10

Moving Beyond Conditional Expectations: Weighted Least Squares, Heteroskedasticity, Local Polynomial Regression

So far, all our estimates have been based on the mean squared error, giving equal importance to all observations, as is generally appropriate when looking at conditional expectations. In this chapter, we'll start to work with giving more or less weight to different observations, through *weighted* least squares. The oldest reason to want to use weighted least squares is to deal with non-constant variance, or heteroskedasticity, by giving more weight to lower-variance observations. This leads us naturally to estimating the conditional variance function, just as we've been estimating conditional expectations. On the other hand, weighted least squares lets us general kernel regression to locally polynomial regression.

10.1 Weighted Least Squares

When we use ordinary least squares to estimate linear regression, we (naturally) minimize the mean squared error:

$$MSE(\beta) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \vec{x}_i \cdot \beta)^2 \tag{10.1}$$

The solution is of course

$$\widehat{\beta}_{OLS} = (\mathbf{x}^T\mathbf{x})^{-1}\mathbf{x}^T\mathbf{y} \tag{10.2}$$

We could instead minimize the *weighted* mean squared error,

$$WMSE(\beta, \vec{w}) = \frac{1}{n}\sum_{i=1}^{n} w_i(y_i - \vec{x}_i \cdot \beta)^2 \tag{10.3}$$

This includes ordinary least squares as the special case where all the weights $w_i = 1$. We can solve it by the same kind of linear algebra we used to solve the ordinary linear least squares problem. If we write $\mathbf{w}$ for the matrix with the w_i on the diagonal and zeroes everywhere else, the solution is

$$\widehat{\beta}_{WLS} = (\mathbf{x}^T\mathbf{w}\mathbf{x})^{-1}\mathbf{x}^T\mathbf{w}\mathbf{y} \tag{10.4}$$

But why would we want to minimize Eq. 10.3?

1. *Focusing accuracy.* We may care very strongly about predicting the response for certain values of the input — ones we expect to see often again, ones where mistakes are especially costly or embarrassing or painful, etc. — than others.

If we give the points $\vec{x}_i$ near that region big weights w_i, and points elsewhere smaller weights, the regression will be pulled towards matching the data in that region.

2. *Discounting imprecision.* Ordinary least squares is the maximum likelihood estimate when the ϵ in $Y = \vec{X} \cdot \beta + \epsilon$ is IID Gaussian white noise. This means that the variance of ϵ has to be constant, and we measure the regression curve with the same precision elsewhere. This situation, of constant noise variance, is called **homoskedasticity**. Often however the magnitude of the noise is not constant, and the data are **heteroskedastic**.

   When we have heteroskedasticity, even if each noise term is still Gaussian, ordinary least squares is no longer the maximum likelihood estimate, and so no longer efficient. If however we know the noise variance σ_i^2 at each measurement i, and set $w_i = 1/\sigma_i^2$, we get the heteroskedastic MLE, and recover efficiency. (See below.)

   To say the same thing slightly differently, there's just no way that we can estimate the regression function as accurately where the noise is large as we can where the noise is small. Trying to give equal attention to all parts of the input space is a waste of time; we should be more concerned about fitting well where the noise is small, and expect to fit poorly where the noise is big.

3. *Sampling bias.* In many situations, our data comes from a survey, and some members of the population may be more likely to be included in the sample than others. When this happens, the sample is a biased representation of the population. If we want to draw inferences about the population, it can help to give more weight to the kinds of data points which we've under-sampled, and less to those which were over-sampled. In fact, typically the weight put on data point i would be inversely proportional to the probability of i being included in the sample (exercise 10.1). Strictly speaking, if we are willing to believe that linear model is exactly correct, that there are no omitted variables, and that the inclusion probabilities p_i do not vary with y_i, then this sort of survey weighting is redundant (DuMouchel and Duncan, 1983). When those assumptions are not met — when there're non-linearities, omitted variables, or "selection on the dependent variable" — survey weighting is advisable, if we know the inclusion probabilities fairly well.

   The same trick works under the same conditions when we deal with "covariate shift", a change in the distribution of X. If the old probability density function was $p(x)$ and the new one is $q(x)$, the weight we'd want to use is $w_i = q(x_i)/p(x_i)$ (Quiñonero-Candela *et al.*, 2009). This can involve estimating both densities, or their ratio (chapter 14).

4. *Doing something else.* There are a number of other optimization problems which can be transformed into, or approximated by, weighted least squares. The most important of these arises from generalized linear models, where the mean response is some nonlinear function of a linear predictor; we will look at them in Chapters 11 and 12.

In the first case, we decide on the weights to reflect our priorities. In the

third case, the weights come from the optimization problem we'd really rather be solving. What about the second case, of heteroskedasticity?

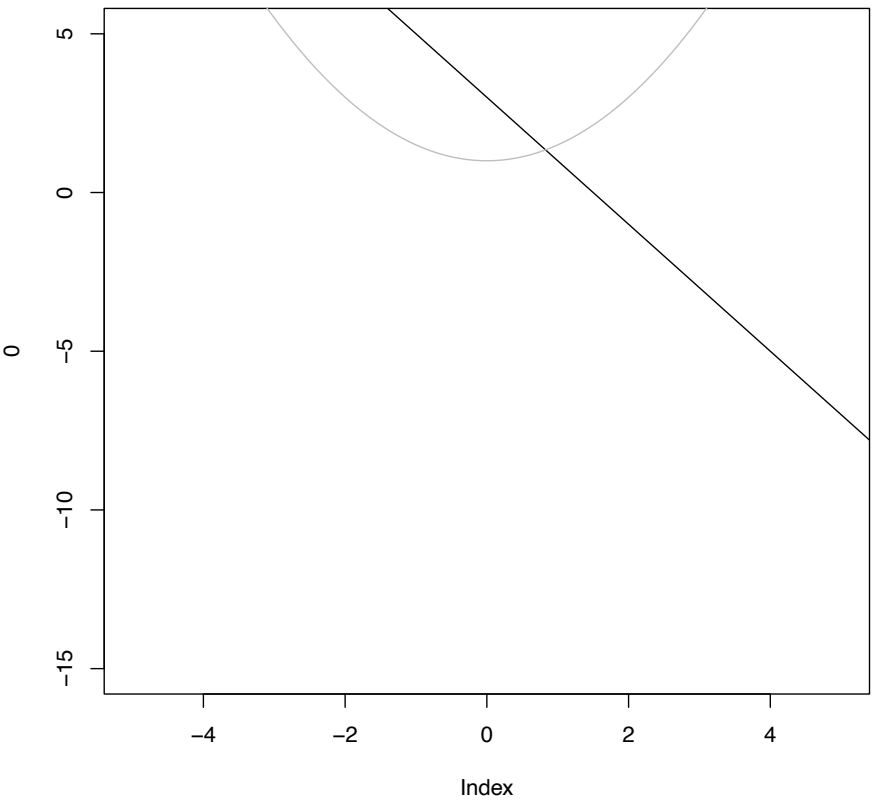
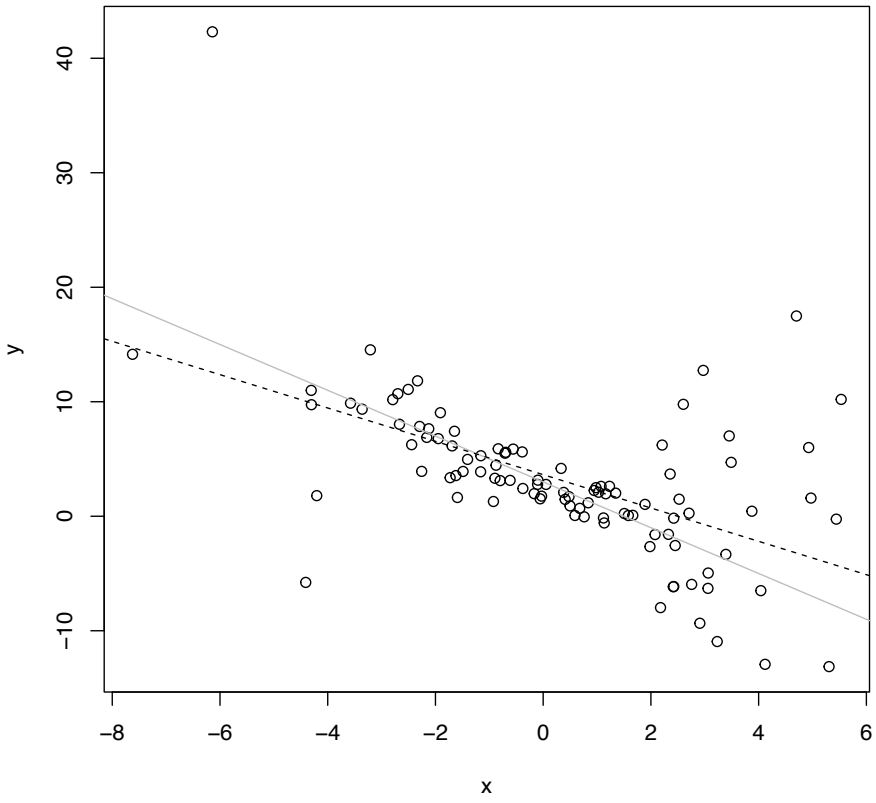


Figure 10.1 Black line: Linear response function ($y = 3 - 2x$). Grey curve: standard deviation as a function of x ($\sigma(x) = 1 + x^2/2$). (Code deliberately omitted; can you reproduce this figure?)

10.2 Heteroskedasticity

Suppose the noise variance is itself variable. For example, the figure shows a simple linear relationship between the input X and the response Y, but also a nonlinear relationship between X and $\mathbb{V}[Y]$.

In this particular case, the ordinary least squares estimate of the regression line is $3.63 - -1.46x$, with R reporting standard errors in the coefficients of ± 0.59 and 0.22, respectively. Those are however calculated under the assumption that the noise is homoskedastic, which it isn't. And in fact we can see, pretty much,

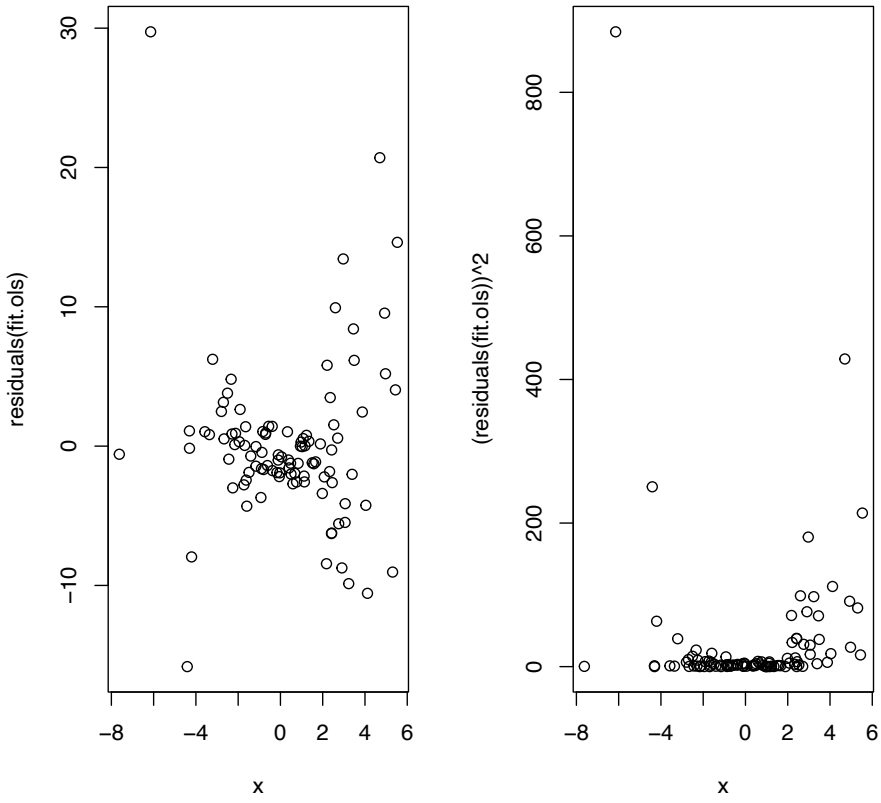



```
plot(x, y)
abline(a = 3, b = -2, col = "grey")
fit.ols = lm(y ~ x)
abline(fit.ols, lty = "dashed")
```

Figure 10.2 Scatter-plot of $n = 100$ data points from the above model. (Here X is Gaussian with mean 0 and variance 9.) Grey: True regression line. Dashed: ordinary least squares regression line.

that there is heteroskedasticity — if looking at the scatter-plot didn't convince us, we could always plot the residuals against x, which we should do anyway.

To see whether that makes a difference, let's re-do this many times with different draws from the same model (Example 24).




```
par(mfrow = c(1, 2))
plot(x, residuals(fit.ols))
plot(x, (residuals(fit.ols))^2)
par(mfrow = c(1, 1))
```

Figure 10.3 Residuals (left) and squared residuals (right) of the ordinary least squares regression as a function of x. Note the much greater range of the residuals at large absolute values of x than towards the center; this changing dispersion is a sign of heteroskedasticity.

Running `ols.heterosked.error.stats(1e4)` produces 10^4 random simulated data sets, which all have the same x values as the first one, but different values of y, generated however from the same model. It then uses those samples to get

```
ols.heterosked.example = function(n) {
    y = 3 - 2 * x + rnorm(n, 0, sapply(x, function(x) {
        1 + 0.5 * x^2
    }))
    fit.ols = lm(y ~ x)
    return(fit.ols$coefficients - c(3, -2))
}

ols.heterosked.error.stats = function(n, m = 10000) {
    ols.errors.raw = t(replicate(m, ols.heterosked.example(n)))
    intercept.se = sd(ols.errors.raw[, "(Intercept)"])
    slope.se = sd(ols.errors.raw[, "x"])
    return(c(intercept.se = intercept.se, slope.se = slope.se))
}
```

Code Example 24: *Functions to generate heteroskedastic data and fit OLS regression to it, and to collect error statistics on the results.*

the standard error of the ordinary least squares estimates. (Bias remains a non-issue.) What we find is the standard error of the intercept is only a little inflated (simulation value of 0.7 versus official value of 0.59), but the standard error of the slope is much larger than what R reports, 0.52 versus 0.22. Since the intercept is fixed by the need to make the regression line go through the center of the data (Chapter 2), the real issue here is that our estimate of the slope is much less precise than ordinary least squares makes it out to be. Our estimate is still consistent, but not as good as it was when things were homoskedastic. Can we get back some of that efficiency?

10.2.1 Weighted Least Squares as a Solution to Heteroskedasticity

Suppose we visit the Oracle of Regression (Figure 10.4), who tells us that the noise has a standard deviation that goes as $1 + x^2/2$. We can then use this to improve our regression, by solving the weighted least squares problem rather than ordinary least squares (Figure 10.5).

This not only looks better, it is better: the estimated line is now $2.95 - 1.53x$, with reported standard errors of 0.32 and 0.2. This checks check out with simulation (Example 25): the standard errors from the simulation are 0.23 for the intercept and 0.23 for the slope, so R's internal calculations are working very well.

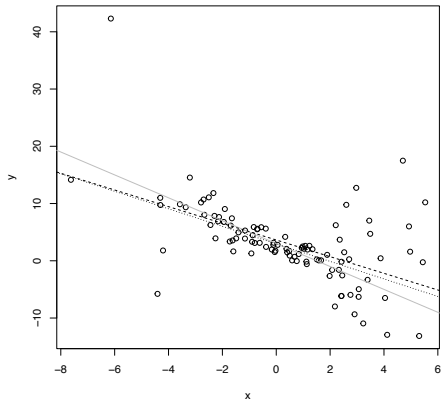
Why does putting these weights into WLS improve things?

Figure 10.4 Statistician (right) consulting the Oracle of Regression (left) about the proper weights to use to overcome heteroskedasticity. (Image from http://en.wikipedia.org/wiki/Image:Pythia1.jpg.)

```
wls.heterosked.example = function(n) {
    y = 3 - 2 * x + rnorm(n, 0, sapply(x, function(x) {
        1 + 0.5 * x^2
    }))
    fit.wls = lm(y ~ x, weights = 1/(1 + 0.5 * x^2))
    return(fit.wls$coefficients - c(3, -2))
}

wls.heterosked.error.stats = function(n, m = 10000) {
    wls.errors.raw = t(replicate(m, wls.heterosked.example(n)))
    intercept.se = sd(wls.errors.raw[, "(Intercept)"])
    slope.se = sd(wls.errors.raw[, "x"])
    return(c(intercept.se = intercept.se, slope.se = slope.se))
}
```

Code Example 25: *Linear regression of heteroskedastic data, using weighted least-squared regression.*




```
plot(x, y)
abline(a = 3, b = -2, col = "grey")
fit.ols = lm(y ~ x)
abline(fit.ols, lty = "dashed")
fit.wls = lm(y ~ x, weights = 1/(1 + 0.5 * x^2))
abline(fit.wls, lty = "dotted")
```

Figure 10.5 Figure 10.2 plus the weighted least squares regression line (dotted).

10.2.2 Some Explanations for Weighted Least Squares

Qualitatively, the reason WLS with inverse variance weights works is the following. OLS tries equally hard to match observations at each data point.[1] Weighted least squares, naturally enough, tries harder to match observations where the weights are big, and less hard to match them where the weights are small. But each y_i contains not only the true regression function $\mu(x_i)$ but also some noise ϵ_i. The noise terms have large magnitudes where the variance is large. So we should want to have small weights where the noise variance is large, because there the data tends to be far from the true regression. Conversely, we should put big weights where the noise variance is small, and the data points are close to the true regression.

The qualitative reasoning in the last paragraph doesn't explain why the weights should be inversely proportional to the variances, $w_i \propto 1/\sigma^2_{x_i}$ — why not $w_i \propto 1/\sigma_{x_i}$, for instance? Seeing why those are the right weights requires investigating how well different, indeed *arbitrary*, choices of weights would work.

Look at the equation for the WLS estimates again:

$$\widehat{\beta}_{WLS} = (\mathbf{x}^T\mathbf{w}\mathbf{x})^{-1}\mathbf{x}^T\mathbf{w}\mathbf{y} \tag{10.5}$$

$$= \mathbf{h}(\mathbf{w})\mathbf{y} \tag{10.6}$$

defining the matrix $\mathbf{h}(\mathbf{w}) = (\mathbf{x}^T\mathbf{w}\mathbf{x})^{-1}\mathbf{x}^T\mathbf{w}$ for brevity. (The notation reminds us that everything depends on the weights in $\mathbf{w}$.) Imagine holding $\mathbf{x}$ constant, but repeating the experiment multiple times, so that we get noisy values of $\mathbf{y}$. In each experiment, $Y_i = \vec{x}_i \cdot \beta + \epsilon_i$, where $\mathbb{E}[\epsilon_i] = 0$ and $\mathbb{V}[\epsilon_i] = \sigma^2_{x_i}$. So

$$\widehat{\beta}_{WLS} = \mathbf{h}(\mathbf{w})\mathbf{x}\beta + \mathbf{h}(\mathbf{w})\epsilon \tag{10.7}$$

$$= \beta + \mathbf{h}(\mathbf{w})\epsilon \tag{10.8}$$

Since $\mathbb{E}[\epsilon] = 0$, the WLS estimator is unbiased:

$$\mathbb{E}\left[\widehat{\beta}_{WLS}\right] = \beta \tag{10.9}$$

In fact, for the j^{th} coefficient,

$$\widehat{\beta}_j = \beta_j + [\mathbf{h}(\mathbf{w})\epsilon]_j \tag{10.10}$$

$$= \beta_j + \sum_{i=1}^{n} h_{ji}(\mathbf{w})\epsilon_i \tag{10.11}$$

Since the WLS estimate is unbiased, it's natural to want it to also have a small variance, and

$$\mathbb{V}\left[\widehat{\beta}_j\right] = \sum_{i=1}^{n} h_{ji}(\mathbf{w})\sigma^2_{x_i} \tag{10.12}$$

It can be shown — the result is called the **Gauss-Markov theorem** — that

[1] Less anthropomorphically, the objective function in Eq. 10.1 has the same derivative with respect to the squared error at each point, $\frac{\partial MSE}{\partial (y_i - \vec{x}_i \cdot \beta)^2} = \frac{1}{n}$.

picking weights to minimize the variance in the WLS estimate has the unique solution $w_i = 1/\sigma^2_{x_i}$. It does not require us to assume the noise is Gaussian[2], but the proof is a bit tricky, so I will confine it to §10.2.2.1 below.

A less general but easier-to-grasp result comes from adding the assumption that the noise around the regression line is Gaussian — that

$$Y = \vec{x} \cdot \beta + \epsilon, \ \epsilon \sim \mathcal{N}(0, \sigma^2_x) \tag{10.13}$$

The log-likelihood is then (Exercise 10.2)

$$-\frac{n}{2}\ln 2\pi - \frac{1}{2}\sum_{i=1}^{n}\log \sigma^2_{x_i} - \frac{1}{2}\sum_{i=1}^{n}\frac{(y_i - \vec{x}_i \cdot \beta)^2}{\sigma^2_{x_i}} \tag{10.14}$$

If we maximize this with respect to β, everything except the final sum is irrelevant, and so we minimize

$$\sum_{i=1}^{n}\frac{(y_i - \vec{x}_i \cdot \beta)^2}{\sigma^2_{x_i}} \tag{10.15}$$

which is just weighted least squares with $w_i = 1/\sigma^2_{x_i}$. So, if the probabilistic assumption holds, WLS is the efficient maximum likelihood estimator.

[2] Despite the first part of the name! Gauss himself was much less committed to assuming Gaussian distributions than many later statisticians.

10.2.2.1 Proof of the Gauss-Markov Theorem[3]

We want to prove that, when we are doing weighted least squares for linear regression, the best choice of weights $w_i = 1/\sigma^2_{x_i}$. We saw that that WLS is unbiased (Eq. 10.9), so "best" here means minimizing the variance. We have also already seen (Eq. 10.6) that

$$\widehat{\beta}_{WLS} = \mathbf{h}(\mathbf{w})\mathbf{y} \tag{10.16}$$

where the matrix $\mathbf{h}(\mathbf{w})$ is

$$\mathbf{h}(\mathbf{w}) = (\mathbf{x}^T\mathbf{w}\mathbf{x})^{-1}\mathbf{x}^T\mathbf{w} \tag{10.17}$$

It would be natural to try to write out the variance as a function of the weights w, set the derivative equal to zero, and solve. This is tricky, partly because we need to make sure that all the weights are positive and add up to one, but mostly because of the matrix inversion in the definition of $\mathbf{h}(\mathbf{w})$. A slightly less direct approach is actually much cleaner.

Write $\mathbf{w_0}$ for the inverse-variance weight matrix, and $\mathbf{h_0}$ for the hat matrix we get with those weights. Then for any other choice of weights, we have $\mathbf{h}(\mathbf{w}) = \mathbf{h_0} + \mathbf{c}$. ($\mathbf{c}$ is implicitly a function of the weights, but let's suppress that in the notation for brevity.) Since we know all WLS estimates are unbiased, we must have

$$(\mathbf{h_0} + \mathbf{c})\mathbf{x}\beta = \beta \tag{10.18}$$

but using the inverse-variance weights is a particular WLS estimate so

$$\mathbf{h_0}\mathbf{x}\beta = \beta \tag{10.19}$$

and so we can deduce that

$$\mathbf{c}\mathbf{x} = 0 \tag{10.20}$$

from unbiasedness.

Now consider the covariance matrix of the estimates, $\mathbb{V}\left[\tilde{\beta}\right]$. This will be $\mathbb{V}\left[(\mathbf{h_0} + \mathbf{c})\mathbf{Y}\right]$,

[3] You can skip this section, without loss of continuity.

which we can expand:

$$\begin{aligned}
\mathbb{V}\left[\tilde{\beta}\right] &= \mathbb{V}\left[(\mathbf{h_0}+\mathbf{c})\mathbf{Y}\right] && (10.21)\\
&= (\mathbf{h_0}+\mathbf{c})\mathbb{V}\left[Y\right](\mathbf{h_0}+\mathbf{c})^T && (10.22)\\
&= (\mathbf{h_0}+\mathbf{c})\mathbf{w_0}^{-1}(\mathbf{h_0}+\mathbf{c})^T && (10.23)\\
&= \mathbf{h_0}\mathbf{w_0}^{-1}\mathbf{h_0}^T + \mathbf{c}\mathbf{w_0}^{-1}\mathbf{h_0}^T + \mathbf{h_0}\mathbf{w_0}^{-1}\mathbf{c}^T + \mathbf{c}\mathbf{w_0}^{-1}\mathbf{c}^T && (10.24)\\
&= (\mathbf{x}^T\mathbf{w_0}\mathbf{x})^{-1}\mathbf{x}^T\mathbf{w_0}\mathbf{w_0}^{-1}\mathbf{w_0}\mathbf{x}(\mathbf{x}^T\mathbf{w_0}\mathbf{x})^{-1} && (10.25)\\
&\quad +\mathbf{c}\mathbf{w_0}^{-1}\mathbf{w_0}\mathbf{x}(\mathbf{x}^T\mathbf{w_0}\mathbf{x})^{-1}\\
&\quad +(\mathbf{x}^T\mathbf{w_0}\mathbf{x})^{-1}\mathbf{x}^T\mathbf{w_0}\mathbf{w_0}^{-1}\mathbf{c}^T\\
&\quad +\mathbf{c}\mathbf{w_0}^{-1}\mathbf{c}^T\\
&= (\mathbf{x}^T\mathbf{w_0}\mathbf{x})^{-1}\mathbf{x}^T\mathbf{w_0}\mathbf{x}(\mathbf{x}^T\mathbf{w_0}\mathbf{x})^{-1} && (10.26)\\
&\quad +\mathbf{c}\mathbf{x}(\mathbf{x}^T\mathbf{w_0}\mathbf{x})^{-1} + (\mathbf{x}^T\mathbf{w_0}\mathbf{x})^{-1}\mathbf{x}^T\mathbf{c}^T\\
&\quad +\mathbf{c}\mathbf{w_0}^{-1}\mathbf{c}^T\\
&= (\mathbf{x}^T\mathbf{w_0}\mathbf{x})^{-1} + \mathbf{c}\mathbf{w_0}^{-1}\mathbf{c}^T && (10.27)
\end{aligned}$$

where in the last step we use the fact that $\mathbf{cx} = 0$ (and so $\mathbf{x}^T\mathbf{c}^T = 0^T = 0$). Since $\mathbf{c}\mathbf{w_0}^{-1}\mathbf{c}^T \geq 0$, because $\mathbf{w_0}$ is a positive-definite matrix, we see that the variance is minimized by setting $\mathbf{c} = 0$, and using the inverse variance weights.

Notes:

1. If all the variances are equal, then we've proved the optimality of OLS.
2. The proof actually works when comparing the inverse-variance weights to any other linear, unbiased estimator; WLS with different weights is just a special case.
3. We can write the WLS problem as that of minimizing $(\mathbf{y}-\mathbf{x}\beta)^T\mathbf{w}(\mathbf{y}-\mathbf{x}\beta)$. If we allow $\mathbf{w}$ to be a non-diagonal, but still positive-definite, matrix, then we have the **generalized least squares** problem. This is appropriate when there are correlations between the noise terms at different observations, i.e., when $\mathrm{Cov}\left[\epsilon_i, \epsilon_j\right] \neq 0$ even though $i \neq j$. In this case, the proof is easily adapted to show that the optimal weight matrix $\mathbf{w}$ is the inverse of the noise covariance matrix. (This is why I wrote everything as a function of $\mathbf{w}$.)

Figure 10.6 The Oracle may be out (left), or too creepy to go visit (right). What then? (Left, the sacred oak of the Oracle of Dodona, copyright 2006 by Flickr user "essayen", http://flickr.com/photos/essayen/245236125/ right, the entrace to the cave of the Sibyl of Cumæ, copyright 2005 by Flickr user "pverdicchio", http://flickr.com/photos/occhio/17923096/. Both used under Creative Commons license.) [[ATTN: Both are only licensed for non-commercial use, so find substitutes OR obtain rights for the for-money version of the book]]

10.2.3 Finding the Variance and Weights

All of this was possible because the Oracle told us what the variance function was. What do we do when the Oracle is not available (Figure 10.6)?

Sometimes we can work things out for ourselves, without needing an oracle.

- We know, empirically, the precision of our measurement of the response variable — we know how precise our instruments are, or the response is really an average of several measurements with known standard deviations, etc.
- We know how the noise in the response must depend on the input variables. For example, when taking polls or surveys, the variance of the proportions we find should be inversely proportional to the sample size. So we can make the weights proportional to the sample size.

Both of these outs rely on kinds of background knowledge which are easier to get in the natural or even the social sciences than in many industrial applications. However, there are approaches for other situations which try to use the observed residuals to get estimates of the heteroskedasticity; this is the topic of the next section.

10.3 Estimating Conditional Variance Functions

Remember that there are two equivalent ways of defining the variance:

$$\mathbb{V}[X] = \mathbb{E}\left[X^2\right] - (\mathbb{E}[X])^2 = \mathbb{E}\left[(X - \mathbb{E}[X])^2\right] \tag{10.28}$$

The latter is more useful for us when it comes to estimating variance functions. We have already figured out how to estimate means — that's what all this previous work on smoothing and regression is for — and the deviation of a random variable from its mean shows up as a residual.

There are two generic ways to estimate conditional variances, which differ slightly in how they use non-parametric smoothing. We can call these the **squared residuals method** and the **log squared residuals method**. Here is how the first one goes.

1. Estimate $\mu(x)$ with your favorite regression method, getting $\widehat{\mu}(x)$.
2. Construct the **squared residuals**, $u_i = (y_i - \widehat{\mu}(x_i))^2$.
3. Use your favorite *non-parametric* method to estimate the conditional mean of the u_i, call it $\widehat{q}(x)$.
4. Predict the variance using $\widehat{\sigma}_x^2 = \widehat{q}(x)$.

The log-squared residuals method goes very similarly.

1. Estimate $\mu(x)$ with your favorite regression method, getting $\widehat{\mu}(x)$.
2. Construct the **log squared residuals**, $z_i = \log (y_i - \widehat{\mu}(x_i))^2$.
3. Use your favorite *non-parametric* method to estimate the conditional mean of the z_i, call it $\widehat{s}(x)$.
4. Predict the variance using $\widehat{\sigma}_x^2 = \exp \widehat{s}(x)$.

The quantity $y_i - \widehat{\mu}(x_i)$ is the i^{th} residual. If $\widehat{\mu} \approx \mu$, then the residuals should have mean zero. Consequently the variance of the residuals (which is what we want) should equal the expected squared residual. So squaring the residuals makes sense, and the first method just smoothes these values to get at their expectations.

What about the second method — why the log? Basically, this is a convenience — squares are necessarily non-negative numbers, but lots of regression methods don't easily include constraints like that, and we really don't want to predict negative variances.[4] Taking the log gives us an unbounded range for the regression.

Strictly speaking, we don't need to use non-parametric smoothing for either method. If we had a parametric model for σ_x^2, we could just fit the parametric model to the squared residuals (or their logs). But even if you think you know what the variance function should look like it, why not check it?

We came to estimating the variance function because of wanting to do weighted least squares, but these methods can be used more generally. It's often important

[4] Occasionally people do things like claiming that gene differences explains more than 100% of the variance in some psychological trait, and so environment and up-bringing contribute negative variance. Some of them — like Alford *et al.* (2005) — say this with a straight face.

to understand variance in its own right, and this is a general method for estimating it. Our estimate of the variance function depends on first having a good estimate of the regression function

10.3.1 Iterative Refinement of Mean and Variance: An Example

The estimate $\widehat{\sigma}_x^2$ depends on the initial estimate of the regression function $\widehat{\mu}$. But, as we saw when we looked at weighted least squares, taking heteroskedasticity into account can change our estimates of the regression function. This suggests an iterative approach, where we alternate between estimating the regression function and the variance function, using each to improve the other. That is, we take either method above, and then, once we have estimated the variance function $\widehat{\sigma}_x^2$, we re-estimate $\widehat{\mu}$ using weighted least squares, with weights inversely proportional to our estimated variance. Since this will generally change our estimated regression, it will change the residuals as well. Once the residuals have changed, we should re-estimate the variance function. We keep going around this cycle until the change in the regression function becomes so small that we don't care about further modifications. It's hard to give a strict guarantee, but *usually* this sort of iterative improvement will converge.

Let's apply this idea to our example. Figure 10.3b already plotted the residuals from OLS. Figure 10.7 shows those squared residuals again, along with the true variance function and the estimated variance function.

The OLS estimate of the regression line is not especially good ($\widehat{\beta}_0 = 3.63$ versus $\beta_0 = 3$, $\widehat{\beta}_1 = -1.46$ versus $\beta_1 = -2$), so the residuals are systematically off, but it's clear from the figure that kernel smoothing of the squared residuals is picking up on the heteroskedasticity, and getting a pretty reasonable picture of the variance function.

Now we use the estimated variance function to re-estimate the regression line, with weighted least squares.

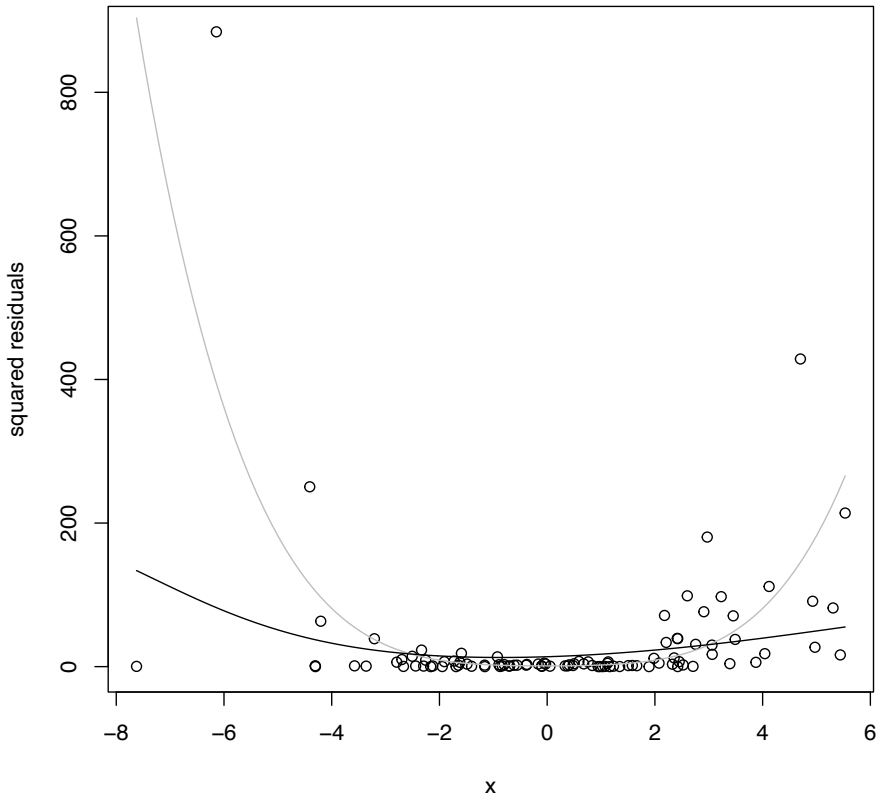
```
fit.wls1 <- lm(y ~ x, weights = 1/fitted(var1))
coefficients(fit.wls1)
## (Intercept)           x
##    3.212162   -1.476736
var2 <- npreg(residuals(fit.wls1)^2 ~ x)
```

The slope has changed substantially, and in the right direction (Figure 10.8a). The residuals have also changed (Figure 10.8b), and the new variance function is closer to the truth than the old one.

Since we have a new variance function, we can re-weight the data points and re-estimate the regression:

```
fit.wls2 <- lm(y ~ x, weights = 1/fitted(var2))
coefficients(fit.wls2)
## (Intercept)           x
##    3.203988   -1.480743
var3 <- npreg(residuals(fit.wls2)^2 ~ x)
```

Since we know that the true coefficients are 3 and -2, we know that this is moving in the right direction. If I hadn't told you what they were, you could still observe that the difference in coefficients between `fit.wls1` and `fit.wls2` is smaller than that between `fit.ols` and `fit.wls1`, which is a sign that this is converging.




```
plot(x, residuals(fit.ols)^2, ylab = "squared residuals")
curve((1 + x^2/2)^2, col = "grey", add = TRUE)
require(np)
var1 <- npreg(residuals(fit.ols)^2 ~ x)
grid.x <- seq(from = min(x), to = max(x), length.out = 300)
lines(grid.x, predict(var1, exdat = grid.x))
```

Figure 10.7 Points: actual squared residuals from the OLS line. Grey curve: true variance function, $\sigma_x^2 = (1 + x^2/2)^2$. Black curve: kernel smoothing of the squared residuals, using `npreg`.

I will spare you the plot of the new regression and of the new residuals. Let's iterate a few more times:

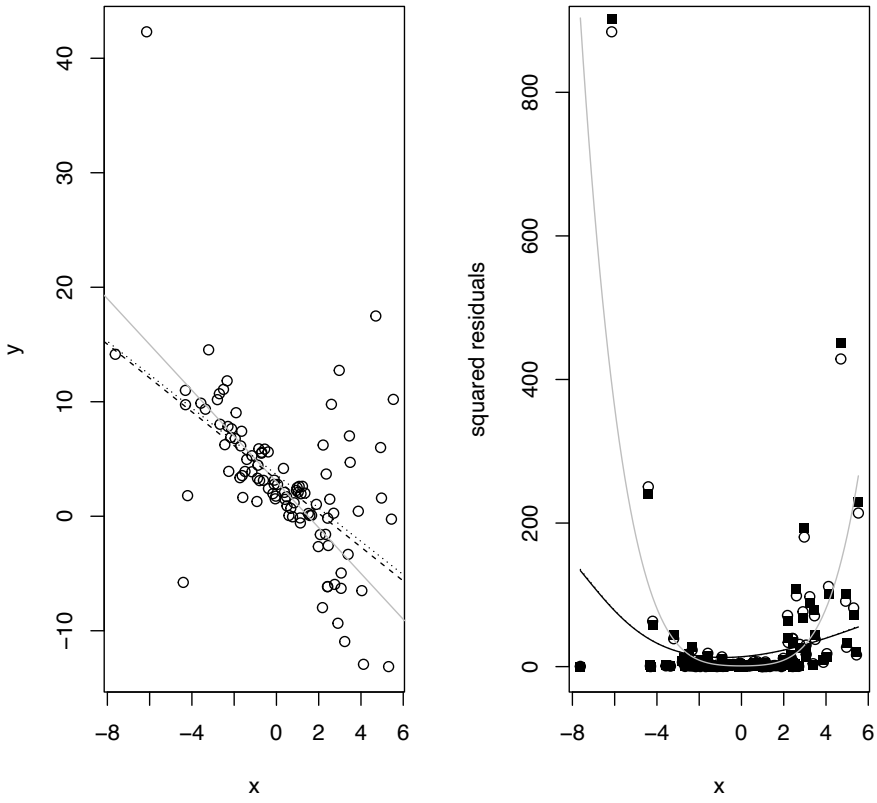


Figure 10.8 Left: As in Figure 10.2, but with the addition of the weighted least squares regression line (dotted), using the estimated variance from Figure 10.7 for weights. Right: As in Figure 10.7, but with the addition of the residuals from the WLS regression (black squares), and the new estimated variance function (dotted curve).

```
fit.wls3 <- lm(y ~ x, weights = 1/fitted(var3))
coefficients(fit.wls3)
## (Intercept)           x
##    3.203520   -1.481161
var4 <- npreg(residuals(fit.wls3)^2 ~ x)
fit.wls4 <- lm(y ~ x, weights = 1/fitted(var4))
coefficients(fit.wls4)
## (Intercept)           x
##    3.203475   -1.481204
```

By now, the coefficients of the regression are changing in the fourth significant digit, and we only have 100 data points, so the imprecision from a limited sample surely swamps the changes we're making, and we might as well stop.

Manually going back and forth between estimating the regression function and estimating the variance function is tedious. We could automate it with a function, which would look something like this:

```
iterative.wls <- function(x, y, tol = 0.01, max.iter = 100) {
    iteration <- 1
    old.coefs <- NA
    regression <- lm(y ~ x)
    coefs <- coefficients(regression)
    while (is.na(old.coefs) || ((max(coefs - old.coefs) > tol) && (iteration < max.iter))) {
        variance <- npreg(residuals(regression)^2 ~ x)
        old.coefs <- coefs
        iteration <- iteration + 1
        regression <- lm(y ~ x, weights = 1/fitted(variance))
        coefs <- coefficients(regression)
    }
    return(list(regression = regression, variance = variance, iterations = iteration))
}
```

This starts by doing an unweighted linear regression, and then alternates between WLS for the getting the regression and kernel smoothing for getting the variance. It stops when no parameter of the regression changes by more than `tol`, or when it's gone around the cycle `max.iter` times.[5] This code is a bit too inflexible to be really "industrial strength" (what if we wanted to use a data frame, or a more complex regression formula?), but shows the core idea.

[5] The condition in the `while` loop is a bit complicated, to ensure that the loop is executed at least once. Some languages have an `until` control structure which would simplify this.

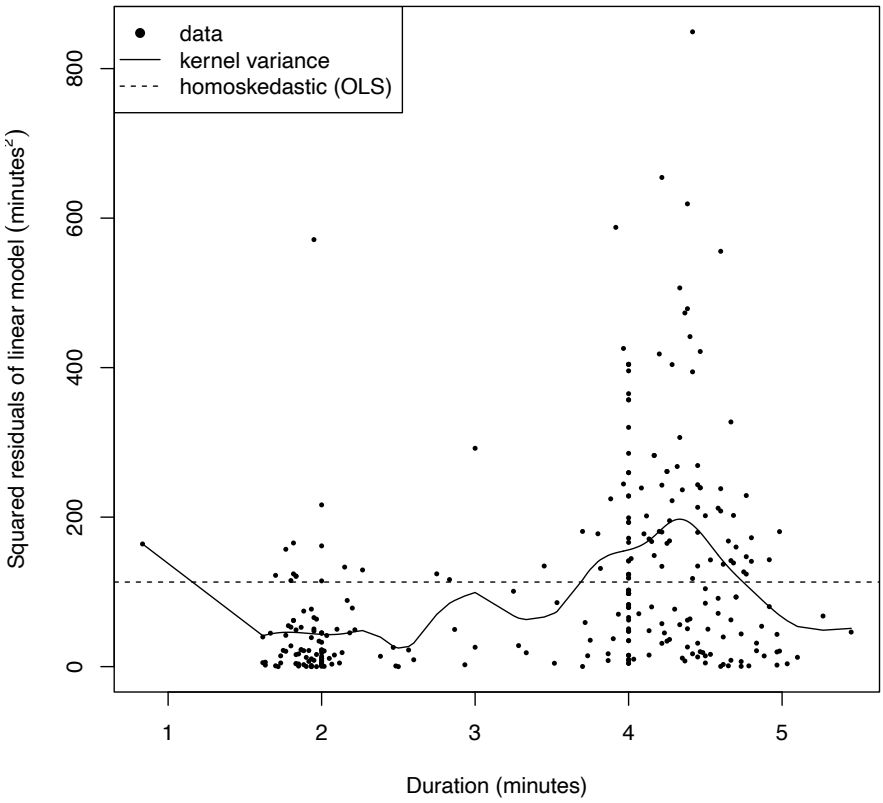
10.3.2 Real Data Example: Old Heteroskedastic

§5.4.2 introduced the `geyser` data set, which is about predicting the waiting time between consecutive eruptions of the "Old Faithful" geyser at Yellowstone National Park from the duration of the latest eruption. Our exploration there showed that a simple linear model (of the kind often fit to this data in textbooks and elementary classes) is not very good, and raised the suspicion that one important problem was heteroskedasticity. Let's follow up on that, building on the computational work done in that section.

The estimated variance function `geyser.var` does not look particularly flat, but it comes from applying a fairly complicated procedure (kernel smoothing with data-driven bandwidth selection) to a fairly limited amount of data (299 observations). Maybe that's the amount of wiggliness we should *expect* to see due to finite-sample fluctuations? To rule this out, we can make surrogate data from the homoskedastic model, treat it the same way as the real data, and plot the resulting variance functions (Figure 10.10). The conditional variance functions estimated from the homoskedastic model are flat or gently varying, with much less range than what's seen in the data.

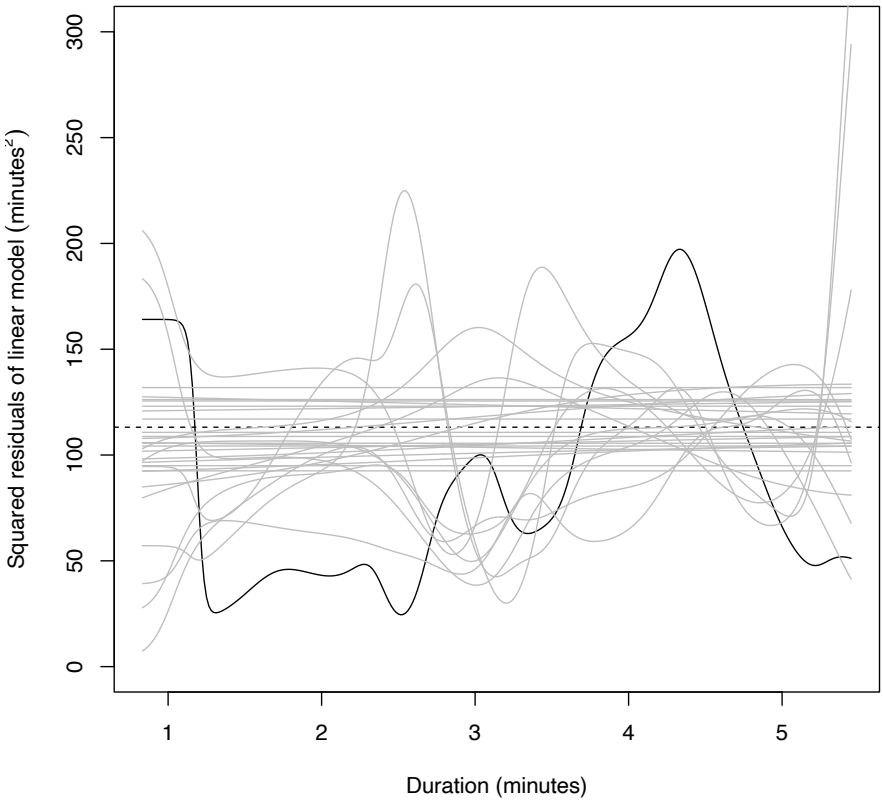
While that sort of qualitative comparison is genuinely informative, one can also be more quantitative. One might measure heteroskedasticity by, say, evaluating the conditional variance at all the data points, and looking at the ratio of the interquartile range to the median. This would be zero for perfect homoskedasticity, and grow as the dispersion of actual variances around the "typical" variance increased. For the data, this is `IQR(fitted(geyser.var))/median(fitted(geyser.var))` =. Simulations from the OLS model give values around 10^{-15}.

There is nothing particularly special about this measure of heteroskedasticity — after all, I just made it up. The broad point it illustrates is the one made in §5.4.2.1: whenever we have some sort of quantitative summary statistic we can calculate on our real data, we can also calculate the same statistic on realizations of the model, and the difference will then tell us something about how close the simulations, and so the model, come to the data. In this case, we learn that the linear, homoskedastic model seriously understates the variability of this data. That leaves open the question of whether the problem is the linearity or the homoskedasticity; I will leave that question to Exercise 10.6.




```
library(MASS)
data(geyser)
geyser.ols <- lm(waiting ~ duration, data = geyser)
plot(geyser$duration, residuals(geyser.ols)^2, cex = 0.5, pch = 16, xlab = "Duration (minutes)",
    ylab = expression(`Squared residuals of linear model `(minutes^2)))
geyser.var <- npreg(residuals(geyser.ols)^2 ~ geyser$duration)
duration.order <- order(geyser$duration)
lines(geyser$duration[duration.order], fitted(geyser.var)[duration.order])
abline(h = summary(geyser.ols)$sigma^2, lty = "dashed")
legend("topleft", legend = c("data", "kernel variance", "homoskedastic (OLS)"), lty = c("blank",
    "solid", "dashed"), pch = c(16, NA, NA))
```

Figure 10.9 Squared residuals from the linear model of Figure 5.1, plotted against duration, along with the unconditional, homoskedastic variance implicit in OLS (dashed), and a kernel-regression estimate of the conditional variance (solid).




```
duration.grid <- seq(from = min(geyser$duration), to = max(geyser$duration), length.out = 300)
plot(duration.grid, predict(geyser.var, exdat = duration.grid), ylim = c(0, 300),
    type = "l", xlab = "Duration (minutes)", ylab = expression(`Squared residuals of linear model `(
abline(h = summary(geyser.ols)$sigma^2, lty = "dashed")
one.var.func <- function() {
    fit <- lm(waiting ~ duration, data = rgeyser())
    var.func <- npreg(residuals(fit)^2 ~ geyser$duration)
    lines(duration.grid, predict(var.func, exdat = duration.grid), col = "grey")
}
invisible(replicate(30, one.var.func()))
```

Figure 10.10 The actual conditional variance function estimated from the Old Faithful data (and the linear regression), in black, plus the results of applying the same procedure to simulations from the homoskedastic linear regression model (grey lines; see §5.4.2 for the `rgeyser` function). The fact that the estimates from the simulations are mostly flat or gently sloped suggests that the changes in variance found in the data are likely too large to just be sampling noise.

10.4 Re-sampling Residuals with Heteroskedasticity

Re-sampling the residuals of a regression, as described in §6.4, assumes that the distribution of fluctuations around the regression curve is the same for all values of the input x. Under heteroskedasticity, this is of course not the case. Nonetheless, we can still re-sample residuals to get bootstrap confidence intervals, standard errors, and so forth, provided we define and scale them properly. If we have a conditional variance function $\hat{\sigma}^2(x)$, as well as the estimated regression function $\widehat{\mu}(x)$, we can combine them to re-sample heteroskedastic residuals.

1. Construct the standardized residuals, by dividing the actual residuals by the conditional standard deviation:
$$\eta_i = \epsilon_i / \hat{\sigma}(x_i) \tag{10.29}$$
The η_i should now be all the same magnitude (in distribution!), no matter where x_i is in the space of predictors.
2. Re-sample the η_i with replacement, to get $\tilde{\eta}_1, \ldots \tilde{\eta}_n$.
3. Set $\tilde{x}_i = x_i$.
4. Set $\tilde{y}_i = \widehat{\mu}(\tilde{x}_i) + \hat{\sigma}(\tilde{x}_i)\tilde{\eta}_i$.
5. Analyze the surrogate data $(\tilde{x}_1, \tilde{y}_1), \ldots (\tilde{x}_n, \tilde{y}_n)$ like it was real data.

Of course, this still assumes that the *only* difference in distribution for the noise at different values of x is the scale.

10.5 Local Linear Regression

Switching gears, recall from Chapter 2 that one reason it can be sensible to use a linear approximation to the true regression function μ is that we can typically Taylor-expand (App. B) the latter around any point x_0,

$$\mu(x) = \mu(x_0) + \sum_{k=1}^{\infty} \frac{(x-x_0)^k}{k!} \left. \frac{d^k \mu}{dx^k} \right|_{x=x_0} \tag{10.30}$$

and similarly with all the partial derivatives in higher dimensions. Truncating the series at first order, $\mu(x) \approx \mu(x_0) + (x-x_0)\mu'(x_0)$, we see the first derivative $\mu'(x_0)$ is the best linear prediction coefficient, at least if x close enough to x_0. The snag in this line of argument is that if $\mu(x)$ is nonlinear, then μ' isn't a constant, and the optimal linear predictor changes depending on where we want to make predictions.

However, statisticians are thrifty people, and having assembled all the machinery for linear regression, they are loathe to throw it away just because the fundamental model is wrong. If we can't fit one line, why not fit many? If each point has a different best linear regression, why not estimate them all? Thus the idea of **local** linear regression: fit a different linear regression everywhere, weighting the data points by how close they are to the point of interest[6].

The simplest approach we could take would be to divide up the range of x into so many bins, and fit a separate linear regression for each bin. This has at least three drawbacks. First, we get weird discontinuities at the boundaries between bins. Second, we induce an odd sort of bias, where our predictions near the boundaries of a bin depend strongly on data from one side of the bin, and not at all on nearby data points just across the border. Third, we need to pick the bins.

The next simplest approach would be to first figure out where we want to make a prediction (say x), and do a linear regression with all the data points which were sufficiently close, $|x_i - x| \leq h$ for some h. Now we are basically using a uniform-density kernel to weight the data points. This eliminates two problems from the binning idea — the examples we include are always centered on the x we're trying to get a prediction for, and we just need to pick one bandwidth h rather than placing all the bin boundaries. But still, each example point always has either weight 0 or weight 1, so our predictions change jerkily as training points fall into or out of the window. It generally works nicer to have the weights change more smoothly with the distance, starting off large and then gradually trailing to zero.

By now bells may be going off, as this sounds very similar to the kernel regression. In fact, kernel regression is what happens when we truncate Eq. 10.30 at *zeroth* order, getting **locally constant** regression. We set up the problem

$$\widehat{\mu}(x) = \operatorname*{argmin}_m \frac{1}{n} \sum_{i=1}^{n} w_i(x)(y_i - m)^2 \tag{10.31}$$

[6] Some people say "local linear" and some "local*ly* linear".

and get the solution

$$\widehat{\mu}(x) = \sum_{i=1}^{n} y_i \frac{w_i(x)}{\sum_{j=1}^{n} w_j(x)} \tag{10.32}$$

which just is our kernel regression, when the weights are proportional to the kernels, $w_i(x) \propto K(x_i, x)$. (Without loss of generality, we can take the constant of proportionality to be 1.)

What about **locally linear** regression? The optimization problem is

$$\left(\widehat{\mu}(x), \widehat{\beta}(x)\right) = \underset{m,\beta}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} w_i(x)(y_i - m - (x_i - x) \cdot \beta)^2 \tag{10.33}$$

where again we can make $w_i(x)$ proportional to some kernel function, $w_i(x) \propto K(x_i, x)$. To solve this, abuse notation slightly to define $z_i = (1, x_i - x)$, i.e., the displacement from x, with a 1 stuck at the beginning to (as usual) handle the intercept. Now, by the machinery above,

$$\left(\widehat{\mu}(x), \widehat{\beta}(x)\right) = (\mathbf{z}^T \mathbf{w}(x) \mathbf{z})^{-1} \mathbf{z}^T \mathbf{w}(x) \mathbf{y} \tag{10.34}$$

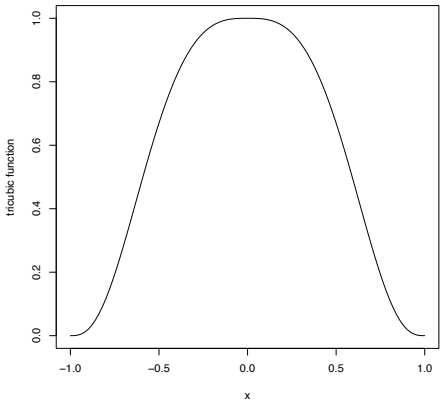
and the prediction is just the intercept, $\widehat{\mu}(x)$. If you need an estimate of the first derivatives, those are the $\widehat{\beta}(x)$. Eq. 10.34 guarantees that the weights given to each training point change smoothly with x, so the predictions will also change smoothly.[7]

Using a smooth kernel whose density is positive everywhere, like the Gaussian, ensures that the weights will change smoothly. But we could also use a kernel which goes to zero outside some finite range, so long as the kernel rises gradually from zero inside the range. For locally linear regression, a common choice of kernel is therefore the **tri-cubic**,

$$K(x_i, x) = \left(1 - \left(\frac{|x_i - x_0|}{h}\right)^3\right)^3 \tag{10.35}$$

if $|x - x_i| < h$, and $= 0$ otherwise (Figure 10.11).

[7] Notice that local linear predictors are still linear smoothers as defined in Chapter 1 (i.e., the predictions are linear in the y_i), but they are not, strictly speaking, *kernel* smoothers, since you can't re-write the last equation in the form of a kernel average.




```
curve((1 - abs(x)^3)^3, from = -1, to = 1, ylab = "tricubic function")
```

Figure 10.11 The tricubic kernel, with broad plateau where $|x| \approx 0$, and the smooth fall-off to zero at $|x| = 1$.

10.5.1 For and Against Locally Linear Regression

Why would we use locally linear regression, if we already have kernel regression?

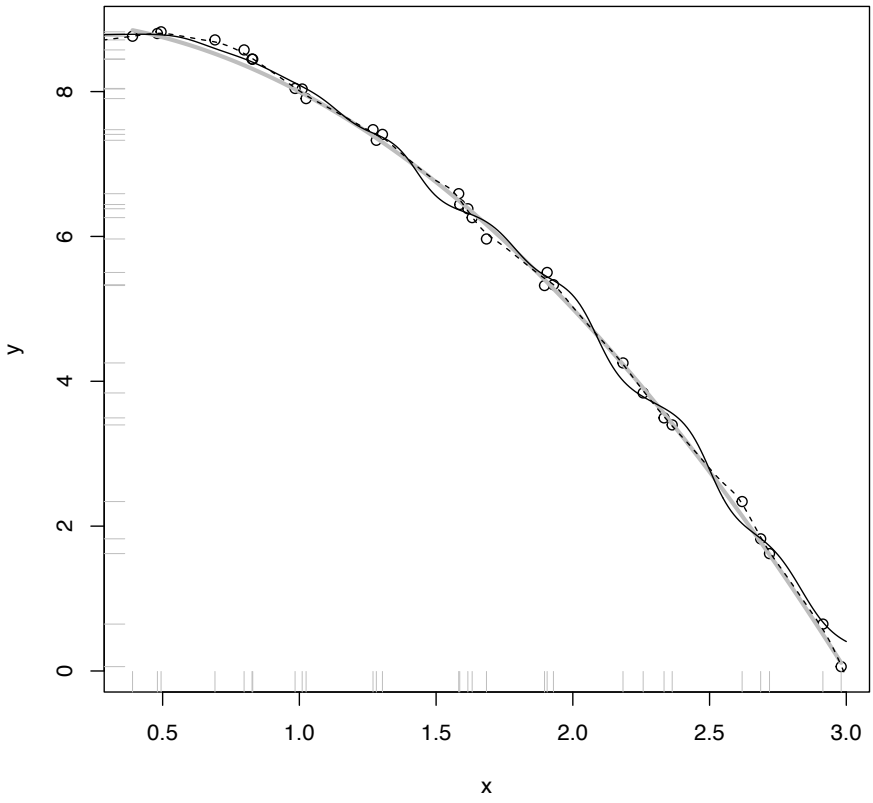
1. You may recall that when we worked out the bias of kernel smoothers (Eq. 4.10 in Chapter 4), we got a contribution that was proportional to $\mu'(x)$. If we do an analogous analysis for locally linear regression, the bias is the same, *except* that this derivative term goes away.
2. Relatedly, that analysis we did of kernel regression tacitly assumed the point we were looking at was in the middle of the training data (or at least rather more than h from the border). The bias gets worse near the edges of the training data. Suppose that the true $\mu(x)$ is decreasing in the vicinity of the largest x_i. (See the grey curve in Figure 10.12.) When we make our predictions there, in kernel regression we can only average values of y_i which tend to be systematically larger than the value we want to predict. This means that our kernel predictions are systematically biased upwards, and the size of the bias grows with $\mu'(x)$. (See the black line in Figure 10.12 at the lower right.) If we use a locally linear model, however, it can pick up that there is a trend, and reduce the edge bias by extrapolating it (dashed line in the figure).
3. The predictions of locally linear regression tend to be smoother than those of kernel regression, simply because we are locally fitting a smooth line rather than a flat constant. As a consequence, estimates of the derivative $\frac{d\widehat{\mu}}{dx}$ tend to be less noisy when $\widehat{\mu}$ comes from a locally linear model than a kernel regression.

Of course, total prediction error depends not only on the bias but also on the variance. Remarkably enough, the variance for kernel regression and locally linear regression is the same, at least asymptotically. Since locally linear regression has smaller bias, local-linear fits are often better predictors.

Despite all these advantages, local linear models have a real drawback. To make a prediction with a kernel smoother, we have to calculate a weighted average. To make a prediction with a local linear model, we have to solve a (weighted) linear least squares problem for each point, or each prediction. This takes much more computing time[8].

There are several packages which implement locally linear regression. Since we are already using `np`, one of the simplest is to set the `regtype="ll"` in

[8] Let's think this through. To find $\widehat{\mu}(x)$ with a kernel smoother, we need to calculate $K(x_i, x)$ for each x_i. If we've got p predictor variables and use a product kernel, that takes $O(pn)$ computational steps. We then need to add up the kernels to get the denominator, which we could certainly do in $O(n)$ more steps. (Could you do it faster?) Multiplying each weight by its y_i is a further $O(n)$, and the final adding up is at most $O(n)$; total, $O(pn)$. To make a prediction with a local linear model, we need to calculate the right-hand side of Eq. 10.34. Finding $(\mathbf{z}^T\mathbf{w}(x)\mathbf{z})$ means multiplying $[(p+1)\times n][n\times n][n\times(p+1)]$ matrices, which will take $O((p+1)^2 n) = O(p^2 n)$ steps. Inverting a $q\times q$ matrix takes $O(q^3)$ steps, so our inversion takes $O((p+1)^3) = O(p^3)$ steps. Just getting $(\mathbf{z}^T\mathbf{w}(x)\mathbf{z})^{-1}$ thus requires $O(p^3 + p^2 n)$. Finding the $(p+1)\times 1$ matrix $\mathbf{z}^T\mathbf{w}(x)\mathbf{y}$ similarly takes $O((p+1)n) = O(pn)$ steps, and the final matrix multiplication is $O((p+1)(p+1)) = O(p^2)$. Total, $O(p^2 n) + O(p^3)$. The speed advantage of kernel smoothing thus gets increasingly extreme as the number of predictor variables p grows.




```
x <- runif(30, max = 3)
y <- 9 - x^2 + rnorm(30, sd = 0.1)
plot(x, y)
rug(x, side = 1, col = "grey")
rug(y, side = 2, col = "grey")
curve(9 - x^2, col = "grey", add = TRUE, lwd = 3)
grid.x <- seq(from = 0, to = 3, length.out = 300)
np0 <- npreg(y ~ x)
lines(grid.x, predict(np0, exdat = grid.x))
np1 <- npreg(y ~ x, regtype = "ll")
lines(grid.x, predict(np1, exdat = grid.x), lty = "dashed")
```

Figure 10.12 Points are samples from the true, nonlinear regression function shown in grey. The solid black line is a kernel regression, and the dashed line is a locally linear regression. Note that the locally linear model is smoother than the kernel regression, and less biased when the true curve has a non-zero bias at a boundary of the data (far right).

`npreg`.[9] There are several other packages which support it, notably `KernSmooth` and `locpoly`.

As the name of the latter suggests, there is no reason we *have* to stop at locally linear models, and we could use local polynomials of any order. The main reason to use a higher-order local polynomial, rather than a locally-linear or locally-constant model, is to estimate higher derivatives. Since this is a somewhat specialized topic, I will not say more about it.

[9] `"ll"` stands for "locally linear", of course; the default is `regtype="lc"`, for "locally constant".

10.5.2 Lowess

There is however one additional topic in locally linear models which is worth mentioning. This is the variant called **lowess** or **loess**.[10] The basic idea is to fit a locally linear model, with a kernel which goes to zero outside a finite window and rises gradually inside it, typically the tri-cubic I plotted earlier. The wrinkle, however, is that rather than solving a least *squares* problem, it minimizes a different and more "robust" loss function,

$$\underset{\beta(x)}{\operatorname{argmin}} \frac{1}{n}\sum_{i=1}^{n} w_i(x)\ell(y - \vec{x}_i \cdot \beta(x)) \tag{10.36}$$

where $\ell(a)$ doesn't grow as rapidly for large a as a^2. The idea is to make the fitting less vulnerable to occasional large outliers, which would have very large squared errors, unless the regression curve went far out of its way to accommodate them. For instance, we might have $\ell(a) = a^2$ if $|a| < 1$, and $\ell(a) = 2|a| - 1$ otherwise.[11] There is a large theory of robust estimation, largely parallel to the more familiar least-squares theory. In the interest of space, we won't pursue it further, but lowess is worth mentioning because it's such a common smoothing technique, especially for sheer visualization.

Lowess smoothing is implemented in base R through the function `lowess` (rather basic), and through the function `loess` (more sophisticated), as well as in the CRAN package `locfit` (more sophisticated still). The lowess idea can be combined with local fitting of higher-order polynomials; the `loess` and `locfit` commands both support this.

[10] I have heard this name explained as an acronym for both "locally weighted scatterplot smoothing" and "locally weight sum of squares".

[11] This is called the **Huber loss**; it continuously interpolates between looking like squared error and looking like absolute error. This means that when errors are small, it gives results very like least-squares, but it is resistant to outliers. See also App. J.6.1.

10.6 Further Reading

Weighted least squares goes back to the 19th century, almost as far back as ordinary least squares; see the references in chapter 1 and 2.

I am not sure who invented the use of smoothing to estimate variance functions; I learned it from Wasserman (2006, pp. 87–88). I've occasionally seen it done with a linear model for the conditional variance, which I can't recommend.

Simonoff (1996) is a good reference on local linear and local polynomial models, including actually doing the bias-variance analyses where I've just made empty "it can be shown" promises. Fan and Gijbels (1996) is more comprehensive, but also a much harder read. Lowess was introduced by Cleveland (1979), but the name evidently came later (since it doesn't appear in that paper).

Exercises

10.1 Imagine we are trying to estimate the mean value of Y from a large population of size n_0, so $\overline{y} = n_0^{-1} \sum_{j=1}^{n} y_j$. We observe $n \ll n_0$ members of the population, with individual i being included in our sample with a probability proportional to π_i.

1. Show that $\left(\sum_{i=1}^{n} y_i/\pi_i\right) / \sum_{i'=1}^{n} 1/\pi_{i'}$ is a consistent estimator of $\overline{y}$, by showing that that it is unbiased and it has a variance that shrinks with n towards 0.
2. Is the unweighted sample mean $n^{-1} \sum_{i=1}^{n} y_i$ a consistent estimator of $\overline{y}$ when the π_i are not all equal?

10.2 Show that the model of Eq. 10.13 has the log-likelihood given by Eq. 10.14.

10.3 Do the calculus to verify Eq. 10.4.

10.4 Is $w_i = 1$ a necessary as well as a sufficient condition for Eq. 10.3 and Eq. 10.1 to have the same minimum?

10.5 §10.2.2 showed that WLS gives better parameter estimates than OLS when there is heteroskedasticity, and we know and use the variance. Modify the code for to see which one has better generalization error.

10.6 §10.3.2 looked at the residuals of the linear regression model for the Old Faithful geyser data, and showed that they would imply lots of heteroskedasticity. This might, however, be an artifact of inappropriately using a linear model. Use either kernel regression (cf. §6.4.2) or local linear regression to estimate the conditional mean of waiting given duration, and see whether the apparent heteroskedasticity goes away.

10.7 Should local linear regression do better or worse than ordinary least squares under heteroskedasticity? What exactly would this mean, and how might you test your ideas?