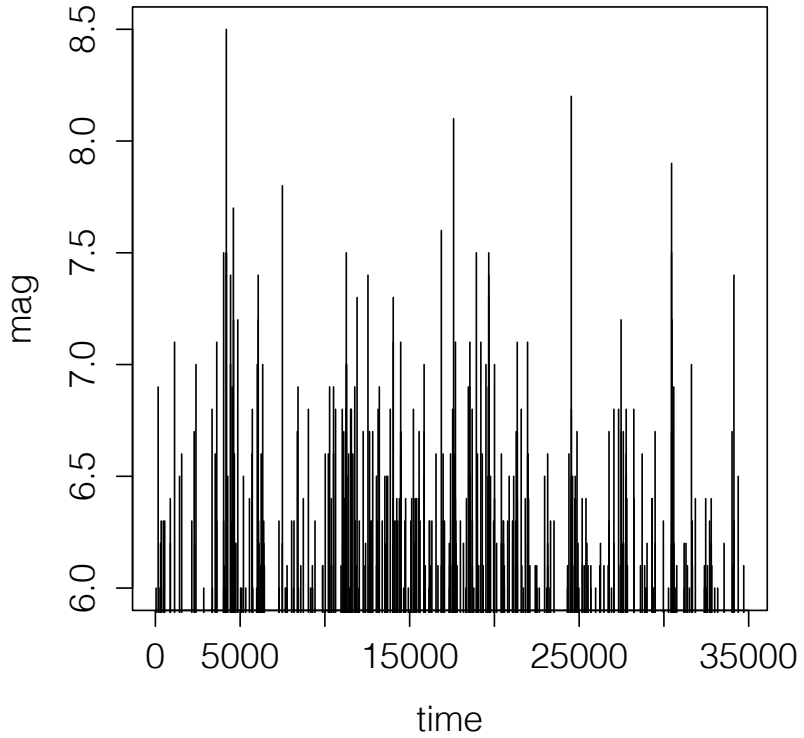
Today

- HW 3 due April 1
- Project due April 15
- nonparametric and semi-parametric regression
- In the News:
  Statschat
  New Yorker
- recap: kernel regression – summarize; plus where it is implemented
- honey

... example 10.31

SM




```
library(SMPracticals); data(quake)
with(quake, plot(time, mag, type="h")
```

Recap: nonparametric Regression

ELM, Ch. 11; SM, §10.7

- model $y_i = f(x_i) + \epsilon_i, \quad i = 1, \ldots, n \quad x_i$ scalar
- mean function $f(\cdot)$ assumed to be "smooth"
- introduce a kernel function $K(u)$ and define a set of weights

$$w_i = \frac{1}{\lambda} K\left(\frac{x_i - x_0}{\lambda}\right)$$

- estimate of $f(x)$, at $x = x_0$:

$$\hat{f}_\lambda(x_0) = \frac{\sum_{i=1}^n w_i y_i}{\sum_{i=1}^n w_i}$$

- Nadaraya-Watson estimator – local averaging

Local Polynomials

- better estimates can be obtained using local regression at point x_0 instead of local averaging

- $$\begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix} = \begin{pmatrix} 1 & (x_1 - x_0) & \cdots & (x_1 - x_0)^k \\ \vdots & \vdots & & \vdots \\ 1 & (x_n - x_0) & \cdots & (x_n - x_0)^k \end{pmatrix} \begin{pmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_k \end{pmatrix} + \begin{pmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \vdots \\ \varepsilon_n \end{pmatrix},$$

- $$\hat{\beta} = (X^T W X)^{-1} X^T W y, \quad w_i = \frac{1}{\lambda} K\left(\frac{x_i - x_0}{\lambda}\right)$$

- $$\hat{f}_\lambda(x_0) = \hat{\beta}_0$$

... local polynomials

- odd-order polynomials work better than even; usually local linear fits are used
- kernel function is often a Gaussian density, or the tricube function

$$K(u) = (1 - |u|^3)^3, \quad |u| < 1$$

- as with Nadaraya-Watson estimators, choice of bandwidth, λ controls smoothness of function
- `loess` computes local linear regression, and includes a robust option

```
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(...), ...)
```

- `"symmetric"` implements robust version

... cross-validation

- $$CV(\lambda) = \sum_{i=1}^{n} \{y_i - \hat{f}_{-i}(x_i)\}^2$$
- for local polynomials

$$CV(\lambda) = \sum_{i=1}^{n} \left\{ \frac{y_i - \hat{f}_\lambda(x_i)}{1 - S_{ii}(\lambda)} \right\}^2$$

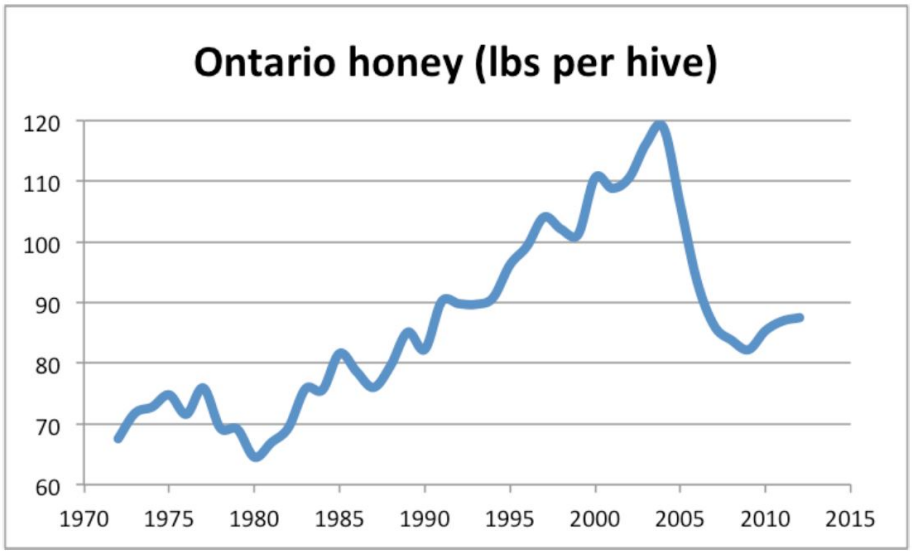
- simpler

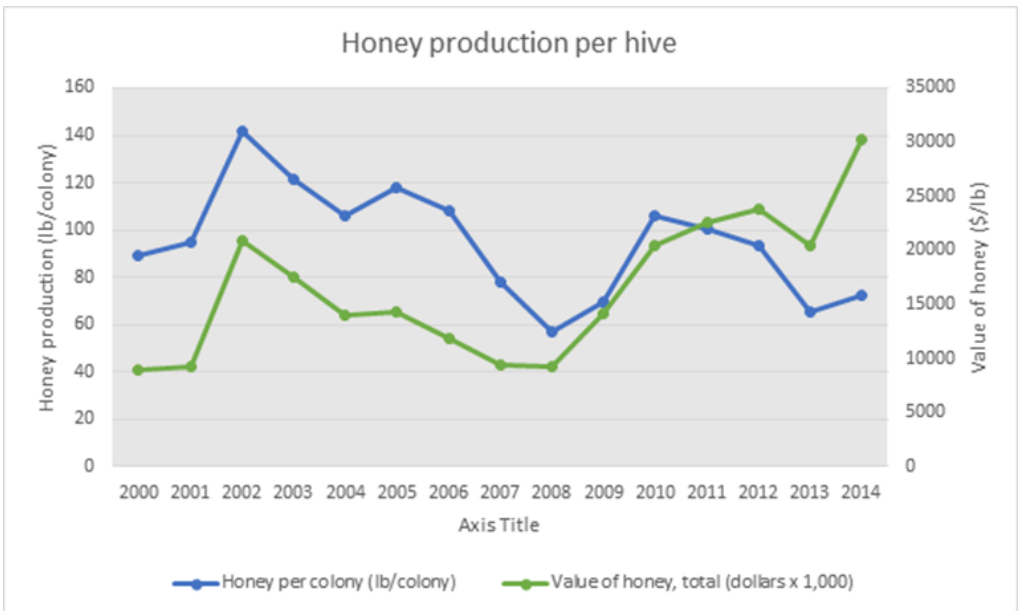
$$GCV(\lambda) = \sum_{i=1}^{n} \left\{ \frac{y_i - \hat{f}_\lambda(x_i)}{1 - \mathrm{tr}(S_\lambda)/n} \right\}^2$$

- library `sm` does local linear regression, computes optimal bandwidth via `hcv`

Example: honey production

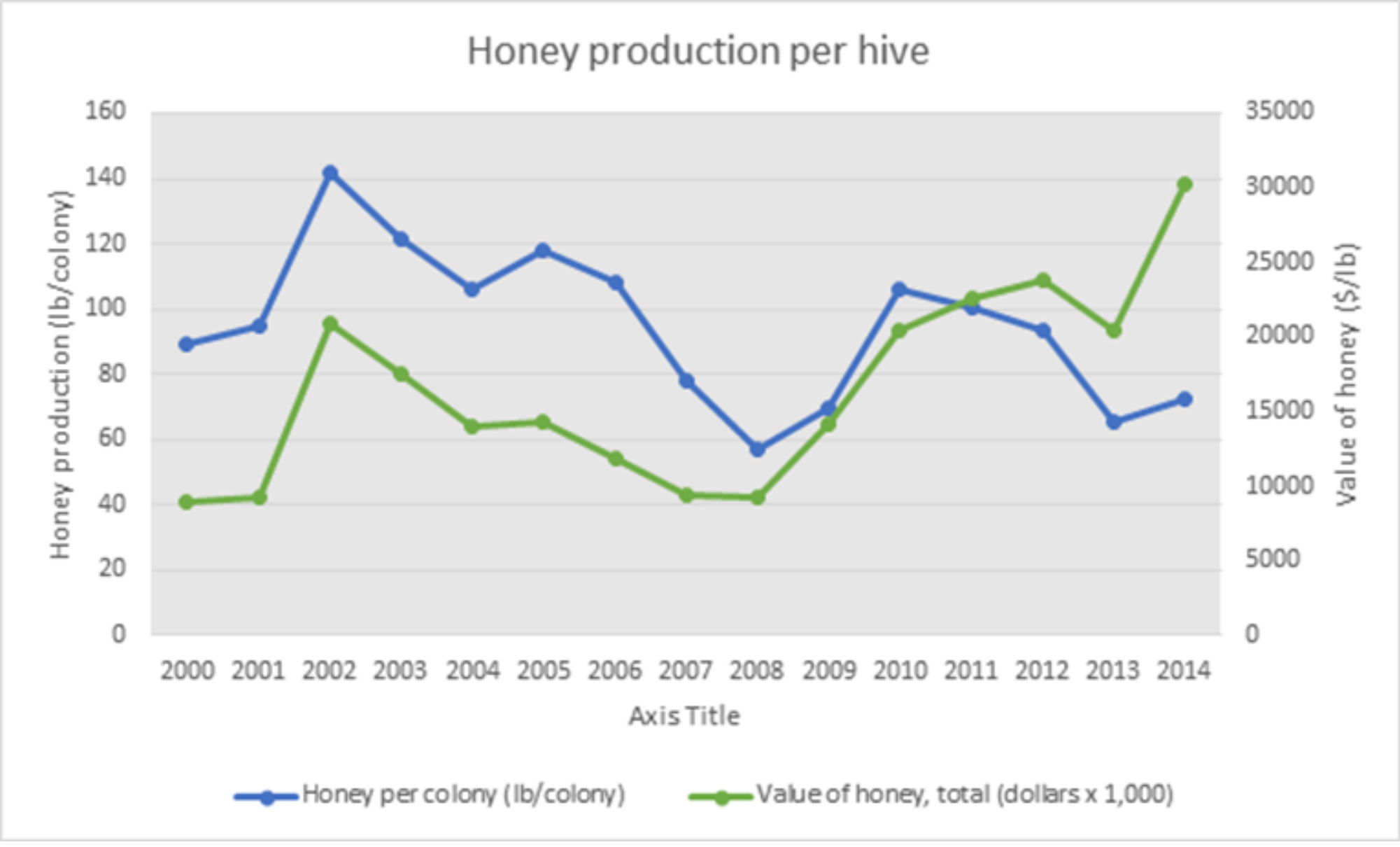
Stat Can



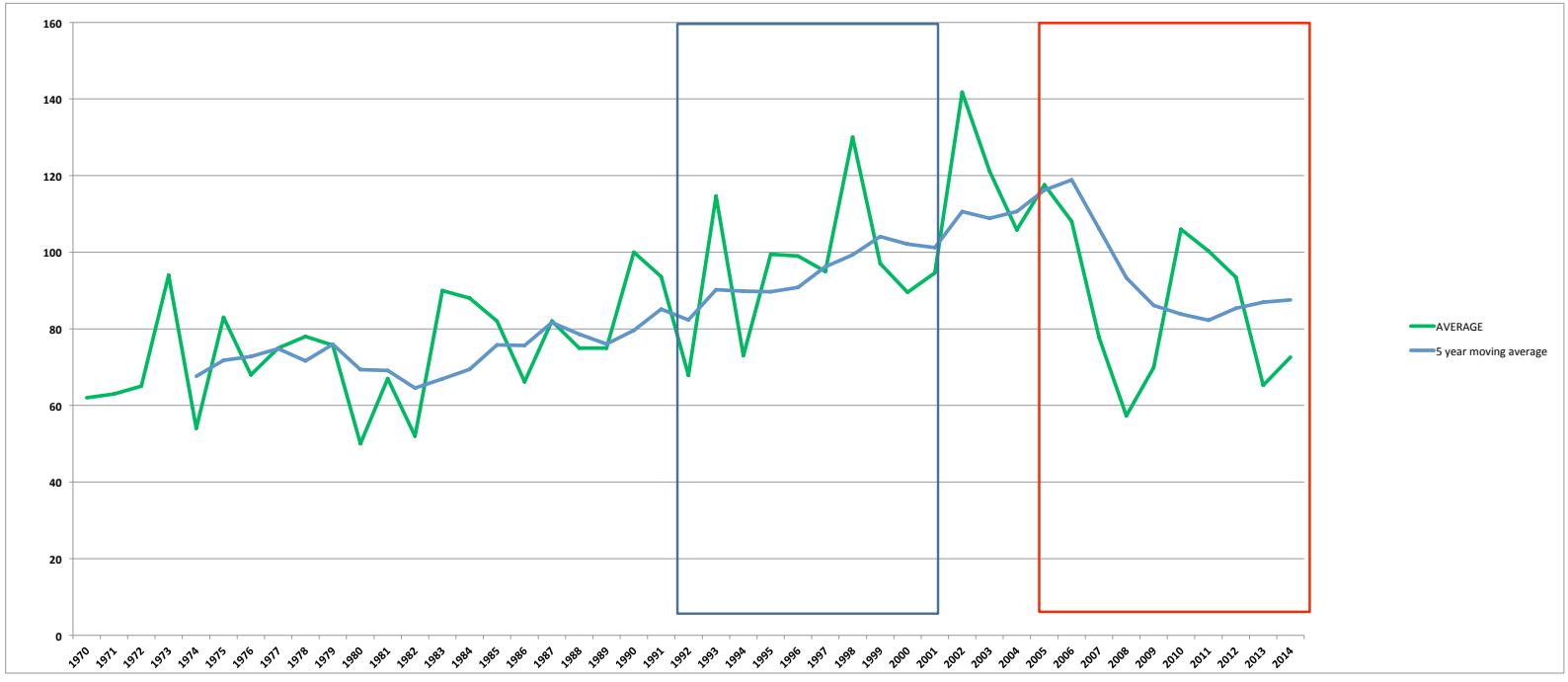


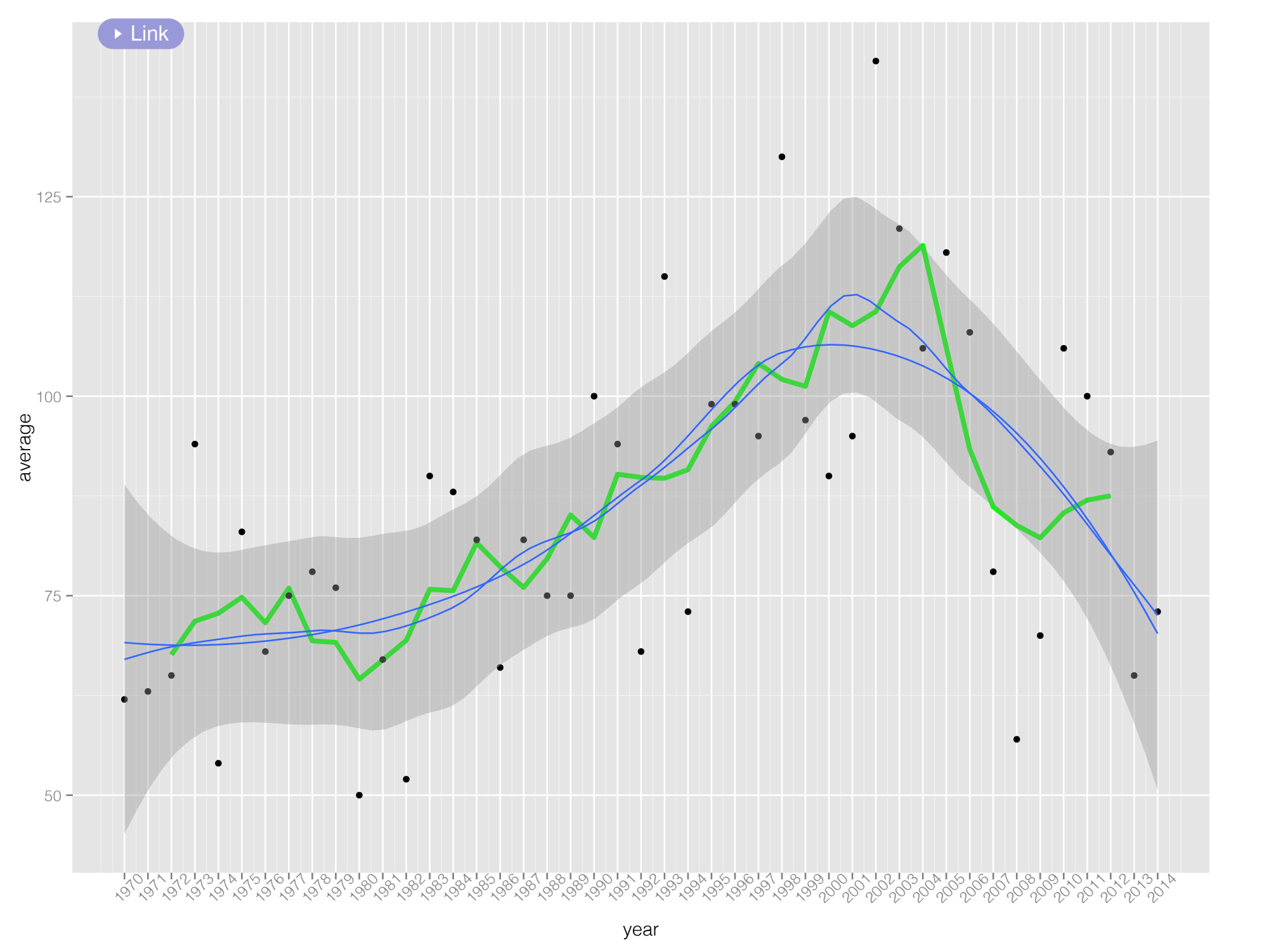
5-year running mean

annual averages


Honey production per hive
Honey production (lb/colony)
Value of honey ($/lb)
160
140
120
100
80
60
40
20
0
35000
30000
25000
20000
15000
10000
5000
0
2000 2001 2002 2003 2004 2005 2006 2007 2008 2009 2010 2011 2012 2013 2014
Axis Title
Honey per colony (lb/colony)
Value of honey, total (dollars x 1,000)

YEAR	1970	1971	1972	1973	1974	1975	1976	1977	1978	1979	1980	1981	1982	1983	1984	1985	1986	1987	1988	1989	1990	1991	1992
AVERAGE	62	63	65	94	54	83	68	75	78	76	50	67	52	90	88	82	66	82	75	75	100	94	68
5 year moving average					68	72	73	75	72	76	69	69	65	67	69	76	76	82	79	76	80	85	82




Link
125
100
75
50
average
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
year

Inference after fitting local polynomials

SM §10.7

- $\hat{\beta} = (X^T W X)^{-1} X^T W y$
- $W = \text{diag}(w_1, \ldots, w_n)$
- $\hat{f}_\lambda(x_0) = \hat{\beta}_0 = \sum_{i=1}^n S(x_0; x_i, \lambda) y_i$; $\hat{f} = (\hat{f}_\lambda(x_1), \ldots, \hat{f}_\lambda(x_n)) = \mathcal{S}_\lambda y$
- $S(x_0; x_1, \lambda), \ldots, S(x_0; x_n, \lambda)$ first row of "hat" matrix $(X^T W X)^{-1} X^T W$
- $\text{E}\{\hat{f}_\lambda(x_0)\} = \sum_{i=1}^n S(x_0; x_i, \lambda) f_\lambda(x_i)$
- $\text{var}\{\hat{f}_\lambda(x_0)\} = \sigma^2 \sum_{i=1}^n S^2(x_0; x_i, \lambda)$
- $\tilde{\sigma}^2 = \frac{1}{n - 2\nu_1 + \nu_2} \sum_{i=1}^n \{y_i - \hat{f}_\lambda(x_i)\}^2$
- $\nu_1 = \text{tr}(S_\lambda)$, $\nu_2 = \text{tr}(S_\lambda^T S_\lambda)$

potential estimates of 'degrees of freedom'

Flexible modelling using basis expansions

ELM §11.2; SM §10.7.2

- $y_i = f(x_i) + \epsilon_i$
- Flexible linear modelling

$$f(x) = \Sigma_{m=1}^{M} \beta_m h_m(x)$$

- This is called a linear basis expansion, and h_m is the mth basis function
- For example: $f(x) = \beta_0 + \beta_1 x + \beta_2 x^2$, or $f(x) = \beta_0 + \beta_1 \sin(x) + \beta_2 \cos(x)$, etc.
- Simple linear regression has $h_1(x) = 1$, $h_2(x) = x$

Piecewise polynomials

- piecewise constant basis functions
  $h_1(x) = I(x < \xi_1), \quad h_2(x) = I(\xi_1 \leq x < \xi_2),$
  $h_3(x) = I(\xi_2 \leq x)$
- equivalent to fitting by local averaging

- piecewise linear basis functions , with constraints
  $h_1(x) = 1, \quad h_2(x) = x$
  $h_3(x) = (x - \xi_1)_+, \quad h_4(x) = (x - \xi_2)_+$
- windows defined by knots $\xi_1, \xi_2, \ldots$

- piecewise cubic basis functions
  $h_1(x) = 1, h_2(x) = x, h_3(x) = x^2, h_4(x) = x^3$

- continuity $h_5(x) = (x - \xi_1)^3_+, \quad h_6(x) = (x - \xi_2)^3_+$

- continuous function, continuous first and second derivs

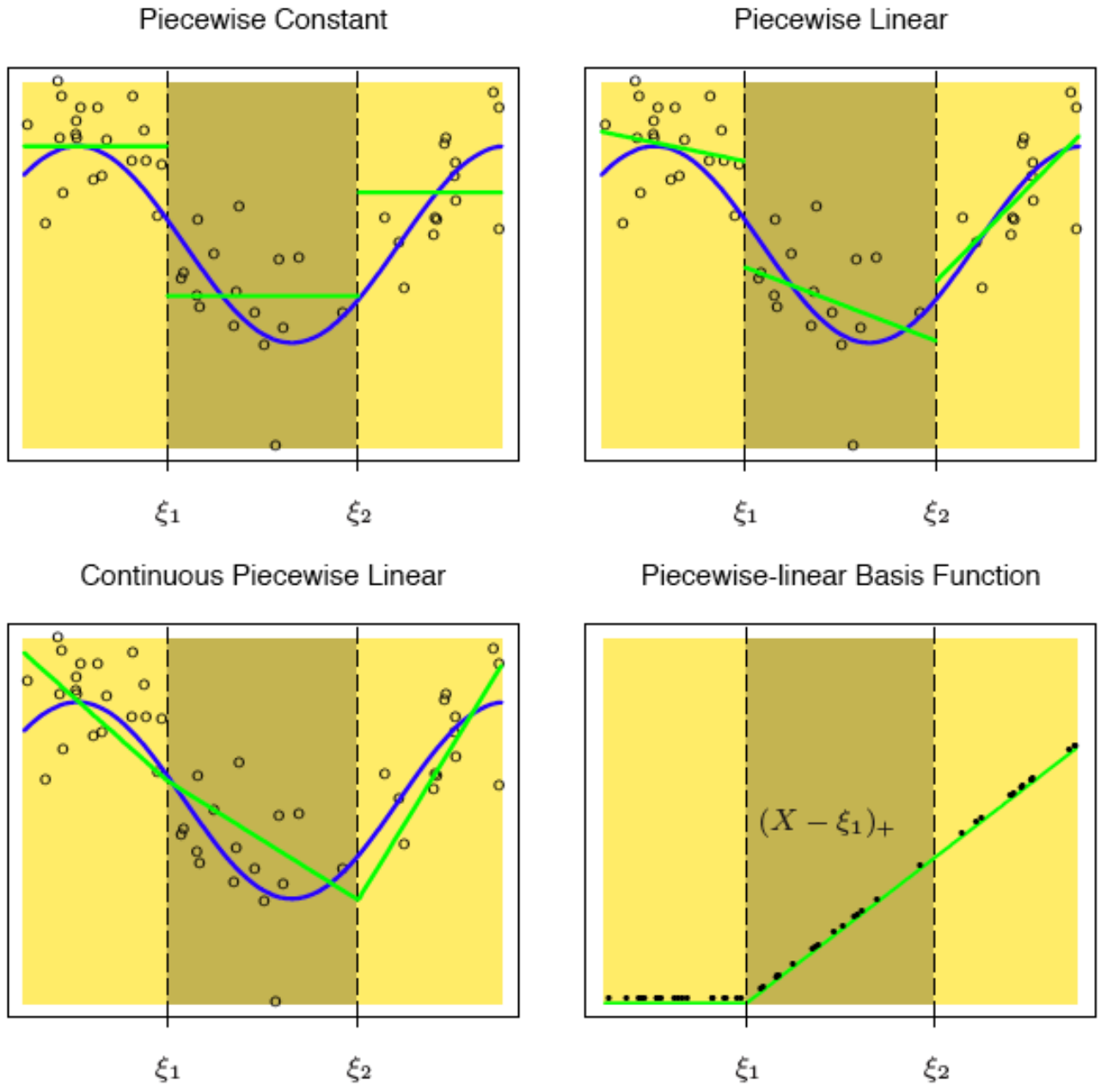


FIGURE 5.1. *The top left panel shows a piecewise constant function fit to some artificial data. The broken vertical lines indicate the positions of the two* knots *ξ_1 and ξ_2. The blue curve represents the true function, from which the data were*

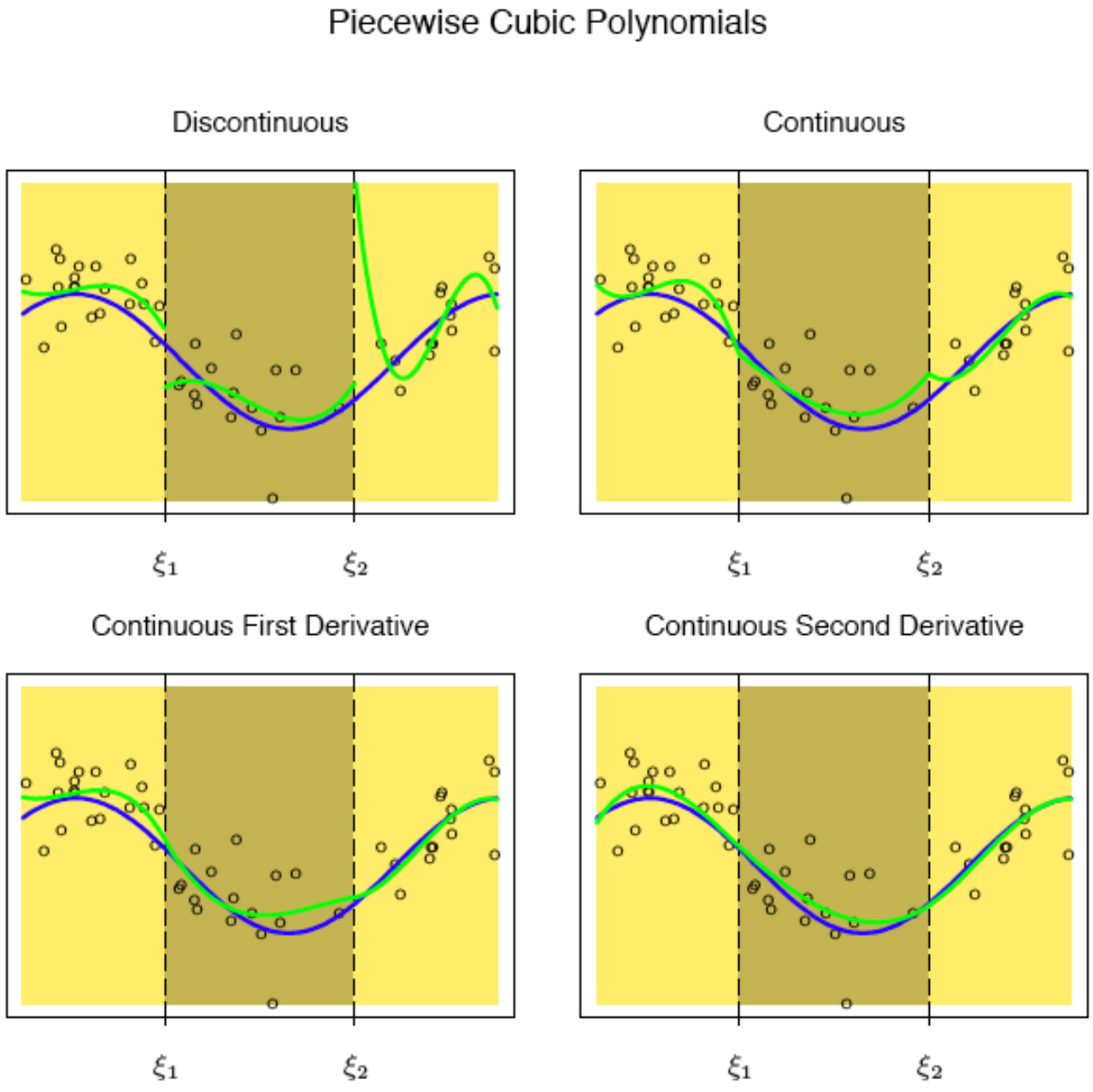
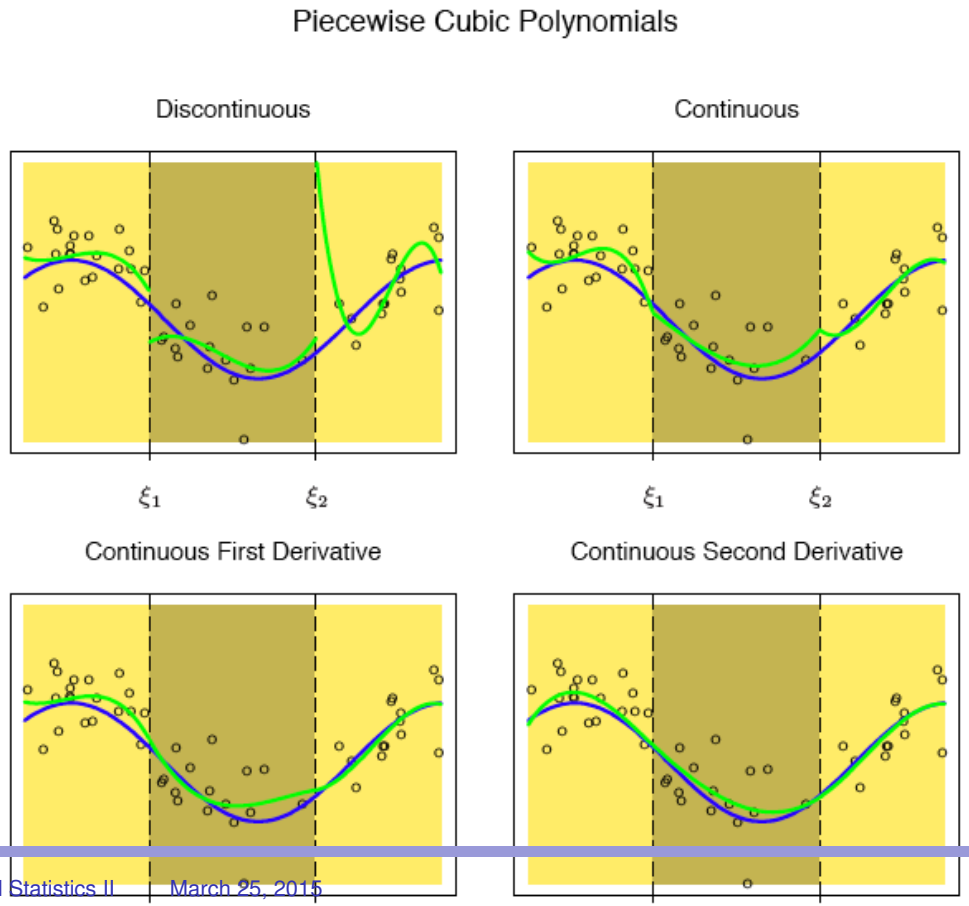


FIGURE 5.2. *A series of piecewise-cubic polynomials, with increasing orders of continuity.*

Cubic splines

- truncated power basis of degree 3
- need to choose number of knots K and placement of knots $\xi_1, \ldots \xi_K$
- construct features matrix using truncated power basis set
- use constructed matrix as set of predictors



... cubic splines

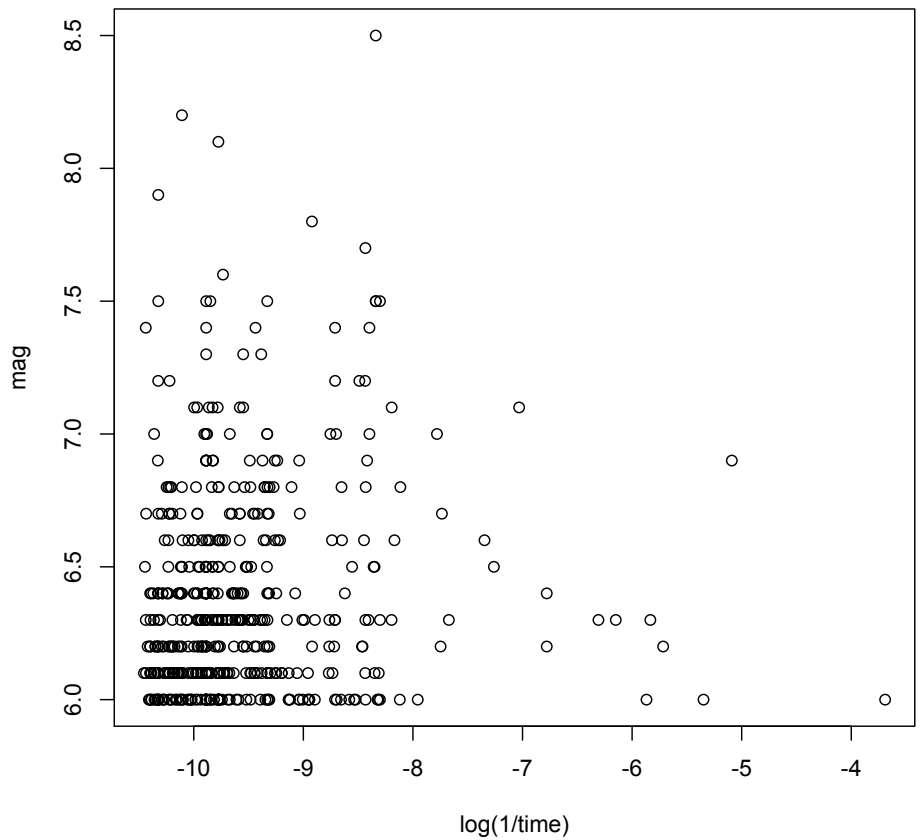
```
> with(quake, bs(log(1/time))[1:10,])
#bs(x) with no other arguments just gives a
# single cubic polynomial
                1         2         3
 [1,] 0.0000000 0.0000000 1.0000000
 [2,] 0.1018013 0.3903714 0.4989780
 [3,] 0.1359705 0.4189773 0.4303434
 [4,] 0.1884790 0.4408886 0.3437743
 [5,] 0.2056632 0.4436068 0.3189471
...
attr(,"degree")
[1] 3
attr(,"knots")
numeric(0)
attr(,"Boundary.knots")
[1] -10.454784  -3.690961
attr(,"intercept")
[1] FALSE
attr(,"class")
[1] "bs"     "basis"  "matrix"
```

... cubic splines

```
 > with(quake,bs(log(1/time), df=5)[1:10,])
# gives a proper cubic spline basis, here with 5 df
      1          2          3          4          5
 [1,] 0 0.00000000 0.0000000 0.0000000 1.0000000
 [2,] 0 0.01110655 0.1250814 0.4247847 0.4390274
 [3,] 0 0.01846075 0.1661869 0.4486889 0.3666635
 [4,] 0 0.03370916 0.2283997 0.4600092 0.2778819
 [5,] 0 0.03989014 0.2484715 0.4585984 0.2530400
...
attr(,"degree")
[1] 3
attr(,"knots")
33.33333% 66.66667%
-9.943294 -9.520987
attr(,"Boundary.knots")
[1] -10.454784  -3.690961
```

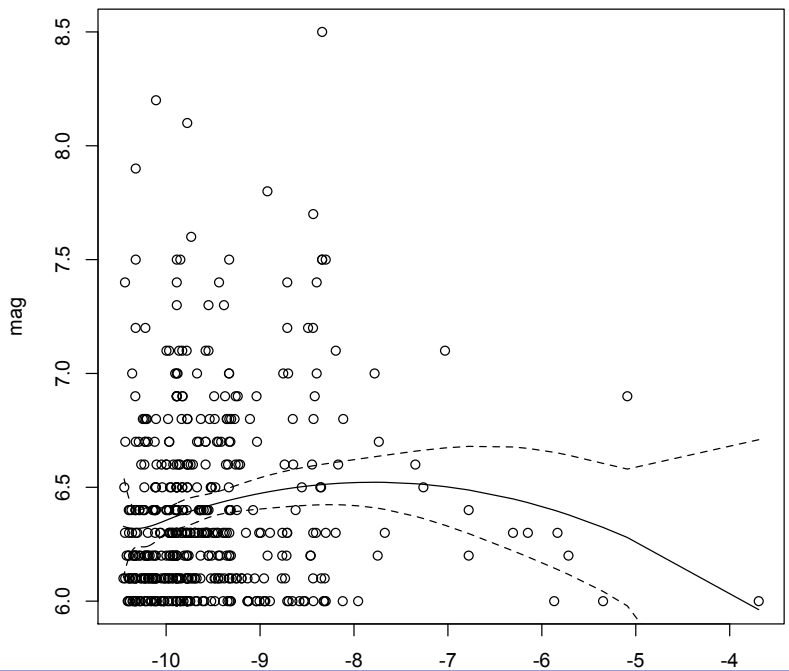
Example: earthquake data

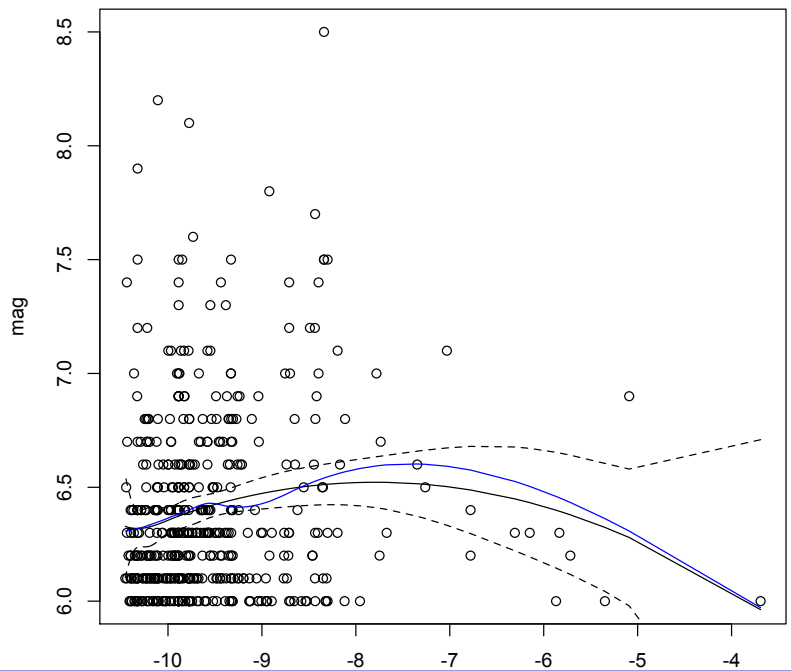
```
> data(quake,package="SMPracticals")
> head(quake)
           time mag
1      40.08333 6.0
2     162.38889 6.9
3     210.22917 6.0
> with(quake, plot(log(1/time),mag))## using a different measure of
###  intensity here than in Figure 10.36
```



... earthquake data

```
> quake.bs = lm(mag ~ bs(log(1/time),df=5),data = quake)
> quake.pred = predict(quake.bs, se.fit = TRUE, interval = "confidence")
> quake.pred
$fit
         fit      lwr      upr
1   5.962665 5.216283 6.709047
2   6.279641 5.979190 6.580092
3   6.323859 6.042772 6.604946
> lines(log(1/quake$time),quake.pred[[1]])
> lines(log(1/quake$time),quake.pred[[1]]+quake.pred$se.fit, lty=2)
> lines(log(1/quake$time),quake.pred[[1]]-quake.pred$se.fit, lty=2)
> quake.lo = loess(mag ~ log(1/time), data = quake)
> lines(quake.lo$x,quake.lo$fitted,col="blue")
```





Regression splines

- regression splines are 'hand-crafted': `bs(x, df = ?)`
- once crafted, model is parametric
- $f(x) = \sum_{m=1}^{M} \beta_m h_m(x)$ `df` = M
- degrees of freedom controls how many knots are placed
- equivalent to how many parameters are fit
- ELM, §11.2 constructs linear splines 'by hand' Figure 11.7
- a different approach is smoothing splines – put a LOT of knots, and regularize

Smoothing splines

ELM §11.2, SM §10.7.2

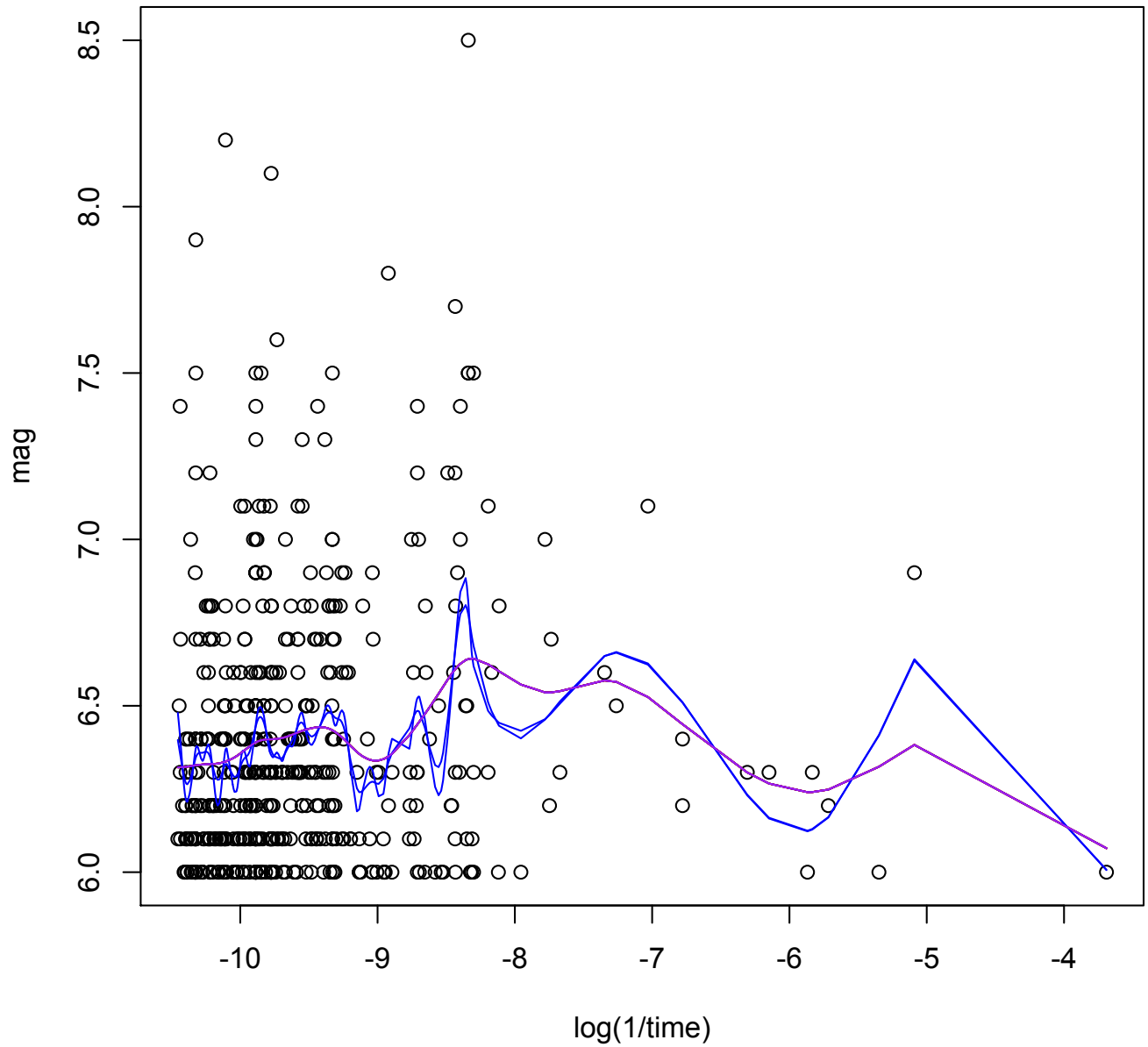
- $y_i = f(x_i) + \epsilon_i, \quad i = 1, \ldots, n$
- choose $f(\cdot)$ to solve

$$\min_f \sum_{i=1}^{n} \frac{\{y - f(x_i)\}^2}{2\sigma^2} - \frac{\lambda}{2\sigma^2} \int_a^b \{f''(t)\}^2 dt, \quad , \lambda > 0$$

- solution is a cubic spline, with knots at each observed x_i value

see SM Figure 10.18 for a non-regularized solution

- has an explicit, finite dimensional solution
- $\hat{f} = \{\hat{f}(x_1), \ldots, \hat{f}(x_n)\} = (I + \lambda K)^{-1} y$
- K is a symmetric $n \times n$ matrix of rank $n - 2$

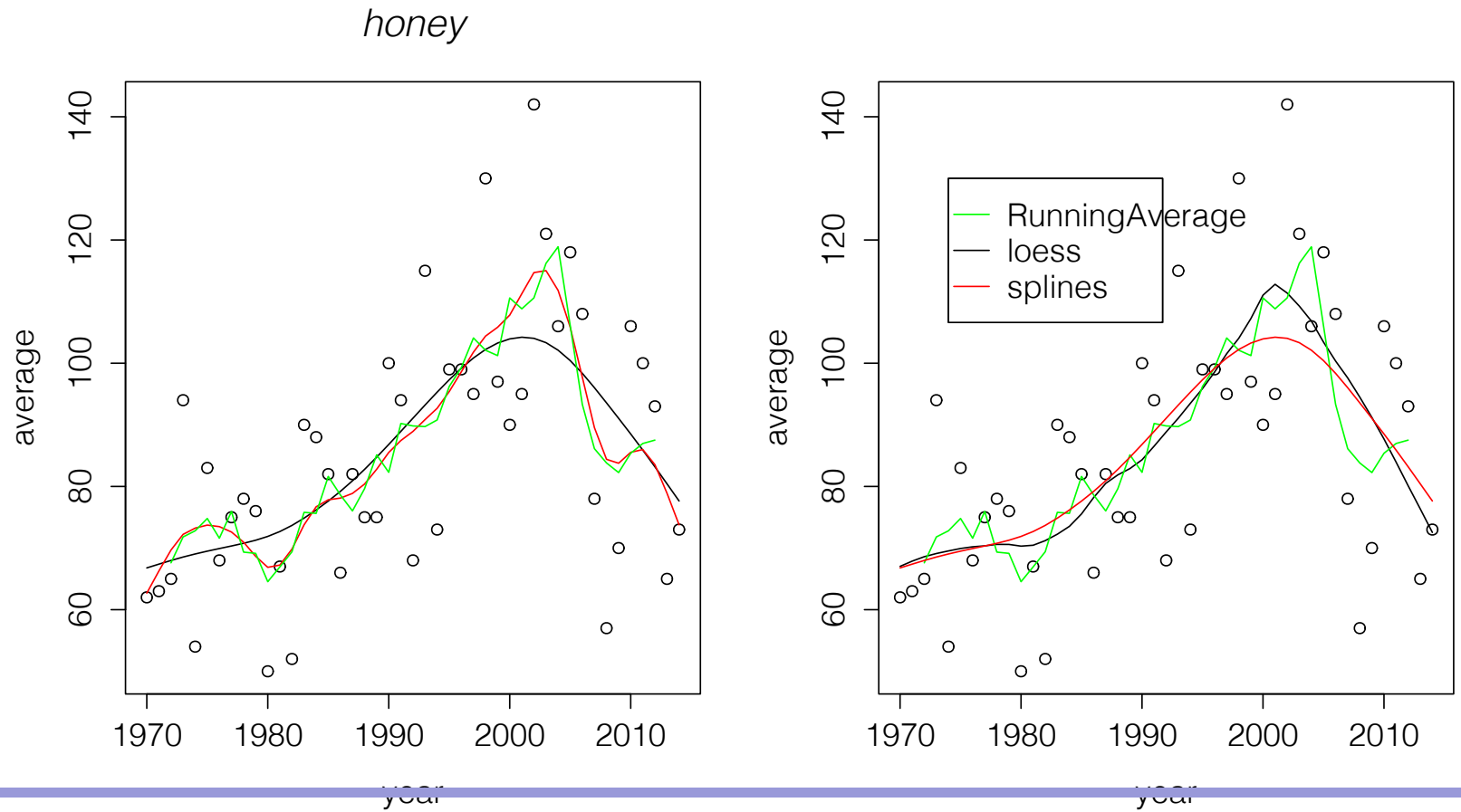

mag
8.5
8.0
7.5
7.0
6.5
6.0
-10
-9
-8
-7
-6
-5
-4
log(1/time)

... smoothing splines

```
> quake$int = log(1/quake$time)
> quake[1:4,]
       time mag       int
1  40.08333 6.0 -3.690961
2 162.38889 6.9 -5.089994
3 210.22917 6.0 -5.348198
4 303.85417 6.2 -5.716548

> attach(quake)
> plot(int,mag)
> quake.ss2 = smooth.spline(x = int, y = mag, df = 5)
> lines(quake.ss2, col="red")
> quake.ss3
Call:
smooth.spline(x = int, y = mag, cv = TRUE)

Smoothing Parameter  spar= 1.499945  lambda= 0.0001340604 (25 iterations)
Equivalent Degrees of Freedom (Df): 11.35023
Penalized Criterion: 64.57512
PRESS: 0.1730025
> lines(quake.ss3, col="blue")
```

... smoothing splines

```
> attach(honey)

> plot(year,average)
> plot(year,average, main = "honey")
> lines(smooth.spline(honey$year,honey$average))
> lines(smooth.spline(honey$year,honey$average, df = 11),col="red")
> lines(honey$year,honey$runmean,col="green")
```



Multidimensional splines

- so far we are considering just 1 X at a time
- for regression splines we replace each X by the new columns of the basis matrix
- for smoothing splines we get a univariate regression
- it is possible to construct smoothing splines for two or more inputs simultaneously, but computational difficulty increases rapidly
- these are called thin plate splines
- alternative:
  $E(Y \mid X_1, \ldots, X_p) = f_1(X_1) + f_2(X_2) + \cdots + f_p(X_p)$

  additive models

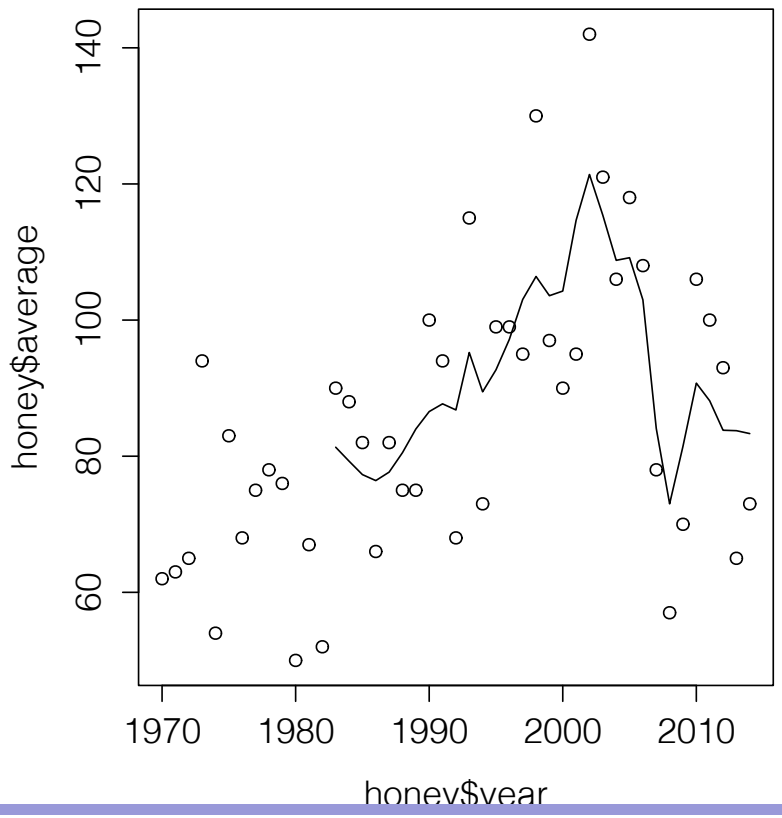
ELM Ch. 12

Wavelets

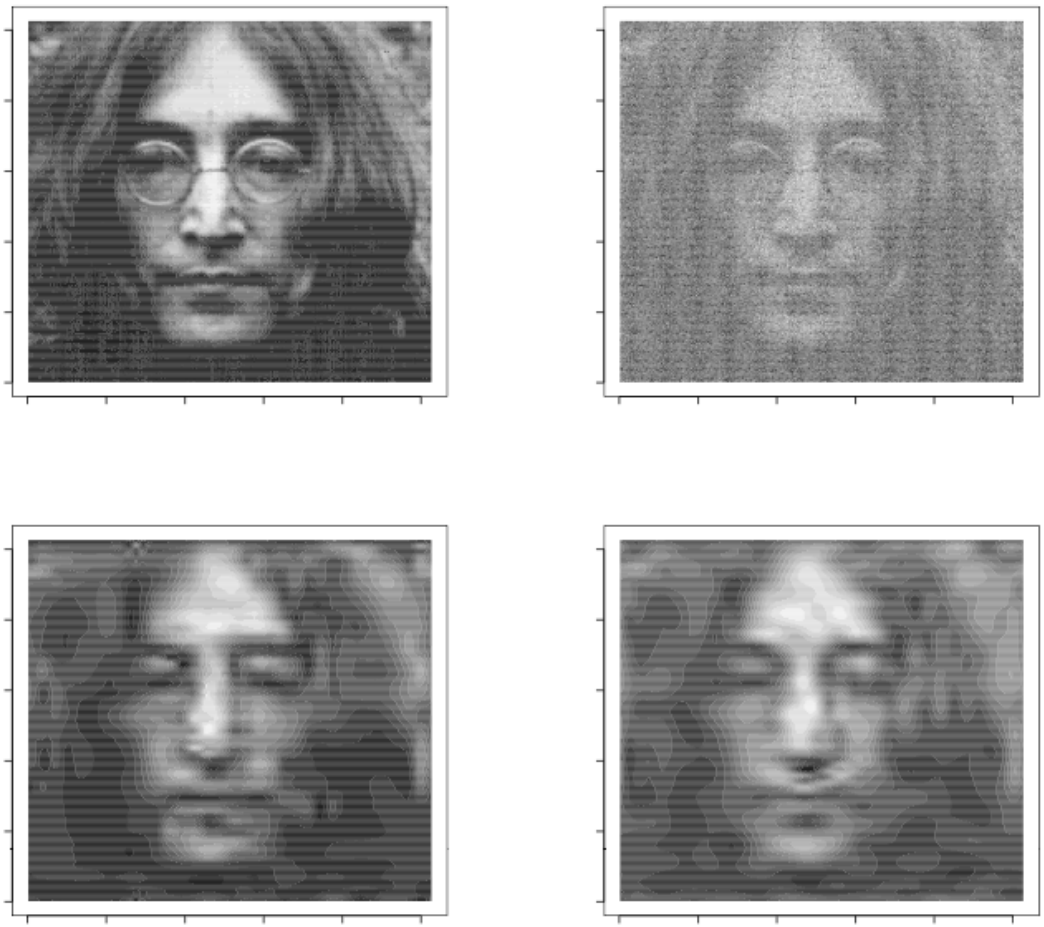
ELM §11.4

- basis functions with very 'local' support
- useful for capturing regions of high variation
- useful for data compression – transmit only local components

Fig 11.11



... wavelets



Vidaković and Mueller, ”Wavelets for kids (Part I)” 1994.

Which smoothing method?

ELM §11.6

- basis functions: natural splines, Fourier, wavelet bases
- regularization via cubic smoothing splines
- kernel smoothers: locally constant/linear/polynomial
- adaptive bandwidth, running medians, running *M*-estimates
- with very little noise, a small amount of local smoothing (e.g. nearest neighbours)
- with moderate amounts of noise, kernel and spline methods are effective
- with large amounts of noise, parametric methods are more attractive
- "It is not reasonable to claim that any one smoother is better than the rest"
  - `loess` is robust to outliers, and provides smooth fits
  - spline smoothers are more efficient, but potentially sensitive to outliers
  - kernel smoothers are very sensitive to bandwidth

Shrinkage Methods

Elements of Statistical Learning, Hastie et al. §3.4

- Ridge regression
-

$$\begin{aligned} \hat{\beta}_{LS} &= (X^T X)^{-1} X^T y \\ \hat{\beta}_{ridge} &= (X^T X + \lambda I)^{-1} X^T y \end{aligned}$$

- can show that $\hat{\beta}_{ridge}$ satisfies

$$\min_{\beta} \left(\Sigma \{ y_i - \beta_0 - \Sigma_{j=1}^{p} x_{ij} \beta_j \}^2 + \lambda \Sigma_{j=1}^{p} \beta_j^2 \right)$$

$$\min_{\beta} \Sigma \{ y_i - \beta_0 - \Sigma_{j=1}^{p} x_{ij} \beta_j \}^2 \quad \text{s.t. } \Sigma \beta_j^2 \leq t$$

- Assume x_j's are centered and put these in matrix X (with no column of 1's:

$$\min_{\beta} (y - X\beta)^T (y - X\beta) \qquad \text{s.t. } ||\beta||^2 \leq t$$

... ridge regression

- $$\min_{\beta}\{(y - X\beta)^T(y - X\beta) + \lambda||\beta||^2\}$$
- λ is a tuning parameter: $\lambda = 0$ gives $\hat{\beta}_{LS}$, $\lambda \to \infty$
- in R the `library MASS` `library(MASS)` has a ridge regression version of `lm` called `lm.ridge`
- if columns of X are nearly linearly dependent (multicollinearity), $\hat{\beta}$'s for these columns should be shrunk towards 0.
- essential that the predictors are all scaled to the same units
- this is difficult for interpretation of the coefficients
- $$df(\lambda) = \text{tr}[X(X^TX + \lambda I)^{-1}X^T] = \Sigma_{j=1}^{p} \frac{d_j^2}{d_j^2 + \lambda}$$

effective number of parameters

Lasso

- $$\min_{\beta} \left(\Sigma \{ y_i - \beta_0 - \Sigma_{j=1}^{p} x_{ij} \beta_j \}^2 + \lambda \Sigma_{j=1}^{p} |\beta_j| \right)$$
- $$\min_{\beta} \Sigma \{ y_i - \beta_0 - \Sigma_{j=1}^{p} x_{ij} \beta_j \}^2 \quad \text{s.t. } \Sigma |\beta_j| \leq t$$
- quadratic programming problem
- $\hat{\beta}^{lasso}$ is nonlinear function of y
- Tibshirani (1996), JRSS B and (2011), JRSS B
- `http://http://www-stat.stanford.edu/~tibs/lasso.html`

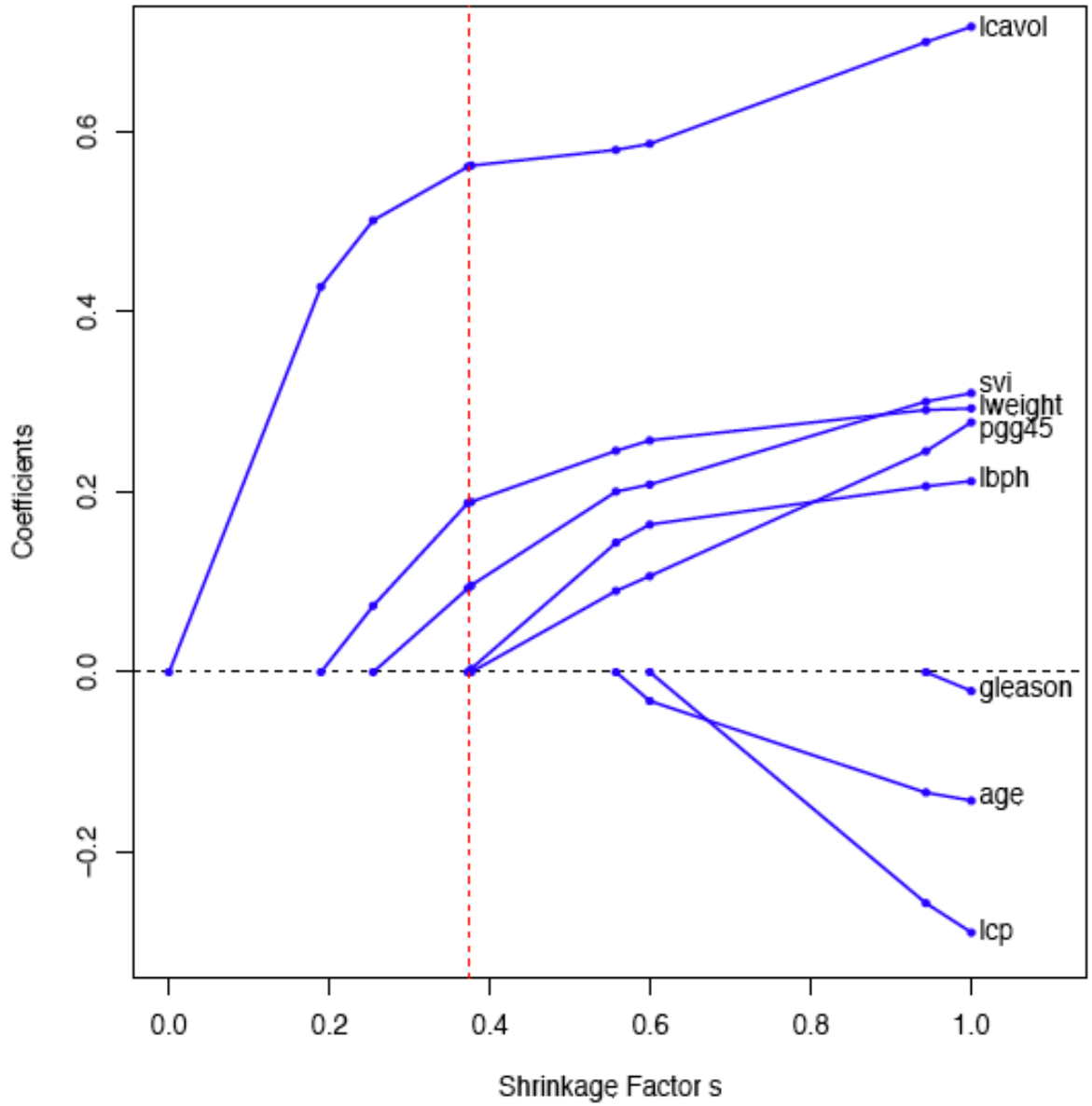


FIGURE 3.10. *Profiles of lasso coefficients, as the tuning parameter t is varied. Coefficients are plotted versus $s = t/\sum_1^p |\hat{\beta}_j|$. A vertical line is drawn at $s = 0.36$, the value chosen by cross-validation. Compare Figure 3.8 on page 65; the lasso*

... shrinkage

- ridge regression gives "proportional shrinkage"
- subset selection gives "hard thresholding" (some $\beta_j \to 0$)
- lasso gives "soft thresholding": blend of shrinkage and zeroing

- elastic net combines lasso and ridge regression

$$\min_{\beta} \left(\sum \{y_i - \beta_0 - \Sigma_{j=1}^{p} x_{ij}\beta_j\}^2 + \lambda_1 \sum_{j=1}^{p} |\beta_j| + \lambda_2 \sum_{j=1}^{p} \beta_j^2 \right)$$

- implemented in `R` in `library(glmnet)`

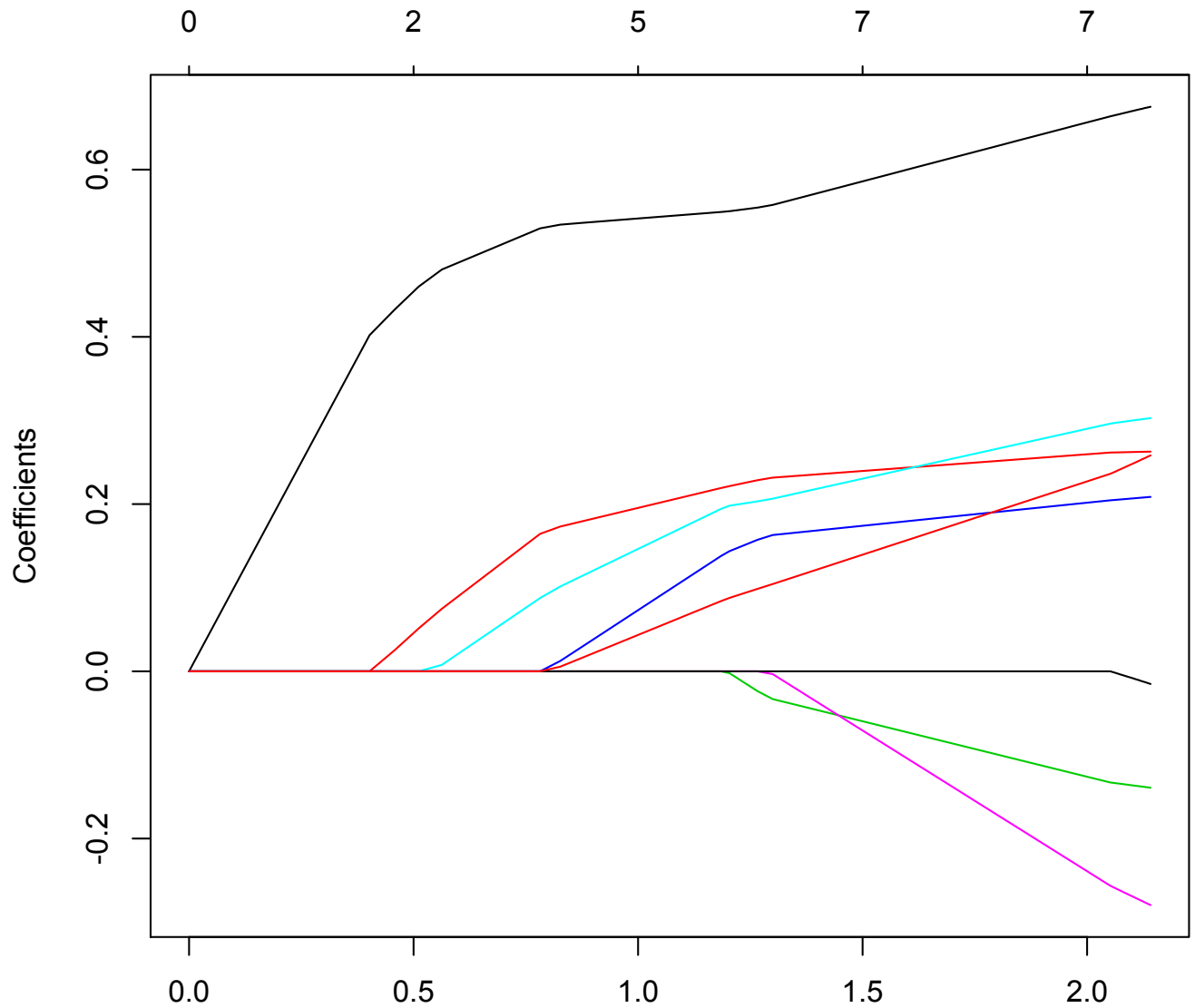
- estimates of coefficients are biased (but may have small mean-squared error)
- Lasso is now used as a variable selection method
- improvements in algorithms allow fast computation even for $p > n$

Prostate data

Ch.3, HTF

```
> prostate <- read.csv(file="prostate.data",sep="\t")
## data is at http://statweb.stanford.edu/~tibs/ElemStatLearn/datasets/prostate.data

> head(prostate)
      lcavol  lweight age      lbph svi       lcp gleason pgg45
1 -0.5798185 2.769459  50 -1.386294   0 -1.386294       6     0
2 -0.9942523 3.319626  58 -1.386294   0 -1.386294       6     0
3 -0.5108256 2.691243  74 -1.386294   0 -1.386294       7    20
4 -1.2039728 3.282789  58 -1.386294   0 -1.386294       6     0
5  0.7514161 3.432373  62 -1.386294   0 -1.386294       6     0
6 -1.0498221 3.228826  50 -1.386294   0 -1.386294       6     0
        lpsa train
1 -0.4307829  TRUE
2 -0.1625189  TRUE
3 -0.1625189  TRUE
4 -0.1625189  TRUE
5  0.3715636  TRUE
6  0.7654678  TRUE

> xp <- scale(prostate[,1:8])
> y <- prostate[,9]
> train <- prostate[,10]
## standardize data; y is the response (log psa); extract training data
##
> library(glmnet)
> pr.lasso <- glmnet(xp[train,],y[train]) #Lasso
> pr.lasso2 <- glmnet(xp[train,],y[train], alpha=0) # ridge
> plot(pr.lasso); plot(pr.lasso2)
```

... prostate data



... prostate data