Polynomial Regression

There are many ways to estimate "flexible" models that allow for all sorts of nonlinearity. We have already discussed K-nearest-neighbour regression. Others include smoothing splines and kernel regression.

- KNN regression is an extremely local method. It estimates $\mathrm{E}(y \mid x_0)$ using only the K points nearest x_0.
- When there is just one predictor, the KNN estimate of $\mathrm{E}(y \mid x_0)$ takes the same value for all $x_0 \geq \max(x_i)$.
- It also takes the same value for all $x_0 \leq \min(x_i)$.
- In both cases, the K nearest neighbors do not change as x_0 gets larger relative to the largest values, or smaller relative to the smallest values, in the sample.
- This makes no sense if $\mathrm{E}(y \mid x_0)$ is changing systematically as x_0 moves within the range of observed x_i. Thus KNN is unlikely to work well for extreme values of x_0.

A more global method (often too global!) is **polynomial regression**.

In the univariate case, we just regress the y_i on a constant, x_i, x_i^2, and so on up to x_i^d, where d is the **degree** of the polynomial.

- If d is large enough, this can be very flexible, perhaps too flexible.
- For $d > 2$, extrapolation often works very badly.
- We can choose d by sequential testing, by comparing C_p, AIC, or BIC, or (less commonly) by cross-validation.
- Unlike with KNN, the fit in any region is affected by *all* of the data. Distant points may have more influence than nearby ones.

After we discuss polynomial regression, we will discuss a number of methods that are less global than it is but less local than KNN.

These will include **step functions**, **basis functions**, **regression splines**, **smoothing splines**, **local regression**, **generalized additive models**, and **kernel regression**.

With the exception of kernel regression, these are all discussed in Chapter 7 of ISLR/ISLP.

Figures 12.1 though 12.3 plot earnings (ISLR/ISLP calls them wages) against age, for 3000 observations.

The figures put the log of earnings on the vertical axis. The comparable ones in ISLR/ISLP use earnings itself.

Why take logs? Is it a good idea in this case?

The figures show fitted values and confidence intervals for three polynomial regressions, where $f(x)$ is quadratic, cubic, or quartic.

The regressions really should have included other explanatory variables, but these ones did not.

The confidence intervals are two standard errors above and below the fitted values. The standard errors for $f(x_0)$ are

$$\text{s.e.}\left(\hat{f}(x_0)\right) = \left(z_0^\top \widehat{\text{Var}}(\hat{\boldsymbol{\beta}}) z_0\right)^{1/2}, \tag{1}$$

where $z_0^\top = [1 \;\; x_0 \;\; x_0^2 \;\; \ldots \;\; x_0^d]$. Notice that $\text{s.e.}\left(\hat{f}(\boldsymbol{x}_0)\right)$ depends on the entire covariance matrix.

Figure 12.1 — Polynomial Regression of Degree 2

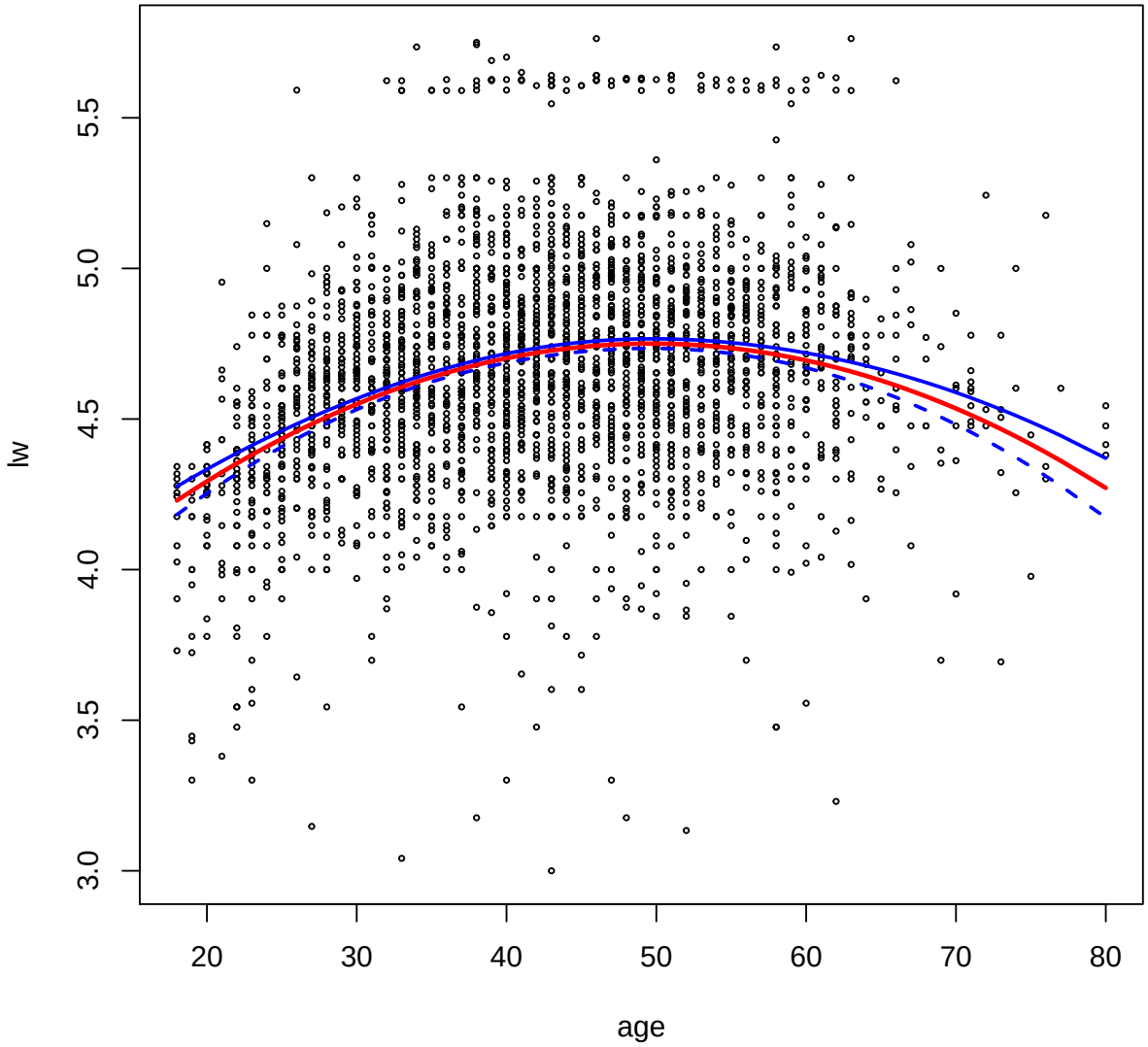


Figure 12.2 — Polynomial Regression of Degree 3

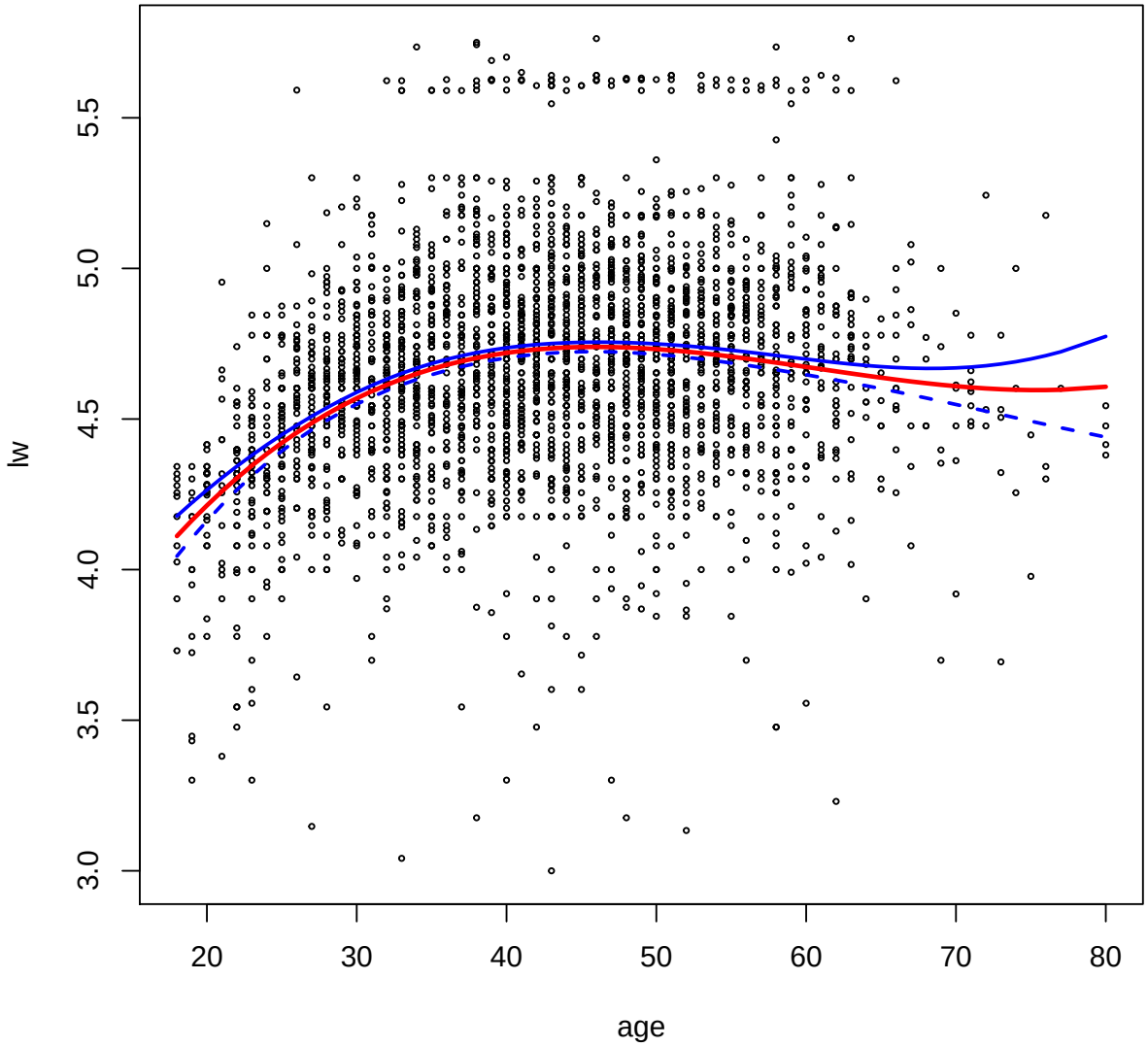
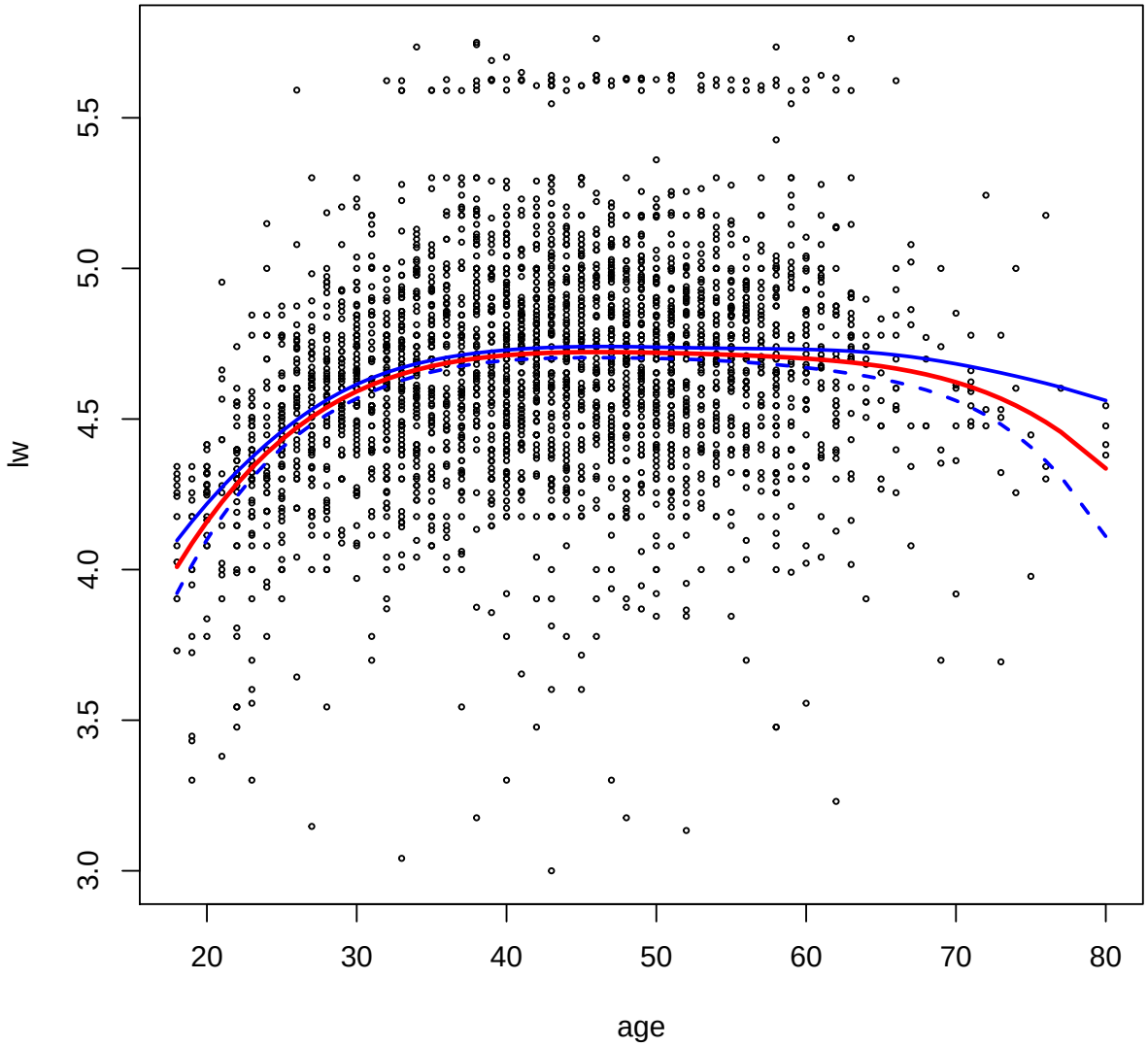


Figure 12.3 — Polynomial Regression of Degree 4



The confidence intervals are narrow for most ages, but they get wider for extreme ages, especially for $d = 3$ and $d = 4$.

The earnings data have some odd features.

- Top coding? Missing data?
- There seems to be a gap between about \$200K and \$250K.

The natural way to create polynomials in age is to define predictors like `agesq = age`^{2} and `agecube = age`^{3}.

- It is easier to use the `poly()` function within `lm()` itself.
- By default, `poly(age,4)` creates four **orthogonal polynomials**.
- These collectively have exactly the same explanatory power as `age` through `age`^{4}, but they are orthogonal to each other.
- Each column of the matrix of orthogonal polynomials is a linear combination of `age`, `age`^{2}, `age`^{3}, and `age`^{4}.

Using orthogonal polynomials instead of ordinary ones (available with the "`raw = TRUE`" option) has several advantages:

- Because the correlations among x, x^2, x^3, x^4, and so on tend to be very high, a regressor matrix that includes all of them may be close to singular, or even numerically singular.
- With orthogonal polynomials, the individual regressors have zero correlation with each other, so those columns of $\boldsymbol{X}$ are as far as possible from being singular.
- If we increase the order of the polynomial, the intercept and the lower-order coefficients do not change as we add higher-order terms. Other coefficients (if any) may change, however.
- The t-statistics on previous terms will change because the estimate of σ changes, but usually by just a little.
- As with principal components, we can obtain all the coefficients from a set of univariate regressions.

Step Functions

We can use **step functions** instead of polynomials.

We pick a number of **cut points** c_1 through c_K and define $C_0(x)$ through $C_K(x)$ as

$$\begin{aligned} C_0(x) &= \mathbb{I}(x < c_1); \\ C_1(x) &= \mathbb{I}(c_1 \leq x < c_2); \\ &\vdots \\ C_{K-1}(x) &= \mathbb{I}(c_{K-1} \leq x < c_K); \\ C_K(x) &= \mathbb{I}(c_K \leq x) \end{aligned}$$

Then we regress y_i on $C_0(x_i)$ through $C_K(x_i)$, dropping the intercept, or on $C_1(x_i)$ through $C_K(x_i)$, keeping the intercept.

This yields a **piecewise-constant** regression. See Figures 12.4, 12.5, and 12.6. The middle cut is very hard to see in Figure 12.4.

Figure 12.4 — Step Function Regression with 3 Cuts

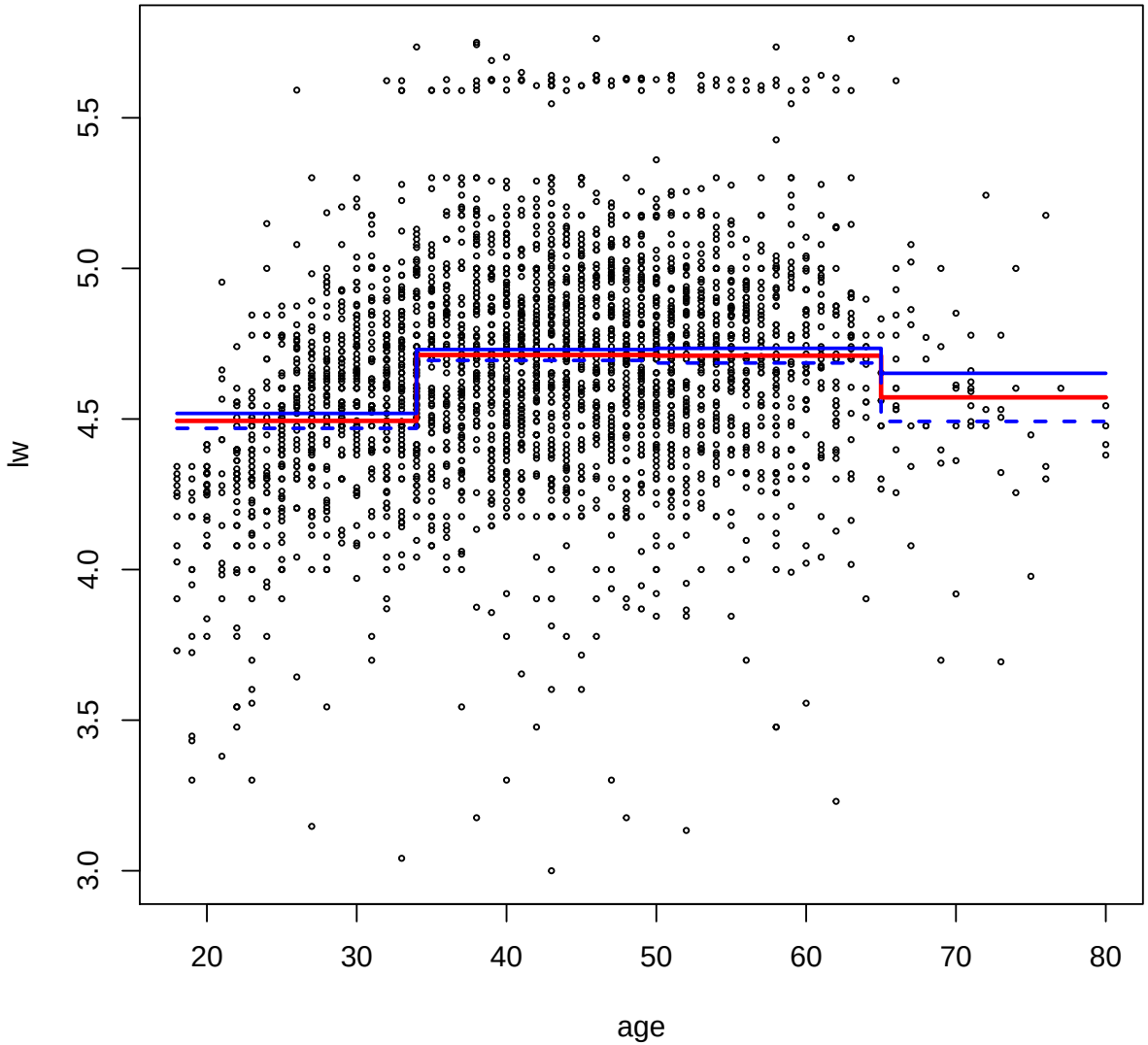


Figure 12.5 — Step Function Regression with 7 Cuts

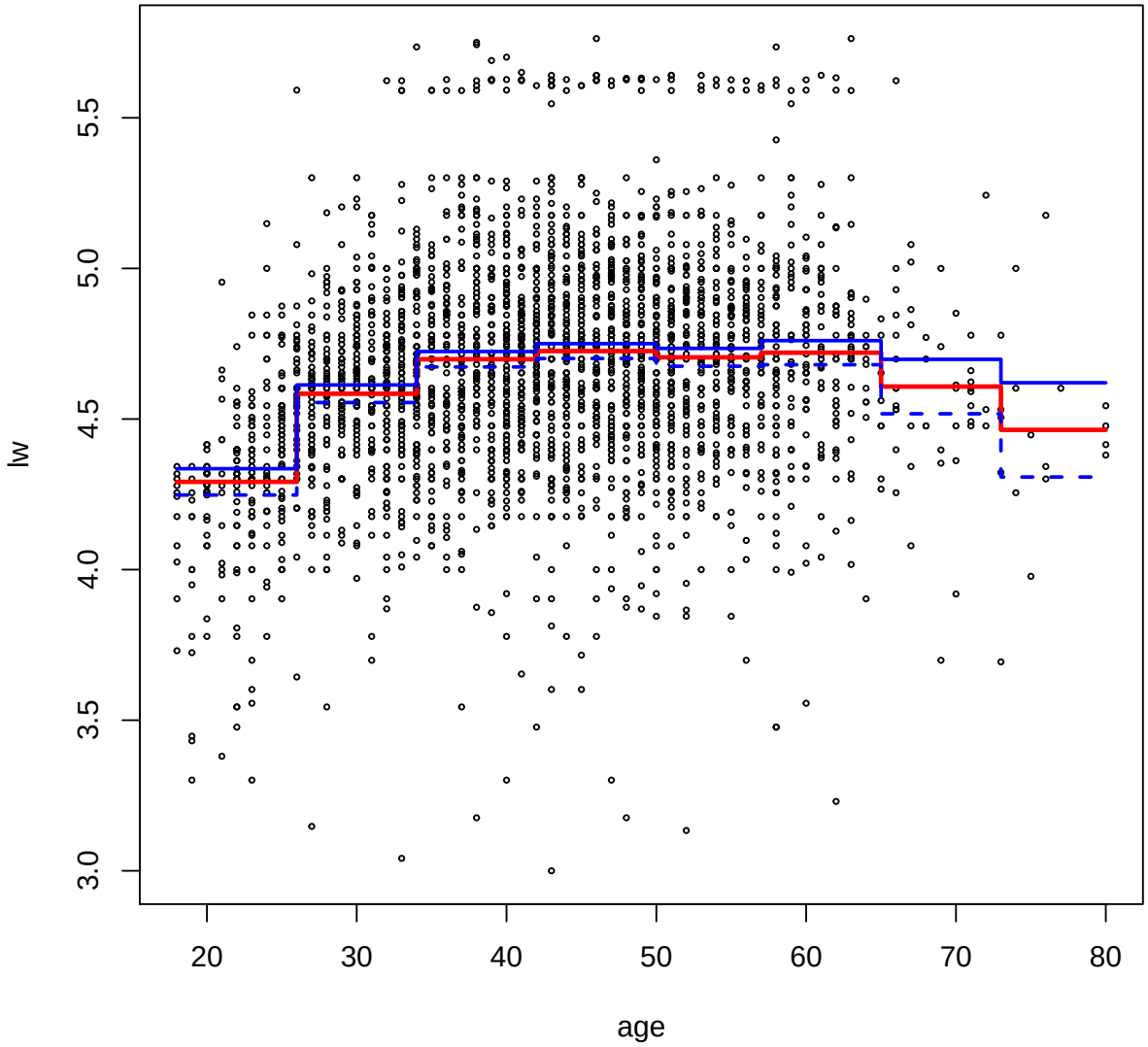
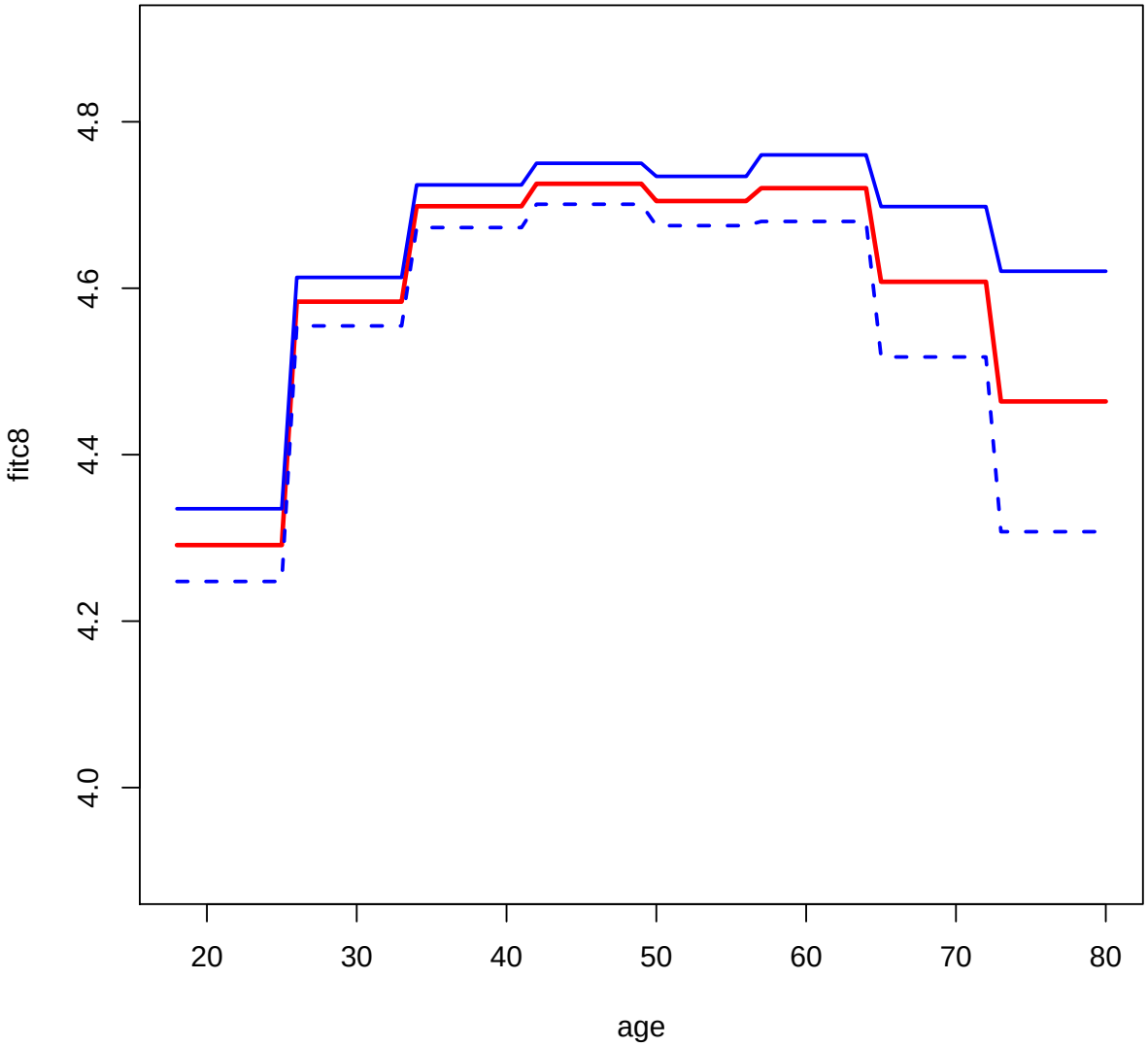


Figure 12.6 — Step Function Regression with 7 Cuts



The step functions in Figures 12.4 and 12.5 do not look as appealing as the polynomials in Figures 12.1 through 12.3.

However, they are better behaved near the boundaries. Extrapolation works the same way as for KNN.

The general shape seems reasonable:

- Earning rise for about 20 years, stay roughly constant for the next 20 years, and then fall fairly sharply.
- Figures 12.5 and 12.6 (with 7 cuts) give a better picture than Figure 12.4 (with 3 cuts).
- However, there is a small downward blip between ages 49 and 57 that is probably just random.

The cut points here are chosen by the `cut()` function without reference to the data. Tree-based methods will choose them more intelligently.

Of course, a more realistic earnings equation would have to include other explanatory variables, such as education and marital status.

Basis Functions

Polynomials and piecewise-constant regressors are special cases of **basis functions**.

These are functions of one or more predictors, determined in advance, that are then used as regressors.

If there are K basis functions $b_1(x)$ through $b_K(x)$, we just regress y_i on a constant and $b_1(x_i)$ through $b_K(x_i)$.

In the case of a polynomial of degree d, we have d basis functions.

In the case of a piecewise-constant regression with K cuts, we have K basis functions. Counting the intercept, the conditional mean can take on $K+1$ different values.

For $p >> 1$, the number of basis functions can get out of hand.

When $d = 2$, we have p linear basis functions, p quadratic basis functions, and $p(p-1)/2$ cross-product basis functions.

Piecewise Polynomials

A **piecewise polynomial** function simply divides the domain of x (assumed one-dimensional for now) into $K+1$ contiguous intervals.

The K points that define the intervals are called **knots**. These are like the cut points for piecewise-constant regression.

We have to estimate $K+1$ low-order polynomials, one for each interval.

This is easily done using any standard regression routine. With $d = 2$ and $K = 1$, for example, the regression is

$$\begin{aligned} y_i = &\beta_0^1 \mathbb{I}(x_i < c_1) + \beta_1^1 x_i \mathbb{I}(x_i < c_1) + \beta_2^1 x_i^2 \mathbb{I}(x_i < c_1) \\ &+ \beta_0^2 \mathbb{I}(x_i \geq c_1) + \beta_1^2 x_i \mathbb{I}(x_i \geq c_1) + \beta_2^2 x_i^2 \mathbb{I}(x_i \geq c_1) + u_i \end{aligned} \tag{2}$$

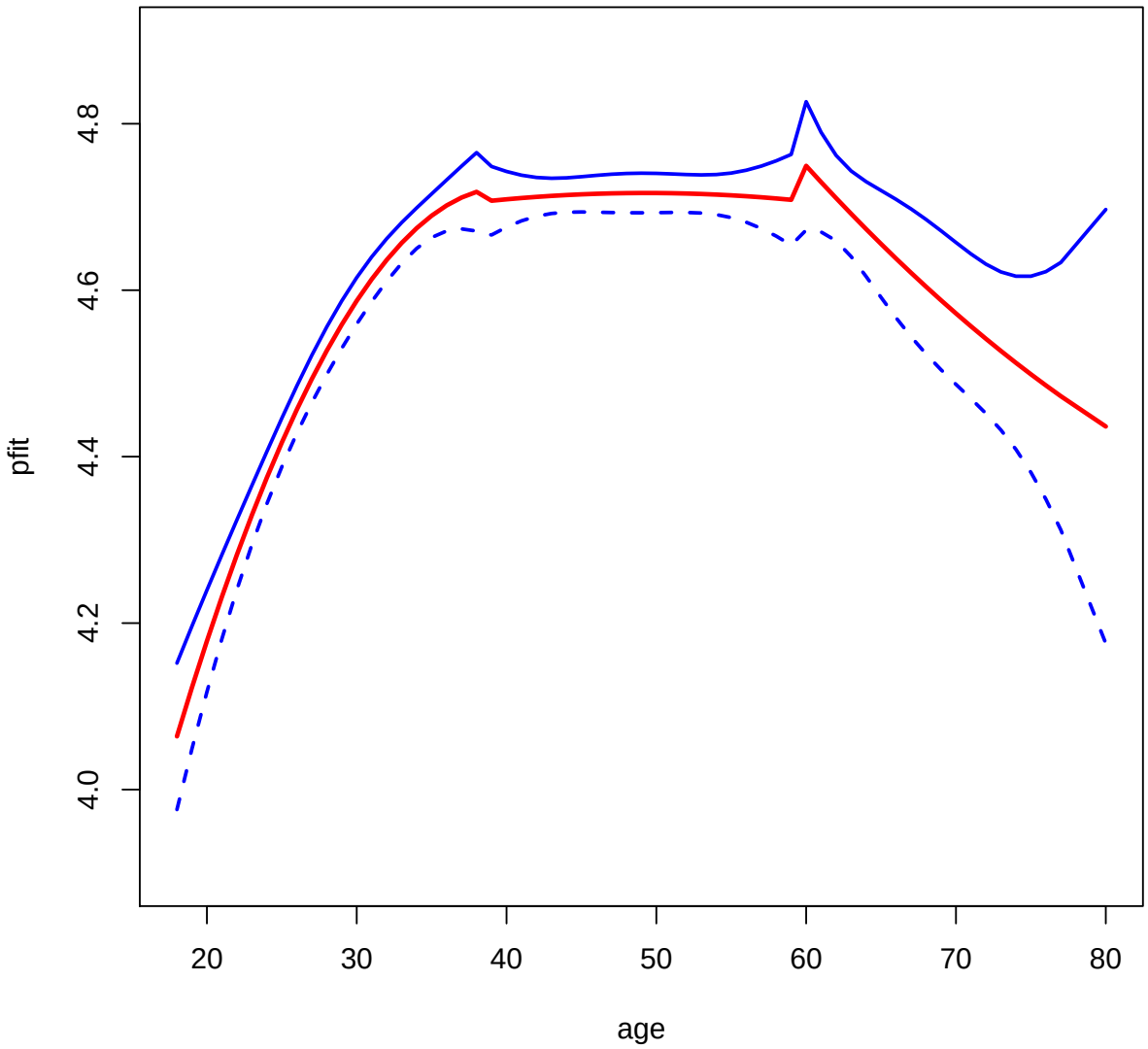
There are six regressors, each of which is zero for one group of observations.

But piecewise polynomial regressions like (2) yield strange-looking predictions. There will be a jump, or a vertical gap, at every knot.

See Figure 12.7, where we used $d = 2$ and $K = 2$.

- The overall shape is reasonable, but strange things happen at the knots, especially the second one.
- The confidence interval also becomes very wide at the top end, probably because there are not a lot of observations in the subsample of older workers.
- That fact combined with the presence of a quadratic term leads to a very wide confidence interval for $\texttt{age} > 75$.
- Even though all predictions for the third subsample depend on the same three coefficient estimates, the prediction standard errors increase a lot with age.
- This is a consequence of (1). The weights on the components of the covariance matrix change with age.

Figure 12.6 — Piecewise Quadratic with 2 Cuts



Regression Splines

The solution is to constrain the curves in various ways.

The simplest constraint is that the two curves that meet at any knot should be equal. This means that the function is continuous.

But a continuous curve can (and in this case usually will) have kinks.

We can eliminate the kinks by requiring that the curves also have the same derivatives (up to some order) at every knot.

When we impose this sort of constraint, we are using **splines**.

A widely-used method is **cubic splines**. These involve polynomials of degree 3 with constraints on the level and the first two derivatives.

- Cubic splines are remarkably parsiminious. The number of basis functions is just $3 + K$. This a *global* method.
- Adding one more knot only requires us to estimate one more parameter (versus 4 more for piecewise cubic regressions).

Using cubic splines turns out to be surprisingly easy.

- Unlike for piecewise polynomials, there is always an intercept.
- The first three basis functions are just x_i, x_i^2, and x_i^3.
- The next K basis functions are **truncated power basis functions**, say $h(x_i, \xi_k), k = 1, \ldots, K$, where ξ_k is the k^{th} knot.
- There is only one truncated power basis function per knot.

In the cubic spline case, the truncated power basis function is

$$h(x, \xi) = \begin{cases} (x - \xi)^3 & \text{if } x > \xi \\ 0 & \text{otherwise.} \end{cases} \tag{3}$$

So we start with an unrestricted cubic polynomial and successively add as many truncated power basis functions as we want.

The numbers make sense. Without restrictions, each knot would add 4 regressors. But there are 3 restrictions (on the level and the first two derivatives), so it actually adds just 1 regressor.

The `splines` library makes this all pretty easy.

The command

```
sfit = lm(lw~bs(age,knots=c(25,40,55)))
```

creates the six basis functions needed for a cubic spline with knots at ages 25, 40, and 55. It then regresses `lw` on them.

If we change the "bs" command to

```
bs(age,knots=c(25,35,45,55,65)),
```

we instead get eight basis functions. Note that `bs` means "B-spline."

- The predictions from the model with 3 knots look much better than the the ones from the model with 5 knots; compare Figures 12.8 and 12.9
- The more complicated model fits only a little bit better. The F statistic (not really valid) is 2.65, which has a P value of 0.071.
- Even if it did fit substantially better, the more complicated model makes far less sense than the simpler one.

Figure 12.8 — Cubic Spline with 3 Knots

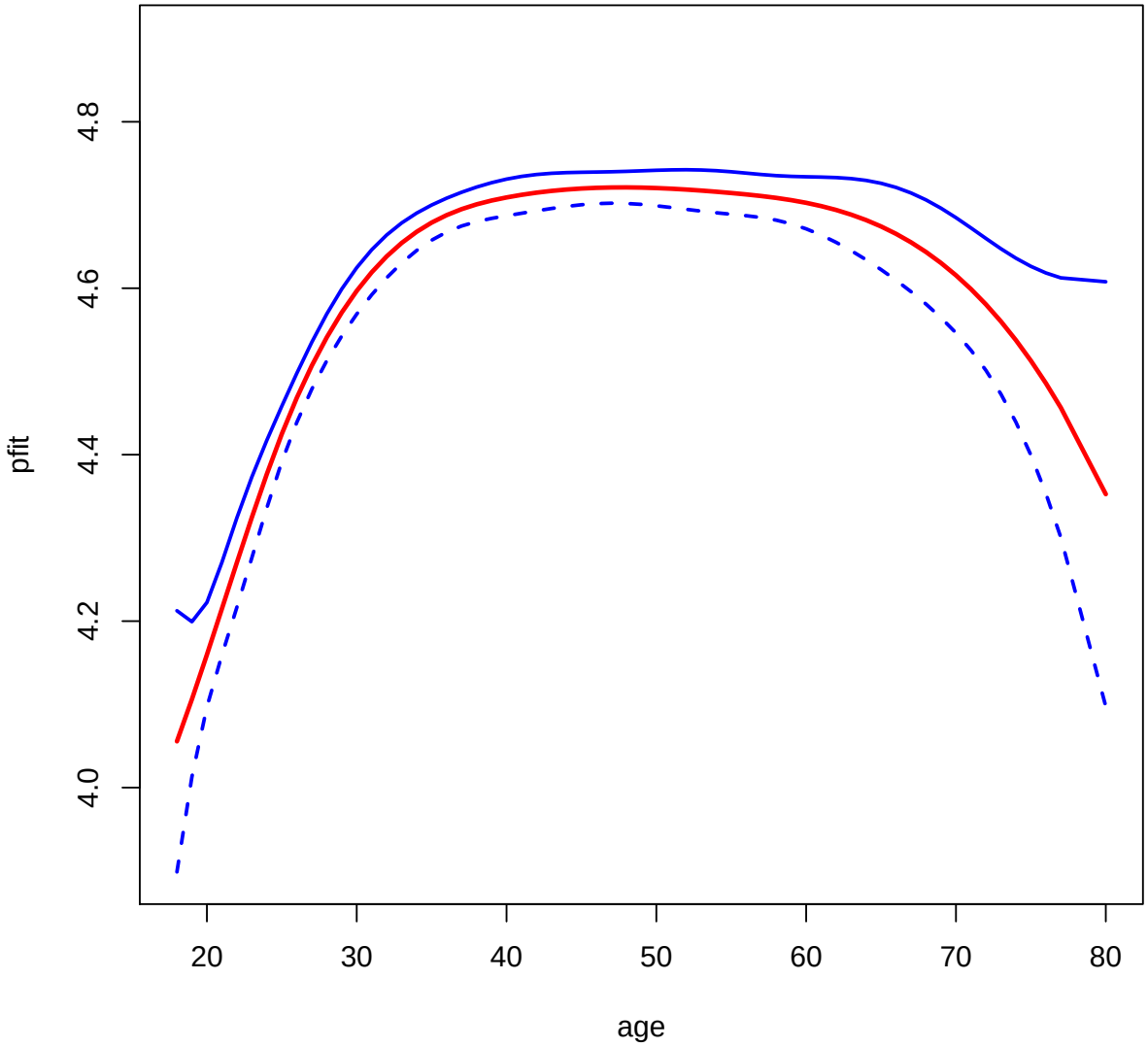
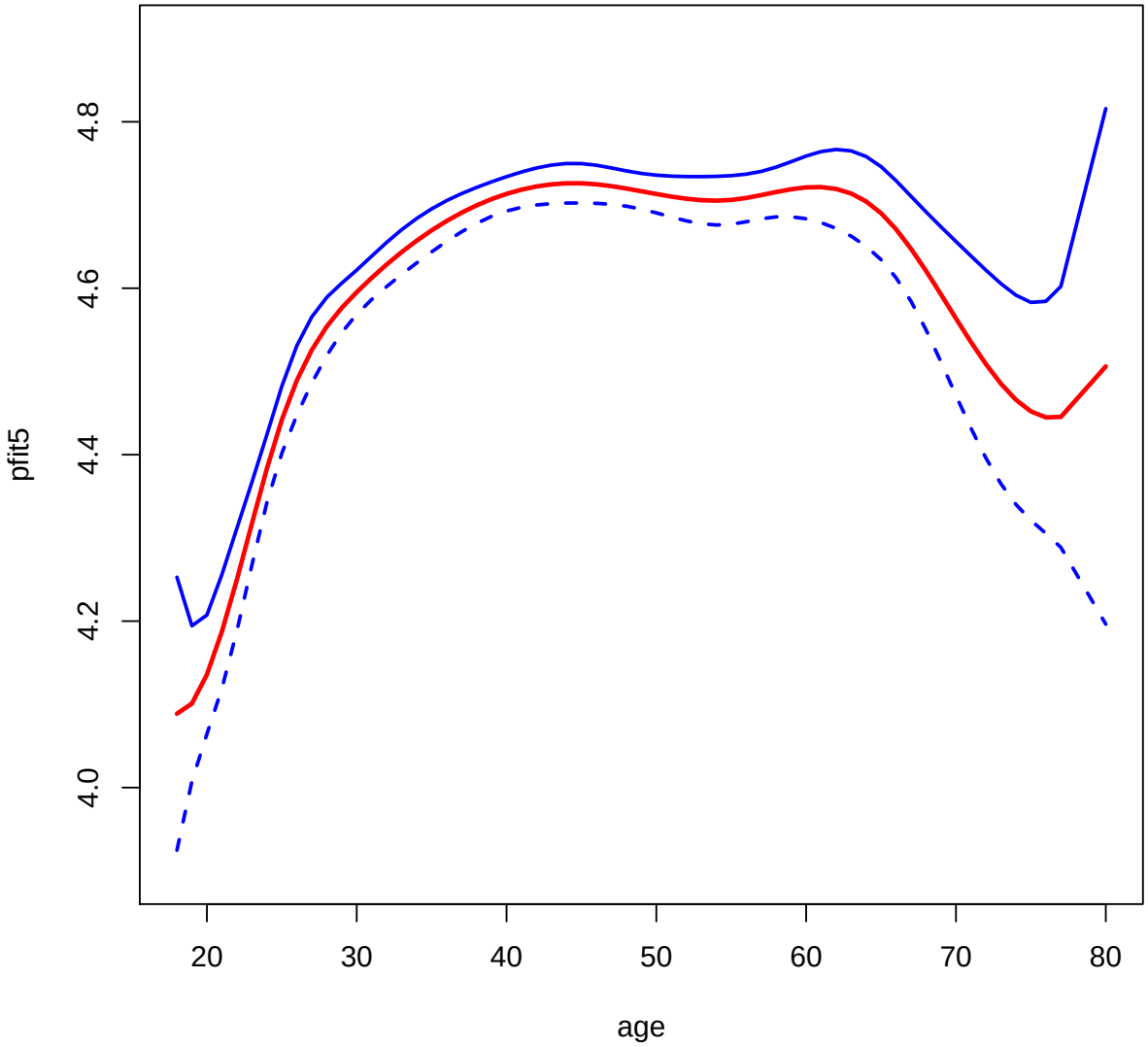


Figure 12.9 — Cubic Spline with 5 Knots



The practical issue with cubic splines is to determine the number of knots and their locations.

One possibility is to use L-fold cross-validation (where L is used instead of K because K has a completely different meaning).

Given a rule for placing the K knots (say at the quantiles), we can

- generate L subsamples of size $N - N/L$;
- estimate the spline regression for each of them;
- use those estimates to make predictions for the N/L omitted observations.
- Choose the value of K to minimize the average of the L cross-validated MSEs.

In principle, we could also use cross-validation to decide where to place the knots, but that would be a lot harder, unless we were just comparing a few simple rules.

Cubic splines can behave badly near the boundaries.

This can be "solved" by adding two additional constraints, namely, that the function be linear at each of the boundaries.

For a **natural cubic spline**, we force the functions to be linear (instead of cubic) below the lowest knot and above the highest knot.

This means adding 4 constraints. So we now have just K parameters to estimate, instead of $K + 4$. But the boundary knots are counted.

A natural cubic spline should therefore probably have more knots than an ordinary cubic spline.

The first two basis functions are just 1 and x. The remaining $K - 2$ are

$$b_{k+2}(x) = \frac{(x - \xi_k)_+^3 - (x - \xi_K)_+^3}{\xi_K - \xi_k} - \frac{(x - \xi_{K-1})_+^3 - (x - \xi_K)_+^3}{\xi_K - \xi_{K-1}} \tag{4}$$

for $k = 1, \ldots, K - 2$, where $(x - \xi_k)_+^3 = \max\left((x - \xi_k)^3, 0\right) = h(x_i, \xi_k)$.

In order to use natural cubic splines, we simply invoke the `ns()` function instead of the `bs()` function.

To get a natural cubic spline with 6 knots, we use

```
ns(age,knots=c(25,30,40,50,60,70)),
```

and we obtain the predictions shown in Figure 12.10.

This model estimates 7 parameters, one for each of the 6 knots that we specified, plus the intercept.

Figure 12.10 looks worse than Figure 12.8 to me, although the fit is marginally better.

The `lm()` command allows us to combine polynomials, step functions, ordinary cubic splines, and natural cubic splines, not to mention all sorts of other regressors.

We might want to use different basis functions for different predictors.

Figure 12.10 — Natural Cubic Spline with 6 Interior Knots