SDS 383D Ex 03:
Linear Smoothing and Gaussian Processes

February 18, 2016

Jennifer Starling

Basic Concepts

Bias-Variance Decomposition

Let $\hat{f}(x)$ be a noisy estimate of some function $f(x)$, evaluated at some point x. Define the mean-squared error of the estimate as

$$MSE(\hat{f}, f) = E\{[f(x) - \hat{f}]^2\}$$

Prove that $MSE\left(f, \hat{f}\right) = B^2 + v$, where

$$B = E\{\hat{f}(x)\} - f(x) \text{ and } v = var\{f(x)\} = E\left[\left(\hat{f} - E\left(\hat{f}\right)\right)^2\right]$$

Begin with the definition of MSE.

$$\begin{aligned}
MSE\left[f, \hat{f}\right] &= E\left[\left(f - \hat{f}\right)^2\right] \\
&= E\left[\left(\hat{f} - f\right)^2\right] \\
&= E\left[\left(\hat{f} - E\left(\hat{f}\right) + E\left(\hat{f}\right) - f\right)^2\right], \text{ adding/subtracting } E\left(\hat{f}\right) \\
&= E\left[\left(\underbrace{\hat{f} - E\left(\hat{f}\right)} + \underbrace{E\left(\hat{f}\right) - f}\right)\left(\underbrace{\hat{f} - E\left(\hat{f}\right)} + \underbrace{E\left(\hat{f}\right) - f}\right)\right] \\
&= E\left[\left(\hat{f} - E\left(\hat{f}\right)\right)^2\right] - E\left[\left(E\left(\hat{f}\right) - f\right)^2\right] + 2E\left[\left(\hat{f} - E\left(\hat{f}\right)\right)\left(E\left(\hat{f}\right) - f\right)\right] \\
&= E\left[\left(\hat{f} - E\left(\hat{f}\right)\right)^2\right] - \left(E\left(\hat{f}\right) - f\right)^2 + 2E\left[\left(\hat{f} - E\left(\hat{f}\right)\right)\left(E\left(\hat{f}\right) - f\right)\right] \\
&\qquad \text{since } E\left(E\left(X\right)\right) = E\left(X\right) \\
&= Var\left(\hat{f}\right) - \left(Bias\left(\hat{f}, f\right)\right)^2 + 0
\end{aligned}$$

The last term reduces to zero as shown below.

$$\begin{aligned}
&2E\left[\left(\hat{f} - E\left(\hat{f}\right)\right)\left(E\left(\hat{f}\right) - f\right)\right] \\
&= E\left[\hat{f}E\left(\hat{f}\right) - E\left(\hat{f}\right)E\left(\hat{f}\right) - \hat{f}f + E\left(\hat{f}\right)f\right]
\end{aligned}$$

Then $\hat{f} = E\left(\hat{f}\right)$, giving us

$$\begin{aligned}
&= E\left[E\left(\hat{f}\right)E\left(\hat{f}\right) - E\left(\hat{f}\right)E\left(\hat{f}\right) - E\left(\hat{f}\right)f + E\left(\hat{f}\right)f\right] \\
&= 0
\end{aligned}$$

Part A

Suppose we observe $x_1, \ldots, x_n$ from some distribution F, and want to estimate $f(0)$, the value of the probability density function at 0. Let h be a small positive number, called the bandwidth, and define the quantity

$$\pi_h = P\left(-\frac{h}{2} < X < \frac{h}{2}\right) = \int_{-h/2}^{h/2} f(x)dx$$

Clearly $\pi_h \approx hf(0)$. Let Y be the number of observations in a sample of size n that fall within the interval $(-h/2, h/2)$. What is the distribution of Y? What are its mean and variance in terms of n and π_h? Propose a simple estimator $\hat{f}(0)$ involving Y.

Let Y be the number of x_i in $(-\frac{h}{2}, \frac{h}{2})$. Then

$$Y \sim Binom(n, \pi_h)$$

To estimate $\hat{\pi}_h$,

$$\hat{\pi}_h = \frac{y}{n}, \text{ the Binomial MLE}$$

Therefore $y = n\pi_h$.
Then our simple estimator for $\hat{f}(0)$ is

$$\hat{f}(0) = \frac{\hat{\pi}_h}{h} = \frac{y}{nh}$$

Then, since $Y \sim Binom(n, \pi_h)$, expectation and variance are

$$E(Y) = n\pi_h$$
$$Var(Y) = n\pi_h(1 - \pi_h)$$

Part B

Suppose we expand $f(x)$ in a second-order Taylor series about 0:

$$f(x) \approx f(0) + xf'(0) + \frac{x^2}{2} f''(0)\,.$$

Use this in the above expression for π_h, together with the bias–variance decomposition, to show that

$$MSE\{\hat{f}(0), f(0)\} \approx Ah^4 + \frac{B}{nh}$$

for constants A and B that you should (approximately) specify. What happens to the bias and variance when you make h small? When you make h big?

Plug in Taylor series approximation to definition of π_h.

$$\begin{aligned}\hat{\pi}_h &\approx \int_{-\frac{h}{2}}^{\frac{h}{2}} \left[f(0) + xf'(0) + \frac{x^2}{2} f''(0) \right] dx \\ &= hf(0) + \frac{h^3 f''(0)}{24}\end{aligned}$$

Plug in $\hat{\pi}_h$ to $E(Y)$ and $Var(Y)$ to obtain components of $MSE = Var + Bias^2$.
Mean:

$$E\left[\hat{f}(0)\right] = E\left[\frac{Y}{nh}\right] = \frac{1}{nh} E\left[Y\right] = \frac{1}{nh} n\pi_h \approx \frac{1}{h}\left(\frac{h^2 f''(0)}{24}\right) = f(0) + \frac{h^2 f''(0)}{24}$$

Variance:

$$\begin{aligned}Var\left[\hat{f}(0)\right] = Var\left[\frac{Y}{nh}\right] = \frac{Var\left[Y\right]}{n^2h^2} = \frac{n\pi_h(1-\pi_h)}{n^2h^2} = \frac{\pi_h(1-\pi_h)}{nh^2} \approx \frac{\pi_h}{nh^2}^{(**)} \\ = \frac{hf(0) + \frac{h^3 f''(0)}{24}}{nh^2} = \frac{f(0)}{nh} + \frac{hf''(0)}{24n} \approx \frac{f(0)}{nh}^{(***)}\end{aligned}$$

$(**)$ Because h small positive number, so π_h small, so $(1-\pi_h) \approx 1$.
$(***)$ Because h small and $h < 1$, first term bigger than second by 24x, so can simplify.
Bias:

$$bias = E\left[\hat{f}(0)\right] - f(0) = f(0) + \frac{h^2 f''(0)}{24} - f(0) = \frac{h^2 f''(0)}{24}$$

Then MSE is as follows.

$$MSE = var + bias^2 \approx \frac{f(0)}{nh} + \left(\frac{h^2 f''(0)}{24}\right)^2 = \frac{f(0)}{nh} + + \frac{h^4\left(f''(0)\right)^4}{576}$$

- Smaller h $\rightarrow$ smaller bias, but larger variance.
- Large h $\rightarrow$ larger bias, but smaller variance.

Note: Could include all of the algebraic terms, but these two dominate in order of h.

Part C

Use this result to derive an expression for the bandwidth that minimizes mean-squared error, as a function of n. You can approximate any constants that appear, but make sure you get the right functional dependence on the sample size.

Could solve previous MSE expression for the optimal h by taking the first derivative, setting equal to zero, and getting an expression in terms of h. This expression would include n, so the optimal bandwidth depends on sample size.

Curve Fitting by Linear Smoothing

Consider a nonlinear regression problem with one predictor and one response: $y_i = f(x_i) + \epsilon_i$, where the ϵ_i are mean-zero random variables.

Part A

Suppose we want to estimate the value of the regression function y^* at some new point x^*, denoted $\hat{f}(x^*)$. Assume for the moment that $f(x)$ is linear, and that y and x have already had their means subtracted, in which case $y_i = \beta x_i + \epsilon_i$. Return to your least-squares estimator for multiple regression. Show that for the one-predictor case, your prediction $y^* = f(x^*) = \hat{\beta} x^*$ may be expressed as a linear smoother of the following form:

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

for any x^*. Inspect the weighting function you derived. Briefly describe your understanding of how the resulting smoother behaves, compared with the smoother that arises from an alternate form of the weight function $w(x_i, w^*)$:

$$w_K(x_i, x^*) = \begin{cases} 1/K, \, x_i \text{ one of the closest } K \text{ sample points to } x_i \\ 0, \text{ otherwise} \end{cases}$$

This is referred to as K-nearest-neighbor smoothing.

The weighting function for smoothing is derived using the following.

$$\begin{aligned} \hat{f}(x^*) &= \hat{\beta} x^* \\ &= X^*(X'X)^{-1}X'y \\ &= x^* \cdot \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n}(x_i - \bar{x})^2} \\ &= \frac{\sum_{i=1}^{n} x_i x^* y_i}{\sum_{i=1}^{n} x_i^2} \end{aligned}$$

since we are working in the single-predictor, single-response, mean-zero case.

Therefore,

$$\begin{aligned} \hat{f}(x^*) &= \sum_{i=1}^{n} w\left(x_i, x^*\right) y_i, \text{ with} \\ w\left(x_i, x^*\right) &= \frac{\sum_{i=1}^{n} x_i x^*}{\sum_{i=1}^{n} x_i^2} \end{aligned}$$

This is a linear smoother in the sense that all x^* points have their corresponding $y^* = \hat{f}(x^*)$ estimates 'smoothed' to the regression line represented by intercept 0, slope $\hat{\beta}$.

This is a different behavior than the *K-nearest-neighbor* smoothing. KNN smoothing is not fitting a line which is constructed by using all of the points. KNN smoothing is calculating each new y^* as the straight average of the closest K points y_i.

Part B

A kernel function $K(x)$ is a smooth function satisyfing

$$\int_{\mathbb{R}} K(x)dx = 1\,, \quad \int_{\mathbb{R}} xK(x)dx = 0\,, \quad \int_{\mathbb{R}} x^2K(x)dx > 0\,.$$

A very simple example is the uniform kernel,

$$K(x) = \frac{1}{2}I(x) \quad \text{where} \quad I(x) = \begin{cases} 1, & |x| \leq 1 \\ 0, & \text{otherwise}\,. \end{cases}$$

Another common example is the Gaussian kernel:

$$K(x) = \frac{1}{\sqrt{2\pi}}e^{-x^2/2}\,.$$

Kernels are used as weighting functions for taking local averages. Specifically, define the weighting function

$$w(x_i, x^\star) = \frac{1}{h}K\left(\frac{x_i - x^\star}{h}\right)\,,$$

where h is the bandwidth. Using this weighting function in a linear smoother is called kernel regression. (The weighting function gives the unnormalized weights; you should normalize the weights so that they sum to 1.)
Write your own R function that will fit a kernel smoother for an arbitrary set of x-y pairs, and arbitrary choice of (positive real) bandwidth h. Set up an R script that will simulate noisy data from some nonlinear function, $y = f(x) + \epsilon$; subtract the sample means from the simulated x and y; and use your function to fit the kernel smoother for some choice of h. Plot the estimated functions for a range of bandwidths large enough to yield noticeable changes in the qualitative behavior of the prediction functions.

See **R Appendix, R Functions**. Function is called **linear_smoother**. There are also functions to specify which kernel function the linear smoother should use. These functions are called **K_gaussian** and **K_uniform**.

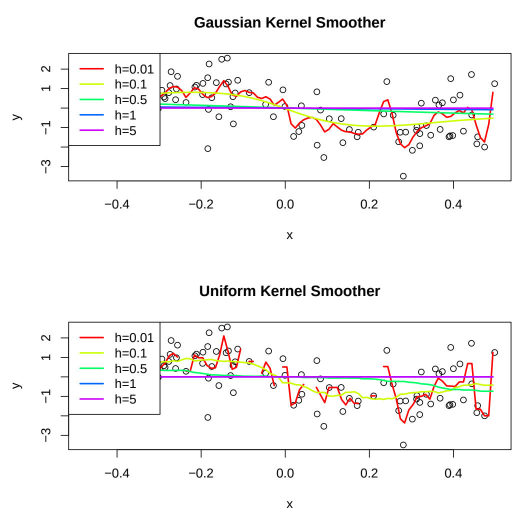

Figure 1: Kernel smoother for varying bandwidths h

Cross Validation

Left unanswered so far in our previous study of kernel regression is the question: how does one choose the bandwidth h used for the kernel? Assume for now that the goal is to predict well, not necessarily to recover the truth. (These are related but distinct goals.)

Part A

Presumably a good choice of h would be one that led to smaller predictive errors on fresh data. Write a function or script that will: (1) accept an old ("training") data set and a new ("testing") data set as inputs; (2) fit the kernel-regression estimator to the training data for specified choices of h; and (3) return the estimated functions and the realized prediction error on the testing data for each value of h. This should involve a fairly straightforward "wrapper" of the function you've already written.

See **R Appendix, R Functions**. Function is called **tune_h**.

Part B

Imagine a conceptual two-by-two table for the unknown, true state of affairs. The rows of the table are "wiggly function" and "smooth function," and the columns are "highly noisy observations" and "not so noisy observations." Simulate one data set (say, 500 points) for each of the four cells of this table, where the x's take values in the unit interval. Then split each data set into training and testing subsets. You choose the functions. Apply your method to each case, using the testing data to select a bandwidth parameter. Choose the estimate that minimizes the average squared error in prediction, which estimates the mean-squared error:

$$L_n(\hat{f}) = \frac{1}{n}\sum_{i=1}^{n^\star}(y_i^\star - \hat{y}_i^\star)^2 \, ,$$

where $(y_i^\star, x_i^\star)$ are the points in the test set, and $\hat{y}_i^\star$ is your predicted value arising from the model you fit using only the training data. Does your out-of-sample predictive validation method lead to reasonable choices of h for each case?

My function found optimal bandwidths of h as below, using a 70/30 train-test split. My functions were $sin(2\pi x)$ for smooth, and $sin(2\pi x)$ for wiggly.

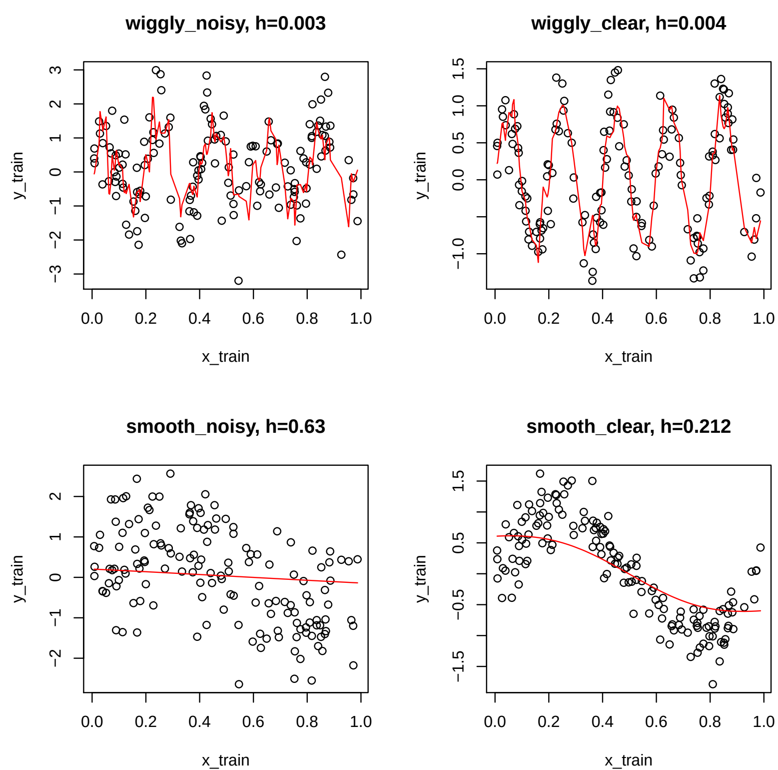

Figure 2: Bandwidth selection for various function types

These bandwidths look generally reasonable in terms of recovering the underlying functions, though the smooth/noisy function's bandwidth was smaller than I anticipated, and looks rather overfitted. My results also varied noticeably in quality as I reran the simulations for various random test/train splits, enough that I would recommend cross-validation each bandwidth selection via a few different test/train splits.

Part C

Splitting a data set into two chunks to choose h by out-of-sample validation has some drawbacks. List at least two. Then consider an alternative: leave-one-out cross validation. Define

$$\text{LOOCV} = \sum_{i=1}^{n} \left(y_i - \hat{y}_i^{(-i)} \right)^2 ,$$

where $\hat{y}_i^{(-i)}$ *is the predicted value of* y_i *obtained by omitting the ith pair and fitting the model to the reduced data set.*

The intuition here is straightforward: for each possible choice of h, you have to predict each data point using all the others. The bandwidth that with the lowest prediction error is the "best" choice by the LOOCV criterion. This is contingent upon a particular bandwidth, and is obviously a function of x_i*, but these dependencies are suppressed for notational ease. This looks expensive to compute: for each value of h, and for each data point to be held out, fit a whole nonlinear regression model. But you will derive a shortcut!*

Observe that for a linear smoother, we can write the whole vector of fitted values as $\hat{y} = Hy$*, where H is called the smoothing matrix (or "hat matrix") and y is the vector of observed outcomes.*

Remember that in multiple linear regression this is also true:

$$\hat{y} = X\hat{\beta} = X(X^T X)^{-1} X^T y = Hy \, .$$

Write $\hat{y}_i$ *in terms of H and y, and show that* $\hat{y}_i^{(-i)} = \hat{y}_i - H_{ii} y_i + H_{ii} \hat{y}_i^{(-i)}$*. Deduce that, for a linear smoother,*

$$\text{LOOCV} = \sum_{i=1}^{n} \left(\frac{y_i - \hat{y}_i}{1 - H_{ii}} \right)^2 .$$

Test/Train Split Issues:

A few problems regarding splitting the data into test/train chunks as in previous problem:

1. Lose potentially valuable information about outliers or patterns in the data by only using a portion of your data to train the model.
2. As mentioned previously, optimal bandwidth selection and the resulting model fit depended on the test/train split.

Plotting of Optimal Bandwidth h:

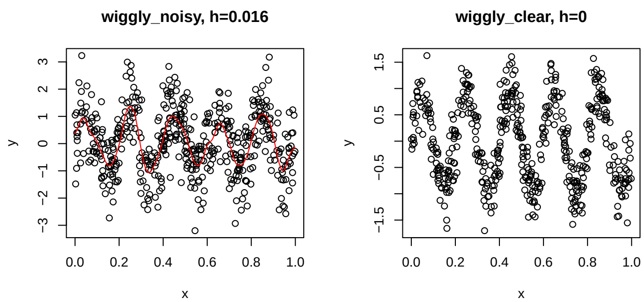

Figure 3: LOOCV Bandwidth selection for various function types

Derivation of H Matrix

Begin with definition of $\hat{y}$ for a single x^* value.

$$\hat{y} = \sum_{i=1}^{n} w(x_i, x^*) y_i \text{ for any value of } x^*$$

Extend to a vector of x^* values, and expression $\hat{y} = Hy$.

$$\begin{bmatrix} \hat{y}_1 \\ \vdots \\ \hat{y}_n \end{bmatrix}_{(nx1)} = \begin{bmatrix} w(x_1,x_1^*) & \dots & w(x_n,x_1^*) \\ \vdots & & \vdots \\ w(x_1,x_p^*) & \dots & w(x_n,x_p^*) \end{bmatrix}_{(nxp)} \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix}_{(nx1)}$$

Note that p is the length of the x^* vector, which will be n in the case of LOOCV, since we are using the x values in the existing data set to compute LOOCV prediction error. So H is (nxn).

Must also normalize the weights, so that $H = H/rowsums(H)$. Can formalize this properly as follows.

$$H = \{H_{ij}\} = \frac{w(x_j, x_i^*)}{\sum_{j=1}^{n} w(x_j, x_i^*)}$$

Accompanying Proof

Goal: Show

$$\hat{y}_i^{(-i)} = \hat{y}_i - H_{ii} y_i + H_{ii} \hat{y}_i^{(-i)}$$

Begin with the following two definitions.

$$\hat{y}_i = H_{ij} y_i \tag{1}$$

$$\hat{y}_i^{(-i)} = \sum_j H_{ij} a_j \tag{2}$$

where we define

$$a_j = \begin{cases} y_i & \text{if } i \neq j \\ \hat{y}_i^{(-i)} & \text{if } i = j \end{cases}$$

We define a_j in this way because we are fixing i, and

$$y^{(-i)} = \begin{bmatrix} y_1 \\ \vdots \\ y_{i-1} \\ y_i^{(-i)} \\ y_{i+1} \\ \vdots \\ y_n \end{bmatrix}$$, where $y_i^{(-i)}$is the LOOCV obs calculated using the non-i data.

Then

$$\begin{aligned} \hat{y}_i - \hat{y}_i^{(-i)} &= \sum_j H_{ij} y_i - \sum_j H_{ij} a_j \\ &= \sum_j H_{ij} \left(y_i - a_j \right) \\ &= \begin{cases} \sum_j H_{ij} \left(y_i - y_i \right) = 0 & \text{if } i \neq j \\ \sum_j H_{ij} \left(y_i - \hat{y}_i^{(-i)} \right) & \text{if } i = j \end{cases} \end{aligned}$$

This zeros out all non-diagonal terms, and we have chosen a fixed i, so are left with this expression.

$$\hat{y}_i - \hat{y}_i^{(-i)} = H_{ii} \left(y_i - \hat{y}_i^{(-i)} \right)$$

Distribute H_{ii} and rearrange to obtain our goal

$$\hat{y}_i^{(-i)} = \hat{y}_i - H_{ii} y_i + H_{ii} \hat{y}_i^{(-i)}$$

Local Polynomial Regression

Kernel regression has a nice interpretation as a "locally constant" estimator, obtained from locally weighted least squares. To see this, suppose we observe pairs (x_i, y_i) for $i = 1, \ldots, n$ from our new favorite model, $y_i = f(x_i) + \epsilon_i$ and wish to estimate the value of the underlying function $f(x)$ at some point x by weighted least squares. Our estimate is the scalar[1] quantity

$$\hat{f}(x) = a = \arg\min_{\mathbb{R}} \sum_{i=1}^{n} w_i (y_i - a)^2 \,,$$

where the w_i are the normalized weights (i.e. they have been rescaled to sum to 1 for fixed x). Clearly if $w_i = 1/n$, the estimate is simply $\bar{y}$, the sample mean, which is the "best" globally constant estimator. Using elementary calculus, it is easy to see that if the unnormalized weights are

$$w_i \equiv w(x, x_i) = \frac{1}{h} K\left(\frac{x_i - x}{h}\right) \,,$$

then the solution is exactly the kernel-regression estimator.

Part A

A natural generalization of locally constant regression is local polynomial regression. For points u in a neighborhood of the target point x, define the polynomial

$$g_x(u; a) = a_0 + \sum_{k=1}^{D} a_j (u - x)^k$$

for some vector of coefficients $a = (a_0, \ldots, a_D)$. As above, we will estimate the coefficients a in $g_x(u; a)$ at some target point x using weighted least squares:

$$\hat{a} = \arg\min_{R^{D+1}} \sum_{i=1}^{n} w_i \left\{y_i - g_x(x_i; a)\right\}^2 \,,$$

where $w_i \equiv w(x_i, x)$ are the kernel weights defined just above, normalized to sum to one.[2] Derive a concise (matrix) form of the weight vector $\hat{a}$, and by extension, the local function estimate $\hat{f}(x)$ at the target value x.[3] Life will be easier if you define the matrix R_x whose (i, j) entry is $(x_i - x)^{j-1}$, and remember that (weighted) polynomial regression is the same thing as (weighted) linear regression with a polynomial basis.

<u>Matrix form of $\hat{a}$</u>

$$\hat{a} = \underset{a \in \mathbb{R}^{D+1}}{\text{argmin}} \sum_{i=1}^{n} w_i \left[y_i - g_x\left(x_i; a\right)\right]^2$$

Sub in expression for $g_x\left(x_i; a\right)$.

$$\hat{a} = \underset{a \in \mathbb{R}^{D+1}}{\text{argmin}} \sum_{i=1}^{n} w_i \left[y_i - a_0 - \sum_{k=1}^{D} a_j \left(x_i - x\right)^k\right]^2$$

[1]Because we are only talking about the value of the function at a specific point x, not the whole function.

[2]We are fitting a different polynomial function for every possible choice of x. Thus $\hat{a}$ depends on the target point x, but we have suppressed this dependence for notational ease.

[3]Observe that at the target point x, $g_x(u = x; a) = a_0$. That is, only the constant term appears. But this is not the same thing as fitting only a constant term!

Switch to j indices, instead of k, for ease of notation. For clarify, we can expand the summation expression.

$$a_0 - \sum_{k=1}^{D} a_j(x_i - x)^k = a_0 + a_1(x_i - x) + a_2(x_i - x)^2 + ... + a_D(x_i - x)^{D+1}$$
$$= a_0(x_i - x)^0 + a_1(x_i - x)^1 + a_2(x_i - x)^2 + ... + a_D(x_i - x)^{D+1}$$

We will therefore define two matrices:

$W = diag(w_1, ..., w_n)$, a diagonal (nxn) matrix containing weights

$$R = \begin{bmatrix} (x_1-x)^0 & (x_1-x)^1 & (x_1-x)^2 & ... & (x_1-x)^D \\ \vdots & & & & \vdots \\ (x_n-x^0) & (x_n-x)^1 & (x_n-x)^2 & ... & (x_n-x)^D \end{bmatrix} \rightarrow \{R_{ij}\} = (x_i - x)^{j=1}, \text{ for } j = \{1, 2, ..., D+1\}$$

Then we can rewrite $\hat{a}$ as

$$\hat{a} = \underset{a \in \mathbb{R}^{D+1}}{\operatorname{argmin}} (y - Ra)^T W (y - Ra)$$

Minimize by taking derivative wrt a and solving for zero.

$$\begin{aligned} &\frac{d}{da}(y - Ra)^T W (y - Ra) \\ &= \frac{d}{da}\left[y^T W y - 2y^T W R a + a^T R^T W R a\right] \\ &- 2y^T W R + 2R^T W R a = 0 \\ &R^T W R a = R^T W y \\ &\hat{a} = (R^T W R)^{-1} R^T W y \end{aligned}$$

in a result with the same form as our usual weighted regression.

<u>Form of $f(\hat{x})$</u>

$$f(\hat{x}) = \hat{a}_0, \text{ the first element of } \hat{a}$$

This is because at target point x, only the constant term appears. This is not same as fitting only a constant term. A Taylor Approximation is the underlying intuition here. We can approximate polynomial $f(x)$ around some target point x_0 using a D-degree Taylor polynomial.

$$f(x) \approx f(x_0) + f'(x_0)(x - x_0) + \frac{f''(x_0)}{2!}(x - x_0)^2 + ...$$

When we approximate $f(x)$ centered at x_0 for the value x_0, only the first term $f(x_0)$ is left, and all other terms include $(x_0 - x_0)$ and so drop out.

Therefore, $\hat{f}(x^*) = \hat{a}_0$.

We can write this more compactly as:

$$\hat{f}(x) = e_1^T \hat{a} = e_1^T (R^T W R)^{-1} R^T W y, \text{ with } e_1 = (1, 0, 0, ...)_{(D+1x1)}$$

Part B

From this, conclude that for the special case of the local linear estimator ($D = 1$), we can write $\hat{f}(x)$ as a linear smoother of the form

$$\hat{f}(x) = \frac{\sum_{i=1}^n w_i(x) y_i}{\sum_{i=1}^n w_i(x)},$$

where the unnormalized weights are

$$\begin{aligned} w_i(x) &= K\left(\frac{x - x_i}{h}\right)\{s_2(x) - (x_i - x)s_1(x)\} \\ s_j(x) &= \sum_{i=1}^n K\left(\frac{x - x_i}{h}\right)(x_i - x)^j . \end{aligned}$$

Begin with the previous result for $\hat{f}(x)$.

$$\hat{f}(x) = e_1^T \hat{a} = e_1^T (R^T W R)^{-1} R^T W y \qquad \text{where } R = \begin{bmatrix} 1 & (x_1 - x) \\ \vdots & \vdots \\ 1 & (x_1 n - x) \end{bmatrix} \text{ and } w_i = K\left(\frac{x - x_i}{h}\right)$$

Then $\hat{f}(x)$ can be expanded as follows.

$$\hat{f}(x) = \underbrace{\begin{bmatrix} 1 & 0 \end{bmatrix}}_{e_1^T} \underbrace{\left[\begin{bmatrix} 1 & \dots & 1 \\ (x_1 - x) & \dots & (x_n - x) \end{bmatrix} \begin{bmatrix} w_1 & & 0 \\ & \ddots & \\ 0 & & w_n \end{bmatrix} \begin{bmatrix} 1 & (x_1 - x) \\ \vdots & \vdots \\ 1 & (x_1 n - x) \end{bmatrix} \right]^{-1}}_{(R^T W R)^{-1}} \underbrace{\begin{bmatrix} 1 & \dots & 1 \\ (x_1 - x) & \dots & (x_n - x) \end{bmatrix} \begin{bmatrix} w_1 & & 0 \\ & \ddots & \\ 0 & & w_n \end{bmatrix} \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix}}_{R^T W y}$$

$$= \underbrace{\begin{bmatrix} 1 & 0 \end{bmatrix}}_{(2x1)} \left[\underbrace{\begin{bmatrix} w_1 & \dots & w_n \\ w_1(x_1 - x) & \dots & w_n(x_n - x) \end{bmatrix}}_{(2xn)} \underbrace{\begin{bmatrix} 1 & (x_1 - x) \\ \vdots & \vdots \\ 1 & (x_n - x) \end{bmatrix}}_{(nx2)} \right]^{-1} \underbrace{\begin{bmatrix} w_1 & \dots & w_n \\ w_1(x_1 - x) & \dots & w_n(x_n - x) \end{bmatrix}}_{(2xn)} \underbrace{\begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix}}_{(nx1)}$$

$$= \underbrace{\begin{bmatrix} 1 & 0 \end{bmatrix}}_{(2x1)} \underbrace{\begin{bmatrix} \sum_{i=1}^n w_i & \sum_{i=1}^n w_i(x_i - x) \\ \sum_{i=1}^n w_i(x_i - x) & \sum_{i=1}^n w_i(x_i - x)^2 \end{bmatrix}^{-1}}_{(2x2)} \underbrace{\begin{bmatrix} \sum_{i=1}^n w_i y_i \\ \sum_{i=1}^n w_i(x_i - x) y_i \end{bmatrix}}_{(2x1)}$$

$$= \underbrace{\begin{bmatrix} 1 & 0 \end{bmatrix}}_{(2x1)} \underbrace{\begin{bmatrix} \sum_{i=1}^n w_i(x_i - x)^2 & -\sum_{i=1}^n w_i(x_i - x) \\ -\sum_{i=1}^n w_i(x_i - x) & \sum_{i=1}^n w_i \end{bmatrix}}_{(2x2)} \left(\frac{1}{\sum_{i=1}^n w_i \sum_{i=1}^n w_i(x_i - x)^2 - \left(\sum_{i=1}^n w_i(x_i - x)\right)^2} \right) \underbrace{\begin{bmatrix} \sum_{i=1}^n w_i y_i \\ \sum_{i=1}^n w_i(x_i - x) y_i \end{bmatrix}}_{(2x1)}$$

$$= \begin{bmatrix} \frac{\sum_{i=1}^n w_i(x_i - x)^2}{\sum_{i=1}^n w_i \sum_{i=1}^n w_i(x_i - x)^2 - \left(\sum_{i=1}^n w_i(x_i - x)\right)^2} & \frac{-\sum_{i=1}^n w_i(x_i - x)}{\sum_{i=1}^n w_i \sum_{i=1}^n w_i(x_i - x)^2 - \left(\sum_{i=1}^n w_i(x_i - x)\right)^2} \end{bmatrix} \begin{bmatrix} \sum_{i=1}^n w_i y_i \\ \sum_{i=1}^n w_i(x_i - x) y_i \end{bmatrix}$$

$$= \frac{\sum_{i=1}^n w_i(x_i - x)^2 \sum_{i=1}^n w_i y_i - \sum_{i=1}^n w_i(x_i - x) \sum_{i=1}^n w_i(x_i - x) y_i}{\sum_{i=1}^n w_i \sum_{i=1}^n w_i(x_i - x)^2 - \sum_{i=1}^n w_i \sum_{i=1}^n w_i(x_i - x)^2}$$

Let $s_1 = \sum_{i=1}^n w_i(x_i - x)$ and $s_2 = \sum_{i=1}^n w_i(x_i - x)^2$

$$= \frac{\sum_{i=1}^n w_i y_i s_2 - \sum_{i=1}^n w_i (x_i - x) y_i s_1}{\sum_{i=1}^n w_i \left(s_2 - (x_i - x) s_1\right)}$$

$$= \frac{\sum_{i=1}^n K\left(\frac{x-x_i}{h}\right)\left[s_2 - (x_i - x)s_1\right] y_i}{\sum_{i=1}^n K\left(\frac{x-x_i}{h}\right)\left[s_2 - (x_i - x)s_1\right]}$$

We can further simplify by defining an updated 'polynomial weight', call it w_i^* so that we can write $\hat{f}(x)$ in linear smoother form.

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

Then rewrite the previous expression in the desired form.

$$\hat{f}(x) = \frac{\sum_{i=1}^n w_i^* y_i}{\sum_{i=1}^n w_i^*}$$

Part C

Suppose that the residuals have constant variance σ^2 (that is, the spread of the residuals does not depend on x). Derive the mean and variance of the sampling distribution for the local polynomial estimate $\hat{f}(x)$ at some arbitrary point x. Note: the random variable $\hat{f}(x)$ is just a scalar quantity at x, not the whole function.

Note that $E(y) = E(y)$ and $Var(y) = \sigma^2 I$.

Expectation.

$$\begin{aligned}
E\left[\hat{f}(x)\right] &= E\left[e_1^T R\hat{a}\right] \\
&= e_1^T (R^T W R)^{-1} R^T W E(y) \\
&= e_1^T R^{-1} W^{-1} R^{-T} R^T W E(y) \\
&= e_1^T R^{-1} E(y)
\end{aligned}$$

Variance:

$$\begin{aligned}
Var\left[\hat{f}(x)\right] &= Var\left[e_1^T \hat{a}\right] \\
&= e_1^T (R^T W R)^{-1} R^T W Var(y) \left[e_1^T (R^T W R)^{-1} R^T W\right]^T \\
&= \sigma^2 e_1^T R^{-1} W^{-1} R^{-T} R^T W \left[e_1^T (R^T W R)^{-1} R^T W\right]^T \\
&= \sigma^2 e_1^T R^{-1} W^T R R^{-1} W^{-T} R^{-T} e_1 \\
&= \sigma^2 e_1^T e_1 \\
&= \sigma^2
\end{aligned}$$

Part D

We don't know the residual variance, but we can estimate it. A basic fact is that if x is a random vector with mean μ and covariance matrix Σ, then for any symmetric matrix Q of appropriate dimension, the quadratic form x^TQx has expectation

$$E(x^TQx) = \text{tr}(Q\Sigma) + \mu^TQ\mu\,.$$

Write the vector of residuals as $r = y - \hat{y} = y - Hy$, where H is the smoothing matrix. Compute the expected value of the estimator

$$\hat{\sigma}^2 = \frac{\|r\|_2^2}{n - 2\text{tr}(H) + \text{tr}(H^TH)}\,,$$

and simplify things as much as possible. Roughly under what circumstances will this estimator be nearly unbiased for large n? Note: the quantity $2tr(H) - tr(H^TH)$ is often referred to as the "effective degrees of freedom" in such problems.

$$\begin{aligned}
E\left[\hat{\sigma}^2\right] &= E\left[\frac{||r||_2^2}{n - 2tr(H) + tr(H^TH)}\right] \\
&= E\left[\frac{(y-Hy)^T(y-Hy)}{n - 2tr(H) + tr(H^TH)}\right] \\
&= \frac{E\left[y^Ty\right] - 2E\left[y^THy\right] + E\left[y^TH^THy\right]}{n - 2tr(H) + tr(H^TH)}
\end{aligned}$$

Then $E\left[x^TQx\right] = tr(Q\Sigma) + \mu^TQ\mu$, where $E(X) = \mu = f(x)$ and $Var(X) = \Sigma = \sigma^2 I$.

$$\begin{aligned}
E\left[\hat{\sigma}^2\right] &= \frac{tr(\Sigma) + \mu^T\mu - 2tr(H\Sigma) - 2\mu^TH\mu + tr(H^TH\Sigma) + \mu H^TH\mu}{n - 2tr(H) + tr(H^TH)} \\
&= \frac{tr(\sigma^2 I) + f(x)^Tf(x) - 2tr(H\sigma^2 I) - 2f(x)^THf(x) + tr(H^TH\sigma^2 I) + f(x)H^THf(x)}{n - 2tr(H) + tr(H^TH)} \\
&= \frac{\sigma^2 n + f(x)^Tf(x) - 2\sigma^2 tr(H) - 2f(x)^THf(x) + \sigma^2 tr(H^TH) + f(x)^TH^THf(x)}{n - 2tr(H) + tr(H^TH)} \\
&= \frac{\sigma^2\left(n - 2tr(H) + tr(H^TH)\right) + f(x)^T\left[I - 2H + H^TH\right]f(x)}{n - 2tr(H) + tr(H^TH)} \\
&= \sigma^2 + \frac{f(x)^T\left[I - 2H + H^TH\right]f(x)}{n - 2tr(H) + tr(H^TH)} \\
&= \sigma^2 + \frac{f(x)^T\left(I-H\right)^T\left(I-H\right)f(x)}{n - 2tr(H) + tr(H^TH)} \\
&= \sigma^2 + \frac{\left(f(x) - Hf(x)\right)^T\left(f(x) - Hf(x)\right)}{n - 2tr(H) + tr(H^TH)}
\end{aligned}$$

This estimator will be unbiased when the second term equals zero. This will occur when smoothing matrix H is a projection matrix into the space of $f(x)$. If H is a projection matrix into the space of $f(x)$, by definition, $Hf(x) = f(x)$ as projections leave elements in their target space unchanged.

This is a fancy way of saying that the estimator will be unbiased if we are recovering the function well using the smoothing matrix.

Part E

Write a new R function that fits the local linear estimator using a Gaussian kernel for a specified choice of bandwidth h. Then load the data in "utilities.csv" into R. This data set shows the monthly gas bill (in dollars) for a single-family home in Minnesota, along with the average temperature in that month (in degrees F), and the number of billing days in that month. Let y be the average daily gas bill in a given month (i.e. dollars divided by billing days), and let x be the average temperature. Fit y versus x using local linear regression and some choice of kernel. Choose a bandwidth by leave-one-out cross-validation.

See R code appendix for function details.

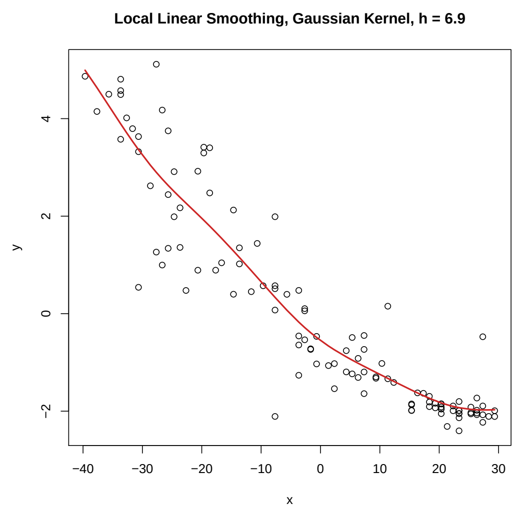

Figure 4: Local Linear Regression with LOOCV Bandwidth

For fun, an illustration of how fit changes with increased degrees.

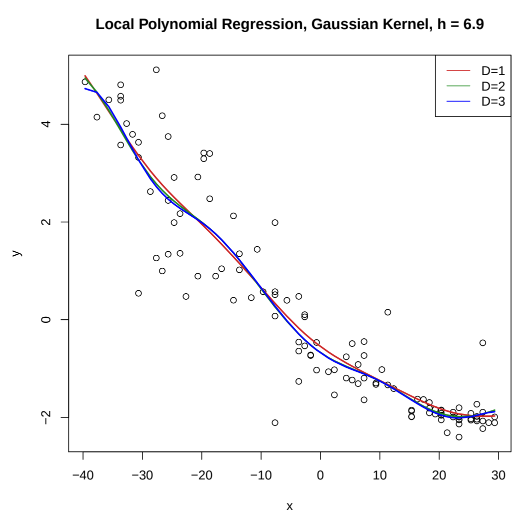

Figure 5: Local Polynomial Regression for Varying Degrees

Part F

Inspect the residuals from the model you just fit. Does the assumption of constant variance (homoscedasticity) look reasonable? If not, do you have any suggestion for fixing it?

The residuals do not look homoscedastic. Some kind of variance stabilizing transform could improve this, such as Anscombe.

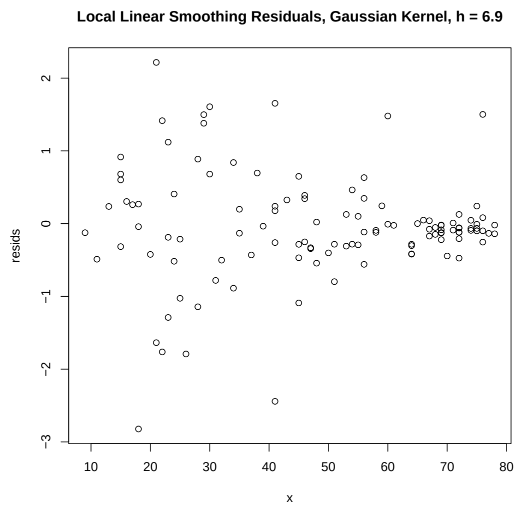

Figure 6: Local Linear Residuals

Part G

Put everything together to construct an approximate point-wise 95% confidence interval for the local linear model (using your chosen bandwidth) for the value of the function at each of the observed points x_i for the utilities data. Plot these confidence bands, along with the estimated function, on top of a scatter plot of the data. (It's fine to use Gaussian critical values for your confidence set.)

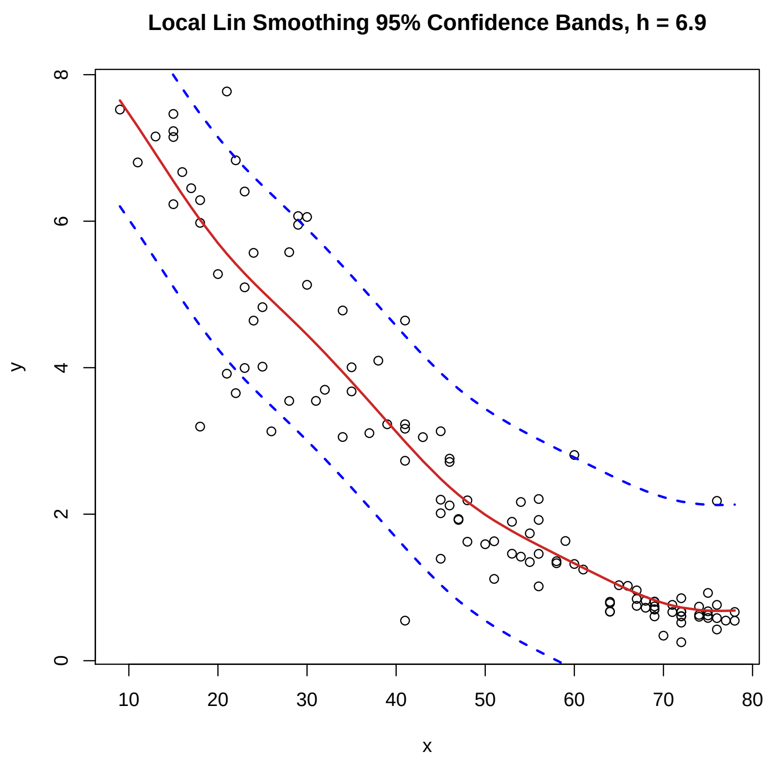

Figure 7: Confidence Bands

Appendix: R Code

R Functions

```
#SDS 383D — Exercise 3
#Functions
#Jennifer Starling
#Feb 2017

#================================================================
# Kernel Functions  =============================================
#================================================================

K_uniform = function(x){
    #------------------------------------------------------------
    #FUNCTION:  Uniform kernel.
    #------------------------------------------------------------
    #INPUTS:    x = a scalar or vector of values.
    #OUTPUT:    k = a scalar or vector of smoothed x values.
    #------------------------------------------------------------
    k = .5 * ifelse(abs(x)<=1,rep(1,length(x)),rep(0,length(x)))
    return(k)
}

K_gaussian = function(x){
    #------------------------------------------------------------
    #FUNCTION: Gaussian kernel.
    #------------------------------------------------------------
    #INPUTS:    x = a scalar or vector of values.
    #OUTPUT:    k = a scalar or vector of smoothed x values.
    #------------------------------------------------------------
    k = (1/sqrt(2*pi)) * exp(-x^2/2)
    return(k)
}

#================================================================
# Noisy Data Simulation =========================================
#================================================================

sim_noisy_data = function(x,f,sig2){
    #------------------------------------------------------------
    #FUNCTION:  Simulates noisy data from some nonlinear function f(x).
    #------------------------------------------------------------
    #INPUTS:    x = independent observations.
    #           f = function f(x) to simulate from
    #           sig2 = variance of the e ~ N(0,sig2) noise.
    #------------------------------------------------------------
    #OUTPUTS:   x = generated x values.
    #           y = generated y = f(x) + e values.
    #------------------------------------------------------------
    fx = f(x)
    e = rnorm(length(x),0,sqrt(sig2))
    return(y = fx+e)
}

#================================================================
# Smoothing Functions ===========================================
#================================================================

linear_smoother = function(x,y,x_star,h=1,K){
    #------------------------------------------------------------
```

```
    #FUNCTION:  Linear smoothing function for kernel regression.
    #------------------------------------------------------------------
    #INPUTS:    x = a scalar or vector of regression covariates.
    #           x_star = scalar or vector of new x values for prediction.
    #           h = a positive bandwidth.
    #           K = a kernel function.  Default is set to Gaussian kernel.
    #OUTPUT:    yhat = a scalar or vector of smoothed x values.
    #------------------------------------------------------------------
    yhat=0  #Initialize yhat.

    for (i in 1:length(x_star)){
        w = (1/h) * K((x-x_star[i])/h) #Calculates weights.
        w = w / sum(w)                  #Normalize weights.
        yhat[i] = crossprod(w,y)
    }
    return(yhat) #UPDATE TO INCLUDE WEIGHTS AS PART OF OUTPUT
}

local_linear_smoother = function(x,y,x_star,h=1,K){
    #------------------------------------------------------------------
    #FUNCTION:  Local linear smoothing function for special case D=1 polynomial
        regression.
    #------------------------------------------------------------------
    #INPUTS:    x = a scalar or vector of regression covariates.
    #           x_star = a scalar or vector of target point(s)
    #           h = a positive bandwidth.
    #           K = a kernel function.  Default is set to Gaussian kernel.
    #------------------------------------------------------------------
    #OUTPUT:    yhat = a scalar or vector of smoothed x values.

    n = length(x)                   #Number of obs.
    D = 1                           #Degrees of local polynomial fctn.
    n_star = length(x_star)         #Number of target points at which to predict.
    yhat = rep(0,n_star)        #Vector to hold predicted function values at target
        point(s).

    #Error check for appropriate h value.
    #try(if(h < 2/n || h > 1) stop("Enter h between (D+1)/n and 1"))

    #Matrix to hold diags of weight matrices. (Col of weights for each x_star.)
    weights = matrix(0,n,n_star)
    #NOTE: There is a separate H matrix for each x_star. Diagonals stored as columns.

    #Loop through each target point.
    for (b in 1:n_star){
        #Calculate unnormalized weights.
        s = rep(0,2)

        for (j in 1:2){
            s[j] = sum( K((x_star[b]-x)/h) * (x-x_star[b])^j )
        }

        w = K((x_star[b]-x)/h) * (s[2] - (x-x_star[b])*s[1])

        #Normalize weights.
        w = w / sum(w)

        #Save normalized weights.
        weights[,b] = w

        #Calculate function value for target point b.
        yhat[b] = crossprod(w,y)
```

```
        #CHECK: Print non-zero weight contributors.
        #print('Non-zero contributors to estimate:')
        #test_df = cbind(x=x,y=round(y,2),w=round(w,2),fhat=as.numeric(fhat_xstar[b]))
        #print(test_df[w>.001,])
    }

    #Return function output.
    return(list(yhat=yhat,weights=weights))
}

local_poly_smoother = function(x,y,x_star,h=1,K=K_gaussian,D=1){
    #------------------------------------------------------------------
    #FUNCTION:  Smoothing function for local polynomial regression.
    #------------------------------------------------------------------
    #INPUTS:    x = a scalar or vector of regression covariates.
    #           x_star = scalar new x value for prediction.
    #           h = a positive bandwidth.
    #           K = a kernel function.  Default is set to Gaussian kernel.
    #           D = degree of polynomial.  Default = 1.
    #------------------------------------------------------------------
    #OUTPUT:    yhat = a scalar or vector of smoothed x values.
    #------------------------------------------------------------------
    n = length(x)   #For i indices.
    yhat = rep(0,length(x_star))    #To store function outputs.
    weights.mat = matrix(0,nrow=n,ncol=length(x_star)) #To hold weights for each x-
        star.
    ahat = matrix(0,ncol=length(x_star),nrow=D+1)
    Hat = matrix(0,length(x_star),n)    #Store projection matrix.  Consists of row 1
        of R times each xstar intercept element.

    for (b in 1:length(x_star)){    #Loop through x_star points.

        xstar.i = x_star[b] #Pick off current x_star.

        #Calculate (nxn) weights matrix W.
        W = diag( (1/h) * K((x-xstar.i)/h) )

        #Set up R matrix.  {R_ij} = (x_i-x)^j for j in 1...D+1.
        R = matrix(0,nrow=n,ncol=D+1)

        for (i in 1:n){
            for (j in 1:(D+1)){
                R[i,j] = (x[i]-xstar.i)^(j-1)
            }
        }

        #Precache t(R) %*% W.
        RtW = t(R) %*% W

        #Calculate ahat.
        ahat.xstar = solve(RtW %*% R) %*% RtW %*% y

        #Calculate hat matrix. First row of ahat.xstar, without y.
        Hat[b,] = (solve(RtW %*% R) %*% RtW)[1,]

        #Estimated function value.
        yhat[b] = ahat.xstar[1]

        #Save ahat parameters for each x_star.
        ahat[,b] = ahat.xstar
    }
```

```
    return(list(yhat=yhat,weights=weights.mat,ahat=ahat,Hat = Hat))
}

#==============================================================
# Cross-Validation / Bandwidth Tuning ==========================
#==============================================================

tune_h = function(test,train,K,h){
    #------------------------------------------------------------
    #FUNCTION:  Function to tune bandwidth h for
    #               specified test/train data sets and
    #               specified kernel K for linear smoother.
    #------------------------------------------------------------
    #INPUTS:    test = a test data set. Must have two cols, x and y.
    #           train = a training data set. Must have two cols, x and y.
    #           K = the kernel function
    #           h = a scalar or vector of bandwidths to try.
    #OUTPUTS:   pred_err_test = prediction error for testing data for each h.
    #           fhat_test = predicted values for testing data's x vals.
    #------------------------------------------------------------
    #Extract training and test x and y vectors.
    x = train[,1]
    y = train[,2]
    x_star = test[,1]
    y_star = test[,2]

    #Calculate predicted points for test data set, and prediction error.
    yhat = linear_smoother(x,y,x_star,h,K)

    #Calculate predicted points for test data set, and prediction error.
    pred_err_test = sum(yhat-y_star)^2 / (length(x))

    #Return function outputs:
    return(list(yhat=yhat, pred_err_test = pred_err_test))
}

tune_h_loocv = function(x,y,K,h){
    #------------------------------------------------------------
    #FUNCTION:  Function to use leave on out LOOCV to tune
    #               bandwidth h for specified test/train data sets and
    #               specified kernel K for linear smoother.
    #------------------------------------------------------------
    #INPUTS:    x = a scalar or vector; the dependent variable.
    #           y = a scalar or vector; the independent var.  Same length as x.
    #           K = the kernel function
    #           h = a scalar or vector of bandwidths to try.
    #OUTPUTS:   pred_err_test = prediction error for testing data for each h.
    #           fhat_test = predicted values for testing data's x vals.
    #------------------------------------------------------------

    #Define ppm H, the 'smoothing matrix'.  {H_ij} = 1/h * K((xi-xj*)/h)
    #For loocv, x=x*, since calculating on single data set instead of test/train.

    Hat = matrix(0,nrow=length(x),ncol=length(x)) #Empty matrix.

    for (i in 1:length(x)){          #Loop through H rows.
        for (j in 1:length(x)){      #Loop through H cols.
            Hat[i,j] = (1/h) * K((x[j] - x[i])/h)
        }
        #Normalize weights by dividing H by rowsums(H).
        Hat[i,] = Hat[i,] / sum(Hat[i,])
    }
```

```
    #Calculate predicted values.
    yhat = Hat %*% y

    #Calculate loocv prediction error.
    loocv_err = sum(((y-yhat)/(1-diag(Hat)))^2)

    #Return function outputs:
    return(list(yhat=yhat,loocv_err=loocv_err))
}

tune_h_local_poly_loocv = function(x,y,K,h){
    #--------------------------------------------------------------
    #FUNCTION:  Function to use leave on out LOOCV to tune
    #               bandwidth h for specified test/train data sets and
    #               specified kernel K for local linear smoother.
    #--------------------------------------------------------------
    #INPUTS:    x = a scalar or vector; the dependent variable.
    #           y = a scalar or vector; the independent var.  Same length as x.
    #           K = the kernel function
    #           h = a scalar or vector of bandwidths to try.
    #--------------------------------------------------------------
    #OUTPUTS:   pred_err_test = prediction error for testing data for each h.
    #           fhat_test = predicted values for testing data's x vals.
    #--------------------------------------------------------------
    #Use existing x obs as target points for LOOCV.
    x_star = x

    #Call local_linear_smoother function to obtain yhat for each x point.
    output = local_poly_smoother(x,y,x_star,h=h,K_gaussian,D=1)
    yhat = output$yhat
    Hat = output$Hat

    Hii = diag(Hat)

    #Calculate loocv prediction error.
    loocv_err = sum(((y-yhat)/(1-Hii))^2)

    return(list(yhat=yhat,loocv_err=loocv_err))
}
```

Part B Script - Curve Fitting by Linear Smoothing

```
#Stats Modeling 2
#Exercise 3
#Curve Fitting by Linear Smoothing — Part B
#Jennifer Starling
#Feb 15, 2017

#————————————————————————————————————————————————————————————
###     Simulate noisy data from a non—linear function,
###     subtract the sample means from the simulated x and y,
###     and use the smoother function to fit the kernel smoother
###     for some choice of h.  Plot estimated functions for a
###     range of bandwidths large enough to yield noticeable changes
###     in the qualitative behavior of the prediction functions.
#————————————————————————————————————————————————————————————

### Environment setup.

#Housekeeping.
rm(list=ls())

#Load functions.
source("/Users/jennstarling/UTAustin/2017S_Stats Modeling 2/Exercise-03/RCode/SDS383D_
    Ex3_FUNCTIONS.R")

#————————————————————————————————————————————————————————————
### Data generation.

# Simulate x and y from a non—linear function x.
x = runif(100,0,1)  #Generate a sample of x values.
f = function(x) sin(2*pi*x) #Set the non—linear function.

#Generate noisy data, and extract x and y.
y = sim_noisy_data(x,f,sig2=.75)

#Subtract means from simulated x and y.
x = x - mean(x)
y = y - mean(y)

#Plot noisy data, with means subtracted.
pdf(file='/Users/jennstarling/UTAustin/2017S_Stats Modeling 2/Exercise-03/Figures/
    noisydata.pdf')
plot(x,y,main='Noisy data',xlab='x',ylab='y')
dev.off()

#Set up a vector of x_star values, and estimate function value at these points.
x_star = seq(min(x),max(x),by=.01)
yhat = linear_smoother(x,y,x_star,h=.01,K=K_gaussian)

plot(x,y,main='Gaussian Kernel Smoother')
lines(x_star,yhat,col='red')

#————————————————————————————————————————————————————————————
### Plot linear smoothing output for various bandwidths.

#Set up bandwidths to try, and corresponding colors.
H = c(.01,.1,.5,1,5)
col = rainbow(length(H))

#Open pdf file for two plots.
```

```
pdf(file='/Users/jennstarling/UTAustin/2017S_Stats Modeling 2/Exercise-03/Figures/
    Smoothers.pdf')
par(mfrow=c(2,1))

#Plot for a variety of bandwidths using Gaussian kernel.
plot(x,y,main='Gaussian Kernel Smoother')
for (i in 1:length(H)){
    yhat = linear_smoother(x,y,x_star,h=H[i],K=K_gaussian)
    lines(x_star,yhat,col=col[i],lwd=2)
}
legend('topleft',legend=paste("h=",H,sep=''),lwd=2,lty=1,col=col,bg='white')

#Plot for a variety of bandwidths using uniform kernel.
plot(x,y,main='Uniform Kernel Smoother')
for (i in 1:length(H)){
    yhat = linear_smoother(x,y,x_star,h=H[i],K=K_uniform)
    lines(x_star,yhat,col=col[i],lwd=2)
}
legend('topleft',legend=paste("h=",H,sep=''),lwd=2,lty=1,col=col,bg='white')

dev.off() #Close pdf file for two plots.
```

Part C Script - Cross Validation

```
#Stats Modeling 2
#Exercise 3
#Cross-Validation - Part B
#Jennifer Starling
#Feb 15, 2017

#------------------------------------------------------------
### Imagine a 2x2 tablefor the unknown, true
### state of affairs.  Rows are 'wiggly' and 'smooth' functions,
### and cols are 'highly noisy obs' and 'less noisy obs'.
### Simulate one data set (n=500) for each of four cells of table.
### Split each data set into test and train sets.  Apply method to
### each case.  Apply function from part A to select bandwidth.
#------------------------------------------------------------

#================================================================
# Environment Setup. ============================================
#================================================================

#Housekeeping.
rm(list=ls())

#Load latex table library.
library(xtable) #For table output to latex.
options(xtable.floating = FALSE)
options(xtable.timestamp = "")

#Load functions.
source('/Users/jennstarling/UTAustin/2017S_Stats Modeling 2/Exercise-03/RCode/SDS383D_
    Ex3_FUNCTIONS.R')

#================================================================
# Data and function generation. =================================
#================================================================

#Set up wiggly and less-wiggly functions.
fwiggly = function(x) sin(10*pi*x)
fsmooth = function(x) sin(2*pi*x)

#Set up x values on the unit interval.
n = 500
x = runif(n,0,1)

#Generate noisy and less-noisy data for
#each fwiggly and fsmooth function.
y_wiggly_noisy  = sim_noisy_data(x,fwiggly,sig2=.75)
y_wiggly_clear  = sim_noisy_data(x,fwiggly,sig2=.1)
y_smooth_noisy  = sim_noisy_data(x,fsmooth,sig2=.75)
y_smooth_clear  = sim_noisy_data(x,fsmooth,sig2=.1)

#Combine into single matrix for convenience.
y = matrix(c(y_wiggly_noisy,y_wiggly_clear,y_smooth_noisy,y_smooth_clear),ncol=4,byrow
    =F)
colnames(y) = c('wiggly_noisy','wiggly_clear','smooth_noisy','smooth_clear')

#Scatter plots to preview the four data sets.
#pdf(file='/Users/jennstarling/UTAustin/2017S_Stats Modeling 2/Exercise-03/Figures/cv_
    function_grid.pdf')
    par(mfrow=c(2,2))
    for(j in 1:ncol(y)){
```

```
        plot(x,y[,j],main=paste(colnames(y)[j]),xlab='x',ylab='y',ylim=c(-5,5))
    }
#dev.off()

#===============================================================
# Cross-Validation to Tune h (One Split) =======================
#  (Part B)                              =======================
#===============================================================

#-----------------------------------------------------------
### Split each data set into train and test sets. (70-30)

test_idx = sample(1:n,size=n*.3,replace=F)
test = cbind(x[test_idx],y[test_idx,])
train = cbind(x[-test_idx],y[-test_idx,])

#-----------------------------------------------------------
### Optimize bandwidth h.

# For each case, select a bandwidth parameter. (Used Gaussian kernel.)
h_opt = rep(0,4)    #Vector to hold optimal h values.
names(h_opt) = colnames(test)[-1]
H = seq(.001,1,by=.001) #Candidate h values.
yhat = list()           #To hold estimated function values.
x_star = test[,1]

#Iterate through each setup.
for (i in 1:4){
    #Extract x and just the y column required.
    tr = train[,c(1,i+1)]
    te = test[,c(1,i+1)]

    #Temp vector to hold prediction errors.
    temp_pred_err = rep(0,length(H))

    for (j in 1:length(H)){
        h = H[j]
        results = tune_h(test=te,train=tr,K=K_gaussian,h=h)
        temp_pred_err[j] = results$pred_err_test
    }

    h_opt[i] = H[which.min(temp_pred_err)]
    yhat[[i]] = tune_h(te,tr,K=K_gaussian,h_opt[i])$fhat_star
}

#-----------------------------------------------------------
### Output results as table and plot.

#Format optimal h results as matrix.
h_opt_mat = matrix(h_opt,nrow=2,byrow=T)
colnames(h_opt_mat) = c('noisy','clear')
rownames(h_opt_mat) = c('wiggly','smooth')
xtable(h_opt_mat,digits=3)  #Output latex table.

#Plot output with fitted data using optimal h values.
pdf('/Users/jennstarling/UTAustin/2017S_Stats Modeling 2/Exercise-03/Figures/Bandwidth
    _selection.pdf')
par(mfrow=c(2,2))
for (i in 1:4){
    #Scatterplot of test x/y.
    plot(test[,1],test[,i+1],
        main=paste(names(h_opt)[i],", h=",h_opt[i],sep=''),
```

```
        xlab='x_train',ylab='y_train')
    #Overlay estimated fit.
    idx = sort(x_star, index.return = T)$ix
    lines(sort(x_star),yhat[[i]][idx],col='red') #Fitted line.
}
dev.off()

#=================================================================
# LOOCV to Tune h  ===============================================
# (Part C)         ===============================================
#=================================================================

# For each case, select a bandwidth parameter. (Used Gaussian kernel.)
h_opt_loocv = rep(0,4)              #Vector to hold optimal h values.
names(h_opt_loocv) = colnames(y)
H = seq(.001,1,by=.001)             #Candidate h values.
yhat_loocv = list()             #To hold estimated function values.

#Iterate through each setup.
for (i in 1:4){

    #Extract y column.
    ytemp = y[,i]

    #Temp vector to hold prediction errors.
    temp_pred_err = rep(0,length(H))

    for (j in 1:length(H)){
        h = H[j]
        results = tune_h_loocv(x=x,y=ytemp,K=K_gaussian,h=h)
        temp_pred_err[j] = results$loocv_err
    }

    h_opt_loocv[i] = H[which.min(temp_pred_err)]
    yhat_loocv[[i]] = tune_h_loocv(x,ytemp,K=K_gaussian,h_opt_loocv[i])$yhat
}

#-------------------------------------------------------------
### Output results as table and plot.

#Format optimal h results as matrix.
h_opt_mat_loocv = matrix(h_opt_loocv,nrow=2,byrow=T)
colnames(h_opt_mat_loocv) = c('noisy','clear')
rownames(h_opt_mat_loocv) = c('wiggly','smooth')
xtable(h_opt_mat_loocv,digits=3)    #Output latex table.

#Plot output with fitted data using optimal h values.
pdf('/Users/jennstarling/UTAustin/2017S_Stats Modeling 2/Exercise-03/Figures/Bandwidth
    _selection_loocv.pdf')
par(mfrow=c(2,2))
for (i in 1:4){
    #Scatterplot of test x/y.
    plot(x,y[,i],
        main=paste(names(h_opt_loocv)[i],", h=",h_opt_loocv[i],sep=''),
        xlab='x',ylab='y')
    #Overlay estimated fit.
    idx = sort(x, index.return = T)$ix
    lines(sort(x),yhat_loocv[[i]][idx],col='red') #Fitted line.
}
dev.off()
```

Part D Script - Local Polynomial Regression

```
#Stats Modeling 2
#Exercise 3
#Local Polynomial Regression — Part E—G
#Jennifer Starling
#Feb 21, 2017

#================================================================
# Environment Setup & Data Load =================================
#================================================================

#Housekeeping.
rm(list=ls())

#Load functions.
source('/Users/jennstarling/UTAustin/2017S_Stats Modeling 2/Exercise-03/RCode/SDS383D_
    Ex3_FUNCTIONS.R')

#Load data.
utilities = read.csv('/Users/jennstarling/UTAustin/2017S_Stats Modeling 2/statsmod/
    Course-Data/utilities.csv',header=T)

#Extract data for model.
x = utilities$temp                                  #average temp.
y = utilities$gasbill / utilities$billingdays    #avg daily bill

#Center variables
x = x - mean(x)
y = y - mean(y)

#================================================================
# Local Linear Smoother: Optimize bandwidth h using loocv =======
#================================================================

#Candidate h values.
H = seq(1,8,by=.1)

#Vectors to hold prediction errors and estimated function values for each h.
pred_err = rep(0,length(H))

#Loop through h values.
for (m in 1:length(H)){
    h = H[m]
    results = tune_h_local_poly_loocv(x,y,K=K_gaussian,h=h)
    pred_err[m] = results$loocv_err
}

#Select optimal h and obtain fitted values.
h_opt = H[which.min(pred_err)]
yhat = local_linear_smoother(x,y,x,h=h_opt,K_gaussian)$yhat

#————————————————————————————————————————————————————————————
### Plot output with fitted data using optimal h value.

pdf('/Users/jennstarling/UTAustin/2017S_Stats Modeling 2/Exercise-03/Figures/Local_
    Linear_Opt_h.pdf')

#Scatterplot of x/y.
plot(x,y,main=paste('Local Linear Smoothing, Gaussian Kernel, h = ',sep="",h_opt))

#Overlay estimated fit.
```

```
x_star=x
idx = sort(x_star, index.return = T)$ix
    lines(sort(x_star),yhat[idx],col='firebrick3',lwd=2) #Fitted line.

dev.off()

#==============================================================
# Analysis of Residuals: =======================================
#==============================================================

#Fit D=1 local polynomial model (local linear) using optimal h.
#Inspect residuals.  Does homoscedasticity look reasonable?  If not, propose fix.

yhat = local_linear_smoother(x,y,x,h=h_opt,K_gaussian)$yhat
resids = (y-yhat)

### Plotting
pdf('/Users/jennstarling/UTAustin/2017S_Stats Modeling 2/Exercise-03/Figures/Local_
    Linear_Residuals.pdf')
plot(x,resids,main=paste('Local Linear Smoothing Residuals, Gaussian Kernel, h = ',sep
    ="",h_opt))
dev.off()

#Looks like residuals follows a trend.
#Try an Anscombe variance stabilizing transformation of y to fix this.
#Or other appropriate var stabilizing transforms.

#==============================================================
# Confidence Bands: ============================================
#==============================================================
RSS = sum(resids^2)
sigma2_hat = RSS / (length(yhat)-1)

lb = yhat - 1.96*sqrt(sigma2_hat)
ub = yhat + 1.96*sqrt(sigma2_hat)

#Plot bands.
#Scatterplot of x/y.
plot(x,y,main=paste('Local Lin Smoothing 95% Confidence Bands, h = ',sep="",h_opt))

#Overlay estimated fit.
x_star=x
idx = sort(x_star, index.return = T)$ix
    lines(sort(x_star),yhat[idx],col='firebrick3',lwd=2) #Fitted line.

#Overlay confidence bands.
lines(sort(x_star),lb[idx],col='blue',lwd=2,lty=2)
lines(sort(x_star),ub[idx],col='blue',lwd=2,lty=2)

#==============================================================
# Local Polynomial Smoother Degrees Demo: ======================
#==============================================================

x_star = x
h=6.9

#Local linear smoother - James solution.
fx_hat = local_poly_smoother(x,y,x_star,h,K=K_gaussian,D=1)$yhat

#Local linear smoother - via polynomial smoother with D=1.
fx_hat2 = local_poly_smoother(x,y,x_star,h,K=K_gaussian,D=2)$yhat
```

```
#Compare to linear smoother from previous ex part.
fx_hat3 = local_poly_smoother(x,y,x_star,h,K=K_gaussian,D=3)$yhat

pdf('/Users/jennstarling/UTAustin/2017S_Stats Modeling 2/Exercise-03/Figures/
    Polynomial_Degree_Comparison.pdf')
#Scatterplot of x/y.
plot(x,y,main=paste('Local Polynomial Regression, Gaussian Kernel, h = ',sep="",h))

#Overlay estimated fits.
idx = sort(x_star, index.return = T)$ix
    lines(sort(x_star),fx_hat[idx],col='firebrick3',lwd=2) #Fitted line.

idx = sort(x_star, index.return = T)$ix
    lines(sort(x_star),fx_hat2[idx],col='forestgreen',lwd=2) #Fitted line.

idx = sort(x_star, index.return = T)$ix
    lines(sort(x_star),fx_hat3[idx],col='blue',lwd=2) #Fitted line.

legend('topright',col=c('firebrick3','forestgreen','blue'),legend=c('D=1','D=2','D=3')
    ,lty=1)
dev.off()
```