

## Basic Concepts

### *Bias-variance decomposition*

$$\begin{aligned}
E \left\{ [f(x) - \hat{f}(x)]^2 \right\} &= E[f^2(x) - 2f(x)\hat{f}(x) + \hat{f}^2(x)] \\
&= E[f^2(x)] - 2E[f(x)\hat{f}(x)] + E[\hat{f}^2(x)] \\
&= f^2(x) - 2f(x)E[\hat{f}(x)] + E[\hat{f}^2(x)] \\
&= f^2(x) - 2f(x)E[\hat{f}(x)] + E[\hat{f}^2(x)] \\
&= f^2(x) - 2f(x)E[\hat{f}(x)] + E[\hat{f}^2(x)] + \underbrace{(E[\hat{f}(x)])^2 - (E[\hat{f}(x)])^2}_{=0} \\
&= \left( f^2(x) - 2f(x)E[\hat{f}(x)] + (E[\hat{f}(x)])^2 \right) + \underbrace{(E[\hat{f}^2(x)] - (E[\hat{f}(x)])^2)}_{\text{Var}(Y)=E[Y^2]-(E[Y])^2} \\
&= (f(x) - E[\hat{f}(x)])^2 + \text{Var}(\hat{f}(x)).
\end{aligned}$$

(A)

Since the  $X_1, \dots, X_n$  are iid and have the same probability of “success” of landing in the interval equal to  $\pi_h$ , we can define the random variable  $W_i = I\left(-\frac{h}{2} < X_i < \frac{h}{2}\right)$  for all  $i$ , so that each  $W_i$  is a Bernoulli random variable with parameter  $\pi_h$ . We can then think of the distribution of  $Y = \sum_{i=1}^n W_i$  as being the sum of iid Bernoulli random variables. Hence,  $Y$  has a **binomial distribution, with mean  $n\pi_h$  and variance  $n\pi_h(1 - \pi_h)$ .**

(B)

Since  $\frac{Y}{n}$  is an estimate of the proportion of observations that fall in the interval  $\left(-\frac{h}{2}, \frac{h}{2}\right)$ , it is an estimate of  $\pi_h$ . And because  $\pi_h \approx hf(0)$ , we can let  $\frac{Y}{nh}$  be our estimator of  $f(0)$ , i.e.,  $\frac{Y}{nh} = \hat{f}(0)$ .

To find the MSE of this estimator, we note that

$$\begin{aligned}\text{Bias}\left(\hat{f}(0)\right) &= E\left[\hat{f}(0) - f(0)\right] \\&= E\left[\hat{f}(0)\right] - f(0) \\&= E\left[\frac{Y}{nh}\right] - f(0) \\&= \frac{E[Y]}{nh} - f(0) \\&= \frac{n\pi_h}{nh} - f(0) \\&= \frac{\pi_h}{h} - f(0).\end{aligned}$$

Also,

$$\begin{aligned}\text{Var}\left(\hat{f}(0)\right) &= \text{Var}\left(\frac{Y}{nh}\right) \\&= \frac{1}{n^2h^2}\text{Var}(Y) \\&= \frac{n\pi_h(1 - \pi_h)}{n^2h^2} \\&= \frac{\pi_h(1 - \pi_h)}{nh^2}.\end{aligned}$$

We can approximate  $\pi_h$  with a Taylor approximation of  $f(x)$ .

$$\begin{aligned}\pi_h &= \int_{-\frac{h}{2}}^{\frac{h}{2}} f(x) dx \\&\approx \int_{-\frac{h}{2}}^{\frac{h}{2}} \left[ f(0) + xf'(0) + \frac{x^2}{2}f''(0) \right] dx\end{aligned}$$

$$\begin{aligned}
&= \left[ xf(0) + \frac{x^2}{2} f'(0) + \frac{x^3}{6} f''(0) \right] \Big|_{-\frac{h}{2}}^{\frac{h}{2}} \\
&= hf(0) + \frac{h^3}{24} f''(0).
\end{aligned}$$

The MSE then becomes:

$$\text{MSE}\{\hat{f}(0), f(0)\}$$

$$\begin{aligned}
&= \left[ \text{Bias}(\hat{f}(0)) \right]^2 + \text{Var}(\hat{f}(0)) \\
&\approx \left( \frac{\pi_h}{h} - f(0) \right)^2 + \frac{\pi_h(1 - \pi_h)}{nh^2} \\
&= \left( \frac{h^2}{24} f''(0) \right)^2 + \frac{1}{nh^2} \left[ hf(0) + \frac{h^3}{24} f''(0) \right] \left[ 1 - hf(0) - \frac{h^3}{24} f''(0) \right] \\
&= \frac{h^4}{576} [f''(0)]^2 + \frac{1}{nh^2} \left[ hf(0) - h^2 f^2(0) - \frac{h^4}{12} f(0) f''(0) + \frac{h^3}{24} f''(0) - \frac{h^6}{576} [f''(0)]^2 \right] \\
&= \frac{h^4}{576} [f''(0)]^2 + \frac{1}{nh} f(0) - \frac{f^2(0)}{n} - \frac{h^2}{12n} f(0) f''(0) + \frac{h}{24n} f''(0) - \frac{h^4}{576n} [f''(0)]^2 \\
&\approx h^4 \underbrace{\left\{ \frac{[f''(0)]^2}{576} \left( 1 - \frac{1}{n} \right) \right\}}_A + \frac{1}{nh} \underbrace{f(0)}_B,
\end{aligned}$$

thereby showing that  $A = \frac{[f''(0)]^2}{576} \left( 1 - \frac{1}{n} \right)$ , and  $B = f(0)$ .

Looking at this last expression, we can see that for small  $h$ , the first term will become very small, while the second will become large. On the other hand, if  $h$  is large, the first term becomes very large, while the second goes to zero. If we think about the first term as a rough measure of the bias, and a second as a measure of the variance, we then see that a tradeoff occurs; we can't shrink one without causing the other one to grow dramatically.

(C)

We can write the above MSE as a function of  $h$  as follows:

$$\text{MSE}(h) = h^4 \left( \frac{[f''(0)]^2}{576} \left( 1 - \frac{1}{n} \right) \right) + \frac{1}{nh} f(0).$$

To optimize this function with respect to  $h$ , we differentiate wrt to  $h$  and set to zero.

$$\frac{d\text{MSE}}{dh} = 4h^3 \left( \frac{[f''(0)]^2}{576} \left( 1 - \frac{1}{n} \right) \right) - \frac{1}{nh^2} f(0) = 0.$$

Solving this last equation for  $h$  yields

$$h_{\text{opt}} = \left[ \frac{144f(0)}{[f''(0)]^2(n-1)} \right]^{\frac{1}{5}}.$$

## Curve fitting by linear smoothing

(A)

In Exercise 1 we showed that for a simple linear regression model  $y_i = f(x_i) = \beta_0 + \beta_1 x_i + \epsilon_i$  that

$$\hat{\beta}_1 = \frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^n (x_i - \bar{x})^2}.$$

Since we are taking  $\beta_0$  to be zero, and also taking  $\bar{x} = \bar{y} = 0$ , we have

$$\hat{\beta}_1 = \frac{\sum_{i=1}^n x_i y_i}{\sum_{i=1}^n x_i^2},$$

so that the prediction function becomes

$$\begin{aligned}\hat{f}(x^*) &= \hat{\beta}_1 x^* \\ &= \left( \frac{\sum_{i=1}^n x_i y_i}{\sum_{i=1}^n x_i^2} \right) x^* \\ &= \frac{(\sum_{i=1}^n x_i y_i) x^*}{\sum_{i=1}^n x_i^2} \\ &= \sum_{i=1}^n \frac{x^* x_i y_i}{\sum_{j=1}^n x_j^2} \\ &= \sum_{i=1}^n \left( \frac{x^* x_i}{s_{xx}} \right) y_i,\end{aligned}$$

where  $s_{xx} = \sum_{j=1}^n x_j^2$ . Thus, we can define

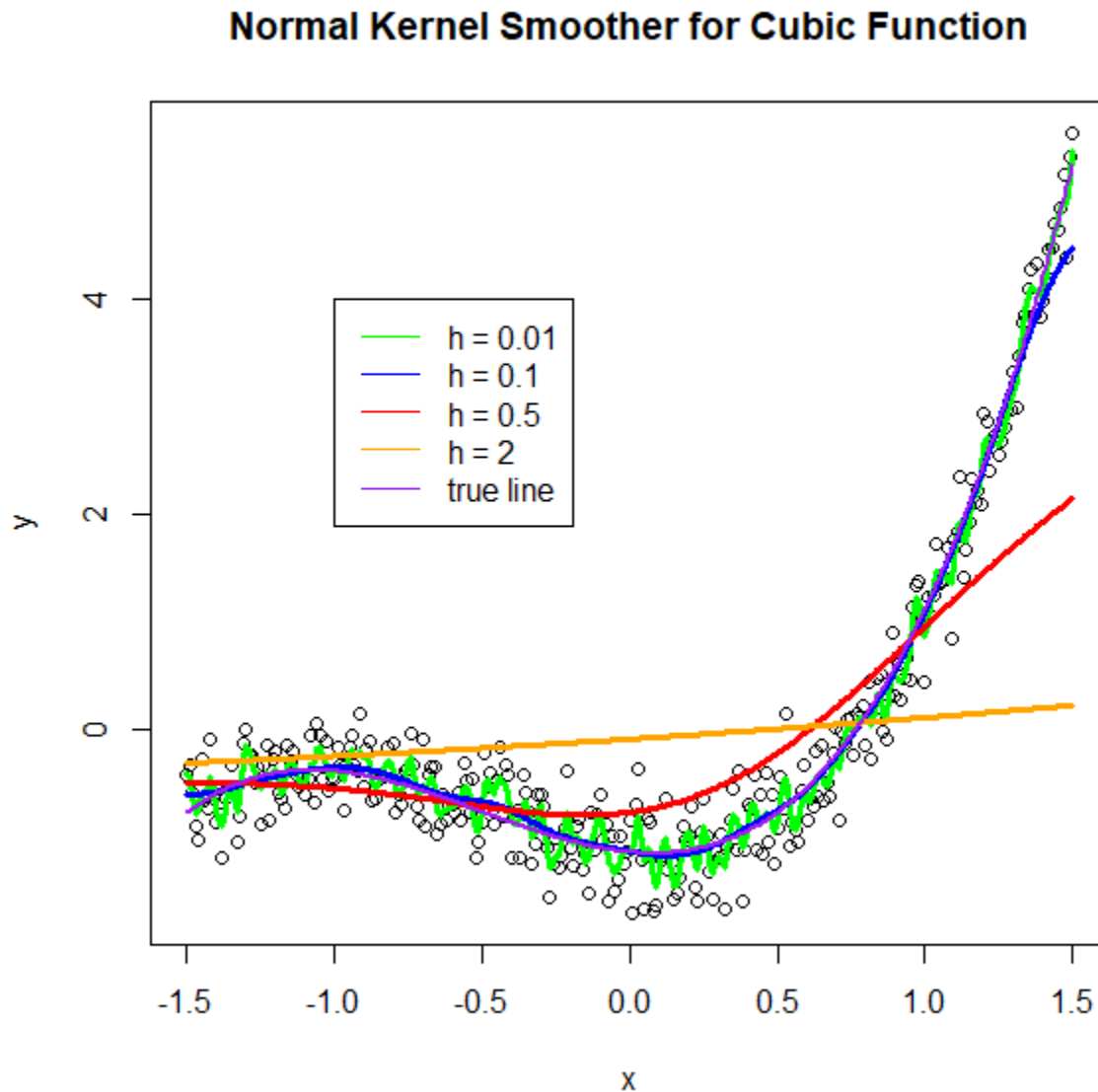
$$w(x_i, x^*) = \frac{x^* x_i}{s_{xx}},$$

so that  $\hat{f}(x^*) = \sum_{i=1}^n w(x_i, x^*) y_i$ .

This weight function seems to give weight to more data points that are closer to  $x^*$ . That is, when  $x^*$  is very close to  $x_i$ , then the weight on  $y_i$  will be larger. Note, however, that we could actually have negative weights if the sign of  $x^*$  and  $x_i$  are opposite. The meaning of “negative” weights is perhaps unclear. In the case of the  $K$ -nearest-neighbor-smoother, we don’t encounter these “negative” weights—either an observation has a positive weight or no weight at all. In this sense, the  $K$  smoother is perhaps more interpretable. On the other hand, note that the  $K$ -smoother will allow only  $K$  points to contribute weight to the predicted value; not all the points will be used. Thus, in a sense, the  $K$  smoother is intentionally omitting some possibly useful information. Also, the  $K$  smoother weights these  $K$  points equally; this could have potentially beneficial or negative consequences.

(B)

As we can see, there appears to be an optimal range for the smoothing parameter  $h$ . If  $h$  is too small, then the line becomes extremely wiggly, and the data are essentially overfit. On the other hand, if  $h$  is too large, then the smoothing function over-smooths, so that we miss important features in the data. In the extreme case, we end up with a line.

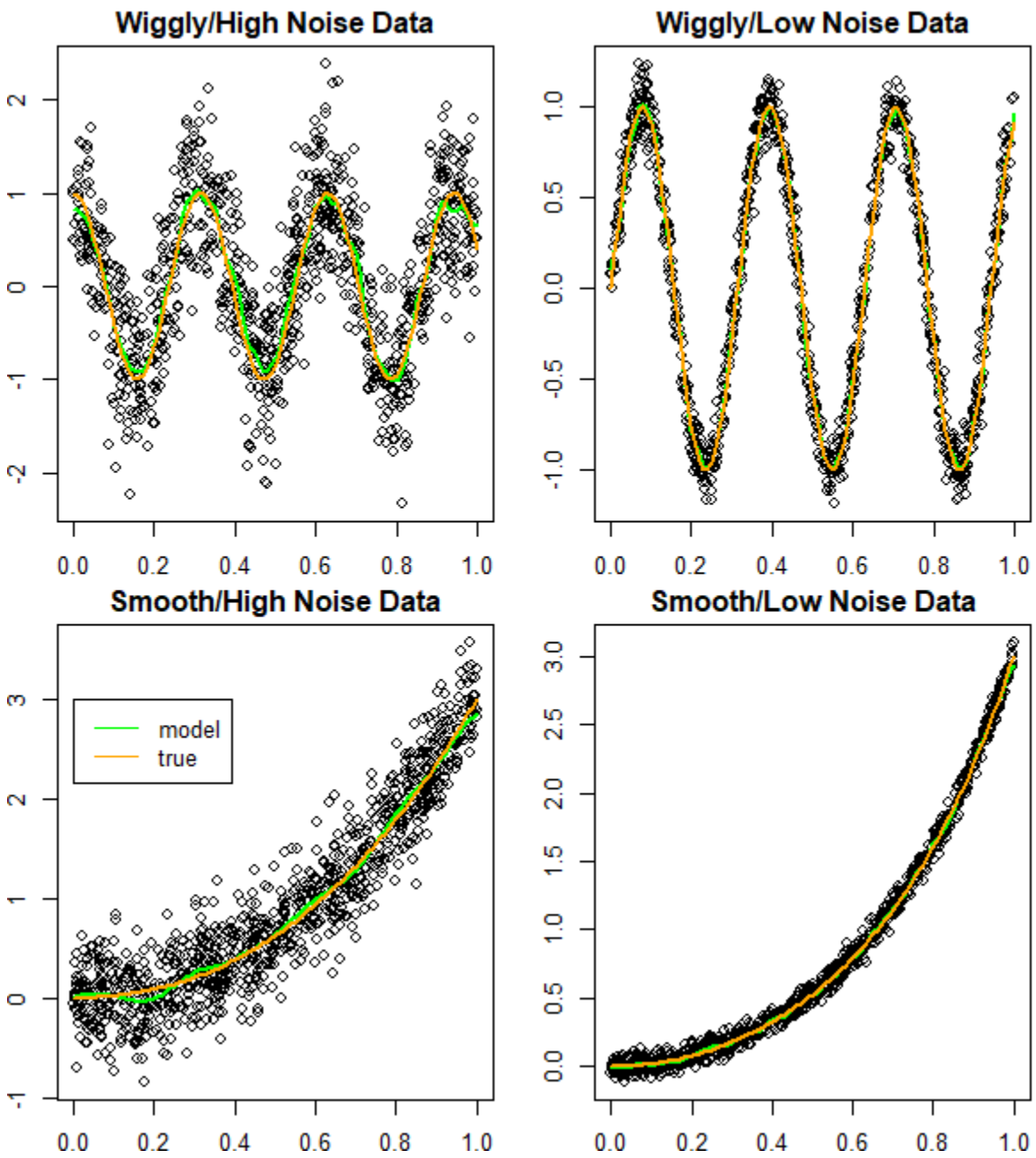


## Cross Validation

(A)

See R Code at end of document.

(B)



## Local Polynomial Regression

(A)

We can write the model for a single target point  $x$  as follows:

$$y_i = a_0 + a_1(x_i - x) + a_2(x_i - x)^2 + \dots + a_D(x_i - x)^D + \epsilon_i,$$

for  $i = 1, \dots, n$ , and where  $\epsilon_i$  is some zero-mean noise. Next, define

$$\mathbf{y} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}_{n \times 1}, \quad \mathbf{a} = \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_D \end{pmatrix}_{(D+1) \times 1}, \quad \boldsymbol{\epsilon} = \begin{pmatrix} \epsilon_1 \\ \epsilon_2 \\ \vdots \\ \epsilon_n \end{pmatrix}_{n \times 1}$$

$$\mathbf{R}_x = \begin{pmatrix} 1 & (x_1 - x) & (x_1 - x)^2 & \dots & (x_1 - x)^D \\ 1 & (x_2 - x) & (x_2 - x)^2 & \dots & (x_2 - x)^D \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & (x_n - x) & (x_n - x)^2 & \dots & (x_n - x)^D \end{pmatrix}_{n \times (D+1)},$$

so that the entire system can be written in matrix form:  $\mathbf{y} = \mathbf{R}_x \mathbf{a} + \boldsymbol{\epsilon}$ .

Furthermore, let

$$\mathbf{W} = \begin{pmatrix} w(x_1, x) & 0 & 0 & \dots & 0 \\ 0 & w(x_2, x) & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & w(x_n, x) \end{pmatrix}_{n \times n}.$$

Then we can minimize the following sum of squared errors with respect to  $\mathbf{a}$  as follows:

$$\begin{aligned} \text{WSS} &= \sum_{i=1}^n w_i [y_i - a_0 - \dots - a_D(x_i - x)^D]^2 \\ &= (\mathbf{y} - \mathbf{R}_x \mathbf{a})^T \mathbf{W} (\mathbf{y} - \mathbf{R}_x \mathbf{a}) \\ &= (\mathbf{y}^T - \mathbf{a}^T \mathbf{R}_x^T) \mathbf{W} (\mathbf{y} - \mathbf{R}_x \mathbf{a}) \\ &= \mathbf{y}^T \mathbf{W} \mathbf{y} - 2\mathbf{y}^T \mathbf{W} \mathbf{R}_x \mathbf{a} + \mathbf{a}^T \mathbf{R}_x^T \mathbf{W} \mathbf{R}_x \mathbf{a}, \end{aligned}$$

so that

$$\frac{\partial \text{WSS}}{\partial \mathbf{a}} = -2\mathbf{R}_x^T \mathbf{W} \mathbf{y} + 2\mathbf{a}^T \mathbf{R}_x^T \mathbf{W} \mathbf{R}_x = 0$$

Solving this last equation yields,

$$\hat{\mathbf{a}} = (\mathbf{R}_x^T \mathbf{W} \mathbf{R}_x)^{-1} \mathbf{R}_x^T \mathbf{W} \mathbf{y}.$$



(B)

First, define

$$s_j(x) = \sum_{i=1}^n K\left(\frac{x-x_i}{h}\right) (x_i - x)^j = \sum_{i=1}^n hw(x_i, x)(x_i - x)^j, \text{ for } j = 0, 1, 2,$$

and

$$w_i(x) = K\left(\frac{x-x_i}{h}\right) [s_2(x) - (x_i - x)s_1(x)] = hw(x_i, x)[s_2(x) - (x_i - x)s_1(x)].$$

From the previous problem  $\hat{\mathbf{a}} = (\mathbf{R}_x^T \mathbf{W} \mathbf{R}_x)^{-1} \mathbf{R}_x^T \mathbf{W} \mathbf{y}$  is the least squares estimator of  $\mathbf{a}$ . Suppose

$$\mathbf{R}_x = \begin{pmatrix} 1 & (x_1 - x) \\ 1 & (x_2 - x) \\ \vdots & \vdots \\ 1 & (x_n - x) \end{pmatrix},$$

so that

$$\begin{aligned} \hat{\mathbf{a}} &= (\mathbf{R}_x^T \mathbf{W} \mathbf{R}_x)^{-1} \mathbf{R}_x^T \mathbf{W} \mathbf{y} \\ &= \left[ \begin{pmatrix} 1 & (x_1 - x) \\ 1 & (x_2 - x) \\ \vdots & \vdots \\ 1 & (x_n - x) \end{pmatrix}^T \begin{pmatrix} w(x_1, x) & 0 & 0 & \cdots & 0 \\ 0 & w(x_2, x) & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & w(x_n, x) \end{pmatrix} \begin{pmatrix} 1 & (x_1 - x) \\ 1 & (x_2 - x) \\ \vdots & \vdots \\ 1 & (x_n - x) \end{pmatrix} \right]^{-1} \\ &\quad \times \begin{pmatrix} 1 & (x_1 - x) \\ 1 & (x_2 - x) \\ \vdots & \vdots \\ 1 & (x_n - x) \end{pmatrix}^T \begin{pmatrix} w(x_1, x) & 0 & 0 & \cdots & 0 \\ 0 & w(x_2, x) & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & w(x_n, x) \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix} \\ &= \left[ \begin{pmatrix} (x_1 - x)w(x_1, x) & (x_2 - x)w(x_2, x) & \cdots & (x_n - x)w(x_n, x) \\ w(x_1, x) & w(x_2, x) & \cdots & w(x_n, x) \end{pmatrix} \begin{pmatrix} 1 & (x_1 - x) \\ 1 & (x_2 - x) \\ \vdots & \vdots \\ 1 & (x_n - x) \end{pmatrix} \right]^{-1} \\ &\quad \times \begin{pmatrix} 1 & (x_1 - x) \\ 1 & (x_2 - x) \\ \vdots & \vdots \\ 1 & (x_n - x) \end{pmatrix}^T \begin{pmatrix} w(x_1, x) & 0 & 0 & \cdots & 0 \\ 0 & w(x_2, x) & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & w(x_n, x) \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix} \\ &= \begin{pmatrix} s_1(x) & s_2(x) \\ s_0(x) & s_1(x) \end{pmatrix}^{-1} \begin{pmatrix} 1 & (x_1 - x) \\ 1 & (x_2 - x) \\ \vdots & \vdots \\ 1 & (x_n - x) \end{pmatrix}^T \begin{pmatrix} w(x_1, x) & 0 & 0 & \cdots & 0 \\ 0 & w(x_2, x) & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & w(x_n, x) \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix} \end{aligned}$$

$$\begin{aligned}
&= \frac{1}{s_1^2(x) - s_2(x)s_0(x)} \begin{pmatrix} s_1(x) & -s_2(x) \\ -s_0(x) & s_1(x) \end{pmatrix} \begin{pmatrix} 1 & (x_1 - x) \\ 1 & (x_2 - x) \\ \vdots & \vdots \\ 1 & (x_n - x) \end{pmatrix}^T \begin{pmatrix} w(x_1, x) & 0 & 0 & \cdots & 0 \\ 0 & w(x_2, x) & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & w(x_n, x) \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix} \\
&= \frac{1}{s_1^2(x) - s_2(x)s_0(x)} \begin{pmatrix} s_1(x) & -s_2(x) \\ -s_0(x) & s_1(x) \end{pmatrix} \\
&\quad \times \begin{pmatrix} (x_1 - x)w(x_1, x) & (x_2 - x)w(x_2, x) & \cdots & (x_n - x)w(x_n, x) \\ w(x_1, x) & w(x_2, x) & \cdots & w(x_n, x) \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix} \\
&= \frac{1}{s_1^2(x) - s_2(x)s_0(x)} \begin{pmatrix} s_1(x) & -s_2(x) \\ -s_0(x) & s_1(x) \end{pmatrix} \begin{pmatrix} \sum y_i w(x_i, x)(x_i - x) \\ \sum y_i w(x_i, x) \end{pmatrix} \\
&= \frac{1}{s_1^2(x) - s_2(x)s_0(x)} \begin{pmatrix} \sum [y_i s_1(x) w(x_i, x)(x_i - x) - y_i s_2(x) w(x_i, x)] \\ \sum [y_i s_1(x) w(x_i, x) - y_i w(x_i, x)(x_i - x)s_0(x)] \end{pmatrix} \\
&= \begin{pmatrix} \hat{a}_0 \\ \hat{a}_1 \end{pmatrix}.
\end{aligned}$$

But because  $\hat{f}(x) = \hat{a}_0 + \hat{a}_1(x - x) = \hat{a}_0$ , we have

$$\begin{aligned}
\hat{f}(x) &= \hat{a}_0 \\
&= \frac{\sum [y_i s_1(x) w(x_i, x)(x_i - x) - y_i s_2(x) w(x_i, x)]}{s_1^2(x) - s_2(x)s_0(x)} \\
&= \frac{\sum (y_i h w(x_i, x) [s_1(x)(x_i - x) - s_2(x)])}{[s_1(x) \sum h w(x_i, x)(x_i - x) - s_2(x) \sum h w(x_i, x)]} \\
&= \frac{\sum \left( y_i K \left( \frac{x_i - x}{h} \right) [s_1(x)(x_i - x) - s_2(x)] \right)}{\sum (h w(x_i, x) [s_1(x)(x_i - x) - s_2(x)])} \\
&= \frac{\sum \left( y_i K \left( \frac{x_i - x}{h} \right) [s_2(x) - s_1(x)(x_i - x)] \right)}{\sum \left( K \left( \frac{x_i - x}{h} \right) [s_2(x) - s_1(x)(x_i - x)] \right)} \\
&= \frac{\sum w_i(x) y_i}{\sum w_i(x)}.
\end{aligned}$$

(C)

Recall that the first component  $\hat{\alpha}_0$  of the vector  $\hat{\boldsymbol{\alpha}}$  represents  $\hat{f}(x)$ . Thus, if we let  $\mathbf{t}^T = (1, 0)_{1 \times 2}$ , then  $\hat{f}(x) = \underbrace{\mathbf{t}^T (\mathbf{R}_x^T \mathbf{W} \mathbf{R}_x)^{-1} \mathbf{R}_x^T \mathbf{W}}_{\mathbf{h}_x^T} \mathbf{y} = \mathbf{h}_x^T \mathbf{y}$ . Also, we note that

$$\begin{aligned}\boldsymbol{\mu} &= E[\mathbf{y}] \\ &= E \left[ \begin{pmatrix} f(x_1) + \epsilon_1 \\ f(x_2) + \epsilon_2 \\ \vdots \\ f(x_n) + \epsilon_n \end{pmatrix} \right] \\ &= \begin{pmatrix} E[f(x_1)] + E[\epsilon_1] \\ E[f(x_2)] + E[\epsilon_2] \\ \vdots \\ E[f(x_n)] + E[\epsilon_n] \end{pmatrix} \\ &= \begin{pmatrix} f(x_1) \\ f(x_2) \\ \vdots \\ f(x_n) \end{pmatrix},\end{aligned}$$

and, assuming that the  $\epsilon_i$  are independent and have equal variance  $\sigma^2$ ,

$$\begin{aligned}\text{Cov}(\mathbf{y})_{(ij)} &= \text{Cov}(f(x_i) + \epsilon_i, f(x_j) + \epsilon_j) \\ &= \text{Cov}(\epsilon_i, \epsilon_j) \\ &= \begin{cases} \sigma^2 & \text{if } i = j \\ 0, & \text{otherwise,} \end{cases}\end{aligned}$$

where  $\text{Cov}(\mathbf{y})_{(ij)}$  is the  $ij^{\text{th}}$  element of  $\text{Cov}(\mathbf{y})$ , so that

$$\text{Cov}(\mathbf{y}) = \sigma^2 \mathbf{I}.$$

Hence,

$$\begin{aligned}E[\hat{f}(x)] &= E[\mathbf{t}^T (\mathbf{R}_x^T \mathbf{W} \mathbf{R}_x)^{-1} \mathbf{R}_x^T \mathbf{W} \mathbf{y}] \\ &= \mathbf{t}^T (\mathbf{R}_x^T \mathbf{W} \mathbf{R}_x)^{-1} \mathbf{R}_x^T \mathbf{W} E[\mathbf{y}], \\ &= \mathbf{h}_x^T \boldsymbol{\mu} = \sum_{i=1}^n h_i f(x_i),\end{aligned}$$

where  $h_i$  is the  $i^{\text{th}}$  element of  $\mathbf{h}$ .

and

$$\begin{aligned}
 \text{Cov}\left(\hat{f}(x)\right) &= \text{Cov}[\mathbf{t}^T (\mathbf{R}_x^T \mathbf{W} \mathbf{R}_x)^{-1} \mathbf{R}_x^T \mathbf{W} \mathbf{y}] \\
 &= [\mathbf{t}^T (\mathbf{R}_x^T \mathbf{W} \mathbf{R}_x)^{-1} \mathbf{R}_x^T \mathbf{W}] (\text{Cov}(\mathbf{y})) [\mathbf{t}^T (\mathbf{R}_x^T \mathbf{W} \mathbf{R}_x)^{-1} \mathbf{R}_x^T \mathbf{W}]^T \\
 &= \mathbf{h}_x^T (\text{Cov}(\mathbf{y})) \mathbf{h}_x \\
 &= \mathbf{h}_x^T (\sigma^2 \mathbf{I}) \mathbf{h}_x \\
 &= \sigma^2 \mathbf{h}_x^T \mathbf{h}_x = \sigma^2 \sum_{i=1}^n h_i^2
 \end{aligned}$$

(D)

We define

$$\mathbf{H} = \begin{pmatrix} \mathbf{h}_{x_1}^T \\ \mathbf{h}_{x_2}^T \\ \vdots \\ \mathbf{h}_{x_n}^T \end{pmatrix},$$

where the  $\mathbf{h}_{x_i}^T$  are defined as in part (C).

Now, note that

$$\begin{aligned} E[\mathbf{r}] &= E[\mathbf{y} - \mathbf{H}\mathbf{y}] \\ &= E[(\mathbf{I} - \mathbf{H})\mathbf{y}] \\ &= (\mathbf{I} - \mathbf{H})E[\mathbf{y}] \\ &= (\mathbf{I} - \mathbf{H})\boldsymbol{\mu}, \end{aligned}$$

where  $\boldsymbol{\mu}$  is defined as before. Also,

$$\begin{aligned} \text{Cov}(\mathbf{r}) &= \text{Cov}(\mathbf{y} - \mathbf{H}\mathbf{y}) \\ &= \text{Cov}((\mathbf{I} - \mathbf{H})\mathbf{y}) \\ &= (\mathbf{I} - \mathbf{H})\text{Cov}(\mathbf{y})(\mathbf{I} - \mathbf{H})^T \\ &= \sigma^2(\mathbf{I} - \mathbf{H})(\mathbf{I} - \mathbf{H}^T) \\ &= \sigma^2(\mathbf{I} - \mathbf{H} - \mathbf{H}^T + \mathbf{H}\mathbf{H}^T). \end{aligned}$$

Thus,

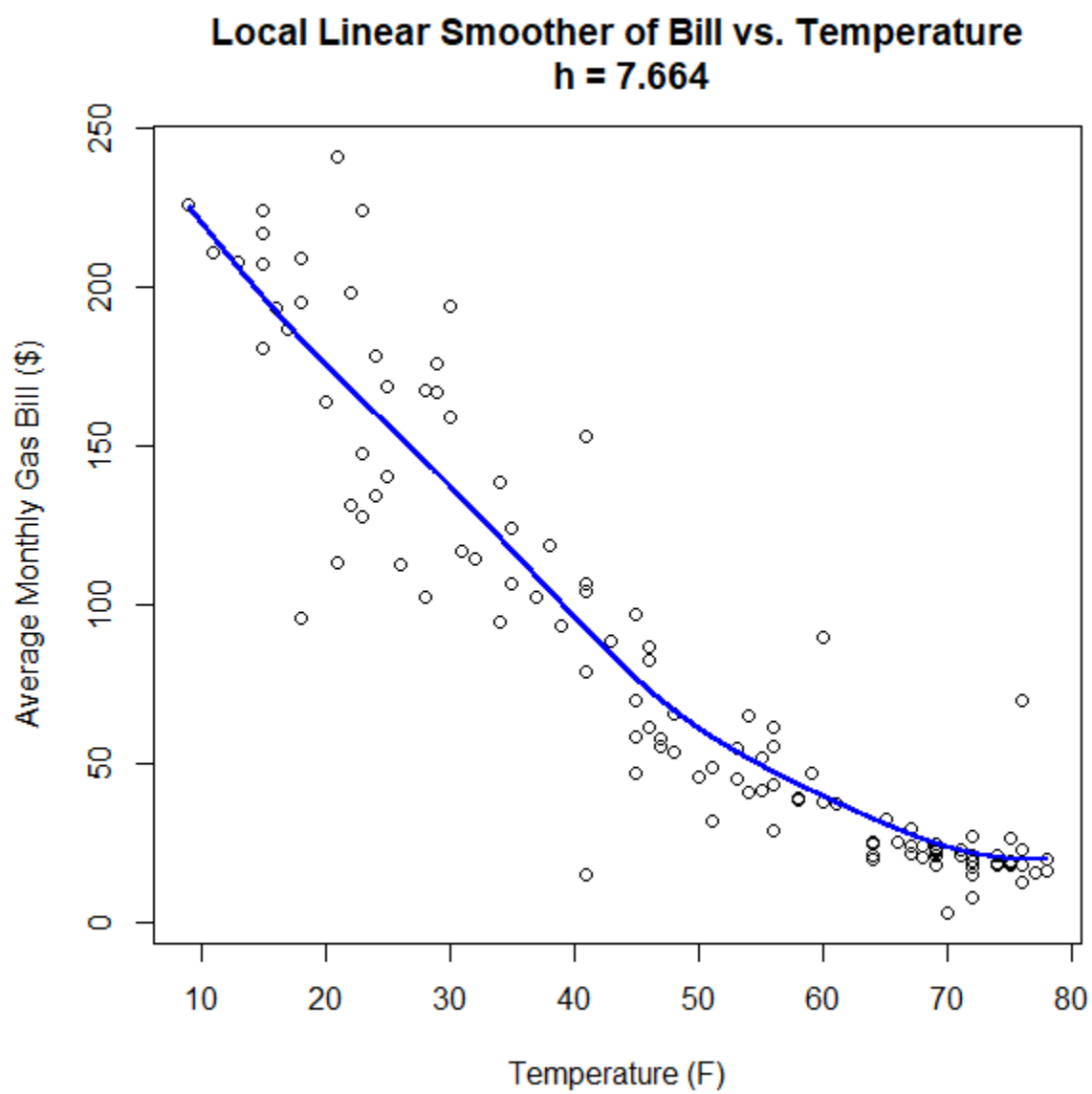
$$\begin{aligned} E[\|\mathbf{r}\|] &= E[(\mathbf{y} - \mathbf{H}\mathbf{y})^T(\mathbf{y} - \mathbf{H}\mathbf{y})] \\ &= \text{tr}(\sigma^2(\mathbf{I} - \mathbf{H} - \mathbf{H}^T + \mathbf{H}\mathbf{H}^T)) + (\mathbf{I} - \mathbf{H})^T\boldsymbol{\mu} \\ &= \sigma^2\text{tr}(\mathbf{I} - \mathbf{H} - \mathbf{H}^T + \mathbf{H}\mathbf{H}^T) + \boldsymbol{\mu} - \mathbf{H}^T\boldsymbol{\mu} \\ &= \sigma^2[n - \text{tr}(\mathbf{H}) - \text{tr}(\mathbf{H}^T) + \text{tr}(\mathbf{H}\mathbf{H}^T)] + \boldsymbol{\mu} - \mathbf{H}^T\boldsymbol{\mu} \\ &= \sigma^2[n - \text{tr}(\mathbf{H}) - \text{tr}(\mathbf{H}) + \text{tr}(\mathbf{H}^T\mathbf{H})] + \boldsymbol{\mu} - \mathbf{H}^T\boldsymbol{\mu} \\ &= \sigma^2[n - 2\text{tr}(\mathbf{H}) + \text{tr}(\mathbf{H}^T\mathbf{H})] + \boldsymbol{\mu} - \mathbf{H}^T\boldsymbol{\mu}, \end{aligned}$$

so that

$$\begin{aligned}
E[\hat{\sigma}^2] &= E\left[\frac{\|\mathbf{r}\|}{n - 2\text{tr}(\mathbf{H}) + \text{tr}(\mathbf{H}^T \mathbf{H})}\right] \\
&= \frac{\sigma^2[n - 2\text{tr}(\mathbf{H}) + \text{tr}(\mathbf{H}^T \mathbf{H})] + \boldsymbol{\mu} - \mathbf{H}^T \boldsymbol{\mu}}{n - 2\text{tr}(\mathbf{H}) + \text{tr}(\mathbf{H}^T \mathbf{H})} \\
&= \sigma^2 + \frac{\boldsymbol{\mu} - \mathbf{H}^T \boldsymbol{\mu}}{n - 2\text{tr}(\mathbf{H}) + \text{tr}(\mathbf{H}^T \mathbf{H})}.
\end{aligned}$$

We see that as  $n$  increases, the bias of  $\hat{\sigma}^2$  tends toward zero. Moreover,  $\hat{\sigma}^2$  would be unbiased if  $\mathbf{H} = \mathbf{I}$ , but this seems unlikely, if not impossible, in local polynomial regression.

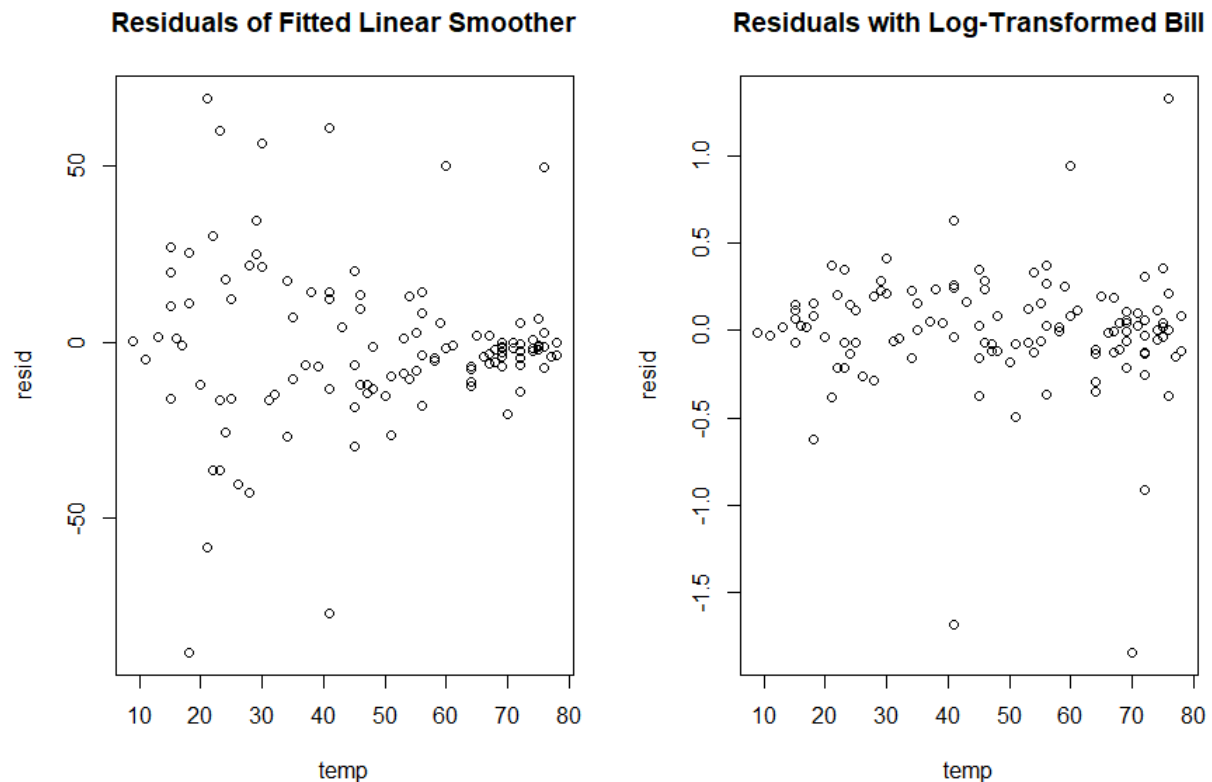
(E)



(F)

We see from the left graph below that there appears to be some heteroskedasticity in the original data. The residuals make a sort of fan shape as we move from left to right, indicating that the variance in the data decreases as the temperature increases.

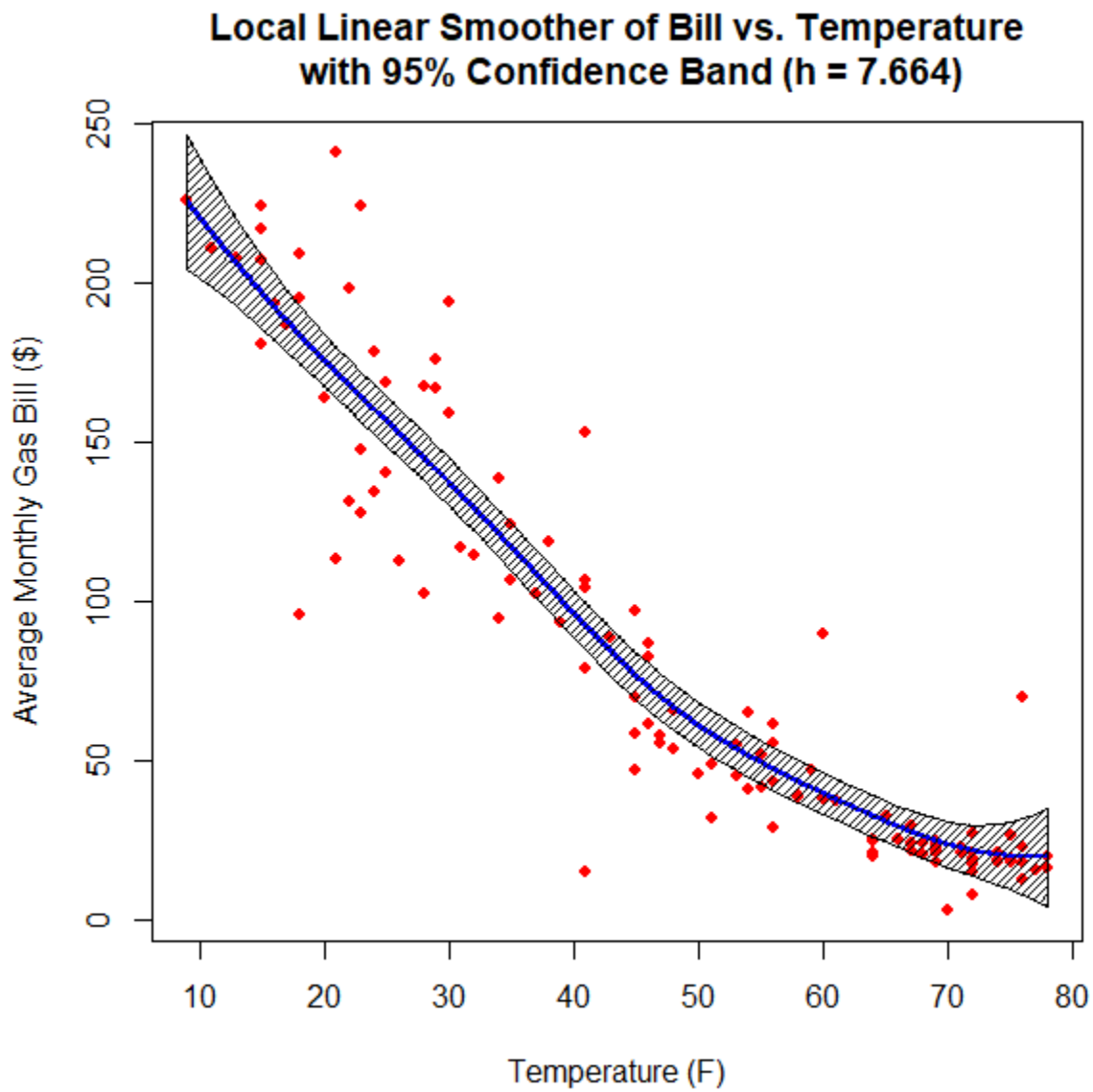
On the right graph below, we show the residual plot after using the linear smoothing when the response variable has first been log-transformed. We see that this somewhat improves the heteroskedasticity of the model. We might consider using this model instead.





(G)

We construct confidence bands for a particular  $f(x)$  in the form  $\hat{f}(x) \pm c \cdot \hat{\sigma}$ , where  $\hat{\sigma}$  is defined as previously, and where  $c$  is the upper 0.025 quantile of a standard normal distribution; i.e.,  $c \approx 2$ .



## R Code

```
# SDS 383D: Statistical Modeling II
# Exercise 3
# Curve Fitting by Linear Smoothing

# (B)

# Function with normal smoother

# Input:
# @x: vector of obs x values
# @y: vector of obs responses
# @h: bandwidth for normal kernel
# @x_star: target x vector

# Output:
# @y_star: predicted value

smooth_kern <- function(x, y, h, x_star, kern = dnorm){
  w <- 1/h*kern((x-x_star)/h)
  w_norm <- w/sum(w)
  y_star <- sum(w_norm*y)
  return(y_star)
}

# Generate random data
x <- seq(0, 3, by = 0.01)
y <- x^3 - 3*x^2 + 2*x + rnorm(length(x), 0, .3)

# Mean center data
x <- x - mean(x)
y <- y - mean(y)

# Plot data
plot(x, y, main = "Normal Kernel Smoother for Cubic Function")

# Plot predicted values

# h = 0.01
z <- seq(min(x), max(x), by = 0.01)
z_star <- rep(0, length(z))
for(i in 1:length(z)){
  z_star[i] <- smooth_kern(x, y, .01, z[i])
}

lines(z, z_star, col = "green", lwd = 3)

# h = 0.1
z <- seq(min(x), max(x), by = 0.01)
z_star <- rep(0, length(z))
for(i in 1:length(z)){
  z_star[i] <- smooth_kern(x, y, .1, z[i])
}

lines(z, z_star, col = "blue", lwd = 3)

# h = 0.5
z <- seq(min(x), max(x), by = 0.01)
z_star <- rep(0, length(z))
for(i in 1:length(z)){
  z_star[i] <- smooth_kern(x, y, .5, z[i])
}

lines(z, z_star, col = "red", lwd = 3)

# h = 2
z <- seq(min(x), max(x), by = 0.01)
z_star <- rep(0, length(z))
for(i in 1:length(z)){
  z_star[i] <- smooth_kern(x, y, 2, z[i])
}
```

```

}
lines(z, z_star, col = "orange", lwd = 3)

# True function
tx <- seq(0, 3, by = 0.01)
ty <- tx^3 - 3*tx^2 + 2*tx

tx <- tx - mean(tx)
ty <- ty - mean(ty)
lines(tx, ty, col = "purple", lwd = 2)

# Legend
legend(-1, 4, c("h = 0.01", "h = 0.1", "h = 0.5", "h = 2", "true line"),
      col = c("green", "blue", "red", "orange", "purple"),
      lty = 1)

```

```

# SDS 383D: Statistical Modeling II
# Exercise 3
# Cross Validation

# (A)

# Function smooth() will compute kernel
# density estimates of data with CV

# Input:
  #@ train: training data,
  #@ test: testing data
  #@ h: bandwidth

# Output:
  #@ result: matrix containing
  #          test predictions and errors

smooth <- function(train, test, h){
  # Vectors to hold training
  x_train <- train[, 1]
  y_train <- train[, 2]

  # Vectors to hold test
  x_test <- test[, 1]
  y_test <- test[, 2]

  # Vectors to hold pred and error
  test_pred <- matrix(NA, nrow = length(h),
                     ncol = length(x_test))

  test_error <- matrix(NA, nrow = length(h),
                     ncol = length(x_test))

  # For each h, compute prediction and error
  # for every test data point, using a model
  # fit by training data
  for(i in 1:length(h)){
    for(j in 1:length(x_test)){
      # Compute weights
      # N(0, 1) kernel
      w <- 1/h[i]*dnorm((x_test[j] - x_train)/h[i])

      # Normalize weights
      w <- w/sum(w)

      # Compute prediction and error
      test_pred[i, j] <- sum(w * y_train)
      test_error[i, j] <- y_test[j] - test_pred[i, j]
    }
  }

  # Object that contains two matrices;
  # (1) Matrix of predictions for each h
  # (2) Matrix of errors for each h
  result <- list("Test Prediction" = test_pred,
                "Test Error" = test_error)

  # Return object
  return(result)
}

graph_pred <- function(x1, x2, y, h){
  pred <- rep(NA, length(x1))

  for(i in 1:length(x1)){
    w <- 1/h*dnorm((x1[i] - x2)/h)
    w <- w/sum(w)
  }
}

```

```

        }
        pred[i] <- sum(w*y)
    }
    lines(x1, pred, col = "green",
          lwd = 2)
}

# (B)

# Sample size
N <- 1000

# Create different types of data

# wiggly/high noise
x_wig_high <- runif(N, 0, 1)
y_wig_high <- cos(x_wig_high*20) + rnorm(N, 0, 0.5)
plot(x_wig_high, y_wig_high)

# wiggly/low noise
x_wig_low <- runif(N, 0, 1)
y_wig_low <- sin(x_wig_low*20) + rnorm(N, 0, 0.1)
plot(x_wig_low, y_wig_low)

# Smooth/high noise
x_sm_high <- runif(N, 0, 1)
y_sm_high <- x_sm_high^3 + 2*x_sm_high^2 + rnorm(N, 0, 0.3)
plot(x_sm_high, y_sm_high)

# Smooth/low noise
x_sm_low <- runif(N, 0, 1)
y_sm_low <- x_sm_low^5 + 2*x_sm_low^2 + rnorm(N, 0, 0.05)
plot(x_sm_low, y_sm_low)

# Partition data into train and test
i1 <- sample(1:N, 2*N/3)
wig_high_train <- cbind(x_wig_high[i1], y_wig_high[i1])
wig_high_test <- cbind(x_wig_high[-i1], y_wig_high[-i1])

i2 <- sample(1:N, 2*N/3)
wig_low_train <- cbind(x_wig_low[i2], y_wig_low[i2])
wig_low_test <- cbind(x_wig_low[-i2], y_wig_low[-i2])

i3 <- sample(1:N, 2*N/3)
sm_high_train <- cbind(x_sm_high[i3], y_sm_high[i3])
sm_high_test <- cbind(x_sm_high[-i3], y_sm_high[-i3])

i4 <- sample(1:N, 2*N/3)
sm_low_train <- cbind(x_sm_low[i4], y_sm_low[i4])
sm_low_test <- cbind(x_sm_low[-i4], y_sm_low[-i4])

# Call smooth function
h <- seq(0.001, 1, by = 0.005)
wig_high <- smooth(wig_high_train, wig_high_test, h)
wig_low <- smooth(wig_low_train, wig_low_test, h)
sm_high <- smooth(sm_high_train, sm_low_test, h)
sm_low <- smooth(sm_low_train, sm_low_test, h)

# Extract optimal h
h_wig_high_opt <- h[which.min(apply(wig_high[[2]], 1,
                                   function(x){sum(x^2)}))]
h_wig_low_opt <- h[which.min(apply(wig_low[[2]], 1,
                                   function(x){sum(x^2)}))]
h_sm_high_opt <- h[which.min(apply(sm_high[[2]], 1,
                                   function(x){sum(x^2)}))]
h_sm_low_opt <- h[which.min(apply(sm_low[[2]], 1,
                                   function(x){sum(x^2)}))]

# Plot all data and kernel density estimate,
# using optimal h chosen by test data
par(mfrow = c(2, 2))

```

```

par(mai = c(0.3, 0.3, 0.3, 0.3))
t <- seq(0, 1, by = 0.01)
plot(x_wig_high, y_wig_high, xlab = "", ylab = "")
graph_pred(t, x_wig_high, y_wig_high, h_wig_high_opt)
lines(t, cos(t*20), col = "orange", lwd = 2)
title(main = "Wiggly/High Noise Data", xlab = "x", ylab = "y")

plot(x_wig_low, y_wig_low, xlab = "", ylab = "")
graph_pred(t, x_wig_low, y_wig_low, h_wig_low_opt)
lines(t, sin(t*20), col = "orange", lwd = 2)
title(main = "Wiggly/Low Noise Data", xlab = "x", ylab = "y")

plot(x_sm_high, y_sm_high, xlab = "", ylab = "")
graph_pred(t, x_sm_high, y_sm_high, h_sm_high_opt)
lines(t, t^3 + 2*t^2, col = "orange", lwd = 2)
title(main = "Smooth/High Noise Data", xlab = "x", ylab = "y")
legend(0, 3, c("model", "true"), col = c("green", "orange"), lty = 1)

plot(x_sm_low, y_sm_low, xlab = "", ylab = "")
graph_pred(t, x_sm_low, y_sm_low, h_sm_low_opt)
lines(t, t^5 + 2*t^2, col = "orange", lwd = 2)
title(main = "Smooth/Low Noise Data", xlab = "x", ylab = "y")

```

```

# SDS 383D: Statistical Modeling II
# Exercise 3

# Local linear smoothing

# Load utilities.csv data
data <- read.csv("C:\\Users\\tr1il\\Google Drive\\UT Austin\\SDS 383D Statistical
Modeling II\\Homework\\Exercises 3\\utilities.csv", head = T)

# Create vectors of data

# Predictor (independent) variable
temp <- data$temp

# Response (dependent) variable
bill <- data$gasbill

# Sample size
n <- length(temp)

# Define smooth function
# Input:
# @: xi: x values for weighted regression
# @: yi: y values for weighted regression
# @: x: new x to predict
# @: h: bandwidth

# Output:
# @: f_hat: estimate of function at new x

smooth <- function(xi, yi, x, h){
  # Formulas from derivations
  s1 <- sum(dnorm((x-xi)/h)*(xi-x))
  s2 <- sum(dnorm((x-xi)/h)*(xi-x)^2)
  w <- dnorm((x-xi)/h)*(s2 - (xi - x)*s1)
  f_hat <- sum(w*yi)/sum(w)
  return(f_hat)
}

# Define hat function
# Function returns smoothing vector for specified x
# Input
# @ x: scalar input
# Output
# @ hx: hat (smoothing) vector

hat <- function(x){
  s <- rep(1, 2)
  W <- diag(1/h_opt*dnorm((x-temp)/h_opt))
  w <- W/sum(W)
  Rx <- cbind(1, temp-x)
  hx <- t(s) %*% solve(t(Rx) %*% W %*% Rx) %*% t(Rx) %*% W
  return(hx)
}

# Leave-one-out cross validation
# Vector of bandwidths to test
h <- seq(7, 8, by = 0.001)

# Vector to hold error of each prediction
err <- rep(NA, n)

# Vector to hold sum of squared errors
sse <- rep(NA, length(h))

# Perform LOOCV
# For each h,
for(i in 1:length(h)){
  # Remove one data point
  # Fit model on remaining data

```

```

    for(j in 1:n){
      train_x <- temp[-j]
      train_y <- bill[-j]

      # Compute error of training model
      err[j] <- smooth(train_x, train_y, temp[j], h[i]) - bill[j]
    }

    # Sum of squared errors
    sse[i] <- sum(err^2)
  }

# Determine optimal bandwidth
h_opt <- h[which.min(sse)]

# Fit model with optimal bandwidth
x_new <- seq(min(temp), max(temp), by = 0.01)
pred <- rep(NA, length(x_new))

for(i in 1:length(x_new)){
  pred[i] <- smooth(temp, bill, x_new[i], h_opt)
}

# Plot data and fitted model
plot(temp, bill, main = "Local Linear Smoother of Bill vs. Temperature",
      h = 7.664,
      xlab = "Temperature (F)",
      ylab = "Average Monthly Gas Bill ($)")
lines(x_new, pred, col = "blue", lwd = 2)

# Plot residuals
resid <- rep(NA, n)

for(i in 1:length(temp)){
  resid[i] <- bill[i] - smooth(temp, bill, temp[i], h_opt)
}

plot(temp, resid, main = "Residuals of Fitted Linear Smoother")

# Try log-transforming response
log_bill <- log(bill)

# Plot transformed resid
for(i in 1:length(temp)){
  resid[i] <- log_bill[i] - smooth(temp, log_bill, temp[i], h_opt)
}

plot(temp, resid, main = "Residuals with Log-Transformed Bill")

# Plot confidence bands
n <- length(temp)

# Create H
# It is concatenation of hat vectors
# for all observed x values
H <- NULL
for(i in 1:n){
  H <- rbind(H, hat(temp[i]))
}

# Estimate variance

# Compute residuals
resid <- rep(NA, n)

for(i in 1:length(temp)){
  resid[i] <- bill[i] - smooth(temp, bill, temp[i], h_opt)
}

# Using variance estimator from exercise
sigma2_hat <- sum(resid^2)/(n - 2*sum(diag(H)) + sum(diag(t(H) %*% H)))

```



```

# Some x values repeat
# Estimate variance only once at each
# uniquely observed x value
x_band <- unique(sort(temp))

# Vector to hold estimated st. err of prediction
se_x <- rep(NA, length(x_band))

# For each observed x value
# estimate standard error
for(i in 1:length(x_band)){
  se_x[i] <- sqrt(sigma2_hat * sum(hat(x_band[i])^2))
}

# Upper and lower 95% conf bands
lower <- rep(NA, length(x_band))
upper <- rep(NA, length(x_band))

# For each observed x value, compute
# the upper and lower 95% limits
for(i in 1:length(x_band)){
  lower[i] <- smooth(temp, bill, x_band[i], h_opt) - 2*se_x[i]
  upper[i] <- smooth(temp, bill, x_band[i], h_opt) + 2*se_x[i]
}

# Plot data and fitted model
plot(temp, bill, main = "Local Linear Smoother of Bill vs. Temperature
                        with 95% Confidence Band (h = 7.664)",
      xlab = "Temperature (F)",
      ylab = "Average Monthly Gas Bill ($)",
      pch = 20,
      cex = 1.2,
      col = "red")
lines(x_new, pred, col = "blue", lwd = 2)

# Plot 95% confidence bands
polygon(c(x_band, rev(x_band)), c(lower, rev(upper)), density = 30)

```