Regression splines have much to recommend them:

- We estimate them using linear regression, which is easy and makes it easy to include other explanatory variables.
- These can include all sorts of dummy variables and/or polynomials in other predictors.
- Natural cubic splines handle boundaries well.
- But choosing the number and location of the knots involves quite a bit of judgement.

A more automated procedure is to use smoothing splines. The idea is to minimize a sum of squares plus a penalty function that penalizes roughness. The penalty function shrinks the coefficients towards linearity.
The minimization problem is

\[ \min_g \left( \sum_{i=1}^{n} (y_i - g(x_i))^2 + \lambda \int (g''(t))^2 dt \right), \]  

(1)

where \( \lambda \) is a smoothing (tuning) parameter.

The first term in (1) is a **loss function**. It will be small when all the \( g(x_i) \) fit the data well.

The second term in (1) is a **penalty term**. It will be small when the second derivatives of \( g(x_i) \) are small.

If all the second derivatives were zero, then the function \( g(x) \) would be linear.

Thus, when \( \lambda = \infty \), minimizing (1) yields an OLS regression line.

When \( \lambda = 0 \), minimizing (1) yields a perfect fit if every \( x_i \) is unique, because \( g(x) \) can be any function that goes through every data point.
It can be shown that the minimizer of (1) is a natural cubic spline with knots at all of the (unique) $x_i$.

However, it is not the natural cubic spline that we would get if we used `ns()` to create $N$ splines and then used OLS.

Instead, the spline coefficients are shrunk towards the linear least squares fit because of the penalty term.

Recall the Shiller lag, in which second differences entered into the penalty term. That is what happens here.

What we are actually minimizing can be written as

$$(y - N\theta)^\top (y - N\theta) + \lambda \theta^\top \Omega_N \theta. \tag{2}$$

Here $N$ is an $n \times n$ matrix with typical element $N_j(x_i)$, where the $N_j(x)$ form an $n$-dimensional set of basis functions for the family of natural splines.
The matrix $\Omega_N$ can be defined as

$$
(\Omega_N)_{jk} \equiv \int N'_j(t)N'_k(t)dt.
$$

(3)

This involves the second derivatives of the basis functions.

As $\lambda$ increases from 0 to $\infty$, the effective degrees of freedom decreases from $n_u$ to 2.

Here $n_u$ is the number of unique values of $x_i$. In the earnings dataset, there are 63 unique values of age.

When $\text{edf} = n_u$, we are setting $g(x_i) = \bar{y}_i$ for all observations, where $\bar{y}_i$ is the average value of $y_i$ for all observations with the same $x_i$ as the observation $(x_i, y_1)$.

When $\text{edf} = 2$, we are estimating a linear regression by OLS.

Thus for the earnings dataset, the most flexible model would be equivalent to regressing log earnings on 63 age dummies.
In practice, we do not actually need to put knots at every value of $x_i$. We can get away with a much smaller number.

If there are $n_u$ unique values, we actually need a number on the order of $O(n_u^{1/5})$ for $n_u > 49$.

The function `smooth.spline()` will estimate smoothing splines, using a variety of computational tricks.

It actually uses $B$-splines instead of natural cubic splines, for computational reasons.

Although `smooth.spline()` lets you specify $\lambda$, it prefers to use a parameter called `spar` that is typically in the 0-1 interval.

There is a relationship between `spar` and $\lambda$, but it is quite complicated.

LOO cross-validation can be used, and it works very efficiently; see Subsection 7.5.2. But it only works for unique $x_i$.

Figure 12.1 shows the results of estimating the relationship between age and the log of earnings using `smooth.spline()`.
Figure 12.1 — Smoothing Splines for Log Earnings

![Smoothing Splines for Log Earnings](image-url)
The red curve was obtained by letting the function pick \( \text{spar} \). It chose 0.669875 using generalized cross-validation.

It refused to use cross-validation because the \( x_i \) were not unique. Instead, it used \textbf{generalized cross-validation}; see below.

The blue curve used \( \text{spar} = 0.2 \), which is evidently much too small.

The purple curve used \( \text{spar} = 0.8 \), which appears to be somewhat too big. The fit is noticeably higher at both ends.

How to obtain confidence intervals will be discussed in the context of generalized additive models.

The main advantage of using smoothing splines is that there is no need to choose the number and location of the knots.

I would be tempted to pick the knots so that the natural cubic spline and smoothing spline (using GCV) yield similar results.
Generalized Cross-Validation

**Generalized cross-validation** is a convenient alternative to leave-one-out cross-validation for linear models with squared-error loss, including smoothing splines.

Any linear fitting method can be written as

$$\hat{y} = Sy$$

for some $n \times n$ matrix $S$.

Such methods include OLS, where $S = P_X$, and linear regression on basis functions, where $S = P_B$, and $B$ is a matrix of basis functions.

They also include methods that use quadratic smoothing, such as ridge regression and smoothing splines.

In all such cases, the **effective number of parameters**, or **effective degrees of freedom**, is simply the trace of $S$. 
For many linear models, including smoothing splines,

\[ \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{g}^{-i}(x_i))^2 = \frac{1}{n} \sum_{i=1}^{n} \left( \frac{y_i - \hat{g}(x_i)}{1 - S_{ii}} \right)^2, \] (5)

where \( S_{ii} \) is the \( i \)th diagonal element of \( S \); we will define \( S_\lambda \) below.

Recall that, for OLS, \( S_{ii} \) is the \( i \)th diagonal element of the "hat matrix" \( P_X \). In that case, we have seen (5) before.

It can be shown that, for linear regression models,

\[ \text{Tr}(S)\sigma^2 = \sum_{i=1}^{n} \text{Cov}(\hat{y}_i, u_i). \] (6)

When there is no covariance between between \( \hat{y}_i \) and \( u_i \), the fitting method does not cause \( \text{Var}(\hat{y}_i) \) to be less than \( \text{Var}(y_i) \).

In other words, fitting the model does not use any degrees of freedom.
Consider OLS regression, for which the fitted values are $\hat{y} = P_X y$. Thus the sample covariance of $u$ and $\hat{y}$ for a correctly specified linear regression model is

$$u^T P_X y = u^T P_X u + u^T X \beta.$$  \hfill (7)

The second term here has expectation zero, and the expectation of the first term is $\sigma^2 \text{Tr}(P_X) = p \sigma^2$. It follows that

$$\sum_{i=1}^n \text{Cov}(\hat{y}_i, u_i) = p \sigma^2.$$  \hfill (8)

This suggests that, whether or not a fitting method is linear, it is reasonable to define edf as

$$\text{edf}(\hat{y}) = \frac{1}{\sigma^2} \sum_{i=1}^n \text{Cov}(\hat{y}_i, u_i).$$  \hfill (9)
The GCV approximation to (5) is

\[
GCV(\hat{g}) = \frac{1}{n} \sum_{i=1}^{n} \left( \frac{y_i - \hat{g}(x_i)}{1 - \text{Tr}(S)/n} \right)^2 = \frac{1}{n} \sum_{i=1}^{n} \left( \frac{y_i - \hat{g}(x_i)}{1 - \text{edf}/n} \right)^2.
\]  

(10)

For OLS regression, edf = p.

For other methods, edf < p, perhaps much less if there is a lot of regularization.

We shall see what S is for smoothing splines shortly.

Notice that (10) blows up every residual by the same proportion. Actual cross-validation blows up some of them more than others. This suggests that generalized cross-validation may not perform well when some observations are much more influential than others.
The fitted values for the smoothing spline that minimizes (2) can be written as
\[ \hat{g} = N(N^\top N + \lambda \Omega_N)^{-1}N^\top y = S_\lambda y \]  
(11)
This looks a lot like the vector fitted values for ridge regression. In that case, \( N \) is replaced by \( X \), and \( \Omega_N \) is replaced by \( I \).

Here \( S_\lambda \) is a **smoother matrix**. It is a linear operator with full rank \( n \), and it is symmetric and positive semidefinite.

Unlike the hat matrix \( P_X \), the smoother matrix \( S_\lambda \) is not idempotent:
\[ S_\lambda S_\lambda \preceq S_\lambda, \]  
(12)
which means that the difference between \( S_\lambda \) and \( S_\lambda S_\lambda \) is a positive semidefinite matrix.

Thus the smoother matrix \( S_\lambda \) shrinks whatever it multiplies.

There is only one tuning parameter, \( \lambda \), to choose.
If we prefer, we could pick edf and use it to solve for $\lambda$.

Or we can use spar, which is what `smooth.spline()` prefers.

The variance of $\hat{g}$ is

$$S_\lambda \text{Var}(y) S_\lambda^\top \propto S_\lambda S_\lambda.$$  \hfill (13)

If the disturbances are independent and homoskedastic, then $\text{Var}(y) = \sigma^2 I$.

The bias of $\hat{g}$ is

$$E(\hat{g}) - g = S_\lambda g - g.$$  \hfill (14)

Thus the mean squared error is

$$\sigma^2 S_\lambda S_\lambda + (S_\lambda - I)gg^\top (S_\lambda - I)^\top.$$  \hfill (15)

Making $\lambda$ too small (edf too big) causes excessive variance, and making $\lambda$ too big (edf too small) causes excessive bias.
Smoothing Splines for Logit

Instead of \( g(x) \) denoting the fitted value conditional on \( x \), it now denotes the log of the odds:

\[
\log \frac{\Pr(y = 1 \mid x)}{\Pr(y = 0 \mid x)} = g(x).
\] (16)

Therefore

\[
p(x) \equiv \Pr(y = 1 \mid x) = \frac{\exp(g(x))}{1 + \exp(g(x))}.
\] (17)

The penalized criterion function is based on the loglikelihood function:

\[
\sum_{i=1}^{n} \left( y_i \log (p(x_i)) + (1 - y_i) \log (1 - p(x_i)) \right) - \frac{1}{2} \lambda \int (g''(t))^2 dt.
\] (18)

This is minimized by making \( g \) a natural cubic spline with knots at every unique value of \( x \).
The prediction at $x$ is simply

$$\hat{g}(x) = \sum_{j=1}^{n} \hat{\theta}_j N_j(x).$$  \hspace{1cm} (19)

Of course, if we set $\lambda = 0$, maximizing (18) would yield the maximum likelihood logit estimates.

Maximizing (18) requires nonlinear optimization, which would be prohibitively expensive for large $n$ if we actually used $n$ basis functions.

But, as in the regression case, we can get away with far fewer than $n$ basis functions when $n$ is large.

Of course, we could also use ordinary or natural cubic splines with a binary dependent variable.

We would need to choose how many knots and their locations. The basis functions could then be constructed using the `bs()` or `ns()` functions in the usual way.
Smoothing Splines versus Kernel Regression

It is perhaps not obvious, but smoothing splines, like kernel regression, are \textit{local} methods.

The smoothing matrix $S_\lambda$ gives large weights to points near $x_0$ and small weights to points far from it.

Figure 12.2, which is taken from Chapter 5 of ESL, illustrates this. The colorful square shows $S_\lambda$ for a particular problem with around 125 observations, where the $x_i$ have been sorted.

The “equivalent kernels” on the right show the implied weights for a number of rows of the smoother matrix.

These equivalent kernels look very much like real kernels, although they change a bit as $x_0$ changes, which is a good thing. They sometimes take on small negative values.
Figure 12.2 — Smoothing Splines vs. Kernels

Equivalent Kernels
Comparing Fully Automated Methods

Smoothing splines with $\lambda$ chosen by some sort of cross-validation and kernel regression with $h$ chosen in the same way are both fully automatic.

How do they compare for the earnings data?

The smoothing spline chosen by `smooth.spline` is in red, and the locally linear kernel regression (Epanechnikov kernel) chosen by `npreg` is in blue.

The optimal `spar` for the smoothing spline was 0.670. This took just 0.0023 seconds.

The optimal bandwidth ($h$) for the kernel regression was 2.846. This took 5.10 seconds.

For large $n$, smoothing splines should be much cheaper than kernel regression.
Figure 12.3 — Smoothing Spline vs. Kernel Regression