Tutorial on Nonparametric Inference

With R

Chad Schafer and Larry Wasserman

cschafer@stat.cmu.edu larry@stat.cmu.edu

Carnegie Mellon University

Outline

- General Concepts of Smoothing, Bias-Variance Tradeoff
- Linear Smoothers
- Cross Validation
- Local Polynomial Regression
- Confidence Bands
- Basis Methods: Splines and Wavelets
- Multiple Regression
- Density Estimation
- Measurement Error
- Inverse Probems
- Classification
- Nonparametric Bayes

Basic Concepts in Smoothing

Problem I: Regression. Observe $(X_1, Y_1), \ldots, (X_n, Y_n)$. Estimate $f(x) = \mathbb{E}(Y|X = x)$. Equivalently:

$$Y_i = f(X_i) + \epsilon_i$$

where $\mathbb{E}(\epsilon_i) = 0$. Simple estimator:

$$\widehat{f}(x) = \text{mean}\{Y_i : \ |X_i - x| \leq h\}.$$

Problem II: Density Estimation. Observe $X_1, \ldots, X_n \sim f$. Estimate f. Simple estimator: $\widehat{f}(x) =$ histogram.

Nonparametric regression: Dark Energy

$$Y_i = f(z_i) + \epsilon_i, \quad i = 1, \ldots, n$$

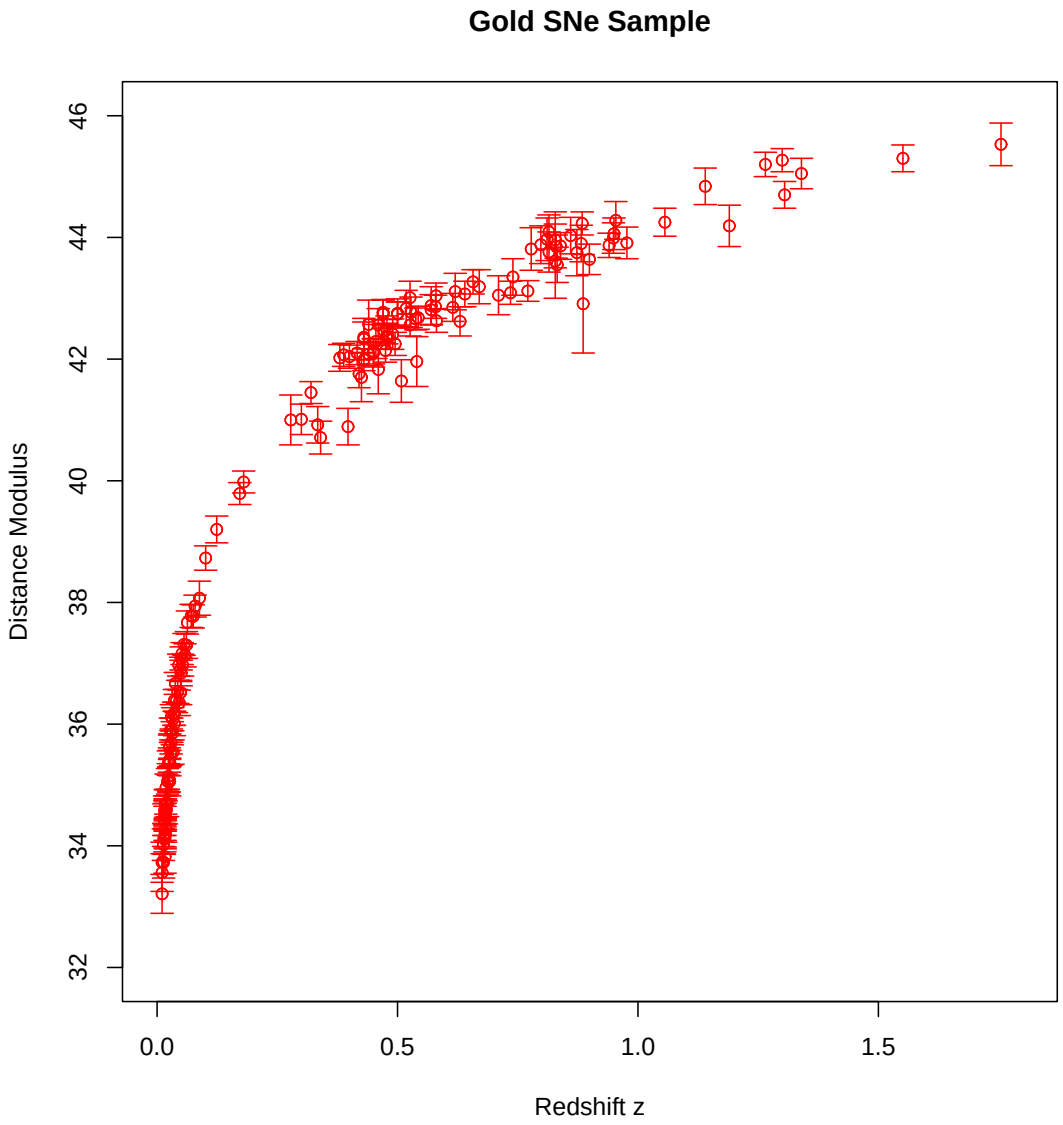
$Y_i =$ luminosity of i^{th} supernova
$z_i =$ redshift of i^{th} supernova
Want to estimate equation of state $w(z)$:

$$w = T(f, f', f'')$$

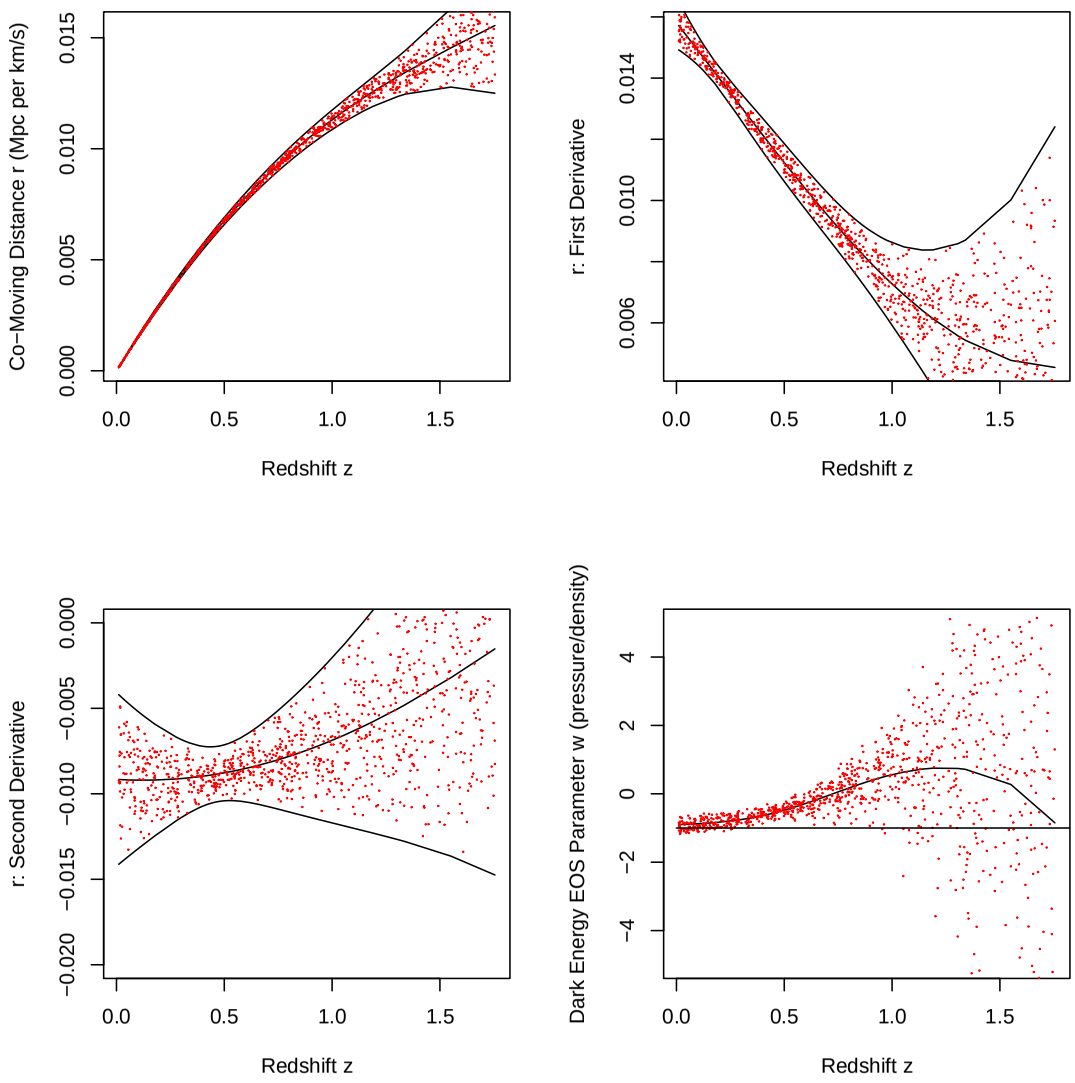
where

$$w(z) = \frac{1+z}{3} \frac{3H_0^2\Omega_M(1+z)^2 + 2\frac{f''(z)}{(f'(z))^3}}{H_0^2\Omega_M(1+z)^3 - \frac{1}{(f'(z))^2}}.$$

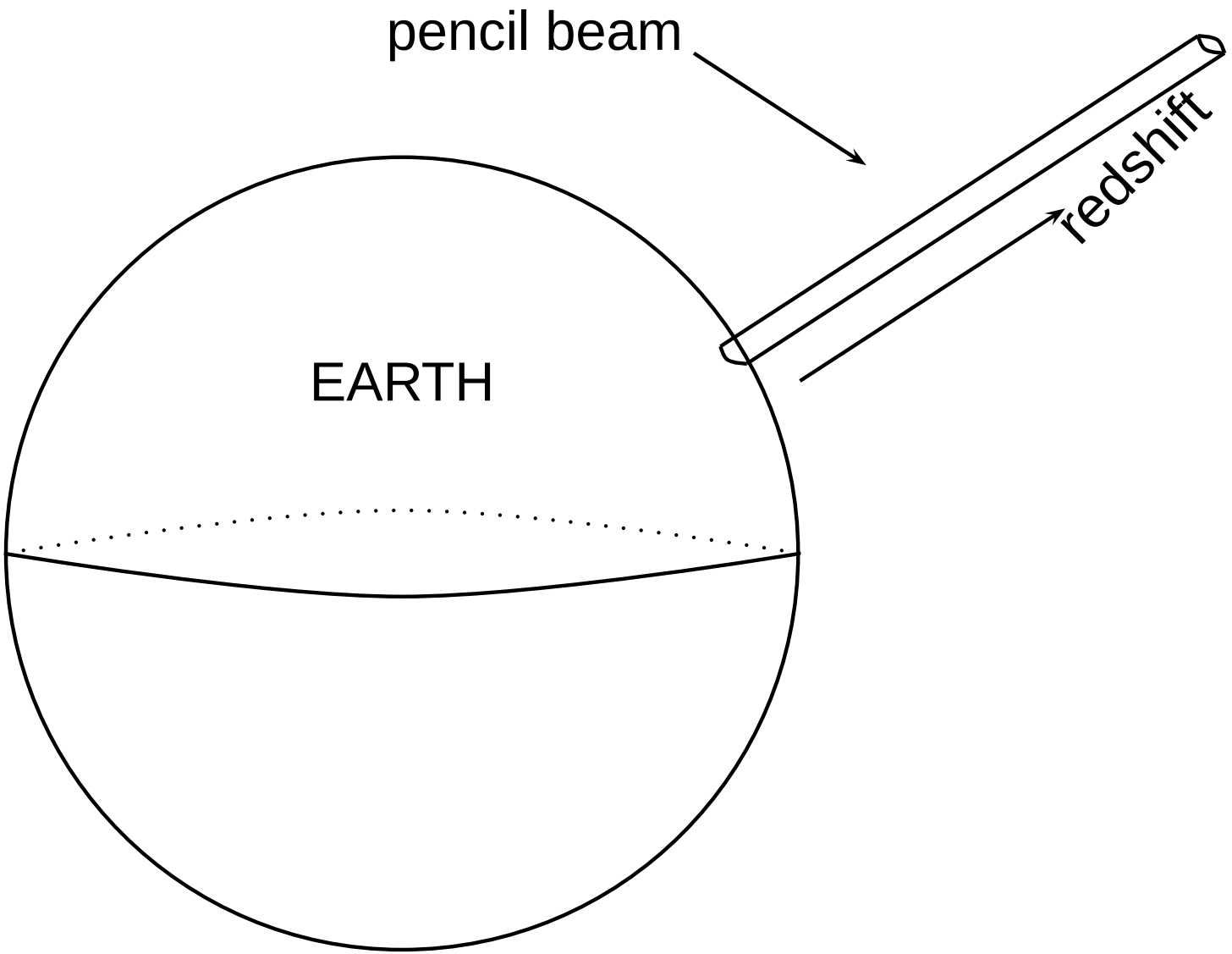
Nonparametric regression: Dark Energy



Nonparametric regression: Dark Energy

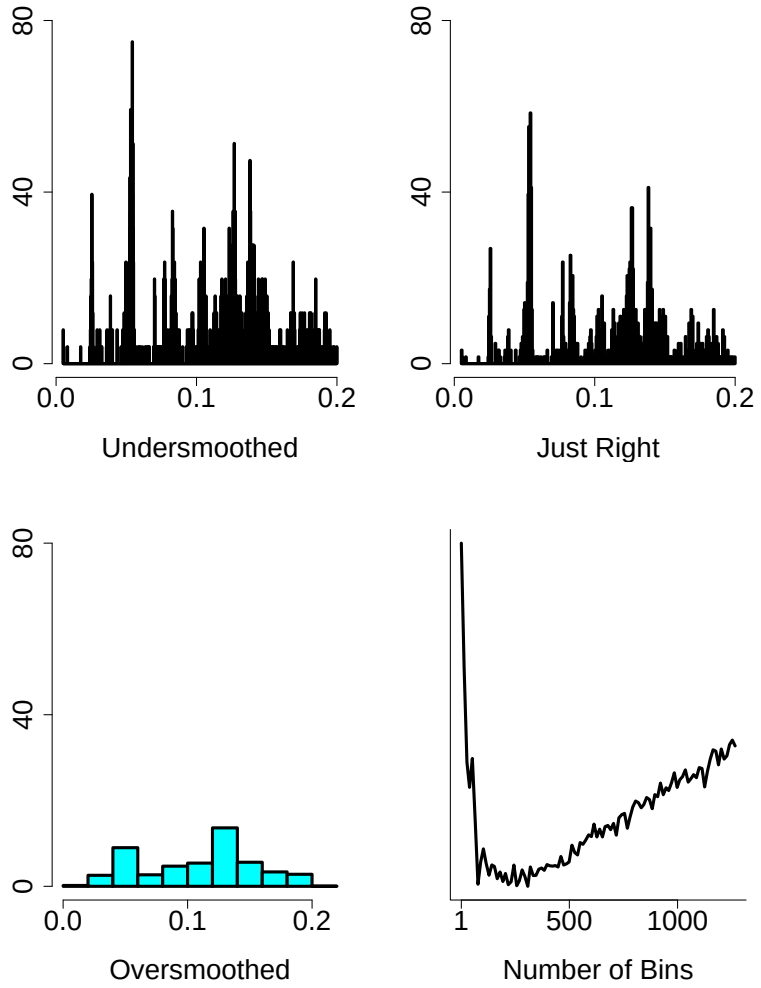


Density Estimation

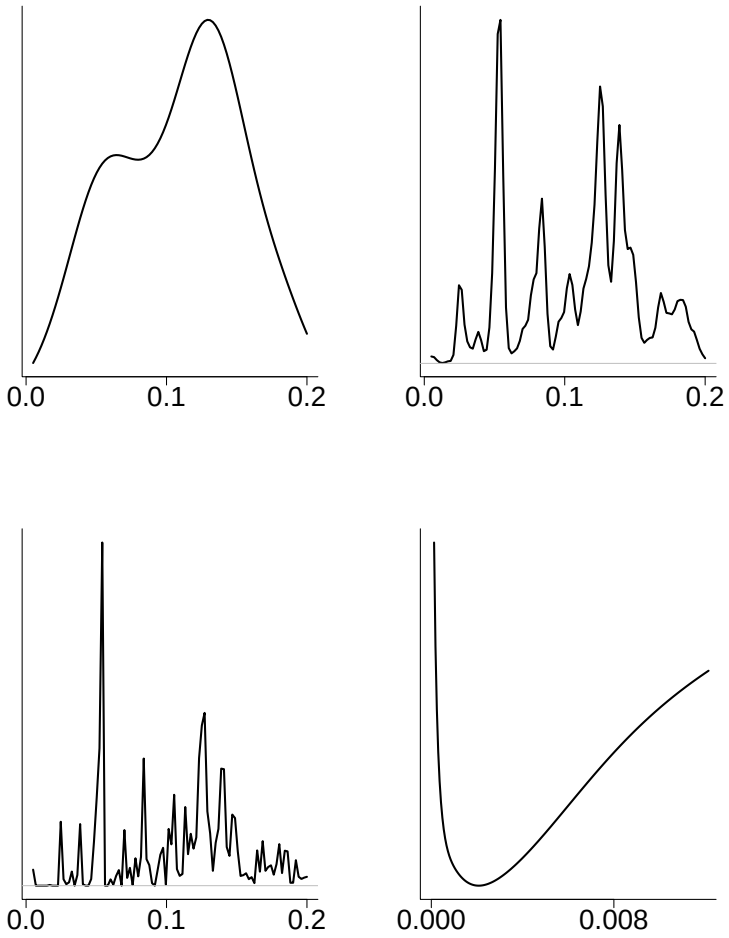


Density Estimation: Histogram

Example: Redshifts (pencil beam).



Density Estimation: Kernel Smoother



The Bias–Variance Tradeoff

Every smoother requires choosing a smoothing parameter h. For a histogram, h = binwidth. Consider the regression estimator based on local averaging:

$$\widehat{f}(x) = \text{mean}\{Y_i : \ |X_i - x| \leq h\}.$$

In both case, $h \uparrow$ implies $\widehat{f}$ is smoother.

Squared error loss:

$$L(f(x), \widehat{f}_n(x)) = (f(x) - \widehat{f}_n(x))^2.$$

Mean squared error MSE (risk)

$$\text{MSE} = R(f(x), \widehat{f}_n(x)) = \mathbb{E}(L(f(x), \widehat{f}_n(x))).$$

The Bias–Variance Tradeoff

$$R(f(x), \widehat{f}_n(x)) = \text{bias}_x^2 + \text{variance}_x$$

where

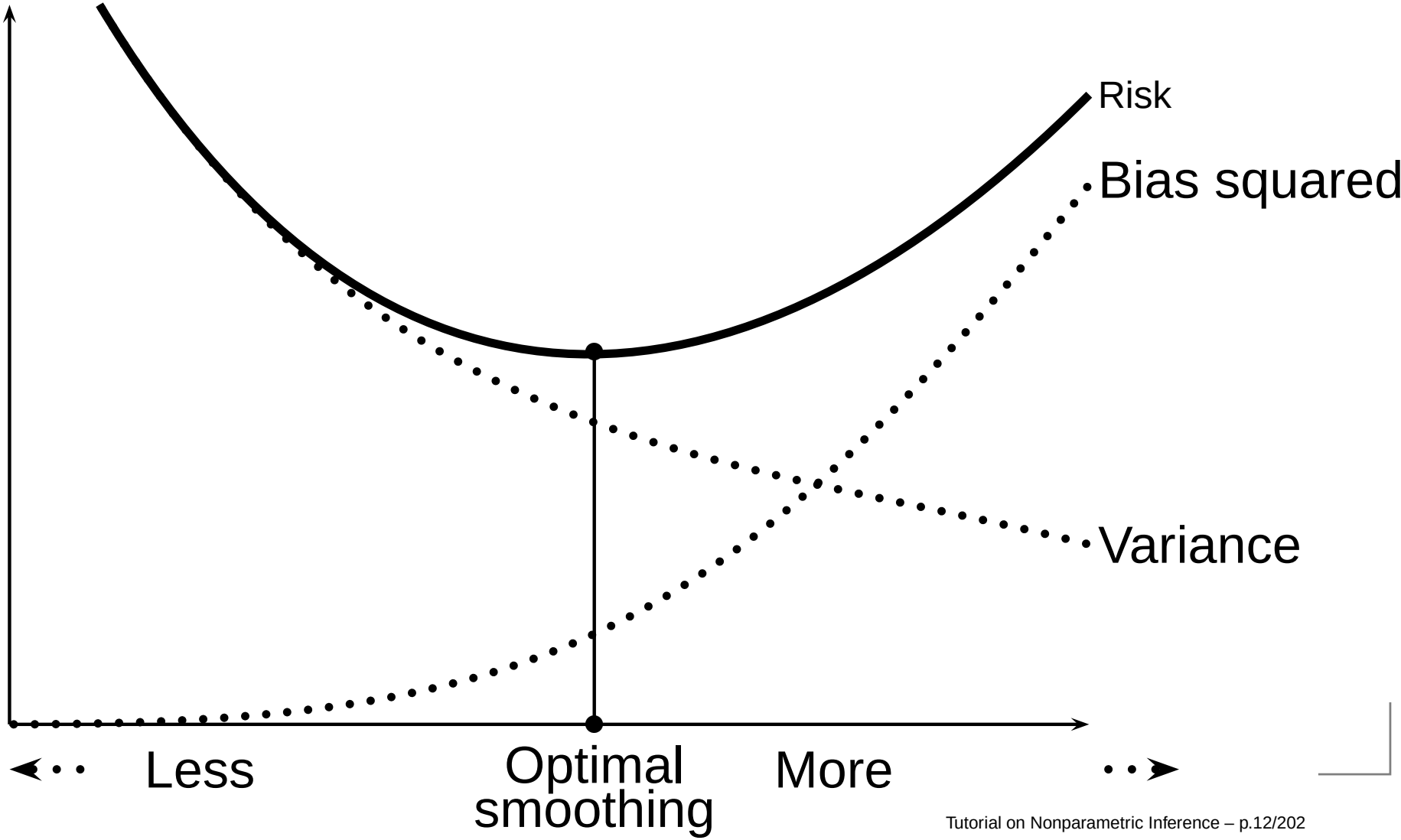
$$\text{bias}_x = \mathbb{E}(\widehat{f}_n(x)) - f(x).$$

$$\text{MSE} = \text{BIAS}^2 + \text{VARIANCE}.$$

Average MSE:

$$\int R(f(x), \widehat{f}_n(x))dx \quad \text{or} \quad \frac{1}{n}\sum_{i=1}^{n} R(f(x_i), \widehat{f}_n(x_i)).$$

The Bias–Variance Tradeoff



The Bias–Variance Tradeoff

For many smoothers:

$$\text{MSE} \approx c_1 h^4 + \frac{c_2}{nh}$$

which is minimized at

$$h = O\left(\frac{1}{n^{1/5}}\right)$$

Hence,

$$\text{MSE} = O\left(\frac{1}{n^{4/5}}\right)$$

whereas, for parametric problems

$$\text{MSE} = O\left(\frac{1}{n}\right)$$

Regression

Parametric Regression:

$$
\begin{aligned}
Y &= \beta_0 + \beta_1 X + \epsilon \\
Y &= \beta_0 + \beta_1 X_1 + \cdots + \beta_d X_d + \epsilon \\
Y &= \beta_0 e^{\beta_1 X_1} + \frac{\beta_2}{X_2} + \epsilon
\end{aligned}
$$

Nonparametric Regression:

$$
\begin{aligned}
Y &= f(X) + \epsilon \\
Y &= f_1(X_1) + \cdots + f_d(X_d) + \epsilon \\
Y &= f(X_1, X_2) + \epsilon
\end{aligned}
$$

Regression

Methods:

- Binning
- Local averaging
- Kernels
- Local polynomials
- Splines
- Wavelets

Regression

All (except wavelets) are **linear smoothers**:

$$\widehat{f}(x) = \sum_i Y_i \ell_i(x)$$

for some weights:

$$\ell_1(x), \ldots, \ell_n(x)$$

The vector of weights depends on the target point x.
Each method has a smoothing parameter h.

Linear Smoothers

If $\widehat{f}(x) = \sum_{i=1}^{n} \ell_i(x) Y_i$ then

$$\underbrace{\begin{pmatrix} \widehat{Y}_1 \\ \vdots \\ \widehat{Y}_n \end{pmatrix}}_{\widehat{Y}} \equiv \begin{pmatrix} \widehat{f}(X_1) \\ \vdots \\ \widehat{f}(X_n) \end{pmatrix} = \underbrace{\begin{pmatrix} \ell_1(X_1) & \ell_2(X_1) & \cdots & \ell_n(X_1) \\ \ell_1(X_2) & \ell_2(X_2) & \cdots & \ell_n(X_2) \\ \vdots & \vdots & \vdots & \vdots \\ \ell_1(X_n) & \ell_2(X_n) & \cdots & \ell_n(X_n) \end{pmatrix}}_{L} \underbrace{\begin{pmatrix} Y_1 \\ \vdots \\ Y_n \end{pmatrix}}_{Y}$$

$$\widehat{Y} = L\ Y$$

The **effective degrees of freedom** is:

$$\nu = \text{trace}(L) = \sum_{i=1}^{n} L_{ii}.$$

Binning

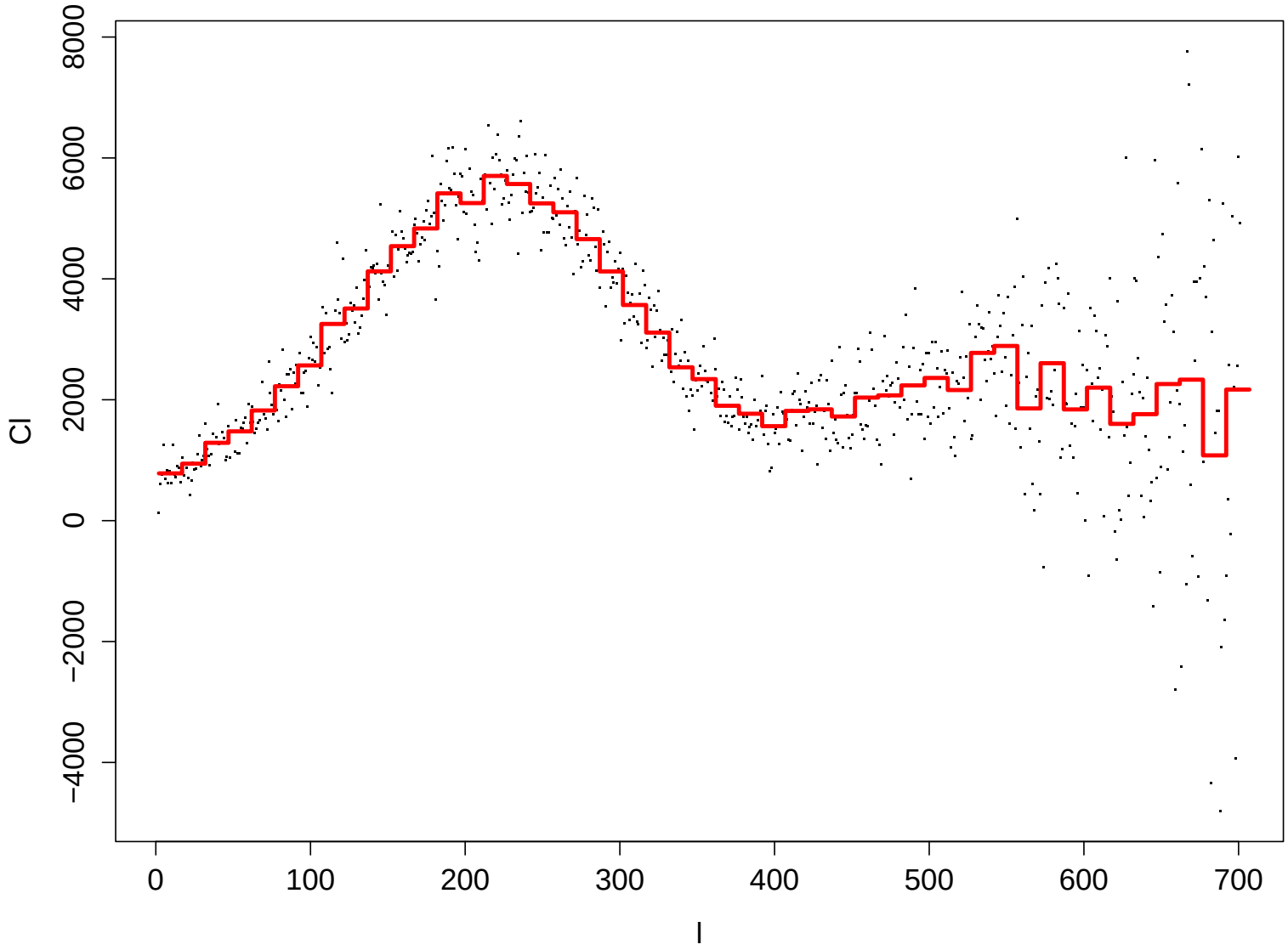
Divide the x-axis into bins $B_1, B_2, \ldots$ of width h. $\widehat{f}$ is a step function based on averaging the Y_i's in each bin:

$$\text{for } x \in B_j: \quad \widehat{f}(x) = \text{mean}\left\{ Y_i : \ X_i \in B_j \right\}.$$

The (arbitrary) choice of the boundaries of the bins can affect inference, especially when h large.

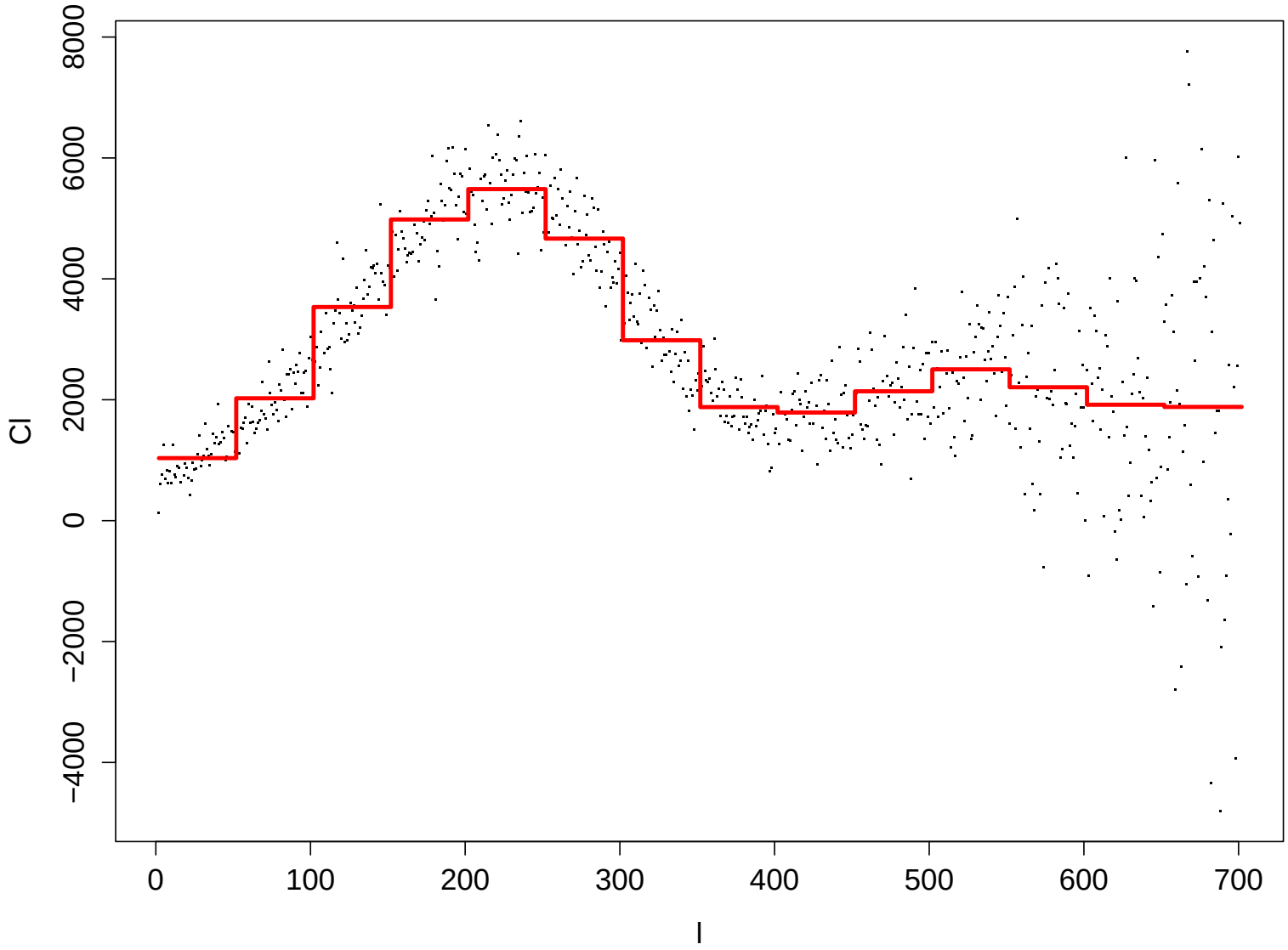
Binning

WMAP data, binned estimate with $h = 15$.



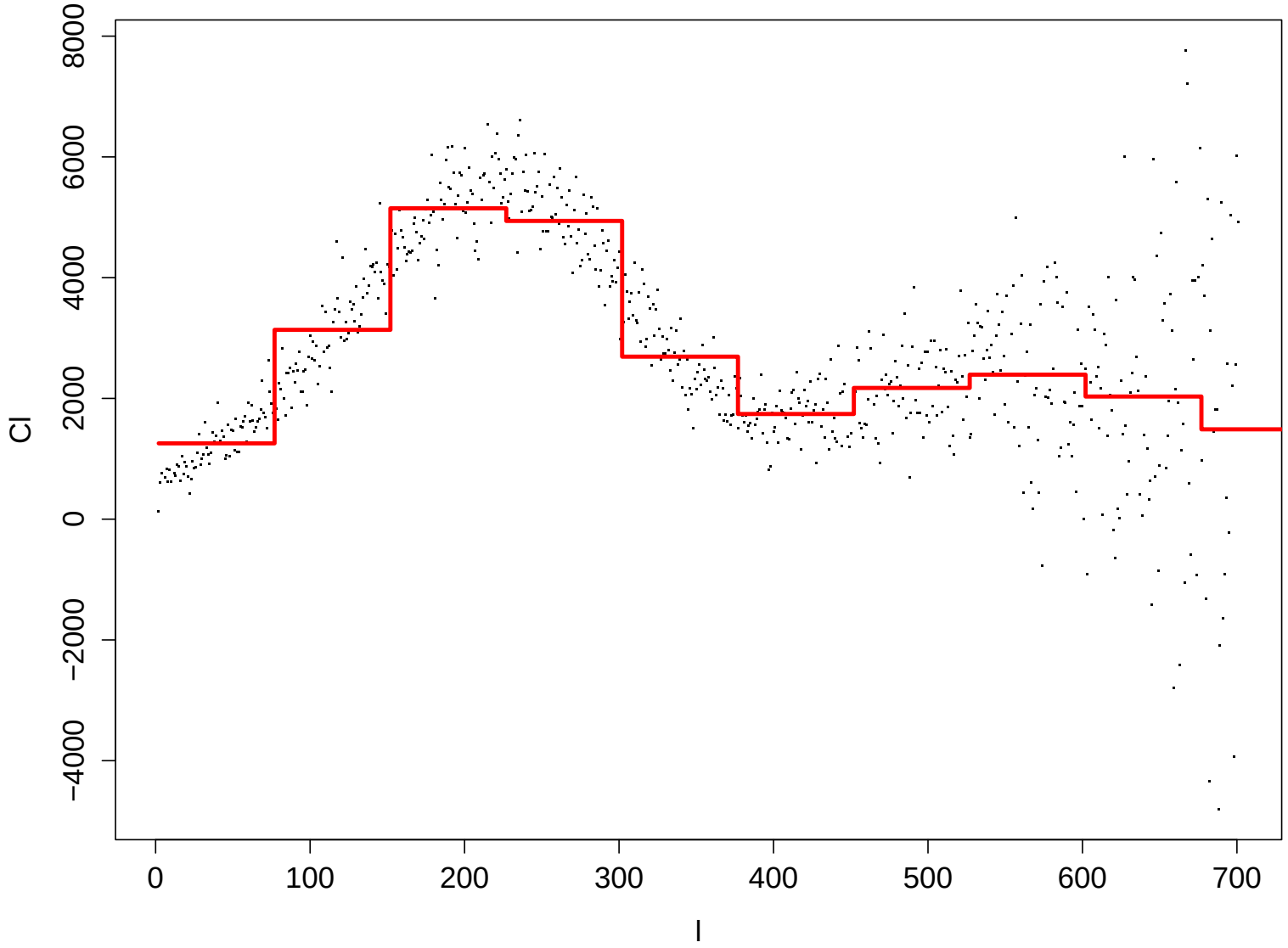
Binning

WMAP data, binned estimate with $h = 50$.



Binning

WMAP data, binned estimate with $h = 75$.



Local Averaging

For each x let

$$N_x = \left[x - \frac{h}{2},\ x + \frac{h}{2}\right].$$

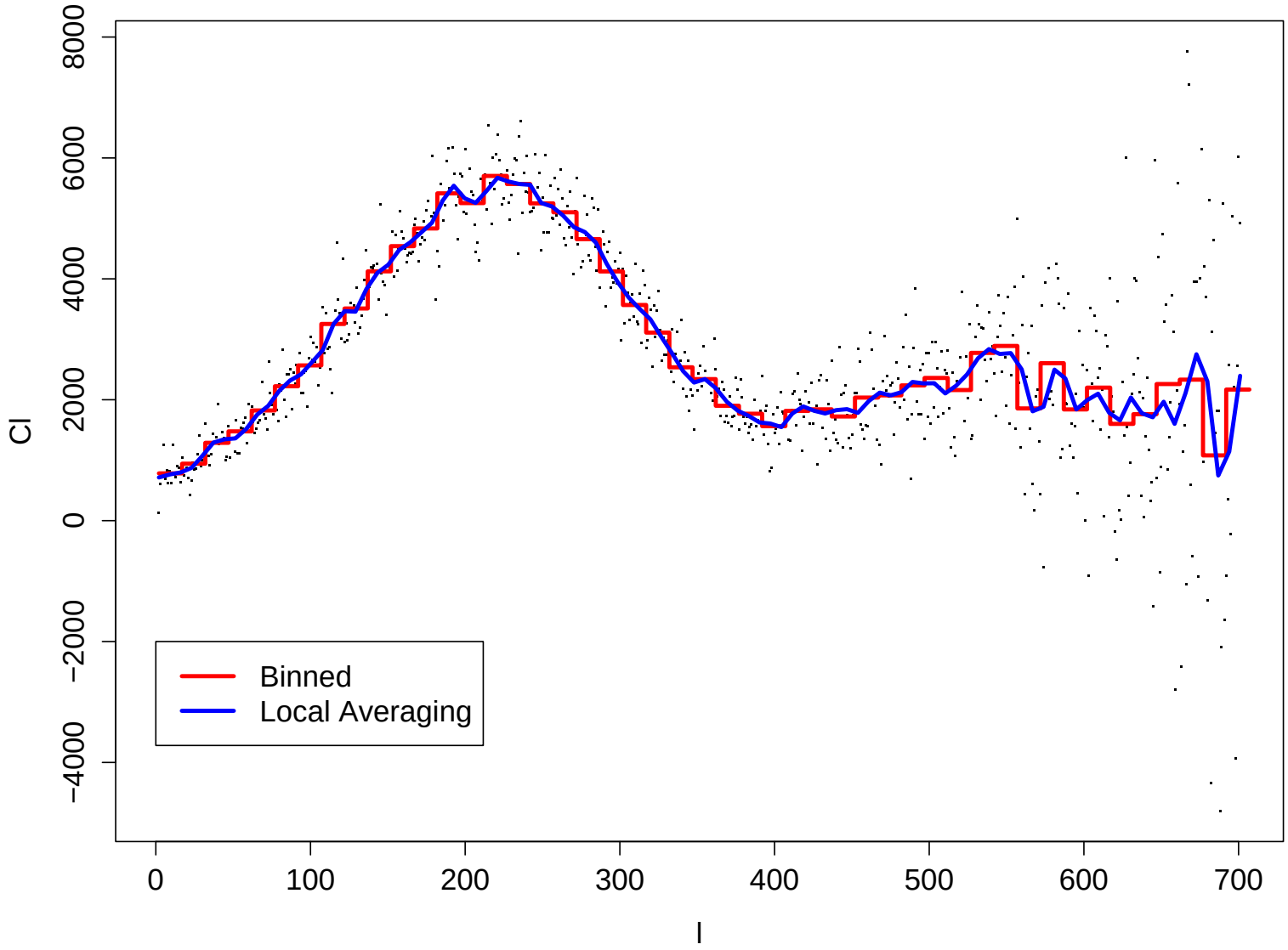
This is a moving window of length h, centered at x. Define

$$\widehat{f}(x) = \text{mean}\left\{Y_i :\ X_i \in N_x\right\}.$$

This is like binning but removes the arbitrary boundaries.

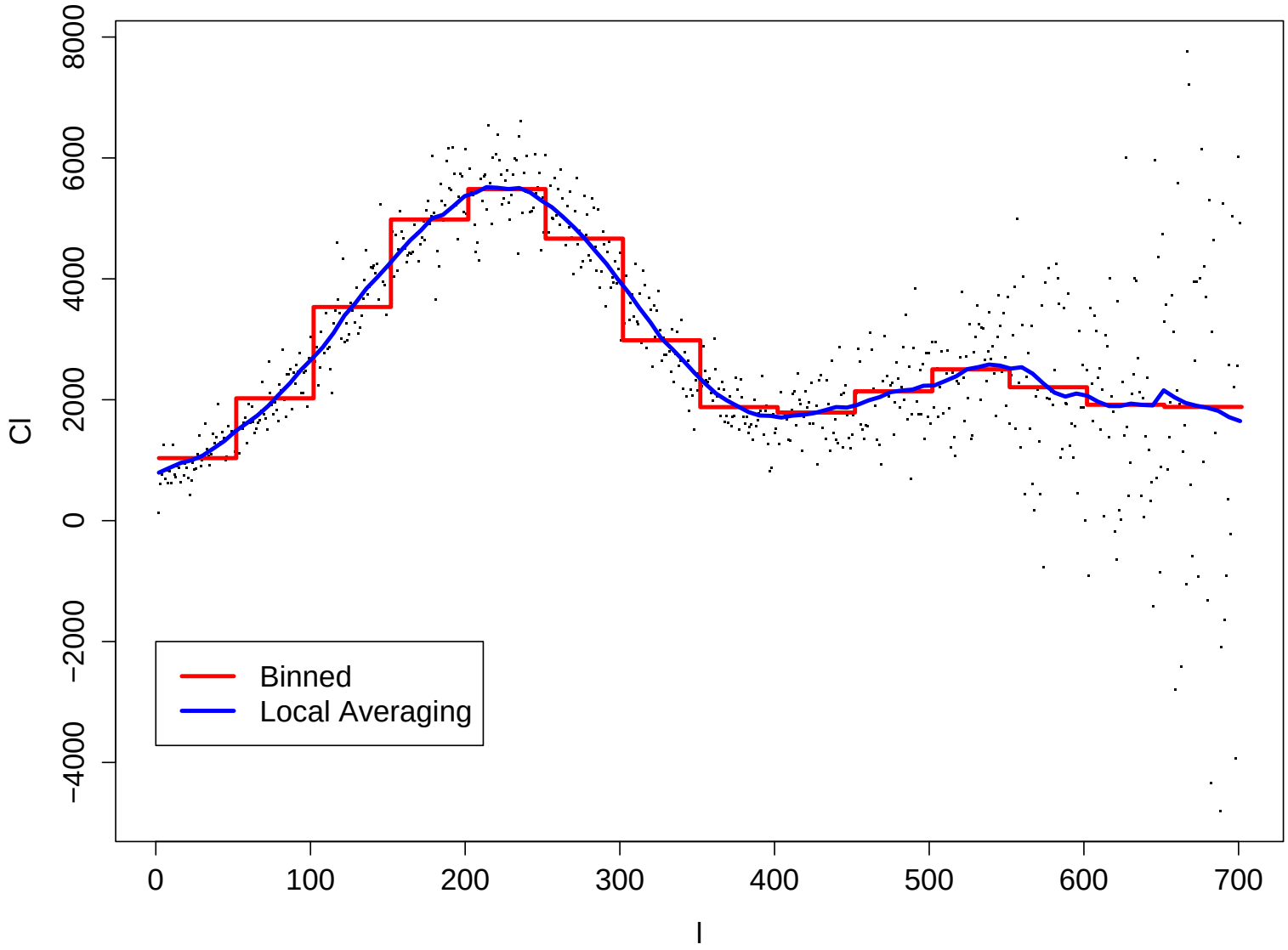
Local Averaging

WMAP data, local average estimate with $h = 15$.



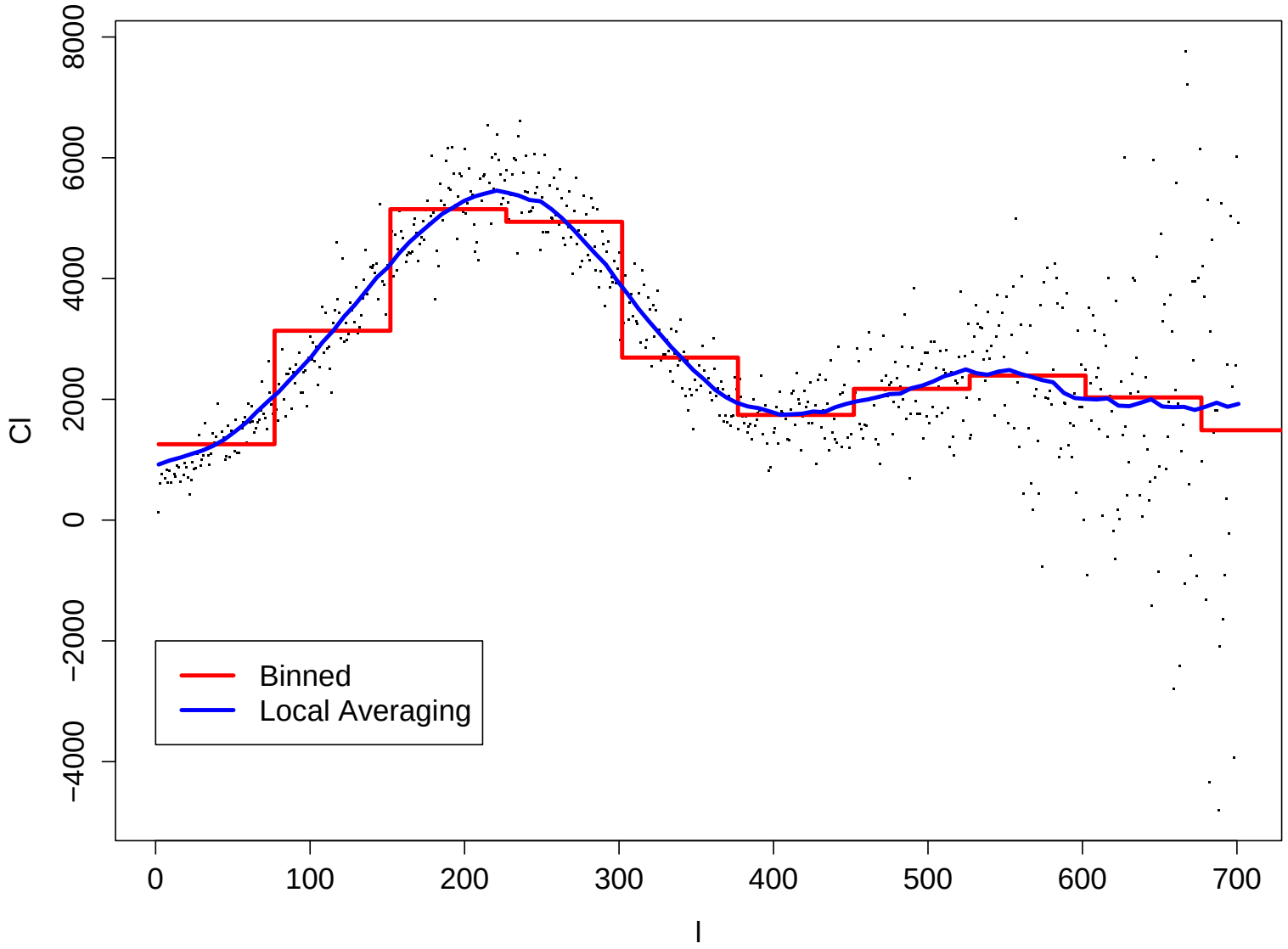
Local Averaging

WMAP data, local average estimate with $h = 50$.



Local Averaging

WMAP data, local average estimate with $h = 75$.



Kernel Regression

The local average estimator can be written:

$$\widehat{f}(x) = \frac{\sum_{i=1}^{n} Y_i \; K\left(\frac{x - X_i}{h}\right)}{\sum_{i=1}^{n} K\left(\frac{x - X_i}{h}\right)}$$

where

$$K(x) = \begin{cases} 1 & |x| < 1/2 \\ 0 & \text{otherwise.} \end{cases}$$

Can improve this by using a function K which is smoother.

Kernels

A **kernel** is any smooth function K such that $K(x) \geq 0$ and

$$\int K(x)\,dx = 1, \quad \int xK(x)dx = 0 \quad \text{and} \quad \sigma_K^2 \equiv \int x^2 K(x)dx > 0.$$

Some commonly used kernels are the following:

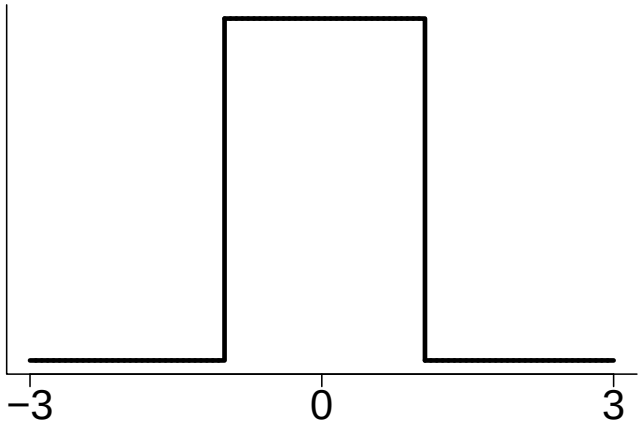
$$\textbf{the boxcar kernel}: \qquad K(x) = \frac{1}{2} I(|x| < 1),$$

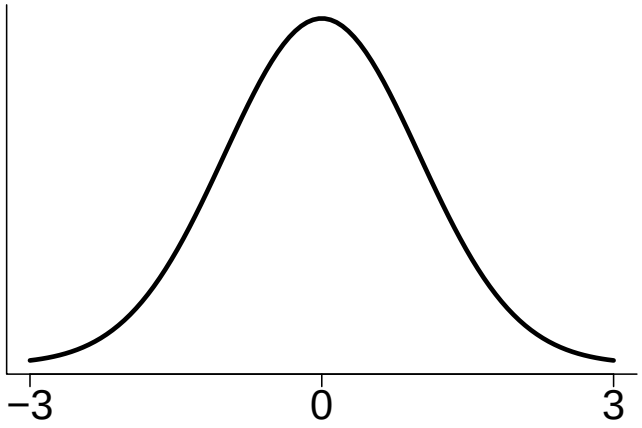
$$\textbf{the Gaussian kernel}: \qquad K(x) = \frac{1}{\sqrt{2\pi}} e^{-x^2/2},$$

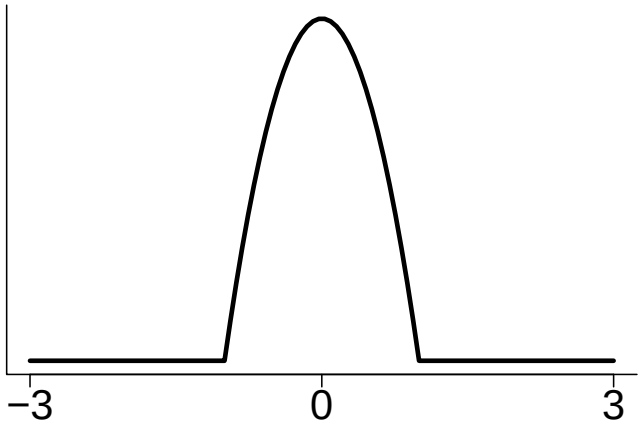
$$\textbf{the Epanechnikov kernel}: \qquad K(x) = \frac{3}{4}(1 - x^2) I(|x| < 1)$$

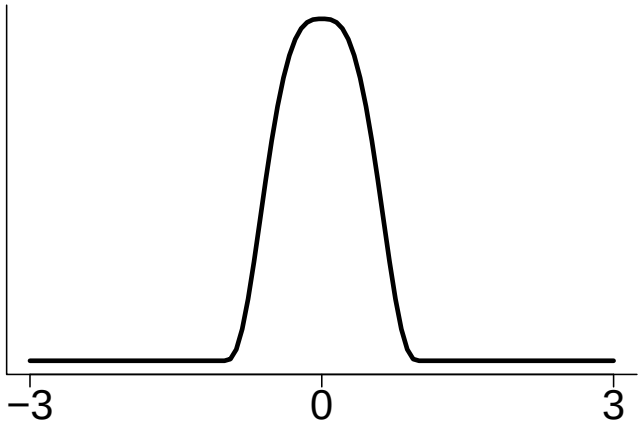
$$\textbf{the tricube kernel}: \qquad K(x) = \frac{70}{81}(1 - |x|^3)^3 I(|x| < 1).$$

Kernels









Kernel Regression

$$\widehat{f}(x) = \frac{\sum_{i=1}^{n} Y_i \; K\left(\frac{x-X_i}{h}\right)}{\sum_{i=1}^{n} K\left(\frac{x-X_i}{h}\right)}$$

We can write this as

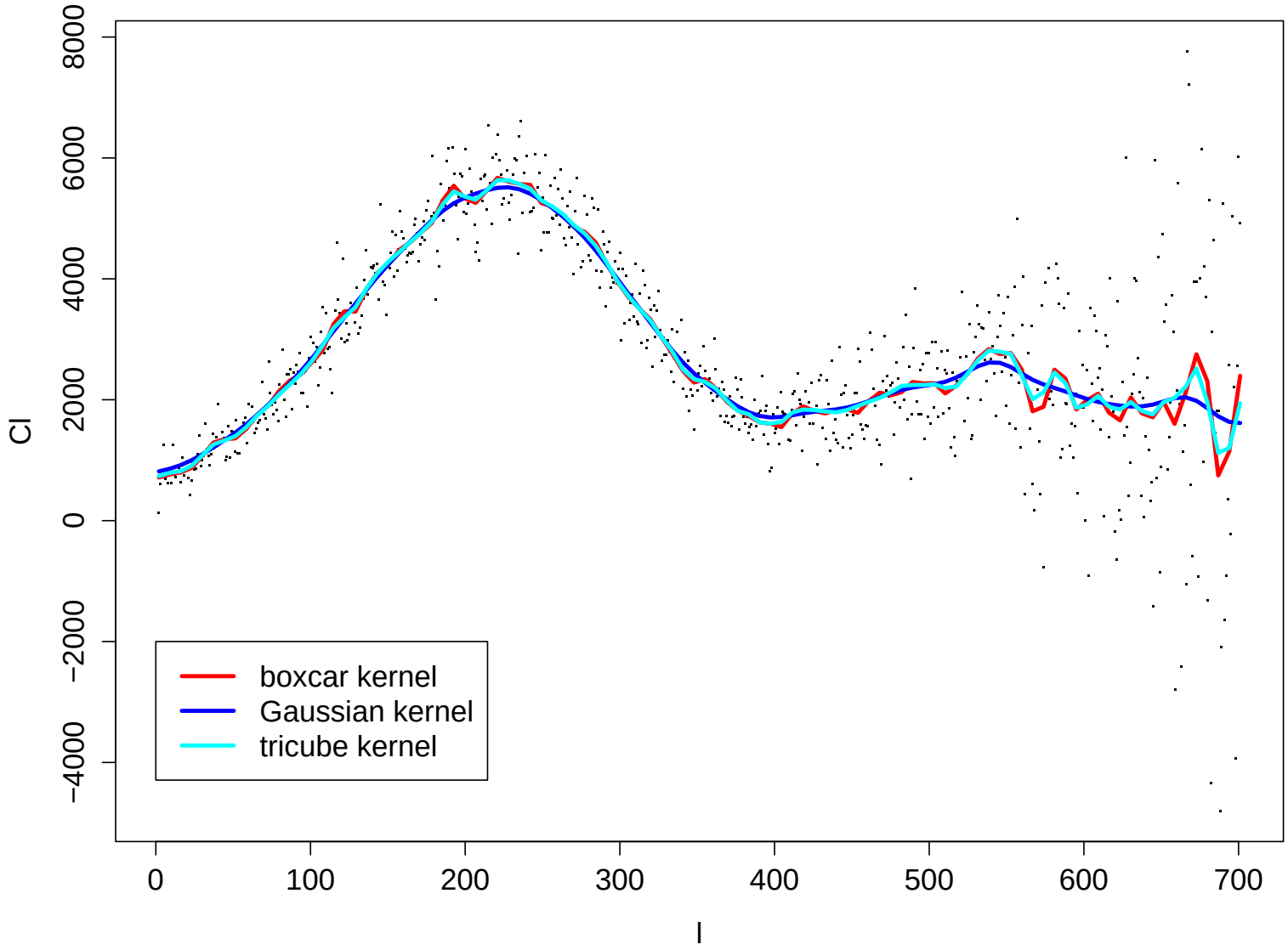
$$\widehat{f}(x) = \sum_i Y_i \; \ell_i(x)$$

where

$$\ell_i(x) = \frac{K\left(\frac{x-X_i}{h}\right)}{\sum_{i=1}^{n} K\left(\frac{x-X_i}{h}\right)}.$$

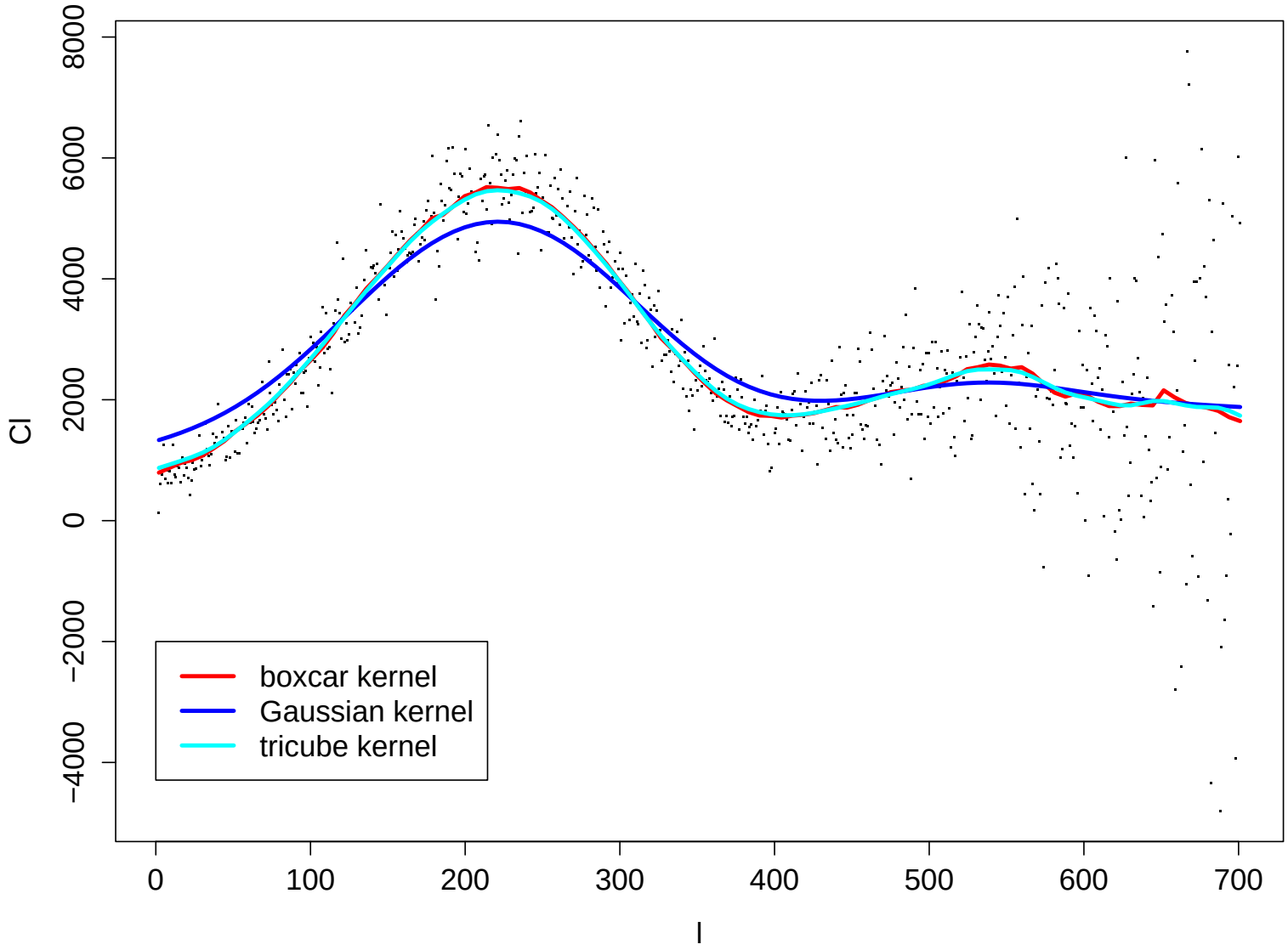
Kernel Regression

WMAP data, kernel regression estimates, $h = 15$.



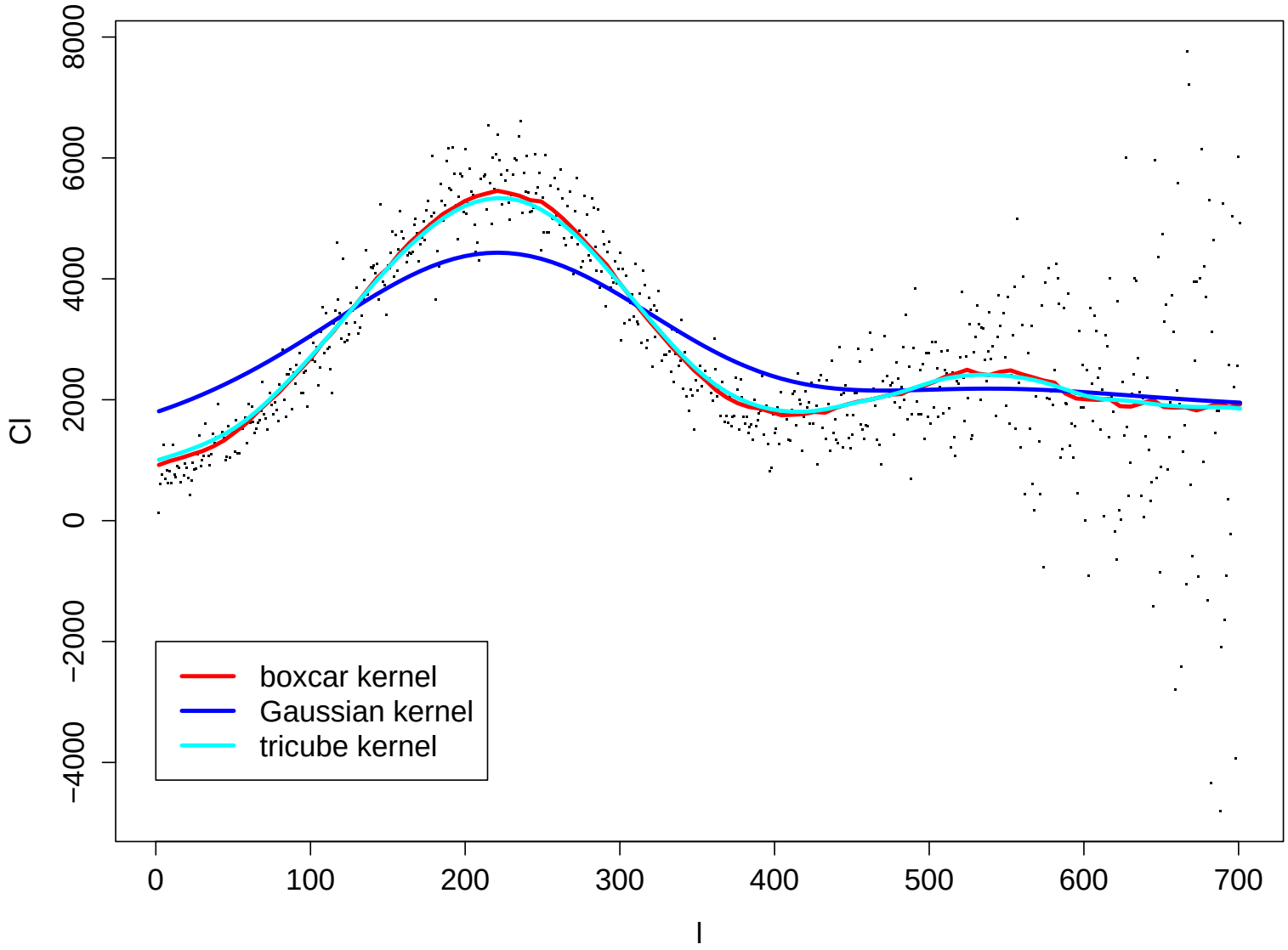
Kernel Regression

WMAP data, kernel regression estimates, $h = 50$.



Kernel Regression

WMAP data, kernel regression estimates, $h = 75$.



Kernel Regression

$$\text{MSE} \approx \frac{h^4}{4}\left(\int x^2 K(x)dx\right)^2 \int \left(f''(x) + 2f'(x)\frac{g'(x)}{g(x)}\right)^2 dx + \frac{\sigma^2 \int K^2(x)dx}{nh}\int \frac{1}{g(x)}dx.$$

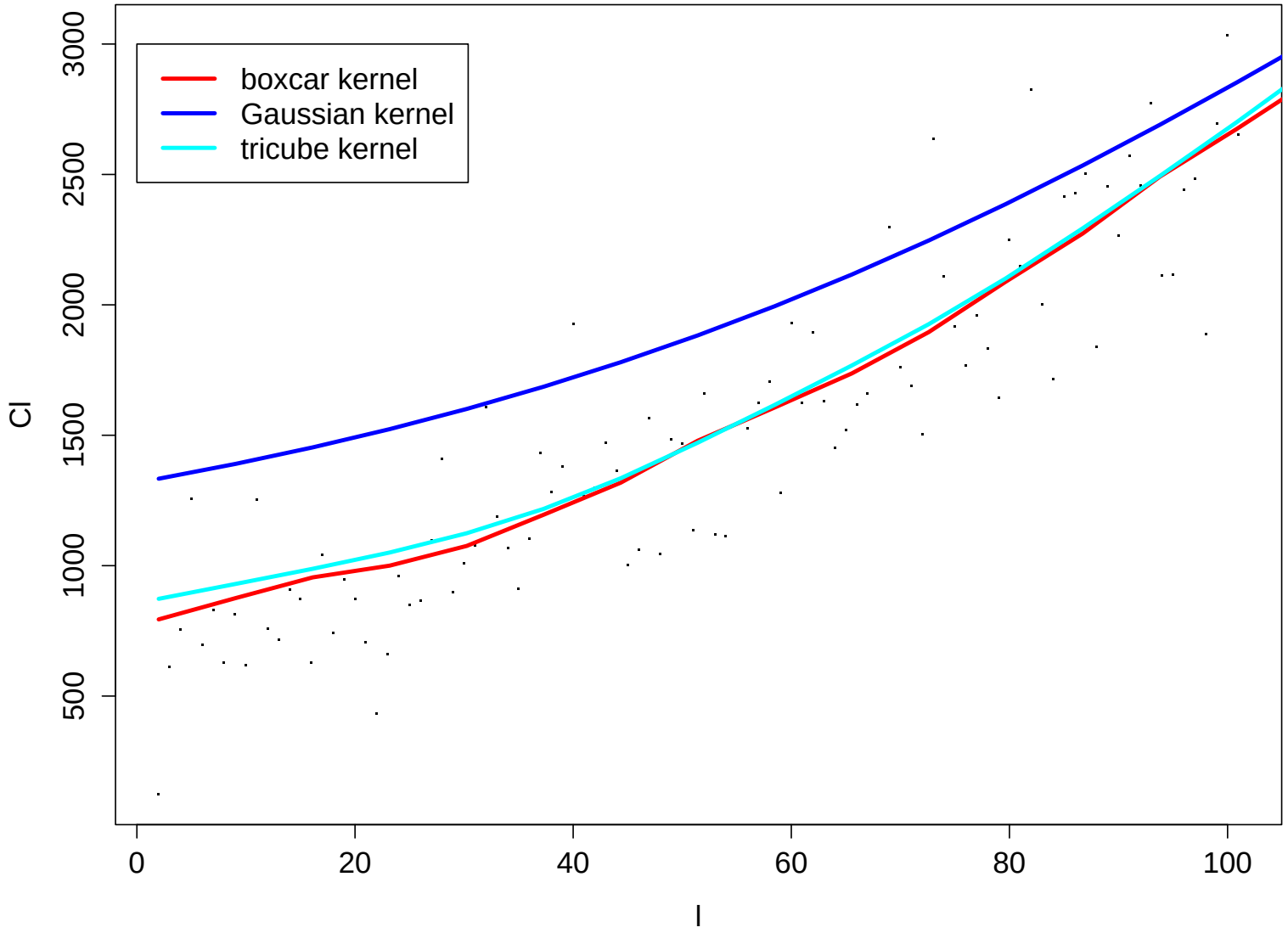
where $g(x)$ is the density for X.
What is especially notable is the presence of the term

$$2f'(x)\frac{g'(x)}{g(x)} = \text{design bias}.$$

Also, bias is large near the boundary. We can reduce these biases using local polynomials.

Kernel Regression

WMAP data, $h = 50$. Note the boundary bias.



Local Polynomial Regression

Recall polynomial regression:

$$\widehat{f}(x) = \widehat{\beta}_0 + \widehat{\beta}_1 x + \widehat{\beta}_2 x^2 + \cdots + \widehat{\beta}_p x^p$$

where $\widehat{\beta} = (\widehat{\beta}_0, \ldots, \widehat{\beta}_p)$ are obtained by least squares:

$$\text{minimize} \sum_{i=1}^{n} \Big(Y_i - [\beta_0 + \beta_1 x + \beta_2 x^2 + \cdots + \beta_p x^p] \Big)^2$$

Local Polynomial Regression

Local polynomial regression: approximate $f(x)$ locally by a **different** polynomial for every x:

$$f(u) \approx \beta_0(x) + \beta_1(x)(u-x) + \beta_2(x)(u-x)^2 + \cdots + \beta_p(x)(u-x)^p$$

for u near x. Estimate $(\widehat{\beta}_0(x), \ldots, \widehat{\beta}_p(x))$ by **local least squares**: minimize

$$\sum_{i=1}^{n} (Y_i - [\beta_0(x) + \beta_1(x)x + \beta_2(x)x^2 + \cdots + \beta_p(x)x^p])^2 \underbrace{K\left(\frac{x - X_i}{h}\right)}_{\text{kernel}}$$

$$\widehat{f}(x) = \widehat{\beta}_0(x)$$

Local Polynomial Regression

Taking $p = 0$ yields the **kernel regression estimator:**

$$\widehat{f}_n(x) = \sum_{i=1}^{n} \ell_i(x) Y_i$$

$$\ell_i(x) = \frac{K\left(\frac{x-x_i}{h}\right)}{\sum_{j=1}^{n} K\left(\frac{x-x_j}{h}\right)}.$$

Taking $p = 1$ yields the **local linear estimator. This is the best, all-purpose smoother.**

Choice of Kernel K: not important

Choice of **bandwidth** h: crucial

Local Polynomial Regression

The local polynomial regression estimate is

$$\widehat{f}_n(x) = \sum_{i=1}^{n} \ell_i(x) Y_i$$

where $\ell(x)^T = (\ell_1(x), \ldots, \ell_n(x))$,

$$\ell(x)^T = e_1^T (X_x^T W_x X_x)^{-1} X_x^T W_x,$$

$e_1 = (1, 0, \ldots, 0)^T$ and X_x and W_x are defined by

$$X_x = \begin{pmatrix} 1 & X_1 - x & \cdots & \frac{(X_1-x)^p}{p!} \\ 1 & X_2 - x & \cdots & \frac{(X_2-x)^p}{p!} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & X_n - x & \cdots & \frac{(X_n-x)^p}{p!} \end{pmatrix} \quad W_x = \begin{pmatrix} K\left(\frac{x-X_1}{h}\right) & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & K\left(\frac{x-X_n}{h}\right) \end{pmatrix}.$$

Local Polynomial Regression

Note that $\widehat{f}(x) = \sum_{i=1}^n \ell_i(x) Y_i$ is a **linear smoother.** Define $\widehat{Y} = (\widehat{Y}_1, \ldots, \widehat{Y}_n)$ where $\widehat{Y}_i = \widehat{f}(X_i)$. Then

$$\widehat{Y} = LY$$

where L is the **smoothing matrix**:

$$L = \begin{pmatrix} \ell_1(X_1) & \ell_2(X_1) & \cdots & \ell_n(X_1) \\ \ell_1(X_2) & \ell_2(X_2) & \cdots & \ell_n(X_2) \\ \vdots & \vdots & \vdots & \vdots \\ \ell_1(X_n) & \ell_2(X_n) & \cdots & \ell_n(X_n) \end{pmatrix}.$$

The **effective degrees of freedom** is:

$$\nu = \text{trace}(L) = \sum_{i=1}^n L_{ii}.$$

Choosing the Bandwidth

Estimate the risk

$$\frac{1}{n}\sum_{i=1}^{n}\mathbb{E}(\widehat{f}(X_i)-f(X_i))^2$$

with the **leave-one-out cross-validation score**:

$$CV=\frac{1}{n}\sum_{i=1}^{n}(Y_i-\widehat{f}_{(-i)}(X_i))^2$$

where $\widehat{f}_{(-i)}$ is the estimator obtained by omitting the i^{th} pair (X_i,Y_i).

Choosing the Bandwidth

Amazing shortcut formula:

$$CV = \frac{1}{n}\sum_{i=1}^{n}\left(\frac{Y_i - \widehat{f}_n(x_i)}{1 - L_{ii}}\right)^2 .$$

An commonly used approximation is GCV (generalized cross-validation):

$$GCV = \frac{1}{n\left(1 - \frac{\nu}{n}\right)^2}\sum_{i=1}^{n}(Y_i - \widehat{f}(X_i))^2$$

$$\nu = \text{trace}(L).$$

Theoretical Aside

Why local linear ($p = 1$) is better than kernel ($p = 0$). Both have (approximate) variance

$$\frac{\sigma^2(x)}{g(x)nh} \int K^2(u)du$$

The kernel estimator has bias

$$h^2 \left(\frac{1}{2} f''(x) + \frac{f'(x)g'(x)}{g(x)} \right) \int u^2 K(u)du$$

whereas the local linear estimator has asymptotic bias

$$h^2 \frac{1}{2} f''(x) \int u^2 K(u)du$$

The local linear estimator is free from design bias. At the boundary points, the kernel estimator has asymptotic bias of $O(h)$ while the local linear estimator has bias $O(h^2)$.

Using `locfit()`

Need to include the `locfit` library:

```
> install.packages("locfit")
> library(locfit)
> result = locfit(y~x, alpha=c(0, 1.5),
deg=1)
```

`y` and `x` are vectors

the second argument to `alpha` gives the bandwidth (h)

the first argument to `alpha` specifies the **nearest neighbor fraction**, an alternative to the bandwidth

`fitted(result)` gives the **fitted values**, $\widehat{f}(X_i)$

`residuals(result)` gives the **residuals**, $Y_i - \widehat{f}(X_i)$

Using `locfit()`

See the R code `locfit_R_example`, the function `locfit_simdata()`.

Allows specification the true function $f(x)$, simulate data $Y_i = f(X_i) + \epsilon_i, i = 1, 2, \ldots, n$, where ϵ_i is normal(0,σ).

Illustrates the use of the function `gcvplot()`, which calculates GCV for specified bandwidths.

An Aside: Functions in R

```
locfit_simdata = function(f,sigma,n,h,deg,xlo,xhi)
{

# GENERATE THE DATA

   x = runif(n,xlo,xhi)
   y = eval(f) + rnorm(n,sd=sigma)


# FIT THE MODEL USING LOCFIT

   locfitfit = locfit(y~x,alpha=c(0,h),deg=deg,maxk=1000)
```

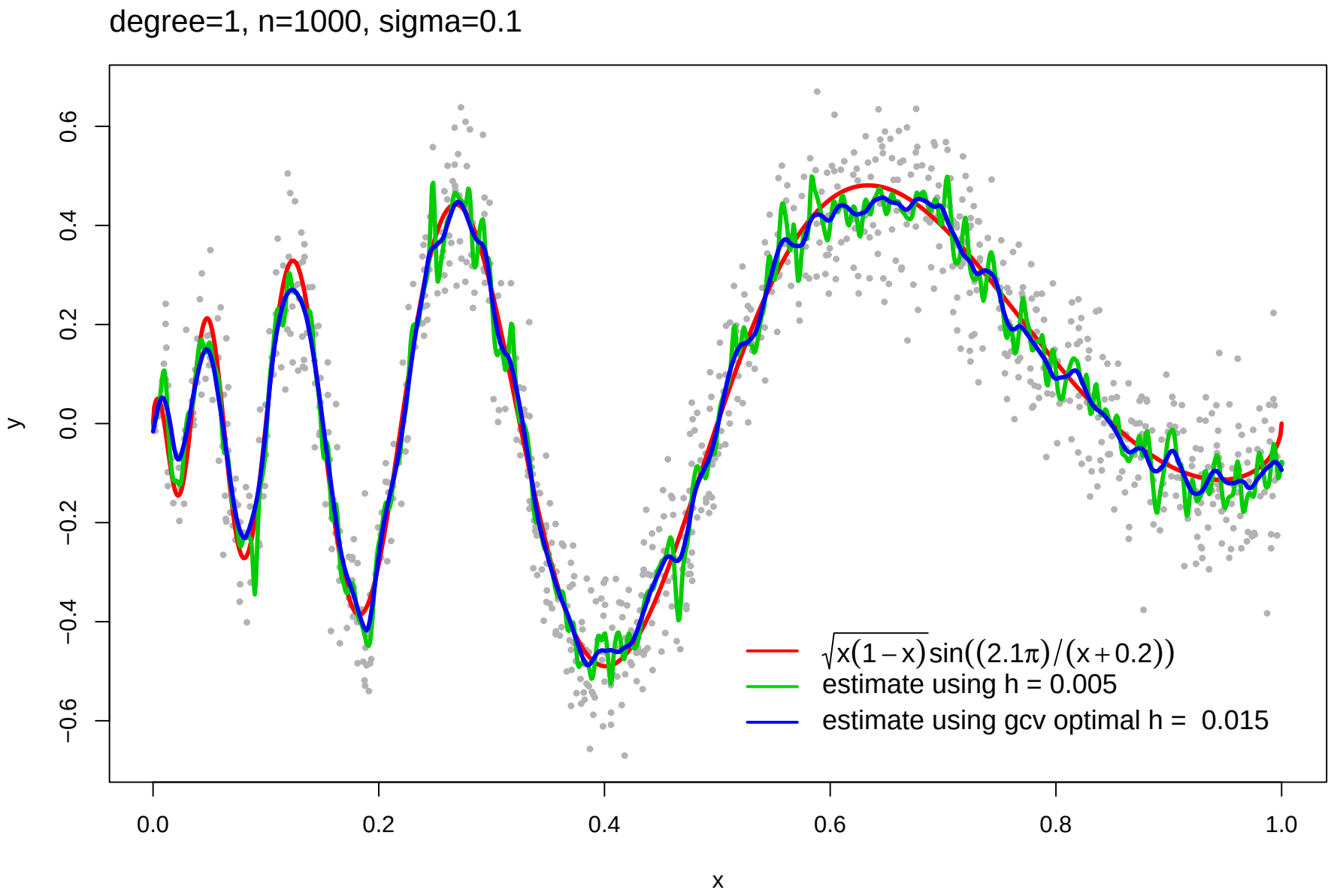
An Aside: Functions in R

```
# USE GCV TO CHOOSE BANDWIDTH

   alphamat = matrix(0,ncol=2,nrow=30)
   alphamat[,2] = (1.2^(seq(30)-30))*2*(xhi-xlo)
   gcvs = gcvplot(y~x,alpha=alphamat,deg=deg,maxk=1000)
   optband = max(gcvs$alpha[gcvs$values == min(gcvs$values),2])
   locfitopt = locfit(y~x,alpha=c(0,optband),deg=deg,maxk=1000)
```

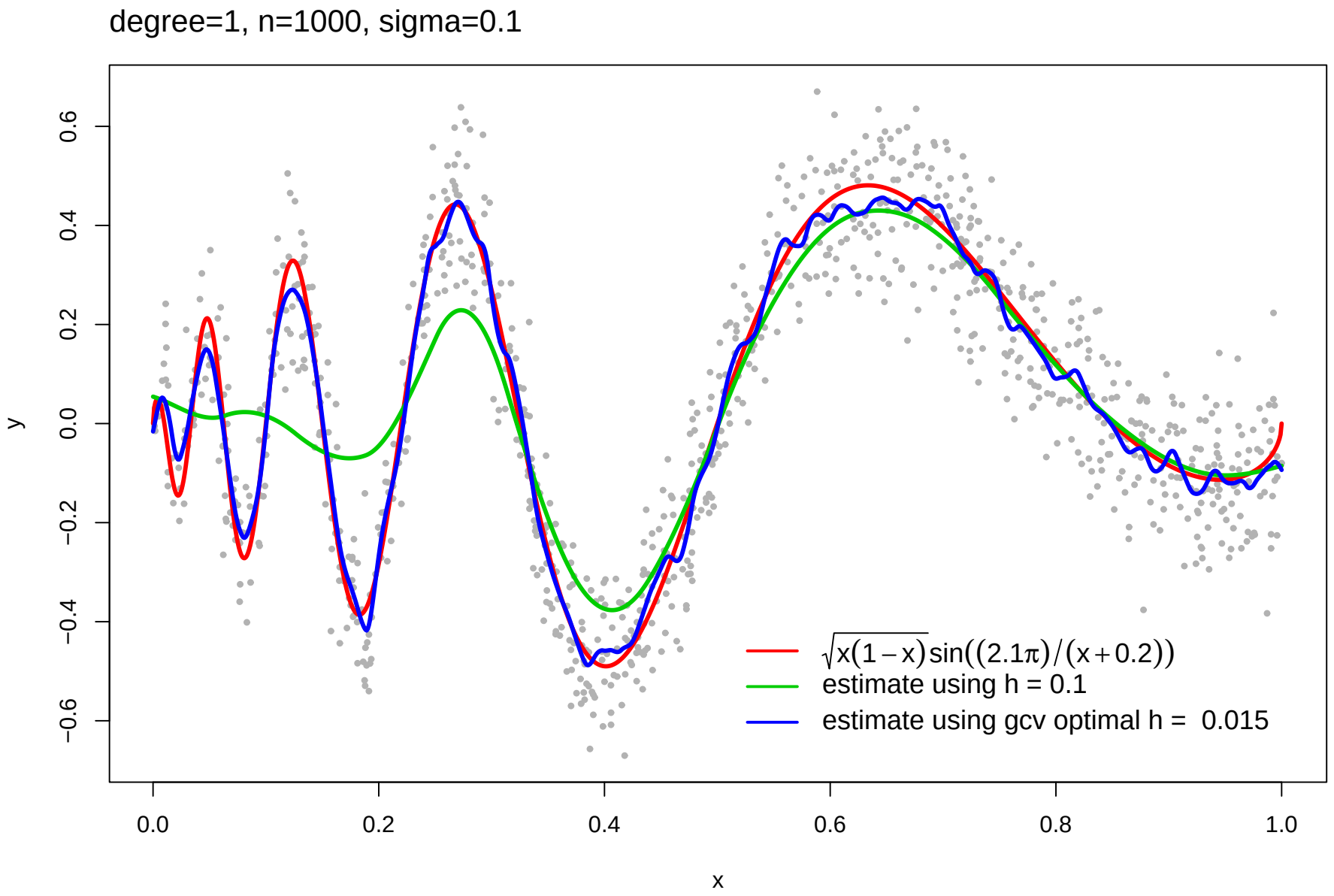
An Aside: Functions in R

```
# MAKE PLOTS

   xg = seq(xlo,xhi,length=1000)
   plot(x,y,xlab="x",ylab="y",pch=16,cex=0.5,
      col=rgb(0.7,0.7,0.7))
   lines(xg,eval(f,list(x=xg)),col=2,lwd=3)
   lines(xg,predict(locfitfit,newdata=xg),col=3,lwd=3)
   lines(xg,predict(locfitopt,newdata=xg),col=4,lwd=3)
   legend(xhi,min(y),legend=c(f,paste("estimate using h =",h),
      paste("estimate using gcv optimal h = ",round(optband,3))),
      col=c(2,3,4),lwd=2,bty="n",cex=1.2,yjust=0,xjust=1)
   mtext(paste("degree=",deg,", n=",n,", sigma=",sigma,
      sep=""),line=1,adj=0,cex=1.3)
}
```

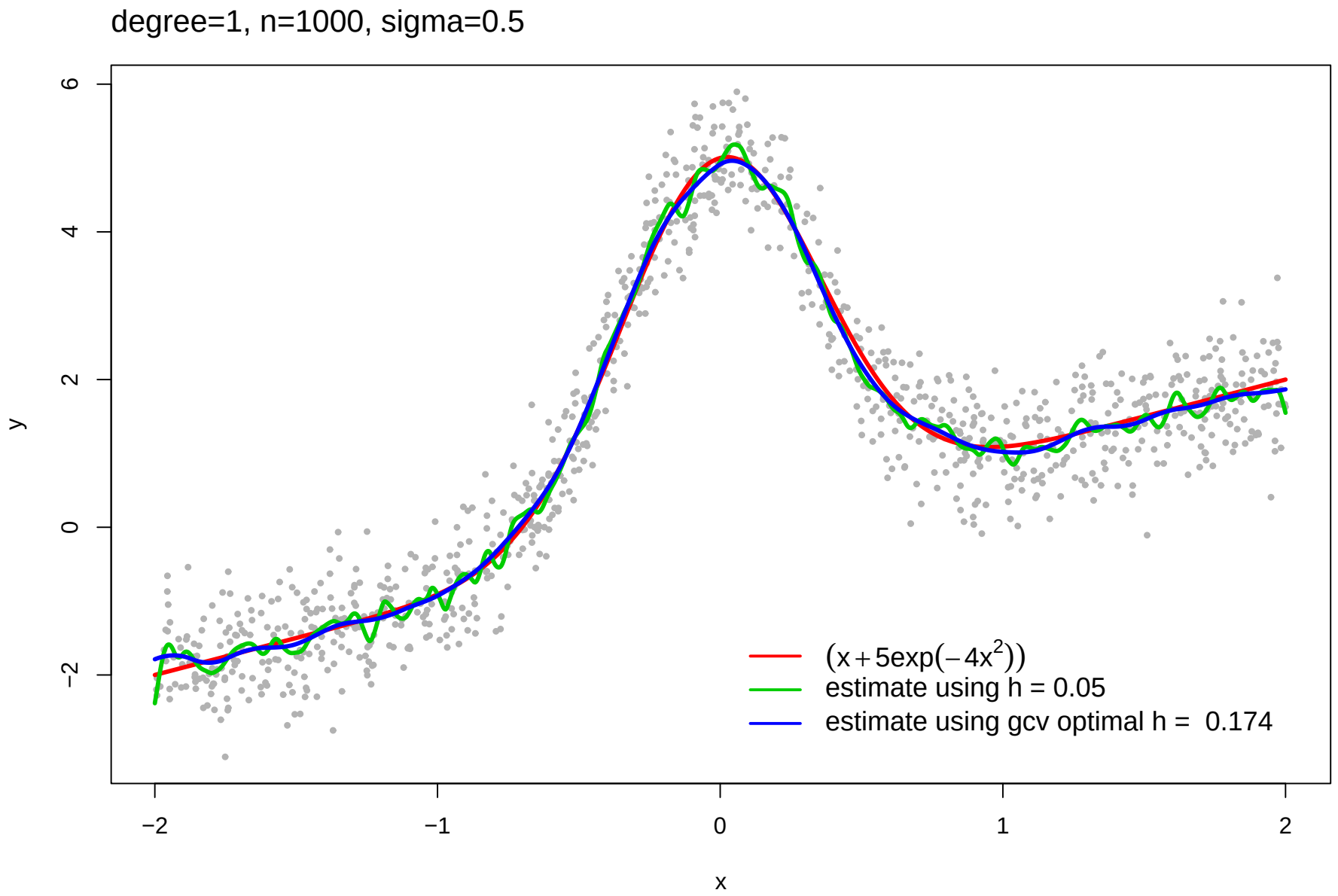
Using `locfit()`



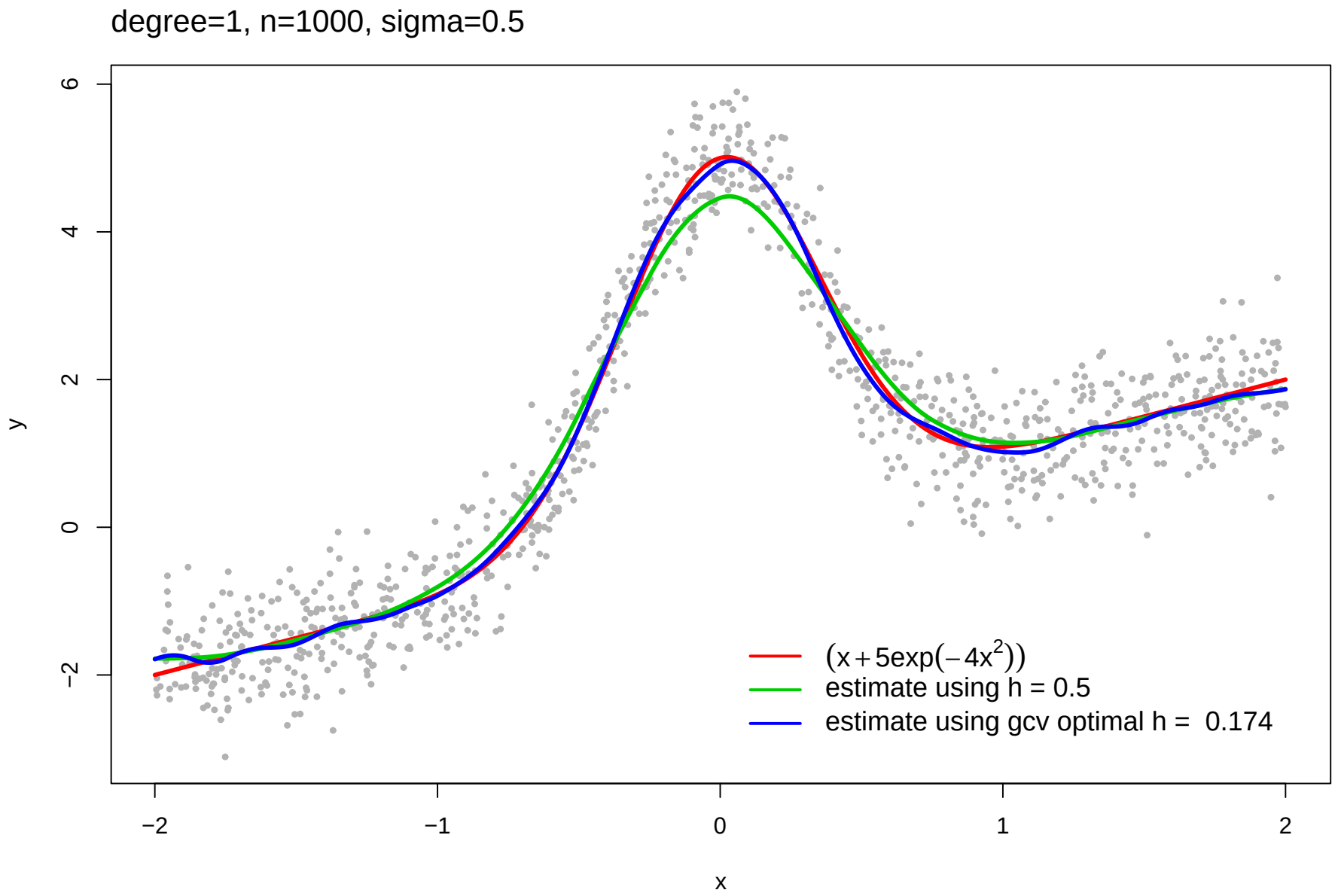
Using `locfit()`



Using `locfit()`

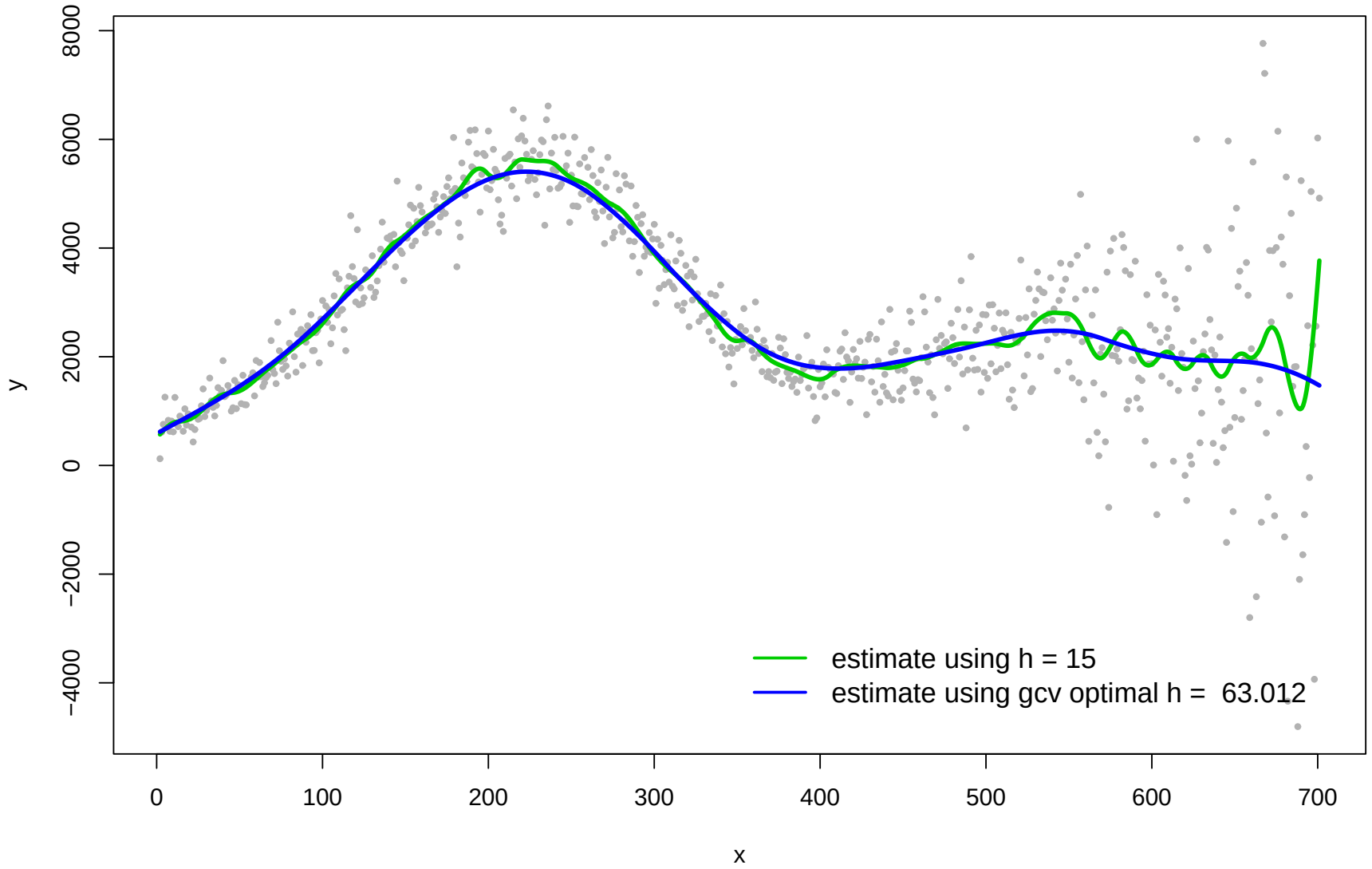


Using `locfit()`



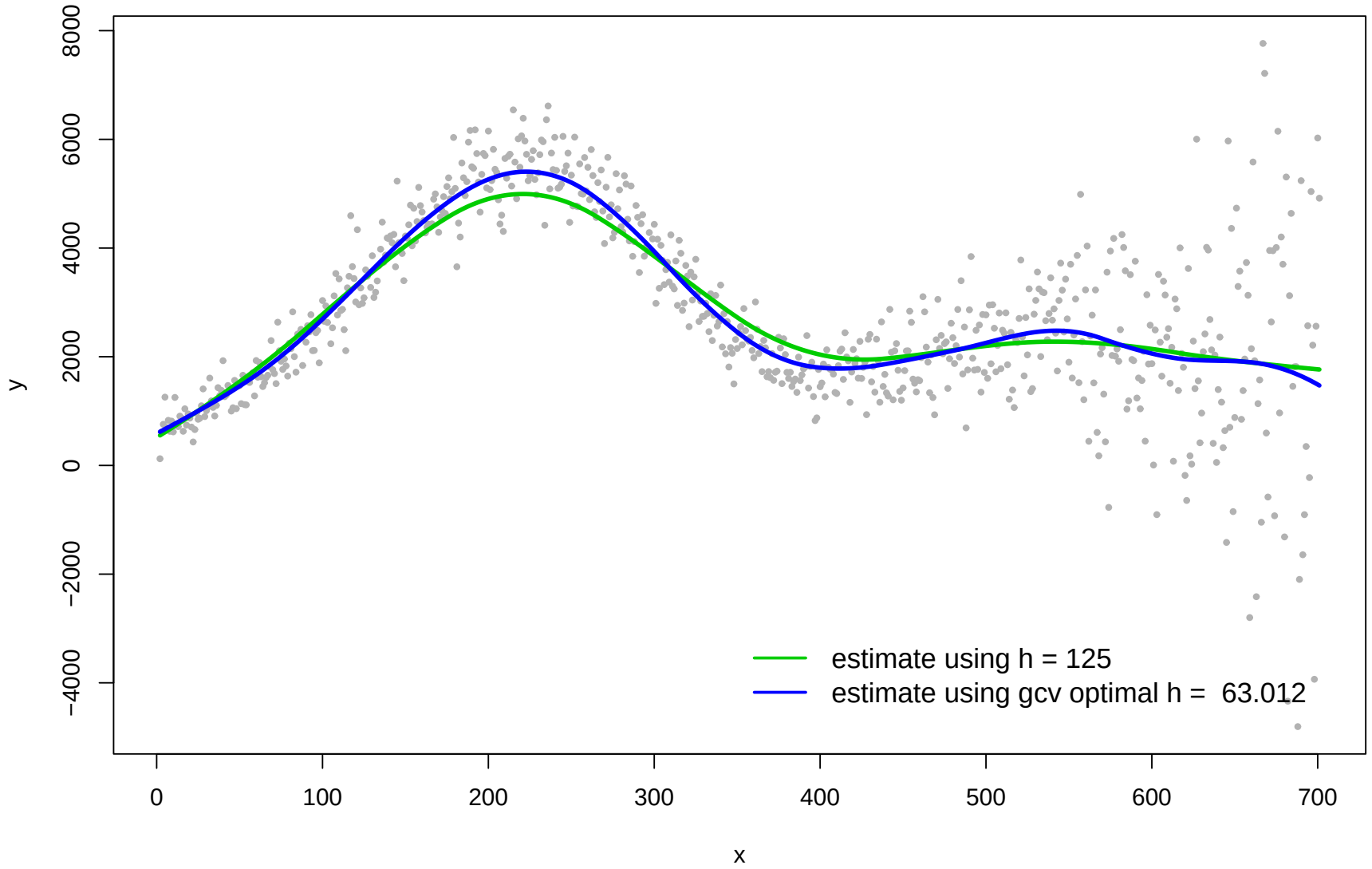
Example

WMAP data, local linear fit, $h = 15$.



Example

WMAP data, local linear fit, $h = 125$.



Variance Estimation

Let

$$\widehat{\sigma}^2 = \frac{\sum_{i=1}^n (Y_i - \widehat{f}(x_i))^2}{n - 2\nu + \widetilde{\nu}}$$

where

$$\nu = \mathrm{tr}(L), \quad \widetilde{\nu} = \mathrm{tr}(L^T L) = \sum_{i=1}^n ||\ell(x_i)||^2.$$

If f is sufficiently smooth, then $\widehat{\sigma}^2$ is a consistent estimator of σ^2.

Variance Estimation

For the WMAP data, using local linear fit.

```
> wmap = read.table("wmap.dat",header=T)
> opth = 63.0

> locfitwmap = locfit(wmap$Cl[1:700]~wmap$ell[1:700],
   alpha=c(0,opth),deg=1)
> nu = as.numeric(locfitwmap$dp[6])
> nutilde = as.numeric(locfitwmap$dp[7])

> sigmasqrhat = sum(residuals(locfitwmap)^2)/(700-2*nu+nutilde)
> sigmasqrhat
[1] 1122214
```

But, does not seem reasonable to assume homoscedasticity...

Variance Estimation

Allow σ to be a function of x:

$$Y_i = f(x_i) + \sigma(x_i)\epsilon_i.$$

Let $Z_i = \log(Y_i - f(x_i))^2$ and $\delta_i = \log \epsilon_i^2$. Then,

$$Z_i = \log(\sigma^2(x_i)) + \delta_i.$$

This suggests estimating $\log \sigma^2(x)$ by regressing the log squared residuals on x.

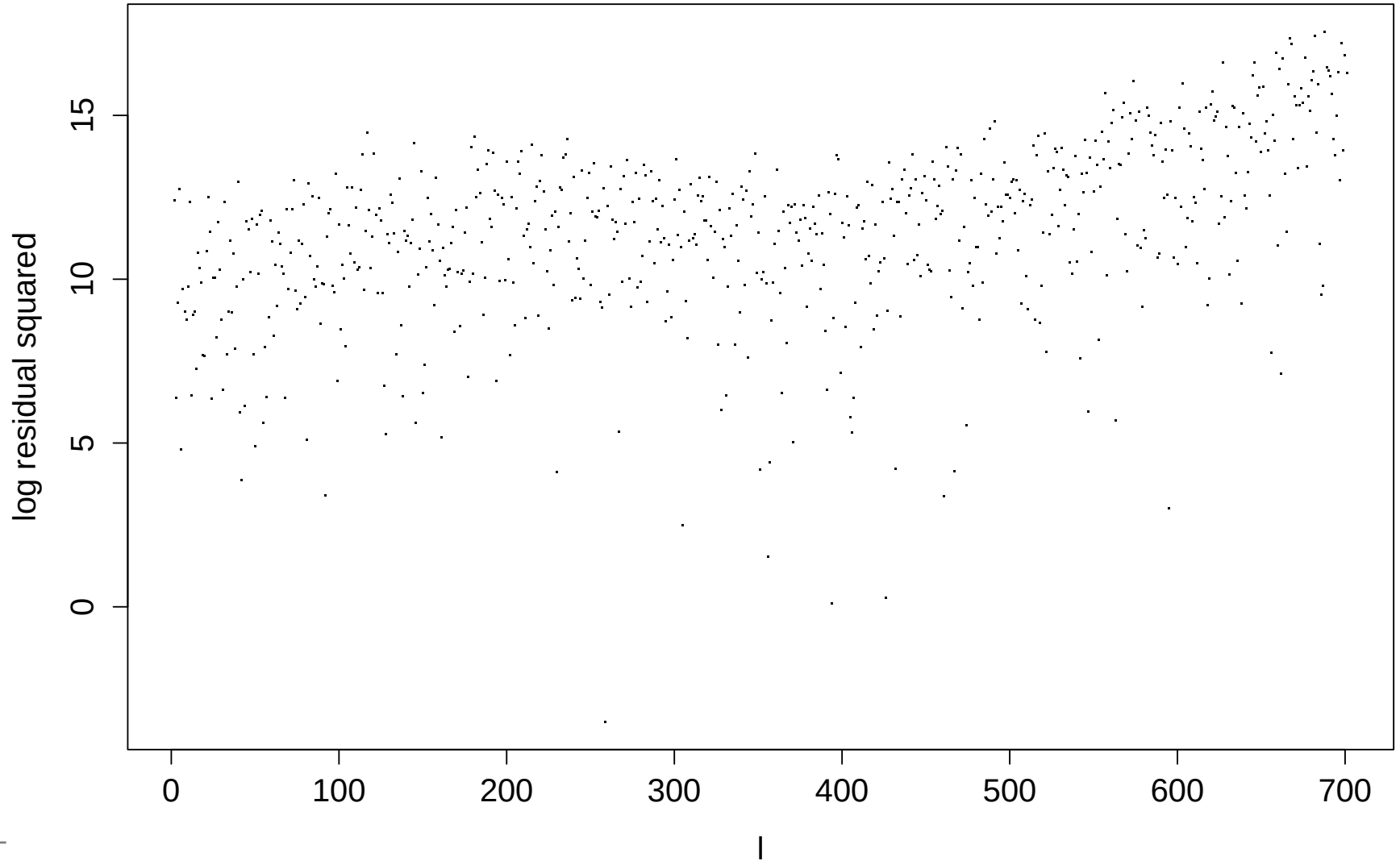
Variance Estimation

1. Estimate $f(x)$ with any nonparametric method to get an estimate $\widehat{f}_n(x)$.
2. Define $Z_i = \log(Y_i - \widehat{f}_n(x_i))^2$.
3. Regress the Z_i's on the x_i's (again using any nonparametric method) to get an estimate $\widehat{q}(x)$ of $\log \sigma^2(x)$ and let

$$\widehat{\sigma}^2(x) = e^{\widehat{q}(x)}.$$

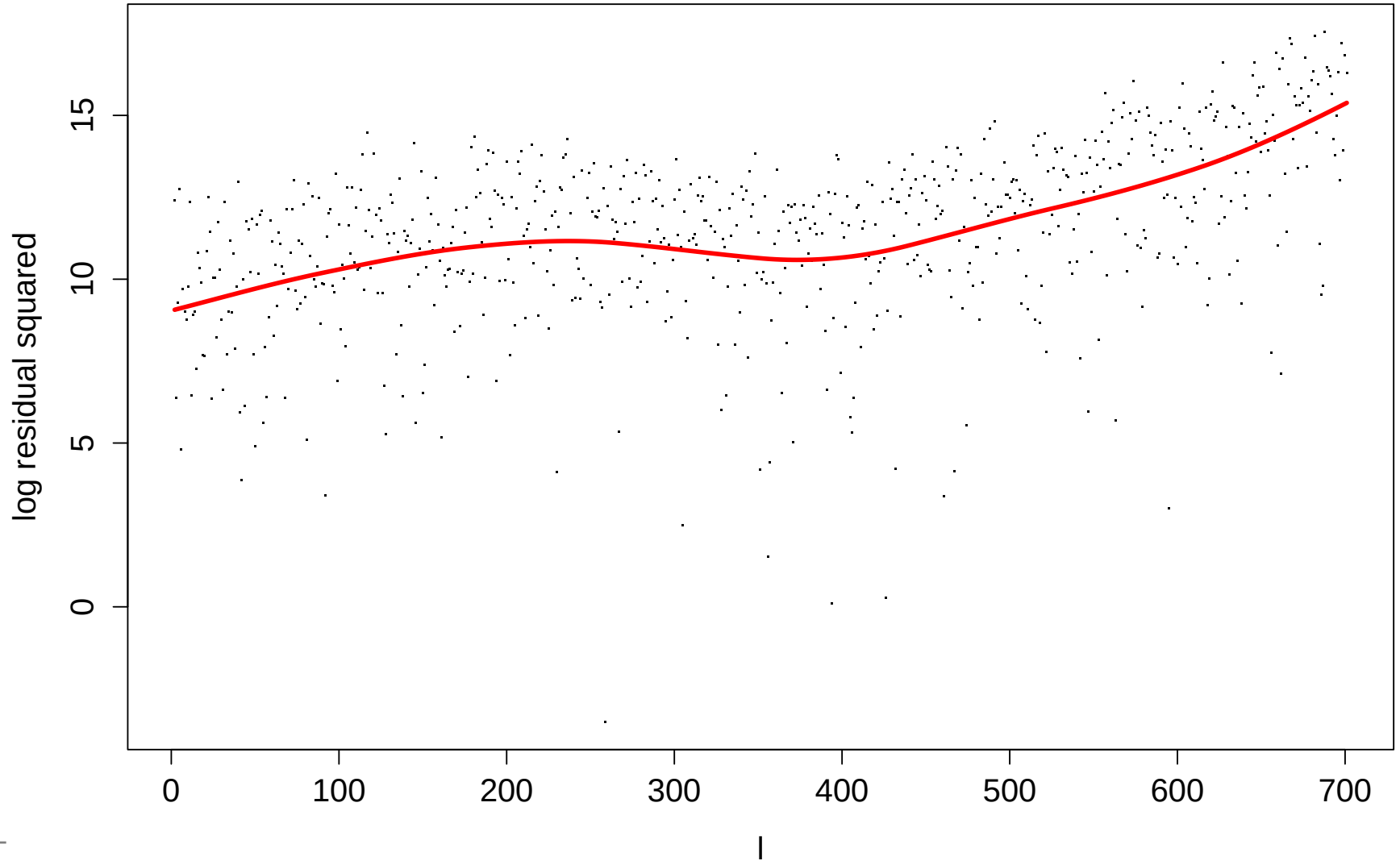
Example

WMAP data, log squared residuals, local linear fit, $h = 63.0$.



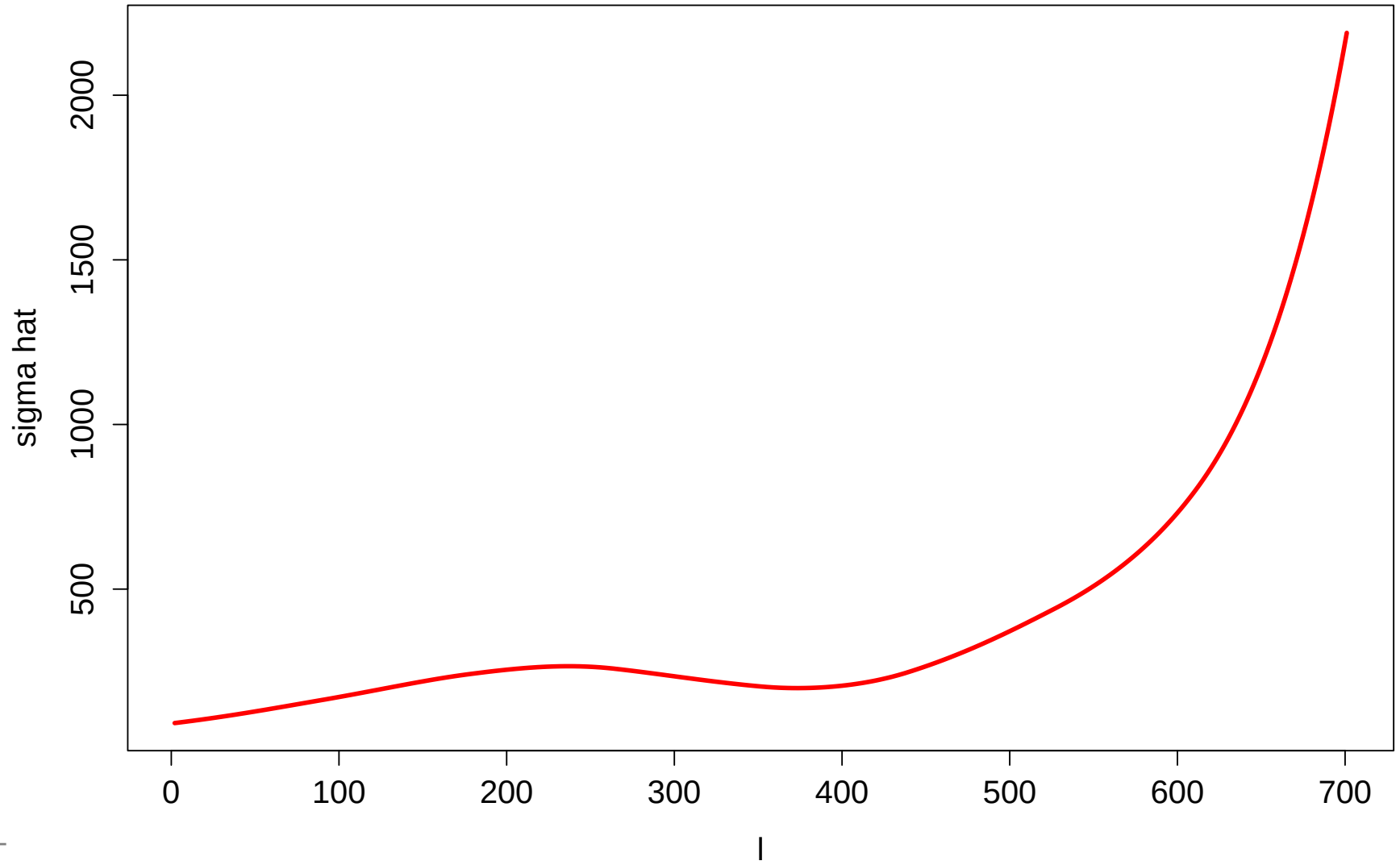
Example

With local linear fit, $h = 130$, chosen via GCV



Example

Estimating $\sigma(x)$:



Confidence Bands

Recall that

$$\widehat{f}(x) = \sum_i Y_i \ell_i(x)$$

so

$$\mathsf{Var}(\widehat{f}(x)) = \sum_i \sigma^2(X_i)\ell_i^2(x).$$

An approximate $1-\alpha$ confidence interval for $f(x)$ is

$$\widehat{f}(x) \pm z_{\alpha/2}\sqrt{\sum_i \widehat{\sigma}^2(X_i)\ell_i^2(x)}.$$

When $\sigma(x)$ is smooth, we can approximate

$$\sqrt{\sum_i \widehat{\sigma}^2(X_i)\ell_i^2(x)} \approx \widehat{\sigma}(x)\|\ell_i(x)\|.$$

Confidence Bands

Two caveats:

1. $\widehat{f}$ is biased so this is realy an interval for $\mathbb{E}(\widehat{f}(x))$. Result: bands can miss sharp peaks in the function.
2. Pointwise coverage does not imply simultaneous coverage for all x.

Solution for 2 is to replace $z_{\alpha/2}$ with a larger number (Sun and Loader 1994). `locfit()` does this for you.

More on `locfit()`

```
> diaghat = predict.locfit(locfitwmap,where="data",what="infl")
> normell = predict.locfit(locfitwmap,where="data",what="vari")
```

`diaghat` will be L_{ii}, $i = 1, 2, \ldots, n$.
`normell` will be $\|\ell_i(x)\|$, $i = 1, 2, \ldots, n$

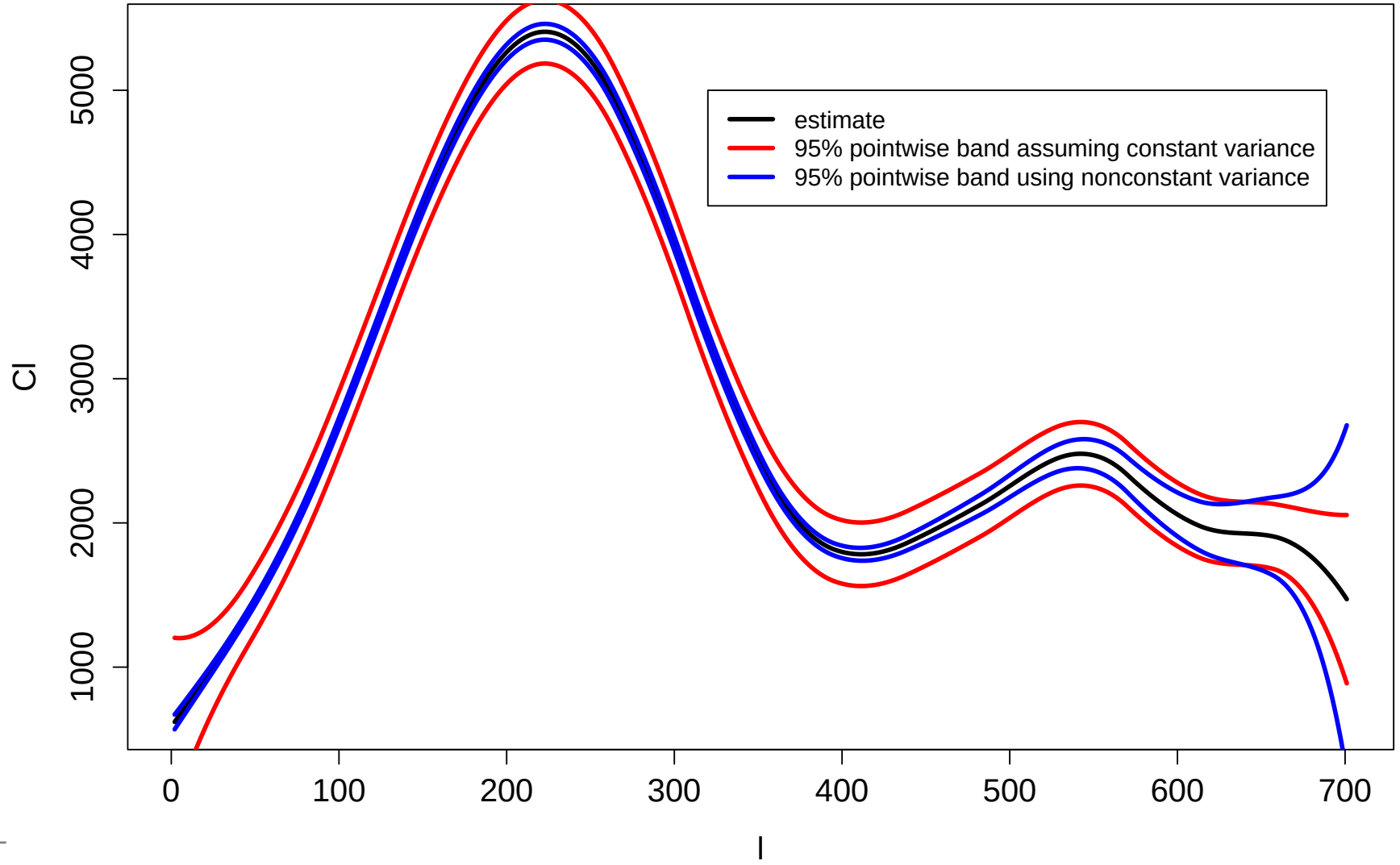
The Sun and Loader replacement for $z_{\alpha/2}$ is found using `kappa0(locfitwmap)$crit.val`.

More on `locfit()`

```
> critval = kappa0(locfitwmap)$crit.val

> postscript("confbandssimul.eps",width=10,height=7)
> plot(wmap$ell[1:700],fitwmap,lwd=3, xlab="l",ylab="Cl",
    cex=3,cex.axis=1.3, cex.lab=1.3,type="l")
> lines(wmap$ell[1:700],fitwmap+
    critval*sqrt(sigmasqrhat*normell),col=2,lwd=3)
> lines(wmap$ell[1:700],fitwmap-
    critval*sqrt(sigmasqrhat*normell),col=2,lwd=3)
...
```

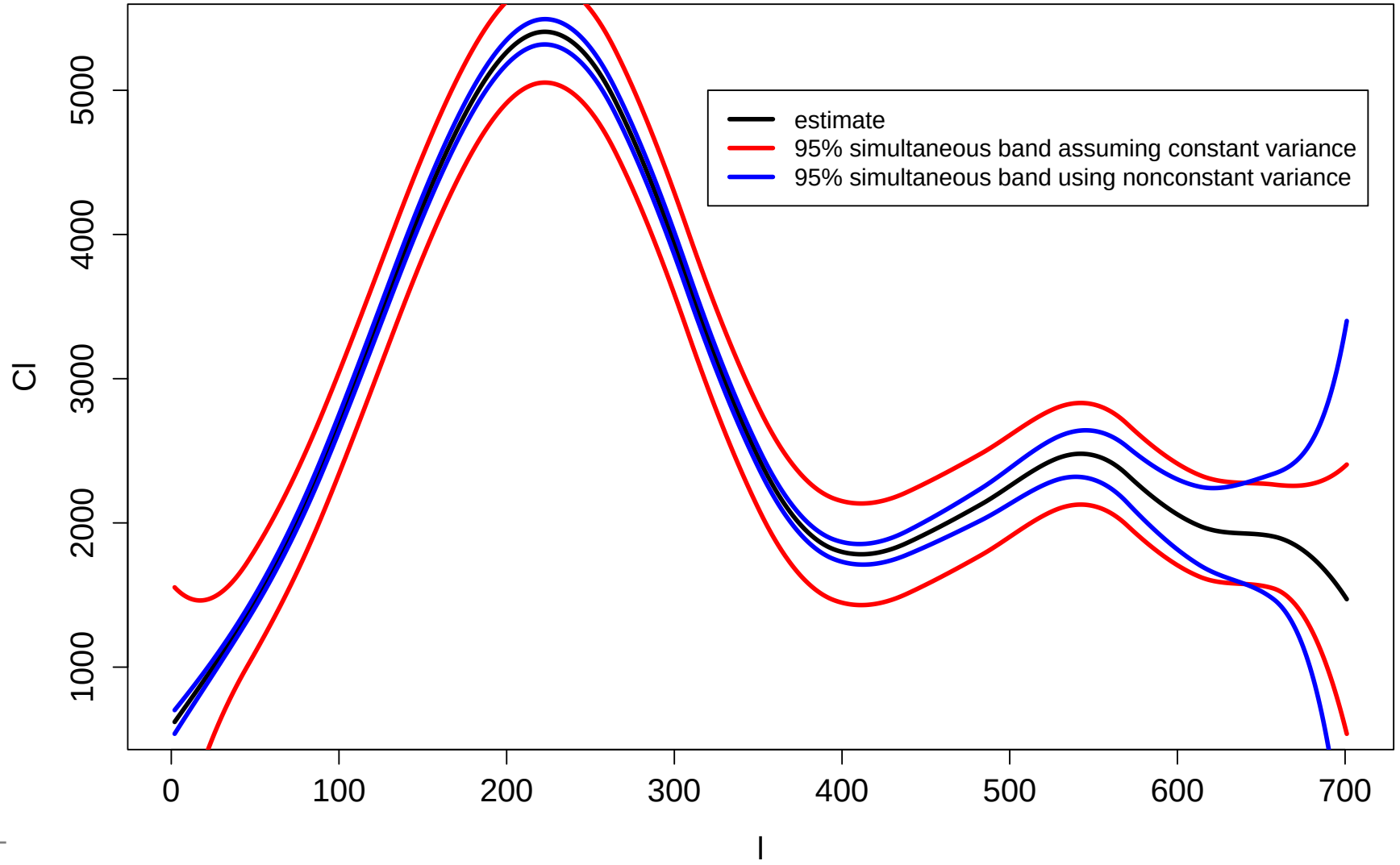
Example

95% pointwise confidence bands:



Example

95% simultaneous confidence bands:



Basis Methods

Idea: expand f as

$$f(x) = \sum_j \beta_j \psi_j(x)$$

where $\psi_1(x), \psi_2(x), \ldots$ are specially chosen, known functions. Then estimate β_j and set

$$f(x) = \sum_j \widehat{\beta}_j \psi_j(x).$$

We consider two versions: (i) splines, (ii) wavelets.

Splines and Penalization

Define $\widehat{f}_n$ to be the function that minimizes

$$M(\lambda) = \sum_i (Y_i - \widehat{f}_n(x_i))^2 + \lambda \int (f''(x))^2 dx.$$

$$\begin{aligned}
\lambda = 0 &\implies \widehat{f}_n(X_i) = Y_i \ \ (\text{no smoothing}) \\
\lambda = \infty &\implies \widehat{f}_n(x) = \widehat{\beta}_0 + \widehat{\beta}_1 x \ \ (\text{linear}) \\
0 < \lambda < \infty &\implies \widehat{f}_n(x) = \text{cubic spline with knots at } X_i.
\end{aligned}$$

A **cubic spline** is a continuous function f such that

1. f is a cubic polynomial between the X_i's
2. f has continuous first and second derivatives at the X_i's.

Basis For Splines

Define $(z)_+ = \max\{z, 0\}$, $N = n + 4$,

$$\psi_1(x) = 1 \qquad \psi_2(x) = x \qquad \psi_3(x) = x^2 \qquad \psi_4(x) = x^3$$
$$\psi_5(x) = (x - X_1)^3_+ \qquad \psi_6(x) = (x - X_2)^3_+ \qquad \cdots \qquad \psi_N(x) = (x - X_n)^3_+.$$

These functions form a basis for the splines: we can write

$$f(x) = \sum_{j=1}^{N} \beta_j \psi_j(x).$$

(For numerical calculations it is actually more efficient to use other spline bases.) We can thus write

(1)
$$\widehat{f}_n(x) = \sum_{j=1}^{N} \widehat{\beta}_j \psi_j(x),$$

Basis For Splines

We can now rewrite the minimization as follows:

$$\text{minimize}: \quad (Y - \Psi\beta)^T(Y - \Psi\beta) + \lambda\beta^T\Omega\beta$$

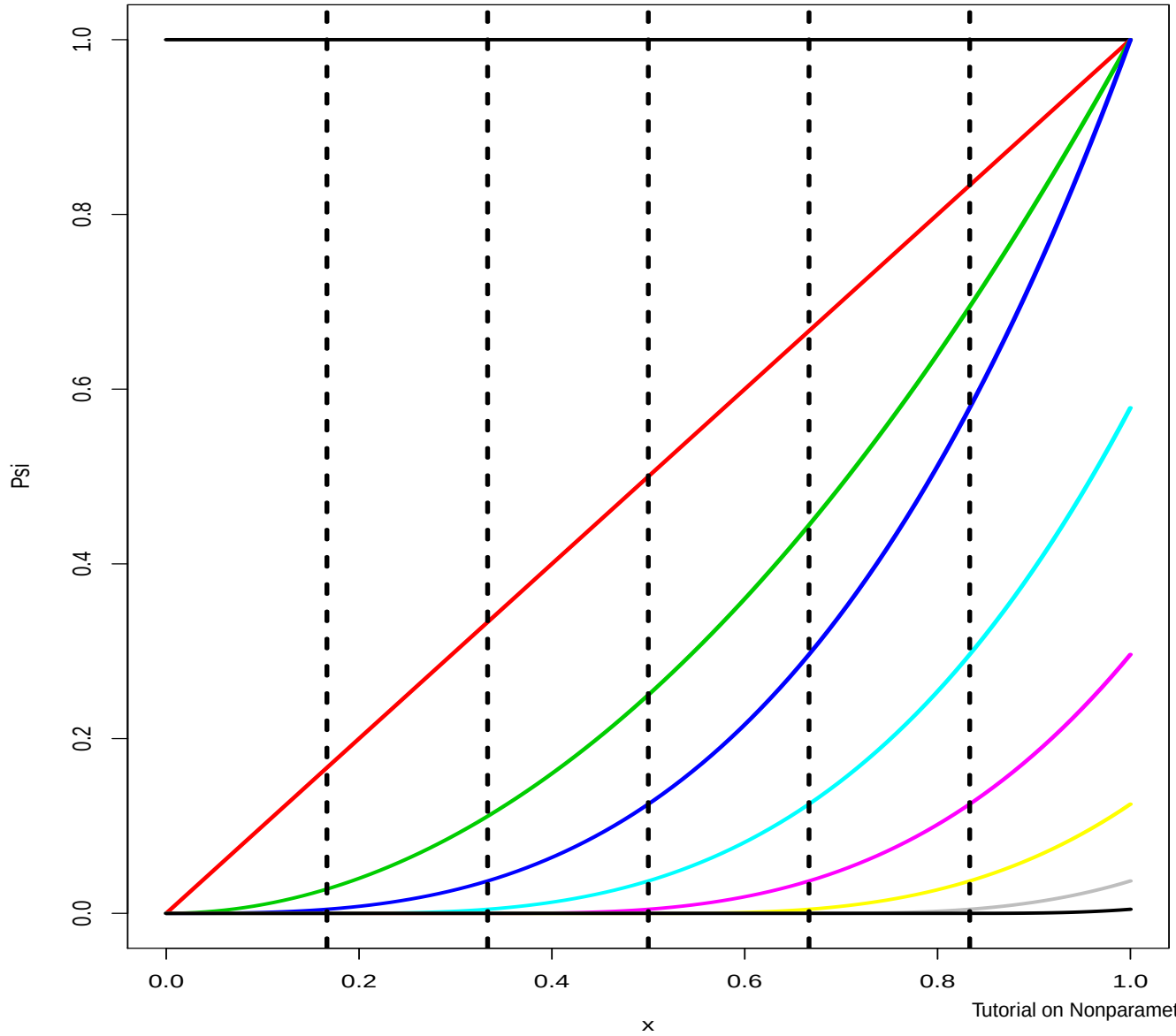
where $\Psi_{ij} = \psi_j(X_i)$ and $\Omega_{jk} = \int \psi_j''(x)\psi_k''(x)dx$. The value of β that minimizes this is

$$\widehat{\beta} = (\Psi^T\Psi + \lambda\Omega)^{-1}\Psi^T Y.$$

The smoothing spline $\widehat{f}_n(x)$ is a linear smoother, that is, there exist weights $\ell(x)$ such that $\widehat{f}_n(x) = \sum_{i=1}^n Y_i\ell_i(x)$.

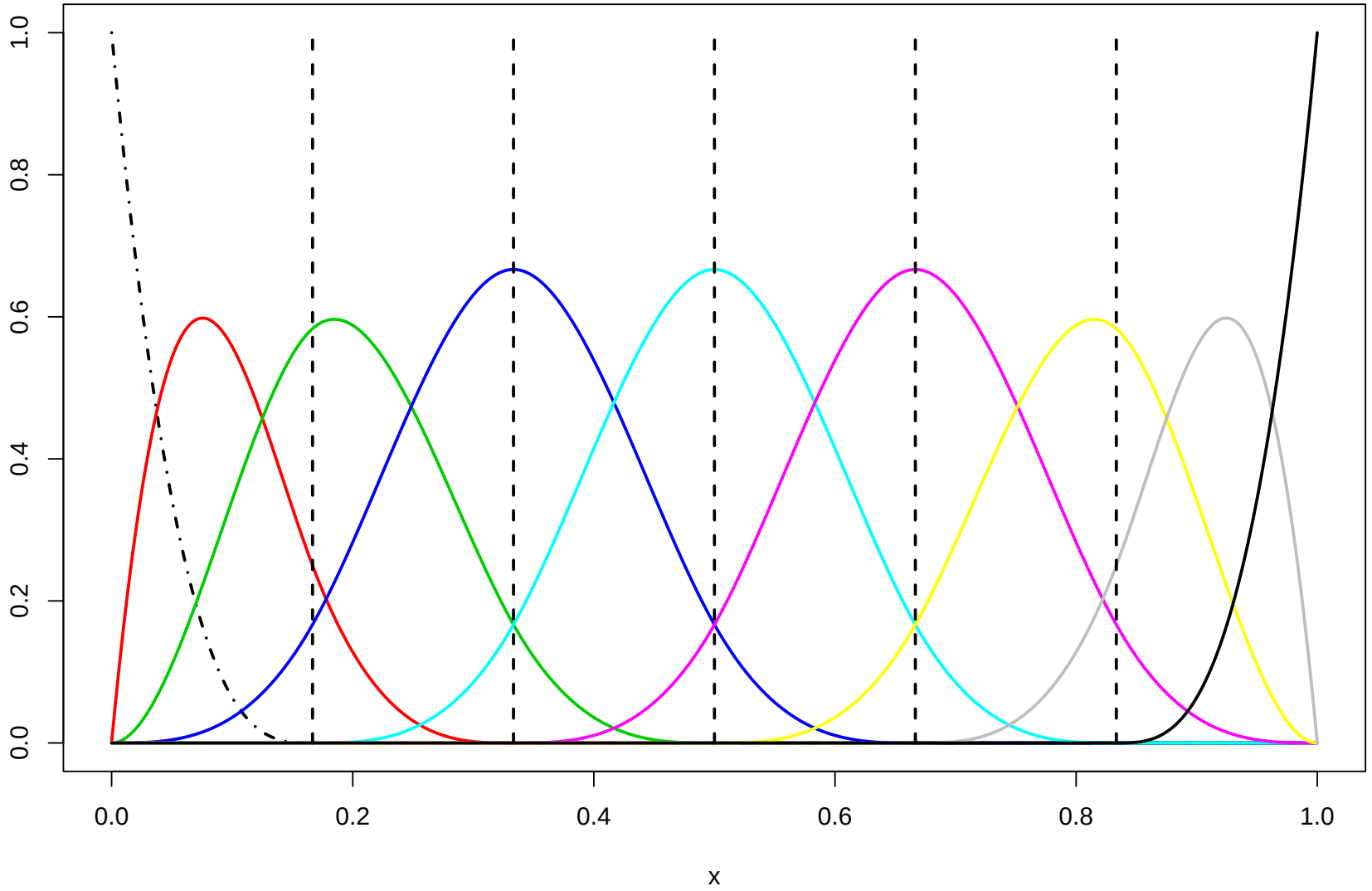
Basis For Splines

Basis functions with 5 knots.



Basis For Splines

Same span as previous slide, the **B-spline basis**, 5 knots:



Smoothing Splines in R

```
> smosplresult = spline.smooth(x,y,
cv=FALSE, all.knots=TRUE)
```

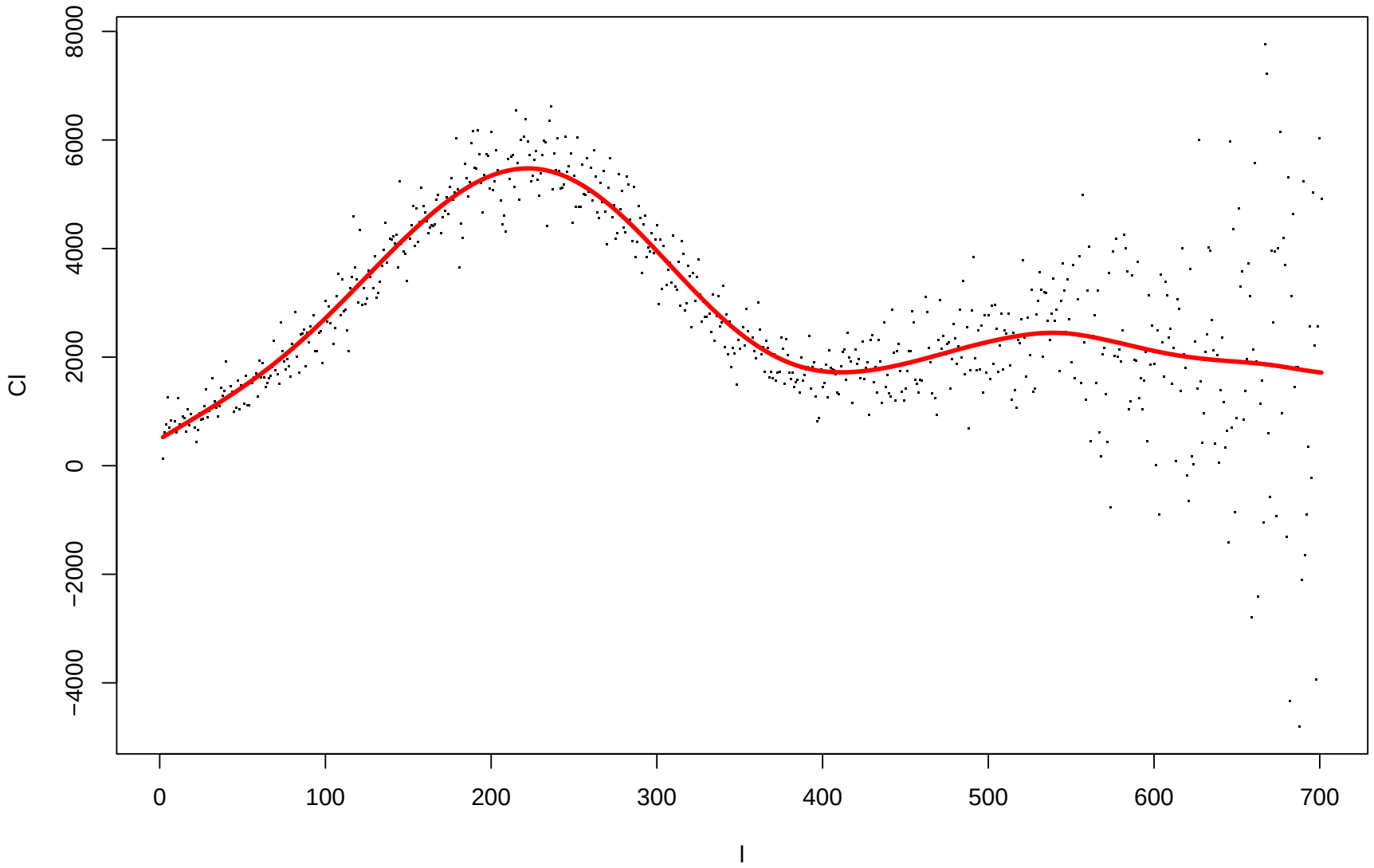
If `cv=TRUE`, then cross-validiation used to choose λ; if `cv=FALSE`, gcv is used

If `all.knots=TRUE`, then knots are placed at all data points; if `all.knots=FALSE`, then set `nknots` to specify the number of knots to be used. Using fewer knots eases the computational cost.

`predict(smosplresult)` gives fitted values.

Example

WMAP data, λ chosen using GCV.



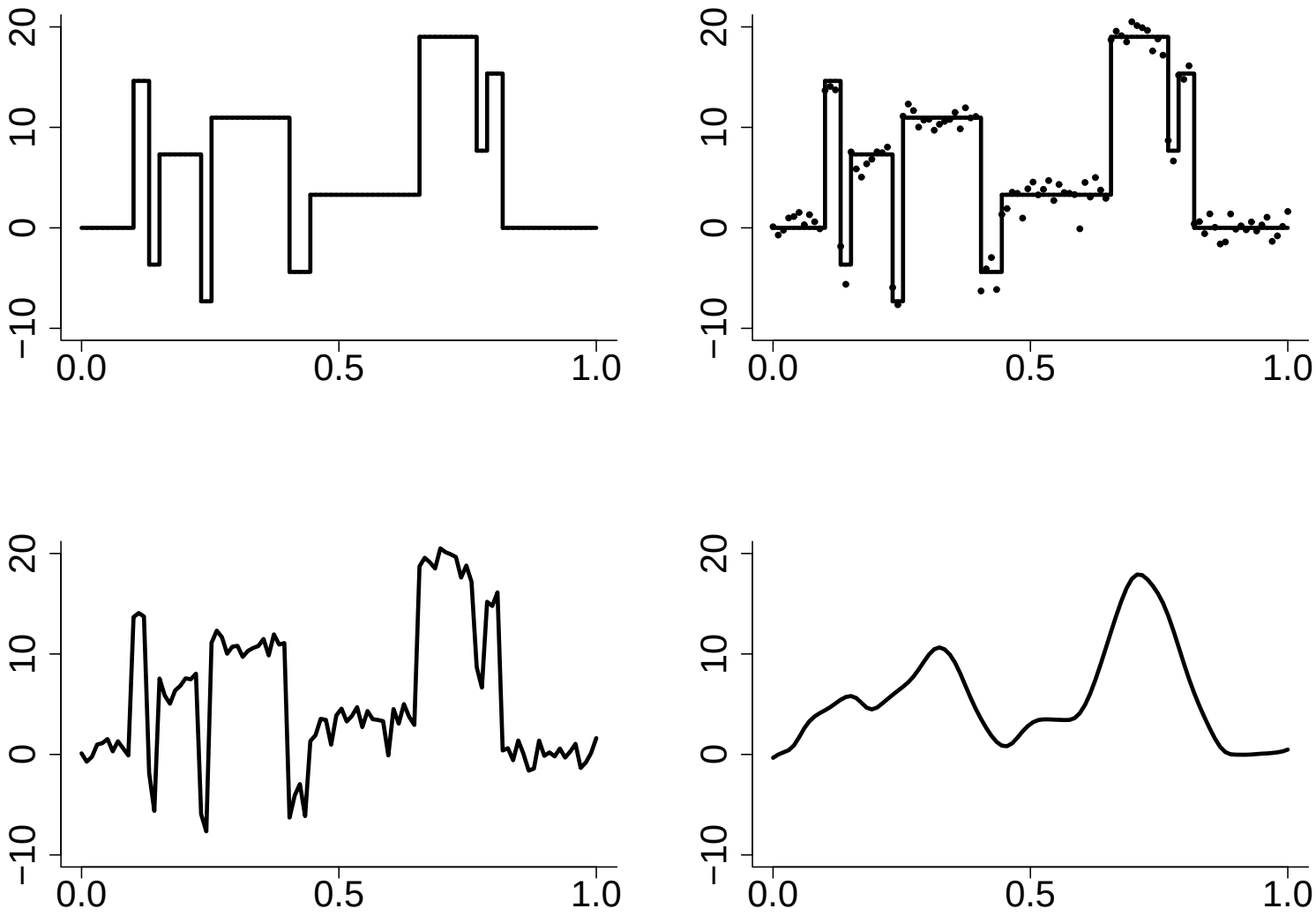
Wavelets

Wavelets are special basis functions with two appealing features:

1. Can be computed quickly.
2. The resulting estimators are **spatially adaptive.**

This means we can accomodate local features in the data.

Wavelets



Small bandwidth (lower left) picks up the jumps but adds many wiggles. Large bandwidth (lower right) is smooth but misses the jumps.

Haar Wavelets

We start with the simplest wavelet, the Haar wavelet. The **Haar father wavelet** or **Haar scaling function** is defined by

$$\phi(x) = \begin{cases} 1 & \text{if } 0 \leq x < 1 \\ 0 & \text{otherwise.} \end{cases} \tag{2}$$

The **mother Haar wavelet** is defined by

$$\psi(x) = \begin{cases} -1 & \text{if } 0 \leq x \leq \frac{1}{2} \\ 1 & \text{if } \frac{1}{2} < x \leq 1. \end{cases} \tag{3}$$

Haar Wavelets

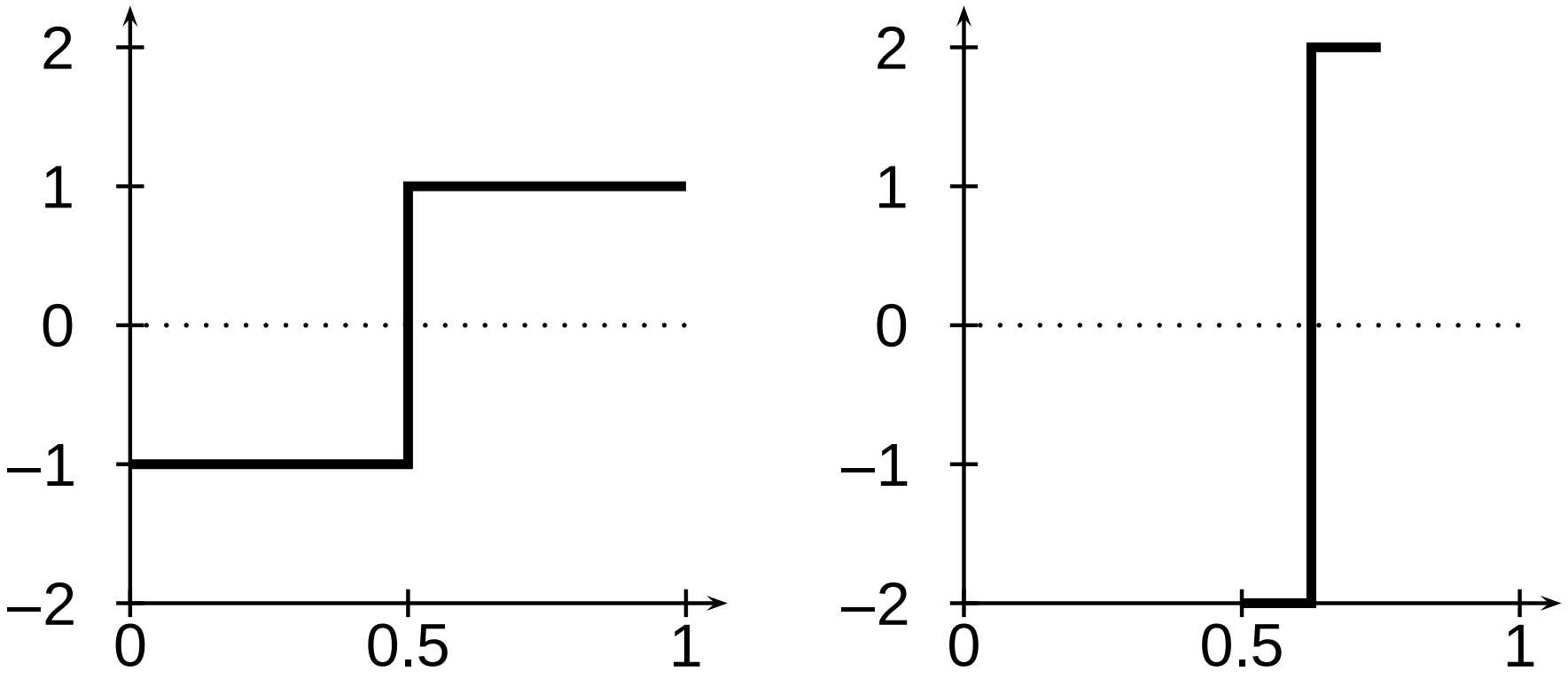
Let

$$\psi_{jk}(x) = 2^{j/2}\psi(2^j x - k).$$

Father	ϕ
Level 1	ψ
Level 2	ψ_{10} ψ_{11}
Level 3	ψ_{20} ψ_{21} ψ_{22} ψ_{23}
Level 4	ψ_{30} ψ_{31} ψ_{32} ψ_{33} ψ_{34} ψ_{35} ψ_{36} ψ_{37}

The set of wavelets is: etc.

Haar Wavelets



Haar wavelets. Left: the mother wavelet $\psi(x)$; right: $\psi_{2,2}(x)$.

Haar Wavelets

The set of functions

$$\left\{\{\phi\}, \{\psi_{00}\}, \{\psi_{10}, \psi_{11}\}, \{\psi_{20}, \psi_{21}, \psi_{22}, \psi_{23}\}, \cdots\right\}$$

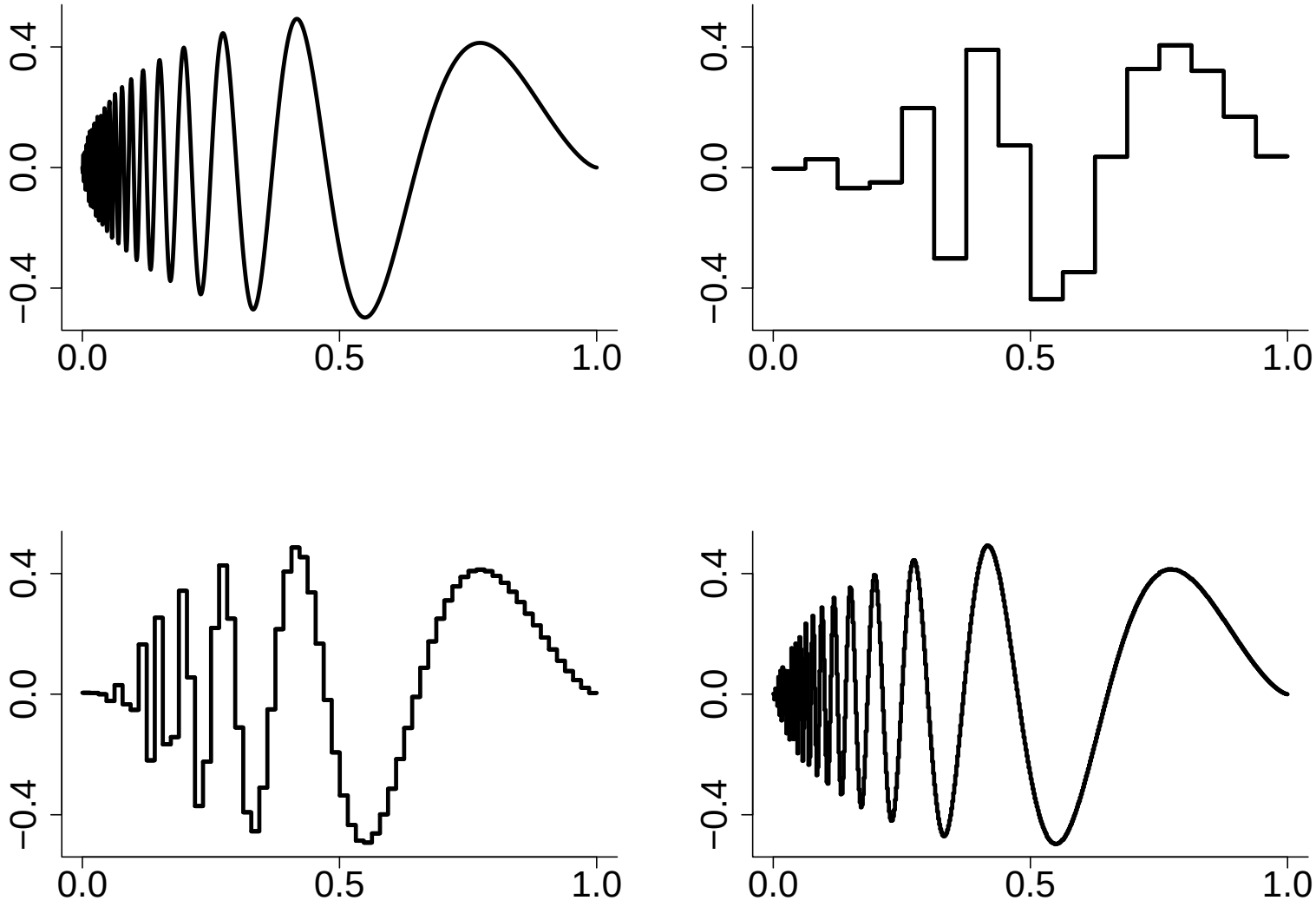
is an orthonormal basis for $L_2(0,1)$. So, as $J \to \infty$,

$$f(x) \approx \alpha\, \phi(x) + \sum_{j=0}^{J} \sum_{k=0}^{2^j - 1} \beta_{jk} \psi_{jk}(x)$$

where

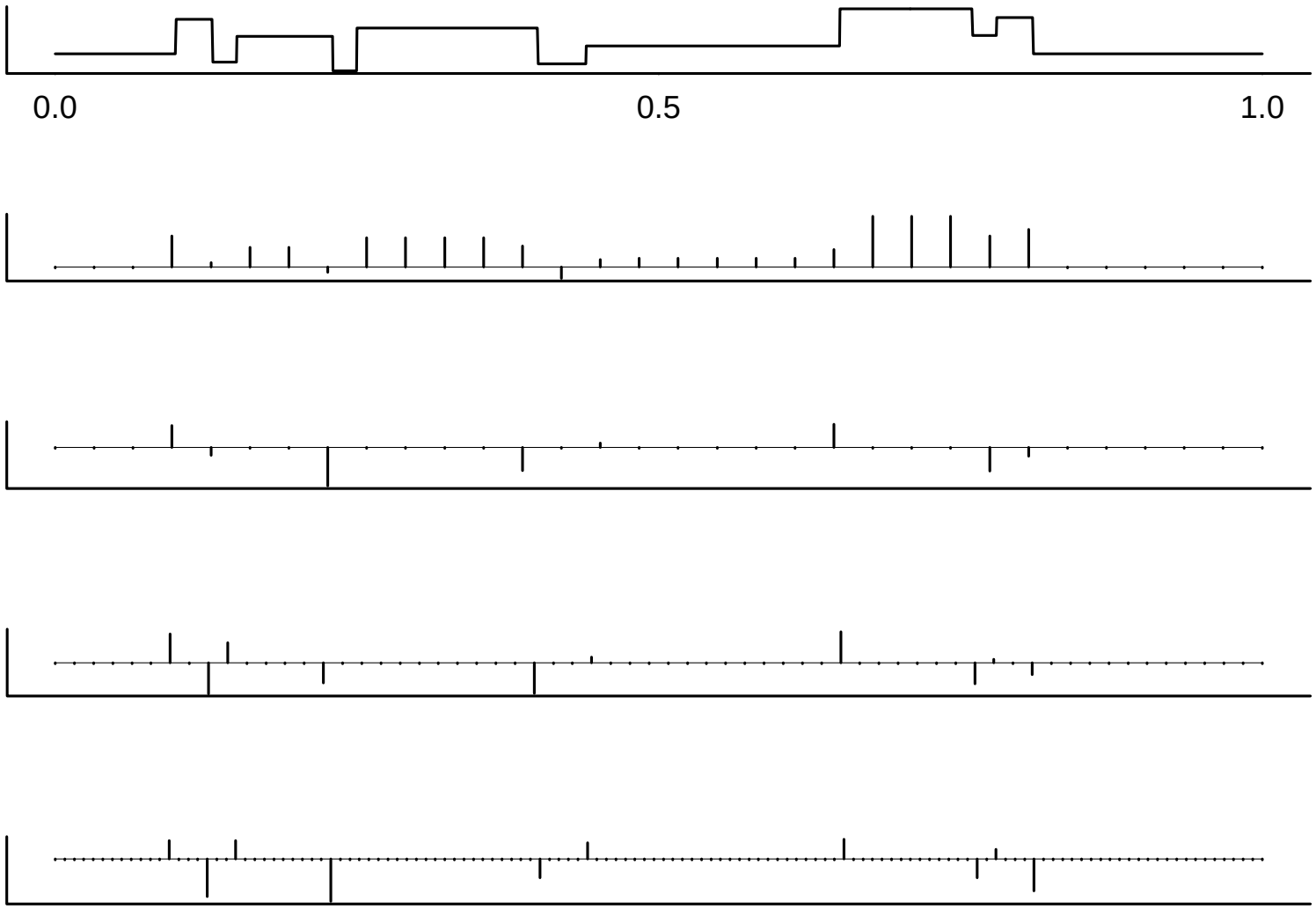
$$\alpha = \int_0^1 f(x)\phi(x)\, dx, \quad \beta_{jk} = \int_0^1 f(x)\psi_{jk}(x)\, dx.$$

Haar Wavelets



The Doppler signal, $J = 3$, $J = 5$, and $J = 8$.

Haar Wavelets



Haar Wavelets

Many functions f have **sparse** expansions in a wavelet basis. Smooth functions are sparse. (Smooth + jumps) is also sparse. So, we expect that for many functions we can write

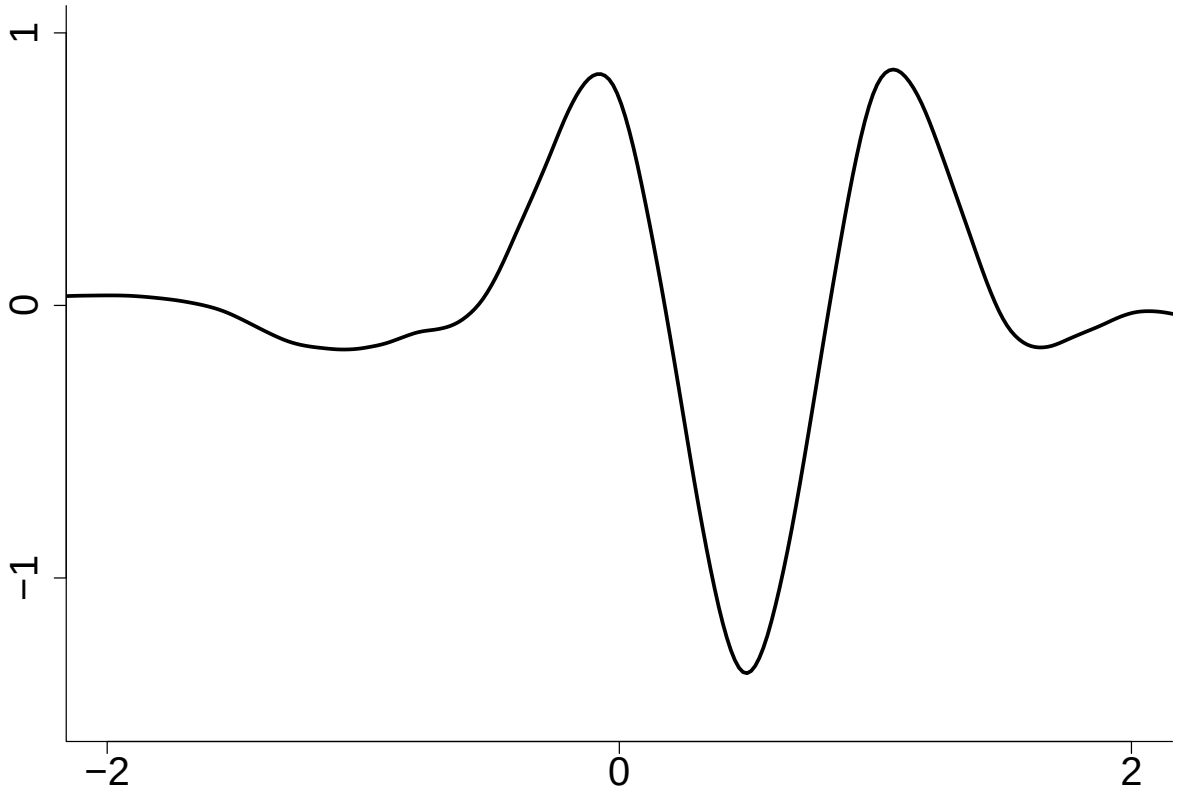
$$f(x) = \alpha\phi(x) + \sum_j \sum_j \beta_{jk}\psi_{jk}(x)$$

where most $\beta_{jk} \approx 0$.

The idea is to estimate the $\beta'_{jk}s$ then set all $\widehat{\beta}_{jk} = 0$ except for a few large coefficients.

Smoother Wavelets

symmlet mother wavelet (Daubechies), $N = 8$ vanishing moments:



Smoother Wavelets

Father ϕ and mother ψ. Smooth wavelets cannot be written in closed form but they can be computed quickly. Still have:

$$f(x) = \sum_k \alpha_{0k}\, \phi_{0k}(x) + \sum_{j=0}^{\infty} \sum_k \beta_{jk}\, \psi_{jk}(x)$$

where $\alpha_{0k} = \int f(x)\phi_{0k}(x)dx$ and $\beta_{jk} = \int f(x)\psi_{jk}(x)dx$.

Wavelet Regression

Let

$$Y_i = f(x_i) + \sigma\epsilon_i$$

where $x_i = i/n$. (Adjustments are needed fot non-equally spaced data.) Procedure:

1. Form preliminary estimate:

$$\widetilde{\beta}_{jk} = \frac{1}{n}\sum_i Y_i \psi_{jk}(x_i) \left(\approx \int f(x)\psi_{jk}(x)dx = \beta_{jk} \right).$$

2. **Shrink**: $\widehat{\beta}_{jk} \longleftarrow \text{shrink}(\widetilde{\beta}_{jk})$.

3. Reconstruct function:

$$\widehat{f}(x) = \sum_k \widehat{\alpha}_{0k}\, \phi_{0k}(x) + \sum_{j=0}^{\infty}\sum_k \widehat{\beta}_{jk}\, \psi_{jk}(x).$$

Wavelet Regression

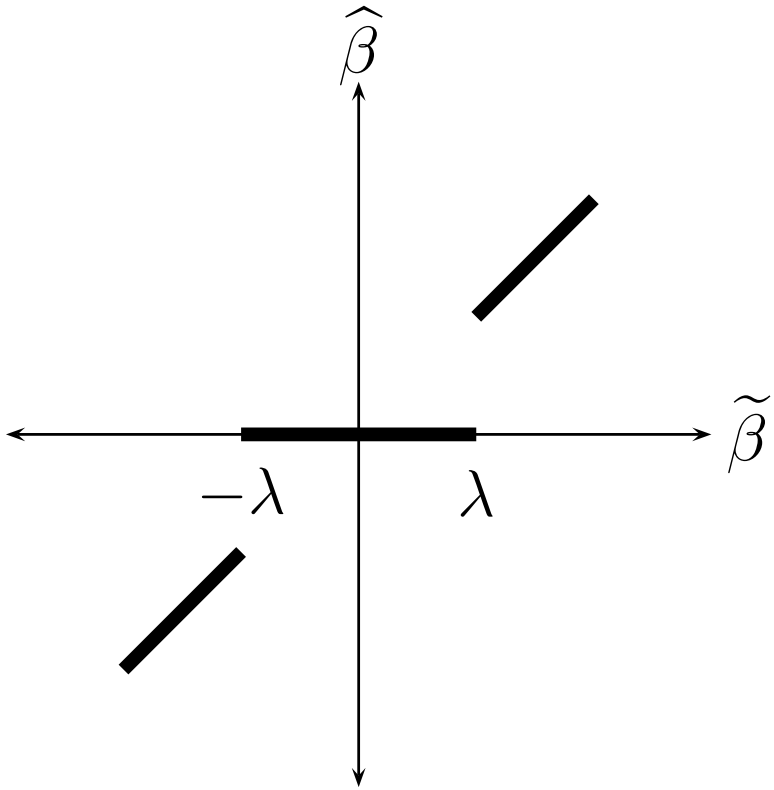
In practice, the preliminary estimates are computed using the discrete wavelet transform (DWT). Two types of shrinkage are used: **hard thresholding** and **soft thresholding.** The hard threshold estimator is

$$\widehat{\beta}_{jk} = \begin{cases} 0 & \text{if } |\widetilde{\beta}_{jk}| < \lambda \\ \widetilde{\beta}_{jk} & \text{if } |\widetilde{\beta}_{jk}| \geq \lambda. \end{cases}$$

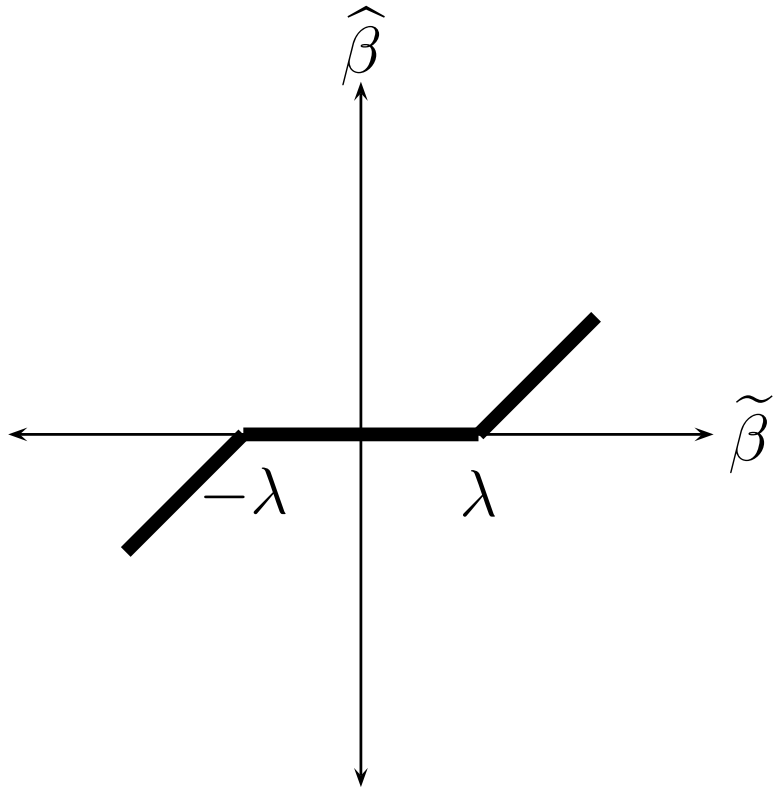
The soft threshold estimator is

$$\widehat{\beta}_{jk} = \text{sign}(\widetilde{\beta}_{jk})(|\widetilde{\beta}_{jk}| - \lambda)_+.$$

Wavelet Regression



Hard thresholding



Soft thresholding

Estimating σ

The highest level coefficients should be mostly noise except, possibly, a few large coefficients.

$$\widehat{\sigma} = \sqrt{n} \times \frac{\text{median}\left(|\widetilde{\beta}_{J-1,k} - m| : \ k = 0, \ldots, 2^{J-1} - 1\right)}{0.6745}.$$

where

$$m = \text{median}\left(\widetilde{\beta}_{J-1,k}) : \ k = 0, \ldots, 2^{J-1} - 1\right)$$

Wavelet Regression

We still need to choose the threshold λ. There are several methods for choosing λ. The simplest rule is the **universal threshold** defined by

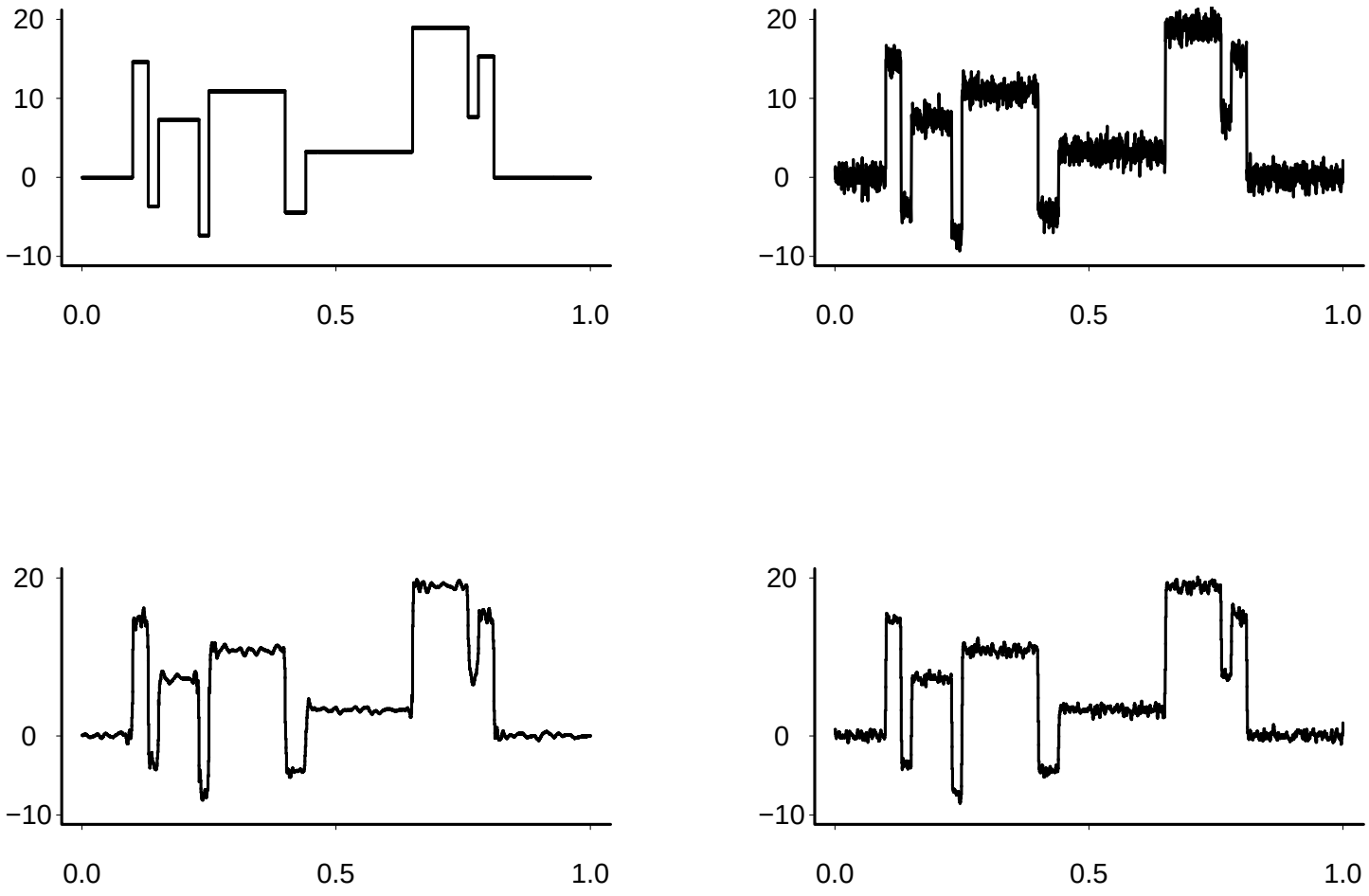
$$\lambda = \widehat{\sigma}\sqrt{\frac{2\log n}{n}}.$$

Another estimator, called **SureShrink** is obtained by using a different threshold λ_j for each level. The threshold λ_j is chosen to minimize SURE (Stein's Unbiased Risk Estimator):

$$S(\lambda_j) = \sum_{k=1}^{n_j}\left[\frac{\widehat{\sigma}^2}{n} - 2\frac{\widehat{\sigma}^2}{n}\, I(|\widetilde{\beta}_{jk}| \leq \lambda_j) + \min(\widetilde{\beta}_{jk}^2, \lambda_j^2)\right] \tag{4}$$

where $n_j = 2^{j-1}$ is the number of parameters at level j.

Example



Top left: the function $f(x)$. Top right: 2048 data points. Bottom left: Using wavelets. Bottom right: Using local linear regression with bandwidth chosen by CV.

Wavelet Regression in R

Include library `wavethresh`.

Assumes y has length which is power of two: Interpolate as needed:

```
> library(wavethresh)
> xs = seq(1,700,length=512)
> lo = floor(xs)
> hi = ceiling(xs)
> ys = (xs-lo)*wmap$Cl[hi] + (hi-xs)*wmap$Cl[lo] +
   wmap$Cl[lo]*(lo==hi)
```

Wavelet Regression in R

The function `wd( )` does the initial wavelet transform:

```
> waveletwmap = wd(ys, family="DaubLeAsymm", filter.number=8)
```

Argument `family="DaubLeAsymm"` gives the Daubechies symmlet

Argument `filter.number=8` sets $N = 8$ (eight vanishing moments)

Wavelet Regression in R

The function `threshold()` does soft and hard thresholding:

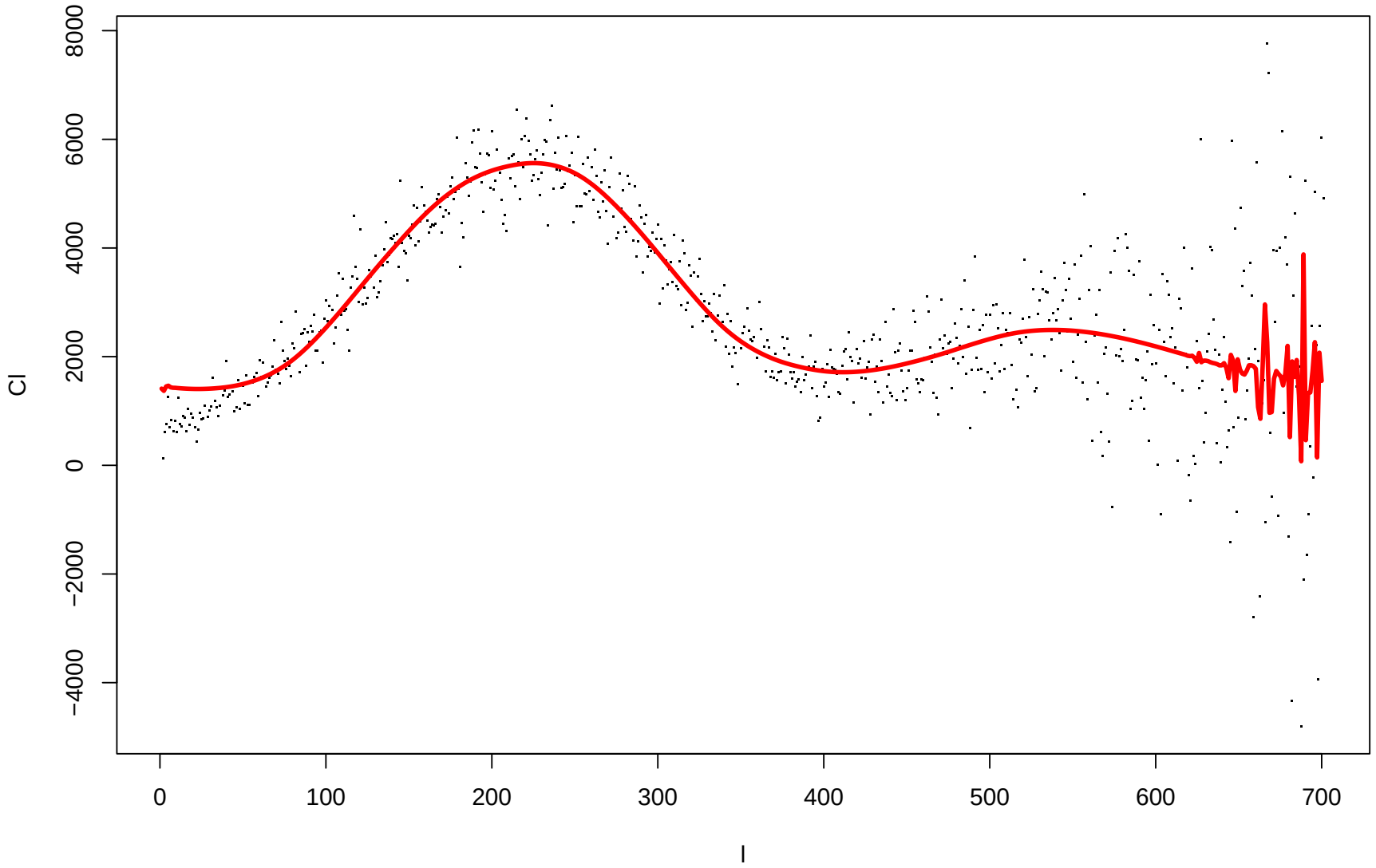
```
> softthreshwmap = threshold(waveletwmap,type="soft",
   policy="universal")
> hardthreshwmap = threshold(waveletwmap,type="hard",
   policy="universal")
```

Argument `policy="universal"` specifies universal thresholding

To invert the transform (i.e., get fitted values) use `wr(softthreshwmap)`

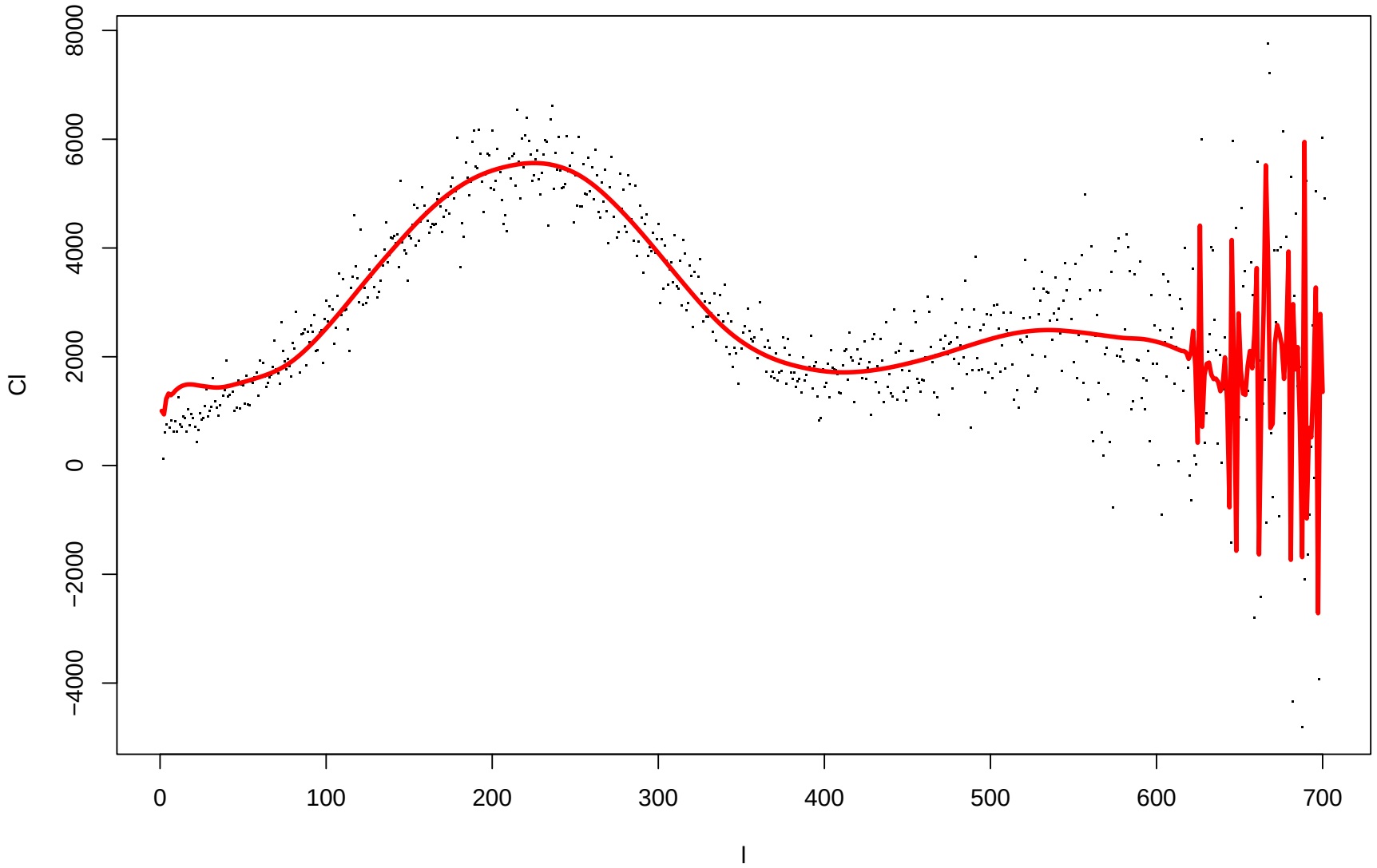
Example

WMAP data, wavelet regression with soft thresholding.



Example

WMAP data, wavelet regression with hard thresholding.



Multiple Regression

$$Y = f(X_1, X_2, \ldots, X_d) + \epsilon$$

The curse of dimensionality:

Optimal rate of convergence for $d = 1$ is $n^{-4/5}$. In d dimensions the optimal rate of convergence is is $n^{-4/(4+d)}$. Thus, the sample size m required for a d-dimensional problem to have the same accuracy as a sample size n in a one-dimensional problem is $m \propto n^{cd}$ where $c = (4+d)/(5d) > 0$.

> To maintain a given degree of accuracy of an estimator, the sample size must increase exponentially with the dimension d.

Put another way, confidence bands get very large as the dimension d increases.

Multiple Local Linear

Given a nonsingular positive definite $d \times d$ bandwidth matrix H, we define

$$K_H(x) = \frac{1}{|H|^{1/2}} K(H^{-1/2}x).$$

Often, one scales each covariate to have the same mean and variance and then we use the kernel

$$h^{-d} K(||x||/h)$$

where K is any one-dimensional kernel. Then there is a single bandwidth parameter h.

Multiple Local Linear

At a target value $x = (x_1, \ldots, x_d)^T$, the local sum of squares is given by

$$\sum_{i=1}^{n} w_i(x) \left(Y_i - a_0 - \sum_{j=1}^{d} a_j (x_{ij} - x_j) \right)^2$$

where

$$w_i(x) = K(||x_i - x||/h).$$

The estimator is $\widehat{f}_n(x) = \widehat{a}_0$ $\widehat{a} = (\widehat{a}_0, \ldots, \widehat{a}_d)^T$ is the value of $a = (a_0, \ldots, a_d)^T$ that minimizes the weighted sums of squares.

Multiple Local Linear

The solution $\widehat{a}$ is

$$\widehat{a} = (X_x^T W_x X_x)^{-1} X_x^T W_x Y$$

where

$$X_x = \begin{pmatrix} 1 & (x_{11} - x_1) & \cdots & (x_{1d} - x_d) \\ 1 & (x_{21} - x_1) & \cdots & (x_{2d} - x_d) \\ \vdots & \vdots & \ddots & \vdots \\ 1 & (x_{n1} - x_1) & \cdots & (x_{nd} - x_d) \end{pmatrix}$$

and W_x is the diagonal matrix whose (i, i) element is $w_i(x)$.

Additive Models

$$Y = \alpha + \sum_{j=1}^{d} f_j(x_j) + \epsilon$$

Usually take $\widehat{\alpha} = \overline{Y}$. Then estimate the f_j's by **backfitting**.

1. set $\widehat{\alpha} = \overline{Y}$, $\widehat{f}_1 = \cdots \widehat{f}_d = 0$.
2. Iterate until convergence: for $j = 1, \ldots, d$:
   - Compute $\widetilde{Y}_i = Y_i - \widehat{\alpha} - \sum_{k \neq j} \widehat{f}_k(X_i)$, $i = 1, \ldots, n$.
   - Apply a smoother to $\widetilde{Y}_i$ on X_j to obtain $\widehat{f}_j$.
   - Set $\widehat{f}_j(x)$ equal to $\widehat{f}_j(x) - n^{-1} \sum_{i=1}^{n} \widehat{f}_j(x_i)$.

The last step ensures that $\sum_i \widehat{f}_j(X_i) = 0$ (identifiability).

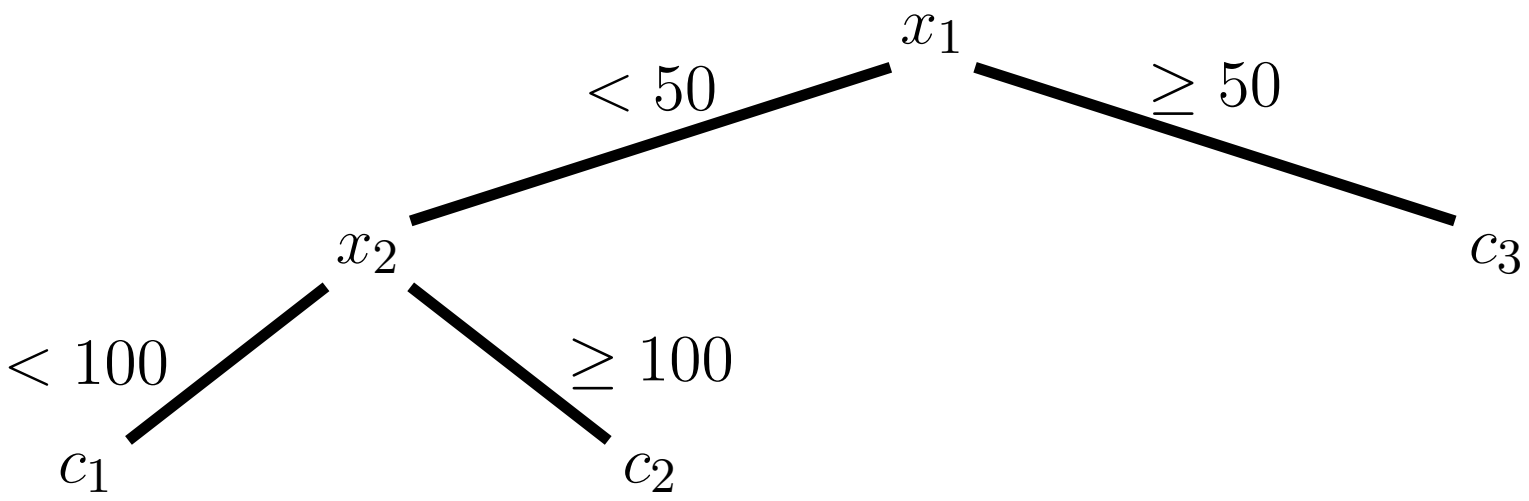
Regression Trees

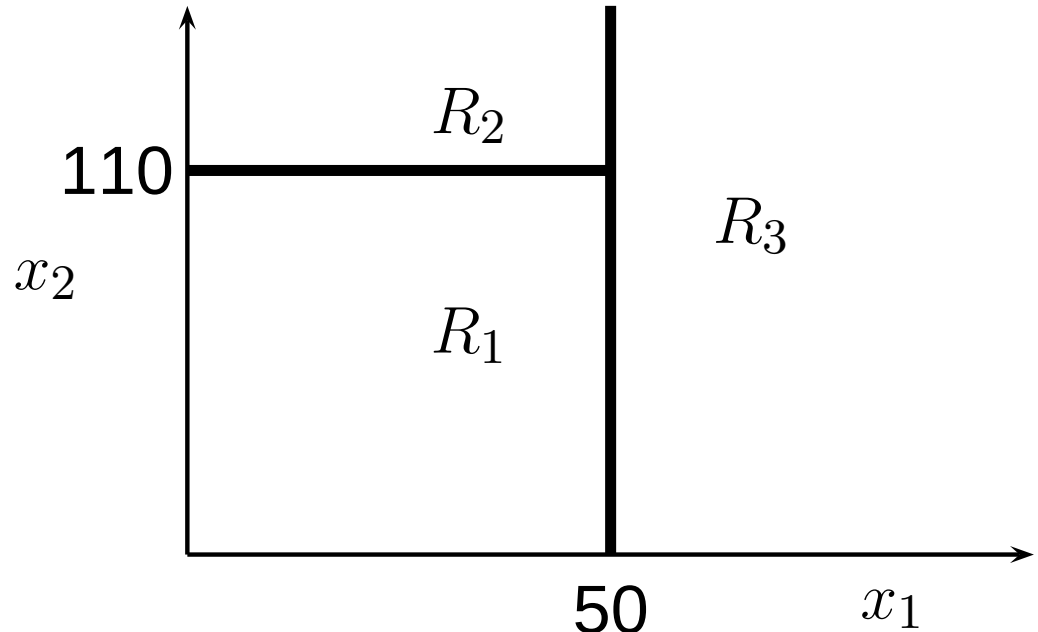
A regression tree is a model of the form

$$f(x) = \sum_{m=1}^{M} \overline{Y}_m I(x \in R_m)$$

where $R_1, \ldots, R_M$ are disjoint rectangles.

Regression Trees





Regression Trees

Generally one grows a very large tree, then the tree is pruned to form a subtree by collapsing regions together. The size of the tree is a tuning parameter chosen as follows. Let N_m denote the number of points in a rectangle R_m of a subtree T and define

$$c_m = \frac{1}{N_m} \sum_{x_i \in R_m} Y_i, \quad Q_m(T) = \frac{1}{N_m} \sum_{x_i \in R_m} (Y_i - c_m)^2.$$

Regression Trees

Define the complexity of T by

$$\begin{aligned} C_k(T) &= \sum_{m=1}^{|T|} N_m Q_m(T) + k|T| \\ &= \sum_{i=1}^{n} (Y_i - \widehat{f}(X_i))^2 + k|T| \\ &= \textbf{Residual sum of squares} + k|T| \end{aligned}$$

where $k > 0$ and $|T|$ is the number of terminal nodes of the tree. Let T_k be the smallest subtree that minimizes C_k. The value $\widehat{k}$ of k can be chosen by cross-validation. The final estimate is based on the tree $T_{\widehat{k}}$.

Example

From
`http://astrostatistics.psu.edu/datasets/Shapley_galaxy.html`:

Redshifts (i.e. velocities in km/s with respect to us) are now measured for 4215 galaxies in the Shapley Concentration regions (Drinkwater et al. 2004).

The dataset has the following columns:

Right ascension: Coordinate in the sky similar to longitude on Earth, 0 to 360 degrees

Declination: Coordinate in the sky similar to latitude on Earth, -90 to +90 degrees

Magnitude: An inverted logarithmic measure of galaxy brightness in the optical band

Velocity: Speed of the galaxy moving away from Earth, after various corrections are applied

Sigma of velocity: Heteroscedastic measurement error known for each individual velocity mesurement

Regression Trees in R

Initial fit using `tree()`:

```
> library(tree)

> galaxy = read.table("ShapleyGalaxy.dat",skip=2,header=T)
> galaxy = galaxy[galaxy$Magnitude>0,]

> galtree = tree(Magnitude~ RightAsc + Declination +
   Velocity, data=galaxy)
```

There are 3,858 galaxies after removing those with missing Magnitude.

Regression Trees in R

Prune tree using `prune.tree()`:

```
> RSS = rep(0,summary(galtree)$size)
> RSS[1] = sum((galaxy$Magnitude -
   mean(galaxy$Magnitude))^2)
> for(i in 2:summary(galtree)$size)
  {
     RSS[i] = summary(prune.tree(galtree,best=i))$dev
  }
```

`summary(galtree)$size` returns $|T|$

`summary(galtree)$dev` returns RSS

For `prune.tree()`, specify either `k` or `best` (`best` sets maximum value of $|T|$)

Regression Trees in R

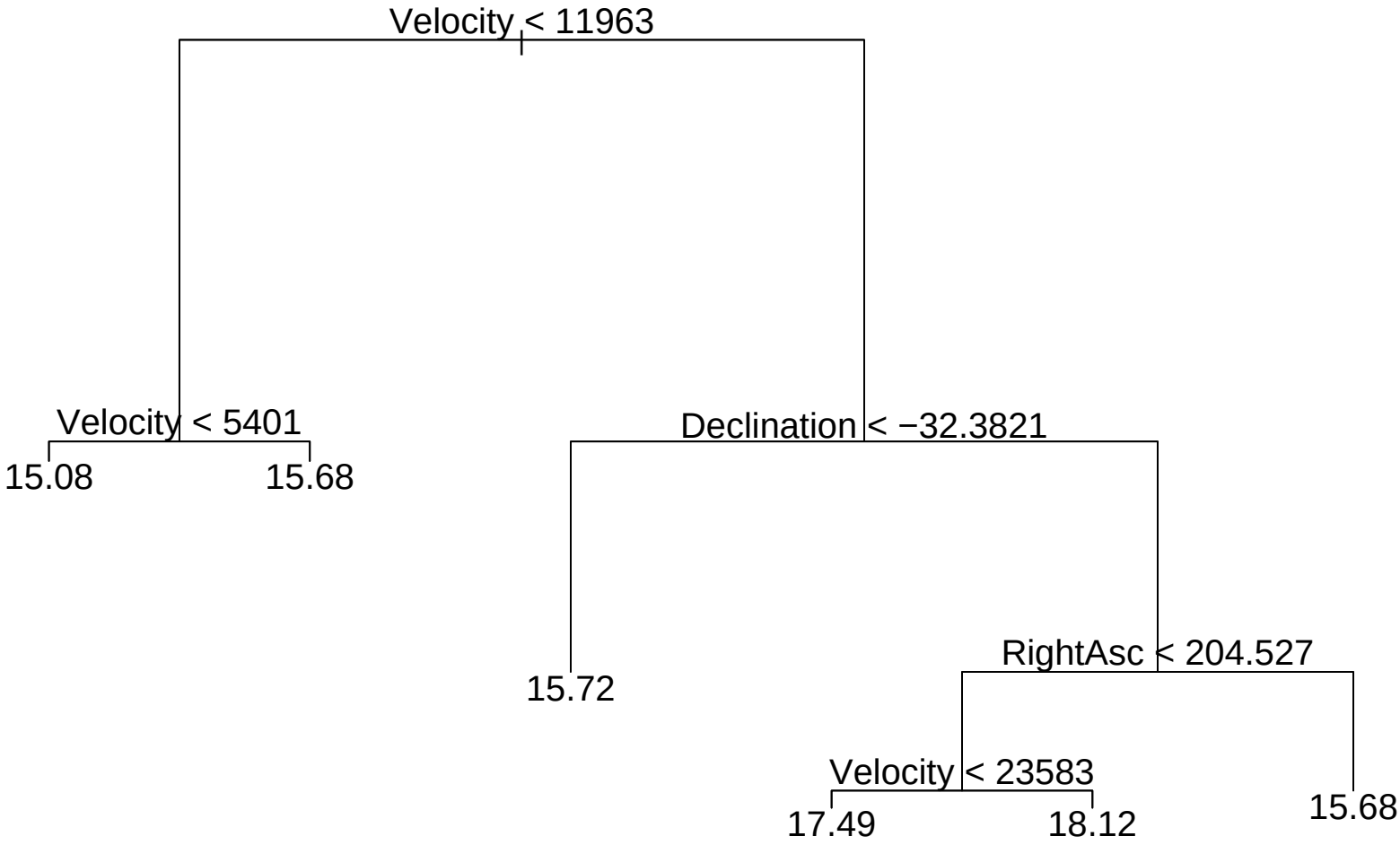
Plot using `plot.tree()` and `text.tree()`:

```
> postscript(file="galtree6.eps",width=10,height=7)
> plot(galtree)
> text(galtree,cex=1.3)
> dev.off()
```

Note that using `plot()` with an object of type “tree” is equivalent to using `plot.tree()`.

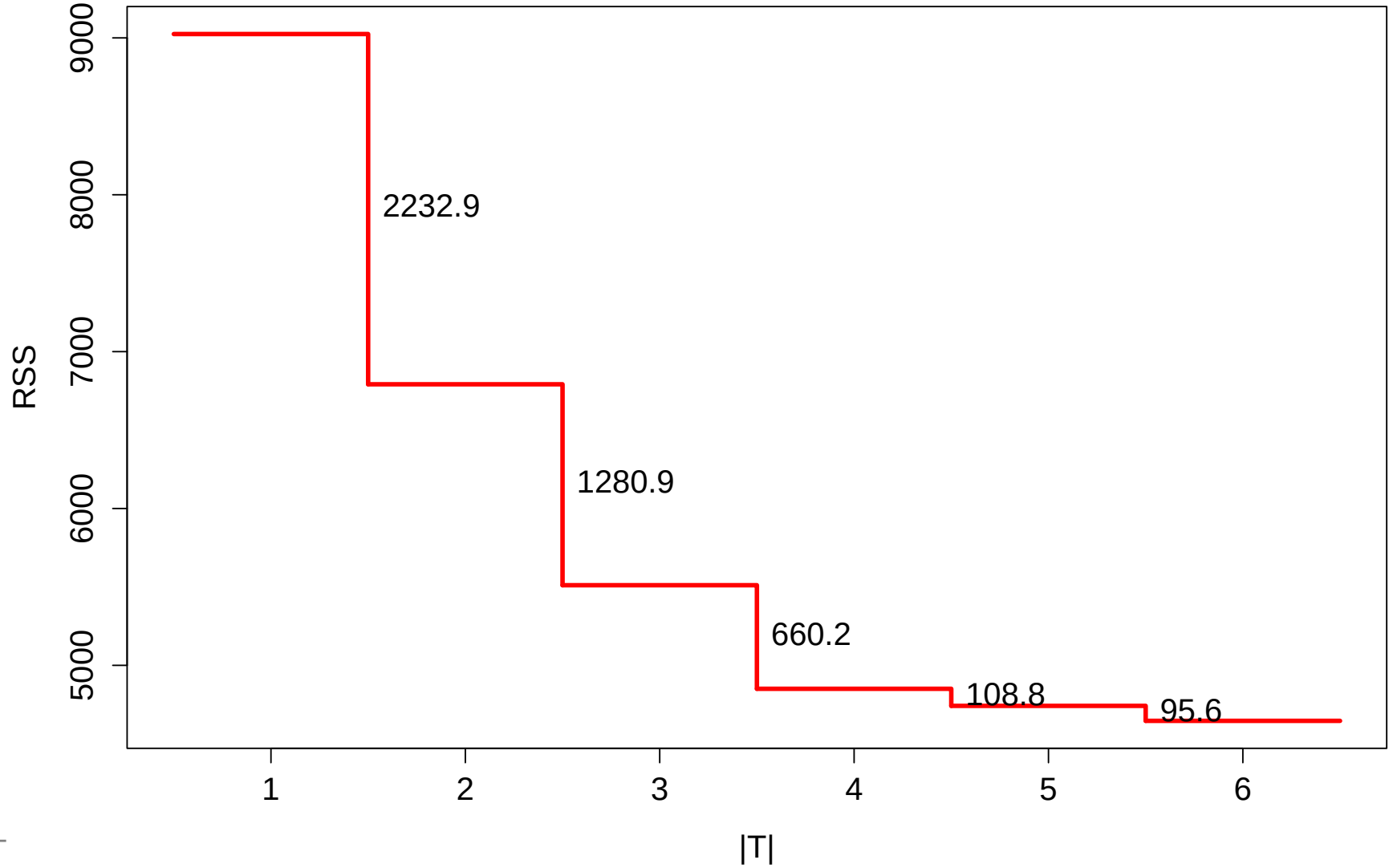
Example

Galaxy data, regression tree, response `Magnitude`



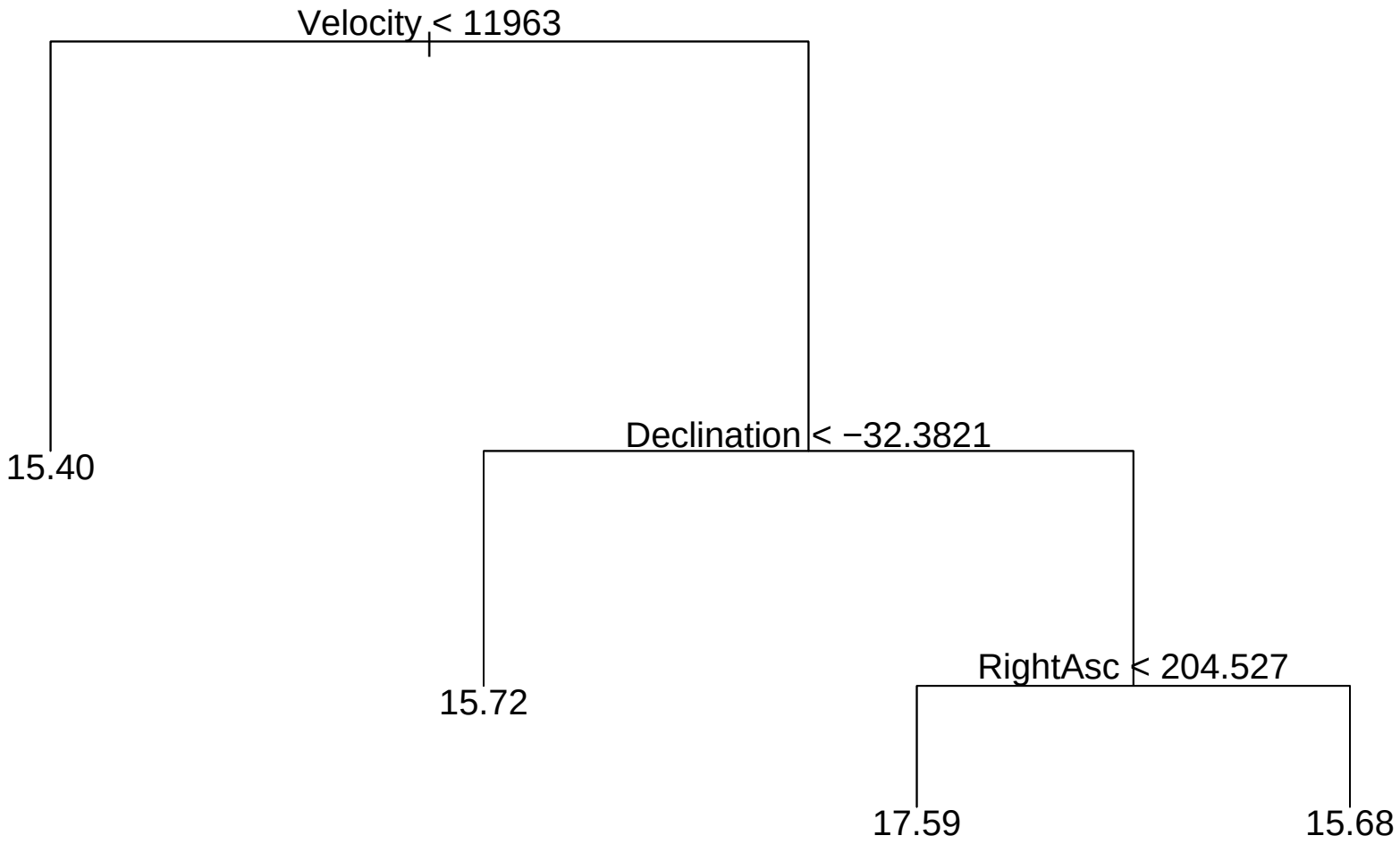
Example

Galaxy data, RSS as a function of $|T|$.



Example

Galaxy data, pruned tree with $k = 109$, thus $|T| = 4$.



Density Estimation

Observe

$$X_1, \ldots, X_n \sim f.$$

Want to estimate f. Methods include:

1. binning (histogram)
2. kernel estimator
3. local likelihood
4. wavelets

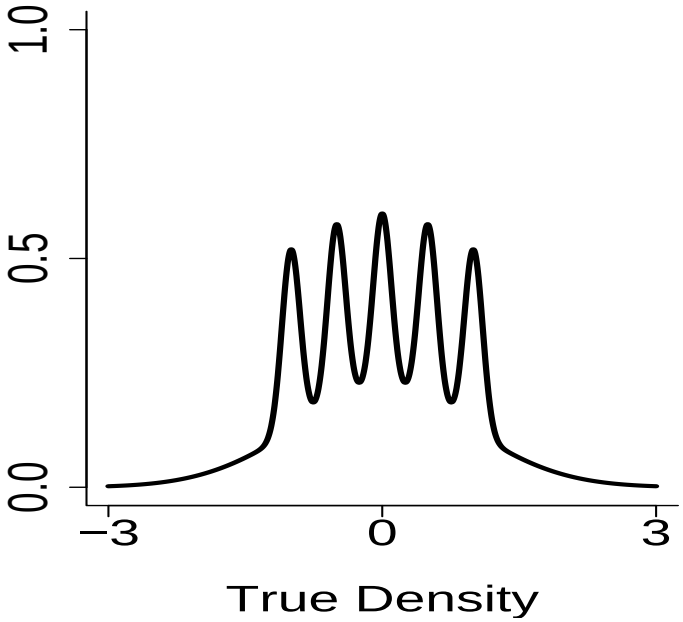
Density Estimation

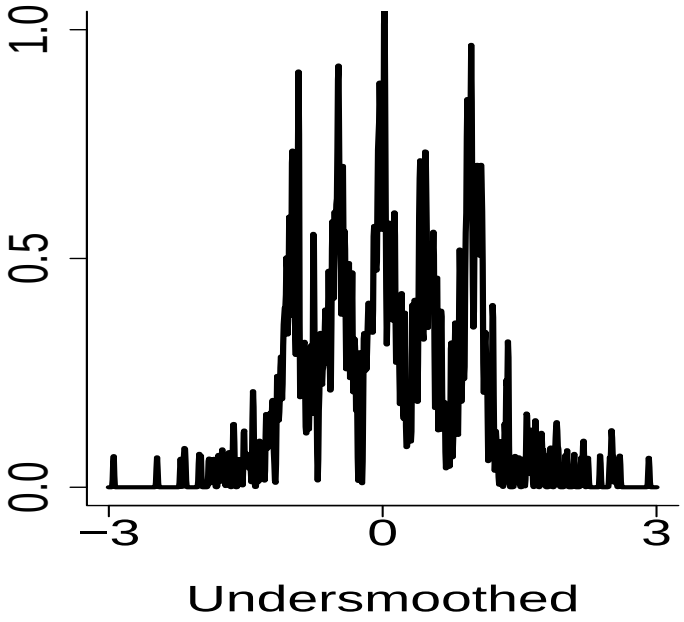
Example: Bart Simpson.

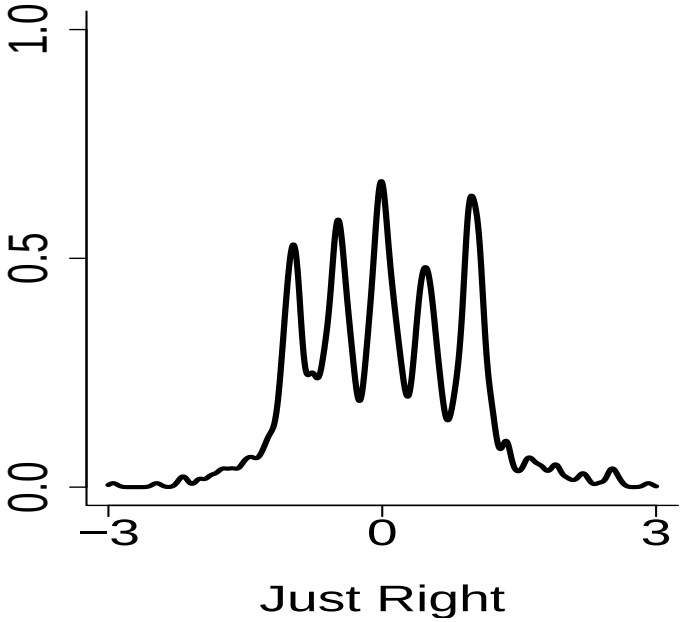
$$f(x) = \frac{1}{2}\phi(x; 0, 1) + \frac{1}{10}\sum_{j=0}^{4} \phi(x; (j/2) - 1, 1/10)$$

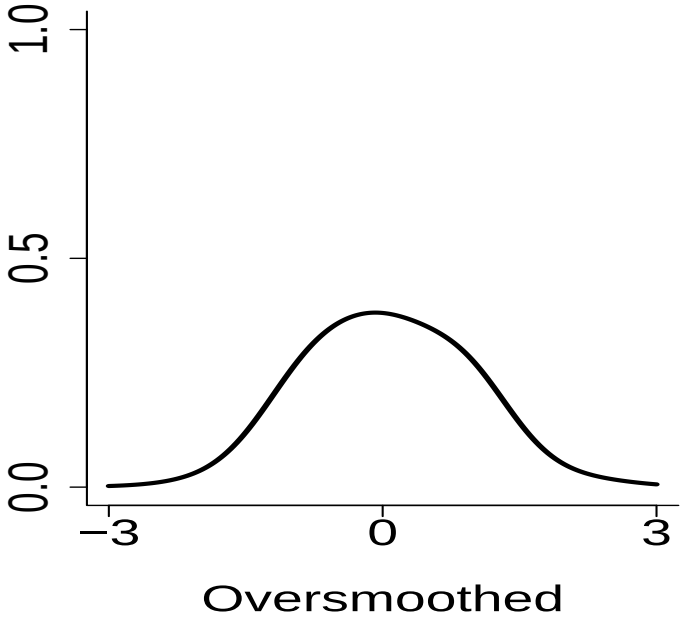
where $\phi(x; \mu, \sigma)$ denotes a Normal density with mean μ and standard deviation σ. This is a nasty density.

Density Estimation









histogram

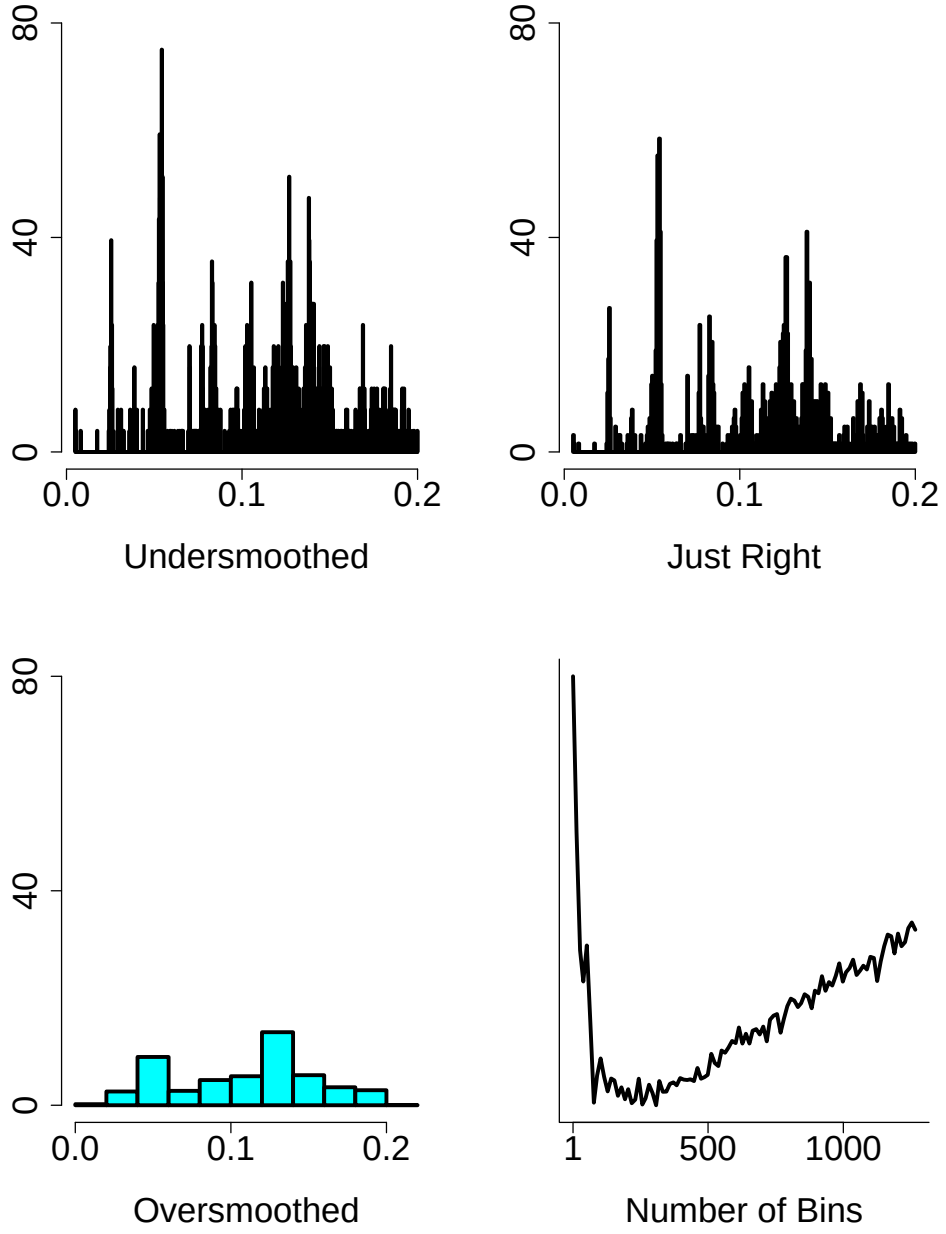
Create bins $B_1, B_2, \ldots, B_m$ of width h. Define

$$\widehat{f}_n(x) = \sum_{j=1}^{m} \frac{\widehat{p}_j}{h} I(x \in B_j).$$

where $\widehat{p}_j$ is proportion of observations in B_j. Note that

$$\int \widehat{f}_n(x) dx = 1.$$

redshift data



Theory

Loss

$$\int (f(x) - \widehat{f}_n(x))^2 dx$$

Risk (MSE)

$$R = \mathbb{E}\left(\int (f(x) - \widehat{f}_n(x))^2 dx \right) \approx \frac{h^2}{12} \int (f'(u))^2 du + \frac{1}{nh}$$

The value h^* that minimizes this is

$$h^* = \frac{1}{n^{1/3}} \left(\frac{6}{\int (f'(u))^2 du} \right)^{1/3}.$$

and then

$$R(\widehat{f}_n, f) \sim \frac{C}{n^{2/3}}$$

Cross-Validation

$$\begin{aligned} L(h) &= \int (\widehat{f}_n(x) - f(x))^2 \, dx \\ &= \int \widehat{f}_n^2(x) \, dx - 2 \int \widehat{f}_n(x) f(x) dx + \int f^2(x) \, dx. \end{aligned}$$

The last term does not depend on h so minimizing the loss is equivalent to minimizing the expected value of

$$J(h) = \int \widehat{f}_n^2(x) \, dx - 2 \int \widehat{f}_n(x) f(x) dx.$$

Estimate this by

$$\widehat{J}(h) = \int \left(\widehat{f}_n(x)\right)^2 dx - \frac{2}{n} \sum_{i=1}^{n} \widehat{f}_{(-i)}(X_i)$$

where $\widehat{f}_{(-i)}$ is the density estimator obtained after removing the i^{th} observation.

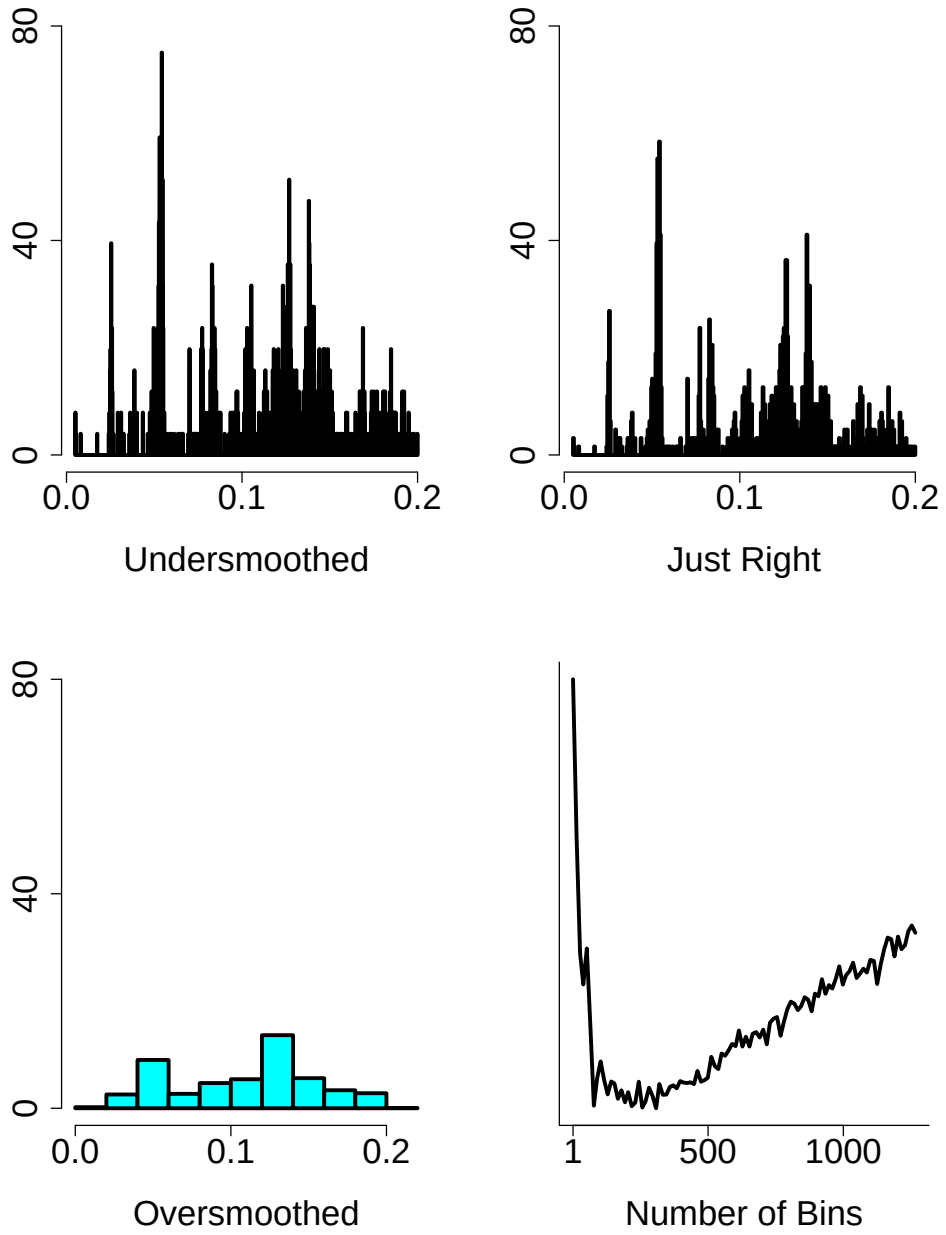
Cross-Validation

$$\widehat{J}(h) = \int \left(\widehat{f}_n(x)\right)^2 dx - \frac{2}{n}\sum_{i=1}^{n} \widehat{f}_{(-i)}(X_i)$$

Can show that, for histograms,

$$\widehat{J}(h) = \frac{2}{h(n-1)} - \frac{n+1}{h(n-1)}\sum_{j=1}^{m} \widehat{p}_j^2.$$

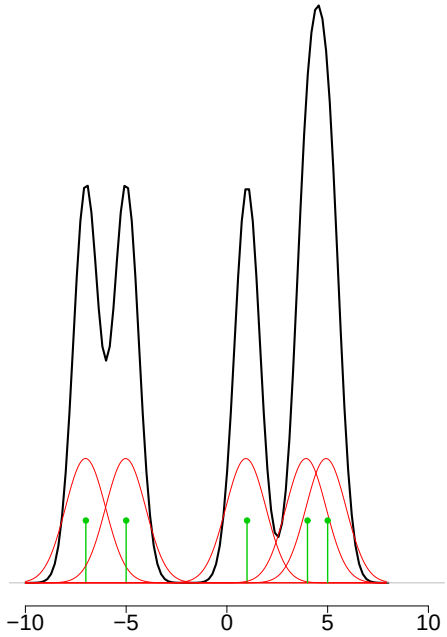
Cross-Validation



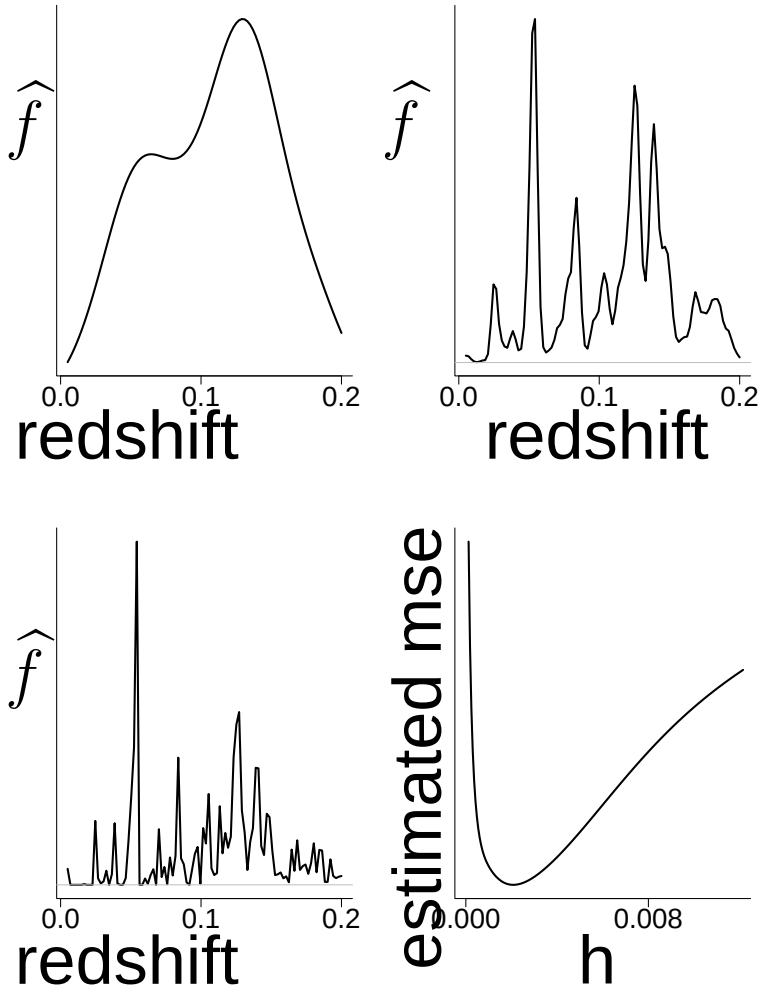
Kernel Density Estimation

$$\widehat{f}_n(x) = \frac{1}{n}\sum_{i=1}^{n} \frac{1}{h} K\left(\frac{x - X_i}{h}\right).$$

This amounts to placing a smoothed out lump of mass of size $1/n$ over each data point X_i.



Kernel Density Estimation



Theory

$$R \approx \frac{1}{4}\sigma_K^4 h_n^4 \int (f''(x))^2 dx + \frac{\int K^2(x)dx}{nh}$$

$$h_* = \left(\frac{c_2}{c_1^2 A(f) n} \right)^{1/5}$$

where $c_1 = \int x^2 K(x)dx$, $c_2 = \int K(x)^2 dx$ and $A(f) = \int (f''(x))^2 dx$.

$$R = O(n^{-4/5}).$$

As we saw, histograms converge at rate $O(n^{-2/3})$ showing that kernel estimators are superior in rate to histograms. There does not exist an estimator that converges faster than $O(n^{-4/5})$.

Bandwidth Selection

For smooth densities and a Normal kernel, use the bandwidth

$$h_n = \frac{1.06\,\widehat{\sigma}}{n^{1/5}}$$

where

$$\widehat{\sigma} = \min\left\{s,\ \frac{Q}{1.34}\right\}.$$

Recall that the cross-validation score is

$$\widehat{J}(h) = \int \widehat{f}^2(x)dx - \frac{2}{n}\sum_{i=1}^{n}\widehat{f}_{-i}(X_i).$$

In fact,

$$\widehat{J}(h) = \frac{1}{hn^2}\sum_i\sum_j K^*\left(\frac{X_i - X_j}{h}\right) + \frac{2}{nh}K(0)$$

where $K^*(x) = K^{(2)}(x) - 2K(x)$ and $K^{(2)}(z) = \int K(z -$

Density Estimation with R

See `hist()` to make histograms
See `density()` for kernel density estimator and `bw.nrd()` for bandwidth selection.

Measurement Error

Suppose we are interested in regressing the outcome Y on a covariate X but we cannot observe X directly. Rather, we observe X plus noise U.

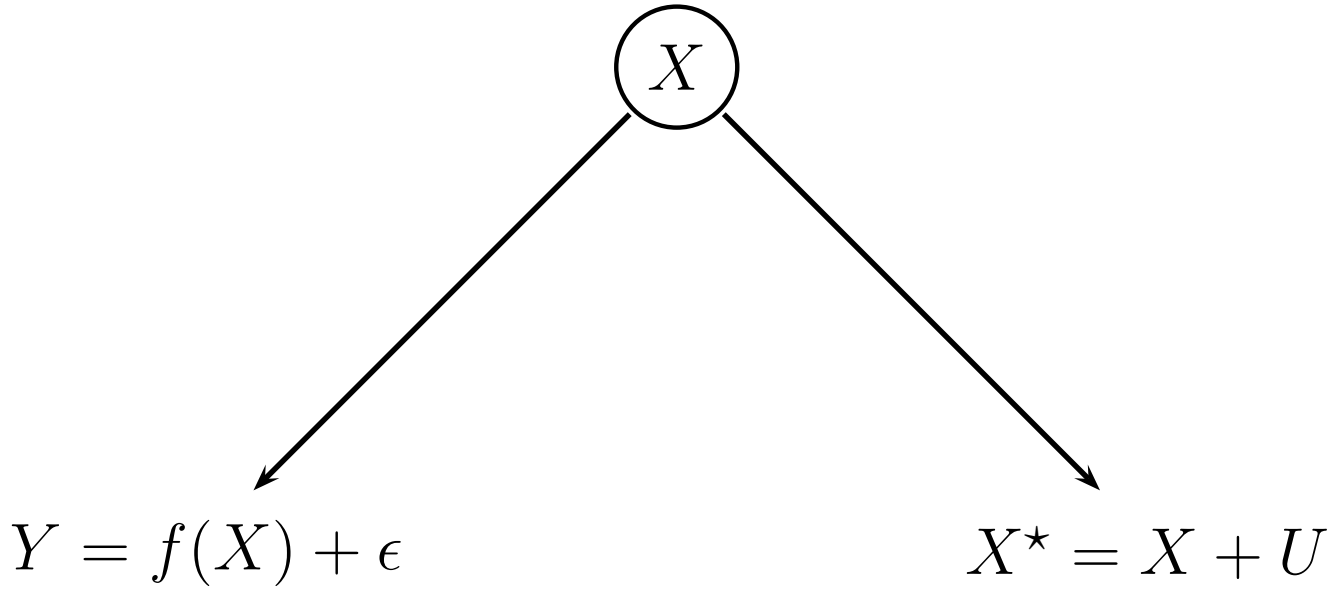
The observed data are $(X_1^\star, Y_1), \ldots, (X_n^\star, Y_n)$ where

$$\begin{aligned} Y_i &= f(X_i) + \epsilon_i \\ X_i^\star &= X_i + U_i, \qquad \mathbb{E}(U_i) = 0. \end{aligned}$$

This is called a **measurement error** problem or an **errors-in-variables** problem.

It is tempting to ignore the error and just regress Y on $X^\star$ but this leads to inconsistent estimates of $f(x)$.

Measurement Error



X is circled to show that it is not observed. $X^\star$ is a noisy version of X. If you regress Y on $X^\star$, you will get an inconsistent estimate of $f(x)$.

Measurement Error

Start with linear regression. The model is

$$\begin{aligned} Y_i &= \beta_0 + \beta_1 X_i + \epsilon_i \\ X_i^\star &= X_i + U_i. \end{aligned}$$

Let $\widehat{\beta}_1$ be the least squares estimator of β_1 obtained by regressing the Y_i's on the $X_i^\star$'s. Then

$$\widehat{\beta} \xrightarrow{as} \lambda\,\beta_1$$

where $\lambda = \frac{\sigma_x^2}{\sigma_x^2+\sigma_u^2} < 1.$

This is called **attenuation bias.**

Measurement Error

A similar result holds for nonparametric regression. Local estimator has excess bias of

$$\sigma_u^2 \left(\frac{g'(x)}{g(x)} f'(x) + \frac{f''(x)}{2} \right)$$

where g is the density of X.

Measurement Error: Linear Case

Since, $\sigma_\star^2 = \sigma_x^2 + \sigma_u^2$, we can estimate σ_x^2 by

$$\widehat{\sigma}_x^2 = \widehat{\sigma}_\star^2 - \sigma_u^2$$

where $\widehat{\sigma}_\star^2$ is the sample variance of the $X_i^\star$s. An estimate of β_1 is

$$\widetilde{\beta}_1 = \frac{\widehat{\beta}_1}{\widehat{\lambda}} = \frac{\widehat{\sigma}_\star^2}{\widehat{\sigma}_\star^2 - \sigma_u^2}\widehat{\beta}_1.$$

This is called the **method of moments estimator**

SIMEX

Another method for correcting the attenuation bias is SIMEX which stands for **simulation extrapolation** (Cook and Stefanksi).

Recall that the least squares estimate $\widehat{\beta}_1$ is a consistent estimate of

$$\frac{\beta_1 \sigma_x^2}{\sigma_x^2 + \sigma_u^2}.$$

Generate new random variables

$$\widetilde{X}_i = X_i^\star + \sqrt{\rho}\sigma_u U_i$$

where $U_i \sim N(0,1)$.

SIMEX

The least squares estimate obtained by regressing the Y_i's on the $\widetilde{X}_i$'s is a consistent estimate of

$$\Omega(\rho) = \frac{\beta_1 \sigma_x^2}{\sigma_x^2 + (1+\rho)\sigma_u^2}. \tag{5}$$

Repeat this process B times (where B is large) and denote the resulting estimators by $\widehat{\beta}_{1,1}(\rho), \ldots, \widehat{\beta}_{1,B}(\rho)$.

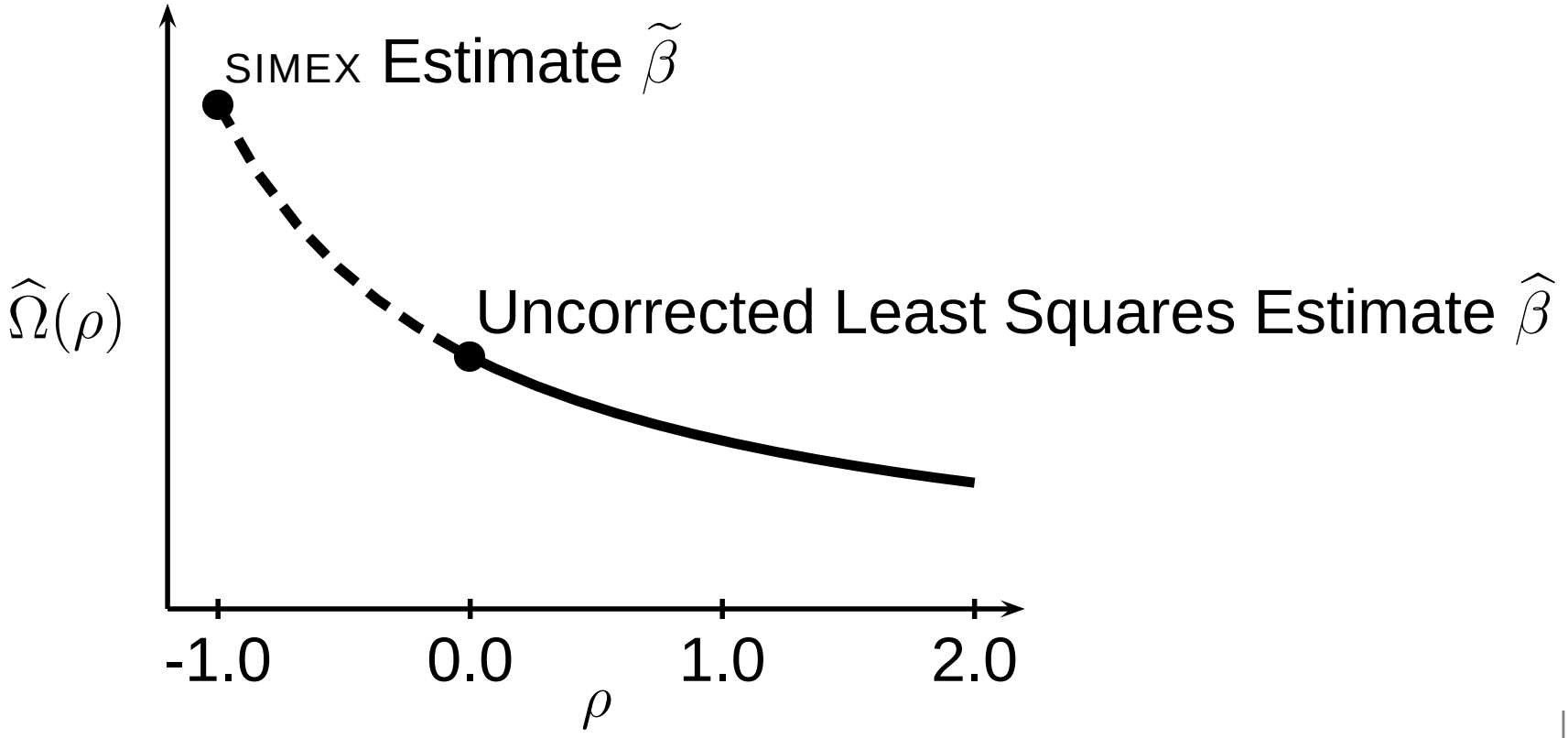
Then define

$$\widehat{\Omega}(\rho) = \frac{1}{B} \sum_{b=1}^{B} \widehat{\beta}_{1,b}(\rho).$$

Setting $\rho = -1$ in (**??**) we see that $\Omega(-1) = \beta_1$ which is the quantity we want to estimate.

SIMEX

Compute $\widehat{\Omega}(\rho)$ for a range of values of ρ such as $0, 0.5, 1.0, 1.5, 2.0$. Then extrapolate the curve $\widehat{\Omega}(\rho)$ back to $\rho = -1$ using quadratic regression.



Nonparametric Case

An advantage of SIMEX is that it extends readily to nonparametric regression. Let $\widehat{f}_n(x)$ be an uncorrected estimate of $f(x)$ obtained by regressing the Y_i's on the $X_i^\star$'s in the nonparametric problem

$$\begin{aligned} Y_i &= f(X_i) + \epsilon_i \\ X_i^\star &= X_i + U_i. \end{aligned}$$

Now perform the **simex** algorithm to get $\widehat{f}_n(x, \rho)$ and define the corrected estimator $\widetilde{f}_n(x) = \widehat{f}_n(x, -1)$.

Nonparametric Case

A more direct way to deal with measurement error is suggested by Fan and Truong. They propose the kernel estimator

$$\widehat{f}_n(x) = \frac{\sum_{i=1}^n K_n\left(\frac{x-X_i^\star}{h_n}\right) Y_i}{\sum_{i=1}^n K_n\left(\frac{x-X_i^\star}{h_n}\right)}$$

where

$$K_n(x) = \frac{1}{2\pi}\int e^{-itx}\frac{\phi_K(t)}{\phi_U(t/h_n)}dt,$$

where ϕ_K is the Fourier transform of a kernel K and ϕ_U is the characteristic function of U.

Nonparametric Case

Yet another way. Write the uncorrected estimator as

$$\widehat{f}_n(x) = \sum_{i=1}^{n} Y_i \, \ell_i(x, X_i^\star).$$

If the X_i's had been observed, the estimator of r would be

$$f_n^*(x) = \sum_{i=1}^{n} Y_i \, \ell_i(x, X_i).$$

Expanding $\ell_i(x, X_i^\star)$ around X_i we have

$$\widehat{f}_n(x) \approx f_n^*(x) + \sum_{i=1}^{n} Y_i (X_i^\star - X_i) \ell'(x, X_i) + \frac{1}{2} \sum_{i=1}^{n} Y_i (X_i^\star - X_i)^2 \ell''(x, X_i).$$

Nonparametric Case

Taking expectations, we see that the excess bias due to measurement error (conditional on the X_is) is

$$b(x) = \frac{\sigma_u^2}{2} \sum_{i=1}^{n} f(X_i) \ell''(x, X_i).$$

We can estimate $b(x)$ with

$$\widehat{b}(x) = \frac{\sigma_u^2}{2} \sum_{i=1}^{n} \widehat{f}(X_i^\star) \ell''(x, X_i^\star).$$

Example

The function `locfitsimex()` allows you to estimate $f(x)$ via local linearusing SIMEX to correct for bias due to measurement error.

```
> locfitsimex(xerr, y, measerrvar, simexreps, h, deg=deg)
```

This function returns a matrix.

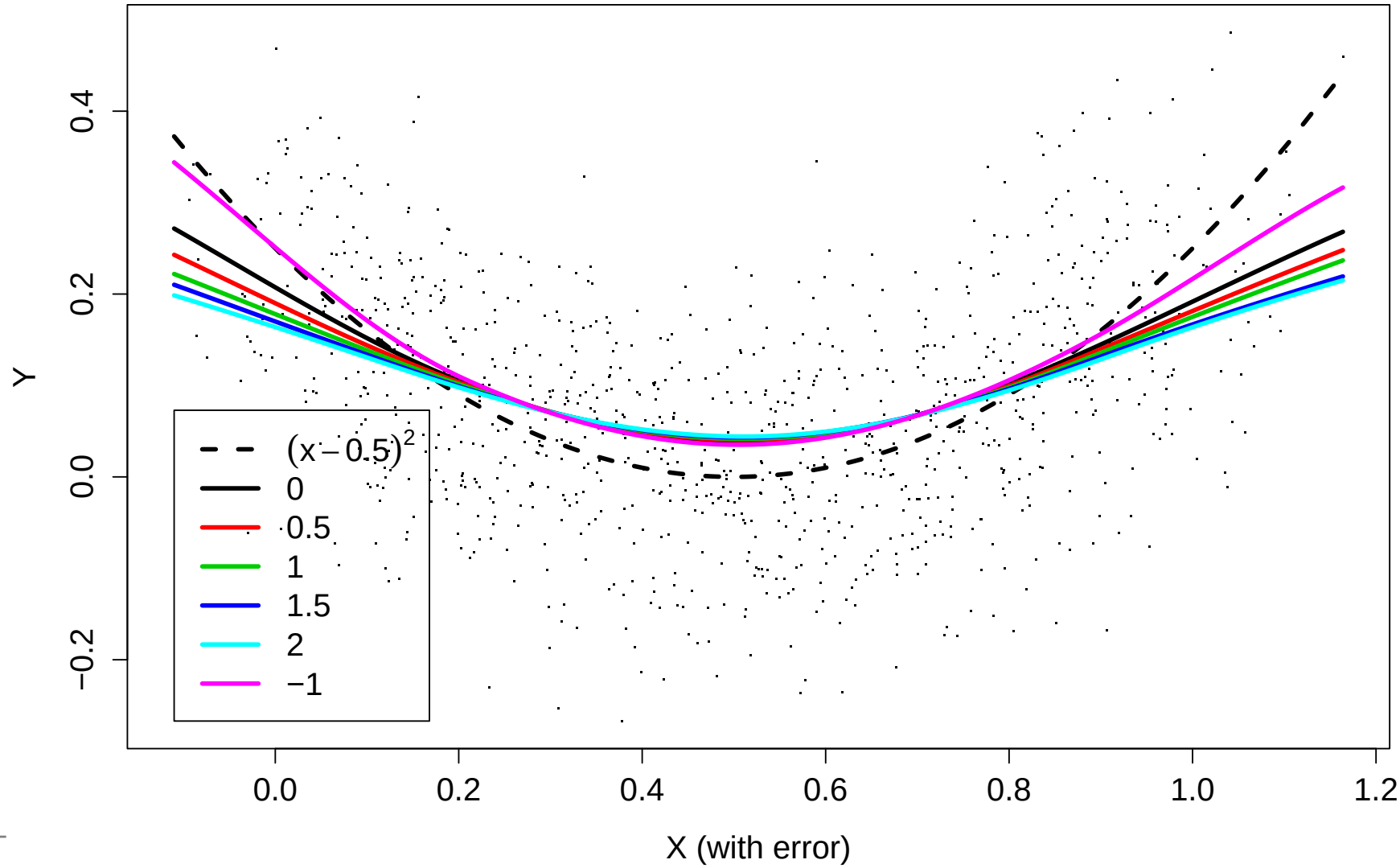
Rows correspond to $\rho = 0, 0.5, 1, 1.5, 2, -1$.

Columns are the fit evaluated at 100 X values ranging from `min(xerr)` to `max(xerr)`, i.e.

```
> seq(min(xerr),max(xerr),length=100)
```

Example

Simulated data, SIMEX with `locfit()`, $\sigma_u^2 = 0.005$.



Inverse Problems

Suppose

$$Y_i = T_i(f) + \epsilon_i, \quad i = 1, \ldots, n$$

For example, blurring,

$$T_i(r) = \int K_i(s) r(s) ds$$

where $K_i(s) = e^{-(s-x_i)^2/2}$. If K_i is a delta function at x_i, then this becomes the usual nonparametric regression model $Y_i = f(x_i) + \epsilon_i$.

Inverse Problems

If

$$\widehat{f}_n(x) = \sum_{i=1}^{n} Y_i \, \ell_i(x).$$

then

$$\mathbb{E}(\widehat{f}_n(x)) = \sum_{i=1}^{n} \ell_i(x) \int K_i(s) r(s) ds = \int A(x,s) r(s) ds$$

where

$$A(x,s) = \sum_{i=1}^{n} \ell_i(x) K_i(s)$$

is called the **Backus–Gilbert averaging kernel.**

Inverse Problems

Suppose $f(x) = \sum_{j=1}^{k} \theta_j \phi_j(x)$. Then,

$$\int K_i(s) r(s) ds = \int K_i(s) \sum_{j=1}^{k} \theta_j \phi_j(s) ds = Z_i^T \theta$$

where $\theta = (\theta_1, \ldots, \theta_k)^T$ and

$$Z_i = \begin{pmatrix} \int K_i(s)\phi_1(s)ds \\ \int K_i(s)\phi_2(s)ds \\ \vdots \\ \int K_i(s)\phi_k(s)ds \end{pmatrix}.$$

Inverse Problems

The model can then be written as

$$Y = Z\theta + \epsilon$$

where Z is an $n \times k$ matrix with i^{th} row equal to Z_i^T, $Y = (Y_1, \ldots, Y_n)^T$ and $\epsilon = (\epsilon_1, \ldots, \epsilon_n)^T$.

It is tempting to estimate θ by the least squares estimator $(Z^T Z)^{-1} Z^T Y$. This may fail since $Z^T Z$ is typically not invertible in which case the problem is said to be **ill-posed.**

This is a hallmark of inverse problems: The function f cannot be recovered, even in the absence of noise, due to the information loss incurred by blurring.

Inverse Problems

Instead, it is common to use a regularized estimator such as $\widehat{\theta} = LY$ where

$$L = (Z^T Z + \lambda I)^{-1} Z^T,$$

I is the identity matrix and $\lambda > 0$ is a smoothing parameter that can be chosen by cross-validation.

Note that cross-validation is estimating the prediction error

$$n^{-1} \sum_{i=1}^{n} (\int K_i(s) r(s) ds - \int K_i(s) \widehat{r}(s) ds)^2$$

rather than

$$\int (r(s) - \widehat{r}(s))^2 ds.$$

Classification

The problem of predicting a discrete random variable Y from another random variable X is called **classification** or **supervised learning** or **discrimination** or **pattern recognition** or **machine learning.**

Consider iid data $(X_1, Y_1), \ldots, (X_n, Y_n)$ where

$$X_i = (X_{i1}, \ldots, X_{id})^T \in \mathcal{X} \subset \mathbb{R}^d$$

is a d-dimensional vector and Y_i takes values in $\{0, 1\}$.

A **classification rule** is a function $h : \mathcal{X} \to \{0, 1\}$. Observe X, predict $Y = h(X)$. The **classification risk (or error rate)** of h is

$$R(h) = \mathbb{P}(Y \neq h(X)).$$

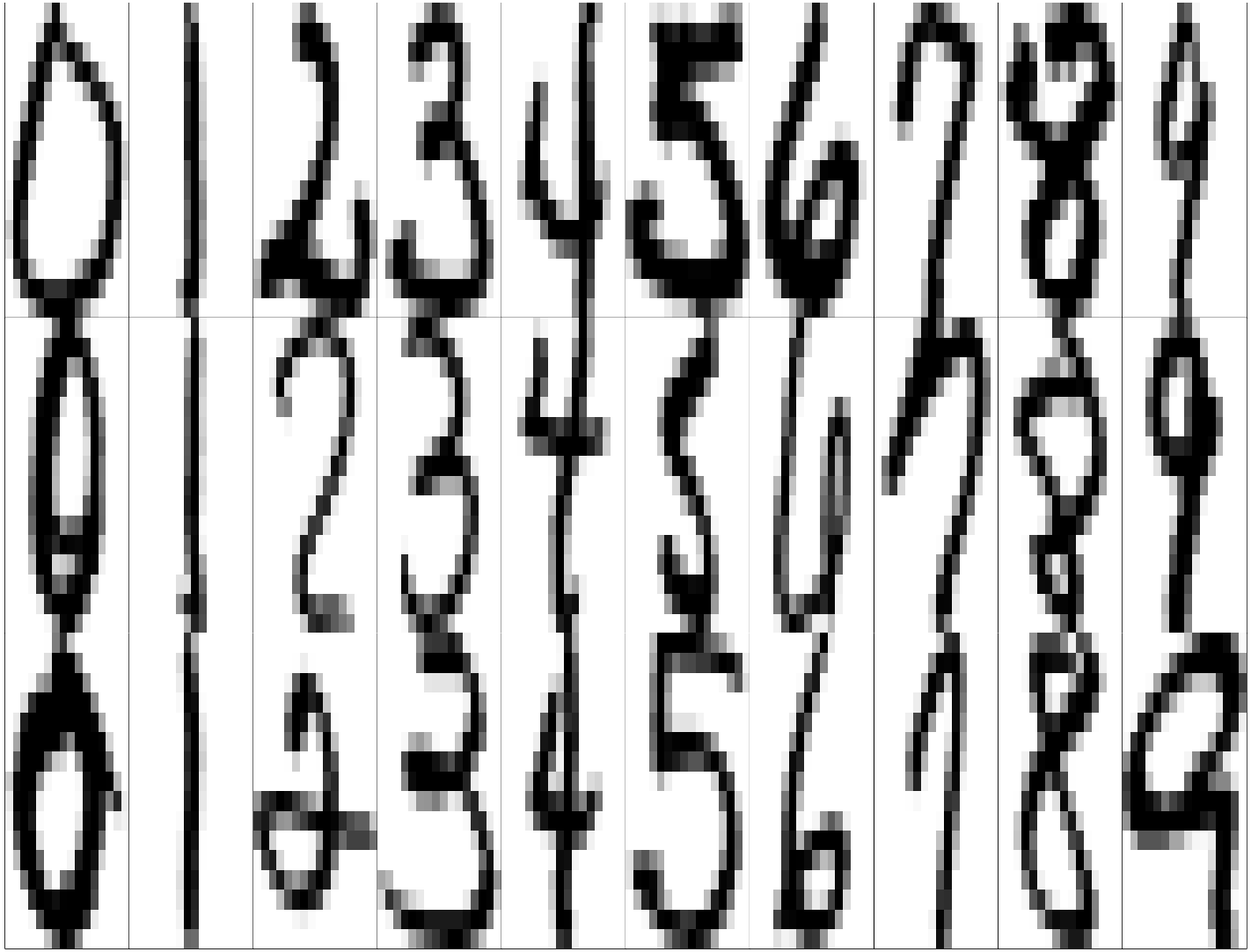
Classification

EXAMPLE:

Identify handwritten digits from images. Each Y is a digit from 0 to 9. There are 256 covariates $x_1, \ldots, x_{256}$ corresponding to the intensity values from the pixels of the 16 X 16 image.

Classification

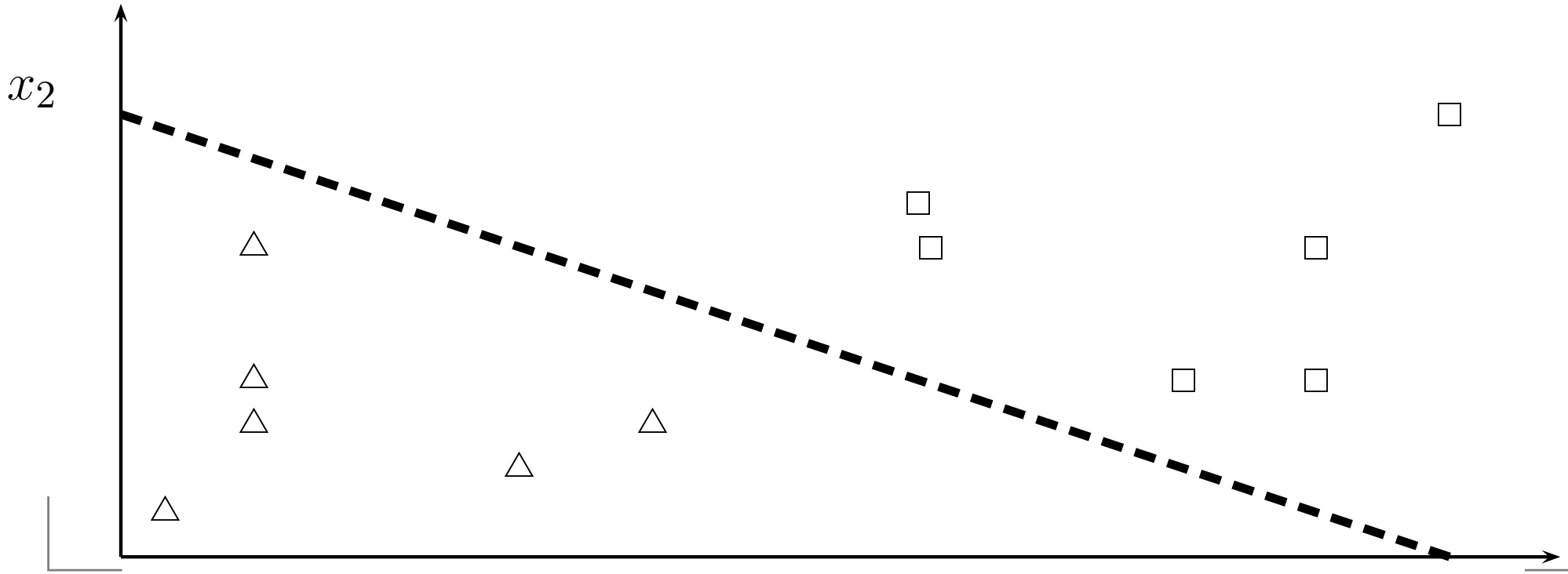
EXAMPLE:



Classification

EXAMPLE: (synthetic data)
100 data points, $d = 2$. Linear classification rule:

$$h(x) = \begin{cases} 1 & \text{if } a + b_1 x_1 + b_2 x_2 > 0 \\ 0 & \text{otherwise.} \end{cases}$$



Error Rates

The **true error rate (or classification risk)** of a classifier h is

$$R(h) = \mathbb{P}(\{h(X) \neq Y\})$$

and the **empirical error rate** or **training error rate** is

$$\widehat{R}_n(h) = \frac{1}{n}\sum_{i=1}^{n} I(h(X_i) \neq Y_i).$$

The Bayes Rule

The rule h that minimizes $R(h)$ is

$$h^*(x) = \begin{cases} 1 & \text{if } r(x) > \frac{1}{2} \\ 0 & \text{otherwise} \end{cases}$$

where

$$r(x) = \mathbb{E}(Y|X = x) = \mathbb{P}(Y = 1|X = x)$$

denote the **regression function.** The rule h^* is called the **Bayes' rule.** Note: the Bayes rule has nothing to do with Bayesian inferencd. The set

$$\mathcal{D}(h) = \{x : \ r(x) = 1/2\}$$

is called the **decision boundary.**

Three Approaches

1. **Empirical Risk Minimization** Choose a set of classifiers $\mathcal{H}$ and find $\widehat{h} \in \mathcal{H}$ that minimizes some estimate of $L(h)$.
2. **Regression.** Find an estimate $\widehat{r}$ of the regression function r and substitute into the Bayes rule.
3. **Density Estimation.** Estimate f_0 from the X_i's for which $Y_i = 0$, estimate f_1 from the X_i's for which $Y_i = 1$ and let $\widehat{\pi} = n^{-1} \sum_{i=1}^{n} Y_i$. Define

$$\widehat{r}(x) = \widehat{\mathbb{P}}(Y = 1 | X = x) = \frac{\widehat{\pi}\widehat{f}_1(x)}{\widehat{\pi}\widehat{f}_1(x) + (1 - \widehat{\pi})\widehat{f}_0(x)}$$

and

$$\widehat{h}(x) = \begin{cases} 1 & \text{if } \widehat{r}(x) > \frac{1}{2} \\ 0 & \text{otherwise.} \end{cases}$$

Linear and Logistic Regression

Regression approach is to estimate $r(x) = \mathbb{E}(Y|X = x) = \mathbb{P}(Y = 1|X = x)$. Can use linear

$$Y = r(x) + \epsilon = \beta_0 + \sum_{j=1}^{d} \beta_j X_j + \epsilon$$

or logistic

$$r(x) = \mathbb{P}(Y = 1|X = x) = \frac{e^{\beta_0 + \sum_j \beta_j x_j}}{1 + e^{\beta_0 + \sum_j \beta_j x_j}}.$$

Even if the model is wrong this might work well since we only need to approximate the decision boundary.

Nearest Neighbors

The k-nearest neighbor rule is

$$(6) h(x) = \begin{cases} 1 & \sum_{i=1}^{n} w_i(x) I(Y_i = 1) > \sum_{i=1}^{n} w_i(x) I(Y_i = 0) \\ 0 & \text{otherwise} \end{cases}$$

where $w_i(x) = 1$ if X_i is one of the k nearest neighbors of x, $w_i(x) = 0$, otherwise. "Nearest" depends on how you define the distance. Often we use Euclidean distance $||X_i - X_j||$.

Nearest Neighbors

Example: Digits.

```
> ### knn
> library(class)
> yhat = knn(train = xtrain, cl = ytrain,
          test = xtest, k = 1)
> b = table(ytest,yhat)
> print(b)
     yhat
ytest 0   1
    0 594   0
    1   0 505
> print((b[1,2]+b[2,1])/sum(b))
[1] 0
```

Nearest Neighbors

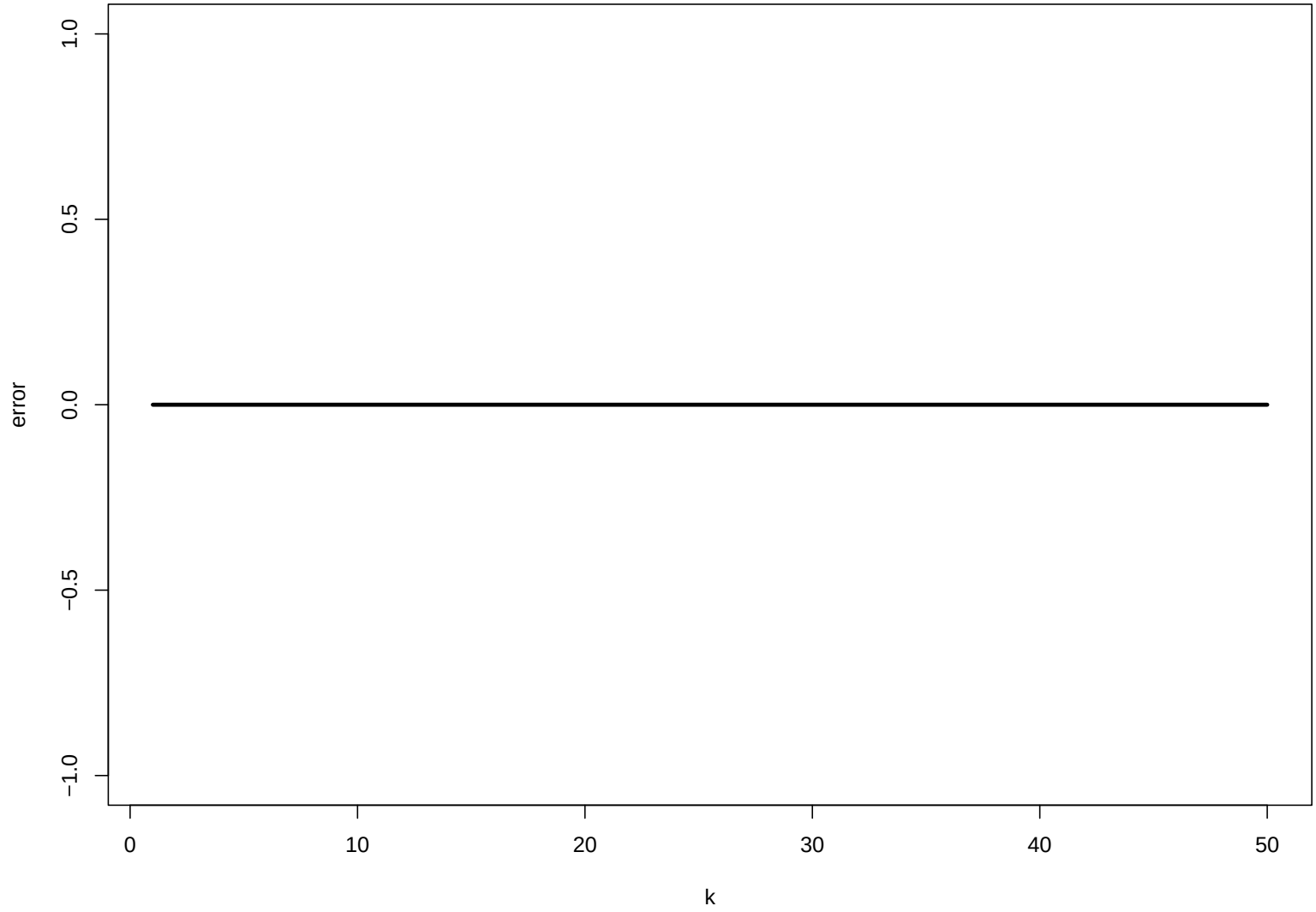
```
> yhat = knn.cv(train = xtrain,
          cl = ytrain, k = 1)
> b = table(ytrain,yhat)
> print(b)
       yhat
ytrain 0   1
     0 599   1
     1   0 500
> print((b[1,2]+b[2,1])/sum(b))
[1] 0.0009090909
```

Nearest Neighbors

Should use cross-validation to choose k. Example: South African heart disease.

```
> library(class)
> m = 50
> error = rep(0,m)
> for(i in 1:m){
      out = knn.cv(train=x,cl=y,k=i)
      error[i] = sum(y != out)/n
      }
> postscript("knn.sa.ps")
> plot(1:m,error,type="l",
      lwd=3,xlab="k",ylab="error")
> dev.off()
```

Nearest Neighbors



Gaussian, Linear and Quadratic Classifier

Suppose that $X|Y=0 \sim N(\mu_0, \Sigma_0)$ and $X|Y=1 \sim N(\mu_1, \Sigma_1)$. Then the Bayes rule is

$$h^*(x) = \text{argmax}_k \delta_k(x)$$

where

$$\delta_k(x) = -\frac{1}{2}\log|\Sigma_k| - \frac{1}{2}(x-\mu_k)^T\Sigma_k^{-1}(x-\mu_k) + \log\pi_k.$$

In practice, insert sample estimates for μ_k, Σ_k, π_k. Decision boundary is quadratic (Quadratic Discriminant Analysis). Set $\Sigma_0 = \Sigma_1 = \Sigma$ to get linear decision boundary (LDA).

Example

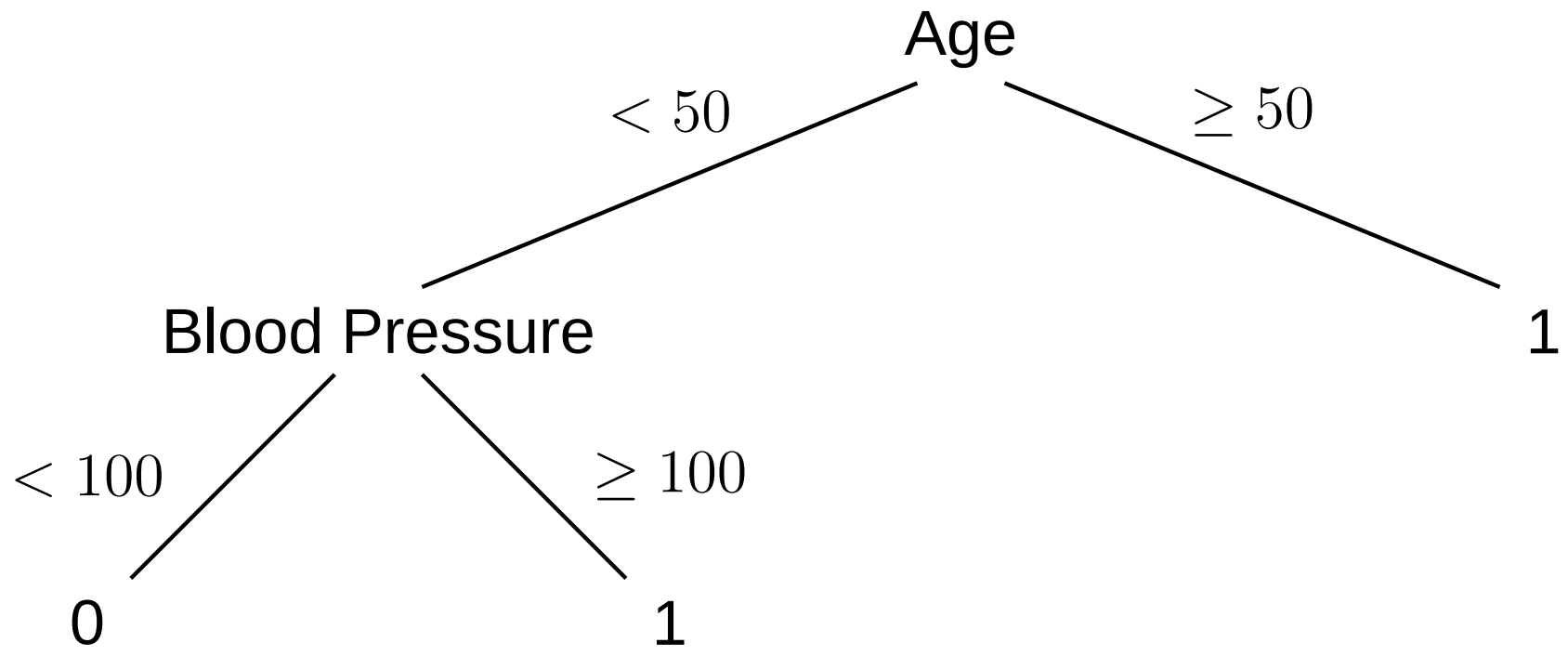
South African heart disease data. In R use:

```
> out  =  lda(x,y)  ### or qda for quadratic
> yhat = predict(out)$class
```

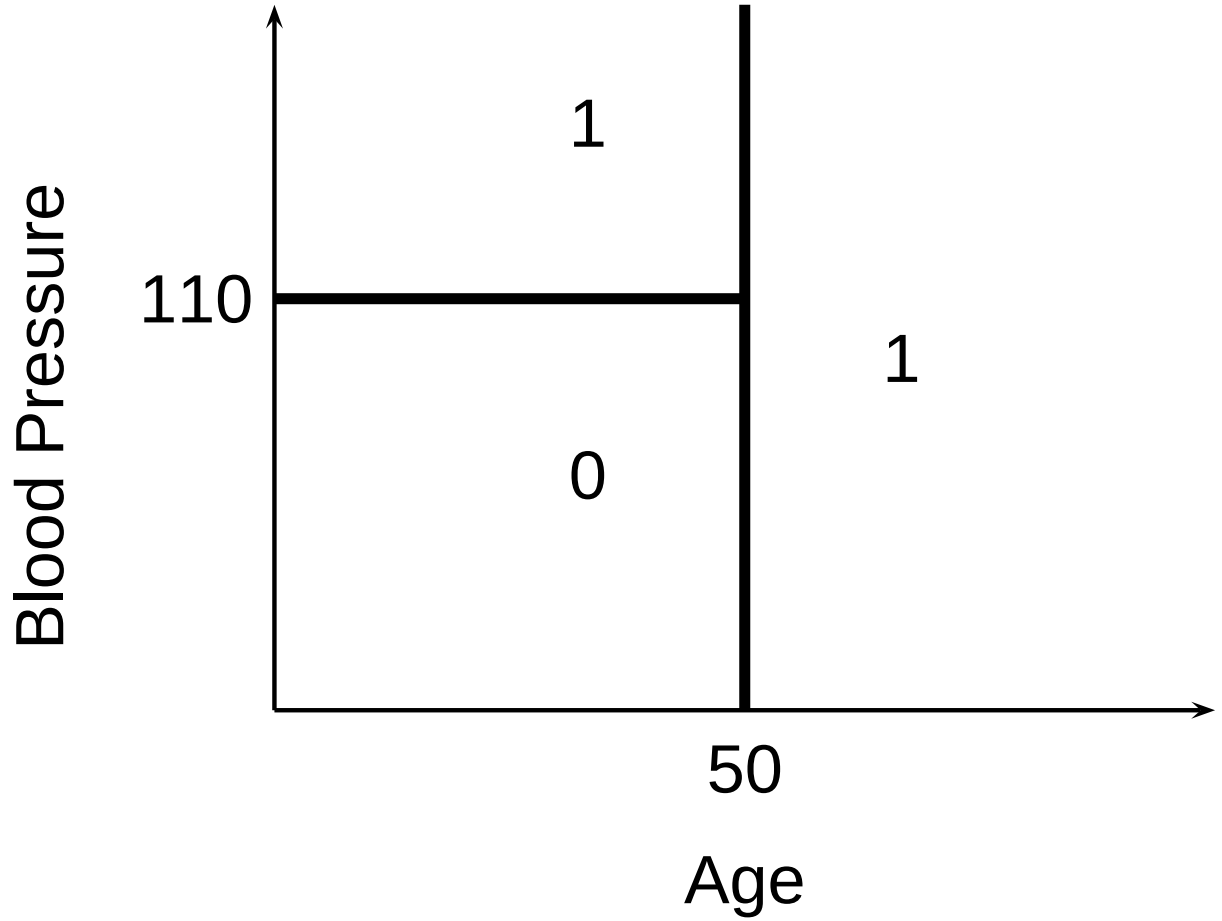
The error rate of LDA is .25. For QDA we get .24. In this example, there is little advantage to QDA over LDA.

Trees

Classification tree is like a regression tree except outcome is binary. For illustration, suppose there are two covariates, $X_1 =$ age and $X_2 =$ blood pressure.



Trees



Trees

Suppose there is a single covariate X. We choose a split point t that divides the real line into two sets $A_1 = (-\infty, t]$ and $A_2 = (t, \infty)$. Let $\widehat{p}_s(j)$ be the proportion of observations in A_s such that $Y_i = j$:

$$\widehat{p}_s(j) = \frac{\sum_{i=1}^n I(Y_i = j, X_i \in A_s)}{\sum_{i=1}^n I(X_i \in A_s)}$$

for $s = 1, 2$ and $j = 0, 1$. The **impurity** of the split t is defined to be

$$I(t) = \sum_{s=1}^{2} \gamma_s$$

where

$$\gamma_s = 1 - \sum_{j=0}^{1} \widehat{p}_s(j)^2. \quad \text{Gini index}$$

Trees

When there are several covariates, we choose whichever covariate and split that leads to the lowest impurity. This process is continued until some stopping criterion is met. For example, we might stop when every partition element has fewer than n_0 data points, where n_0 is some fixed number. The bottom nodes of the tree are called the **leaves**. Each leaf is assigned a 0 or 1 depending on whether there are more data points with $Y = 0$ or $Y = 1$ in that partition element.

Example

```
> library(tree)

> sadat = read.table("sa.dat",sep=",",header=T)
> n = dim(sadat)[[1]]
> chd = sadat[,11]

> names = c("sbp","tobacco","ldl","adiposity","famhist",
     "typea","obesity", "alcohol","age")
> for(i in 2:10){
     assign(names[i-1],sadat[,i])
}
```

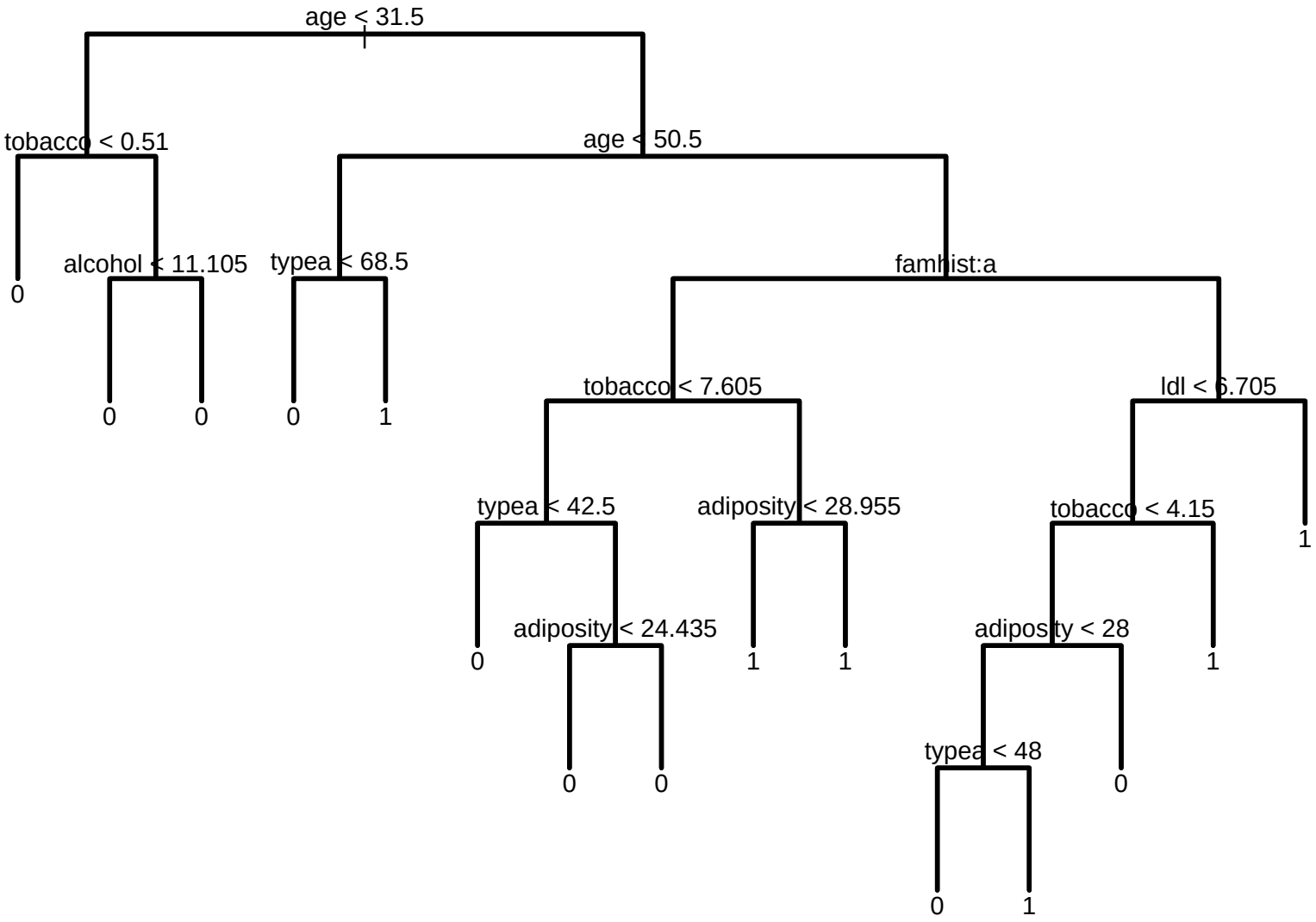
Example

```
> famhist = as.factor(famhist)
> formula = paste(names,sep="",collapse="+")
> formula = paste("chd ~ ",formula)
> formula = as.formula(formula)

> chd = as.factor(chd)
> d = data.frame(chd,sbp,tobacco,ldl,adiposity,
   famhist,typea,obesity,alcohol,age)
```

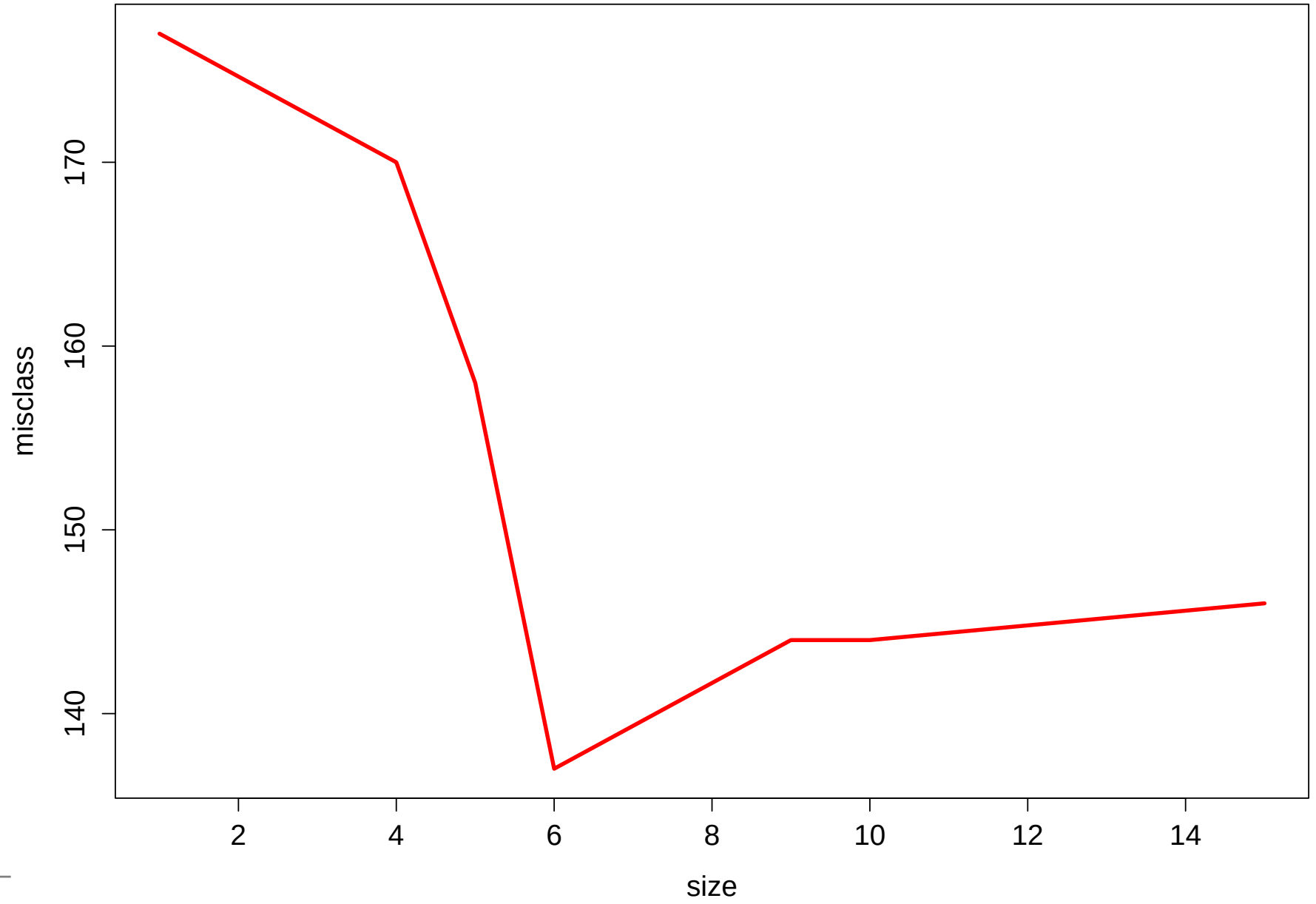
Example

```
> postscript("south.africa.tree.plot1.ps")
> out = tree(formula,data=d)
> plot(out,type="u",lwd=3)
> text(out)
> dev.off()

> cv = cv.tree(out,method="misclass")
> postscript("south.africa.tree.plot2.ps")
> plot(cv$size,cv$dev,xlab="size",ylab="misclass",
   lwd=3,type="l",col=2,cex.axis=1.3,cex.lab=1.3)
> dev.off()
```

Example


age < 31.5
tobacco < 0.51
age < 50.5
0
alcohol < 11.105
typea < 68.5
famhist:a
0
0
0
1
tobacco < 7.605
ldl < 6.705
typea < 42.5
adiposity < 28.955
tobacco < 4.15
1
adiposity < 24.435
adiposity < 28
0
1
1
1
typea < 48
0
0
0
0
1

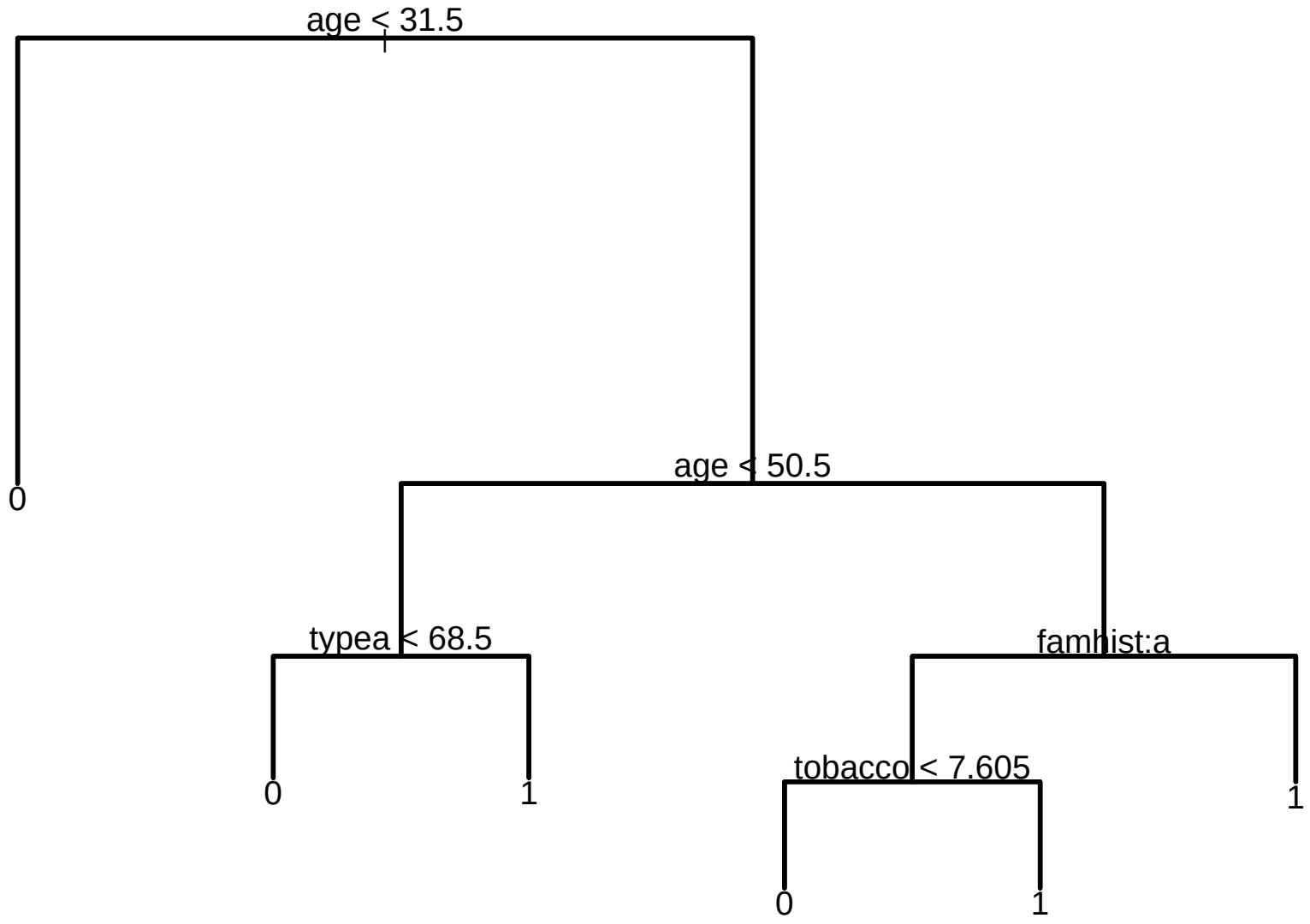
Example



Example

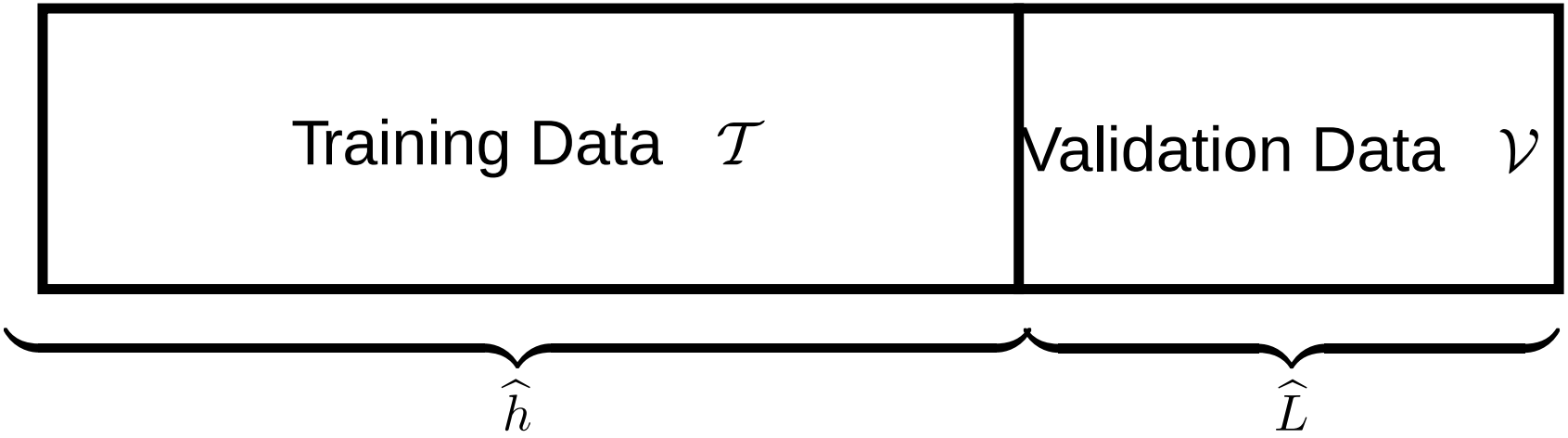
```
> newtree = prune.tree(out,best=6,method="misclass")
> print(summary(newtree))
> snip.tree(tree = out, nodes = c(2, 28, 29, 15))
> postscript("south.africa.tree.plot3.ps")
> plot(newtree,lwd=3)
> text(newtree,cex=1.3)
> dev.off()
```

Example



Cross Validation

Data Splitting:



Or use K-fold. Split data into K groups etc.

Perceptrons and Support Vector Machines

In this section we consider a class of linear classifiers called **support vector machines**. It will be convenient to label the outcomes as -1 and $+1$ instead of 0 and 1. A linear classifier can then be written as

$$h(x) = \text{sign}\Big(H(x)\Big)$$

where $x = (x_1, \ldots, x_d)$,

$$H(x) = a_0 + \sum_{i=1}^{d} a_i x_i$$

and

$$\text{sign}(z) = \begin{cases} -1 & \text{if } z < 0 \\ 0 & \text{if } z = 0 \\ 1 & \text{if } z > 0. \end{cases}$$

Support Vector Machines

Note that:

$$\begin{aligned} \text{classifier correct} &\implies Y_i H(X_i) \geq 0 \\ \text{classifier incorrect} &\implies Y_i H(X_i) \leq 0. \end{aligned}$$

The classification risk is

$$R = \mathbb{P}(Y \neq h(X)) = \mathbb{P}(Y H(X) \leq 0) = \mathbb{E}(L(Y H(X)))$$

where the loss function L is $L(a) = 1$ if $a < 0$ and $L(a) = 0$ if $a \geq 0$.

Support Vector Machines

Suppose that the data are **linearly separable**, that is, there exists a hyperplane that perfectly separates the two classes. How can we find a separating hyperplane? A separating hyperplane will minimize

$$-\sum_{\text{misclassified}} Y_i H(X_i).$$

Rosenblatt's perceptron algorithm takes starting values and updates them:

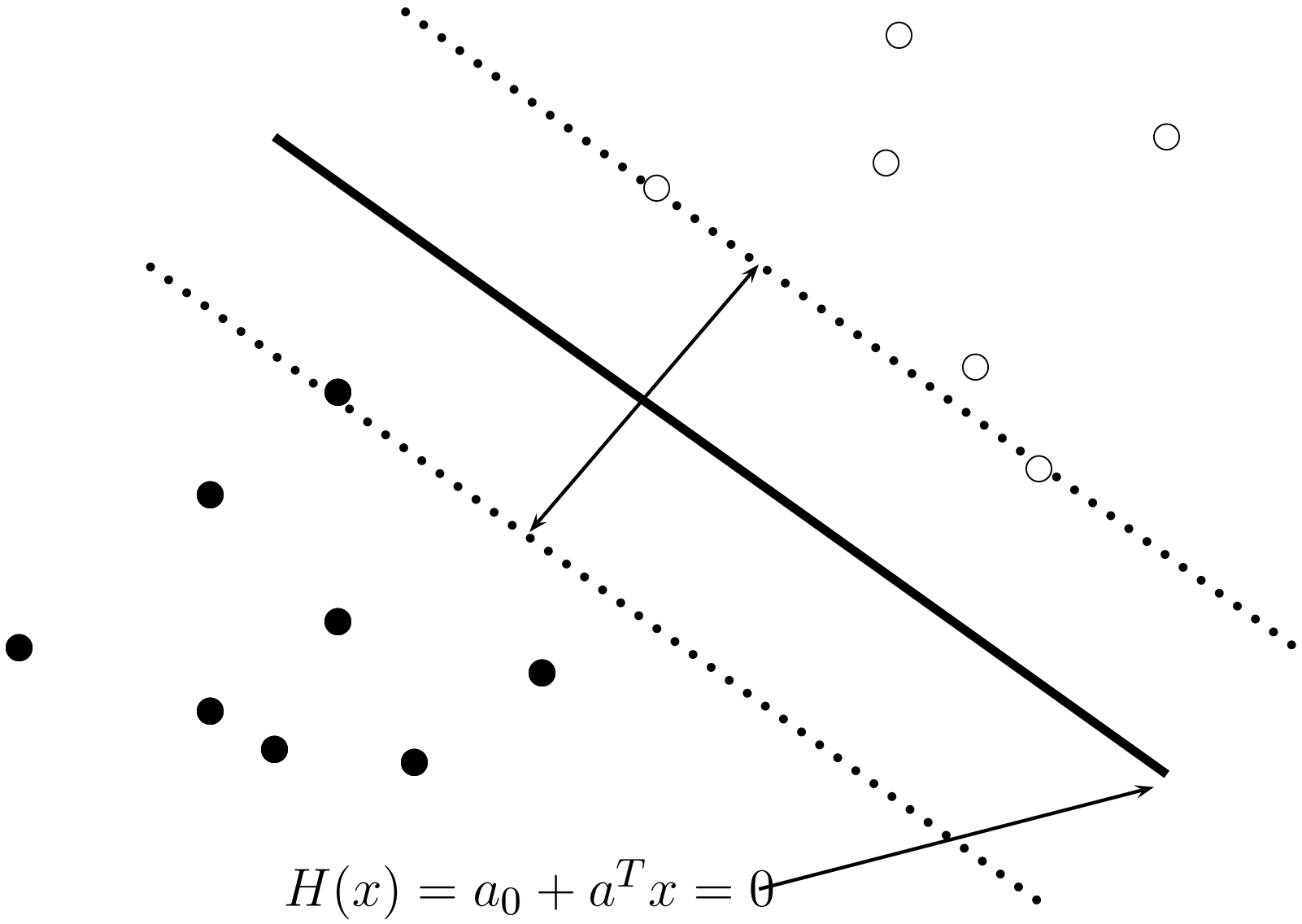
$$\begin{pmatrix} \beta \\ \beta_0 \end{pmatrix} \longleftarrow \begin{pmatrix} \beta \\ \beta_0 \end{pmatrix} + \rho \begin{pmatrix} Y_i X_i \\ Y_i \end{pmatrix}.$$

However, there are many separating hyperplanes. The particular separating hyperplane that this algorithm converges to depends on the starting values.

Support Vector Machines

Intuitively, it seems reasonable to choose the hyperplane "furthest" from the data in the sense that it separates the +1s and -1s and maximizes the distance to the closest point. This hyperplane is called the **maximum margin hyperplane.** The margin is the distance to from the hyperplane to the nearest point. Points on the boundary of the margin are called **support vectors.**

Support Vector Machines



$$H(x) = a_0 + a^T x = 0$$

Support Vector Machines

The data can be separated by some hyperplane if and only if there exists a hyperplane $H(x) = a_0 + \sum_{i=1}^{d} a_i x_i$ such that

$$Y_i H(x_i) \geq 1, \quad i = 1, \ldots, n.$$

The goal, then, is to maximize the margin, subject to this condition.

Support Vector Machines

Given two vectors a and b let $\langle a, b \rangle = a^T b = \sum_j a_j b_j$ denote the inner product of a and b. Let $\widehat{H}(x) = \widehat{a}_0 + \sum_{i=1}^d \widehat{a}_i x_i$ denote the optimal (largest margin) hyperplane. Then, for $j = 1, \ldots, d$,

$$\widehat{a}_j = \sum_{i=1}^n \widehat{\alpha}_i Y_i X_j(i)$$

where $X_j(i)$ is the value of the covariate X_j for the i^{th} data point, and $\widehat{\alpha} = (\widehat{\alpha}_1, \ldots, \widehat{\alpha}_n)$ is the vector that maximizes

$$\sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i=1}^n \sum_{k=1}^n \alpha_i \alpha_k Y_i Y_k \langle X_i, X_k \rangle$$

subject to

$$\alpha_i \geq 0, \quad \text{and} \quad 0 = \sum_i \alpha_i Y_i.$$

Support Vector Machines

The points X_i for which $\widehat{\alpha} \neq 0$ are called **support vectors.** $\widehat{a}_0$ can be found by solving

$$\widehat{\alpha}_i \Big(Y_i (X_i^T \widehat{a} + \widehat{a}_0 \Big) = 0$$

for any support point X_i. $\widehat{H}$ may be written as

$$\widehat{H}(x) = \widehat{\alpha}_0 + \sum_{i=1}^{n} \widehat{\alpha}_i Y_i \langle x, X_i \rangle.$$

Support Vector Machines

If there is no perfect linear classifier, then one allows overlap between the groups by replacing the condition with

$$Y_i H(x_i) \geq 1 - \xi_i, \quad \xi_i \geq 0, \quad i = 1, \ldots, n.$$

The variables $\xi_1, \ldots, \xi_n$ are called **slack variables.** We now maximize subject to

$$0 \leq \xi_i \leq c, \quad i = 1, \ldots, n$$

and

$$\sum_{i=1}^{n} \alpha_i Y_i = 0.$$

The constant c is a tuning parameter that controls the amount of overlap.

Support Vector Machines

In R we can use the package `e1071`.
The iris data.

```
> library(e1071)
> data(iris)
> x = iris[51:150,]
> a = x[,5]
> x = x[,-5]
> attributes(a)
$levels
[1] "setosa"     "versicolor" "virginica"

$class
[1] "factor"
```

Support Vector Machines

```
> n = length(a)
> y = rep(0,n)
> y[a == "versicolor"] = 1
> y = as.factor(y)
> out = svm(x, y)
> print(out)

Call:
 svm.default(x = x, y = y)

Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  radial
       cost:  1
      gamma:  0.25

Number of Support Vectors:  33
```

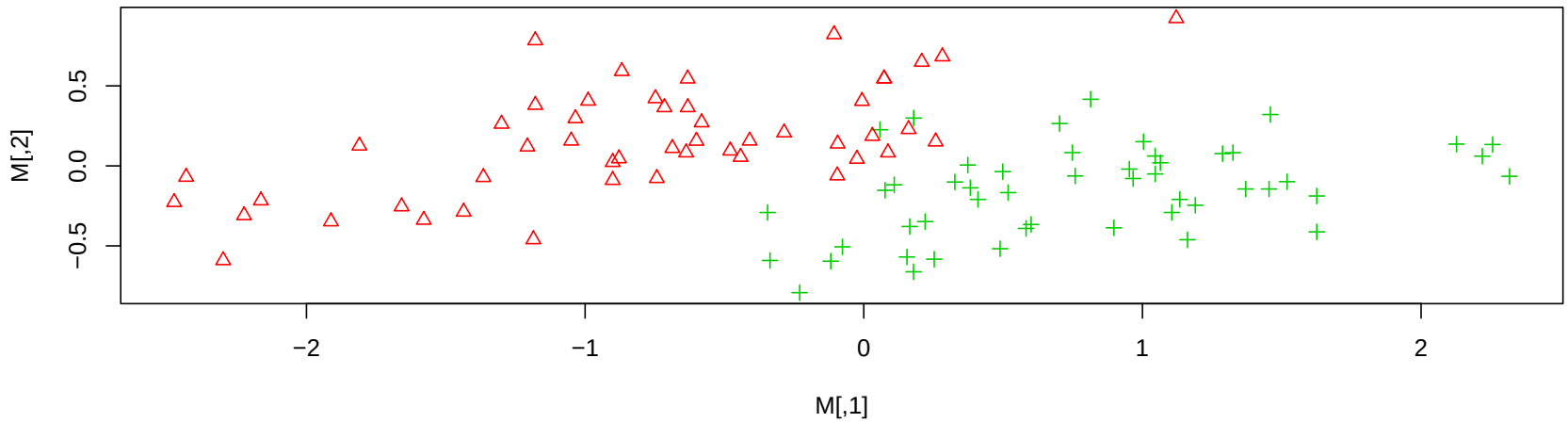
Support Vector Machines

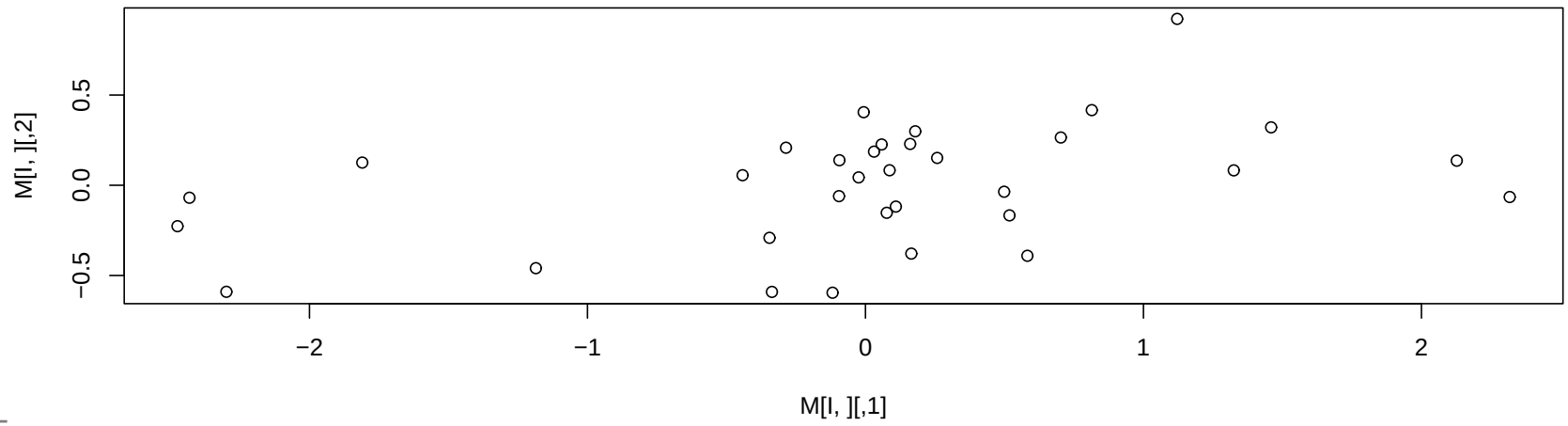
```
> summary(out)

Call:
 svm.default(x = x, y = y)

Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  radial
       cost:  1
      gamma:  0.25

Number of Support Vectors:  33
 ( 17 16 )
Number of Classes:  2

Levels:
 0 1
```

Support Vector Machines

```
## test with train data
> pred = predict(out, x)
> table(pred, y)
    y
pred 0  1
   0 49  2
   1  1 48
> M = cmdscale(dist(x))
> plot(M,col = as.integer(y)+1,pch = as.integer(y)+1)
## support vectors
> I = 1:n %in% out$index
 points(M[I,],lwd=2)
```

Support Vector Machines





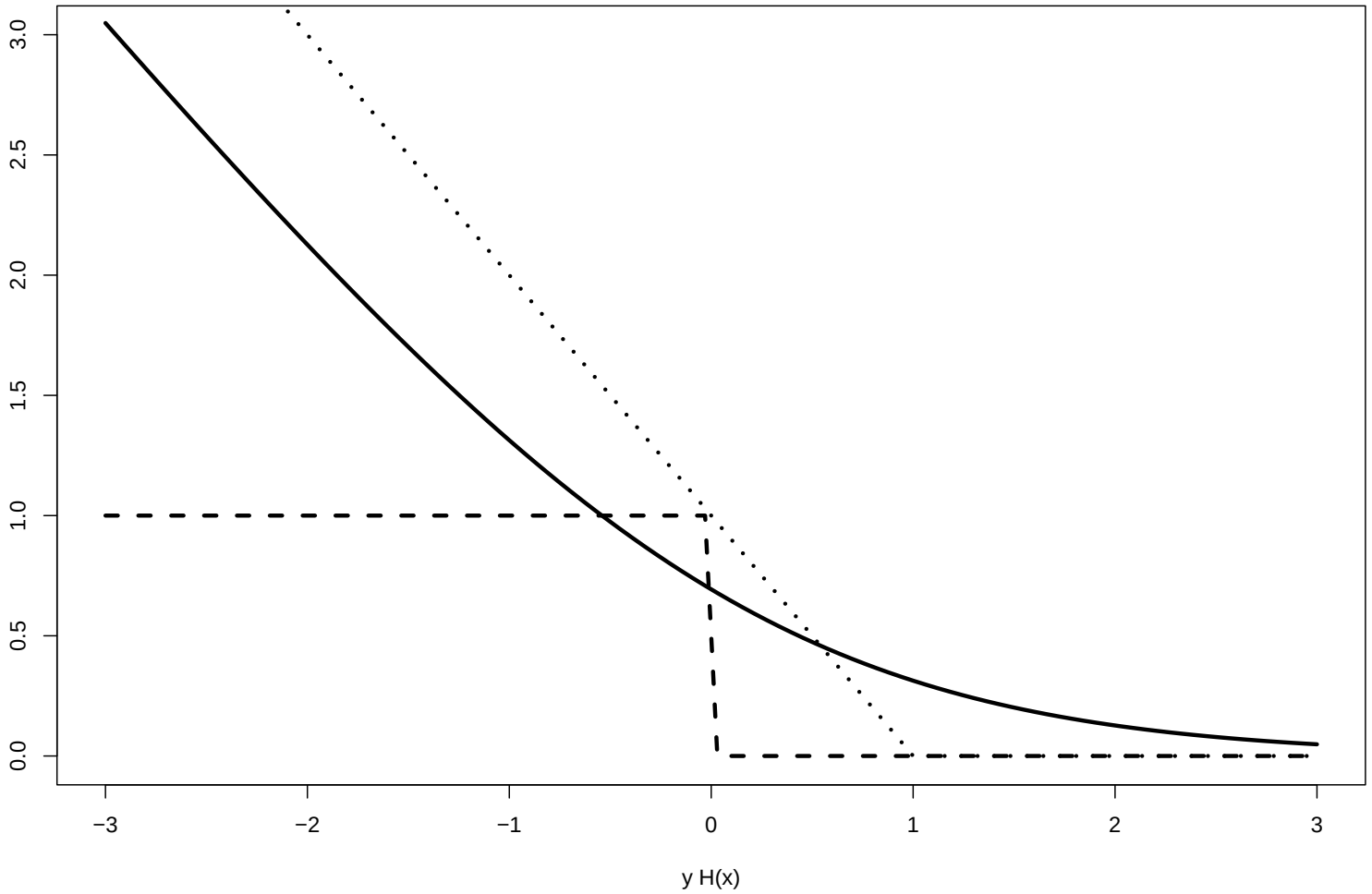
Support Vector Machines

Here is another (easier) way to think about the SVM. The SVM hyperplan $H(x) = \beta_0 + x^T x$ can be obtained by minimizing

$$\sum_{i=1}^{n}(1 - Y_i H(X_i))_+ + \lambda||\beta||^2.$$

The following figure compares the svm loss, squared loss, classification error and logistic loss $\log(1 + e^{-yH(x)})$.

Support Vector Machines



Kernelization

There is a trick called **kernelization** for improving a computationally simple classifier h. The idea is to map the covariate X — which takes values in $\mathcal{X}$ — into a higher dimensional space $\mathcal{Z}$ and apply the classifier in the bigger space $\mathcal{Z}$. This can yield a more flexible classifier while retaining computationally simplicity.

Kernelization

Example: The covariate $x = (x_1, x_2)$. The Y_is can be separated into two groups using an ellipse. Define a mapping ϕ by

$$z = (z_1, z_2, z_3) = \phi(x) = (x_1^2, \sqrt{2}x_1x_2, x_2^2).$$

Thus, ϕ maps $\mathcal{X} = \mathbb{R}^2$ into $\mathcal{Z} = \mathbb{R}^3$. In the higher-dimensional space $\mathcal{Z}$, the Y_i's are separable by a linear decision boundary.

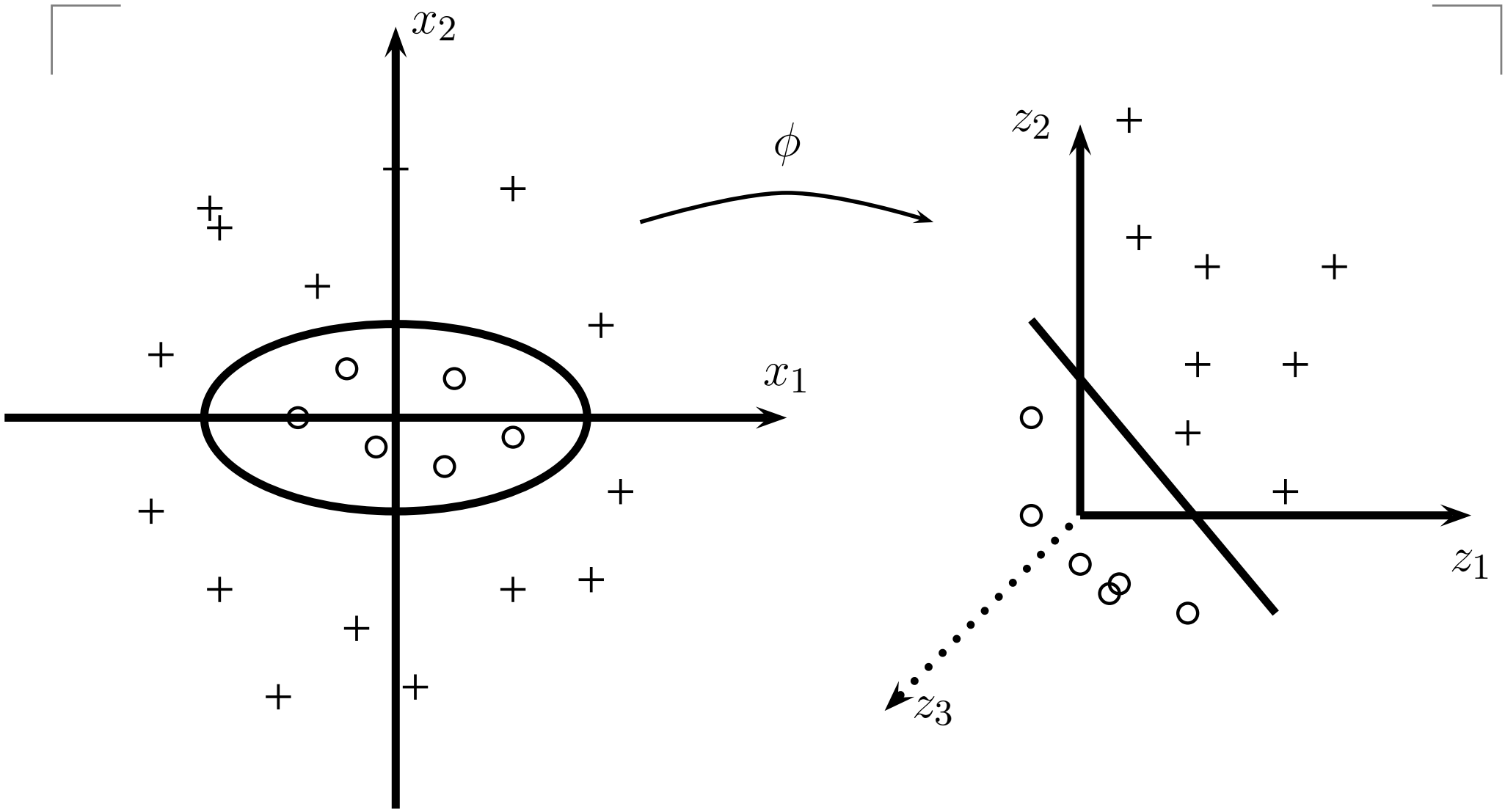
Kernelization

In other words,

> a linear classifier in a higher-dimensional space corresponds to a non-linear classifier in the original space.

The point is that to get a richer set of classifiers we do not need to give up the convenience of linear classifiers. We simply map the covariates to a higher-dimensional space. This is akin to making linear regression more flexible by using polynomials.

Kernelization



Kernelization

If we significantly expand the dimension of the problem, we might increase the computational burden. For example, if x has dimension $d = 256$ and we wanted to use all fourth-order terms, then $z = \phi(x)$ has dimension 183,181,376. We are spared this computational nightmare by the following two facts. First, many classifiers just use the inner product between pairs of points. Second, the inner product in $\mathcal{Z}$ can be written

$$\begin{aligned} \langle z, \widetilde{z} \rangle &= \langle \phi(x), \phi(\widetilde{x}) \rangle \\ &= x_1^2 \widetilde{x}_1^2 + 2 x_1 \widetilde{x}_1 x_2 \widetilde{x}_2 + x_2^2 \widetilde{x}_2^2 \\ &= (\langle x, \widetilde{x} \rangle)^2 \equiv K(x, \widetilde{x}). \end{aligned}$$

Thus, we can compute $\langle z, \widetilde{z} \rangle$ without ever computing $Z_i = \phi(X_i)$.

Kernelization

To summarize, kernelization involves finding a mapping $\phi : \mathcal{X} \to \mathcal{Z}$ and a classifier such that:

1. $\mathcal{Z}$ has higher dimension than $\mathcal{X}$ and so leads a richer set of classifiers.
2. The classifier only requires computing inner products.
3. There is a function K, called a kernel, such that $\langle \phi(x), \phi(\widetilde{x}) \rangle = K(x, \widetilde{x})$.
4. Everywhere the term $\langle x, \widetilde{x} \rangle$ appears in the algorithm, replace it with $K(x, \widetilde{x})$.

Kernelization

In fact, we never need to construct the mapping ϕ at all. We only need to specify a kernel $K(x, \widetilde{x})$ that corresponds to $\langle \phi(x), \phi(\widetilde{x}) \rangle$ for some ϕ. This raises an interesting question: given a function of two variables $K(x, y)$, does there exist a function $\phi(x)$ such that $K(x, y) = \langle \phi(x), \phi(y) \rangle$? The answer is provided by **Mercer's theorem** which says, roughly, that if K is positive definite — meaning that

$$\int \int K(x, y) f(x) f(y) dx dy \geq 0$$

for square integrable functions f — then such a ϕ exists.

Kernelization

Examples of commonly used kernels are:

$$\begin{aligned}
\text{polynomial} \quad & K(x,\widetilde{x}) &=& \Big(\langle x,\widetilde{x}\rangle + a\Big)^r \\
\text{sigmoid} \quad & K(x,\widetilde{x}) &=& \tanh(a\langle x,\widetilde{x}\rangle + b) \\
\text{Gaussian} \quad & K(x,\widetilde{x}) &=& \exp\Big(-||x-\widetilde{x}||^2/(2\sigma^2)\Big)
\end{aligned}$$

Kernelization

The support vector machine can be kernelized as follows. We simply replace $\langle X_i, X_j \rangle$ with $K(X_i, X_j)$. We now maximize

$$\sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{k=1}^{n} \alpha_i \alpha_k Y_i Y_k K(X_i, X_j). \tag{7}$$

The hyperplane can be written as $\widehat{H}(x) = \widehat{a}_0 + \sum_{i=1}^{n} \widehat{\alpha}_i Y_i K(X, X_i)$.

Other Classifiers

1. Bagging
2. Boosting
3. Neural Networks

A Few Words on Nonparametric Bayes

Nonparametric Bayes is becoming very popular. It is appealing because of (i) conceptual simplicity and (ii) can be implmented by simulation.
Advantages:

1. Easy to understand.
2. Can incorporate prior information.

Disadvantages:

1. Requires specifying an infinite dimensional prior.
2. Not falsifiable (no long run guarantees). In fact, they typically have near 0 frequency coverage.

Nonparametric Bayes

$$Y_i = f(X_i) + \epsilon_i$$

Assume f lives in some functions space, for example:

$$f \in \mathcal{F} = \left\{ f : \int (f''(x))^2 dx < \infty \right\}.$$

Now put a prior π on f. Then get the posterior by Bayes' theorem:

$$\underbrace{\pi(f \in A)}_{\text{prior}} \stackrel{\text{data } D}{\longrightarrow} \underbrace{\pi(f \in A|D)}_{\text{posterior}}$$

Can now find Bayesian confidence bands (L, U):

$$\mathbb{P}(L(x) \leq f(x) \leq U(x)|D) = 1 - \alpha.$$

But ...

How often will (L, U) tarp f in the frequency sense? In other words, what is:

$$\mathbb{P}_f(L(x) \leq f(x) \leq U(x))??$$

Answer: typically,

$$\mathbb{P}_f(L(x) \leq f(x) \leq U(x)) \approx 0.$$

Why? The prior adds bias which causes the bands to be centered away from the true f. That's why choosing smoothing parameters is so hard!

Some References

- Diaconis and Freedman (1986)
- Shen and Wasserman (2001)
- Ghosal, Ghosh and van der Vaart (2000)
- Zhao (2000)
- Freedman (1963)
- Cox (1993)