Harvard Department of Government 2003
Faraway Chapter 14, Nonparametric Regression

JEFF GILL

Visiting Professor, Fall 2024

Motivation

- Sometimes we do not a priori have a specific model or parametric assumption in mind.
- Two typical uses: bivariate visualization and modeling.
- Two modeling purposes: general data exploration (a good thing), a commitment to reduce the usual number of distributional assumptions (sometimes a good thing).
- So sometimes these tools are used as a precursor to full model specification in the traditional parametric (especially Bayesian) sense.
- Also, sometimes this will suggest transformations of the data to convenient forms (sometimes referred to as the "non-linear in the parameters" approach).
- Note: nothing is truly "nonparametric," but this term is too ingrained to avoid.

Smoothing, Goals

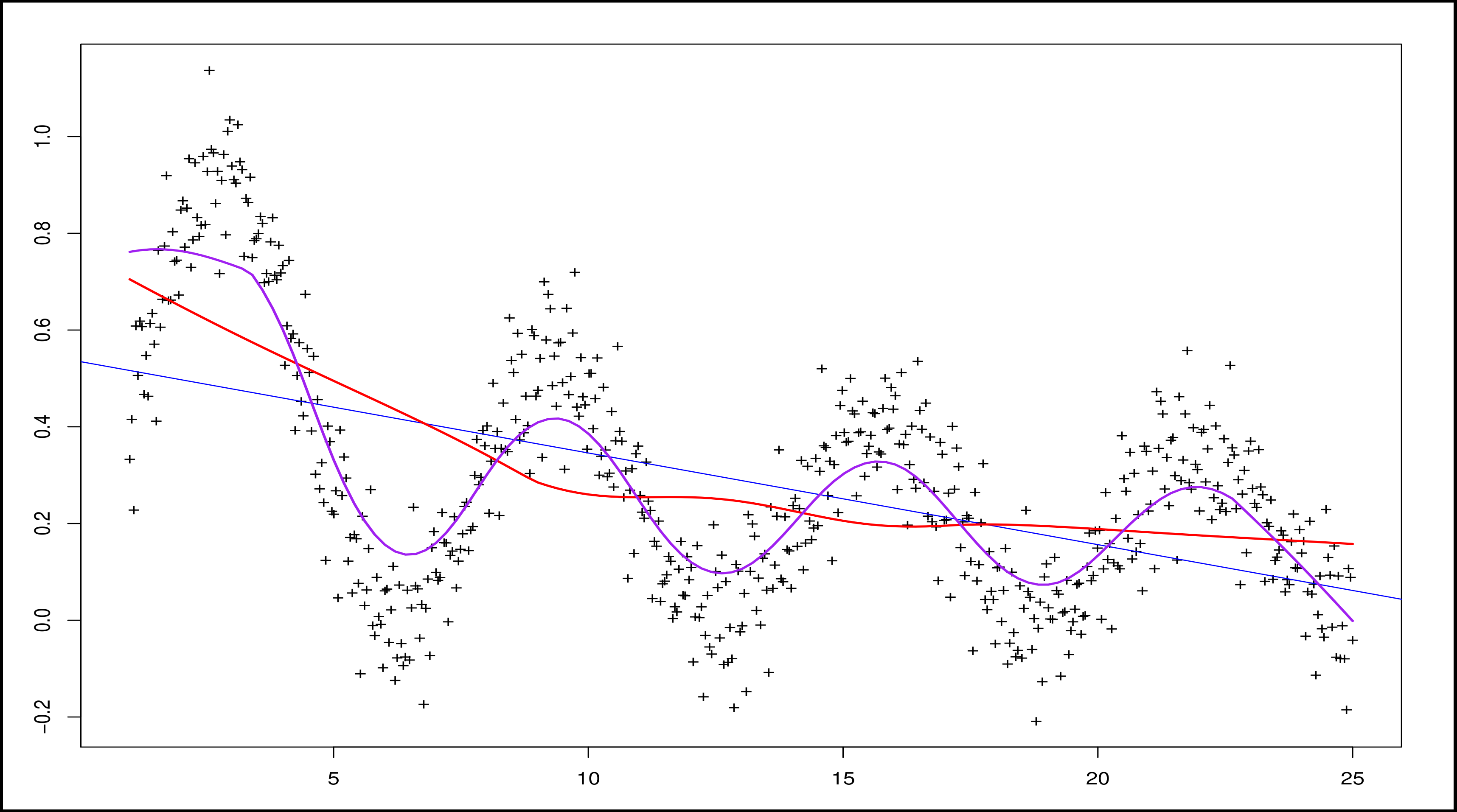
- A tool for summarizing the trend of an outcome variable as a function of explanatory variables (often only one).
- Designed to be less variable than the data itself (hence “smooth”).
- How smooth do we want to be?
- For a nonlinear trend:
  - too much smoothing: variance $\downarrow$, and bias $\uparrow$,
  - too little smoothing: variance $\uparrow$, and bias $\downarrow$,

  where bias in this context means missing curvilinear features.
- Linear regression is then infinitely smooth.
- Pointwise interpolation is then infinitely unsmooth (rough).

Running the Lowess Smoother

```
x <- seq(1,25,length=600)
y <- (2/(pi*x))^(0.5)*(1-cos(x)) + rnorm(100,0,1/10)

par(mar=c(3,3,2,2), bg="white")
plot(x,y,pch="+")

ols.object <- lm(y~x)
abline(ols.object,col="blue")

lo.object <- lowess(y~x,f=2/3)
lines(lo.object$x,lo.object$y,lwd=2,col="red")

lo.object <- lowess(y~x,f=1/5)
lines(lo.object$x,lo.object$y,lwd=2,col="purple")
```

Running the Lowess Smoother



Smoothing, Starting Vocabulary

- We smooth by adjusting data points vertically through weighting to be more “harmonious” with their neighbors.
- The bivariate case is usually called scatterplot smoothing.
- The key smoothing decision is the determination of the size of the neighborhood around each point.
- Larger neighborhoods lead to more smoothness since points further out are included in the weighting.
- We then slide this neighborhood from left to right adjusting the point in the middle.
- The span is defined as the proportion of the total points included in the neighborhood:

  $$\omega = \frac{2K+1}{n}$$

  so there are K points on either side of the point to be smoothed.
- One complication: the ends of the data.

Illustrative Beginning Example

- To test memory retrieval Kail and Nippold ("Unconstrained Retrieval From Semantic Memory", 1984, *Child Development*, 944-951) asked 8, 12, and 21 year olds to name as many animals and pieces of furniture as possible in separate seven minute intervals.

- They find that this number increases across the tested age range but that the rate of retrieval slows down as the period continues.

- In fact, the responses often came in "clusters" of related responses ("lion," "tiger," "cheetah," etc.), where the relation of time in seconds to cluster size is fitted to be

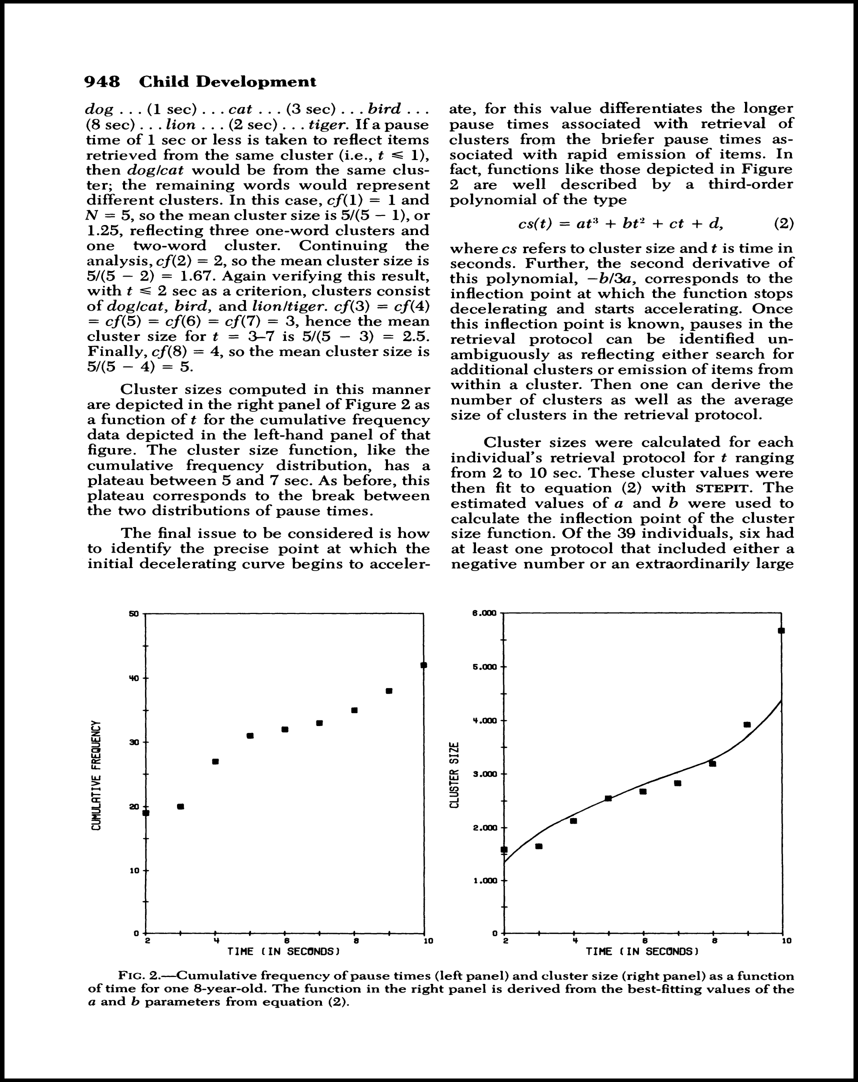
  $$cs(t) = at^3 + bt^2 + ct + d,$$

  where time is t, and the others are estimated parameters (which differ by topic, age group and subject).

- There are strong theoretical reasons that $b = -18a$ from the literature.

- The researchers were very interested in the inflection point of this function since it suggests a change of cognitive process.

Illustrative Beginning Example



dog . . . (1 sec) . . . *cat* . . . (3 sec) . . . *bird* . . . (8 sec) . . . *lion* . . . (2 sec) . . . *tiger*. If a pause time of 1 sec or less is taken to reflect items retrieved from the same cluster (i.e., $t \leq 1$), then *dog/cat* would be from the same cluster; the remaining words would represent different clusters. In this case, $cf(1) = 1$ and $N = 5$, so the mean cluster size is $5/(5 - 1)$, or 1.25, reflecting three one-word clusters and one two-word cluster. Continuing the analysis, $cf(2) = 2$, so the mean cluster size is $5/(5 - 2) = 1.67$. Again verifying this result, with $t \leq 2$ sec as a criterion, clusters consist of *dog/cat*, *bird*, and *lion/tiger*. $cf(3) = cf(4) = cf(5) = cf(6) = cf(7) = 3$, hence the mean cluster size for t = 3–7 is $5/(5 - 3) = 2.5$. Finally, $cf(8) = 4$, so the mean cluster size is $5/(5 - 4) = 5$.

Cluster sizes computed in this manner are depicted in the right panel of Figure 2 as a function of t for the cumulative frequency data depicted in the left-hand panel of that figure. The cluster size function, like the cumulative frequency distribution, has a plateau between 5 and 7 sec. As before, this plateau corresponds to the break between the two distributions of pause times.

The final issue to be considered is how to identify the precise point at which the initial decelerating curve begins to accelerate, for this value differentiates the longer pause times associated with retrieval of clusters from the briefer pause times associated with rapid emission of items. In fact, functions like those depicted in Figure 2 are well described by a third-order polynomial of the type

$$cs(t) = at^3 + bt^2 + ct + d, \qquad (2)$$

where cs refers to cluster size and t is time in seconds. Further, the second derivative of this polynomial, $-b/3a$, corresponds to the inflection point at which the function stops decelerating and starts accelerating. Once this inflection point is known, pauses in the retrieval protocol can be identified unambiguously as reflecting either search for additional clusters or emission of items from within a cluster. Then one can derive the number of clusters as well as the average size of clusters in the retrieval protocol.

Cluster sizes were calculated for each individual's retrieval protocol for t ranging from 2 to 10 sec. These cluster values were then fit to equation (2) with STEPIT. The estimated values of a and b were used to calculate the inflection point of the cluster size function. Of the 39 individuals, six had at least one protocol that included either a negative number or an extraordinarily large

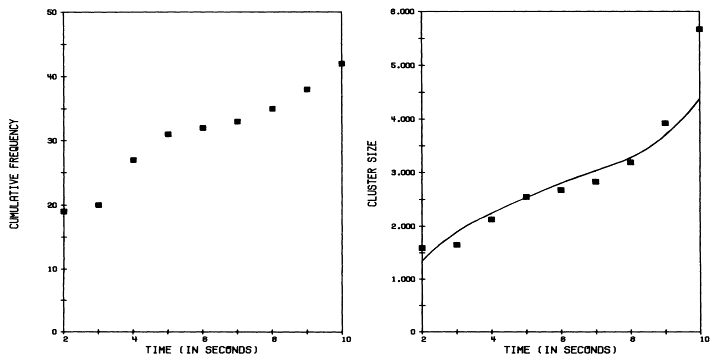


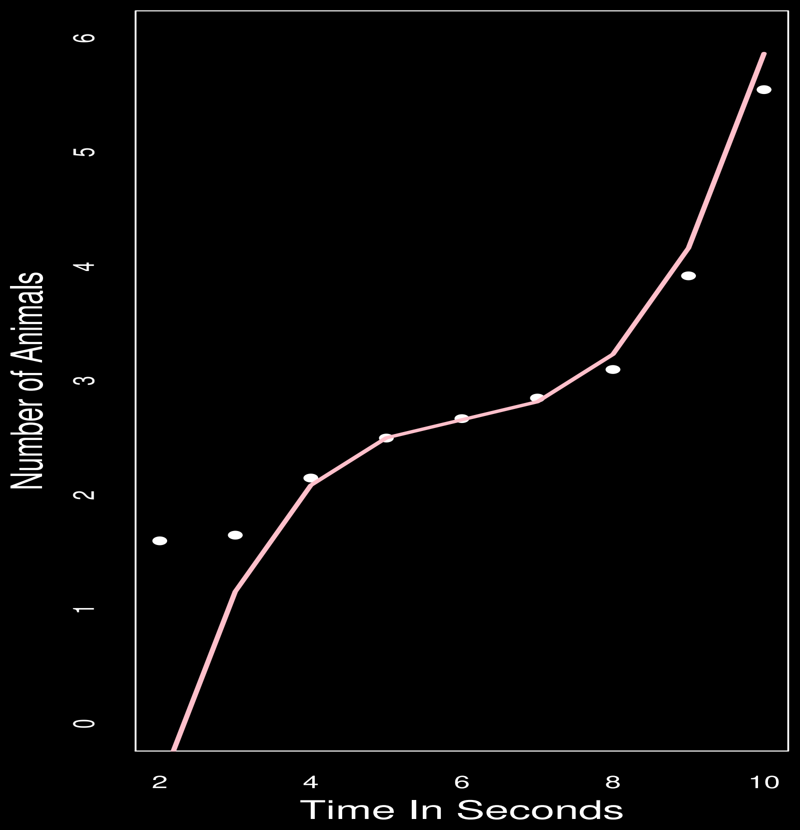
FIG. 2.—Cumulative frequency of pause times (left panel) and cluster size (right panel) as a function of time for one 8-year-old. The function in the right panel is derived from the best-fitting values of the a and b parameters from equation (2).

Illustrative Beginning Example

- We can specify hard-coded values of the parameters (below) by trial and error.

```
cs <- c(1.6,1.65,2.15,2.5,2.67,2.85,3.1,3.92,5.55)
seconds <- 2:10

cog <- function(a,c,d,t) a*t^3 + (-18*a)*t^2 + c*t + d

postscript("Class.Stat.Comp/cognitive2a.ps")
par(mfrow=c(1,1),mar=c(5,5,3,3),oma=c(6,6,6,6),col.axis="white",col.lab="white",
    col.sub="white",col="white",bg="black")
plot(seconds,cs,pch=19,ylim=c(0,6),xlab="",ylab="")
cs.vals <- cog(a=0.04291667,c=4.75,d=-7.3,t=seconds)
# try a=0.0405,c=5.03,d=-6.36
lines(seconds,cs.vals,col="pink",lwd=3)
mtext(side=1,line=2.5,cex=1.5,"Time In Seconds")
mtext(side=2,line=2.5,cex=1.5,"Number of Animals")
dev.off()
```

Nonlinear (Weighted) Least-Squares



Illustrative Beginning Example

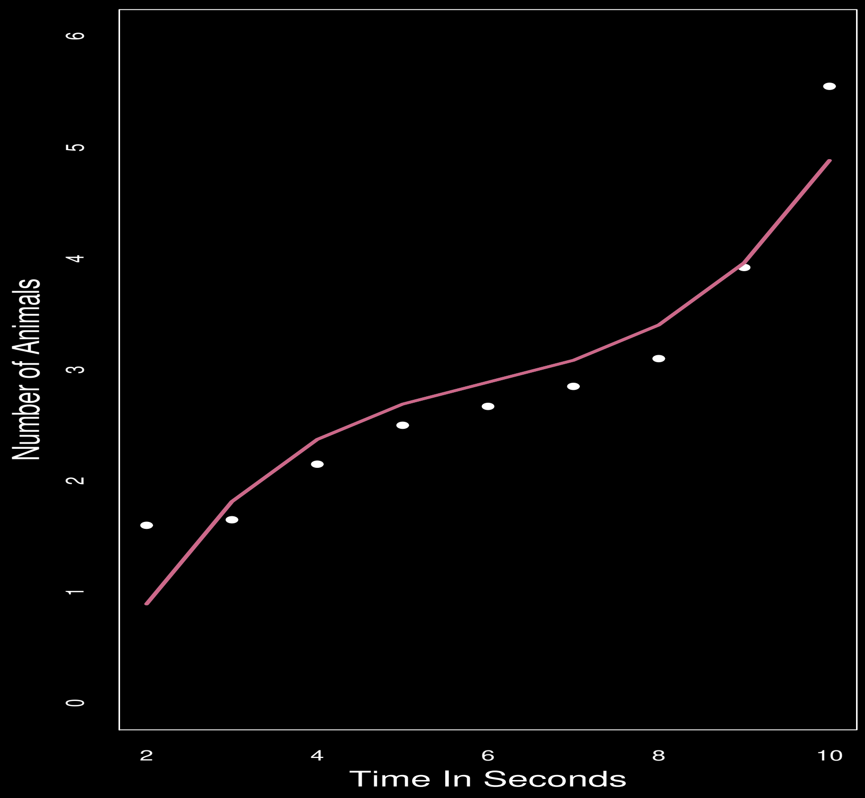
- We can also use the `R` function `nls` to estimate these by minimizing residuals:

```
cog.df <- data.frame(seconds=seconds,cs=cs)
cog.nls <- nls(cs ~ a*seconds^3 + (-18*a)*seconds^2 + c*seconds + d,
               start=c(a=10,c=10,d=-10),trace=TRUE); summary(cog.nls)

  Estimate Std. Error t value Pr(>|t|)
a  0.02013    0.01211   1.662   0.1476
c  2.35048    1.16637   2.015   0.0905
d -2.52056    1.79998  -1.400   0.2109
Residual standard error: 0.4572 on 6 degrees of freedom
```

Illustrative Beginning Example

```
postscript("Class.Stat.Comp/cognitive2b.ps")
par(mfrow=c(1,1),mar=c(4,4,4,4),oma=c(3,3,3,3),col.axis="white",col.lab="white",
    col.sub="white",col="white",bg="black")
plot(seconds,cs,pch=19,ylim=c(0,6),xlab="",ylab="")
lines(seconds,cs.vals,col="lightsteelblue4",lwd=3)
mtext(side=1,line=2.5,cex=1.5,"Time In Seconds")
mtext(side=2,line=2.5,cex=1.5,"Number of Animals")
cs.vals <- cog(a=summary(cog.nls)$parameters[1,1],
               c=summary(cog.nls)$parameters[2,1],
               d=summary(cog.nls)$parameters[3,1],
               t=seconds)
lines(seconds,cs.vals,col="palevioletred3",lwd=3)
dev.off()
```

Illustrative Beginning Example



General Expression For Smoothers

- Now consider the general model:
$$y_i = f(x_i) + \epsilon_i$$
where $f()$ is an unspecified (for now) smooth, nonlinear function, and $\epsilon \sim \mathcal{N}(0, \sigma^2)$.

- One choice of the function: **Scatterplot Smoother:**
$$\hat{f}(x_i) = \sum_{j=1}^{n} s_{ij} y_j$$
$$s_{ij} = s(x_i, x_j), \text{ some weighting function},$$
$$x_i, \text{ point to be smoothed (moved)},$$
$$x_j, \text{ all other points: } 1, \dots, n$$
$$y_j, \text{all outcome variable values: } 1, \dots n$$

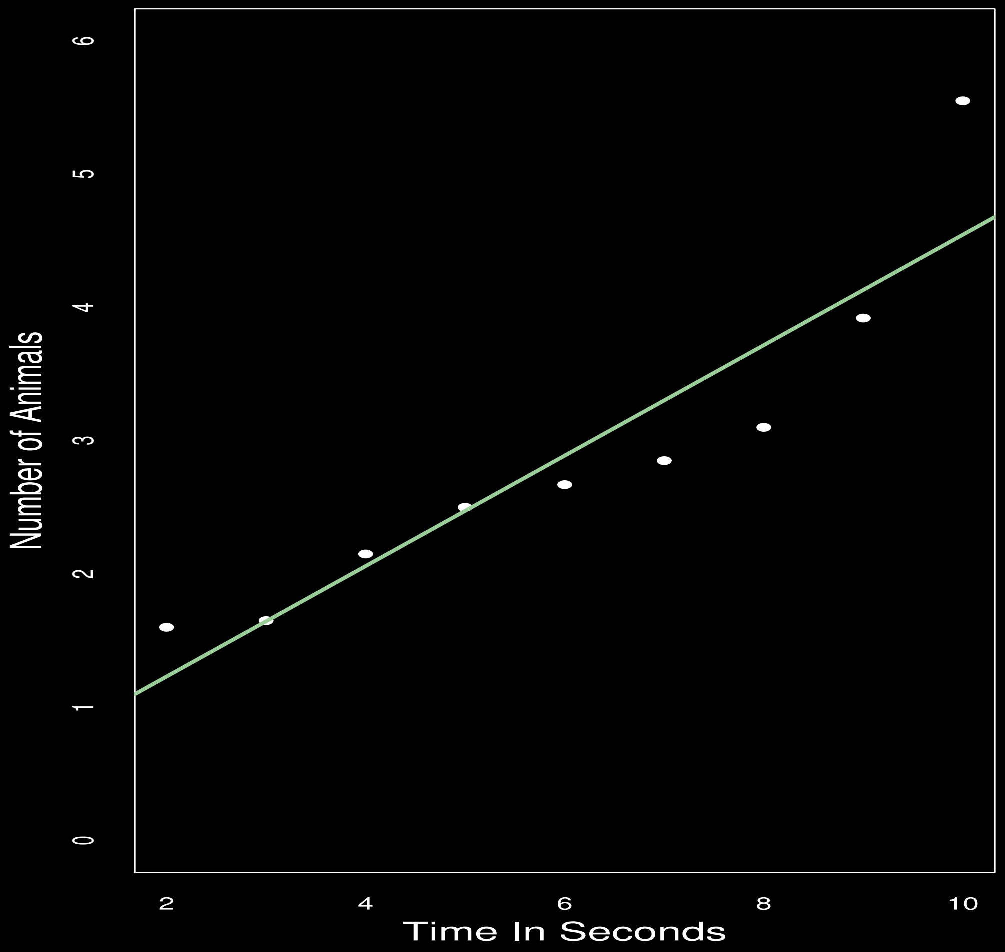
- The key decision (as we'll see) is the choice of s_{ij} through neighborhood treatment: large neighborhoods or diffuse functions produce less variable and more smooth fits with greater bias, and small neighborhoods or narrow functions produce more variable and less smooth fits with less bias.

Linear Regression as a Smoother

- Smoother #1: linear regression.
  - $\hat{f}(x_i) = \alpha + x_i\beta$.
  - Called "infinitely smooth" since the second derivative is zero everywhere on the line.

```
postscript("Class.Stat.Comp/cognitive2c.ps")
par(mfrow=c(1,1),mar=c(5,5,3,3),oma=c(3,3,3,3),col.axis="white",col.lab="white",
    col.sub="white",col="white",bg="black")
plot(seconds,cs,pch=19,ylim=c(0,6),xlab="",ylab="")
abline(lm(cs.vals~seconds),col="darkseagreen3",lwd=3)
mtext(side=1,line=2.5,cex=1.5,"Time In Seconds")
mtext(side=2,line=2.5,cex=1.5,"Number of Animals")
dev.off()
```

Linear Regression as a Smoother

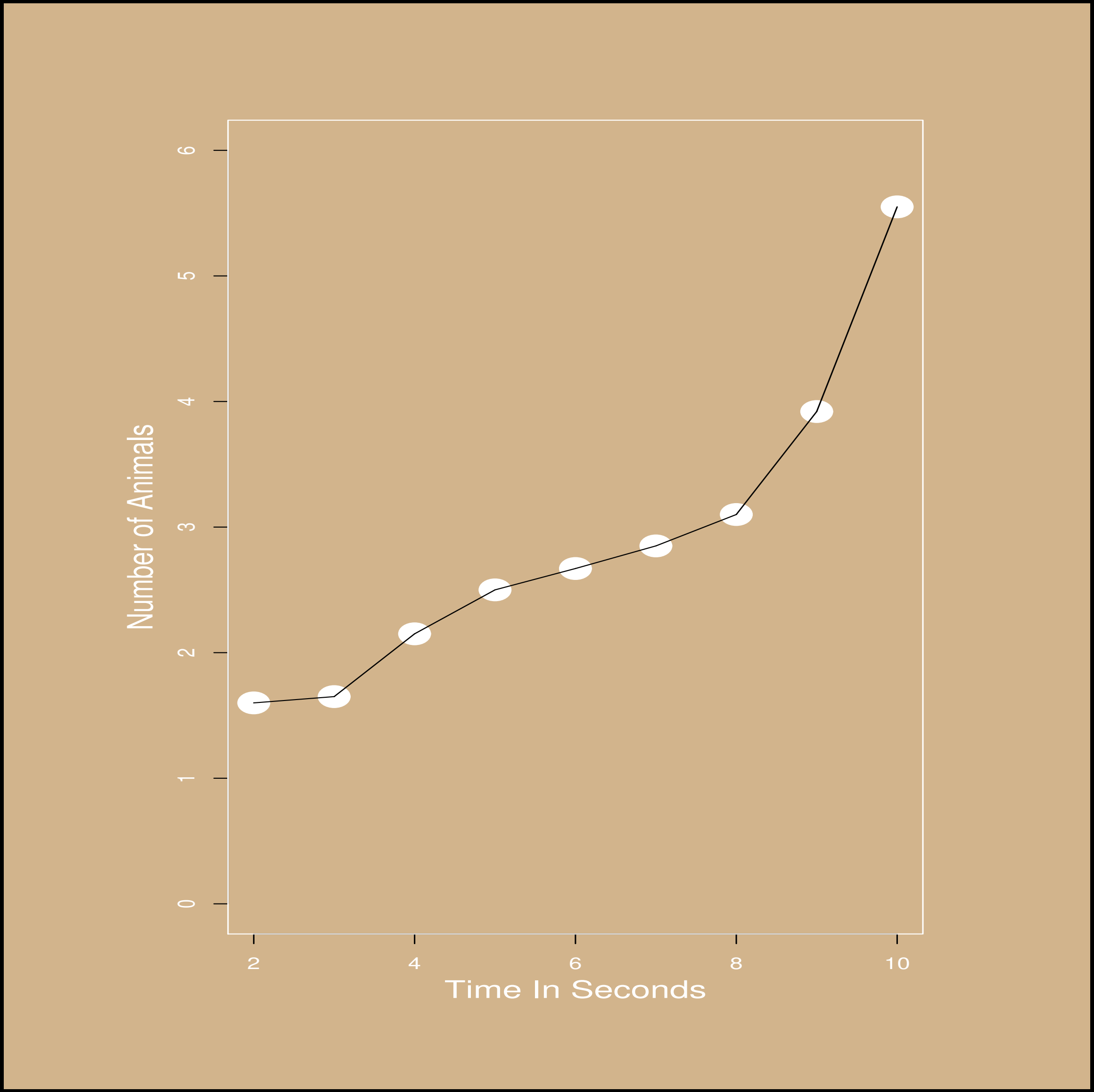


Interpolation as a Smoother

- Smoother #2: interpolation.
  - $\hat{f}(x_i) = x_i$.
  - Called "infinitely rough" since it just connects the points.

```
postscript("Class.Stat.Comp/cognitive2d.ps")
par(mfrow=c(1,1),mar=c(5,5,3,3),oma=c(3,3,3,3),col.axis="white",col.lab="black",
    col.sub="white",col="white",bg="tan")
plot(seconds,cs,pch=19,ylim=c(0,6),xlab="",ylab="",cex=3)
for(i in 1:(length(seconds)-1))
    segments(seconds[i],cs[i],seconds[i+1],cs[i+1],cex=2)
mtext(side=1,line=2.5,cex=1.5,"Time In Seconds")
mtext(side=2,line=2.5,cex=1.5,"Number of Animals")
dev.off()
```

Interpolation as a Smoother



Bin Smoother

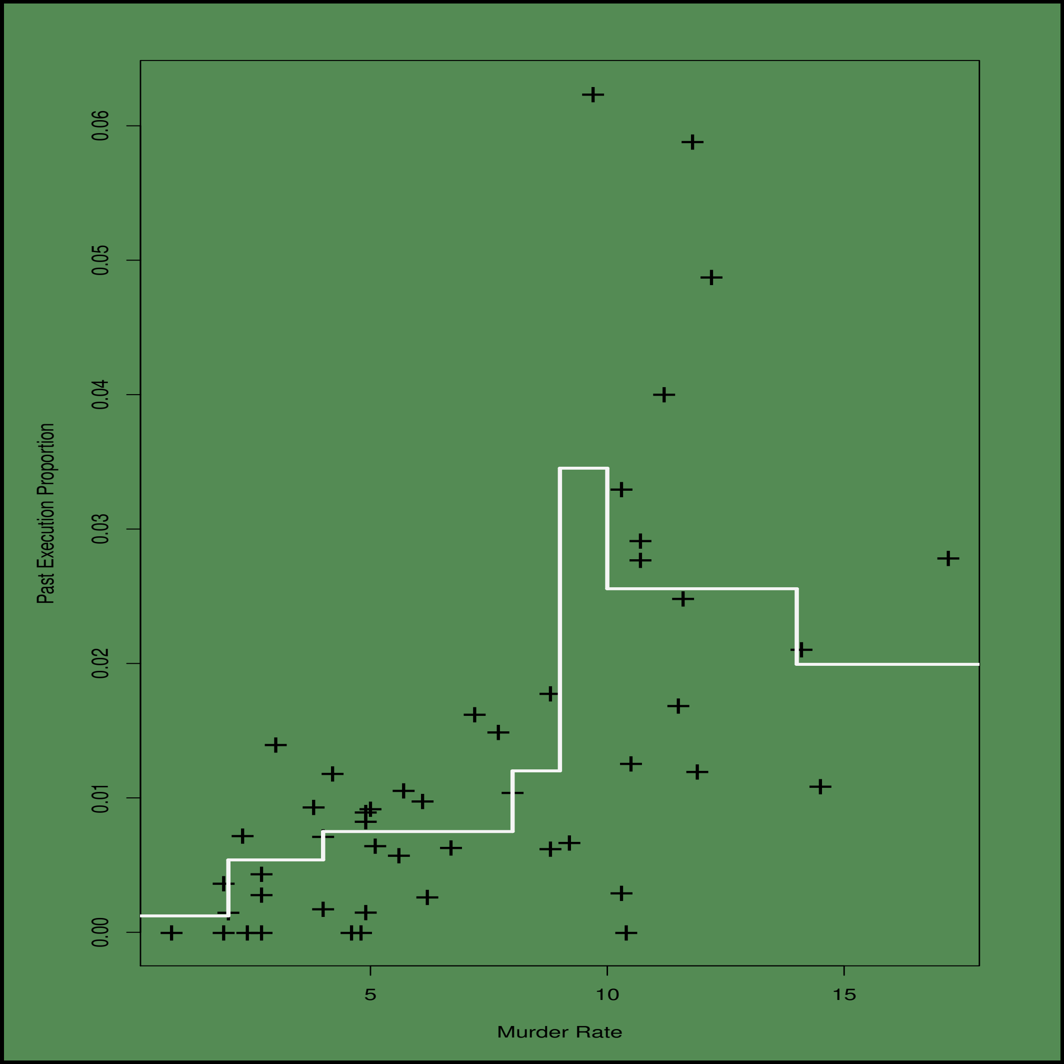
- Smoother #3: bin smoother (regressogram). Looks very "notchy" like a city skyline. Steps:
  1. Choose $J+1$ thresholds (cutpoints) along the x-axis:
  $$-\infty < c_0 < c_1 < \ldots < c_{J-1} < c_J < \infty$$
  2. Now "bin" the data by collecting values between the thresholds. The j th bin contains points such that:
  $$R_j = \{i : c_j \leqslant x_i < c_{j+1}\}$$
  (i.e. collect the points in R_j that are greater than c_j and less than c_{j+1}).
  3. Within each bin calculate an average AVE (mean, median, or mode), and assign it to each of the bin values:
  $$s_{ij} = AVE(y_i | i \in R_j)$$
- Window widths can be equal (default) or customized.
- Two issues: it looks "choppy" and sometimes picking a good value for J.

Bin Smoother

- Code:

```
postscript("Class.Stat.Comp/cognitive2e.ps")
par(mfrow=c(1,1),mar=c(5,5,3,3),bg="palegreen4")
x1 <- seq(0.25,10,length=40)
y1 <- tan(x1)/x1^2 + rt(length(x1),10)
plot(x1,y1,pch="+",lwd=3,xlab="",ylab="",xlim=c(0,10))

x1.cuts <- c(0,2,4,6,8,10)
y1.means <- rep(NA,(length(x1.cuts)-1))
for (i in 1:length(y1.means))
    y1.means[i] <- mean(y1[x1 > x1.cuts[i] & x1 < x1.cuts[i+1]])

for (i in 1:length(y1.means))  {
    segments(x1.cuts[i],y1.means[i],x1.cuts[i+1],y1.means[i],lwd=2)
    if (i != 1) segments(x1.cuts[i],y1.means[i-1],x1.cuts[i],y1.means[i],lwd=2)
}
dev.off()
```

Bin Smoother



Running Mean Smoothers

- General Idea: for each point in the data, pick k points to the left and k points to the right on the x-axis and take the mean on the y-axis of these points.
- This defines a neighborhood for each point.
- Small values of k produce rough plots and large values of k produce smooth plots.
- Also called the "symmetric nearest neighborhood" smoother or "simple neighborhood" smoother.

Running Mean Smoothers

- Steps:
  - ▻ $N(x_i)$ is the neighborhood of point i determined by w, which is the window size/span defined as the proportion of other points included. Note that $0 < w < 1$ is constant.
  - ▻ More formally, assume n points (odd number for notational convenience), and define:

$$N(x_i) = \left[\max\left(\frac{\lfloor wn \rfloor - 1}{2}, 1 \right), \ldots, i-1, i, i+1, \ldots, \min\left(\frac{\lfloor wn \rfloor - 1}{2}, n \right) \right]$$

    where $\lfloor wn \rfloor$ is the "floor" of wn, i.e. the lower integer component ($\lfloor 3.14 \rfloor = 3$).
  - ▻ So $N(x_i)$ gives indices of x's to include that denote x_i and $\frac{\lfloor wn \rfloor - 1}{2}$ points on either side (smaller near the endpoints though).
  - ▻ Now apply the smoother of choice, $\hat{f}(x_i)$ inside each neighborhood to get values for each x_i.

Running Mean Smoothers

- Running-mean type:

$$\hat{f}(x_i) = AVE(y_j|j \in N(x_i)) = AVE_{j\in N(x_i)}(y_i)$$

- Running-line (OLS fit of points in $N(x_i)$) type:

$$\hat{f}(x_i) = \hat{\alpha} + \hat{\beta}x_i$$

Running Mean Smoothers

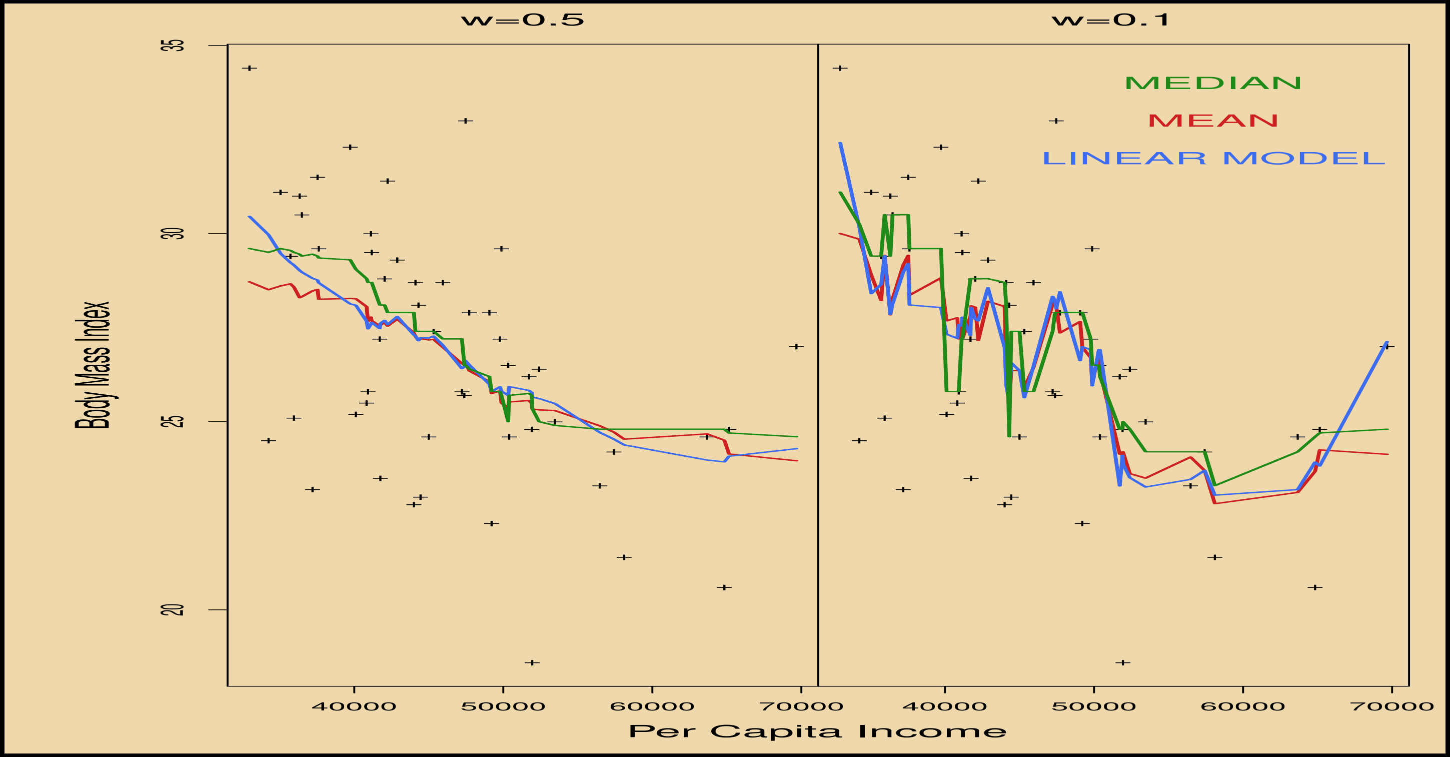
- R function is `smooth`, but we can write our own:

```
x0 <- seq(1,25,length=101); n <- length(x0)
y0 <- exp(2/(pi*x0))^(1.5) + rnorm(n,0,1/10)
f.rm <- f.rl <- f.rd <- rep(NA,length=n)
w <- 0.3

for (i in 1:n)  {
    lower <- max(i-floor((w*n-1)/2),1);  upper <- min(i+floor((w*n-1)/2),n)
    neighborhood <- lower:upper
    f.rm[i] <- mean(y0[neighborhood])
    f.rd[i] <- median(y0[neighborhood])
    lin.fit <- lm(y0[neighborhood] ~ x0[neighborhood])
    f.rl[i] <- lin.fit$coefficients %*% c(1,x0[i])
}
```

Running Mean Smoothers

```
# postscript("Class.Stat.Comp/cognitive2f.ps")
par(mfrow=c(1,1),mar=c(5,5,5,5),bg="wheat2")
plot(x0,y0,pch="+",lwd=1,xlab="",ylab="")
lines(x0,f.rm,col="firebrick3")
lines(x0,f.rl,col="royalblue2")
lines(x0,f.rd,col="forest green")
mtext(side=3,cex=1.3,line=2,"Comparison of Neighborhood Smoothers")
# dev.off()
```

Running Mean Smoothers



Running Line Smoother From R

- The R `smooth` function is actually Tukey's running median smoother, which is not so useful.
- Bratton and Van De Walle (1997) make available data on regime transition for 47 sub-Saharan countries over the period from each country's colonial independence to 1989, with some additional variables collected for the period 1990 to 1994. Included are 99 variables describing governmental, economic, and social conditions for the 47 cases. Also provided are data from 106 presidential and 185 parliamentary elections, including information about parties, turnout, and political openness.
- Focus on two variables: economic debt and growth.
- Let's look at Friedman's "super smoother" instead.

```
supsmu(x, y, wt, span = "cv", periodic = FALSE, bass = 0)
```

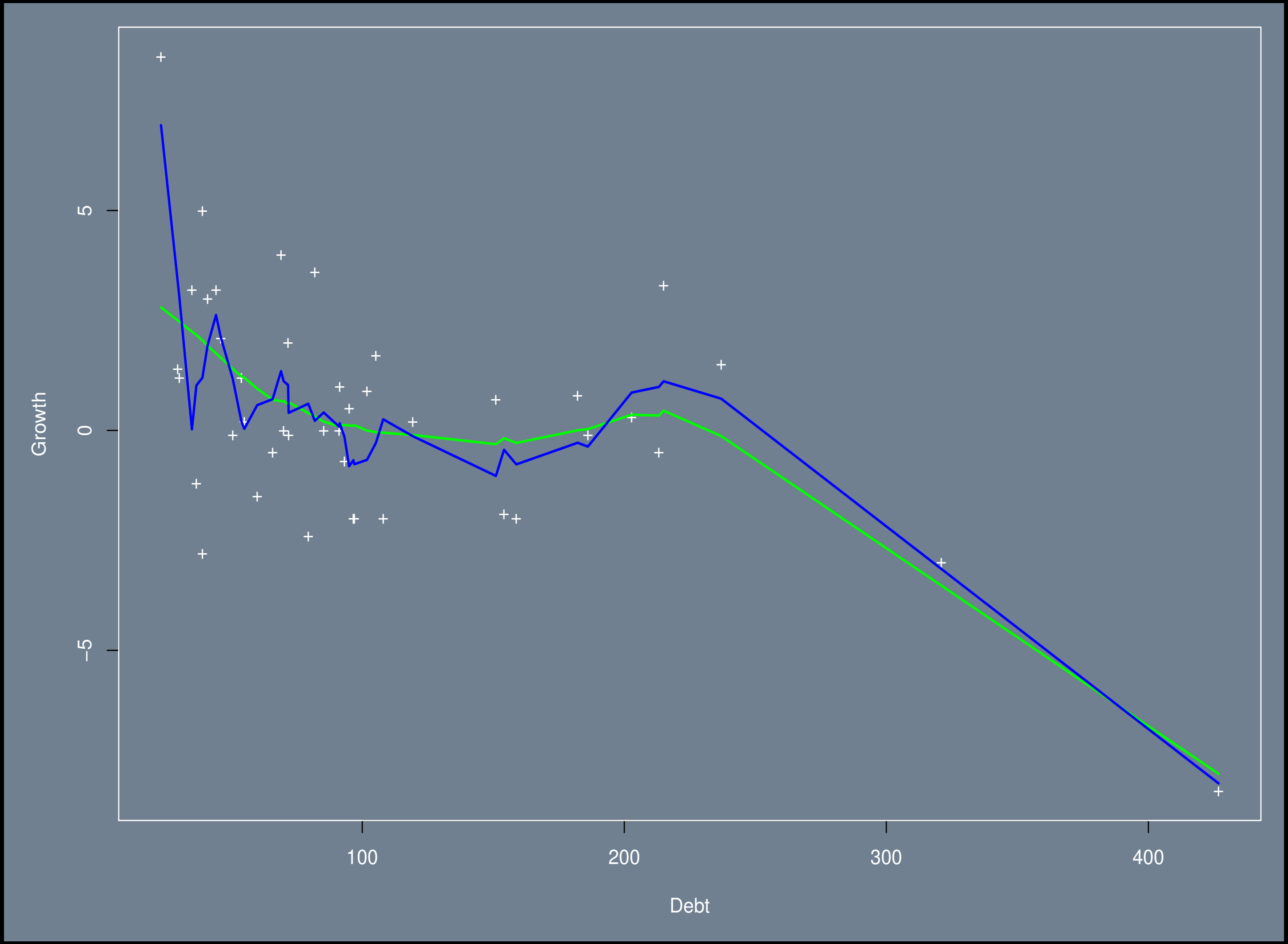
where:

`wt`	case weights, equal by default
`span`	the fraction of observations in the span of the running lines smoother, use "cv" to choose this by cross-validation (default).
`periodic`	if TRUE, x values assumed to be in [0, 1] and of period 1.
`bass`	smoothness control: values of up to 10 indicate increasing smoothness.

Running Line Smoother From R

```
africa <- read.table("http://jgill.wustl.edu/data/africa.dat",
    row.names=1,header=TRUE)
africa2 <- cbind(africa$DEBT,africa$GROWTH)[!is.na(apply(africa2,1,sum)),]
postscript("Class.Stat.Comp/africa.smooth.ps")
par(mar=c(5,5,1,1),col.axis="white",col.lab="white",col.sub="white",
    col="white",bg="slategray")
plot(africa2,pch="+",xlab="Debt",ylab="Growth")
super <- supsmu(africa2[,1],africa2[,2])
lines(super$x,super$y,col="green",lwd=2)
super <- supsmu(africa2[,1],africa2[,2],span=0.01,bass=2)
lines(super$x,super$y,col="blue",lwd=2)
dev.off()
```

Running Line Smoother From R



Kernel Smoothers

- An extension of the running-line smoother where explicit weights are now included.
- Standard idea: weight the points closer to x_i more than remote points.
- Again, pick k points to the left and k points to the right of the x_i point, producing a neighborhood size of $2k+1$.
- Define the jth weight for the ith point as the function:

$$\omega_{ij} = cd\left(\left|\frac{x_i - x_j}{\lambda}\right|\right) \qquad \text{for } j \in N_{2k+1},\ 0 \text{ otherwise}$$

  where:

  - ▷ c is a normalizing constant such that $\int s(u)du = 1$:

$$c = \left[\sum_{i=1}^{2k+1} d\left(\left|\frac{x_i - x_j}{\lambda}\right|\right)\right]^{-1}$$

  - ▷ λ is the window width: $2k+1$,
  - ▷ and $d(t)$ is a decreasing function in $t = |x_i - x_j|/\lambda$.

Kernel Smoothers

- So

$$\hat{y}_i = s(y_i|\mathbf{X}) = \sum_{j=1}^{2k+1} \omega_{ij} y_i.$$

- Common forms:

$$d(t) = \phi(t) \qquad \text{[Gaussian]}$$

$$d(t) = \begin{cases} \frac{3}{4}(1-t^2), & |t| \leqslant 1 \\ 0, & \text{otherwise} \end{cases} \qquad \text{[Epanechnikov]}$$

$$d(t) = \begin{cases} \frac{3}{8}(3-5t^2), & |t| \leqslant 1 \\ 0, & \text{otherwise} \end{cases} \qquad \text{[minimum variance]}$$

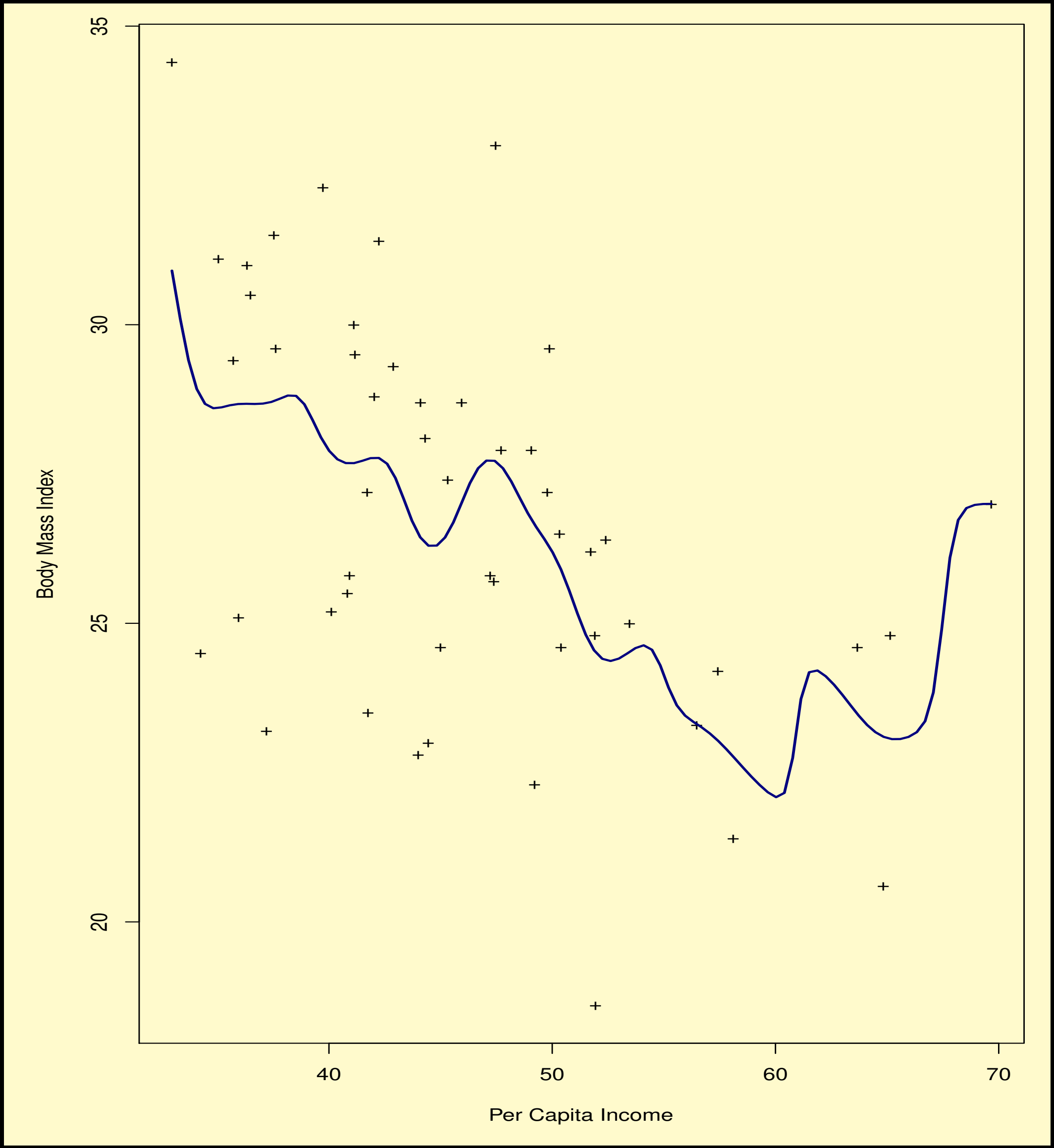
- Others: box, triangle, Parzen, miscellaneous polynomials.

Kernel Smoothers

- Using the R function `ksmooth`:

```
postscript("Class.Stat.Comp/cognitive2g.ps")
par(mfrow=c(1,1),mar=c(5,5,5,5),bg="lemonchiffon")
plot(x0,y0,pch="+",lwd=1,xlab="",ylab="")
lines(ksmooth(x0,y0,kern="normal",bandwidth=2),col="navy")
mtext(side=3,cex=1.3,line=2,"Gaussian Kernel Smoother")
dev.off()
```

Kernel Smoothers



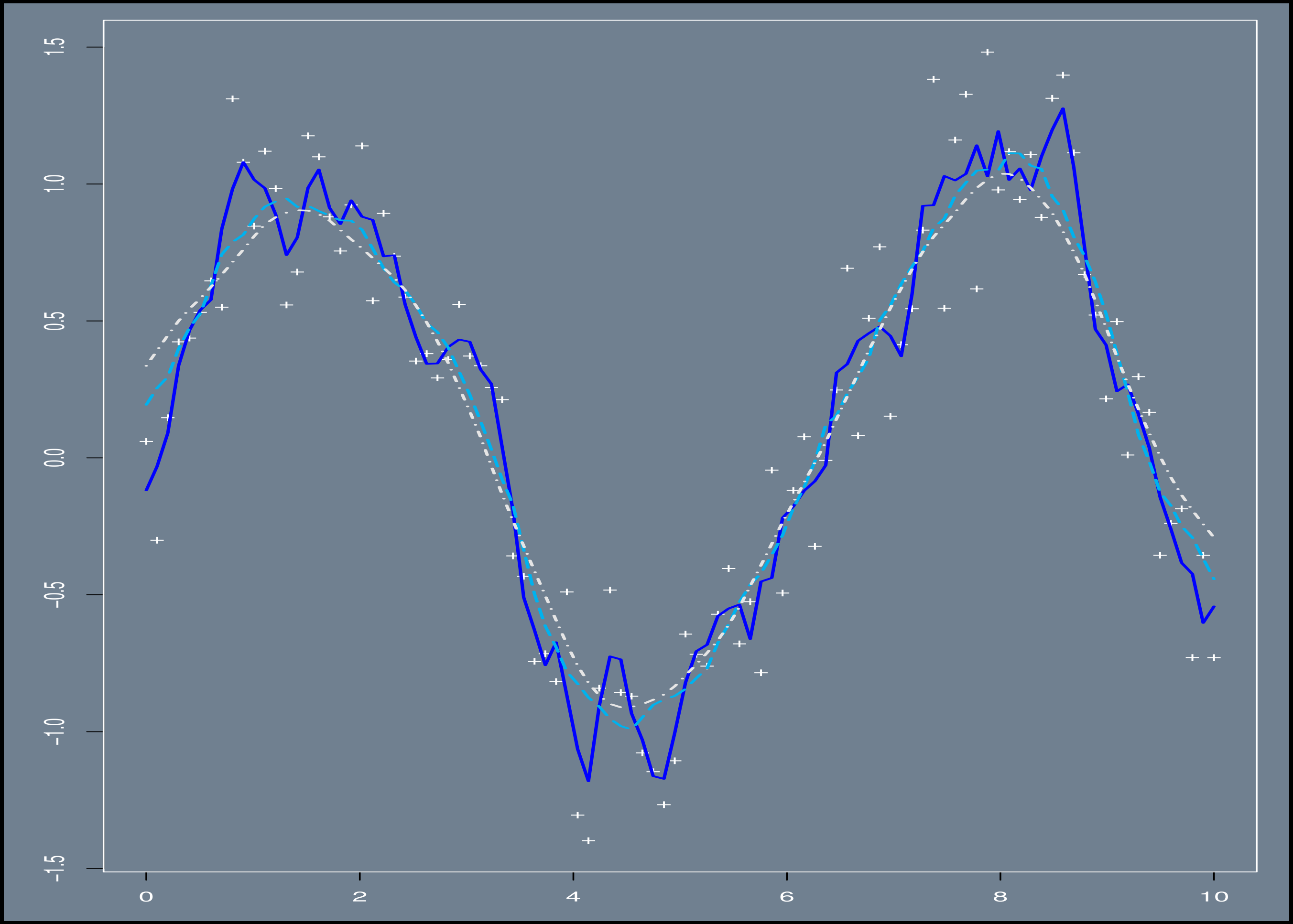
Kernel Smoothers, Rolling Our Own

```
epan <- function(x) ifelse(abs(x) <= 1, 0.75*(1-x^2), 0)

k.sm <- function(x,y,k)  {
    s.y <- y
    for (i in 1:length(x))  {
        lo <- ifelse(i-k >= 1, i-k, 1)
        hi <- ifelse(i+k <= length(x), i+k, length(x))
        w <- epan(x[i] - x[lo:hi])/sum(epan(x[i] - x[lo:hi]))
        s.y[i] <- y[lo:hi] %*% w
    }
    s.y
}
```

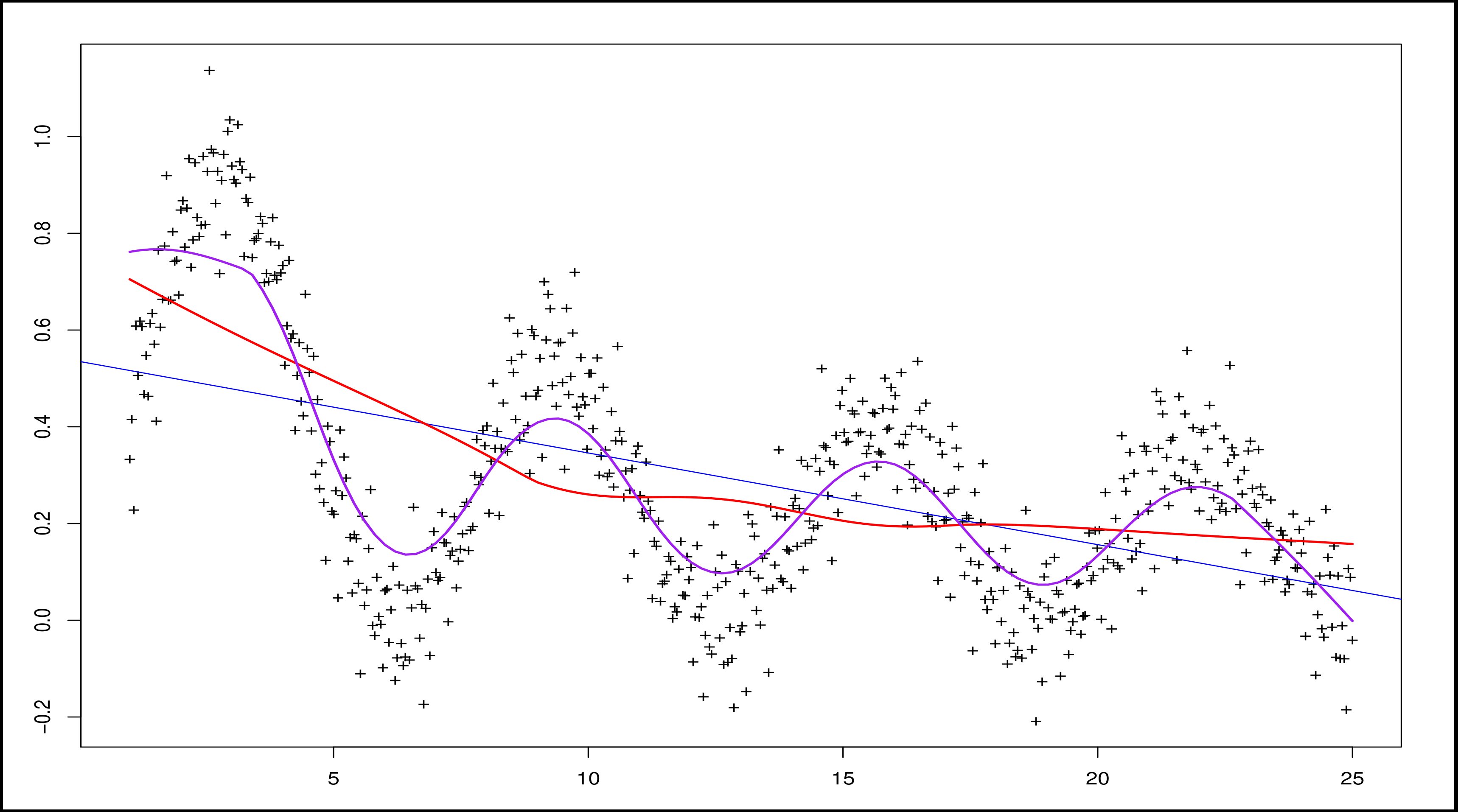
Kernel Smoothers, Rolling Our Own

```
X <- seq(0,10,length=100)
Y <- sin(X) + rnorm(length(X),0,0.3)
postscript("Class.Stat.Comp/kernel.smooth.ps")
par(mar=c(3,3,1,1),col.axis="white",col.lab="white",col.sub="white",
    col="white",bg="slategray")
plot(X,Y,pch="+")
for (j in c(1,5,10)) lines(X,k.sm(X,Y,j),col=colors()[3+24*j],lty=j,lwd=2)
dev.off()
```

Kernel Smoothers, Rolling Our Own



Recall the Lowess Smoother. . .



Lowess Smoother

- Lowess Smoother, Locally-weighted running line smoother (Cleveland 1979). Steps, for each point:
  1. Denote k nearest neighbors of x_i as $N(x_i)$. Based on distance, not symmetry.
  2. Determine the furthest neighbor distance:
  $$\mathbf{D}(x_i) = \max_{N(x_i)} |x_i - x|, \qquad \forall x \in N(x_i).$$
  3. Calculate weights for each jth point in $N(x_i)$ using the **tri-cube weighting function**:
  $$u = u_{ij} = \frac{|x_i - x_j|}{\mathbf{D}(x_i)}$$
  $$w(u_{ij}) = \begin{cases} (1-u^3)^3 & \text{for } 0 \leqslant u \leqslant 1 \\ 0 & \text{otherwise} \end{cases}$$
  4. Fit with a weighted running line smoother using these calculated weights:
  $$\hat{f}(x_i) = \hat{\alpha}_N + \hat{\beta}_N x_i = \frac{\sum_{j=1}^{k} w(u_{ij})Y_{ij}}{\sum_{j=1}^{k} w(u_{ij})}$$

  Note: weights are all positive and decreasing with increasing distance. They also decrease with increasing window width.

Lowess Smoother

- For each data-point we are producing a neighborhood definition with weights and new Y points from the fit:

$$\forall x_i, \begin{bmatrix} x_1, & x_2, & \ldots, & x_k \\ w_1, & w_2, & \ldots, & w_k \\ \hat{y}_1, & \hat{y}_2, & \ldots, & \hat{y}_k \end{bmatrix}$$

- Actually two “flavors” of Lowess available:
  - $\lambda = 1$:

$$\min \sum_{j=1}^{k} w(u_{ij})(y_j - \alpha - \beta x_j)^2$$

  - $\lambda = 2$:

$$\min \sum_{j=1}^{k} w(u_{ij})(y_j - \alpha - \beta x_j - \gamma x_j^2)^2$$

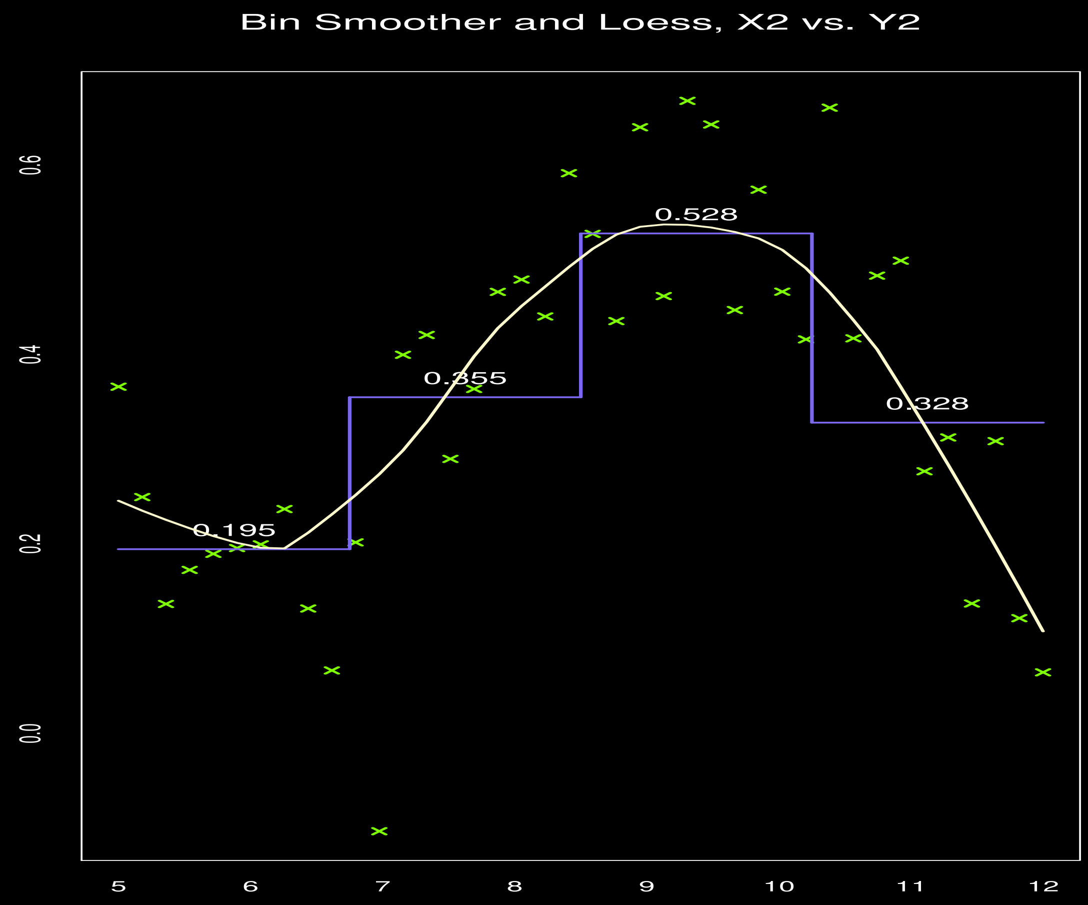
Lowess Smoother In R

```
x2 <- seq(5,12,length=40)
y2 <- (2/(pi*x2))^(0.5)*(1-cos(x2)) + rnorm(length(x2),0,1/10) + 0.1
postscript("Class.Stat.Comp/cognitive2j.ps")
par(mfrow=c(1,1),mar=c(2,2,2,2),oma=c(3,3,3,3),col.axis="white",col.lab="black",
    col.sub="white",col="white",bg="black")
plot(x2,y2,pch=4,col="chartreuse1",lwd=2)
```

Continued. . .

```
# Do a regressogram first
x2.cuts <- quantile(x2,seq(0,1,length=5))
y2.bins <- matrix(y2,ncol=4)
y2.means <- apply(y2.bins,2,mean)
for (i in 1:(length(x2.cuts)-1))  {
    segments(x2.cuts[i],y2.means[i],x2.cuts[i+1],y2.means[i],
        col="mediumslateblue",lwd=2)
    segments(x2.cuts[i+1],y2.means[i],x2.cuts[i+1],y2.means[i+1],
        col="mediumslateblue",lwd=2)
    text((x2.cuts[i]+x2.cuts[i+1])/2,y2.means[i]+0.02,cex=1.2,
        round(y2.means[i],3))
}
lines(lowess(x2,y2,f=0.4),col="lemonchiffon",lwd=2)
mtext(outer=TRUE,side=3,cex=1.5,line=0.25,"Bin Smoother and Loess, X2 vs. Y2")
dev.off()
```

Lowess Smoother

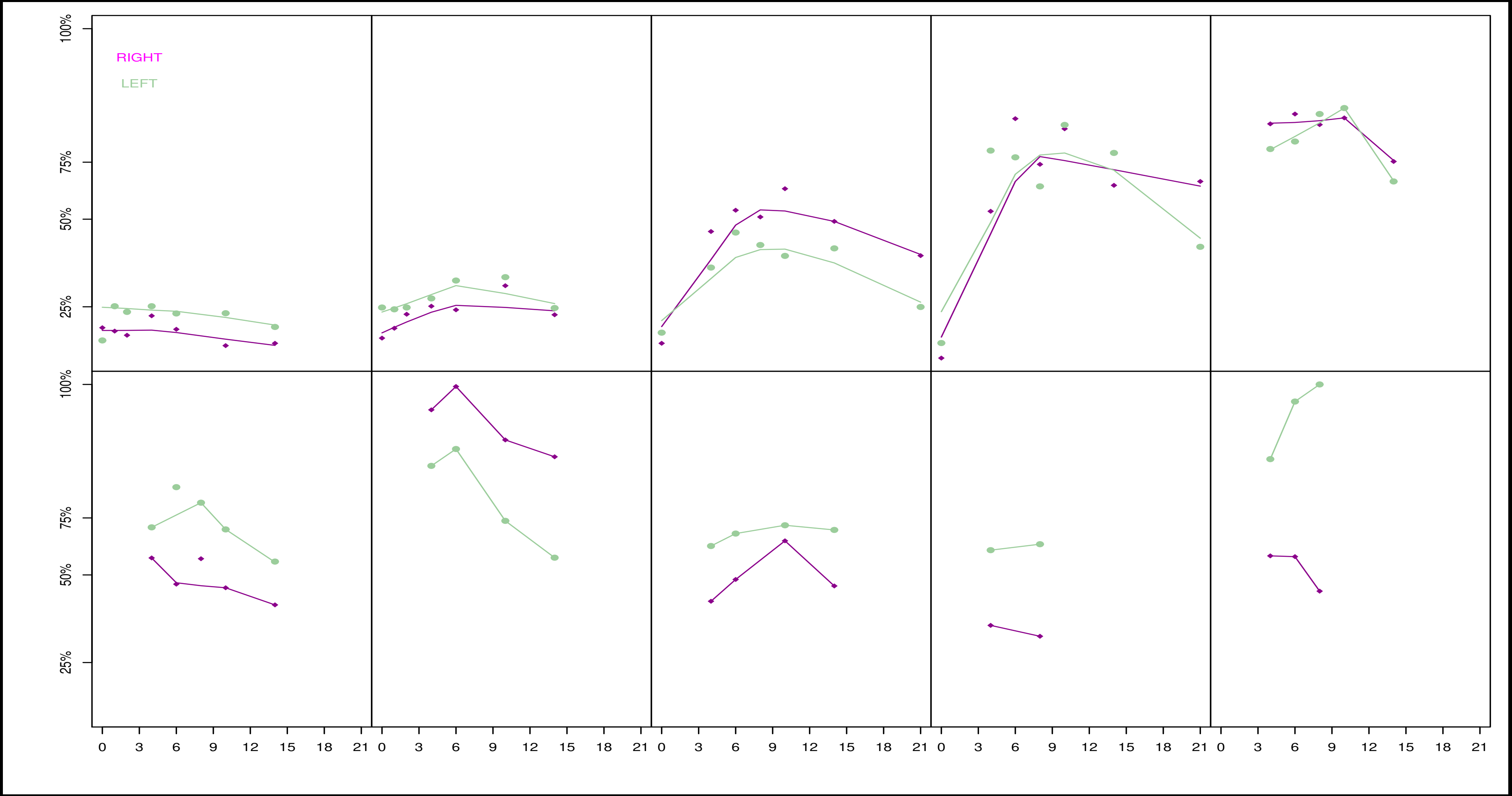


Using Lowess for Nested Classification Factors

- Experiment recording mean pixel intensity of the right and left lymph nodes in the auxiliary region from (tomography) CT scans of 10 dogs over 14 days after intravenous application of a dye contrast.
- Data:

```
connect1 <- url("http://jgill.wustl.edu/data/Pixel.rda")
load(connect1);   close(connect1)
Pixel[,"Side"] <- as.numeric( Pixel[,"Side"] )
Pixel[1:10,]
  Case Side day  pixel
1    1    2   0 1045.8
2    1    2   1 1044.5
3    1    2   2 1042.9
4    1    2   4 1050.4
5    1    2   6 1045.2
6    1    2  10 1038.9
7    1    2  14 1039.8
8    2    2   0 1041.8
9    2    2   1 1045.6
10   2    2   2 1051.0
```

Using Lowess for Nested Classification Factors



Using Lowess for Nested Classification Factors

```
postscript("Class.Multilevel/dogs.ps")
J = max(as.numeric(Pixel$Person))
day.range <- c(min(as.numeric(Pixel$day)), max(as.numeric(Pixel$day)))
pixel.range <- c(min(as.numeric(Pixel$pixel)), max(as.numeric(Pixel$pixel)))
par(oma=c(5,5,1,1),mar=c(0,0,0,0),mfrow=c(J/(J/2),J/2),bg="white")
for (i in 1:J)  {
    patient.i <- Pixel[Pixel["Person"]==i,]
    plot(patient.i[patient.i["Side"]==2,][,3:4], xlim=day.range, ylim=pixel.range,
         pch=18,col="darkmagenta",yaxt="n",xaxt="n")
    lines(lowess(patient.i[patient.i["Side"]==2,][,3:4], f=0.9), col="darkmagenta")
    points(patient.i[patient.i["Side"]==1,][,3:4], pch=19,col="darkseagreen3")
    lines(lowess(patient.i[patient.i["Side"]==1,][,3:4], f=0.9), col="darkseagreen3")
    if (i > 5) axis(side=1,labels=seq(0,21,by=3),at=seq(0,21,by=3))
    if (i == 1 | i==6) axis(side=2,labels=names(quantile(Pixel$pixel)[-1]),
                            at=quantile(Pixel$pixel)[-1])
    if (i == 1)  { text(3,1150,"RIGHT",col="magenta")
                   text(3,1140,"LEFT",col="darkseagreen3") }
}
dev.off()
```

Using Lowess for Nested Classification Factors

- We want to account for nesting of sides within dogs.
- We also need to think carefully about where to put intercepts in the various levels (choices).
- Note that with nesting levels we can qualitatively different results for standard error terms.
- There is an obvious nonlinear effect to account for in days with a peak at about 10.
- A general model:

$$\begin{aligned} y_{ijt} &= \beta_0 + \beta_1 T_i + \beta_2 T_i^2 + \beta_3 S_{ij} + \beta_{4,j[i]} + \beta_{5,j[i]} + \epsilon_{ijt} \\ \beta_4 &\sim N(\gamma_0 + \gamma_1 S_{ij} + \gamma_2 T_i, \sigma^2_{\beta_4,}) \\ \beta_5 &\sim N(\delta_0 + \delta_1 T_i, \sigma^2_{\beta_5,}) \end{aligned}$$

where:

y_{ijt}	pixels for ith patient, side j, at time t
T_{ij}	ith patient's time t (not all measured at regular intervals!)
S_{ij}	ith patient's side: 1 (left) or 2 (right).

Using Lowess for Nested Classification Factors

- Random intercepts model:

```
fm1Pixel <- lmer(pixel ~ day + I(day^2) + (1 | Person), data = Pixel)
summary(fm1Pixel)
   AIC   BIC logLik deviance REMLdev
 895.2 908.4 -442.6    886.9   885.2
Random effects:
 Groups   Name        Variance Std.Dev.
 Person   (Intercept) 633.89   25.177
 Residual             263.95   16.247
Number of obs: 102, groups: Person, 10

Fixed effects:
              Estimate Std. Error t value          Correlation of Fixed Effects:
(Intercept) 1074.16253    9.12496  117.72                   (Intr) day
day            4.94201    1.03686    4.77          day      -0.426
I(day^2)      -0.25047    0.05303   -4.72          I(day^2)  0.355 -0.945
```

Using Lowess for Nested Classification Factors

- Checking for a non-grouped difference between the left and right sides across dogs:

```
fm2Pixel <- lmer(pixel ~ day + I(day^2) + Side + (1 | Person), data = Pixel)
summary(fm2Pixel)
   AIC BIC logLik deviance REMLdev
 890.2 906 -439.1      884   878.2
Random effects:
 Groups   Name        Variance Std.Dev.
 Person   (Intercept) 634.54   25.190
 Residual             258.56   16.080
Number of obs: 102, groups: Person, 10

Fixed effects:
              Estimate Std. Error t value          Correlation of Fixed Effects:
(Intercept) 1082.28413   10.28304  105.25                  (Intr) day    I(d^2)
day            4.93811    1.02628    4.81          day      -0.374
I(day^2)      -0.25029    0.05249   -4.77          I(day^2)  0.311 -0.945
Side          -5.40196    3.18425   -1.70          Side     -0.464  0.000  0.000
```

Using Lowess for Nested Classification Factors

```
anova(fm2Pixel,fm1Pixel)

fm1Pixel: pixel ~ day + I(day^2) + (1 | Person)
fm2Pixel: pixel ~ day + I(day^2) + Side + (1 | Person)
         Df     AIC     BIC  logLik  Chisq Chi Df Pr(>Chisq)
fm1Pixel  5  896.87  910.00 -443.44
fm2Pixel  6  895.94  911.69 -441.97 2.9352      1    0.08667
```

Using Lowess for Nested Classification Factors

- A model that nests the side in patient as well as the days in patient, days has intercept:

```
fm3Pixel <- lmer(pixel ~ day + I(day^2) + (Side | Person) + (-1 + day | Person),
                 data = Pixel); summary(fm3Pixel)

   AIC   BIC logLik deviance REMLdev
 843.4 864.4 -413.7    829.7   827.4
Random effects:
 Groups   Name        Variance  Std.Dev. Corr
 Person   (Intercept) 1937.2268 44.014
          Side         565.3087 23.776   -0.746
 Person   day            3.1791  1.783
 Residual               81.3617  9.020
Number of obs: 102, groups: Person, 10

Fixed effects:
             Estimate Std. Error t value         Correlation of Fixed Effects:
(Intercept) 1073.6530     9.7331  110.31                 (Intr) day
day            6.2335     0.8655    7.20        day      -0.202
I(day^2)      -0.3685     0.0340  -10.84        I(day^2)  0.195 -0.679
```

Using Lowess for Nested Classification Factors

```
anova( fm3Pixel, fm2Pixel )

fm2Pixel: pixel ~ day + I(day^2) + Side + (1 | Person)
fm3Pixel: pixel ~ day + I(day^2) + (Side | Person) + (-1 + day | Person)
         Df     AIC     BIC  logLik Chisq Chi Df Pr(>Chisq)
fm2Pixel  6  895.94  911.69 -441.97
fm3Pixel  8  845.69  866.69 -414.84 54.25      2  1.658e-12
```

Using Lowess for Nested Classification Factors

- Is it worth having Side as a fixed effect too?

```
fm4Pixel <- lmer(pixel ~ day + I(day^2) + Side + (Side|Person) + (-1+day|Person),
                 data = Pixel); summary(fm4Pixel)
 AIC   BIC logLik deviance REMLdev
 838 861.7   -410    828.2     820
Random effects:
 Groups   Name        Variance  Std.Dev. Corr
 Person   (Intercept) 1908.1242 43.6821
          Side         544.6817 23.3384  -0.741
 Person   day            3.1794  1.7831
 Residual               81.2434  9.0135
Number of obs: 102, groups: Person, 10

Fixed effects:
              Estimate Std. Error t value       Correlation of Fixed Effects:
(Intercept) 1086.41495   14.41456   75.37                (Intr) day    I(d^2)
day            6.23301    0.86512    7.20      day       -0.136
I(day^2)      -0.36852    0.03398  -10.85      I(day^2)   0.131 -0.679
Side          -9.19399    7.62640   -1.21      Side      -0.737  0.000  0.000
```

Using Lowess for Nested Classification Factors

```
anova( fm4Pixel, fm3Pixel )

fm3Pixel: pixel ~ day + I(day^2) + (Side | Person) + (-1 + day | Person)
fm4Pixel: pixel ~ day + I(day^2) + Side + (Side | Person) + (-1 + day | Person)
         Df     AIC     BIC  logLik  Chisq Chi Df Pr(>Chisq)
fm3Pixel  8  845.69  866.69 -414.84
fm4Pixel  9  846.20  869.83 -414.10 1.4834      1     0.2232
```

lowess versus loess

- The full syntax of `lowess` is:

  ```
  lowess(x, y = NULL, f = 2/3, iter = 3, delta = 0.01 * diff(range(xy$x[o])))
  ```

  where `iter` is the number of "robustifying" iterations to run through the tricube weighting functions (smaller means it runs faster), and `delta` gives a short-cut: don't calculate fit at data points delta of the last computed point but use linear interpolation instead.

- But `loess` offers:

  ```
  loess(formula, data, weights, subset, na.action, model = FALSE, span = 0.75,
        enp.target, degree = 2, parametric = FALSE, drop.square = FALSE,
        normalize = TRUE, family = c("gaussian", "symmetric"),
        method = c("loess", "model.frame"), control = loess.control(...), ...)
  ```

lowess versus loess

- Where these options provide extra functionality:

formula	a full R modeling formula
data	an optional data frame, list or environment
weights	optional regression-style weights for each case
subset	an optional subset of the data to be used
na.action	the regular model treatment of missing data
model	should the model frame be returned?
span	the parameter which controls the degree of smoothing, like f
enp.target	another way to specify span: the approximate equivalent number of points to be used
degree	the degree of the polynomials to be used, normally 1 or 2
	(this is the main reason people use loess instead of lowess
parametric	allows terms to be fitted globally rather than locally
drop.square	if fits have more than one predictor and degree=2, should the quadratic term be dropped some variables?
family	if ”gaussian” fitting is by least-squares, and if ”symmetric” by a re-descending M-estimator
method	fit the model or just extract the model frame (data plus modeling information)
control	control parameters, see loess.control

Regression Splines

- Choose breakpoints (also called knots).
  - This is the key tradeoff: more knots mean more flexibility, but can be more computationally intensive and sometimes too wavy.
  - Strategies:
    1. cardinal knots: uniform over range of X data,
    2. at quantiles,
    3. adaptive (complex),
    4. at selected X_i.
  - Effects:
    1. bad choices can be dramatic,
    2. bad choices can miss important features.
  - Setup: interior knots given by
$$\xi_1 < \xi_2 < \cdots < \xi_S$$
over the range $(X_{(1)}, X_{(n)})$, along with the boundary knots:
$$\xi_0 < X_{(1)}, \qquad X_{(n)} < \xi_S$$

Regression Splines

- We need to choose a basis function which must join smoothly at knots.
- Truncated Power Series, with S knots:

$$s(x) = \delta_0 + \delta_1 x + \delta_2 x^2 + \delta_3 x^3 + \sum_{i=1}^{S} \delta_{3+i}(x - \xi_i)_+^3$$

Where $(x - \xi_i)_+^3$ means we include only positive terms, else zero:

$$(x - \xi_i)_+^3 = \max[0, (x - \xi_i)^3]$$

- So $s(x)$ is a linear weighted combination of the $S + 4$ functions:

$$\begin{aligned}
P_0(x) &= 1 \\
P_1(x) &= x \\
P_2(x) &= x^2 \\
P_3(x) &= x^3 \\
P_{S_1}(x) &= (x - \xi_1)_+^3, \qquad P_{S_2}(x) = (x - \xi_2)_+^3, \qquad P_{S_3}(x) = (x - \xi_3)_+^3
\end{aligned}$$

meaning that the the function is linear in these $S + 4$ parameters and can be estimated with OLS.

Cubic Regression Splines

- Cubic Splines add the condition that at the boundary knots, $s(x)''$ and $s(x)'''$ are both zero, so $s(x)$ is linear (but not necessarily flat) in the intervals:

$$[\xi_0, \xi_1], \qquad\qquad [\xi_S, \xi_{S+1}].$$

- Now estimate by applying the function:
  1. regress y_i on $f(x_i)$, say for $S = 3$:

$$\begin{aligned}\hat{y}_i &= f(x_i)\\ &= \delta_0 + \delta_1 x_i + \delta_2 x_i^2 + \delta_3 x_i^3\\ &\qquad + \delta_4 (x_i - \xi_1)_+^3\\ &\qquad + \delta_5 (x_i - \xi_2)_+^3\\ &\qquad + \delta_6 (x_i - \xi_3)_+^3\end{aligned}$$

     using OLS.
  2. Obtains 7 coefficient estimates: $\hat{\delta}_i,\ i = 1, \ldots, 7$.

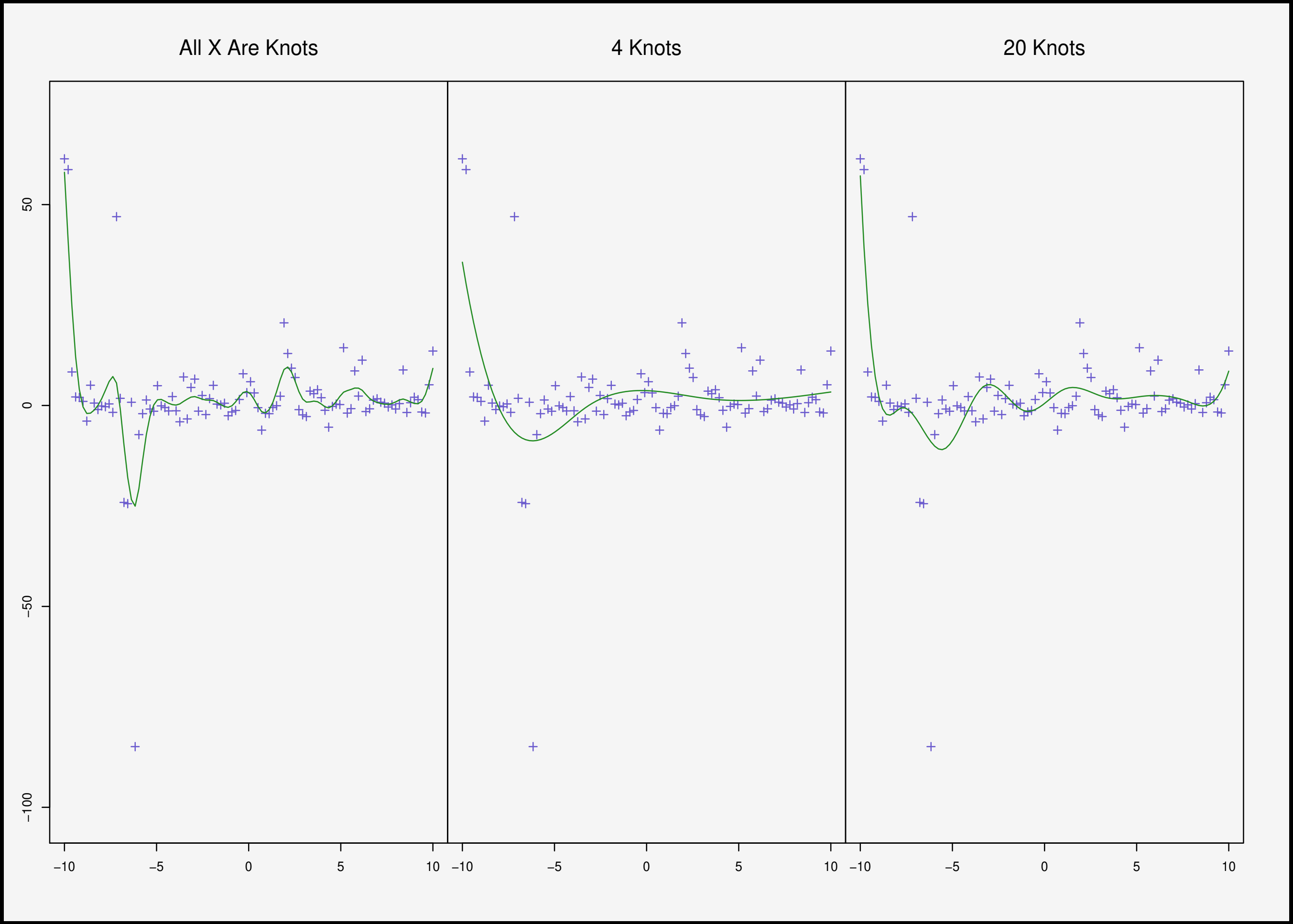
- Extension: B-Splines, different parameterizations,. . .

Cubic Regression Splines

- The R function for Cubic Splines is `smooth.spline`:

```
x1 <- seq(-10,10,length=100);   y1 <- cos(x1)/(rt(length(x1),50)*0.5)
postscript("Class.Stat.Comp/cognitive2h.ps")
par(mfrow=c(1,3),oma=c(3,3,3,3),mar=c(2,0,2,0),bg="whitesmoke")

plot(x1,y1,pch=3,ylim=range(y1)*1.2,col="slateblue")
spline.out <- smooth.spline(x1,y1,all.knots=TRUE)
lines(spline.out$x,spline.out$y,col="forest green")
mtext("All X Are Knots",outer=FALSE,side=3,cex=1.1,line=1.5)

plot(x1,y1,pch=3,ylim=range(y1)*1.2,yaxt="n",col="slateblue")
spline.out <- smooth.spline(x1,y1,all.knots=FALSE,nknots=4)
lines(spline.out$x,spline.out$y,col="forest green")
mtext("4 Knots",outer=FALSE,side=3,cex=1.1,line=1.5)

plot(x1,y1,pch=3,ylim=range(y1)*1.2,yaxt="n",col="slateblue")
spline.out <- smooth.spline(x1,y1,all.knots=FALSE,nknots=10)
lines(spline.out$x,spline.out$y,col="forest green")
mtext("10 Knots",outer=FALSE,side=3,cex=1.1,line=1.5);     dev.off()
```

Cubic Regression Splines



Penalized Splines

- The goal is:
$$\min \sum_{i=1}^{n}(y_i - f(x_i))^2 + \lambda \int_a^b (f''(t))^2 dt$$
where λ is a fixed constant, and $a < x_{(1)}, x_{(n)} < b$.
- The idea is that the first term promotes fit and the second term penalizes overfitting.
- The function $f(x)$ has an explicit and unique form that minimizes the cubic spline with knots at all of the x_i.
- If $\lambda = \infty$, then we get interpolation.
- As λ approaches 0, we get closer linear regression.

Estimating Penalized Splines

- For regression-style models, another way of notating the function to be minimized is:

$$||\mathbf{y} - \mathbf{X}\boldsymbol{\beta}||^2 + \lambda \int_a^b (f''(t))^2 dt.$$

- Because $f(t)$ is linear in the parameters by design, it can be written as quadratic:

$$\int_a^b (f''(t))^2 dt = \boldsymbol{\beta}\mathbf{S}\boldsymbol{\beta}',$$

  where $\mathbf{S}$ is a matrix of known coefficients determined by the form of $f(t)$.

- So the penalized least squares estimator is given by:

$$\hat{B} = (\mathbf{X}'\mathbf{X} + \lambda\mathbf{S})^{-1}\mathbf{X}'\mathbf{y},$$

  with the associated hat (influence) matrix:

$$\mathbf{A} = \mathbf{X}(\mathbf{X}'\mathbf{X} + \lambda\mathbf{S})^{-1}\mathbf{X}'.$$

Estimating Penalized Splines

- The matrix $\mathbf{A}$ also gives the *effective degrees of freedom* for the smoothed fit by its trace.
- The $\max[\text{tr}(\mathbf{A})]$ is the number of parameters minus the number of constraints, and the $\min[\text{tr}(\mathbf{A})]$ is the max minus the rank of the $\mathbf{S}$ matrix.
- As the number of parameters goes from zero to infinity, the edf moves upward between these two quantities.
- We can use edf for hypothesis testing between two models.

Penalized Splines Code

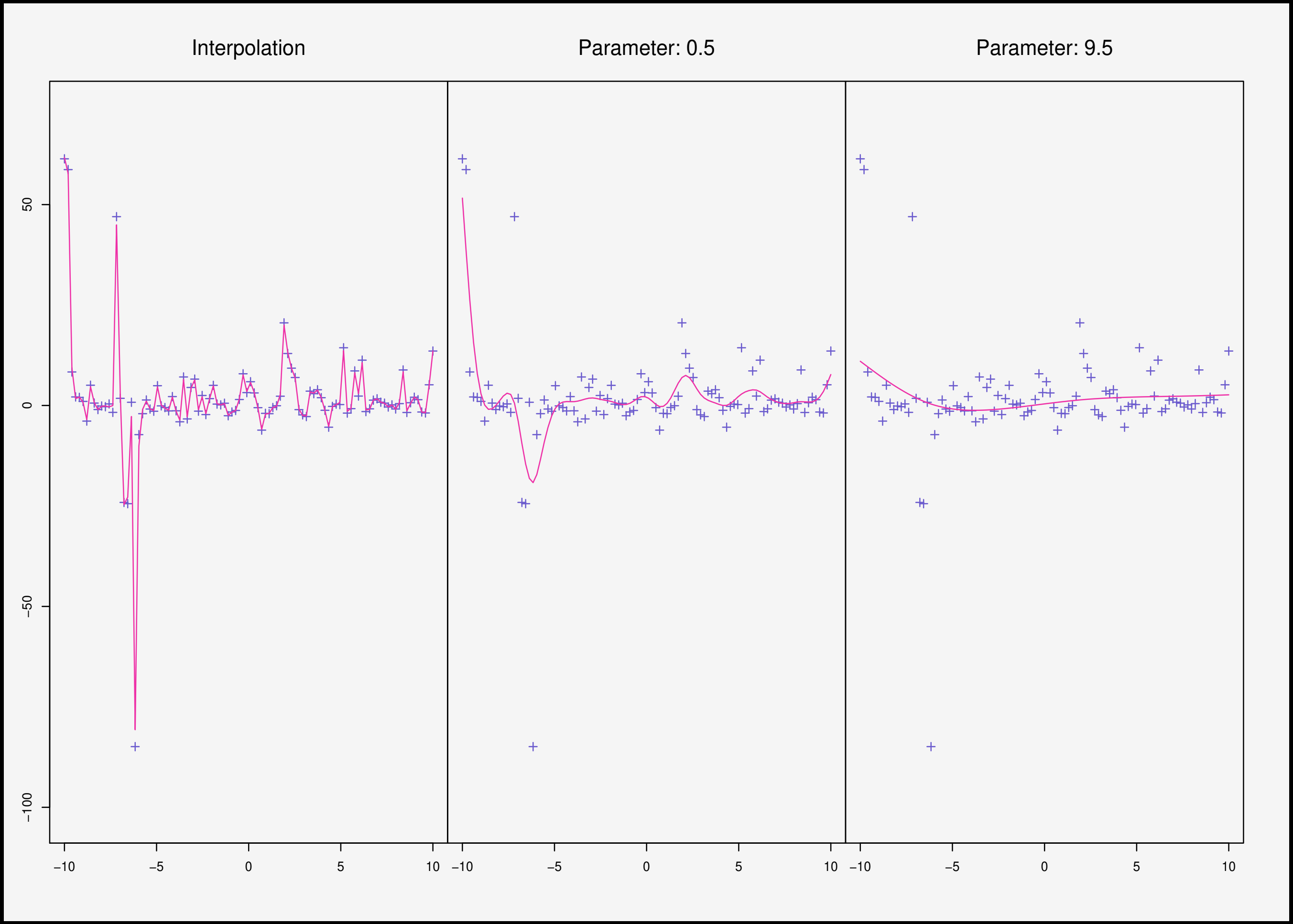
- The R function for Penalized Splines is also `smooth.spline`:

```
nonlin.mat <- read.table("http://jgill.wustl.edu/data/gam.test.dat",header=TRUE)
attach(nonlin.mat)
postscript("Class.Stat.Comp/cognitive2i.ps")
par(mfrow=c(1,3),oma=c(3,3,3,3),mar=c(2,0,2,0),bg="whitesmoke")

plot(x1,y1,pch=3,ylim=range(y1)*1.2,col="slateblue")
spline.out <- smooth.spline(x1,y1,spar=0.1,all.knots=TRUE)
lines(spline.out$x,spline.out$y,col="maroon2")
mtext("Interpolation",outer=FALSE,side=3,cex=1.1,line=1.5)
```

Penalized Splines

```
plot(x1,y1,pch=3,ylim=range(y1)*1.2,yaxt="n",col="slateblue")
spline.out <- smooth.spline(x1,y1,spar=0.5)
lines(spline.out$x,spline.out$y,col="maroon2")
mtext("Parameter: 0.5",outer=FALSE,side=3,cex=1.1,line=1.5)

plot(x1,y1,pch=3,ylim=range(y1)*1.2,yaxt="n",col="slateblue")
spline.out <- smooth.spline(x1,y1,spar=0.95)
lines(spline.out$x,spline.out$y,col="maroon2")
mtext("Parameter: 9.5",outer=FALSE,side=3,cex=1.1,line=1.5)
dev.off()
detach(nonlin.mat)
```

Penalized Splines



Thin Plate Splines

- Some disadvantages of standard spline approaches: user must stipulated knots, bases given for only one variable, criteria for bases unclear.
- We want: knot-free spline bases over any number of explanatory variables that have optimal properties.
- Thin plate splines (Duchon 1977; Wahba 1990, Green & Silverman 1994, Wood 2006, Chapter 4) are a good solution to this problem.
- General Strategy: penalize with derivative functions, to produce a smooth function according to

  $$y_i = g(\mathbf{x}_i) + \boldsymbol{\epsilon}_i$$

  where $\boldsymbol{\epsilon}_i$ is a random error vector with good properties and $\mathbf{x}_i$ is a d-length explanatory variable vector.
- It automatically calculates how much weight to give the conflicting goals of following the data and making the fit as smooth as possible by putting the tradeoff into an explicit function.

Thin Plate Splines

- Objective: find a function $\mathbf{f}$ that minimizes the vector norm:

$$||\mathbf{y} - \mathbf{f}||^2 = \lambda J_{md}(f)$$

  where:

  - $\mathbf{y}$ is the n-length outcome variable vector,
  - $\mathbf{f} = |f(x_1), f(x_2), \dots, f(x_n)|$,
  - λ is a smoothing parameter,
  - J_{md} is a penalty term based on the curviness of the smooth.

- This is very much in line with the spline technology that we have been studying except that there will be more "automatic" rather than human decisions.

Thin Plate Splines Penalty Function

- The penalty is defined with the following function:

$$J_{md} = \int \cdots \int \sum_{\eta_1 + \cdots + \eta_d = m} \frac{m!}{\eta_1! \cdots \eta_k!} \left(\frac{\partial^m f}{\partial x_1^{\eta_1} \cdots \partial x_d^{\eta_d}} \right)^2 dx_1 \cdots dx_d$$

  where $2m > d + 1$.

- For instance, when $d = 2 \;\; \eta_1 = \eta_2 = 1$, and $m = 2$, then:

$$J_{22} = 2 \int \int \left[\left(\frac{\partial^2 f}{\partial x_1^2} \right) + \left(\frac{\partial^2 f}{\partial x_1 \partial x_2} \right) + \left(\frac{\partial^2 f}{\partial x_2^2} \right) \right] dx_1 dx_2.$$

Thin Plate Splines

- One function that minimizes $||\mathbf{y} - \mathbf{f}||^2 = \lambda J_{md}(f)$ is:

$$\hat{f}(\mathbf{x}) = \sum_{i=1}^{n} \mathbf{D}_i \eta_{md}(||\mathbf{y} - \mathbf{f}||) + \sum_{j=1}^{M} \boldsymbol{\alpha}_j \phi_j(\mathbf{x}),$$

  where:

  - $\mathbf{D}$ and $\boldsymbol{\alpha}$ are coefficient vectors to be estimated.
  - $\mathbf{D}$ has the linear constraint $\mathbf{T}'\mathbf{D} = 0$, with the matrix values $T_{ij} = \phi_j(\mathbf{x}_i)$, which are linearly independent polynomials spanning $\Re^d$ of degree less than m as well as spanning the null space of J_{mn}. Returning to the example where $m = d = 3$,

$$\phi_1 = 1, \qquad \phi_2 = x_1, \qquad \phi_3 = x_2.$$

  - Finally:

$$\eta_{md}(\chi) = \begin{cases} \frac{(-1)^{m+1+d/2}}{2^{2m}\pi^{d/2}(m-1)!(m-d/2)!}\chi^{2m-d}\log(\chi) & d \text{ even} \\ \frac{\Gamma(d/2-m)}{2^{2m}\pi^{d/2}(m-1)!}\chi^{2m-d}\log(\chi) & d \text{ odd} \end{cases}$$

Thin Plate Splines

- Now define the matrix E with elements:

$$\mathbf{E}_{ij} = \eta_{md}(||\mathbf{x}_i - \mathbf{x}_j||).$$

- The fitting problem is now expressible as:

$$\text{minimize } ||\mathbf{y} - \mathbf{E} - \mathbf{T}||^2 + \lambda\mathbf{D}'\mathbf{E}\mathbf{D}, \quad \text{subject to } \mathbf{T}'\mathbf{D} = 0, \text{with respect to } \mathbf{T}, \boldsymbol{\alpha},$$

  where (again) we specify the $\mathbf{T}$ and estimated the $\boldsymbol{\alpha}$ vector.

- This truncates the space of rough components, those with $\mathbf{D}$ parameters, while leaving the smooth components untouched.

- Primary challenge (besides all the math): computational efficiency: there are as many unknown quantities as data-points and estimation time is proportional to d^3.

- Of course this concern goes down every year (thank you Gordon Moore).

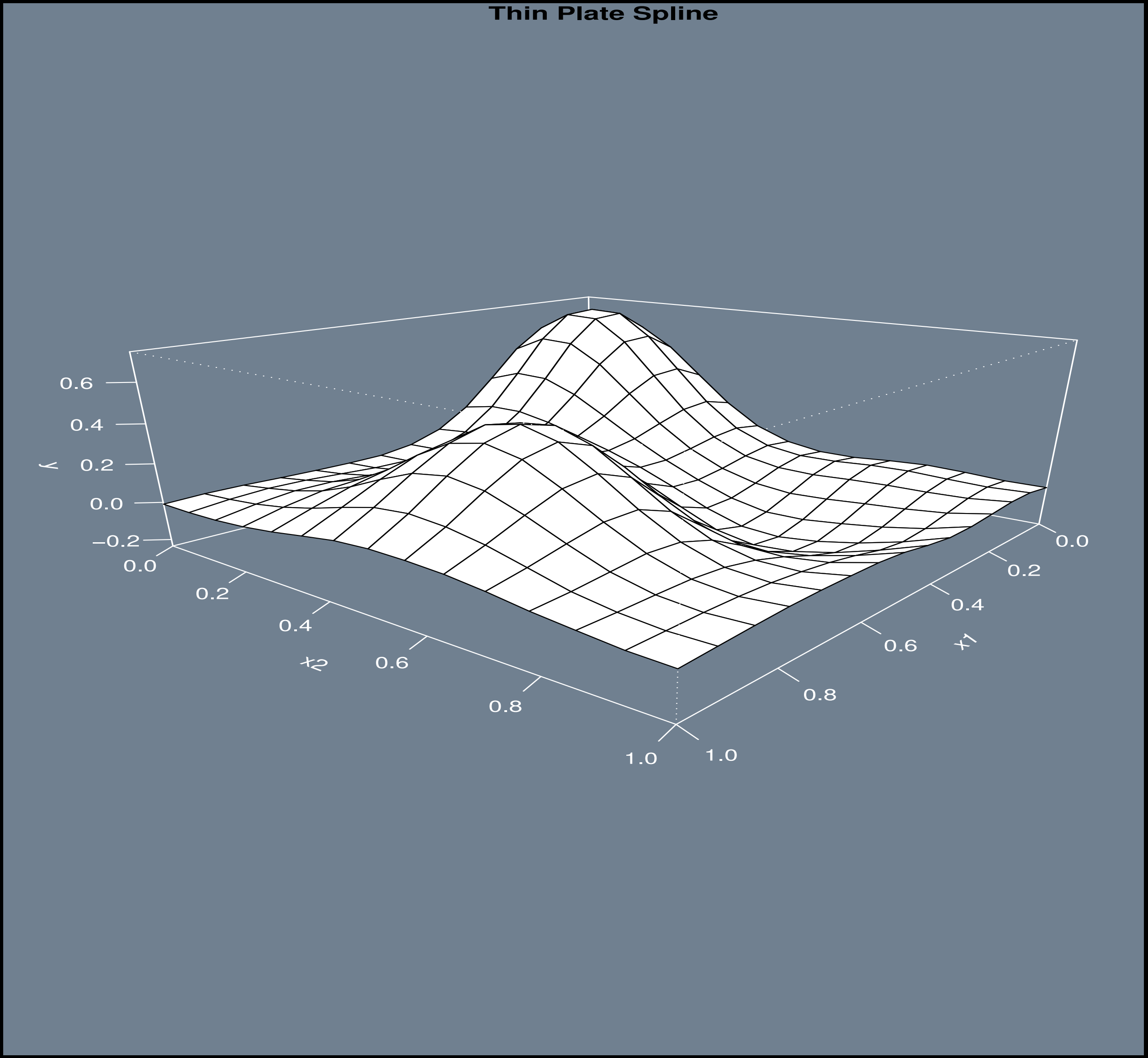
Thin Plate Splines in R

```
library(rgcvpack)

# DEFINE A THREE-DIMENSIONAL FUNCTION (2 IN, 1 OUT)
f <- function(x, y)  {
          0.75*exp( -((10*x-1)^2 + (10*y-1)^2)/5 ) +
          0.50*exp( -((10*x-7)^2 + (10*y-5)^2)/5 ) -
          0.25*exp( -((10*x-4)^2 + (10*y-7)^2)/5 )
}

# CREATE A FAKE DATASET USING THIS FUNCTION
set.seed(pi);   n <- 15;
x2 <- x1 <- seq(0,1,length=n)
y <- outer(x1, x2, f); y <- y + rnorm(n^2,0,0.05*max(abs(y)))

# THE FUNCTION NEEDS THESE AS VECTORS
x1.vec <- rep(x1, n); x2.vec <- rep(x2, rep(n,n)); y.vec <- as.vector(y)
```

Thin Plate Splines in R

```
# RUN THE THIN PLATE SPLINE WITH ALL DATA POINTS AS KNOTS, ORDER 3
thinpl.out <- fitTps(cbind(x1.vec,x2.vec), y.vec, m=3)

# GRAPH
postscript("Class.Stat.Comp/thin.plate.ps")
x2 <- x1 <- seq(0,1,length=length(y.vec))
par(mar=c(3,3,1,1),col.axis="white",col.lab="white",col.sub="white",
    col="white",bg="slategray")
persp(x1, x2, matrix(predict(thinpl.out),n,n), theta=130, phi=20,
      expand=0.50, xlab="x1", ylab="x2", zlab="y", xlim=c(0,1),
      ylim=c(0,1),zlim=range(y), ticktype="detailed", scale=FALSE,
      main="Thin Plate Spline")
dev.off()
```

Thin Plate Splines in R



Smoothing Parameter Selection

- Recall as λ approaches 0, we get closer to interpolation (for a scalar parameter).
- Penalized maximum likelihood methods can only estimate $\boldsymbol{\beta}$ coefficients conditional on smoothing parameters, $\boldsymbol{\lambda}$.
- Two basic scenarios to estimation via minimizing the error quantity, $E(M) = E\left(||\mu - \hat{\mu}||^2/n\right)$:
  - σ^2 known or assumed true, then estimation uses Mallow's C_p-UBRE (Unbiased Risk Estimator).
  - σ^2 unknown, then estimation uses generalized cross validation (GCV).

Smoothing Parameter Selection, Scale Parameter Known

- For regression, the *expected mean square error* is given by:

$$E(M) = E\left(||\boldsymbol{\mu} - \mathbf{X}\hat{\boldsymbol{\beta}}||^2/n\right) = E\left(||\mathbf{y} - \mathbf{A}\mathbf{y}||^2\right)/n - \sigma^2 + 2\mathrm{tr}(\mathbf{A})\sigma^2/n$$

  where $\mathbf{A} = \mathbf{X}(\mathbf{X}'\mathbf{X} + \lambda\mathbf{S})^{-1}\mathbf{X}'$.

- Which means we minimize

$$||\mathbf{y} - \mathbf{A}\mathbf{y}||^2/n - \sigma^2 + 2\mathrm{tr}(\mathbf{A})\sigma^2/n$$

- So the smoothing parameters affect the estimation through $\mathbf{A}$.

Smoothing Parameter Selection, Scale Parameter Unknown

- In this case minimize the *mean square prediction error*:

$$\boldsymbol{\psi} = \sigma + M,$$

  which is the average squared error in predicting a new observation, $\mathbf{y}_{n+1}$, using the fitted model.

- $\boldsymbol{\psi}$ is most easily estimated with *cross validation* (later generalized cross validation):
  - ▻ Jackknife out each case iteratively.
  - ▻ At each step, calculate $\hat{\mu}_{[i]}$, which is the prediction of $\mathbf{y}_i$ from the model that does not include case i.
  - ▻ Finally, calculate:

$$\hat{\boldsymbol{\psi}} = \frac{1}{n}\sum_{i-1}^{n}(y_i - \hat{\mu}_{[i]})^2.$$

Smoothing Parameter Selection, Scale Parameter Unknown

- The $\hat{\mu}_{[i]}$ term means that we have to do the full jackknifing loop through all of the data.
- Actually this can be done in one step using the complete-data model:

$$\hat{\psi} = \frac{1}{n}\sum_{i=1}^{n}\frac{(y_i - \hat{\mu}_i)^2}{(1 - A_{ii})^2}.$$

- This is analogous to the short-hand method for calculating the jackknifed standard error.

Tensor Product Smoothing

- For a two-variable problem, start with the marginal smoothing functions:

$$f_x(x) = \sum_{i=1}^{I} \alpha_i a_i(x) \qquad\qquad f_y(y) = \sum_{\ell=1}^{L} \delta_\ell d_\ell(y),$$

  where α_i, δ_ℓ are parameters, and $a_i(x)$, $d_\ell(y)$ are known basis functions.

- We want $f_x()$ to vary smoothly with y, which can be done by requiring the α_i to vary smoothly with y, which means conditioning on some function of y.

- A convenient way to do this is to use:

$$\alpha_i(y) = \sum_{\ell=1}^{L} \delta_{i,\ell} d_\ell(y),$$

  which explicitly accounts for the smoothed value of y at each i in the smooth of x.

- Therefore from the x side the tensor *product* smooth is the function:

$$f_{x,y}(x,y) = \sum_{i=1}^{I} \sum_{\ell=1}^{L} \delta_{i,\ell} d_\ell(y) a_i(x).$$