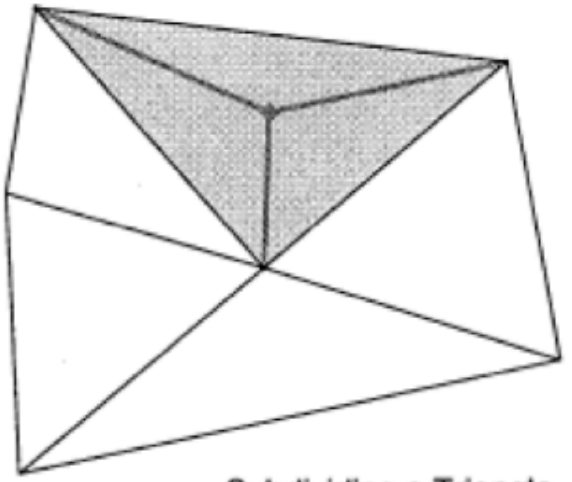
Original Triangulation



Splitting Boundary Edge



Splitting an Interior Edge

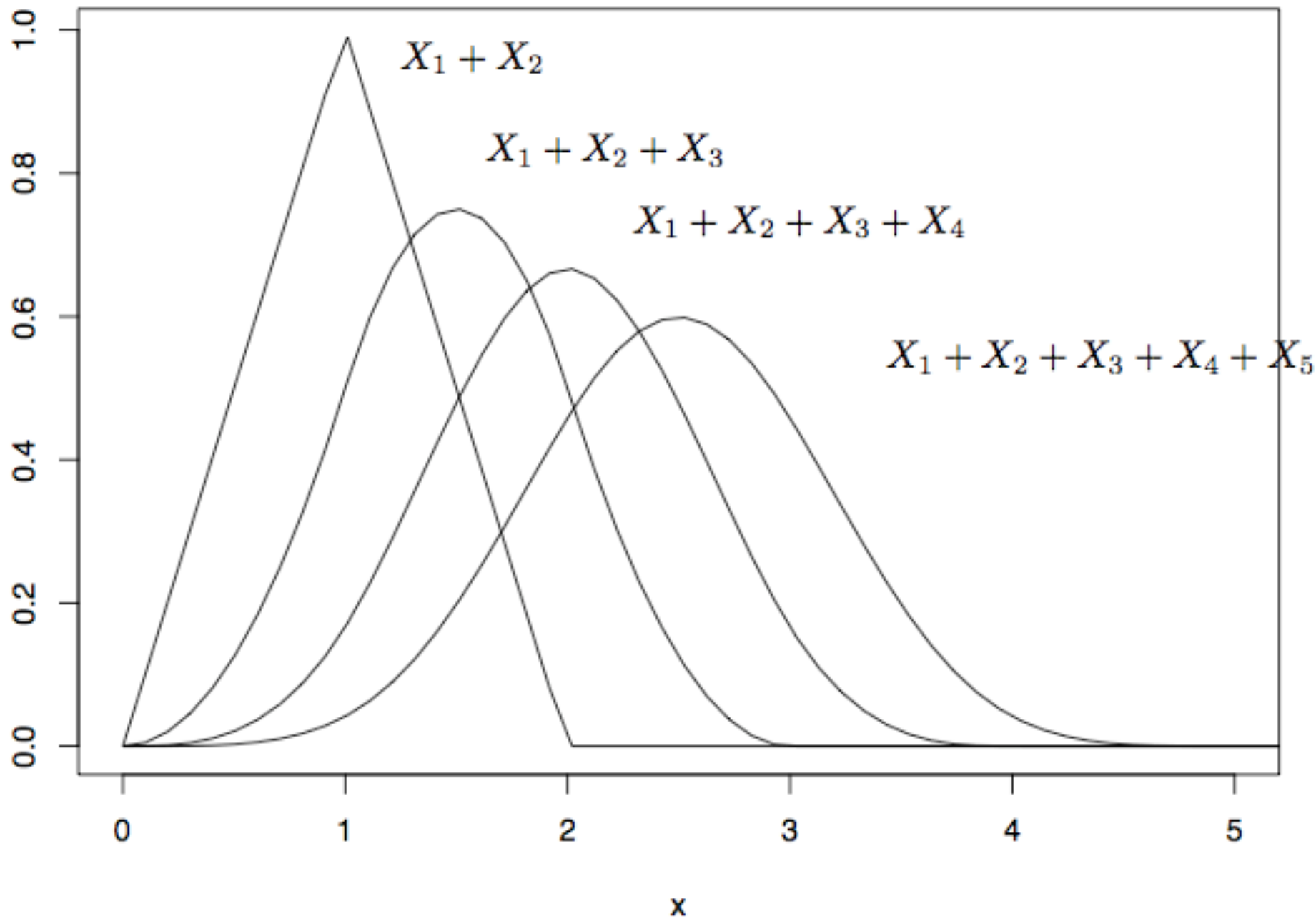


Subdividing a Triangle

Multivariate splines

There is also an extension (truly beautiful) known as simplex splines that take the convolution construction we alluded to earlier and creates multivariate bumps over triangulations

Again, for equally spaced knots we have...


$X_1 + X_2$
$X_1 + X_2 + X_3$
$X_1 + X_2 + X_3 + X_4$
$X_1 + X_2 + X_3 + X_4 + X_5$
0.0
0.2
0.4
0.6
0.8
1.0
0
1
2
3
4
5
x

Function estimation and OLS

Now, given n observations $(x_1, y_1), \ldots, (x_n, y_n)$ and a p-dimensional linear space of functions $\mathbb{G}$, we choose the coefficients $\beta = (\beta_1, \ldots, \beta_p)$ by ordinary least squares -- That is, we seek to find the $g \in \mathbb{G}$ that minimizes

$$\sum_{i=1}^{n} [y_i - g(x_i; \beta)]^2$$

We haven't gone too far because we still have $\widehat{\beta} = (M^t M)^{-1} M^t y$ where M is the model matrix $[M]_{ij} = B_j(x_i)$ and $\mathbb{G}$ has basis functions $B_1(x), \ldots, B_p(x)$

Approximation spaces

For the rest of the lecture (and maybe the next?) we'll focus on so-called approximation spaces -- That is, linear spaces that are adaptable in the sense that they can describe a wide range of smooth functions

The notion of adaptability is quantified through the approximation error -- Given a function f and a linear space $\mathbb{G}$, we define the distance

$$d(f, \mathbb{G}) = \min_{g \in \mathbb{G}} \|f - g\|_\infty$$

where $\|f - g\|_\infty = \sup_{x \in \mathcal{X}} |f(x) - g(x)|$

In-sample error revisited

A few lectures ago, we examined the expected error in our estimates and this quantity emerged -- We'll call it an (in-sample) model error

$$\mathrm{ME}(\mathbb{G}) = \frac{1}{n}\sum_{i=1}^{n} \mathrm{E}\left[f(x_i) - g(x_i;\widehat{\beta})\right]^2$$

If we let $\widetilde{\beta} = \mathrm{E}\widehat{\beta}$ then we saw that we could write

$$\mathrm{ME}(\mathbb{G}) = \frac{1}{n}\sum_{i=1}^{n} \left[f(x_i) - g(x_i;\widetilde{\beta})\right]^2 + \frac{1}{n}\sum_{i=1}^{n} \mathrm{E}\left[g(x_i;\widetilde{\beta}) - g(x_i;\widehat{\beta})\right]^2$$

which is a bias (first term) variance (second term) tradeoff

In-sample error

Now, because $\widetilde{\beta} = \mathsf{E}\widehat{\beta}$ we see that

$$\widetilde{\beta} = \mathsf{E}\widehat{\beta} = (\mathsf{M}^{\mathsf{t}}\mathsf{M})^{-1}\mathsf{M}^{\mathsf{t}}\mathsf{E}\mathsf{Y} = (\mathsf{M}^{\mathsf{t}}\mathsf{M})^{-1}\mathsf{M}^{\mathsf{t}}\mathsf{f}$$

where $\mathsf{f} = (\mathsf{f}_1, \ldots, \mathsf{f}_n)$ for $\mathsf{f}_i = \mathsf{f}(\mathsf{x}_i)$

This says that $\mathbb{G}$ is just the least squares projection of the true regression function into the space

In-sample error

Put another way, $\widetilde{\beta}$ satisfies

$$\frac{1}{n}\sum_{i=1}^{n}\left[f(x_i) - g(x_i;\widetilde{\beta})\right]^2 = \min_{g\in\mathbb{G}}\frac{1}{n}\sum_{i=1}^{n}\left[f(x_i) - g(x_i)\right]^2$$

Now, if we let $g^* = \text{argmin}_{g\in\mathbb{G}}\|f-g\|_\infty$, the "best" approximation to f in $\mathbb{G}$, we have that

$$\begin{aligned}\min_{g\in\mathbb{G}}\frac{1}{n}\sum_{i=1}^{n}\left[f(x_i) - g(x_i)\right]^2 &\leq \frac{1}{n}\sum_{i=1}^{n}\left[f(x_i) - g^*(x_i)\right]^2 \\ &\leq \frac{1}{n}\sum_{i=1}^{n} d^2(f,\mathbb{G}) \\ &= d^2(f,\mathbb{G})\end{aligned}$$

In-sample error

This implies that the bias term in our in-sample model error is bounded from above by $d^2(f, \mathbb{G})$ -- We recall that the variance term (because we have p predictors in our regression equation) is $p\sigma^2/n$

Therefore, we can bound our model error associated with the space $\mathbb{G}$ as

$$ME(\mathbb{G}) \leq d^2(f, \mathbb{G}) + \frac{p\sigma^2}{n}$$

Approximation power of polynomials

OK, let's put this to work! As baby example, Jackson's Theorem (a standard result in numerical analysis) tells us the approximation power of univariate polynomials of order k (which we'll denote $\mathcal{P}_k$)

Specifically, if f has a bounded rth derivative and $\mathcal{X} = [a, b]$, then

$$d(f, \mathcal{P}_k) \leq C \left(\frac{b - a}{2k} \right)^r$$

This says that we can have the same effect on approximation error by dividing the interval into 5 pieces and use cubics (k=4) as we can by approximating f by a single polynomial of order 20

Approximation power of splines

A more refined result can be derived that relates to splines and not just (piecewise) polynomials -- If f has r continuous derivatives, then for k>r, the approximation rate for $\mathcal{S}_k(m)$, the space of splines with m equally-spaced knots in $\mathcal{X} = [a, b]$

$$d(f, \mathcal{S}_k(m)) = C \left(\frac{1}{m}\right)^r$$

In-sample error

This means that our model error associated with $\mathcal{S}_k(m)$ can be bounded from above by the expression

$$C\left(\frac{1}{m}\right)^{2r} + \frac{(m+k)\sigma^2}{n}$$

where the spline space of order k with m knots has dimension (m+k)

In-sample error

Now, for each sample size n we can balance the bias and variance terms -- We're going to be more concerned now with rates meaning how these quantities scale with increasing sample size n

The rate game is something you'll get in Stat 200c next term, but for now, let's just see roughly how big ME can be...

In-sample error

After a little algebra we find that m should increase with n at the rate

$$m \sim n^{\frac{1}{2r+1}}$$

Substituting this into the ME bound, we find that this choice produces a ME that is bounded by

$$n^{-\frac{2r}{2r+1}}$$

How do we make sense of this?

In-sample error

Notice that if we could assume that f was a member of $\mathbb{G}$, then we wouldn't have a bias term and the model error would drop like 1/n as our sample size increased (it's customary to talk about the square root of an error instead in which case you have the more familiar $1/\sqrt{n}$ "rate")

Instead, our error is decreasing at a slower rate, one that depends on the smoothness of the unknown regression function -- This, then, becomes an assumption to replace the strong "parametric" form we started our quarter with

Univariate function estimation

All of this analysis can be beefed up and there are excellent papers by Stone and Huang and Stone on the subject -- But even these rough calculations show that if we are only able to assume something about the smoothness of our regression function, our approximation spaces should become more complex as we collect more and more data

Working in this way, our estimates are always biased (we never assume the true regression function f is a polynomial or a spline) but that as we collect more and more data, our estimates are closer and closer to f

Multivariate function estimation

Now, moving from d=1 to general dimensional input spaces, we can derive a similar result, but with one important difference -- If we model with the tensor product $\mathbb{G}_1 \otimes \mathbb{G}_2 \otimes \cdots \otimes \mathbb{G}_d$ of d spline spaces, each with order k and m equally spaced knots then our model error bound is

$$n^{-\frac{2r}{2r+d}}$$

under multivariate smoothness assumptions that we won't really get into -- Still, what do you notice?

Summary

The upshot is that high-dimensional input spaces slow the rate of convergence considerably -- To achieve the same rate of decay for a 11-dimensional input space as a univariate one we have to collect n^3 points as opposed to n, assuming r=2

The escalating difficulty of the estimation problem with n is referred to as the “curse of dimensionality” -- This is a term coined by Bellman in 1961 to describe problems that scale exponentially with dimension

Curses

There are plenty of consequences of the curse to go around -- For example, suppose we were determined to use high-order polynomials as our approximation space of choice

Well, we saw that these tools had the tendency to behave badly in regions with little data -- The curse of dimensionality appears here to open holes in the input space as d gets large

To see this, assume our predictors are uniformly distributed in $[0,1]^d$ and consider a “hole” that is a subcube with side length δ -- The chance that this hole is empty (none of our n data points fall inside) is $(1-\delta^d)^n$

The chance that a given subcube of side-length $\delta = 0.5$ is empty is nearly zero for d=1,2,3 but for d=12 it’s 0.78 and for d=20 it’s effectively 1

Curses

Using the tensor product idea, we also have an exponential explosion in basis functions -- Here we can provide some structure that will help us tame the curse to some extent

Suppose for some dimension d we want to work with a space of polynomials of (coordinate) order 1 -- Then we can rewrite any function g in the space as

$$g(x) = \beta_0 + \sum_{j=1}^{d} \beta_j x_j + \sum_{1 \le j < k \le d} \beta_{jk} x_j x_k + \sum_{1 \le j < k < l \le d} \beta_{jkl} x_j x_k x_l + \cdots$$

This is the basis of the ANOVA expansions you studied in Stat 201b

Cures

The same kind of construction can be applied to any tensor product space -- Recall we can write any function g in $\mathbb{G}_1 \otimes \cdots \otimes \mathbb{G}_d$ as

$$g(x) = \sum_{j_1=1}^{p_1} \cdots \sum_{j_d=1}^{p_d} \beta_{j_1,\ldots,j_d} B_{1,j_1}(x_1) \cdots B_{d,j_d}(x_d)$$

Now, suppose that each $\mathbb{G}_j$, j=1,...,d, contains the constant function and for simplicity assume that it's the first basis function for each space, $B_{j,1}$ -- Then we can collect all the terms in our tensor product expansion that involve x_1 and write

$$g_1(x_1) = \sum_{j_1=2}^{p_1} \beta_{j_1,1,\ldots,1} B_{1,j_1}(x_1)$$

Cures

We can do the same thing for the first d “main effects” -- The two-way interaction between variables 1 and 2 can be similarly written as

$$g_{1,2}(x_1, x_2) = \sum_{j_1=2}^{p_1} \sum_{j_2=2}^{p_2} \beta_{j_1,j_2,1,\ldots,1} B_{1,j_2}(x_1) B_{2,j_2}(x_2)$$

and extended to any two-way interaction

If we keep this up, we can eventually write any function g in the tensor product space as

$$g(x) = g_0 + \sum_{j_1} g_{j_1}(x_{j_1}) + \sum_{j_1<j_2} g_{j_1,j_2}(x_{j_1}, x_{j_2}) + \sum_{j_1<j_2<j_3} g_{j_1,j_2,j_3}(x_{j_1}, x_{j_2}, x_{j_3}) + \cdots$$

Cures

If we terminate the expansion with say two-way interactions, then the target of our estimation procedure is no longer the complete regression function, but a similarly truncated expansion for the regression function -- That is, the “best” approximation to f of the form

$$f^*(x) = f_0^* + \sum_{j_1} f_{j_1}^*(x_{j_1}) + \sum_{j_1<j_2} f_{j_1,j_2}^*(x_{j_1}, x_{j_2})$$

If we worked a bit harder and forced the higher order terms in our expansion of g to be orthogonal (using the data inner product) to the lower order terms, then the individual terms in the expansion of g approach their counterparts in f* as n gets large

Cures

This expansion gives us a way to tame the curse of dimensionality -- If we choose to not work with the full tensor product basis but instead maybe a model with just main effects or maybe all main effects and two-way interactions, we can improve the rate at which our model error decays

In particular, the rate becomes $n^{-\frac{2r}{2r+s}}$ where s is the count of the largest number of variables included in any one interaction term in the model on the previous page -- A main-effects model has s=1, while all two-way interactions mean that s=2

The important thing is that with this construction, estimating a d main effects is “as hard as” estimating a univariate function -- We can undercut the curse by dropping higher-order terms

But what do we lose in the process?

Functional ANOVA

A functional ANOVA model, then, involves undercutting the curse of dimensionality by dropping higher-order interactions in our estimate -- The target is no longer f but f*, the best approximation to f in the given form

A simple version of this is the so-called additive model -- Here we consider just the main effects

$$g(x) = g_0 + g_1(x_1) + \cdots + g_d(x_d)$$

where again each 1-factor term is a spline in the corresponding variable

Additive models

There are, of course, more direct ways to motivate an additive model -- In particular, you can think of it as the simplest elaboration of the ordinary linear model

$$g(x) = \beta_0 + \beta_1 x_1 + \cdots + \beta_d x_d$$

We've gone to the trouble of providing an (at least back of the envelope) model error assessment to motivate why this form is interesting generally

Fitting (part I)

One strategy for fitting this kind of model is to learn the important interactive components using the data at hand -- The process mimics stepwise addition of variables

We've already seen the correspondence between elements of the truncated power basis and knot (breakpoint) addition or deletion in univariate fitting -- For the full multivariate model, we do essentially the same thing, but have to fold in the introduction of variables

Here's a sketch of one algorithm...

Specify an initial spline model

Stepwise addition: Continue until the maximum model size is reached or no candidates can be found

- Decide which basis functions are candidates for addition
- Add the best candidate
- Fit the model
- Evaluate the model
- If the model is better than the best previous one, save it

Stepwise deletion: Continue until the minimum model size is reached

- Decide which basis function can be removed from the model
- Remove the one that is the worst predictor
- Fit the model
- Evaluate the model
- If the model is better than the best previous one, save it

Output the best fit

Fitting (part I)

Broadly, this algorithm lets us start with a relatively simple model (maybe just an intercept-only fit) and then add components gradually

With the truncated power basis there are two kinds of basis functions -- One is a global polynomial built from $1, x, x^2, \ldots, x^k$ and the other are ramp functions of the form $(x - t)_+^k$

Typically, we constrain the addition process so that we only add spline terms after the full corresponding polynomial has been entered -- And during deletion, we remove the spline terms before the polynomial pieces

Fitting (part I)

For linear splines, the functions are broken lines in each variable -- The selection process first enters a simple linear function in a variable x_j and then entertains the spline terms $(x_j - t)_+$

Here's how these constraints play out in terms of the candidates available to add or delete at any step in the general algorithm two slides back...

Fitting (part I)

For an additive model using piecewise linear splines (a broken line in each variable), our candidate basis functions available at each step consist of

$x_j, j = 1, \ldots, d$ and

$(x_j - t_{j,m})_+$ if x_j is in the model

During backward deletion, we want to make sure we remove truncated power basis elements before the corresponding linear term -- At each step of the deletion process, the candidates for removal are

$(x_j - t_{j,m})_+$ and

x_j if there are no terms of the form $(x_j - t_{j,m})_+$ in the model

Looking at the chain of models generated in this way, the “best” can be selected using one of the selection criterion (Cp, AIC, BIC) or via a test set or even using cross validation

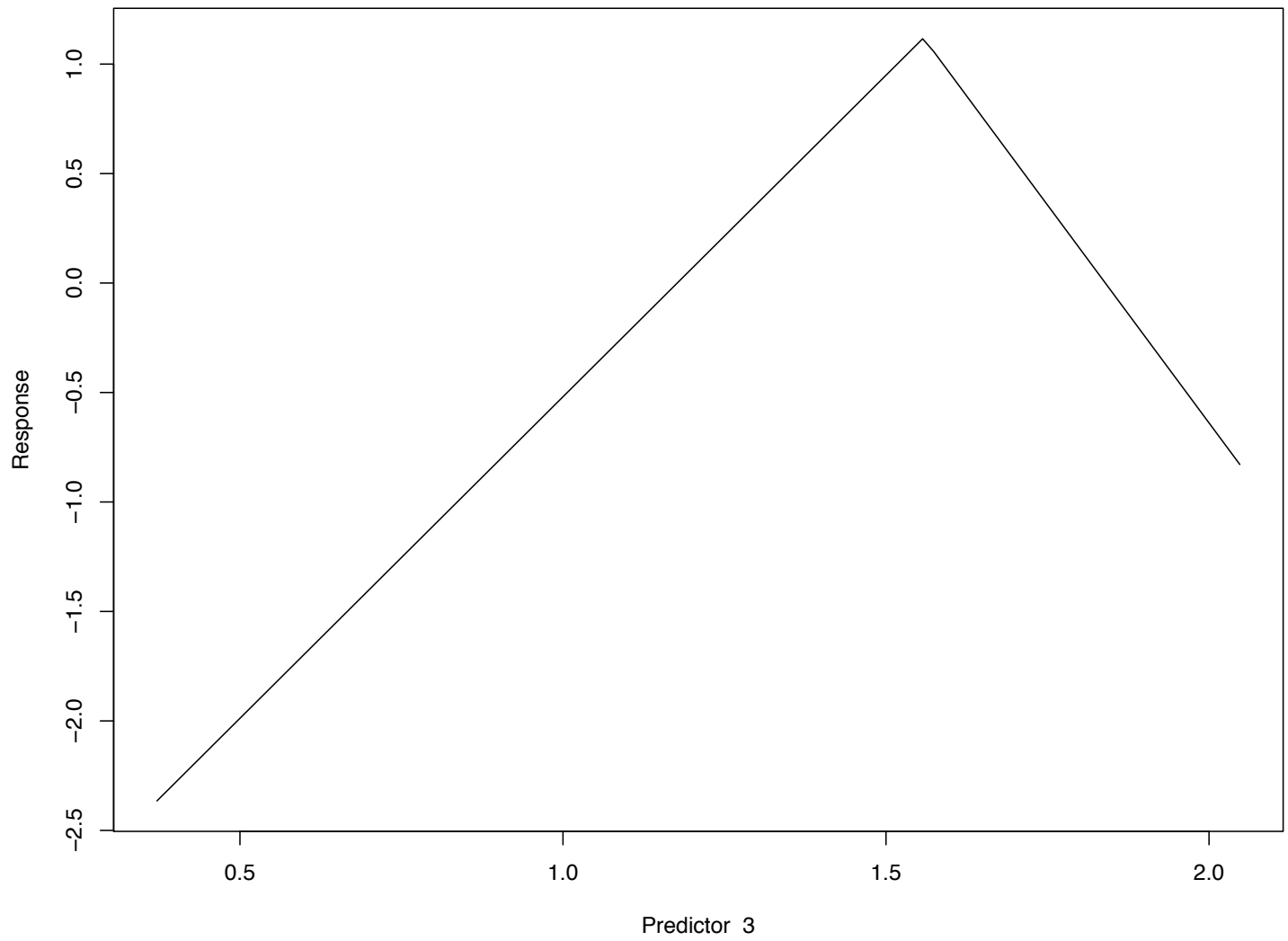
Fitting (part I)

If we want a model with at most two-factor interactions, we might impose the following constraints on the candidate addition set

$x_j, j = 1, \ldots, d$ and
$(x_j - t_{j,m})_+$ if x_j is in the model
$x_{j_1} x_{j_2}$ if x_{j_1} and x_{j_2} are in the model
$x_{j_1}(x_{j_2} - t_{j_2,m})_+$ if x_{j_1} x_{j_2} and $(x_{j_2} - t_{j_2,m})_+$ are in the model
$(x_{j_1} - t_{j_1,m_1})_+(x_{j_2} - t_{j_2,m_2})_+$ if $x_{j_1}(x_{j_2} - t_{j_2,m_2})_+$ and $x_{j_2}(x_{j_1} - t_{j_1,m_1})_+$ are in the model

Clearly this is just a bookkeeping exercise, and thankfully one that we don't have to do on our own -- The PolyMARS algorithm from Charles Kooperberg implements this strategy

```
> polymars(vul$ln_death_risk,vul[,c("ln_pop","ln_events","ln_fert","hdi")])
Call:
polymars(responses = vul$ln_death_risk, predictors = vul[, c("ln_pop",
    "ln_events", "ln_fert", "hdi")])

Model fitting

   0/1 size      RSS       GCV
1    1    1 438.2049  3.219465
2    1    2 383.5404  2.986041
3    1    3 286.1273  2.364689
4    1    4 243.2221  2.137694
5    1    5 203.6456  1.907191
6    1    6 198.8474  1.988474
7    1    7 189.3154  2.025968
8    1    8 182.6697  2.096973
9    1    9 175.1646  2.162526
10   1   10 168.4340  2.242465
11   1   11 162.4136  2.338755
12   1   12 158.1545  2.471164
13   1   13 156.0660  2.655187
14   1   14 154.1593  2.866599
...
62   0    2 401.4299  3.125320
63   0    1 438.2049  3.219465


Model produced

  pred1    knot1 pred2 knot2      coefs        SE
1     0       NA     0    NA -4.3963974 0.5157676
2     3       NA     0    NA  2.9370508 0.3132665
3     3 1.558145     0    NA -6.9173484 1.1380678
4     2       NA     0    NA  1.1998149 0.1606159
5     1       NA     0    NA -0.4721077 0.0908349

Rsquared : 0.535
> fit <- polymars(vul$ln_death_risk,vul[,c("ln_pop","ln_events","ln_fert","hdi")])
> plot(fit,3)
```


Response
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
−2.5
0.5
1.0
1.5
2.0
Predictor 3

Fitting (part I)

PolyMARS is half acronym -- MARS stands for Multivariate Adaptive Regression Splines, a procedure developed by Jerry Friedman in 1990, citing similarities with the tree-based procedures we saw earlier

Similar algorithms were developed by Breiman (DKCV -- Delete knot, cross validate, and PIMPLE, the PI implementation) and Hastie and Silverman (TURBO)

In short, these routines take the adaptive strategy we mentioned for univariate fitting using the truncated power basis -- A method that's essentially the approach taken by Patricia Smith

Comments

These procedures are aggressively adaptive, identifying both the important variables and their functional form -- They are also useful because they provide simple graphics to examine the different functional components, viewing the main effects and interactions

The selection criteria embedded in these procedures account for the expansive search by increasing the penalties associated with model size (in effect correcting the degrees of freedom used in finding the basis elements)

The truncated power basis is not the only technique for working with “free knot” splines -- Other work attempts to treat the knot locations as another parameter and minimizing their configuration in a less greedy way

Finally, Luo and Wahba have proposed a hybrid selection/shrinkage routine in which knots are placed adaptively and then penalized fit is performed in this space (using the second derivative penalty brought up last lecture)

In short, the 90s was a time of incredible activity in this area!

Fitting (part II)

Using splines, of course, doesn't mean we have to place knots in an adaptive way -- Instead we can just place them at equally spaced quantiles of the input data

This puts them in the same class of smoothers as local polynomials and smoothing splines, at least superficially, in that they have a single "handle" to control the amount of smoothness and the flexibility of these spaces at some point is related to the density input data nearby

In this case, they can, like all these other procedures, be written as a linear smoother with $\hat{y} = Sy$ for a (possibly hard to actually compute) n-by-n matrix S

Fitting (part II)

Buja, Hastie and Tibshirani refer to these as “linear smoothers” in the sense that the predictions at the design points are just linear combinations of the observed data -- Contrast this to the adaptive knot schemes we just presented in which the smoother depends heavily on the input data

These authors present a simple approach to using these “black box” smoothers to fit an additive model -- It involves a process known as backfitting...

Iterative solutions for linear systems

When looking at how to derive OLS estimates, we focused mainly on “direct methods” that involved various matrix decompositions of the model matrix -- The QR decomposition, the SVD and even the LU decomposition implied by Gaussian elimination

In contrast to these direct methods are so-called “iterative methods” which generate a sequence of approximate solutions -- Backfitting builds from one such method, Gauss-Seidel iterations

Iterative approaches

If we are trying to solve a linear system of the form $Ax = b$ where A is a 3-by-3 matrix -- Assuming it has no zero diagonal elements, the solution can be written as

$$
\begin{aligned}
x_1 &= (b_1 - a_{12}x_2 - a_{13}x_3)/a_{11} \\
x_2 &= (b_2 - a_{21}x_1 - a_{23}x_3)/a_{22} \\
x_3 &= (b_3 - a_{31}x_1 - a_{32}x_2)/a_{33}
\end{aligned}
$$

If we have an approximation x^k to the solution, a natural way to generate a new approximation x^{k+1} is to compute

$$
\begin{aligned}
x_1^{k+1} &= (b_1 - a_{12}x_2^k - a_{13}x_3^k)/a_{11} \\
x_2^{k+1} &= (b_2 - a_{21}x_1^k - a_{23}x_3^k)/a_{22} \\
x_3^{k+1} &= (b_3 - a_{31}x_1^k - a_{32}x_2^k)/a_{33}
\end{aligned}
$$

Gauss-Seidel iterations

Gauss-Seidel is only slightly different, making use of the updates as they are available -- That is

$$
\begin{aligned}
x_1^{k+1} &= (b_1 - a_{12}x_2^k - a_{13}x_3^k)/a_{11} \\
x_2^{k+1} &= (b_2 - a_{21}x_1^{k+1} - a_{23}x_3^k)/a_{22} \\
x_3^{k+1} &= (b_3 - a_{31}x_1^{k+1} - a_{32}x_2^{k+1})/a_{33}
\end{aligned}
$$

Another view

For the previous version of our normal linear model we assumed that

$$E[Y|X = x] = \beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p$$

If instead of conditioning on the complete vector X, we use just a single variable, we find that

$$\begin{aligned} E[Y|X_k = x_k] &= E[E[Y|X_1, \ldots, X_k, \ldots, X_p]|X_k = x_k] \\ &= E\left[\sum_{j=0}^{p} \beta_j X_j | X_k = x_k\right] \\ &= \beta_k x_k + E\left[\sum_{j \neq k}^{p} \beta_j X_j | X_k = x_k\right] \end{aligned}$$

Another view

Turning this around we find that

$$\begin{aligned}\beta_k x_k &= E[Y|X_k = x_k] - E\left[\sum_{j\neq k}\beta_j X_j | X_k = x_k\right] \\ &= E\left[Y - \left(\sum_{j\neq k}\beta_j X_j\right) | X_k = x_k\right]\end{aligned}$$

The term in the second line is a partial residual -- It's the difference between Y and what we expect it to be ignoring the kth variable

Backfitting

This equivalence suggests an iterative process in which we cycle through the predictors, regressing the partial residuals on the “left out” variable -- This is known as backfitting

It’s an legitimate alternative to fitting least squares problems (or more generally solving linear systems) especially when we have lots of variables or if our data are sparse in some way

In statistics, this method comes up not just to fit a linear regression but also in the context of fitting an additive model with one of the “linear smoothers” we mentioned earlier...

Center the response and all of predictors

Until the estimates don't change much

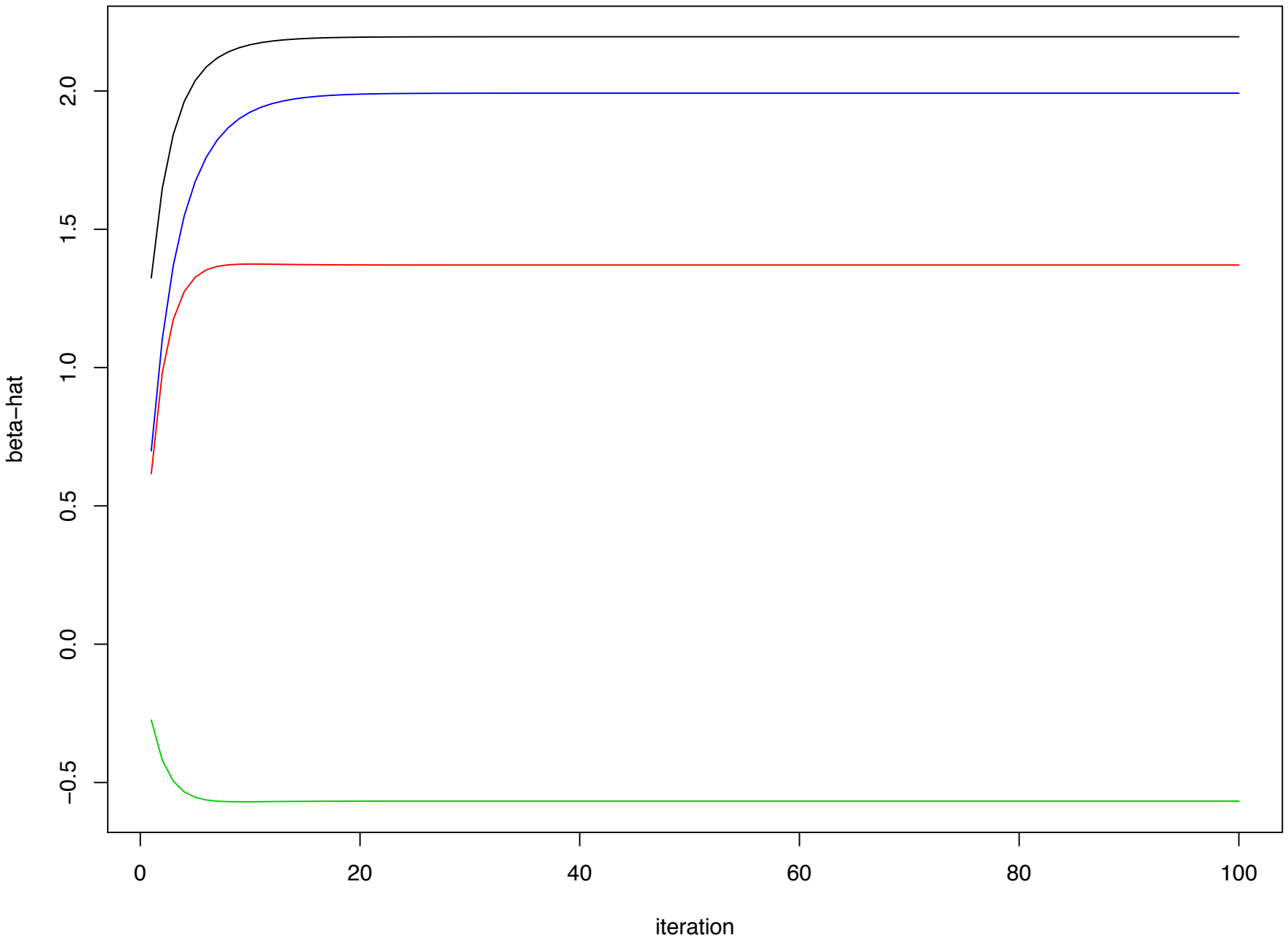
For k = 1,...,p

Form the kth partial residuals $y_k = y - \sum_{j \neq k} \widehat{\beta}_j x_j$

Regress y_k on the kth predictor $x_k = (x_{1k}, \ldots, x_{nk})^t$ and call the coefficient c_k

Set $\widehat{\beta}_k = c_k$

```
# gauss-seidel iterations to fit the vulnerability regression

y <- vul$ln_death_risk
M <- as.matrix(vul[,c("ln_fert","ln_events","ln_pop","hdi")])
M <- M-matrix(apply(M,2,mean),ncol=ncol(M),nrow=nrow(M),byrow=T)

y <- y-mean(y)
beta <- rep(0,ncol(M))

# store all the coefficients as we iterate (not strictly needed
# but we want a pretty picture at the end)
Beta <- NULL

for(i in 1:100){

   for(k in 1:ncol(M)){

      yk <- y - M[,-k]%*%beta[-k]
      ck <- coefficients(lm(yk~M[,k]-1))
      beta[k] <- ck
   }
   Beta <- rbind(Beta,beta)
}

matplot(1:100,Beta,type="l",xlab="iteration",ylab="beta-hat",lty=1)
```


beta−hat
2.0
1.5
1.0
0.5
0.0
−0.5
0
20
40
60
80
100
iteration

Backfitting and the additive model

If we really believed that our regression function has an additive form

$$E[Y|X = x] = \alpha + f(x_1) + \cdots + f(x_p)$$

If instead of conditioning on the complete vector X, we use just a single variable, we find that

$$\begin{aligned} E[Y|X_k = x_k] &= E[E[Y|X_1, \ldots, X_k, \ldots, X_p]|X_k = x_k] \\ &= E\left[\alpha + \sum_{j=1}^{p} f(X_j)|X_k = x_k\right] \\ &= f(x_k) + E\left[\alpha + \sum_{j\neq k}^{p} f(X_j)|X_k = x_k\right] \end{aligned}$$

Another view

Turning this around we find that

$$\begin{aligned} f(x_k) &= E[Y|X_k = x_k] - E\left[\alpha + \sum_{j \neq k} f(X_j)|X_k = x_k\right] \\ &= E\left[Y - \left(\alpha + \sum_{j \neq k} f(X_j)\right)|X_k = x_k\right] \end{aligned}$$

The term in the second line is a partial residual -- It's the difference between Y and what we expect it to be ignoring the kth variable

Backfitting and the additive model

Therefore, if we could estimate arbitrary one-dimensional functions we could fold that into a backfitting routine to estimate an additive model -- Ah, but here's where our linear smoothers come into play!

This is a second sense in which fitting an additive model is "no harder" than a simple univariate smooth -- We are just applying the same tool over and over again...

Center the response and all of predictors and form the smoother matrices $S_1, \ldots, S_p$

Until the estimates don't change much

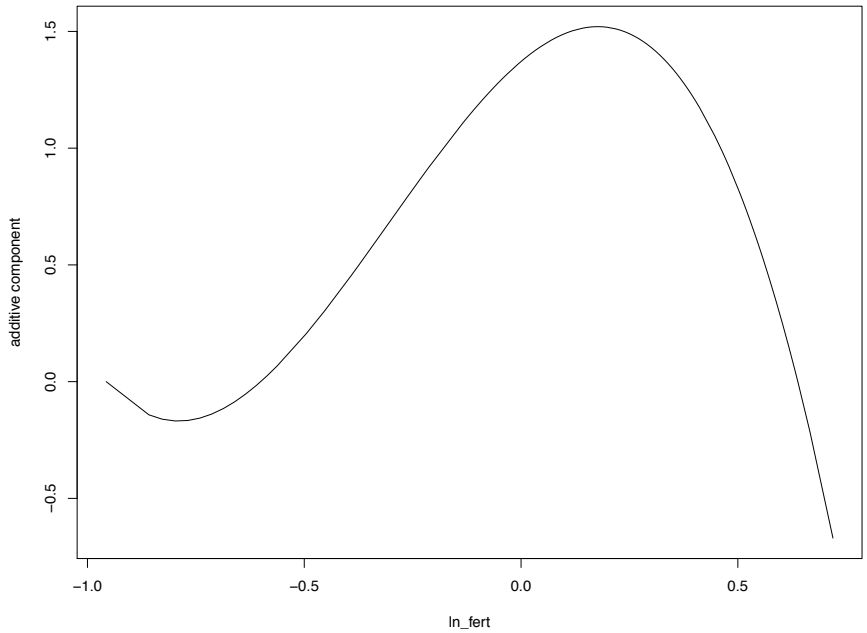
For k = 1,...,p

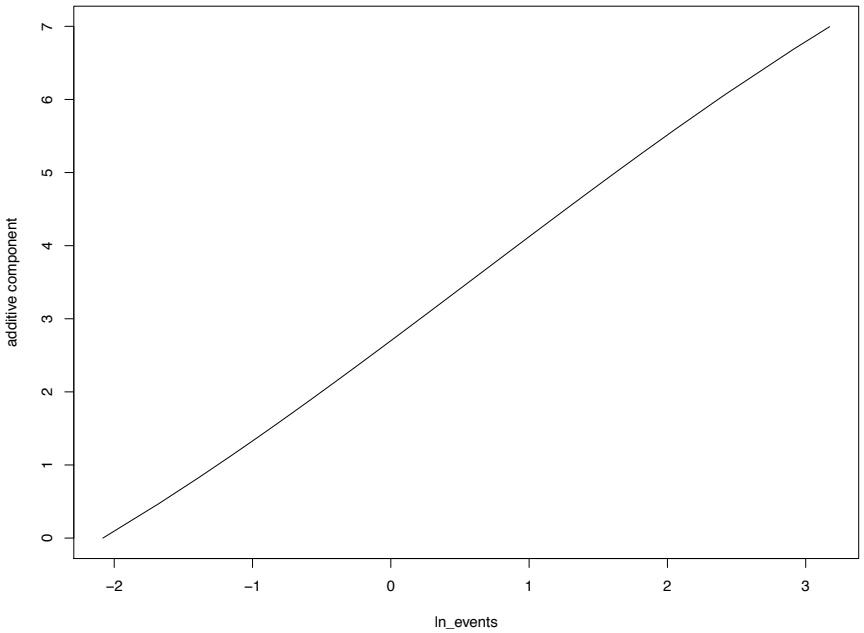
Form the kth partial residuals $y_k = y - \sum_{j \neq k} \widehat{f}_j$

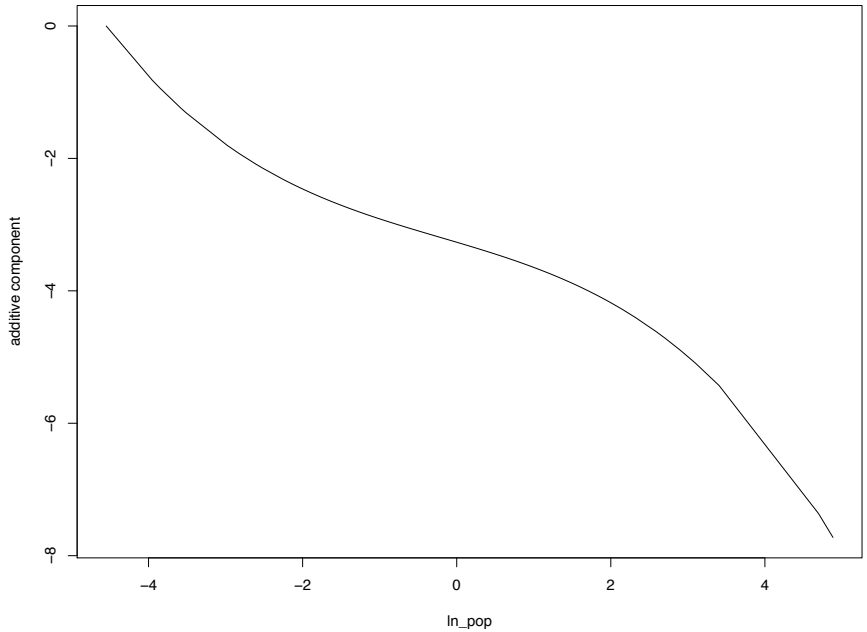
Smooth y_k using the kth predictor $x_k = (x_{1k}, \ldots, x_{nk})^t$ so that $s_k = S_k y_k$

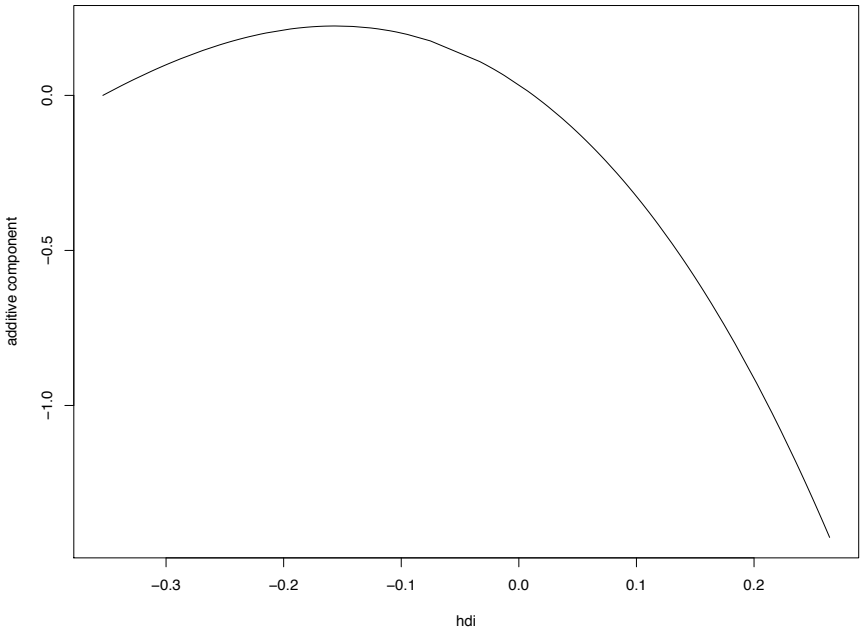
Set $\widehat{f}_k = s_k$

```
y <- vul$ln_death_risk
M <- as.matrix(vul[,c("ln_fert","ln_events","ln_pop","hdi")])
M <- M-matrix(apply(M,2,mean),ncol=ncol(M),nrow=nrow(M),byrow=T)

y <- y-mean(y)
fits <- matrix(0,ncol=ncol(M),nrow=nrow(M))

for(i in 1:1000){

   for(k in 1:ncol(M)){

      yk <- y - apply(fits[,-k],1,sum)
      fk <- predict(lm(yk~bs(M[,k],df=3)-1))
      fits[,k] <- fk
   }
}

plot(sort(M[,1]),fits[order(M[,1]),1],
     xlab="ln_fert",ylab="additive component",type="l")

plot(sort(M[,2]),fits[order(M[,2]),2],
     xlab="ln_events",ylab="additive component",type="l")

plot(sort(M[,3]),fits[order(M[,3]),3],
     xlab="ln_pop",ylab="additive component",type="l")

plot(sort(M[,4]),fits[order(M[,4]),4],
     xlab="hdi",ylab="additive component",type="l")
```


additive component
1.5
1.0
0.5
0.0
−0.5
−1.0
−0.5
0.0
0.5
ln_fert


additive component
7
6
5
4
3
2
1
0
−2
−1
0
1
2
3
ln_events


additive component
0
−2
−4
−6
−8
−4
−2
0
2
4
ln_pop


additive component
0.0
−0.5
−1.0
−0.3
−0.2
−0.1
0.0
0.1
0.2
hdi