Basis Functions and Local Regression

Mark M. Fredrickson (`mfredric@umich.edu`)

Computational Methods in Statistics and Data Science (Stats 406)

Review

- Expanded mean functions to include $p > 1$ variables $\mathbf{x}$:

$$Y = \mu(x_1, x_2, \ldots, x_p) + \epsilon, \quad \mathrm{E}(\epsilon) = 0$$

- Considered smoothing estimators but hit up against the **curse of dimensionality**
- Decided to force $\mu(a)$ to be a function of a single argument, so required $\eta(\mathbf{x}; \boldsymbol{\beta})$ to map p variables to a single value.
- Simplest possible case: $\mu(a) = a, \eta(\mathbf{x}; \boldsymbol{\beta}) = \mathbf{x}^T\boldsymbol{\beta}$. Implies

$$Y = \mathbf{x}^T\boldsymbol{\beta} + \epsilon$$

Review cont.

- Need to pick $\boldsymbol{\beta}$; considered loss functions $R(\beta) = \sum_{i=1}^{n} h(Y_i - \mathbf{x}^T\boldsymbol{\beta})$, such as $h_{\text{abs}}(a) = |a|$ and $h_{\text{sqd}}(a) = a^2$
- Ordinary Least Squares (OLS): minimizer of squared loss is the solution $\hat{\boldsymbol{\beta}}$ to $\mathbf{X}^T\mathbf{X}\boldsymbol{\beta} = \mathbf{X}^T\mathbf{y}$
- Interpretation and inference for $\hat{\boldsymbol{\beta}}$, finding "best" linear combination to fit the line $\mu(a) = a$.

Basis Functions

Introducing Non-Linearity

Recall we are modeling **conditional mean of Y given $\mathbf{x}$** with a linear function:

$$\mathrm{E}\left(Y \mid \mathbf{x}\right) = \mathbf{x}^T \boldsymbol{\beta}$$

What if we think **$\mathrm{E}\left(Y \mid \mathbf{x}\right)$ is not linear in $\mathbf{x}$?**

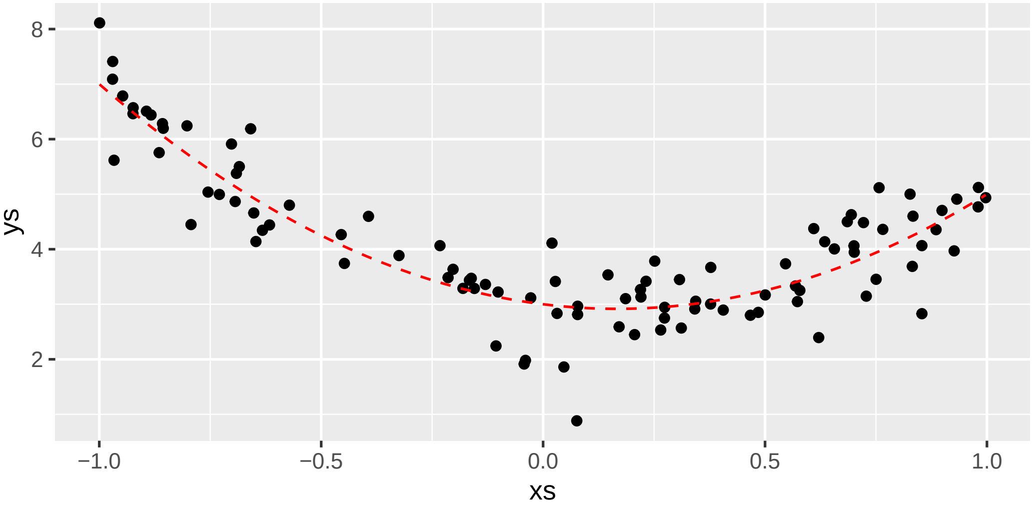
OLS can still be useful if **$\mathrm{E}\left(Y \mid \mathbf{x}\right)$ is linear in $f(\mathbf{x})$** for some function f:

$$\mathrm{E}\left(Y \mid \mathbf{x}\right) = f(\mathbf{x})^T \boldsymbol{\beta}, \quad f : \mathbb{R}^p \to \mathbb{R}^q, \quad \boldsymbol{\beta} \in \mathbb{R}^q$$

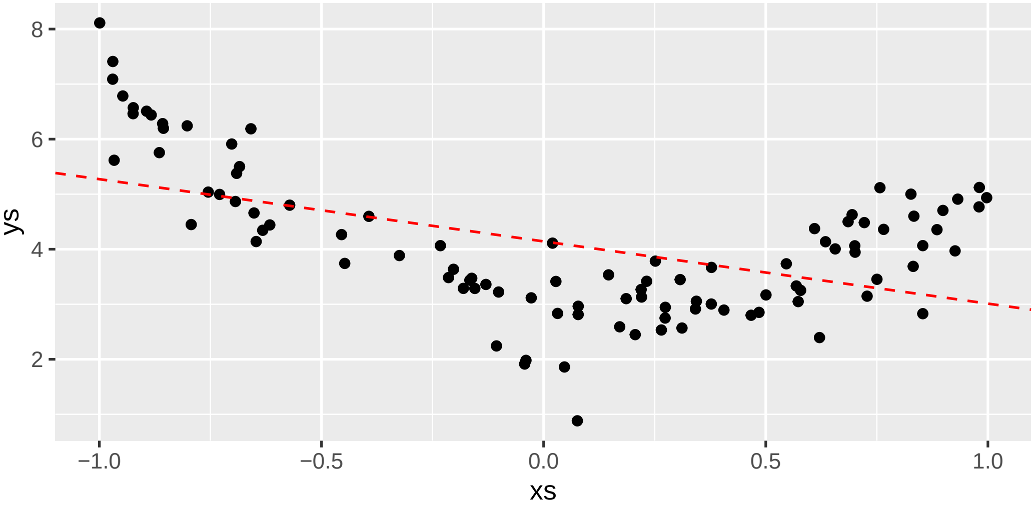
Note: We can allow $p = q$, $p < q$ or $p > q$.

Selecting f can be done on theoretical grounds. Today we'll see some generally useful f functions.

Some simulated data



Linear Fit: $\mathbf{E}(Y \mid \mathbf{x}) = \beta_0 + \beta_1 x$



Non-linear transformation

Here we have $\mathbf{x} = \begin{pmatrix} 1 & x \end{pmatrix}^T$. One way we could try to fit this model is:

$$y = f(\mathbf{x})^T \boldsymbol{\beta} + \epsilon = \begin{pmatrix} 1 & x & x^2 \end{pmatrix} \begin{pmatrix} \beta_1 \\ \beta_2 \\ \beta_3 \end{pmatrix} + \epsilon$$

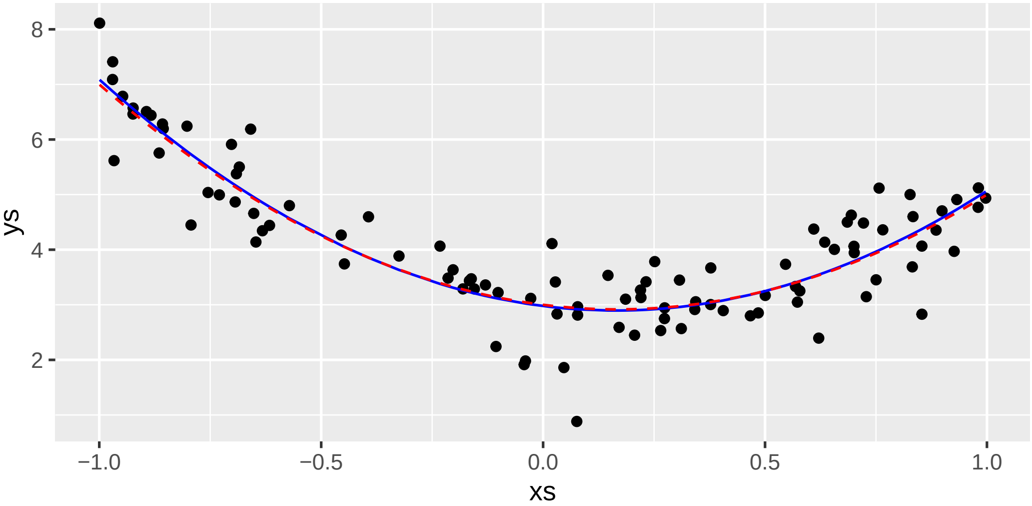
(f adds a squared term for the second component).

```
> quadmod <- lm(ys ~ xs + I(xs^2)) # intercept included automatically
> coef(quadmod)
```

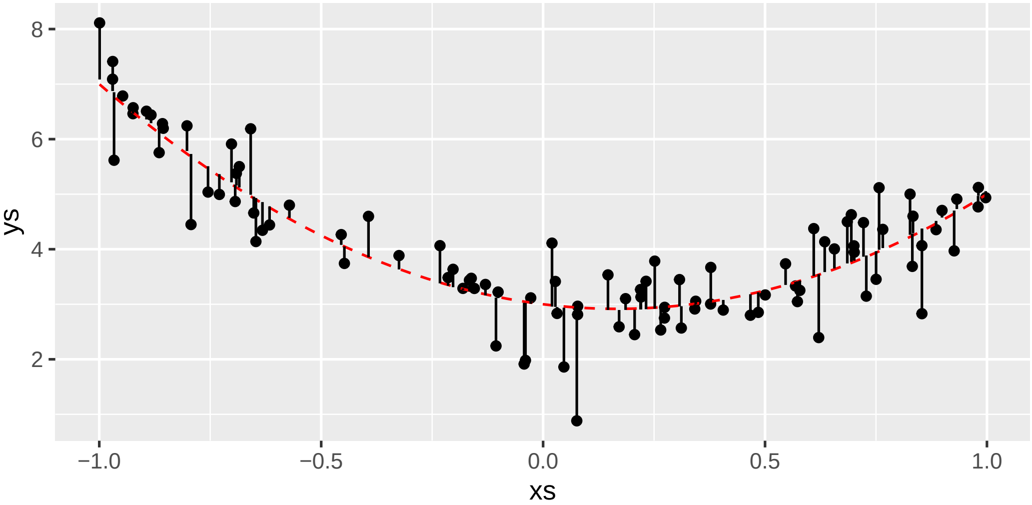
```
(Intercept)          xs     I(xs^2)
      2.978      -1.010       3.101
```

Estimated mean function

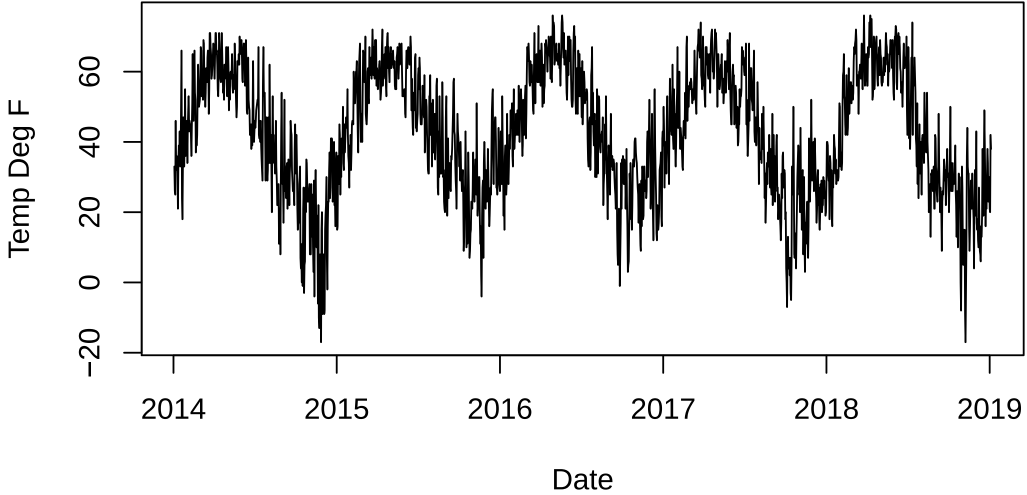
```
> estmean <- predict(quadmod)
```



Interpretation: Minimized Squared Errors to Curve



Average Temperature Data: Ann Arbor, 2014-03 to 2019-03



Picking a transformation $f(x)$

Idea 1: There are **four seasons**; Use the season to predict the temperature. Then $f(x)$ maps to a vector of length 4, with one 1 and three 0s.

```
> day_id    <- cycle(temp_ts)
> ## days start on 1 for 2014-03-29
> is_spring <- as.numeric(day_id > 3 & day_id <= 65) # April  and May
> is_summer <- as.numeric(day_id > 65 & day_id <= 157) # June to Aug
> is_fall   <- as.numeric(day_id > 157 & day_id <= 218 ) # Sept and Oct
> is_winter <- as.numeric(day_id > 218 | day_id <= 3) # Nov - Mar
```

Fitting the model

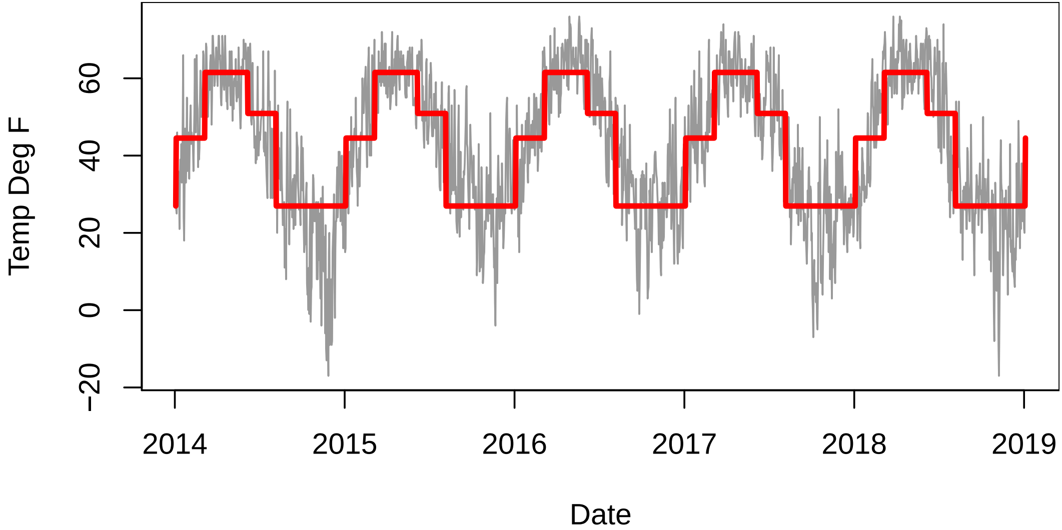
We will fit a model that is **piece-wise constant by season**:

```
> temp_pwc <- lm(weather$TOBS ~ is_winter + is_spring
+                  + is_summer + is_fall - 1) # no intercept term
> coef(temp_pwc)

is_winter is_spring is_summer   is_fall
    26.96     44.53     61.52     50.90
```

This is equivalent to finding the average response within group:

```
> mean(weather$TOBS[is_winter == 1])

[1] 26.96
```

Plotting Piece-wise Constant $\mu(x)$



Idea 2: Angle to the Sun

Since seasonal variation is due to **the Earth's axial tilt**, perhaps we can think about the angle to the sun at every time point.

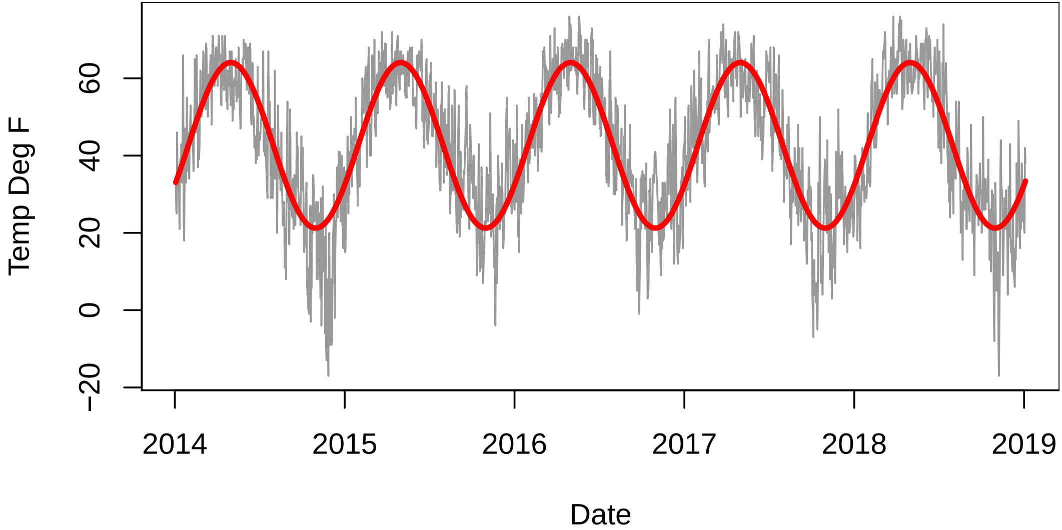
Since our period is 365 days (or so), we can transform x into:

$$f(x) = \sin\left(2\pi \frac{x-a}{365}\right)$$

where a is picked to make $x - a = 0$ around the mid point of the temperature series ($a = 30$).

```
> temp_sin <- lm(weather$TOBS ~ I(sin(2 * pi * (day_id - 30) / 365)))
```

Plotting Sinusoidal $\mu(x)$



Basis Functions

Recall in the quadratic mean function example we wrote:

$$\mu(x) = \beta_1 + \beta_2 x + \beta_3 x^2 = \sum_{j=1}^{j} \beta_j x^j$$

In the first weather example, we used indicator functions for season ($x \in \{1, 2, 3, 4\}$):

$$\mu(x) = \sum_{j=1}^{4} \beta_j I(x = j)$$

Both of these are examples of **basis functions**: writing $\mu(x)$ as a linear combination of functions evaluated at x.

$$\mu(x) = \sum_{j=1}^{k} \beta_j f_j(x)$$

(we'll focus on finite or truncated basis functions with $k < \infty$)

More on Basis Functions

In the following, suppose have the following two conditions:

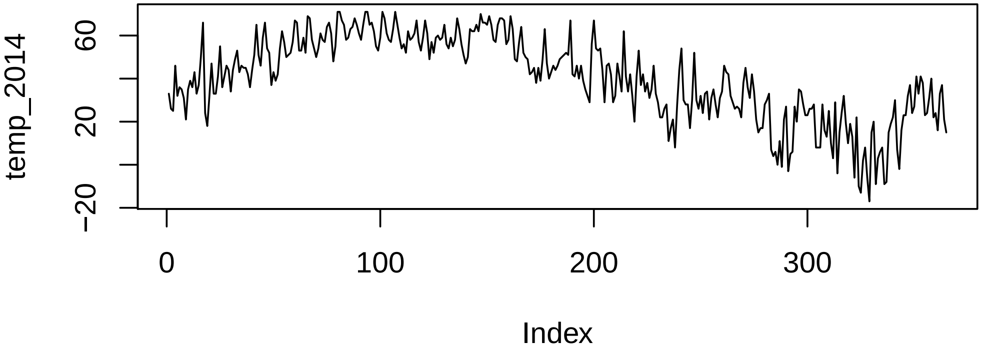
- $x \in [0, 1]$ (we can always scale $x' = (x - x_{\text{min}})/(x_{\text{max}} - x_{\text{min}})$)
- $y \geq 0$ (we can always add an intercept so that $y' = y - y_{\text{min}}$)

Typically, we will pick the **class of functions f_j** and the **order of the basis k**. For example, the **monomial basis functions**:

$$f_j(x) = x^j$$

Using OLS, we can then fit appropriate parameters (as always, under the squared error loss function).

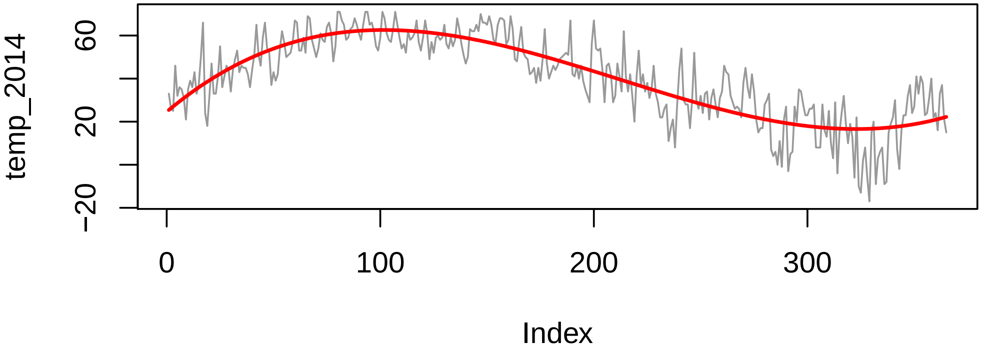
Ann Arbor Temperature, March 2014 to March 2015



3rd Order Monomial Basis

```
> x <- 0:364 / 364
> monoMod <- lm(temp_2014 ~ 1 + x + I(x^2) + I(x^3))
> coef(monoMod)

(Intercept)           x      I(x^2)      I(x^3)
       25.4       300.2      -712.8       409.4
```

Ann Arbor Temperature, March 2014 to March 2015



Splines

When we did the first investigation for the weather data, we

- Reduced the complete time to just "day of the year"
- Chopped up a year into four seasons. Let the start of season j be s_j and the end be $s_{j+1} - 1$
- Created basis functions $f_j(x) = I(s_j \leq x)I(x \leq s_{j+1} - 1)$

In general, we call the locations of the breaks **knots**. When we pick polynomials as the basis, functions, we call the overall approach **a spline** (term comes from fitting curved pieces in woodworking).

There was no particular need for the hard breaks at the knot locations. Could have the basis functions **span the knots** in some way.

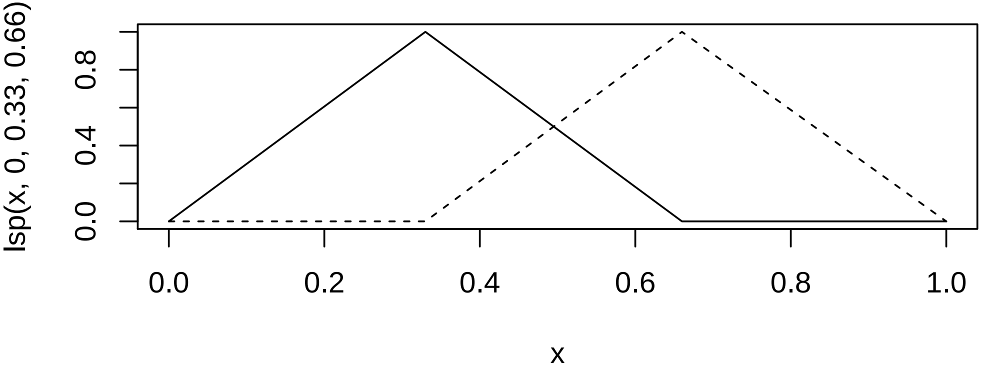
For example, suppose we picked knots $v_1, \ldots, v_k$ (and augment with $v_0 = 0, v_{k+1} = 1$. Then we could use **linear basis functions**:

$$f_j(x) = \begin{cases} (x - v_{j-1})/(v_j - v_{j-1}) & : v_{j-1} < x \leq v_j \\ (v_{j+1} - x)/(v_{j+1} - v_j) & : v_j < x \leq v_{j+1} \\ 0 & : \text{otherwise} \end{cases}$$

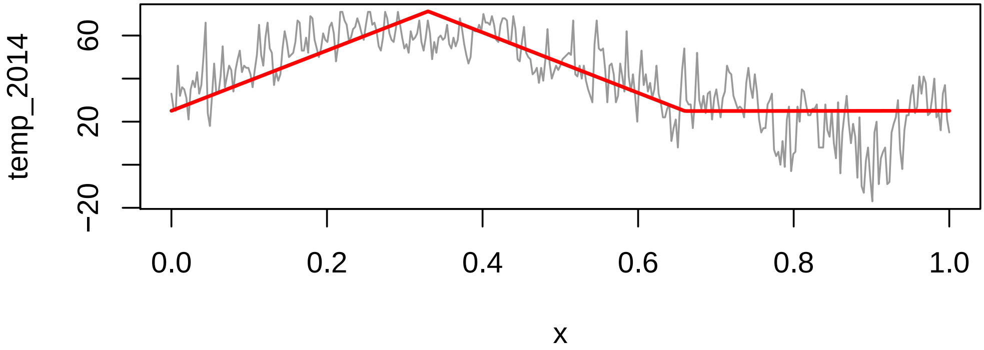
Linear spline, 2nd order

```
> lsp <- function(x, v1, v2, v3) {
+     slopes <- ifelse(x < v2,
+     (x >= v1) * (x - v1) / (v2 - v1),
+     (x <= v3) * (v3 - x) / (v3 - v2))
+ }
> b1 <- lsp(x, 0, 0.33, 0.66)
> b2 <- lsp(x, 0.33, 0.66, 1)
> lspmod <- lm(temp_2014 ~ b1 + b2)
```

Piece-wise Linear Functions



Fit



Estimated Coefficients

```
> summary(lspmod)

Call:
lm(formula = temp_2014 ~ b1 + b2)

Residuals:
   Min      1Q  Median      3Q     Max
-42.02  -6.89   1.05   7.47  34.79

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)   25.055      1.368   18.32   <2e-16 ***
b1            46.195      2.129   21.70   <2e-16 ***
b2            -0.114      2.129   -0.05     0.96
---
```

B-splines and Natural Splines

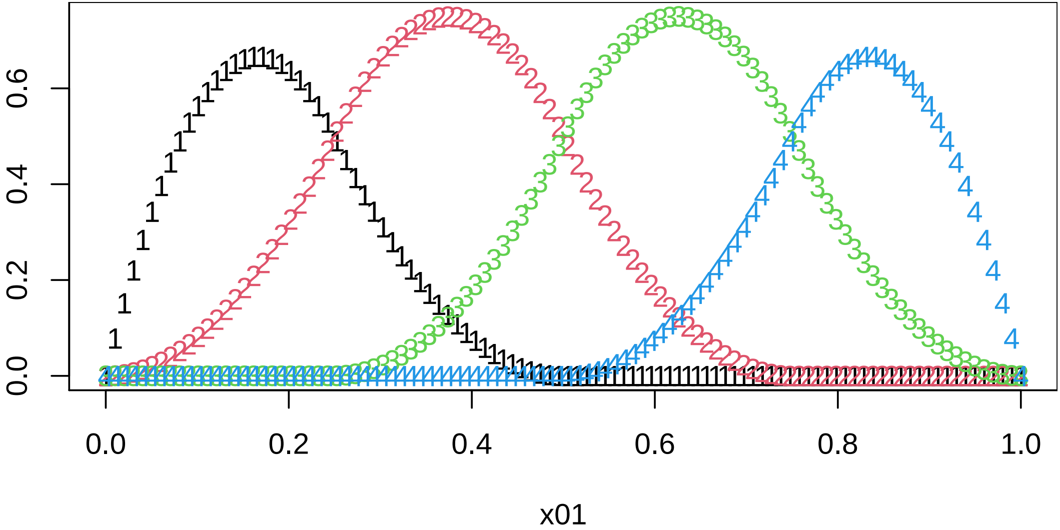
In previous graph there were **clear changes at the knots** as the basis functions weren't required to smoothly connect.

Adding **smoothness requirements** (i.e., constraints on the second derivatives) and constraints that the **functions meet at the knots**, requires using **third order polynomials**.

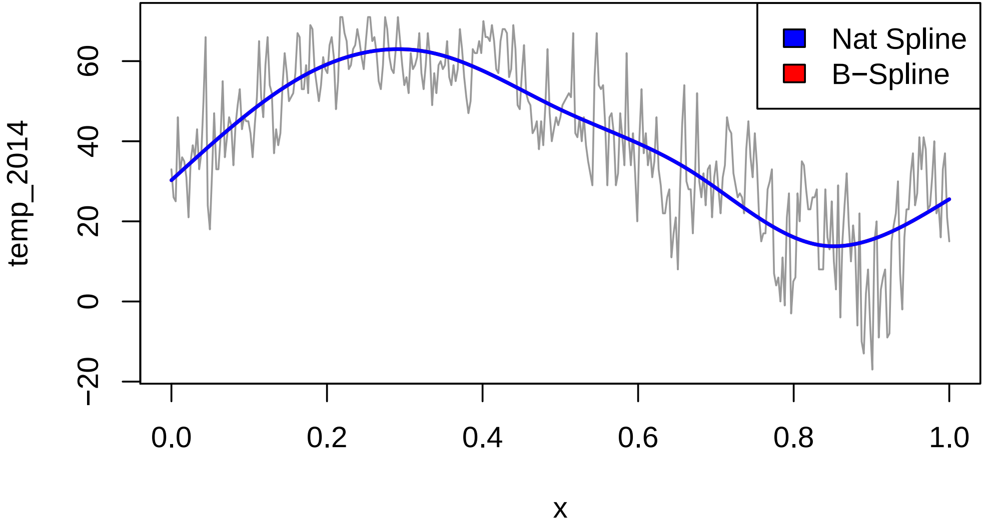
Such basis functions are called **b-splines** (of a given order). If we add the requirement that the **functions go to zero linearly** outside of $[0, 1]$, they are called **natural splines**.

Practical differences are small.

Three (interior) knots, cubic polynomials for $x \in (0, 1)$



```
> library(splines)
> temp_ns <- lm(temp_2014 ~ ns(x, 6)) # "natrual" (goes to zero)
> (temp_bs <- lm(temp_2014 ~ bs(x, 6))) # less constrained

Call:
lm(formula = temp_2014 ~ bs(x, 6))

Coefficients:
(Intercept)   bs(x, 6)1   bs(x, 6)2   bs(x, 6)3
      30.91       12.86       44.73       14.93
  bs(x, 6)4   bs(x, 6)5   bs(x, 6)6
      -2.73      -32.63        3.15
```


Nat Spline
B−Spline
temp_2014
−20
0
20
40
60
0.0
0.2
0.4
0.6
0.8
1.0
x

Notes on Basis Functions

- The basis function **decomposes $\mu(x)$ into linear combination of functions $f_j(x)$**
- The **coefficients can be selected by OLS** (i.e., they minimized squared error for y).
- **Picking order and knot location** can be done via **cross validation**.
- High orders tend to be "wiggly" and chase noise, particularly at the boundary of x (Runge's phenonmenon)
- Other kinds of basis functions exist:
  - Orthogonal polynomials where $\int f_i(x) f_j(x)\, dx = 0$ for all i and j.
  - Harmonic series of sin and cos
- Difficult to interpret parameters. Changes in y depend on more than $x_1 - x_0$ (depend on values of x_0 and x_1)

Local Regression

Linear Estimators

A useful class of estimators of the conditional mean are **linear estimators** of $\mathrm{E}(Y \mid x)$:

$$\hat{\mu}(x) = \sum_{i=1}^{n} l_i(x) Y_i$$

Some examples include:

- The **sample mean** of Y: $l_i(x) = 1/n$
- **Ordinary Least Squares**: Recall $\hat{\boldsymbol{\beta}} = \left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T\mathbf{y} = \mathbf{P}^T\mathbf{y}$,

$$\hat{\mu}(\mathbf{x}) = \mathbf{x}^T\hat{\boldsymbol{\beta}} = \mathbf{x}^T\mathbf{P}^T\mathbf{y} = \sum_{i=1}^{n} \mathbf{x}^T\mathbf{p_i}\, y_i$$

- Nadaraya-Watson **kernel smoothing estimator**:

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

Statistical Properties of Linear Estimators: Bias

Recall our model:

$$Y_i = \mu(x_i) + \epsilon_i, \quad x_i \perp \epsilon_i, \mathrm{E}(\epsilon_i) = 0$$

Then the **bias** of a linear smoother **at the point x** is:

$$\begin{aligned}
\mathrm{E}(\hat{\mu}(x)) - \mu(x)) &= \mathrm{E}(\hat{\mu}(x)) - \mu(x) \\
&= \mathrm{E}\left(\sum_{i=1}^{n} l_i(x) Y_i\right) - \mu(x) \\
&= \sum_{i=1}^{n} l_i(x) \mathrm{E}(Y_i \mid x_i) - \mu(x) \\
&= \sum_{i=1}^{n} l_i(x) \mu(x_i) - \mu(x)
\end{aligned}$$

Of course, **we don't know $\mu(x_i)$ or $\mu(x)$**, but we can still reason about when the bias will be small.

Taylor Expansion

Recall (from a previous calculus class perhaps) that a **Taylor's expansion** approximates $f(x)$ using the series

$$f(x) = \sum_{n=0}^{\infty} (x - x^*)^n \frac{f^{(n)}(x^*)}{n!}$$

- $f^{(n)}$ is the nth derivative (f needs to be continuously differentiable at x^*)
- x^* is some other point, often one for which we know $f(x^*)$ and/or its derivatives
- $n!$ gets large fast, so a good approximation can be made from only a few terms

Taylor expansion for $\mu(x_i)$ at x

We want to approximate:

$$E(\hat{\mu}(x)) - \mu(x) = \sum_{i=1}^{n} l_i(x)\mu(x_i) - \mu(x)$$

The **first order Taylor approximation** for $\mu(x_i)$:

$$\mu(x_i) \approx \mu(x) + (x_i - x)\mu'(x)$$

$$E(\hat{\mu}(x)) - \mu(x) \approx \mu(x)\left(\sum_{i=1}^{n} l_i(x) - 1\right) + \mu'(x)\left(\sum_{i=1}^{n}(x_i - x)l_i(x)\right)$$

Minimizing bias

From the last slide:

$$E(\hat{\mu}(x)) - \mu(x) \approx \mu(x)\left(\sum_{i=1}^{n} l_i(x) - 1\right) + \mu'(x)\left(\sum_{i=1}^{n}(x_i - x)l_i(x)\right)$$

In general, we don't know the true $\mu(x)$ or $\mu'(x)$, but we can construct $\sum_{i=1}^{n} l_i(x) = 1$.

E.g., for the Nardaraya-Watson estimator,

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

Some estimators (linear regression, local linear regression later) have $\sum_{i=1}^{n}(x_i - x)l_i(x) = 0$.

Brief Aside: Adding Weights

Suppose for each observation we had a **weight** $w_i > 0$, indicating the importance of the ith observation.

It may be natural **to pay more cost to certain observations** when computing the **loss function**. For example,

$$R(\beta) = \sum_{i=1}^{n} w_i (y_i - \mathbf{x}^T \beta)^2$$

or

$$R(\beta) = \mathbf{W}(\mathbf{y} - \mathbf{X}\beta)^T (\mathbf{y} - \mathbf{X}\beta)$$

where $\mathbf{W}$ $(n \times n)$ is a matrix with w_i on the diagonal and zero elsewhere.

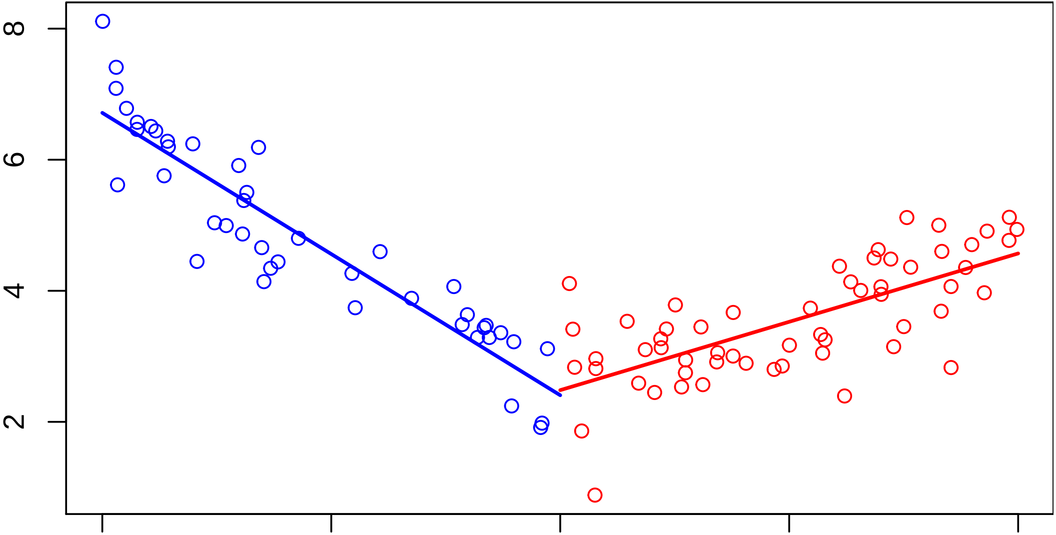
Since $\mathbf{W}$ has the structure $\mathbf{W_{ii}} = w_i$, we can factor it as $\mathbf{W}^{1/2}\mathbf{W}^{1/2}$ ($\sqrt{w_i}$ on the diag) and $\mathbf{W}^{1/2} = \left(\mathbf{W}^{1/2}\right)^T$.

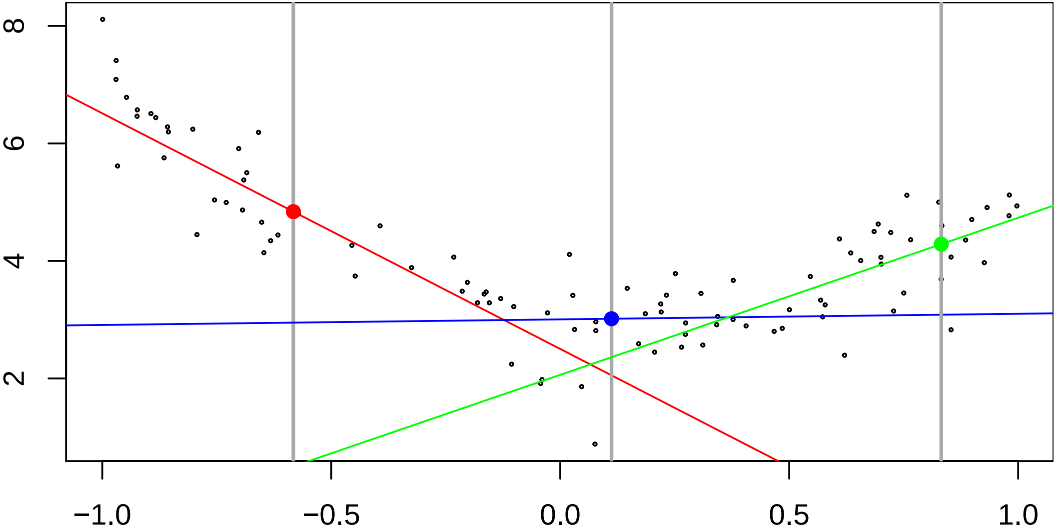
Then

$$\begin{aligned} R(\boldsymbol{\beta}) &= (\mathbf{W}^{1/2}\mathbf{W}^{1/2})(\mathbf{y} - \mathbf{X}\beta)^T(\mathbf{y} - \mathbf{X}\beta) \\ &= \mathbf{W}^{1/2}(\mathbf{y} - \mathbf{X}\beta)^T\mathbf{W}^{1/2}(\mathbf{y} - \mathbf{X}\beta) \\ &= (\mathbf{W}^{1/2}\mathbf{y} - \mathbf{W}^{1/2}\mathbf{X}\beta)^T(\mathbf{W}^{1/2}\mathbf{y} - \mathbf{W}^{1/2}\mathbf{X}\beta) \end{aligned}$$

which is OLS on $\tilde{\mathbf{y}} = \mathbf{W}^{1/2}\mathbf{y}$ and $\tilde{\mathbf{X}} = \mathbf{W}^{1/2}\mathbf{X}$.

Two fits (simulated data)




2
4
6
8
−1.0
−0.5
0.0
0.5
1.0

Local linear fits

To make a linear fit **about the point x**, the slope of the line would not change if we replaced

$$\tilde{x}_i = x_i - x$$

The **weighted least squares** fit would minimize:

$$\sum_{i=1}^{n} w_i (Y_i - (a_0 + a_1 \tilde{x}_i))^2$$

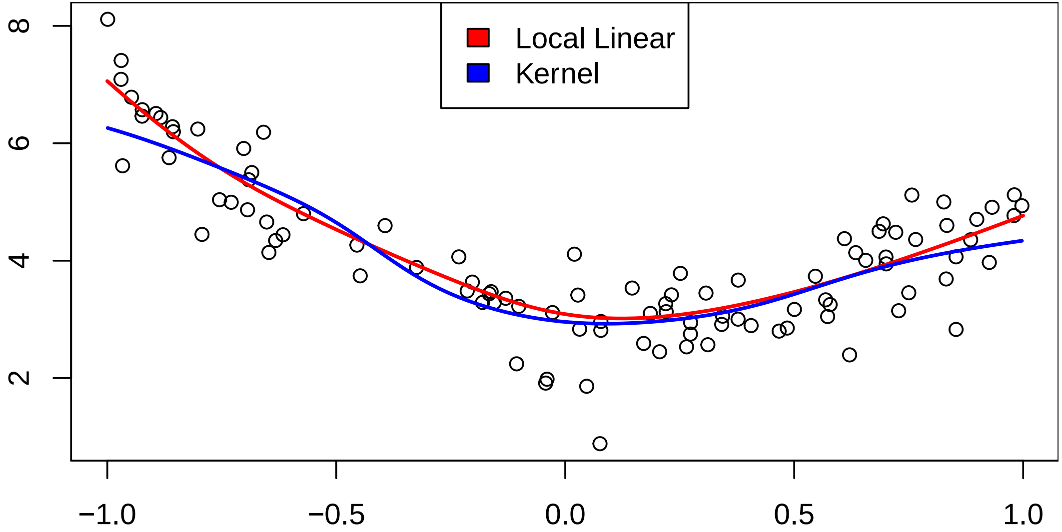
where $w_i = K\left(\frac{x_i - x}{h}\right)$ and K is a kernel function. If we find $\hat{a}_0$ and $\hat{a}_1$, then

$$\hat{\mu}(x - x) = \hat{a}_0$$

Implementation

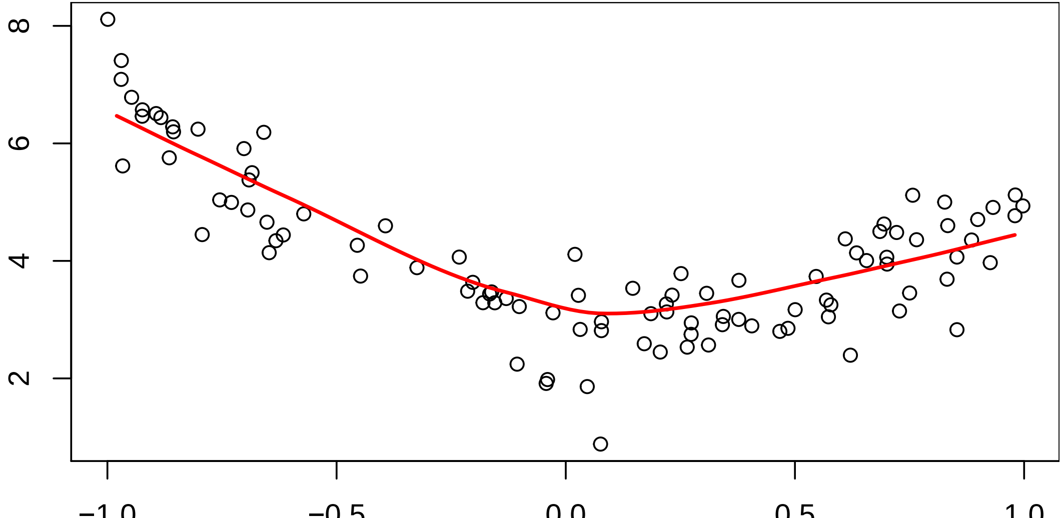
We'll use the Gaussian distribution as our weight function with $h = 0.25$

```
> grid <- seq(-1, 1, length.out = 100)
> a0s <- sapply(grid, function (g) {
+   ws <- dnorm((xs - g) / 0.25)
+   fit <- lm(ys ~ I(xs - g), weights = ws)
+   return(coef(fit)[1])
+ })
```


Local Linear
Kernel
8
6
4
2
−1.0
−0.5
0.0
0.5
1.0

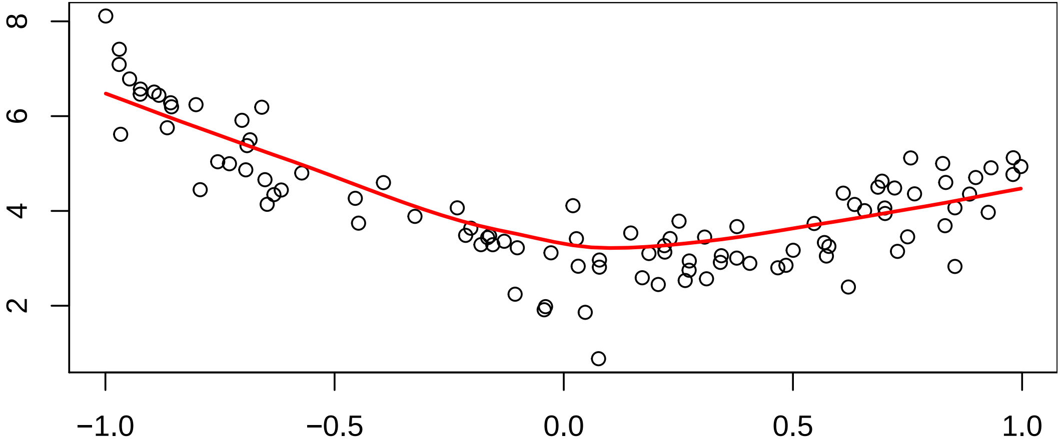
Implementation in R using `loess`

```
> ll <- loess(ys ~ xs, degree = 1, span = 0.75)
```



Implementation in R using `scatter.smooth`

```
> par(mar = c(2, 2, 0, 0))
> scatter.smooth(ys ~ xs, degree = 1, span = 0.75,
+                lpars = list(col = "red", lwd = 2))
```



Higher degree polynomial fits

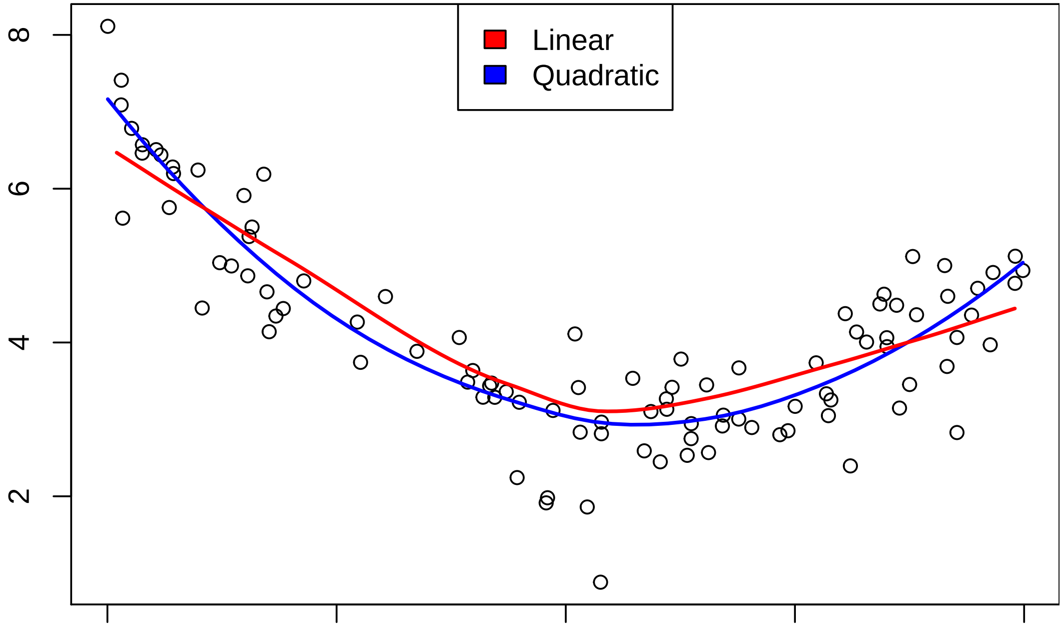
We can extend our approximation for $\mu(x_i)$ with a **quadratic term**:

$$\mu(x_i) \approx a_0 + a_1(x_i - x) + a_2 \frac{(x_i - x)^2}{2}$$

Instead of fitting a line at each point, we are fitting a parabola.

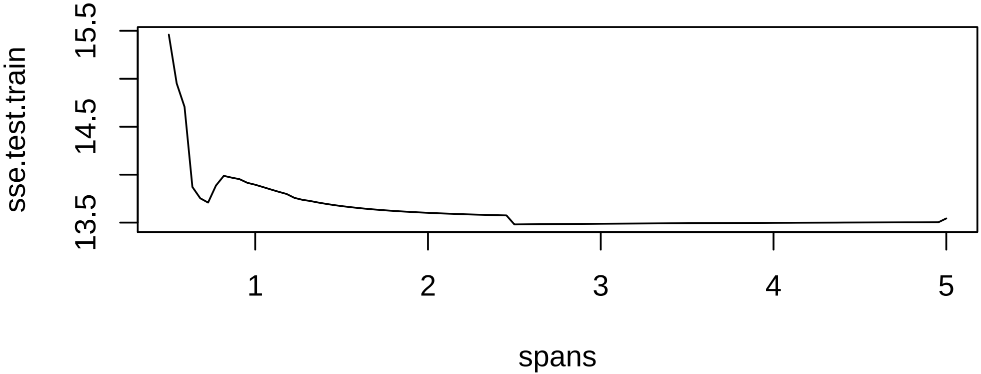
We could implement this ourselves, but let's just use `loess`'s `degree` parameter:

```
> ll2 <- loess(ys ~ xs, degree = 2, span = 0.75)
```


Linear
Quadratic
8
6
4
2

Picking loess parameters using train-test

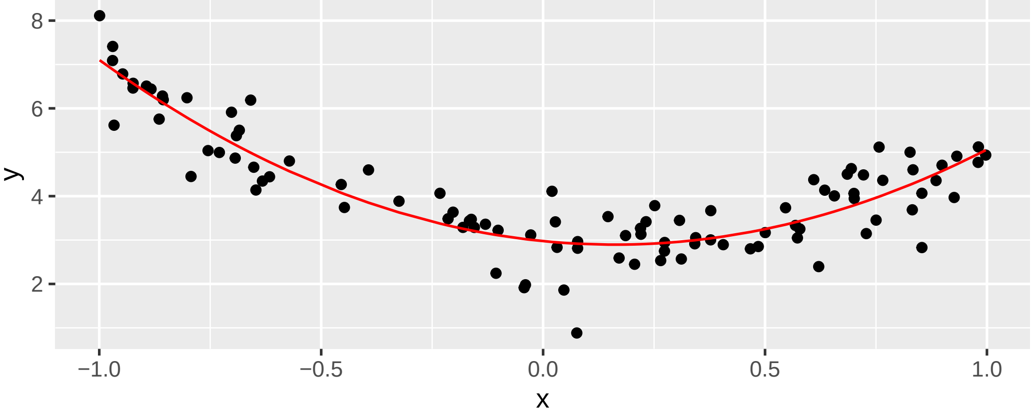
```
> spans <- seq(0.5, 5, length.out = 100)
> train_ids <- sample.int(length(xs), size = round(length(xs) / 2))
> training <- data.frame(y = ys[train_ids], x = xs[train_ids] )
> testing  <- data.frame(y = ys[-train_ids], x = xs[-train_ids] )
> sse.test.train <- sapply(spans, function(s) {
+     mod <- loess(y ~ x, training, span = s)
+     preds <- predict(mod, newdata = testing)
+     sum((preds - testing$y)^2, na.rm = TRUE)
+ })
```



sse.test.train
13.5
14.5
15.5
1
2
3
4
5
spans

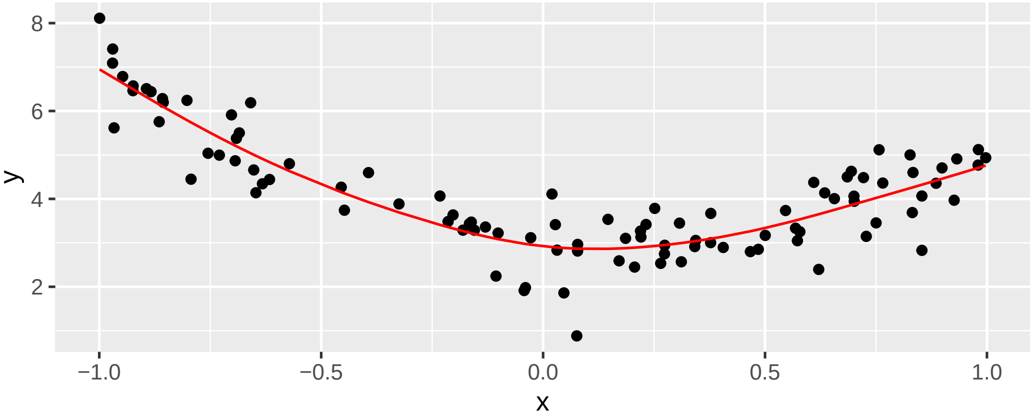
Fit all data at best span

```
> d$pred <- predict(loess(y ~ x, d,
+   span = spans[which.min(sse.test.train)]))
```



Built in CV

```
> u <- smooth.spline(d$x, d$y, cv = TRUE) # does leave one out
> d$ss <- u$y[match(d$x, u$x)]
```



Local regression summary

- For any point x, we estimate $\hat{\mu}(x)$ using a **weighted regression**, where weights come from the **kernel weights** $K((x_i - x)/h)$.
- Can be **combined with other basis methods**, such as quadratic fits or splines.
- **Local linear** has **better bias properties** than kernel estimators.
- **Computational cost** of fitting a regression at each point x.
- Still suffers from **curse of dimensionality** and **lack of data summarizing**.
- Need to pick **smoothing parameters**, usually with CV.