STAT 545A
Class meeting #12 (The End)
Wednesday, October 17, 2012

Dr. Jennifer (Jenny) Bryan

Department of Statistics and Michael Smith Laboratories

Review of last class

Demonstrated the bootstrap approach to “two groups” testing on the yeast growth data.

Got bootstrap p-values for a non-standard test statistic and also for classical test statistics. Results very compatible.

Saw some tricks for bootstrapping or, for that matter, any resampling or simulation type of work. Explicit loop avoidance techniques include [1] generating the data at once and [2] apply-type functions for computing bootstrap statistics. set.seed() useful for making stochastic work repeatable, which can be invaluable for debugging.

Review of last class (cont'd)

Tried out some robust regression techniques using the life expectancy data for Rwanda in the Gapminder dataset.

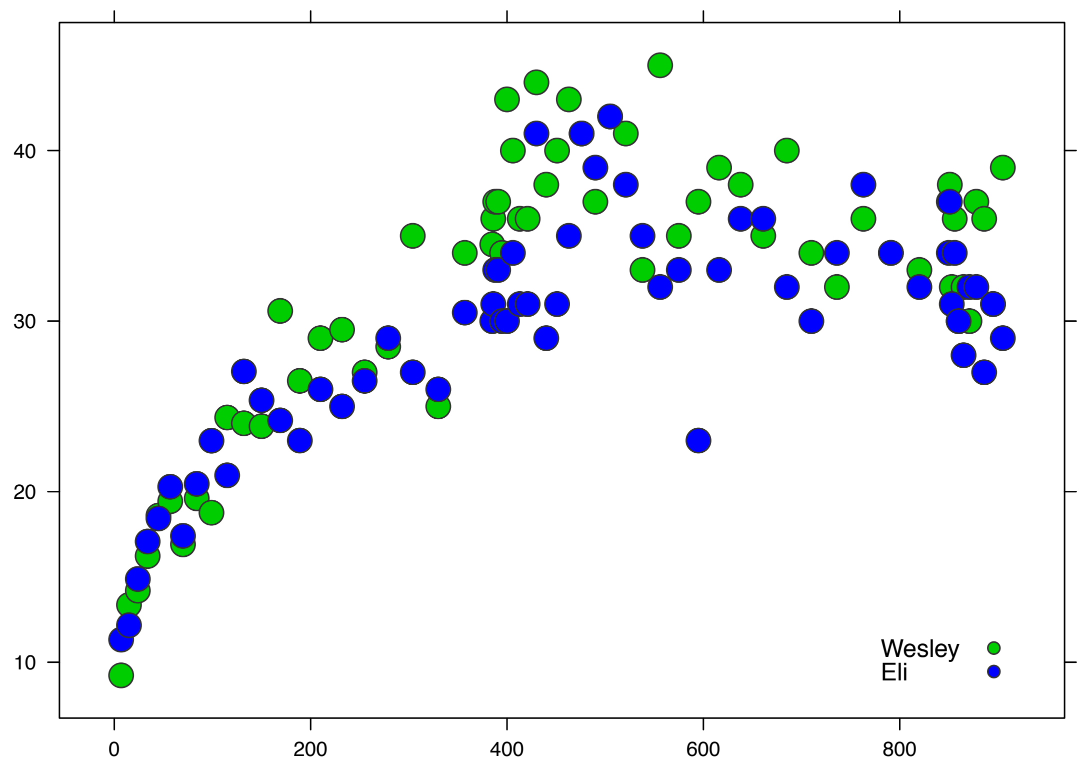
MM estimation is probably best general purpose option, implemented in lmrob() in robustbase package.

Revisited bootstrap for checking applicability of asymptotic standard errors and distribution. Results reassuring.

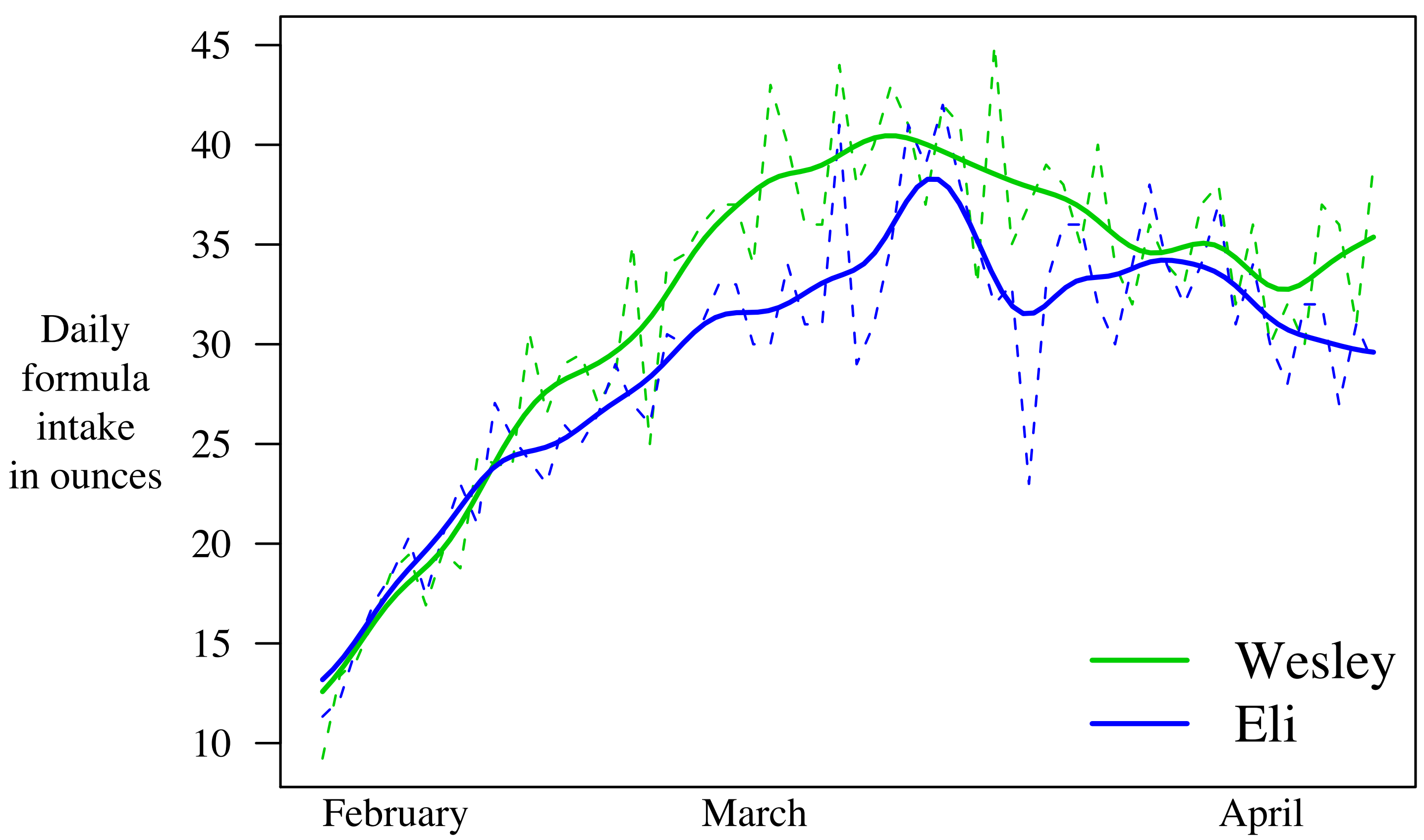
Brief remarks on data reshaping. Tips on avoiding it and on giving in and doing it "by hand" or with fancier functions and packages.

smoothing

Facilitate comparisons, identify trends.



Managing Inputs ... Nicer than Outputs



Two quantitative variables: X and Y

- X ... ‘independent variable’, ‘covariate’, ‘predictor’, ‘explanatory variable’
- Y ... ‘dependent variable’, ‘response’, ‘outcome’
- Regression ≈ study of the conditional expectation of Y given X=x

$$y_i = f(x_i) + \varepsilon_i$$

- Our focus: We believe f is smooth, but don’t wish to specify much else about f.
  - Nonparametric regression, smoothing, etc.

Two quantitative variables: X and Y

- X ... quantitative, often called the ‘independent variable’, ‘covariate’, ‘predictor’, ‘explanatory variable’
- Y ... quantitative, often called the ‘dependent variable’, ‘response’, ‘outcome’
- Regression refers generally to the study of the conditional distribution -- or, often, just the conditional expectation -- of Y given that X=x

$$Y \mid X = x \sim F_{\theta,\delta}$$

- Some parameters, referred to here as θ, are of direct interest. Others, referred to here as δ, are of secondary or no interest at all (“nuisance parameter”).

Nature of the parameter space

- The nature of the parameter (θ, δ) is one of the defining characteristics of the regression model
  - $(\theta, \delta) \in R^p \Rightarrow$ parametric model
  - $(\theta, \delta) \in R^p \otimes$ (sthg complicated, such as a function space) $\Rightarrow$ semi-parametric model
  - $(\theta, \delta) \in$ (sthg complicated) $\Rightarrow$ nonparametric model
- Examples of the 'complicated' spaces above: all possible distributions, all possible distributions with mean zero, etc.

Narrowing focus to expectation

- Next half-course covers generalized linear models, so I will not.
- Frees us to focus on the conditional expectation of Y given X = x

$$Y_{X=x} = f(x;\beta) + \varepsilon_x, E(\varepsilon_x) = 0$$

- Common scenario: parameter β is of primary interest and parameter(s) relating to the distribution of the error term ε are a nuisance.
- It is very common for $\beta \in R^p$, in which case assumptions about ε will determine whether the model is parametric or semi-parametric.

Nature of the regression function

- The nature of the regression function $f(x;\beta)$ is one of the defining characteristics of the regression model
  - f linear in $\beta \Rightarrow$ linear model
  - f not linear in $\beta \Rightarrow$ nonlinear model

$$y = \beta_0 + \beta_1 x + \underset{\uparrow}{\beta_2 x^2} + \varepsilon$$

Note: this IS a linear model.

Nonlinear regression

- Consider the case where the regression function -- the conditional expectation of Y given X = x -- is nonlinear in the parameter β:

$$Y_{X=x} = f(x;\beta) + \varepsilon_x, E(\varepsilon_x) = 0$$

- ‘Nonlinear regression’ is a huge area. Helpful to distinguish at least two broad classes within that:
    - Parametric nonlinear regression, e.g. $f(x_i) = \beta_0 + \beta_1 x_i^{\beta_2}$
    - Nonparametric regression, aka ‘smoothing’

Smoothing methods / nonparametric regression

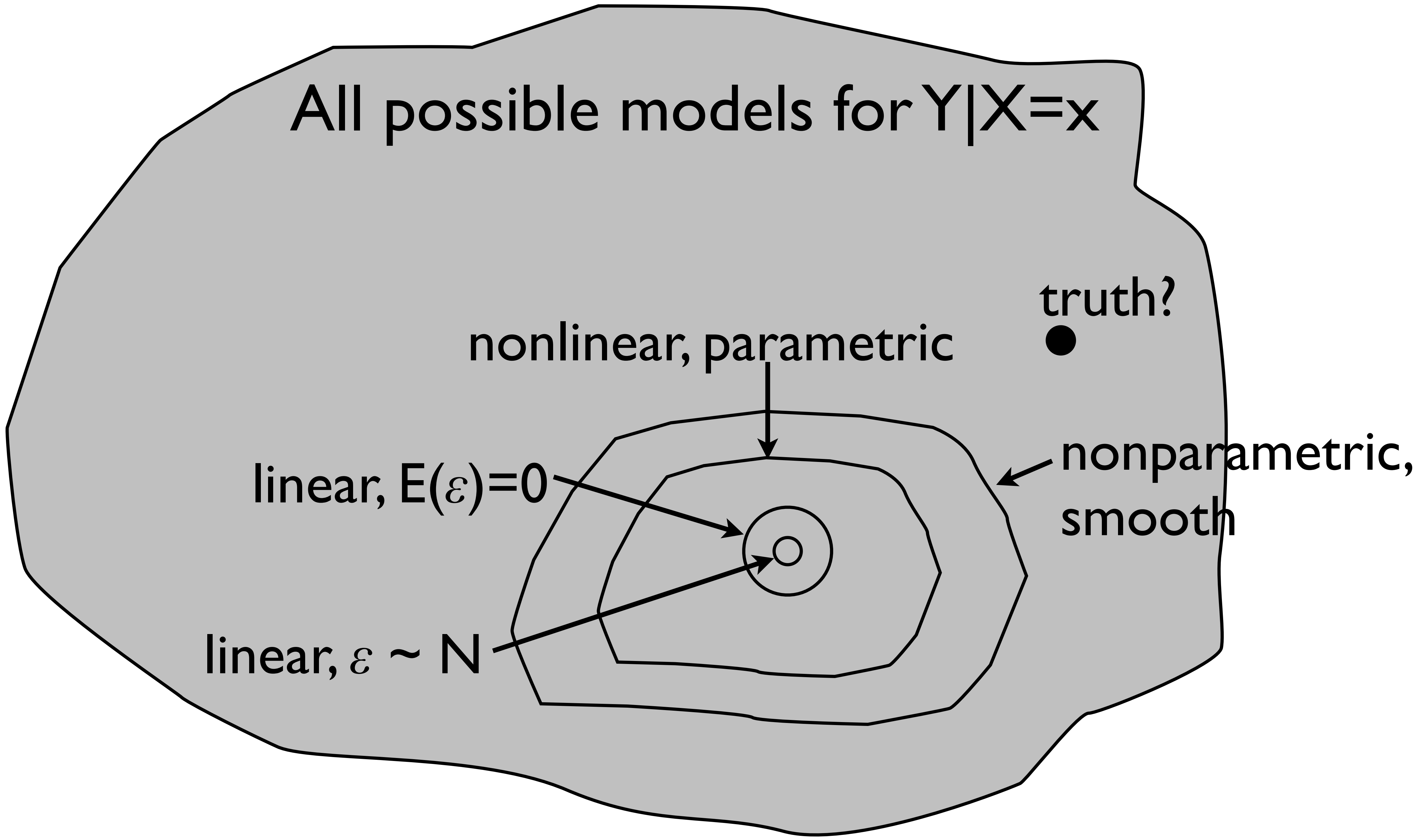
- Kernel-based methods
  - will tie in nicely to kernel-based density estimation
- Local polynomials (e.g. loess)
- Splines -- WILL NOT COVER

$$y_i = x_i\beta + \varepsilon_i$$

linear model
so plain yet so useful
why must I move on?

$$y_i = f(x_i) + \varepsilon_i$$

nonlinear f
my head hurts already
what’s in it for me?



Nonlinear models allow us to explore a larger portion of the space of 'all possible models' and, therefore, increase the chance that we'll study a model that's 'close enough' to the truth to actually learn something useful.

✧Please note: THIS IS JUST A CARTOON! Not to be over-interpreted.

Some pros and cons

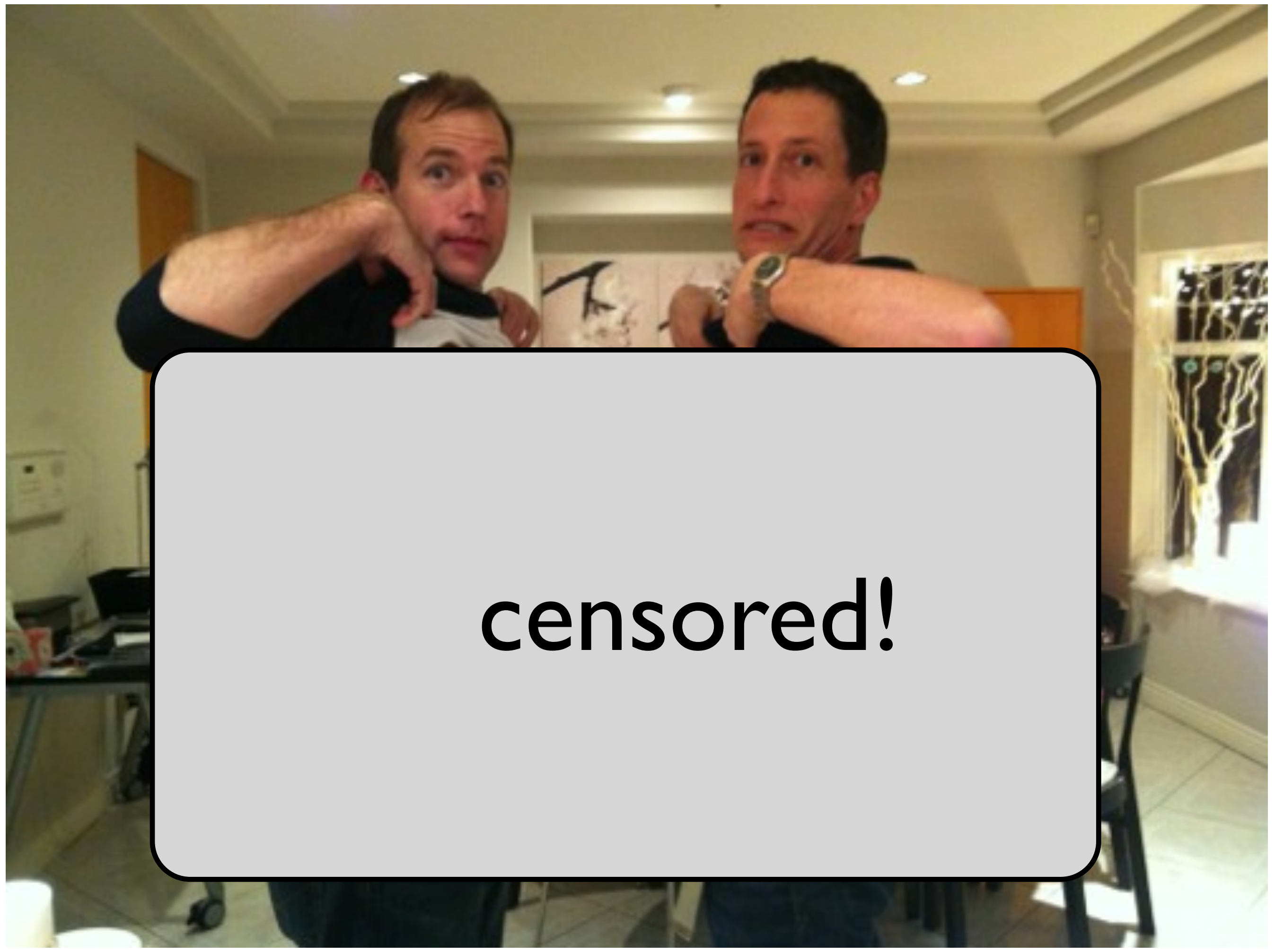
	Parametric regression function	Nonparametric regression function
Pros	Easy to write down Easy to fit Efficient, if model correct Parameters often have meaning Can easily incorporate prior/ external info	So flexible! Less biased, most of the time Leads to appealing figures accessible to many
Cons	Almost certainly doesn't contain true data-generating model	Hard to store, communicate, utilize the fitted result Path to inference not so clear

✧Please note: Broad generalities but their usefulness seems to outweigh any potential harm.

my husband, Jim
weight loss bet w/ friend Craig
late August 2011 to New Year's Eve 2011
winner = he who loses most as a percentage of starting weight
Craig won, by the way :(
data collection continues ....

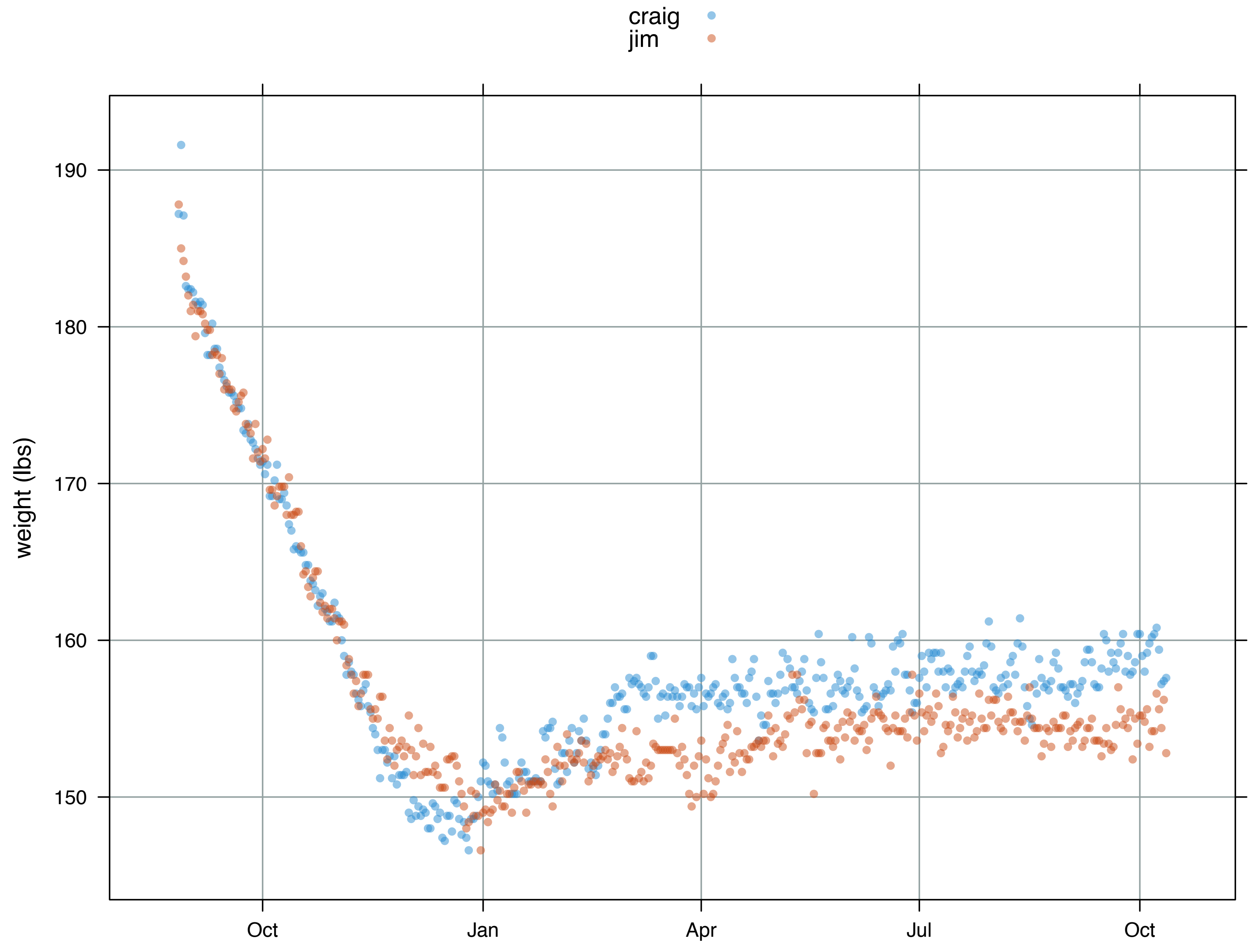
during

before

after!

censored!

```
> peek(wDat)
           date   jim craig
55   2011-10-20 163.4 164.8
105  2011-12-09 151.6 148.0
131  2012-01-04 149.0 150.8
170  2012-02-12 152.2 155.0
174  2012-02-16 152.0 151.8
233  2012-04-15 152.2 157.6
249  2012-05-01 152.6 156.6

> dim(wDat)
[1] 413   3
```

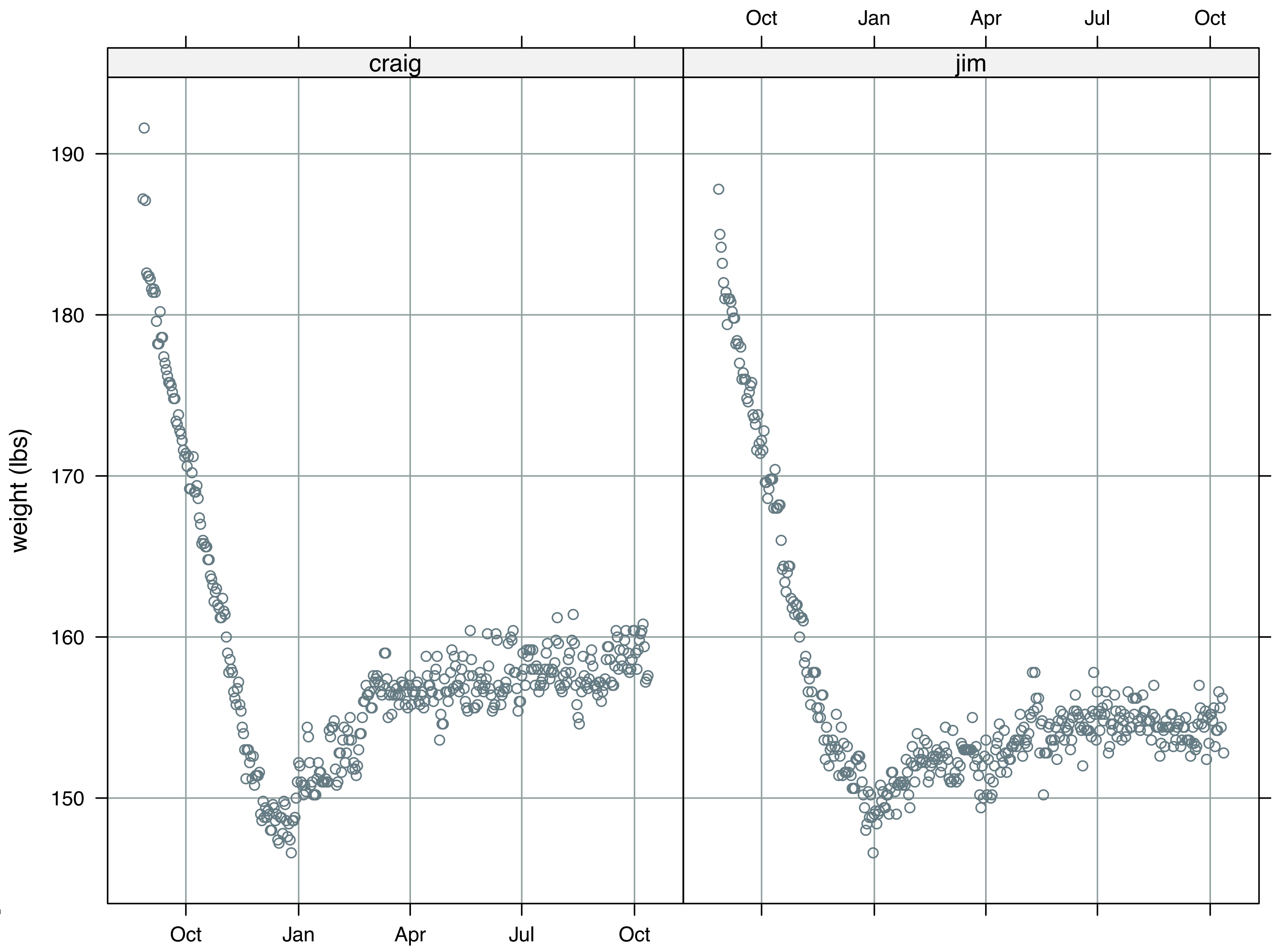
most natural way to record the data is ... short and fat



```
xyplot(craig + jim ~ date, wDat,
       grid = TRUE, auto.key = TRUE,
       xlab = jXlab, ylab = jYlab)
```

"extended formula interface" in lattice lets us do fake grouping / superposition

```
> peek(wDat)
          date   jim craig
55  2011-10-20 163.4 164.8
105 2011-12-09 151.6 148.0
131 2012-01-04 149.0 150.8
170 2012-02-12 152.2 155.0
174 2012-02-16 152.0 151.8
233 2012-04-15 152.2 157.6
249 2012-05-01 152.6 156.6

> dim(wDat)
[1] 413   3
```



most natural way to record the data is ... short and fat

```
xyplot(craig + jim ~ date, wDat,
       grid = TRUE, auto.key = TRUE, outer = TRUE,
       xlab = jXlab, ylab = jYlab, pch = 1)
```

"extended formula interface" in lattice lets us do fake multi-panel conditioning too

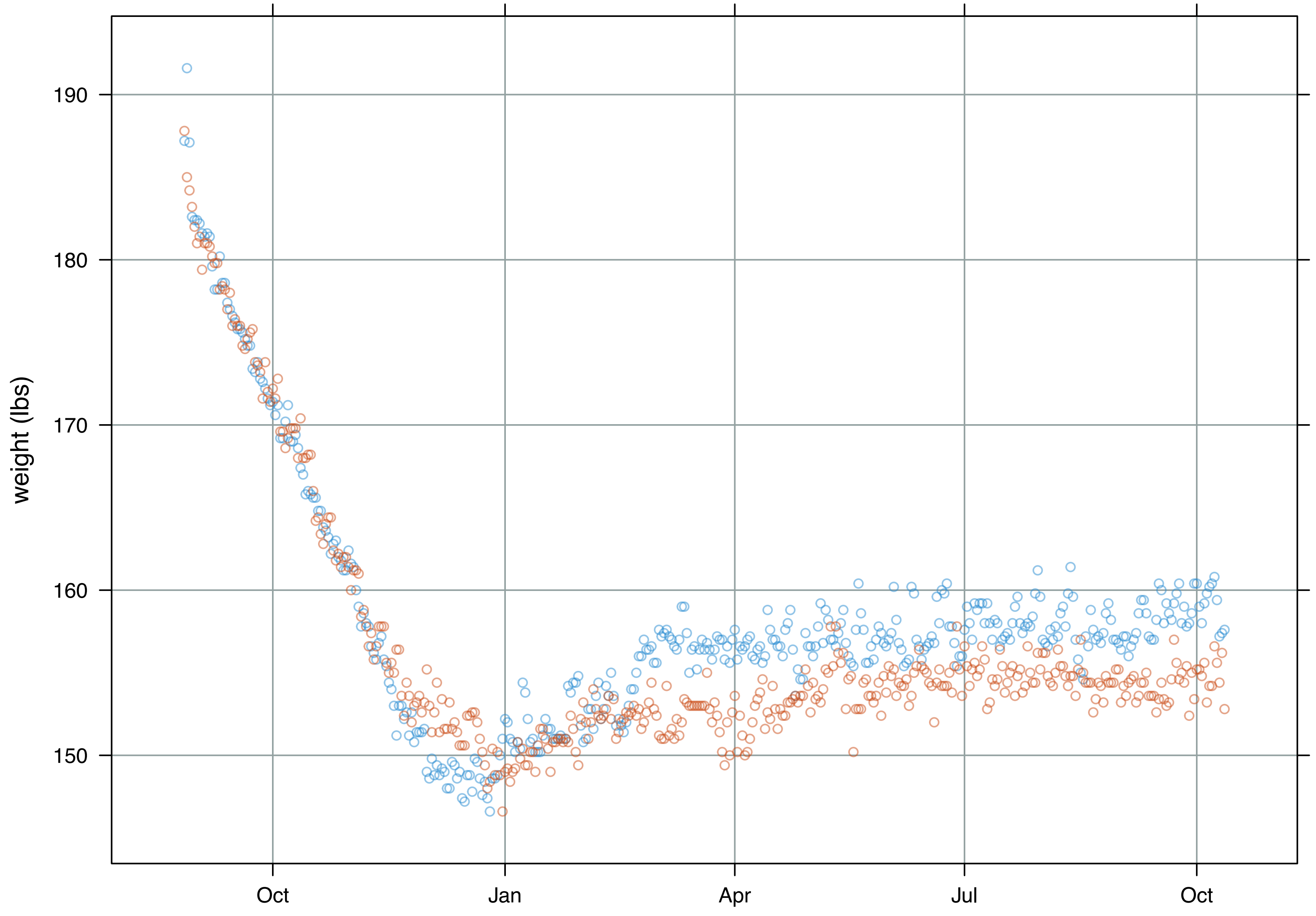
low-tech 'manual' data reshaping

```
> peek(wDat)
          date   jim craig
55  2011-10-20 163.4 164.8
105 2011-12-09 151.6 148.0
131 2012-01-04 149.0 150.8
170 2012-02-12 152.2 155.0
174 2012-02-16 152.0 151.8
233 2012-04-15 152.2 157.6
249 2012-05-01 152.6 156.6

> dim(wDat)
[1] 413   3
```

```
## reshape the data
xDat <- with(wDat,
             data.frame(date = date,
                        wt = c(jim, craig),
                        who = factor(rep(c("jim", "craig"),
                                         each = nrow(wDat)))))
```

```
> peek(xDat)
          date    wt   who
85  2011-11-19 156.4   jim
235 2012-04-17 152.8   jim
243 2012-04-25 153.6   jim
276 2012-05-28 154.4   jim
359 2012-08-19 154.4   jim
555 2012-01-15 150.2 craig
773 2012-08-20 158.8 craig

> dim(xDat)
[1] 826   3
```

```
> peek(xDat)
           date    wt   who
85   2011-11-19 156.4   jim
235  2012-04-17 152.8   jim
243  2012-04-25 153.6   jim
276  2012-05-28 154.4   jim
359  2012-08-19 154.4   jim
555  2012-01-15 150.2 craig
773  2012-08-20 158.8 craig

> dim(xDat)
[1] 826   3
```

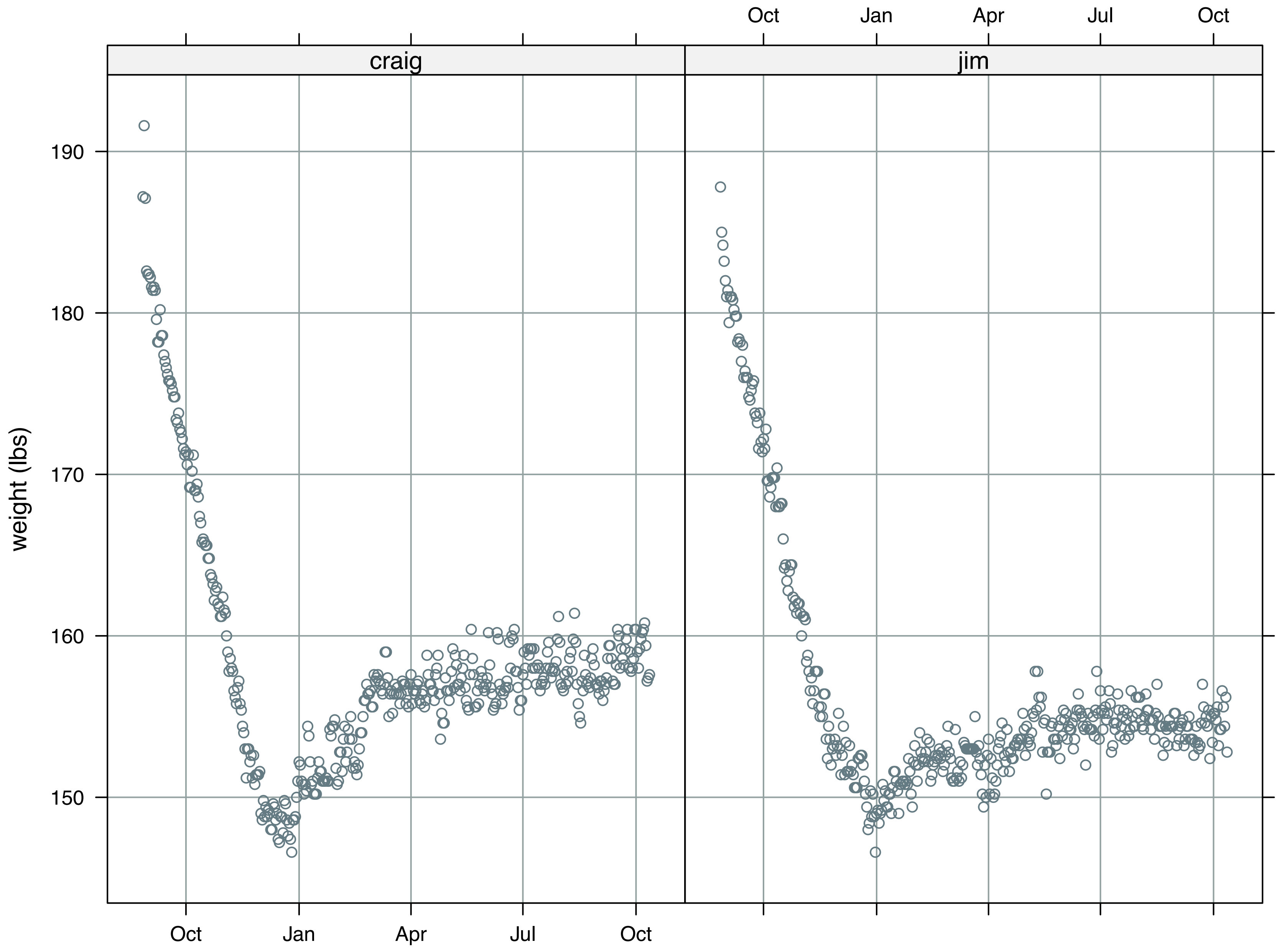


```
xyplot(wt ~ date, xDat,
       groups = who,
       grid = TRUE, auto.key = TRUE,
       xlab = jXlab, ylab = jYlab, pch = 1)
```

store tall in tall and skinny format to facilitate data aggregation and visualization

proper grouping / superposition via ‘groups’

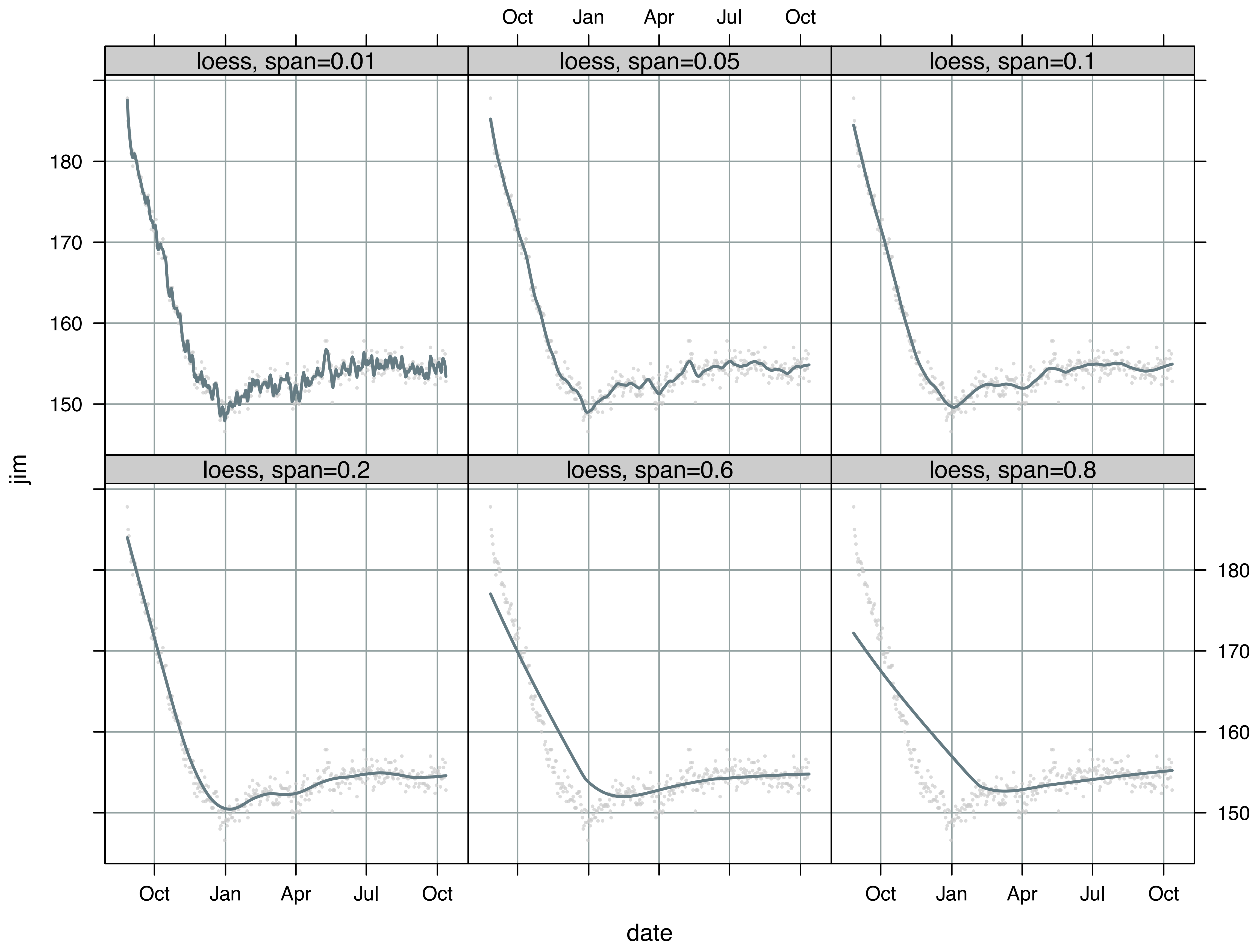
```
> peek(xDat)
           date    wt   who
85   2011-11-19 156.4   jim
235  2012-04-17 152.8   jim
243  2012-04-25 153.6   jim
276  2012-05-28 154.4   jim
359  2012-08-19 154.4   jim
555  2012-01-15 150.2 craig
773  2012-08-20 158.8 craig

> dim(xDat)
[1] 826   3
```

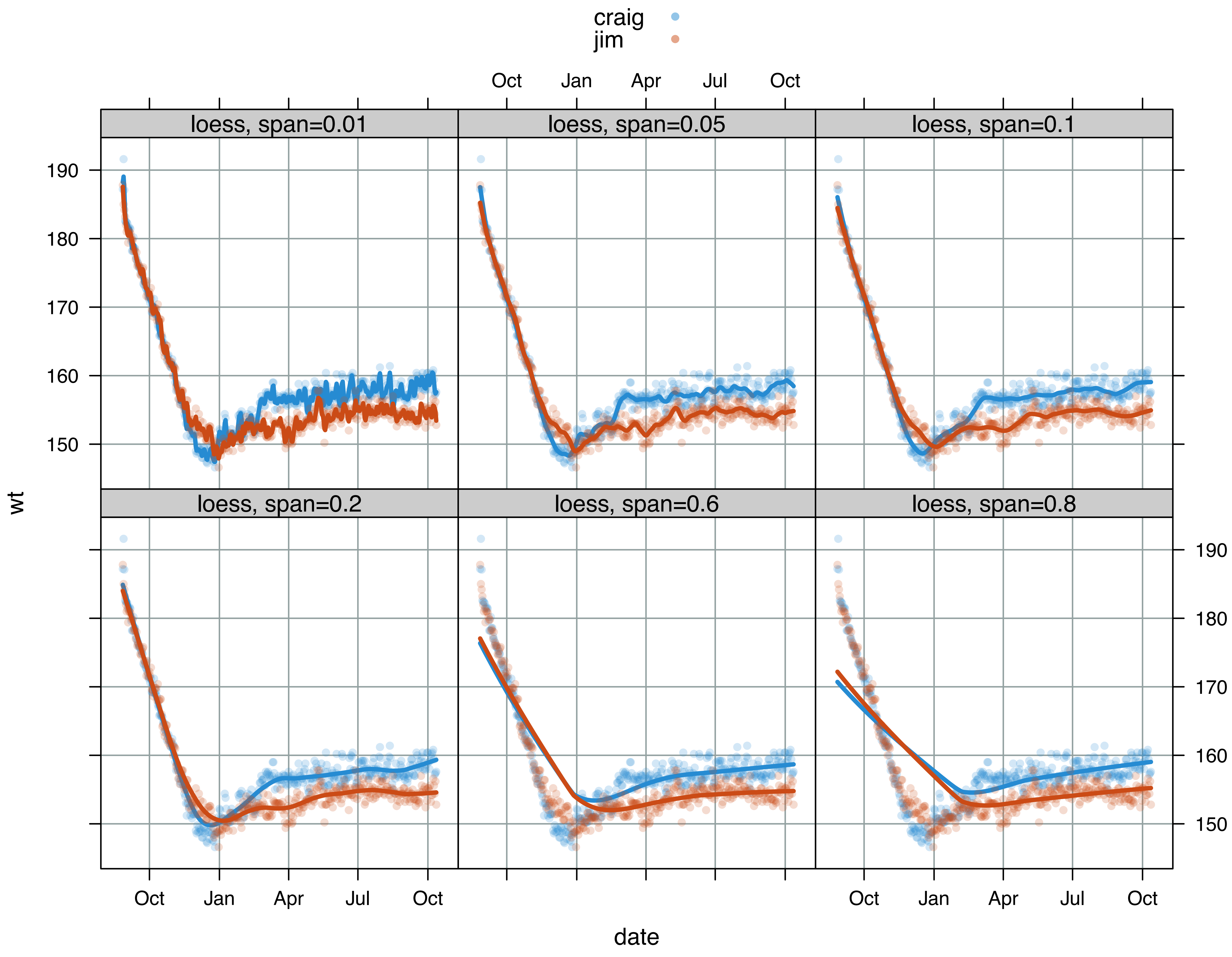


store tall in tall and skinny format to facilitate data aggregation and visualization

```
xyplot(wt ~ date | who, xDat,
       grid = TRUE,
       xlab = jXlab, ylab = jYlab, pch = 1)
```

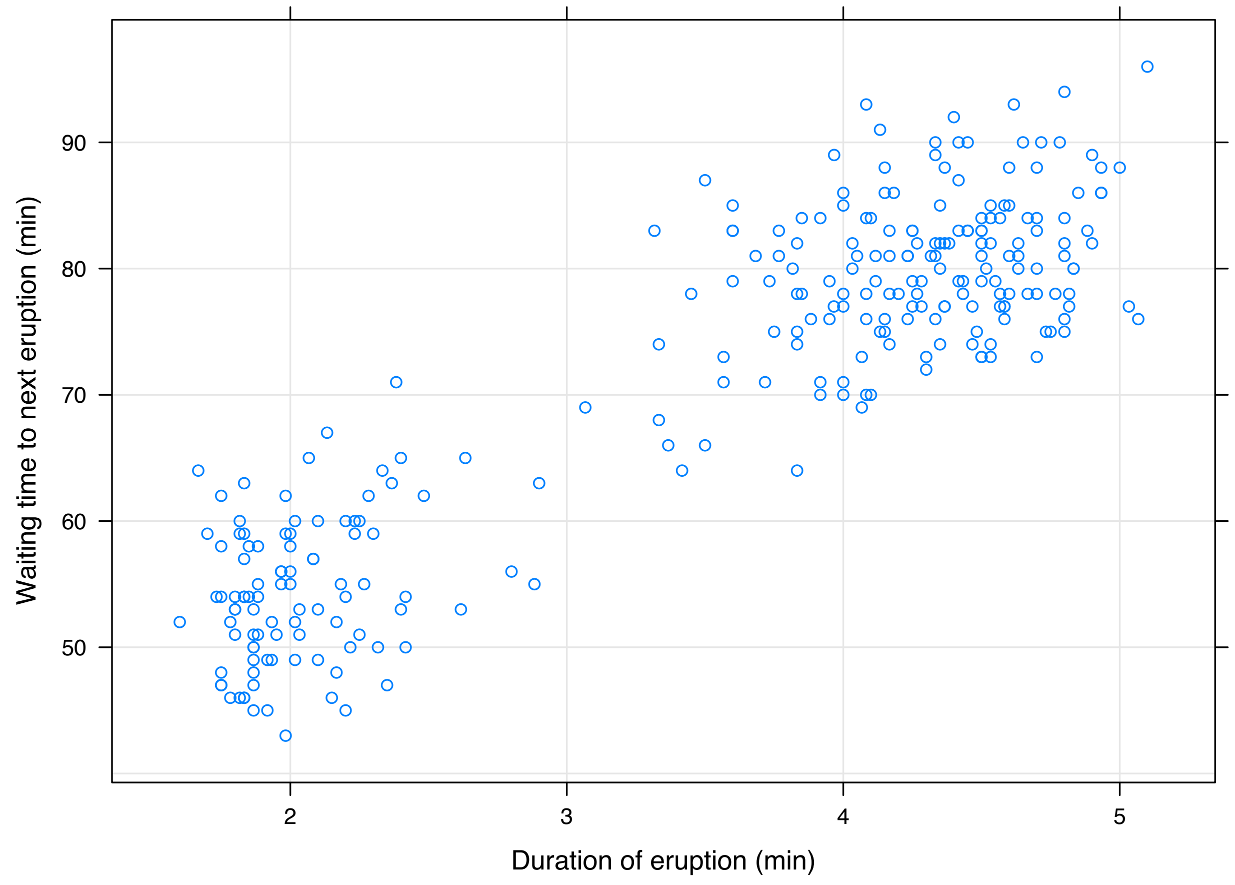
proper multi-panel conditioning via ‘y ~ x | z’


loess, span=0.01
loess, span=0.05
loess, span=0.1
loess, span=0.2
loess, span=0.6
loess, span=0.8
jim
date
150
160
170
180
Oct
Jan
Apr
Jul
Oct

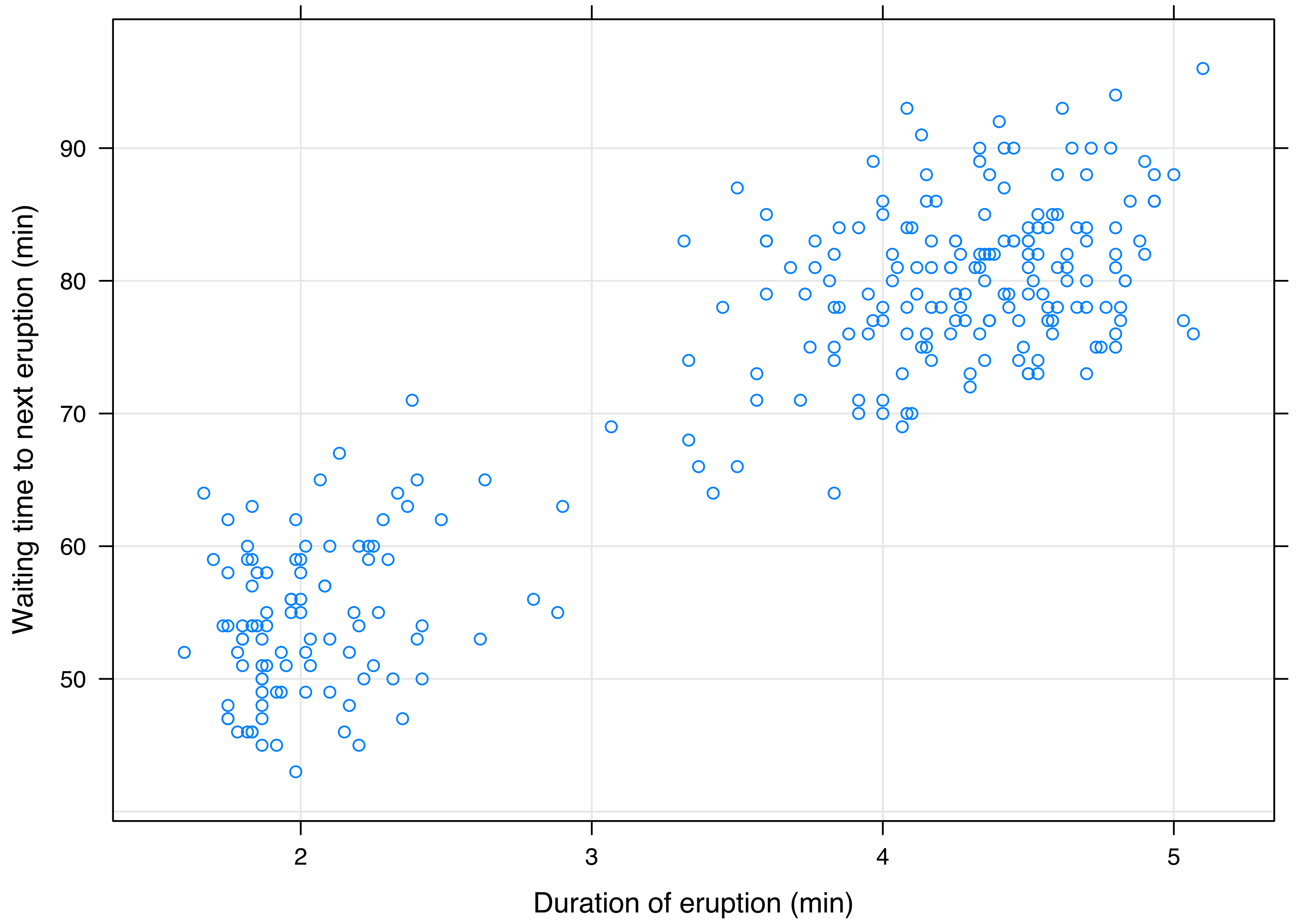

craig
jim
loess, span=0.01
loess, span=0.05
loess, span=0.1
loess, span=0.2
loess, span=0.6
loess, span=0.8
Oct
Jan
Apr
Jul
Oct
150
160
170
180
190
wt
date

Selected methods for smoothing / nonparametric regression

- Kernel-based methods
- Local polynomials (e.g. loess)
- (Splines)

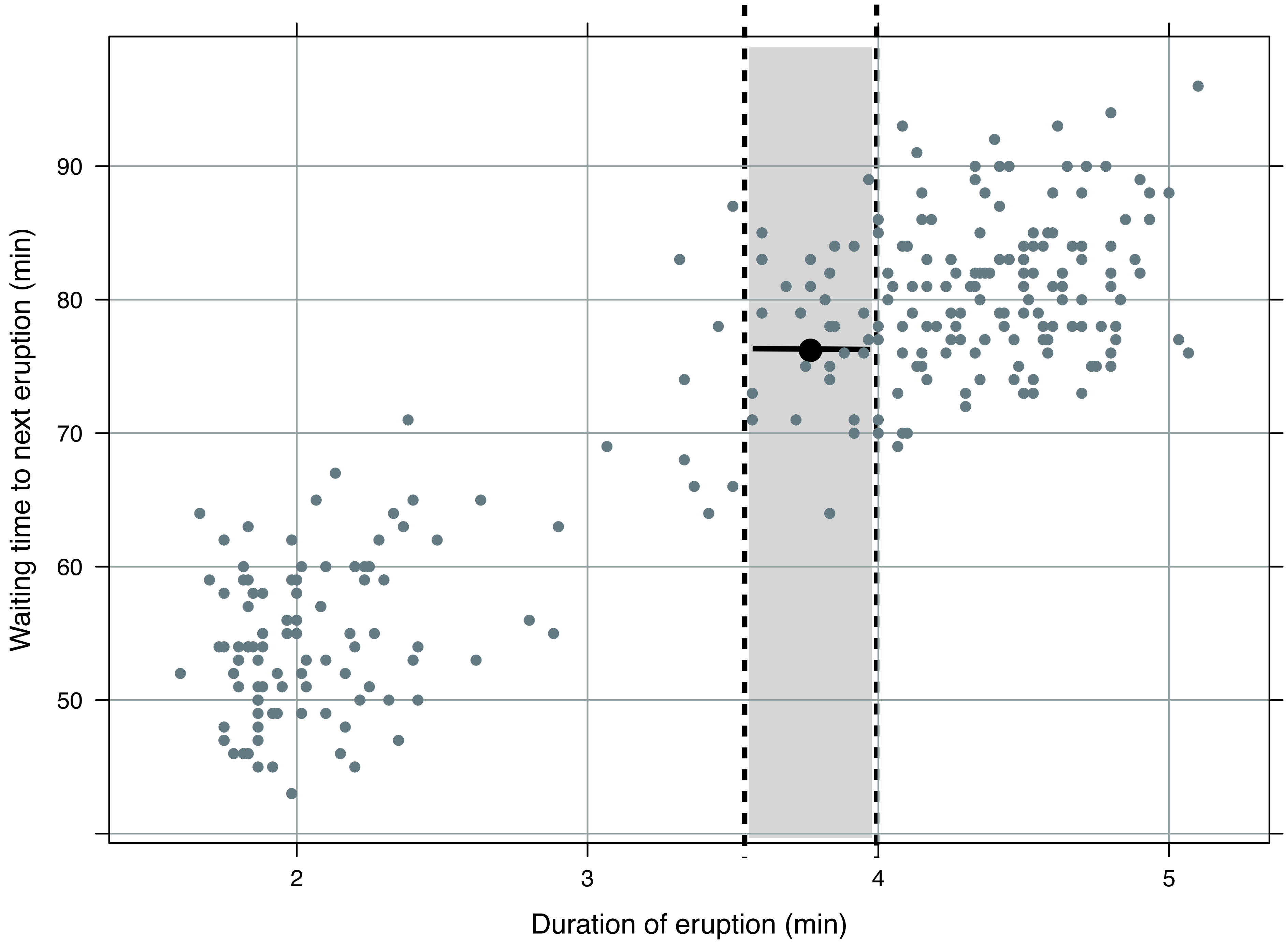


Old Faithful geyser data (`faithful`)



Key concept: Moving or running average

$\hat{f}(x_0)$ = avg of *y*'s from all observations with *x*'s falling in a window around x_0 (e.g. nearest 10 observations or obs where $x \in x_0 \pm$ sthg)



Old Faithful geyser data (`faithful`)

Key innovation for *kernel smoothing*: Use a *weighted* moving average. The kernel specifies the weights. The weight of a point close *x_0* is generally greater than that of a point that is far.

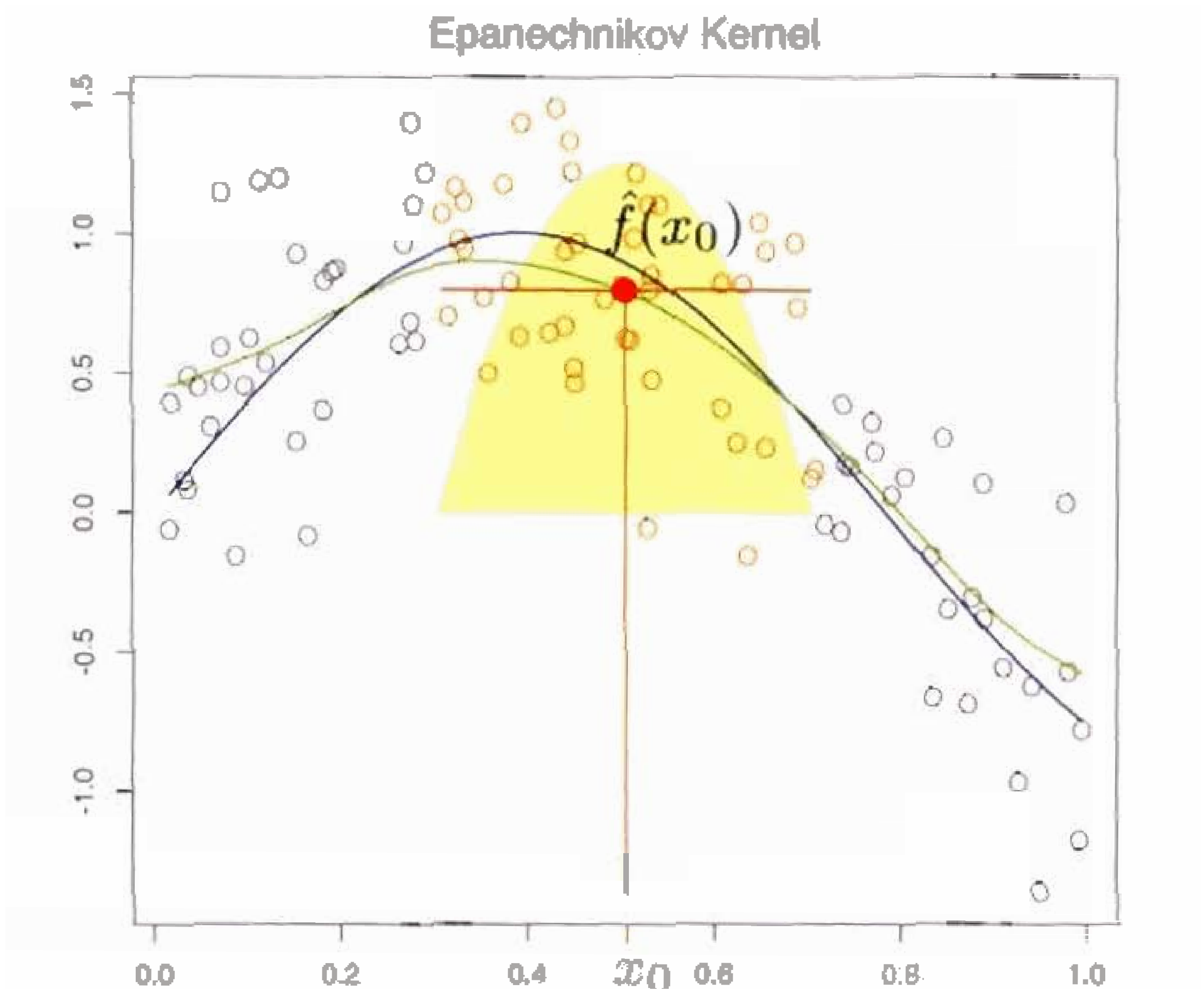
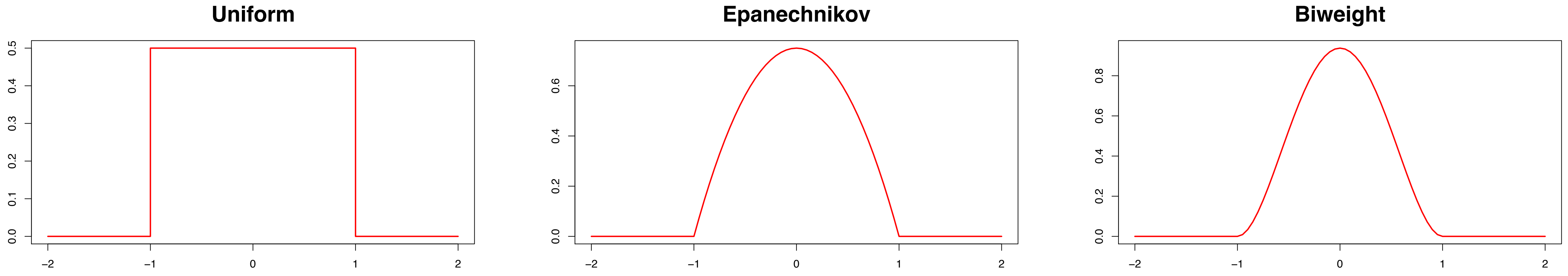
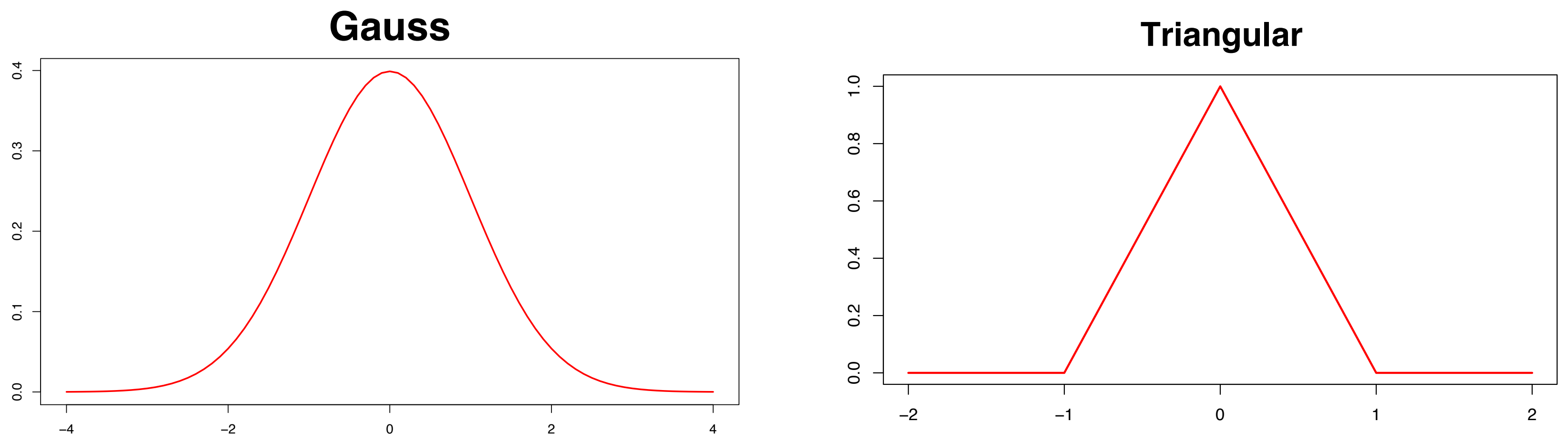


Figure from "The Elements of Statistical Learning", by by Hastie, Tibshirani, and Friedman (2001), Springer-Verlag.

http://compdiag.molgen.mpg.de/docs/talk_05_01_04_stefanie.pdf





Some popular kernel functions

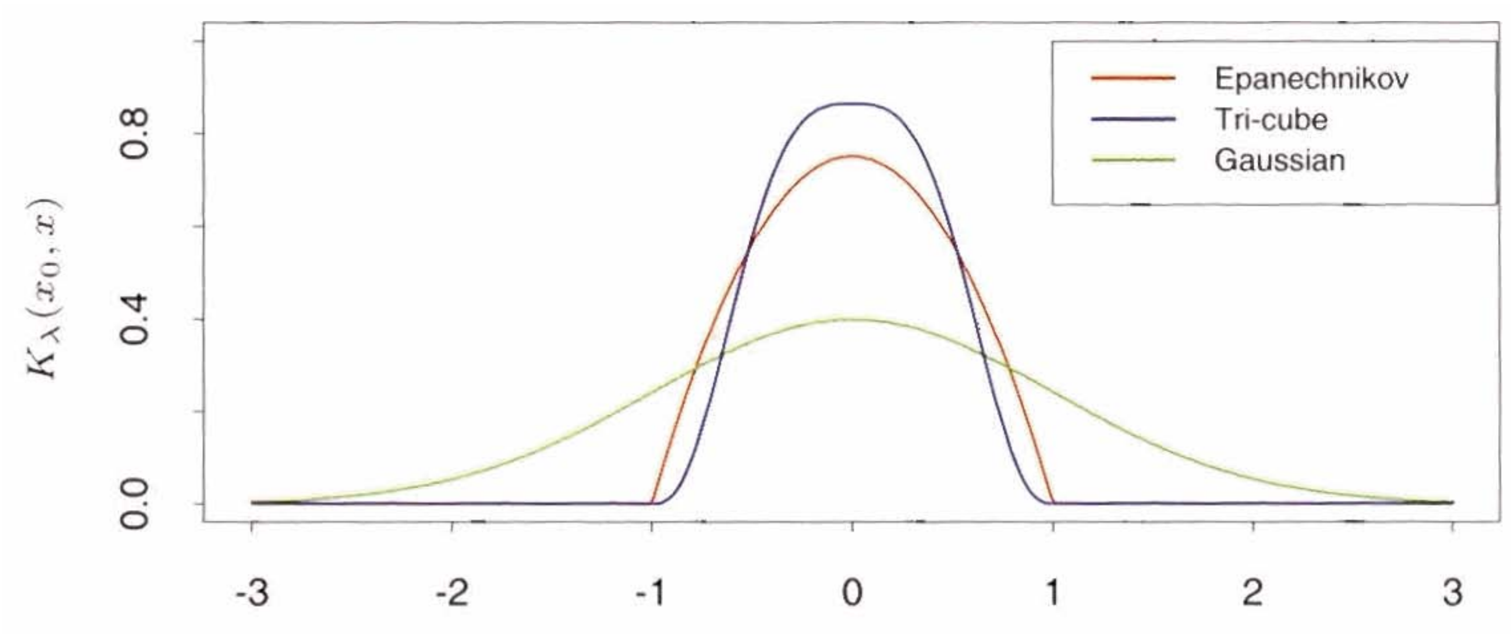


Figure from "The Elements of Statistical Learning", by by Hastie, Tibshirani, and Friedman (2001), Springer-Verlag.

Recall kernel density estimation?

$$\hat{f}_b(x) = \frac{1}{nb}\sum_i K\left(\frac{x - x_i}{b}\right) = \frac{1}{n}\sum_i w_i$$

where the weight w_i is given by

$$w_i = \frac{1}{b}K\left(\frac{x - x_i}{b}\right)$$

f is a density

Well, here's a kernel smoother:

$$\hat{f}_b(x) = \frac{1}{nb}\sum_i K\left(\frac{x - x_i}{b}\right) y_i = \frac{1}{n}\sum_i w_i y_i$$

where the weight w_i is given by

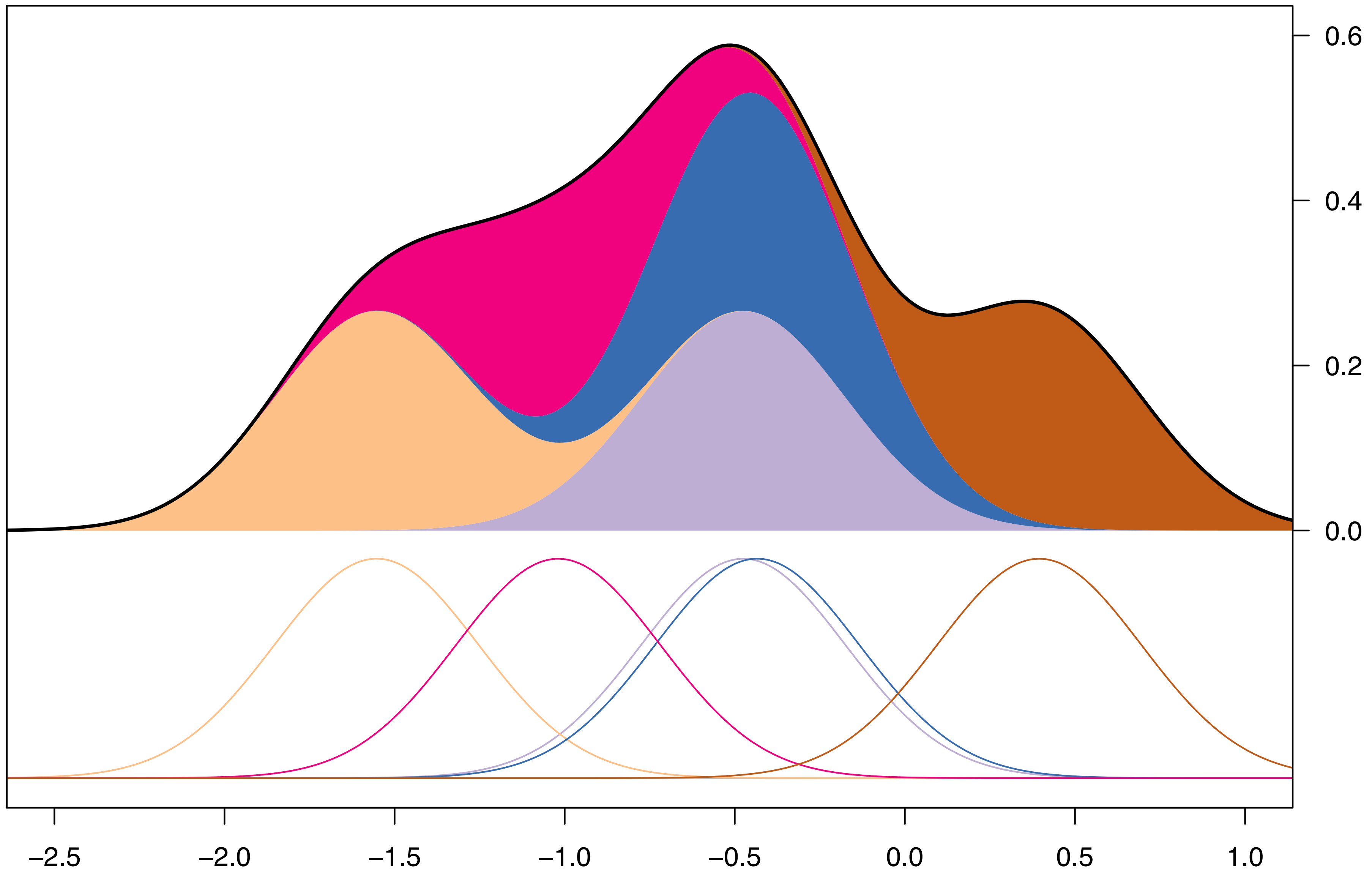
$$w_i = \frac{1}{b}K\left(\frac{x - x_i}{b}\right)$$

f is the regression function

Themes of kernel-based methods

- A kernel density estimator is a glorified histogram.
- A kernel smoother is a glorified ‘moving average’.
- Key concept: instead of the ‘hard’ contribution made by each point in a histogram or moving average (two options: contributes / does not contribute), contribution of each point to the density estimate or regression function varies with distance, i.e. is high in neighborhood of the point and low elsewhere
  - The kernel function describes exactly how this contribution varies with distance.
  - Introduces a smoothing parameter, denoted b earlier. Often called the bandwidth in this context.

Illustration of kernel density estimation



Produced from code at the R graph gallery

Kernel smoother

- Moving average corresponds to a rectangular or nearest-neighbor kernel.
- Qualities we often like in a kernel
  - Compact support: Otherwise every point makes a contribution to the regression function everywhere.
  - Smooth.
- Far more crucial than choice of kernel is ... choice of bandwidth.
  - Subjective method: eye ball it!
  - Cross validation -- and even the bootstrap -- can be extremely useful here.

Nadaraya-Watson estimator

$$\hat{f}_b(x) = \frac{\sum_i w_i y_i}{\sum_i w_i}$$

Modifies the 'moving average' estimator to make it a more truly a moving average. When forming the estimate at x, the effective weights will sum to one.

Using this modification, here's the typical kernel smoother, again:

$$\hat{f}_b(x) = \frac{\sum_i K_b(x,x_i) y_i}{\sum_i K_b(x,x_i)}$$

$$\text{where } K_b(x,x_i) = K\left(\frac{x - x_i}{b}\right)$$

Epanechnikov kernel

$$K(z_i) = \begin{cases} \frac{3}{4}(1 - z^2) \text{ if } |z| < 1 \\ 0 \text{ otherwise} \end{cases}$$

Optimal choice under some standard assumptions.
Minimizes asymptotic mean integrated squared error.

*Despite its optimality, in R it is much easier to find software for Gaussian kernel.

Packages and functions for kernel smoothing

ksmooth, a really basic function, only does rectangular and Gaussian kernel. Don't use this. Even the help file says that!

KernSmooth package (only seems to offer Gaussian kernel).

sm package (only seems to offer Gaussian kernel).

lpridge package offers Epanechnikov kernel.

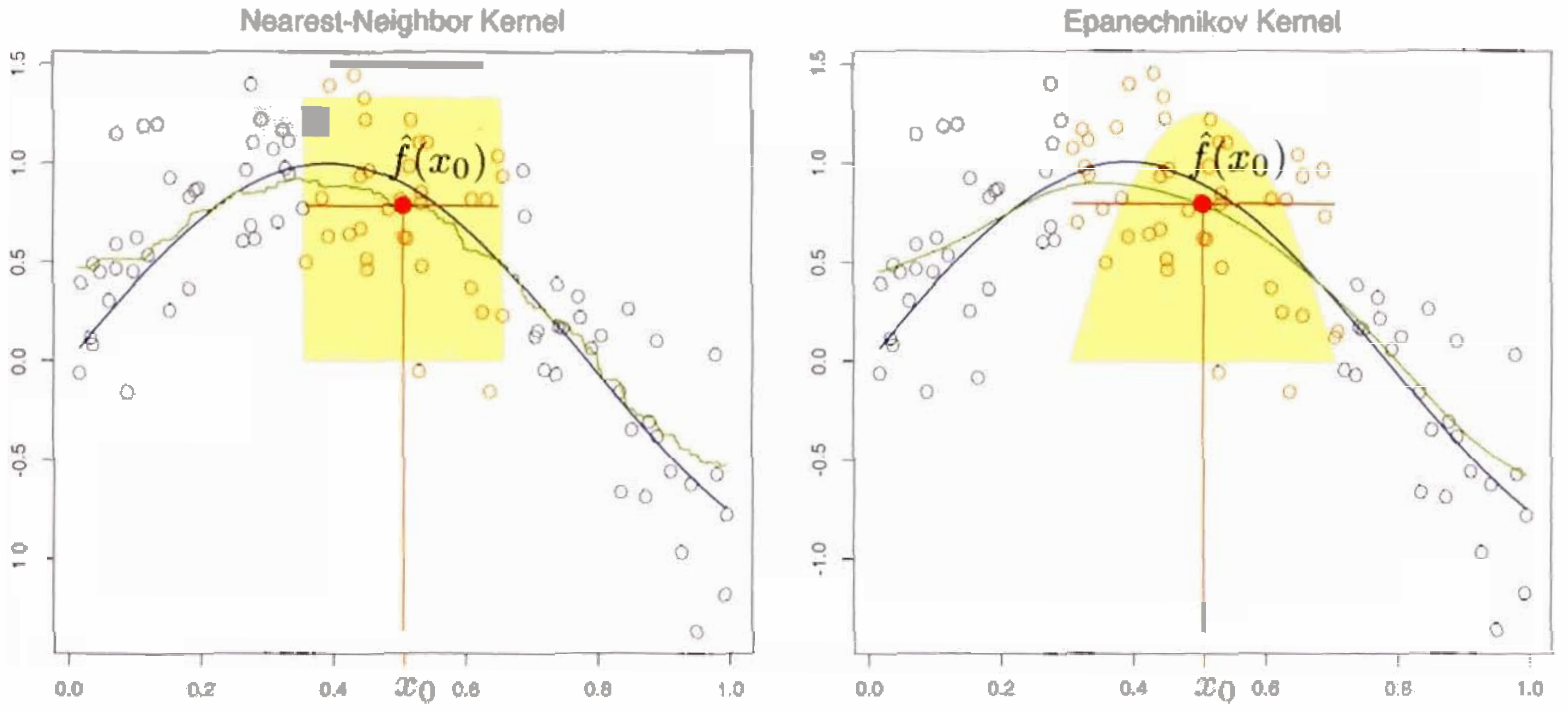


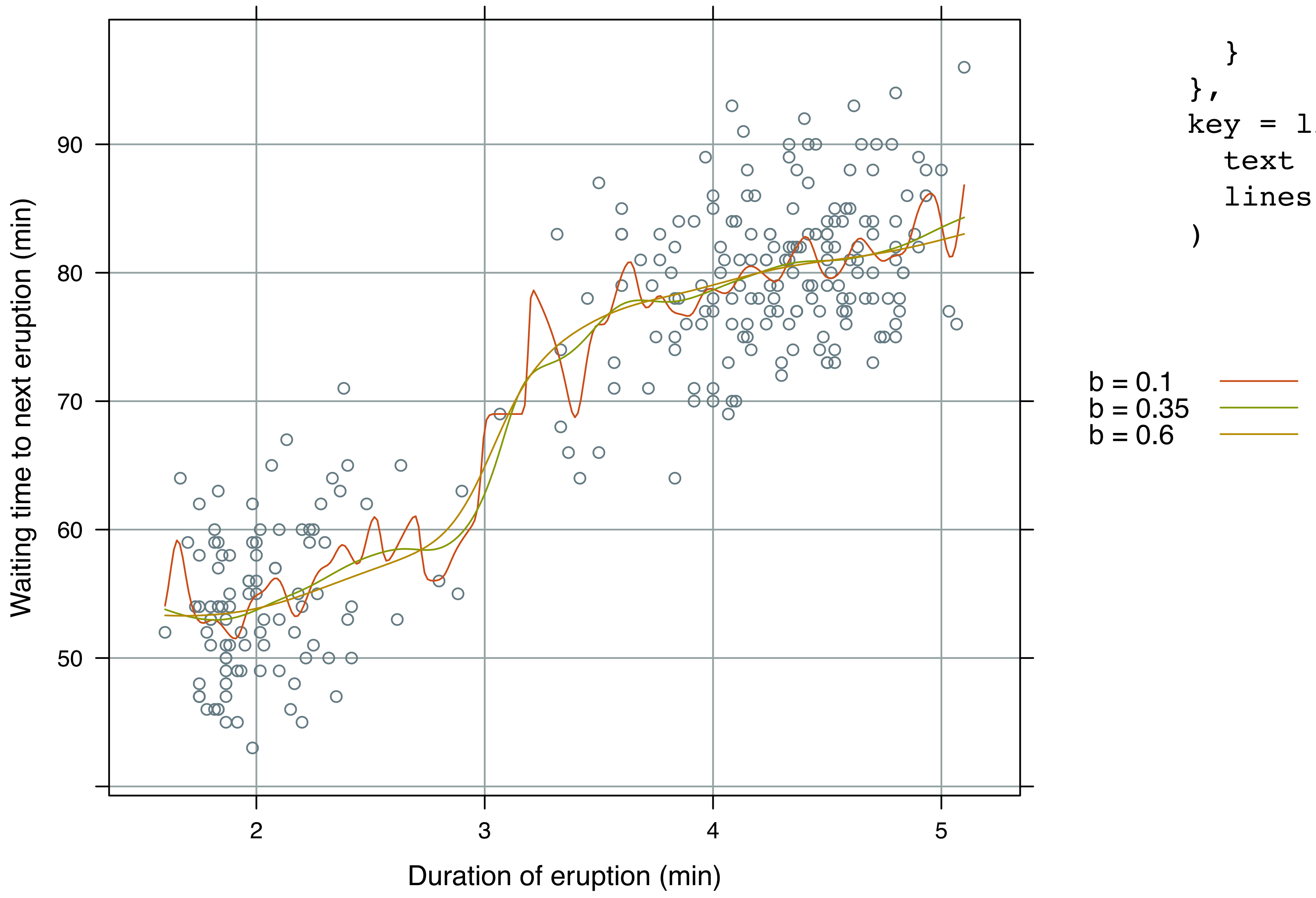
FIGURE 6.1. *In each panel* 100 *pairs* x_i, y_i *are generated at random from the blue curve with Gaussian errors:* $Y = \sin(4X) + \varepsilon$, $X \sim U[0,1]$, $\varepsilon \sim N(0, 1/3)$. *In the left panel the green curve is the result of a* 30*-nearest-neighbor running-mean smoother. The red point is the fitted constant* $\hat{f}(x_0)$, *and the orange shaded circles indicate those observations contributing to the fit at* x_0. *The solid orange region indicates the weights assigned to observations. In the right panel, the green curve is the kernel-weighted average, using an Epanechnikov kernel with (half) window width* $\lambda = 0.2$.

Figure from "The Elements of Statistical Learning", by by Hastie, Tibshirani, and Friedman (2001), Springer-Verlag.

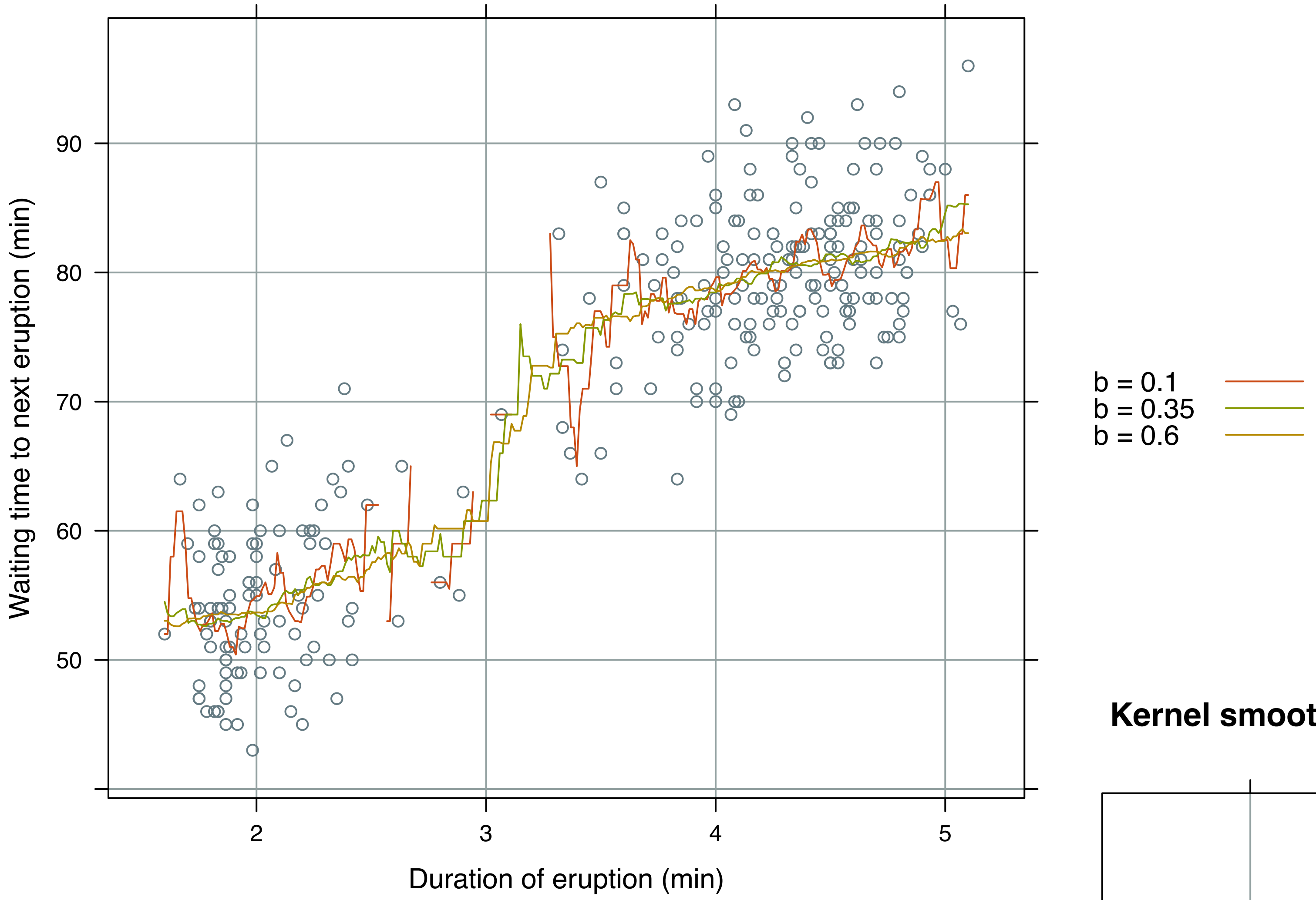
```
> ## trying ksmooth out for first time
> jSmooth <-
+   ksmooth(faithful$eruptions, faithful$waiting,
+           kernel = "normal", bandwidth = 1)
> str(jSmooth)
List of 2
 $ x: num [1:272] 1.60 1.61 1.63 1.64 1.65 ...
 $ y: num [1:272] 53.6 53.6 53.6 53.6 53.6 ...
```

```
xyplot(waiting ~ eruptions, faithful,
       xlab = "Duration of eruption (min)",
       ylab = "Waiting time to next eruption (min)",
       main = "Kernel smoother, ...",
       panel = function(x, y, ...) {
         panel.grid(h = -1, v = -1)
         panel.xyplot(x, y, ...)
         for(i in seq_along(jBand)) {
           jSmooth <- ksmooth(x, y, kernel = "normal",
                              bandwidth = jBand[i])
           panel.lines(jSmooth$x, jSmooth$y,
                       col.line = jCols[i], ...)
         }
       },
       key = list(space = "right",
         text  = list(paste("b =", jBand)),
         lines = list(col = jCols[1:length(jBand)]))
       )
```

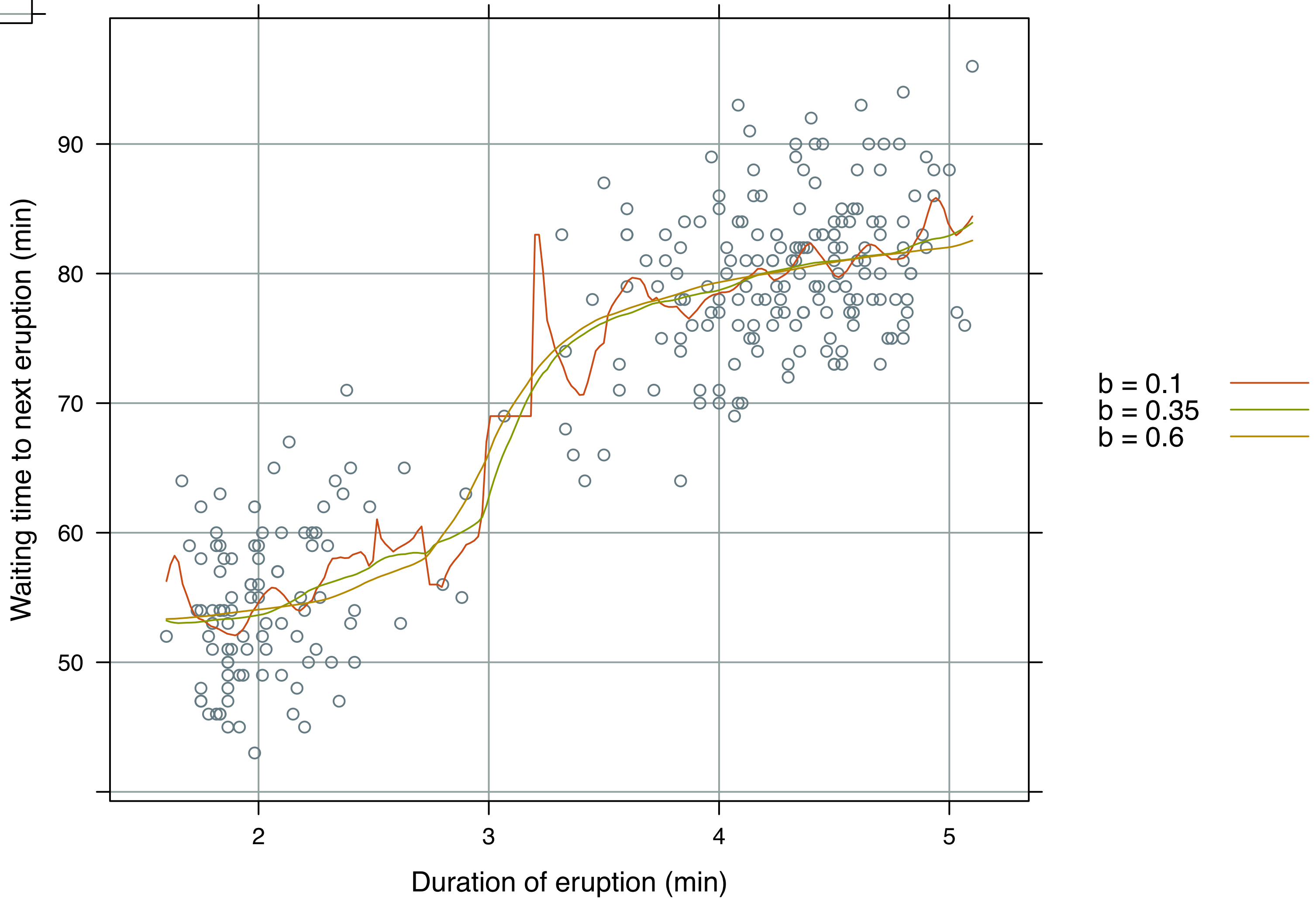
Kernel smoother, implemented by ksmooth, Gaussian kernel



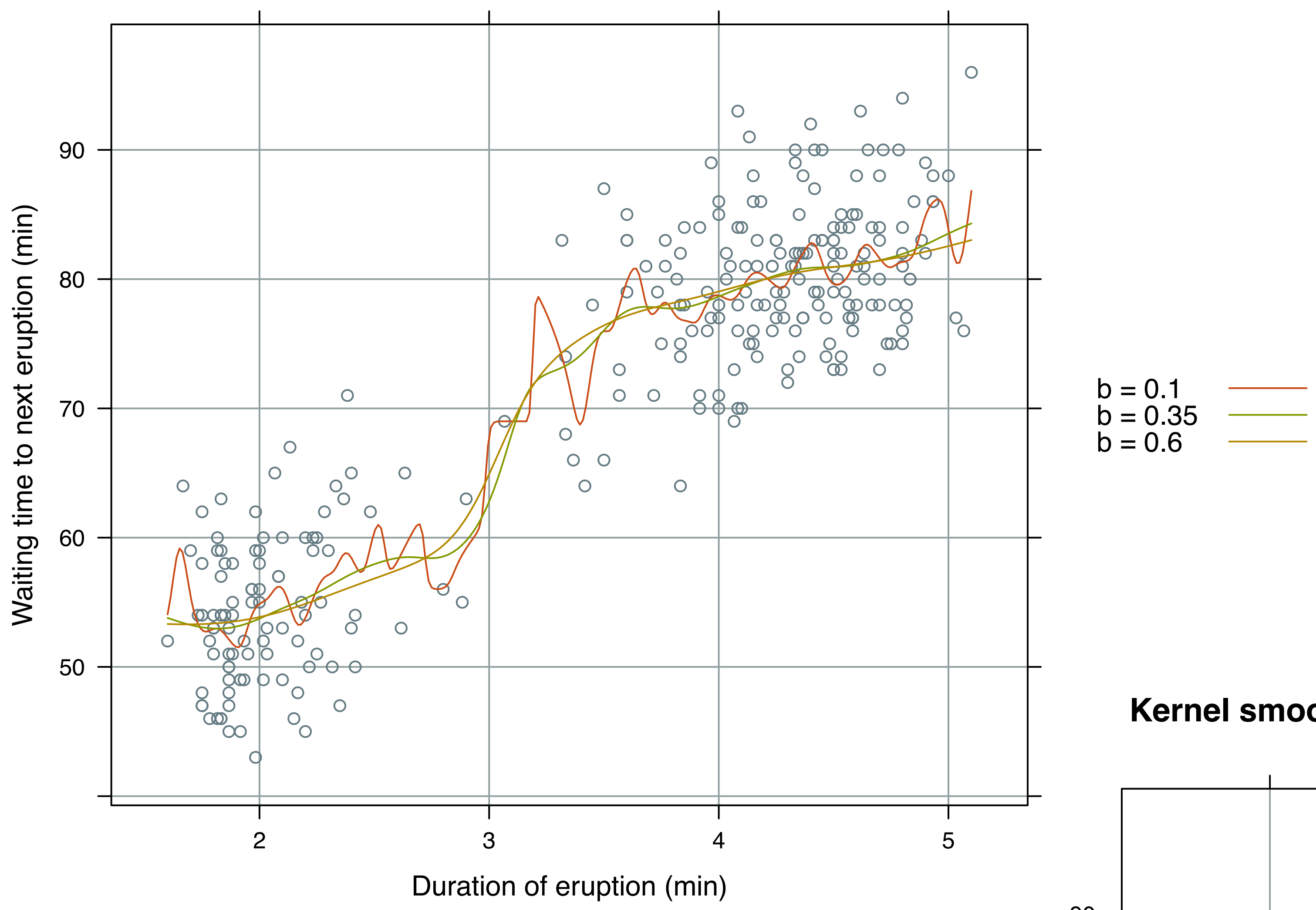
Kernel smoother, implemented by ksmooth, 'box' kernel



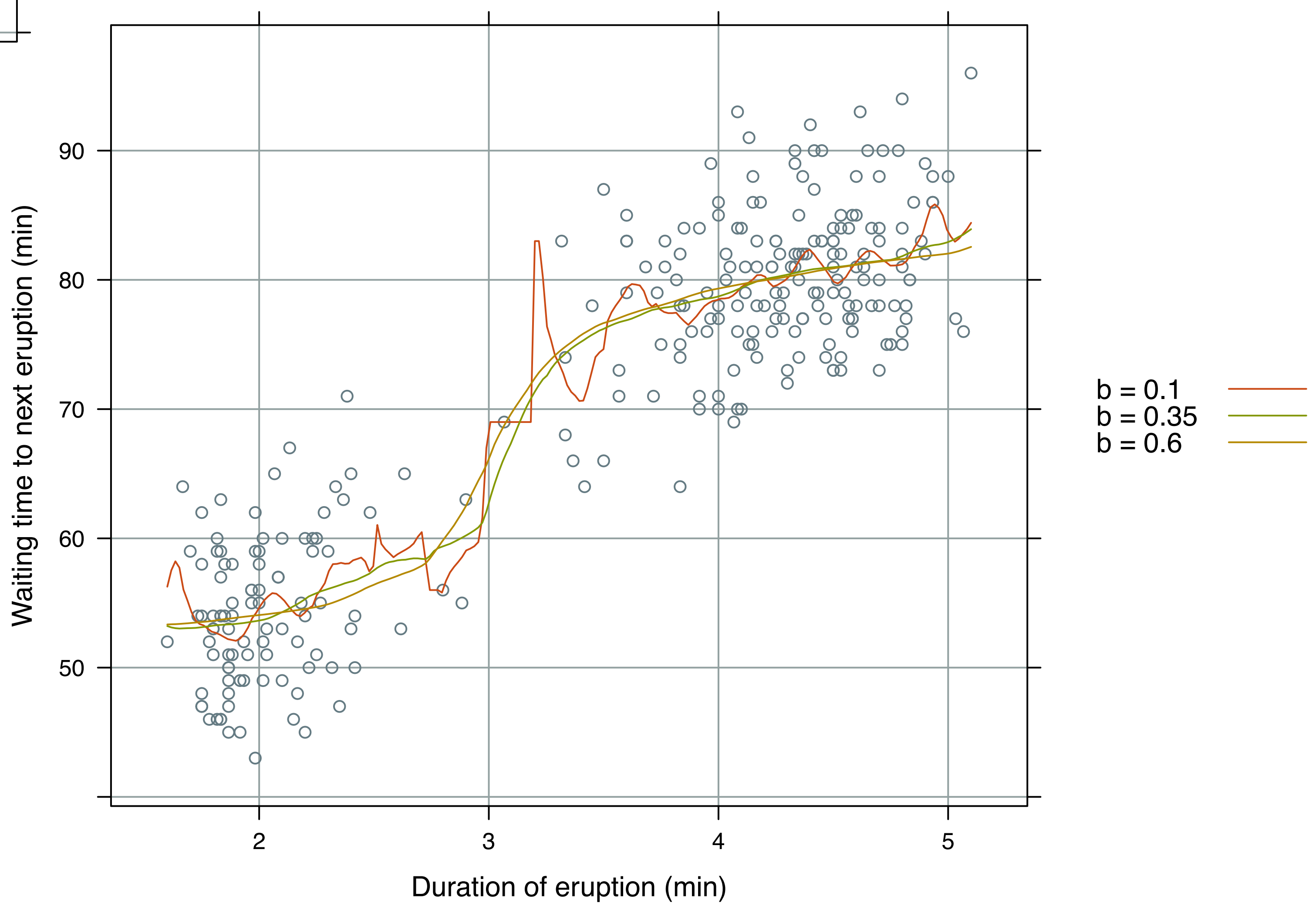
Kernel smoother, implemented by lpepa (lpridge), Epanechnikov kernel

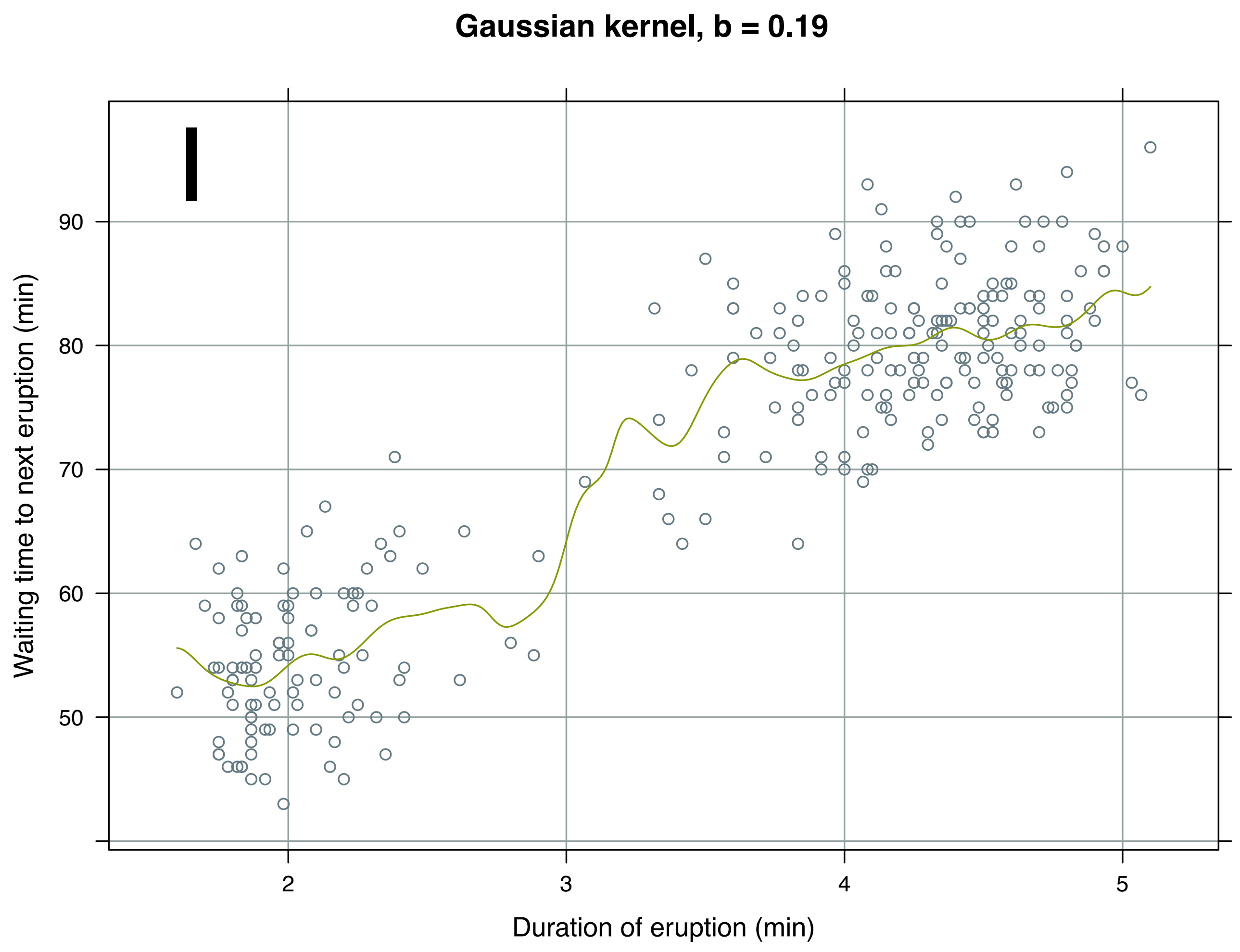


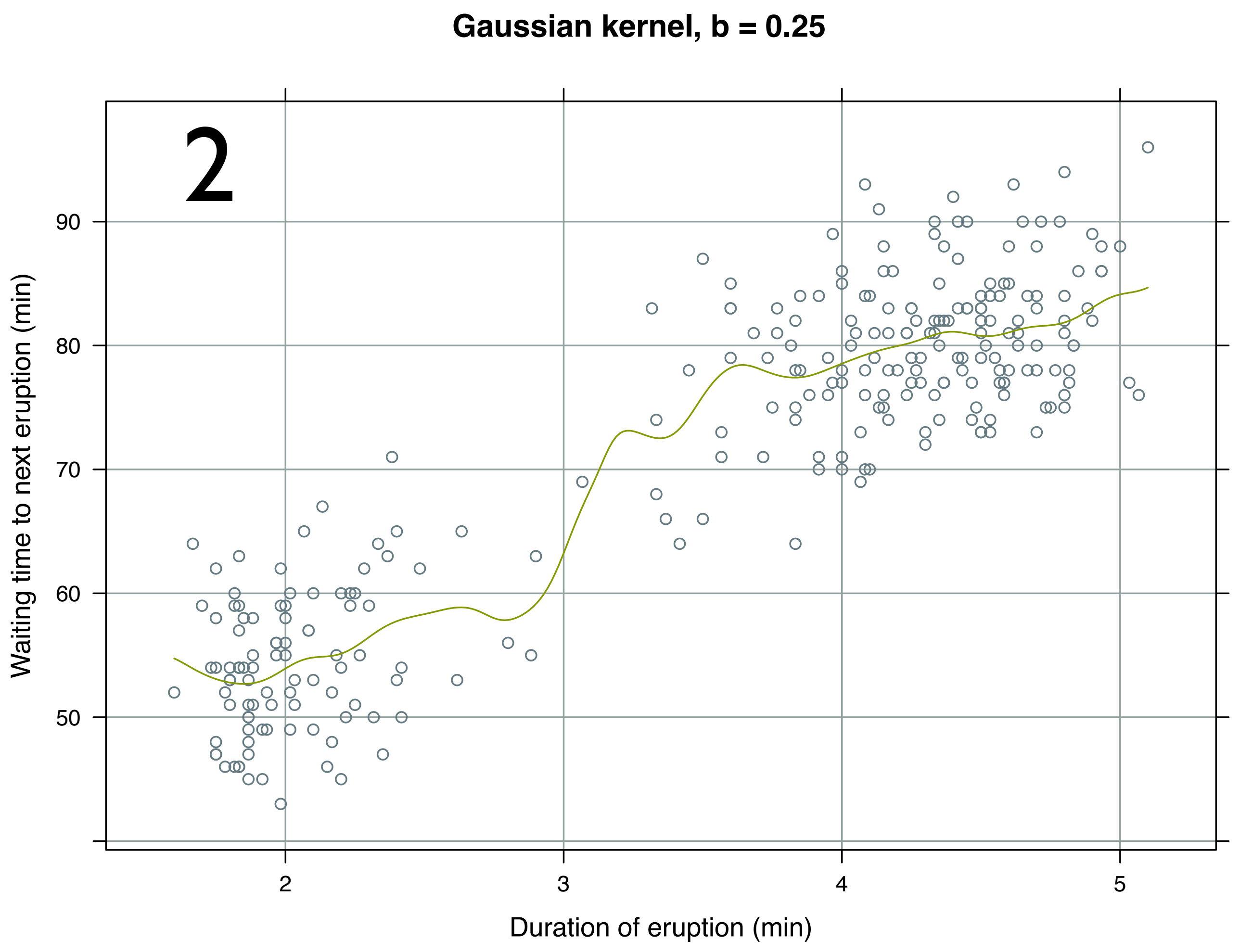
Kernel smoother, implemented by ksmooth, Gaussian kernel



Kernel smoother, implemented by lpepa (lpridge), Epanechnikov kernel

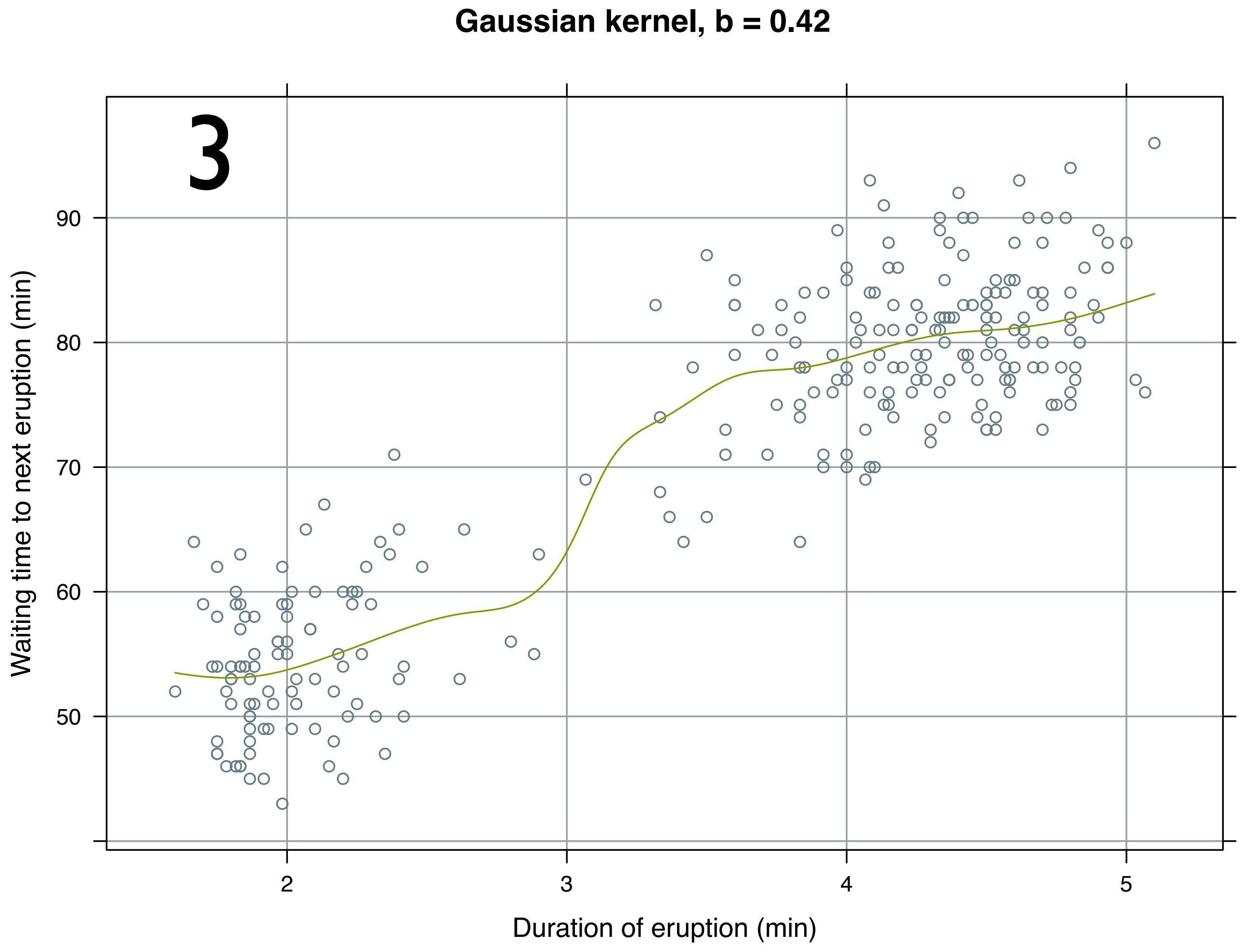


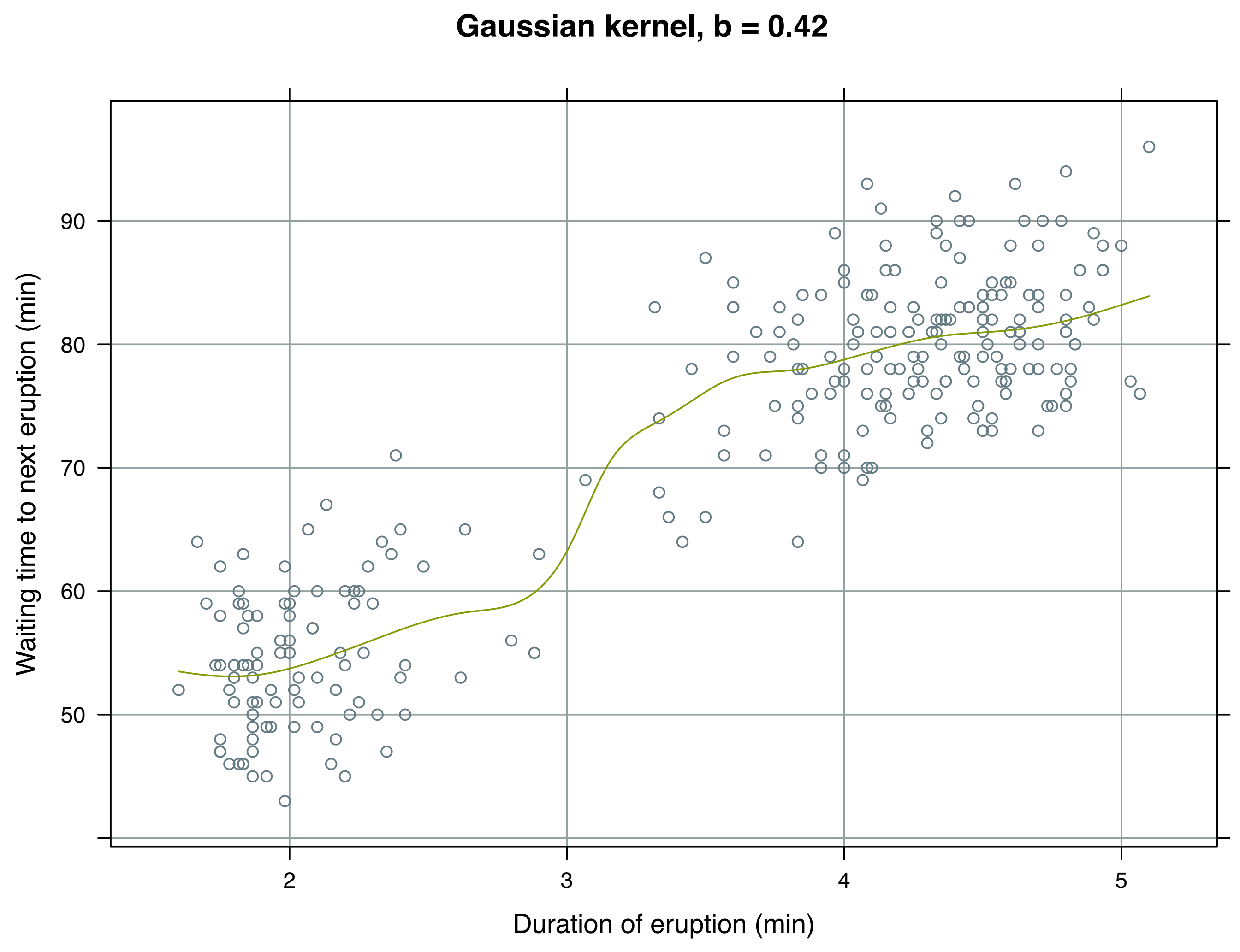


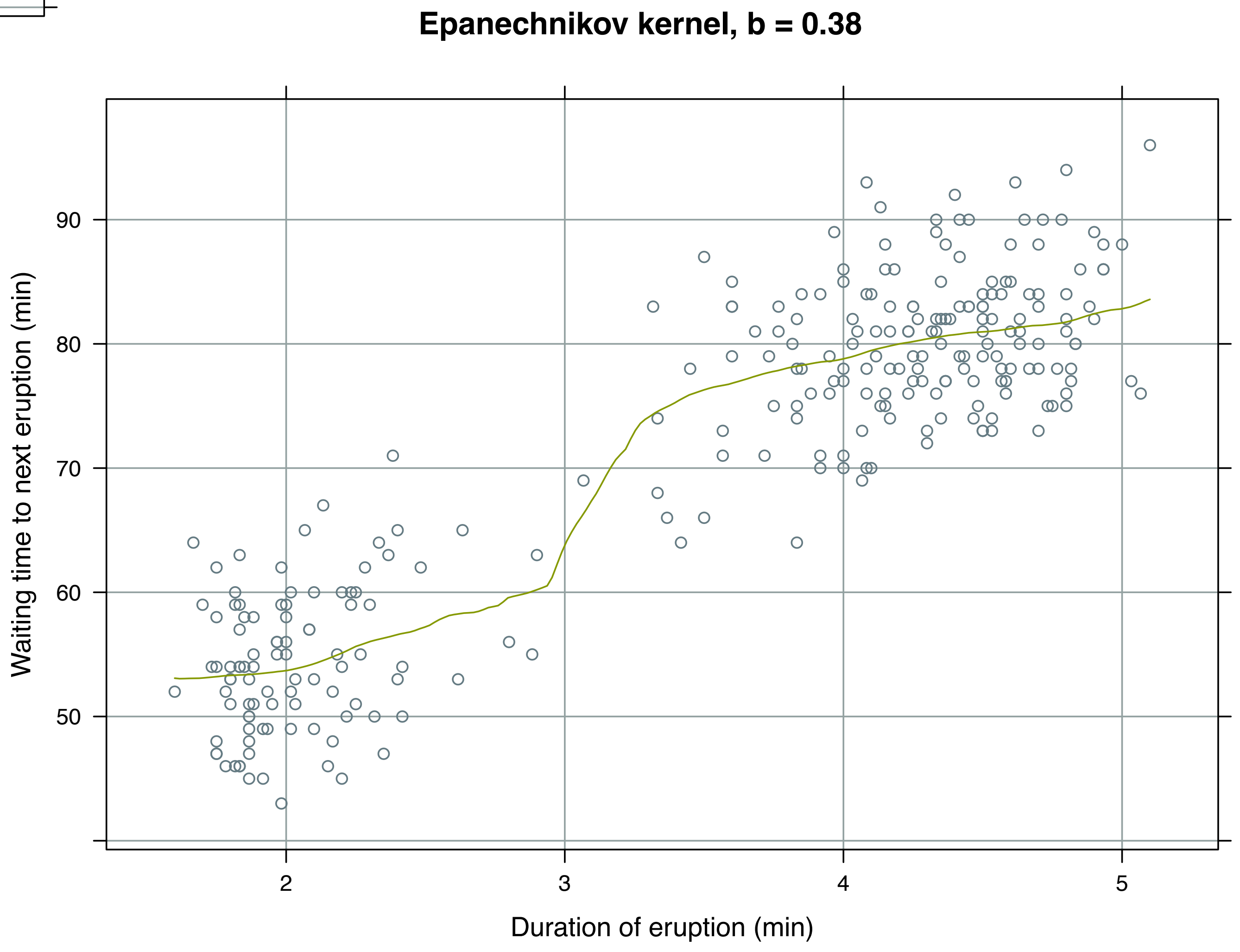


Bandwidth selection via ...

1. "direct plug-in methodology ... as described by Ruppert, Sheather and Wand (1995)", implemented by `dpill` in `KernSmooth` package
2. cross validation, implemented by `h.select` in `sm` package
3. cross validation, as reported in Faraway (2006)




Gaussian kernel, b = 0.42
Waiting time to next eruption (min)
Duration of eruption (min)
90
80
70
60
50
2
3
4
5


Epanechnikov kernel, b = 0.38
Waiting time to next eruption (min)
Duration of eruption (min)
90
80
70
60
50
2
3
4
5

Local constant →local polynomial

- Kernel smoother fits a local constant.
- Why not fit local lines or polynomials?
  - Built-in function `loess` does this
  - Packages `KernSmooth` and `locfit` also fit local polynomials
  - The weighting of observations via a kernel function persists in most of these local regression approaches
  - Robust local regression is also a nice add-on

Key innovation for *local polynomials*: Use the predicted y from *fitting a linear or quadratic function* to the (x,y)'s in the "neighborhood" of x_0. Often a kernel is used to weight the observations. Sometimes a robust procedure is used to fit the regression.

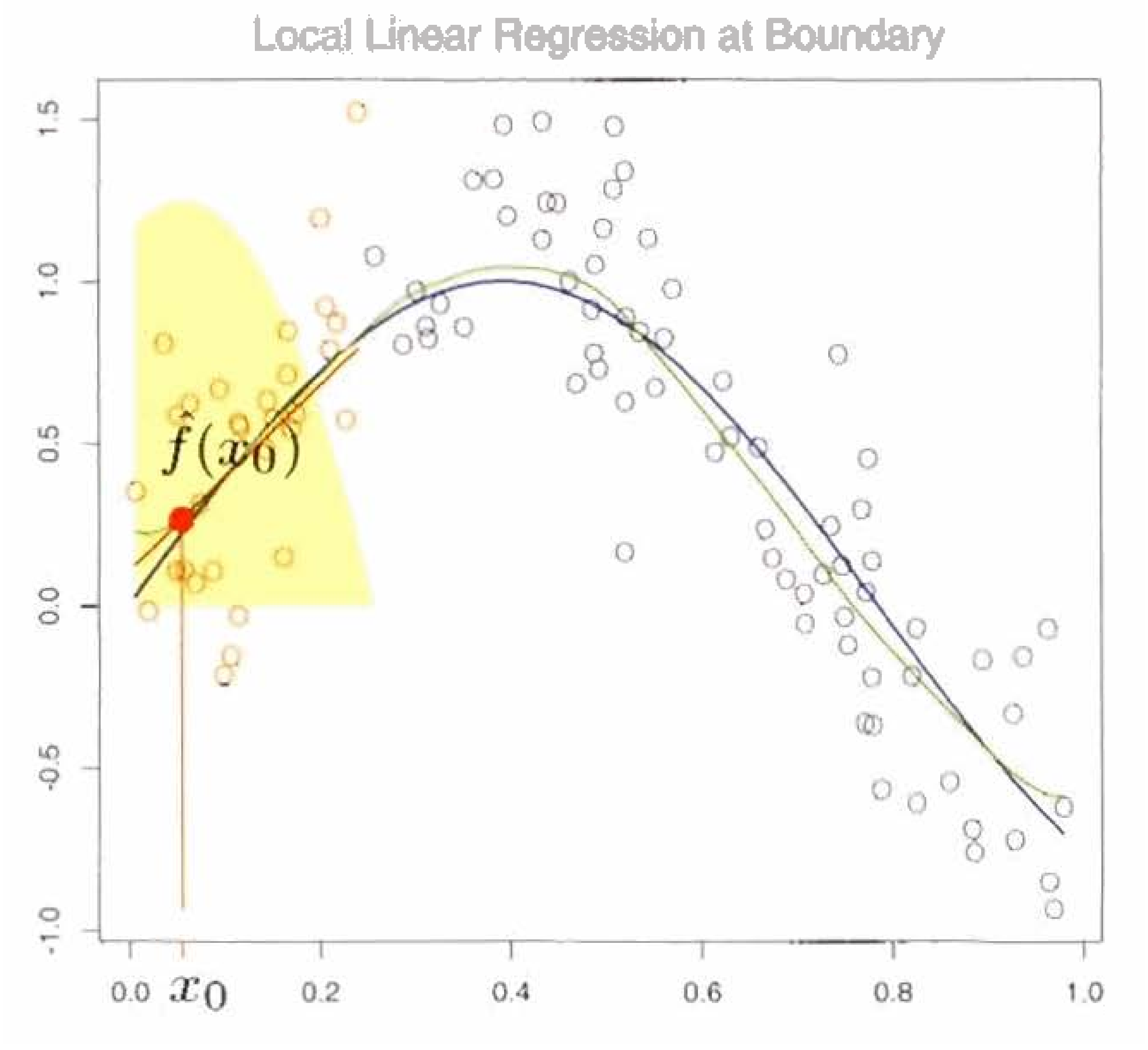


Figure from "The Elements of Statistical Learning", by by Hastie, Tibshirani, and Friedman (2001), Springer-Verlag.

Local polynomials

- Kernel smoothers can be quite biased near the edges
- Local linear fits work much better here
- Local linear fits are sometimes biased in the interior, in areas of curvature ("trimming the hills, filling the valleys") -- local quadratic fits can do better here
- Bias-variance trade-off
  - lower degree polynomials more biased, less variable
  - higher degree are less biased, more variable
- Proposal: default to local linear, use higher degrees with good reason.

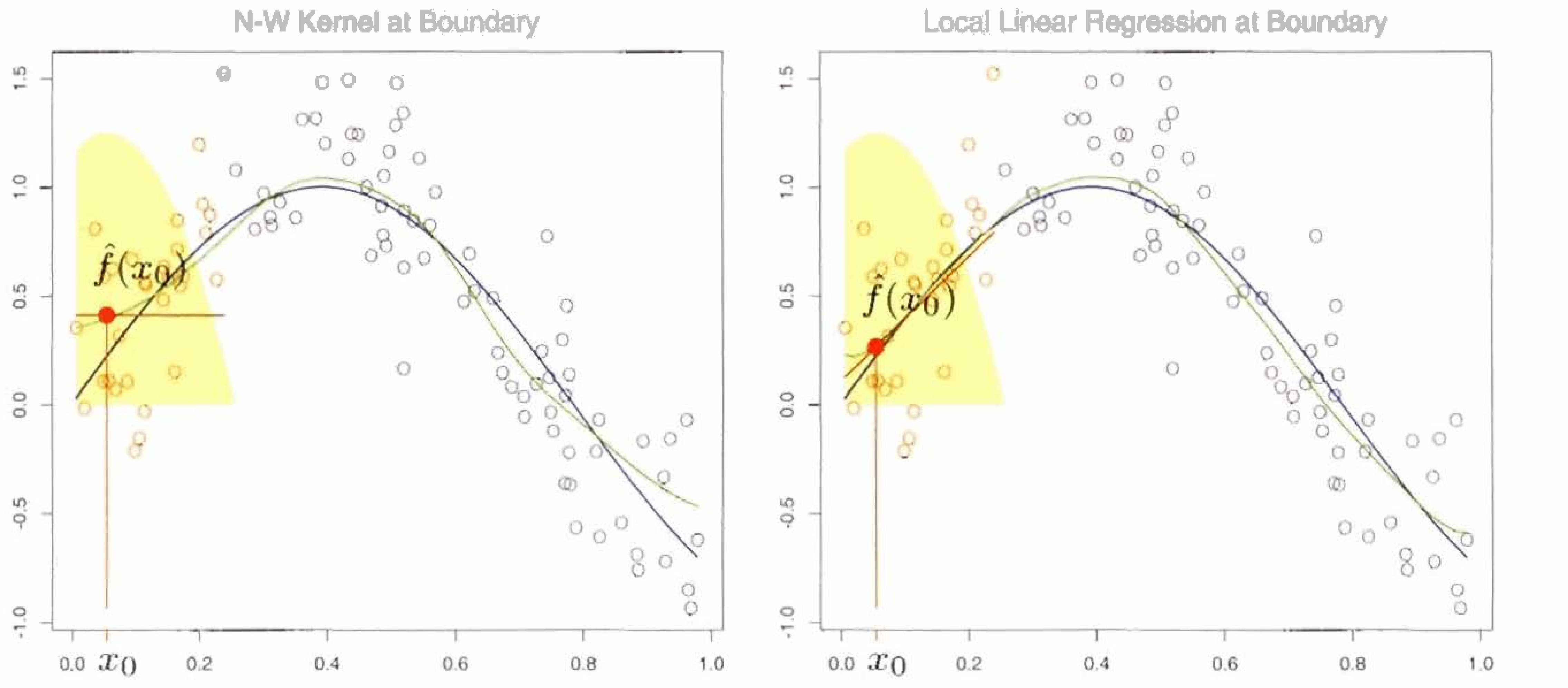


FIGURE 6.3. *The locally weighted average has bias problems at or near the boundaries of the domain. The true function is approximately linear here, but most of the observations in the neighborhood have a higher mean than the target point, so despite weighting, their mean will be biased upwards. By fitting a locally weighted linear regression (right panel), this bias is removed to first order*

Figure from "The Elements of Statistical Learning", by by Hastie, Tibshirani, and Friedman (2001), Springer-Verlag.

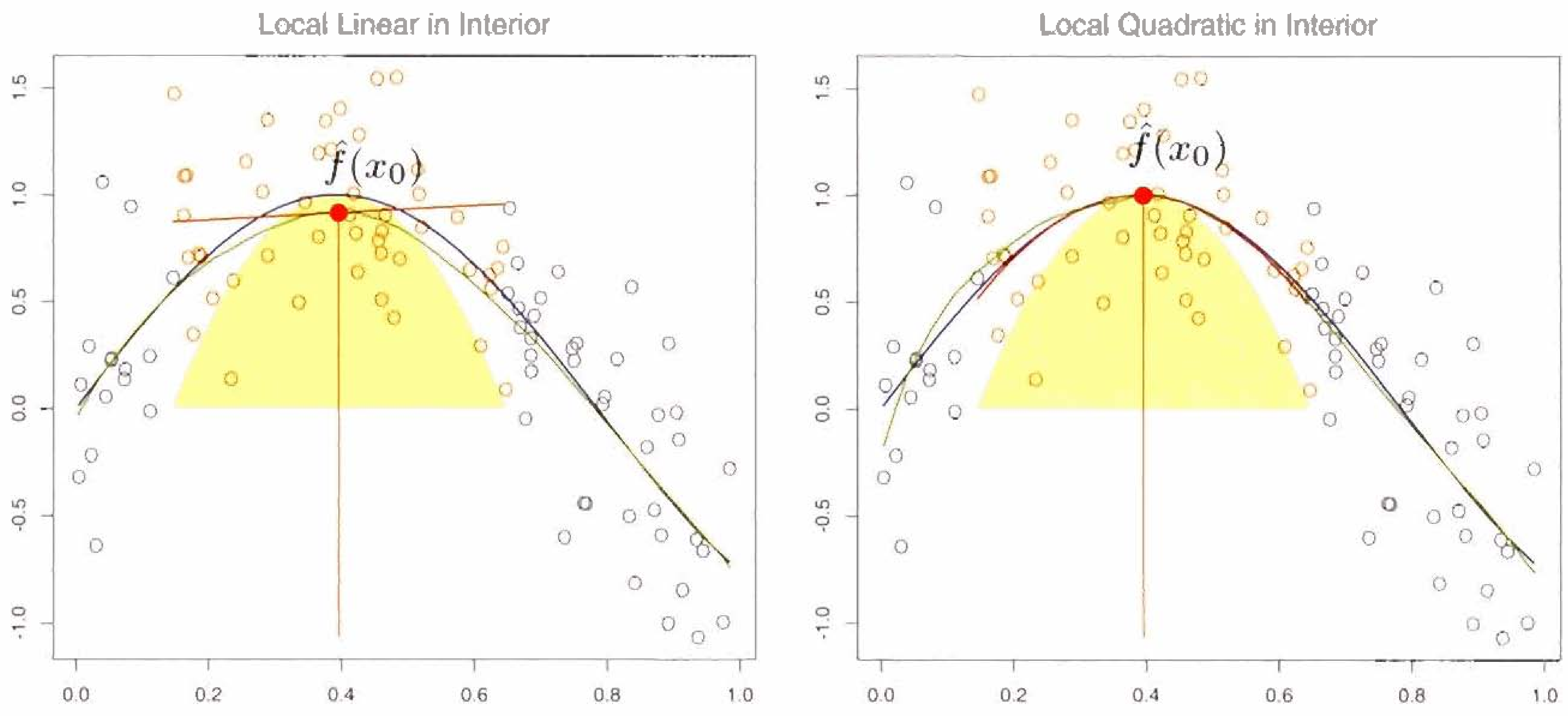


FIGURE 6.5. *Local linear fits exhibit bias in regions of curvature of the true function. Local quadratic fits tend to eliminate this bias.*

Figure from "The Elements of Statistical Learning", by by Hastie, Tibshirani, and Friedman (2001), Springer-Verlag.

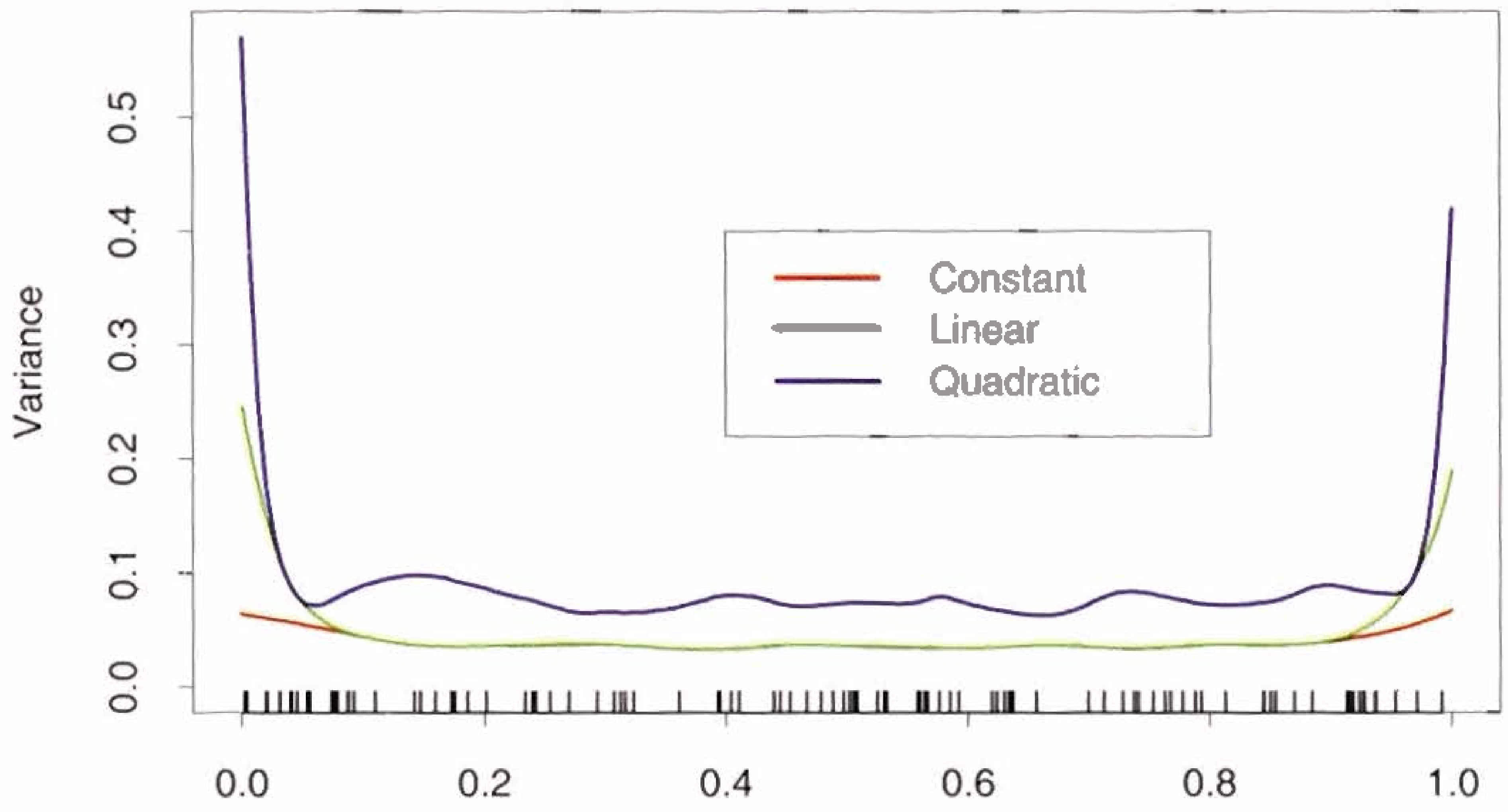


FIGURE 6.6. *The variances functions* $||l(x)||^2$ *for local constant, linear and quadratic regression, for a metric bandwidth* ($\lambda = 0.2$) *tri-cube kernel.*

Figure from "The Elements of Statistical Learning", by by Hastie, Tibshirani, and Friedman (2001), Springer-Verlag.

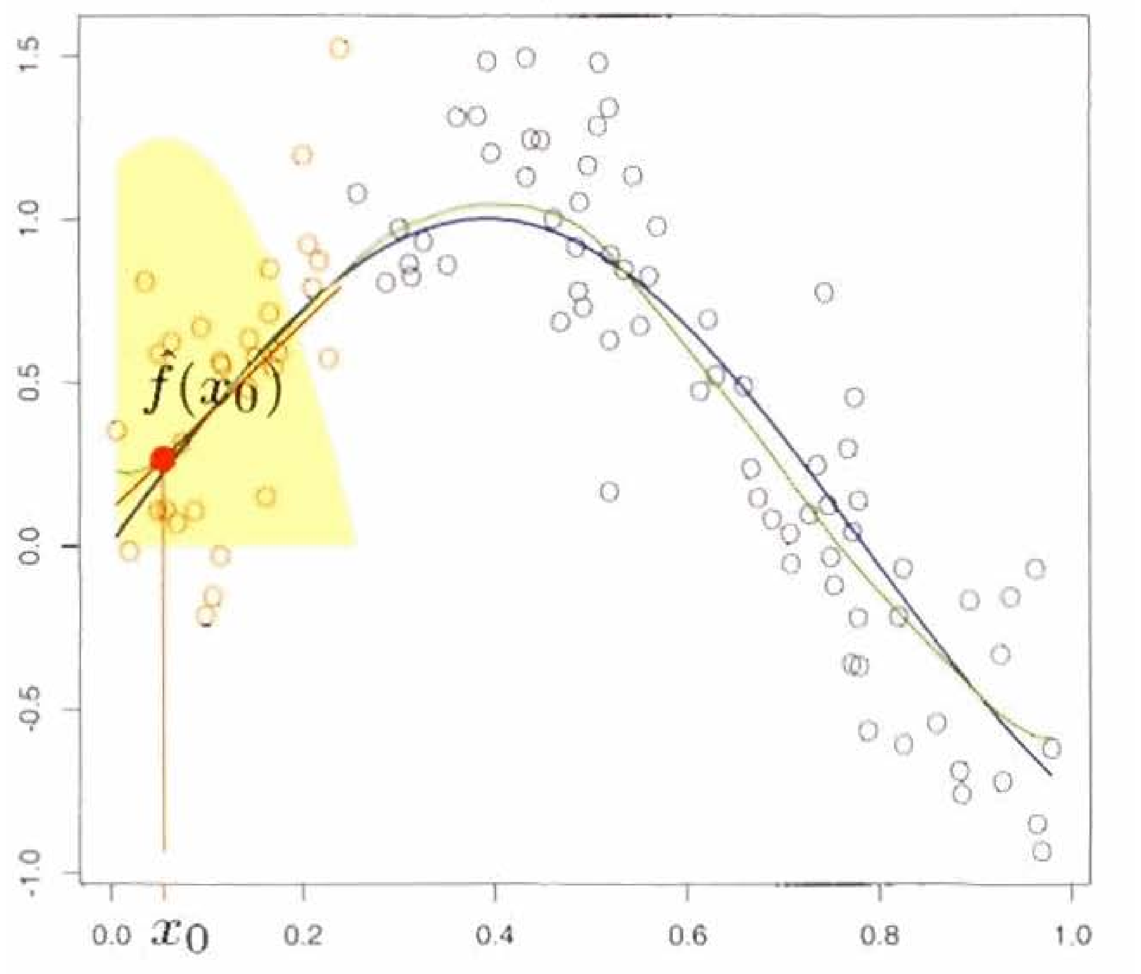
Local polynomial approach (using least squares)

$$\hat{\beta}(x) = \min^{-1} \sum_i K_b(x, x_i)(y_i - X\beta)^2$$

where the i-th row of X might be $\begin{pmatrix} 1 & x_i \end{pmatrix}$ or $\begin{pmatrix} 1 & x_i & x_i^2 \end{pmatrix}$

$$\hat{f}_b(x) = \tilde{x}\hat{\beta}(x)$$

where $\tilde{x}$ is an appropriately defined row vector, such as $\begin{pmatrix} 1 & x \end{pmatrix}$ or $\begin{pmatrix} 1 & x & x^2 \end{pmatrix}$



Local polynomial approach (using least squares)

Let $z = \begin{pmatrix} 1 & x \end{pmatrix}$ or $\begin{pmatrix} 1 & x & x^2 \end{pmatrix}$, i.e. the 'regression ready' predictor for the point x.

Let Z be the n by 2 or 3 design matrix, with ith row $\begin{pmatrix} 1 & x_i \end{pmatrix}$ or $\begin{pmatrix} 1 & x_i & x_i^2 \end{pmatrix}$.

Let $W(x)$ be the n by n diagonal matrix with ith diagonal element $K_b(x, x_i)$.

Then the local polynomial fit at x can be written as:

$$\hat{f}_b(x) = z(Z^T W(x) Z)^{-1} Z^T W(x) y = \sum_i l_i(x) y_i$$

weighted least squares regression

linear in the observed response y_i

The weights $l_i(x)$ combine the weighting from the kernel function with that from the least squares fitting approach. Sometimes called the *equivalent kernel*.

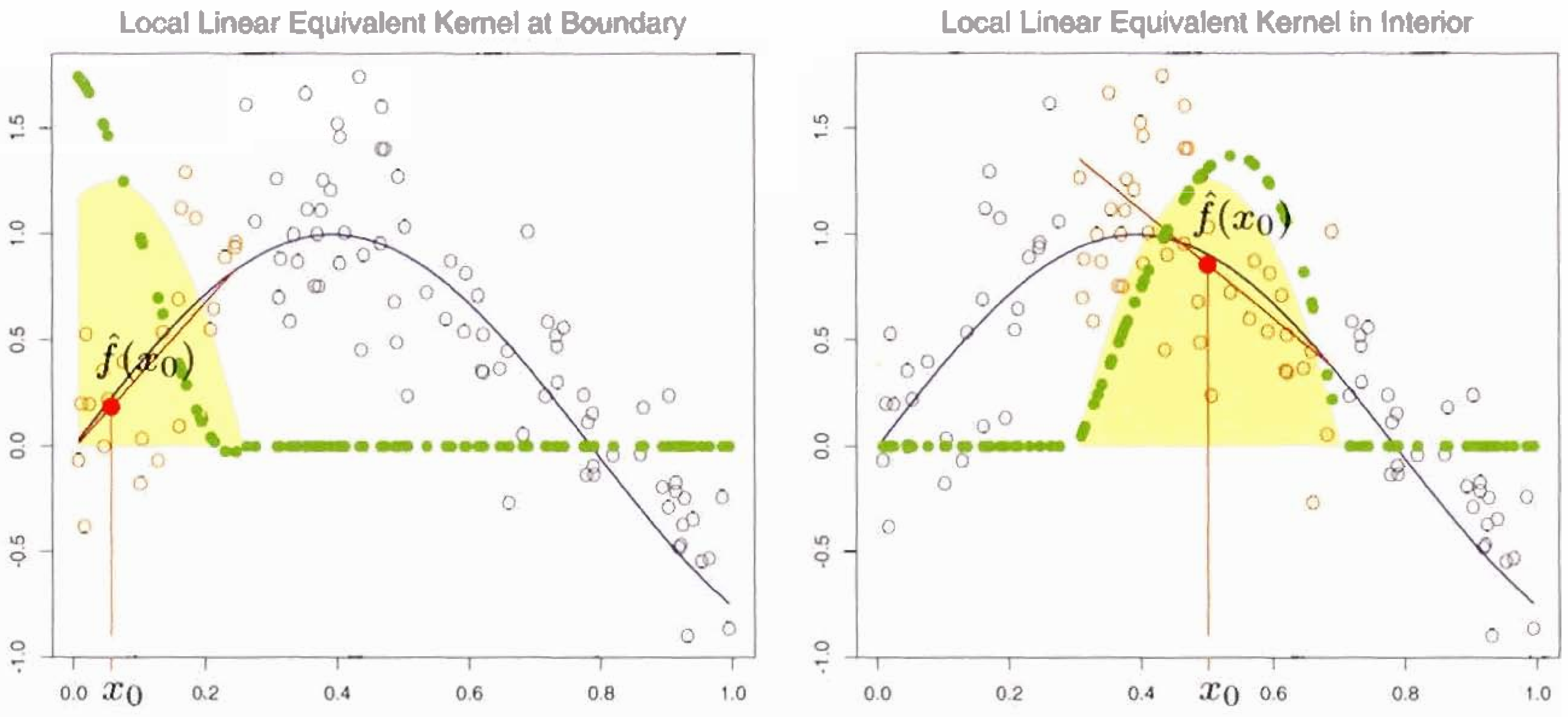


FIGURE 6.4. *The green points show the equivalent kernel $l_i(x_0)$ for local regression. These are the weights in $\hat{f}(x_0) = \sum_{i=1}^{N} l_i(x_0)y_i$, plotted against their corresponding x_i. For display purposes, these have been rescaled, since in fact they sum to 1. Since the orange shaded region is the (rescaled) equivalent kernel for the Nadaraya–Watson local average, we see how local regression automatically modifies the weighting kernel to correct for biases due to asymmetry in the smoothing window.*

Figure from "The Elements of Statistical Learning", by by Hastie, Tibshirani, and Friedman (2001), Springer-Verlag.

Local polynomial approach (using least squares)

$$\hat{f}_b(x) = z(Z^T W(x) Z)^{-1} Z^T W(x) y = \sum_i l_i(x) y_i$$

$$\hat{f} = S_b y$$

"smoother matrix"

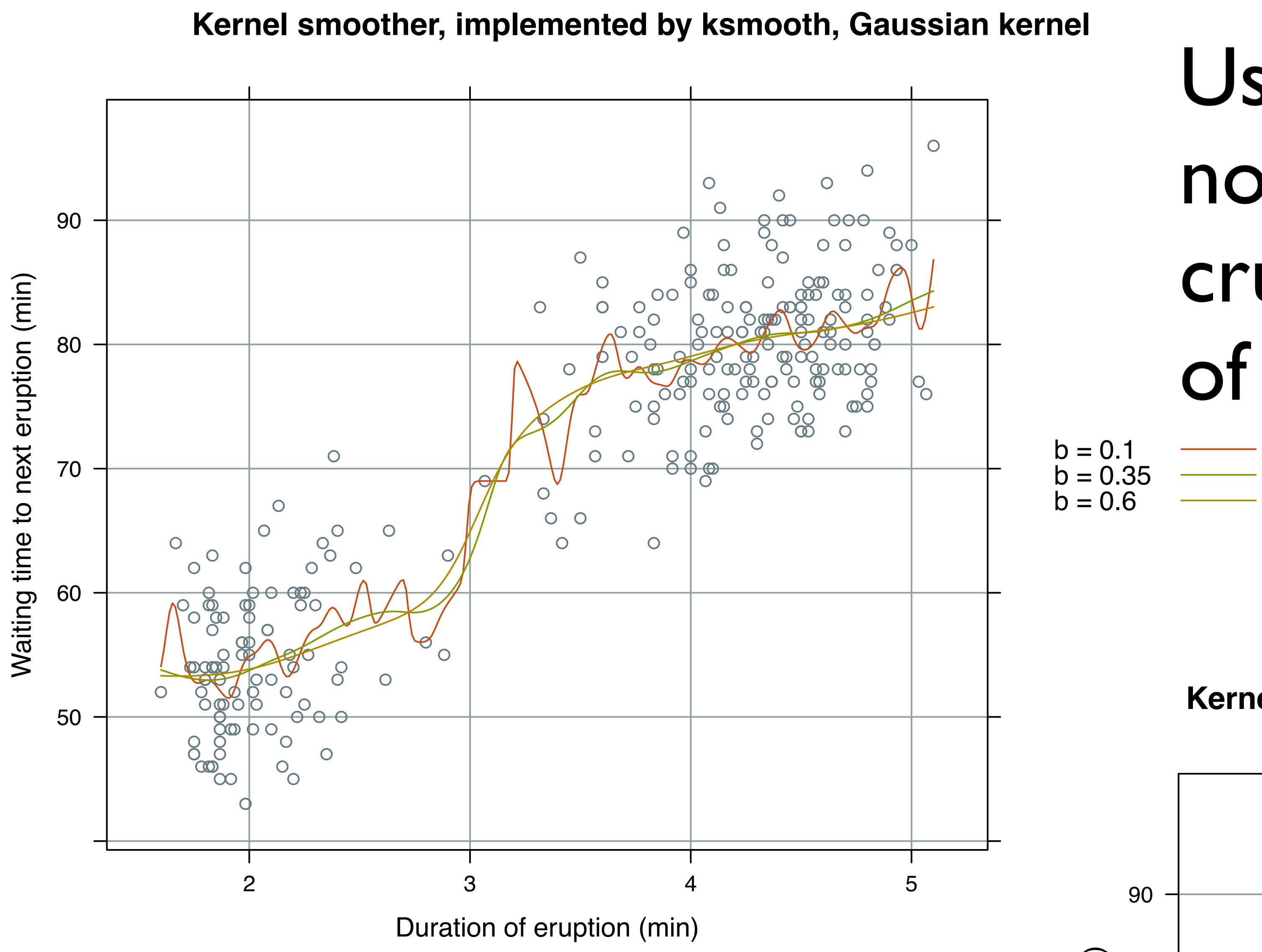
fitted values in a plain vanilla linear moc

$$\hat{y} = \underbrace{X(X^T X)^{-1} X^T}_{\text{the "hat matrix"}} y$$

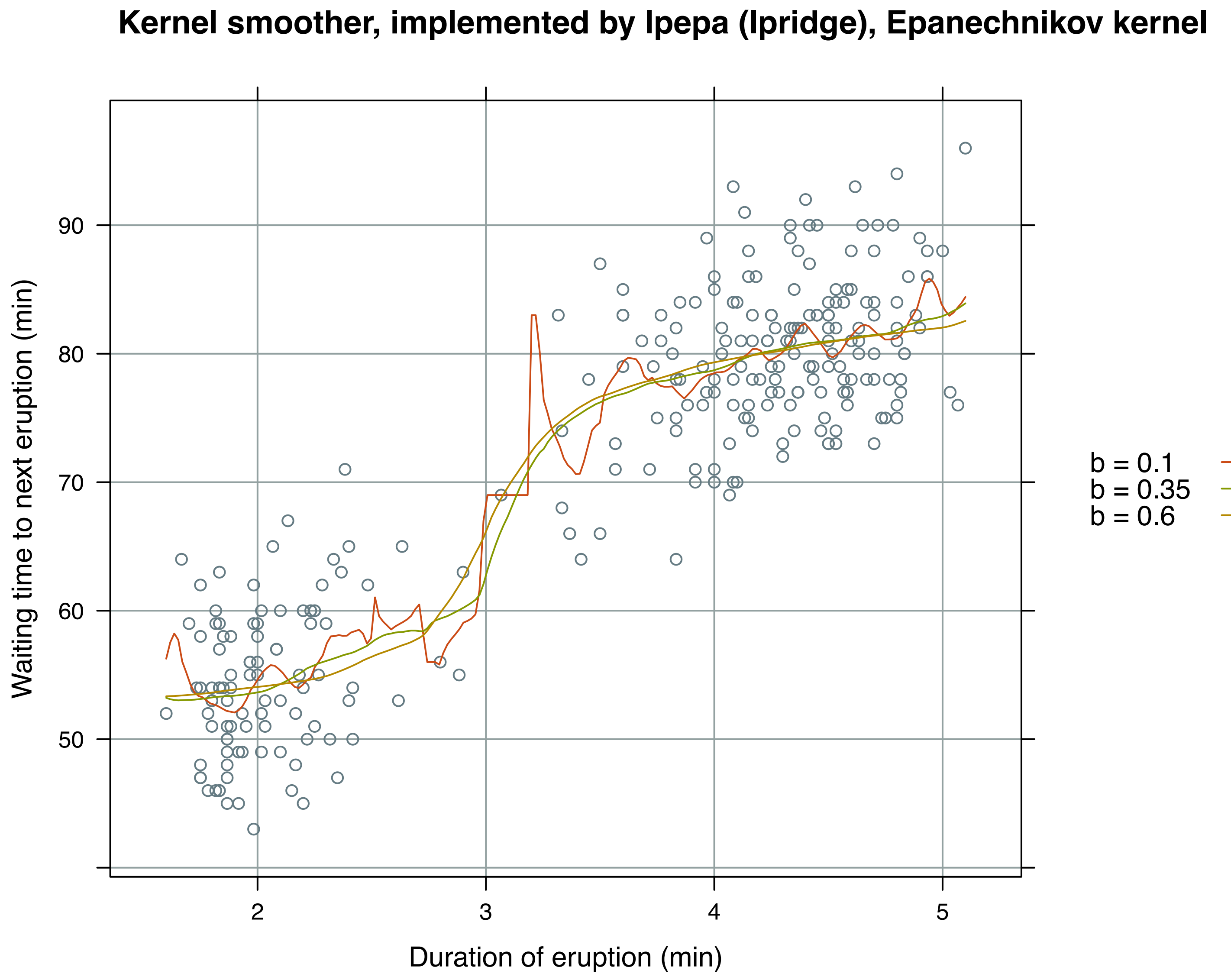
The trace of the smoother matrix is used as the 'effective degrees of freedom' for the smoother. Recall that the trace of the analogous 'hat' matrix in linear regression is p = the number of parameters. (A similar smoother matrix arises with splines as well.)
This 'pseudo degrees of freedom' can be used to calibrate different smoothers or as an alternative way to specify the amount of smoothing.

`loess` (and `lowess`)

- Local, weighted polynomial regression
- Require specification of a span = proportion of points that influence fit at any given point
- Use tricubic weighting
- `lowess`: an older function that provides a local linear (or constant) fit, superceded by `loess`
- `loess`: local linear or quadratic fit, can work with multiple predictors, can use some predictors globally, local fits can be least squares or something more robust,



Usually, choice of kernel is not terribly interesting / crucial from the perspective of exploratory figures.



In contrast, the choice of bandwidth -- or whatever tuning parameter controls the amount of smoothing -- **is** quite critical.

approach	tuning parameter(s) that controls smoothness	some leads on actual functions or packages
kernel smoother	bandwidth sd (or some other measure of spread) of the kernel	ksmooth() KernSmooth sm lpridge
local polynomial regression	effective degrees of freedom trace of the smoother matrix span degree of polynomial	loess() locfit package

$\hat{f}_b(x)$ Nonparametric regression estimate of E(Y|X=x) for smoothing parameter value b

How to choose b?

We'd like to set b to minimize prediction error:

$$PE = E(Y - \hat{f}_b(X))^2$$

In practice, it is tempting to estimate PE with this simple-to-compute empirical quantity:

$$\frac{1}{n}\sum_i (y_i - \hat{f}_b(x_i))^2$$

However, this is too naive.

Average squared residual error will be too "optimistic".

Average squared residual error will tend to underestimate prediction error.

Circular logic: use the observed data to build the predictor THEN use the same data to evaluate the performance of the predictor.

If we pick the amount of smoothing to minimize avg residual error, we will "connect the dots" (i.e. do *no* smoothing) and get a value of zero, which is just silly.

Oops!

What we want:

$$E(Y - \hat{f}_b(X))^2$$

What we might settle for:

$$\frac{1}{n}\sum_i (y_i - \hat{f}_b(x_i))^2$$

Fundamental issue:

We need to use some data to build the predictor and other data to characterize its performance.

"Training vs. test"

Cross validation is an attractive approach.

What we want:

$$E(Y - \hat{f}_b(X))^2$$

What we almost settled for:

$$\frac{1}{n}\sum_i (y_i - \hat{f}_b(x_i))^2$$

Cross validation estimate of prediction error:

$$\frac{1}{n}\sum_i (y_i - \hat{f}_b^{-k(i)}(x_i))^2$$

prediction for obs *i*, based on a regression estimate that excluded the *k(i)*th part of the data

Cross validation estimate of prediction error:

$$\frac{1}{n}\sum_i (y_i - \hat{f}_b^{-k(i)}(x_i))^2$$

prediction for obs *i*, based on a regression estimate that excluded the *k(i)*th part of the data

What's the deal with the "*k(i)*th" part of the data?

If *k(i)* = *i*, we get "leave one out" cross validation.

If *k(i)* = a sub-sample of the dataset, of size ~ *n*/*K*, we get "K-fold" cross validation, e.g. "10-fold cross validation."

Cross validation estimate of prediction error associated with smoothing parameter value b:

$$CV(b) = \frac{1}{n}\sum_i (y_i - \hat{f}_b^{-k(i)}(x_i))^2$$

Compute *CV(b)* for many values of *b*.

You should probably plot this.

The *b* value that minimizes *CV(b)*, call it b_{CV}, should receive serious consideration as the "best" value of *b*.

216 NONPARAMETRIC REGRESSION

```
> sm.regression(faithful$duration,faithful$waiting,h=hm,
  xlab="duration",ylab="waiting")
```

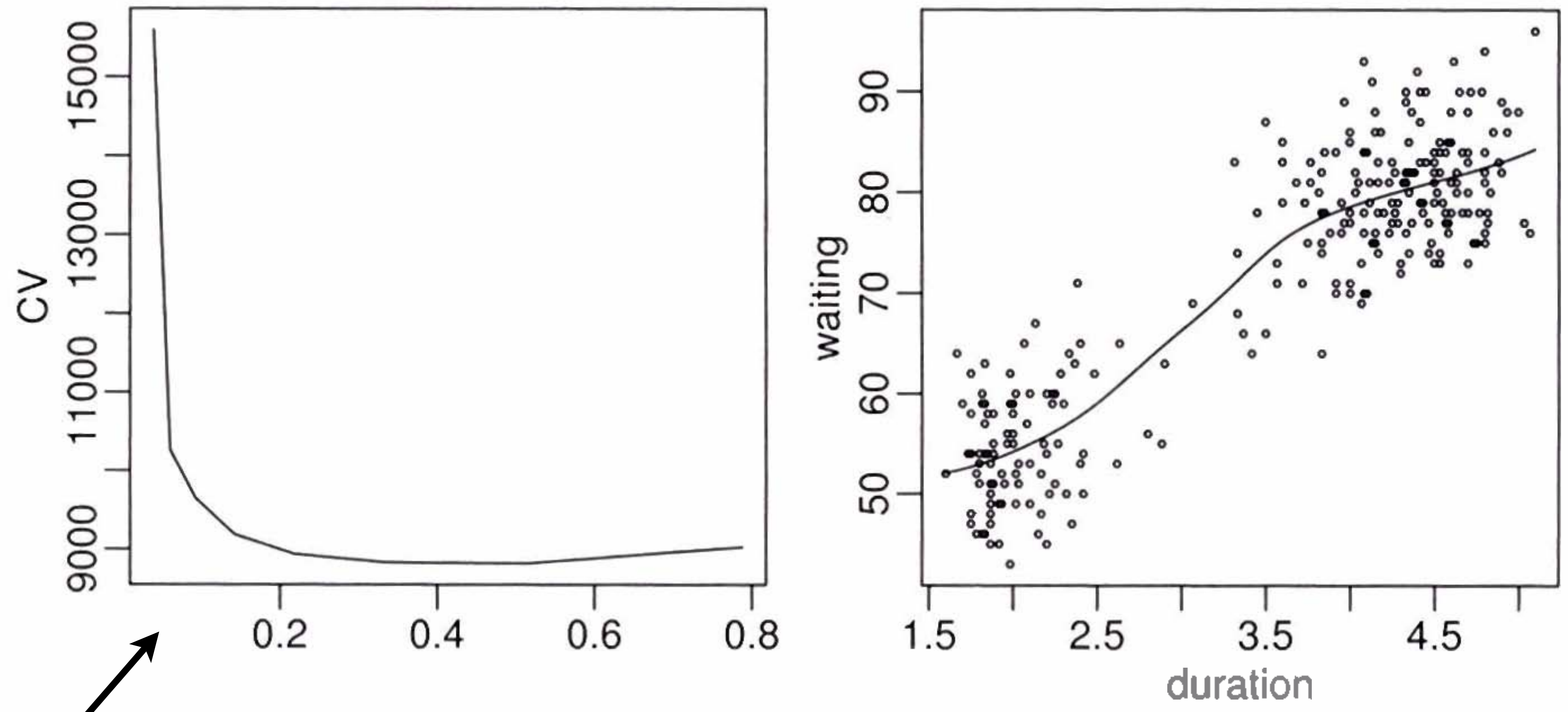
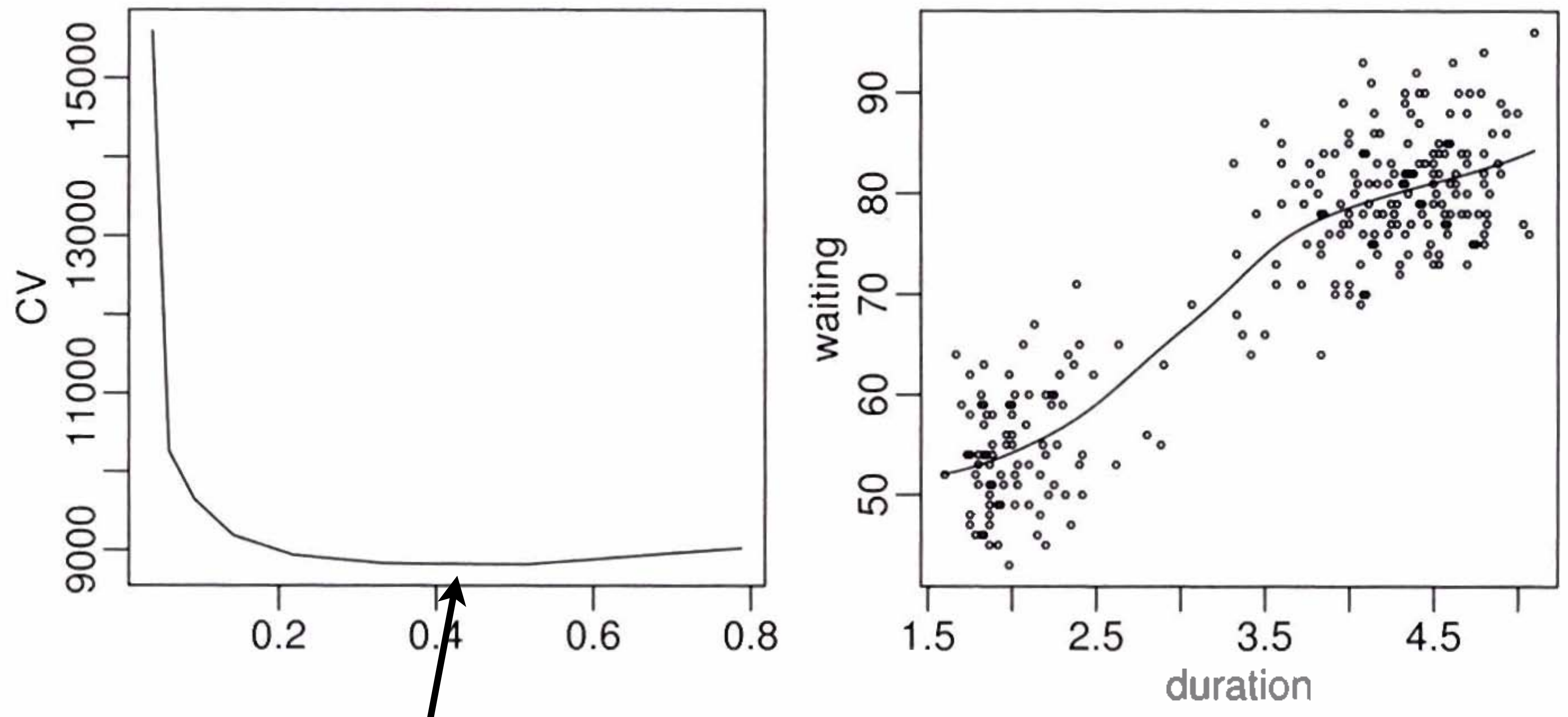


Figure from “Extending the Linear Model With R”. Faraway, Julian (2006) Chapman & Hall/CRC Press.

This is our goal -- the error curve that relates the CV criterion to the tuning parameter b.

216 NONPARAMETRIC REGRESSION

```
> sm.regression(faithful$duration,faithful$waiting,h=hm,
  xlab="duration",ylab="waiting")
```

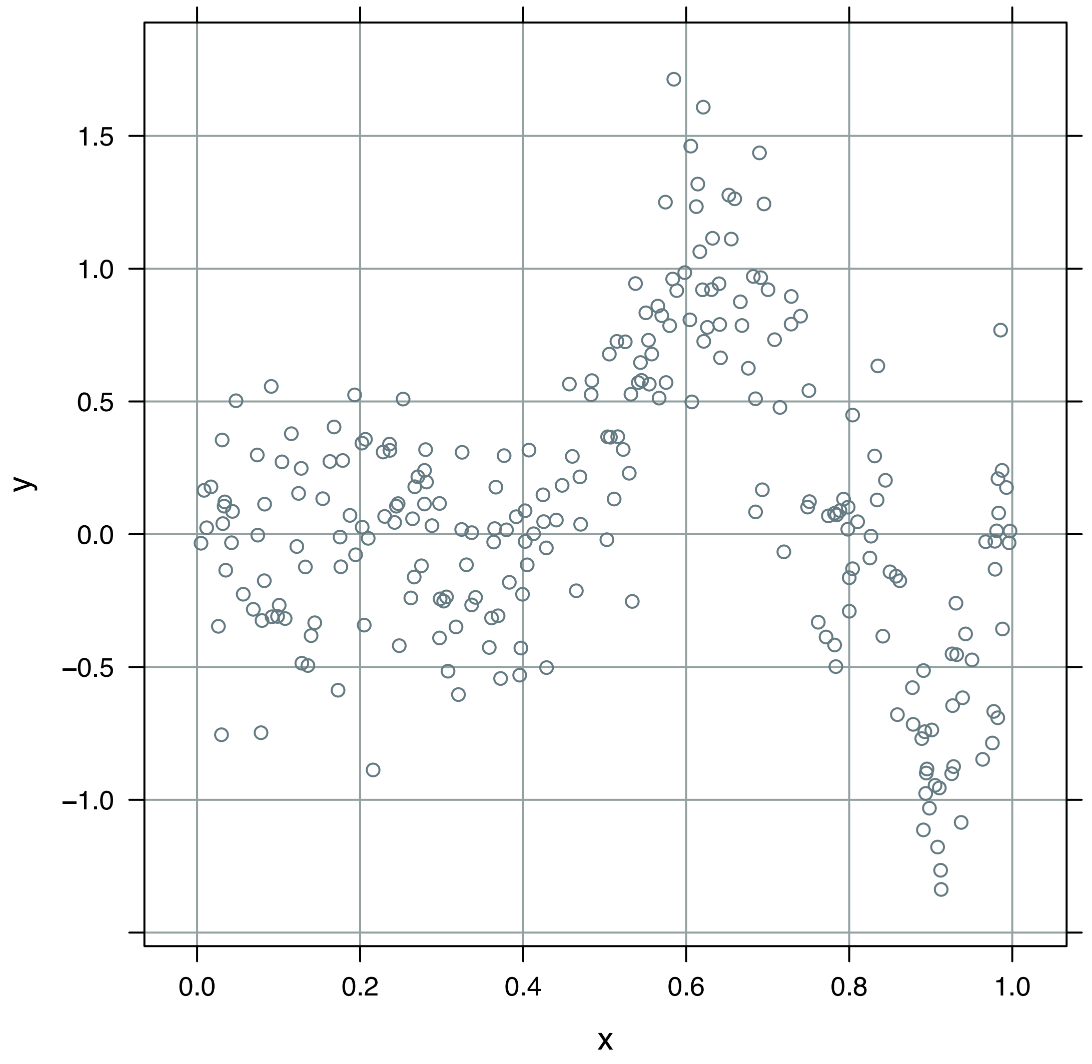


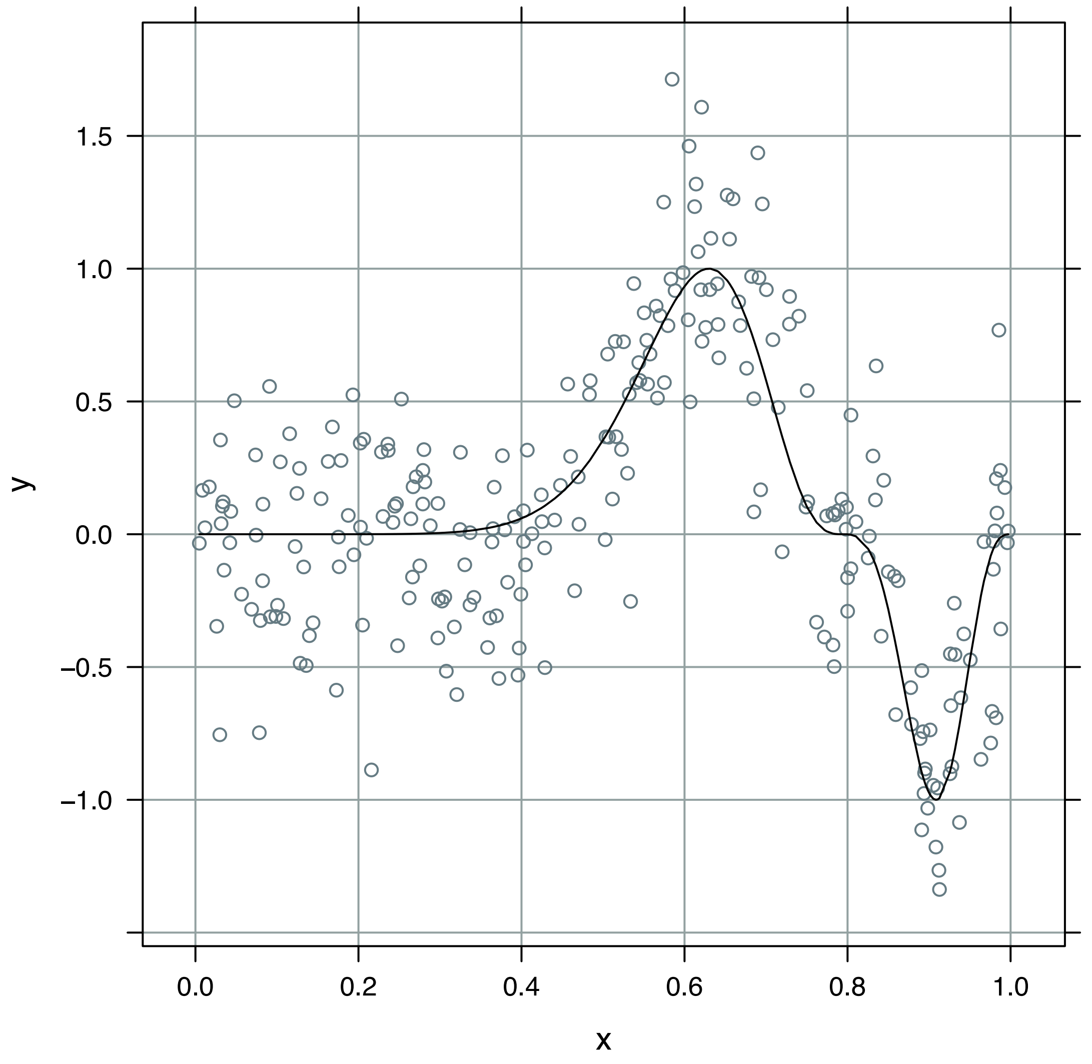
$b_{CV} = 0.42$

But also note how flat that curve is ...
many values of b perform about the same.

Figure from "Extending the Linear Model With R". Faraway, Julian (2006) Chapman & Hall/CRC Press.

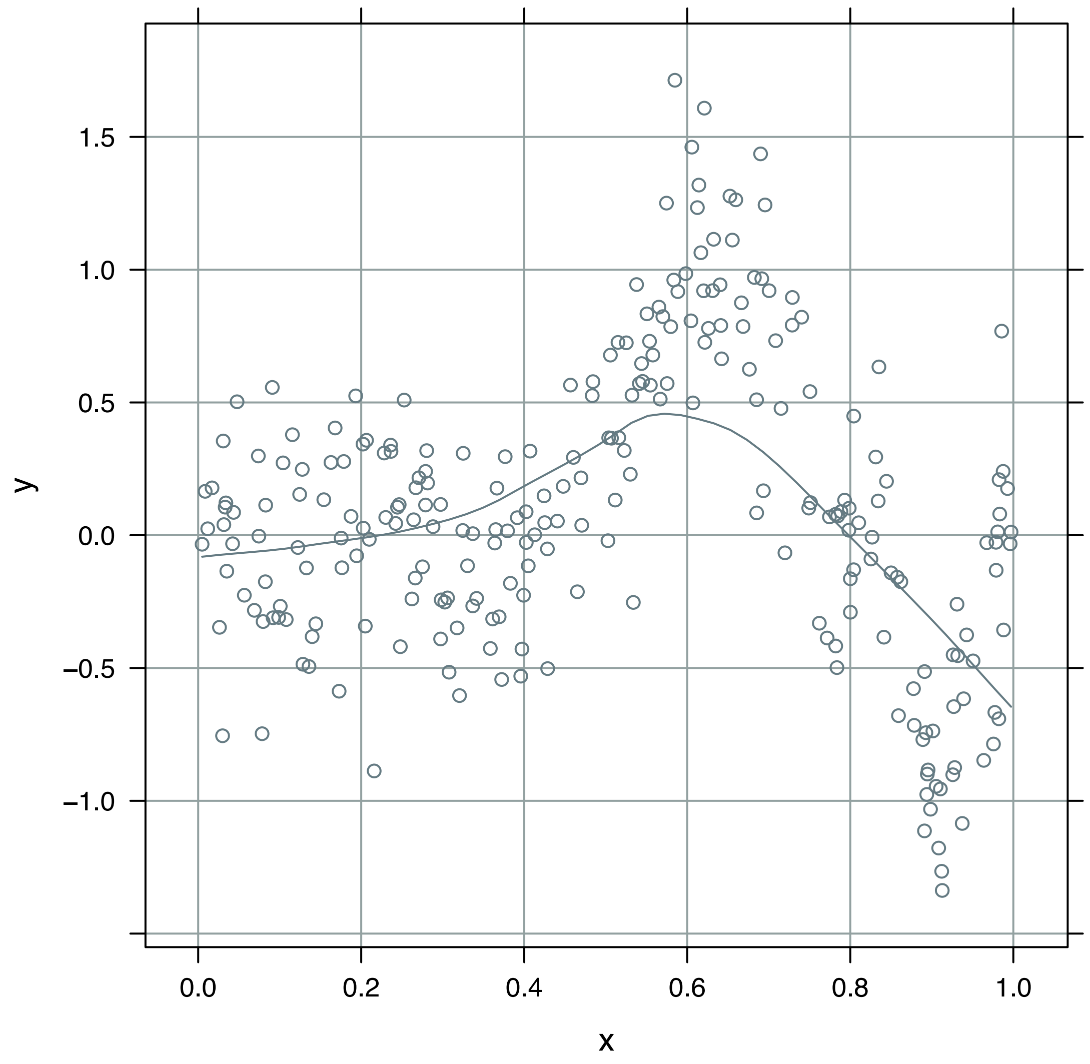
Synthetic dataset from Faraway (2006).



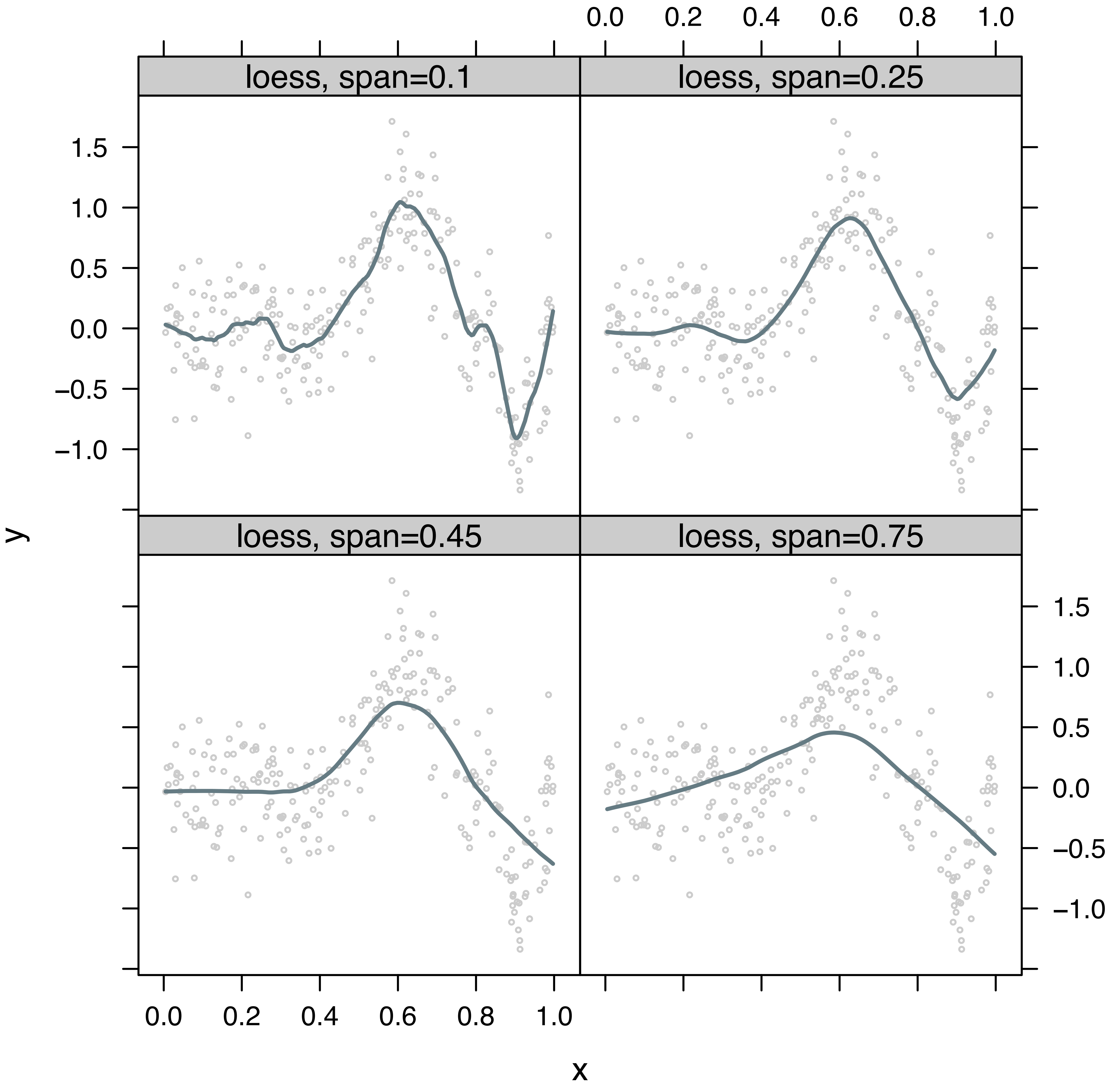


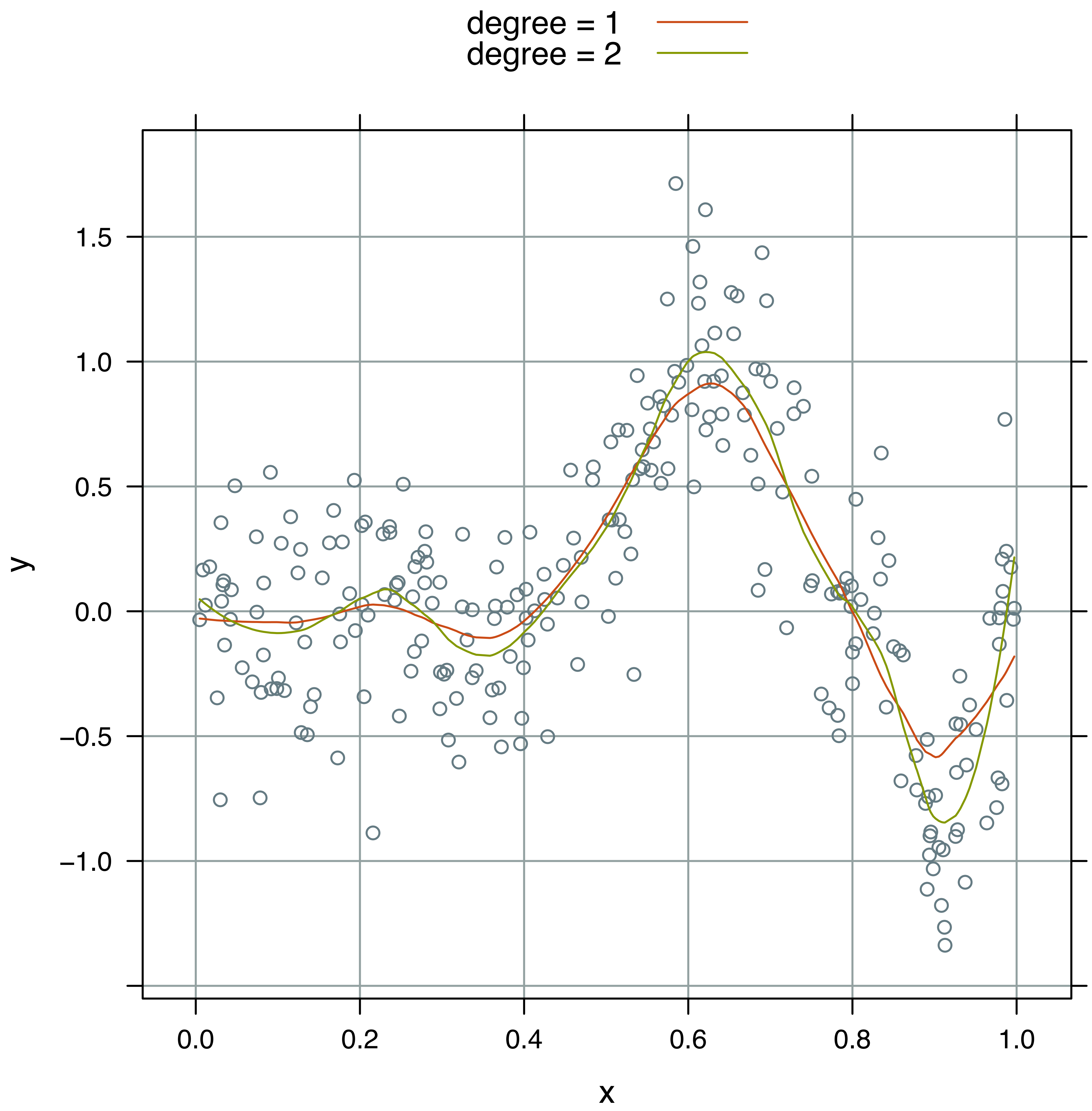
black line is the *data-generating truth*

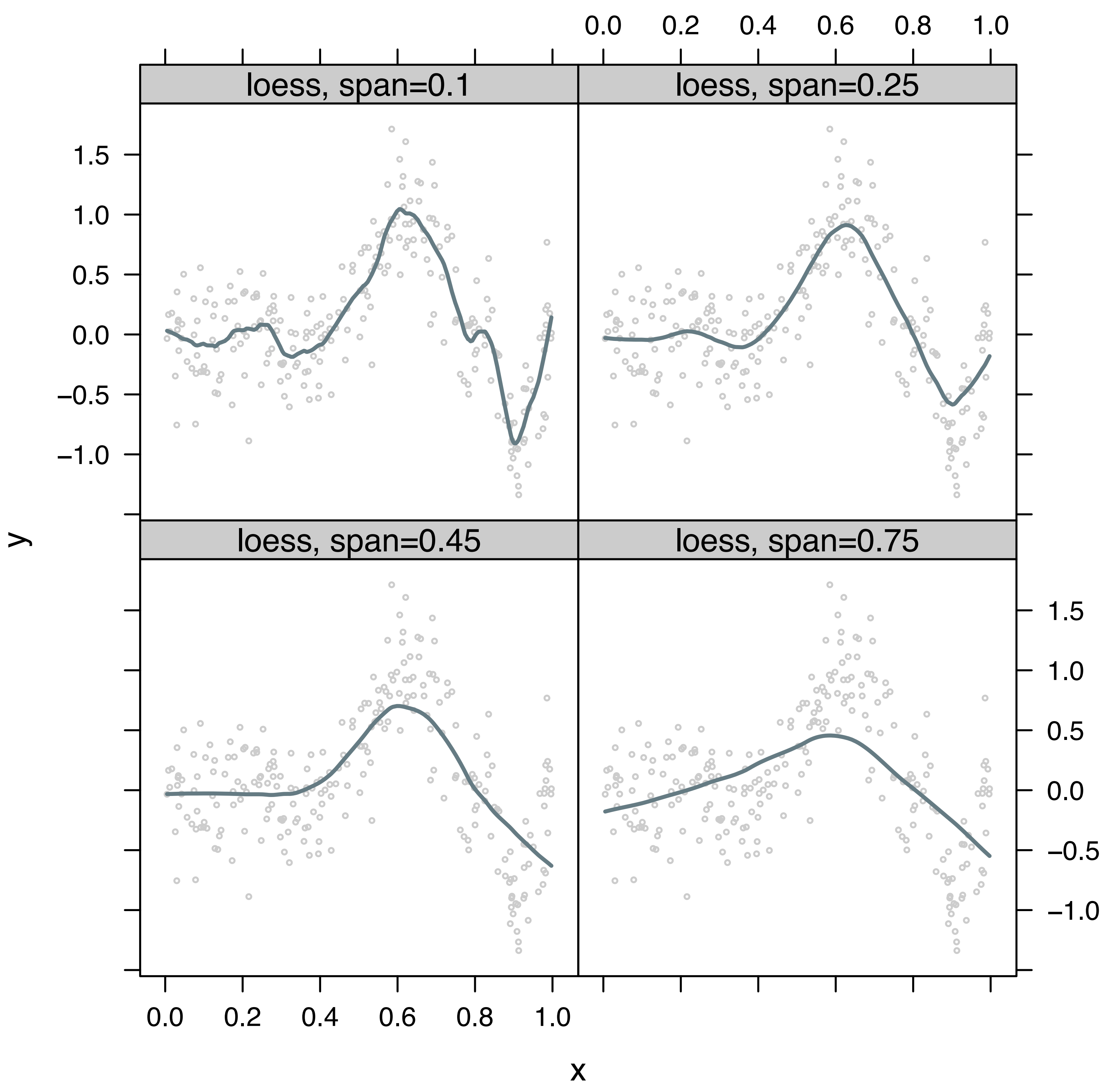
```
xyplot(y ~ x, exa, type = c('p','smooth'), grid = TRUE)
```



`loess` is the smoother used by lattice when type includes "smooth", so it's easy to get loess fit at the default settings (span = 0.75). Doesn't this seem *too* smooth, though?


loess, span=0.1
loess, span=0.25
loess, span=0.45
loess, span=0.75
x
y
0.0
0.2
0.4
0.6
0.8
1.0
−1.0
−0.5
0.0
0.5
1.0
1.5

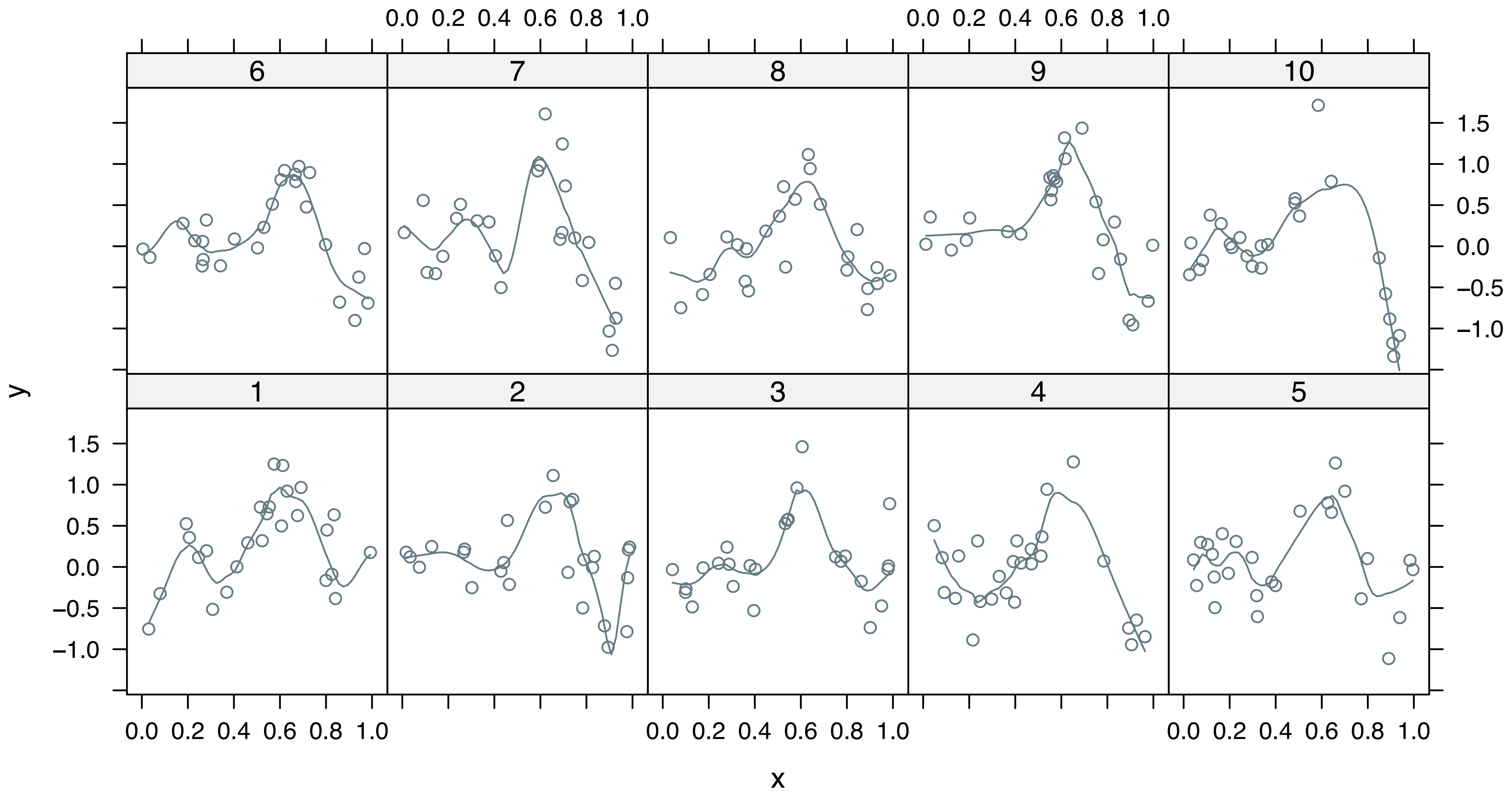

degree = 1
degree = 2
1.5
1.0
0.5
0.0
−0.5
−1.0
y
0.0
0.2
0.4
0.6
0.8
1.0
x



Let's implement the use of cross-validation to select the span.

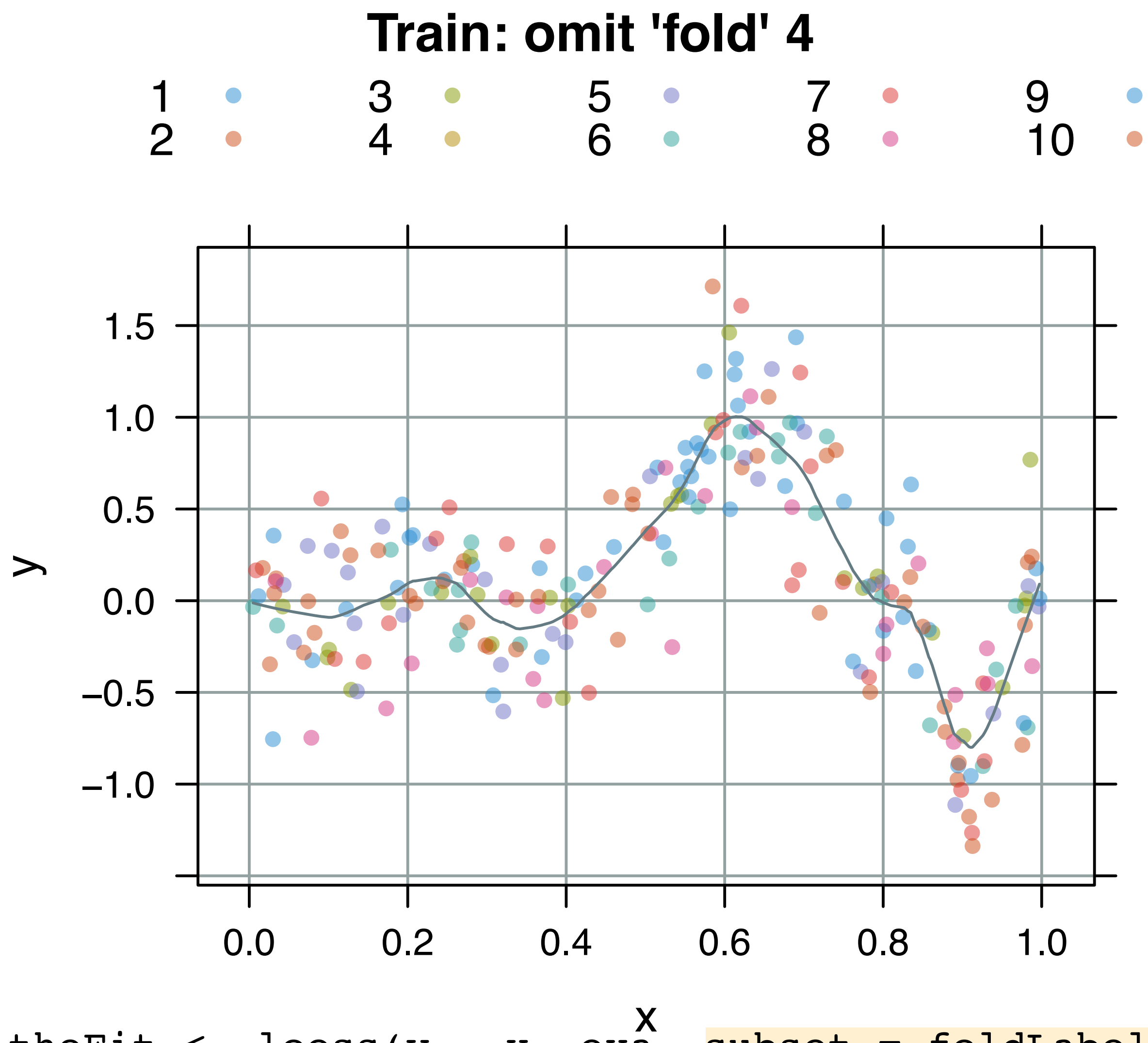
K-fold: randomly divide data into K parts

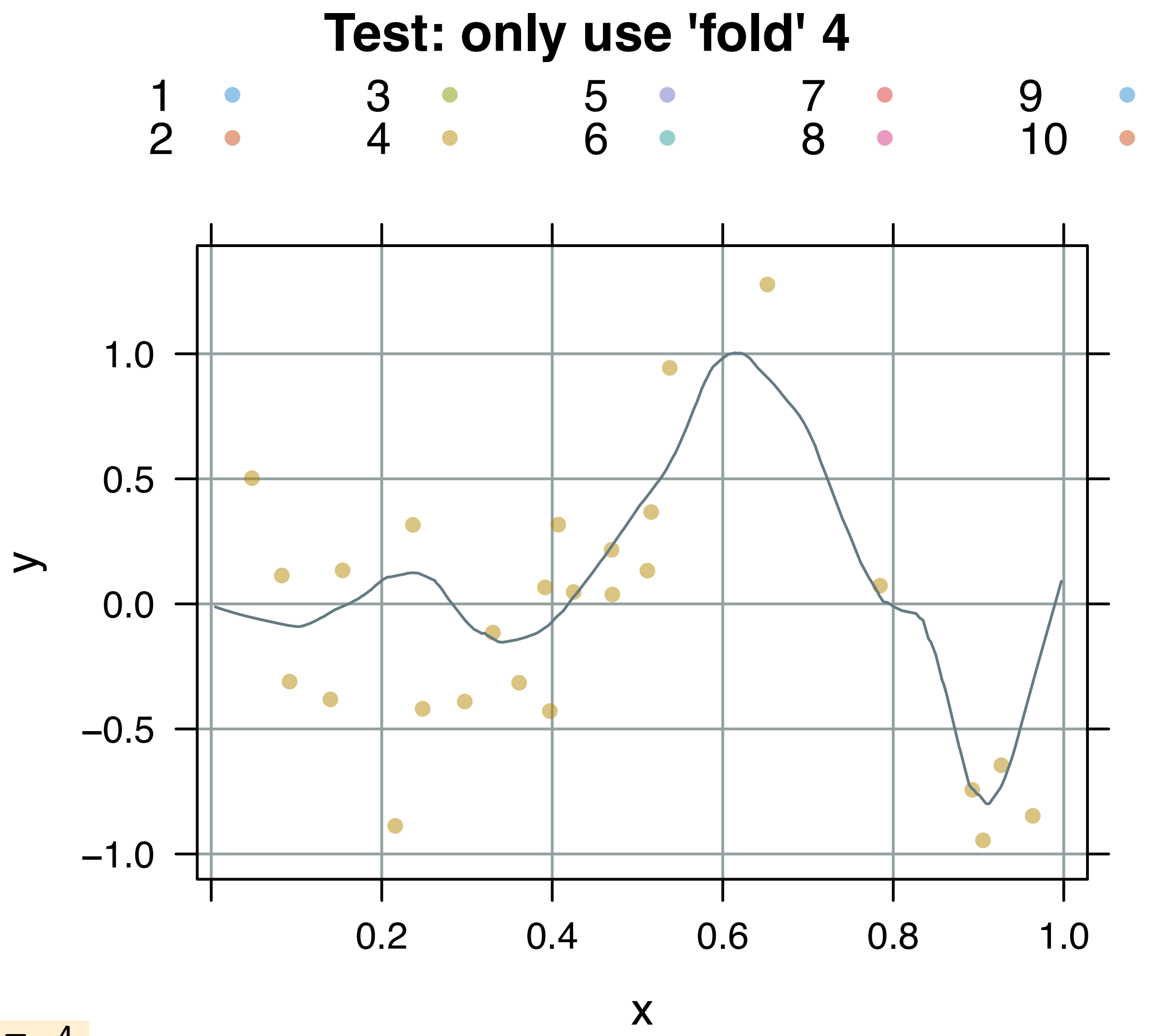
```
> (n <- nrow(exa))                               # 256
[1] 256
> K <- 10
> set.seed(88)
> exa$foldLabels <- sample(rep(1:K, length = n), n, replace = FALSE)
> table(exa$foldLabels)

 1  2  3  4  5  6  7  8  9 10 
26 26 26 26 26 26 25 25 25 25 

> xyplot(y ~ x | factor(foldLabels), exa,
+        type = c('p', 'smooth'), span = 0.3,
+        layout = c(5, 2))
```



▨ 1	2	3	4	5	6	7	8	9	10
1	▨ 2	3	4	5	6	7	8	9	10
1	2	▨ 3	4	5	6	7	8	9	10
1	2	3	▨ 4	5	6	7	8	9	10
1	2	3	4	▨ 5	6	7	8	9	10
1	2	3	4	5	▨ 6	7	8	9	10
1	2	3	4	5	6	▨ 7	8	9	10
1	2	3	4	5	6	7	▨ 8	9	10
1	2	3	4	5	6	7	8	▨ 9	10
1	2	3	4	5	6	7	8	9	▨ 10

□ train	▨ test



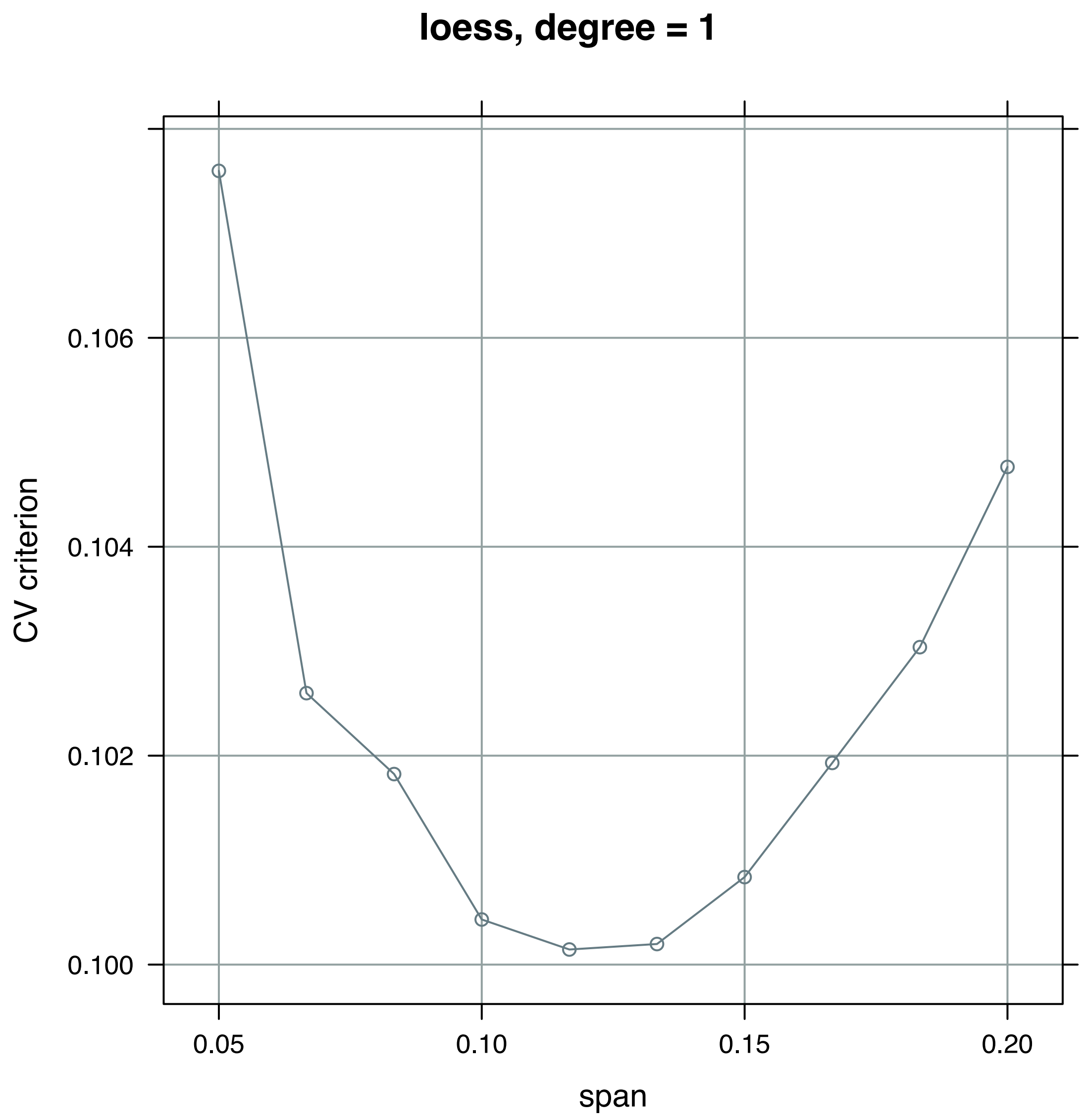


```
> theFit <- loess(y ~ x, exa, subset = foldLabels != 4,
+                 span = theSpan, degree = 1,
+                 control = loess.control(surface = "direct"))
> summary(theFit)
...
Number of Observations: 230
Equivalent Number of Parameters: 10.72
Residual Standard Error: 0.3062
Trace of smoother matrix: 12.74

Control settings:
  normalize:  TRUE
  span       :  0.15
  degree    :  1
  family    :  gaussian
  surface   :  direct
```

```
> thePred <- predict(theFit,
+                    newdata = subset(exa, subset = foldLabels == 4))
> sqrt(mean((exa$y[exa$foldLabels == 4] - thePred)^2))
[1] 0.3402023
```

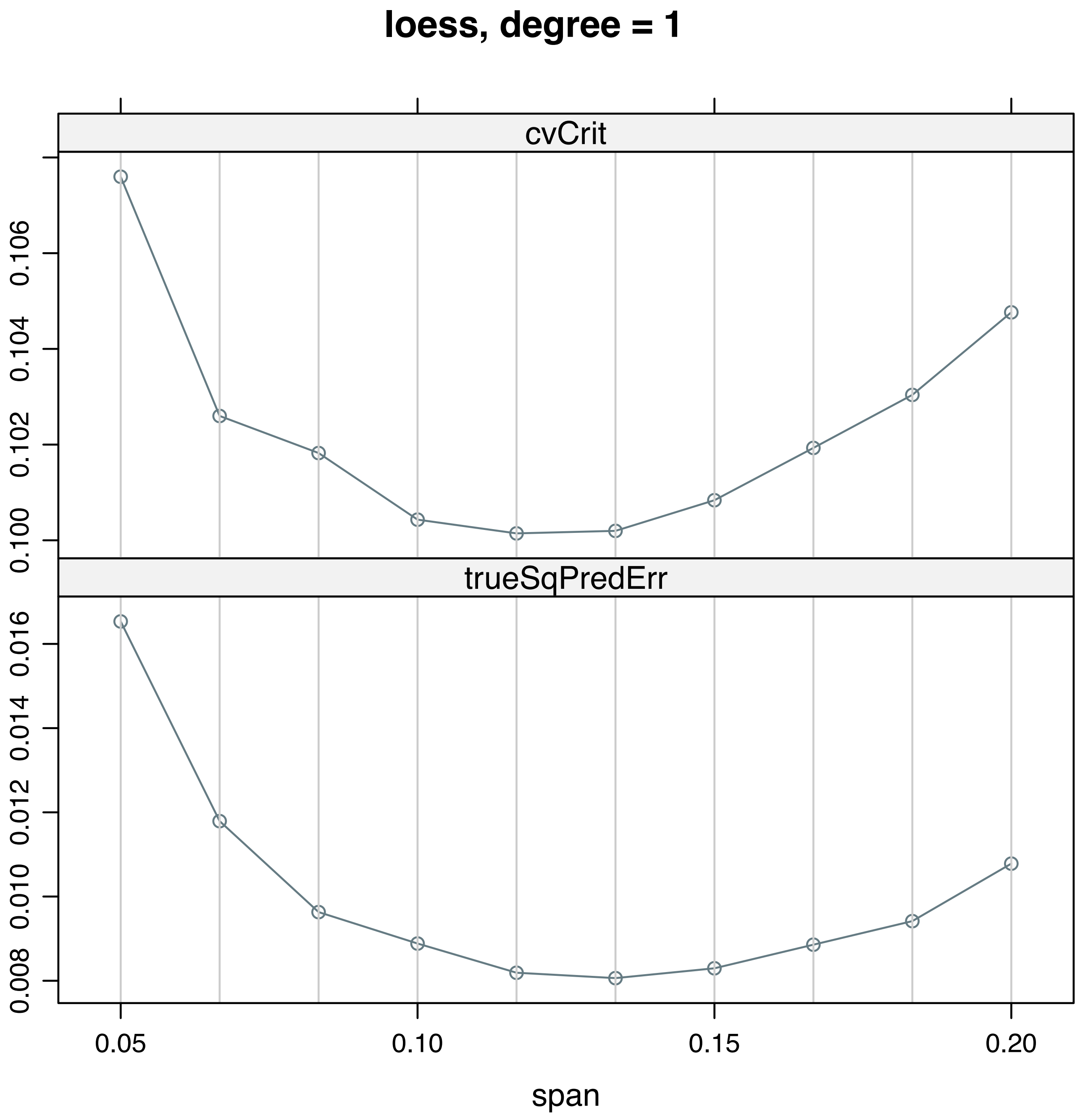
```
> cvSquaredErrors <-
+   sapply(1:K, function(k) {
+     cat(paste("fold =", k, "\n"))
+     theFit <- loess(y ~ x, exa, subset = foldLabels != k,
+                     span = theSpan, degree = 1,
+                     control = loess.control(surface = "direct"))
+     thePred <-
+       predict(theFit,
+               newdata = subset(exa, subset = foldLabels == k))
+     theSquaredError <- (exa$y[exa$foldLabels == k] - thePred)^2
+     return(theSquaredError)
+   })
fold = 1
fold = 2
fold = 3
fold = 4
fold = 5
fold = 6
fold = 7
fold = 8
fold = 9
fold = 10
> sapply(cvSquaredErrors, mean)           # avg sq err by fold
 [1] 0.11730766 0.09362097 0.07410273 0.11573757 0.08207289 0.06686242
 [7] 0.15973350 0.12620991 0.07783665 0.09710683
> sqrt(sapply(cvSquaredErrors, mean))     # sqrt(avg sq err by fold)
 [1] 0.3425021 0.3059754 0.2722182 0.3402023 0.2864837 0.2585777 0.3996667
 [8] 0.3552603 0.2789922 0.3116197
> (cvCrit <- mean(unlist(cvSquaredErrors)))
[1] 0.1008378
> sqrt(cvCrit)
[1] 0.3175497
```

```
## pkg one CV as a function
cvFun <- function(jDat, span) {
  K <- length(unique(jDat$foldLabels))
  cvSquaredErrors <-
    sapply(1:K, function(k) {
      cat(paste("fold =", k, "\n"))
      theFit <- loess(y ~ x, jDat, subset = foldLabels != k,
                      span = span, degree = 1,
                      control = loess.control(surface = "direct"))
      thePred <-
        predict(theFit,
                newdata = subset(jDat, subset = foldLabels == k))
      return((exa$y[exa$foldLabels == k] - thePred)^2)
    })
  return(mean(unlist(cvSquaredErrors)))
}

jSpan <- seq(from = 0.05, to = 0.2, length = 10)

cvCrit <- sapply(seq_along(jSpan), function(i) {
  cat(paste("span =", jSpan[i], "\n"))
  return(cvFun(exa, jSpan[i]))
})

xyplot(cvCrit ~ jSpan, type = 'b',
       xlab = 'span', ylab = 'CV criterion',
       main = 'loess, degree = 1')
```

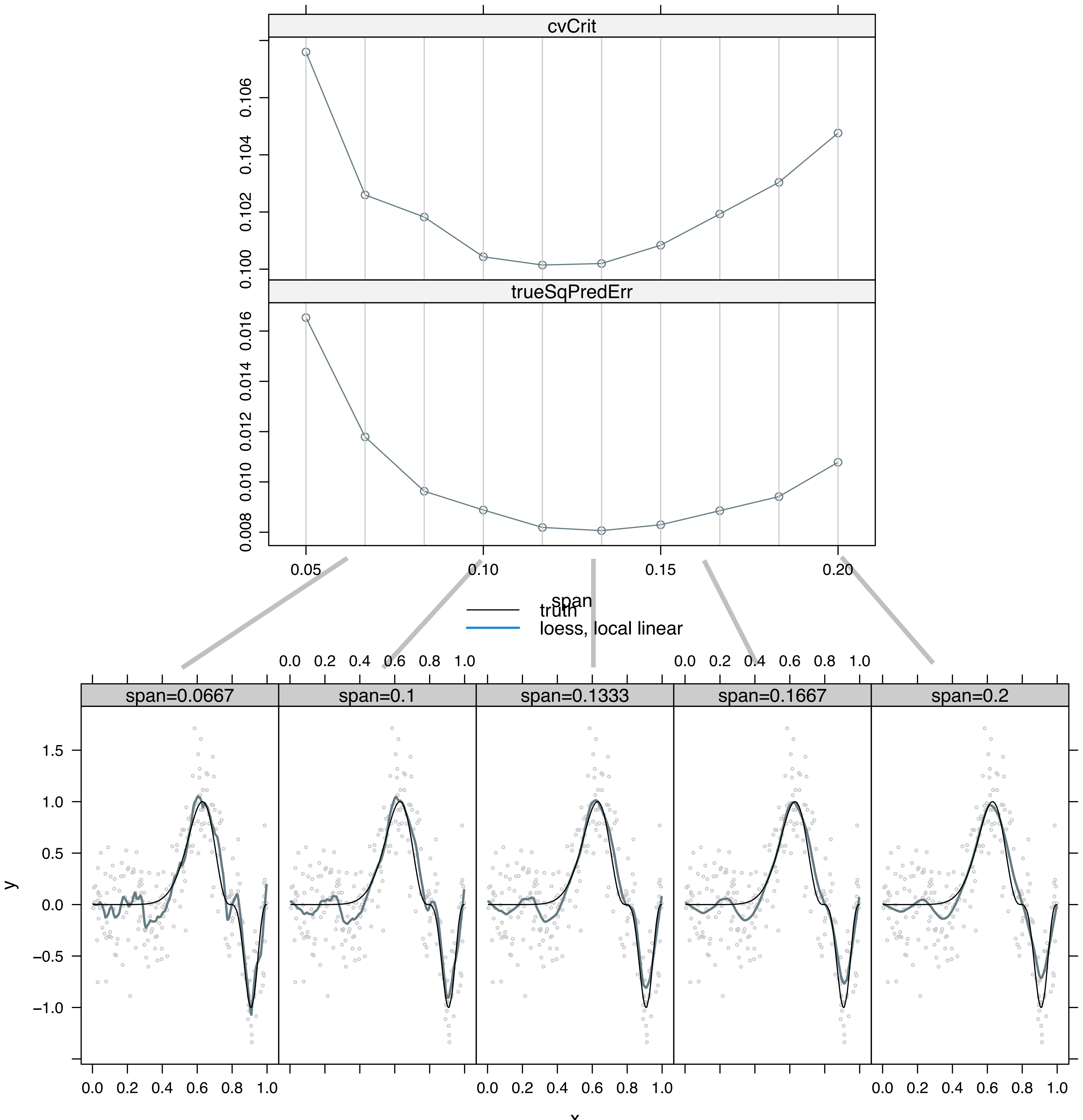


```
## since we know the truth, we can assess the error associated with
## each span
trueSqPredErr <-
  sapply(seq_along(jSpan), function(i) {
    cat(paste("span =", jSpan[i], "\n"))
    theFit <- loess(y ~ x, exa, span = jSpan[i], degree = 1,
                    control = loess.control(surface = "direct"))
    return(mean((exa$m - theFit$fitted)^2))
})

xyplot(trueSqPredErr + cvCrit ~ jSpan, type = 'b',
       scales = list(y = list(relation = 'free')),
       xlab = 'span', ylab = '', layout = c(1, 2),
       main = 'loess, degree = 1', outer = TRUE,
       panel = function(x, y, ...) {
         panel.xyplot(x, y, ...)
         panel.abline(v = jSpan, col = jGray)
       })
```

Minimizer of CV criterion also is the minimizer of true squared prediction error. Illustrates the key idea of CV.



loess, degree = 1



In defense of eyeball-o-metrics

- Methods that are more formal, more mathematical almost always make very specific assumptions.
- These assumptions are often quite inappropriate.
- So, you can assume a nice but wrong model, in order to permit you to Do The Math.
- Or, you can use common sense, years of experience, etc. to make a subjective choice.
- Both approaches are undoubtedly wrong and subjective. In general, we cannot say which one is more wrong.
- Theoretical solutions are incredibly valuable for offering constructive insight into how/why a system works (or doesn't!) but there is still a big role for judgement and common sense in real applications.

Summary on smoothing

Kernel smoothing is an extension of kernel density estimation; histogram --> kernel density estimate, moving average --> kernel smoother

Local polynomials are an extension of kernel smoothers

In the absence of any other direction, use loess to get a local linear fit

Summary on smoothing

Cross validation provides a good way to select tuning parameters; feel free to use pre-packaged CV implementations

Don’t let yourself completely off the hook for choosing the amount of smoothing that is appropriate to your goals; use your own brain and eyeballs

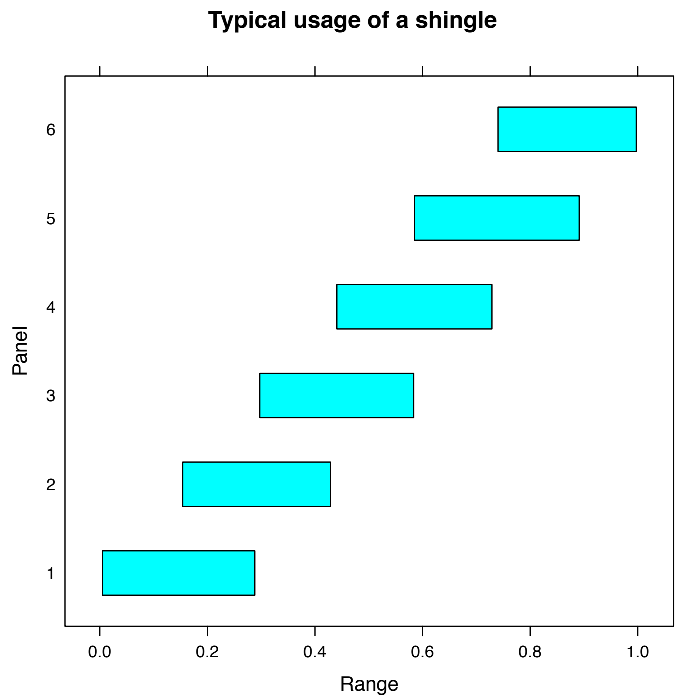
Note that CV is also used to estimate prediction error (as opposed to ‘model selection’ as done here, where we just seek to find the tuning parameter value that minimizes it) and there are some differences

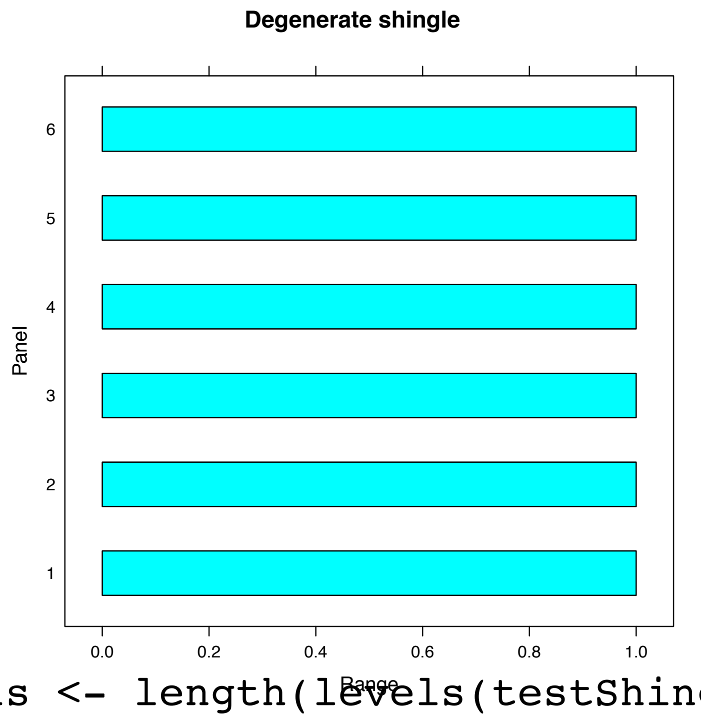
References for smoothing, CV

- Faraway, Julian. Extending the Linear Model With R. Chapman & Hall/CRC 2006. Available via MyiLibrary.
- Hastie, T., Tibshirani, R., Friedman, J. The Elements of Statistical Learning. Springer 2001. Available via SpringerLink.
- Efron, B., Tibshirani, R. An Introduction to the Bootstrap. Chapman & Hall/CRC 1998.

"Mechanics" Sidebar

How to 'trick' lattice into exploring a tuning parameter ... degenerate shingles





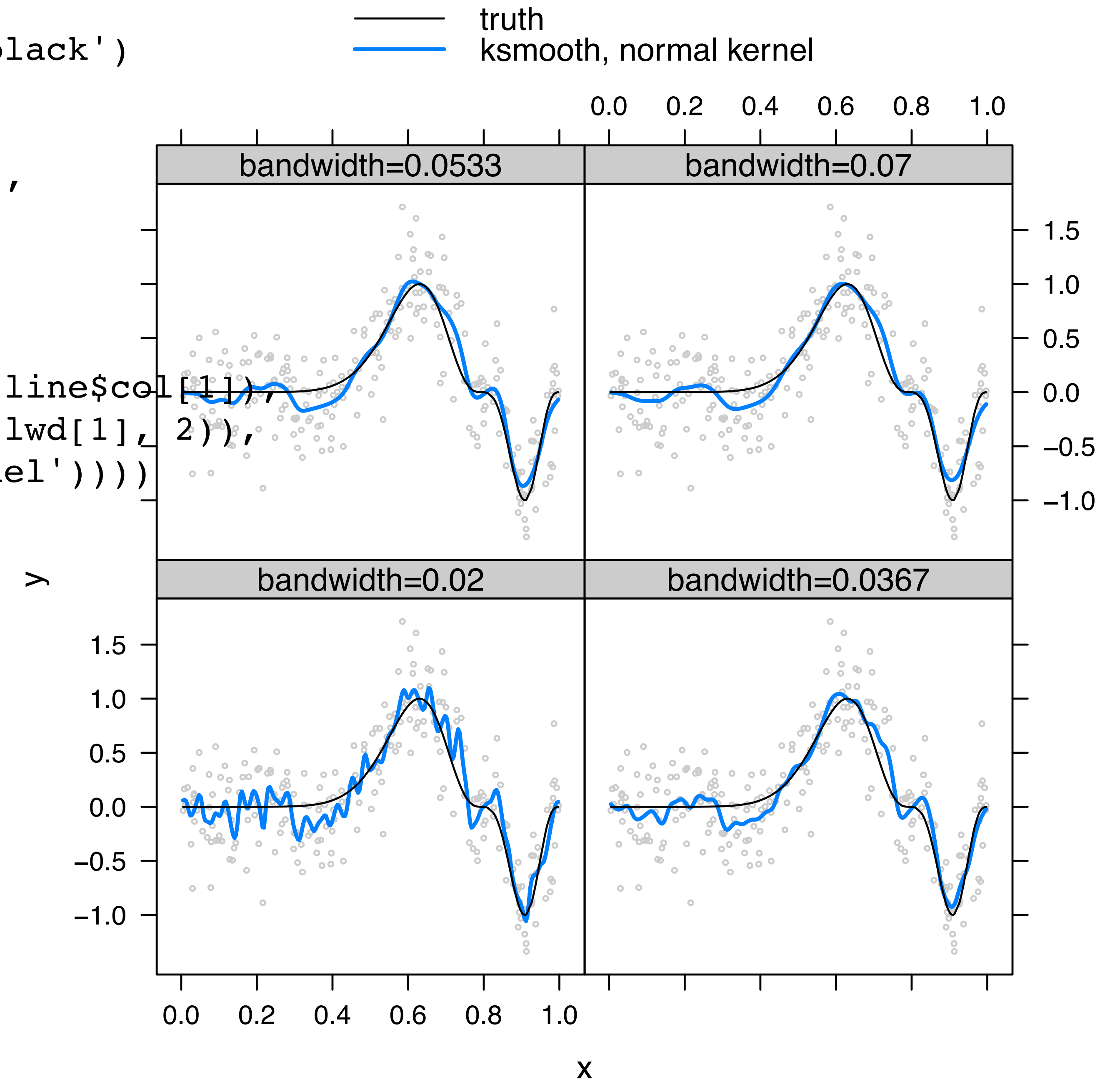
```
> testShingle1 <- equal.count(exa$x)
> summary(testShingle1)

Intervals:
      min     max count
1 0.00475 0.28815    73
2 0.15405 0.42875    73
3 0.29735 0.58345    73
4 0.44065 0.72895    73
5 0.58475 0.89115    73
6 0.74025 0.99725    73

Overlap between adjacent intervals:
[1] 36 37 36 37 36
> plot(testShingle1,
+      main = "Typical usage of a shingle")
```

```
> nLevels <- length(levels(testShingle1))
> jIntervals <- cbind(rep(0, nLevels), rep(1,
nLevels))
> testShingle2 <- shingle(exa$x, jIntervals)
> summary(testShingle2)

Intervals:
  min max count
1   0   1   256
2   0   1   256
3   0   1   256
4   0   1   256
5   0   1   256
6   0   1   256

Overlap between adjacent intervals:
[1] 256 256 256 256 256
> plot(testShingle2, main = "Degenerate shingle")
```

Sidebar

How the degenerate shingle comes in

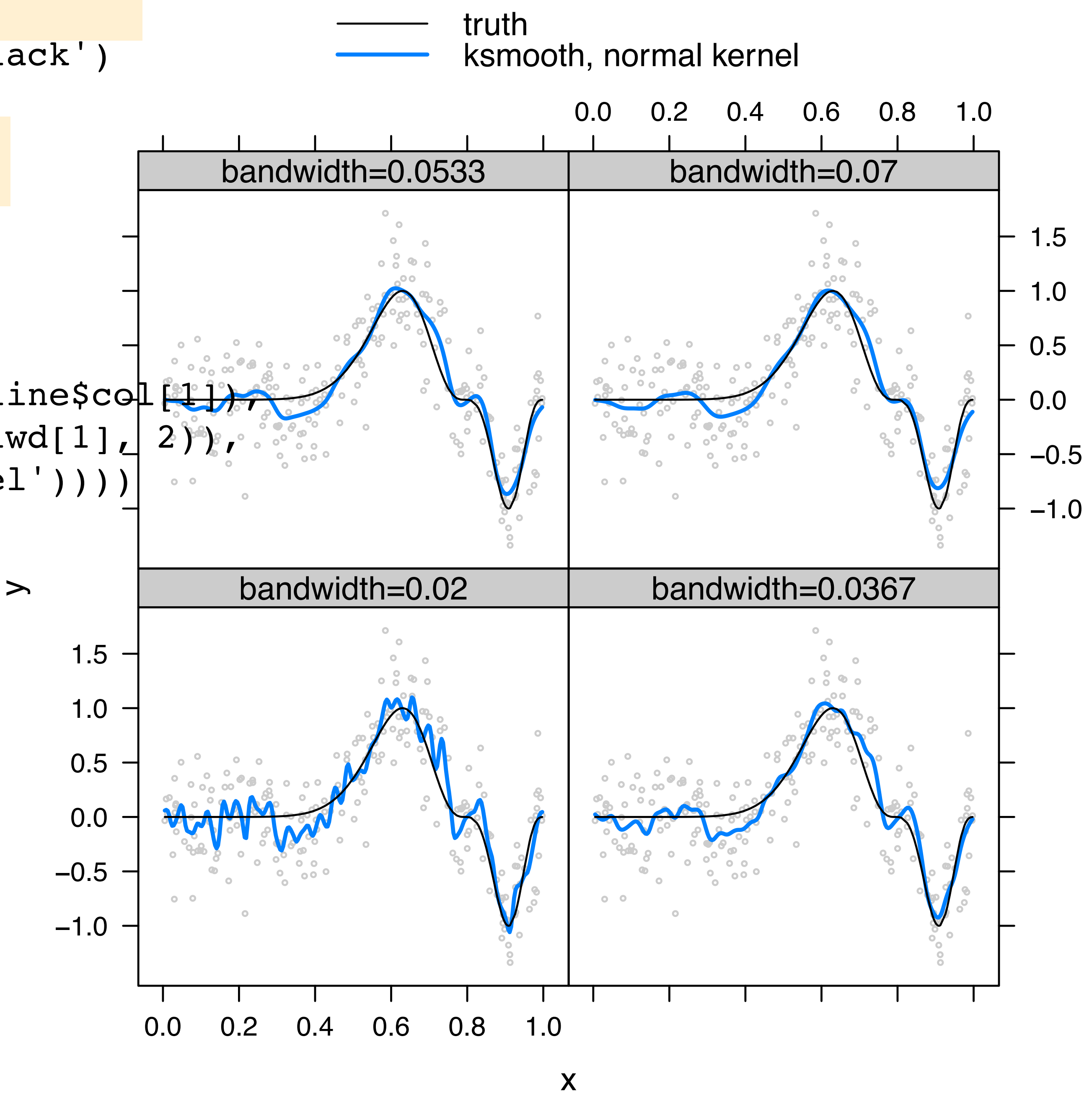
```
## kernel smoothing w/ different bandwidths
nBw <- 4
jBw <- seq(from = 0.02, to = 0.07, length = nBw)
jIntervals <- cbind(rep(0, nBw), rep(1, nBw))
exa$xShingle <- shingle(exa$x, jIntervals)

xyplot(y ~ x | xShingle, exa,
       panel = function(x, y, ...) {
         panel.xyplot(x, y, cex = 0.3, col = jGray, ...)
         z <- ksmooth(x, y, "normal", jBw[packet.number()])
         panel.lines(z$x, z$y, lwd = 2, ...)
         panel.xyplot(x, exa$m, type = 'l', col = 'black')
       },
       strip = strip.custom(var.name = 'bandwidth',
         factor.levels = as.character(round(jBw, 4)),
         strip.levels = c(TRUE, TRUE),
         fg = jGray, sep = "="),
       key = list(space = 'top',
         lines = list(col = c('black',
                        trellis.par.get()$superpose.line$col[1]),
           lwd = c(trellis.par.get()$superpose.line$lwd[1], 2)),
         text = list(c('truth','ksmooth, normal kernel'))))
```



Sidebar

How the different bandwidths come in

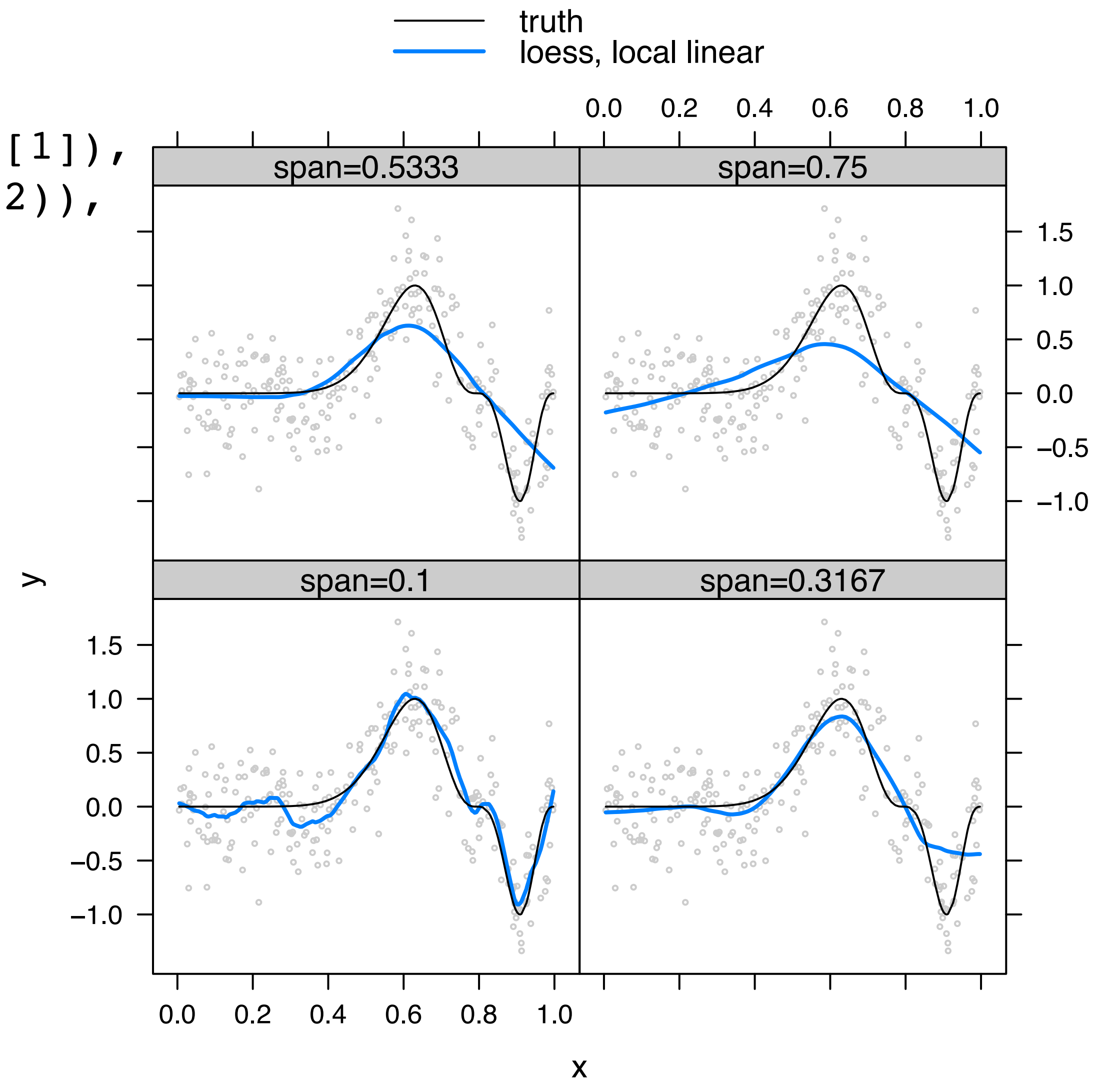
```
## kernel smoothing w/ different bandwidths
nBw <- 4
jBw <- seq(from = 0.02, to = 0.07, length = nBw)
jIntervals <- cbind(rep(0, nBw), rep(1, nBw))
exa$xShingle <- shingle(exa$x, jIntervals)

xyplot(y ~ x | xShingle, exa,
       panel = function(x, y, ...) {
         panel.xyplot(x, y, cex = 0.3, col = jGray, ...)
         z <- ksmooth(x, y, "normal", jBw[packet.number()])
         panel.lines(z$x, z$y, lwd = 2, ...)
         panel.xyplot(x, exa$m, type = 'l', col = 'black')
       },
       strip = strip.custom(var.name = 'bandwidth',
         factor.levels = as.character(round(jBw, 4)),
         strip.levels = c(TRUE, TRUE),
         fg = jGray, sep = "="),
       key = list(space = 'top',
         lines = list(col = c('black',
                        trellis.par.get()$superpose.line$col[1]),
           lwd = c(trellis.par.get()$superpose.line$lwd[1], 2)),
         text = list(c('truth','ksmooth, normal kernel'))))
```

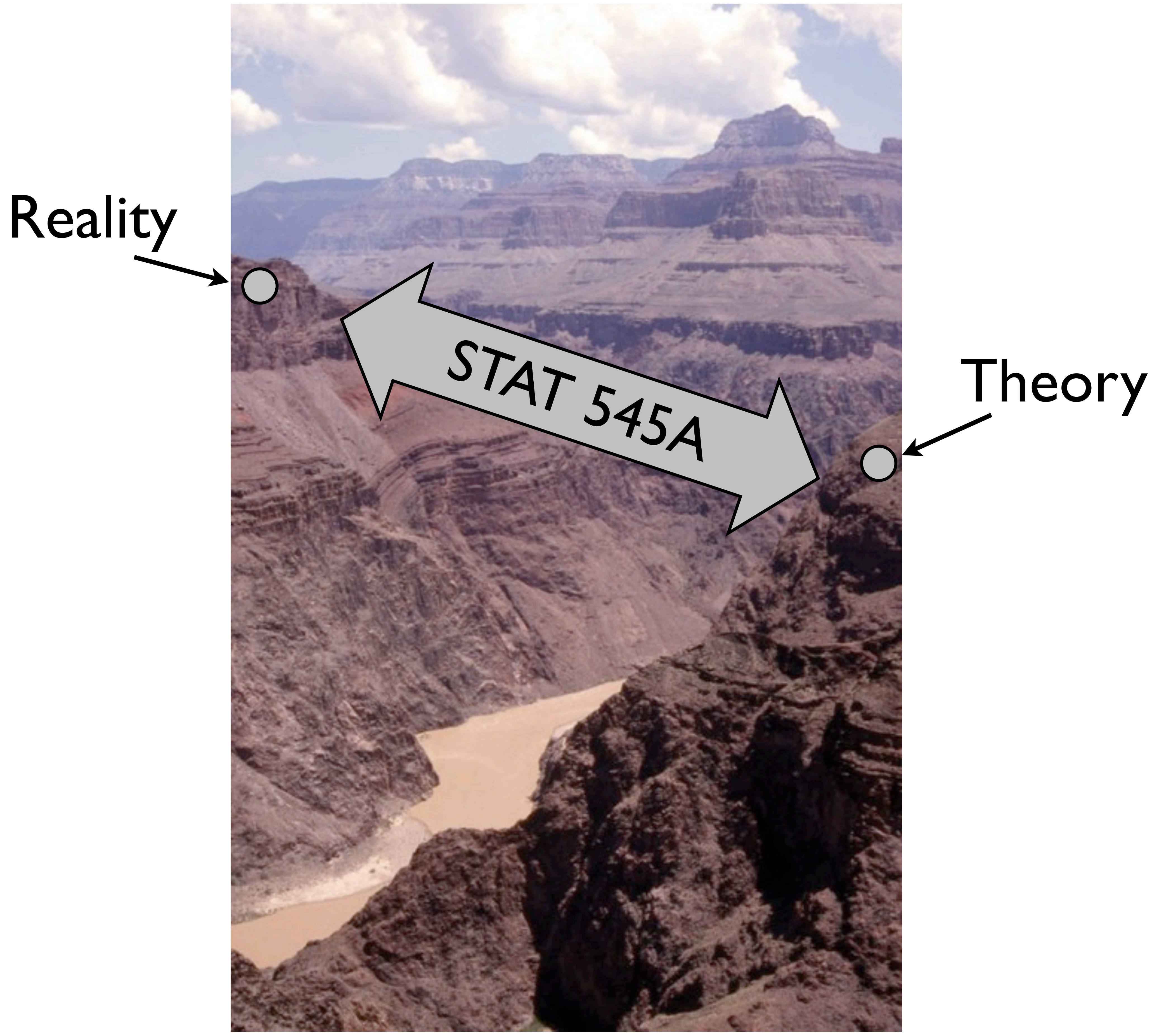


Sidebar

Same idea, but with loess and span

```
## local linear fits
jSpan <- seq(from = 0.1, to = 0.75, length = nBw)

xyplot(y ~ x | xShingle, exa,
       panel = function(x, y, ...) {
         panel.xyplot(x, y, cex = 0.3, col = jGray, ...)
         z <- loess(y ~ x, span = jSpan[packet.number()],
                    degree = 1)
         panel.lines(z$x, z$fitted, lwd = 2, ...)
         panel.xyplot(x, exa$m, type = 'l', col = 'black')
       },
       strip = strip.custom(var.name = 'span',
         factor.levels = as.character(round(jSpan, 4)),
         strip.levels = c(TRUE, TRUE),
         fg = jGray, sep = "="),
       key = list(space = 'top',
         lines = list(col = c('black',
                          trellis.par.get()$superpose.line$col[1]),
           lwd = c(trellis.par.get()$superpose.line$lwd[1], 2)),
         text = list(c('truth','loess, local linear'))))
```




Reality
STAT 545A
Theory

Two inter-related goals

- Foster your development of a personal philosophy on data analysis, especially exploratory analysis.
- Strengthen your data analysis skills.

My hope:

You'll leave this course with (at least the beginnings of) a confident, deliberate attitude about how to approach data analysis and a base level of practical skills to put your attitude into action.