Lecture 8

Connections

We're now going to make a deeper connection between these methods -- Specifically we'll consider the relationship between subset selection and the lasso

This will lead us to something called least angle regression, a relatively recent (say, 2002) development in modeling technology...

Forward stagewise modeling

Suppose we have a model g that involves a so-called basis expansion

$$g(x;\beta,\gamma) = \sum_{k=1}^{K} \beta_k B(x;\gamma_k)$$

where the B's are basis elements (think of them as your original variables from an OLS fit or maybe polynomial terms or -- later in the quarter -- spline terms or trees) and they in turn depend (possibly) on some auxiliary parameters γ_k

Then, given data $y_i, (x_{i1}, \ldots, x_{ip})^t$, i=1,..,n, we would like to select the parameters β_k, γ_k to minimize some loss function

$$\sum_{i=1}^{n} L\big(y_i, g(x_i;\beta,\gamma)\big)$$

Forward stagewise modeling approximates the solution to this problem by sequentially adding new basis functions to the expansion without adjusting the parameters and coefficients of those already in the fit

Forward stagewise modeling

That was a bit opaque -- While a stagewise approach has connections to optimization, it might be better to consider its action first and then talk about what it's trying to do

Stagewise modeling constructs a model in small steps...

Forward stagewise modeling

Initialize by standardizing the predictors $\widetilde{x}_{ij} = (x_{ij} - \bar{x}_j)/\mathrm{sd}(x_j)$ (center and scale to norm 1) and centering the responses $\widetilde{y}_i = y_i - \bar{y}$ -- Then set

$$\widehat{\mu} = (0, \ldots, 0)^t \quad r = \widetilde{y}, \text{ and } \widehat{\beta}_1 = \cdots, \widehat{\beta}_p = 0$$

Repeat a large number of times

Find the predictor $\widetilde{x}_j$ most correlated with r

Set $\delta = \epsilon \ \mathrm{sign}(\widetilde{x}_j^t r)$ and form the updates

$$\widehat{\beta}_j \leftarrow \widehat{\beta}_j + \delta$$

$$\widehat{\mu} \leftarrow \widehat{\mu} + \delta \widetilde{x}_j$$

$$r \leftarrow r - \delta \widetilde{x}_j$$

Forward stagewise modeling

Because of the standardization at the first step, each of the correlations in question are just inner products $\widetilde{x}_j^t r$ for j=1,...,p -- Also the correlation criterion is equivalent to choosing the variable that would produce the largest drop in the sum of squares of r

Now, at each step selecting $\epsilon = |\widetilde{x}_j^t r|$ would advance us to the next regression involving $\widetilde{x}_j$ in one step -- In short, this “big” value of ϵ **reduces forward stagewise modeling to the classic forward stepwise procedure**

The idea behind stagewise procedures is that the big-step greedy nature of stepwise approaches can make them notoriously unstable -- Small changes in the data, for example, can create different paths on the model tree

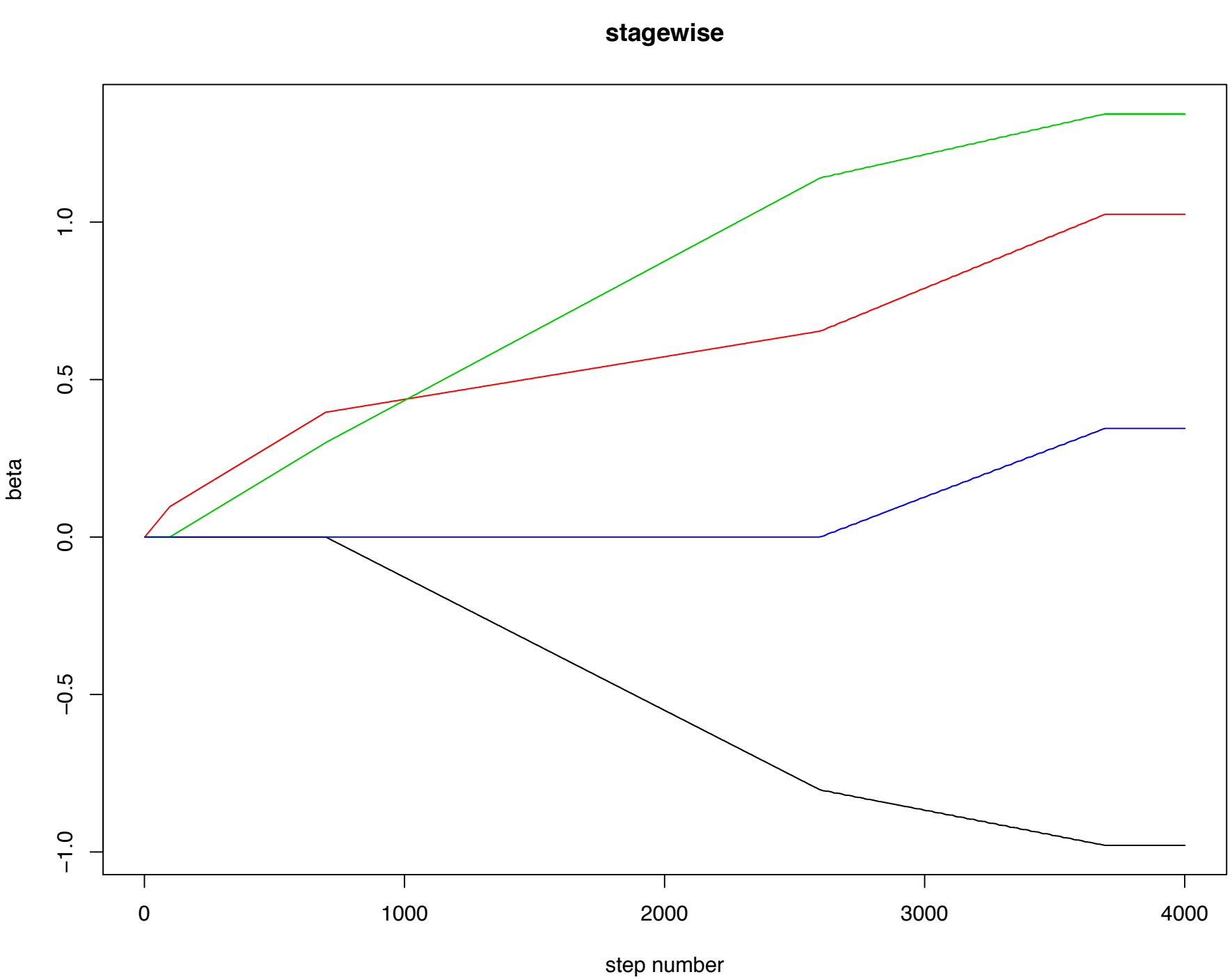
Forward stagewise modeling

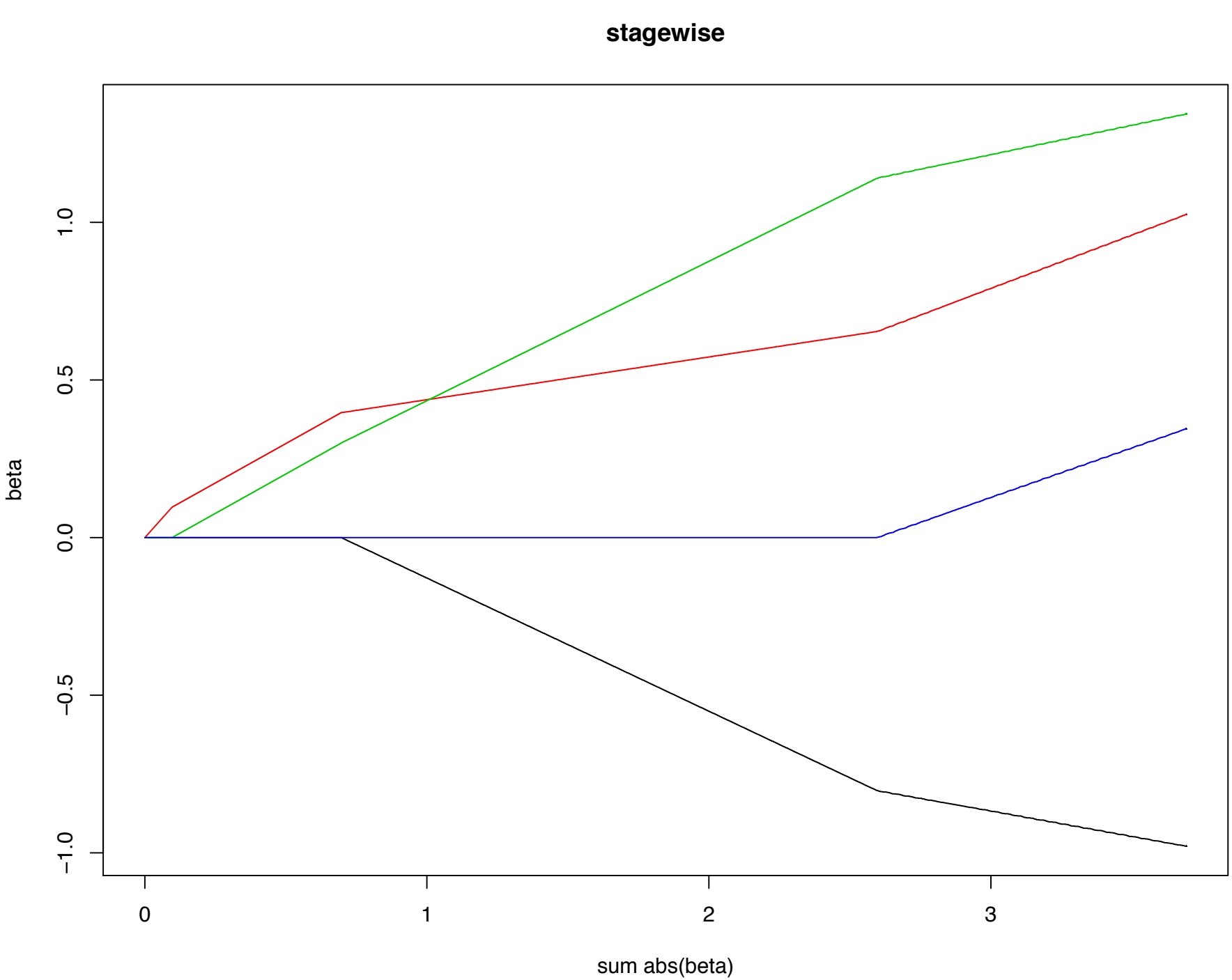
This approach to modeling was inspired by another relatively recent advance in machine learning -- It's basic idea is to combine many "weak" models to build a powerful "committee"

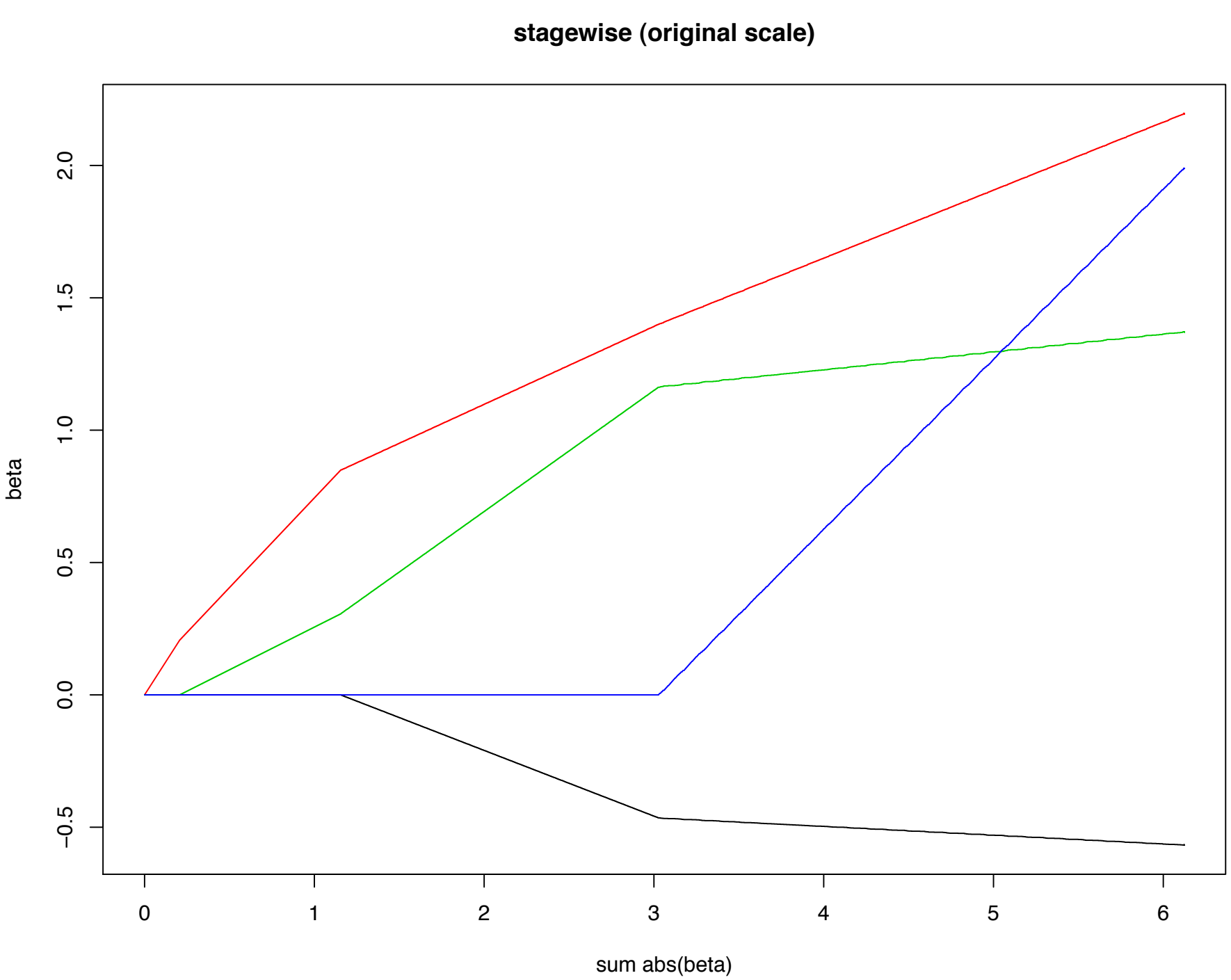
In short, we apply our "learning scheme" (here finding the single variable that best describes the data) successively to the residuals from our current model, our current set of errors

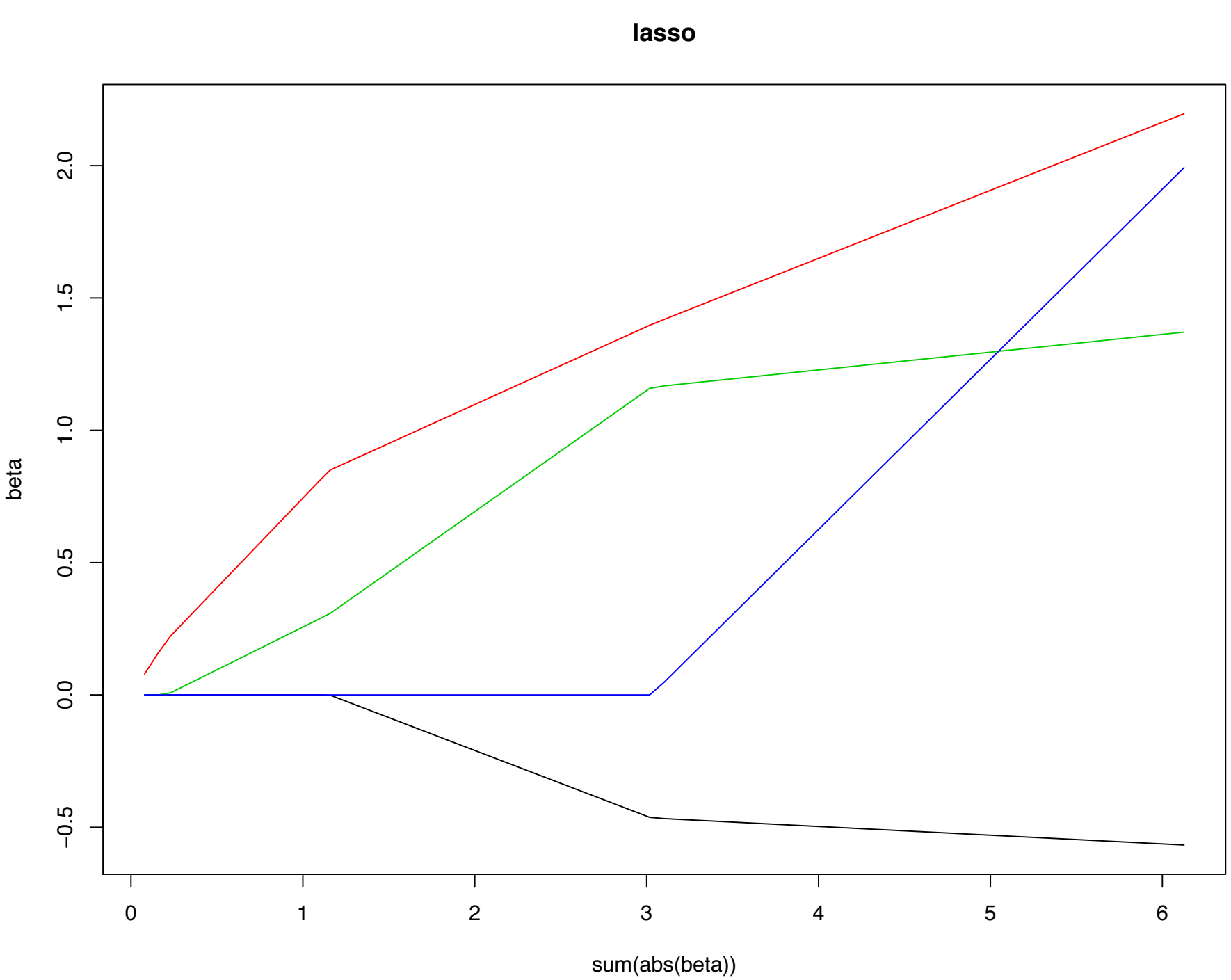
You'll also see boosting referred to as "slow learning" and from this stagewise example you can see why -- Now, let's have a look at what this does in practice... Any ideas?

```
y <- y-mean(y)
M <- M-matrix(apply(M,2,mean),ncol=ncol(M),nrow=nrow(M),byrow=T)
M <- M/matrix(apply(M,2,sd),ncol=ncol(M),nrow=nrow(M),byrow=T)

beta <- matrix(0,ncol=ncol(M),nrow=1)
r <- y
eps <- 0.001

lots <- 4000

for(i in 1:lots){

   co <- t(M)%*%r
   j <- (1:ncol(M))[abs(co)==max(abs(co))][1]
   delta <- eps*sign(co[j])

   b <- beta[nrow(beta),]

   b[j] <- b[j] + delta
   beta <- rbind(beta,b)

   r <- r - delta*M[,j]
}

matplot(beta,type="l",lty=1,xlab="step number",ylab="beta",main="stagewise")

matplot(apply(beta,1,function(x) sum(abs(x))),
      beta,type="l",lty=1,xlab="sum abs(beta)",ylab="beta",main="stagewise")

matplot(apply(
        beta/matrix(s,ncol=ncol(beta),nrow=nrow(beta),byrow=T),1,function(x) sum(abs(x))),
        beta/matrix(s,ncol=ncol(beta),nrow=nrow(beta),byrow=T),
        type="l",lty=1,xlab="sum abs(beta)",ylab="beta",main="stagewise (original scale)")
```


stagewise
beta
1.0
0.5
0.0
−0.5
−1.0
0
1000
2000
3000
4000
step number


stagewise
beta
1.0
0.5
0.0
−0.5
−1.0
0
1
2
3
sum abs(beta)


stagewise (original scale)
beta
2.0
1.5
1.0
0.5
0.0
−0.5
0
1
2
3
4
5
6
sum abs(beta)


lasso
beta
2.0
1.5
1.0
0.5
0.0
−0.5
0
1
2
3
4
5
6
sum(abs(beta))

A connection

It would appear that the two paths are nearly (if not exactly) the same, if only because the stagewise approach is jagged -- This is a remarkable fact that allows us to gain insight into a variety of properties of the lasso

For orthogonal problems, the two paths are guaranteed to be the same -- To be precise, if you could imagine taking a very large number of infinitesimally small steps, you would end up walking the same path as the lasso

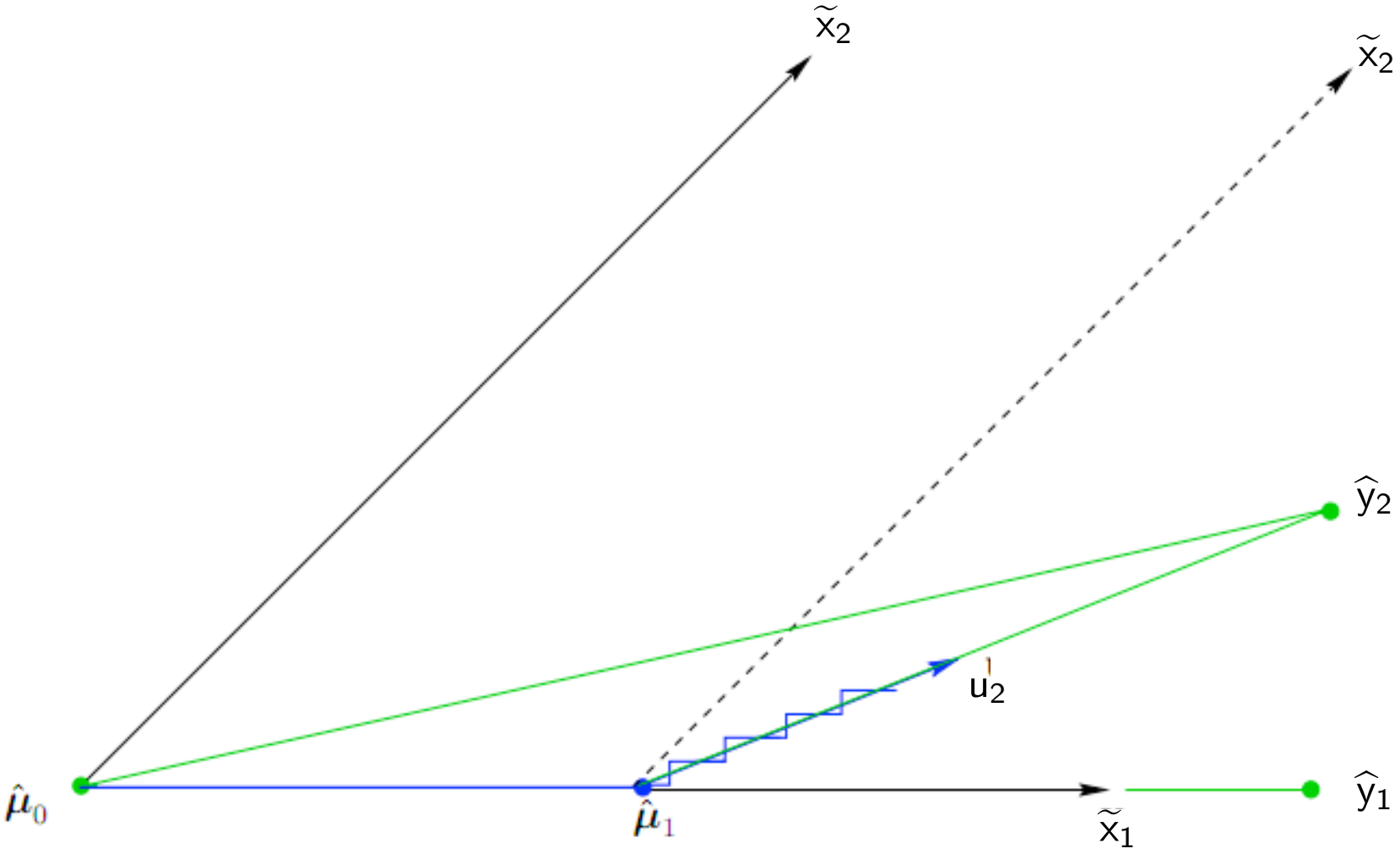
Stagewise, a closer look

On the next page we have a simple diagram of how the algorithm works (at least in 2d) -- We let $\widehat{y}_2$ be the projection of our data y into the space spanned by $\widetilde{x}_1$ and $\widetilde{x}_2$

We start with $\widehat{\mu}_0 = 0$ and $r_0 = \widehat{y}_2 - \widehat{\mu}_0$ and consider the variable ($\widetilde{x}_1$ or $\widetilde{x}_2$) that has the greatest correlation with r_0 -- In this case, it's $\widetilde{x}_1$ and we take steps in that direction

Classical stepwise regression would take a big step and move from $\widehat{\mu}_0$ to $\widehat{y}_1$, the projection of y onto the space spanned by the vector $\widetilde{x}_1$ -- The stagewise approach is different in that it will only go as far as $\widehat{\mu}_1$

At that point, the correlation between $\widehat{y}_2 - \widehat{\mu}_1$ and each of $\widetilde{x}_1$ and $\widetilde{x}_2$ is the same and we let u_2 denote the unit vector lying along the bisector -- The procedure marches stepwise along this path until it reaches $\widehat{y}_2$


$\widetilde{x}_2$
$\widetilde{x}_2$
$\widehat{y}_2$
u_2
$\hat{\mu}_0$
$\hat{\mu}_1$
$\widetilde{x}_1$
$\widehat{y}_1$

Least angle regression

LARS is motivated by the fact that it's relatively easy to identify the points in the figure where the "infinitesimal" version of the stagewise procedure would turn a corner and start following a new vector

The LARS procedure works (essentially) as follows -- We start with all the coefficients in our model set to zero and we look for the variable $\widetilde{x}_{j_1}$ that is most correlated (makes the smallest angle) with our response y

We then take the largest step possible in the direction $\widetilde{x}_{j_1}$ until some other predictor, $\widetilde{x}_{j_2}$, has as much correlation with the accompanying residual -- At that point LARS switches to a direction that is "equiangular" between the two predictors $\widetilde{x}_{j_1}$ and $\widetilde{x}_{j_2}$

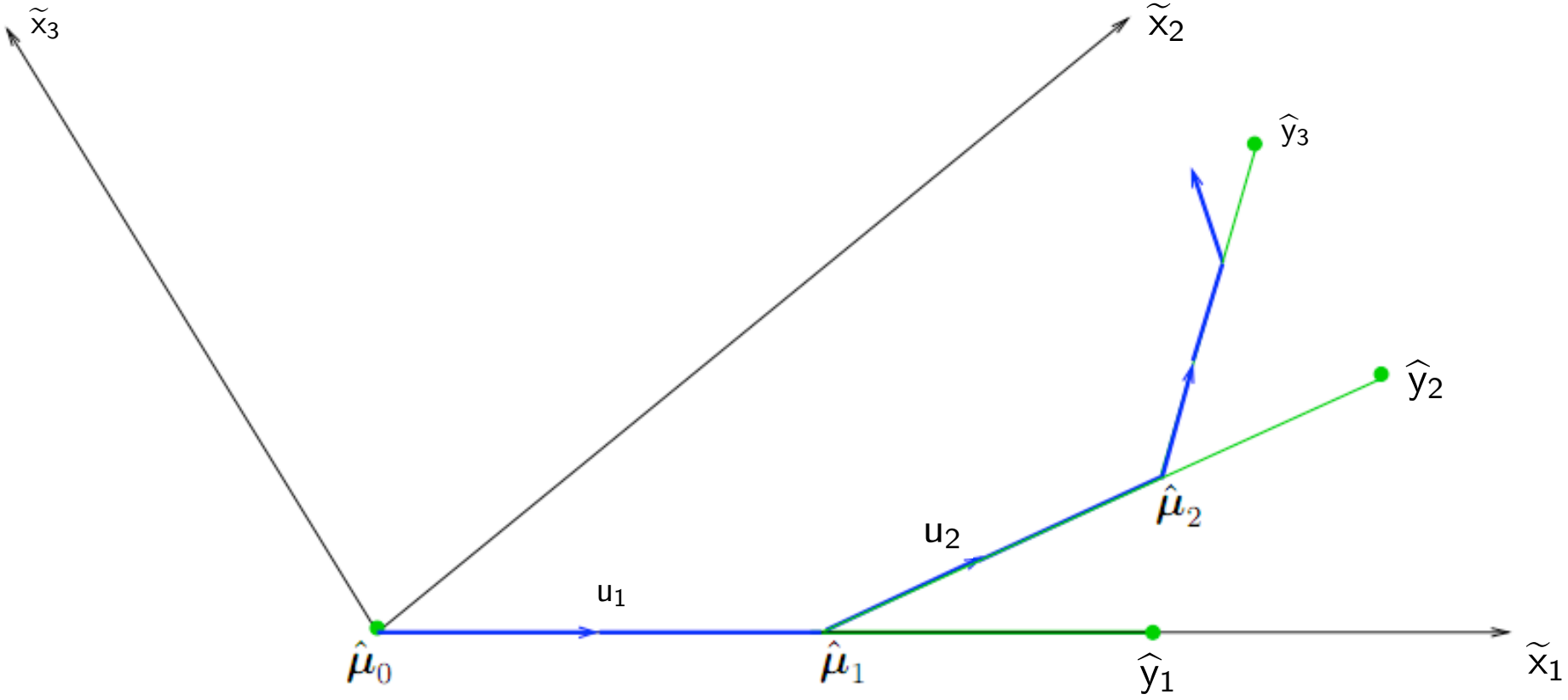
We continue in this way until a third variable $\widetilde{x}_{j_3}$ joins the "most correlated" set and we move equiangularly between $\widetilde{x}_{j_1}, \widetilde{x}_{j_2}$ and $\widetilde{x}_{j_3}$ -- If we had p original predictors we continue in this way for p-1 steps and take as our last move a jump to the OLS fit using all p predictors

Least angle regression

Without getting into too many details, suppose we have completed k-1 steps of the LARS algorithm -- The kth step will see us introduce a new variable to the set and we can label them $x_{j_1}, \ldots, x_{j_k}$

One can show that the LARS estimate for the kth step $\widehat{\mu}_k$ lies along the line between $\widehat{\mu}_{k-1}$ and $\widehat{y}_k$, the OLS fit to the response using $x_{j_1}, \ldots, x_{j_k}$

In this sense, the LARS estimate is always approaching, but never reaching the OLS estimates at each stage


$\widetilde{x}_3$
$\widetilde{x}_2$
$\widehat{y}_3$
$\widehat{y}_2$
$\hat{\mu}_2$
u_2
u_1
$\hat{\mu}_0$
$\hat{\mu}_1$
$\widehat{y}_1$
$\widetilde{x}_1$

Some facts about least angle regression

First, it is computationally “thrifty” in the words of Efron and company -- It is said to be no harder than fitting a full least squares solution to all p input variables

Next, LARS can be modified so that it can produce the complete paths for both stagewise regression and the lasso -- This has the effect of both providing each with a computationally efficient algorithm as well as offering insight into their operations (LARS moves along a “compromise” direction, equiangular, while the lasso and stagewise restrict strategy in some way)

Finally, the “tuning” parameter for a LARS fit is the number of steps k -- To choose k, Efron et al appeal to a Cp statistic where the “degrees of freedom” for a LARS model after k steps is, well, just k

$$C_p(k) = \frac{RSS}{n} + \frac{2k}{n}\widehat{\sigma}^2$$

where k is the number of steps and $\widehat{\sigma}^2$ is the estimated error variance (from the full model, say if n > p)

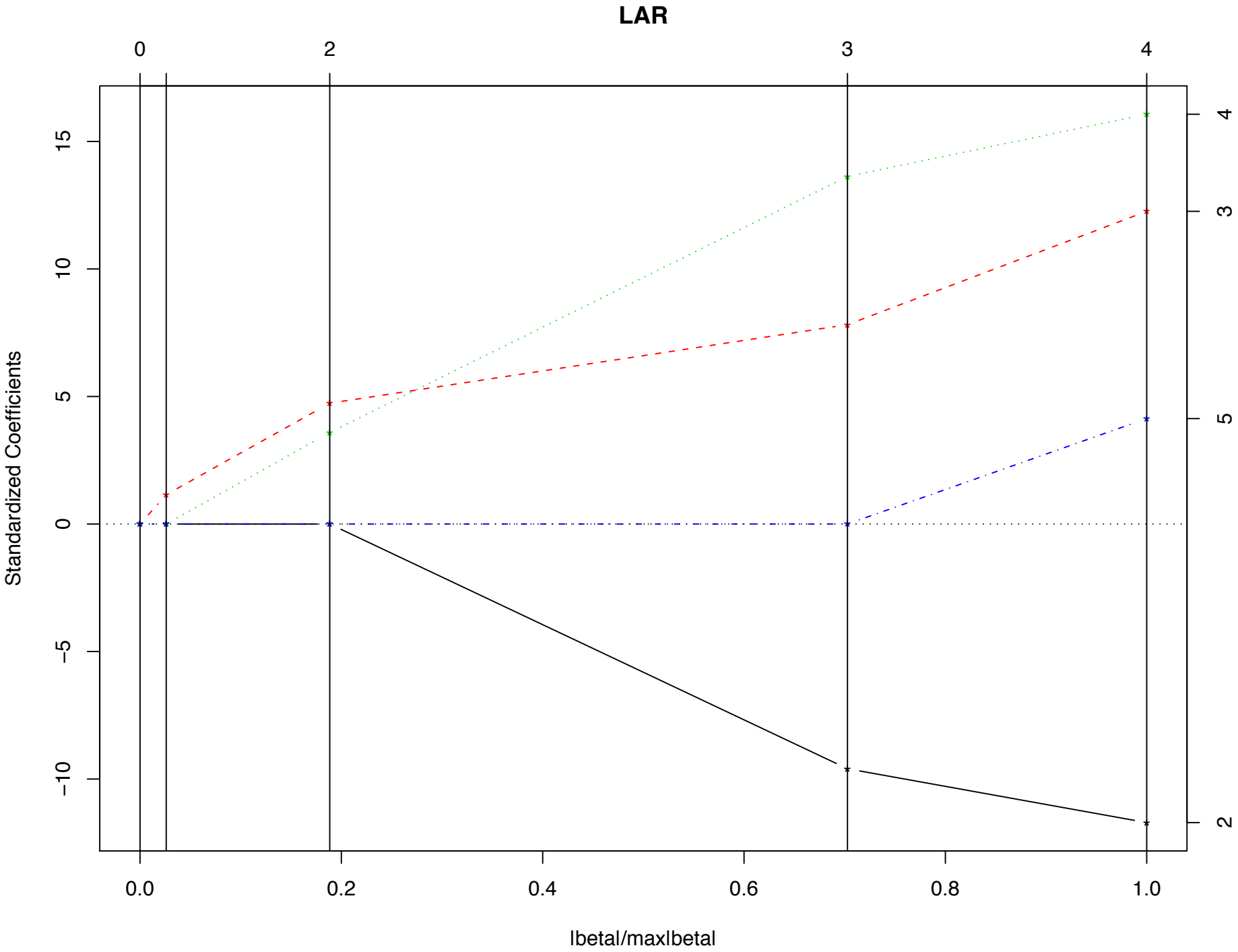
Degrees of freedom

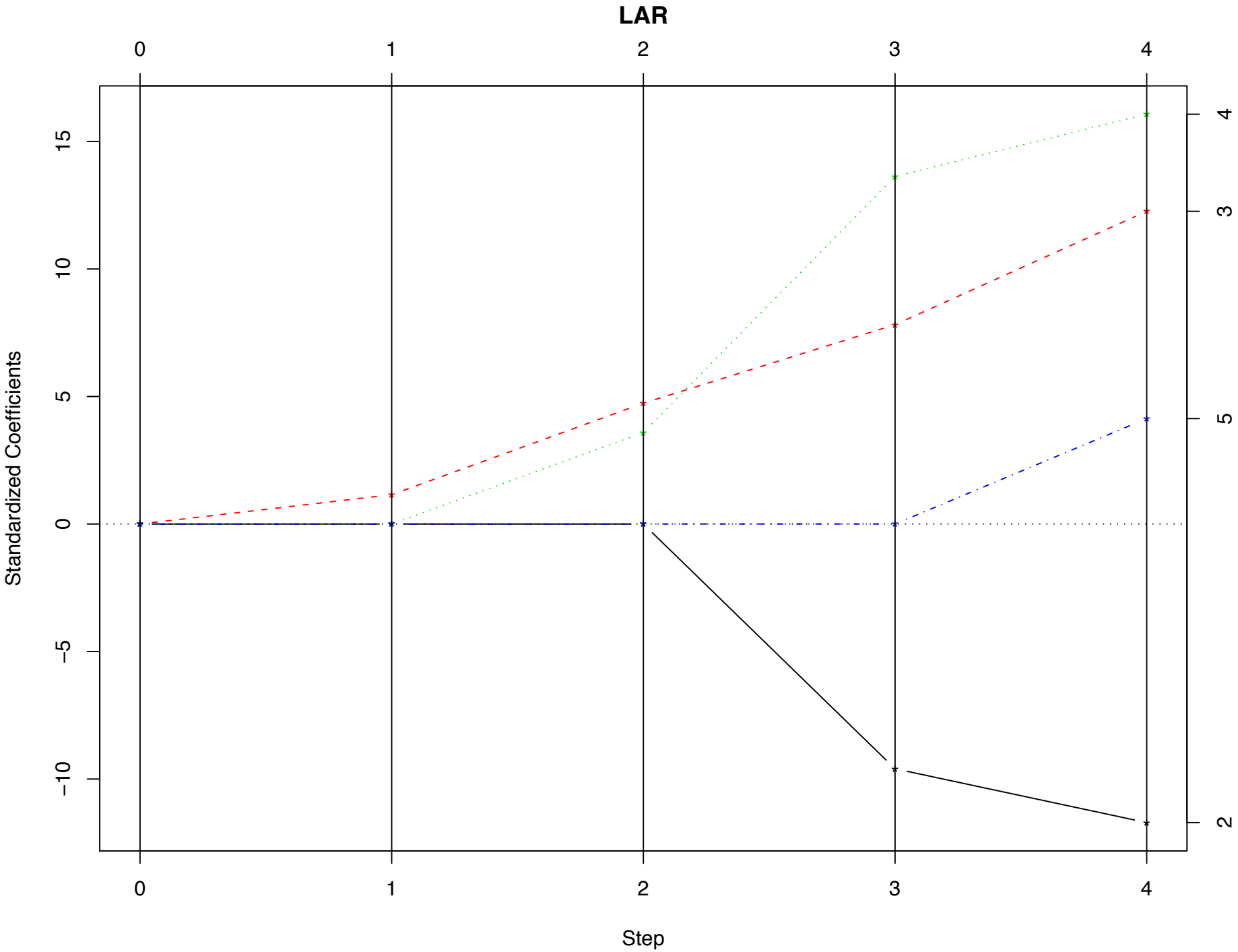
The simple expression for the degrees of freedom in the fit comes from a simulation experiment in which the optimism

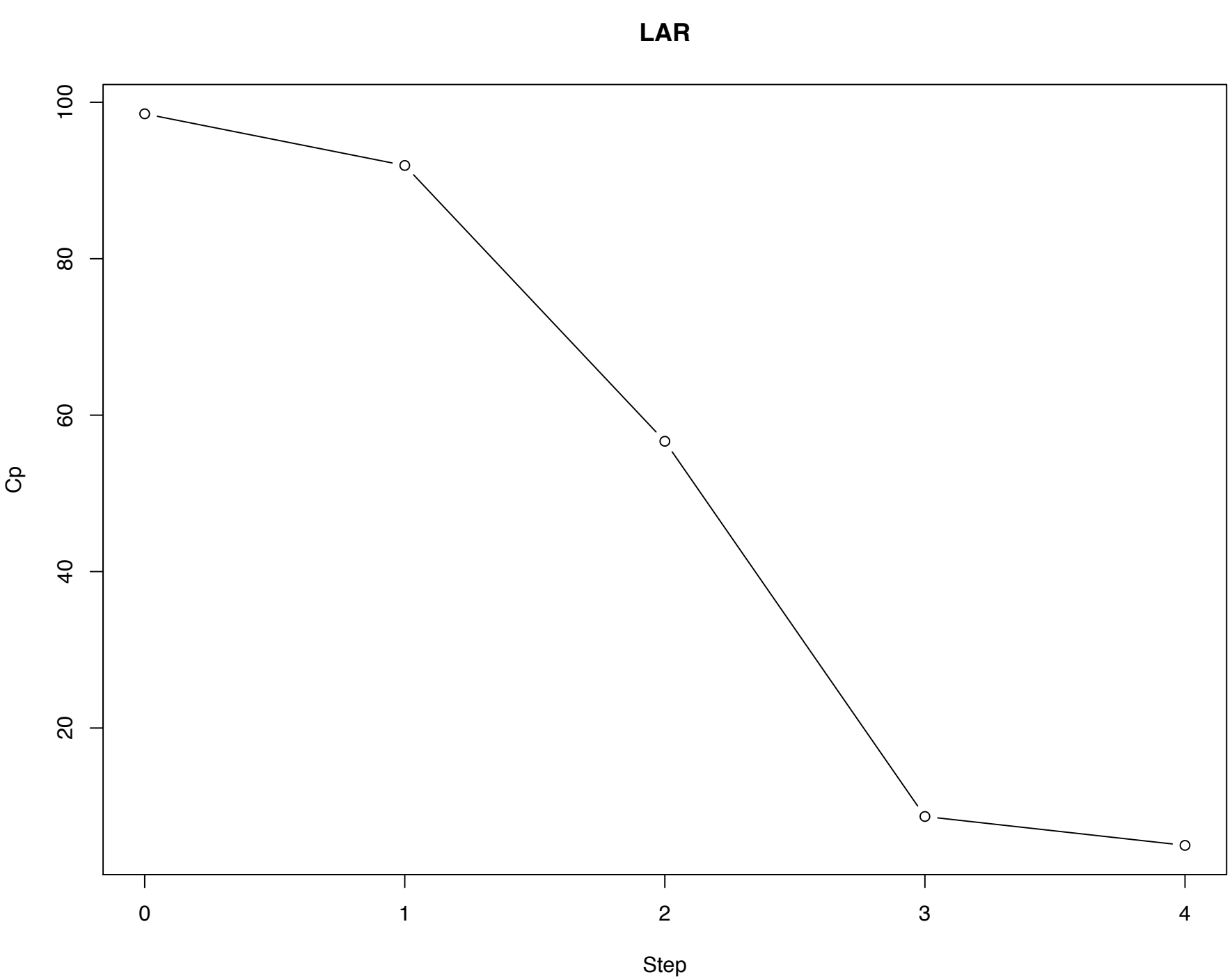
$$\frac{2}{n}\sum_{i=1}^{n}\mathrm{cov}\left(y_i, \widehat{\mu}_{ik}\right)$$

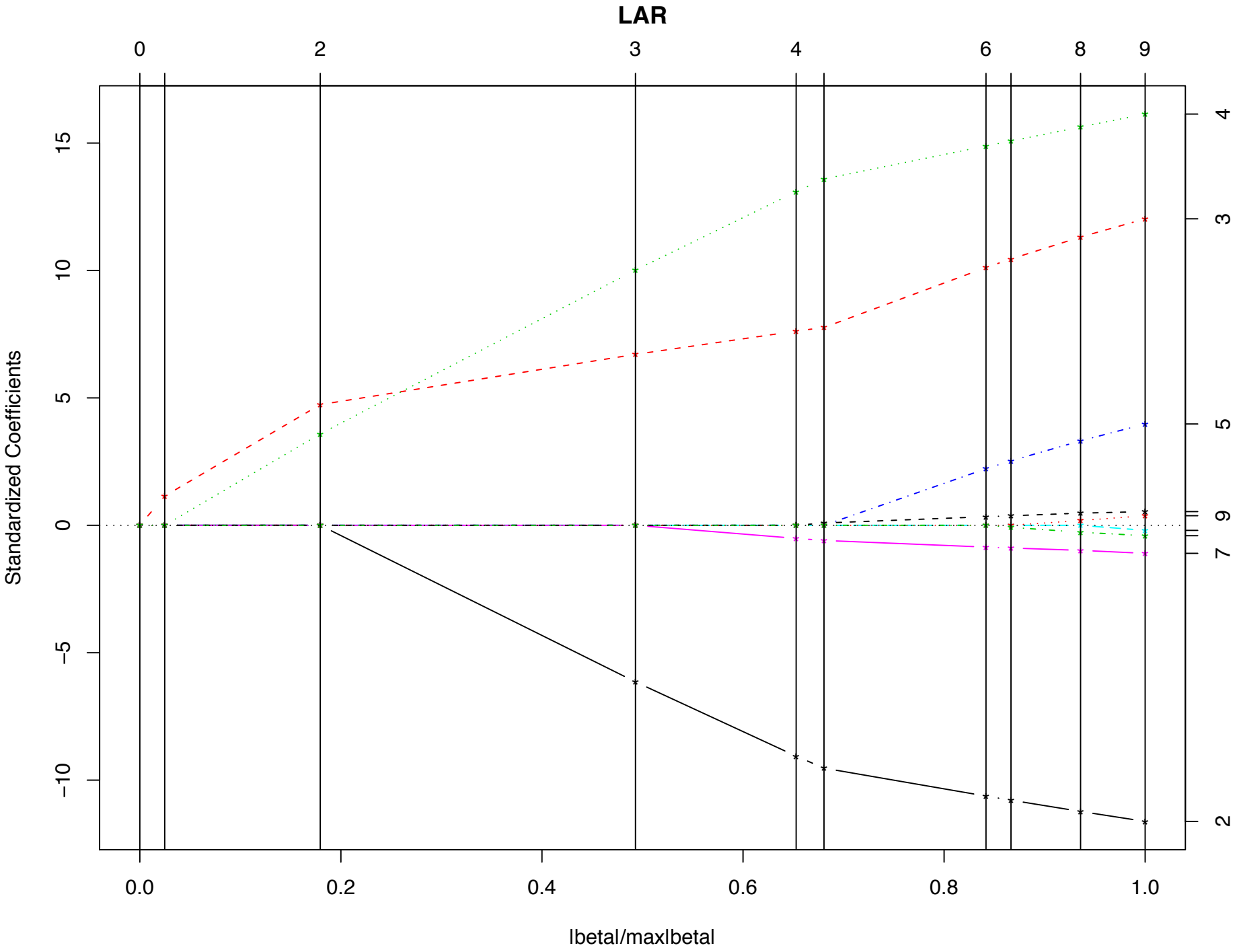
was estimated via the bootstrap (later) and then plots were made against k -- The result can be established theoretically for orthogonal models and for designs satisfying certain (easily checked) properties

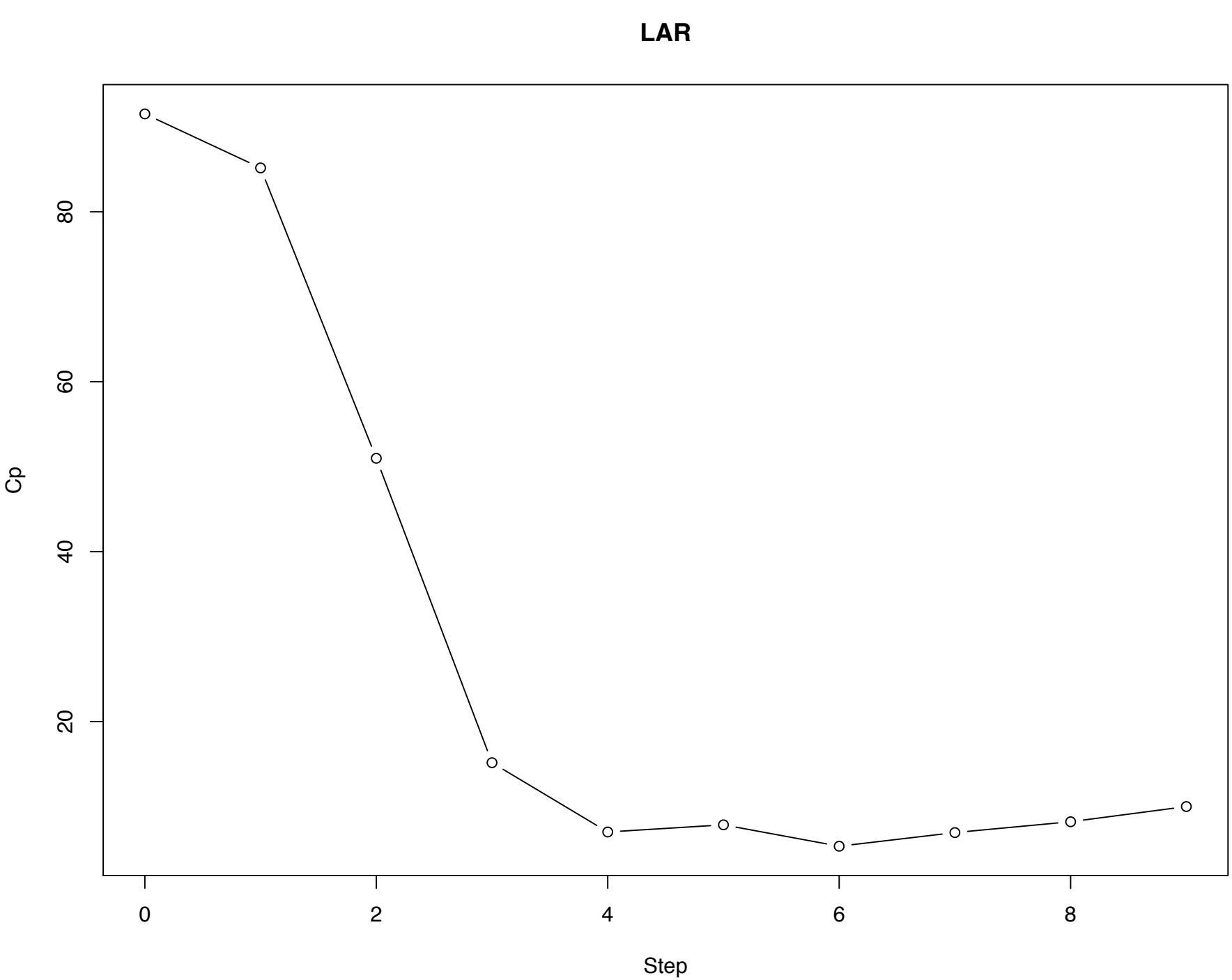
```
library(lars)

M <- model.matrix(ln_death_risk~ln_pop+ln_fert+ln_events+hdi,data=vul)
y <- vul$ln_death_risk

fit <- lars(M,y,"lar")

plot(fit)
plot(fit,xvar="step")

plot(fit,plottype="Cp",xvar="step")

# add noise variables

M <- cbind(M,matrix(rnorm(nrow(M)*5),ncol=5))
fit <- lars(M,y,"lar")

plot(fit)
plot(fit,plottype="Cp",xvar="step")
```


LAR
0
2
3
4
15
10
5
0
−5
−10
Standardized Coefficients
4
3
5
2
0.0
0.2
0.4
0.6
0.8
1.0
|beta|/max|beta|


LAR
0
1
2
3
4
15
10
5
0
−5
−10
Standardized Coefficients
4
3
5
2
0
1
2
3
4
Step


LAR
Cp
100
80
60
40
20
0
1
2
3
4
Step


LAR
0
2
3
4
6
8
9
Standardized Coefficients
15
10
5
0
−5
−10
4
3
5
9
7
2
0.0
0.2
0.4
0.6
0.8
1.0
|beta|/max|beta|


LAR
Cp
80
60
40
20
0
2
4
6
8
Step

To sum

Stepwise regression: Pick the predictor most closely correlated with y and add it completely to the model

Forward stagewise: Pick the predictor most correlated with y and increment the coefficient for that predictor

Least angle regression: Pick the predictor most correlated with y but add it to the model only to the extent that it has greater correlation than the others -- Move in the least-squares direction until another variable is as correlated

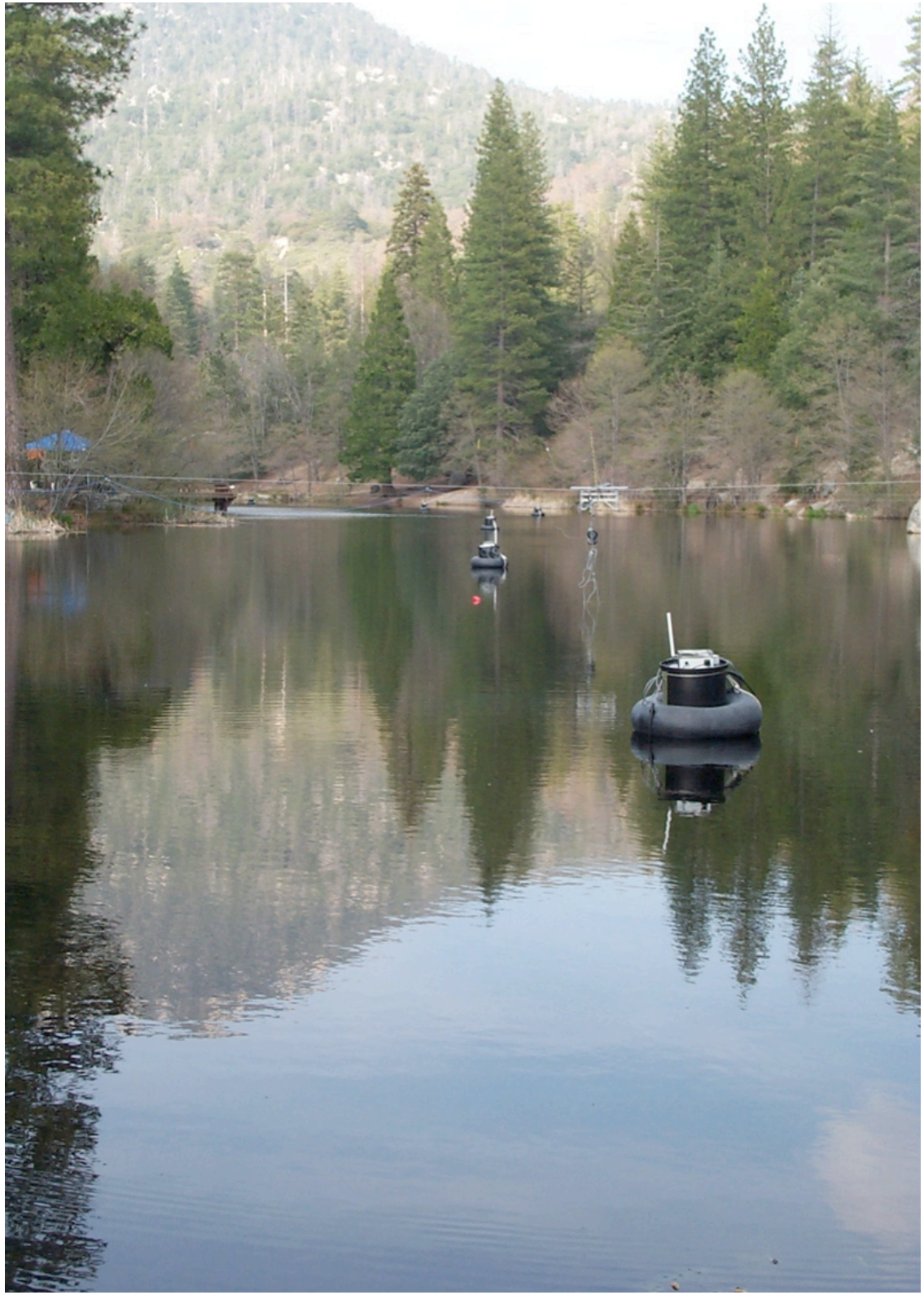
Quo vadis?

We started with ordinary least squares as a methodology -- We studied some of its basic properties, developing an intuition for the various pieces that went into the fitting procedure

We discussed some diagnostics that helped us assess when things might be going wrong -- Perhaps we need to add something to a model or consider dropping something

We then looked at penalization and how that affects a fit, first from the standpoint of stability and then from pure predictive power -- We saw two different kinds of penalization that resulted in very different kinds of fits

We finally made a connection between these penalties and a relatively new procedure for model building called least angle regression -- Ultimately our geometric perspective came in handy here!

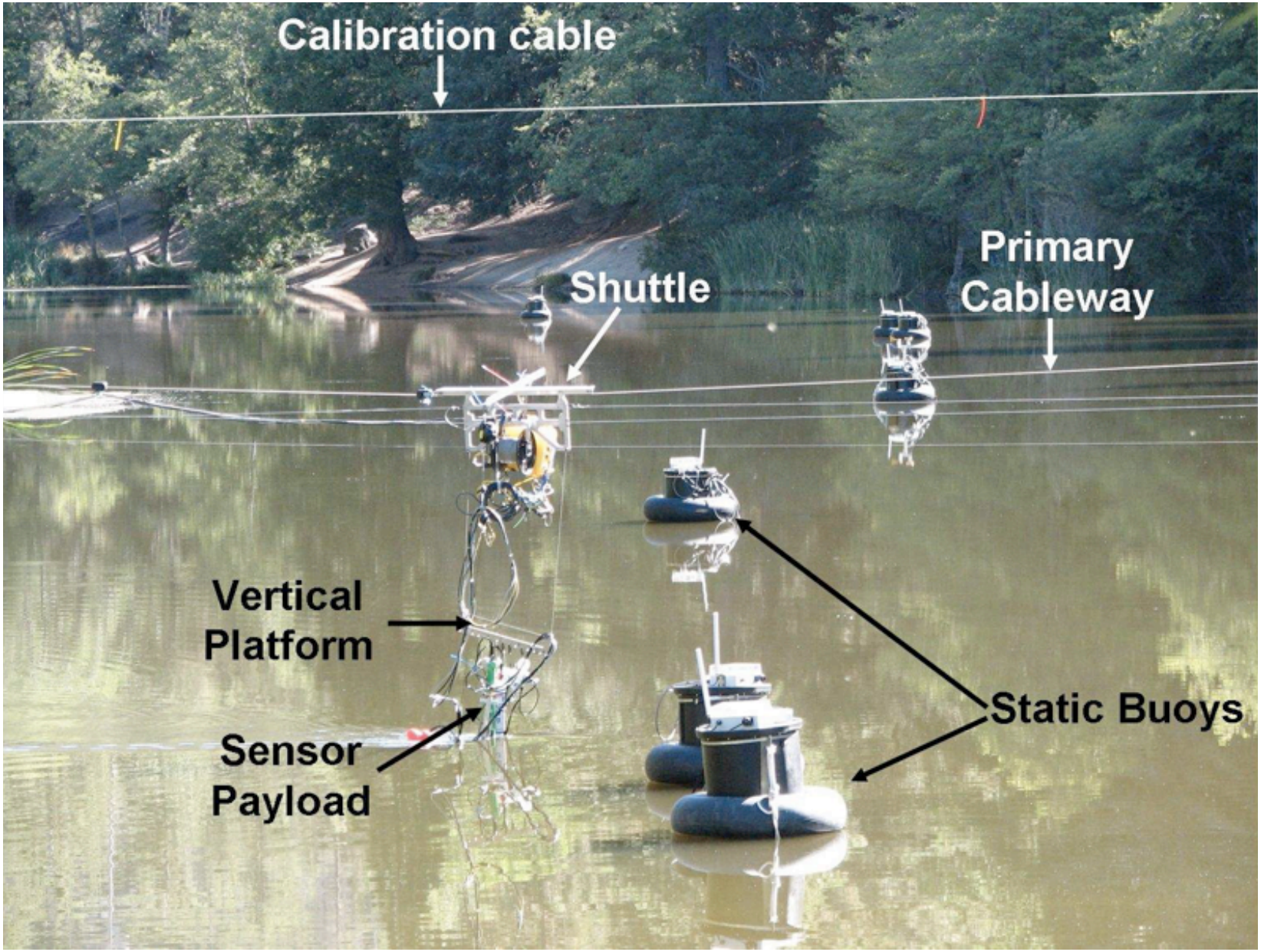


Dynamics of an urban lake

This is Lake Fulmor, located in the San Jacinto Mountains; it is adjacent to James Reserve, part of the UC Natural Reserve System

James Reserve runs several testbeds for embedded sensing systems; sensors have been deployed throughout the area to study microclimates, plant phenology, climate change, you name it

The data we will study come from a week-long deployment that attempted to assess the dynamics of various biological and chemical processes in the lake


Calibration cable
Shuttle
Primary
Cableway
Vertical
Platform
Sensor
Payload
Static Buoys

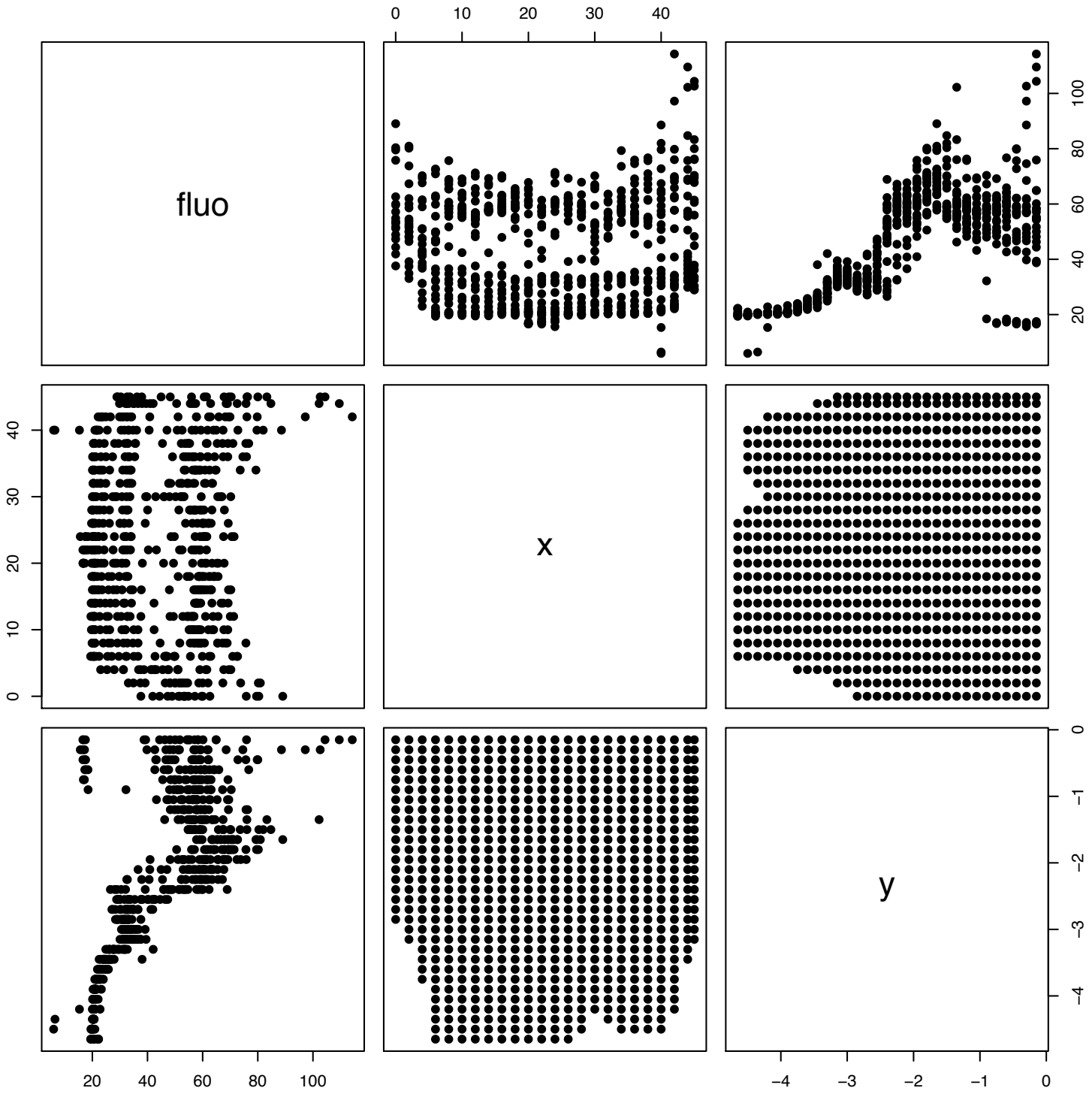
Dynamics of an urban lake

The measurements were collected by a robotic sensing system, producing a vertical profile of about 10 different kinds of measurements

The robot itself consisted of two parts: A shuttle that rode along a tether stretched across the lake, and a sensor pack that could be lowered into the water at any depth

Over the course of an hour, the robot visited 685 different locations; we will focus on data from the fluorometer, a device that responds to chlorophyll levels... here are some plots

```
# 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)
```

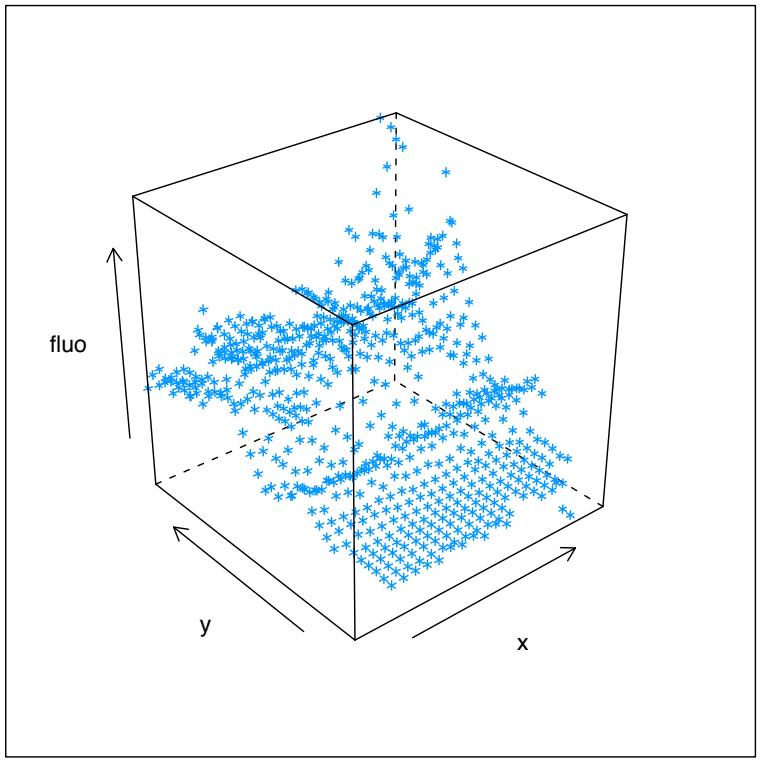

fluo
x
y
0
10
20
30
40
20
40
60
80
100
−4
−3
−2
−1
0

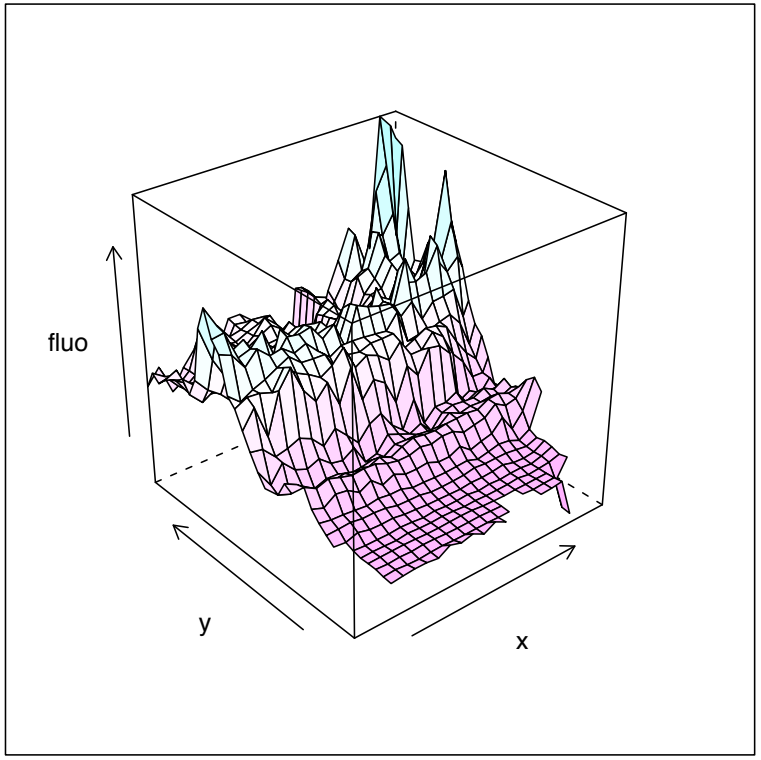
Dynamics of an urban lake

The variable x denotes distance along the lake, and y represents depth below the surface; we see from the pairs plot that the locations were part of a (mostly) regular grid

The plots on the right show that the chlorophyll levels start low near the surface, peak at about 2m deep, and 100 then trail off as you approach the bottom of the lake

It turns out that this pattern is chlorophyll is expected, in that it closely matches the thermal profile in the lake

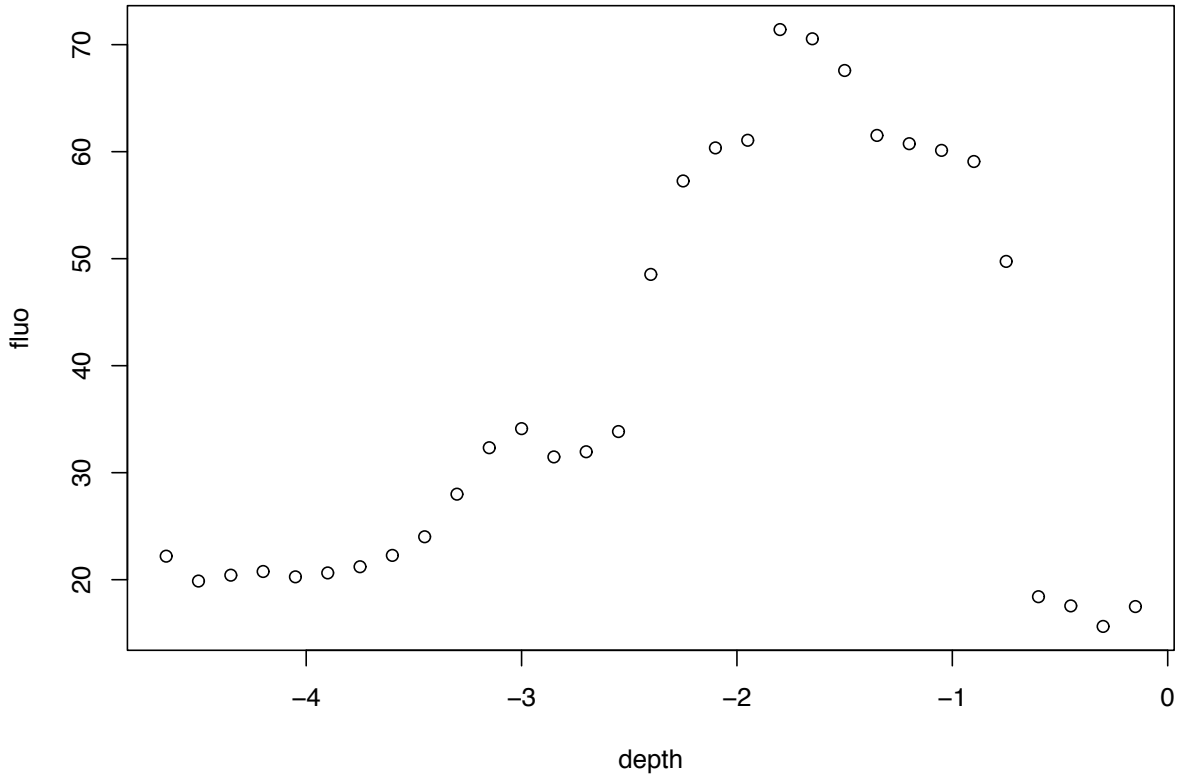




A single slice

We'll start by considering simple smoothing; therefore, we'll select a single slice through the data, the chlorophyll profile corresponding to x=24m across the lake

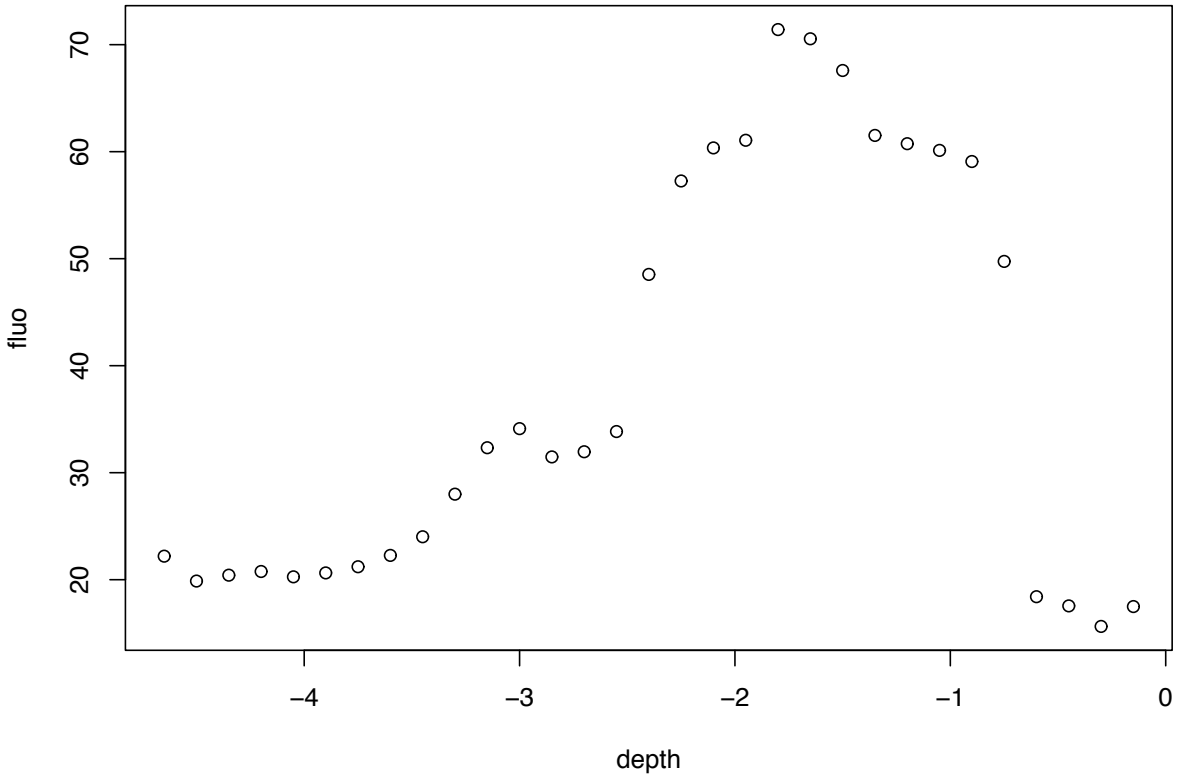
The robot was lowered to 31 different depths, starting at 15cm below the surface to 465cm in steps of 15cm



Smoothing

How do we model data like these? Intuitively the goal seems clear in that we'd like to run a smooth curve of some kind through the "center" of these data

It seems unlikely that a linear or quadratic or even a cubic polynomial will be flexible enough -- What principle should we leverage here to come up with this curve?



Parametric...

The regression models we have seen so far are defined by a set of parameters; the relationship between the predictors and the response is governed by, say, p parameters $\widehat{\beta}_1, \ldots, \widehat{\beta}_p$ that we need to estimate

Over the last few lectures, we saw that many of the common analysis methodologies in this context focus on the presence or absence of terms in the model, their sign and their relative magnitude

... vs. Nonparametric

In the statistics literature, smoothing is often referred to as “nonparametric” -- In short, the object of our attention shifts from parameters to features of the smooth curve

We are interested in looking at the shape of the curve, in counting modes or bumps, in examining regions of sharp change -- You might say that our analysis becomes a lot more visual

Rather than “seeing” the model through a table of parameters, we instead look at the curve itself in a lot more detail; with this shift, our underlying model also changes

Smoothing (Nonparametric regression)

Let's assume that our underlying model is of the form

$$y = f(x) + \epsilon$$

for x in some region $\mathcal{X}$, where again we assume normal errors with mean zero and variance σ^2; as usual we have observations $(x_1, y_1), \ldots, (x_n, y_n)$

In the case of the lake slice, we might take $\mathcal{X} = [-4.65, 0]$, depths ranging from the surface to the bottom of the lake; if we were to model the entire set of data, then $\mathcal{X}$ could be the 2-dimensional region that stretches across the lake in one dimension and from the surface to the lake bottom in the other (the upper boundary being straight, the lower boundary jagged)

If we don't make any assumptions on $f(x)$, there's nothing to anchor an estimation problem on; that is, if we cannot relate $f(x_1)$ to $f(x_2)$ for two points $x_1, x_2 \in \mathcal{X}$, then the best we can do is estimate $\widehat{f}(x_i) = y_i$

Smoothing (Nonparametric regression)

For the normal linear model, we assumed a particular parametric form for f ; that is, we said that the predictors and the response were related via a linear equation

In the case of smoothing, our implicit assumption has to do with the functional characteristics of f ; one common assumption of this type is that f is, well, smooth

We can make this formal by saying that f has two continuous derivatives or something like that; smoothness means that if x_1 and x_2 are near each other $f(x_1)$ and $f(x_2)$ should be close

Smoothing via local polynomials

It also provides some guidance about how we might create a good smoother: Recall from Taylor's Theorem that any (sufficiently) smooth function can be written locally as a polynomial

That is, given some point $x_0 \in \mathcal{X}$, we have the approximation

$$\begin{aligned} f(x) &= f(x_0) + f'(x_0)(x - x_0) + \\ &\quad \frac{1}{2} f''(x_0)(x - x_0)^2 + \frac{1}{6} f'''(x_0)(x - x_0)^3 + \cdots \end{aligned}$$

This, then will be the basis of a class of smoothers called local polynomials

An aside

Note that Taylor's theorem could be used to justify higher and higher degree polynomials -- In a theoretical sense, the more terms we add the better the approximation to the underlying function

Keep in mind, however, that there's a difference between theoretical approximation and what we can do with data -- We'll see a little later what will happen when we try ramping up the degree of an approximating polynomial

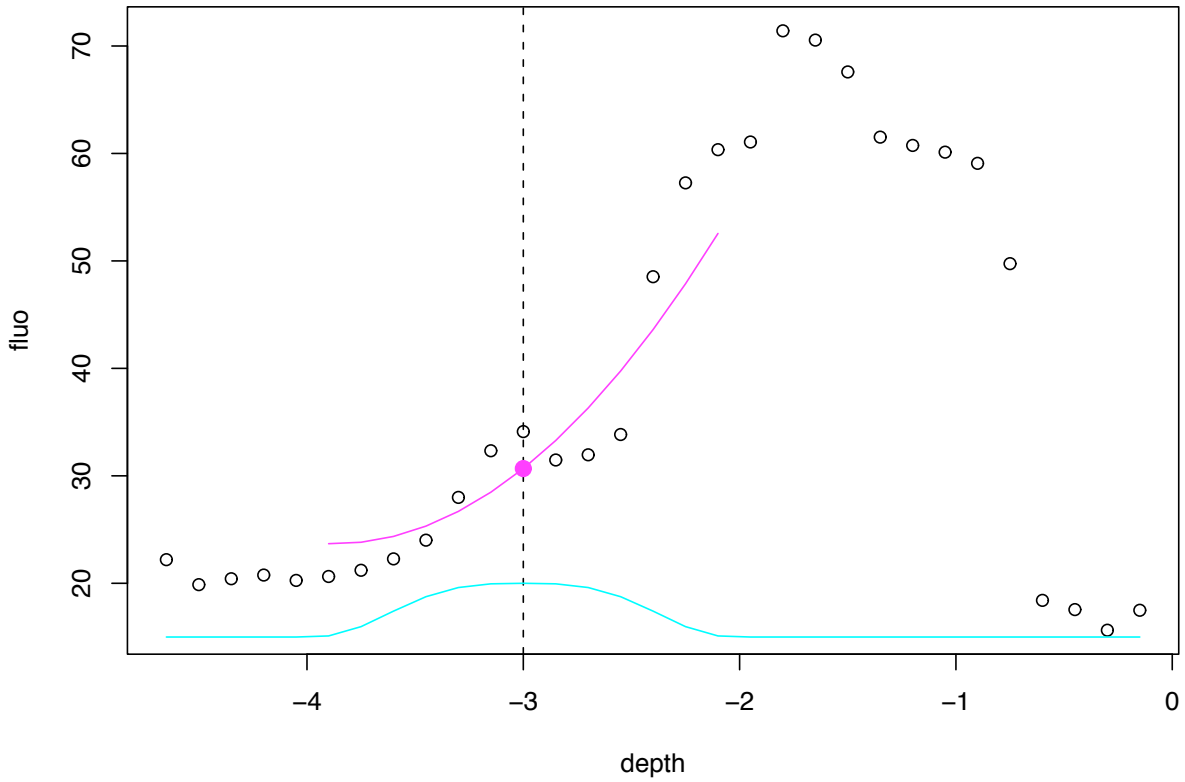
And it isn't pretty!

An example

Rather than fit these data by working with higher and higher degree polynomials, we might instead fit the polynomials locally

Here, for example, we are interested in estimating at the value of $f(x)$ at $x = -3$, or 3 meters below the surface

At the right, we perform a "local" quadratic fit that weights data according to their distance from -3

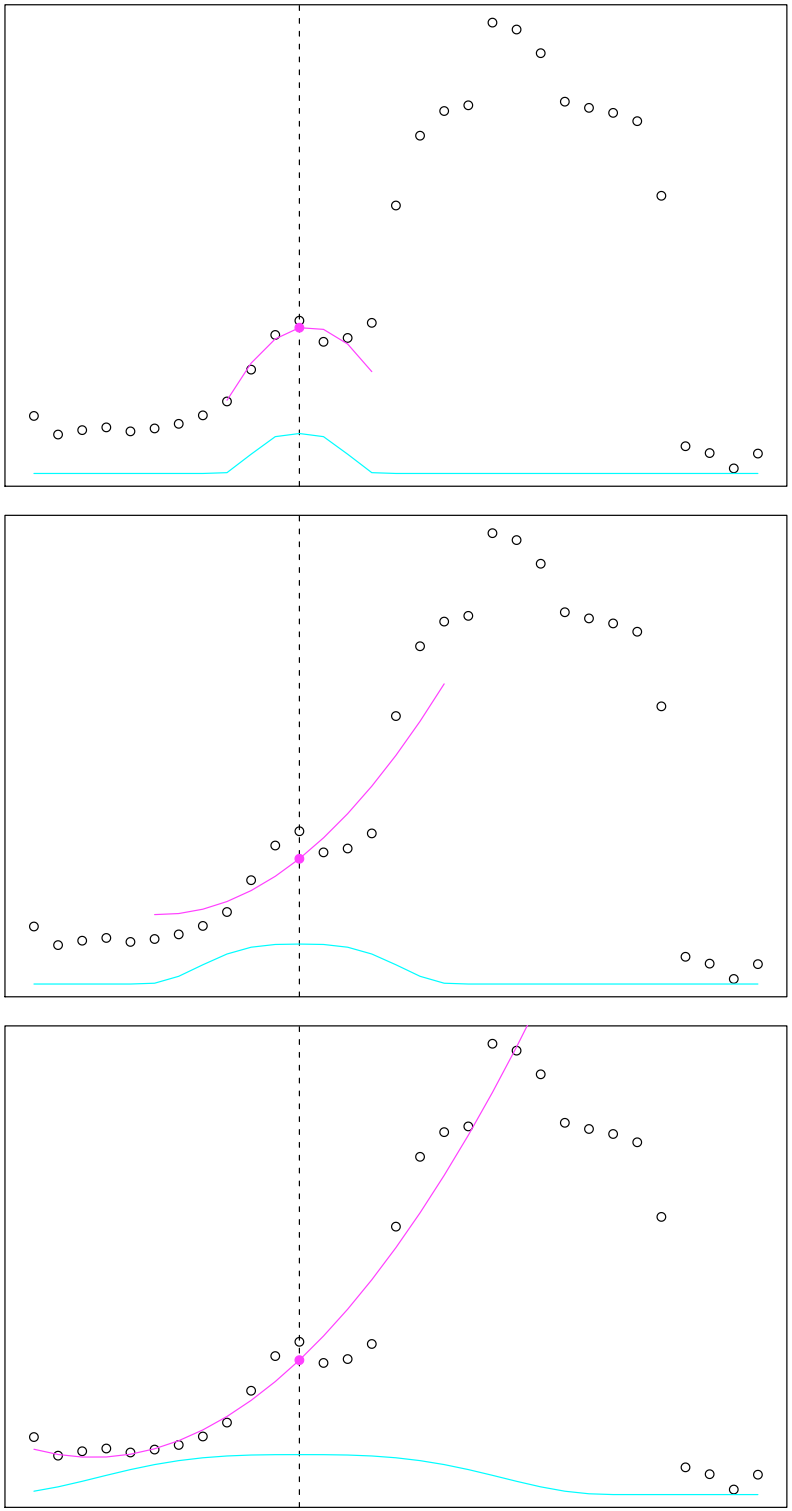


Local polynomials

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

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

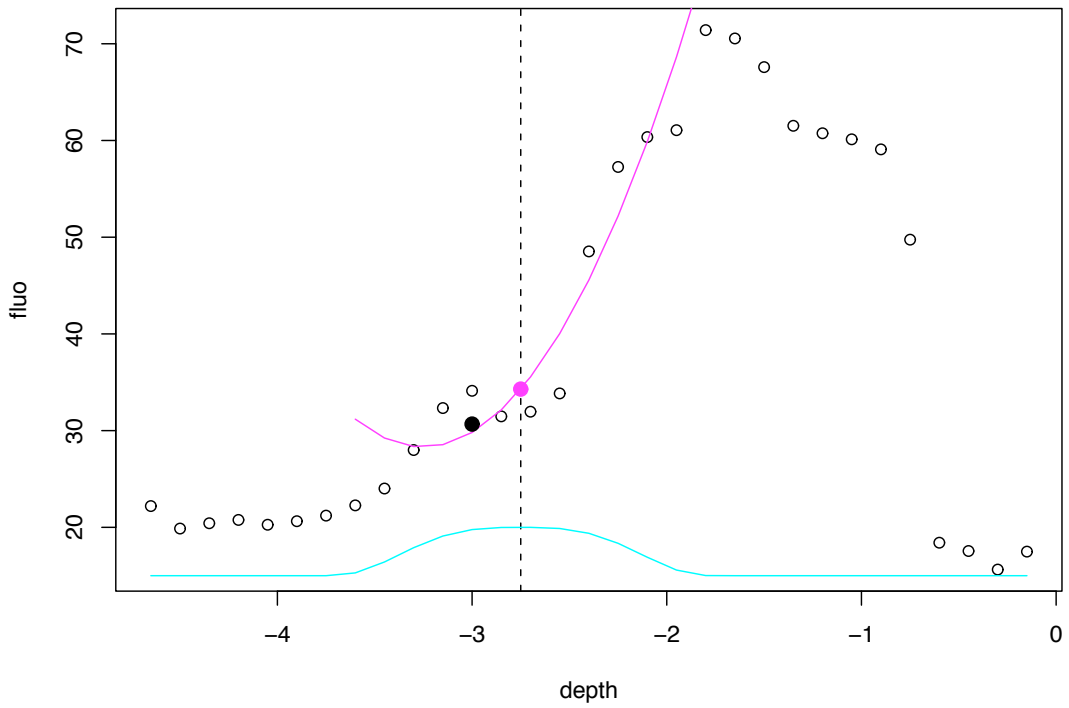
which gives $\widehat{\beta}_0, \widehat{\beta}_1, \widehat{\beta}_2$; we then take our estimate of $f(x_0)$ to be $\widehat{\beta}_0$

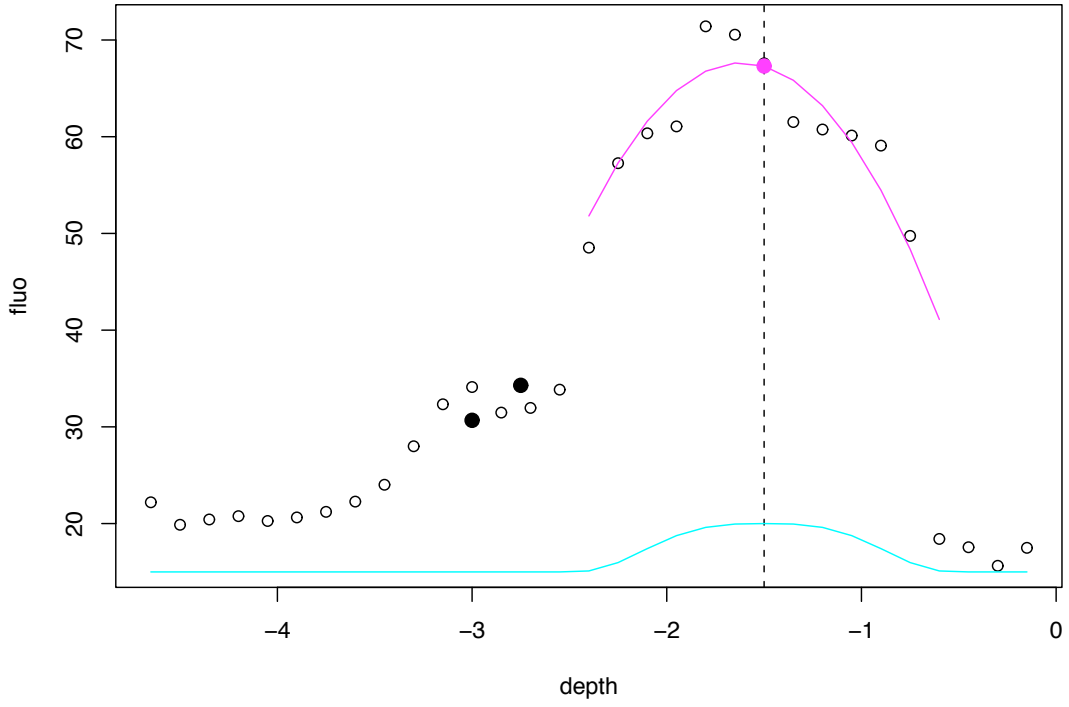
An example

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



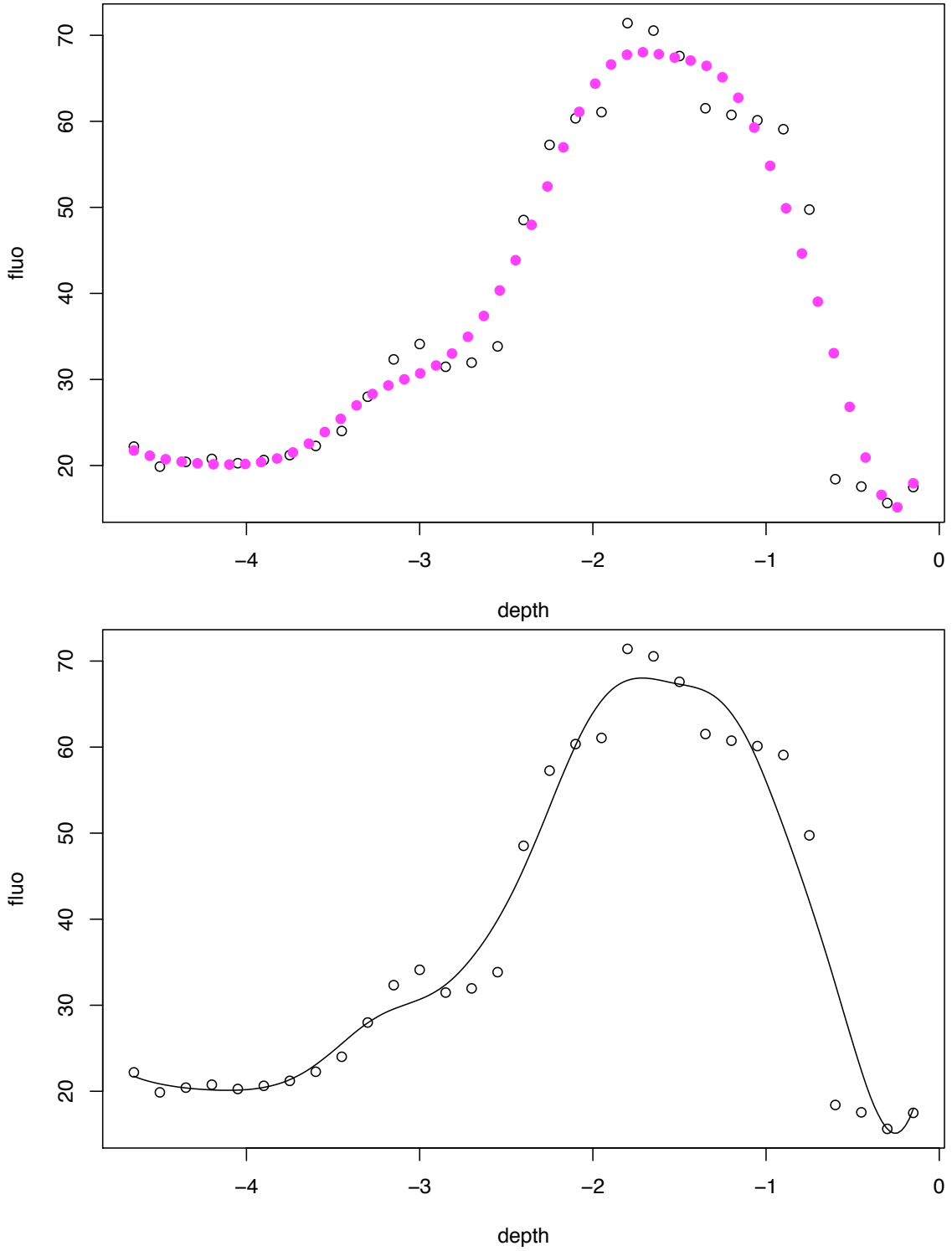


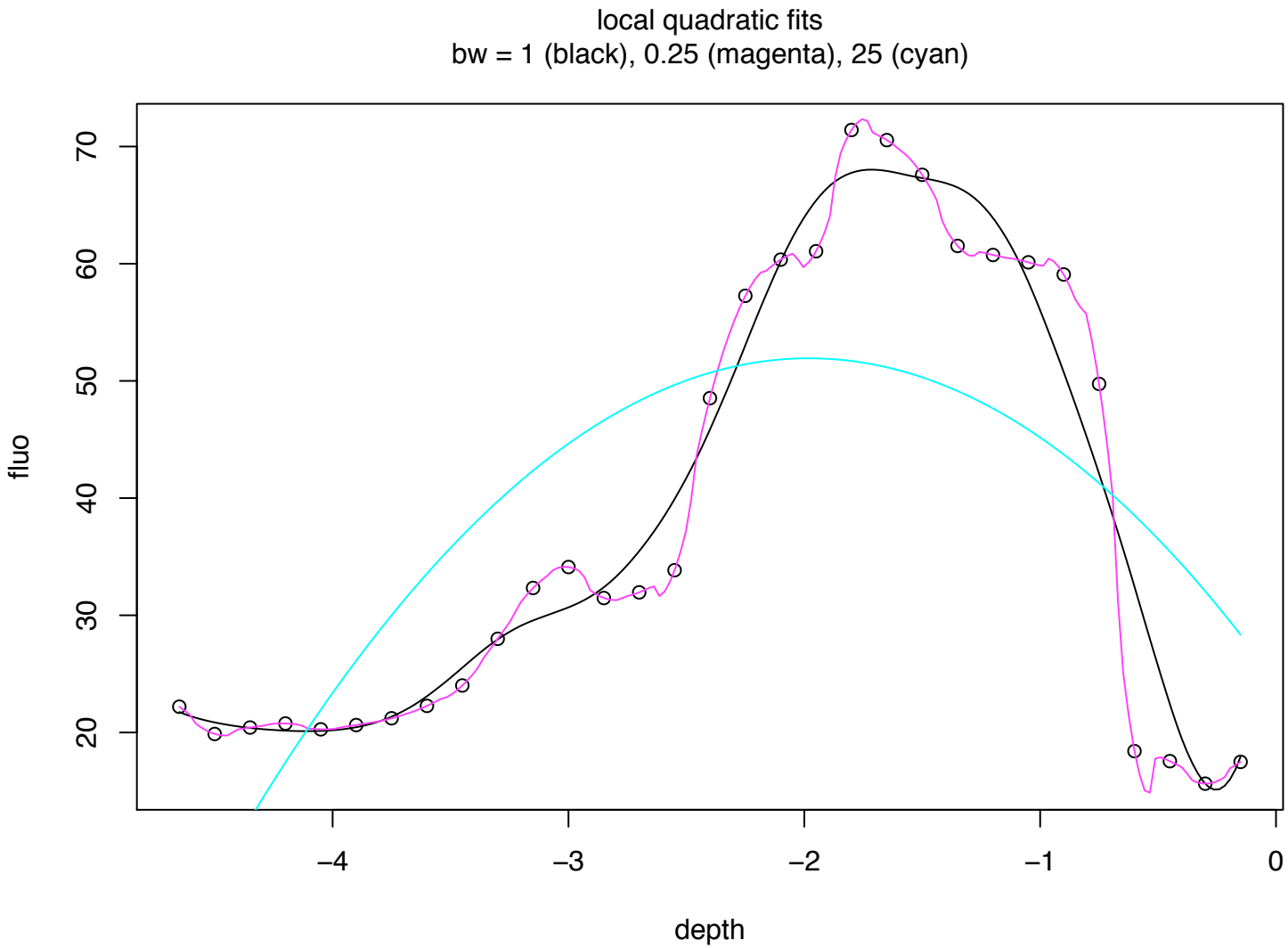
Local polynomials

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”




local quadratic fits
bw = 1 (black), 0.25 (magenta), 25 (cyan)
fluo
20
30
40
50
60
70
−4
−3
−2
−1
0
depth

```
# first, load in a function to compute the weights; it is called tric()

source(url("http://www.stat.ucla.edu/~cocteau/tric.R"))

# now, focus on the point 24 meters across the lake
# recall that the condition before the comma selects rows;
# here we take just those rows that correspond to x=24

slice = lake[lake$x==24,]
plot(slice$y,slice$fluo,xlab="depth",ylab="fluo")

# now, let's fit a local polynmoial at x0=-3; the weight
# function takes arguments of your data, the point x0 and
# the bandwidth (which here is 1)

x0=-3
weights = tric(slice$y,x0,1)
plot(slice$y,weights)

# now do the local fit

plot(slice$y,slice$fluo,xlab="depth",ylab="fluo")

fit = lm(fluo~I(y)+I(y^2),weight=tric(y,x0,1),data=slice)
points(x0,predict(fit,newdata=data.frame(y=x0)),pch=20,col=6)

# or do a series of them

x0 = seq(-4.65,-0.15,len=50)

for(i in 1:50){

   fit = lm(fluo~I(y)+I(y^2),weight=tric(y,x0[i],1),data=slice)
   points(x0[i],predict(fit,newdata=data.frame(y=x0[i])),pch=20,col=6)
}
```

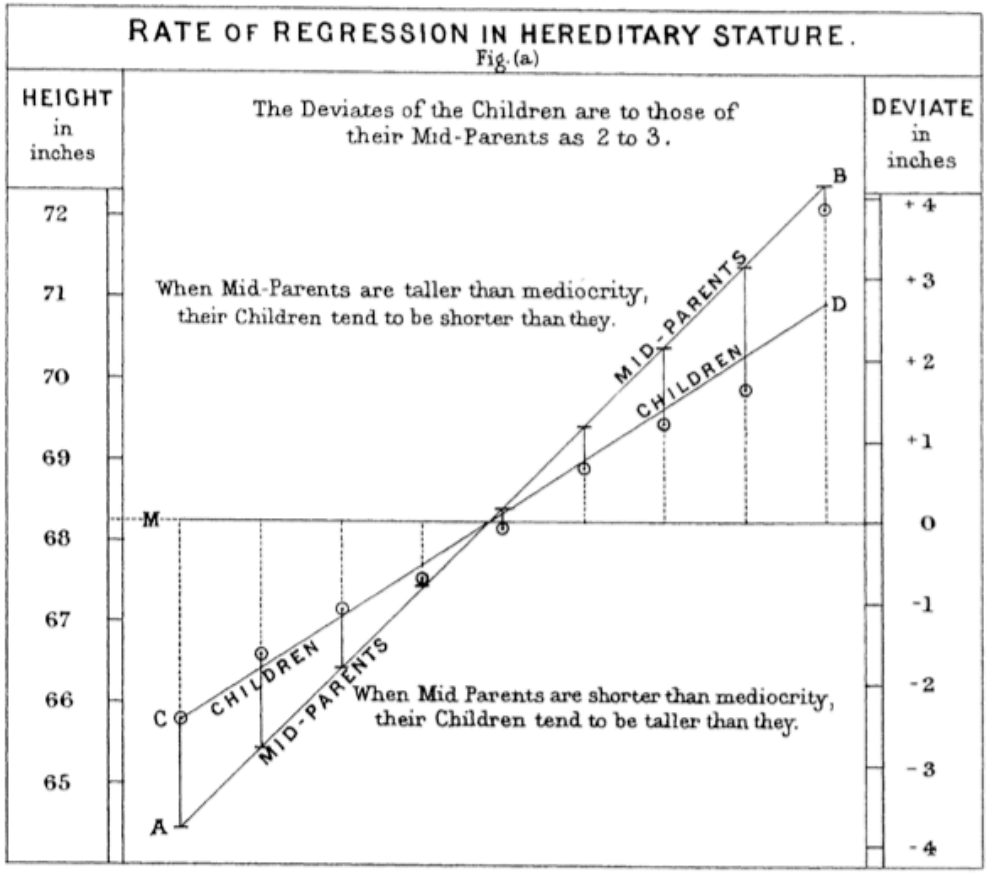
Binning?

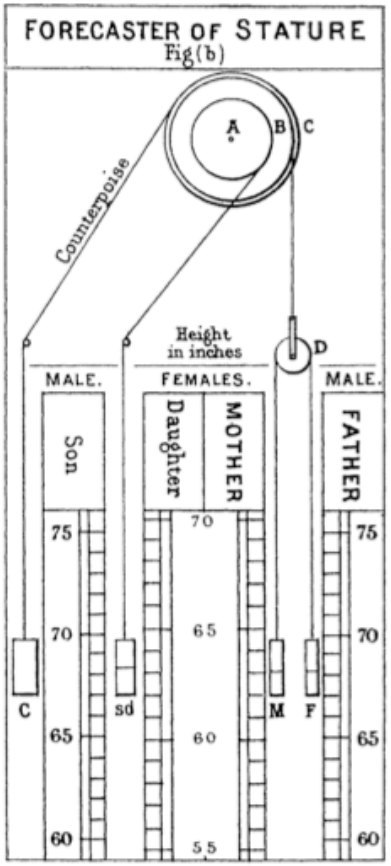
In some sense, this is a continuous version of the **binning** that Galton performed when he was relating the heights of children to mid-parents -- Galton, you'll recall, looked at averages of children's heights across (mid-)parents who had heights in separate groups

The mechanics behind binning involves (essentially) **fitting models piecewise**, carving the data into pieces and fitting a separate model in each -- We'll see more piecewise fits later...

In Galton's case, these binned averages followed (essentially) a straight line, providing evidence that his model was reasonable -- Interestingly, Galton appealed to a "nonparametric" tool when examining the goodness of his model

Plate IX.





J P & W R Emslie, lith.

Journ. Anthropolog. Inst., Vol. XV., Pl. IX

TABLE I.

NUMBER OF ADULT CHILDREN OF VARIOUS STATURES BORN OF 205 MID-PARENTS OF VARIOUS STATURES.

(All Female heights have been multiplied by 1·08).

Heights of the Mid-parents in inches.	Heights of the Adult Children.														Total Number of		Medians.
	Below	62·2	63·2	64·2	65·2	66·2	67·2	68·2	69·2	70·2	71·2	72·2	73·2	Above	Adult Children.	Mid-parents.	
Above ..	..	..	..	..	..	..	..	..	..	..	..	1	3	..	4	5	..
72·5	..	..	..	..	..	..	..	1	2	1	2	7	2	4	19	6	72·2
71·5	..	..	..	..	1	3	4	3	5	10	4	9	2	2	43	11	69·9
70·5	1	..	1	..	1	1	3	12	18	14	7	4	3	3	68	22	69·5
69·5	..	..	1	16	4	17	27	20	33	25	20	11	4	5	183	41	68·9
68·5	1	..	7	11	16	25	31	34	48	21	18	4	3	..	219	49	68·2
67·5	..	3	5	14	15	36	38	28	38	19	11	4	..	..	211	33	67·6
66·5	..	3	3	5	2	17	17	14	13	4	..	..	..	..	78	20	67·2
65·5	1	..	9	5	7	11	11	7	7	5	2	1	..	..	66	12	66·7
64·5	1	1	4	4	1	5	5	..	2	..	..	..	..	..	23	5	65·8
Below ..	1	..	2	4	1	2	2	1	1	..	..	..	..	..	14	1	..
Totals ..	5	7	32	59	48	117	138	120	167	99	64	41	17	14	928	205	..
Medians ..	..	..	66·3	67·8	67·9	67·7	67·9	68·3	68·5	69·0	69·0	70·0	..	..	..	..	..

NOTE.—In calculating the Medians, the entries have been taken as referring to the middle of the squares in which they stand. The reason why the headings run 62·2, 63·2, &c., instead of 62·5, 63·5, &c., is that the observations are unequally distributed between 62 and 63, 63 and 64, &c., there being a strong bias in favour of integral inches. After careful consideration, I concluded that the headings, as adopted, best satisfied the conditions. This inequality was not apparent in the case of the Mid-parents.

FIG. 10.

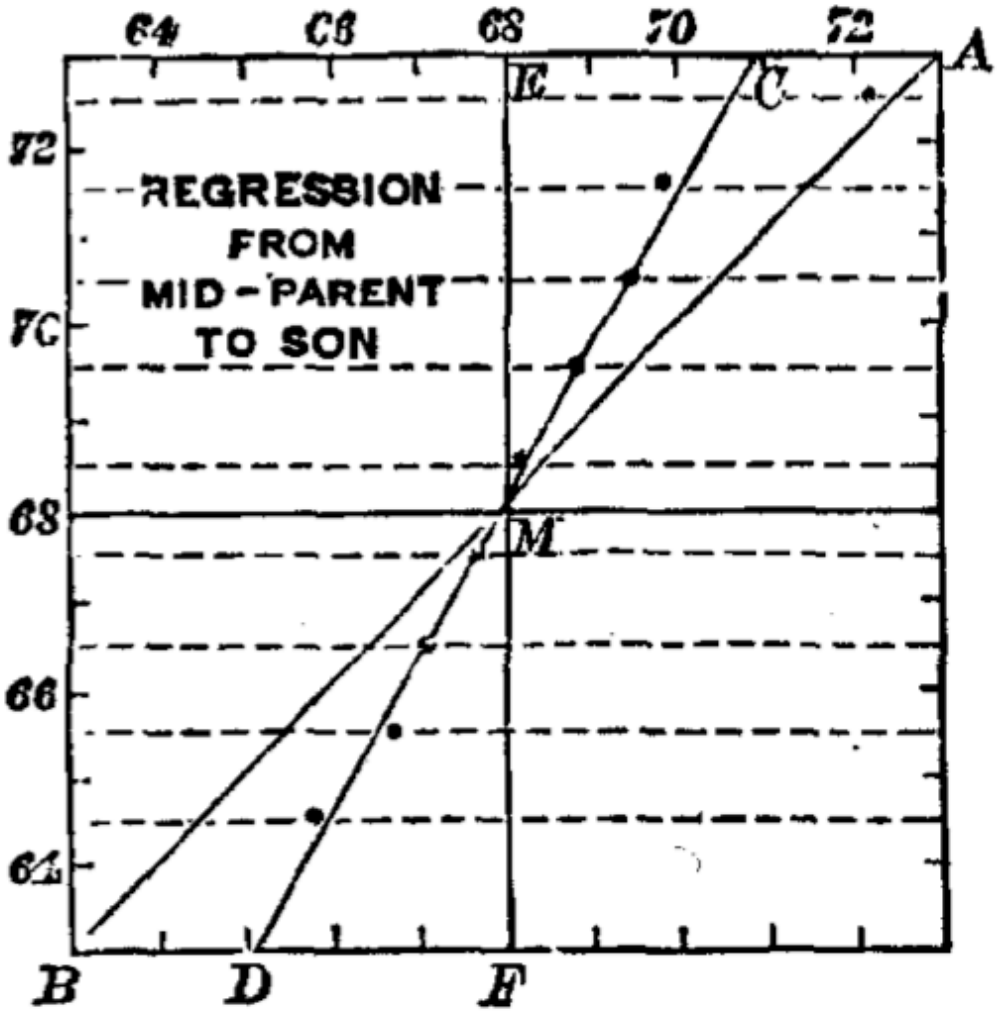
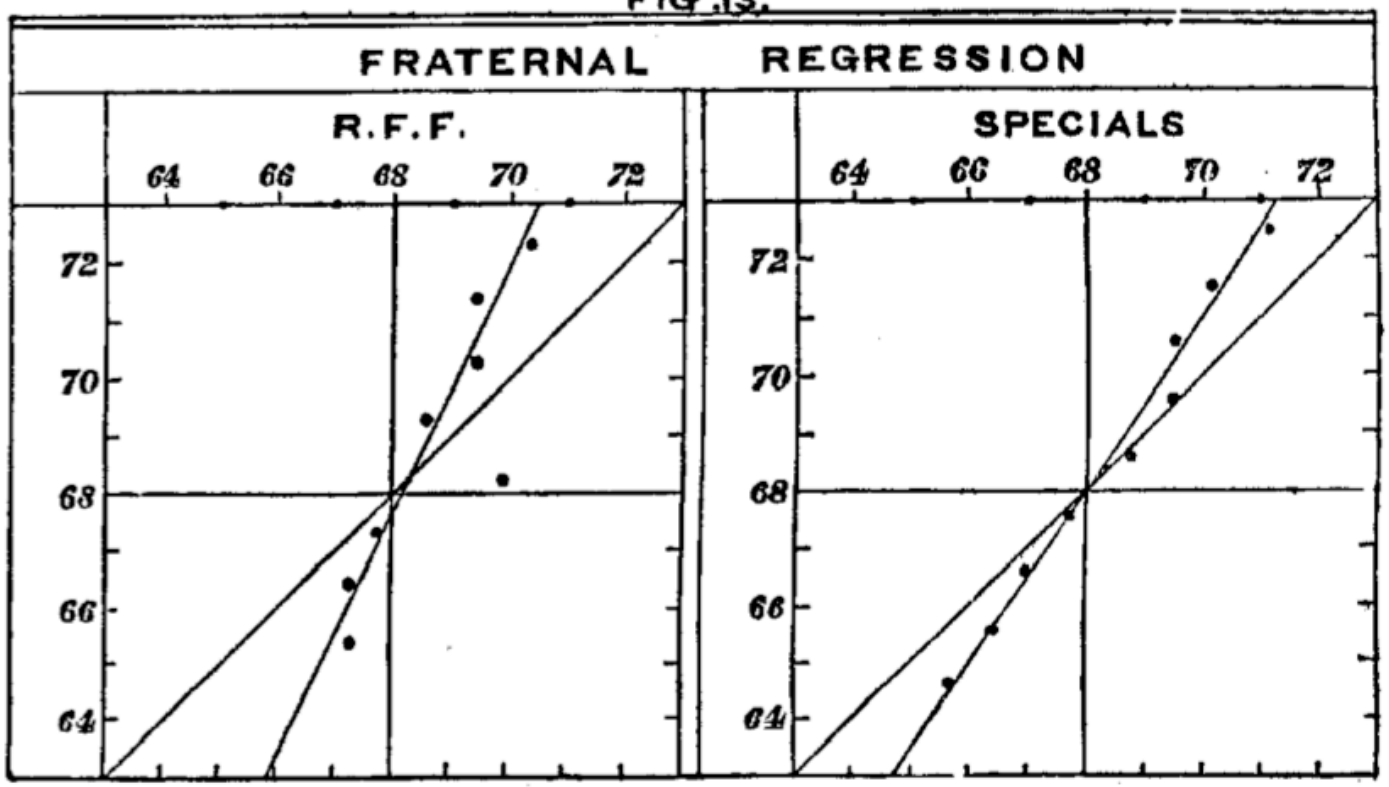

64
66
68
70
72
A
E
C
72
REGRESSION
FROM
MID-PARENT
TO SON
70
68
M
66
64
B
D
F

FIG. 13.


FRATERNAL REGRESSION
R.F.F.
SPECIALS
64
66
68
70
72
72
70
68
66
64

Defining locality

Right, the idea is pretty clear (although there are lots and lots of questions about practicalities); but first, let's figure out how best to define "local"

What properties do we want from the weights? Well, we need them to be centered at x_0 ; it's probably good if they are symmetric around that point also; they should decrease the farther you are from

A common choice is the so-called tri-cube function (gosh, I wonder why it's called that?)

$$w(u) = \begin{cases} (1 - |u|^3)^3 & |u| < 1 \\ 0 & \text{else} \end{cases}$$

and with it, given a bandwidth h , you define weights

$$w_i = w\left(\frac{x_i - x_0}{h}\right)$$

Local polynomials

As you might expect, this idea has been around for a little while (dare I mention the fact that it was published in 1979?); there is lots of code implementing the procedure

In R, the function we want is called `loess` for local polynomial regression; rather than work with bandwidths, it exposes two ways to control the amount of smoothing

LOWESS: A Program for Smoothing Scatterplots by Robust Locally Weighted Regression

The visual information on a scatterplot can be greatly enhanced with little additional cost by plotting smoothed points. Suppose the points of the scatterplot are $(x_i,y_i), i = 1, \ldots, n$. The smoothed points are $(x_i,\hat{y}_i)$ where $\hat{y}_i$, the fitted value at x_i, portrays the location of Y given $X = x_i$. Plotting the smoothed points, which form a nonparametric regression of Y on X, frequently allows the perception of effects on the scatterplot that are otherwise difficult to detect.

Robust locally weighted regression (Cleveland 1979) is a method for smoothing scatterplots in which the fitted value at x_k is the value of a line fit to the data using weighted least squares where the weight for (x_i,y_i) is large if x_i is close to x_k and small if x_i is not close to x_k. A robust fitting procedure guards against outliers distorting the smoothed points. Information on obtaining FORTRAN routines for robust locally weighted regression is available from the Computing Information Library at Bell Laboratories.

The routine LOWESS, which is directly called by the user, consists of 107 lines of code. LOWESS calls a support routine LOWEST, which has 79 lines of code, and requires a routine that sorts an array and a routine that sorts an array and passively sorts a second array according to the first. Two sort routines with 69 and 83 lines of code are provided for users who do not have sort routines on their systems. The user may elect to receive, along with the program documentation and listings, either a tape or a punched deck of the programs.

William S. Cleveland
Bell Laboratories
600 Mountain Ave.
Murray Hill, NJ 07974

REFERENCE

CLEVELAND, WILLIAM S. (1979), "Robust Locally Weighted Regression and Smoothing Scatterplots," *Journal of the American Statistical Association*, 74, 829–836.

Local polynomials

`loess()` exposes two ways to control the "amount" of smoothing

1. The first is a parameter called `span`; it is a number between 0 and 1 and determines the fraction of the data that should be included in the fit (using the tri-cube function, notice that data points beyond a certain distance are left out or assigned zero weight)

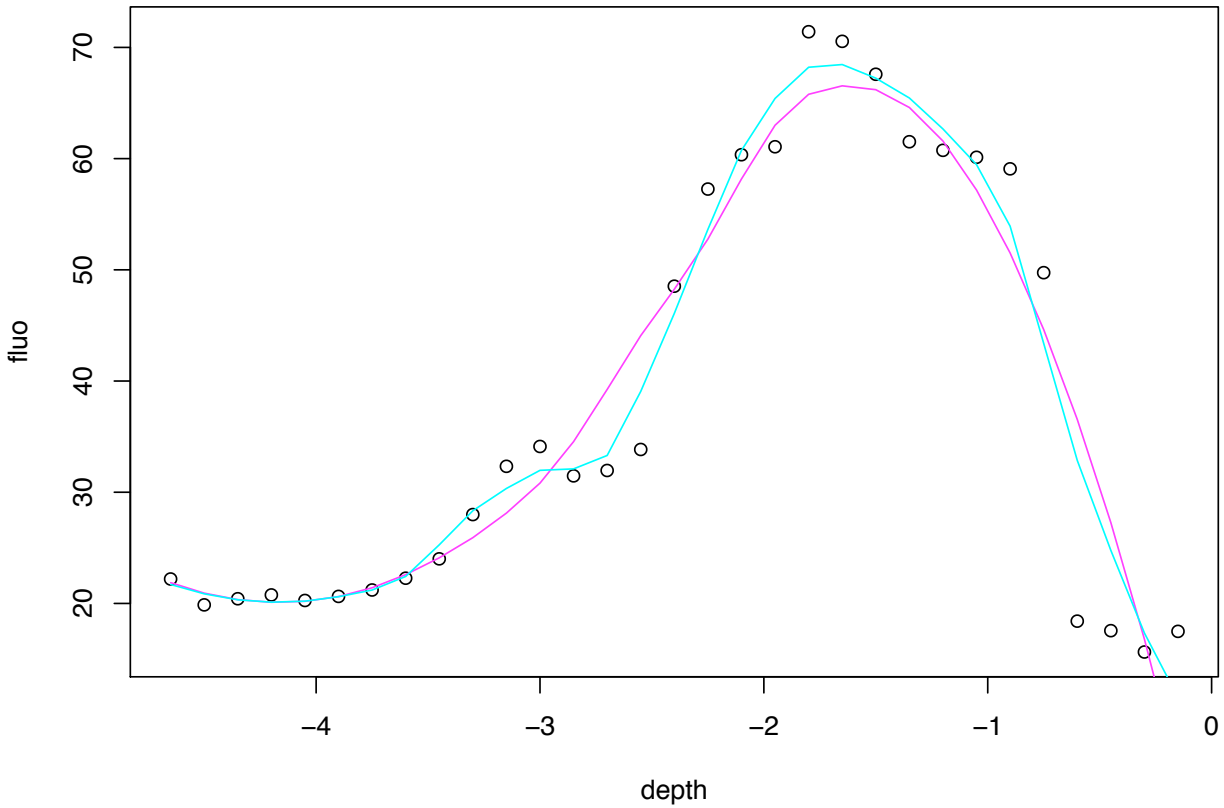
2. The second parameter is called `enp.target`; this translates the amount of smoothing into an equivalent number of parameters or degrees of freedom (this is essentially the same trick we played with ridge regression when thinking about the amount of shrinkage taking place)

Local polynomials

The way we've described them, local polynomials can be written in the form

$$\widehat{y} = Sy$$

where S depends only on the input variables -- This means we can apply the same degrees of freedom computations we entertained for ridge regression

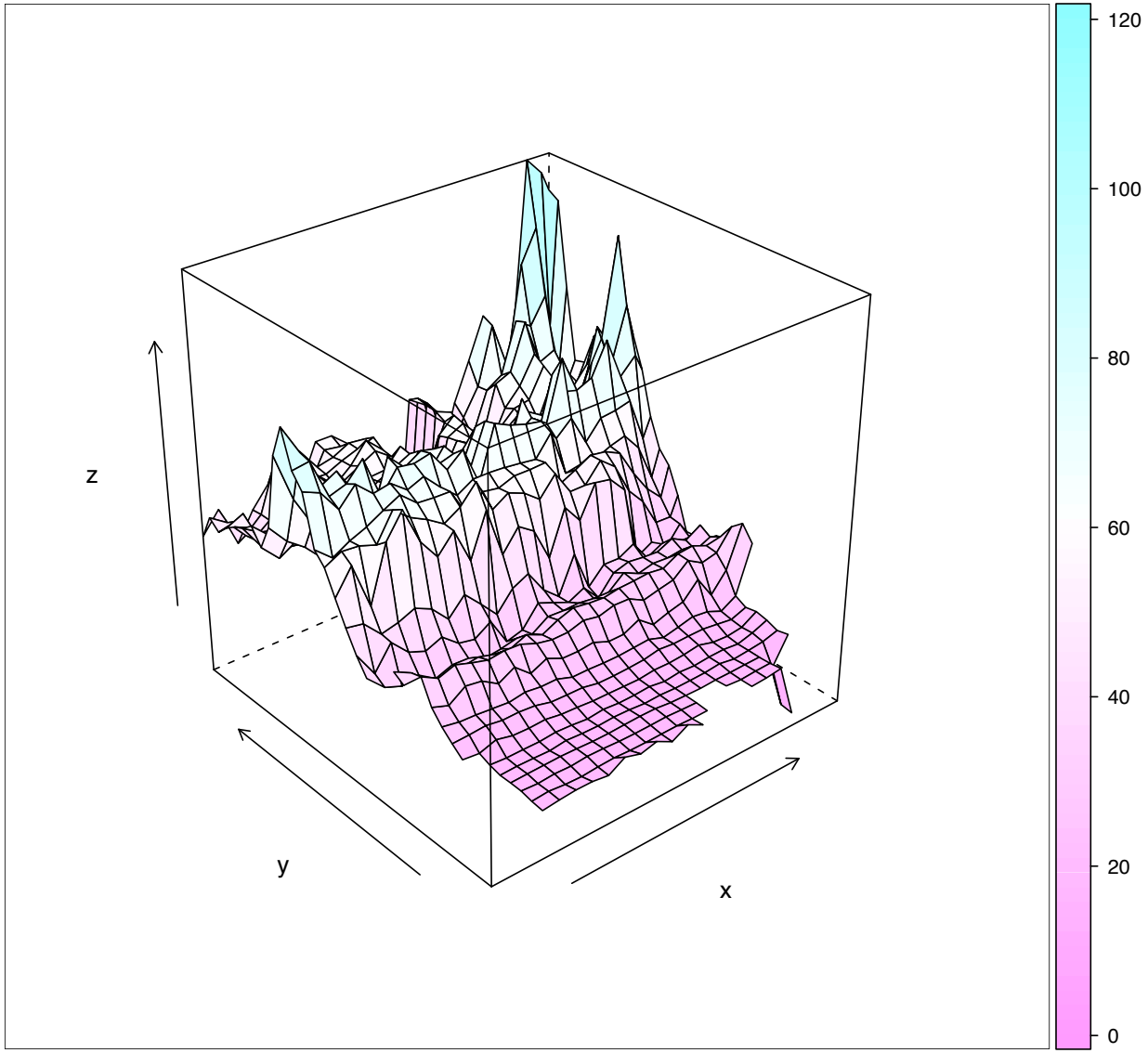



```
slice = lake[lake$x==24,]
plot(slice$y,slice$fluo,xlab="depth",ylab="fluo")

# fit with about 5 degrees of freedom

fit = loess(fluo~y,data=slice,enp.target=5)
lines(slice$y,predict(fit),col=6)

# fit with about 10 degrees of freedom

fit = loess(fluo~y,data=slice,enp.target=10)
lines(slice$y,predict(fit),col=5)
```

Multivariate modeling

Here we have the chlorophyll values in both depth and distance along the transect

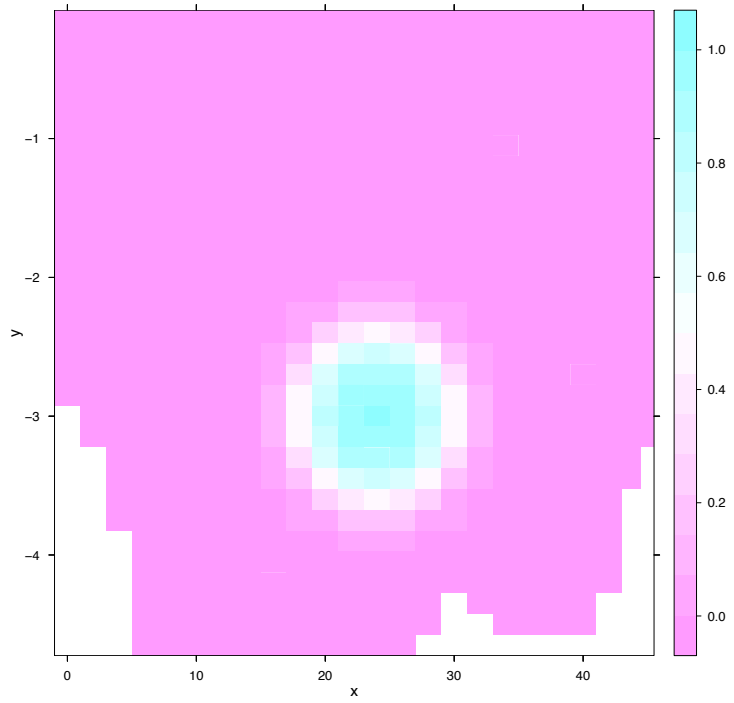
Applying the local polynomial ideas here, we select a location (x,y) at which we want to make a prediction and fit a quadratic polynomial (now in both x and y)...

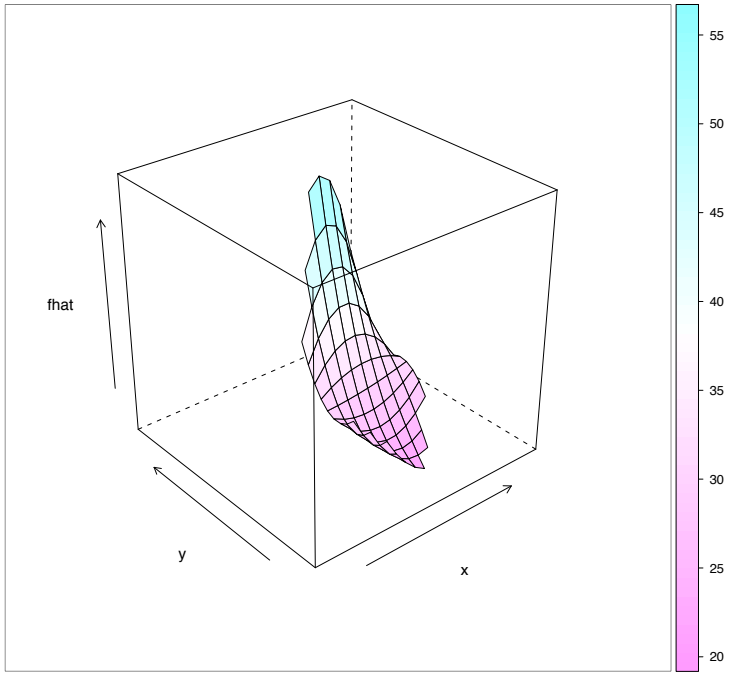


Multivariate modeling

Here we choose the point (2.4,-3) and plot the tri-cube kernel centered at this point

This defines a region for us and the resulting polynomial fit in the region is given in the lower panel

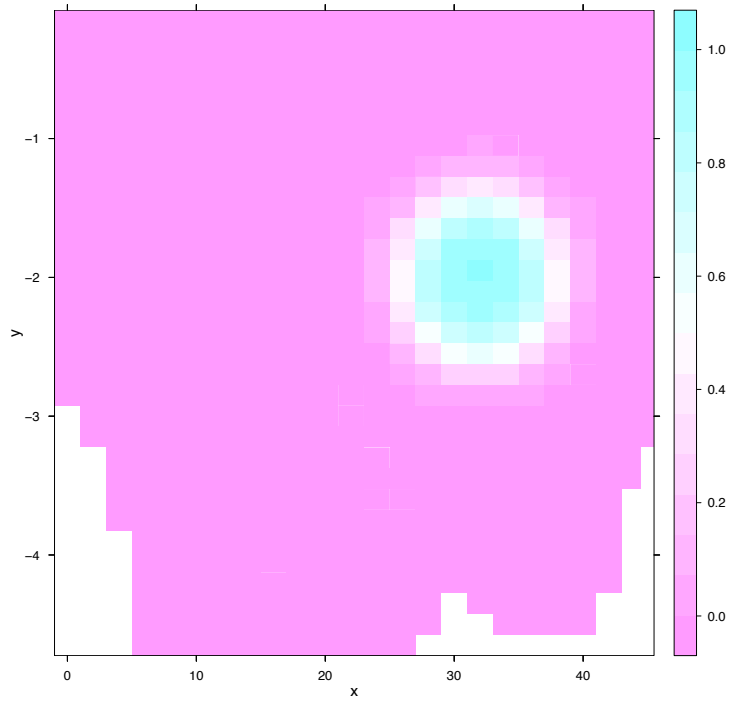


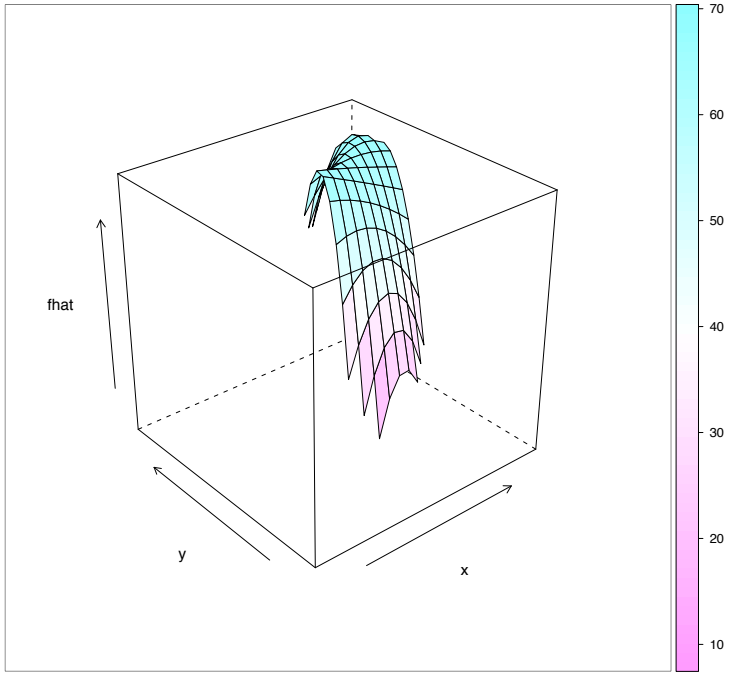


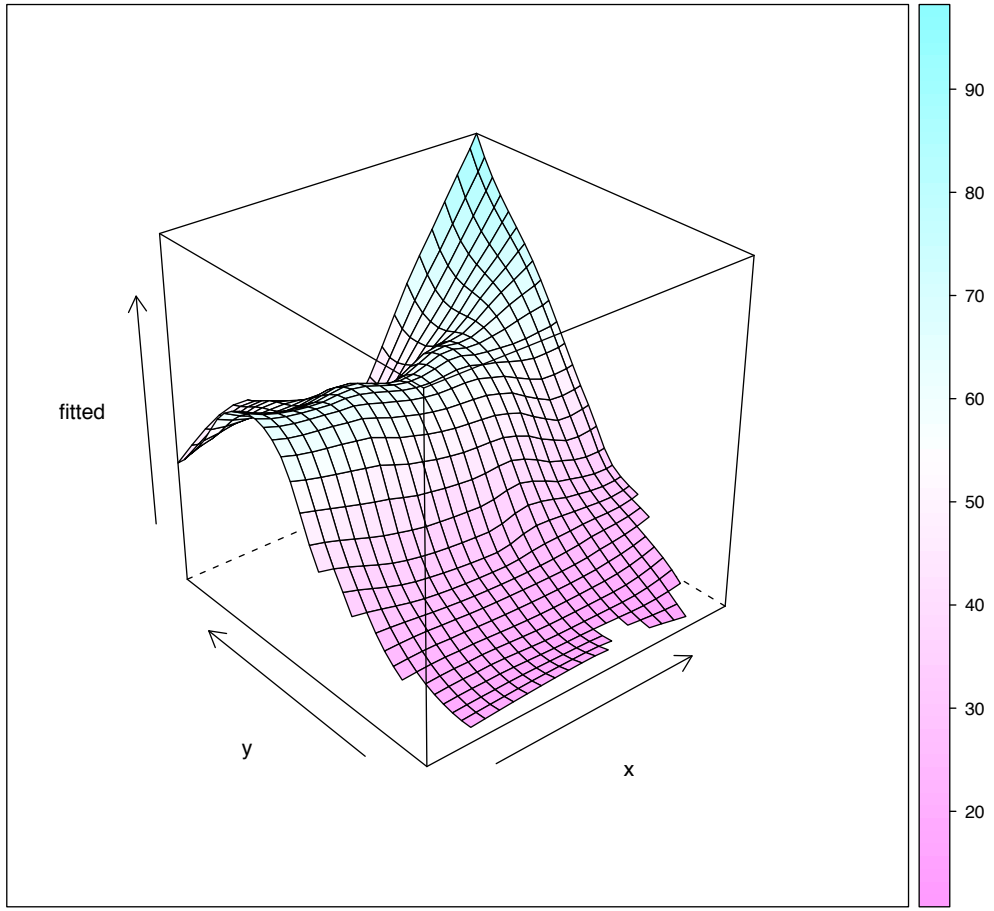
Multivariate modeling

Here we choose the point (3.2,-2) and plot the tri-cube kernel centered at this point

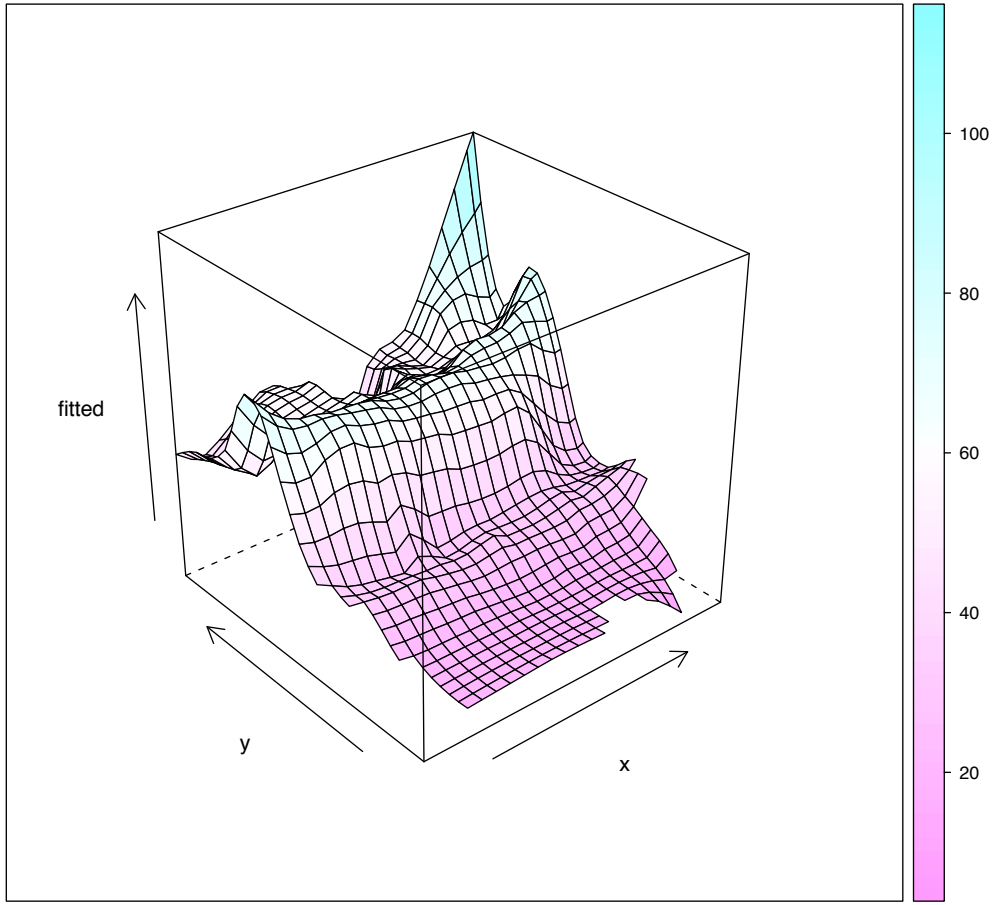
This defines a region for us and the resulting polynomial fit in the region is given in the lower panel








```
# 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)
```




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

library(lattice)

# fit a local polynomial in two dimensions using the full
# lake data set; this time with 100 degrees of freedom

fit = loess(fluo~x+y,data=lake,enp.target=100)

# 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)
```

Extending simple polynomials

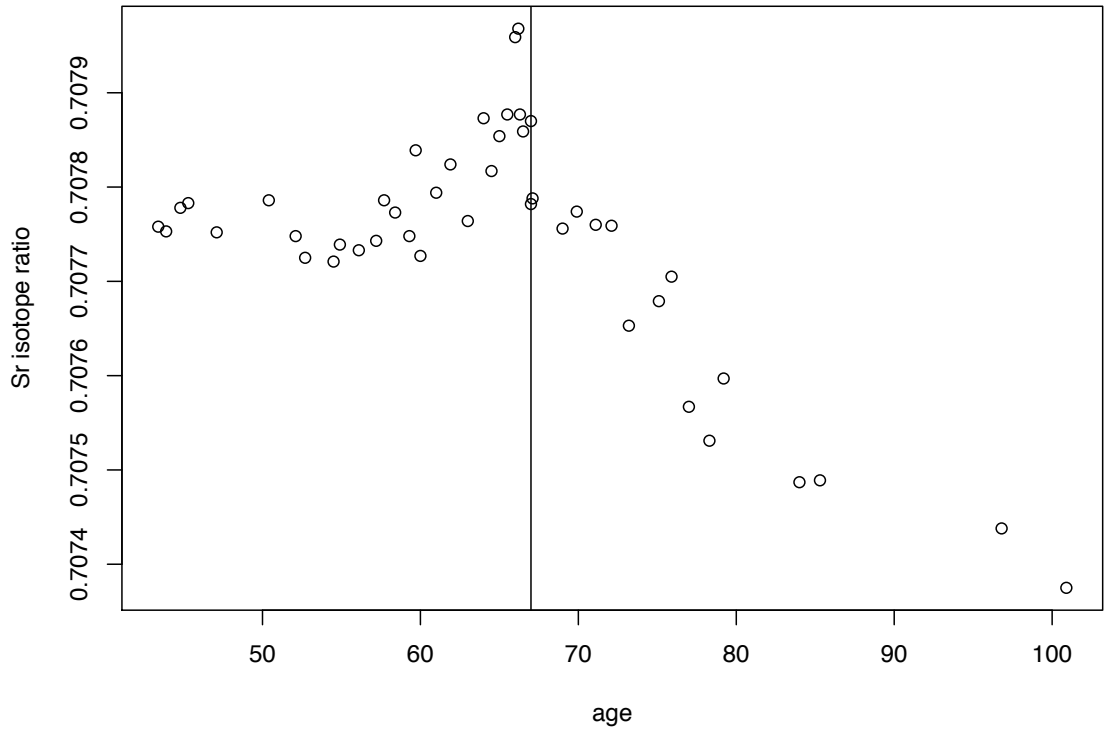
The local polynomial approach is just one way to beef up the descriptive power of polynomials (stably) -- We'll now consider another way to extend polynomials, this time by fitting them piecewise...

Sr isotopes

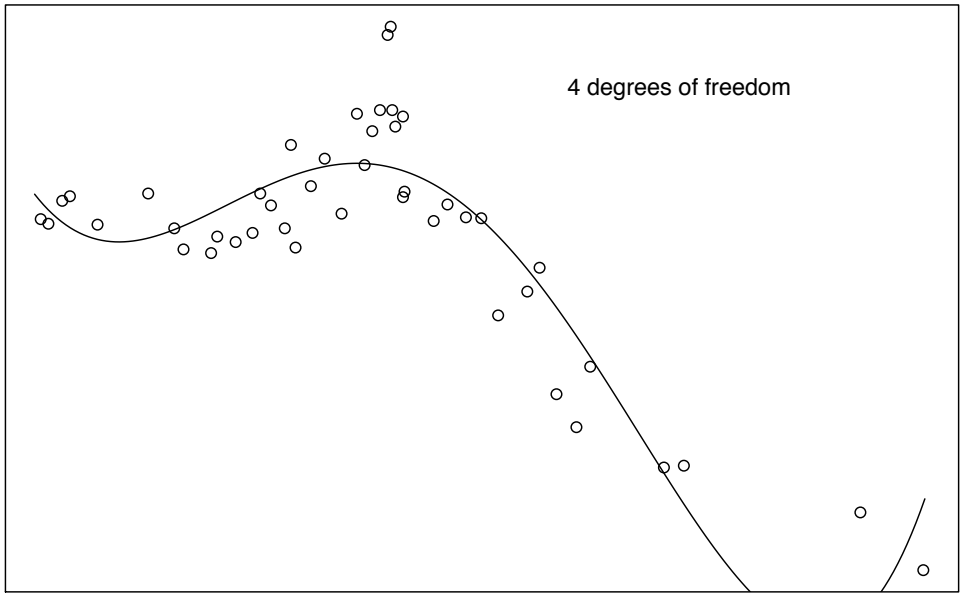
We're going to start by looking at what is now a fairly old data set but it illustrates some of
the reasons why one might want to appeal to constructions like piecewise polynomials

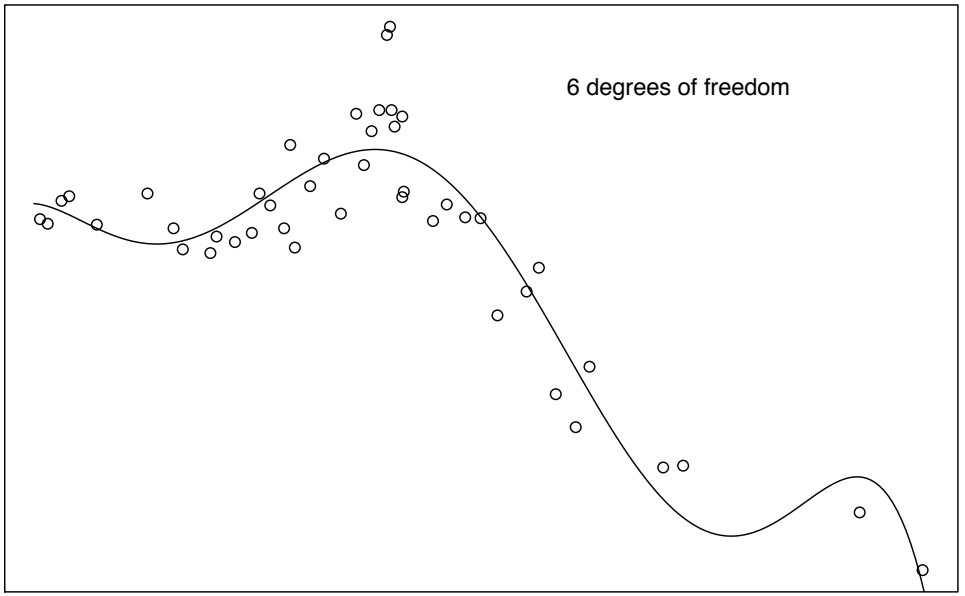
1. The "predictors" are unequally spaced (notice the data points trail off after about 80 million years ago)

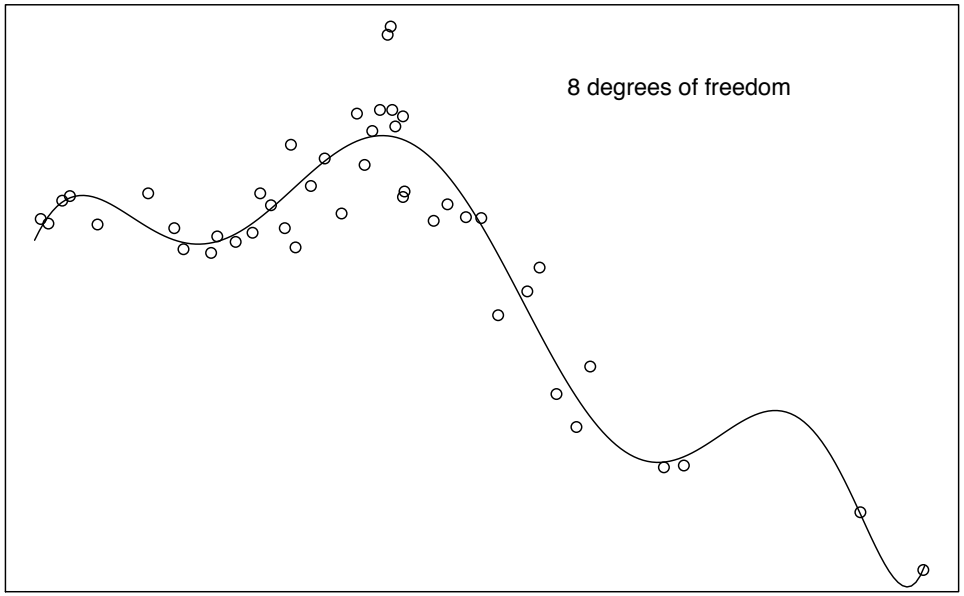
2. There is scientific interest in a feature of the estimated curve (the jump at 67 million years ago tells us something about the catastrophe that resulted in a mass extinction)

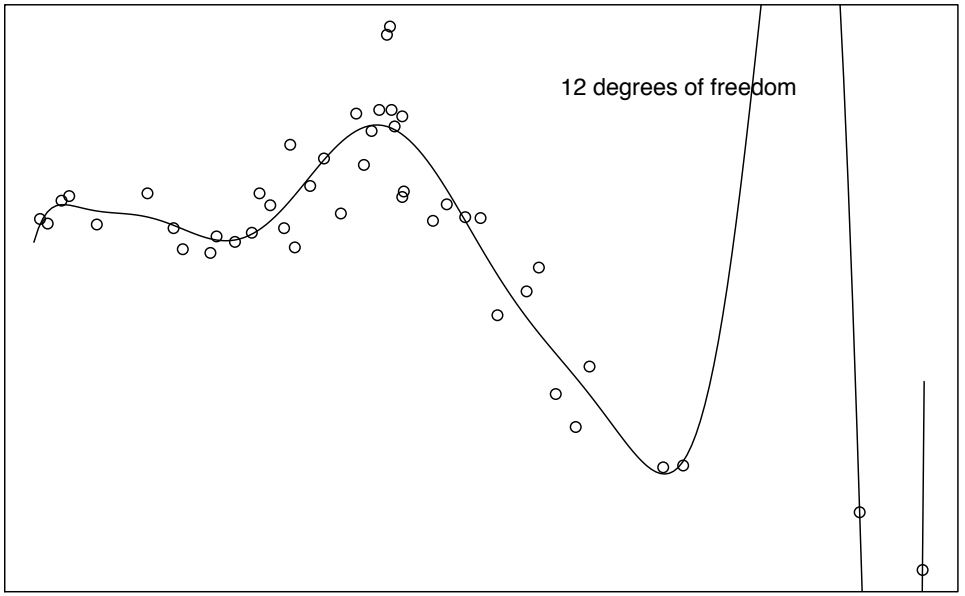


```
# first, load in the data and then play with some polynomial fits

kt = read.csv(url("http://www.stat.ucla.edu/~cocteau/kt.csv"),head=T)
newkt = data.frame(age=seq(43,101,len=200))

# 4 degrees of freedom

plot(kt$age,kt$sr,xlab="age",ylab="Sr isotope ratio")
fit = lm(sr.ratio~poly(age,4),data=kt)
lines(newkt$age,predict(fit,newdata=newkt))
text(85,0.7079,"4 degrees of freedom")

# 6 degrees of freedom

plot(kt$age,kt$sr,xlab="age",ylab="Sr isotope ratio")
fit = lm(sr.ratio~poly(age,6),data=kt)
lines(newkt$age,predict(fit,newdata=newkt))
text(85,0.7079,"6 degrees of freedom")

# 8 degrees of freedom

plot(kt$age,kt$sr,xlab="age",ylab="Sr isotope ratio")
fit = lm(sr.ratio~poly(age,8),data=kt)
lines(newkt$age,predict(fit,newdata=newkt))
text(85,0.7079,"8 degrees of freedom")

# 12 degrees of freedom

plot(kt$age,kt$sr,xlab="age",ylab="Sr isotope ratio")
fit = lm(sr.ratio~poly(age,12),data=kt)
lines(newkt$age,predict(fit,newdata=newkt))
text(85,0.7079,"12 degrees of freedom")
```







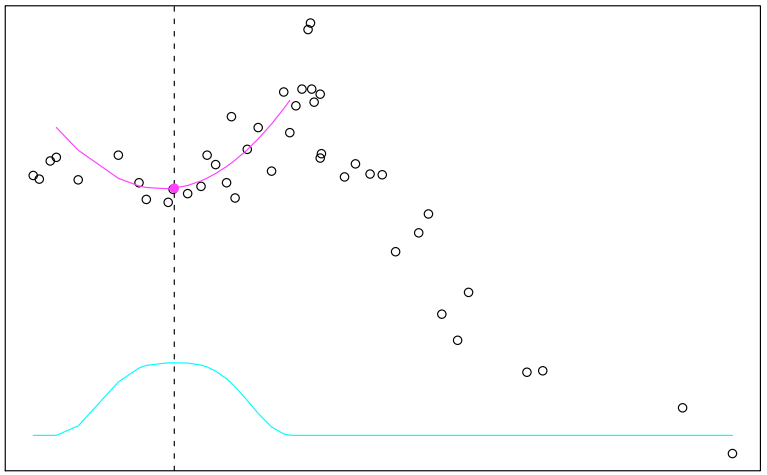


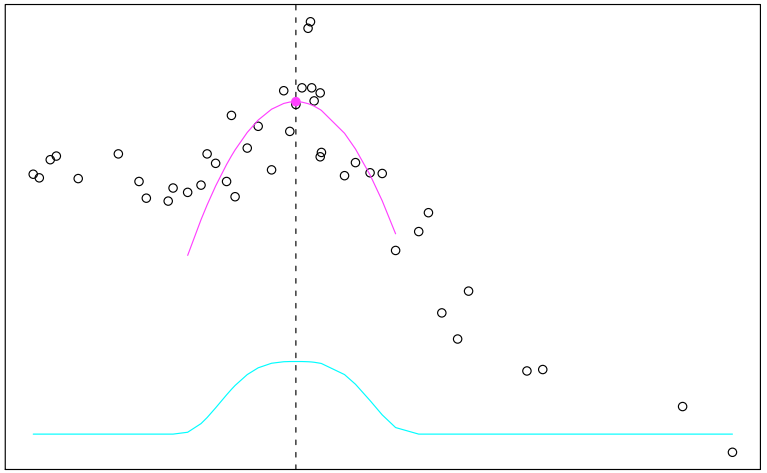
*This is why Tukey referred to high-degree polynomials as “sharp” tools

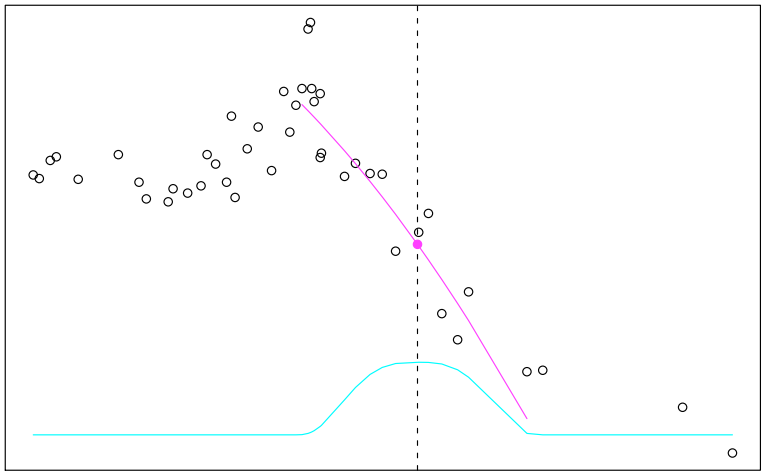
Smoothing

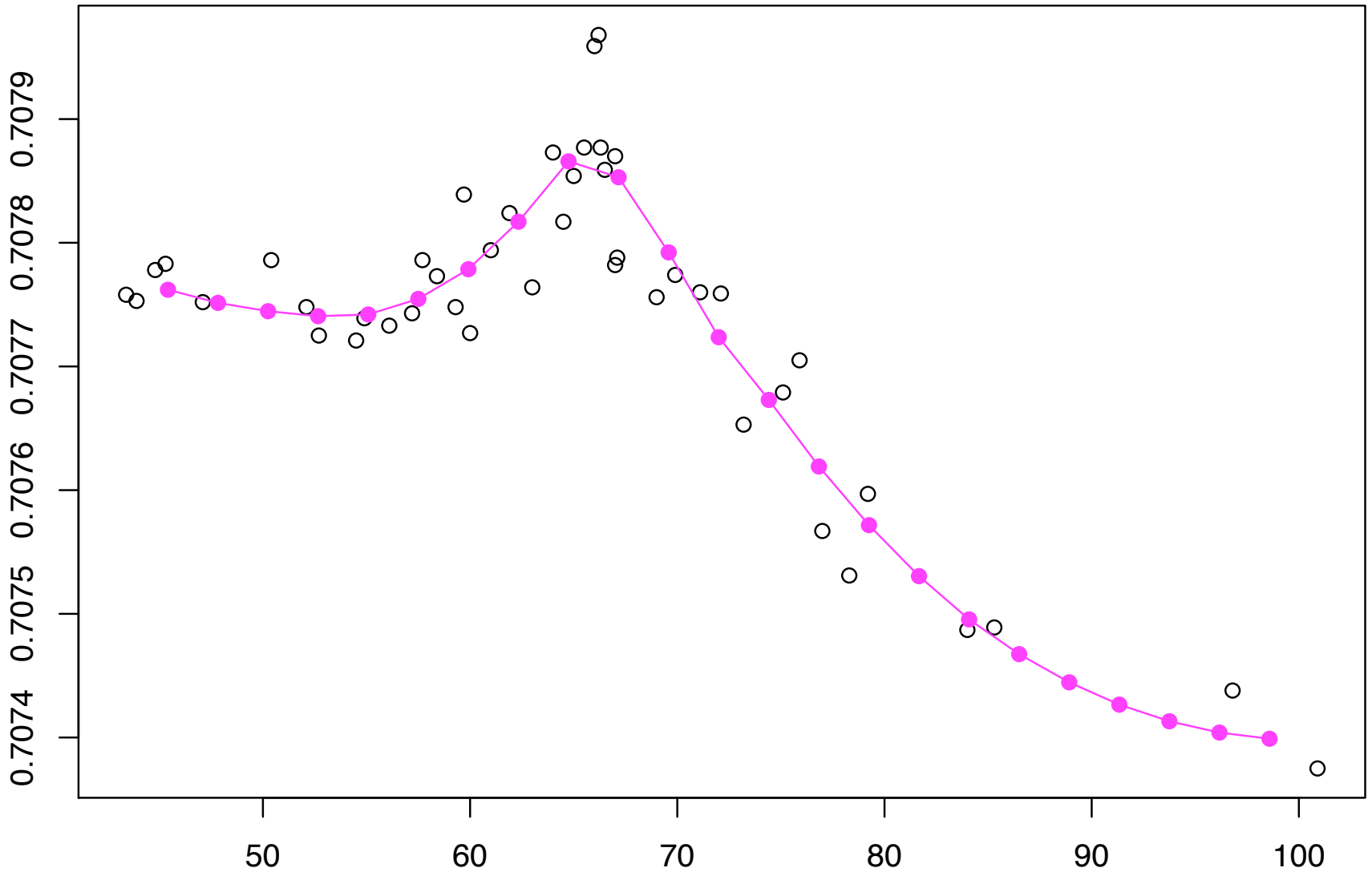
Clearly, a high-dimensional polynomial here doesn't work -- Even when fitting with orthogonal polynomials, we see that the fit is so highly constrained to be "well behaved" on the design points, it jumps erratically outside the support of the input data

So while in principle polynomials are extremely useful descriptive analytical tools, their use in data analysis needs to be constrained in some way -- At the right we have a few fitted values from a local polynomial

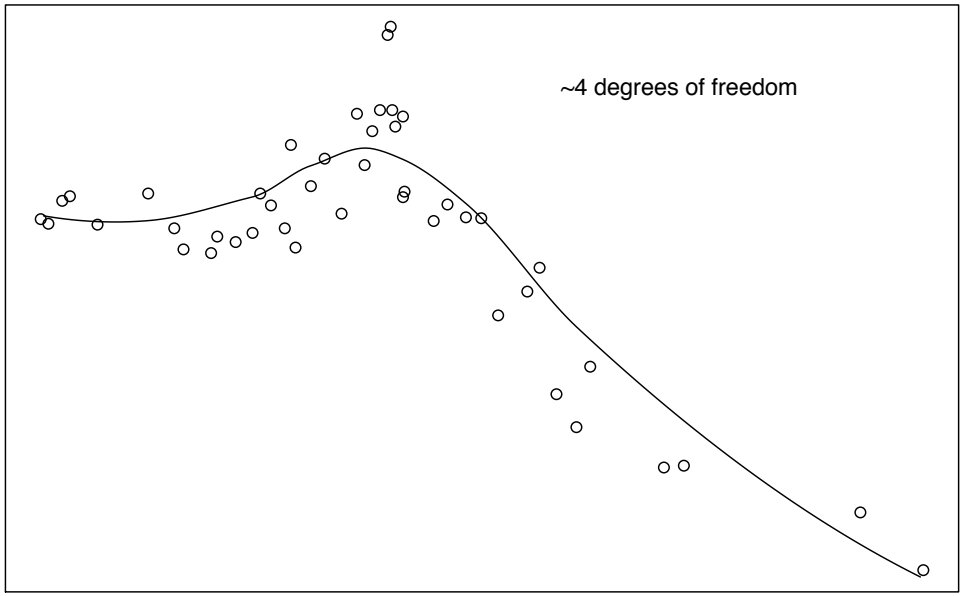


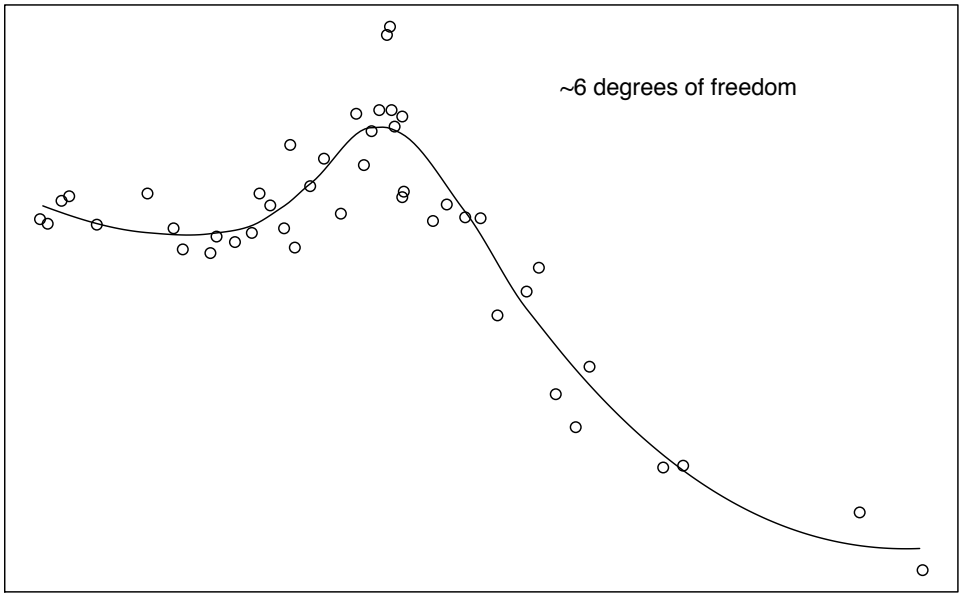


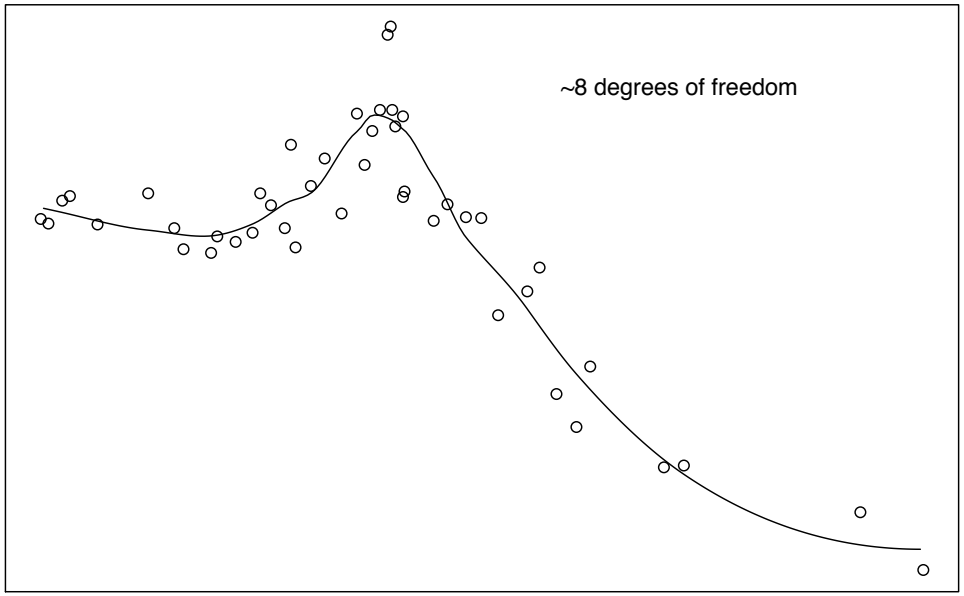


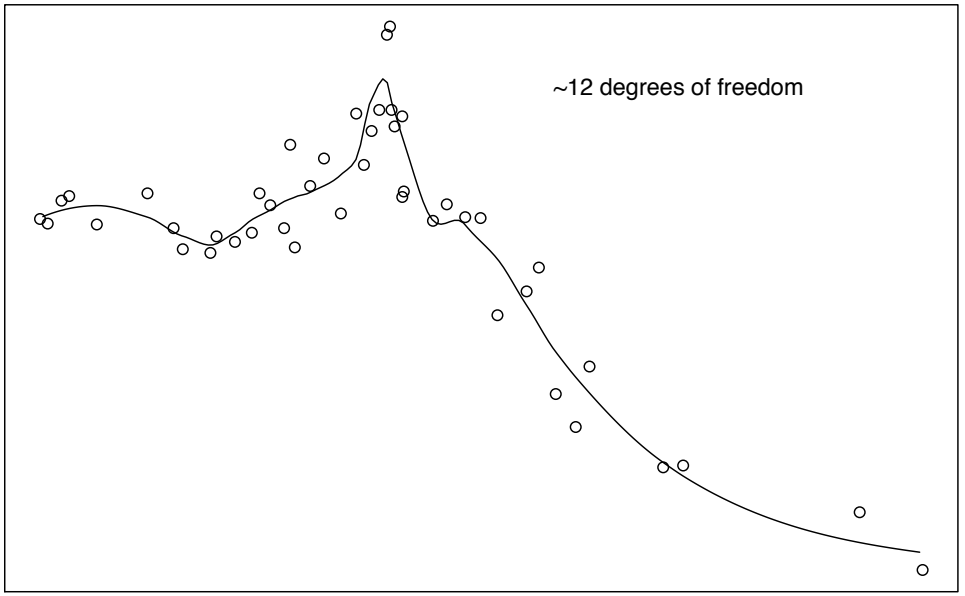

0.7074
0.7075
0.7076
0.7077
0.7078
0.7079
50
60
70
80
90
100

```
# first, load in the data and then play with some polynomial fits

kt = read.csv(url("http://www.stat.ucla.edu/~cocteau/kt.csv"),head=T)
newkt = data.frame(age=seq(43,101,len=200))

# 4 degrees of freedom

plot(kt$age,kt$sr,xlab="age",ylab="Sr isotope ratio")
fit = loess(sr.ratio~age,data=kt,enp=4)
lines(newkt$age,predict(fit,newdata=newkt))
text(85,0.7079,"~4 degrees of freedom")

# 6 degrees of freedom

plot(kt$age,kt$sr,xlab="age",ylab="Sr isotope ratio")
fit = loess(sr.ratio~age,data=kt,enp=6)
lines(newkt$age,predict(fit,newdata=newkt))
text(85,0.7079,"~6 degrees of freedom")

# 8 degrees of freedom

plot(kt$age,kt$sr,xlab="age",ylab="Sr isotope ratio")
fit = loess(sr.ratio~age,data=kt,enp=8)
lines(newkt$age,predict(fit,newdata=newkt))
text(85,0.7079,"~8 degrees of freedom")

# 12 degrees of freedom

plot(kt$age,kt$sr,xlab="age",ylab="Sr isotope ratio")
fit = loess(sr.ratio~age,data=kt,enp=12)
lines(newkt$age,predict(fit,newdata=newkt))
text(85,0.7079,"~12 degrees of freedom")
```


~4 degrees of freedom


~6 degrees of freedom


~8 degrees of freedom


~12 degrees of freedom

Pros and Cons

Because this smoothing procedure has a single knob to play with (the bandwidth), it is extremely convenient for data analysis -- You can quickly and easily compare a large number of fits

The procedure is also quite flexible and can be extended to handle so- called “robust” regression (which is resistant to outliers) as well as generalized linear models (which we will learn about next)

One might argue, however, that that since you don’t have a closed- form expression for the curve, you have to carry around a lot of points to represent your fit (this only really gets nasty in high-dimensional settings)

Also, that single knob, while convenient, means that to capture features like the peak at 67 million years ago, we have to choose a small bandwidth that produces an under-smoothed fit in other places (compare the 6 and 12 dof fits at 55 million years ago) -- This is why we might look at a number of different bandwidths to see what’s going on

Another extension

Finally, local polynomial fitting is in some sense less intuitive than other procedures we might consider that also use polynomials as a building block

For example, suppose that rather than fit a different curve at each prediction point, we instead divide our data up into regions and fit separate curves in each region

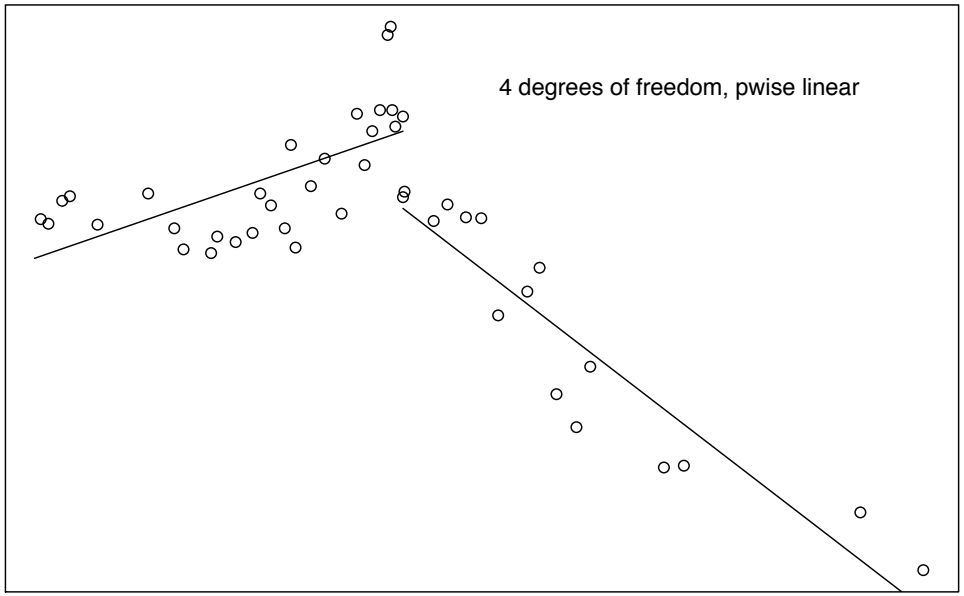
Simply put, we can move from global polynomials to piecewise polynomials -- This will be the subject of the rest of the lecture

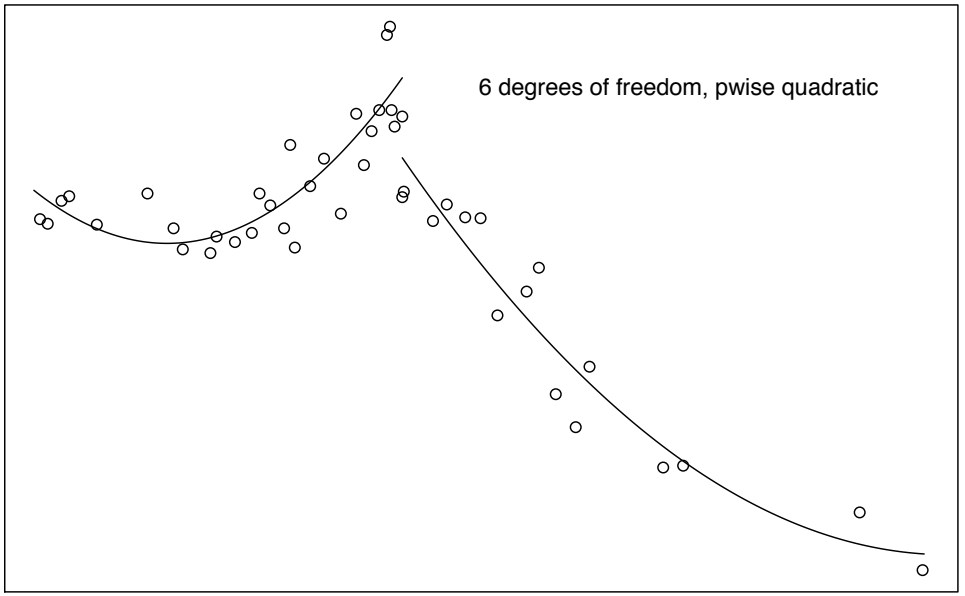
Piecewise polynomials

On the next page I present a number of piecewise polynomial fits that match the degrees of freedom for the global polynomials and the local polynomials from several slides back

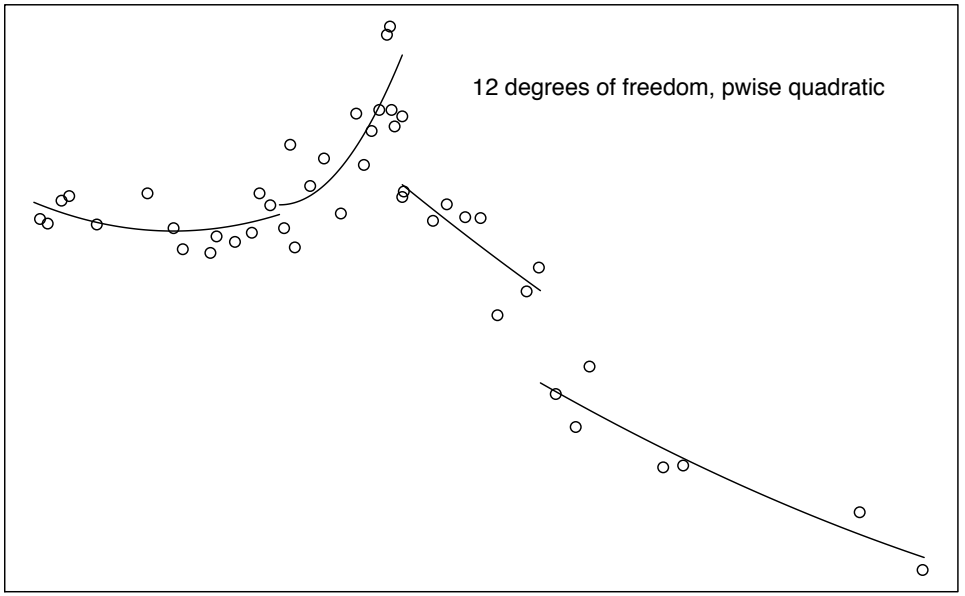
What do you notice? How do these fits compare?

```
# the piecewise linear fit, with 4 degrees of freedom

plot(kt$age,kt$sr)

newkt = data.frame(age=seq(43,67,len=100))
fit = lm(sr.ratio~poly(age,1),data=kt[kt$age<67,])
lines(newkt$age,predict(fit,newdata=newkt))

newkt = data.frame(age=seq(67,101,len=100))
fit = lm(sr.ratio~poly(age,1),data=kt[kt$age>67,])
lines(newkt$age,predict(fit,newdata=newkt))

text(85,0.7079,"4 degrees of freedom, pwise linear")

# the piecewise linear fit, with 6 degrees of freedom

plot(kt$age,kt$sr)

newkt = data.frame(age=seq(43,67,len=100))
fit = lm(sr.ratio~poly(age,2),data=kt[kt$age<67,])
lines(newkt$age,predict(fit,newdata=newkt))

newkt = data.frame(age=seq(67,101,len=100))
fit = lm(sr.ratio~poly(age,2),data=kt[kt$age>67,])
lines(newkt$age,predict(fit,newdata=newkt))

text(85,0.7079,"6 degrees of freedom, pwise quadratic")
```


4 degrees of freedom, pwise linear


6 degrees of freedom, pwise quadratic


8 degrees of freedom, pwise linear


12 degrees of freedom, pwise quadratic

Note

The code on the previous slide is pretty simple-minded -- We are literally dividing the data into two pieces and then fitting a separate linear polynomial in each piece

That means we can write down our “basis” functions pretty easily -- They are simply monomials constrained to the different intervals

We will generalize this in a minute...

Piecewise polynomials

Notice that accounting for the number of degrees of freedom in this case is easy -- If we have a piecewise linear fit, then we have two degrees of freedom in each “piece”

While this sort of model is intuitive, it has its own set of problems; for example, it’s not clear that we need the extra degrees of freedom that are involved in fitting separate polynomials in each interval

We might consider constraining our fits so that the final result is “smooth” -- How much smoothness can we ask from a piecewise linear fit? A piecewise 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)

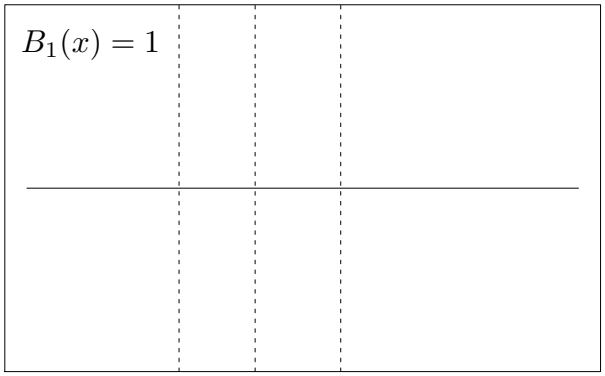
How many degrees of freedom do we have now? What "basis" functions do we use here?

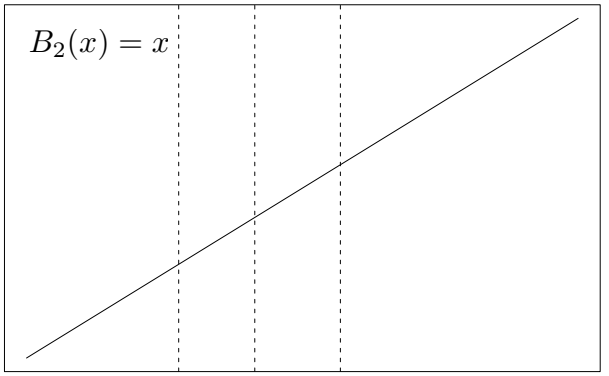
Basis choice

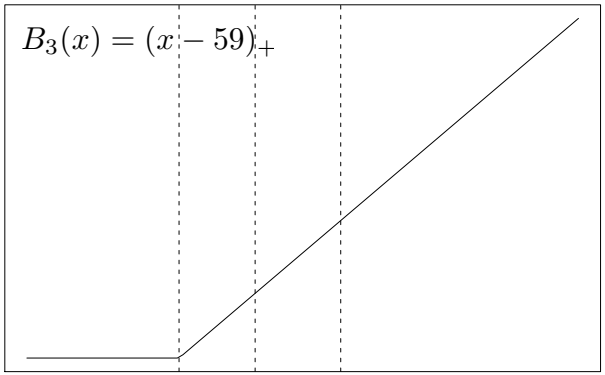
Rather than try to solve a "constrained" least squares problem in which we force the ends of the lines to join, it's easier to work with basis functions that have the conditions we're after

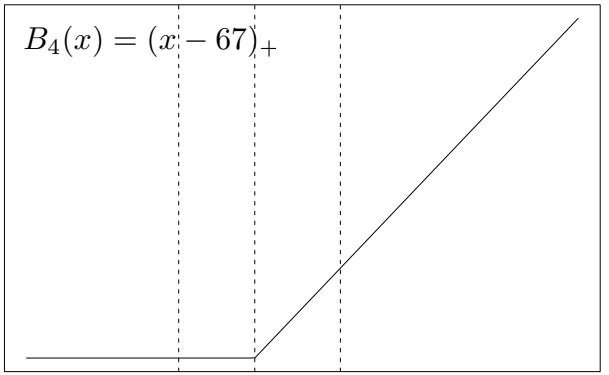
At the right we show five basis functions that can be used in a fitting routine to create the broken-stick model on the previous slide

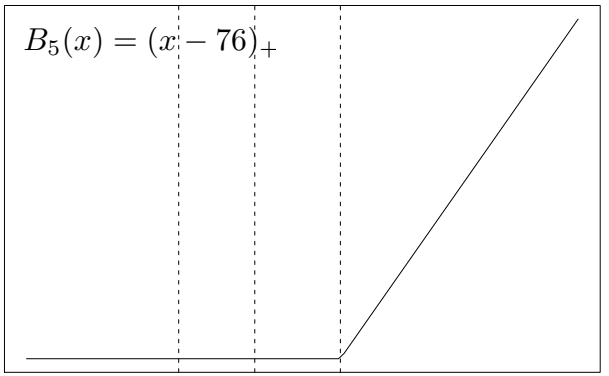
This is called the **truncated power basis**; anything interesting here?











Truncated power basis

The truncated power basis is especially convenient for statistical applications; in particular, you see that there is one basis function per breakpoint or knot

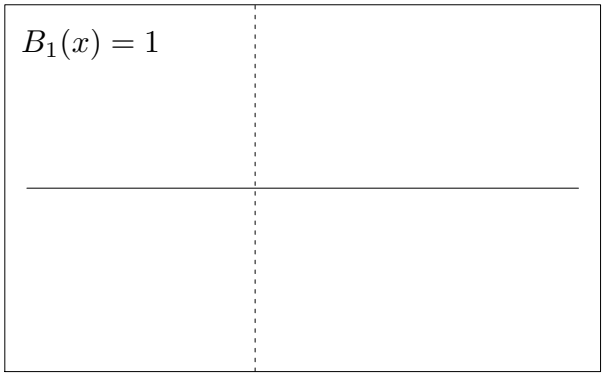
That means, that adding or deleting a single basis function adds or removes flexibility to the estimated regression function

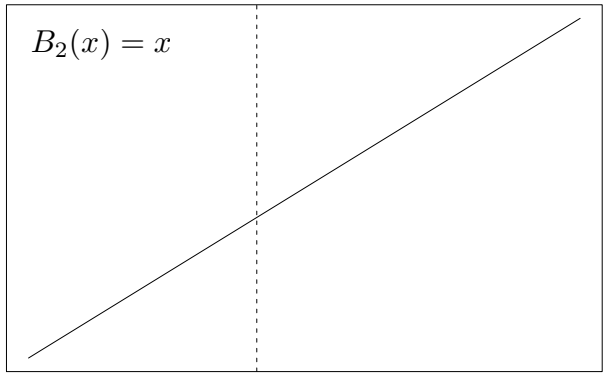
How might this prove useful?

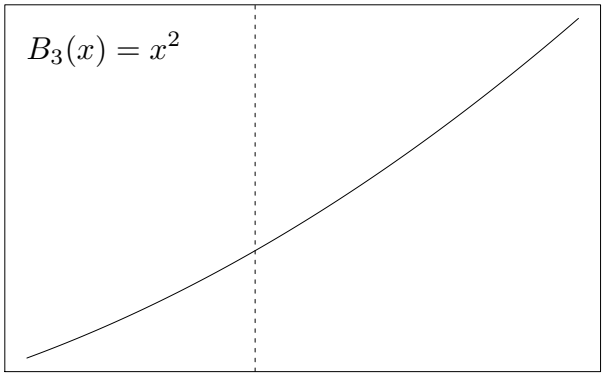
Truncated power basis

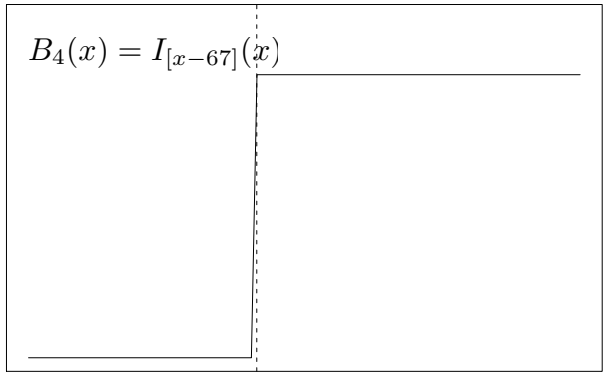
The truncation idea generalizes pretty easily; here is the basis associated with the full set of piecewise quadratics

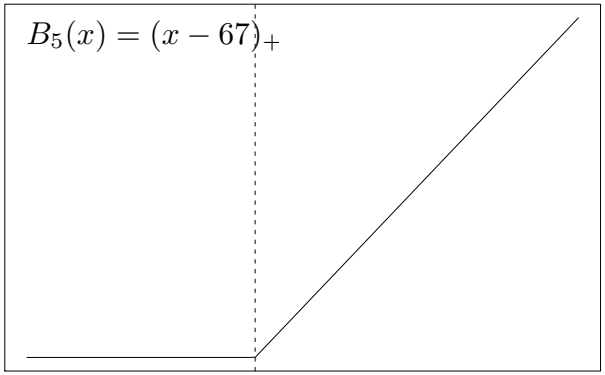
If we wanted to enforce continuity across $x = 67$ what would we do? What about creating one continuous derivative? Two continuous derivatives?

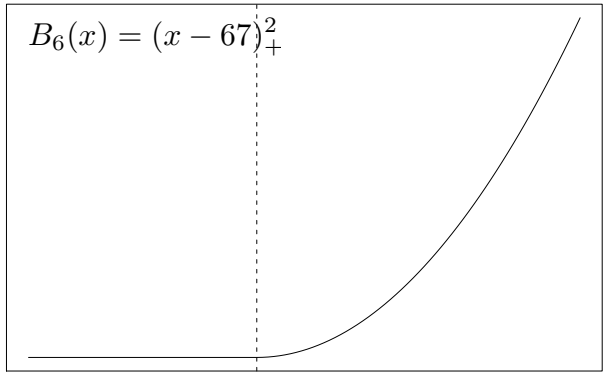


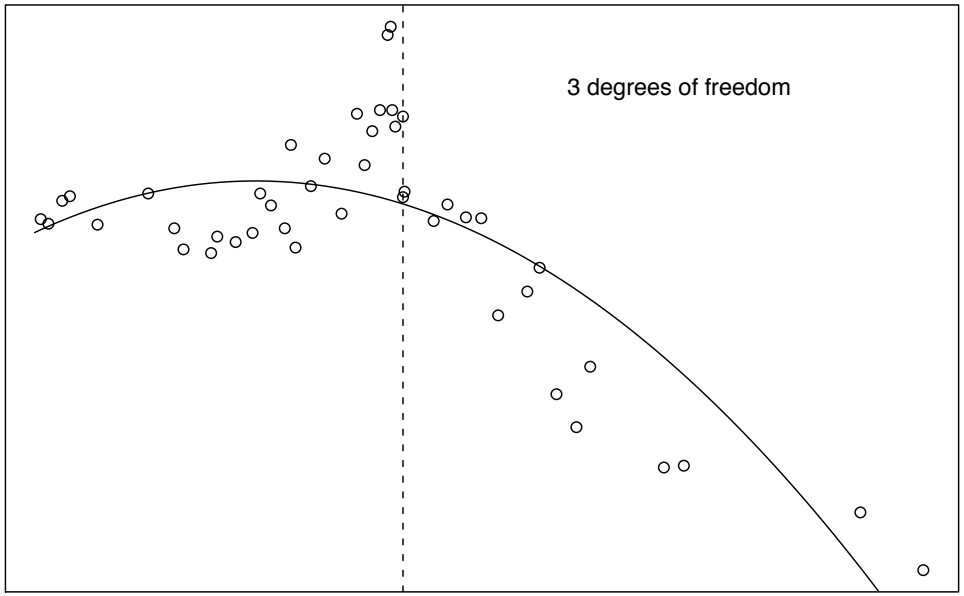


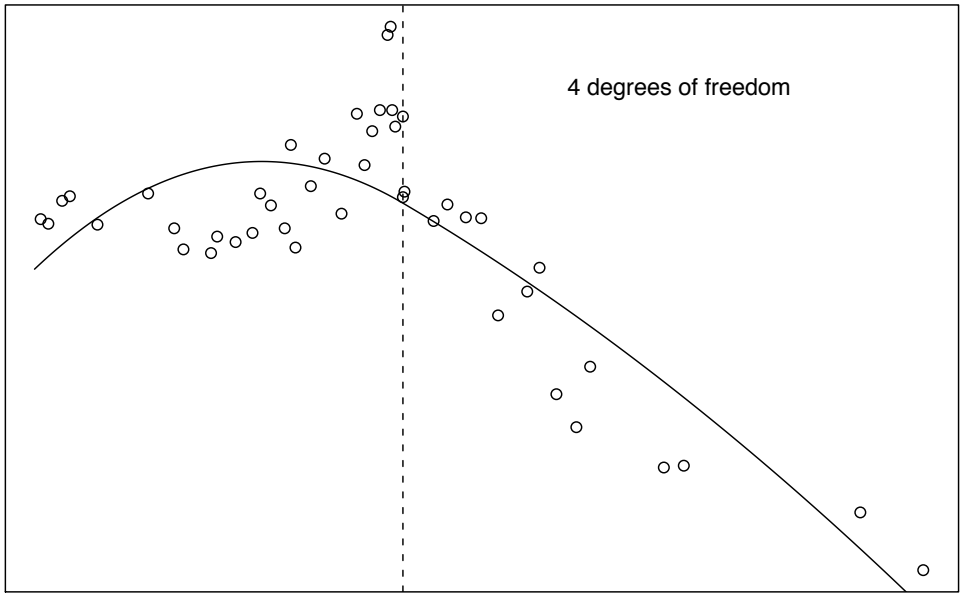


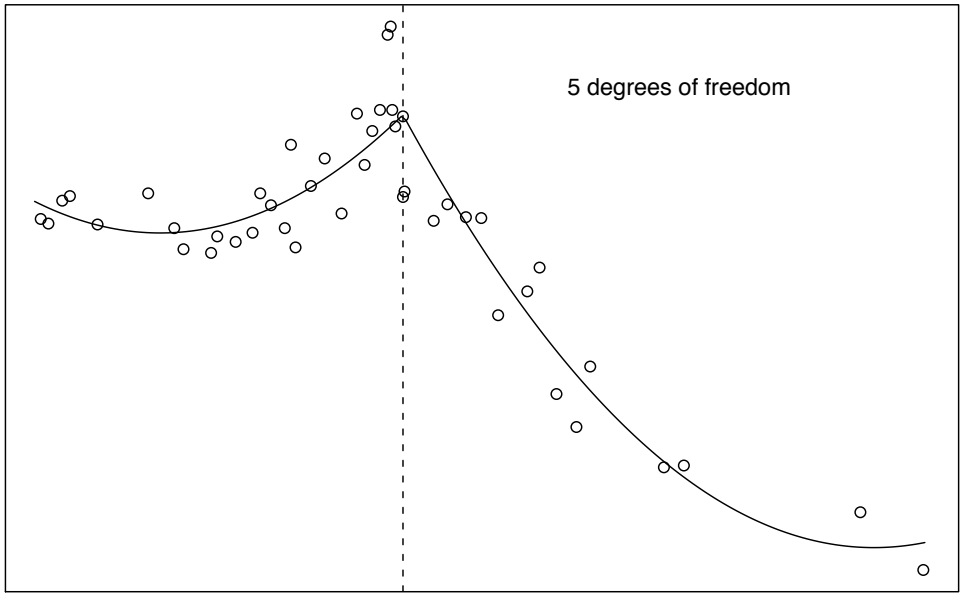


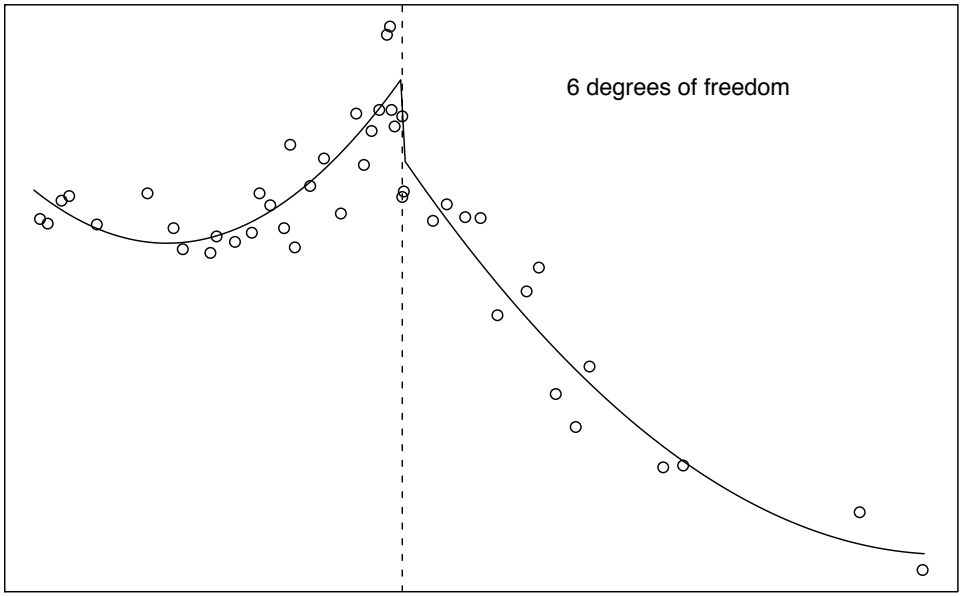





3 degrees of freedom


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
```

Splines

While testing for a break (or a changepoint as it is commonly called) is an important task, we often appeal to piecewise polynomials to provide a good, general purpose smoother

For that, then, we typically work with the “smoothest” space that does not degenerate into a single, polynomial piece: For piecewise linear functions, that is continuity; for quadratics, we enforce one continuous derivative; and for cubics, two continuous derivatives

We will refer to piecewise polynomial spaces that have this “maximal” smoothness property as spline spaces; in particular, linear splines, quadratic splines and cubic splines, respectively

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

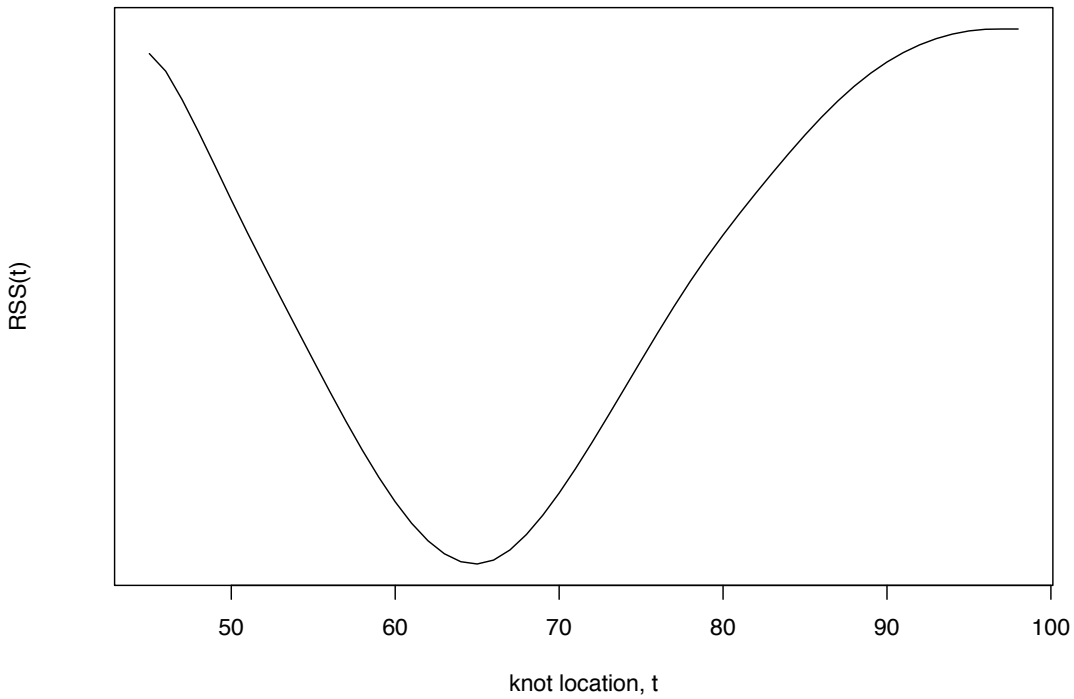
For each value of t, we can fit an ordinary regression; we can then associate with each value of t the error $RSS(t)$

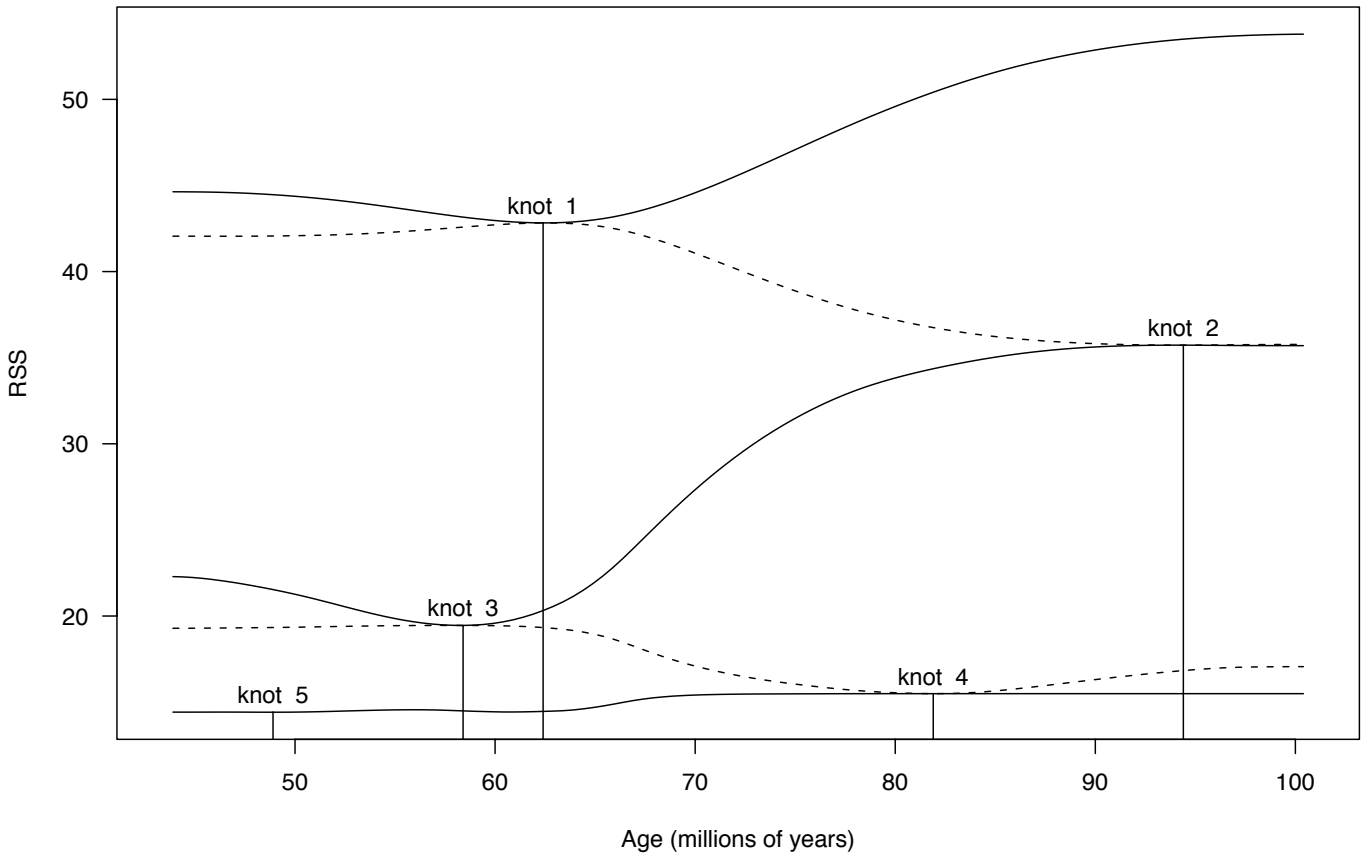
We could then find the location that minimizes $RSS(t)$; in this case it occurs at $t = 65$

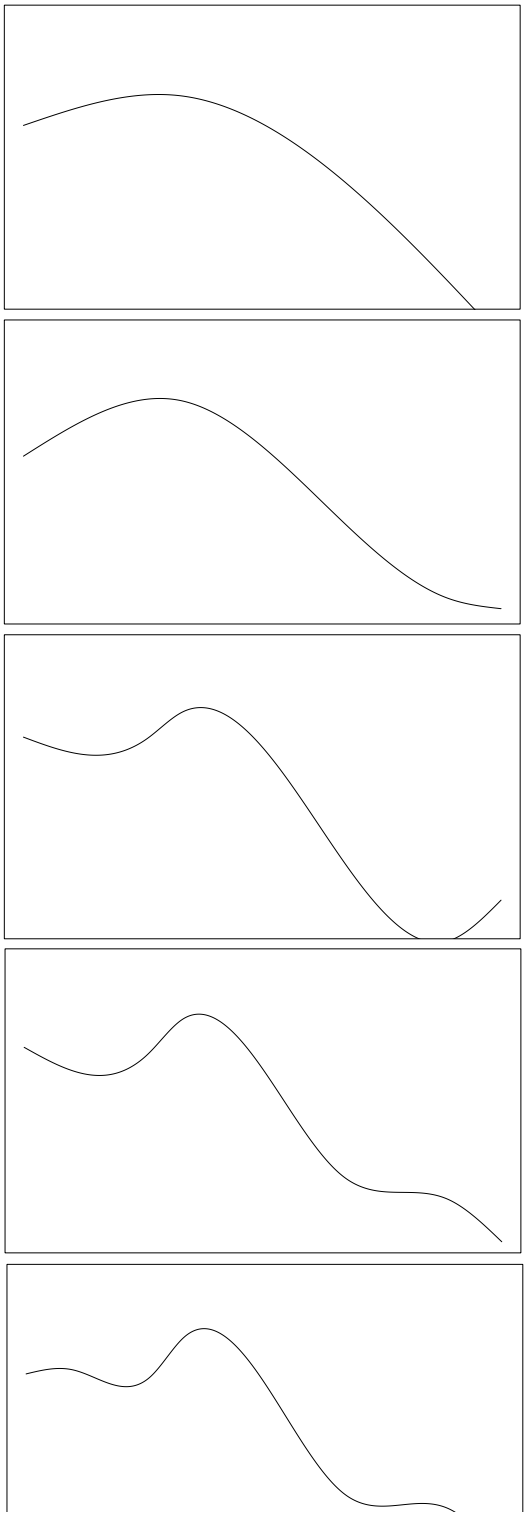
Having picked the first knot, we can then start with the basis

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

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)



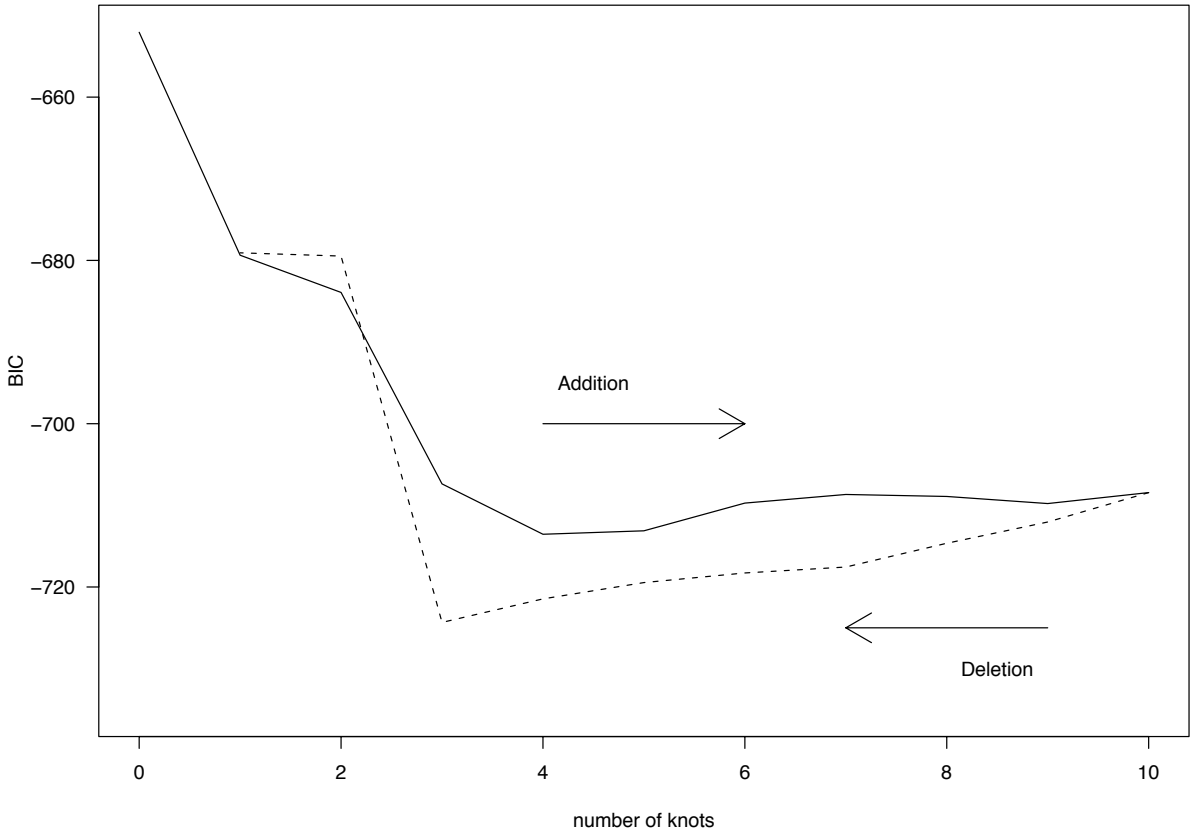
What does this remind you of?

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

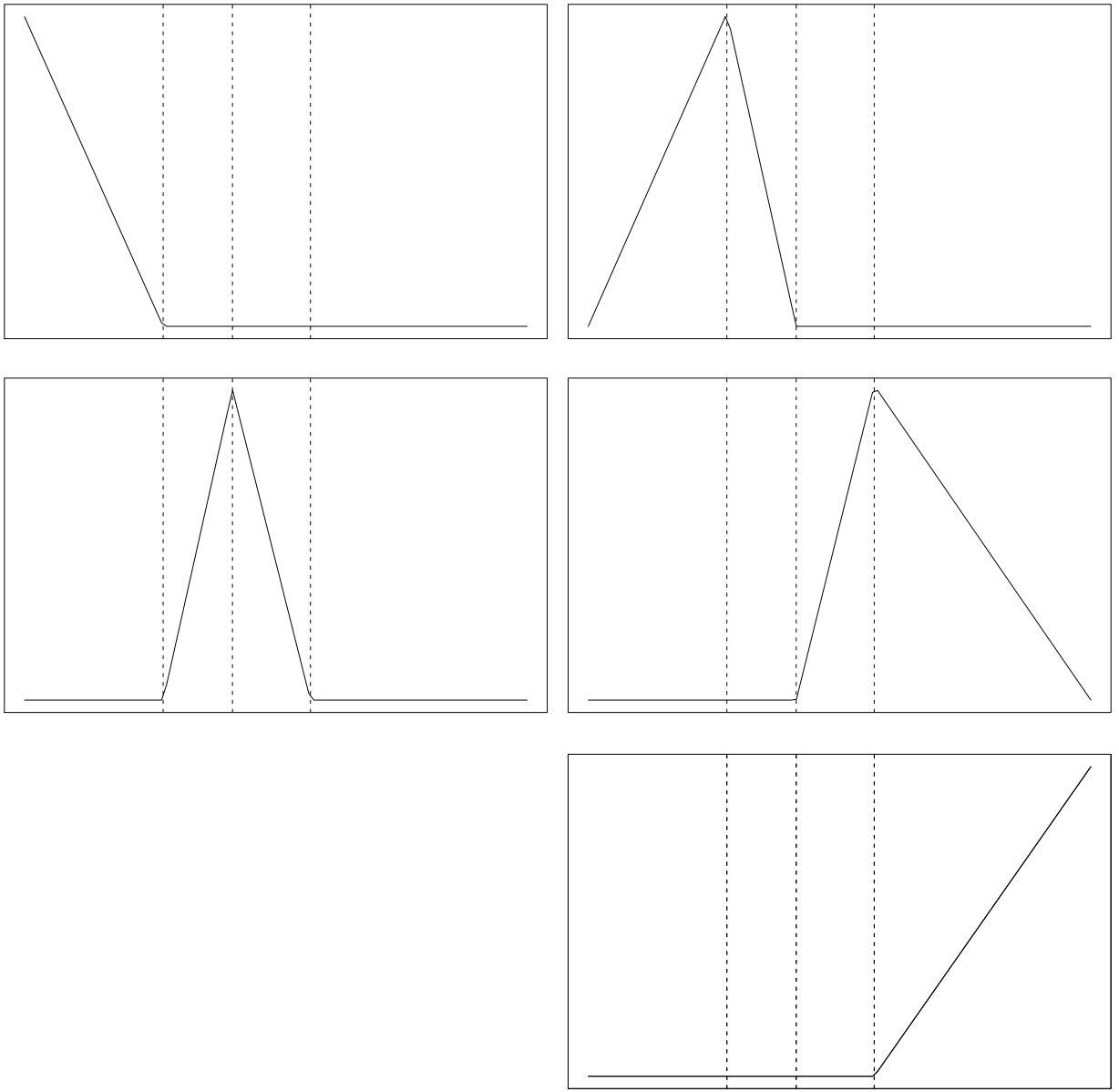
That said, it is not the only basis representation...

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

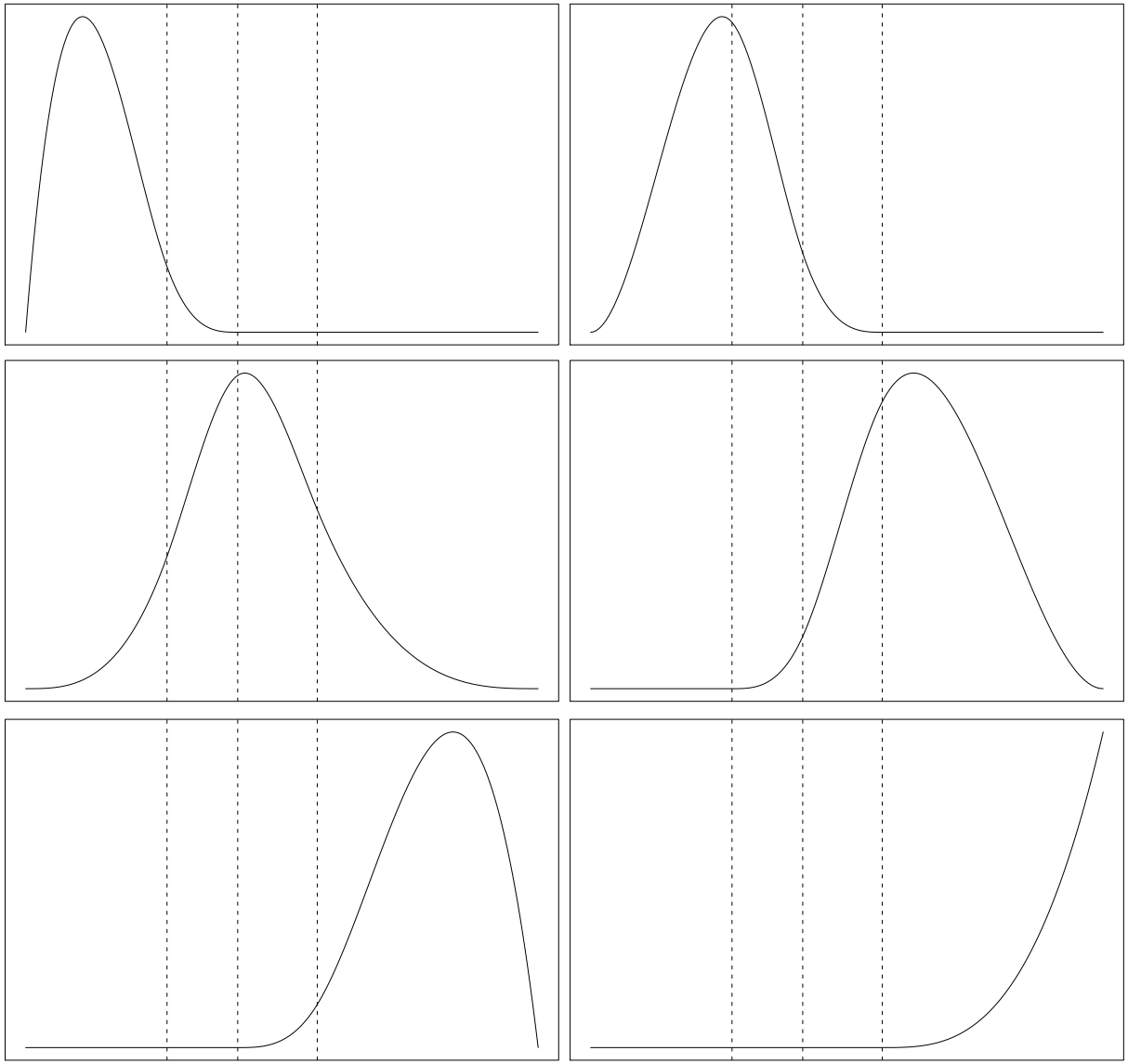
An obvious down-side to the truncated power basis is that it is just that, a series of truncated powers; all of the instability we associated with polynomials (that led us to orthogonal polynomials) applies for this basis

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

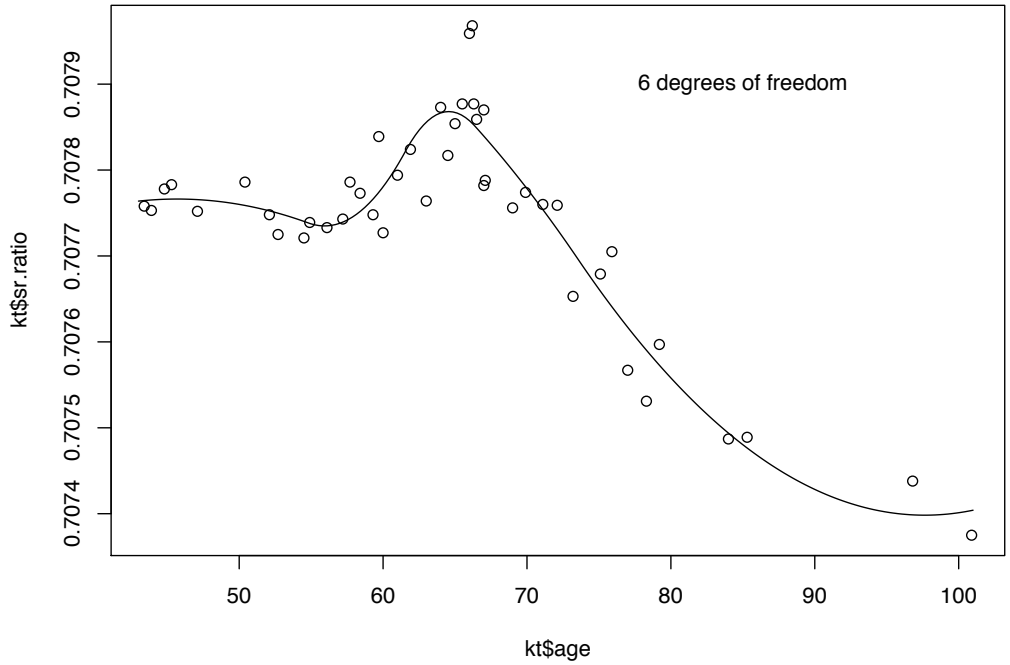
Here we present the basis for the quadratic B-splines (piecewise quadratic functions that are continuously differentiable); what does their shape remind you of?

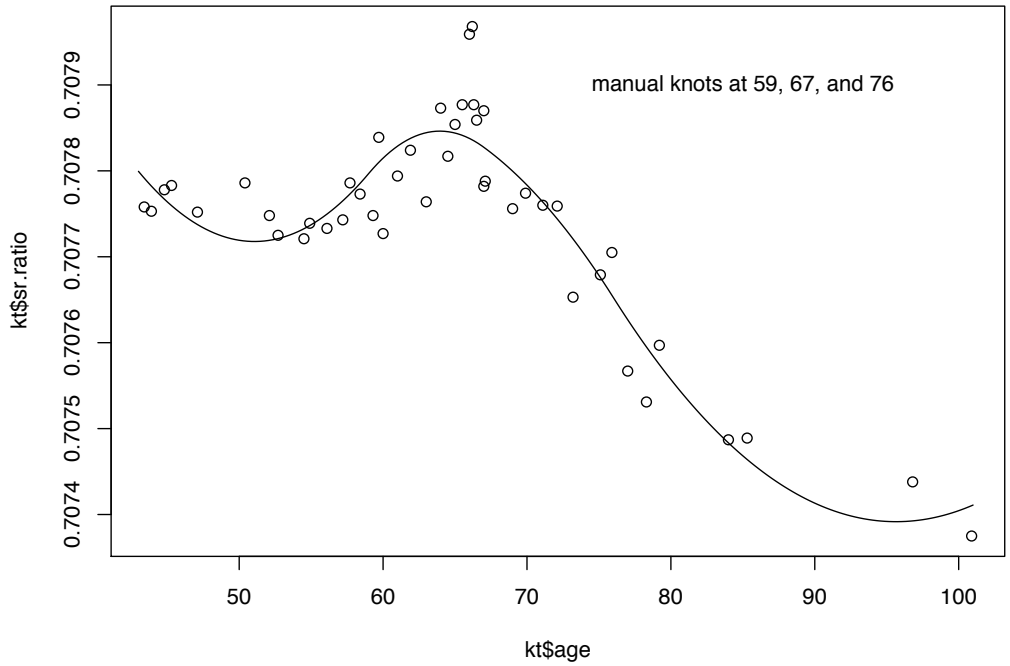


Modeling with B-splines

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



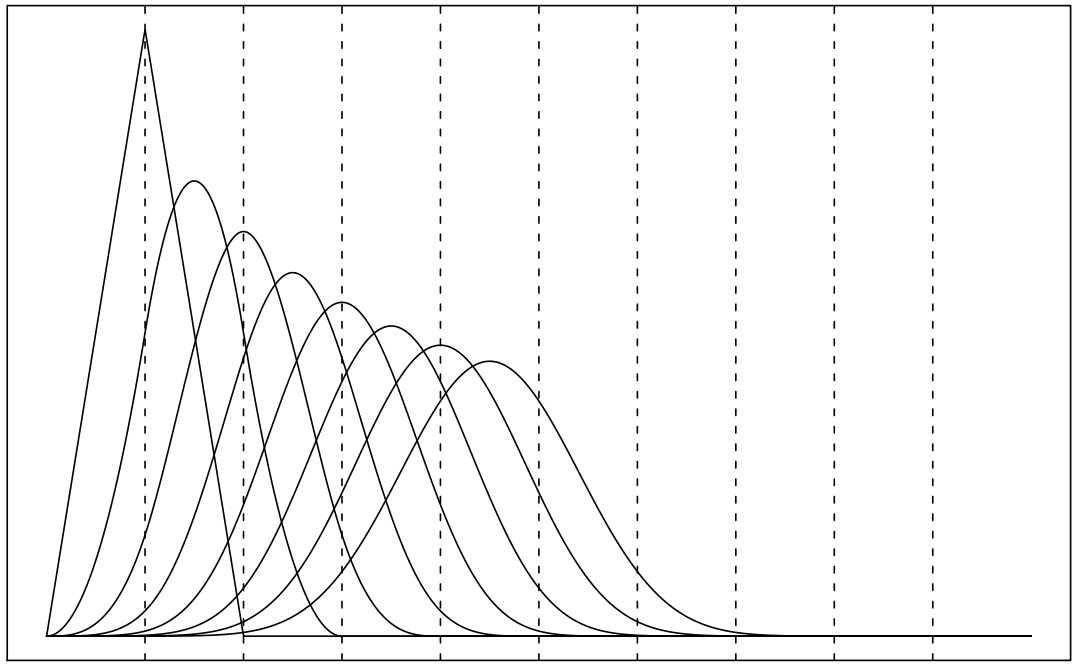


```
library(splines)

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

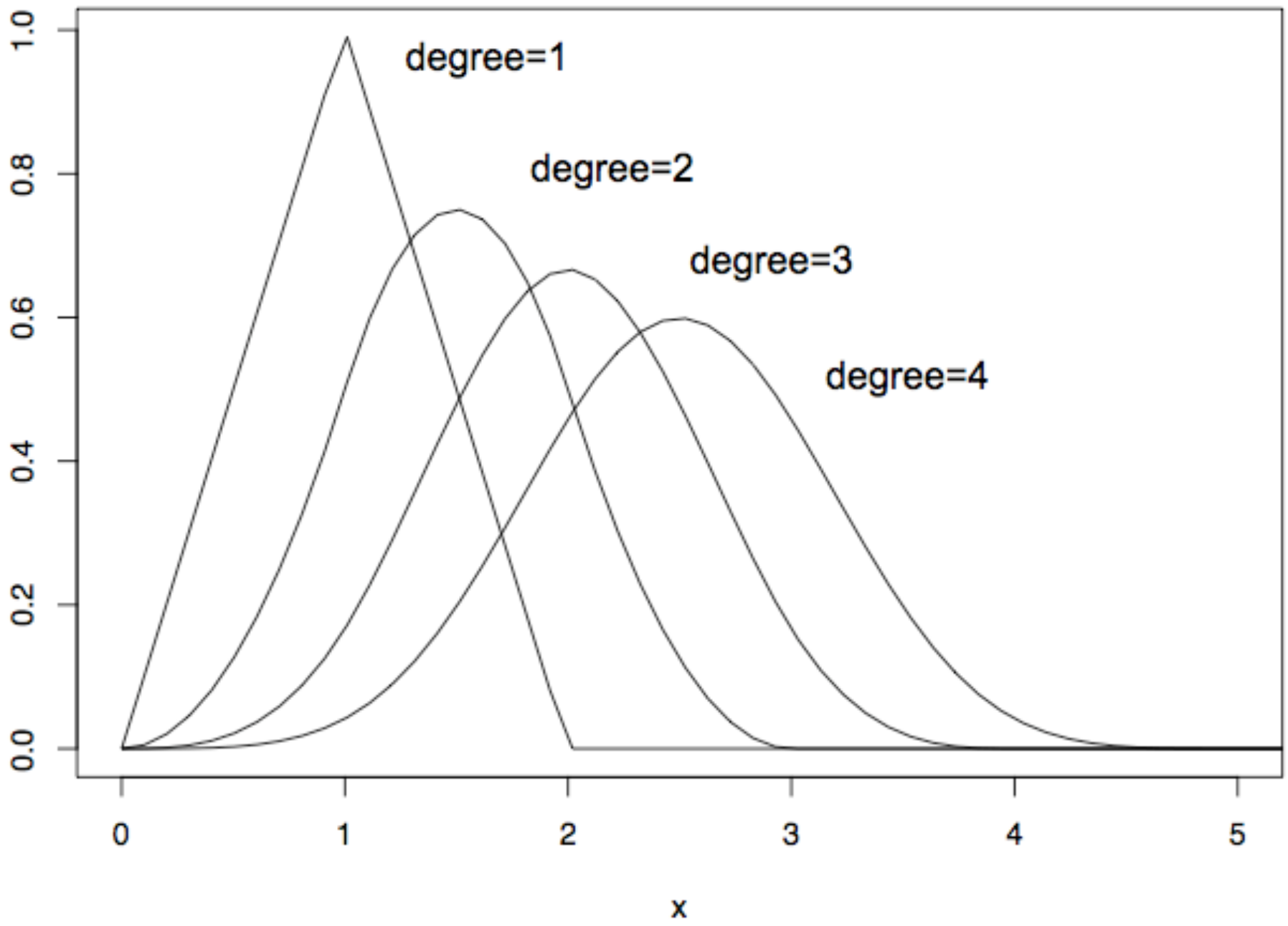
I leave you with a question...

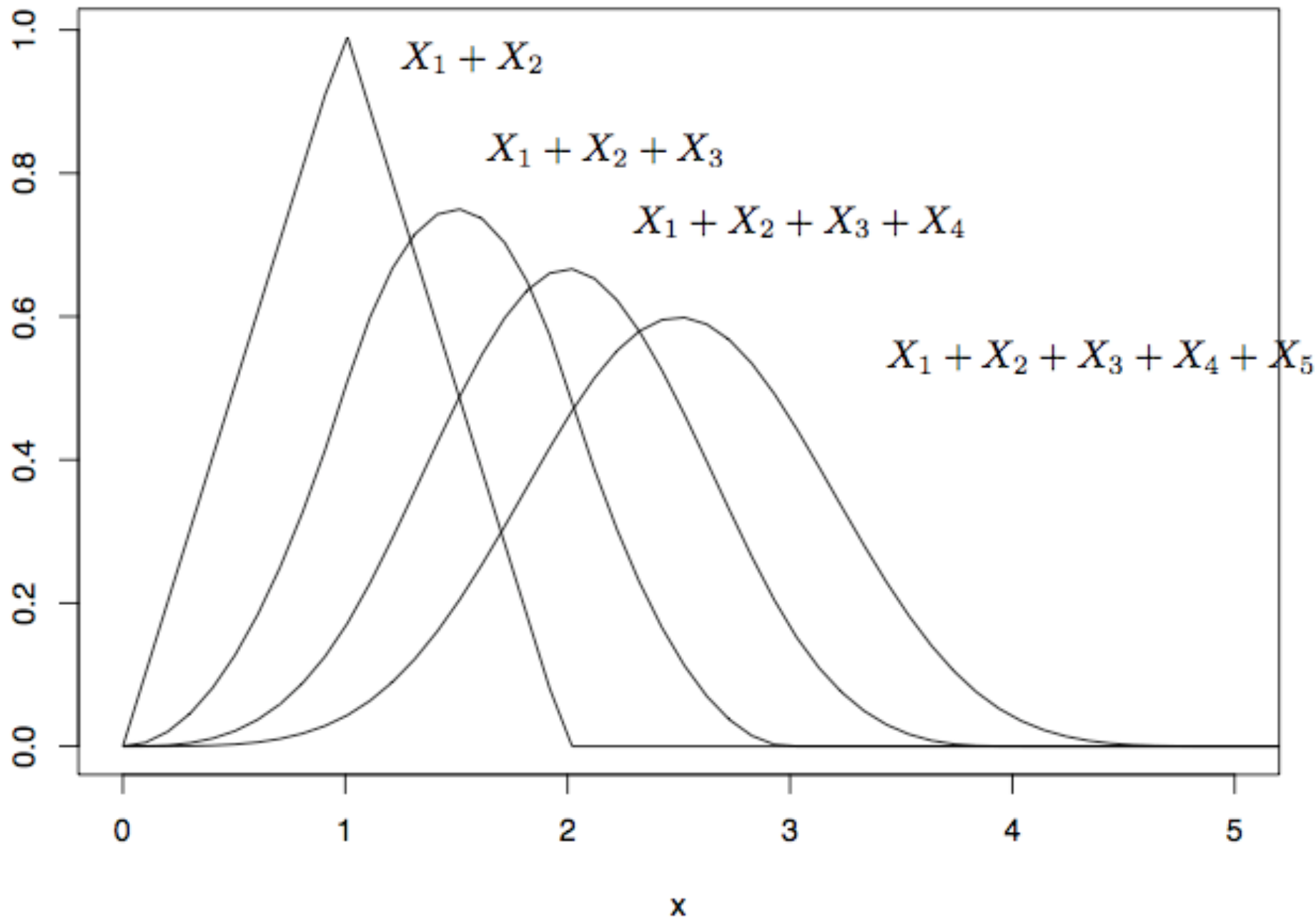
At the right we have a series of spline basis functions, each associated with a higher and higher degree space; the first corresponds to linear splines, then quadratic, the cubic and so on



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

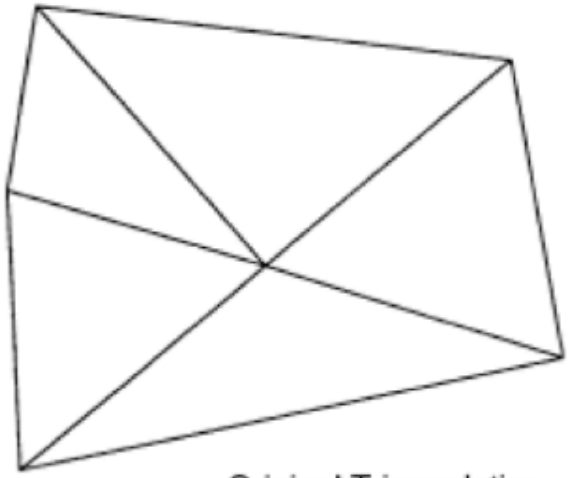
Multivariate splines

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

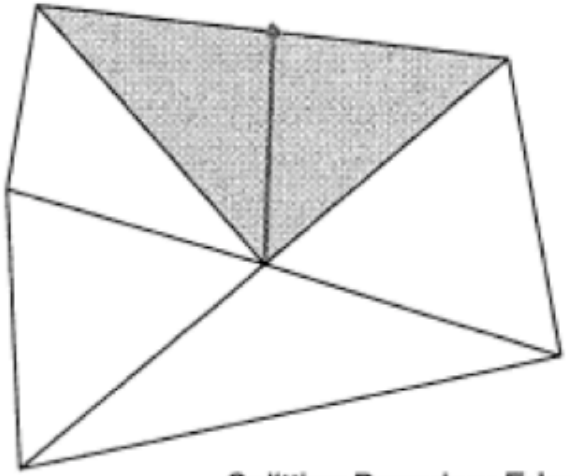
There is one line of reasoning that tries to build genuinely multivariate spline spaces; starting with an extension of "piecewise" polynomials, we could define regions in the plane, say, to be our pieces

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

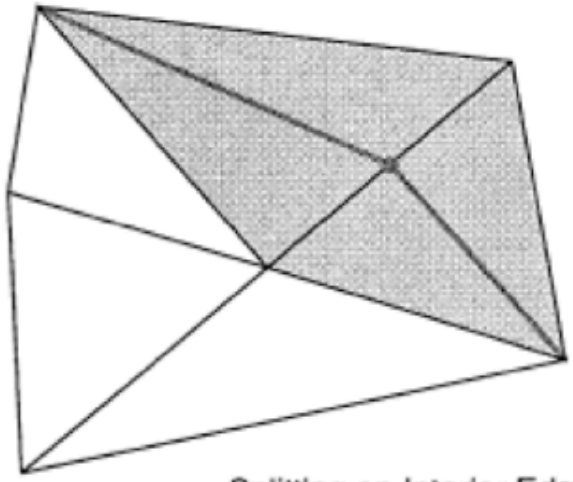
Here's one simple example...



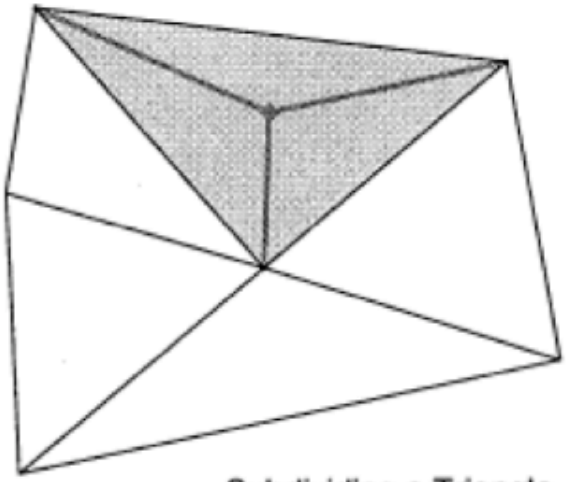
Original Triangulation



Splitting Boundary Edge



Splitting an Interior Edge

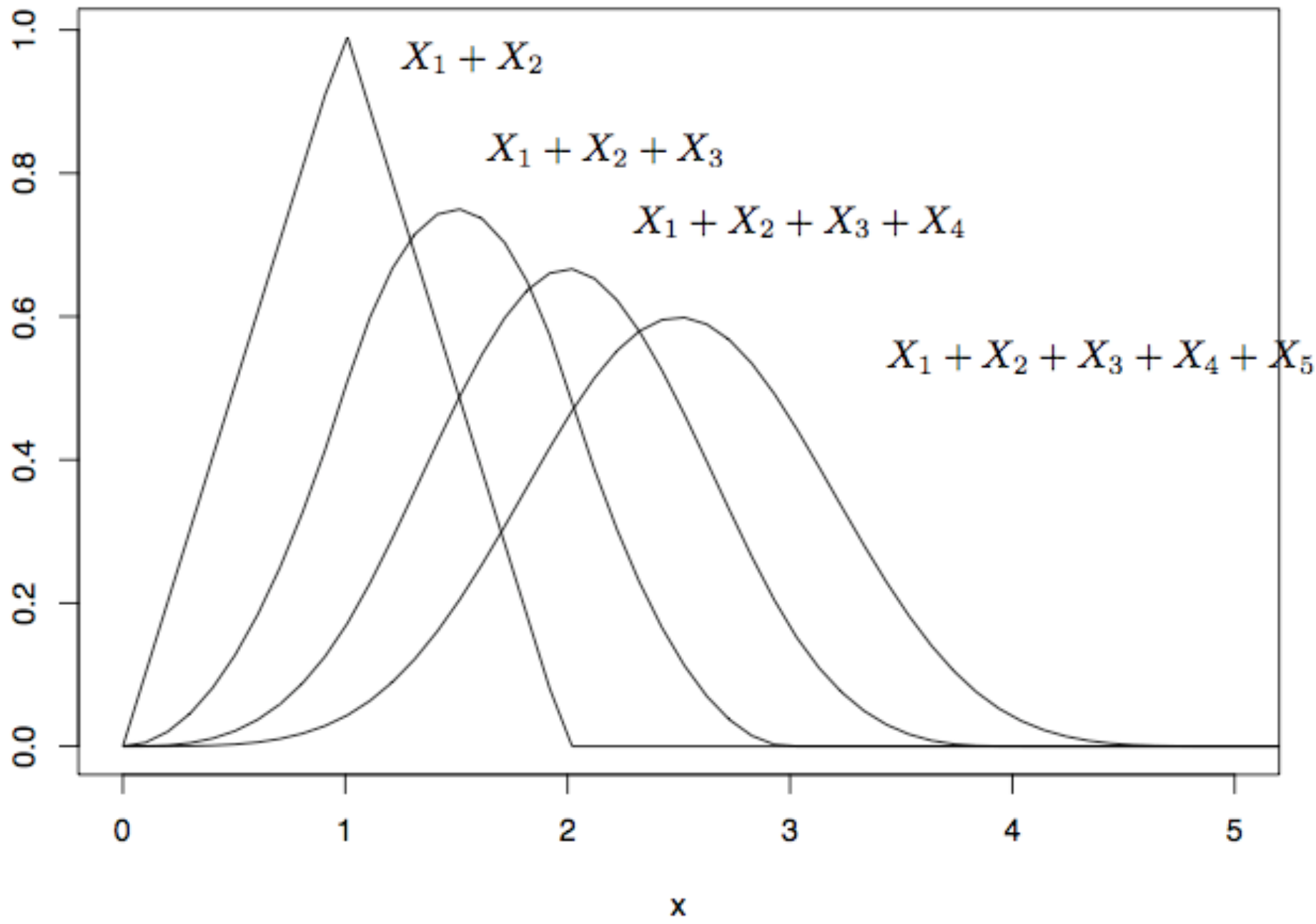


Subdividing a Triangle

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

Adaptive fits

Strategies for knot placement that depend only on the distribution of the design points are non-adaptive -- They can be written in the form $\widehat{\mu} = \mathsf{S}(\lambda)\mathrm{y}$ for some design-dependent complexity parameter (degree of the polynomial, size of the bandwidth, number of knots)

When we peak at the response y, we venture into the territory of adaptive estimation -- When we use subset selection procedures, for example, to place knots, we are giving our smoother the ability to recognize and capture peaks in the response data, adding, perhaps, complexity in regions which seem to require it

Adaptive fits

Interestingly, many of the strategies we have examined for working with the standard linear model have analogs in adaptive fitting -- The interplay between selection and shrinkage also has a story here...

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

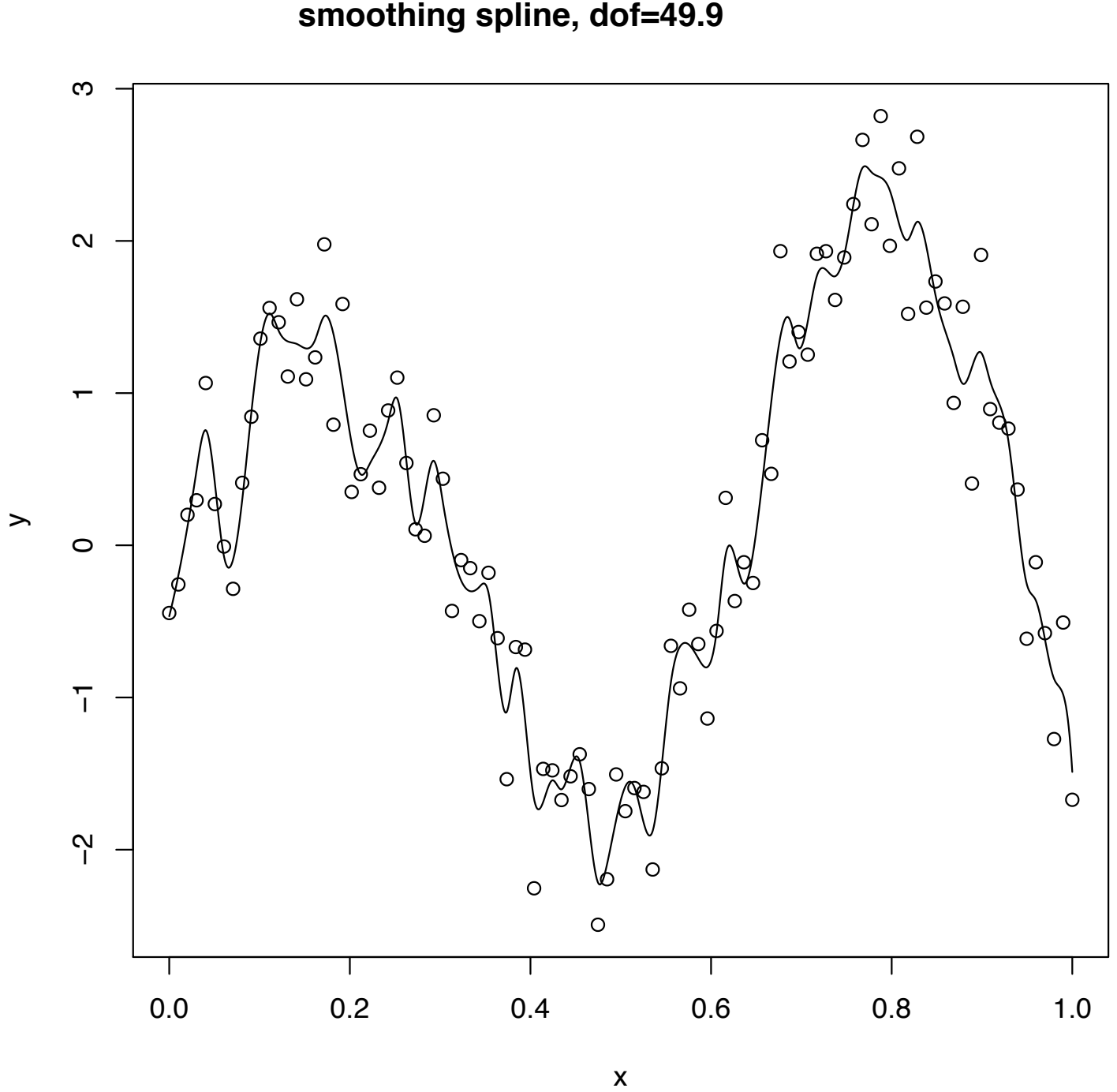
Univariate smoothing splines

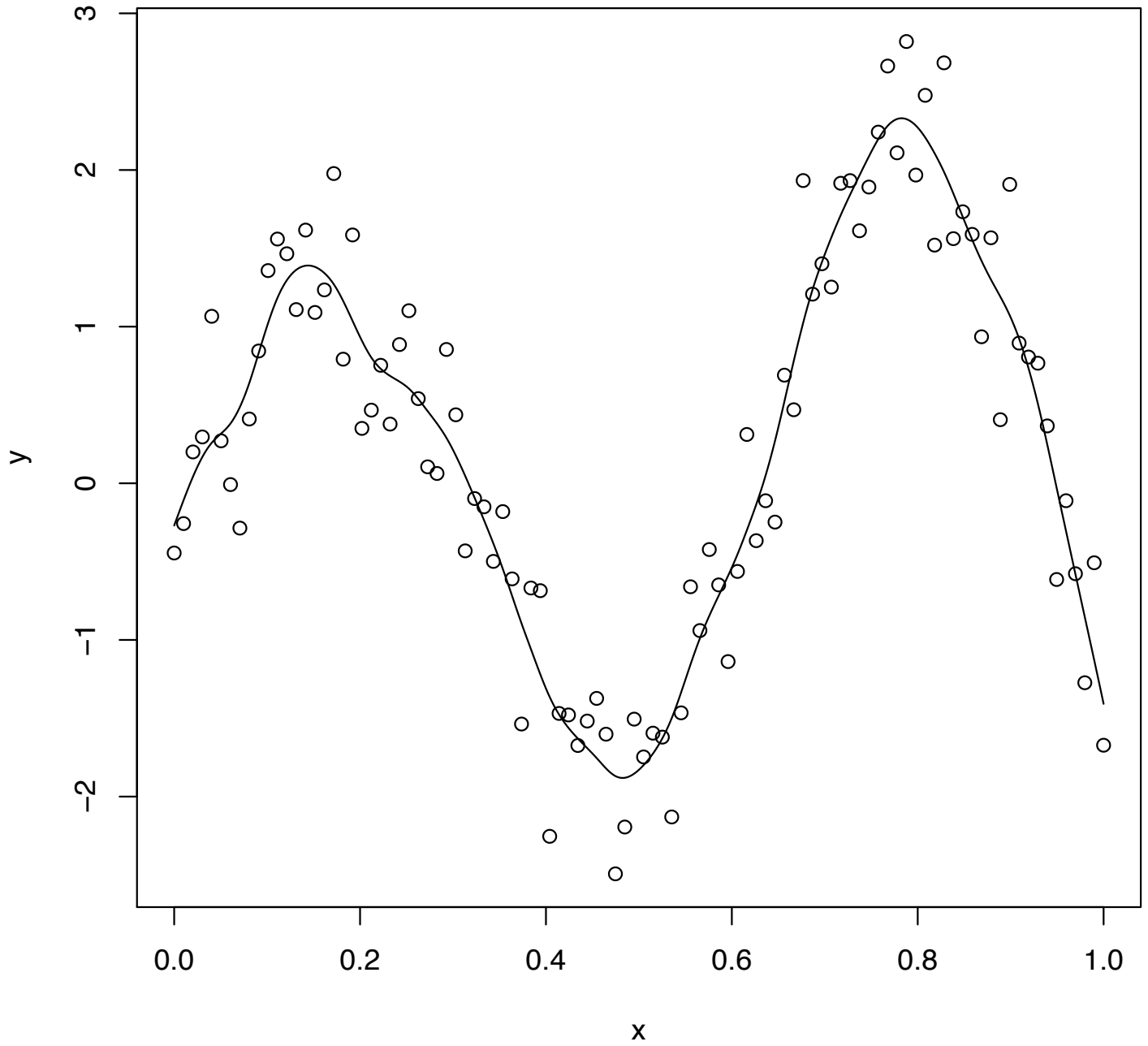
Remarkably, this problem has a closed-form solution; assuming that the ordinary regression problem (linear) has a unique solution, then the penalized criterion has a minimizer of the form

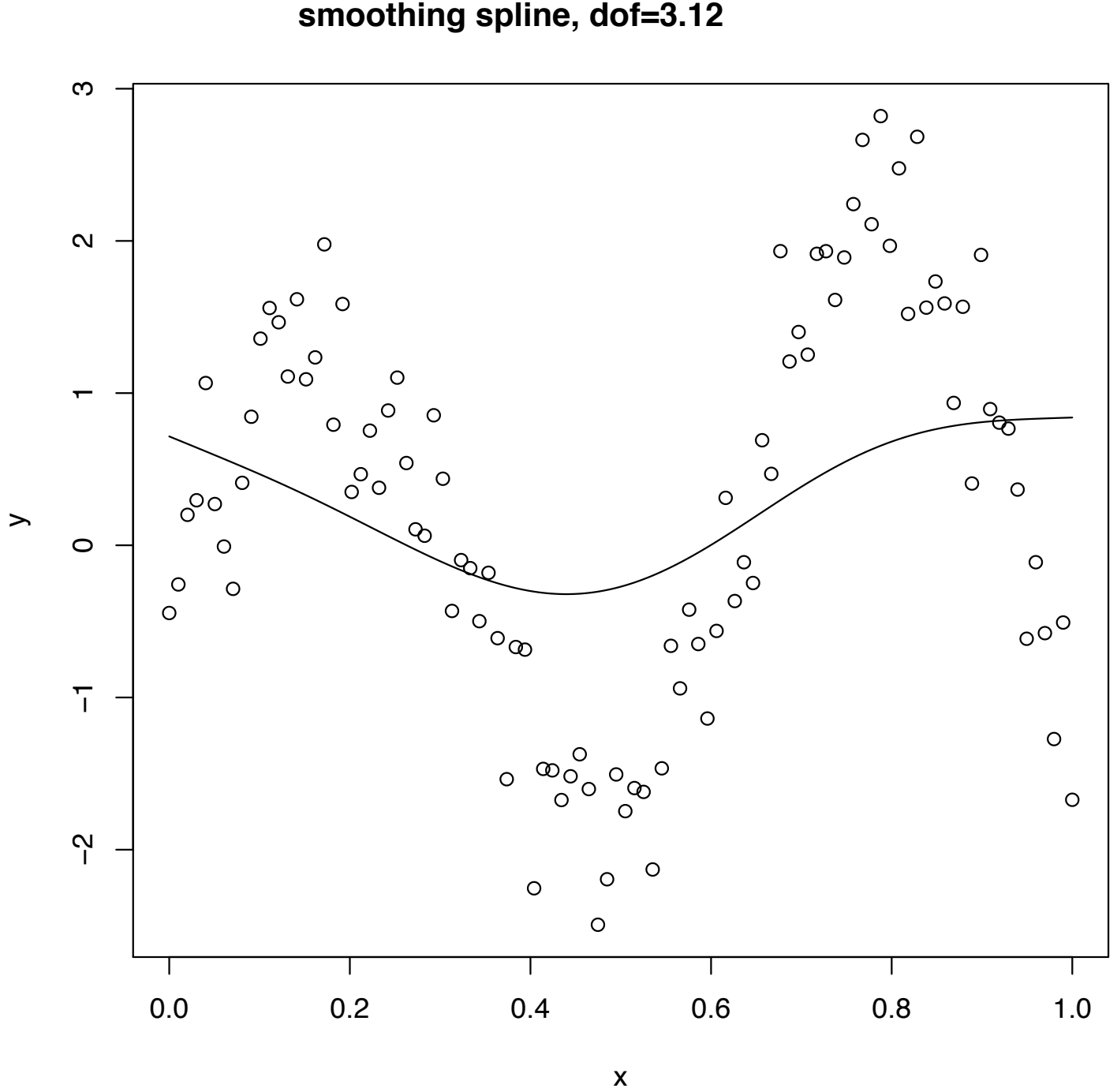
$$\widehat{f}(x) = \begin{cases} \text{linear polynomial} & x \in (-\infty, x_1] \\ \text{cubic polynomial} & x \in [x_j, x_{j+1}] \\ \text{linear polynomial} & x \in [x_n, \infty) \end{cases}$$

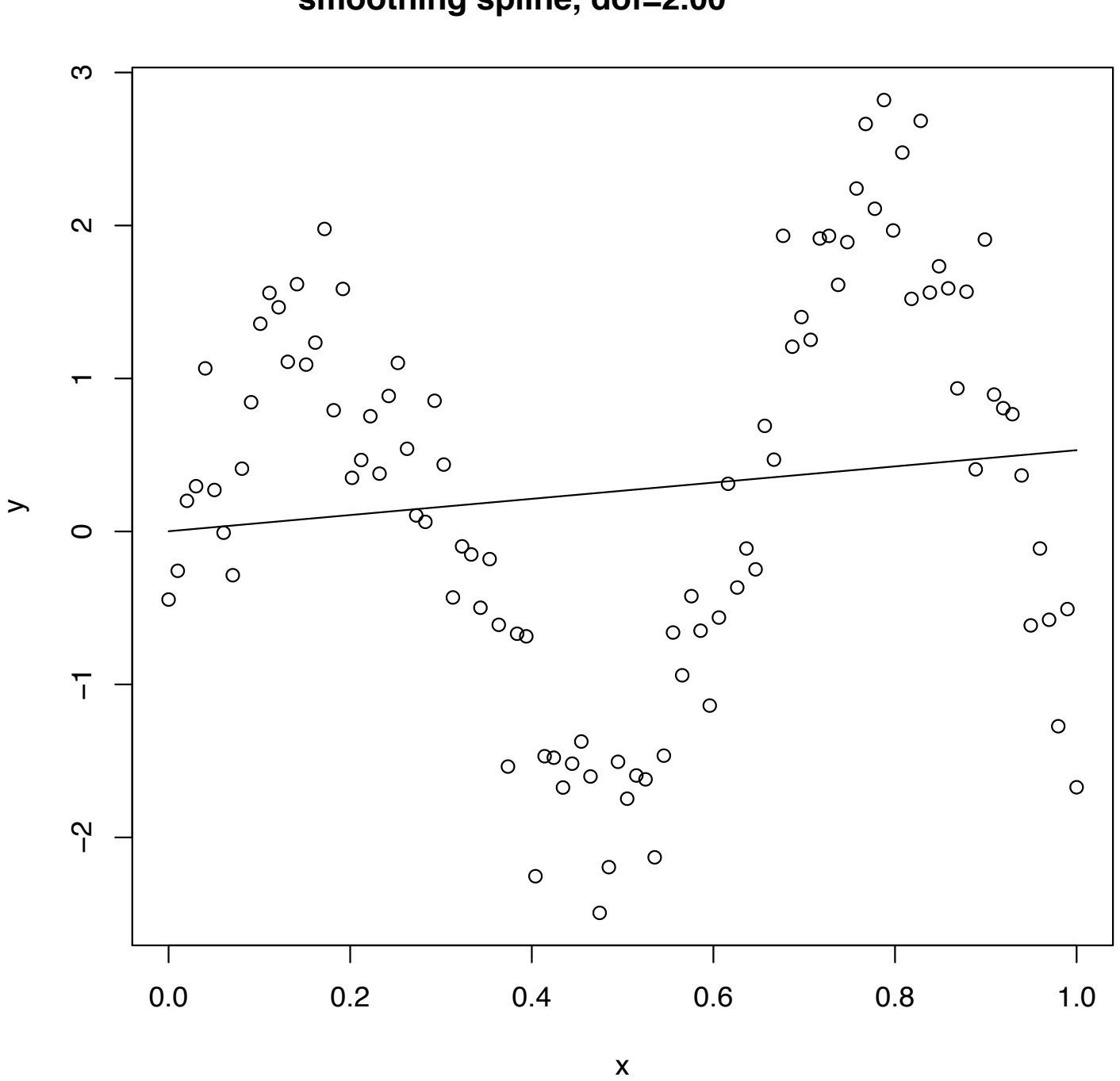
Further, $\widehat{f}(x)$ has two continuous derivatives; so, piecewise cubic polynomial, smooth across the knots -- the solution is a cubic spline with breaks at each of the data points!

```
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

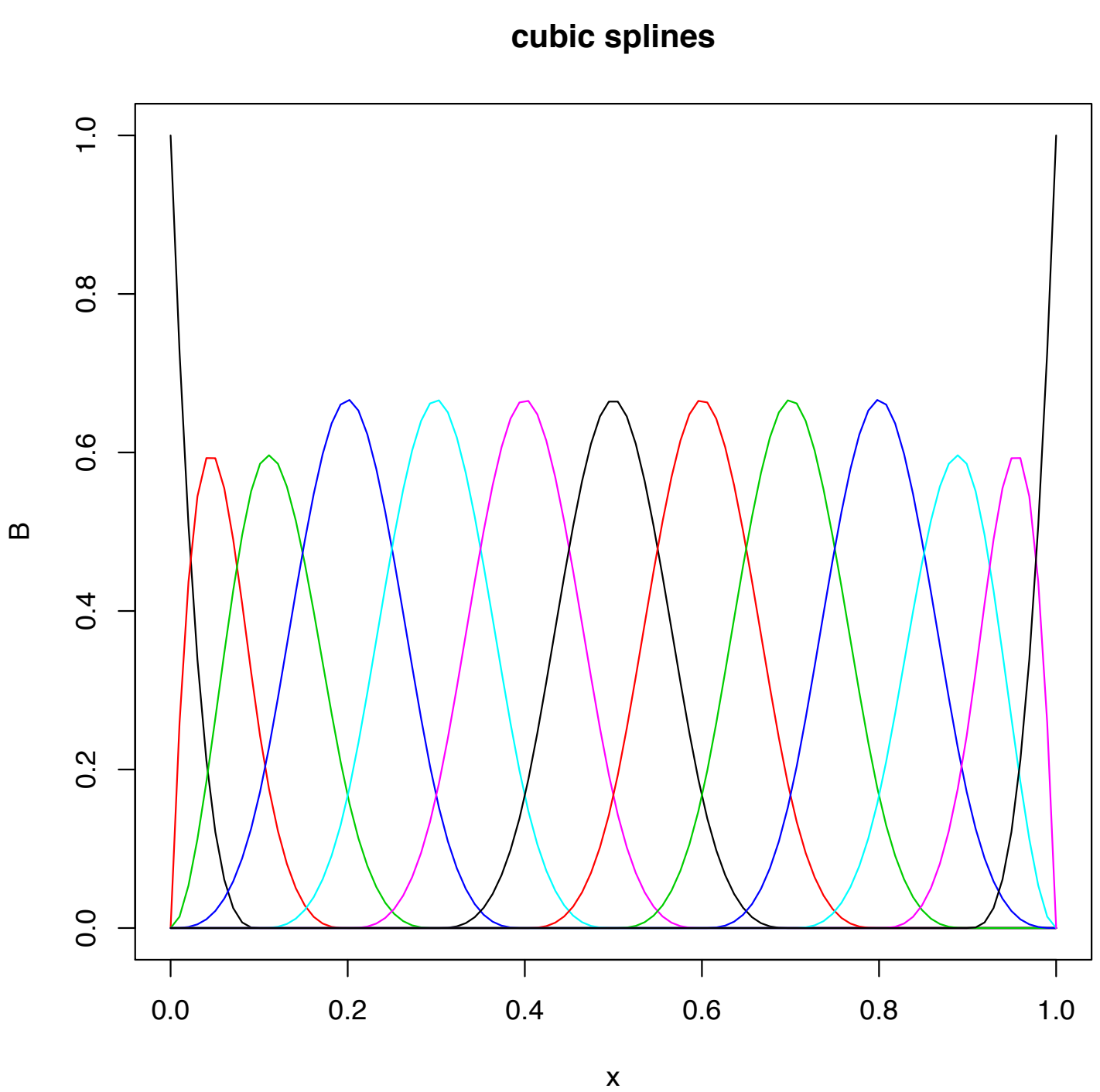
Basis set

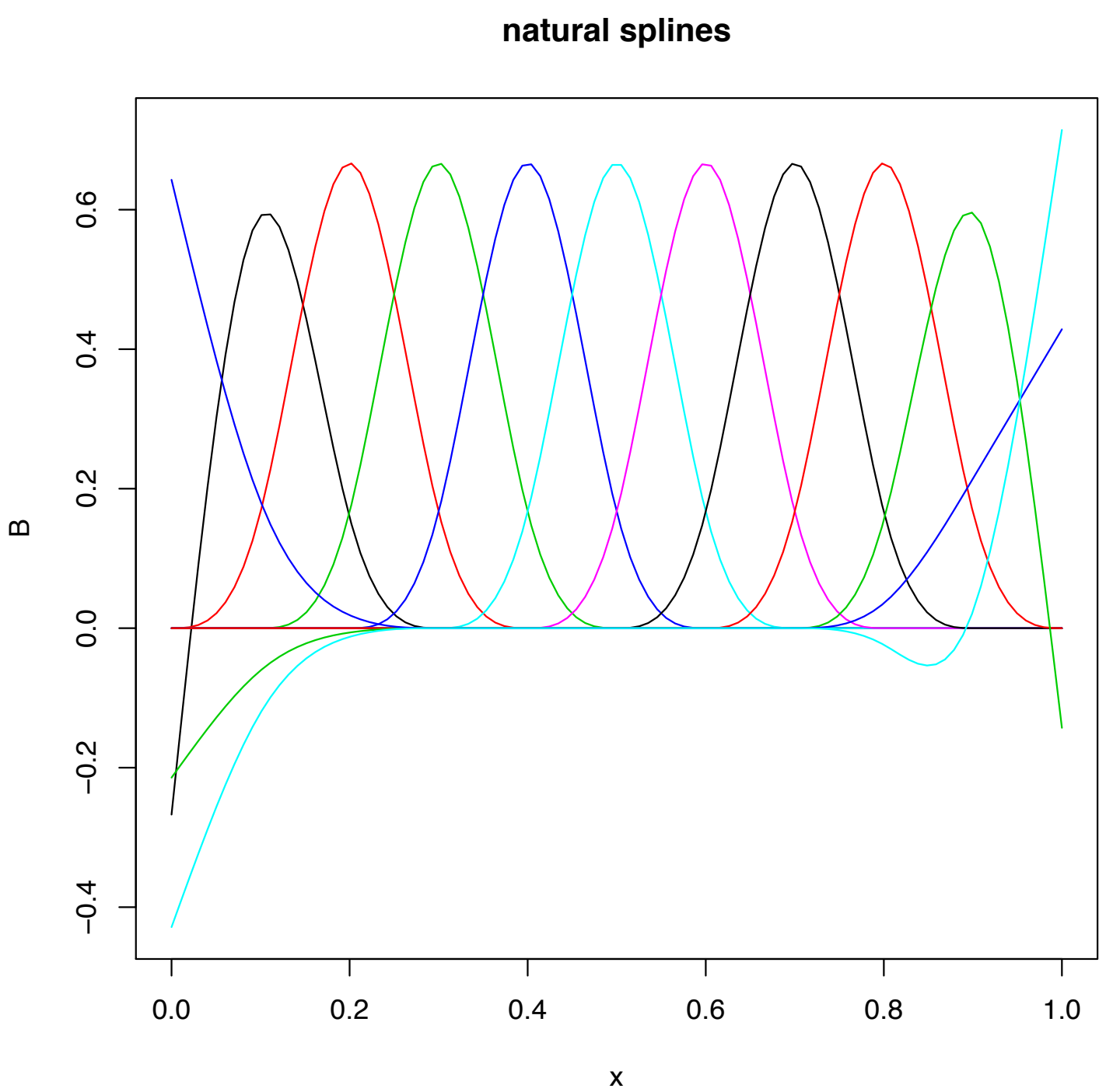
The solution to the smoothing spline problem is a little different than a simple spline space; beyond the first and last knot, the function is a line -- this means that at the first and last knots we have

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

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

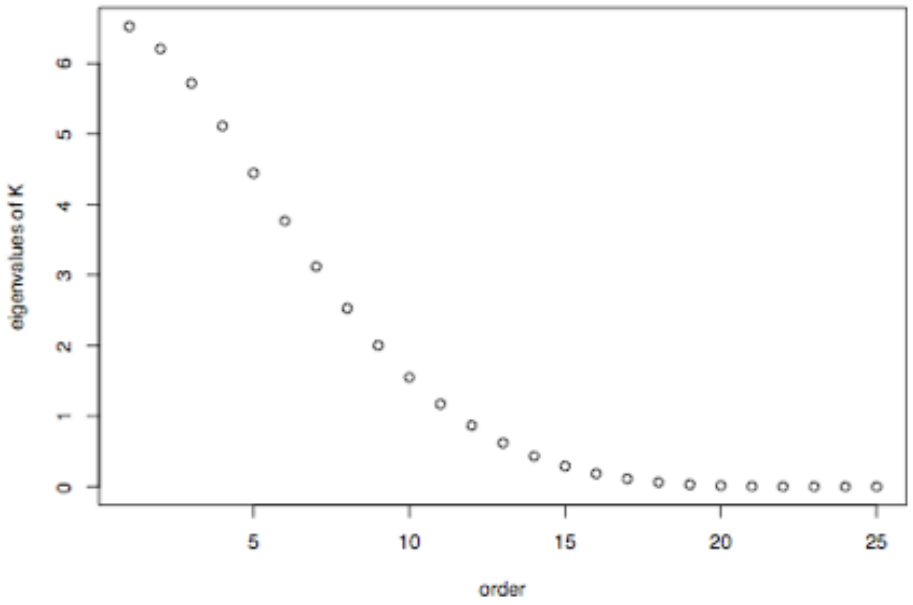
$$\begin{aligned}
\widehat{\boldsymbol{y}} &= \mathbf{B}\left(\mathbf{B}^t[\boldsymbol{I} + \lambda\mathbf{B}^{-t}\Lambda\mathbf{B}^{-1}]\mathbf{B}\right)^{-1}\mathbf{B}^t\boldsymbol{y} \\
&= \left(\boldsymbol{I} + \lambda\mathbf{B}^{-t}\Lambda\mathbf{B}^{-1}\right)^{-1}\boldsymbol{y} \\
&= (\boldsymbol{I}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

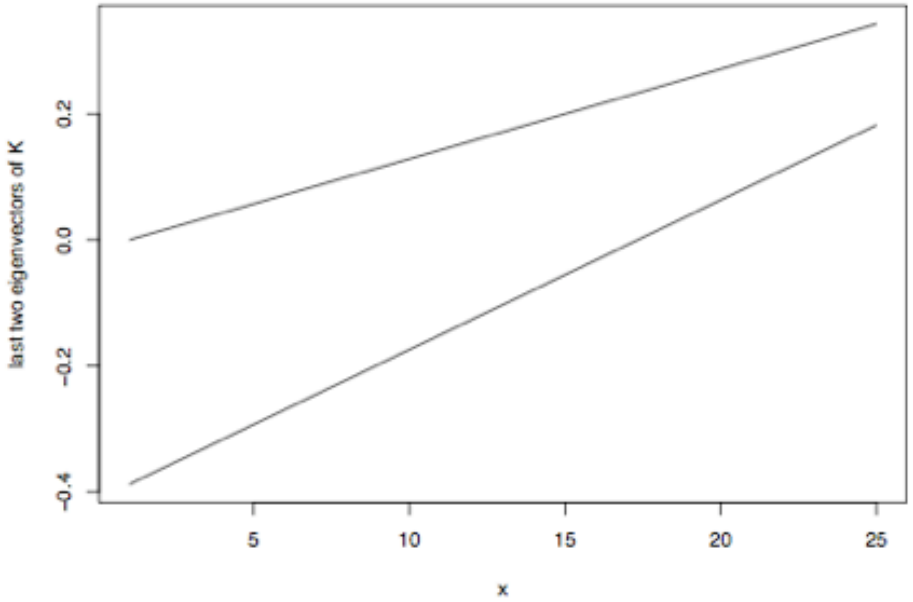
A little more linear algebra

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

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

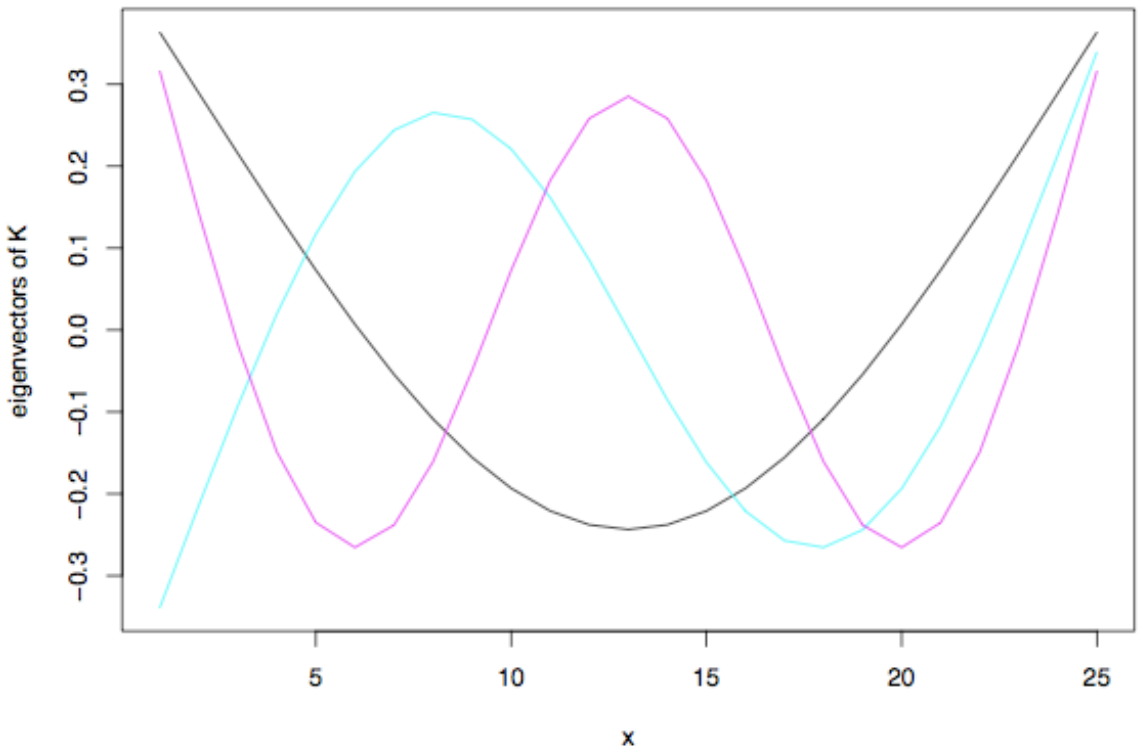
In effect, we are shrinking these coefficients by

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

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?

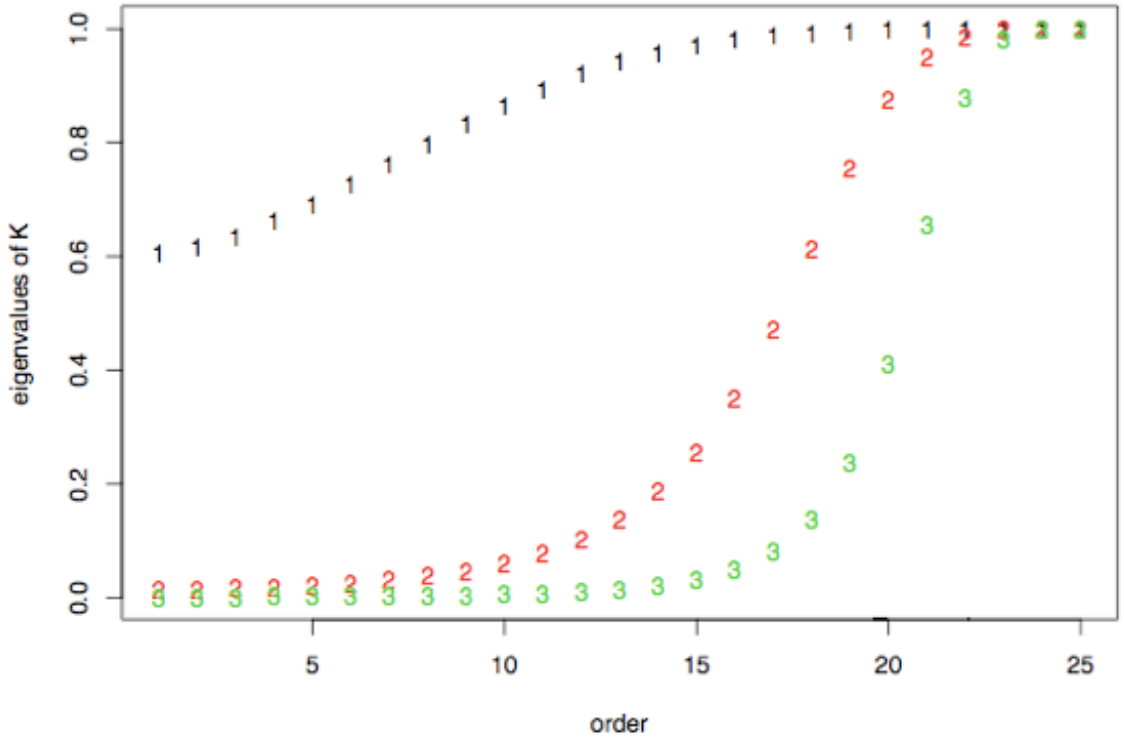


The upshot

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



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

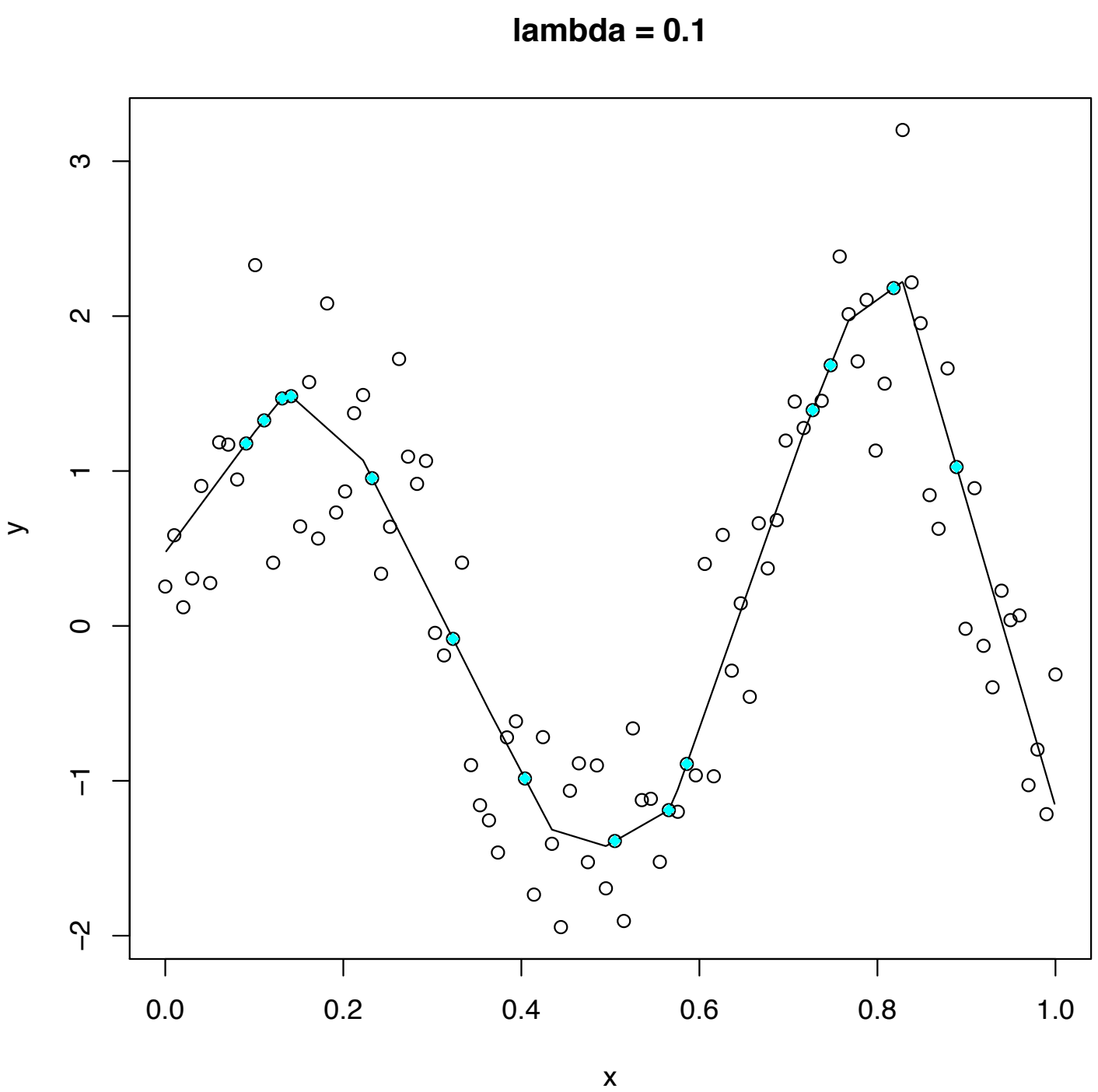
$$V(h) = \int |h'(u)| du$$

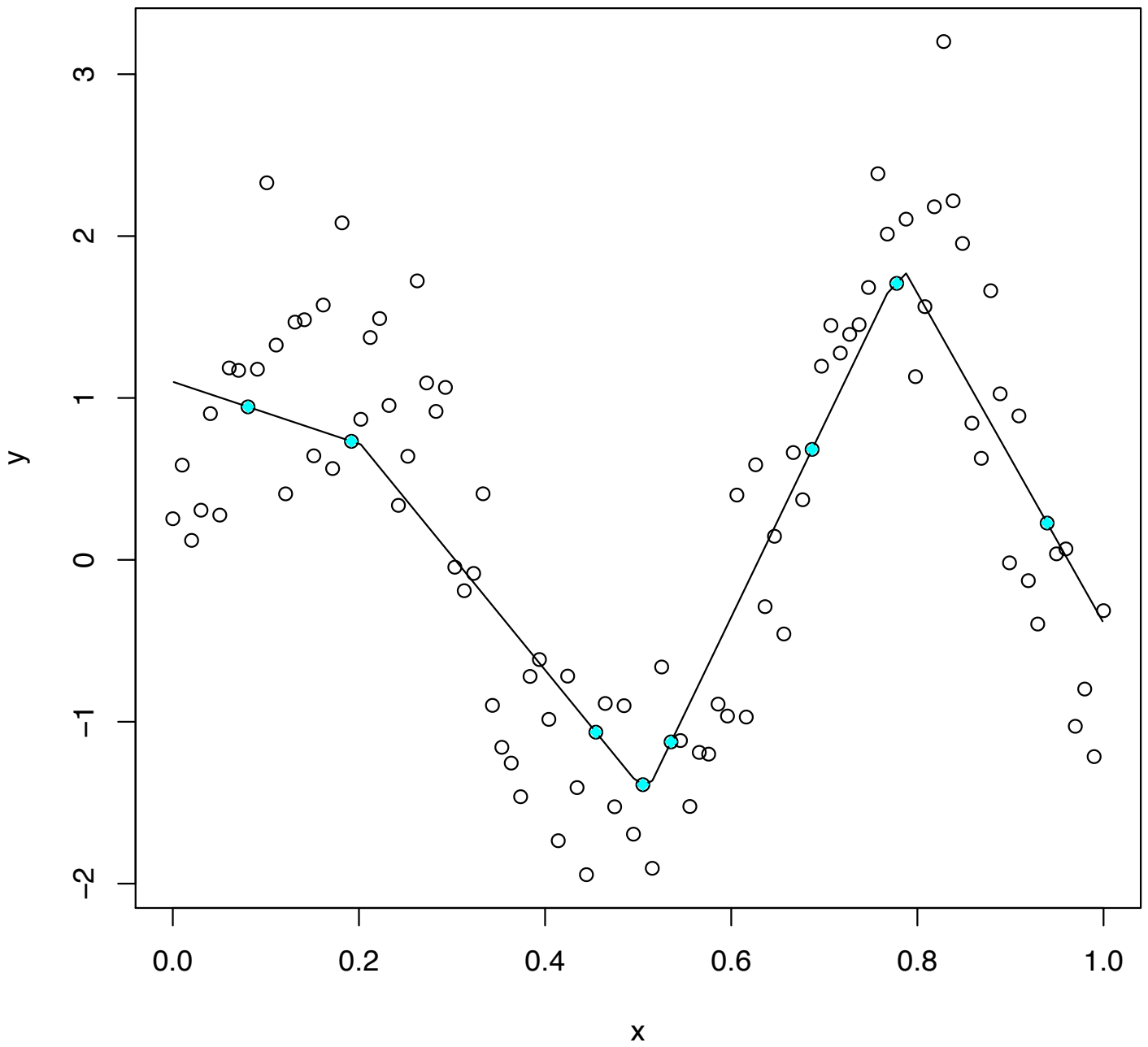
assuming certain smoothness conditions on h

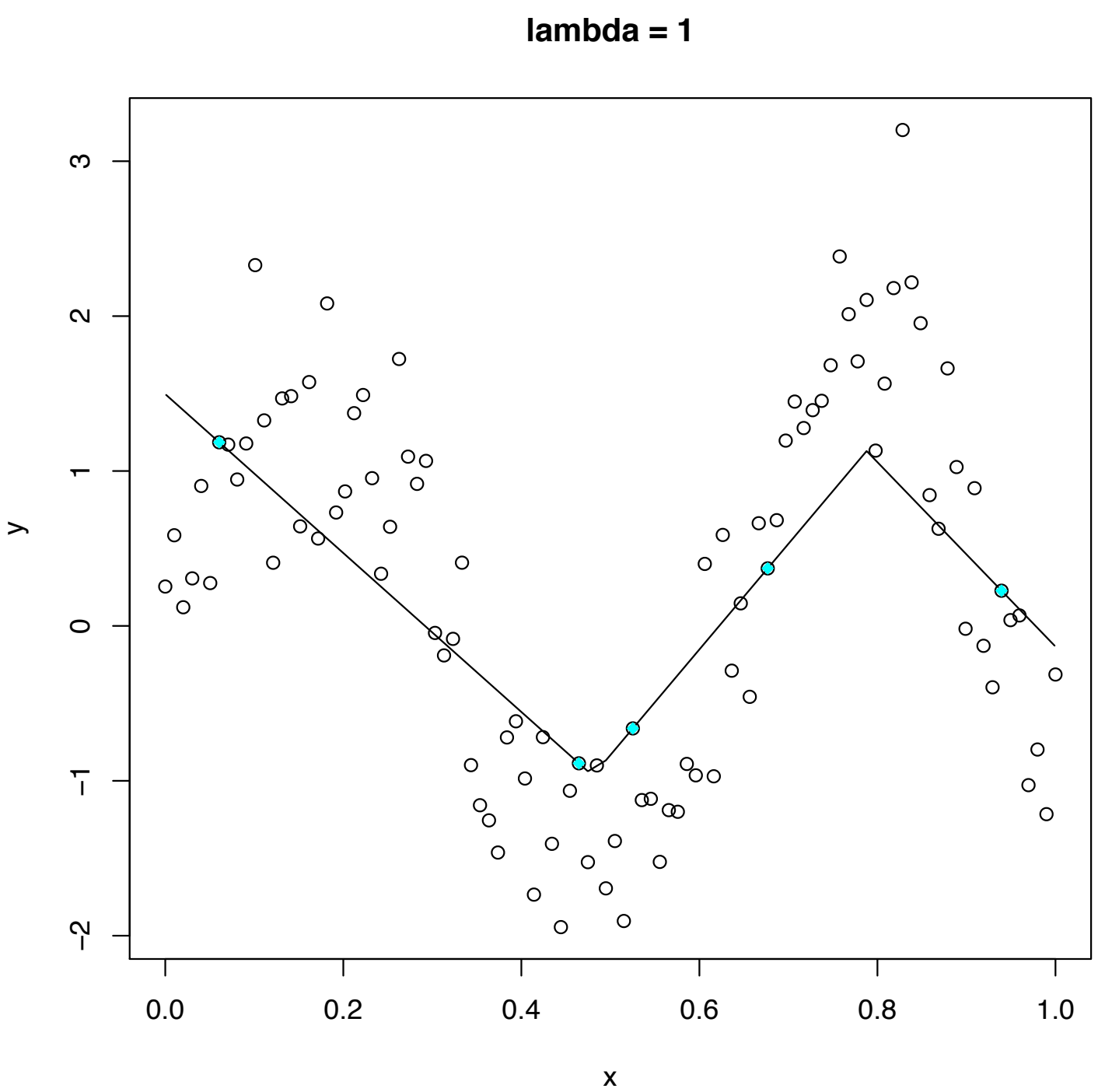
Quantile smoothing splines

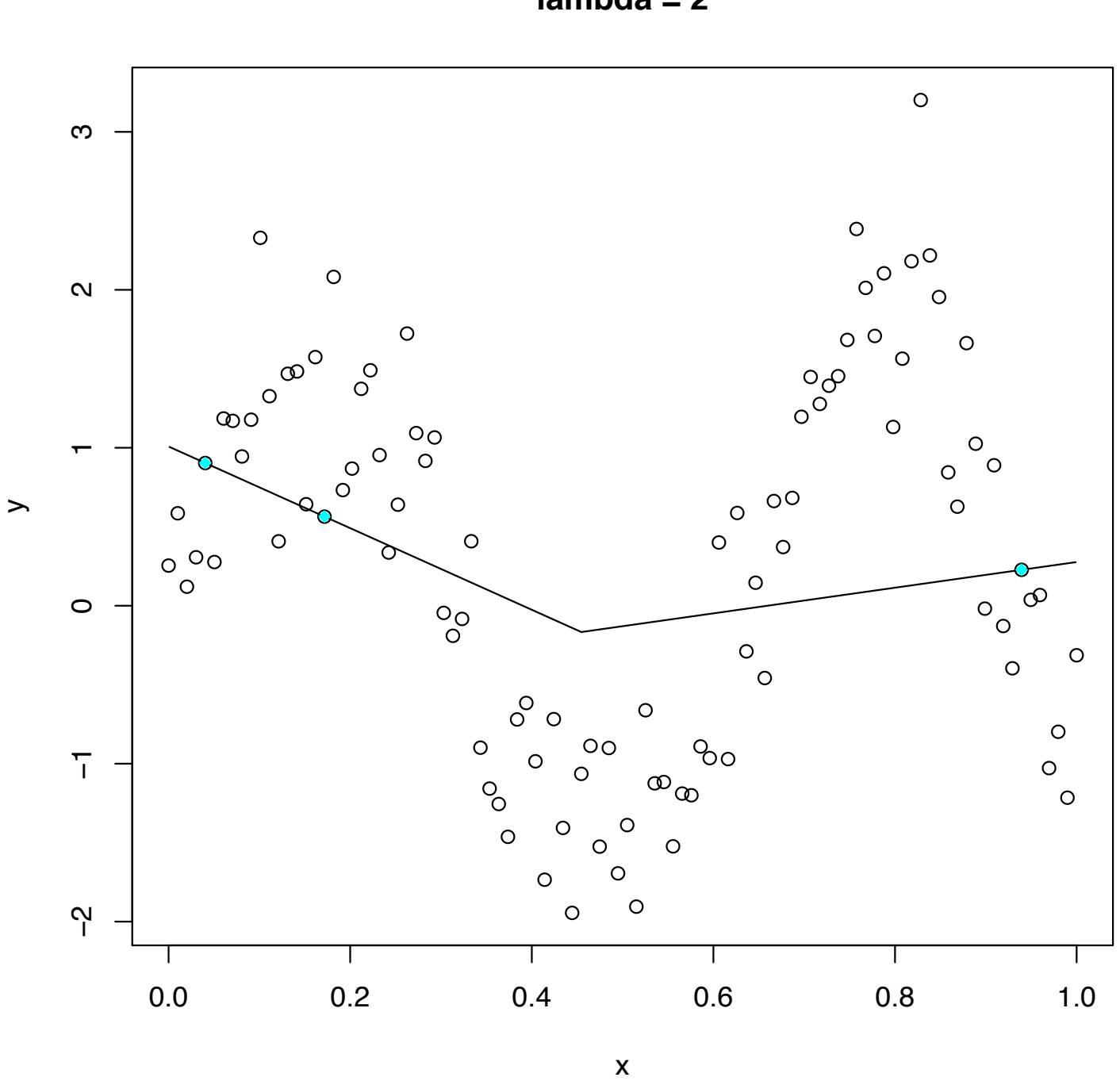
The solution to this problem can be shown to be a linear spline with knots at the input points $x_1, \ldots, x_n$

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