Travis Lilley
SDS 383D: Statistical Modeling II
Exercises 3

Basic Concepts

Bias-variance decomposition

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

(A)

Since the $X_1, \ldots, X_n$ are iid and have the same probability of "success" of landing in the interval equal to π_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 π_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 π_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 π_h with a Taylor approximation of $f(x)$.

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

$$= \left[xf(0) + \frac{x^2}{2} f'(0) + \frac{x^3}{6} f''(0) \right] \Big|_{-\frac{h}{2}}^{\frac{h}{2}}$$

$$= hf(0) + \frac{h^3}{24} f''(0).$$

The MSE then becomes:

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

$$= \left[\text{Bias}\left(\hat{f}(0)\right) \right]^2 + \text{Var}\left(\hat{f}(0)\right)$$

$$\approx \left(\frac{\pi_h}{h} - f(0) \right)^2 + \frac{\pi_h(1-\pi_h)}{nh^2}$$

$$= \left(\frac{h^2}{24} f''(0) \right)^2 + \frac{1}{nh^2} \left[hf(0) + \frac{h^3}{24} f''(0) \right] \left[1 - hf(0) - \frac{h^3}{24} f''(0) \right]$$

$$= \frac{h^4}{576} [f''(0)]^2 + \frac{1}{nh^2} \left[hf(0) - h^2 f^2(0) - \frac{h^4}{12} f(0) f''(0) + \frac{h^3}{24} f''(0) - \frac{h^6}{576} [f''(0)]^2 \right]$$

$$= \frac{h^4}{576} [f''(0)]^2 + \frac{1}{nh} f(0) - \frac{f^2(0)}{n} - \frac{h^2}{12n} f(0) f''(0) + \frac{h}{24n} f''(0) - \frac{h^4}{576n} [f''(0)]^2$$

$$\approx h^4 \underbrace{\left\{ \frac{[f''(0)]^2}{576} \left(1 - \frac{1}{n} \right) \right\}}_{A} + \frac{1}{nh} \underbrace{f(0)}_{B},$$

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:

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

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

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

Solving this last equation for h yields

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

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 β_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

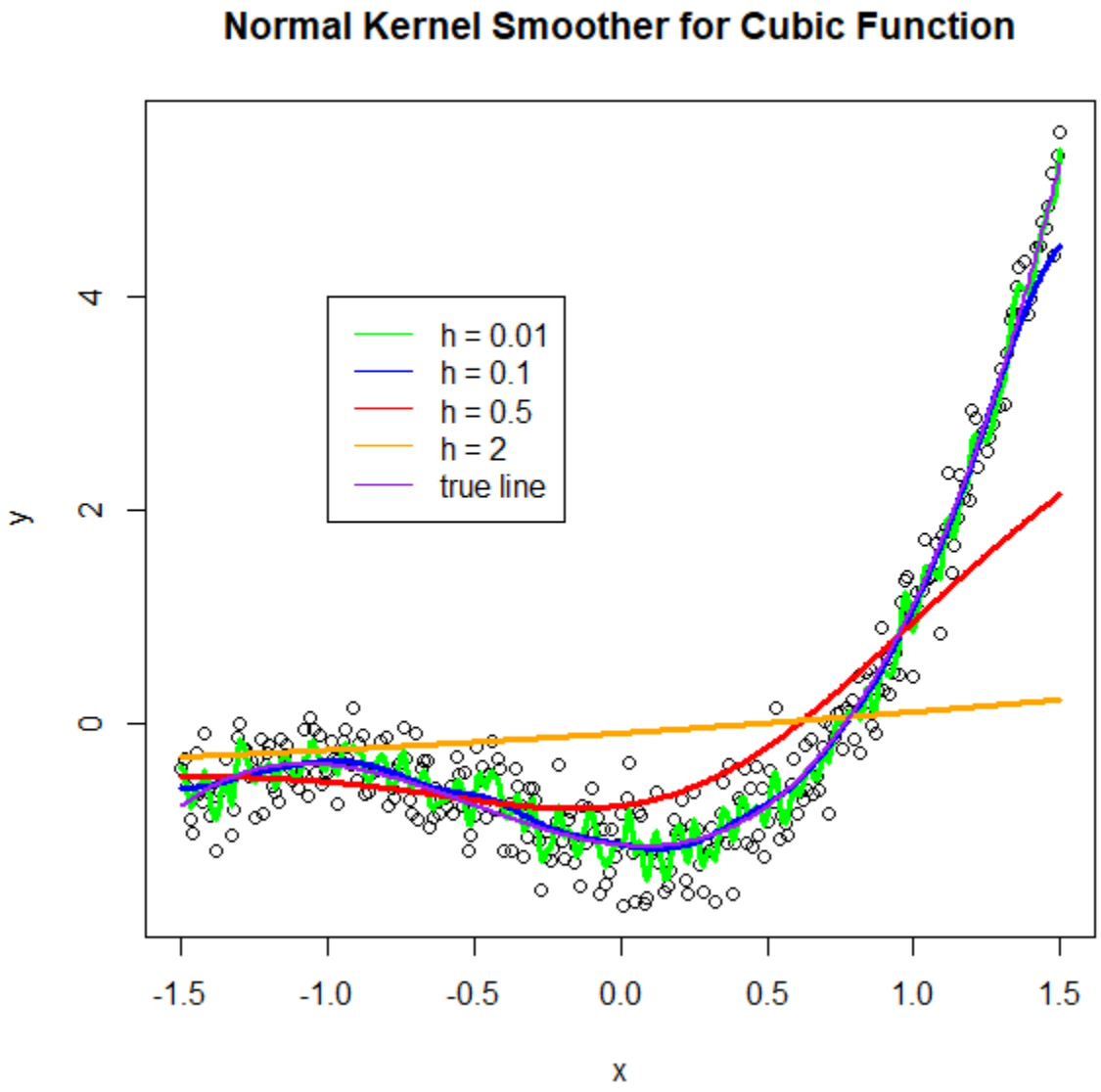
$$w(x_i, x^*) = \frac{x^* x_i}{s_{xx}},$$

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.

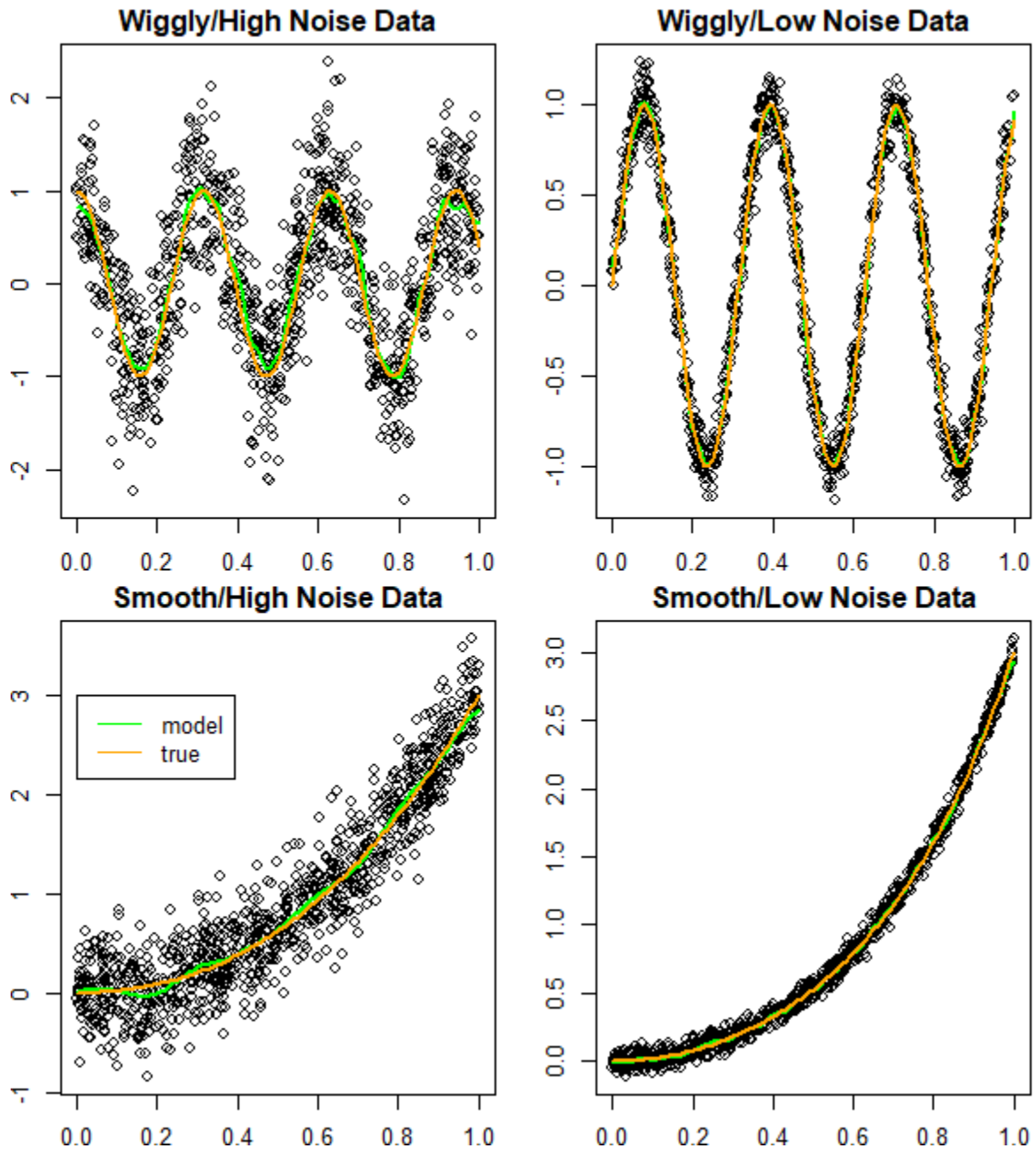


Cross Validation

(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:

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

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

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

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

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

Furthermore, let

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

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

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

so that

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

Solving this last equation yields,

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

(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 $\widehat{\boldsymbol{a}} = (\boldsymbol{R}_x^T \boldsymbol{W} \boldsymbol{R}_x)^{-1} \boldsymbol{R}_x^T \boldsymbol{W} \boldsymbol{y}$ is the least squares estimator of $\boldsymbol{a}$. Suppose

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

so that

$$\widehat{\boldsymbol{a}} = (\boldsymbol{R}_x^T \boldsymbol{W} \boldsymbol{R}_x)^{-1} \boldsymbol{R}_x^T \boldsymbol{W} \boldsymbol{y}$$

$$= \left[\begin{pmatrix} 1 & (x_1 - x) \\ 1 & (x_2 - x) \\ \vdots & \vdots \\ 1 & (x_n - x) \end{pmatrix}^T \begin{pmatrix} w(x_1, x) & 0 & 0 & \cdots & 0 \\ 0 & w(x_2, x) & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & w(x_n, x) \end{pmatrix} \begin{pmatrix} 1 & (x_1 - x) \\ 1 & (x_2 - x) \\ \vdots & \vdots \\ 1 & (x_n - x) \end{pmatrix} \right]^{-1}$$

$$\times \begin{pmatrix} 1 & (x_1 - x) \\ 1 & (x_2 - x) \\ \vdots & \vdots \\ 1 & (x_n - x) \end{pmatrix}^T \begin{pmatrix} w(x_1, x) & 0 & 0 & \cdots & 0 \\ 0 & w(x_2, x) & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & w(x_n, x) \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}$$

$$= \left[\begin{pmatrix} (x_1 - x)w(x_1, x) & (x_2 - x)w(x_2, x) & \cdots & (x_n - x)w(x_n, x) \\ w(x_1, x) & w(x_2, x) & \cdots & w(x_n, x) \end{pmatrix} \begin{pmatrix} 1 & (x_1 - x) \\ 1 & (x_2 - x) \\ \vdots & \vdots \\ 1 & (x_n - x) \end{pmatrix} \right]^{-1}$$

$$\times \begin{pmatrix} 1 & (x_1 - x) \\ 1 & (x_2 - x) \\ \vdots & \vdots \\ 1 & (x_n - x) \end{pmatrix}^T \begin{pmatrix} w(x_1, x) & 0 & 0 & \cdots & 0 \\ 0 & w(x_2, x) & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & w(x_n, x) \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}$$

$$= \begin{pmatrix} s_1(x) & s_2(x) \\ s_0(x) & s_1(x) \end{pmatrix}^{-1} \begin{pmatrix} 1 & (x_1 - x) \\ 1 & (x_2 - x) \\ \vdots & \vdots \\ 1 & (x_n - x) \end{pmatrix}^T \begin{pmatrix} w(x_1, x) & 0 & 0 & \cdots & 0 \\ 0 & w(x_2, x) & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & w(x_n, x) \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}$$

$$= \frac{1}{s_1^2(x) - s_2(x)s_0(x)} \begin{pmatrix} s_1(x) & -s_2(x) \\ -s_0(x) & s_1(x) \end{pmatrix} \begin{pmatrix} 1 & (x_1 - x) \\ 1 & (x_2 - x) \\ \vdots & \vdots \\ 1 & (x_n - x) \end{pmatrix}^T \begin{pmatrix} w(x_1, x) & 0 & 0 & \cdots & 0 \\ 0 & w(x_2, x) & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & w(x_n, x) \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}$$

$$= \frac{1}{s_1^2(x) - s_2(x)s_0(x)} \begin{pmatrix} s_1(x) & -s_2(x) \\ -s_0(x) & s_1(x) \end{pmatrix}$$

$$\times \begin{pmatrix} (x_1 - x)w(x_1, x) & (x_2 - x)w(x_2, x) & \cdots & (x_n - x)w(x_n, x) \\ w(x_1, x) & w(x_2, x) & \cdots & w(x_n, x) \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}$$

$$= \frac{1}{s_1^2(x) - s_2(x)s_0(x)} \begin{pmatrix} s_1(x) & -s_2(x) \\ -s_0(x) & s_1(x) \end{pmatrix} \begin{pmatrix} \sum y_i w(x_i, x)(x_i - x) \\ \sum y_i w(x_i, x) \end{pmatrix}$$

$$= \frac{1}{s_1^2(x) - s_2(x)s_0(x)} \begin{pmatrix} \sum [y_i s_1(x) w(x_i, x)(x_i - x) - y_i s_2(x) w(x_i, x)] \\ \sum [y_i s_1(x) w(x_i, x) - y_i w(x_i, x)(x_i - x) s_0(x)] \end{pmatrix}$$

$$= \begin{pmatrix} \hat{\alpha}_0 \\ \hat{\alpha}_1 \end{pmatrix}.$$

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

$$\hat{f}(x) = \hat{\alpha}_0$$

$$= \frac{\sum [y_i s_1(x) w(x_i, x)(x_i - x) - y_i s_2(x) w(x_i, x)]}{s_1^2(x) - s_2(x)s_0(x)}$$

$$= \frac{\sum (y_i h w(x_i, x)[s_1(x)(x_i - x) - s_2(x)])}{[s_1(x) \sum h w(x_i, x)(x_i - x) - s_2(x) \sum h w(x_i, x)]}$$

$$= \frac{\sum \left(y_i K \left(\frac{x_i - x}{h} \right) [s_1(x)(x_i - x) - s_2(x)] \right)}{\sum (h w(x_i, x)[s_1(x)(x_i - x) - s_2(x)])}$$

$$= \frac{\sum \left(y_i K \left(\frac{x_i - x}{h} \right) [s_2(x) - s_1(x)(x_i - x)] \right)}{\sum \left(K \left(\frac{x_i - x}{h} \right) [s_2(x) - s_1(x)(x_i - x)] \right)}$$

$$= \frac{\sum w_i(x) y_i}{\sum w_i(x)}.$$

(C)

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

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

and, assuming that the ϵ_i are independent and have equal variance σ^2,

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

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

$$\mathrm{Cov}(\boldsymbol{y}) = \sigma^2\boldsymbol{I}.$$

Hence,

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

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

and

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

(D)

We define

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

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

Now, note that

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

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

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

Thus,

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

so that

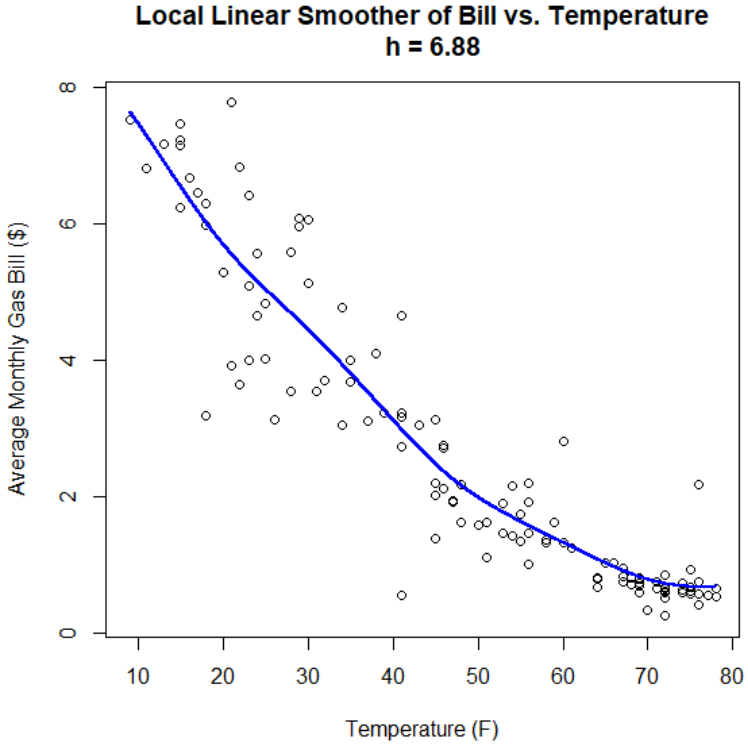
$$E[\hat{\sigma}^2] = E\left[\frac{\|\boldsymbol{r}\|}{n - 2\text{tr}(\boldsymbol{H}) + \text{tr}(\boldsymbol{H}^T\boldsymbol{H})}\right]$$

$$= \frac{\sigma^2[n - 2\text{tr}(\boldsymbol{H}) + \text{tr}(\boldsymbol{H}^T\boldsymbol{H})] + \boldsymbol{\mu} - \boldsymbol{H}^T\boldsymbol{\mu}}{n - 2\text{tr}(\boldsymbol{H}) + \text{tr}(\boldsymbol{H}^T\boldsymbol{H})}$$

$$= \sigma^2 + \frac{\boldsymbol{\mu} - \boldsymbol{H}^T\boldsymbol{\mu}}{n - 2\text{tr}(\boldsymbol{H}) + \text{tr}(\boldsymbol{H}^T\boldsymbol{H})}.$$

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

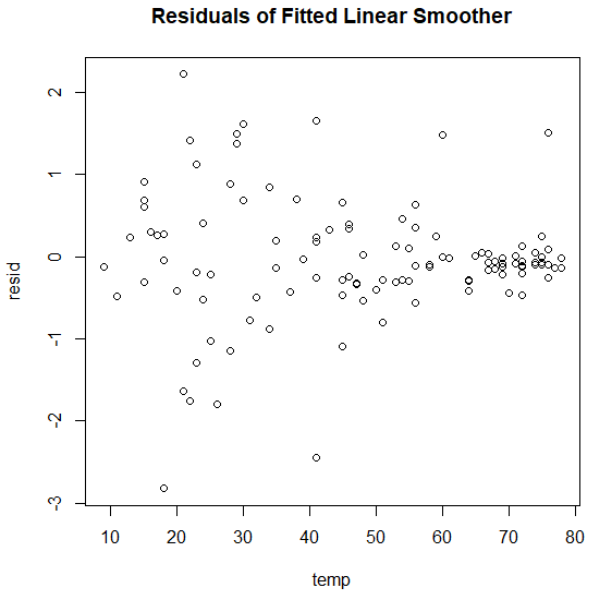
(E)


Local Linear Smoother of Bill vs. Temperature
h = 6.88
Average Monthly Gas Bill ($)
Temperature (F)
0
2
4
6
8
10
20
30
40
50
60
70
80

(F)

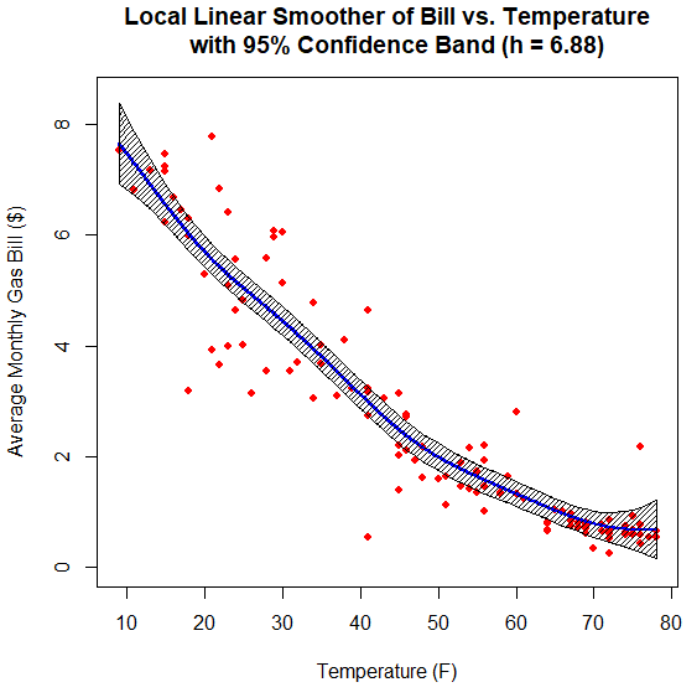
We see from the left graph below that there appears to be some heteroskedasticity in the original data. The residuals make a sort of fan shape as me move from left to right, indicating that the variance in the data decreases as the temperature increases.

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)

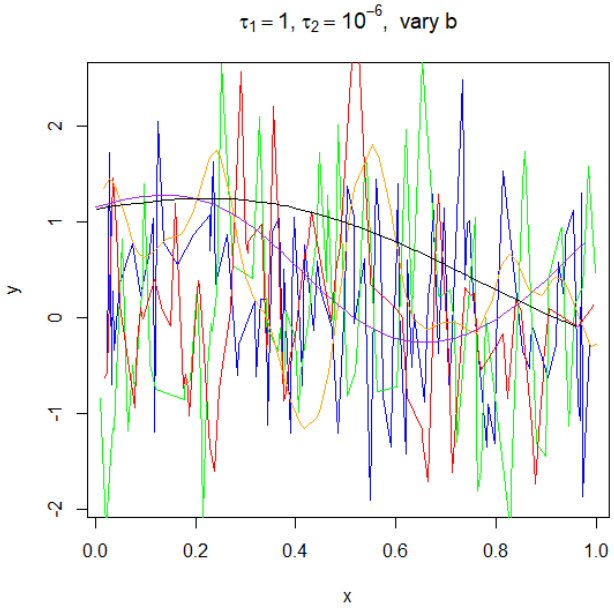
We construct confidence bands for a particular $f(x)$ in the form $\hat{f}(x) \pm c \cdot \hat{\sigma}$, where $\hat{\sigma}$ is defined as previously, and where c is the upper 0.025 quantile of a standard normal distribution; i.e., $c \approx 2$.



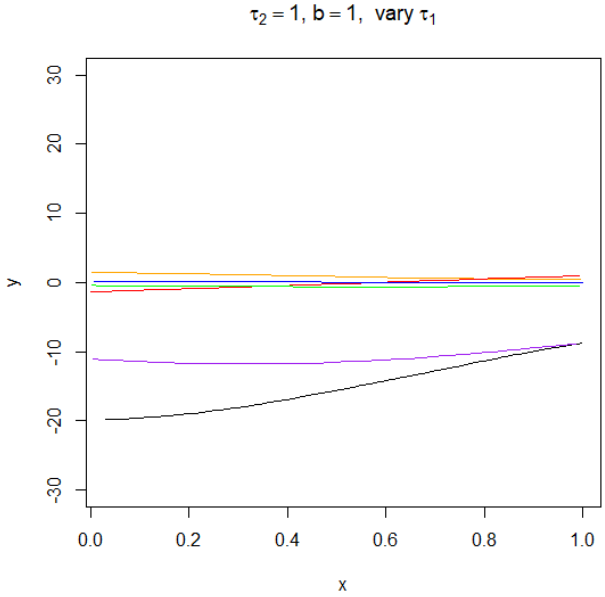
Gaussian processes

(A)

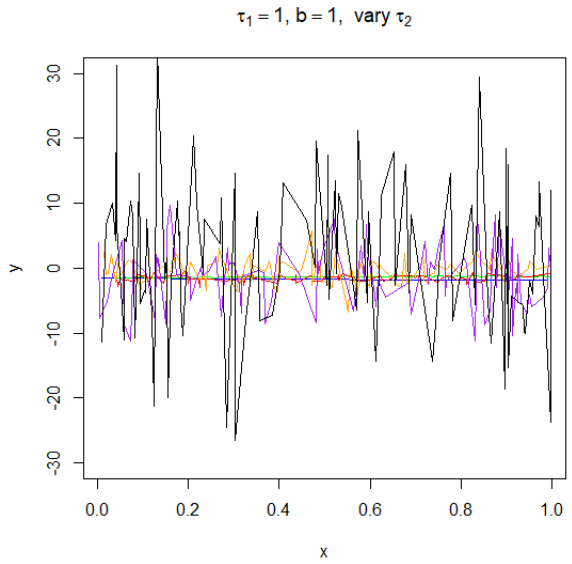
As we can see in the graphs below, the different parameters control different aspects of the realizations of the Gaussian process samples. It appears that b controls the curvature in the graph; small values cause high curvature, while larger values mean less curvature, so that graphs appear more linear. It seems that τ_1 has some effect on the distance that the sample has from the mean of zero. If τ_1 is large, the curve will be farther away from the center. Finally, τ_2 seems to control how far the high areas of curvature are from the center, analogous to the amplitudes of waves.



b
0.001
0.005
0.01
0.05
0.25
0.5

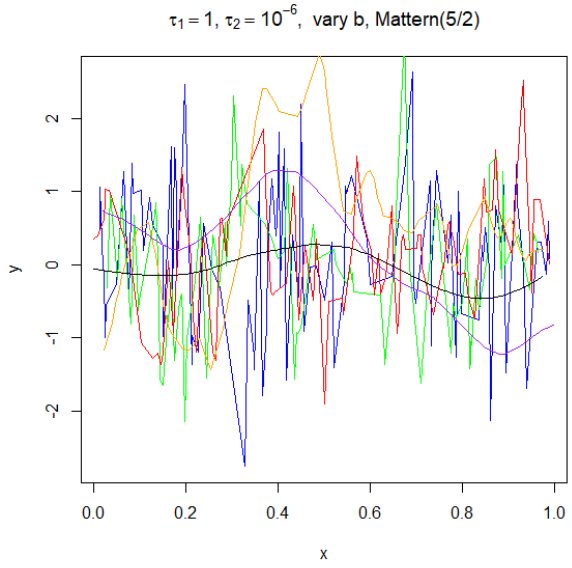


τ_1
0.1
0.5
1
2
5
10

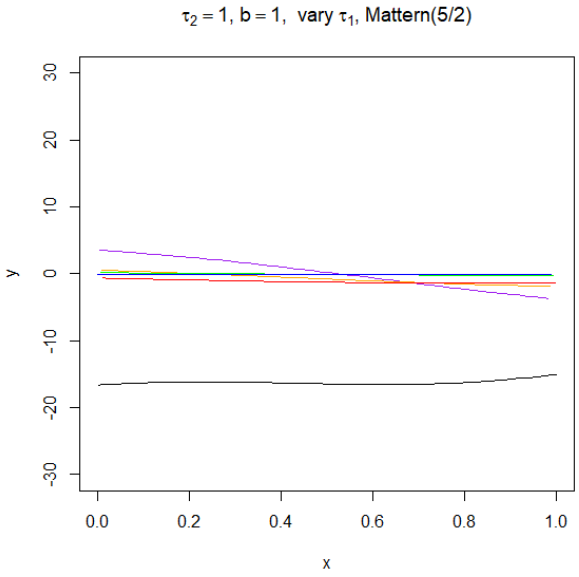


τ_2
0.01
0.05
0.5
2
5
10

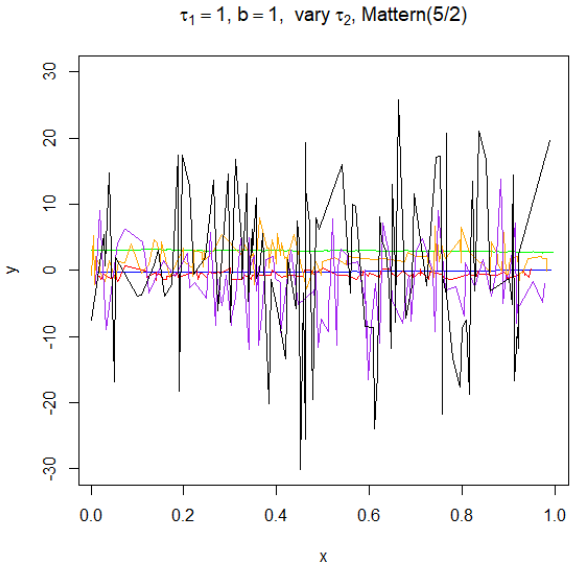
Below are similar experiments with the Mattern(5/2) covariance function.



b
0.001
0.005
0.01
0.05
0.25
0.5



τ_1
0.1
0.5
1
2
5
10



τ_2
0.01
0.05
0.5
2
5
10

When $\tau_2 = 0$, we obtain singular covariances matrices. In order to generate MVN samples with a singular covariance matrix, we define a function `sing()` that samples using the singular value decomposition of the covariance matrix. (See R code.) Below are three samples from different Gaussian process, all of which have $\tau_2 = 0$.

To see why this approach works mathematically, let $\boldsymbol{M}$ be an arbitrary positive semidefinite matrix, possibly singular. (All covariance matrices are positive semidefinite and can sometimes be singular.) Then the singular value decomposition of $\boldsymbol{M}$ can be written as:

$$\boldsymbol{M}_{n\times n} = \boldsymbol{U}_{n\times n}\boldsymbol{S}_{n\times n}\boldsymbol{V}^T_{n\times n},$$

where $\boldsymbol{U}$ is an orthogonal matrix (i.e., $\boldsymbol{U}^T\boldsymbol{U} = \boldsymbol{U}\boldsymbol{U}^T = \boldsymbol{I}$) whose columns are the eigenvectors of $\boldsymbol{M}\boldsymbol{M}^T$, $\boldsymbol{V}$ is an orthogonal matrix whose columns are the eigenvectors of $\boldsymbol{M}^T\boldsymbol{M}$, and $\boldsymbol{S}$ is a diagonal matrix whose diagonal elements are the square roots of the eigenvalues of $\boldsymbol{M}$. But since $\boldsymbol{M}$ is symmetric, we have $\boldsymbol{M}^T\boldsymbol{M} = \boldsymbol{M}^2 = \boldsymbol{M}\boldsymbol{M}^T$, so that the eigenvalues of $\boldsymbol{M}^T\boldsymbol{M}$ and $\boldsymbol{M}\boldsymbol{M}^T$ coincide, by which we can conclude that $\boldsymbol{U} = \boldsymbol{V}$, and thus $\boldsymbol{M} = \boldsymbol{U}\boldsymbol{S}\boldsymbol{U}^T$.

Next, if $\boldsymbol{z}\sim\mathcal{N}_p(\boldsymbol{0}, \boldsymbol{I})$ is a standard multivariate normal random variable, we claim that

$$\boldsymbol{U}\boldsymbol{S}^{\frac{1}{2}}\boldsymbol{U}^T\boldsymbol{z} + \boldsymbol{\mu}$$

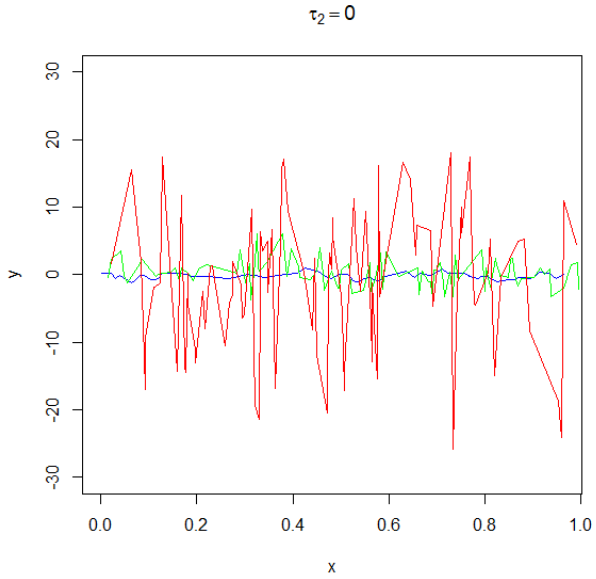
is multivariate normal with mean $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{U}\boldsymbol{S}^{\frac{1}{2}}\boldsymbol{U}^T$. To see why, we note first that this is an affine transformation of a multivariate random vector, so that our results earlier indicate that it is also multivariate normal. Hence, we need only find its mean and variance to fully characterize it.

$$E\left[\boldsymbol{U}\boldsymbol{S}^{\frac{1}{2}}\boldsymbol{U}^T\boldsymbol{z} + \boldsymbol{\mu}\right] = \boldsymbol{U}\boldsymbol{S}^{\frac{1}{2}}\boldsymbol{U}^T E[\boldsymbol{z}] + \boldsymbol{\mu} = \boldsymbol{0} + \boldsymbol{\mu} = \boldsymbol{\mu},$$

and

$$\begin{aligned}
\mathrm{Var}\left(\boldsymbol{U}\boldsymbol{S}^{\frac{1}{2}}\boldsymbol{U}^T\boldsymbol{z} + \boldsymbol{\mu}\right) &= \mathrm{Var}\left(\boldsymbol{U}\boldsymbol{S}^{\frac{1}{2}}\boldsymbol{U}^T\boldsymbol{z}\right) \\
&= \left(\boldsymbol{U}\boldsymbol{S}^{\frac{1}{2}}\boldsymbol{U}^T\right)\boldsymbol{I}\left(\boldsymbol{U}\boldsymbol{S}^{\frac{1}{2}}\boldsymbol{U}^T\right)^T \\
&= \boldsymbol{U}\boldsymbol{S}^{\frac{1}{2}}\boldsymbol{U}^T\boldsymbol{U}\boldsymbol{S}^{\frac{1}{2}}\boldsymbol{U}^T \\
&= \boldsymbol{U}\boldsymbol{S}^{\frac{1}{2}}\boldsymbol{I}\boldsymbol{S}^{\frac{1}{2}}\boldsymbol{U}^T \\
&= \boldsymbol{U}\boldsymbol{S}\boldsymbol{U}^T \\
&= \boldsymbol{M},
\end{aligned}$$

where we have used the orthogonality of $\boldsymbol{U}$. Thus, we see that sampling from a $\mathcal{N}_p(\boldsymbol{\mu}, \boldsymbol{M})$ distribution is the same as sampling from the standard normal distribution and then applying the transformation $\boldsymbol{U}\boldsymbol{S}^{\frac{1}{2}}\boldsymbol{U}^T\boldsymbol{z} + \boldsymbol{\mu}$. This is the approach we take in the simulations below when $\tau_2 = 0$.


$\tau_2 = 0$
30
20
10
0
-10
-20
-30
y
0.0
0.2
0.4
0.6
0.8
1.0
x

(B)

By the definition of Gaussian processes, the joint distribution of $x^*, x_1, \dots, x_N$ is multivariate normal with mean parameter

$$\boldsymbol{m} = \begin{pmatrix} \mu_{x^*} \\ \mu_{x_1} \\ \vdots \\ \mu_{x_N} \end{pmatrix}$$

and covariance matrix

$$\boldsymbol{\Sigma} = \begin{pmatrix} C(x^*, x^*) & C(x^*, x_1) & \cdots & C(x^*, x_N) \\ C(x_1, x^*) & C(x_1, x_1) & \cdots & C(x_1, x_N) \\ \vdots & \vdots & \ddots & \vdots \\ C(x_N, x^*) & C(x_N, x_1^*) & \vdots & C(x_N, x_N) \end{pmatrix}.$$

We can partition the mean and covariance matrix as follows in order to derive the conditional distribution of $x^*|x_1, \dots, x_N$.

$$\boldsymbol{m} = \begin{pmatrix} \mu_{x^*} \\ \mu_{x_1} \\ \vdots \\ \mu_{x_N} \end{pmatrix} = \begin{pmatrix} \mu_1 \\ \boldsymbol{\mu}_2 \end{pmatrix},$$

where $\mu_1 = \mu_{x^*}$, and $\boldsymbol{\mu}_2 = \left(\mu_{x_1}, \dots, \mu_{x_N}\right)^T$. Moreover,

$$\boldsymbol{\Sigma} = \begin{pmatrix} C(x^*, x^*) & C(x^*, x_1) & \cdots & C(x^*, x_N) \\ C(x_1, x^*) & C(x_1, x_1) & \cdots & C(x_1, x_N) \\ \vdots & \vdots & \ddots & \vdots \\ C(x_N, x^*) & C(x_N, x_1) & \vdots & C(x_N, x_N) \end{pmatrix} = \begin{pmatrix} \Sigma_{11} & \boldsymbol{\Sigma}_{12} \\ \boldsymbol{\Sigma}_{12}^T & \boldsymbol{\Sigma}_{22} \end{pmatrix},$$

where

$$\Sigma_{11} = C(x^*, x^*), \qquad \boldsymbol{\Sigma}_{12}^T = \begin{pmatrix} C(x_1, x^*) \\ \vdots \\ C(x_N, x^*) \end{pmatrix},$$

and

$$\boldsymbol{\Sigma}_{22} = \begin{pmatrix} C(x_1, x_2) & \cdots & C(x_1, x_N) \\ \vdots & \ddots & \vdots \\ C(x_N, x_1) & \cdots & C(x_N, x_N) \end{pmatrix}.$$

Our results from Exercise 1 then tell us that $x^*|x_1, \dots, x_N$ is multivariate normal with mean

$$m^* = \mu_1 + \boldsymbol{\Sigma}_{12}\boldsymbol{\Sigma}_{22}^{-1}((x_1, \dots, x_N)^T - \boldsymbol{\mu}_2)$$

and covariance

$$\Sigma^* = \Sigma_{11} - \boldsymbol{\Sigma}_{12}\boldsymbol{\Sigma}_{22}^{-1}\boldsymbol{\Sigma}_{12}^T.$$

(C)

The joint distribution of $\boldsymbol{y}$ and $\boldsymbol{\theta}$ is as follows:

$$\begin{aligned} f(\boldsymbol{y},\boldsymbol{\theta}) &= f(\boldsymbol{y}|\boldsymbol{\theta})f(\boldsymbol{\theta}) \\ &\propto \exp\left\{-\frac{1}{2}(\boldsymbol{y}-\boldsymbol{R\theta})^T\boldsymbol{\Sigma}^{-1}(\boldsymbol{y}-\boldsymbol{R\theta})\right\}\exp\left\{-\frac{1}{2}(\boldsymbol{\theta}-\boldsymbol{m})^T\boldsymbol{V}^{-1}(\boldsymbol{\theta}-\boldsymbol{m})\right\} \\ &= \exp\left\{-\frac{1}{2}[\boldsymbol{y}^T\boldsymbol{\Sigma}^{-1}\boldsymbol{y}-2\boldsymbol{\theta}^T\boldsymbol{R}^T\boldsymbol{\Sigma}^{-1}\boldsymbol{y}+\boldsymbol{\theta}^T\boldsymbol{V}^{-1}\boldsymbol{\theta}-2\boldsymbol{m}^T\boldsymbol{V}^{-1}\boldsymbol{\theta}]\right\}, \end{aligned}$$

which we recognize as the quadratic form of a partitioned vector. Hence, the joint distribution of $\boldsymbol{y}$ and $\boldsymbol{\theta}$ must be multivariate normal. (Exponentiation of quadratic forms uniquely determines the kernel of a multivariate normal distribution.) More explicitly, consider the exponentiation of the following quadratic form with partitioned matrices

$$\exp\left\{\left[\begin{pmatrix}\boldsymbol{x}_1\\ \boldsymbol{x}_2\end{pmatrix}-\begin{pmatrix}\boldsymbol{\mu}_1\\ \boldsymbol{\mu}_2\end{pmatrix}\right]^T\begin{pmatrix}\boldsymbol{A}_{11} & \boldsymbol{A}_{12}\\ \boldsymbol{A}_{12}^T & \boldsymbol{A}_{22}\end{pmatrix}\left[\begin{pmatrix}\boldsymbol{x}_1\\ \boldsymbol{x}_2\end{pmatrix}-\begin{pmatrix}\boldsymbol{\mu}_1\\ \boldsymbol{\mu}_2\end{pmatrix}\right]\right\}$$

$$\begin{aligned} &\propto \exp\{\boldsymbol{x}_1^T\boldsymbol{A}_{11}\boldsymbol{x}_1-\boldsymbol{\mu}_1^T\boldsymbol{A}_{11}\boldsymbol{x}_1+\boldsymbol{x}_2^T\boldsymbol{A}_{12}^T\boldsymbol{x}_1-\boldsymbol{\mu}_2^T\boldsymbol{A}_{12}^T\boldsymbol{x}_1-\boldsymbol{x}_1^T\boldsymbol{A}_{11}\boldsymbol{\mu}_1-\boldsymbol{x}_2^T\boldsymbol{A}_{12}^T\boldsymbol{\mu}_1+\boldsymbol{x}_1^T\boldsymbol{A}_{12}\boldsymbol{x}_2-\boldsymbol{\mu}_1^T\boldsymbol{A}_{12}\boldsymbol{x}_2 \\ &\qquad +\boldsymbol{x}_2^T\boldsymbol{A}_{22}\boldsymbol{x}_2-\boldsymbol{\mu}_2^T\boldsymbol{A}_{22}\boldsymbol{x}_2-\boldsymbol{x}_1^T\boldsymbol{A}_{12}\boldsymbol{\mu}_2-\boldsymbol{x}_2^T\boldsymbol{A}_{22}\boldsymbol{\mu}_2\} \\ &= \exp\{\boldsymbol{x}_1^T\boldsymbol{A}_{11}\boldsymbol{x}_1-2(\boldsymbol{\mu}_1^T\boldsymbol{A}_{11}+\boldsymbol{\mu}_2^T\boldsymbol{A}_{12}^T)\boldsymbol{x}_1+2\boldsymbol{x}_1^T(\boldsymbol{A}_{12})\boldsymbol{x}_2-2(\boldsymbol{\mu}_1^T\boldsymbol{A}_{12}+\boldsymbol{\mu}_2^T\boldsymbol{A}_{22})\boldsymbol{x}_2+\boldsymbol{x}_2^T\boldsymbol{A}_{22}\boldsymbol{x}_2\}, \end{aligned}$$

which we see as analogous to the form for the joint of $\boldsymbol{y}$ and $\boldsymbol{\theta}$ above.

Since we know that $\boldsymbol{y}$ and $\boldsymbol{\theta}$ are jointly multivariate normal, all we need is the mean and covariance matrix to fully characterize the distribution. Recalling the partitioned form of multivariate normal distributions, we have

$$E[f(\boldsymbol{y},\boldsymbol{\theta})] = E\left[\begin{pmatrix}\boldsymbol{y}\\ \boldsymbol{\theta}\end{pmatrix}\right] = \begin{pmatrix}E[\boldsymbol{y}]\\ E[\boldsymbol{\theta}]\end{pmatrix}$$

and

$$\mathrm{Var}\big(f(\boldsymbol{y},\boldsymbol{\theta})\big) = \begin{pmatrix}\mathrm{Var}(\boldsymbol{y}) & \mathrm{Cov}(\boldsymbol{y},\boldsymbol{\theta})\\ \mathrm{Cov}(\boldsymbol{y},\boldsymbol{\theta})^T & \mathrm{Var}(\boldsymbol{\theta})\end{pmatrix}.$$

We know immediately that $E[\boldsymbol{\theta}] = \boldsymbol{m}$ and $\mathrm{Var}(\boldsymbol{\theta}) = \boldsymbol{V}$. To find $E[\boldsymbol{y}]$ and $\mathrm{Var}(\boldsymbol{y})$ we can use iterated expectations.

$$\begin{aligned} E[\boldsymbol{y}] &= E\big[E[\boldsymbol{y}|\boldsymbol{\theta}]\big] \\ &= E[\boldsymbol{R\theta}] \\ &= \boldsymbol{R}\,E[\boldsymbol{\theta}] \\ &= \boldsymbol{Rm}, \end{aligned}$$

and

$$\begin{aligned}\mathrm{Var}(\boldsymbol{y}) &= E[\mathrm{Var}(\boldsymbol{y}|\boldsymbol{\theta})] + \mathrm{Var}(E[\boldsymbol{y}|\boldsymbol{\theta}]) \\ &= E[\boldsymbol{\Sigma}] + \mathrm{Var}(\boldsymbol{R\theta}) \\ &= \boldsymbol{\Sigma} + \boldsymbol{R}\,\mathrm{Var}(\boldsymbol{\theta})\boldsymbol{R}^T \\ &= \boldsymbol{\Sigma} + \boldsymbol{RVR}^T.\end{aligned}$$

Lastly, to find $\mathrm{Cov}(\boldsymbol{y}, \boldsymbol{\theta})$, we use the formula $\mathrm{Cov}(\boldsymbol{y}, \boldsymbol{\theta}) = E[\boldsymbol{y\theta}^T] - E[\boldsymbol{y}]E[\boldsymbol{\theta}]^T$, and then evaluate $E[\boldsymbol{y\theta}^T]$ via iterated expectations again.

$$\begin{aligned}E[\boldsymbol{y\theta}^T] - E[\boldsymbol{y}]E[\boldsymbol{\theta}]^T &= E\big[E[\boldsymbol{y\theta}^T|\boldsymbol{\theta}]\big] - \boldsymbol{Rmm}^T \\ &= E[E[\boldsymbol{y}|\boldsymbol{\theta}]\boldsymbol{\theta}^T] - \boldsymbol{Rmm}^T \\ &= E[\boldsymbol{R\theta\theta}^T] - \boldsymbol{Rmm}^T \\ &= \boldsymbol{R}\,E[\boldsymbol{\theta\theta}^T] - \boldsymbol{Rmm}^T \\ &= \boldsymbol{R}(\mathrm{Var}(\boldsymbol{\theta}) + E[\boldsymbol{\theta}]E[\boldsymbol{\theta}]^T) - \boldsymbol{Rmm}^T \\ &= \boldsymbol{R}(\boldsymbol{V} + \boldsymbol{mm}^T) - \boldsymbol{Rmm}^T \\ &= \boldsymbol{RV}.\end{aligned}$$

And since $\mathrm{Cov}(\boldsymbol{\theta}, \boldsymbol{y}) = \mathrm{Cov}(\boldsymbol{y}, \boldsymbol{\theta})^T = \boldsymbol{V}^T\boldsymbol{R}^T$, the final result is

$$(\boldsymbol{\theta}, \boldsymbol{y}) \sim \mathcal{N}\left(\begin{pmatrix}\boldsymbol{Rm} \\ \boldsymbol{m}\end{pmatrix}, \begin{pmatrix}\boldsymbol{\Sigma} + \boldsymbol{RVR}^T & \boldsymbol{RV} \\ \boldsymbol{V}^T\boldsymbol{R}^T & \boldsymbol{V}\end{pmatrix}\right)$$

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, comupte 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\\trlil\\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$gasbill/data$billingdays

# 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(2, 8, by = 0.01)

# 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 = 6.88",
	     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 resids
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 exercicse
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 = 6.88)",
               xlab = "Temperature (F)",
               ylab = "Average Monthly Gas Bill ($)",
               pch = 20,
               cex = 1.2,
               col = "red",
               ylim = c(0, 8.5))
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)
```

```
# SDS 383D: Statistical Modeling I
# Exercise 3
# Gaussian Processes

# (A)

# Load library to generate MVN
library(mvtnorm)
library(MASS)

# Function to generate sample from GP
# Default covariance is first in notes
# Otherwise, Mattern(5/2) covariance

# Input:
	#@ x: vector of values to sample GP
	#@ b: scalar GP covar parameter
	#@ tau1: scalar GP covar parameter
	#@ tau2: scalar GP covar parameter

# Output:
	#@ sam: n x 2 data frame
		 column 1: x
		 column 2: sample values

GP <- function(x, b, tau1, tau2, k = 1){

	# Sample size
	n <- length(x)

	# Compute covariance
	covar <- matrix(NA, nrow = n, ncol = n)

	# Default covaiance
	if(k == 1){
		for(i in 1:n){
			for(j in 1:n){
				covar[i, j] <- tau1^2*
						exp(-1/2*(abs(x[i]-x[j])/b)^2)+
						tau2^2*(x[i] == x[j])
			}
		}
	}

	# Mattern(5/2) Covariance
	else{
		for(i in 1:n){
			for(j in 1:n){
				covar[i, j] <- tau1^2*
						(1 + sqrt(5)*abs(x[i] - x[j])/b +
						 5*abs(x[i] - x[j])^2/(3*b^2))*
						 exp(-sqrt(5)*abs(x[i]-x[j])/b) +
						 tau2^2*(x[i] == x[j])
			}
		}
	}

	# If covar is singular (i.e., determinant ~ 0)
	# Use sing() function to sample
	if(abs(det(covar)) < 10^(-9)){
		sam <- cbind(x, sing(covar))
	}

	# If covariance is nonsingular,
	# Use mvrnorm() to sample
	else{
	# Sample from MVN norm
		sam <- cbind(x, mvrnorm(1, rep(0, n), covar))
	}

	# Sort points for ease of plotting
	sam <- sam[order(x), ]
```

```
        return(sam)
}

# Function to sample from singular covar matrix
# Samples from MVN using SVD of covar matrix

# Input:
        #@ M: a singular covar matrix

# Output:
        #@ s: an MVN sample using covar

sing <- function(M){

        # Find M^(1/2) using SVD
        sqrtM <- svd(M)$v %*% sqrt(diag(svd(M)$d)) %*% t(svd(M)$v)

        # Sample from Z
        u <- rnorm(dim(M)[1])

        # Return (M^(1/2))*u
        return(sqrtM %*% u)
}


# Vary b
# tau2 = 10^(-6), tau1 = 1,

# b = 0.001
S <- GP(runif(100), b = 0.001, tau1 = 1, tau2 = 10^(-6))
x <- S[, 1]
y <- S[, 2]
plot(x, y, col = "blue", typ = "l",
            main = expression(paste(tau[1]==1, ", ",
                                    tau[2]==10^-6, ", ",
                                    " vary ", b)))

# b = 0.005
S <- GP(runif(100), b = 0.005, tau1 = 1, tau2 = 10^(-6))
x <- S[, 1]
y <- S[, 2]
lines(x, y, col = "green", typ = "l")

# b = 0.01
S <- GP(runif(100), b = 0.01, tau1 = 1, tau2 = 10^(-6))
x <- S[, 1]
y <- S[, 2]
lines(x, y, col = "red", typ = "l")

# b = 0.05
S <- GP(runif(100), b = 0.05, tau1 = 1, tau2 = 10^(-6))
x <- S[, 1]
y <- S[, 2]
lines(x, y, col = "orange", typ = "l")

# b = 0.25
S <- GP(runif(100), b = 0.25, tau1 = 1, tau2 = 10^(-6))
x <- S[, 1]
y <- S[, 2]
lines(x, y, col = "purple", typ = "l")

# b = 0.5
S <- GP(runif(100), b = 0.5, tau1 = 1, tau2 = 10^(-6))
x <- S[, 1]
y <- S[, 2]
lines(x, y, col = "black", typ = "l")

# Vary tau1
# tau2 = 10^(-6), b = 1

# tau1 = 10
S <- GP(runif(100), b = 1, tau1 = 10, tau2 = 10^(-6))
```

```
x <- S[, 1]
y <- S[, 2]
plot(x, y, col = "black", typ = "l",
            main = expression(paste(tau[2]==1, ", ",
                                    b==1, ", ",
                                    " vary ", tau[1])),
            ylim = c(-30, 30))

# tau1 = 5
S <- GP(runif(100), b = 1, tau1 = 5, tau2 = 10^(-6))
x <- S[, 1]
y <- S[, 2]
lines(x, y, col = "purple", typ = "l")

# tau1 = 2
S <- GP(runif(100), b = 1, tau1 = 2, tau2 = 10^(-6))
x <- S[, 1]
y <- S[, 2]
lines(x, y, col = "orange", typ = "l")

# tau1 = 1
S <- GP(runif(100), b = 1, tau1 = 1, tau2 = 10^(-6))
x <- S[, 1]
y <- S[, 2]
lines(x, y, col = "red", typ = "l")

# tau1 = 0.5
S <- GP(runif(100), b = 1, tau1 = 0.5, tau2 = 10^(-6))
x <- S[, 1]
y <- S[, 2]
lines(x, y, col = "green", typ = "l")

# tau1 = 0.1
S <- GP(runif(100), b = 1, tau1 = 0.1, tau2 = 10^(-6))
x <- S[, 1]
y <- S[, 2]
lines(x, y, col = "blue", typ = "l")


# tau1 = 1, b = 1
# Vary tau2

# tau2 = 0.01
S <- GP(runif(100), b = 1, tau1 = 1, tau2 = .01)
x <- S[, 1]
y <- S[, 2]
plot(x, y, col = "blue", typ = "l",
            main = expression(paste(tau[1]==1, ", ",
                                    b==1, ", ",
                                    " vary ", tau[2])),
            ylim = c(-30, 30))

# tau2 = 0.05
S <- GP(runif(100), b = 1, tau1 = 1, tau2 = .05)
x <- S[, 1]
y <- S[, 2]
lines(x, y, col = "green", typ = "l")

# tau2 = 0.5
S <- GP(runif(100), b = 1, tau1 = 1, tau2 = .5)
x <- S[, 1]
y <- S[, 2]
lines(x, y, col = "red", typ = "l")

# tau2 = 2
S <- GP(runif(100), b = 1, tau1 = 1, tau2 = 2)
x <- S[, 1]
y <- S[, 2]
lines(x, y, col = "orange", typ = "l")

# tau2 = 5
S <- GP(runif(100), b = 1, tau1 = 1, tau2 = 5)
```

```
x <- S[, 1]
y <- S[, 2]
lines(x, y, col = "purple", typ = "l")

# tau2 = 10
S <- GP(runif(100), b = 1, tau1 = 1, tau2 = 10)
x <- S[, 1]
y <- S[, 2]
lines(x, y, col = "black", typ = "l")

# What happens if tau2^2 = 0?
# Draw sample with tau2^2 = 0 using sing()

# tau2 = 0
S <- GP(runif(100), b = 0.01, tau1 = 0.5, tau2 = 0)
x <- S[, 1]
y <- S[, 2]
plot(x, y, col = "blue", typ = "l",
           main = expression(paste(tau[2]==0)),
           ylim = c(-30, 30))

S <- GP(runif(100), b = 0.001, tau1 = 2, tau2 = 0)
x <- S[, 1]
y <- S[, 2]
lines(x, y, col = "green")

S <- GP(runif(100), b = 0.001, tau1 = 10, tau2 = 0)
x <- S[, 1]
y <- S[, 2]
lines(x, y, col = "red")



# Experiments with Mattern(5/2) covariance

# Vary b
# tau2 = 10^(-6), tau1 = 1,

# b = 0.001
S <- GP(runif(100), b = 0.001, tau1 = 1, tau2 = 10^(-6), k = 2)
x <- S[, 1]
y <- S[, 2]
plot(x, y, col = "blue", typ = "l",
           main = expression(paste(tau[1]==1, ", ",
                                   tau[2]==10^-6, ", ",
                                   " vary ", b, ", Mattern(5/2)")))

# b = 0.005
S <- GP(runif(100), b = 0.005, tau1 = 1, tau2 = 10^(-6), k = 2)
x <- S[, 1]
y <- S[, 2]
lines(x, y, col = "green", typ = "l")

# b = 0.01
S <- GP(runif(100), b = 0.01, tau1 = 1, tau2 = 10^(-6), k = 2)
x <- S[, 1]
y <- S[, 2]
lines(x, y, col = "red", typ = "l")

# b = 0.05
S <- GP(runif(100), b = 0.05, tau1 = 1, tau2 = 10^(-6), k = 2)
x <- S[, 1]
y <- S[, 2]
lines(x, y, col = "orange", typ = "l")

# b = 0.25
S <- GP(runif(100), b = 0.25, tau1 = 1, tau2 = 10^(-6), k = 2)
x <- S[, 1]
y <- S[, 2]
lines(x, y, col = "purple", typ = "l")

# b = 0.5
```

```
S <- GP(runif(100), b = 0.5, tau1 = 1, tau2 = 10^(-6), k = 2)
x <- S[, 1]
y <- S[, 2]
lines(x, y, col = "black", typ = "l")

# Vary tau1
# tau2 = 10^(-6), b = 1

# tau1 = 10
S <- GP(runif(100), b = 1, tau1 = 10, tau2 = 10^(-6))
x <- S[, 1]
y <- S[, 2]
plot(x, y, col = "black", typ = "l",
           main = expression(paste(tau[2]==1, ", ",
                                   b==1, ", ",
                                   " vary ", tau[1], ", Mattern(5/2)")),
           ylim = c(-30, 30))

# tau1 = 5
S <- GP(runif(100), b = 1, tau1 = 5, tau2 = 10^(-6))
x <- S[, 1]
y <- S[, 2]
lines(x, y, col = "purple", typ = "l")

# tau1 = 2
S <- GP(runif(100), b = 1, tau1 = 2, tau2 = 10^(-6))
x <- S[, 1]
y <- S[, 2]
lines(x, y, col = "orange", typ = "l")

# tau1 = 1
S <- GP(runif(100), b = 1, tau1 = 1, tau2 = 10^(-6))
x <- S[, 1]
y <- S[, 2]
lines(x, y, col = "red", typ = "l")

# tau1 = 0.5
S <- GP(runif(100), b = 1, tau1 = 0.5, tau2 = 10^(-6))
x <- S[, 1]
y <- S[, 2]
lines(x, y, col = "green", typ = "l")

# tau1 = 0.1
S <- GP(runif(100), b = 1, tau1 = 0.1, tau2 = 10^(-6))
x <- S[, 1]
y <- S[, 2]
lines(x, y, col = "blue", typ = "l")

# tau1 = 1, b = 1
# Vary tau2

# tau2 = 0.01
S <- GP(runif(100), b = 1, tau1 = 1, tau2 = .01)
x <- S[, 1]
y <- S[, 2]
plot(x, y, col = "blue", typ = "l",
           main = expression(paste(tau[1]==1, ", ",
                                   b==1, ", ",
                                   " vary ", tau[2], ", Mattern(5/2)")),
           ylim = c(-30, 30))

# tau2 = 0.05
S <- GP(runif(100), b = 1, tau1 = 1, tau2 = .05)
x <- S[, 1]
y <- S[, 2]
lines(x, y, col = "green", typ = "l")

# tau2 = 0.5
S <- GP(runif(100), b = 1, tau1 = 1, tau2 = .5)
x <- S[, 1]
y <- S[, 2]
lines(x, y, col = "red", typ = "l")
```

```
# tau2 = 2
S <- GP(runif(100), b = 1, tau1 = 1, tau2 = 2)
x <- S[, 1]
y <- S[, 2]
lines(x, y, col = "orange", typ = "l")

# tau2 = 5
S <- GP(runif(100), b = 1, tau1 = 1, tau2 = 5)
x <- S[, 1]
y <- S[, 2]
lines(x, y, col = "purple", typ = "l")

# tau2 = 10
S <- GP(runif(100), b = 1, tau1 = 1, tau2 = 10)
x <- S[, 1]
y <- S[, 2]
lines(x, y, col = "black", typ = "l")
```