Module 7: Moving Beyond Linearity

TMA4268 Statistical learning

Andreas Strand

24 februar, 2019

Contents

Introduction

In this module we will make modifications to the regression models explored previously. We will consider six variations. We need these models when a straight line is not an accurate description.

- Polynomial regression uses powers of the original predictor.
- Step functions are piece-wise constant.
- Regression splines are regional polynomials joined smoothly.
- Smoothing splines are smooth functions.
- Local regressions are splines with overlapping regions.
- Additive models combines models.

Basis Functions

Previously, we encountered the multiple linear regression model

$$y_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \dots \beta_k x_{ik} + \varepsilon_i,$$

or equivalently

$$\mathbf{y} = \mathbf{X}\beta + \varepsilon,$$

where $\mathbf{y} = (y_1, y_2, \dots, y_n)^\mathsf{T}$, $\beta = (\beta_1, \beta_2, \dots, \beta_k)^\mathsf{T}$ and $\varepsilon = (\varepsilon_1, \varepsilon_2, \dots, \varepsilon_n)^\mathsf{T}$. This model suggets that the expected response is a linear combination of some explanatory variables $X_1, X_2, \dots, X_k$. We call $\mathbf{X}$ the design matrix. Each row of the design matrix represents an observation $i \in \{1, \dots n\}$. Each column represents an explanatory variable. Thus, the design matrix is

$$\mathbf{X} = \begin{pmatrix} 1 & x_{11} & x_{12} & \dots & x_{1k} \\ 1 & x_{21} & x_{22} & \dots & x_{2k} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n1} & x_{n2} & \dots & x_{nk} \end{pmatrix}.$$

The OLS estimator for β is

$$\hat{\beta} = (\mathbf{X}^\mathsf{T}\mathbf{X})^{-1}\mathbf{X}^\mathsf{T}\mathbf{y}.$$

The design matrix defines our regression. The one defined above is the special case for MLR, and we will see other variations. Luckily, the estimator $\hat{\beta}$ can still be used when changing $\mathbf{X}$.

We know that the response sometimes can be related to the square of an explanatory variable. In order to allow for this, we need to change the model. Let us focus on **one explanatory variable** X for now. Then, instead of x_{i1}, we just write x_i. Some possible models are

$$\begin{aligned} y_i &= \beta_0 + \beta_1 x_i + \varepsilon_i, \\ y_i &= \beta_0 + \beta_1 x_i^2 + \varepsilon_i, \\ y_i &= \beta_0 + \beta_1 x_i + \beta_2 x_i^2 + \varepsilon_i \end{aligned}$$

The first model suggests that Y and X have a linear relationship, and the second model suggests a quadratic one. The third version explains Y as a linear combination of both X and X^2. The square X^2 is one of many transformations we can apply to X. We call these transformations *basis functions* $b_1(X), b_2(X), \dots b_k(X)$. For any choice of basis functions we can fit the model

$$y_i = \beta_0 + \beta_1 b_1(x_i) + \beta_2 b_2(x_i) + \dots \beta_k b_k(x_i) + \varepsilon_i.$$

The basis functions are fixed functions of the known values x_i. The corresponding design matrix is

$$\mathbf{X} = \begin{pmatrix} 1 & b_1(x_1) & b_2(x_1) & \dots & b_k(x_1) \\ 1 & b_1(x_2) & b_2(x_2) & \dots & b_k(x_2) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & b_1(x_n) & b_2(x_n) & \dots & b_k(x_n) \end{pmatrix}.$$

Thus, each row represents an observation and each column a basis function. An intercept is included as the first column. It is convenient to let k denote the number of basis functions since the design matrix will have dimensions $n \times (k+1)$. This is the same dimensions as with MLR. The design matrix is fixed and merely a function of $x_1, x_2, \dots, x_n$.

Consider the model with X and X^2. In basis function notation we write $b_1(X) = X$ and $b_2(X) = X^2$. Let a realization be $\mathbf{x} = (6, 3, 6, 8)^\mathsf{T}$ and $\mathbf{y} = (3, -2, 5, 10)^\mathsf{T}$. This results in

$$\mathbf{X} = \begin{pmatrix} 1 & b_1(x_1) & b_2(x_1) \\ 1 & b_1(x_2) & b_2(x_2) \\ 1 & b_1(x_3) & b_2(x_3) \\ 1 & b_1(x_4) & b_2(x_4) \end{pmatrix} = \begin{pmatrix} 1 & x_1 & x_1^2 \\ 1 & x_2 & x_2^2 \\ 1 & x_3 & x_3^2 \\ 1 & x_4 & x_4^2 \end{pmatrix} = \begin{pmatrix} 1 & 6 & 36 \\ 1 & 3 & 9 \\ 1 & 6 & 36 \\ 1 & 8 & 64 \end{pmatrix}$$

and

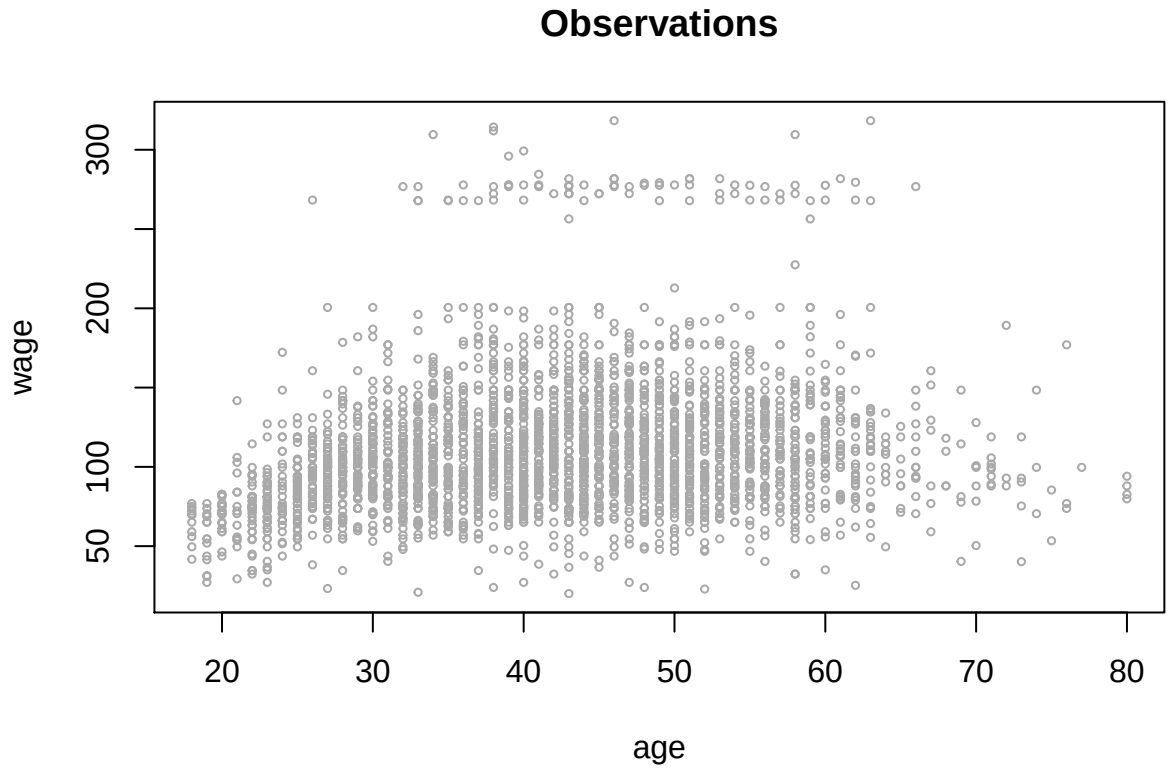
$$\hat{\beta} = (\mathbf{X}^\mathsf{T}\mathbf{X})^{-1}\mathbf{X}^\mathsf{T}\mathbf{y} = (-4.4, 0.2, 0.2)^\mathsf{T}.$$

Predictions

The models presented in this module have coefficients that we estimate. This is done efficiently by R functions. Furthermore, predicted responses are usually provided by `predict()`. Finally, we show the result with `plot()`. For convenience, we will combine the steps in our own R function `Plot()`.

We will focus on the data set with `wage` explained by one characteristic, namely `age`.

```
library(ISLR)
attach(Wage)
plot(age, wage, cex = .5, col = "darkgray", main = "Observations")
```



Each method is a way of drawing a line through these points. We can perform most methods by `lm(wage ~ X)`. We choose the design matrix $\mathbf{X}$ according to the desired method.

Polynomial Regression

The polynomial regression includes powers of X in the regression. This is

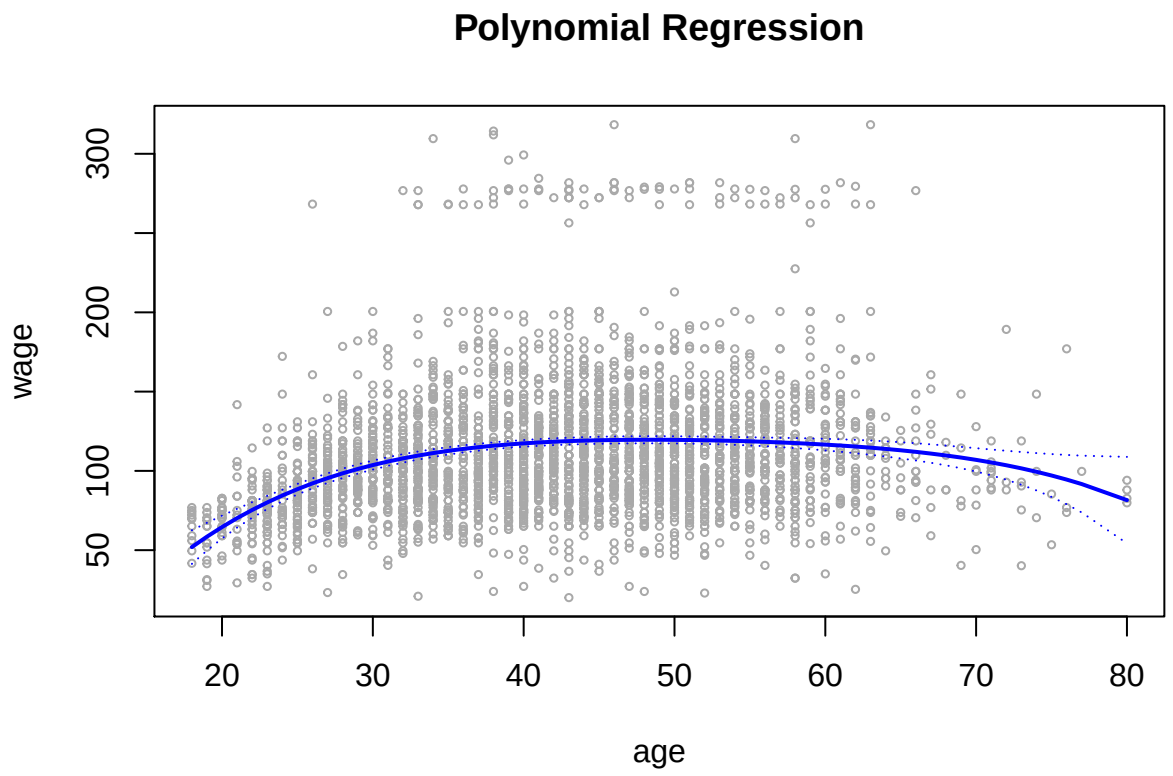
$$y_i = \beta_0 + \beta_1 x_i + \beta_2 x_i^2 + \dots \beta_d x_i^d + \varepsilon_i,$$

where d is the degree. The degree is usually no greater $d = 4$ as higher degree polynomials can lead to wild fits. The simplest choice of basis functions are $b_j(x_i) = x_i^j$ for $j = 1, 2, \dots, d$. Thus, the design matrix is

$$\mathbf{X} = \begin{pmatrix} 1 & x_1 & x_1^2 & \dots & x_1^d \\ 1 & x_2 & x_2^2 & \dots & x_2^d \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & x_n^2 & \dots & x_n^d \end{pmatrix}.$$

In R we can construct a design matrix for `age` using `poly(age)`. The default in this funciton is to return an orthogonal version of the matrix above. This makes no difference in the predictions, but makes $\mathbf{X}^\intercal\mathbf{X}$ diagonal, which is convenient for estimating coefficients. A polynomial regression with `age` of degree 4 is below.

```
fit = lm(wage ~ poly(age,4))
Plot(fit, main = "Polynomial Regression")
```



Step Functions

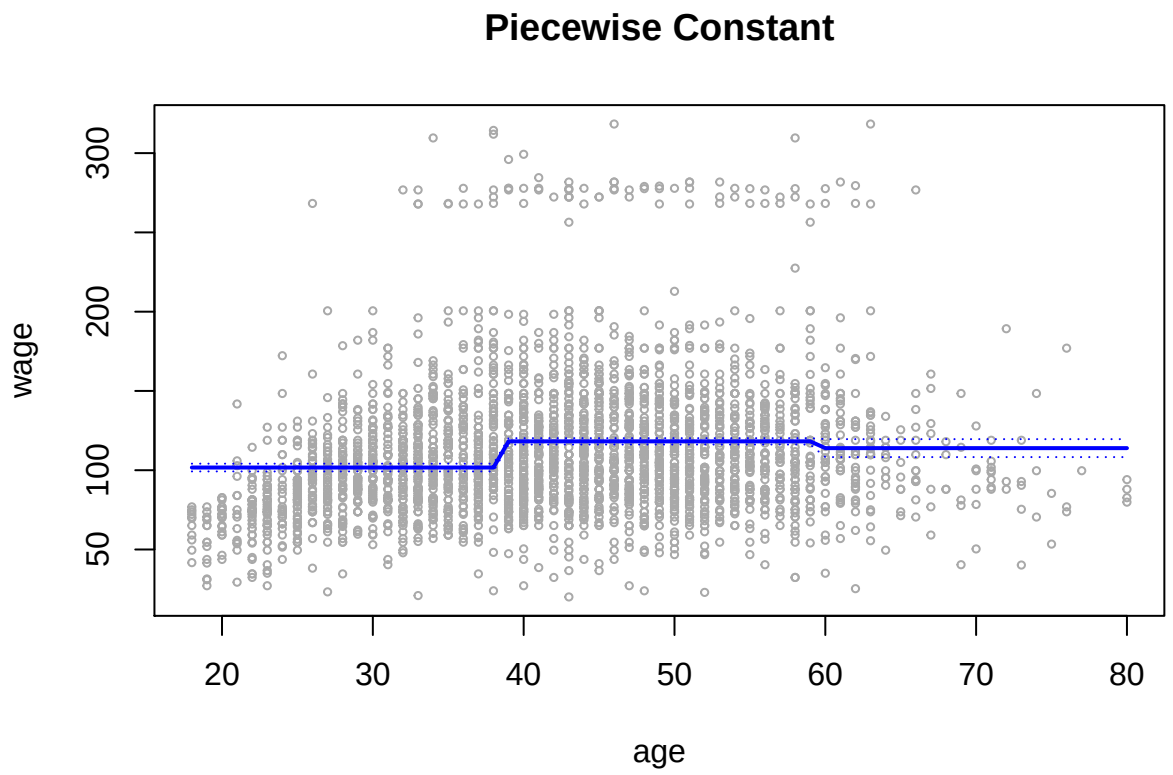
Step-functions can be used to divide `age` into bins. We model `wage` as a constant in each bin. The basis functions are now indicator functions for what bin x_i belongs to. Cutpoints $c_1, c_2, \dots, c_K$ define the bins. The

basis functions becomes $b_j(x_i) = I(c_j \leq x_i < c_{j+1})$ for $j = 1, 2, \ldots, K-1$. The last bin is $b_K(x_i) = I(c_K \leq x_i)$. The intercept is the expected value of `wage` in the first bin, when $x_i < c_1$. The design matrix is now

$$\mathbf{X} = \begin{pmatrix} 1 & I(c_1 \leq x_1 < c_2) & I(c_2 \leq x_1 < c_3) & \ldots & I(c_K \leq x_1) \\ 1 & I(c_1 \leq x_2 < c_2) & I(c_2 \leq x_2 < c_3) & \ldots & I(c_K \leq x_2) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & I(c_1 \leq x_n < c_2) & I(c_2 \leq x_n < c_3) & \ldots & I(c_K \leq x_n) \end{pmatrix}.$$

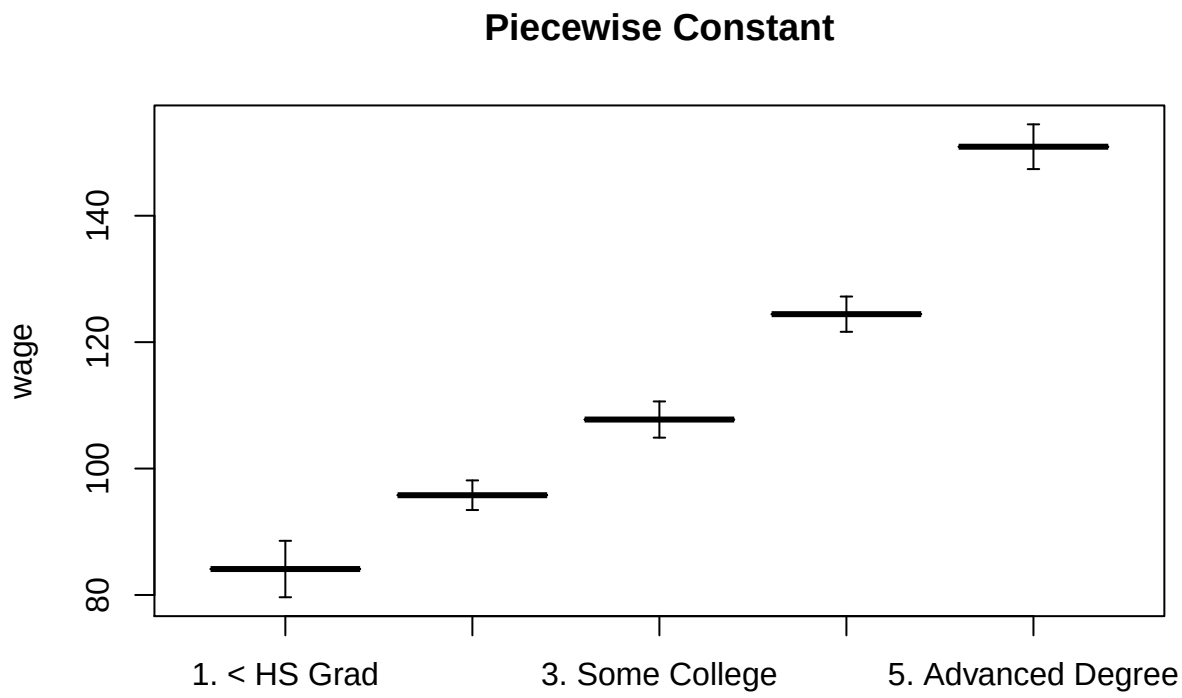
Each row will have a 1 in the first column and in the column given by the bin index. Otherwise zeros. We can create the design matrix with `cut(age, k)`.

```
fit = lm(wage ~ cut(age,3))
Plot(fit, main = "Piecewise Constant")
```



Give custom cutpoints to the `breaks` option if desirable. In the case where the covariate is a factor, `cut()` is unnecessary. Simply giving the name of the covariate to `lm()` will result in a step function. The qualitative variable `education` has levels `< HS Grad`, `HS Grad`, `Some College`, `College Grad` and `Advanced Degree`.

```
fit = lm(wage ~ education)
Plot(fit, main = "Piecewise Constant")
```



Regression Splines

Splines are combinations of the previous two methods. They are polynomials joined in a smooth way at knots $c_1, c_2, \ldots, c_K$. A spline of order M joins polynomials of degree $M-1$. The derivatives up to order $M-2$ are continous, also in the knots. The basis functions are not so different from those we have seen before. However, we will express them using a truncated power function, that is

$$(x-c_j)_+^{M-1} = \begin{cases} (x-c_j)^{M-1} & , x > c_j \\ 0 & , \text{otherwise.} \end{cases}$$

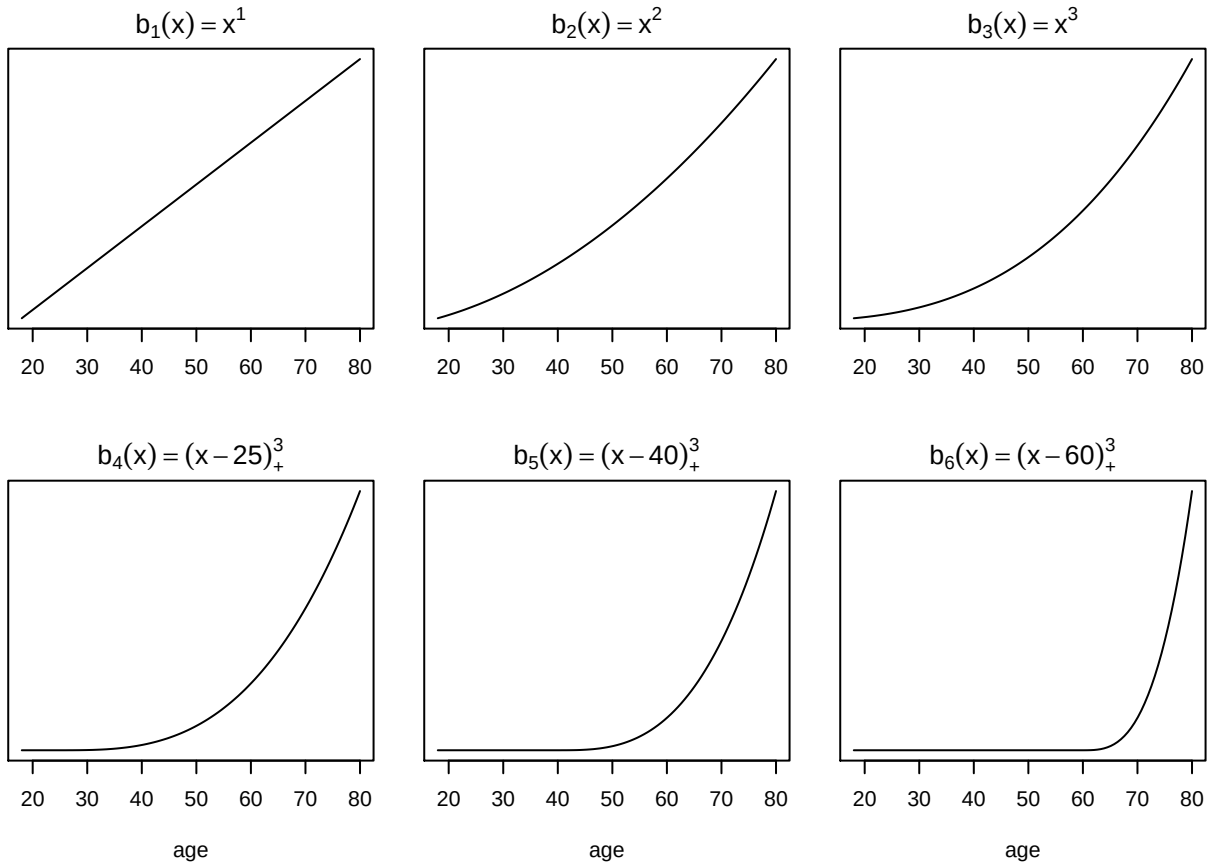
This is a polynomial of degree n truncated at a knot. The standard basis for a order-M spline with K knots is

$$\begin{aligned} b_j(x_i) &= x_i^j && , \ j = 1, \ldots, M-1 \\ b_{M-1+k}(x_i) &= (x_i - c_k)_+^{M-1} && , \ k = 1, \ldots, K. \end{aligned}$$

It may not be obvious why this is a basis for a spline, but you can show that it meets the requirements mentioned above. Note that there are $M+K-1$ basis functions.

Cubic Splines

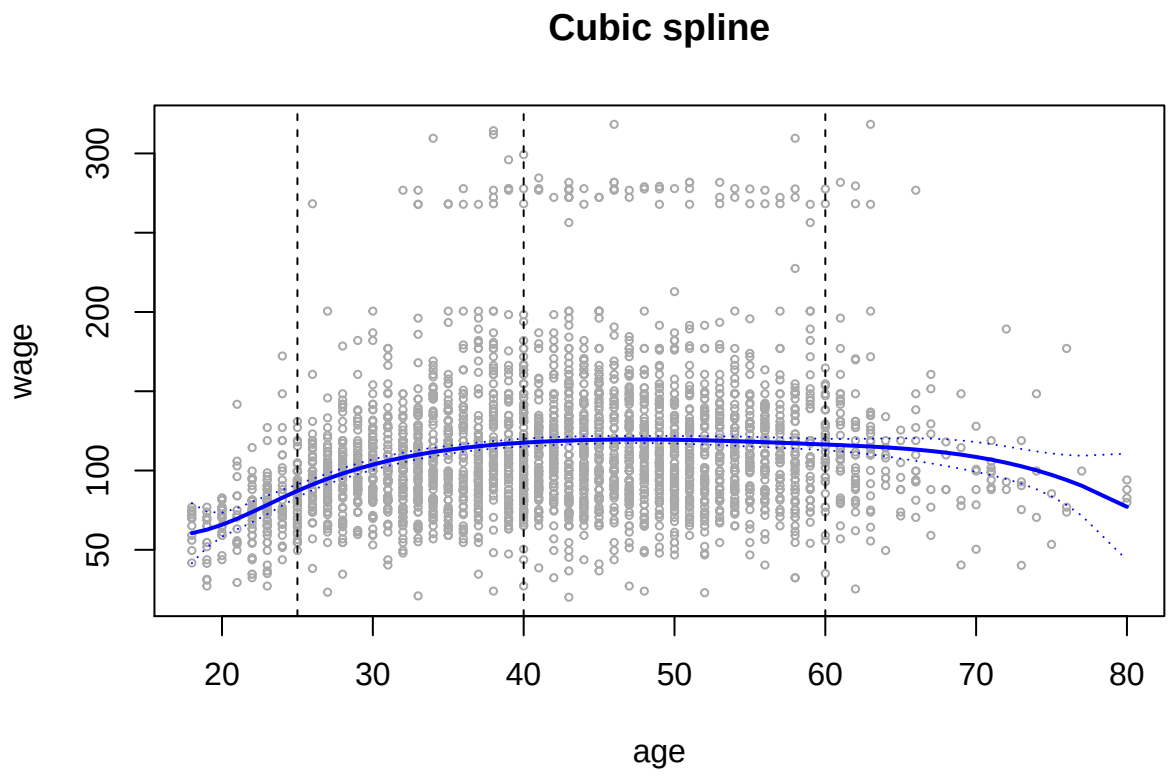
A spline with $M=4$ is cubic. We see that the basis is $X, X^2, X^3, (X-c_1)_+^3, (X-c_2)_+^3, \ldots, (X-c_K)_+^3$. Consider an example with the three knots $c_1 = 25,\ c_2 = 40$ and $c_3 = 60$. This results in 6 basis function that we can plot



in R.

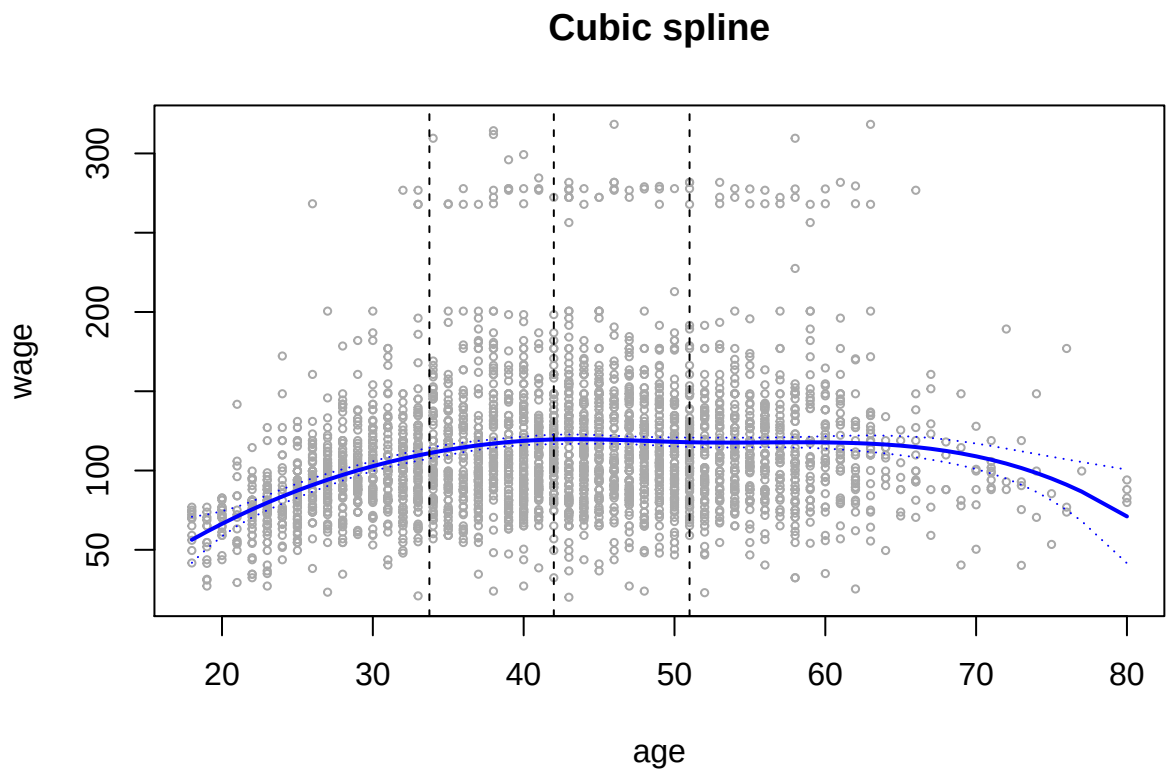
The design matrix is created in R with `bs()`, short for basis splines. The default for `bs()` is cubic splines, but we can specify a different `degree` if desirable. Let us see the result from using the knots above.

```
fit = lm(wage ~ bs(age, knots = c(25,40,60)))
Plot(fit, main = "Cubic spline")
```



Instead of giving the knots explicitely, we can ask R to divide the observations into equally sized bins. The argument `df` is the number of basis functions we want. Thus, specifying `df = 6` results in three knots.

```
fit = lm(wage ~ bs(age, df = 6))
Plot(fit, main = "Cubic spline")
```



In this example, the dotted lines separate the observations into bins with about the same number of observations in each. Note that the cubic splines have high variance at the ends. For other examples, this effect may be even larger and our prediction might have a wild behaviour. This issue may be resolved by using *natural cubic splines*.

Natural Cubic Splines

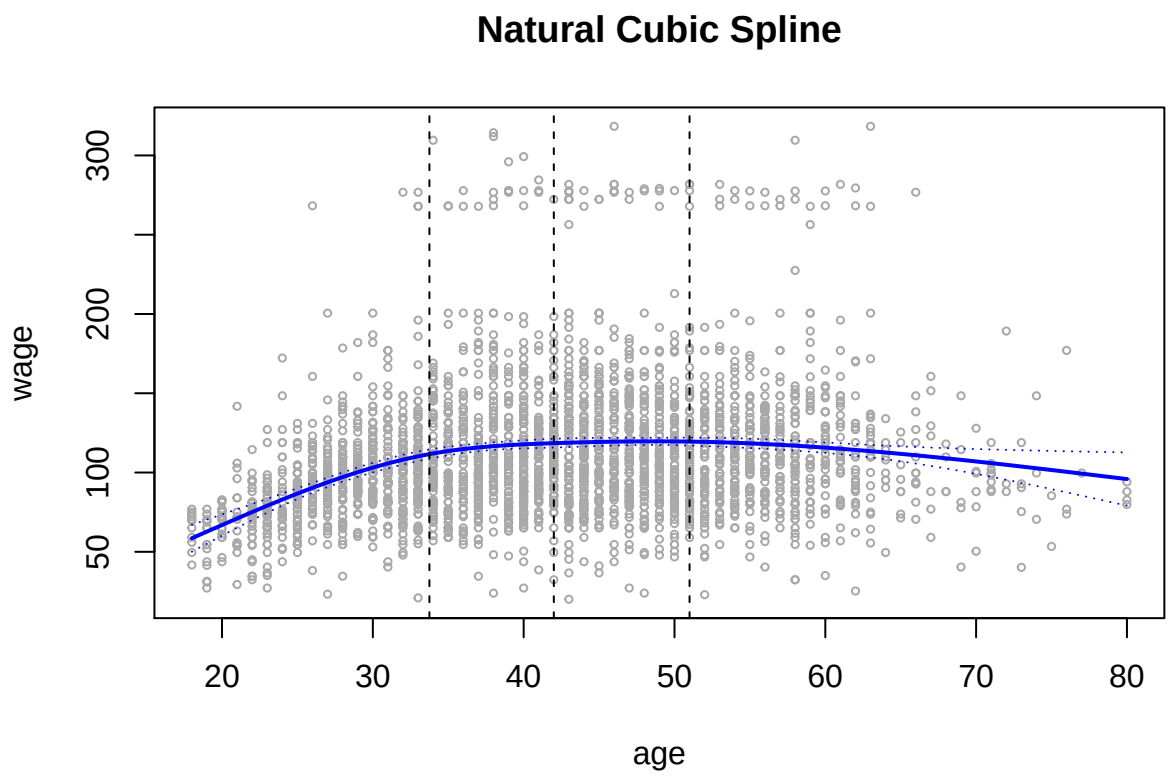
This is a cubic spline that is linear at the ends. Our fitted natural cubic spline will be a straight line when `age` is less than $c_0 = 18$ or greater than $c_{K+1} = 80$. We call these points *boundary knots*, and they typically correspond to the extreme values of x. We will continue referring to the points $c_1, \ldots, c_K$ as interior knots or simply knots. When specifying linearity, we put the second derivative equal to zero on both ends. Two restrictions means two less basis functions as we have less flexibility. We can write the basis as

$$b_1(x_i) = x_i, \quad b_{k+2}(x_i) = d_k(x_i) - d_K(x_i), \; k = 0, \ldots, K-1,$$

$$d_k(x_i) = \frac{(x_i - c_k)_+^3 - (x_i - c_{K+1})_+^3}{c_{K+1} - c_k}.$$

The function `ns()` gives the design matrix for the natural spline. We see that basis functions outnumber knots by 1. Choosing `df = 4` results in 3 knots.

```
fit = lm(wage ~ ns(age, df = 4))
Plot(fit, main = "Natural Cubic Spline")
```



We are now done with the methods that exclusively uses least squares. Henceforth, we cannot use least squares, except for in special cases.

Smoothing Splines

The idea of smoothing splines is very different from that of regression splines. Common for all methods is that we want to minimize the prediction error. In practice, we find a function $f(x)$ that fits the observations well resulting in a small RSS $= \sum_{i=1}^{n}(y_i - f(x_i))^2$. However, we need another requirement in addition to the RSS. Otherwise, our prediction will be a function that interpolates all the observations, with RSS $= 0$. This prediction will have a high variance and a high MSE accordingly. Both bias and variance in our prediction will contribute to the MSE.

Smoothing splines minimize the RSS, but also reduce the variance in the prediction. A smoothing spline is the function f that minimizes

$$\sum_{i=1}^{n}(y_i - f(x_i))^2 + \lambda \int f''(t)^2 \mathrm{d}t,$$

where λ is a non-negative tuning parameter. The first term is loss and puts f close to the observations, while the second term penalizes variability.

Note that $\int f''(t)^2 \mathrm{d}t$ is the total squared change of $f'(t)$, that is how much f turns. The higher λ is, the smoother f will be. In the limit when $\lambda \to \infty$, f is the straight line we would get from linear least squares regression. Conversely, when $\lambda = 0$, f interpolates all the observations. Thus, λ controls the bias-variance trade-off.

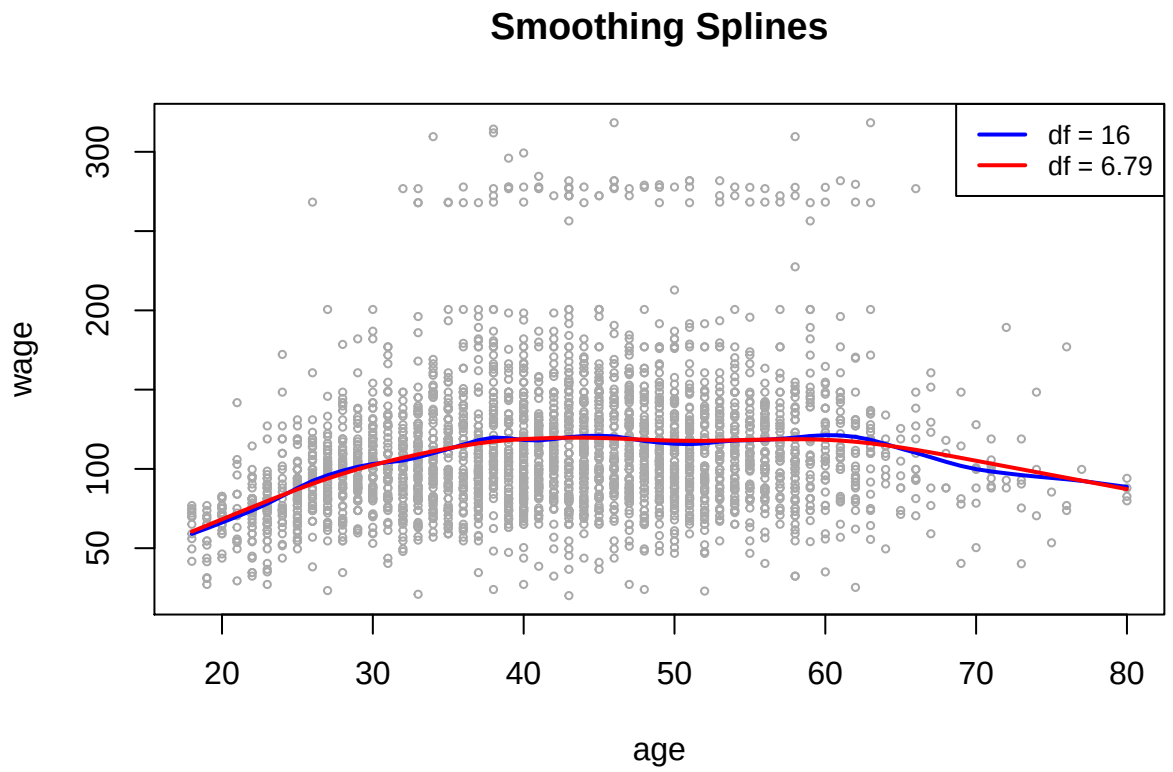
We have not yet discussed why smoothing splines are actually splines. Splines have knots, so that is the case

also for smoothing splines. A smoothing spline is a natural cubic spline with knots at the unique values of x. More precisely, it is a shrunken version of the natural cubic spline we saw in the previous section.

The model is fitted with `smooth.spline()`. A high value of `df` corresponds to a small λ and a lot of freedom. We can either specify `df` or we can ask R to choose a suitable value for λ with leave-one-out cross-validation (LOOCV). The latter is selected by `cv = TRUE`.

```
fit = smooth.spline(age, wage, df = 16)
Plot(fit, main = "Smoothing Splines")

fit = smooth.spline(age, wage, cv = T)
Plot(fit, legend = 16)
```



LOOCV results in `df = 6.79`. As expected, this gives a smoother fit than `df = 16`.

The smoother matrix

This section goes in detail on how we can do LOOCV and how to compute degrees of freedom. A smoothing spline is a linear smoother, which means that we can write the method as

$$\hat{\mathbf{y}} = \mathbf{S}\mathbf{y},$$

where S is the *smoother matrix*. Thus, $\hat{\mathbf{y}}$ is a linear combination of y. The degrees of freedom of a smoothing splines is equal to the number of basis functions, which is the number of unique x_i. However, due to the shrinking penalty, the *effective* degrees of freedom is less. The effective degrees of freedom is defined as the sum of the diagonal elements of the smoothing matrix, that is

$$\mathrm{df}_\lambda = \mathrm{tr}(\mathbf{S}).$$

We can also compute the leave-one-out cross-validation error efficiently with the smoother matrix by

$$\mathrm{RSS}_{cv}(\lambda) = \sum_{i=1}^{n} \left(\frac{y_i - \hat{y}_i}{1 - \mathbf{S}_{ii}} \right)^2.$$

Note that we only need one fit to do cross-validation!

Computing S (optional)

Finding the smoother matrix efficiently involves some steps. Let $\mathbf{X}$ be the $n \times n$ design matrix of the smoothing spline. It consists of an intercept and the natural spline basis functions $b_2, \ldots, b_n$ resulting from knots at the unique $x_i \in [a, b]$. The roughness penalty can also be represented as a matrix Ω, with elements $\Omega_{ij} = \int_a^b b_i''(t) b_j''(t) \mathrm{d}t$. The smoother matrix is now computed as

$$\begin{aligned}
S &= \mathbf{X}(\mathbf{X}^\mathsf{T}\mathbf{X} + \lambda\Omega)^{-1}\mathbf{X}^\mathsf{T} \\
&= \mathbf{X}(\mathbf{X}^\mathsf{T}(\mathbf{I} + \lambda(\mathbf{X}^\mathsf{T})^{-1}\Omega\mathbf{X}^{-1})\mathbf{X})^{-1}\mathbf{X}^\mathsf{T} \\
&= \mathbf{X}\mathbf{X}^{-1}(\mathbf{I} + \lambda(\mathbf{X}^\mathsf{T})^{-1}\Omega\mathbf{X}^{-1})\mathbf{X})^{-1}(\mathbf{X}^\mathsf{T})^{-1}\mathbf{X}^\mathsf{T} \\
&= (\mathbf{I} + \lambda(\mathbf{X}^\mathsf{T})^{-1}\Omega\mathbf{X}^{-1})^{-1} \\
&= (\mathbf{I} + \lambda\mathbf{K})^{-1} \\
&= (\mathbf{U}\mathbf{I}\mathbf{U}^{-1} + \lambda\mathbf{U}\mathbf{D}\mathbf{U}^{-1})^{-1} \\
&= (\mathbf{U}(\mathbf{I} + \lambda\mathbf{D})\mathbf{U}^{-1})^{-1} \\
&= \mathbf{U}(\mathbf{I} + \lambda\mathbf{D})^{-1}\mathbf{U}^\mathsf{T},
\end{aligned}$$

where $K = (\mathbf{X}^\mathsf{T})^{-1}\Omega\mathbf{X}^{-1}$ is the *Reinsch matrix* with the eigendecomposition $K = \mathbf{U}\mathbf{D}\mathbf{U}^{-1} = \mathbf{U}\mathbf{D}\mathbf{U}^\mathsf{T}$. We use this decomposition to avoid computing $\mathbf{X}^{-1}$. Recommended exercise 6 provide code for computing the Reinsch matrix and the eigendecomposition is obtained with `eigen()`. We may use the final expression for $\mathbf{S}$ to compute effective degrees of freedom, that is

$$\begin{aligned}
\mathrm{df}_\lambda &= \mathrm{tr}(\mathbf{S}) \\
&= \mathrm{tr}(\mathbf{U}(\mathbf{I} + \lambda\mathbf{D})^{-1}\mathbf{U}^\mathsf{T}) \\
&= \mathrm{tr}(\mathbf{U}^\mathsf{T}\mathbf{U}(\mathbf{I} + \lambda\mathbf{D})^{-1}) \\
&= \mathrm{tr}((\mathbf{I} + \lambda\mathbf{D})^{-1}) \\
&= \sum_{i=1}^{n} \frac{1}{1 + \lambda d_i},
\end{aligned}$$

where $\mathbf{D} = \mathrm{diag}(d_1, \ldots, d_n)$. Thus, we only need the smoothing parameter and the eigenvalues of the Reinsch matrix to compute the effective degrees of freedom. Note that df_λ is monotonically decreaing in λ.

Connection to ridge regression (optional)

Rigde regression is also a linear smoother. The penalty is on the sum of squared regression coefficients. The smoother matrix is

$$S = \mathbf{X}(\mathbf{X}^\mathsf{T}\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^\mathsf{T}.$$

This is similar to the expression for smoothing splines. Instead of a roughness matrix Ω, rigde regression simply uses the identity $\mathbf{I}$. Note that $\mathbf{X}$ is $n \times k$ for rigde regression and $n \times n$ for smoothing splines. The effective degrees of freedom for rigde regression is

$$\begin{aligned} \mathrm{df}_\lambda &= \mathrm{tr}(\mathbf{S}) \\ &= \mathrm{tr}(\mathbf{X}(\mathbf{X}^\mathsf{T}\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^\mathsf{T}) \\ &= \mathrm{tr}(\mathbf{X}^\mathsf{T}\mathbf{X}(\mathbf{X}^\mathsf{T}\mathbf{X} + \lambda\mathbf{I})^{-1}) \\ &= \sum_{i=1}^{k} \frac{d_i^2}{d_i^2 + \lambda}, \end{aligned}$$

where the values d_i^2 are the eigenvalues of $\mathbf{X}^\mathsf{T}\mathbf{X}$.

Local Regression

Local regression can be thought of as a smoothed k-nearest neighbor algorithm. We first choose a target point x_0 and then make a prediction based on the nearby observations. We draw a line $\beta_0 + \beta_1 x + \beta_2 x^2$ through this neighborhood as close to the observations as possible. More precisely, we are finding the $\hat{\beta}_0$, $\hat{\beta}_1$ and $\hat{\beta}_2$ that minimize

$$\sum_{i=1}^{n} K_{i0}(y_i - \beta_0 - \beta_1 x_i - \beta_2 x_i^2)^2,$$

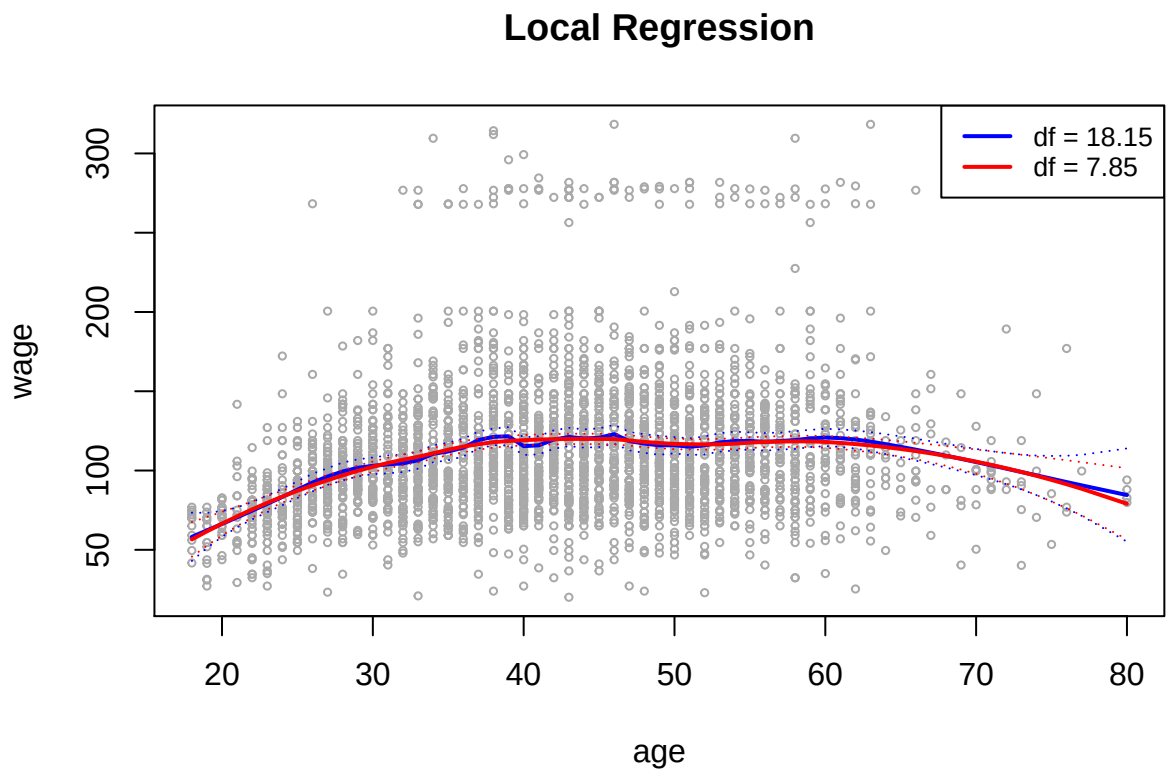
where K_{i0} is a weight we give each observation. Close observations are weighted heavily. Our neighborhood consists of the k observations closest to x_0. Denote the kth closest observation x_κ. The default weights in R are proportional to the tricube

$$K_{i0} = \left(1 - \left|\frac{x_0 - x_i}{x_0 - x_\kappa}\right|^3\right)_+^3$$

Thus, the weights are zero for x_i outside the neighborhood. Our prediction of the wage is $\hat{\beta}_0 + \hat{\beta}_1 x_0 + \hat{\beta}_2 x_0^2$. Writing `loess()` fits the observations by local regression. The size of the neighborhood is regulated by `span`, the fraction k/n. Let us compare `span = 0.2` and `span = 0.5`.

```
fit = loess(wage ~ age, span = .2)
Plot(fit, main = "Local Regression")
df = fit$trace.hat

fit = loess(wage ~ age, span = .5)
Plot(fit, legend = df)
```



Large neighborhoods give little freedom and low variance in the prediction, but at the cost of bias.

Additive Models

We have discussed several methods for predicting `wage`. One option is a cubic spline with $X_1 =$ `age` and knots at 40 and 60. The design matrix when excluding the intercept is

$$\mathbf{X}_1 = \begin{pmatrix} x_{11} & x_{11}^2 & x_{11}^3 & (x_{11}-40)_+^3 & (x_{11}-60)_+^3 \\ x_{21} & x_{21}^2 & x_{21}^3 & (x_{21}-40)_+^3 & (x_{21}-60)_+^3 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ x_{n1} & x_{n1}^2 & x_{n1}^3 & (x_{n1}-40)_+^3 & (x_{n1}-60)_+^3 \end{pmatrix}.$$

Another option is a natural spline with $X_2 =$ `year` and one knot $c_1 = 2006$. We let the boundary knots be the extreme values of `year`, that is $c_0 = 2003$ and $c_2 = 2009$. The design matrix when excluding the intercept is then

$$\mathbf{X}_2 = \begin{pmatrix} x_{12} & \left[\frac{1}{6}(x_{12}-2003)_+^3 - \frac{1}{3}(x_{12}-2006)_+^3 + \frac{1}{6}(x_{12}-2009)_+^3\right] \\ x_{22} & \left[\frac{1}{6}(x_{22}-2003)_+^3 - \frac{1}{3}(x_{22}-2006)_+^3 + \frac{1}{6}(x_{22}-2009)_+^3\right] \\ \vdots & \vdots \\ x_{n2} & \left[\frac{1}{6}(x_{n2}-2003)_+^3 - \frac{1}{3}(x_{n2}-2006)_+^3 + \frac{1}{6}(x_{n2}-2009)_+^3\right] \end{pmatrix}.$$

A third option is using the factor $X_3 =$ `education` which has levels `< HS Grad`, `HS Grad (HSG)` , `Some College (SC)` , `College Grad (CG)` and `Advanced Degree (AD)`. The default representation for this factor in R is dummy variable coding with `< HSG` as the base line reference. The design when excluding the intercept

is

$$\mathbf{X}_3 = \begin{pmatrix} I(x_{13} = \text{HSG}) & I(x_{13} = \text{SC}) & I(x_{13} = \text{CG}) & I(x_{13} = \text{AD}) \\ I(x_{23} = \text{HSG}) & I(x_{23} = \text{SC}) & I(x_{23} = \text{CG}) & I(x_{23} = \text{AD}) \\ \vdots & \vdots & \vdots & \vdots \\ I(x_{n3} = \text{HSG}) & I(x_{n3} = \text{SC}) & I(x_{n3} = \text{CG}) & I(x_{n3} = \text{AD}) \end{pmatrix}.$$

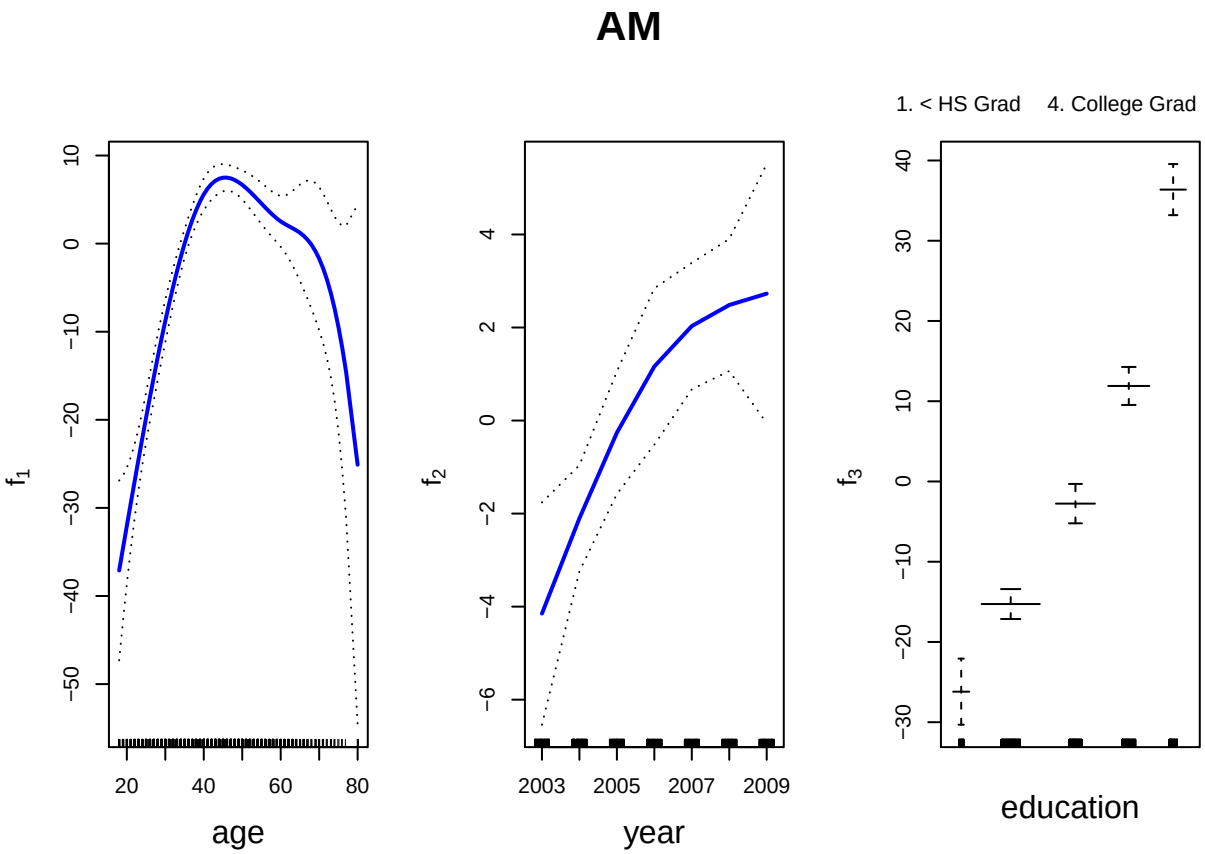
Note that each row of $\mathbf{X}_3$ has a single 1, otherwise zeros.

A generalized additive model combines the models we have discussed so far. Above we fitted `wage` in three different ways. These are a cubic spline in `age`, a natural spline in `year` and a step-function in `education`. We obtain an additive model (AM) when we add these together. Each component is given by a design matrix. The design matrix for the AM is

$$\mathbf{X} = \begin{pmatrix} \mathbf{1} & \mathbf{X}_1 & \mathbf{X}_2 & \mathbf{X}_3 \end{pmatrix},$$

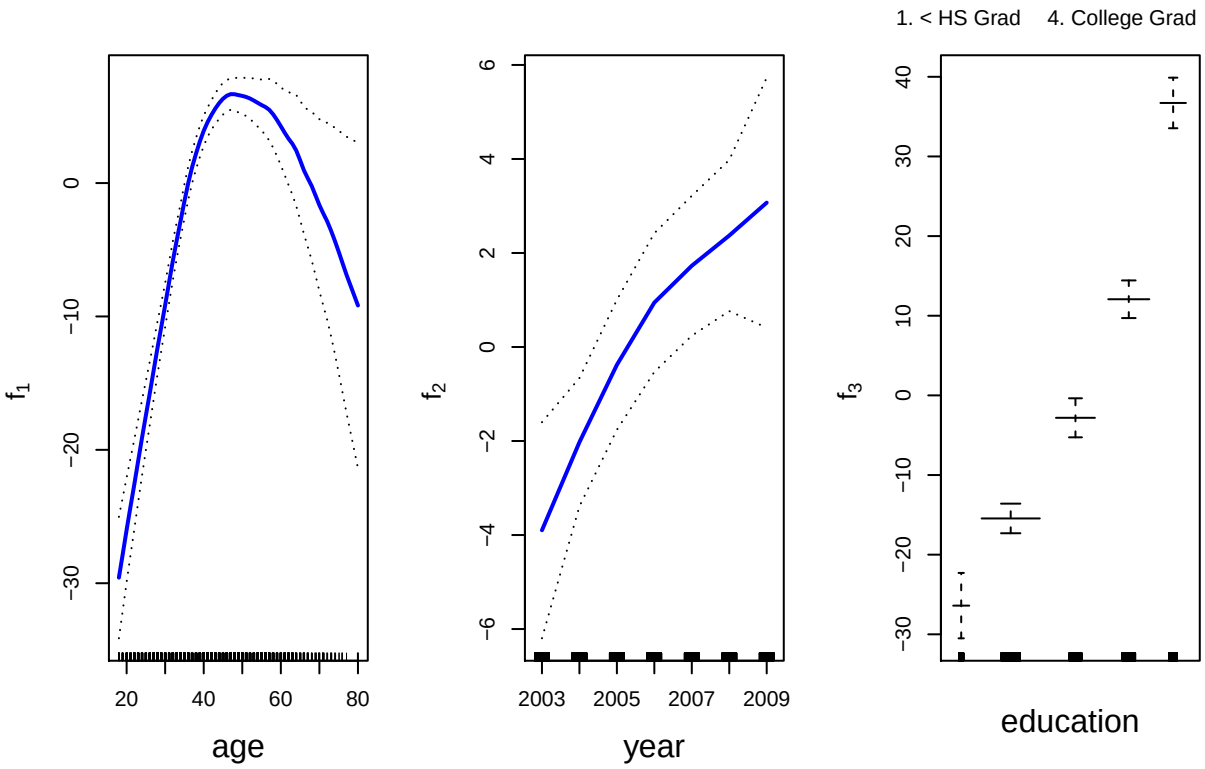
where $\mathbf{1}$ is a column vector of n ones providing an intercept. The easiest way of fitting an AM in R is with `gam()`.

```
fit = gam(wage ~ bs(age,knots=c(40,60)) + ns(year,knots=2006) + education)
Plot(fit)
```



Smoothing splines and local regressions are not expressed by basis functions. Thus, we do not use a design matrix and least squares. Still, we may include smoothing splines and local regressions in an AM. Simply use `s()` and `lo()`. In this case, *backfitting* is used to fit the AM. It is an iterative algorithm where we fit one component at a time, holding the others fixed. Details on this is beyond the scope of what we do here.

```
fit = gam(wage ~ lo(age, span = 0.6) + s(year, df = 2) + education)
Plot(fit)
```



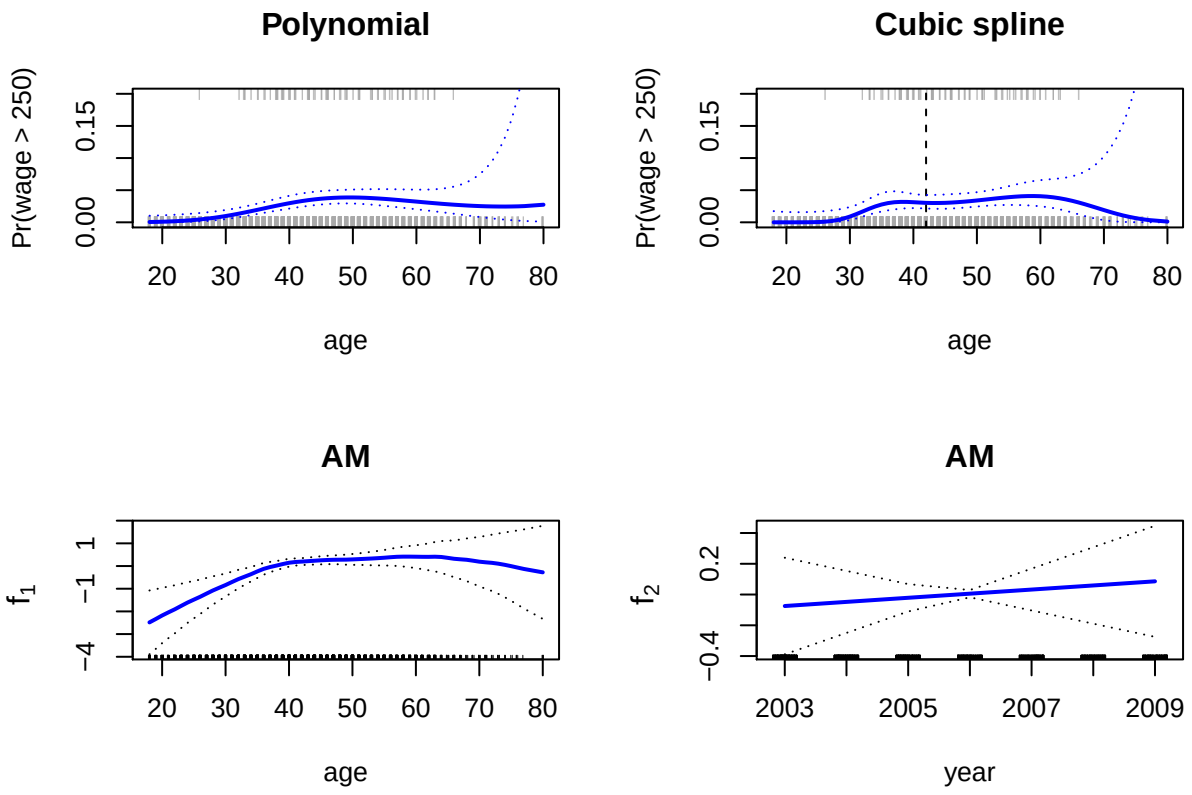
Qualitative Responses

Logistic regression is what we will use for qualitative responses. This is the case whether the model is linear or not, and also when there is no basis function representation. Let us build on the case of predicting `wage`. Only this time, we separate the workers into high earners and low earners with the threshold at $\$250,000$. Then we use X to determine the probability $p(X) = \Pr(Y = 1|X)$. The generalized logistic regression model is

$$\log\left(\frac{p(X)}{1-p(X)}\right) = f(X).$$

Choose the function $f(X)$ from the selection presented in this module. In the first sections we wrote $f(X) = X\beta$. Later we relaxed the linearity. To do logistic regresson, use either `glm()` or `gam()`, with the option `family="binomial"`. The qualitive response is created by `I(wage>250)`. Creating a binary variable like this is in general not recommended, as we lose information. However, let us see some examples.

```
par(mfrow = c(2,2))
Plot(glm(I(wage>250) ~ poly(age,3), family = "binomial"), main = "Polynomial")
Plot(glm(I(wage>250) ~ bs(age, df = 4), family = "binomial"), main = "Cubic spline")
Plot(gam(I(wage>250) ~ lo(age, span = 0.6) + year, family = "binomial"), multi=T)
```



The figure shows estimates of `wage` on the variables $X_1 =$ `age` and $X_2 =$ `year`. The top-left frame is a plot of fitted probabilities for a polynomial logistic regression. The polynomial is of degree 3 in X_1 and the model takes the form

$$\log\left(\frac{p(X_1)}{1-p(X_1)}\right) = \beta_0 + \beta_1 X_1 + \beta_2 X_1^2 + \beta_3 X_1^3.$$

The top-right plot include fitted probabilities for a cubic spline with one knot at 42. We can write the model as

$$\log\left(\frac{p(X_1)}{1-p(X_1)}\right) = \beta_0 + \beta_1 X_1 + \beta_2 X_1^2 + \beta_3 X_1^3 + \beta_4 (X_1 - 42)_+^3.$$

The plots in the lower row shows a fitted GAM with two components $f_1(X_1)$ and $f_2(X_2)$. The model is

$$\log\left(\frac{p(X_1, X_2)}{1-p(X_1, X_2)}\right) = \beta_0 + f_1(X_1) + f_2(X_2).$$

The first component f_1 is a local regression in `age`. The second component f_2 is a simple linear regression in `year`. Consult Module 4 for more on logistic regression.

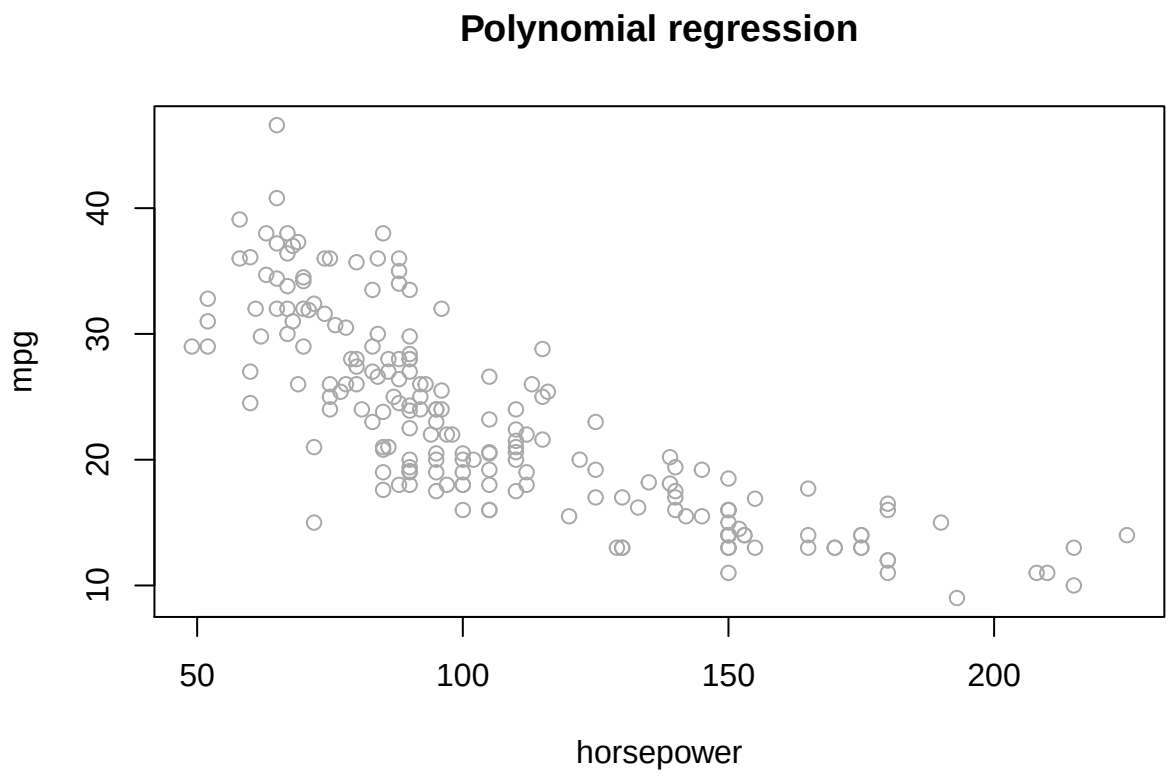
Recommended Exercises

Problem 1: Let us take a look at the `Auto` data set. We want to model miles per gallon `mpg` by engine horsepower `horsepower`. Separate the observations into training and test. A training set is plotted below.

Perform polynomial regression of degree 1, 2, 3 and 4. Use `lines()` to add the fitted values to the plot below.

Also plot the test error by polynomial degree.

```
library(ISLR)
ds = Auto[c("horsepower","mpg")]
n = nrow(ds)
deg = 1:4
set.seed(1)
tr = sample.int(n, n/2)
plot(ds[tr,], col = "darkgrey", main = "Polynomial regression")
```



Problem 2: We will continue working with the `Auto` data set. The variable `origin` is 1,2 or 3, corresponding to American, European or Japanese origin. Use `factor(origin)` for conversion to a factor variable. Predict `mpg` by origin. Plot the fitted values and approximative 95 percent confidence intervals. Selecting `se = T` in `predict()` gives standard errors of the prediction.

Problem 3: Now, let us look at the `Wage` data set. The section on AMs explains how we can create an AM by adding components together. One component we saw is a natural spline in `year` with one knot. Derive the expression for the design matrix $\mathbf{X}_2$ from the natural spline basis above.

Problem 4: We will continue working with the same AM as in problem 3. The R call `model.matrix(~ bs(age,knots=c(40,60)) + ns(year,knots=2006) + education)` gives a design matrix for the AM. This matrix is what `gam()` uses. However, it does not equal our matrix $\mathbf{X}$. The predicted responses will still be the same.

Write code that produces $\mathbf{X}$. The code below may be useful.

```
mybs = function(x,knots) cbind(x,x^2,x^3,sapply(knots,function(y) pmax(0,x-y)^3))

d = function(c, cK, x) (pmax(0,x-c)^3-pmax(0,x-cK)^3)/(cK-c)
myns = function(x,knots){
```

```
  kn = c(min(x), knots, max(x))
  K = length(kn)
  sub = d(kn[K-1],kn[K],x)
  cbind(x,sapply(kn[1:(K-2)],d,kn[K],x)-sub)
}

myfactor = function(x) model.matrix(~x)[,-1]
```

If the code is valid, the predicted response $\hat{\mathbf{y}} = \mathbf{X}(\mathbf{X}^\mathsf{T}\mathbf{X})^{-1}\mathbf{X}^\mathsf{T}\mathbf{y}$ should be the same as when using the built-in R function.

```
X =
myhat = lm(wage ~ X - 1)$fit
yhat = gam(wage ~ bs(age, knots = c(40,60)) + ns(year, knots = 2006) + education)$fit
all.equal(myhat,yhat)
```

How can `myhat` equal `yhat` when the design matrices differ?

Problem 5: Problem 3 on Compulsory 2.

Problem 6 (Advanced): In the part where we discussed smoothing splines there is a section explaining how to compute $\mathbf{S}$, where $\hat{\mathbf{y}} = \mathbf{S}\mathbf{y}$. This is implemented below, with $\mathbf{x}$ as unique observations of `age` and $\mathbf{y}$ the coresponding `wage`.

```
K = function(x){
  xi = sort(unique(x))
  n = length(xi)
  h = xi[-1]-xi[-n]
  i = seq.int(n-2)
  D = diag(1/h[i], ncol = n)
  D[cbind(i,i+1)] = - 1/h[i] - 1/h[i+1]
  D[cbind(i,i+2)] = 1/h[i+1]
  W = diag(h[i]+h[i+1]/3)
  W[cbind(i[-1],i[-1]-1)] = h[i[-1]]/6
  W[cbind(i[-1]-1,i[-1])] = h[i[-1]]/6
  t(D)%*%solve(W)%*%D
}

x = sort(unique(age))
y = wage[order(age[!duplicated(age)])]
eig = eigen(K(x))
U = eig$vectors
d = eig$values
lambda = 2000
S = U%*%diag(1/(1+lambda*d))%*%t(U)
```

Use $\mathbf{S}$ from this code to compute $\hat{\mathbf{y}}$. Also compute $\hat{\mathbf{y}}$ using `smooth.spline()` with the correct degrees of freedom. Finally, plot both sets of fitted values and observe that they are equal.

```
myhat =
yhat = smooth.spline(x, y, df =    )$y

plot(x,y, main = "Comparison of fitted values")
co = c("blue", "red")
w = c(5,2)
lines(x,myhat, lwd = w[1], col = co[1])
lines(x,yhat, lwd = w[2], col = co[2])
```

```
legend("topright", legend = c("myhat", "yhat"), col = co,
       lwd = w, inset=c(0, -0.19), xpd = T)
```

References