5601 Notes: Smoothing

Charles J. Geyer

December 4, 2013

Contents

1 Web Pages

This handout accompanies the web pages

```
http://www.stat.umn.edu/geyer/5601/examp/smoo.html
http://www.stat.umn.edu/geyer/5601/examp/smootoo.html
```

2 The General Smoothing Problem

In simple linear regression, the standard assumptions are that the data are of the form (x_i, y_i), $i = 1$, $\ldots$, n. We are interested in being able to predict y_i values given the corresponding x_i values. For this reason we treat x_i as non-random. If the x_i are actually random, we say we are conditioning on their observed values, which is the same thing as treating them as non-random. The conditional distribution of the y_i given the x_i is determined by

$$y_i = \alpha + \beta x_i + e_i \tag{1}$$

where α and β are unknown parameters (non-random but unknown constants) and the e_i are IID mean zero normal random variables.

More generally, using multiple linear regression, we can generalize the model (1) to

$$y_i = \alpha + \beta_1 g_1(x_i) + \cdots + \beta_k g_k(x_i) + e_i \tag{2}$$

where g_1, $\ldots$, g_k are any known functions and the errors e_i are as before.

For example, polynomial regression is the case where the g_i are monomials

$$g_i(x) = x^i, \qquad i = 1, \ldots, k.$$

But multiple regression works with *any* functions g_i so long as they are *known* not *estimated*, that is, chosen by the data analyst without looking at the data rather than somehow estimated from the data (only the regression parameters α, β_1, $\ldots$, β_k are estimated from the data).

Even more generally (using we don't yet know what) we can generalize the model (2) to

$$y_i = g(x_i) + e_i \tag{3}$$

where g is an *unknown* function and the errors e_i are as before. Unlike the jump from (1) to (2), which involves only a quantitative change from 2 to $k + 1$ regression coefficients, the jump from (2) to (3) involves a qualitative change from $k + 1$ real parameters α, β_1, $\ldots$, β_k to an unknown "parameter" that is a whole function g.

Theoretical statisticians often call such a g an infinite-dimensional parameter because no finite-dimensional parameter vector $\boldsymbol{\theta}$ can parameterize all possible functions, that is, we cannot write the function $g(x)$ as $g_{\boldsymbol{\theta}}(x)$ for some finite-dimensional parameter $\boldsymbol{\theta}$. In particular, we cannot write

$$g_{\boldsymbol{\theta}}(x) = \alpha + \beta_1 g_1(x) + \cdots + \beta_k g_k(x)$$

where $\boldsymbol{\theta} = (\alpha, \beta_1, \ldots, \beta_k)$ where g_1, ..., g_k are *known* functions. If we could to that, this would reduce (3) to a special case of (2). But we can't do that, so (3) is not a special case of (2).

Those who have had advanced calculus may know that we can generally write large classes of functions (for example all continuous functions on a bounded interval) as infinite series

$$g_{\boldsymbol{\theta}}(x) = \alpha + \sum_{i=1}^{\infty} \beta_i x^i$$

where now $\boldsymbol{\theta} = (\alpha, \beta_1, \beta_2, \ldots)$ is an infinite-dimensional vector. This is the sense in which the parameter vector is infinite-dimensional. However, we won't actually use this parameterization or any parameterization. The actual way statisticians estimate the general regression model (3) doesn't explicitly use any parameters.

Summary of Section 2. In all forms of regression there are two kinds of assumptions: (i) those about the regression function and (ii) those about the error distribution.

It is possible to be nonparametric about either (i) or (ii). Robust regression as implemented by the R functions `lmsreg` and `ltsreg` is nonparametric about the error distribution. These methods are robust against non-normality of the errors.

In this handout and the accompanying web pages we are making the standard parametric error assumption IID mean zero normal errors. Instead we are nonparametric about (i). The *regression function* is the statisticians name for the conditional expectation $E(Y \mid x)$ thought of as a function of the conditioning variable x.

From (3) we have

$$E(y_i \mid x_i) = g(x_i)$$

because the errors have mean zero and are independent of the x_i. So g is the regression function. We are allowing it to be completely arbitrary (or almost arbitrary, more on this below). That's nonparametric.

3 Some Smoothers

The basic idea of all *smoothers* is that the unknown regression function g is smooth, meaning that $|g(x) - g(y)|$ is small when $|x - y|$ is small. From calculus, we know that these quantities are involved in the derivative

$$g'(x) = \lim_{y \to x} \frac{g(y) - g(x)}{y - x}$$

Thus we may assume that the unknown regression function is differentiable or even differentiable several times. But differentiability assumptions are not strictly necessary. Some smoothing methods assume derivatives, and some don't.

What all smoothing methods have in common is making some use of (x_i, y_i) pairs with x_i near x in estimating $g(x)$. Only smoothness of g can justify this.

If g were arbitrarily rough, so that $g(x)$ is completely unrelated to $g(x')$ for $x \neq x'$, then the only sensible estimate would be data itself, that is, use y_i as our estimate of $g(x_i)$ and don't even try to estimate $g(x)$ for x not equal to any x_i. This hardly qualifies as doing statistics, since it involves no calculation at all, but without some assumptions about the behavior of the regression function g, there is no justification for anything else. Hence assumptions about smoothness of g. See Section 4.2.2 below for more on this.

3.1 Running Mean Smoother

A *running mean smoother* averages the values y_i such that $|x_i - x| \leq h$ for some fixed h. For now, we will just treat the number h as a magic number pulled out of the air. Later we will see that choosing h or the number or numbers analogous to h in other methods (and every known smoothing method has some such numbers that must be chosen) can be done using statistical methodology. A large part of these notes, culminating in Section 5, is about this issue.

The number h is sometimes called the *smoothing parameter* but it is more often referred to by the cutesy name *bandwidth*, which is a metaphorical use of a term from communications theory. The bandwidth of a radio signal is the range of frequencies used. Although the frequency of a radio station is usually given as a single number, such as KSJN FM 99.5 (the units are megahertz, abbreviated MHz), but the station actually uses the "band" 0.2 MHz wide (from 99.4 to 99.6). The wider the band, the more information that can be transmitted through the communication channel. That's why FM radio has higher fidelity than AM radio, which uses only 5 kHz = 0.005 Mhz bandwidth. It's not clear what this metaphor has to do with smoothing, but it sounds high tech and statisticians like to use it.

For an example we will use the cholostyramine data set provided with the R `bootstrap` package and described in Section 7.3 of Efron and Tibshirani (1993).

The following R creates the data

```
> library(bootstrap)
> compliance <- cholost$z
> improvement <- cholost$y
> plot(compliance, improvement)
```

and makes the plot shown in our Figure 1 and in Figure 7.5 in Efron and Tibshirani (1993). The figures should (and do) look the same except for trivial details (like the shape of plotting points).

The following code adds a running mean smoother to the plot (Figure 2).

```
> plot(compliance, improvement)
> bw <- 10
> lines(ksmooth(compliance, improvement, bandwidth = bw))
```

3.2 General Kernel Smoothing

One of the problems with the simple running mean smoother, so problematic that no one uses it on important data (other fancier smoothers are preferred) is that its estimate of the supposedly smooth function g isn't very smooth. The reason it isn't smooth is that the operation of choosing which y_i contribute to the estimate of $g(x)$ is all or nothing. A given y_i contributes to the estimate for some x and not to other x. It contributes if and only if $|x_i - x| \leq h$. This makes the "smooth" estimate actually a *discontinuous* function of x. No good!

An obvious choice is to replace all-or-nothing choice with partial use. We replace the ordinary average with a weighted average. Let w be an arbitrary known, fixed non-negative function that is symmetric about zero called the

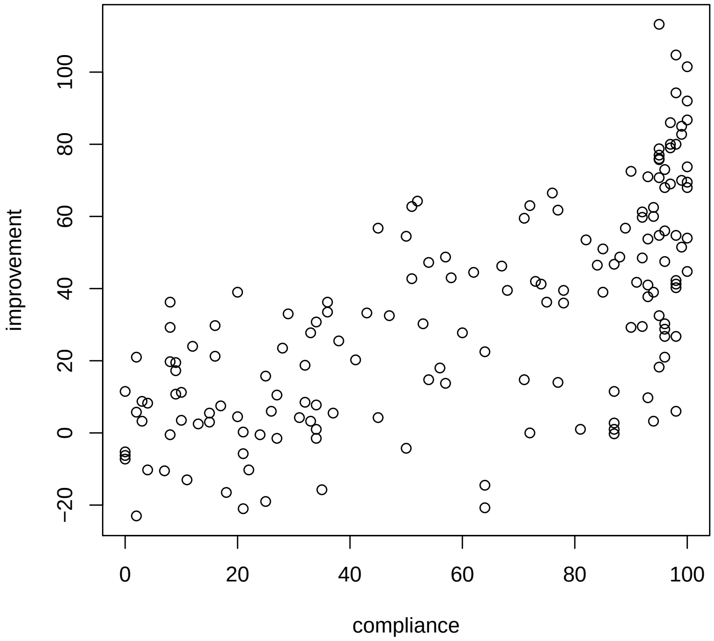

Figure 1: Scatterplot for the cholostyramine data.

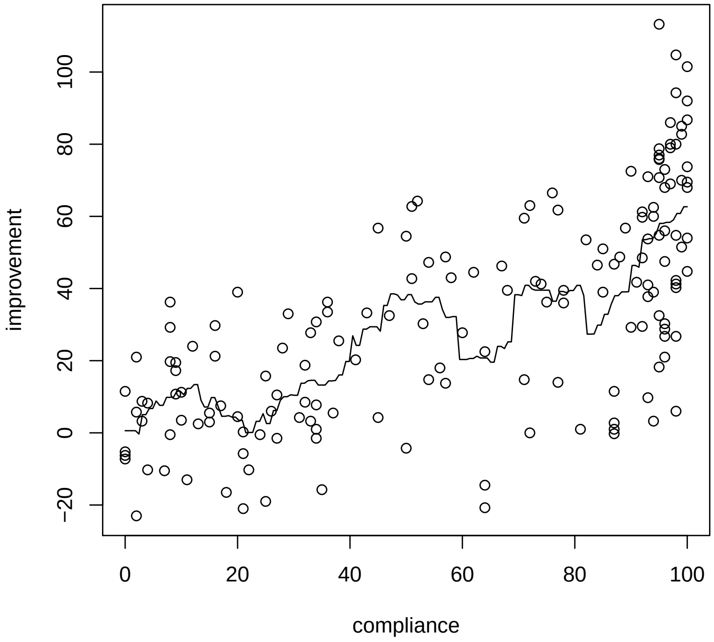

Figure 2: Scatterplot with running mean smooth (bandwidth 10) for the cholostyramine data.

kernel. We estimate $g(x)$ by the *kernel regression estimate*

$$\hat{g}(x) = \frac{\displaystyle\sum_{i=1}^{n} y_i w\left(\frac{x - x_i}{h}\right)}{\displaystyle\sum_{i=1}^{n} \quad w\left(\frac{x - x_i}{h}\right)} \tag{4}$$

(we write the denominator with that peculiar space to emphasize that formulas for the numerator and denominator are exactly the same except for the y_i in the numerator that is missing in the denominator).

Note that each $\hat{g}(x)$ is a *weighted average* of the y_i, which is the operation

$$p_1 y_1 + \cdots + p_n y_n \tag{5}$$

where the p_i are non-negative constants that sum to one. We put (4) in the form (5) by defining the "weights"

$$p_i = \frac{w\left(\frac{x - x_i}{h}\right)}{\displaystyle\sum_{j=1}^{n} w\left(\frac{x - x_j}{h}\right)}$$

Note that a running mean smoother is the special case of a general kernel smoother (4) where the kernel function is a simple "box" function

$$w(x) = \begin{cases} 0, & x < -1 \\ 1, & -1 \le x \le 1 \\ 0, & 1 < x \end{cases}$$

But if we change, for example, to the "gaussian" kernel

$$w(x) = \exp(-x^2/2)$$

we get much smoother behavior.

The kernel estimate $\hat{g}(x)$ of the true unknown regression function $g(x)$ given by (4) obviously is smooth if and only if the kernel $w(x)$ is smooth. For example, $\hat{g}$ is continuous if and only if w is continuous, $\hat{g}$ is differentiable if and only if w is differentiable, and $\hat{g}$ is differentiable k times if and only if w is differentiable k times.

Note finally that "bandwidth" is not comparable between different kernels. For fixed w, it is always true that large h gives smoother smooths and

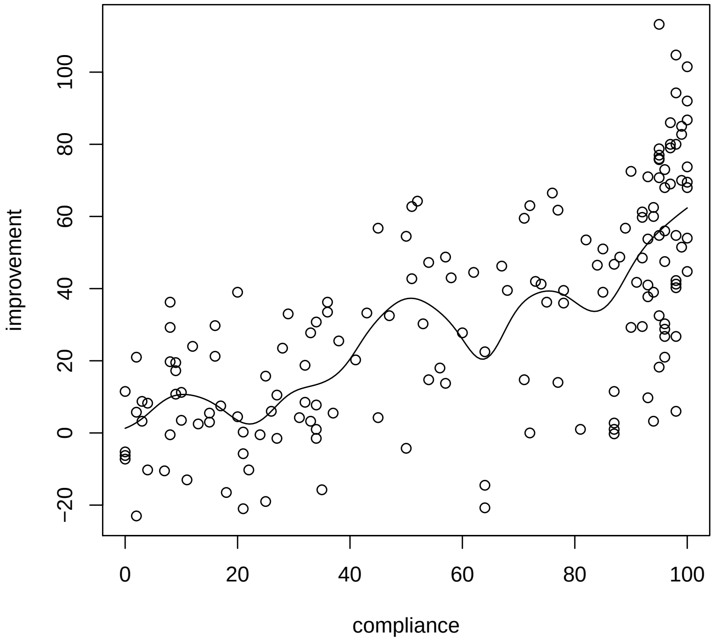

Figure 3: Scatterplot with gaussian kernel smooth (bandwidth 10) for the cholostyramine data.

small h gives rougher smooths. But consider two kernels w_1 and w_2 related by $w_1(x) = w_2(3x)$. Then with smoothing parameters (bandwidths) related by $h_1 = 3h_2$, they give the same smooth. So clearly bandwidth in smoothing is a rather vague metaphor.

The following code adds a kernel smoother with gaussian kernel to the plot (Figure 3).

```
> plot(compliance, improvement)
> lines(ksmooth(compliance, improvement, bandwidth = bw, kernel = "normal"))
```

Observe how the estimated regression function shown in Figure 3 is much, much smoother than the estimated regression function shown in Figure 2.

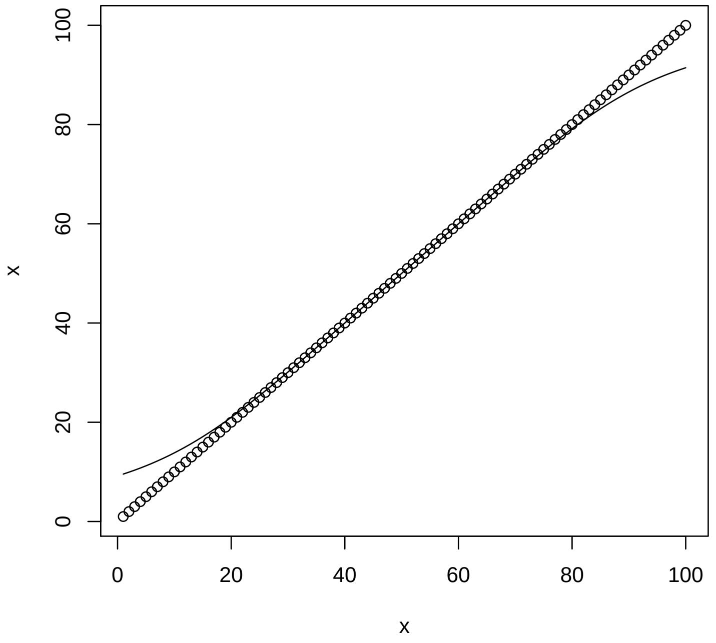

Figure 4: Gaussian kernel smooth (bandwidth 30) for noiseless pseudo-data showing edge effects.

3.3 Local Polynomial Smoothing

An issue with kernel smoothing (including running means) is that these methods have bad behavior at the edges of the plot. Observe what kernel smoothing does with perfectly regular, noiseless, linear data. (Figure 4).

```
> x <- seq(1:100)
> plot(x, x)
> bw <- 30
> lines(ksmooth(x, x, bandwidth = bw, kernel = "normal"))
```

An improvement to kernel smoothing is *local polynomial smoothing*, which does the following. To estimate $g(x)$ fit a polynomial to the data (x_i, y_i),

$i = 1, \ldots, n$ using weighted least squares with weights

$$w_i = w\left(\frac{x - x_i}{h}\right)$$

where w is a kernel function and h the smoothing parameter. Then use the predicted value at x from this regression as $\hat{g}(x)$.

To be a bit more concrete in our description, suppose we choose to use a first degree polynomial (that is, use *linear* regression), then for each x we find $\hat{\alpha}(x)$ and $\hat{\beta}(x)$ that minimize the weighted residual sum of squares

$$\sum_{i=1}^{n} w\left(\frac{x - x_i}{h}\right)(y_i - \alpha - \beta x_i)^2 \tag{6}$$

and set

$$\hat{g}(x) = \hat{\alpha}(x) + \hat{\beta}(x) \cdot x \tag{7}$$

Note that we have to do this for each x for which we wish to evaluate $\hat{g}(x)$.

Local polynomial smoothing, because it is fitting lines locally would do the right thing to the pseudo-data in Figure 4. There would be no curving of the smooth at the edges. The smooth would follow the points.

Let's try local polynomial smoothing on the cholostyramine data (Figure 5).

```
> library(KernSmooth)
> plot(compliance, improvement)
> bw <- 5
> lines(locpoly(compliance, improvement, bandwidth = bw))
```

3.4 Smoothing Splines

In this section we start over with a completely different rationale for smoothing. We are not going to use anything remotely resembling a kernel or "local" use of any well-known statistical procedure. We are going to use a completely different approach of using penalty functions.

The method of least squares finds the best (according to certain criteria) model within a certain parametric class to fit the data. But if the parametric class is too "big" (generally, whenever there are at least as many parameters as data points) we just fit the data perfectly. The data are the "smooth" and effectively we do no analysis at all.

For example, if we have n data points and fit a polynomial of degree $n - 1$ (which has n parameters including the intercept). We get a perfect fit (Figure 6).

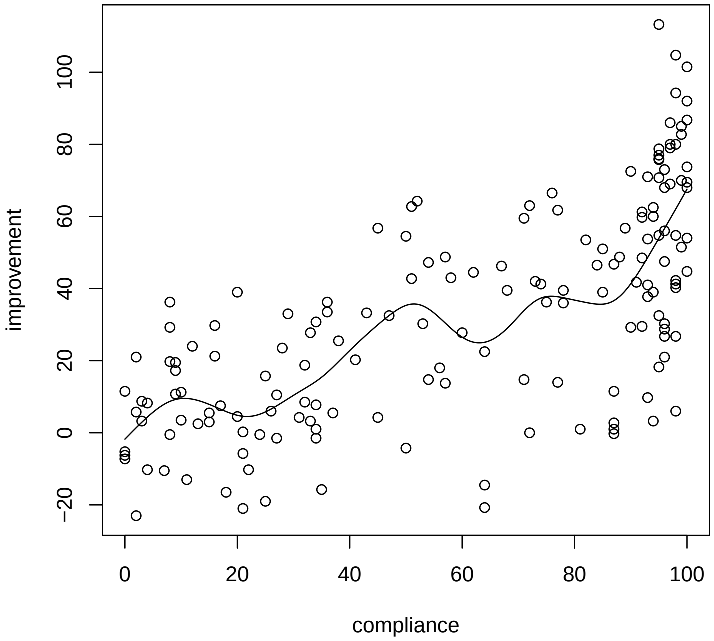

Figure 5: Scatterplot with local polynomial smooth (bandwidth 5) for the cholostyramine data.

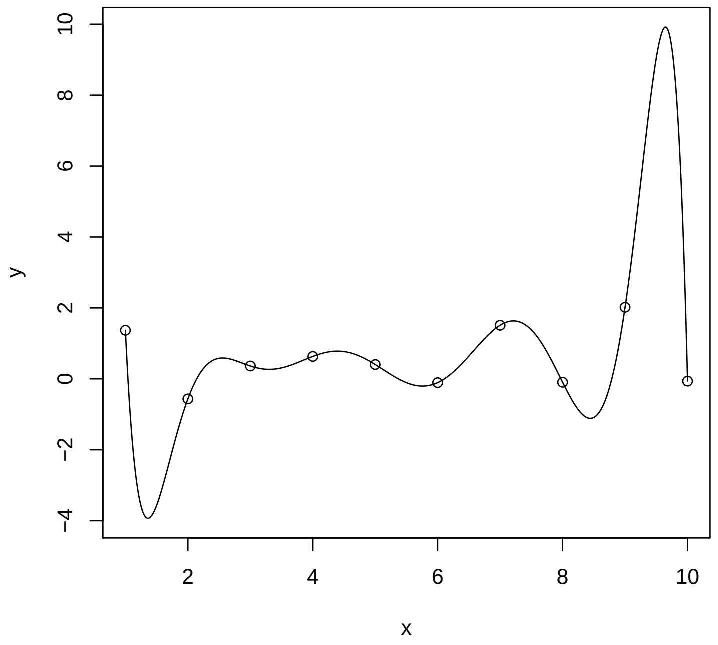

Figure 6: Polynomial interpolating data points.

```
> n <- 10
> x <- seq(1, n)
> set.seed(42)
> y <- rnorm(n)
> out <- lm(y ~ poly(x, n - 1))
> xx <- seq(min(x), max(x), length = 1001)
> yy <- predict(out, newdata = data.frame(x = xx))
> plot(x, y, ylim = range(yy))
> curve(predict(out, newdata = data.frame(x = x)), add = TRUE, n = 1001)
```

And, of course, this "perfect" fit is perfectly useless. At $x = x_i$ for some i it just predicts the data value $\hat{g}(x_i) = y_i$. At other points, it does give nice smooth predictions, but we don't believe these wild oscillations. In fact, the data are simulated from the model with constant regression function and

$$\hat{g}(x) = \bar{y}_n, \qquad \text{for all } x$$

would be a much, much better estimate than the curve in Figure 6.

Thus the method of least squares gives ridiculous results when applied to a model that is too big. One cure is to "penalize" models that seem less reasonable. The penalty function that leads to smoothing splines penalizes integrated squared second derivative. The method of smoothing splines chooses the g that minimizes "residual sum of squares plus penalty"

$$\sum_{i=1}^{n} \bigl[y_i - g(x_i)\bigr]^2 + \lambda \int_{-\infty}^{\infty} g''(x)^2 \, dx \tag{8}$$

where λ is a fixed positive number, called the *smoothing parameter* that plays the role that bandwidth plays in kernel smoothing.

It can be shown that the g that minimizes (8) is always a *natural cubic spline* with *knots* at the observed predictor values. Let $x_{(1)}$, $\ldots$, $x_{(k)}$ be the distinct ordered predictor values, that is, each x_i is some $x_{(j)}$ and

$$x_{(1)} < x_{(2)} < \cdots < x_{(k)}.$$

Then a natural cubic spline has the following properties.

- On each interval between knots $(x_{(j)}, x_{(j+1)})$ the function g is cubic

$$g(x) = \alpha_j + \beta_j x + \gamma_j x^2 + \delta_j x^3, \qquad x_{(j)} < x < x_{(j+1)} \tag{9}$$

- On each semi-infinite interval outside the knots the function g is linear

$$\begin{aligned} g(x) &= \alpha_0 + \beta_0 x, \qquad & x < x_{(1)} \\ g(x) &= \alpha_k + \beta_k x, \qquad & x > x_{(k)} \end{aligned}$$

- This specifies g by $4k$ parameters (α's, β's, γ's, and δ's).

- At knots g, g', and g'' are continuous.

- This imposes $3k$ conditions, which are equations that are linear in the parameters. Hence the $4k$ original parameters can be expressed as linear functions of k free parameters.

This has the following important consequence. The functions g and g'' are linear in the parameters (original or free). Thus the penalized least squares objective function (8) is quadratic in the parameters. To minimize a quadratic function, one finds a point where all the partial derivatives are zero. All partial derivatives of (8) are linear in the parameters. Hence the minimization is carried out by solving linear equations. Moreover, all partial derivatives of (8) are also linear in the data. Thus the penalized least squares estimates are linear functions of the data vector $\mathbf{y} = (y_1, \ldots, y_n)$ and so is the predicted value vector $\hat{\mathbf{y}}$ because $g(x)$ is linear in the parameters.

As we adjust λ between zero and infinity the smoothing spline goes from very rough to very smooth. At $\lambda = \infty$ we only have (8) finite if the integral is zero, which happens only if $g''(x) = 0$ for all x, which happens only if $g(x)$ is a linear function. Thus this case is the same as simple linear regression. At $\lambda = 0$ the smoothing spline interpolates the data (Figure 7, which is qualitatively like Figure 6 but different in detail).

```
> out <- spline(x, y, n = 1001, method = "natural")
> plot(x, y, ylim = range(out$y))
> foo <- par("usr")
> out <- spline(x, y, n = 1001, method = "natural",
+     xmin = foo[1], xmax = foo[2])
> lines(out)
> outr <- lm(y ~ x)
> abline(outr, lty = 2)
```

Note that the oscillations in Figure 7 are much less wild than the oscillations in Figure 6. That is an important reason why people use smoothing splines instead of polynomials. Both can be fitted by multiple regression, both have k free parameters (regression coefficients) for k knots, but splines do a better job fitting most data.

Figure 7 only shows the extreme behavior ($\lambda = 0$) of smoothing splines. In real applications we are interested in intermediate values of λ. Figure 8 is a smoothing spline for the cholostyramine data. It is produced by the R statements

```
> plot(compliance, improvement)
> df <- 10
> lines(smooth.spline(compliance, improvement, df = df))
```

in which the `df` argument specifies the smoothness in terms of “effective degrees of freedom”, which is explained in the following section.

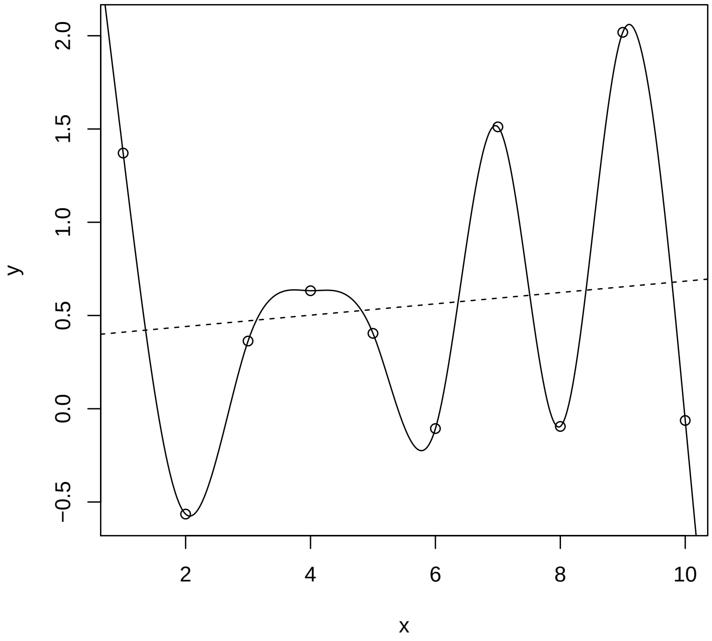

Figure 7: Natural cubic spline interpolating data points (solid curve) and simple linear regression line (dashed line).

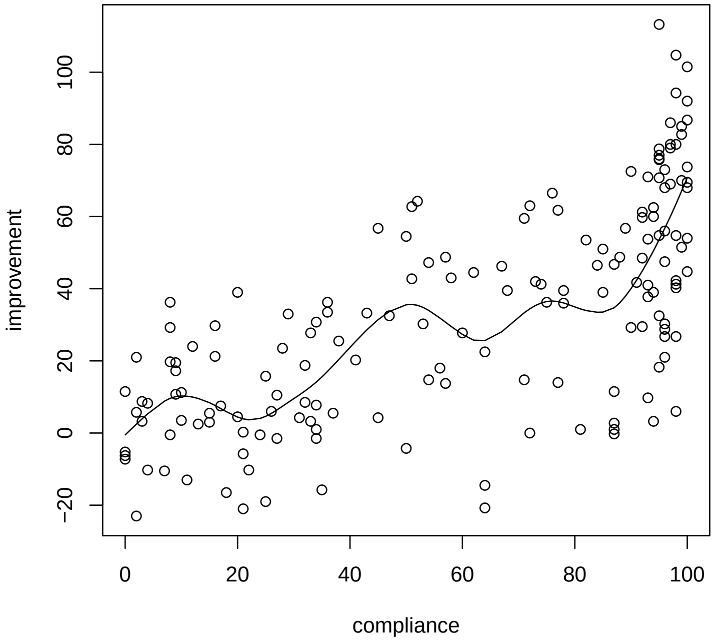

Figure 8: Scatterplot with smoothing spline (effective degrees of freedom = 10) for the cholostyramine data.

4 Some Theory

The theory in this section closely follows the presentation in Chapters 2 and 3 of Hastie and Tibshirani (1990). although it leaves out a lot of what they say and fills in a lot of details. Students interested in more information about smoothing should look at Hastie and Tibshirani (1990). It's a good book.

4.1 Linear Smoothers

Every smoother we have discussed so far is a so-called *linear smoother*. What this means is that the vector $\hat{\mathbf{y}}$ of predicted values at the observed predictor values is a linear function of the data vector $\mathbf{y}$. This is obvious for kernel smoothers. Equation (4) gives predicted values using a formula that is linear in the y_i (the components of $\mathbf{y}$). This is less obvious for local polynomial smoothers and smoothing splines, but we saw in the preceding section that we have $\hat{\mathbf{y}}$ a linear function of $\mathbf{y}$ for smoothing splines, and the same also holds for local polynomial smoothing for the following reasons. The objective function (6) is quadratic in the residuals $y_i - \alpha - \beta x_i$, hence its partial derivatives with respect to α and β are *linear* in the residuals, hence simultaneous linear equations to be solved for α and β that are also linear in the y_i. Hence the solutions $\hat{\alpha}(x)$ and $\hat{\beta}(x)$ are also linear in the y_i. Then applying (7) we see that the $\hat{y}_i$ must be linear in the y_i.

In short, for all of these smoothers we can write

$$\hat{\mathbf{y}}_\lambda = \mathbf{S}_\lambda \mathbf{y} \tag{10}$$

where (as we defined them before) $\mathbf{y}$ is the data vector and $\hat{\mathbf{y}}_\lambda$ is the predicted values vector and (newly defined here) $\mathbf{S}_\lambda$ is a matrix, called the *smoother matrix*, that depends on some smoothing parameter λ and also on the data vector $\mathbf{x} = (x_1, \ldots, x_n)$ although this is not explicitly indicated by the notation. The smoother matrix $\mathbf{S}_\lambda$ can (and typically does) depend on λ and $\mathbf{x}$ in a highly non-linear way. The only linearity we have in (10) is linearity in $\mathbf{y}$.

It is of some interest that multiple regression itself fits into this scheme. In regression theory the smoother matrix is called the "hat matrix" because it puts the hat on $\mathbf{y}$ and has the form

$$\mathbf{H} = \mathbf{X}(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T \tag{11}$$

where $\mathbf{X}$ is the so-called "design" or "model" matrix which has as rows the values of the predictor variables, one row per case. Thus ordinary multiple regression is a special case of "smoothing" where there is no adjustable smoothing parameter. The amount of "smoothness" is just built into the model somehow.

4.2 Distribution Theory

4.2.1 Assumptions

The usual "assumptions" for smoothing are the same as those for linear regression. We assume

$$\mathbf{y} = \boldsymbol{\mu} + \mathbf{e} \tag{12a}$$

where

$$\mathbf{e} \sim \text{Normal}(\mathbf{0}, \sigma^2 \mathbf{I}) \tag{12b}$$

In (12a) the vector $\boldsymbol{\mu} = (\mu_1, \ldots, \mu_n)$ is the regression function evaluated at the "design points"

$$\mu_i = g(x_i)$$

and in (12b) the matrix $\mathbf{I}$ is the $n \times n$ identity matrix, so (12b) says that the components of the error vector $\mathbf{e}$ are IID Normal$(0, \sigma^2)$.

4.2.2 Bias

The (nonparametric) parameter $\boldsymbol{\mu}$ is the unknown parameter of interest. The variance parameter σ^2 is a nuisance parameter.

The parameter of interest is the (conditional) mean of $\mathbf{y}$

$$E(\mathbf{y}) = \boldsymbol{\mu}.$$

The estimate of this parameter is the "fitted values" vector $\hat{\mathbf{y}}_\lambda$. The expectation of this estimate is

$$E(\hat{\mathbf{y}}_\lambda) = E\bigl(\mathbf{S}_\lambda \mathbf{y}\bigr) = \mathbf{S}_\lambda \boldsymbol{\mu}$$

The difference is the *bias* of the estimator

$$\mathbf{b}_\lambda = E(\hat{\mathbf{y}}_\lambda) - \boldsymbol{\mu} = \bigl(\mathbf{S}_\lambda - \mathbf{I}\bigr)\boldsymbol{\mu}$$

Generally, smoothers are *biased.* The bias vector $\mathbf{b}_\lambda$ is generally not equal to zero. But (this is a very important point). When you think nonparametrically, trying to be unbiased is the stupidest thing you can do. An unbiased smoothing method would have to satisfy

$$\boldsymbol{\mu} = \mathbf{S}_\lambda \boldsymbol{\mu}, \qquad \text{for all } \boldsymbol{\mu}$$

and the only matrix $\mathbf{S}_\lambda$ that has that property is the the identity matrix $\mathbf{S}_\lambda = \mathbf{I}$. But that smoother just "estimates" $\hat{\mathbf{y}}_\lambda = \mathbf{y}$. This isn't data analysis. When your "estimate" is the raw data, you're not really doing any statistics.

4.2.3 Variance

The variance of $\mathbf{y}$ is variance of $\mathbf{e}$, that is

$$\operatorname{var}(\mathbf{y}) = \sigma^2 \mathbf{I}$$

From a general theorem about the variance of a linear transformation of a random vector

$$\operatorname{var}(\hat{\mathbf{y}}_\lambda) = \sigma^2 \mathbf{S}_\lambda \mathbf{S}_\lambda^T \tag{13}$$

where the superscript T indicates the matrix transpose operation. Also of interest is the variance of the residual vector

$$\begin{aligned} \operatorname{var}(\mathbf{y} - \hat{\mathbf{y}}_\lambda) &= \operatorname{var}\bigl((\mathbf{I} - \mathbf{S}_\lambda)\mathbf{y}\bigr) \\ &= \sigma^2 (\mathbf{I} - \mathbf{S}_\lambda)(\mathbf{I} - \mathbf{S}_\lambda)^T \\ &= \sigma^2 (\mathbf{I} - \mathbf{S}_\lambda - \mathbf{S}_\lambda^T + \mathbf{S}_\lambda \mathbf{S}_\lambda^T) \end{aligned} \tag{14}$$

4.2.4 Variance Estimate

We need an estimate of σ^2. In multiple regression, we divide the residual sum of squares

$$\|\mathbf{y} - \hat{\mathbf{y}}_\lambda\|^2 = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 \tag{15}$$

by the "degrees of freedom for error" to get an unbiased estimate of σ^2.

In smoothing, things are not so simple. Let us calculate the expectation of (15) and see what we can do with it. First recall from theory that for any random variable W whatsoever

$$\operatorname{var}(W) = E(W^2) - E(W)^2$$

or, equivalently,

$$E(W^2) = \operatorname{var}(W) + E(W)^2.$$

Applying this with

$$W = y_i - \hat{y}_{\lambda,i}$$

we see that $E(W) = b_{\lambda,i}$ and $\operatorname{var}(W)$ is given by the i,i-th element of the variance matrix (14).

The sum of the diagonal elements of a matrix is called its *trace*. We denote the trace of a matrix $\mathbf{A}$ by $\operatorname{tr}(\mathbf{A})$. With this notation and the ideas above, we get

$$\begin{aligned} E\bigl\{\|\mathbf{y} - \hat{\mathbf{y}}_\lambda\|^2\bigr\} &= \sigma^2 \operatorname{tr}(\mathbf{I} - \mathbf{S}_\lambda - \mathbf{S}_\lambda^T + \mathbf{S}_\lambda \mathbf{S}_\lambda^T) + \mathbf{b}_\lambda^T \mathbf{b}_\lambda \\ &= \sigma^2 \bigl(n - 2\operatorname{tr}(\mathbf{S}_\lambda) + \operatorname{tr}(\mathbf{S}_\lambda \mathbf{S}_\lambda^T)\bigr) + \mathbf{b}_\lambda^T \mathbf{b}_\lambda \end{aligned} \tag{16}$$

Now we have a slight problem carrying out our program. No multiple of (15) is an unbiased estimate of σ^2. The best we can do is ignore the bias term and divide (15) by

$$\mathrm{df}_{\text{err}} = n - 2\,\mathrm{tr}(\mathbf{S}_\lambda) + \mathrm{tr}(\mathbf{S}_\lambda \mathbf{S}_\lambda^T) \tag{17}$$

We call (17) the *degrees of freedom for error* of a smoother with smoother matrix $\mathbf{S}_\lambda$. Then

$$\hat{\sigma}^2 = \frac{\|\mathbf{y} - \hat{\mathbf{y}}_\lambda\|^2}{\mathrm{df}_{\text{err}}}$$

is our (biased) estimate of σ^2. At least we can say that $\hat{\sigma}^2$ generally overestimates σ^2 because its bias $\mathbf{b}_\lambda^T \mathbf{b}_\lambda / \mathrm{df}_{\text{err}}$ is necessarily nonnegative.

4.2.5 Degrees of Freedom

The notion of degrees of freedom doesn't really apply to smoothers, but

- people are so used to degrees of freedom, we would like to find an analog, even a strained one, and
- smoothing parameters in general and "bandwidth" in particular mean different things for different smoothers, so we would like some intrinsic notion of how much smoothing a smoother does.

We already have one thing we have called degrees of freedom (17). But (13) gives us another. In ordinary linear regression the analog of the smoother matrix is the hat matrix (11). It is an orthogonal projection, which means $\mathbf{H}^T = \mathbf{H}$ and $\mathbf{H}^2 = \mathbf{H}$. This the analog of (13) for ordinary linear regression is

$$\mathrm{var}(\hat{\mathbf{y}}) = \sigma^2 \mathbf{H}\mathbf{H}^T = \sigma^2 \mathbf{H}$$

and $\mathrm{tr}(\mathbf{H})$ is the number of regression coefficients in the model (this is tricky, we won't try to prove it).

By analogy, this suggests

$$\mathrm{df}_{\text{var}} = \mathrm{tr}(\mathbf{S}_\lambda \mathbf{S}_\lambda^T)$$

as another degrees of freedom notion. Yet a third degrees of freedom notion is

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

This is the simplest of all, because it doesn't involve a matrix multiplication. It arises as the correction in Mallows's C_p statistic (Section 4.3.2 below) and also as the correction in generalized cross validation (Section 4.3.6 below).

These three are not directly comparable, because $\mathrm{df}_{\mathrm{err}}$ is n minus something comparable to the others. The three directly comparable degrees of freedom notions are

$$\begin{gathered} \operatorname{tr}(\mathbf{S}_\lambda) \\ \operatorname{tr}(\mathbf{S}_\lambda \mathbf{S}_\lambda^T) \\ 2 \operatorname{tr}(\mathbf{S}_\lambda) - \operatorname{tr}(\mathbf{S}_\lambda \mathbf{S}_\lambda^T) \end{gathered}$$

All three reduce to $\operatorname{tr} \mathbf{H} = p$ in the case of ordinary linear regression. They are all different for a general smoother.

4.3 Performance Criteria

4.3.1 Mean Squared Error

The *mean squared error* of the estimator $\hat{g}_\lambda(x)$ is

$$\operatorname{mse}(x, \lambda) = E\bigl\{ \bigl(\hat{g}_\lambda(x) - g(x)\bigr)^2 \bigr\}$$

Generally, this is different at each x.

The *predictive squared error* of the estimator $\hat{g}_\lambda(x)$ is

$$\operatorname{pse}(x, \lambda) = E\bigl\{ \bigl(y^* - \hat{g}_\lambda(x)\bigr)^2 \bigr\}$$

where y^* is a new response value associated with predictor value x ("new" here meaning *not* one of the "old" values used in the estimation of $\hat{g}_\lambda$). There is a simple relation between these two quantities.

$$\begin{aligned} \operatorname{pse}(x, \lambda) &= E\bigl\{ \bigl(y^* - \hat{g}_\lambda(x)\bigr)^2 \bigr\} \\ &= E\bigl\{ \bigl(y^* - g(x)\bigr)^2 \bigr\} - 2E\bigl\{ \bigl(y^* - g(x)\bigr)\bigl(\hat{g}_\lambda(x) - g(x)\bigr) \bigr\} \\ &\qquad + E\bigl\{ \bigl(\hat{g}_\lambda(x) - g(x)\bigr)^2 \bigr\} \\ &= E\bigl\{ \bigl(y^* - g(x)\bigr)^2 \bigr\} + E\bigl\{ \bigl(\hat{g}_\lambda(x) - g(x)\bigr)^2 \bigr\} \\ &= \sigma^2 + \operatorname{mse}(x, \lambda) \end{aligned}$$

(the expectation of the cross product term is zero because y^* and the data used to estimate $\hat{g}_\lambda$ are independent).

And there is an analogous decomposition of mean square error. Students who have had theory will recall "mse equals variance plus bias squared." For those who haven't had theory, we derive this here. First we get a notation for bias in this context (bias at x)

$$b_\lambda(x) = E\bigl\{ \hat{g}_\lambda(x) \bigr\} - g(x)$$

Then

$$\begin{aligned}
\operatorname{mse}(x, \lambda) &= E\bigl\{\bigl(\hat{g}_\lambda(x) - g(x)\bigr)^2\bigr\} \\
&= E\bigl\{\bigl(\hat{g}_\lambda(x) - g(x) - b_\lambda(x)\bigr)^2\bigr\} \\
&\qquad + 2b_\lambda(x)E\bigl\{\bigl(\hat{g}_\lambda(x) - g(x) - b_\lambda(x)\bigr)\bigr\} + b_\lambda(x)^2 \\
&= E\bigl\{\bigl(\hat{g}_\lambda(x) - g(x) - b_\lambda(x)\bigr)^2\bigr\} + b_\lambda(x)^2 \\
&= \operatorname{var}\bigl\{\hat{g}_\lambda(x)\bigr\} + b_\lambda(x)^2
\end{aligned}$$

(here the expectation of the cross product term is zero because $g(x) + b_\lambda(x)$ is the expectation of $\hat{g}_\lambda(x)$).

In order to get a single criterion of performance, we average these quantities over the observed x values

$$\operatorname{mse}(\lambda) = \frac{1}{n} \sum_{i=1}^{n} \operatorname{mse}(\lambda, x_i) \tag{18}$$

and

$$\operatorname{pse}(\lambda) = \sigma^2 + \operatorname{mse}(\lambda) \tag{19}$$

We can now go back to our matrix notation

$$\begin{aligned}
\operatorname{mse}(\lambda) &= \frac{1}{n} \sum_{i=1}^{n} \operatorname{mse}(\lambda, x_i) \\
&= \frac{1}{n} \sum_{i=1}^{n} \operatorname{var}\bigl\{\hat{g}_\lambda(x_i)\bigr\} + \frac{1}{n} \sum_{i=1}^{n} b_\lambda(x_i)^2 \\
&= \frac{1}{n} \sum_{i=1}^{n} \operatorname{var}(\hat{y}_{\lambda,i}) + \frac{1}{n} \sum_{i=1}^{n} b_i^2
\end{aligned}$$

where the $\hat{y}_{\lambda,i}$ are the components of $\hat{\mathbf{y}}_\lambda$ as defined above and the $b_{\lambda,i}$ are the components of $\mathbf{b}_\lambda$ as defined above. Thus we see

$$\begin{aligned}
\operatorname{mse}(\lambda) &= \frac{\operatorname{tr}\bigl(\operatorname{var}(\hat{\mathbf{y}}_\lambda)\bigr)}{n} + \frac{\mathbf{b}_\lambda^T \mathbf{b}_\lambda}{n} \\
&= \frac{\sigma^2 \operatorname{tr}\bigl(\mathbf{S}_\lambda \mathbf{S}_\lambda^T\bigr)}{n} + \frac{\mathbf{b}_\lambda^T \mathbf{b}_\lambda}{n}
\end{aligned} \tag{20}$$

Hence

$$\operatorname{pse}(\lambda) = \sigma^2 \left(1 + \frac{\operatorname{tr}\bigl(\mathbf{S}_\lambda \mathbf{S}_\lambda^T\bigr)}{n}\right) + \frac{\mathbf{b}_\lambda^T \mathbf{b}_\lambda}{n} \tag{21}$$

4.3.2 Mallows's C_p

A very bad approximation of $\operatorname{pse}(\lambda)$ is the *average squared residual*

$$\operatorname{asr}(\lambda) = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 = \frac{\|\mathbf{y} - \hat{\mathbf{y}}_\lambda\|^2}{n} \tag{22}$$

This is the criterion that least squares minimizes. The more the model overfits, the smaller it is.

We have already calculated its expectation except for a factor of n in (16)

$$\begin{aligned} E\{\operatorname{asr}(\lambda)\} &= \frac{\sigma^2\bigl(n - 2\operatorname{tr}(\mathbf{S}_\lambda) + \operatorname{tr}(\mathbf{S}_\lambda \mathbf{S}_\lambda^T)\bigr) + \mathbf{b}^T\mathbf{b}}{n} \\ &= \sigma^2 \left(1 - \frac{2\operatorname{tr}(\mathbf{S}_\lambda)}{n} + \frac{\operatorname{tr}(\mathbf{S}_\lambda \mathbf{S}_\lambda^T)}{n}\right) + \frac{\mathbf{b}^T\mathbf{b}}{n} \end{aligned}$$

Comparing this with (21) we see that

$$E\{\operatorname{asr}(\lambda)\} = \operatorname{pse}(\lambda) - \sigma^2 \frac{2\operatorname{tr}(\mathbf{S}_\lambda)}{n}$$

hence

$$C_p(\lambda) = \operatorname{asr}(\lambda) + \hat{\sigma}^2 \frac{2\operatorname{tr}(\mathbf{S}_\lambda)}{n} \tag{23}$$

is a sensible estimate of predictive squared error. This is called *Mallows's C_p statistic.* It was originally introduced for model selection in ordinary regression. In that context p is the number of regression coefficients in the model. Here it has been generalized to a context where p makes no sense, but the name of this thingy is "C_p." Sorry about that. This is why "alphabet soup" terminology is bad.

4.3.3 Cross Validation

Cross validation is an important idea in regression. The idea is to estimate $\operatorname{pse}(\lambda)$ by

$$\operatorname{cv}(\lambda) = \frac{1}{n} \sum_{i=1}^{n} \bigl(y_i - \hat{g}_{\lambda,-i}(x_i)\bigr)^2 \tag{24}$$

where $\hat{g}_{\lambda,-i}$ means the smooth estimate for smoothing parameter λ and data $(x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n)$ that "leaves out" x_i.

The point is that in $\operatorname{pse}(\lambda)$ we need y^* that are independent of the "old" data. We don't have any such y^* so we make do with y_i. But to get "old" data independent of y_i we have leave y_i out of the so-called "old" data. We do that in turn for each i, using a different "old" data with each y_i. Tricky. And rather complicated. But the best we can do without actually obtaining some actually new data.

Another formula for $\text{cv}(\lambda)$ is given in Section 4.3.5 below. To derive it we need some theory from the next section.

4.3.4 Leave One Out

It seems at first sight that (24) would necessitate doing our smoothing procedure n times, one for each y_i left out. It turns out that this is not so. We can calculate all the leave-one-out smooths from the original smoother matrix $\mathbf{S}_\lambda$.

Temporarily, we leave the λ's off to simplify the notation. We'll put them back later. Let $\hat{y}_{-i}$ denote the predicted value for the i-th case when y_i is left out of the data doing the fitting.

Actually, it is not completely clear what "leave one out" means in the context of smoothing. In general, there is no necessary relationship between a smoother for n data pairs and a smoother for $n-1$ data pairs. (Of course, a for a *particular kind of smoother*, such as a kernel smoother with a particular kernel and bandwidth or a smoothing spline with a particular smoothing parameter, there is such a relationship. But there is no relationship in general.)

One method of finding such a general relationship is to note that any reasonable smoother is constant preserving, which can be expressed in the formula $\mathbf{S1} = \mathbf{1}$, where $\mathbf{1}$ is the vector with all elements equal to 1. In words, this says the rows of $\mathbf{S}$ sum to one. Thus if we want to use the same smoother with the i-th row and column deleted to be an $(n-1) \times (n-1)$ smoother matrix, we must renormalize the rows to sum to one. Let s_{ij} denote the elements of the original $n \times n$ smoother matrix $\mathbf{S}$. When we delete the i-th column, then the i-th row now sums to $1 - s_{ii}$. So that's what we divide by to renormalize.

This line of reasoning gives

$$\hat{y}_{-i} = \frac{1}{1 - s_{ii}} \sum_{\substack{j=1 \\ j \neq i}}^{n} s_{ij} y_j \tag{25}$$

for comparison, the ordinary predicted value is

$$\hat{y}_i = \sum_{j=1}^{n} s_{ij} y_j \tag{26}$$

If we multiply (25) by $1 - s_{ii}$ and then move a term from one side to the other, we get

$$\begin{aligned}
\hat{y}_{-i} &= \sum_{\substack{j=1 \\ j \neq i}}^{n} s_{ij} y_j + s_{ii} \hat{y}_{-i} \\
&= \sum_{j=1}^{n} s_{ij} y_j + s_{ii} \hat{y}_{-i} - s_{ii} y_i \\
&= \hat{y}_i + s_{ii} \hat{y}_{-i} - s_{ii} y_i
\end{aligned}$$

From which we conclude

$$y_i - \hat{y}_{-i} = y_i - \hat{y}_i + s_{ii}(y_i - \hat{y}_{-i})$$

and hence

$$y_i - \hat{y}_{-i} = \frac{y_i - \hat{y}_i}{1 - s_{ii}} \tag{27}$$

This equation is very important. The left hand side of (27) is the *leave one out residual.* The right hand side of (27) expresses this in terms of the ordinary residual $y_i - \hat{y}_i$.

Thus we do not need to do n smooths to do cross-validation. As long as we can get the diagonal elements s_{ii} of the smoother matrix $\mathbf{S}$, we can compute the leave one out residuals from the ordinary residuals.

If we put the λ's back, (27) becomes

$$y_i - \hat{y}_{\lambda,-i} = \frac{y_i - \hat{y}_{\lambda,i}}{1 - s_{\lambda,ii}} \tag{28}$$

4.3.5 Cross Validation Revisited

Then

$$\operatorname{cv}(\lambda) = \frac{1}{n} \sum_{i=1}^{n} \left(\frac{y_i - \hat{y}_{\lambda,i}}{1 - s_{\lambda,ii}} \right)^2 \tag{29}$$

4.3.6 Generalized Cross Validation

A minor variant on cross validation is, so-called *generalized cross validation*, which, of course, like most things statisticians call “generalized,” isn't.

It replaces the $s_{\lambda,ii}$ in the denominator with their average $\operatorname{tr}(\mathbf{S}_\lambda)/n$ giving

$$
\begin{aligned}
\operatorname{gcv}(\lambda) &= \frac{1}{n} \sum_{i=1}^{n} \left(\frac{y_i - \hat{y}_{\lambda,i}}{1 - \operatorname{tr}(\mathbf{S}_\lambda)/n} \right)^2 \\
&= \frac{1}{n(1 - \operatorname{tr}(\mathbf{S}_\lambda)/n)^2} \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_{\lambda,i})^2 \\
&= \frac{\|\mathbf{y} - \hat{\mathbf{y}}_\lambda\|^2}{n(1 - \operatorname{tr}(\mathbf{S}_\lambda)/n)^2} \\
&= \frac{\operatorname{asr}(\lambda)}{(1 - \operatorname{tr}(\mathbf{S}_\lambda)/n)^2}
\end{aligned}
\tag{30}
$$

Thus $\operatorname{gcv}(\lambda)$ is a simple function of the average squared residual $\operatorname{asr}(\lambda)$ given by (22).

5 The Bias-Variance Trade-off

The title of this section is about the inevitability of trade-offs in life. You can't have everything.

The "bias" under discussion is the bias term in $\operatorname{mse}(\lambda)$, the second term on the right hand side of (20), which is

$$
\mathbf{b}_\lambda^T \mathbf{b}_\lambda / n \tag{31}
$$

The "variance" under discussion is the variance term in $\operatorname{mse}(\lambda)$, the first term on the right hand side of (20), which is

$$
\sigma^2 \operatorname{tr}(\mathbf{S}_\lambda^T \mathbf{S}_\lambda)/n \tag{32}
$$

We want both to be as small as possible. If the "bias" (31) is big, then our estimate $\hat{\mathbf{y}}_\lambda$ of the true regression function $\boldsymbol{\mu}$ is off center (its sampling distribution is not centered at $\boldsymbol{\mu}$). If the "variance" (32) is big, then our estimate $\hat{\mathbf{y}}_\lambda$ is too variable (its sampling distribution has too much spread).

Unfortunately, we can't have both small at the same time. Reducing λ (less smoothing) reduces bias but increases variance, and vice versa.

To see how the bias-variance trade-off works, let us work through a simple kernel smoothing example with Gaussian kernel. In order to know the bias we must know the true regression function $\boldsymbol{\mu}$. So this example must be on made-up data (where we know the truth).

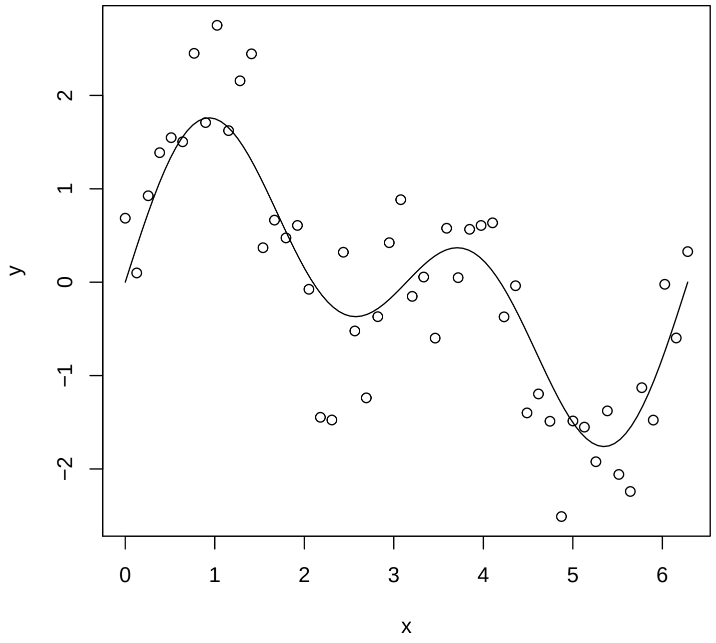

Figure 9: Made-up data with true regression curve.

```
> n <- 50
> x <- seq(0, 2 * pi, length = n)
> mu <- sin(x) + sin(2 * x)
> sigma <- 0.5
> set.seed(42)
> y <- mu + sigma * rnorm(n)
```

Figure 9 shows these made-up data. It is made by the R commands

```
> plot(x, y)
> curve(sin(x) + sin(2 * x), add = TRUE)
```

Because the `x` values are equally spaced, the following code calculates the smoother matrix for bandwidth `h`

```
> h <- 0.5
> xdiff <- mean(diff(x))
> j <- seq(-n, n)
> k <- seq(1, n)
> wj <- exp(- (xdiff * j / h)^2 / 2)
> S <- NULL
> for (i in 1:n) {
+    l <- match(k - i, j)
+    foo <- wj[l]
+    S <- rbind(S, foo / sum(foo))
+ }
```

Figure 10 is the same as Figure 9 except the smooth estimate is added R commands

```
> y.hat <- S %*% y
> plot(x, y)
> curve(sin(x) + sin(2 * x), add = TRUE)
> lines(x, y.hat, lty = 2)
```

Now that we know how to construct the smoother matrix for some smoother, we can calculate all kinds of things.

```
> stuff <- NULL
> hs <- c(seq(0.06, 0.09, 0.01), seq(0.1, 0.9, 0.1))
> for (h in hs) {
+     wj <- exp(- (xdiff * j / h)^2 / 2)
+     S <- NULL
+     for (i in 1:n) {
+        l <- match(k - i, j)
+        foo <- wj[l]
+        S <- rbind(S, foo / sum(foo))
+     }
+     y.hat <- S %*% y
+     b <- S %*% mu - mu
+     bsq <- sum(b^2)
+     df1 <- sum(diag(S))
+     df2 <- sum(diag(S %*% t(S)))
+     df3 <- 2 * df1 - df2
+     v <- sigma^2 * df2
+     mse <- (v + bsq) / n
+     pse <- sigma^2 + mse
+     rss <- sum((y - y.hat)^2)
```

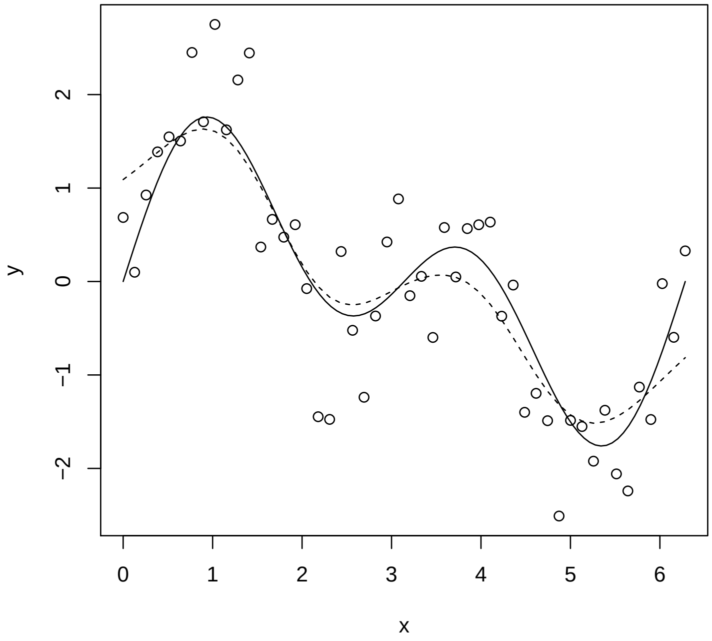

Figure 10: Made-up data with true regression curve (solid curve) and kernel smooth estimate thereof (bandwidth = 0.5).

```
+     sii <- diag(S)
+     cv <- sum(((y - y.hat) / (1 - sii))^2) / n
+     asr <- rss / n
+     gcv <- asr / (1 - df1 / n)^2
+     sigmahatsq <- rss / (n - df3)
+     Cp <- asr + sigmahatsq * 2 * df1 / n
+     stuff <- rbind(stuff, c(h, df1, df2, df3, bsq, v, pse, cv, gcv, Cp, asr))
+ }
> dimnames(stuff) <- list(rep("", nrow(stuff)),
+     c("h", "df1", "df2", "df3", "bias", "var", "pse", "cv", "gcv", "Cp", "asr"))
> stuff[ , 2:4] <- round(stuff[ , 2:4], 1)
> round(stuff, 2)
```

```
    h  df1  df2  df3  bias  var  pse   cv  gcv   Cp  asr
 0.06 41.7 35.5 47.9  0.00 8.87 0.43 0.46 0.46 0.52 0.01
 0.07 36.6 28.6 44.5  0.01 7.15 0.39 0.46 0.46 0.47 0.03
 0.08 32.2 23.9 40.5  0.02 5.98 0.37 0.45 0.45 0.45 0.06
 0.09 28.7 20.8 36.7  0.03 5.20 0.35 0.45 0.44 0.43 0.08
 0.10 25.9 18.6 33.3  0.05 4.64 0.34 0.44 0.44 0.42 0.10
 0.20 13.2  9.5 17.0  0.44 2.37 0.31 0.39 0.39 0.38 0.21
 0.30  9.0  6.5 11.5  1.45 1.61 0.31 0.41 0.41 0.40 0.27
 0.40  6.9  5.0  8.8  3.10 1.24 0.34 0.46 0.46 0.45 0.34
 0.50  5.6  4.1  7.1  5.19 1.01 0.37 0.53 0.53 0.53 0.42
 0.60  4.7  3.4  6.0  7.47 0.86 0.42 0.61 0.61 0.60 0.50
 0.70  4.1  3.0  5.2  9.68 0.75 0.46 0.69 0.68 0.68 0.57
 0.80  3.7  2.7  4.6 11.65 0.67 0.50 0.75 0.75 0.75 0.64
 0.90  3.3  2.4  4.2 13.31 0.61 0.53 0.80 0.80 0.80 0.70
```

Not very pretty, but it does show that cv(λ), gcv(λ) and $C_p(\lambda)$ fairly closely approximate pse(λ). And it also shows that they all choose the same bandwidth ($h = 0.20$) as the optimal choice (among the ones tried), and this is optimal according to the pse(λ) criterion.

Of course, that's partly due to the crudeness of our grid of h values. Let's try again with a finer grid. (We won't repeat the code, which is unchanged except for different h values and different rounding).

```
    h  df1  df2  df3 bias  var    pse     cv    gcv     Cp    asr
 0.18 14.6 10.5 18.8 0.32 2.62 0.3087 0.3940 0.3949 0.3830 0.1975
 0.19 13.9  9.9 17.9 0.38 2.49 0.3072 0.3927 0.3935 0.3825 0.2051
 0.20 13.2  9.5 17.0 0.44 2.37 0.3061 0.3920 0.3925 0.3825 0.2123
```

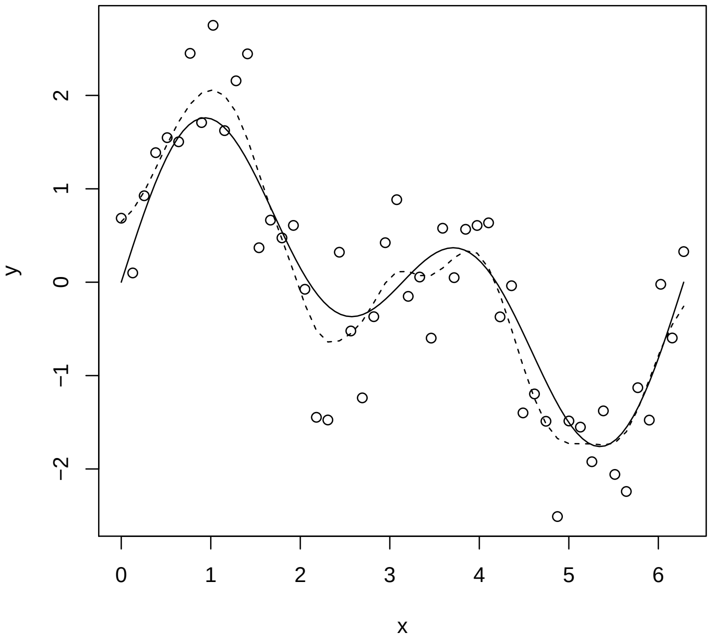

Figure 11: Made-up data with true regression curve (solid curve) and kernel smooth estimate thereof using the optimal bandwidth 0.23.

```
 0.21 12.6  9.0 16.2 0.51 2.26 0.3054 0.3917 0.3921 0.3828 0.2191
 0.22 12.1  8.6 15.5 0.59 2.16 0.3050 0.3919 0.3921 0.3835 0.2256
 0.23 11.6  8.3 14.8 0.67 2.07 0.3049 0.3926 0.3925 0.3845 0.2319
 0.24 11.1  8.0 14.2 0.76 1.99 0.3051 0.3937 0.3934 0.3859 0.2380
 0.25 10.7  7.7 13.7 0.86 1.92 0.3055 0.3953 0.3947 0.3877 0.2441
 0.26 10.3  7.4 13.2 0.97 1.85 0.3062 0.3973 0.3964 0.3898 0.2500
```

Now we see that pse(λ) picks $h = 0.23$, whereas cv(λ) picks $h = 0.21$, gcv(λ) picks either $h = 0.21$ or $h = 0.22$, and $C_p(\lambda)$ picks either $h = 0.19$ or $h = 0.20$.

Figure 11 (page 32) is the same as Figure 9 except the "optimal" smooth estimate is added.

The general story about bias-variance trade-off is much the same as this example. For h too small (not enough smoothing) the bias is small but the variance is huge. For h too big (too much smoothing) the variance is small but the bias is huge. Somewhere in the middle (h just right) the bias and variance are both moderate. That's the amount of smoothing you want.

References

Hastie, T. J. and Tibshirani, R. J. (1990). *Generalized Additive Models.* Chapman & Hall.