



ETC3580: Advanced Statistical Modelling

Week 9: Nonparametric regression

Outline

- 1 Nonlinear regression
- 2 Kernel estimators
- 3 Local polynomials
- 4 Interpolating splines
- 5 Smoothing splines
- 6 Regression splines
- 7 Multivariate predictors

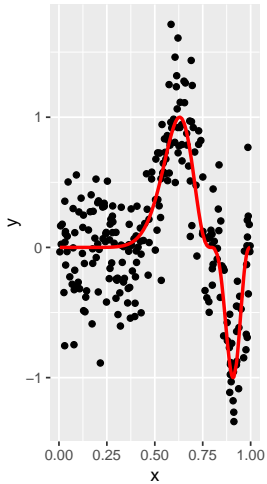
Nonlinear regression

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

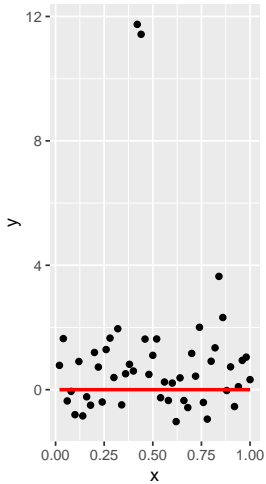
- How to estimate f ?
- Could assume $f(x) = \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3$
- Or $f(x) = \beta_0 + \beta_1 x^{\beta_2}$
- OK if you *know* the right form.
- But often better to assume only that f is continuous and smooth.

Examples

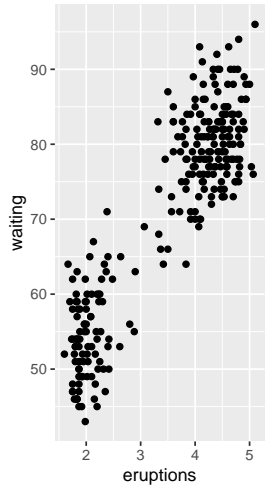
Example A



Example B



Old Faithful



Outline

- 1 Nonlinear regression
- 2 Kernel estimators
- 3 Local polynomials
- 4 Interpolating splines
- 5 Smoothing splines
- 6 Regression splines
- 7 Multivariate predictors

Kernel estimators

Nadaraya-Watson estimator

$$\hat{f}_h(x) = \frac{\sum_{j=1}^n K\left(\frac{x - x_j}{h}\right) y_j}{\sum_{j=1}^n K\left(\frac{x - x_j}{h}\right)}$$

- K is a kernel function where $\int K = 1$,
 $K(a) = K(-a)$, and $K(0) \geq K(a)$, for all a .
- \hat{f} is a weighted moving average
- Need to choose K and h .

Common kernels

Uniform

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

Epanechnikov

$$K(x) = \begin{cases} \frac{3}{4}(1 - x^2) & -1 < x < 1 \\ 0 & \text{otherwise.} \end{cases}$$

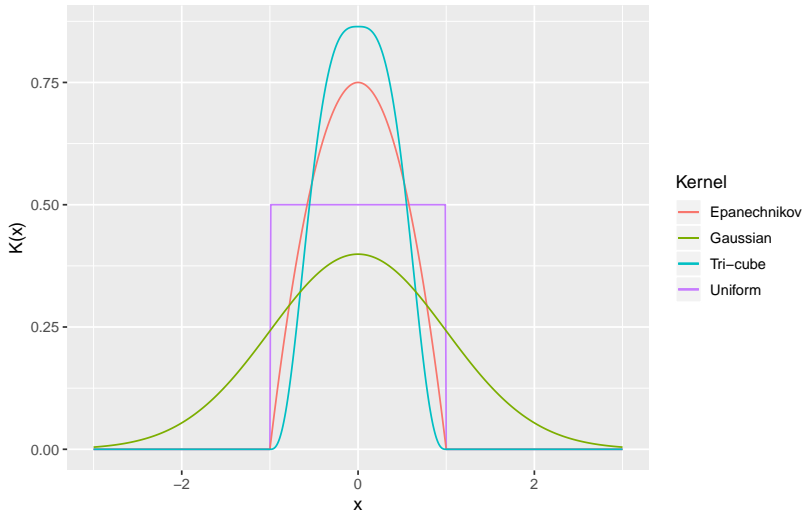
Tri-cube

$$K(x) = \begin{cases} c(1 - |x|^3)^3 & -1 < x < 1 \\ 0 & \text{otherwise.} \end{cases}$$

Gaussian

$$K(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}x^2}$$

Common kernels



Old Faithful

Kernel smoothing

- A smooth kernel is better, but otherwise the choice of kernel makes little difference.
- Optimal kernel (minimizing MSE) is Epanechnikov. It is also fast.
- The choice of h is crucial.

Example A

Example B

Cross-validation

$$CV(h) = \frac{1}{n} \sum_{j=1}^n (y_j - \hat{f}_h^{(-j)}(x_j))^2$$

- $(-j)$ indicates j th point omitted from estimate
- Pick h that minimizes CV.
- Works ok provided there are no duplicate (x, y) pairs. But occasionally odd results.

Let $y = f(x) + \varepsilon$ where $\varepsilon \sim IID(0, \sigma^2)$. Then

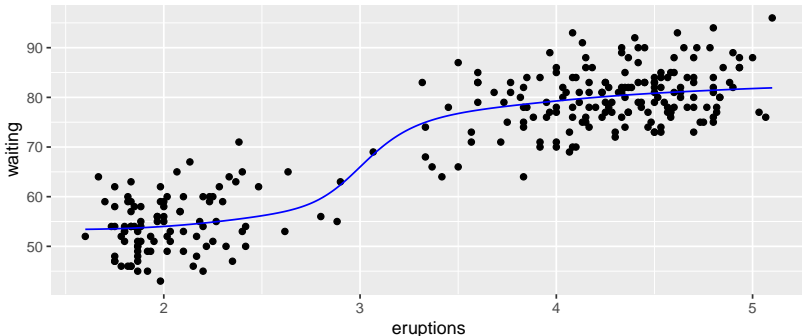
$$MSE(h) = E[f(x) - \hat{f}_h(x)]^2 \approx \frac{\sigma^2 r_K}{nh} + \frac{v_K^2 h^4 [f''(x)]^2}{4}$$

- $r_K = \int K^2(x)dx$, $v_K = \int x^2 K(x)dx$
- Consistent estimator if $nh \rightarrow \infty$ and $h \rightarrow 0$ as $n \rightarrow \infty$.
- $h_{\text{optimal}} \approx \left(\frac{r_K}{n[f''(x)]^2 v_K^2} \right)^{1/5}$

Kernel smoothing in R

Many packages available. One of the better ones is **KernSmooth**:

```
fit <- locpoly(faithful$eruptions, faithful$waiting,  
              degree=0, bandwidth=0.3) %>% as.tibble  
ggplot(faithful) +  
  geom_point(aes(x=eruptions,y=waiting)) +  
  geom_line(data=fit, aes(x=x,y=y), col='blue')
```



Outline

- 1 Nonlinear regression
- 2 Kernel estimators
- 3 Local polynomials**
- 4 Interpolating splines
- 5 Smoothing splines
- 6 Regression splines
- 7 Multivariate predictors

Local polynomials

Kernel smoothing is a local constant method:

$$\text{WLS}(x) = \sum_{j=1}^n w_j(x)(y_j - a_0)^2$$

$$\text{where } w_j(x) = \frac{K\left(\frac{x-x_j}{h}\right)}{\sum_{i=1}^n K\left(\frac{x-x_i}{h}\right)}$$

is minimized by

$$\hat{f}(x) = \hat{a}_0 = \sum_{j=1}^n w_j(x)y_j$$

Local polynomials

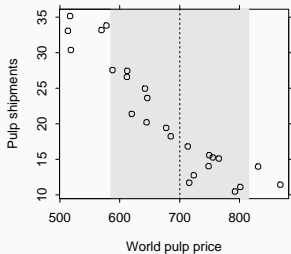
We can compute local linear instead:

$$\text{WLS}(x) = \sum_{j=1}^n w_j(x) (y_j - a_0 - a_1(x_j - x))^2$$

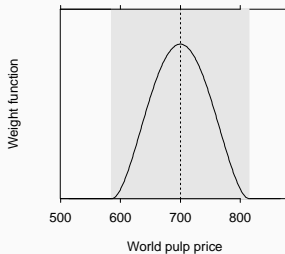
$$\hat{f}(x) = \hat{a}_0$$

Local polynomials

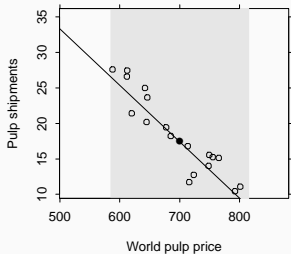
Step 1



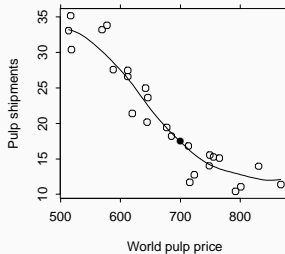
Step 2



Step 3



Result



Local polynomials

$$\text{WLS}(x) = \sum_{j=1}^n w_j(x) (y_j - \sum_{k=0}^p a_k (x_j - x)^k)^2$$

$$\hat{f}(x) = \hat{a}_0$$

- Local linear and local quadratic are commonly used.
- Robust regression can be used instead
- Less biased at boundaries than kernel smoothing
- Local quadratic less biased at peaks and troughs than local linear or kernel

Local polynomials in R

- One useful implementation is `KernSmooth::locpoly(x, y, degree, bandwidth)`
- Uses Gaussian kernel.
- `dpill` can be used to choose the bandwidth h if `degree=1`.
- Many people seem to use trial and error — finding the largest h that captures what they think they see by eye.

Loess

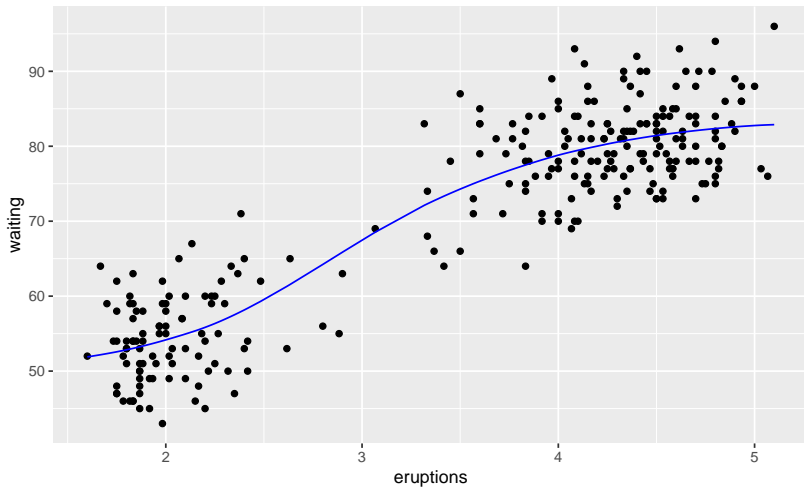
Best known implementation is **loess** (locally quadratic)

```
fit <- loess(y ~ x, span=0.75, degree=2,  
            family="gaussian", data)
```

- Uses tri-cube kernel and variable bandwidth.
- span controls bandwidth. Specified in terms of percentage of data covered.
- degree is order of polynomial
- Use family="symmetric" for a robust fit

Loess

Old Faithful (Loess, span=0.75)

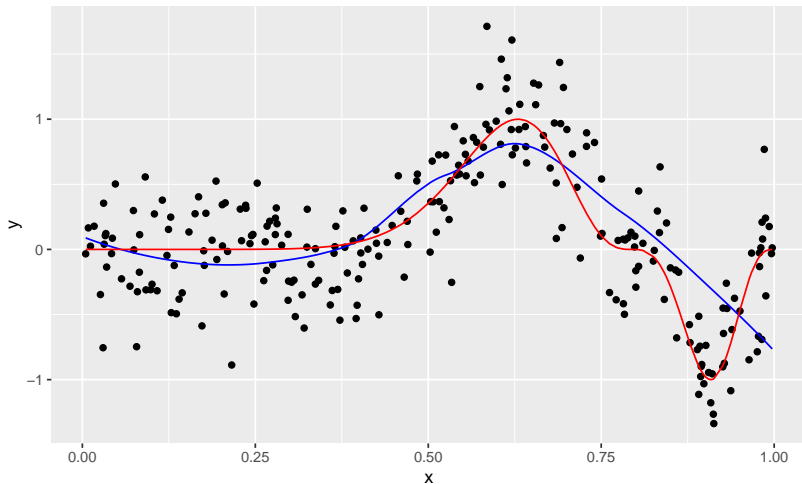


Loess in R

```
smr <- loess(waiting ~ eruptions, data=faithful)
ggplot(faithful) +
  geom_point(aes(x=eruptions,y=waiting)) +
  ggtitle("Old Faithful (Loess, span=0.75)") +
  geom_line(aes(x=eruptions, y=fitted(smr)),
    col='blue')
```

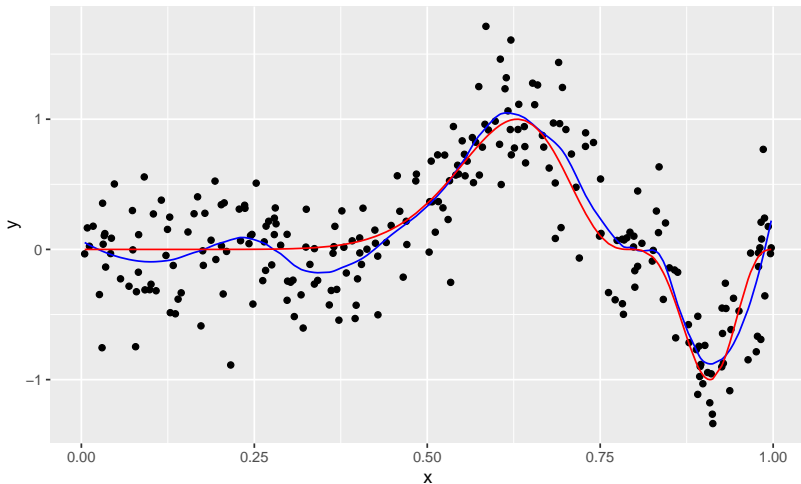

Loess

Example A (Loess, span=0.75)



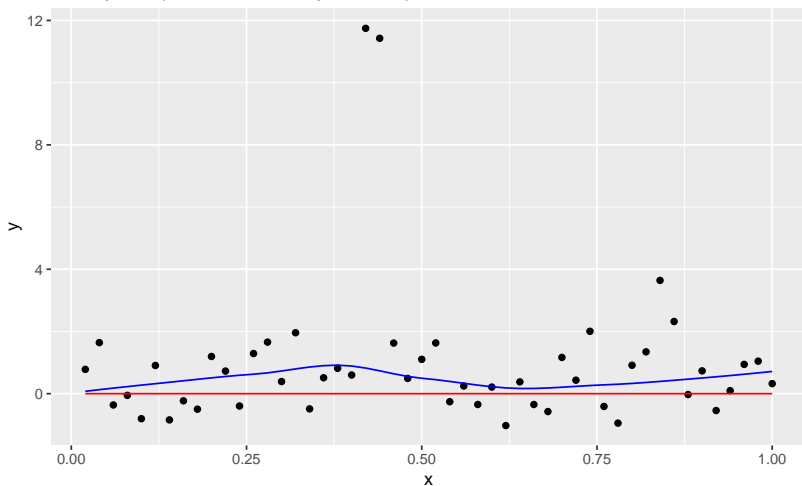
Loess

Example A (Loess, span=0.22)



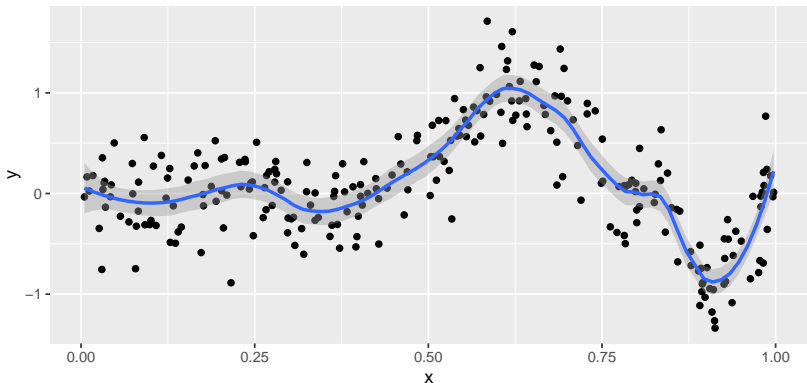
Loess

Example B (Robust Loess, span=0.75)



Loess and geom_smooth()

```
ggplot(exa) +  
  geom_point(aes(x=x,y=y)) +  
  geom_smooth(aes(x=x,y=y), method='loess',  
    span=0.22)
```



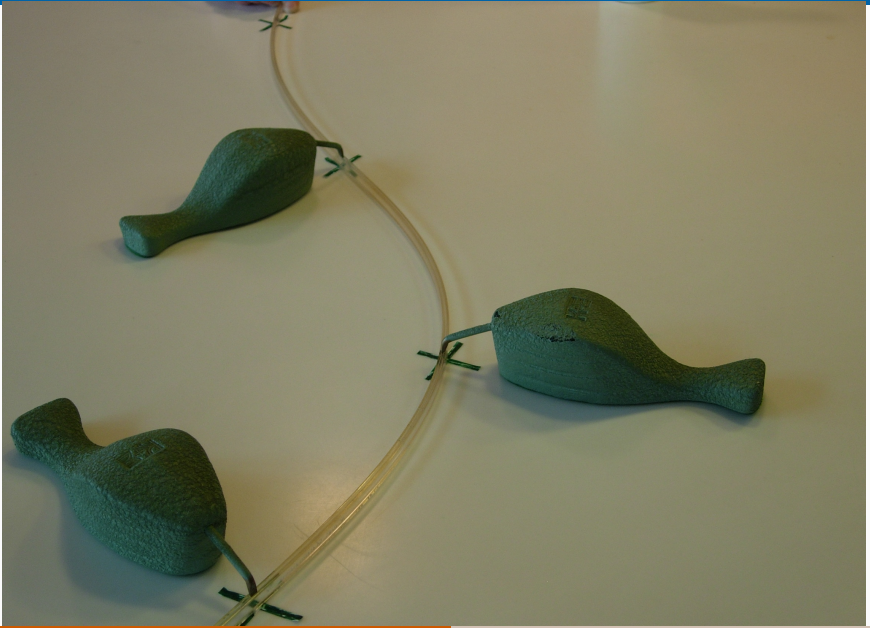
Loess and geom_smooth()

- Because local polynomials use local linear models, we can easily find standard errors for the fitted values.
- Connected together, these form a pointwise confidence band.
- Automatically produced using `geom_smooth`

Outline

- 1 Nonlinear regression
- 2 Kernel estimators
- 3 Local polynomials
- 4 Interpolating splines**
- 5 Smoothing splines
- 6 Regression splines
- 7 Multivariate predictors

Interpolating splines



Interpolating splines



Interpolating splines



Interpolating splines

A spline is a continuous function $f(x)$ interpolating all points (κ_j, y_j) for $j = 1, \dots, K$ and consisting of polynomials between each consecutive pair of 'knots' κ_j and κ_{j+1} .

Interpolating splines

A spline is a continuous function $f(x)$ interpolating all points (κ_j, y_j) for $j = 1, \dots, K$ and consisting of polynomials between each consecutive pair of 'knots' κ_j and κ_{j+1} .

- Parameters constrained so that $f(x)$ is continuous.
- Further constraints imposed to give continuous derivatives.
- Cubic splines most common, with f', f'' continuous.

Outline

- 1 Nonlinear regression
- 2 Kernel estimators
- 3 Local polynomials
- 4 Interpolating splines
- 5 Smoothing splines**
- 6 Regression splines
- 7 Multivariate predictors

Smoothing splines

Let $y = f(x) + \varepsilon$ where $\varepsilon \sim IID(0, \sigma^2)$. Then

Choose \hat{f} to minimize

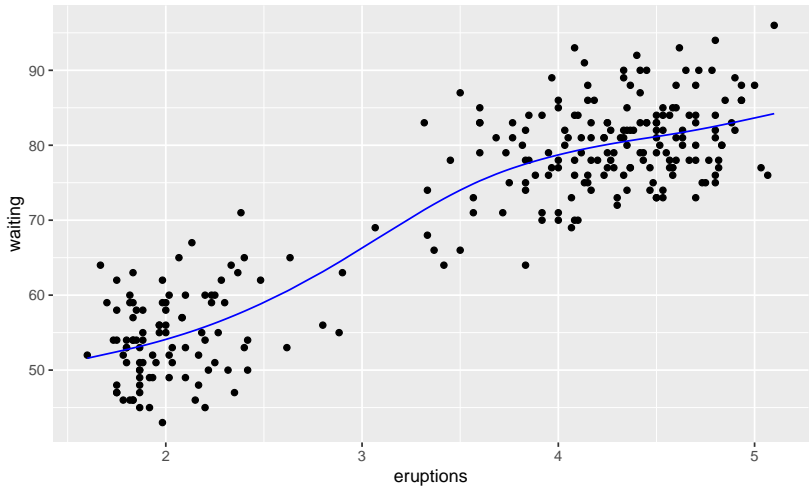
$$\frac{1}{n} \sum_i (y_i - f(x_i))^2 + \lambda \int [f''(x)]^2 dx$$

- λ is smoothing parameter to be chosen
- $\int [f''(x)]^2 dx$ is a measure of roughness.
- Solution: \hat{f} is a cubic spline with knots $\kappa_i = x_i$, $i = 1, \dots, n$ (ignoring duplicates).
- Other penalties lead to higher order splines
- Cross-validation can be used to select λ .

Smoothing splines

Smoothing splines

Old Faithful (Smoothing spline, lambda chosen by CV)

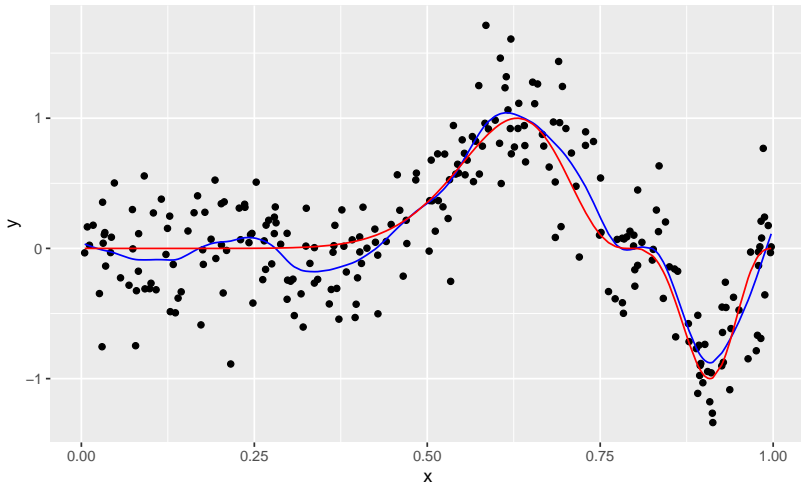


Smoothing splines

```
smr <- smooth.spline(  
  faithful$eruptions,  
  faithful$waiting,  
  cv=TRUE)  
smr <- data.frame(x=smr$x,y=smr$y)  
ggplot(faithful) +  
  geom_point(aes(x=eruptions,y=waiting)) +  
  ggtitle("Old Faithful (Smoothing spline, lambda chosen by  
  geom_line(data=smr, aes(x=x, y=y), col='blue'))
```

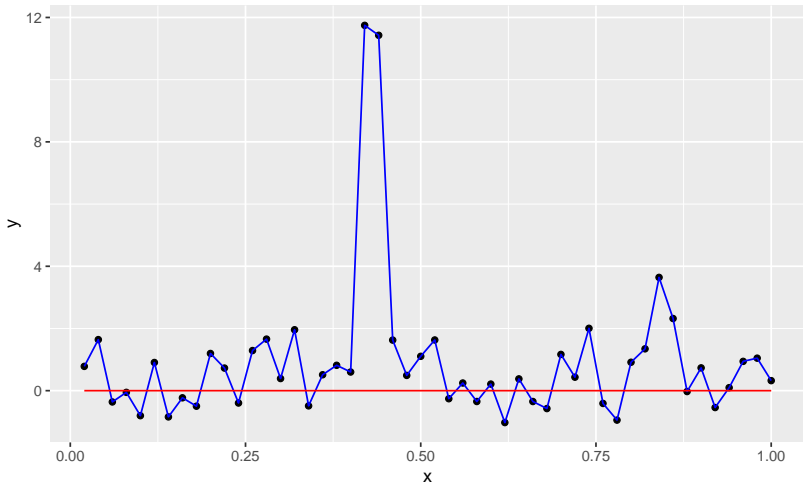

Smoothing splines

Example A (Smoothing spline, lambda chosen by CV)



Smoothing splines

Example B (Smoothing spline, lambda chosen by CV)



Outline

- 1 Nonlinear regression
- 2 Kernel estimators
- 3 Local polynomials
- 4 Interpolating splines
- 5 Smoothing splines
- 6 Regression splines**
- 7 Multivariate predictors

Regression splines

- Fewer knots than smoothing splines
- Need to choose the knots rather than a smoothing parameter.
- Can be estimated as a linear model once knots are selected.

General cubic regression splines

- Let $\kappa_1 < \kappa_2 < \dots < \kappa_K$ be “knots” in interval (a, b) .
- Let $x_1 = x$, $x_2 = x^2$, $x_3 = x^3$, $x_j = (x - \kappa_{j-3})_+^3$ for $j = 4, \dots, K + 3$.
- Then the regression of y on x_1, \dots, x_{K+3} is piecewise cubic, but smooth at the knots.
- Choice of knots can be difficult and arbitrary.
- Automatic knot selection algorithms very slow.
- Often use equally spaced knots. Then only need to choose K .

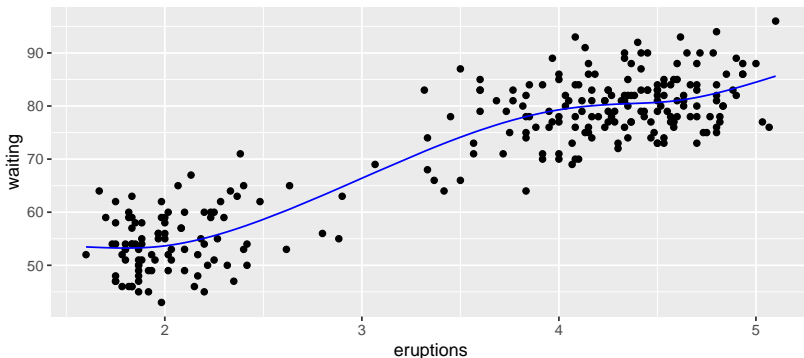
B-splines and natural splines

- B-splines provide an equivalent set of basis functions.
- Natural cubic splines are a variation on B-splines with linear boundary conditions.
- These are usually more stable
- Implemented in `splines::ns` function in R
- Can specify knots explicitly, or `df`. Then $df-2$ knots are selected at quantiles of x .

Natural splines in R

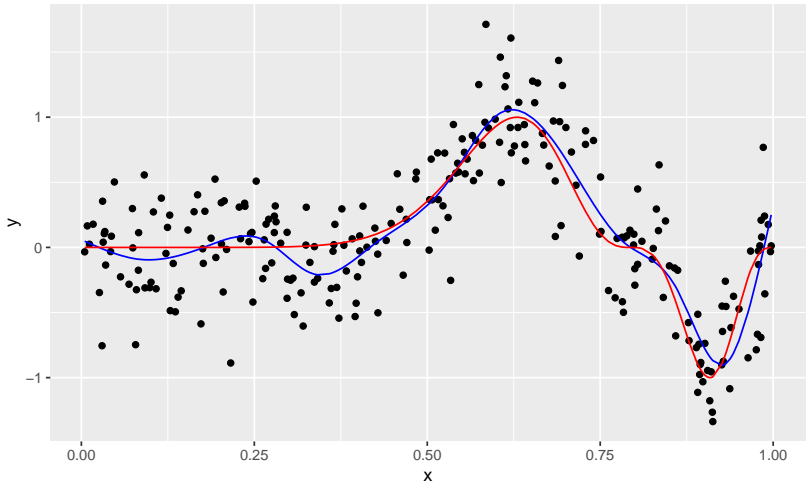
```
fit <- lm(waiting ~ ns(eruptions, df=6), faithful)
ggplot(faithful) +
  geom_point(aes(x=eruptions, y=waiting)) +
  ggtitle("Old Faithful (Natural splines, 6 df)") +
  geom_line(aes(x=eruptions, y=fitted(fit)), col='blue')
```

Old Faithful (Natural splines, 6 df)



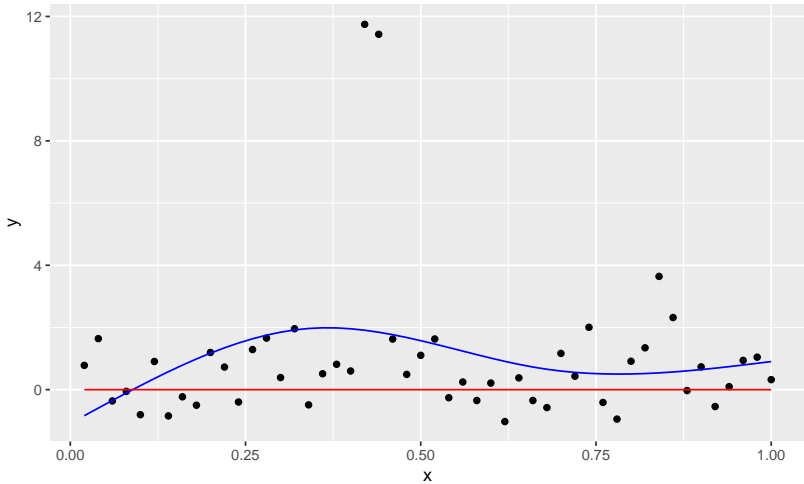
Natural splines in R

Example A (Natural splines, 12 df)



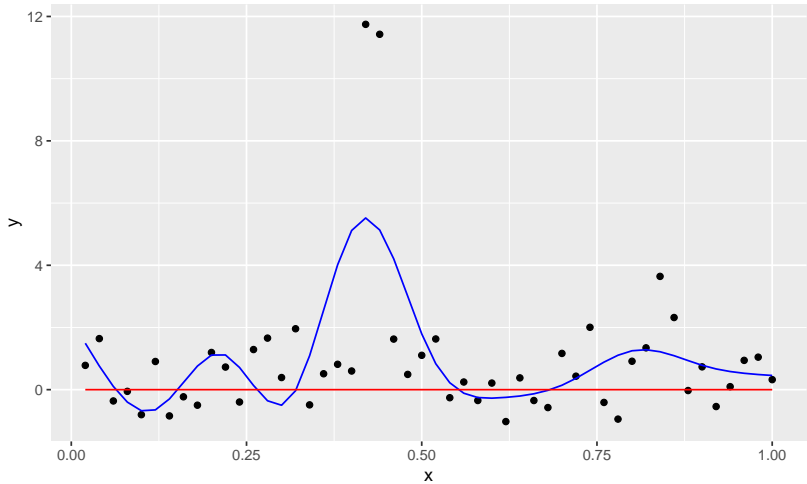
Natural splines in R

Example B (Natural splines, 3 df)



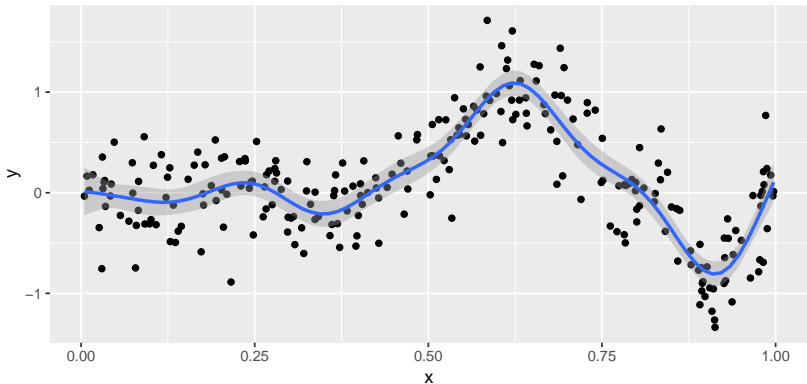
Natural splines in R

Example B (Natural splines, 10 df)



Splines and geom_smooth()

```
ggplot(exa) +  
  geom_point(aes(x=x,y=y)) +  
  geom_smooth(aes(x=x,y=y), method='gam',  
    formula = y ~ s(x,k=12))
```



Splines and `geom_smooth()`

- Because regression splines use local linear models, we can easily find standard errors for the fitted values.
- Connected together, these form a pointwise confidence band.
- Automatically produced using `geom_smooth`

Outline

- 1 Nonlinear regression
- 2 Kernel estimators
- 3 Local polynomials
- 4 Interpolating splines
- 5 Smoothing splines
- 6 Regression splines
- 7 Multivariate predictors

Multivariate predictors

$$y_i = f(\mathbf{x}_i) + \varepsilon_i, \quad \mathbf{x} \in \mathbb{R}^d$$

Most methods extend naturally to higher dimensions.

- Multivariate kernel methods
- Multivariate local quadratic surfaces
- Thin-plate splines (2-d version of smoothing splines)

Multivariate predictors

$$y_i = f(\mathbf{x}_i) + \varepsilon_i, \quad \mathbf{x} \in \mathbb{R}^d$$

Most methods extend naturally to higher dimensions.

- Multivariate kernel methods
- Multivariate local quadratic surfaces
- Thin-plate splines (2-d version of smoothing splines)

Problem

The curse of dimensionality!

Curse of dimensionality

Most data lie near the boundary

```
x <- matrix(runif(1e6,-1,1), ncol=100)
boundary <- function(z) { any(abs(z) > 0.95) }

mean(apply(x[,1,drop=FALSE], 1, boundary))
## [1] 0.0532

mean(apply(x[,1:2], 1, boundary))
## [1] 0.0985

mean(apply(x[,1:5], 1, boundary))
## [1] 0.2285
```


Curse of dimensionality

Most data lie near the boundary

```
x <- matrix(runif(1e6,-1,1), ncol=100)
boundary <- function(z) { any(abs(z) > 0.95) }

mean(apply(x[,1:10], 1, boundary))
## [1] 0.4031

mean(apply(x[,1:50], 1, boundary))
## [1] 0.9221

mean(apply(x[,1:100], 1, boundary))
## [1] 0.9933
```

Curse of dimensionality

Data are sparse

```
x <- matrix(runif(1e6,-1,1), ncol=100)
nearby <- function(z) { all(abs(z) < 0.5) }
mean(apply(x[,1,drop=FALSE], 1, nearby))
```

```
## [1] 0.4966
```

```
mean(apply(x[,1:2], 1, nearby))
```

```
## [1] 0.2397
```

```
mean(apply(x[,1:5], 1, nearby))
```

```
## [1] 0.0286
```

Curse of dimensionality

Data are sparse

```
x <- matrix(runif(1e6, -1, 1), ncol=100)
nearby <- function(z) { all(abs(z) < 0.5) }
mean(apply(x[,1:10], 1, nearby))
```

```
## [1] 7e-04
```

```
mean(apply(x[,1:50], 1, nearby))
```

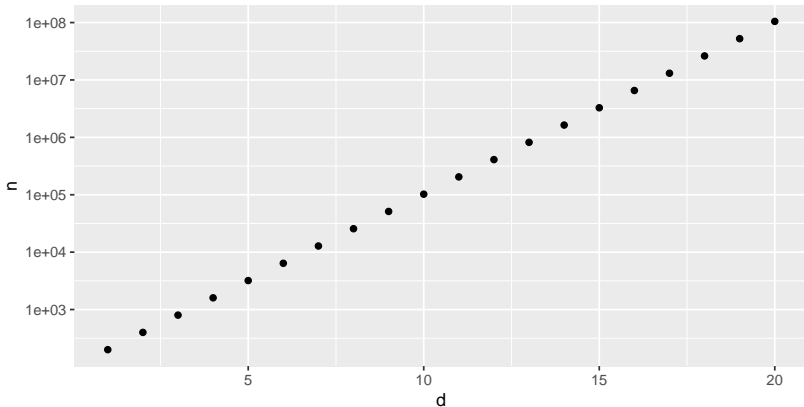
```
## [1] 0
```

```
mean(apply(x[,1:100], 1, nearby))
```

```
## [1] 0
```

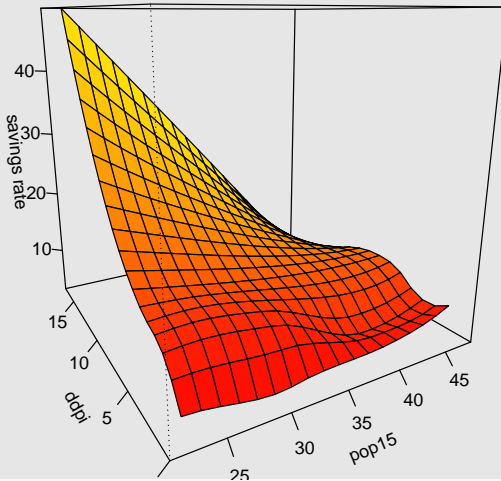
Curse of dimensionality

- Available data in a window is proportional to n^{-d} .
- Let $h = 0.5$, and suppose we need 100 observations to estimate our model locally.



Bivariate smoothing

```
lomod <- loess(sr ~ pop15 + ddpi, data=savings)
```



Bivariate smoothing

```
library(mgcv)  
smod <- gam(sr ~ s(pop15, ddpi), data=savings)
```

