Chapter 10

Testing Parametric Regression Specifications with Nonparametric Regression

10.1 Testing Functional Forms

One important, but under-appreciated, use of nonparametric regression is in testing whether parametric regressions are well-specified.

The typical parametric regression model is something like

$$Y = f(X;\theta) + \epsilon \tag{10.1}$$

where f is some function which is completely specified except for the adjustable parameters θ, and ϵ, as usual, is uncorrelated noise. Usually, but not necessarily, people use a function f that is linear in the variables in X, or perhaps includes some interactions between them.

How can we tell if the specification is right? If, for example, it's a linear model, how can we check whether there might not be some nonlinearity? One common approach is to modify the specification by adding in *specific* departures from the modeling assumptions — say, adding a quadratic term — and seeing whether the coefficients that go with those terms are significantly non-zero, or whether the improvement in fit is significant.[1] For example, one might compare the model

$$Y = \theta_1 x_1 + \theta_2 x_2 + \epsilon \tag{10.2}$$

to the model

$$Y = \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \epsilon \tag{10.3}$$

by checking whether the estimated θ_3 is significantly different from 0, or whether the residuals from the second model are significantly smaller than the residuals from the first.

[1] In my experience, this is second in popularity only to ignoring the issue.

This can work, *if* you have chosen the right nonlinearity to test. It has the power to detect certain mis-specifications, if they exist, but not others. (What if the departure from linearity is not quadratic but cubic?) If you have good reasons to think that when the model is wrong, it can only be wrong in certain ways, fine; if not, though, why only check for those errors?

Nonparametric regression effectively lets you check for *all* kinds of systematic errors, rather than singling out a particular one. There are three basic approaches, which I give in order of increasing sophistication.

- If the parametric model is right, it should predict as well as, or even better than, the non-parametric one, and we can check whether $MSE_p(\widehat{\theta}) - MSE_{np}(\widehat{r})$ is sufficiently small.
- If the parametric model is right, the non-parametric estimated regression curve should be very close to the parametric one. So we can check whether $f(x;\widehat{\theta}) - \widehat{r}(x)$ is approximately zero everywhere.
- If the parametric model is right, then its *residuals* should be patternless and independent of input features, because

$$\mathbf{E}[Y - f(x;\theta)|X] = \mathbf{E}[f(x;\theta) + \epsilon - f(x;\theta)|X] = \mathbf{E}[\epsilon|X] = 0 \tag{10.4}$$

  So we can apply non-parametric smoothing to the parametric residuals, $y - f(x;\widehat{\theta})$, and see if their expectation is approximately zero everywhere.

We'll stick with the first procedure, because it's simpler for us to implement computationally. However, it turns out to be easier to develop theory for the other two, and especially for the third — see Li and Racine (2007, ch. 12), or Hart (1997).

Here is the basic procedure.

1. Get data $(x_1, y_1), (x_2, y_2), \ldots (x_n, y_n)$.
2. Fit the parametric model, getting an estimate $\widehat{\theta}$, and in-sample mean-squared error $MSE_p(\widehat{\theta})$.
3. Fit your favorite nonparametric regression (using cross-validation to pick control settings as necessary), getting curve $\widehat{r}$ and in-sample mean-squared error $MSE_{np}(\widehat{r})$.
4. Calculate $\widehat{t} = MSE_p(\widehat{\theta}) - MSE_{np}(\widehat{r})$.
5. Simulate from the parametric model $\widehat{\theta}$ to get faked data $(x'_1, y'_1), \ldots (x'_n, y'_n)$.
6. Fit the parametric model to the simulated data, getting estimate $\tilde{\theta}$ and $MSE_p(\tilde{\theta})$.
7. Fit the nonparametric model to the simulated data, getting estimate $\tilde{r}$ and $MSE_{np}(\tilde{r})$.

8. Calculate $\tilde{T} = MSE_p(\tilde{\theta}) - MSE_{np}(\tilde{r})$.

9. Repeat steps 5–8 many times to get an estimate of the distribution of T.

10. The p-value is $\frac{1+\#\left\{\tilde{T} > \hat{t}\right\}}{1+\#T}$.

Let's step through the logic. In general, the error of the non-parametric model will be converging to the smallest level compatible with the intrinsic noise of the process. What about the parametric model?

Suppose on the one hand that the parametric model *is* correctly specified. Then its error will also be converging to the minimum — by assumption, it's got the functional form right so bias will go to zero, and as $\widehat{\theta} \to \theta_0$, the variance will also go to zero. In fact, with enough data the correctly-specified parametric model will actually *generalize* better than the non-parametric model[2].

Suppose on the other hand that the parametric model is mis-specified. Then it is predictions are systematically wrong, even with unlimited amounts of data — there's some bias which never goes away, no matter how big the sample. Since the non-parametric smoother does eventually come arbitrarily close to the true regression function, the smoother will end up predicting better than the parametric model.

Smaller errors for the smoother, then, suggest that the parametric model is wrong. But since the smoother has higher capacity, it could easily get smaller errors on a particular sample by chance and/or over-fitting, so only *big* differences in error count as evidence. Simulating from the parametric model gives us surrogate data which looks just like reality ought to, *if* the model is true. We then see how much better we could expect the non-parametric smoother to fit *under the parametric model*. If the non-parametric smoother fits the actual data much better than this, we can reject the parametric model with high confidence: it's really unlikely that we'd see that big an improvement from using the nonparametric model just by luck.[3]

As usual, we simulate from the parametric model simply because we have no hope of working out the distribution of the differences in MSEs from first principles. This is an example of our general strategy of bootstrapping.

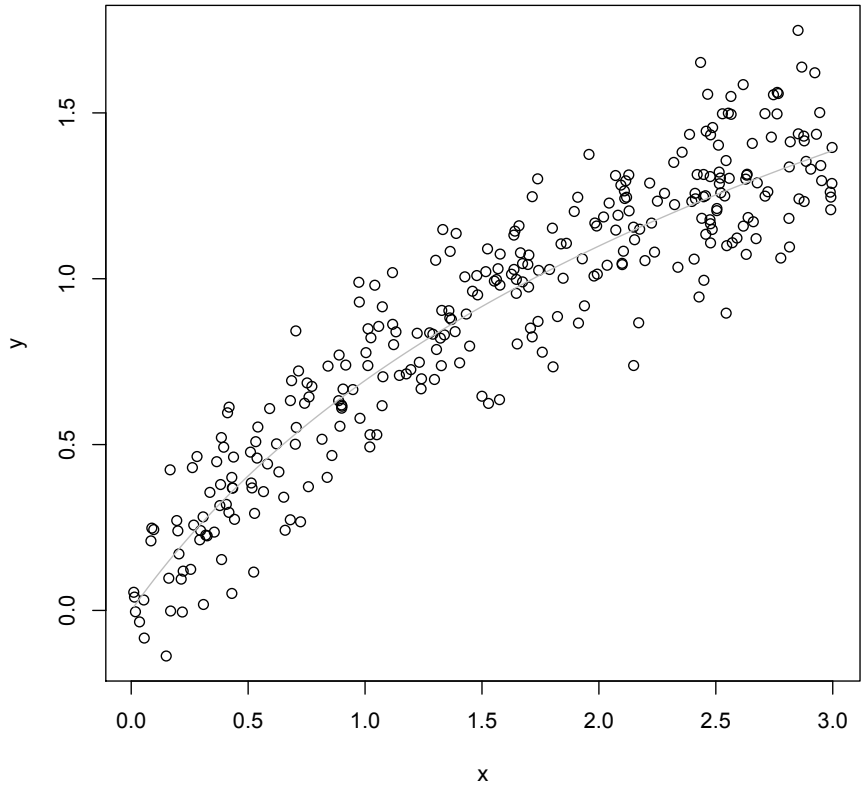
10.1.1 Examples of Testing a Parametric Model

Let's see this in action. First, let's detect a reasonably subtle nonlinearity. Take the non-linear function $g(x) = \log x + 1$, and say that $Y = g(x) + \epsilon$, with ϵ being IID Gaussian noise with mean 0 and standard deviation 0.15. (This is one of the two examples from the notes to Lecture 4.) Figure 10.1 shows the regression function and the data. The nonlinearity is clear with the curve to "guide the eye", but fairly subtle.

A simple linear regression looks pretty good:

[2]Remember that the smoother must, so to speak, use up some of the degrees of freedom in the data to figure out the shape of the regression function. The parametric model, on the other hand, takes the shape of the basic shape regression function as given, and uses all the degrees of freedom to tune its parameters.

[3]As usual with p-values, this is not symmetric. A high p-value might mean that the true regression function is very close to $r(x;\theta)$, or it might just mean that we don't have enough data to draw conclusions.




```
x  <- runif(300,0,3)
yg <- log(x+1)+rnorm(length(x),0,0.15)
gframe <- data.frame(x=x,y=yg)
plot(x,yg,xlab="x",ylab="y")
curve(log(1+x),col="grey",add=TRUE)
```

Figure 10.1: True regression curve (grey) and data points (circles). The curve $g(x) = \log x + 1$.

```
> glinfit = lm(y~x,data=gframe)
> print(summary(glinfit),signif.stars=FALSE,digits=2)

Call:
lm(formula = y ~ x, data = gframe)

Residuals:
   Min     1Q Median     3Q    Max
-0.416 -0.115  0.004  0.118  0.387

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)    0.208      0.019      11   <2e-16
x              0.434      0.011      41   <2e-16

Residual standard error: 0.16 on 298 degrees of freedom
Multiple R-squared: 0.85,Adjusted R-squared: 0.85
F-statistic: 1.7e+03 on 1 and 298 DF,  p-value: <2e-16
```

R^2 is ridiculously high — the regression line preserves 85% of the variance in the data. The p-value reported by R is also very, very low, which seems good, but remember all this really means is "you'd have to be crazy to think a flat line fit better than one with a slope" (Figure 10.2)

The in-sample MSE of the linear fit[4]

```
> mean(residuals(glinfit)^2)
[1] 0.02617729
```

The nonparametric regression has a somewhat smaller MSE[5]

```
> gnpr <- npreg(y~x,data=gframe)

> gnpr$MSE
[1] 0.02163506
```

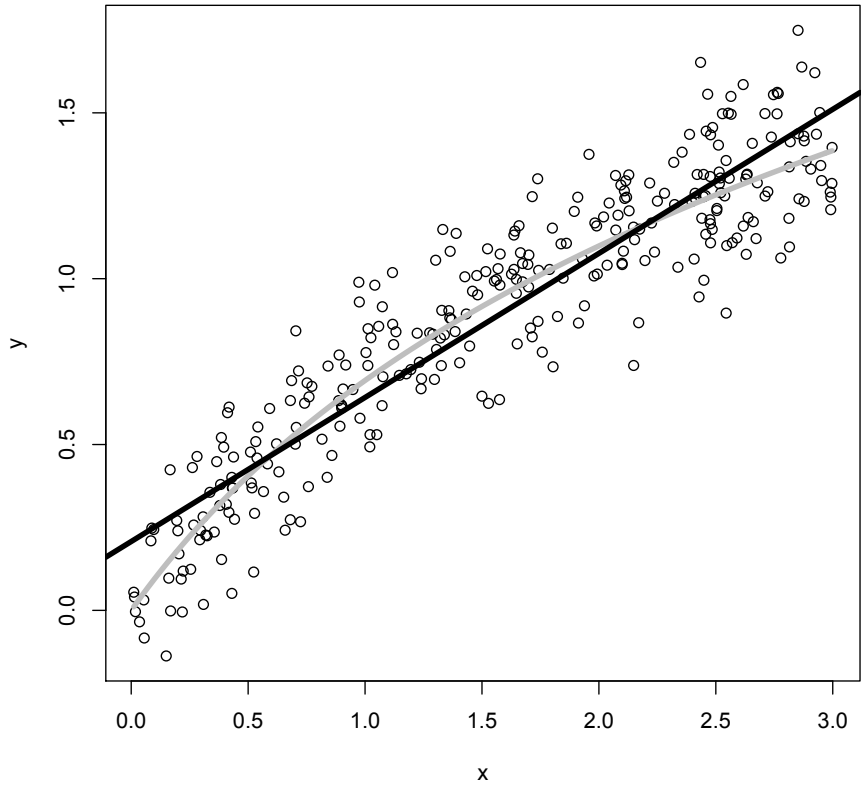
So $\hat{t} = 0.0045$:

```
> t.hat = mean(glinfit$residual^2) - gnpr$MSE
> t.hat
[1] 0.004542232
```

Now we need to simulate from the fitted parametric model, using its estimated coefficients and noise level. We have seen several times now how to do this. The function `sim.lm` in Example 19 does this, along the same lines as the examples in

[4] If we ask R for the MSE, by doing `summary(glinfit)$sigma`$\hat{2}$, we get 0.02635298. These differ by a factor of $n/(n-2) = 300/298 = 1.0067$, because R is trying to estimate the out-of-sample error by scaling up the in-sample error, the same way the estimated population variance scales up the sample variance. We want to compare in-sample fits.

[5] `npreg` does not apply the kind of correction mentioned in the previous footnote.




```
plot(x,yg,xlab="x",ylab="y")
curve(log(1+x),col="grey",add=TRUE,lwd=4)
abline(glinfit,lwd=4)
```

Figure 10.2: As previous figure, but adding the least-squares regression line (black). Line widths exaggerated for clarity.

```
# One surrogate data set for simple linear regression
# Inputs: linear model (linfit), x values at which to
  # simulate (test.x)
# Outputs: Data frame with columns x and y
sim.lm <- function(linfit, test.x) {
  n <- length(test.x)
  sim.frame <- data.frame(x=test.x)
    # Add the y column later
  sigma <- summary(linfit)$sigma*(n-2)/n  # MLE value
  y.sim <- predict(linfit,newdata=sim.frame)
  y.sim <- y.sim + rnorm(n,0,sigma) # Add noise
  sim.frame <- data.frame(sim.frame,y=y.sim) # Adds column
  return(sim.frame)
}
```

Code Example 19: Simulate a new data set from a linear model, assuming homoskedastic Gaussian noise. It also assumes that there is one input variable, `x`, and that the response variable is called `y`. Could you modify it to work with multiple regression?

Chapter 5; it assumes homoskedastic Gaussian noise. Again, as before, we need a function which will calculate the difference in MSEs between a linear model and a kernel smoother fit to the same data set — which will do automatically what we did by hand above. This is `calc.T` in Example 20. Note that the kernel bandwidth has to be re-tuned to each new data set.

If we call `calc.T` on the output of `sim.lm`, we get one value of the test statistic under the null distribution:

```
> calc.T(sim.lm(glinfit,x))
[1] 0.001513319
```

Now we just repeat this a lot to get a good approximation to the sampling distribution of T under the null hypothesis:

```
null.samples.T <- replicate(200,calc.T(sim.lm(glinfit,x)))
```

This takes some time, because each replication involves not just generating a new simulation sample, but also cross-validation to pick a bandwidth. This adds up to about a second per replicate on my laptop, and so a couple of minutes for 200 replicates.

(While the computer is thinking, look at the command a little more closely. It leaves the `x` values alone, and only uses simulation to generate new y values. This is appropriate here because our model doesn't really *say* where the `x` values came from; it's just about the conditional distribution of Y *given* X. If the model we were testing specified a distribution for x, we should generate x each time we invoke `calc.T`. If the specification is vague, like "x is IID" but with no particular distribution, then use the nonparametric bootstrap. The command would be something like

```
# Calculate the difference-in-MSEs test statistic
# Inputs: A data frame (my.data)
# Presumes: data has columns "x" and "y", which are input
  # and response
# Calls: np::npreg
# Output: Difference in MSEs between linear model and
  # kernel smoother
calc.T <- function(data) {
  # Fit the linear model, extract residuals, calculate MSE
  MSE.p <- mean((lm(y~x, data=data)$residuals)^2)
  # npreg gets unhappy when called with a "data" argument
  #  that is defined inside this function; npregbw does
  # not complain
  MSE.np.bw <- npregbw(y~x,data=data)
  MSE.np <- npreg(MSE.np.bw)$MSE
  return(MSE.p - MSE.np)
}
```

Code Example 20: Calculate the difference-in-MSEs test statistic.

```
replicate(200,calc.T(sim.lm(glinfit,resample(x)))
```

using the `resample` function from lecture 8, to draw a different bootstrap sample of x each time.)

When it's done, we can plot the distribution and see that the observed value $\hat{t}$ is pretty far out along the right tail (Figure 10.3). This tells us that it's very unlikely that `npreg` would improve so much on the linear model if the latter were true. In fact, *none* of the bootstrap replicates were that big:

```
> sum(null.samples.T > t.hat)
[1] 0
```

so our estimated p-value is $\frac{1}{201}$. We can thus reject the linear model pretty confidently.[6]

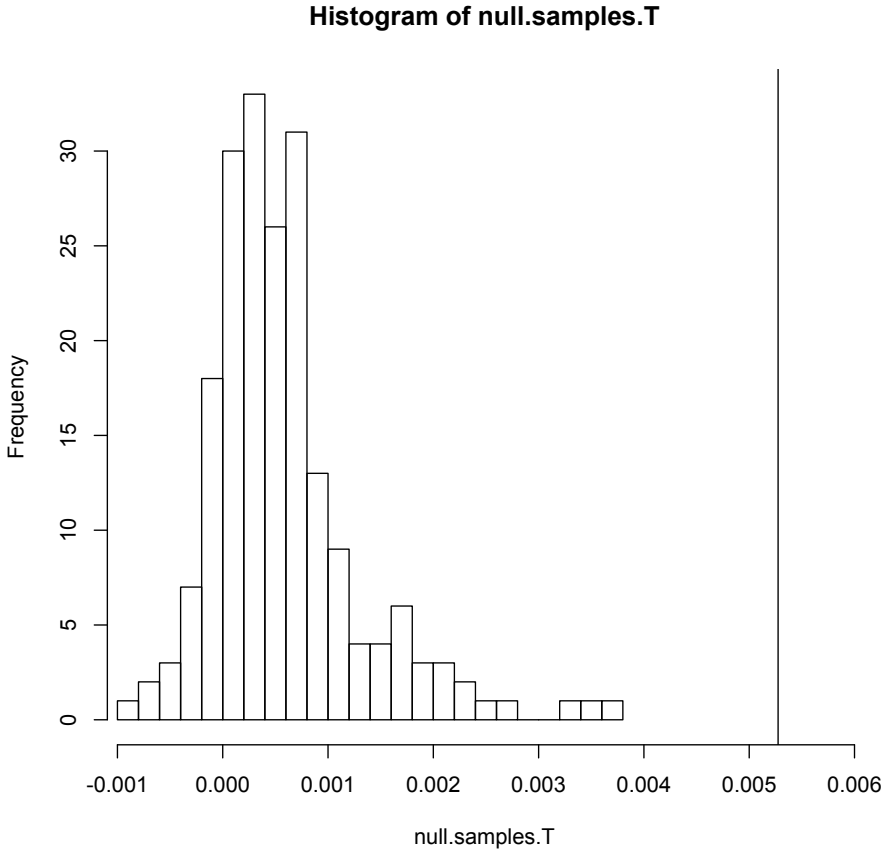
As a second example, let's suppose that the linear model *is* right — then the test should give us a high p-value. So let us stipulate that in reality

$$Y = 0.2 + 0.5x + \eta \tag{10.5}$$

with $\eta \sim \mathcal{N}(0, 0.15^2)$. Figure 10.4 shows data from this, of the same size as before.

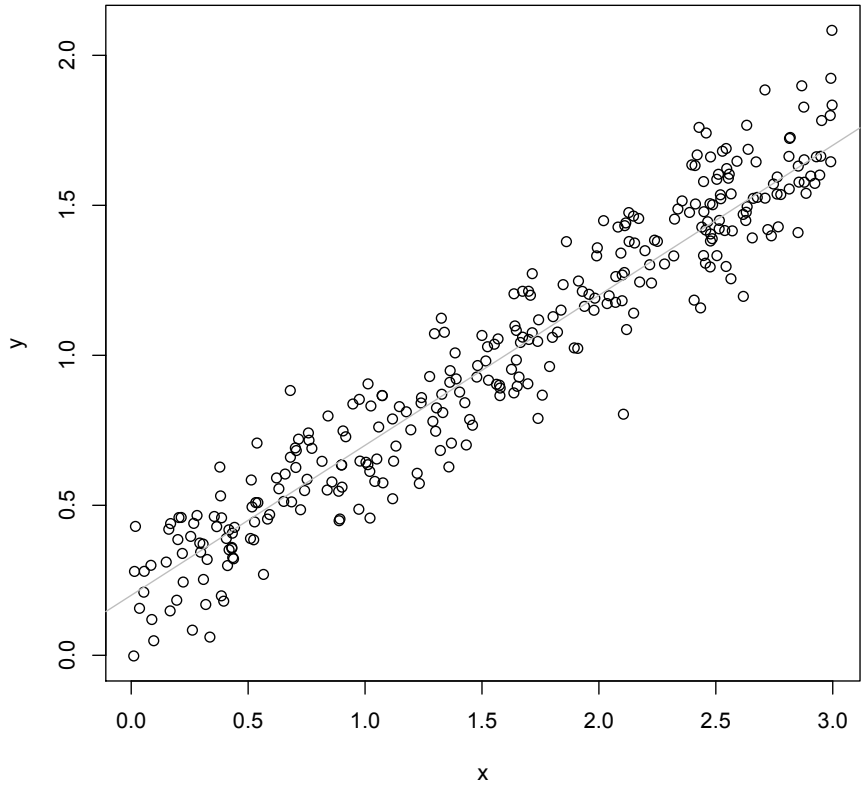
Repeating the same exercise as before, we get that $\hat{t} = 6.8 \times 10^{-4}$, together with a slightly different null distribution (Figure 10.5). Now the p-value is 32%, which one would be quite rash to reject.

[6] If we wanted a more precise estimate of the p-value, we'd need to use more bootstrap samples.



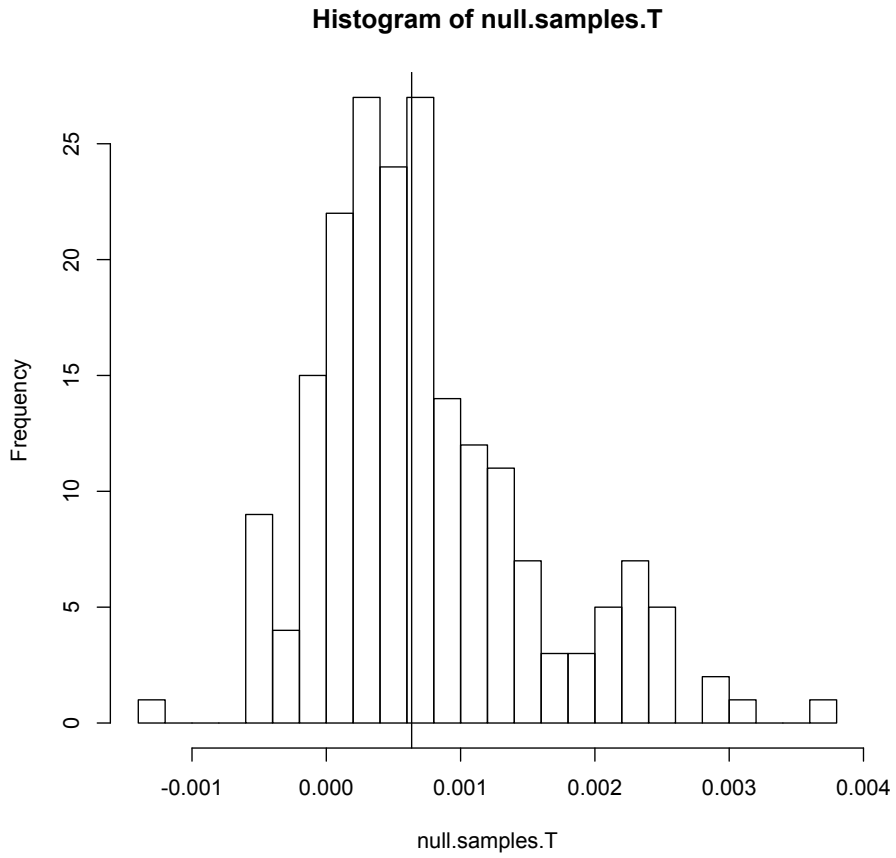

```
hist(null.samples.T,n=31,xlim=c(min(null.samples.T),1.1*t.hat),probability=TRUE)
abline(v=t.hat)
```

Figure 10.3: Histogram of the distribution of $T = MSE_p - MSE_{np}$ for data simulated from the parametric model. The vertical line mark the observed value. Notice that the mode is positive and the distribution is right-skewed; this is typical.




```
y2 = 0.2+0.5*x + rnorm(length(x),0,0.15)
y2.frame <- data.frame(x=x,y=y2)
plot(x,y2,xlab="x",ylab="y")
abline(0.2,0.5,col="grey")
```

Figure 10.4: Data from the linear model (true regression line in grey).




```
y2.fit <- lm(y~x,data=y2.frame)
null.samples.T.y2 <- replicate(200,calc.T(sim.lm(y2.fit,x)))
t.hat2 <- calc.T(y2.frame)
hist(null.samples.T.y2,n=31,probability=TRUE)
abline(v=t.hat2)
```

Figure 10.5: As in Figure 10.3, but using the data and fits from Figure 10.4.

10.1.2 Remarks

Other Nonparametric Regressions There is nothing especially magical about using kernel regression here. Any consistent nonparametric estimator (say, your favorite spline) would work. They may differ somewhat in their answers on particular cases.

Additive Alternatives For multivariate regressions, testing against a fully nonparametric alternative can be very time-consuming, as well as running up against curse-of-dimensionality issues[7]. A compromise is to test the parametric regression against an additive model. Essentially nothing has to change.

Testing $\mathbf{E}[\hat{\epsilon}|X] = 0$ I mentioned at the beginning of the chapter that one way to test whether the parametric model is correctly specified is to test whether the residuals have expectation zero everywhere. This amounts to (i) finding the residuals by fitting the parametric model, and (ii) comparing the MSE of the "model" that they have expectation zero with a nonparametric smoothing of the residuals. We just have to be careful that we simulate from the fitted parametric model, and not just by resampling the residuals.

Stabilizing the Sampling Distribution of the Test Statistic I have just looked at the difference in MSEs. The bootstrap principle being invoked is that the sampling distribution of the test statistic, under the estimated parametric model, should be close the distribution under the true parameter value. As discussed in Chapter 5, sometimes some massaging of the test statistic helps bring these distributions closer. Some modifications to consider:

- Divide the MSE difference by an estimate of the noise σ.
- Divide by an estimate of the noise σ times the difference in degrees of freedom, using the estimated, effective degrees of freedom of the nonparametric regression.
- Use the log ratio of MSEs instead of the MSE difference.

Doing a double bootstrap can help you assess whether these are necessary.

[7] This curse manifests itself here as a loss of power in the test.

10.2 Why Use Parametric Models At All?

It might seem by this point that there is little point to using parametric models at all. Either our favorite parametric model is right, or it isn't. If it is right, then a consistent nonparametric estimate will eventually approximate it arbitrarily closely. If the parametric model is wrong, it will not self-correct, but the non-parametric estimate will eventually show us that the parametric model doesn't work. Either way, the parametric model seems superfluous.

There are two things wrong with this line of reasoning — two good reasons to use parametric models.

1. One use of statistical models, like regression models, is to connect scientific theories to data. The theories are about the mechanisms generating the data. Sometimes these hypotheses are "tight" enough to tell us what the functional form of the regression should be, or even what the distribution of noise terms should be, but still contain unknown parameters. In this case, the parameters themselves are substantively meaningful and interesting — we *don't* just care about prediction. It can be very hard to relate non-parametric smoothing curves to aspects of scientific theories in the same way.[8]

2. Even if all we care about *is* prediction accuracy, there is still the bias-variance trade-off to consider. Non-parametric smoothers will have larger variance in their predictions, at the same sample size, than correctly-specified parametric models, simply because the former are more flexible. Both models are converging on the true regression function, but the parametric model converges faster, because it searches over a more confined space. In terms of total prediction error, the parametric model's low variance plus vanishing bias beats the non-parametric smoother's larger variance plus vanishing bias. (Remember that this is part of the logic of testing parametric models in the previous section.) In the next section, we will see that this argument can actually be pushed further, to work with not-quite-correctly specified models.

Of course, both of these advantages of parametric models only obtain *if* they are well-specified. If we want to claim those advantages, we need to check the specification.

[8] On the other hand, it is not uncommon for scientists to write down theories positing linear relationships between variables, not because they actually believe that, but because that's the only thing they know how to estimate statistically.

10.3 Why We Sometimes Want Mis-Specified Parametric Models

Low-dimensional parametric models have potentially high bias (if the real regression curve is very different from what the model posits), but low variance (because there isn't that much to estimate). Non-parametric regression models have low bias (they're flexible) but high variance (they're flexible). If the parametric model is true, it can converge *faster* than the non-parametric one. Even if the parametric model isn't quite true, a small bias plus low variance can sometimes still beat a non-parametric smoother's smaller bias and substantial variance. With enough data the non-parametric smoother will eventually over-take the mis-specified parametric model, but with small samples we might be better off embracing bias.

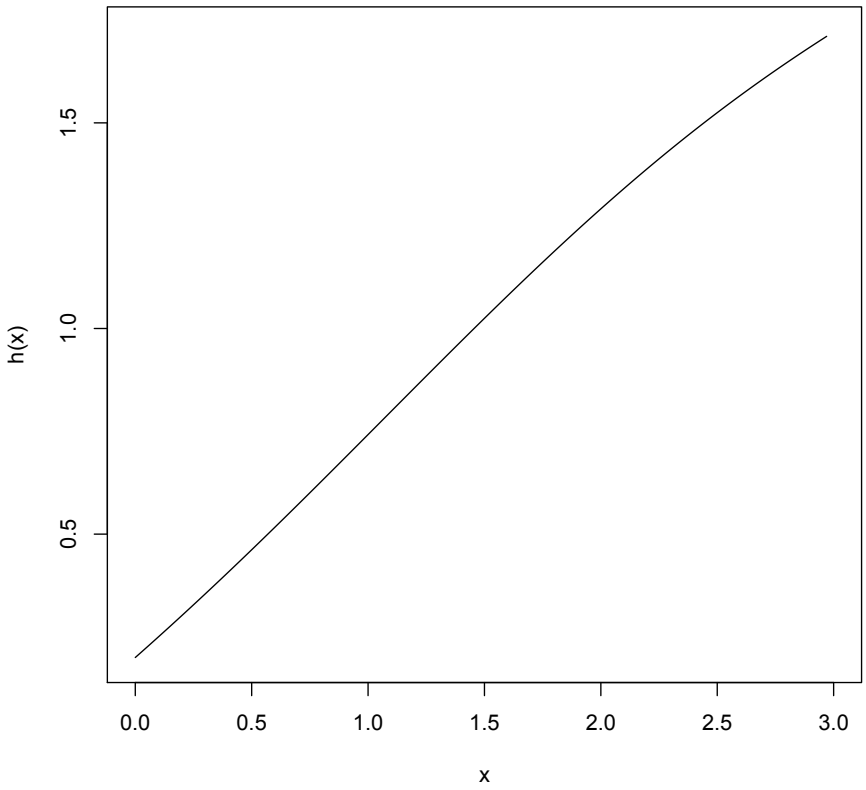
To illustrate, suppose that the true regression function is

$$\mathbf{E}[Y|X=x] = 0.2 + \frac{1}{2}\left(1 + \frac{\sin x}{10}\right)x \tag{10.6}$$

This is very nearly linear over small ranges — say $x \in [0,3]$ (Figure 10.6).

I will use the fact that I know the true model here to calculate the actual expected generalization error, by averaging over many samples (Example 21).

Figure 10.7 shows that, out to a fairly substantial sample size (≈ 500), the lower bias of the non-parametric regression is systematically beaten by the lower variance of the linear model — though admittedly not by much.



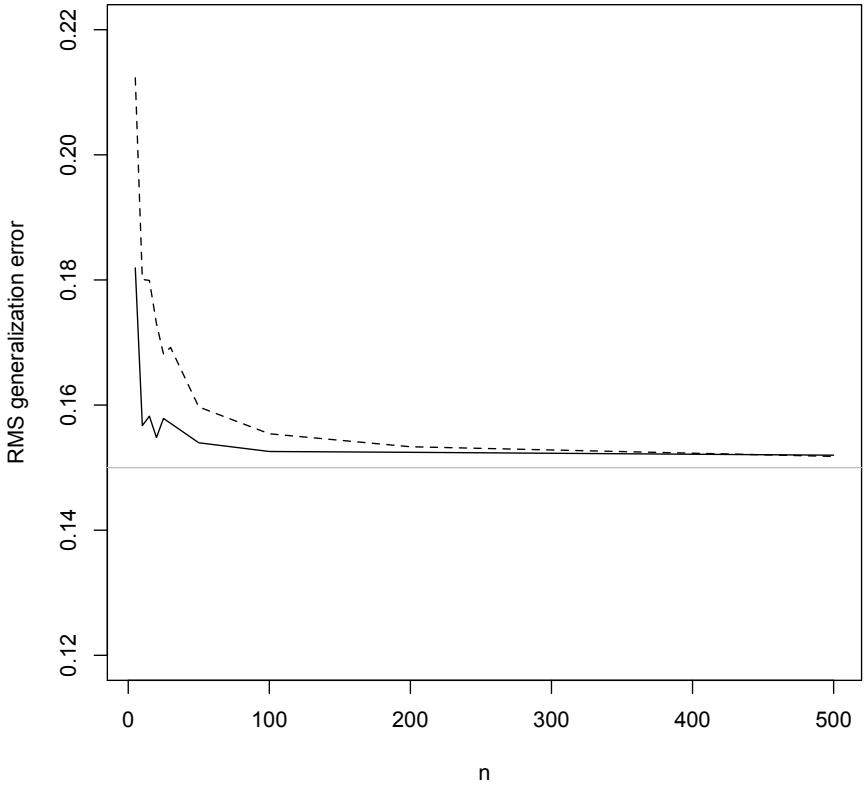

```
h <- function(x) { 0.2 + 0.5*(1+sin(x)/10)*x }
curve(h(x),from=0,to=3)
```

Figure 10.6: Graph of $h(x) = 0.2 + \frac{1}{2}\left(1 + \frac{\sin x}{10}\right) x$ over $[0, 3]$.

```
nearly.linear.out.of.sample = function(n) {
  # Combines simulating the true model with fitting
  # parametric model and smoother, calculating MSEs
  x=seq(from=0,to=3,length.out=n)
  y = h(x) + rnorm(n,0,0.15)
  data <- data.frame(x=x,y=y)
  y.new = h(x) + rnorm(n,0,0.15)
  sim.lm <- lm(y~x,data=data)
  lm.mse = mean(( fitted(sim.lm) - y.new )^2)
  sim.np.bw <- npregbw(y~x,data=data)
  sim.np <- npreg(sim.np.bw)
  np.mse = mean((sim.np$mean - y.new)^2)
  mses <- c(lm.mse,np.mse)
  return(mses)
}

nearly.linear.generalization = function(n,m=100) {
  raw = replicate(m,nearly.linear.out.of.sample(n))
  reduced = rowMeans(raw)
  return(reduced)
}
```

Code Example 21: Evaluating the out-of-sample error for the nearly-linear problem as a function of n, and evaluting the generalization error by averaging over many samples.




```
sizes = c(5,10,15,20,25,30,50,100,200,500)
generalizations = sapply(sizes,nearly.linear.generalization)
plot(sizes,sqrt(generalizations[1,]),ylim=c(0.12,0.22),type="l",
     xlab="n",ylab="RMS generalization error")
lines(sizes,sqrt(generalizations[2,]),lty=2)
abline(h=0.15,col="grey")
```

Figure 10.7: Root-mean-square generalization error for linear model (solid line) and kernel smoother (dashed line), fit to the same sample of the indicated size. The true regression curve is as in 10.6, and observations are corrupted by IID Gaussian noise with $\sigma = 0.15$ (grey horizontal line). The cross-over after which the nonparametric regressor has better generalization performance happens shortly before $n = 500$.