SDS 383D: Homework 3

Giorgio Paulon

May 7, 2017

Problem 1. Basics 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

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

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

$$B = E\{\hat{f}(x)\} - f(x) \quad \textit{and} \quad V = Var\{\hat{f}(x)\}\,.$$

We can write

$$\begin{aligned}
\text{MSE}(\hat{f}, f) &= \text{E}\left[f(x)^2 + \hat{f}(x)^2 - 2f(x)\hat{f}(x)\right] \\
&= f(x)^2 + \text{E}\left[\hat{f}(x)^2\right] - 2f(x)\text{E}\left[\hat{f}(x)\right] \\
&= f(x)^2 - 2f(x)\text{E}\left[\hat{f}(x)\right] + \text{E}\left[\hat{f}(x)\right]^2 + \text{E}\left[\hat{f}(x)^2\right] - \text{E}\left[\hat{f}(x)\right]^2 \\
&= \left(f(x) - \text{E}\left[\hat{f}(x)\right]\right)^2 + \text{Var}\left[\hat{f}(x)\right] \\
&= B^2 + V.
\end{aligned}$$

A simple example

Some people refer to the above decomposition as the bias–variance tradeoff. Why a tradeoff? Here's a simple example to convey the intuition.

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 $\pi_h \approx hf(0)$.

(A) *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.*

The number of observation that fall in the interval $(-h/2, h/2)$ follows a Binomial distribution, i.e.

$$Y \sim \text{Bin}\,(n, \pi_h)\,.$$

Therefore, we get $\mathrm{E}[Y] = n\pi_h$ and $\mathrm{Var}(Y) = n\pi_h(1-\pi_h)$. A simple estimator for $f(0)$, say $\hat{f}(0)$, is

$$\hat{f}(0) = \frac{Y}{nh},$$

so that

$$\mathrm{E}[\hat{f}(0)] = \frac{1}{nh}\mathrm{E}[Y] = \frac{\pi_h}{h} \approx f(0)$$
$$\mathrm{Var}(\hat{f}(0)) = \frac{1}{n^2h^2}\mathrm{Var}(Y) = \frac{\pi_h(1-\pi_h)}{nh^2}$$

(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?

We can re-express the probabilities π_h using a second order Taylor approximation for $f(x)$, that is

$$\begin{aligned}
\pi_h &= \int_{-h/2}^{h/2} f(x)dx \\
&\approx \int_{-h/2}^{h/2} f(0)dx + \int_{-h/2}^{h/2} xf'(0)dx + \int_{-h/2}^{h/2} \frac{x^2}{2}f''(0)dx \\
&= f(0)\cdot h + \frac{1}{2}f'(0)\left[\left(\frac{h}{2}\right)^2 - \left(-\frac{h}{2}\right)^2\right] + \frac{1}{2}f''(0)\left[\frac{x^3}{3}\right]_{-h/2}^{h/2} \\
&= h\left(f(0) + \frac{f''(0)}{24}\cdot h^2\right)
\end{aligned}$$

Recall the formula for the MSE and write

$$\begin{aligned}
\mathrm{MSE}\left[\hat{f}(0), f(0)\right] &= \left[f(0) - \mathrm{E}\left(\hat{f}(0)\right)\right]^2 + \mathrm{Var}\left(\hat{f}(0)\right) \\
&= \left(f(0) - f(0) - \frac{f''(0)}{24}\cdot h^2\right)^2 + \frac{1}{nh}\left(f(0) + \frac{f''(0)}{24}\cdot h^2\right)\left(1 - f(0) - \frac{f''(0)}{24}\cdot h^2\right) \\
&= \left(\frac{f''(0)}{24}\right)^2 h^4 + \frac{1}{nh}\left(f(0) + \frac{f''(0)}{24}\cdot h^2 - f(0)^2 h - \frac{f(0)f''(0)}{12}\cdot h^3 - \frac{f''(0)^2}{24^2}\cdot h^5\right) \\
&\approx \underbrace{\left(\frac{f''(0)}{24}\right)^2\left(1 - \frac{1}{n}\right)}_{\mathrm{A}}\cdot h^4 + \frac{1}{nh}\cdot\underbrace{f(0)}_{\mathrm{B}}.
\end{aligned}$$

When h is small, the bias term gets small and the variance term gets large. The opposite happens when h is large.

(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.*

One can find the minimum of the MSE with a standard calculus procedure.

$$\begin{aligned}
&\underset{h}{\operatorname{argmin}}\, \text{MSE}(n) = \underset{h}{\operatorname{argmin}} \left(Ah^4 + \frac{B}{nh} \right) \\
&\Rightarrow \frac{d}{dh} \left(Ah^4 + \frac{B}{nh} \right) = 4Ah^3 - \frac{B}{nh^2} = 0 \\
&\Rightarrow h = \left(\frac{B}{4An} \right)^{1/5}
\end{aligned}$$

Problem 2. 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.

(A) *Suppose we want to estimate the value of the regression function $y^\star$ at some new point $x^\star$, denoted $\hat{f}(x^\star)$. 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 $\hat{y}^\star = f(x^\star) = \hat{\beta} x^\star$ may be expressed as a linear smoother of the following form:

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

for any $x^\star$. 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 $w(x_i, x^\star)$:

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

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

Let us suppose that $y_i = f(x_i) + \epsilon_i$, where $\epsilon_i \overset{\text{iid}}{\sim} N(0, \sigma^2)$. If the function f is linear, then $\mathrm{E}[y_i] = a\mathrm{E}[x] + b = 0$, which implies $b = 0$. Thus, $f(x) = \beta x$ and $y_i = \beta x + \epsilon_i$.

In the case of Least squares, the estimator is

$$\hat{y}^* = \hat{\beta} x^*,$$

where

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

Therefore, we get

$$\hat{y}^* = \frac{\sum_{i=1}^n x_i x^*}{\sum_{i=1}^n x_i^2} y_i = \sum_{i=1}^{n} w(x_i, x^*) y_i$$

where the weights are

$$w(x_i, x^*) = \frac{x_i x^*}{\sum_{i=1}^n x_i^2}.$$

This estimator uses all of the points x_i's. In the k-nearest neighbours case, it is easy to see how the distance between $x^\star$ and x_i affects the weights $w(x_i, x^\star)$. In the one variable OLS case, the corresponding intuition is that it only matters how $x_i - \bar{x}$ is large, not the distance between $x^\star$ and x_i. The weight $w(x_i, x^\star)$ is negative if x_i and $x^\star$ lie to the opposite sides of $\bar{x}$.

(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,

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

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.

In order to do kernel regression, we evaluate the smoothed function on a fine grid of points. For each point, we compute a set of weights that are given to the observations $x_1, \ldots, x_n$ given the distance to that particular point. The value of the bandwidth is crucial, as one can see in Figure 1 and in Figure 2. In particular, higher values for the bandwidth result in a smoother function, whereas smaller values of the bandwidth will tend to yield an interpolating and wiggly curve.

The linear smoothing problem can be reinterpreted, as we will see in the next section, as a locally constant regression. This is evident in Figure 1 and in Figure 2, where small values of the bandwidth produce flat lines.

Why would we use a uniform kernel instead of a gaussian one? The uniform kernel enforces sparsity in the smoothing matrix H, and so it could be preferable for high dimensional data.

Let us also remark that on the left edge, since the function is increasing, for small bandwidths the function will be biased upwards (**edge bias**). The same happens on the right edge.

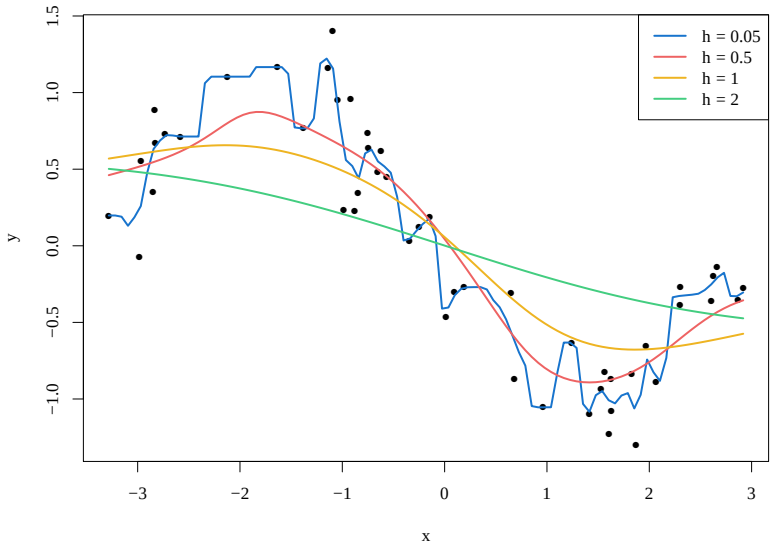


Figure 1: Gaussian kernel regression for the sampled black points. Different values of the bandwidth λ result in different smoothed curves.

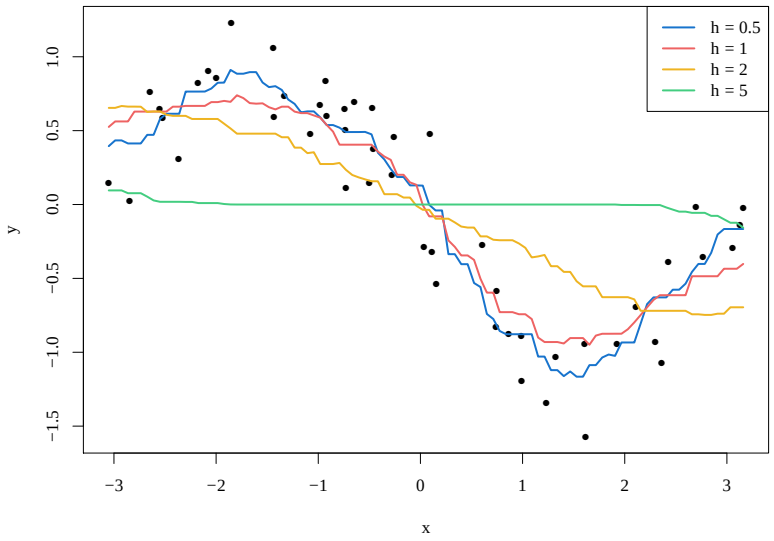


Figure 2: Uniform kernel regression for the sampled black points. Different values of the bandwidth λ result in different smoothed curves.

Problem 3. 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.)

(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.*

The requested functions are displayed in Listing A.1.

(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:*

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

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?

In Table 1 the values of the bandwidth are displayed in the four different cases. We see that, as expected, the highest bandwidth will arise in the smooth and noisy case, since the corresponding function will be flat. On the other hand, a non noisy wiggly function will produce a small value of the bandwidth and therefore a not-so-smooth estimate.

	Wiggly	Smooth
Noisy	0.01556	0.04780
Not noisy	0.00793	0.01947

Table 1: Comparison of the optimal bandwidths for different choices of the underlying function.

In Figure 3 the results are displayed visually. As one can see, the cross-validation procedure yields to reasonable choices for the optimal bandwidth. The true mechanism generating the data seems to be reconstructed pretty well!

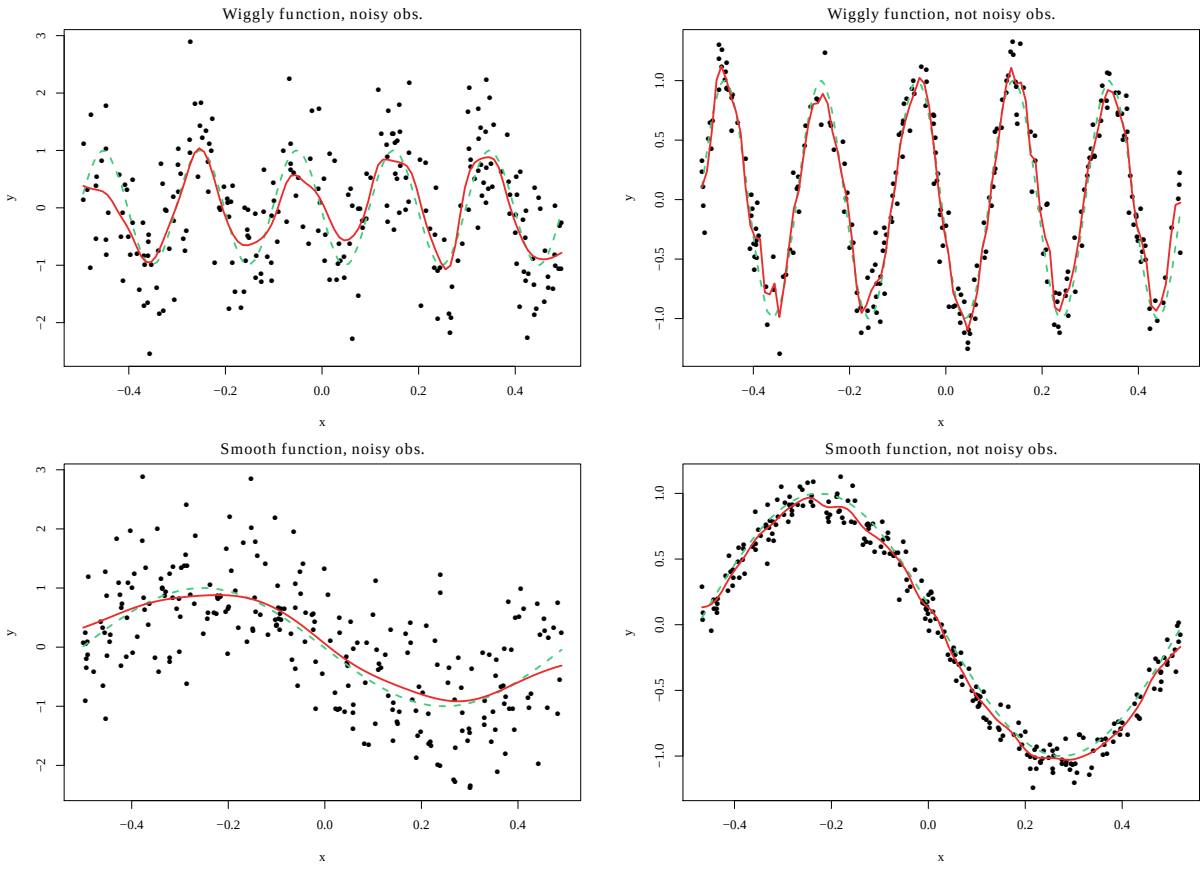


Figure 3: Four different smoothing problems. The smoothed version of the data is represented in red solid line, the "true" density generating the data is overlapped in green dashed line. The wiggly function is $f(x) = \sin(10\pi x)$, the smooth one is $g(x) = \sin(2\pi x)$. The noisy observations have a standard deviation $\sigma_a = 0.75$, the non noisy ones $\sigma_b = 0.25$.

(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*

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

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. 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. 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,

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

Two drawbacks of splitting the dataset in two chunks is surely the fact of not using all of the data to fit the model. In situations where we can't afford to have many observations, we might not want to do that. Moreover, the h chosen via this method leads to low bias for that particular partition of training/test set but high variance for other splits!

Let us now prove the LOOCV lemma: let us start by the assumption that a smoother maps constants into constants, that is $H\mathbf{1} = \mathbf{1}$. This implies that

$$\hat{y}_i^{(-i)} = \frac{\sum_{j \neq i} h_{ij} y_j}{1 - h_{ii}}.$$

By reshuffling the terms we get

$$\begin{aligned}
&\sum_{j \neq i} h_{ij} y_j = \hat{y}_i^{(-i)} - h_{ii}\hat{y}_i^{(-i)} \\
\Rightarrow &\sum_{j=1}^{n} h_{ij} y_j = \hat{y}_i^{(-i)} - h_{ii}\hat{y}_i^{(-i)} + h_{ii} y_i \\
\Rightarrow &\hat{y}_i = \hat{y}_i^{(-i)} - h_{ii}\hat{y}_i^{(-i)} + h_{ii} y_i \\
\Rightarrow &\hat{y}_i^{(-i)} = \frac{\hat{y}_i - h_{ii} y_i}{1 - h_{ii}}.
\end{aligned}$$

Therefore,

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

Problem 4. 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 $i = 1, \dots, n$ 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 quantity

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

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.

We can prove this by simply solving the minimization problem, that is

$$\hat{a} = \operatorname*{argmin}_{\mathbb{R}} \sum_{i=1}^{n} \frac{1}{h} K\left(\frac{x_i - x}{h}\right) (y_i - a)^2,$$

which is equivalent to, taking the derivative with respect to a and setting it equal to 0, to

$$\begin{aligned} \sum_{i=1}^{n} \frac{1}{h} K\left(\frac{x_i - x}{h}\right) 2(y_i - \hat{a}) &= 0 \\ \Rightarrow \hat{a} = \hat{f}(x) = \frac{\sum_{i=1}^{n} K\left(\frac{x_i - x}{h}\right) y_i}{K\left(\frac{x_i - x}{h}\right)} &= \sum_{i=1}^{n} w^\star(x, x_i) y_i, \end{aligned}$$

where $w^\star(x, x_i) = \frac{K\left(\frac{x_i - x}{h}\right)}{\sum_{i=1}^{n} K\left(\frac{x_i - x}{h}\right)}$ are the normalized weights.

(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 $\boldsymbol{a} = (a_0, \dots, a_D)$. As above, we will estimate the coefficients $\boldsymbol{a}$ in $g_x(u; \boldsymbol{a})$ at some target point x using weighted least squares:

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

where $w_i \equiv w(x_i, x)$ are the kernel weights defined just above, normalized to sum to one. Derive a concise (matrix) form of the weight vector $\hat{\boldsymbol{a}}$, and by extension, the local function estimate $\hat{f}(x)$ at

the target value x. 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.

The problem can be rephrased as a weighted least squares problem in which we are trying to estimate the $(D+1)$-dimensional vector $\hat{\boldsymbol{a}}_x$, so that $g_x(x_1, \dots, x_n; \hat{\boldsymbol{a}}) = \hat{f}(x)$.

Let us define the matrix R_x, so that $[R_x]_{ij} = (x_j - x)^{j-1}$, i.e. the $n \times (D+1)$ matrix

$$R_x = \begin{pmatrix} 1 & x_1 - x & \dots & (x_1 - x)^D \\ 1 & x_2 - x & \dots & (x_2 - x)^D \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_n - x & \dots & (x_n - x)^D \end{pmatrix}.$$

Then we can rewrite the minimization problem as

$$\begin{aligned} \hat{\boldsymbol{a}}_x &= \underset{\mathbb{R}^{D+1}}{\operatorname{argmin}} \sum_{i=1}^{n} w_i \{y_i - g_x(x_i; \boldsymbol{a})\}^2 \\ &= \underset{\mathbb{R}^{D+1}}{\operatorname{argmin}} \sum_{i=1}^{n} w_i \{y_i - a_0 - \sum_{k=1}^{D} a_j (x_i - x)^k\}^2 \\ &= \underset{\mathbb{R}^{D+1}}{\operatorname{argmin}} (\boldsymbol{Y} - R_x \boldsymbol{a})^T W (\boldsymbol{Y} - R_x \boldsymbol{a}). \end{aligned}$$

This problem is analogous to the weighted least squares we have already analysed several times, thus the solution is

$$\hat{\boldsymbol{a}}_x = (R_x^T W R_x)^{-1} R_x^T W \boldsymbol{Y},$$

and the corresponding estimated function at the target point x is

$$\hat{f}(x) = g_x(x; \hat{\boldsymbol{a}}_x) = \hat{a}_{x0} = \boldsymbol{e}_1^T \hat{\boldsymbol{a}}_x,$$

where $\boldsymbol{e}_1 = (1, 0, \dots, 0)^T$. Therefore, for each target point x we have to solve a WLS problem, and the corresponding smoothed fitted value is $\hat{y} = Hy$ where H is the first row of the matrix $(R_x^T W R_x)^{-1} R_x^T W$.

(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

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

We can explicitly write the solution of the WLS problem in the case $D = 1$. In fact, let us denote for brevity $k_i = K(\frac{x-x_i}{h})$. Then we have

$$\begin{aligned}
\hat{\boldsymbol{a}}_x &= \left[\begin{pmatrix} 1 & \dots & 1 \\ x_1 - x & \dots & x_n - x \end{pmatrix} \begin{pmatrix} k_1 & & \\ & \ddots & \\ & & k_n \end{pmatrix} \begin{pmatrix} 1 & x_1 - x \\ \vdots & \vdots \\ 1 & x_n - x \end{pmatrix} \right]^{-1} \\
&\quad \cdot \begin{pmatrix} 1 & \dots & 1 \\ x_1 - x & \dots & x_n - x \end{pmatrix} \begin{pmatrix} k_1 & & \\ & \ddots & \\ & & k_n \end{pmatrix} \begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix} \\
&= \begin{pmatrix} \sum_{i=1}^n k_i & \sum_{i=1}^n k_i(x_i - x) \\ \sum_{i=1}^n k_i(x_i - x) & \sum_{i=1}^n k_i(x_i - x)^2 \end{pmatrix}^{-1} \begin{pmatrix} k_1 & \dots & k_n \\ k_1(x_1 - x) & \dots & k_n(x_n - x) \end{pmatrix} \begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix} \\
&= \frac{1}{s_0(x)s_2(x) - s_1(x)^2} \begin{pmatrix} s_2(x) & -s_1(x) \\ -s_1(x) & s_0(x) \end{pmatrix} \begin{pmatrix} k_1 & \dots & k_n \\ k_1(x_1 - x) & \dots & k_n(x_n - x) \end{pmatrix} \begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix}
\end{aligned}$$

and therefore, considering only the first element of $\hat{\boldsymbol{a}}_x$,

$$\begin{aligned}
\hat{a}_{x0} &= \frac{1}{s_0(x)s_2(x) - s_1(x)^2} \begin{pmatrix} k_1[s_2(x) - (x_1 - x)s_1(x)] & \dots & k_n[s_2(x) - (x_n - x)s_1(x)] \end{pmatrix} \begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix} \\
&= \frac{\sum_{i=1}^n w_i(x) y_i}{\sum_{i=1}^n w_i(x)},
\end{aligned}$$

where $w_i(x) = k_i[s_2(x) - (x_i - x)s_1(x)]$, which is the desired result.

(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.*

Recall that $\hat{\boldsymbol{a}}_x = (R_x^T W R_x)^{-1} R_x^T W \boldsymbol{Y}$. Thus,

$$\begin{aligned}
\mathrm{E}[\hat{a}_{x0}] &= \boldsymbol{e}_1^T (R_x^T W R_x)^{-1} R_x^T W \cdot \mathrm{E}[\boldsymbol{Y}] \\
&= \boldsymbol{e}_1^T \underbrace{(R_x^T W R_x)^{-1} R_x^T W}_{H} \boldsymbol{f}(\boldsymbol{x}).
\end{aligned}$$

Thus,

$$\mathrm{E}[\hat{f}(x_i)] = \sum_{j=1}^n H_{ij} f(x_j)$$

This means that the estimator is unbiased only if $H = \mathcal{I}$, which never happens. In fact, the estimator is a smoothed version of the underlying function. It is therefore biased because the corresponding unbiased estimator would be just $\boldsymbol{y}$ (connect the dots without smoothing).

In the same fashion, we find that

$$\text{Var}[\hat{f}(x_i)] = \sum_{j=1}^{n} H_{ij}^2 \sigma^2$$

(D) *We don't know the residual variance, but we can estimate it. A basic fact is that if $\boldsymbol{X}$ is a random vector with mean $\boldsymbol{\mu}$ and covariance matrix Σ, then for any symmetric matrix Q of appropriate dimension, the quadratic form $\boldsymbol{X}^T Q \boldsymbol{X}$ has expectation*

$$E(\boldsymbol{X}^T Q \boldsymbol{X}) = tr(Q\Sigma) + \boldsymbol{\mu}^T Q \boldsymbol{\mu}\,.$$

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

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

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

Let us start by expanding out the term at the numerator. Since $\text{E}[\boldsymbol{y}] = f(\boldsymbol{x})$ and $\text{Cov}[\boldsymbol{y}] = \sigma^2 \mathcal{I}_n$, we can write

$$\begin{aligned}
\text{E}[||\boldsymbol{r}||_2^2] &= \text{E}[(\boldsymbol{y} - H\boldsymbol{y})^T (\boldsymbol{y} - H\boldsymbol{y})] \\
&= \text{E}[\boldsymbol{y}^Y \boldsymbol{y} + \boldsymbol{y}^T H^T H \boldsymbol{y} - 2\boldsymbol{y}^T H^T \boldsymbol{y}] \\
&= tr(\mathcal{I}_n \sigma^2 \mathcal{I}_n) + f(\boldsymbol{x})^T \mathcal{I}_n f(\boldsymbol{x}) + tr(H^T H \sigma^2 \mathcal{I}_n) + f(\boldsymbol{x})^T H^T H f(\boldsymbol{x}) \\
&\quad - 2\left[tr(H^T \sigma^2 \mathcal{I}_n) + f(\boldsymbol{x})^T H^T f(\boldsymbol{x})\right] \\
&= \sigma^2 \left[n + tr(H^T H) - 2tr(H)\right] + \left[f(\boldsymbol{x}) - Hf(\boldsymbol{x})\right]^T \left[f(\boldsymbol{x}) - Hf(\boldsymbol{x})\right].
\end{aligned}$$

Therefore

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

which is nearly unbiased for large n if $f(\boldsymbol{x}) \approx Hf(\boldsymbol{x})$, that is, if the "true" and unknown function $f(x)$ generating the data is similar to its smoothed version, which implies that it is regular enough.

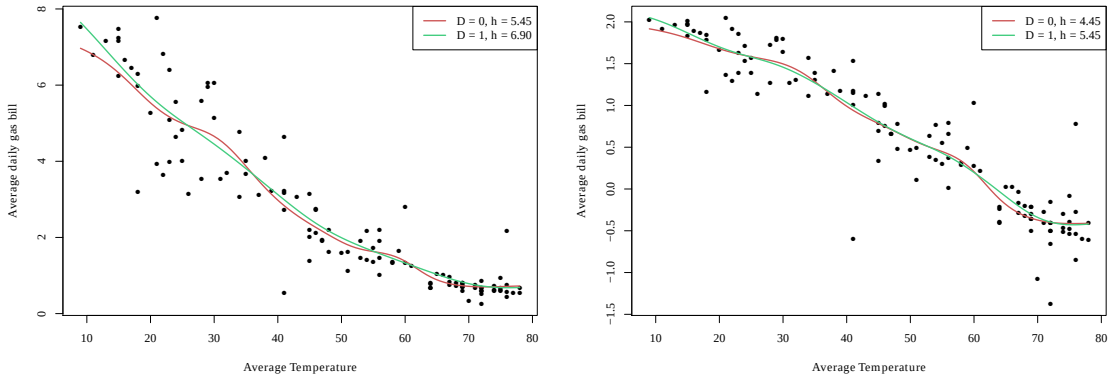
In the case of ordinary least squares, since H is a perpendicular projection matrix, we know that $H = H^T H$ and $tr(H) = p$. Thus,

$$\text{E}[\hat{\sigma}^2] = \sigma^2 + \frac{\left[f(\boldsymbol{x}) - Hf(\boldsymbol{x})\right]^T \left[f(\boldsymbol{x}) - Hf(\boldsymbol{x})\right]}{n - p}.$$

In general, $n - tr(H^T H) - 2tr(H)$ is the effective degrees of freedom. In general, p is not growing as fast as n (the only case when it does is when we are not doing smoothing). In fact, the more we smooth, the less degrees of freedom we are selecting in the model

(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.*

The R function is illustrated in Listing A.1. It has been implemented in the general case, and the user can specify the choice of the polynomial degree D. In the particular cases $D = 0$ and $D = 1$, we obtained the results displayed in Figure 4. As one can see, the local linear smoother seems to outperform the local constant smoother on the edges, where the latter suffers of the edge bias problem previously discussed.



(a) Original data *(b) Log-transformed data*

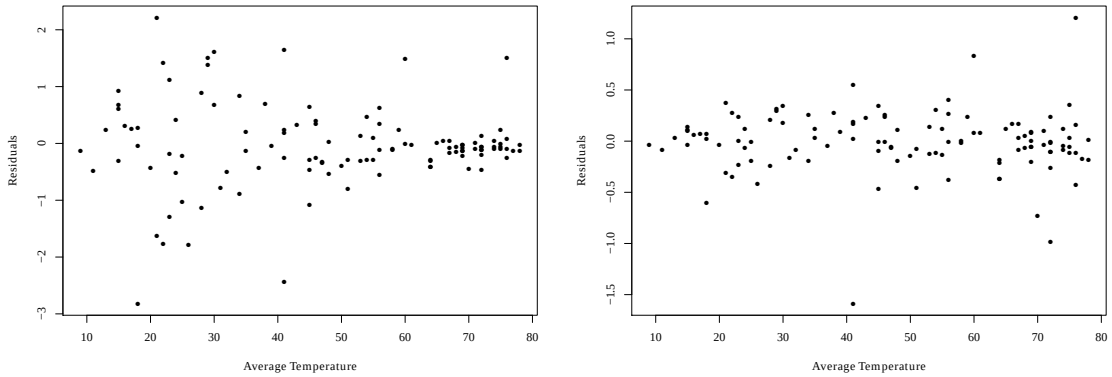
Figure 4

(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?*

No, as one can see in Figure 5a, the residuals do not seem to have the same variance. A variance stabilizing transformation, such as the logarithm, seems to improve the behaviour of the residuals, see Figure 5b.

In alternative, we could model the variance since the observations are clearly more noisy on the left side of the plot. To do this, we can compute the empirical residuals $\hat{\epsilon}_i = y_i - \hat{f}(x_i)$ and model their variance, i.e. $\text{Var}(\epsilon_i) = \text{E}[\epsilon_i^2]$. One can choose to model $E[\epsilon_i^2] = \exp\{g(x)\}$, that is, to fit a linear smoother to $\log \epsilon_i^2$ using the observed residuals $\hat{\epsilon}_i$.

(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.*



(a) Original data *(b) Log-transformed data*

Figure 5: Residuals of the model applied to both the original data and the log-transformed data.

Using the estimated variance in part (D), we can build pointwise confidence intervals for the smoothed curve. In fact, we can use

$$\hat{y} \pm \hat{\sigma} \cdot z_{0.975}.$$

In Figure 6 the result of this procedure is illustrated.

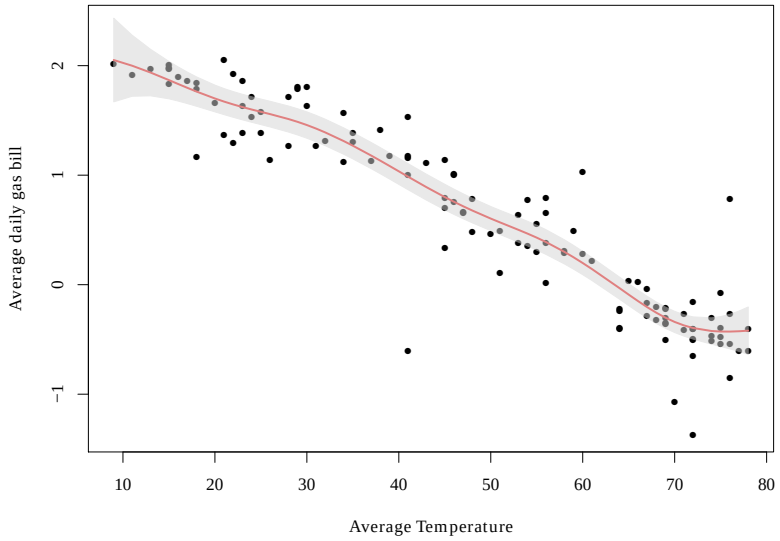


Figure 6: Local linear smoother for the utilities data. The optimal value of the bandwidth has been chosen via LOOCV. The gray shaded area represents the 95% confidence intervals.

Problem 5. Gaussian processes

A Gaussian process is a collection of random variables $\{f(x) : x \in \mathcal{X}\}$ such that, for any finite collection of indices $x_1, \ldots, x_N \in \mathcal{X}$, the random vector $[f(x_1), \ldots, f(x_N)]^T$ has a multivariate normal distribution. It is a generalization of the multivariate normal distribution to infinite-dimensional spaces. The set $\mathcal{X}$ is called the index set or the state space of the process, and need not be countable.

A Gaussian process can be thought of as a random function defined over $\mathcal{X}$, often the real line or $\mathbb{R}^p$. We write $f \sim GP(m, C)$ for some mean function $m : \mathcal{X} \to \mathbb{R}$ and a covariance function $C : \mathcal{X} \times \mathcal{X} \to \mathbb{R}^+$. These functions define the moments of all finite-dimensional marginals of the process, in the sense that

$$E\{f(x_1)\} = m(x_1) \quad \textit{and} \quad cov\{f(x_1), f(x_2)\} = C(x_1, x_2)$$

for all $x_1, x_2 \in \mathcal{X}$. More generally, the random vector $[f(x_1), \ldots, f(x_N)]^T$ has covariance matrix whose (i, j) element is $C(x_i, x_j)$. Typical covariance functions are those that decay as a function of increasing distance between points x_1 and x_2. The notion is that $f(x_1)$ and $f(x_2)$ will have high covariance when x_1 and x_2 are close to each other.

(A) *Read up on the Matern class of covariance functions. The Matern class has the squared exponential covariance function as a special case:*

$$C_{SE}(x_1, x_2) = \tau_1^2 \exp\left\{-\frac{1}{2}\left(\frac{d(x_1, x_2)}{b}\right)^2\right\} + \tau_2^2 \delta(x_1, x_2)\,,$$

where $d(x_1, x_2) = \|x_1 - x_2\|_2$ is Euclidean distance (or just $|x - y|$ for scalars). The constants (b, τ_1^2, τ_2^2) are often called hyperparameters, and $\delta(a, b)$ is the Kronecker delta function that takes the value 1 if $a = b$, and 0 otherwise. But usually this covariance function generates functions that are "too smooth," and so we use other covariance functions in the Matern class as a default.

Let's start with the simple case where $\mathcal{X} = [0, 1]$, the unit interval. Write a function that simulates a mean-zero Gaussian process on $[0, 1]$ under the squared exponential covariance function. The function will accept as arguments: (1) finite set of points $x_1, \ldots, x_N$ on the unit interval; and (2) a triplet (b, τ_1^2, τ_2^2). It will return the value of the random process at each point: $f(x_1), \ldots, f(x_N)$.

Use your function to simulate (and plot) Gaussian processes across a range of values for b, τ_1^2, and τ_2^2. Try starting with a very small value of τ_2^2 (say, 10^{-6}) and playing around with the other two first. On the basis of your experiments, describe the role of these three hyperparameters in controlling the overall behavior of the random functions that result. What happens when you try $\tau_2^2 = 0$? Why? If you can fix this, do—remember our earlier discussion on different ways to simulate the MVN.

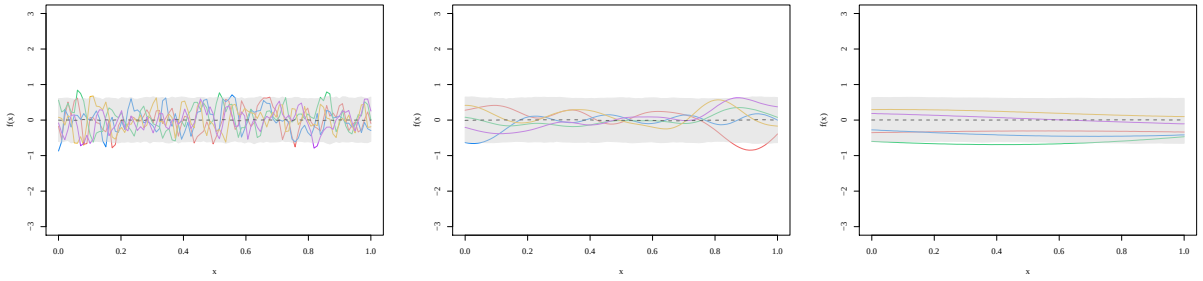
Now simulating a few functions with a different covariance function, the Matérn with parameter $5/2$:

$$C_{M52}(x_1, x_2) = \tau_1^2 \left\{1 + \frac{\sqrt{5}d}{b} + \frac{5d^2}{3b^2}\right\} \exp\left(\frac{-\sqrt{5}d}{b}\right) + \tau_2^2 \delta(x_1, x_2)\,,$$

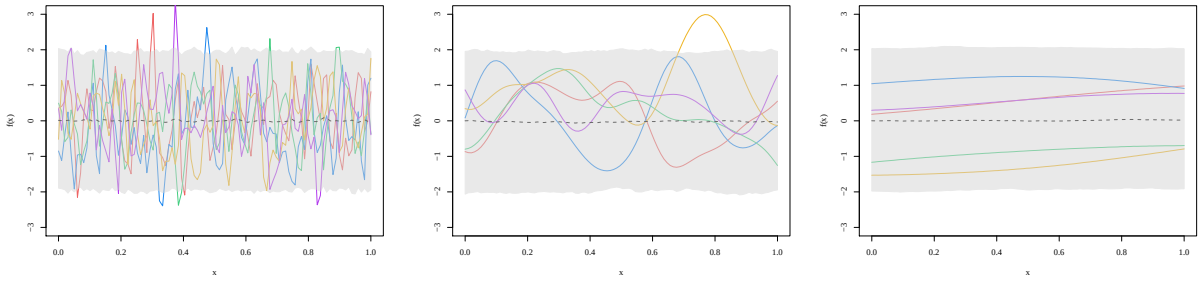
where $d = \|x_1 - x_2\|_2$ is the distance between the two points x_1 and x_2. Comment on the differences between the functions generated from the two covariance kernels.

The role of the parameters for the two covariance function families is the following:

- b controls how much distant points are correlated. Small values of b produce a non correlated process (more noisy), large values of b produce a process with high correlation and, subsequently, smoother functions;
- τ_1^2 is the scaling factor that controls the amplitude of the covariance matrix. Larger values will amplify the values of the process;
- τ_2^2 is an additive scaling factor that controls the amplitude of the variances, that is, how noisy each point is.



(a) $b = 0.01, \tau_1^2 = 0.1, \tau_2^2 = 10^{-6}$ *(b)* $b = 0.1, \tau_1^2 = 0.1, \tau_2^2 = 10^{-6}$ *(c)* $b = 1, \tau_1^2 = 0.1, \tau_2^2 = 10^{-6}$



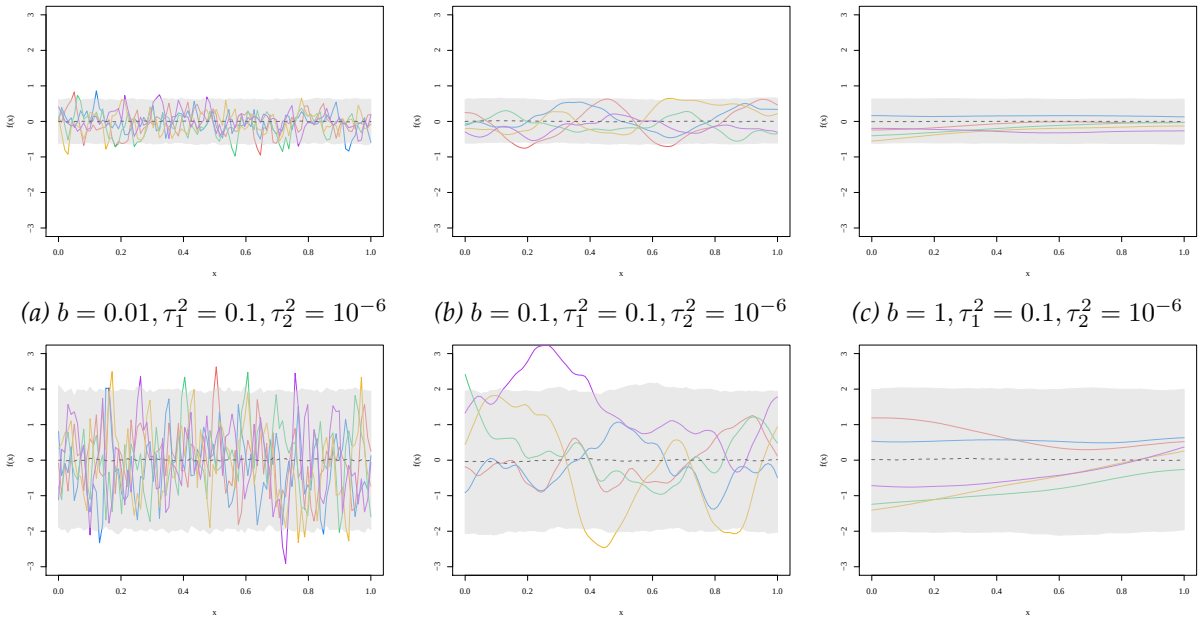
(d) $b = 0.01, \tau_1^2 = 1, \tau_2^2 = 10^{-6}$ *(e)* $b = 0.1, \tau_1^2 = 1, \tau_2^2 = 10^{-6}$ *(f)* $b = 1, \tau_1^2 = 1, \tau_2^2 = 10^{-6}$

Figure 7: Sample of 5 *functions from a Gaussian process with squared exponential covariance function. The mean* $m(x) = 0$ *is represented in dashed black line, the* 95% *credible interval is shadowed in gray.*

As one can remark in Figure 7 and Figure 8, the functions sampled with Matern covariance function tend to be less smooth than the ones with squared exponential covariance function. We also show the 95% credible intervals for the realization from the Gaussian process.

(B) *Suppose you observe the value of a Gaussian process* $f \sim GP(m, C)$ *at points* $x_1, \dots, x_N$. *What is the conditional distribution of the value of the process at some new point* $x^\star$? *For the sake of notational ease simply write the value of the* (i, j) *element of the covariance matrix as* $C_{i,j}$, *rather than expanding it in terms of a specific covariance function.*

Let us denote with $p(f(x^\star)|x^\star, \boldsymbol{x}, \boldsymbol{f}(\boldsymbol{x}))$ the requested predictive distribution for the value of the gaussian process at a new point.



(a) $b = 0.01, \tau_1^2 = 0.1, \tau_2^2 = 10^{-6}$ *(b)* $b = 0.1, \tau_1^2 = 0.1, \tau_2^2 = 10^{-6}$ *(c)* $b = 1, \tau_1^2 = 0.1, \tau_2^2 = 10^{-6}$

(d) $b = 0.01, \tau_1^2 = 1, \tau_2^2 = 10^{-6}$ *(e)* $b = 0.1, \tau_1^2 = 1, \tau_2^2 = 10^{-6}$ *(f)* $b = 1, \tau_1^2 = 1, \tau_2^2 = 10^{-6}$

Figure 8: Sample of 5 *functions from a Gaussian process with Matern* $5/2$ *exponential covariance function. The mean* $m(x) = 0$ *is represented in dashed black line, the* 95% *credible interval is shadowed in gray.*

We know that, by definition of Gaussian process,

$$\begin{pmatrix} f(x^\star) \\ f(x_1) \\ \vdots \\ f(x_N) \end{pmatrix} \sim N\left[\begin{pmatrix} m(x^\star) \\ m(x_1) \\ \vdots \\ m(x_N) \end{pmatrix}; \begin{pmatrix} C(x^\star, x^\star) & C(x_1, x^\star) & \dots & C(x_N, x^\star) \\ C(x^\star, x_1) & C(x_1, x_1) & \dots & C(x_1, x_N) \\ \vdots & \vdots & \ddots & \vdots \\ C(x^\star, x_N) & C(x_N, x_1) & \dots & C(x_N, x_N) \end{pmatrix} \right].$$

Since we are in a multivariate setting, we can now use the results of Homework 1 about the conditional distributions of the multivariate Normal distributions. In fact, we can write

$$f(x^\star)|f(x_1), \dots, f(x_N) \sim N(m(x^\star) + \Sigma_{12}\Sigma_{22}^{-1}[\boldsymbol{f}(\boldsymbol{x}) - \boldsymbol{m}(\boldsymbol{x})]; C(x^\star, x^\star) - \Sigma_{12}\Sigma_{22}^{-1}\Sigma_{21})$$

where

$$\Sigma_{12} = (C(x^\star, x_1), \dots, C(x^\star, x_N))$$
$$\Sigma_{22} = \begin{pmatrix} C(x_1, x_1) & \dots & C(x_1, x_N) \\ \vdots & \ddots & \vdots \\ C(x_N, x_1) & \dots & C(x_N, x_N) \end{pmatrix}$$

(C) *Prove the following lemma.*

Lemma 1. *Suppose that the joint distribution of two vectors* $\boldsymbol{y}$ *and* $\boldsymbol{\theta}$ *has the following properties: (1) the conditional distribution for* $\boldsymbol{y}$ *given* $\boldsymbol{\theta}$ *is multivariate normal,* $(\boldsymbol{y} \mid \boldsymbol{\theta}) \sim N(R\boldsymbol{\theta}, \Sigma)$*; and (2)*

the marginal distribution of $\boldsymbol{\theta}$ is multivariate normal, $\boldsymbol{\theta} \sim N(\boldsymbol{m}, V)$. Assume that R, Σ, $\boldsymbol{m}$, and V are all constants. Then the joint distribution of $\boldsymbol{y}$ and $\boldsymbol{\theta}$ is multivariate normal.

The random vector $\boldsymbol{Y}$ can be written as $\boldsymbol{Y} = R\boldsymbol{\theta} + \boldsymbol{\epsilon}$, where

$$\begin{aligned}\boldsymbol{\epsilon} &\sim N(\mathbf{0}, \Sigma)\\ \boldsymbol{\theta} &\sim N(\boldsymbol{m}, V).\end{aligned}$$

Therefore we can write the joint distribution

$$\begin{pmatrix} \boldsymbol{Y} \\ \boldsymbol{\theta} \end{pmatrix} = \begin{pmatrix} R \\ \mathcal{I} \end{pmatrix} \cdot \boldsymbol{\theta} + \begin{pmatrix} \mathcal{I} \\ \mathcal{O} \end{pmatrix} \cdot \boldsymbol{\epsilon}$$

and we can find the conditionals by simply using the theorems proved in the first set of exercises, obtaining

$$\begin{pmatrix} \boldsymbol{Y} \\ \boldsymbol{\theta} \end{pmatrix} \sim \mathcal{N}_{n+p}\left(\begin{bmatrix} R\boldsymbol{m} \\ \boldsymbol{m} \end{bmatrix}, \begin{bmatrix} \Omega & -\Omega R \\ -R^T\Omega & W + R^T\Omega R \end{bmatrix}^{-1} \right)$$

Problem 6. In nonparametric regression and spatial smoothing

(A) *Suppose we observe data $y_i = f(x_i) + \epsilon_i$, $\epsilon_i \sim N(0, \sigma^2)$, for some unknown function f. Suppose that the prior distribution for the unknown function is a mean-zero Gaussian process: $f \sim GP(0, C)$ for some covariance function C. Let $x_1, \dots, x_N$ denote the previously observed x points. Derive the posterior distribution for the random vector $[f(x_1), \dots, f(x_N)]^T$, given the corresponding outcomes $y_1, \dots, y_N$, assuming that you know σ^2.*

If we observe the data $\{(x_1, y_1); \dots; (x_n, y_n)\}$ where $y_i = f(x_i) + \epsilon_i$, $\epsilon_i \sim N(0, \sigma^2)$ and we put a Gaussian process as a prior for f, we have the model

$$\begin{aligned}
&\boldsymbol{Y} | \boldsymbol{f}(\boldsymbol{x}), \boldsymbol{x}, \sigma^2 \sim N_n(\boldsymbol{f}(\boldsymbol{x}); \sigma^2 \mathcal{I}) \\
&f \sim GP(0; C), \quad \text{i.e. } \boldsymbol{f}(\boldsymbol{x}) \sim N_n(\boldsymbol{0}, C(\boldsymbol{x}, \boldsymbol{x})).
\end{aligned}$$

In this case, we can find the posterior for the random vector $(f(x_1), \dots, f(x_n))$ which we will denote by $\boldsymbol{f}$, i.e.

$$\begin{aligned}
p(\boldsymbol{f} | \boldsymbol{y}, \boldsymbol{x}, \sigma^2) &\propto p(\boldsymbol{y} | \boldsymbol{f}, \boldsymbol{x}, \sigma^2) \cdot p(\boldsymbol{f} | \boldsymbol{x}, \sigma^2) \\
&= \left(\frac{1}{2\pi\sigma^2} \right)^{\frac{n}{2}} \exp\left\{ -\frac{1}{2} (\boldsymbol{y} - \boldsymbol{f})^T \frac{\mathcal{I}}{\sigma^2} (\boldsymbol{y} - \boldsymbol{f}) \right\} \\
&\quad \cdot \left(\frac{1}{2\pi} \right)^{\frac{n}{2}} |C|^{-\frac{1}{2}} \exp\left\{ -\frac{1}{2} \boldsymbol{f}^T C^{-1} \boldsymbol{f} \right\} \\
&\propto \exp\left\{ -\frac{1}{2} \left[(\boldsymbol{y} - \boldsymbol{f})^T \frac{\mathcal{I}}{\sigma^2} (\boldsymbol{y} - \boldsymbol{f}) + \boldsymbol{f}^T C^{-1} \boldsymbol{f} \right] \right\}.
\end{aligned}$$

We can now use the completing the square trick, i.e.

$$\begin{aligned}
&\frac{1}{2} \boldsymbol{y}^T \frac{\mathcal{I}}{\sigma^2} \boldsymbol{y} + \frac{1}{2} \boldsymbol{f}^T \frac{\mathcal{I}}{\sigma^2} \boldsymbol{f} - \boldsymbol{y}^T \frac{\mathcal{I}}{\sigma^2} \boldsymbol{f} + \boldsymbol{f}^T C^{-1} \boldsymbol{f} \\
=&\frac{1}{2} \boldsymbol{f}^T \left[\frac{\mathcal{I}}{\sigma^2} + C^{-1} \right] \boldsymbol{f} - \boldsymbol{y}^T \frac{\mathcal{I}}{\sigma^2} \boldsymbol{f} + \frac{1}{2} \boldsymbol{y}^T \frac{\mathcal{I}}{\sigma^2} \boldsymbol{y} \\
\propto&\frac{1}{2} (\boldsymbol{f} - \boldsymbol{m}^\star)^T C^{\star -1} (\boldsymbol{f} - \boldsymbol{m}^\star),
\end{aligned}$$

where

$$\begin{aligned}
C^{\star -1} &= C^{-1} + \frac{\mathcal{I}}{\sigma^2} \\
\boldsymbol{m}^\star &= C^\star \frac{\mathcal{I}}{\sigma^2} \boldsymbol{y}.
\end{aligned}$$

Therefore,

$$f(x_1), \dots, f(x_n) | \boldsymbol{y}, \boldsymbol{x}, \sigma^2 \sim N(\boldsymbol{m}^\star, C^\star).$$

(B) *As before, suppose we observe data $y_i = f(x_i) + \epsilon_i$, $\epsilon_i \sim N(0, \sigma^2)$, for $i = 1, \dots, N$. Now we wish to predict the value of the function $f(x^\star)$ at some new point $x^\star$ where we haven't seen previous data. Suppose that f has a mean-zero Gaussian process prior, $f \sim GP(0, C)$. Show that the posterior mean*

$E\{f(x^\star) \mid y_1, \dots, y_N\}$ is a linear smoother, and derive expressions both for the smoothing weights and the posterior variance of $f(x^\star)$.

We can replicate the same steps of part (B) of the previous problem in the particular case $m = 0$, and reminding that now we do not observe the true values of the function $(f(x_1), \dots, f(x_n))$ but only its noisy counterparts $y_i = f(x_i) + \epsilon_i$, where $\epsilon_i \overset{\text{iid}}{\sim} N(0, \sigma^2)$. Therefore

$$\begin{pmatrix} \boldsymbol{y} \\ f(x^\star) \end{pmatrix} \sim N\left[\begin{pmatrix} \mathbf{0} \\ 0 \end{pmatrix}; \begin{pmatrix} C(\boldsymbol{x}, \boldsymbol{x}) + \sigma^2 \mathcal{I} & \boldsymbol{C}(x^\star, \boldsymbol{x}) \\ \boldsymbol{C}(\boldsymbol{x}, x^\star) & C(x^\star, x^\star) \end{pmatrix} \right].$$

If we use the conditional rules for the normal distributions obtained in Homework 1, we get:

$$f(x^\star)|\boldsymbol{y} \sim N\Big(\boldsymbol{C}(\boldsymbol{x}, x^\star)[C(\boldsymbol{x}, \boldsymbol{x}) + \sigma^2\mathcal{I}]^{-1}\boldsymbol{y}; C(x^\star, x^\star) - \boldsymbol{C}(\boldsymbol{x}, x^\star)[C(\boldsymbol{x}, \boldsymbol{x}) + \sigma^2\mathcal{I}]^{-1}\boldsymbol{C}(x^\star, \boldsymbol{x})\Big)$$

The posterior mean can be written as

$$\mathrm{E}[f(x^\star)|x^\star, \boldsymbol{y}, \boldsymbol{x}, \sigma^2] = \sum_{i=1}^{n} \alpha_i y_i,$$

where

$$\boldsymbol{\alpha} = \boldsymbol{C}(\boldsymbol{x}, x^\star)[C(\boldsymbol{x}, \boldsymbol{x}) + \sigma^2\mathcal{I}]^{-1}.$$

Therefore, the posterior mean is a linear smoother. However, if we include uncertainty over the parameters τ_1^2, τ_2^2, b, the conditional posterior mean $\mathrm{E}[f(x^\star)|\boldsymbol{y}, \tau_1^2, \tau_2^2, b]$ is still linear but $\mathrm{E}[f(x^\star)|\boldsymbol{y}]$ is not a linear smoother anymore. In fact, the dependence of C over the parameters is not linear and therefore, when we integrate out, the linearity is lost.

(C) *Go back to the utilities data, and plot the pointwise posterior mean and 95% posterior confidence interval for the value of the function at each of the observed points x_i (again, superimposed on top of the scatter plot of the data itself). Choose τ_2^2 to be very small, say 10^{-6}, and choose (b, τ_1^2) that give a sensible-looking answer.*

The code implementing the Gaussian process for fitting a nonlinear curve is shown in Listing A.3.

In Figure 9, we show the posterior mean over a fine grid of points. As one can see, the result seems to overfit the data. We show in the next sections how to pick an optimal choice for the hyperparameters.

(D) *Let $y_i = f(x_i) + \epsilon_i$, and suppose that f has a Gaussian-process prior under the Matern(5/2) covariance function C with scale τ_1^2, range b, and nugget τ_2^2. Derive an expression for the marginal distribution of $y = (y_1 \dots, y_N)$ in terms of (τ_1^2, b, τ_2^2), integrating out the random function f. This is called a marginal likelihood.*

Given the following model

$$Y_i|f(x_i), x_i, \sigma^2 \overset{\text{ind}}{\sim} N(f(x_i), \sigma^2)$$
$$f \sim GP(0, C)$$

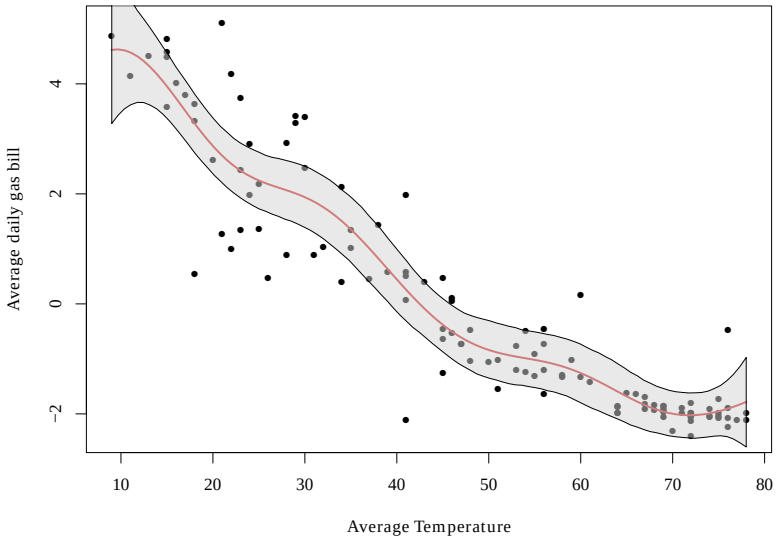


Figure 9: Fit of the utilities data with a Gaussian process, using a non optimal choice of the parameters. The pointwise 95% confidence interval is overlapped in the gray area.

we can compute the marginal likelihood by integrating out the random function $\boldsymbol{f}$, that is the function f evaluated at the points $x_1, \dots, x_N$. Pay attention, in the likelihood of the data we are not conditioning with respect to the random function f, but only to the function evaluated at the points $\boldsymbol{x}$, which we will denote with $\boldsymbol{f}$. Hence we have (the conditioning on σ^2 and $\boldsymbol{x}$ is implicit),

$$\begin{aligned}
p(\boldsymbol{y}) &= \int p(\boldsymbol{y}|\boldsymbol{f}) \cdot p(\boldsymbol{f})\, d\boldsymbol{f} \\
&= \int \left(\frac{1}{2\pi\sigma^2}\right)^{\frac{n}{2}} \exp\left\{-\frac{1}{2}(\boldsymbol{y}-\boldsymbol{f})^T \frac{\mathcal{I}}{\sigma^2}(\boldsymbol{y}-\boldsymbol{f})\right\} \\
&\quad \cdot \left(\frac{1}{2\pi}\right)^{\frac{n}{2}} |C|^{-\frac{1}{2}} \exp\left\{-\frac{1}{2}\boldsymbol{f}^T C^{-1}\boldsymbol{f}\right\}\, d\boldsymbol{f} \\
&= \int \exp\left\{-\frac{1}{2}(\boldsymbol{y}-\boldsymbol{f})^T \frac{\mathcal{I}}{\sigma^2}(\boldsymbol{y}-\boldsymbol{f})\right\} \exp\left\{-\frac{1}{2}\boldsymbol{f}^T C^{-1}\boldsymbol{f}\right\}\, d\boldsymbol{f}.
\end{aligned}$$

The exponent can be rewritten, with the usual trick, as

$$\begin{aligned}
&-\left[\frac{1}{2}\boldsymbol{y}^T\frac{\mathcal{I}}{\sigma^2}\boldsymbol{y} + \frac{1}{2}\boldsymbol{f}^T\frac{\mathcal{I}}{\sigma^2}\boldsymbol{f} - \boldsymbol{y}^T\frac{\mathcal{I}}{\sigma^2}\boldsymbol{f} + \frac{1}{2}\boldsymbol{f}^T C^{-1}\boldsymbol{f}\right] \\
=&-\left[\frac{1}{2}\boldsymbol{f}^T\left(\frac{\mathcal{I}}{\sigma^2}+C^{-1}\right)\boldsymbol{f} - \boldsymbol{y}^T\frac{\mathcal{I}}{\sigma^2}\boldsymbol{f} + \frac{1}{2}\boldsymbol{y}^T\frac{\mathcal{I}}{\sigma^2}\boldsymbol{y}\right] \\
=&-\left[\frac{1}{2}(\boldsymbol{f}-\boldsymbol{m}^\star)^T\left(\frac{\mathcal{I}}{\sigma^2}+C^{-1}\right)(\boldsymbol{f}-\boldsymbol{m}^\star) + v + \frac{1}{2}\boldsymbol{y}^T\frac{\mathcal{I}}{\sigma^2}\boldsymbol{y}\right],
\end{aligned}$$

where

$$\boldsymbol{m}^\star = \left(\frac{\mathcal{I}}{\sigma^2} + C^{-1}\right)^{-1} \frac{\mathcal{I}}{\sigma^2}\boldsymbol{y}$$
$$v = -\frac{1}{2}\boldsymbol{y}^T \frac{\mathcal{I}}{\sigma^2}\left(\frac{\mathcal{I}}{\sigma^2} + C^{-1}\right)^{-1} \frac{\mathcal{I}}{\sigma^2}\boldsymbol{y}.$$

Since the quadratic form (normal kernel) integrates to the proportionality constant that does not depend on $\boldsymbol{y}$, we are left with

$$\begin{aligned} p(\boldsymbol{y}) &\propto \exp\left\{-\frac{1}{2}\boldsymbol{y}^T\frac{\mathcal{I}}{\sigma^2}\boldsymbol{y} + \frac{1}{2}\boldsymbol{y}^T\frac{\mathcal{I}}{\sigma^2}\left(\frac{\mathcal{I}}{\sigma^2} + C^{-1}\right)^{-1}\frac{\mathcal{I}}{\sigma^2}\boldsymbol{y}\right\} \\ &= \exp\left\{-\frac{1}{2}\boldsymbol{y}^T\left[\frac{\mathcal{I}}{\sigma^2} - \frac{\mathcal{I}}{\sigma^2}\left(\frac{\mathcal{I}}{\sigma^2} + C^{-1}\right)^{-1}\frac{\mathcal{I}}{\sigma^2}\right]\boldsymbol{y}\right\} \\ &= \exp\left\{-\frac{1}{2}\boldsymbol{y}^T(\sigma^2\mathcal{I} + C)^{-1}\boldsymbol{y}\right\}, \end{aligned}$$

where in the last step we used the matrix inversion lemma, that states that

$$\left(A^{-1} + B^{-1}\right)^{-1} = A - A(A+B)^{-1}A.$$

Therefore, we recognize that the marginal likelihood of the data is

$$\boldsymbol{y}|\sigma^2 \sim N(\boldsymbol{0}; \sigma^2\mathcal{I} + C).$$

(E) *Return to the utilities or ethanol data sets. Fix $\tau_2^2 = 0$, and evaluate the log of the marginal likelihood function $p(y \mid \tau_1^2, b)$ over a discrete 2-d grid of points. If you're getting errors in your code with $\tau_2^2 = 0$, use something very small instead. Use this plot to choose a set of values $(\hat{\tau}_1^2, \hat{b})$ for the hyperparameters. Then use these hyperparameters to compute the posterior mean for f, given y. Comment on any lingering concerns you have with your fitted model.*

In Figure 10 the value of the marginal log-likelihood on a fine grid of points is shown. The optimal parameters setting is found in $(\hat{b}, \hat{\tau}_1^2) = (57.18, 23.21)$.

(F) *In `weather.csv` you will find data on two variables from 147 weather stations in the American Pacific northwest.*

- *pressure: the difference between the forecasted pressure and the actual pressure reading at that station (in Pascals)*
- *temperature: the difference between the forecasted temperature and the actual temperature reading at that station (in Celsius)*

There are also latitude and longitude coordinates of each station. Fit a Gaussian process model for each of the temperature and pressure variables. Choose hyperparameters appropriately. Visualize your fitted functions (both the posterior mean and posterior standard deviation) on a regular grid

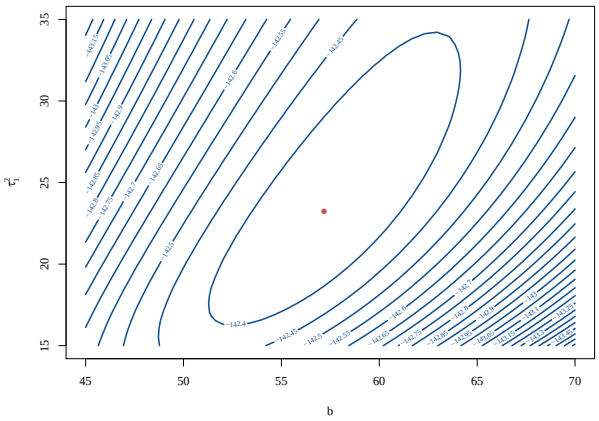


Figure 10: Marginal log-likelihood evaluated on a fine grid of points for the two parameters (b, τ_1^2). The optimal value is the red point.

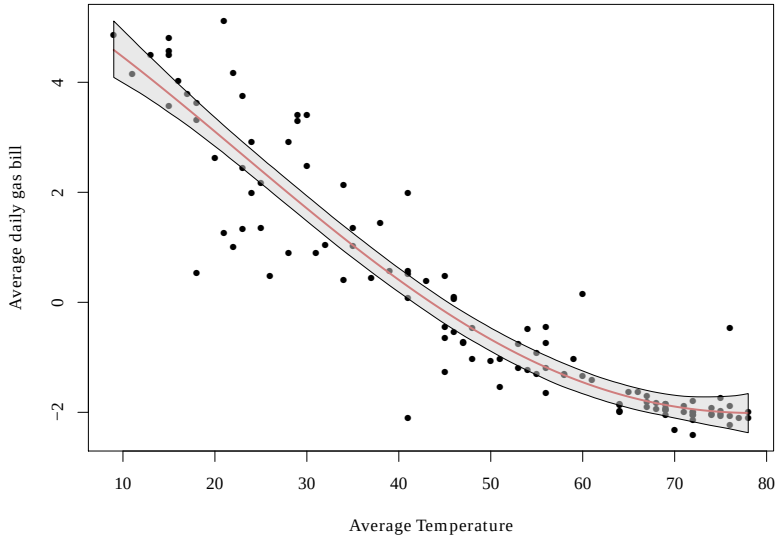


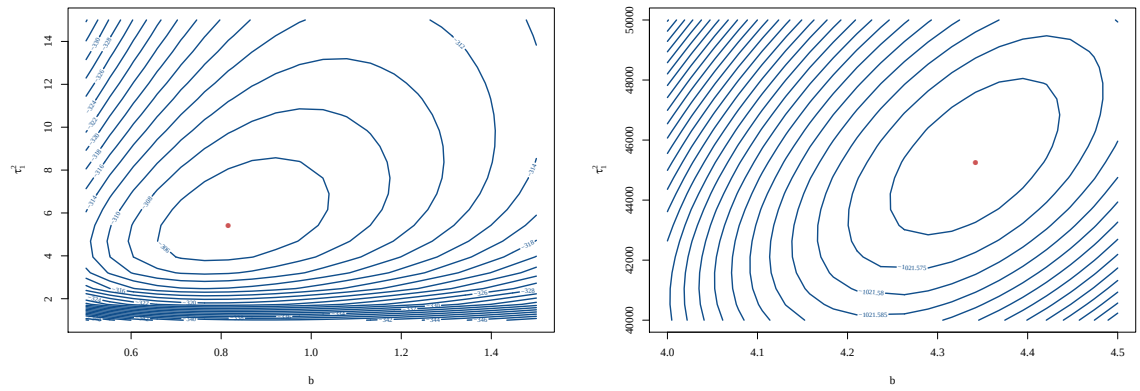
Figure 11: Fit of the utilities data with a Gaussian process, using the optimal choice of the parameters. The pointwise 95% confidence interval is overlapped in the gray area.

using something like a contour plot or color image. Read up on the `image`, `filled.contour`, *or* `contourplot` *functions in R. An important consideration: is Euclidean distance the appropriate measure to go into the covariance function? Or do we need separate length scales for the two dimensions, i.e.*

$$d^2(x, z) = \frac{(x_1 - z_1)^2}{b_1^2} + \frac{(x_2 - z_2)^2}{b_2^2}.$$

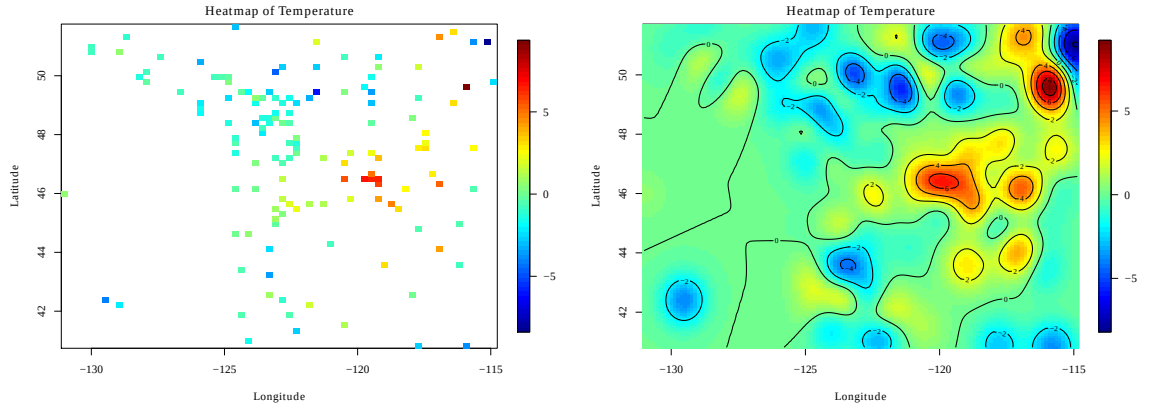
Justify your reasoning for using Euclidean distance or this "nonisotropic" distance.

I implemented both the isotropic and the anisotropic distance. For the sake of brevity, I display here the results obtained with the isotropic distance.



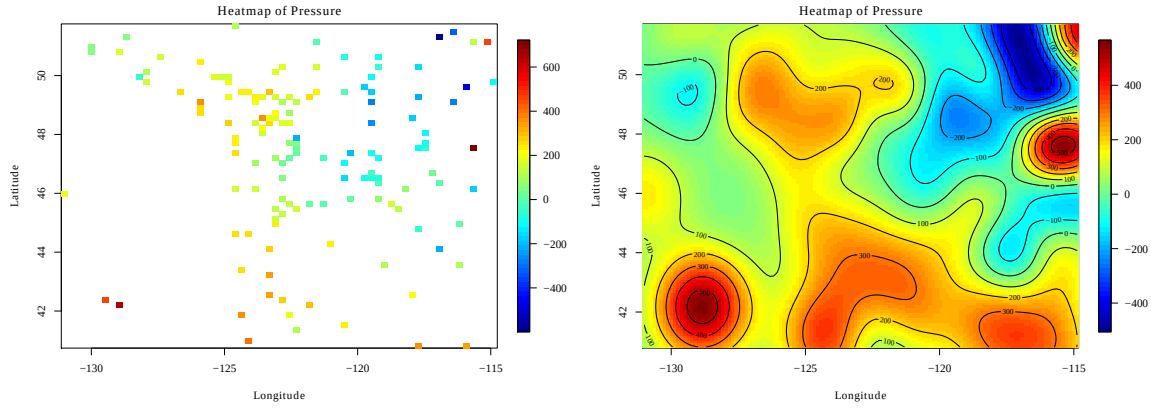
(a) Temperature data *(b) Pressure data*

Figure 12: Marginal log-likelihood as a function of the two parameters (b, τ_1^2) for the two different responses: temperature and pressure



(a) Original data *(b) Smoothed with parameters: $\hat{b} = 0.71; \hat{\tau}_1^2 = 6.38$*

Figure 13: Smoothing of the temperature data



(a) Original data *(b) Smoothed with parameters: $\hat{b} = 4.34; \hat{\tau}_1^2 = 45263$*

Figure 14: Smoothing of the pressure data

Appendix A

R code

```r
gauss.kernel <- function(x){
  # Computes the Gaussian kernel function
  # ------------------------------------------------------------------------------
  # Args:
  #   - x: distance between the two points, scaled by h
  # Returns:
  #   - K: value of the Gaussian kernel
  # ----------------------------------
  K <- (1/sqrt(2*pi)) * exp(-(1/2) * x^2)

  return (K)
}


unif.kernel <- function(x){
  # Computes the Uniform kernel function
  # ------------------------------------------------------------------------------
  # Args:
  #   - distance: distance between the two points, scaled by h
  # Returns:
  #   - K: value of the Gaussian kernel
  # ----------------------------------
  K <- (1/2) * as.numeric(abs(x) <= 1)

  return (K)
}


weights <- function(x, xnew, h, type = 'gaussian'){
  # Computes the weigths for the new observation xnew given the previous ones x
  # ------------------------------------------------------------------------------
  # Args:
  #   - x: vector containing the points
  #   - xnew: new point for which we compute the weights
  #   - h: bandwidth
  # Returns:
  #   - w: vector of weights for the new observation xnew
  # ----------------------------------------------------
  n <- length(x)
  w <- array(NA, dim = n)
```

```
41
42    if (type == 'gaussian'){
43      for (i in 1:n){
44        w[i] <- (1/h) * gauss.kernel((x[i] - xnew)/h)
45      }
46    }
47    else if (type == 'uniform'){
48      for (i in 1:n){
49        w[i] <- (1/h) * unif.kernel((x[i] - xnew)/h)
50      }
51    }
52
53    return (w)
54  }
55
56
57  linsmooth <- function(x, y, xnew, h, type = 'gaussian'){
58    # Computes the predicted values of the linear smoother for the values xnew
59    # ------------------------------------------------------------------------
60    # Args:
61    #   - x: vector of the data to fit the linear smoother
62    #   - y: response vector to fit the linear smoother
63    #   - xnew: grid on which we wish to evaluate the linear smoother
64    #   - h: bandwidth
65    # Returns:
66    #   - ynew: values of the linear smoother on the grid xnew
67    # ------------------------------------------------------------
68    ynew <- array(NA, dim = length(xnew))
69
70    for (i in 1:length(ynew)){
71      w <- weights(x, xnew[i], h, type)
72      if (sum(w) > 0)
73        w <- w/sum(w)
74      ynew[i] <- sum(w * y)
75    }
76
77    return(ynew)
78  }
79
80
81  cv.linsmooth <- function(x.train, y.train, x.test, y.test, hgrid){
82    # Computes the linear smoother functions for several values of the bandwidth h
83    # and the MSE for each case
84    # ------------------------------------------------------------------------------
85    # Args:
86    #   - x.train: vector of the data to fit the linear smoother
87    #   - y.train: response vector to fit the linear smoother
88    #   - x.test: vector of the test data to compute the MSE
89    #   - y.test: response vector of the test data to compute the MSE
90    #   - hgrid: grid of values for the bandwidth
91    # Returns:
92    #   - y.pred: values of the linear smoother on the grid x.test for each of the models
93    #   - MSE: MSE values for each one of the models
94    # ----------------------------------------------
95    k <- length(hgrid)
96
97    y.new <- array(NA, dim = c(k, length(y.test)))
```

```
  MSE <- array(NA, dim = k)
  for (i in 1:k){
    y.new[i,] <- linsmooth(x.train, y.train, x.test, hgrid[i])
    MSE[i] <- sum((y.new[i,] - y.test)^2)
  }

  return (list('y.pred' = y.new, 'MSE' = MSE))
}


polregr <- function(x, y, xnew, D, h, type = 'gaussian'){
  # Computes the predicted values of the linear polynomial regression for the
  # values xnew
  # ----------------------------------------------------------------------
  # Args:
  #   - x: vector of the data to fit the linear smoother
  #   - y: response vector to fit the linear smoother
  #   - xnew: grid on which we wish to evaluate the linear smoother
  #   - D: maximum degree of the polynomial
  #   - h: bandwidth
  # Returns:
  #   - ynew: values of the linear smoother on the grid xnew
  #   - H: the projection matrix, i.e. yhat = H %*% y
  # ----------------------------------------------------------
  n <- length(x)
  nstar <- length(xnew)

  # Initialization of the data structures
  ynew <- array(NA, dim = nstar)
  H <- array(NA, dim = c(nstar, n))
  Rx <- array(NA, dim = c(n, D+1))
  Rx[,1] <- rep(1, n) # Initialization of the R matrix
  for (i in 1:length(ynew)){ # loop through the observations to predict
    # Compute the weights
    W <- diag(weights(x, xnew[i], h, type))

    # We compute the Rx matrix for the target point xnew[i]
    if (D >= 1){
      Rx[,2:(D+1)] <- rep(x - xnew[i], D)
      for (d in 1:(D+1))
        Rx[,d] <- (Rx[,d])^(d-1)
    }

    # We solve the WLS problem
    RtW <- crossprod(Rx, W)        # precaching
    b <- solve(RtW %*% Rx) %*% RtW # solution of the WLS problem

    # Compute the projection matrix for the ith data point
    H[i,] <- b[1,]

    # Remember: at the target point, the estimated function is a_0^hat
    ynew[i] <- (b %*% y)[1]
  }
  return(list('ynew' = ynew, 'H' = H))
}


```

```r
cv.polregr <- function(x, y, D, hgrid, type = 'gaussian'){
  # Computes the LOOCV error for the linear polynomial regression for several values
  # of the bandwidth h
  # ------------------
  # Args:
  #   - x: vector of the data to fit the linear smoother
  #   - y: response vector to fit the linear smoother
  #   - D: maximum degree of the polynomial
  #   - hgrid: grid of values for the bandwidth
  # Returns:
  #   - LOOCV: leave-one-out cross validation error
  # ------------------------------------------------
  k <- length(hgrid)
  n <- length(x)
  H <- array(NA, dim = c(n, n))

  LOOCV <- array(NA, dim = k)
  for (h in 1:k){
    # We compute the H matrix for each possible value of the bandwidth
    H <- polregr(x, y, x, D, hgrid[h])$H

    # We use the LOOCV lemma
    LOOCV[h] <- sum((y - H %*% y)^2 / (1 - diag(H))^2)
  }

  return (LOOCV)
}
```

Listing A.1: Linear smoothing and Polynomial regression functions.

```r
########################################
### CURVE FITTING BY LINEAR SMOOTHING ###
########################################

source('SDS383D/HW3/smoothing.R')

# pdf('./SDS383D/HW3/Notes/Img/h.pdf', width = 8.5, height = 6)

# Data generation
n <- 50
x <- runif(n, 0, 2*pi)
eps <- rnorm(n, 0, 0.25)
y <- sin(x) + eps

# Centering step
x <- x - mean(x)
y <- y - mean(y)

par(mar=c(4,4,2,2), family = 'Palatino', cex = 1.1)
plot(x, y, pch = 16, cex = 0.8, xlab = 'x', ylab = 'y')

h <- 1

xnew <- seq(min(x), max(x), length.out = 100)
ynew <- linsmooth(x, y, xnew, h)

lines(xnew, ynew, col = 'firebrick3', lwd = 2)
```

```
28
29 # pdf('./SDS383D/HW3/Notes/Img/112.pdf', width = 8.5, height = 6)
30 # par(mar=c(4,4,2,2), family = 'Palatino', cex = 1.1)
31 # plot(x, y, pch = 16, cex = 0.8, xlab = 'x', ylab = 'y')
32 #
33 # h <- c(.5, 1, 2, 5)
34 # colors <- c('dodgerblue3','indianred2','goldenrod2','seagreen3','darkorchid2')
35 # for (i in 1:length(h)){
36 #   ynew <- linsmooth(x, y, xnew, h[i], type = 'uniform')
37 #   lines(xnew, ynew, col = colors[i], lwd = 2)
38 # }
39 # legend('topright', legend = c('h = 0.5','h = 1','h = 2','h = 5'), lwd = 2, col = colors[1:4])
40 # dev.off()
41
42 ########################
43 ### CROSS VALIDATION ###
44 ########################
45
46 rm(list=ls())
47
48 source('SDS383D/HW3/smoothing.R')
49
50
51 # Case 1: wiggly function with highly noisy observations
52 n <- 500
53 x <- runif(n, 0, 1)
54 eps <- rnorm(n, 0, 0.75)
55 y <- sin(10*pi*x) + eps
56
57 # Centering step
58 mx <- mean(x); x <- x - mx
59 y <- y - mean(y)
60
61 x.train <- x[1:(n/2)]
62 y.train <- y[1:(n/2)]
63 x.test <- x[(n/2 + 1):n]
64 y.test <- y[(n/2 + 1):n]
65
66 par(mar=c(4,4,2,2), family = 'Palatino', cex = 1.1)
67 plot(x.train, y.train, pch = 16, cex = 0.8, xlab = 'x', ylab = 'y', main = 'Wiggly function,
       noisy obs.')
68 lines(sort(x.train), sin(10*pi*(sort(x.train) + mx)), lwd = 2.5, col = 'seagreen3', lty = 2)
69
70 hgrid <- exp(seq(-10, 1, length.out = 50))
71 cv.lin <- cv.linsmooth(x.train, y.train, x.test, y.test, hgrid)
72 h.opt <- hgrid[which.min(cv.lin$MSE)]
73
74 xgrid <- seq(min(x.train), max(x.train), length.out = 100)
75 y.pred <- linsmooth(x.train, y.train, xgrid, h.opt)
76 lines(xgrid, y.pred, lwd = 2.5, col = 'firebrick2')
77
78
79 # Case 2: wiggly function with not so noisy observations
80 n <- 500
81 x <- runif(n, 0, 1)
82 eps <- rnorm(n, 0, 0.2)
83 y <- sin(10*pi*x) + eps
```

```
84
85  # Centering step
86  mx <- mean(x); x <- x - mx
87  y <- y - mean(y)
88
89  x.train <- x[1:(n/2)]
90  y.train <- y[1:(n/2)]
91  x.test <- x[(n/2 + 1):n]
92  y.test <- y[(n/2 + 1):n]
93
94  par(mar=c(4,4,2,2), family = 'Palatino', cex = 1.1)
95  plot(x.train, y.train, pch = 16, cex = 0.8, xlab = 'x', ylab = 'y', main = 'Wiggly function, not
        noisy obs.')
96  lines(sort(x.train), sin(10*pi*(sort(x.train) + mx)), lwd = 2.5, col = 'seagreen3', lty = 2)
97
98  hgrid <- exp(seq(-10, 1, length.out = 50))
99  cv.lin <- cv.linsmooth(x.train, y.train, x.test, y.test, hgrid)
100 h.opt <- hgrid[which.min(cv.lin$MSE)]
101
102 xgrid <- seq(min(x.train), max(x.train), length.out = 100)
103 y.pred <- linsmooth(x.train, y.train, xgrid, h.opt)
104 lines(xgrid, y.pred, lwd = 2.5, col = 'firebrick2')
105
106
107 # Case 3: smooth function with highly noisy observations
108 n <- 500
109 x <- runif(n, 0, 1)
110 eps <- rnorm(n, 0, 0.75)
111 y <- sin(2*pi*x) + eps
112
113 # Centering step
114 mx <- mean(x); x <- x - mx
115 y <- y - mean(y)
116
117 x.train <- x[1:(n/2)]
118 y.train <- y[1:(n/2)]
119 x.test <- x[(n/2 + 1):n]
120 y.test <- y[(n/2 + 1):n]
121
122 par(mar=c(4,4,2,2), family = 'Palatino', cex = 1.1)
123 plot(x.train, y.train, pch = 16, cex = 0.8, xlab = 'x', ylab = 'y', main = 'Smooth function,
        noisy obs.')
124 lines(sort(x.train), sin(2*pi*(sort(x.train) + mx)), lwd = 2.5, col = 'seagreen3', lty = 2)
125
126 hgrid <- exp(seq(-10, 1, length.out = 50))
127 cv.lin <- cv.linsmooth(x.train, y.train, x.test, y.test, hgrid)
128 h.opt <- hgrid[which.min(cv.lin$MSE)]
129
130 xgrid <- seq(min(x.train), max(x.train), length.out = 100)
131 y.pred <- linsmooth(x.train, y.train, xgrid, h.opt)
132 lines(xgrid, y.pred, lwd = 2.5, col = 'firebrick2')
133
134
135 # Case 4: smooth function with not so noisy observations
136 n <- 500
137 x <- runif(n, 0, 1)
138 eps <- rnorm(n, 0, 0.1)
```

```
y <- sin(2*pi*x) + eps

# Centering step
mx <- mean(x); x <- x - mx
y <- y - mean(y)

x.train <- x[1:(n/2)]
y.train <- y[1:(n/2)]
x.test <- x[(n/2 + 1):n]
y.test <- y[(n/2 + 1):n]

par(mar=c(4,4,2,2), family = 'Palatino', cex = 1.1)
plot(x.train, y.train, pch = 16, cex = 0.8, xlab = 'x', ylab = 'y', main = 'Smooth function, not
    noisy obs.')
lines(sort(x.train), sin(2*pi*(sort(x.train) + mx)), lwd = 2.5, col = 'seagreen3', lty = 2)

hgrid <- exp(seq(-10, 1, length.out = 50))
cv.lin <- cv.linsmooth(x.train, y.train, x.test, y.test, hgrid)
h.opt <- hgrid[which.min(cv.lin$MSE)]

xgrid <- seq(min(x.train), max(x.train), length.out = 100)
y.pred <- linsmooth(x.train, y.train, xgrid, h.opt)
lines(xgrid, y.pred, lwd = 2.5, col = 'firebrick2')


###################################
### LOCAL POLYNOMIAL REGRESSION ###
###################################

rm(list=ls())

source('SDS383D/HW3/smoothing.R')


utilities <- read.csv(file = 'SDS383D-master/data/utilities.csv')

y <- log(utilities[,2]/utilities[,3])
x <- utilities[,1]

n <- length(y)

# # Centering step
# x <- x - mean(x)
# y <- y - mean(y)


par(mar=c(4,4,2,2), family = 'Palatino', cex = 1.1)
plot(x, y, pch = 16, cex = 0.8, xlab = 'Average Temperature', ylab = 'Average daily gas bill')


# Try with different values of h and with different bandwidths
h <- 1
D <- 1
xnew <- seq(min(x), max(x), length.out = 100)
ynew <- polregr(x, y, xnew, D, h)$ynew
lines(xnew, ynew, col = 'seagreen2', lwd = 2)
ynew <- linsmooth(x, y, xnew, h)
```

```r
lines(xnew, ynew, col = 'darkorchid2', lwd = 2)

# We now pick the optimal h via LOOCV
# pdf('./SDS383D/HW3/Notes/Img/prova2.pdf', width = 8.5, height = 6)
par(mar=c(4,4,2,2), family = 'Palatino', cex = 1.1)
plot(x, y, pch = 16, cex = 0.8, xlab = 'Average Temperature', ylab = 'Average daily gas bill')

D <- 1
hgrid <- seq(1, 10, length.out = 100)
LOOCV <- cv.polregr(x, y, D, hgrid)
h.opt <- hgrid[which.min(LOOCV)]

xgrid <- seq(min(x), max(x), length.out = 100)
fit <- polregr(x, y, xgrid, D, h.opt)
y.pred <- fit$ynew
H <- fit$H
lines(xgrid, y.pred, lwd = 2, col = 'seagreen3')


# Inspect the residuals of this optimal model
yhat <- polregr(x, y, x, D, h.opt)$ynew
par(mar=c(4,4,2,2), family = 'Palatino', cex = 1.1)
plot(x, y - yhat, pch = 16, cex = 0.8, xlab = 'Average Temperature', ylab = 'Residuals')


# Let us now construct confidence intervals for the regression line
fit <- polregr(x, y, x, D, h.opt)
y.pred <- fit$ynew
H <- fit$H
sigma2hat <- sum((y - as.numeric(H %*% y))^2) / (n - 2 * sum(diag(H)) + sum(diag(crossprod(H))))
sigma2x <- rowSums(H^2)[order(x)]

par(mar=c(4,4,2,2), family = 'Palatino', cex = 1.1)
plot(x, y, pch = 16, cex = 0.8, xlab = 'Average Temperature', ylab = 'Average daily gas bill',
    ylim = c(min(y), 2.4))
lines(sort(x), y.pred[order(x)], lwd = 2, col = 'firebrick2')
polygon(c(rev(sort(x)), sort(x)), c(rev(y.pred[order(x)] - sqrt(sigma2x) * sqrt(sigma2hat) *
    qnorm(0.975)), y.pred[order(x)] + sqrt(sigma2x) * sqrt(sigma2hat) * qnorm(0.975)), col = rgb
    (0.83, 0.83, 0.83, 0.5), border = rgb(0.83, 0.83, 0.83, 0.5))
```

Listing A.2: Script to call the linear smoothing and polynomial regression functions.

```r
my.dist <- function(x, y){
  # Function that computes the distance matrix D, where D_{ij} is x_i - y_j
  # ------------------------------------------------------------------------
  # Args:
  #   - x: first vector of points
  #   - y: second vector of points
  # Returns:
  #   - D: matrix of pairwise distances
  # ---------------------------------
  n <- length(x)
  m <- length(y)

  D <- array(NA, dim = c(n, m))

  # In order to speed up the code, we vectorize the distance computations: we create
```

```
  # two matrices, X and Y, by replicating the vectors x and y. The only operation we do
  # is the difference between the two matrices.
  X <- matrix(rep(x, m), nrow = n, ncol = m, byrow = F)
  Y <- matrix(rep(y, n), nrow = n, ncol = m, byrow = T)

  D <- abs(X - Y)

  return (D)
}


C.squaredexp <- function(x, y, b, tau1sq, tau2sq){
  # Function that computes the squared exponential covariance function
  # ------------------------------------------------------------------
  # Args:
  #   - x: first vector of points
  #   - y: second vector of points
  #   - b, tau1sq, tau2sq: parameters of the SE covariance function
  # Returns:
  #   - CSE: squared exponential covariance matrix
  # ------------------------------------------------
  n <- length(x)
  m <- length(y)
  CSE <- array(NA, dim = c(n, m))

  d <- my.dist(x, y)
  CSE <- tau1sq * exp(-(1/2)*(d/b)^2)
  diag(CSE) <- diag(CSE) + tau2sq

  return(CSE)
}


C.matern52 <- function(x, y, b, tau1sq, tau2sq){
  # Function that computes the Matern 5/2 covariance function
  # ----------------------------------------------------------
  # Args:
  #   - x: first vector of points
  #   - y: second vector of points
  #   - b, tau1sq, tau2sq: parameters of the M52 covariance function
  # Returns:
  #   - CM52: Matern 5/2 covariance matrix
  n <- length(x)
  m <- length(y)
  CM52 <- array(NA, dim = c(n, m))

  d <- my.dist(x, y)
  CM52 <- tau1sq * (1 + sqrt(5)*d/b + (5*d^2)/(3*b^2)) * exp(- sqrt(5) * d/b)
  diag(CM52) <- diag(CM52) + tau2sq

  return(CM52)
}


rgaussproc <- function(x, m, C){
  # Samples one trajectory from a Gaussian Process
  # ----------------------------------------------
```

```
  # Args:
  #   - x: vector where evaluating the process
  #   - m: mean function for the process evaluated at x
  #   - C: covariance function of the process evaluated at x
  # Returns:
  #   - values of the process at the target points x
  # ------------------------------------------------
  return (as.numeric(rmvnorm(1, mean = m, sigma = C)))
}


pred.GP <- function(x, y, xstar, b, tau1sq, tau2sq, sigma2, type = 'squaredexp'){
  # Computes the log-marginal likelihood of the Gaussian Process for a given set of
  # parameters
  # ----------
  # Args:
  #   - x: vector where we observe the process
  #   - y: observed values for y
  #   - xstar: values where to predict mean and variance of f(xstar)
  #   - b, tau1sq, tau2sq: parameters for the Gaussian Process
  #   - sigma2: noise (unknown) of the data
  # Returns:
  #   - pred.mean: predicted mean of f(xstar)
  #   - pred.var: predicted variance of f(xstar)
  # ------------------------------------------
  n <- length(x)

  if (type == 'squaredexp'){
    Cxx <- C.squaredexp(x, x, b, tau1sq, tau2sq)
    Czx <- C.squaredexp(xstar, x, b, tau1sq, tau2sq)
    Czz <- C.squaredexp(xstar, xstar, b, tau1sq, tau2sq)
  }
  else if (type == 'matern'){
    Cxx <- C.matern52(x, x, b, tau1sq, tau2sq)
    Czx <- C.matern52(xstar, x, b, tau1sq, tau2sq)
    Czz <- C.matern52(xstar, xstar, b, tau1sq, tau2sq)
  }
  else{
    stop ('Select a valid Covariance function family')
  }

  Vinv = solve(Cxx + diag(sigma2, n))

  pred.mean <- Czx %*% Vinv %*% y
  pred.var <- Czz - Czx %*% Vinv %*% t(Czx)

  return (list('pred.mean' = pred.mean, 'pred.var' = pred.var))
}


marginal.likelihood <- function(x, y, b, tau1sq, tau2sq, sigma2, type = 'squaredexp'){
  # Computes the log-marginal likelihood of the Gaussian Process for a given set of
  # parameters
  # ----------
  # Args:
  #   - x: vector where we observe the process
  #   - y: observed values for y
```

```
  #   - b, tau1sq, tau2sq: parameters for the Gaussian Process
  #   - sigma2: noise (unknown) of the data
  # Returns:
  #   - the value of the log-marginal likelihood
  # ------------------------------------------
  n <- length(x)

  if (type == 'squaredexp')
    C <- C.squaredexp(x, x, b, tau1sq, tau2sq)
  else if (type == 'matern')
    C <- C.matern52(x, x, b, tau1sq, tau2sq)
  else
    stop ('Select a valid Covariance function family')

  V = C + diag(sigma2, n)

  return (dmvnorm(y,  mean = rep(0, n), sigma = V, log=TRUE))
}
```

Listing A.3: Gaussian Process functions.

```
my.dist <- function(x, y, b1, b2){
  # Function that computes the distance matrix D, where D_{ij} is x_i - y_j
  # ------------------------------------------------------------------------
  # Args:
  #   - x: first vector of points
  #   - y: second vector of points
  # Returns:
  #   - D: matrix of pairwise distances
  # ---------------------------------
  n <- dim(x)[1]
  m <- dim(y)[1]

  D <- array(NA, dim = c(n, m))

  X1 <- matrix(rep(x[,1], m), nrow = n, ncol = m, byrow = F)
  Y1 <- matrix(rep(y[,1], n), nrow = n, ncol = m, byrow = T)
  X2 <- matrix(rep(x[,2], m), nrow = n, ncol = m, byrow = F)
  Y2 <- matrix(rep(y[,2], n), nrow = n, ncol = m, byrow = T)

  D <- sqrt( ((X1 - Y1) / b1)^2 + ((X2 - Y2) / b2)^2 )

  return (D)
}


C.squaredexp <- function(x, y, b, tau1sq, tau2sq, b1, b2){
  # Function that computes the squared exponential covariance function
  # ------------------------------------------------------------------
  # Args:
  #   - x: first vector of points
  #   - y: second vector of points
  #   - b, tau1sq, tau2sq: parameters of the SE covariance function
  # Returns:
  #   - CSE: squared exponential covariance matrix
  # ---------------------------------------------
  n <- dim(x)[1]
```

```
  m <- dim(y)[1]
  CSE <- array(NA, dim = c(n, m))

  d <- my.dist(x, y, b1, b2)
  CSE <- tau1sq * exp(-(1/2)*(d/b)^2)
  diag(CSE) <- diag(CSE) + tau2sq

  return(CSE)
}


C.matern52 <- function(x, y, b, tau1sq, tau2sq, b1, b2){
  # Function that computes the Matern 5/2 covariance function
  # ----------------------------------------------------------
  # Args:
  #   - x: first vector of points
  #   - y: second vector of points
  #   - b, tau1sq, tau2sq: parameters of the M52 covariance function
  # Returns:
  #   - CM52: Matern 5/2 covariance matrix
  n <- length(x)
  m <- length(y)
  CM52 <- array(NA, dim = c(n, m))

  d <- my.dist(x, y, b1, b2)
  CM52 <- tau1sq * (1 + sqrt(5)*d/b + (5*d^2)/(3*b^2)) * exp(- sqrt(5) * d/b)
  diag(CM52) <- diag(CM52) + tau2sq

  return(CM52)
}


rgausssproc <- function(x, m, C){
  # Samples one trajectory from a Gaussian Process
  # ----------------------------------------------
  # Args:
  #   - x: vector where evaluating the process
  #   - m: mean function for the process evaluated at x
  #   - C: covariance function of the process evaluated at x
  # Returns:
  #   - values of the process at the target points x
  # ----------------------------------------------
  return (as.numeric(rmvnorm(1, mean = m, sigma = C)))
}


pred.GP <- function(x, y, xstar, b, tau1sq, tau2sq, sigma2, b1, b2, type = 'squaredexp'){
  # Computes the log-marginal likelihood of the Gaussian Process for a given set of
  # parameters
  # ----------
  # Args:
  #   - x: vector where we observe the process
  #   - y: observed values for y
  #   - xstar: values where to predict mean and variance of f(xstar)
  #   - b, tau1sq, tau2sq: parameters for the Gaussian Process
  #   - sigma2: noise (unknown) of the data
  # Returns:
```

```
  #   - pred.mean: predicted mean of f(xstar)
  #   - pred.var: predicted variance of f(xstar)
  # ------------------------------------------
  n <- dim(X)[1]

  if (type == 'squaredexp'){
    Cxx <- C.squaredexp(x, x, b, tau1sq, tau2sq, b1, b2)
    Czx <- C.squaredexp(xstar, x, b, tau1sq, tau2sq, b1, b2)
    Czz <- C.squaredexp(xstar, xstar, b, tau1sq, tau2sq, b1, b2)
  }
  else if (type == 'matern'){
    Cxx <- C.matern52(x, x, b, tau1sq, tau2sq, b1, b2)
    Czx <- C.matern52(xstar, x, b, tau1sq, tau2sq, b1, b2)
    Czz <- C.matern52(xstar, xstar, b, tau1sq, tau2sq, b1, b2)
  }
  else{
    stop ('Select a valid Covariance function family')
  }

  Vinv = solve(Cxx + diag(sigma2, n))

  pred.mean <- Czx %*% Vinv %*% y
  pred.var <- Czz - Czx %*% Vinv %*% t(Czx)

  return (pred.mean)
}


marginal.likelihood <- function(x, y, b, tau1sq, tau2sq, sigma2, b1, b2, type = 'squaredexp'){
  # Computes the log-marginal likelihood of the Gaussian Process for a given set of
  # parameters
  # ----------
  # Args:
  #   - x: vector where we observe the process
  #   - y: observed values for y
  #   - b, tau1sq, tau2sq: parameters for the Gaussian Process
  #   - sigma2: noise (unknown) of the data
  # Returns:
  #   - the value of the log-marginal likelihood
  # ------------------------------------------
  n <- dim(x)[1]

  if (type == 'squaredexp')
    C <- C.squaredexp(x, x, b, tau1sq, tau2sq, b1, b2)
  else if (type == 'matern')
    C <- C.matern52(x, x, b, tau1sq, tau2sq, b1, b2)
  else
    stop ('Select a valid Covariance function family')

  V = C + diag(sigma2, n)

  return (dmvnorm(y,  mean = rep(0, n), sigma = V, log=TRUE))
}
```

Listing A.4: Spatial smoothing functions.

```
# ================================================
```

```
# ==== Basics on Gaussian Processes Smoothing ====
# ================================================

source('SDS383D/HW3/gaussianprocess.R')

b <- 1
tau1sq <- 1
tau2sq <- 1E-6
# Role of the parameters:
#   - b: controls how much distant points are correlated. Small values of b produce a non
#     correlated process (more noisy), large values of b produce a process with high
#     correlation and, subsequently, smoother functions.
#   - tau1sq: scaling factor that controls the amplitude of the covariances.
#   - tau2sq: scaling factor that controls the amplitude of the variances, that is,
#             how noisy each point is.


colors <- c('indianred2','dodgerblue2','seagreen3','goldenrod2','darkorchid2')
#pdf('./SDS383D/HW3/Notes/Img/gaussianexp6.pdf', width = 8.5, height = 6)
# Squared Exponential covariance function
par(mar=c(4,4,1,2), family = 'Palatino', cex = 1.1)
xgrid <- seq(0, 1, length.out = 100)
plot(xgrid, rgaussproc(xgrid, m = rep(0, length(xgrid)), C = C.squaredexp(xgrid, xgrid, b, tau1sq
    , tau2sq)), type ='l', col = colors[1], lwd = 2, ylim = c(-3, 3), xlab = 'x', ylab = 'f(x)')
for (i in 2:5){
  lines(xgrid, rgaussproc(xgrid, m = rep(0, length(xgrid)), C = C.squaredexp(xgrid, xgrid, b,
      tau1sq, tau2sq)), col = colors[i], lwd = 2)
}

Npred <- 250
pred <- array(NA, dim = c(Npred, length(xgrid)))
for (i in 1:Npred){
  pred[i,] <- rgaussproc(xgrid, m = rep(0, length(xgrid)), C = C.squaredexp(xgrid, xgrid, b,
      tau1sq, tau2sq))
}
pred.quant <- as.matrix(apply(pred, 2, quantile, prob = c(0.025, 0.5, 0.975)))
lines(xgrid, pred.quant[2,], lty = 2, lwd = 2)
polygon(c(xgrid, rev(xgrid)), c(pred.quant[3,], rev(pred.quant[1,])), col = rgb
    (0.83,0.83,0.83,0.5), border = rgb(0.83, 0.83, 0.83, 0.5))
dev.off()

#pdf('./SDS383D/HW3/Notes/Img/gaussianmatern6.pdf', width = 8.5, height = 6)
# Matern covariance function
par(mar=c(4,4,1,2), family = 'Palatino', cex = 1.1)
xgrid <- seq(0, 1, length.out = 100)
plot(xgrid, rgaussproc(xgrid, m = rep(0, length(xgrid)), C = C.matern52(xgrid, xgrid, b, tau1sq,
    tau2sq)), type ='l', lwd = 2, col = colors[1], ylim = c(-3, 3), xlab = 'x', ylab = 'f(x)')
for (i in 2:5){
  lines(xgrid, rgaussproc(xgrid, m = rep(0, length(xgrid)), C = C.matern52(xgrid, xgrid, b,
      tau1sq, tau2sq)), col = colors[i], lwd = 2)
}
# We notice that, in general, using the same parameters as before the Matern covariance
# function yields to a more noisy Gaussian process.

Npred <- 250
pred <- array(NA, dim = c(Npred, length(xgrid)))
for (i in 1:Npred){
```

```
  pred[i,] <- rgaussproc(xgrid, m = rep(0, length(xgrid)), C = C.matern52(xgrid, xgrid, b, tau1sq
      , tau2sq))
}
pred.quant <- as.matrix(apply(pred, 2, quantile, prob = c(0.025, 0.5, 0.975)))
lines(xgrid, pred.quant[2,], lty = 2, lwd = 2)
polygon(c(xgrid, rev(xgrid)), c(pred.quant[3,], rev(pred.quant[1,])), col = rgb
    (0.83,0.83,0.83,0.5), border = rgb(0.83, 0.83, 0.83, 0.5))
dev.off()



# ==================================
# ==== Nonparametric Regression ====
# ==================================

rm(list = ls())

source('SDS383D/HW3/gaussianprocess.R')

utilities <- read.csv(file = 'SDS383D-master/data/utilities.csv')

y <- utilities[,2]/utilities[,3]
x <- utilities[,1]

# Centering Step
y <- y - mean(y)

n <- length(y)
sigma2 <- 1

#pdf('./SDS383D/HW3/Notes/Img/gaussianfit.pdf', width = 8.5, height = 6)
par(mar=c(4,4,2,2), family = 'Palatino', cex = 1.1)
plot(x, y, pch = 16, cex = 0.8, xlab = 'Average Temperature', ylab = 'Average daily gas bill')

b <- 10
tau1sq <- 10
tau2sq <- 1E-6

xgrid <- seq(min(x), max(x), length.out = 100)

pred <- pred.GP(x, y, xgrid, b, tau1sq, tau2sq, sigma2, type = 'matern')
lines(xgrid, pred$pred.mean, pch = 16, lwd = 2, col = 'firebrick3')

Npred <- 500
pred.grid <- array(NA, dim = c(Npred, length(xgrid)))
for (i in 1:Npred){
  pred.grid[i,] <- rgaussproc(xgrid, m = pred$pred.mean, C = pred$pred.var)
}
pred.quant <- as.matrix(apply(pred.grid, 2, quantile, prob = c(0.025, 0.5, 0.975)))
polygon(c(xgrid, rev(xgrid)), c(pred.quant[1,], rev(pred.quant[3,])), col = rgb
    (0.83,0.83,0.83,0.5), lwd = 0.1)


# Pick the hyperparameters by marginal likelihood
pred <- pred.GP(x, y, x, b, tau1sq, tau2sq, sigma2, type = 'matern')
sigma2 <- as.numeric(var(y - pred$pred.mean))

```

```
tau2sq <- 1E-6

D <- 20
b.grid <- seq(90, 110, length.out = D)
tau1sq.grid <- seq(30, 40, length.out = D)
sigma2.grid <- seq(0.01, 5, length.out = D)

marg.loglik = array(NA, dim = c(D, D, D))
for(i in 1:D){
  for(j in 1:D){
    for(k in 1:D){
      marg.loglik[i,j,k] <- marginal.likelihood(x, y, b.grid[i], tau1sq.grid[j], tau2sq, sigma2.grid[k], type = 'matern')
    }
  }
  cat('Iteration: ', i, "\n")
}
b.opt <- b.grid[which(marg.loglik == max(marg.loglik), arr.ind = T)[1]]
tau1sq.opt <- tau1sq.grid[which(marg.loglik == max(marg.loglik), arr.ind = T)[2]]
sigma2.opt <- sigma2.grid[which(marg.loglik == max(marg.loglik), arr.ind = T)[3]]

# Check if we are looking in the correct region
par(mar=c(4,4.5,2,2), family = 'Palatino')
contour(b.grid, tau1sq.grid, marg.loglik[,,which(marg.loglik == max(marg.loglik), arr.ind = T)[3]], nlevels=20, xlab = 'b', ylab = bquote(tau[1]^2), col = 'dodgerblue4', lwd = 2)

points(b.opt, tau1sq.opt, pch = 16, col = 'indianred3')

par(mar=c(4,4,2,2), family = 'Palatino', cex = 1.1)
plot(x, y, pch = 16, cex = 0.8, xlab = 'Average Temperature', ylab = 'Average daily gas bill')

pred <- pred.GP(x, y, xgrid, b.opt, tau1sq.opt, tau2sq, sigma2.opt, type = 'matern')
lines(xgrid, pred$pred.mean, pch = 16, lwd = 2, col = 'firebrick3')


Npred <- 250
pred.grid <- array(NA, dim = c(Npred, length(xgrid)))
for (i in 1:Npred){
  pred.grid[i,] <- rgausspproc(xgrid, m = pred$pred.mean, C = pred$pred.var)
}
pred.quant <- as.matrix(apply(pred.grid, 2, quantile, prob = c(0.025, 0.5, 0.975)))
polygon(c(xgrid, rev(xgrid)), c(pred.quant[1,], rev(pred.quant[3,])), col = rgb(0.83,0.83,0.83,0.5), lwd = 0.1)


# ============================
# ==== Spatial Smoothing ====
# ============================

rm(list = ls())

source('SDS383D/HW3/spatialsmoothing.R')
library(fields)
library(ggmap)
library(RColorBrewer)
cols <- rev(colorRampPalette(brewer.pal(10, "RdYlBu"))(100))

```

```
161
162 # Load the data
163 weather <- read.csv(file = 'SDS383D-master/data/weather.csv')
164 n <- dim(weather)[1]
165
166 # Download the map and plot the data
167 limits <- c(min(weather$lon), min(weather$lat), max(weather$lon), max(weather$lat))
168 bw.map <- get_map(location = limits, maptype = 'terrain-background')
169 ggmap(bw.map) +
170   scale_fill_distiller(palette = 'RdYlBu') +
171   geom_point(data = weather, aes(x = lon, y = lat, fill = temperature), colour = 'gray', size =
        2.5, pch = 21)
172
173 # Choice of the anisotropy parameters
174 center.lat <- min(weather$lat) + diff(range(weather$lat))/2
175 b1 <- 1/sin(center.lat) # relative to the longitude
176 b2 <- 1                 # relative to the latitude
177
178 # #pdf('./SDS383D/HW3/Notes/Img/pressure.pdf', width = 8.5, height = 6)
179 # par(mar=c(4,4,2,2), family = 'Palatino')
180 # quilt.plot(weather$lon, weather$lat, weather$pressure, xlab = 'Longitude', ylab = 'Latitude',
      main = 'Heatmap of Pressure')
181 # dev.off()
182 # #pdf('./SDS383D/HW3/Notes/Img/temperature.pdf', width = 8.5, height = 6)
183 # par(mar=c(4,4,2,2), family = 'Palatino')
184 # quilt.plot(weather$lon, weather$lat, weather$temperature, xlab = 'Longitude', ylab = 'Latitude
      ', main = 'Heatmap of Temperature')
185 # dev.off()
186
187 # Rename the data
188 X <- cbind(weather$lon, weather$lat)
189 z <- weather$temperature - mean(weather$temperature)
190
191 # First guess for sigma:
192 # - temperature data range from -9 to 9 (use sigma2 = 1)
193 # - pressure data range from -600 to 700 (use sigma2 = 5000)
194 sigma2 <- 1
195
196 # Plot for a guess of the parameters on a coarse grid
197 b <- 1
198 tau1sq <- sigma2
199 tau2sq <- 1E-6
200
201 D <- 64
202 longrid <- seq(min(weather$lon), max(weather$lon), length.out = D)
203 latgrid <- seq(min(weather$lat), max(weather$lat), length.out = D)
204 grid <- expand.grid(longrid, latgrid)
205
206 pred <- pred.GP(X, z, grid, b, tau1sq, tau2sq, sigma2, b1, b2, type = 'matern')
207 par(mar=c(4,4,2,2), family = 'Palatino')
208 quilt.plot(grid[,1], grid[,2], pred, nx = D, ny = D)
209
210 # Estimate the variance of the model
211 pred <- pred.GP(X, z, X, b, tau1sq, tau2sq, sigma2, b1, b2, type = 'matern')
212 sigma2 <- as.numeric(var(z - pred)) # estimate the variance of the model
213
214 # Pick the hyperparameters by marginal likelihood
```

```
tau2sq <- 1E-6 # this is fixed
D <- 20
b.grid <- seq(0.5, 1.5, length.out = D)
#b.grid <- seq(1, 1.6, length.out = D)
tau1sq.grid <- seq(1, 15, length.out = D)
#tau1sq.grid <- seq(50000, 80000, length.out = D)

marg.loglik = array(NA, dim = c(D, D))
for(i in 1:D){
  for(j in 1:D){
      marg.loglik[i,j] <- marginal.likelihood(X, z, b.grid[i], tau1sq.grid[j], tau2sq, sigma2, b1
          , b2, type = 'matern')
  }
  cat('Iteration: ', i, "\n")
}

par(mar=c(4,4.5,2,2), family = 'Palatino')
contour(b.grid, tau1sq.grid, marg.loglik, nlevels=20, xlab = 'b', ylab = bquote(tau[1]^2), col =
    'dodgerblue4', lwd = 2)

b.opt <- b.grid[which(marg.loglik == max(marg.loglik), arr.ind = T)[1]]
tau1sq.opt <- tau1sq.grid[which(marg.loglik == max(marg.loglik), arr.ind = T)[2]]
points(b.opt, tau1sq.opt, pch = 16, col = 'indianred3')

# Fit the final model (fine grid, optimal hyperparameters)
D <- 64
longrid <- seq(min(weather$lon), max(weather$lon), length.out = D)
latgrid <- seq(min(weather$lat), max(weather$lat), length.out = D)
grid <- expand.grid(longrid, latgrid)

pred <- pred.GP(X, z, grid, b.opt, tau1sq.opt, tau2sq, sigma2, b1, b2, type = 'matern')

#load(file = './SDS383D/HW3/pred_press.Rdata')
#load(file = './SDS383D/HW3/pred_temp.Rdata')

#pdf('./SDS383D/HW3/Notes/Img/tempsmooth.pdf', width = 8.5, height = 6)
par(mar=c(4,4,2,2), family = 'Palatino')
quilt.plot(grid[,1], grid[,2], pred, xlab = 'Longitude', ylab = 'Latitude', main = 'Heatmap of
    Temperature', nx = D, ny = D)
contour(x = longrid, y = latgrid, matrix(pred, D, D, byrow = F), nlevels = 10, add = T, lwd =
    1.5, labcex = 0.7)
dev.off()

pred.values <- data.frame(cbind(grid, pred))
names(pred.values) <- c('lon','lat','pred')
#pdf('./SDS383D/HW3/Notes/Img/presssmoothmap.pdf', width = 8.5, height = 6)
ggmap(bw.map) +
  geom_tile(data = pred.values, aes(x = lon, y = lat, fill = pred), alpha = 0.7) +
  scale_fill_distiller(palette = 'RdYlBu') +
  geom_contour(data = pred.values, aes(x = lon, y = lat, z = pred, colour = ..level..), size = 1)
      +
  scale_colour_gradientn(colors = cols) +
  geom_point(data = weather, aes(x = lon, y = lat, fill = temperature), colour = 'gray', size =
      2, pch = 21)
dev.off()


```

```
# ============================
# ==== Spatial Smoothing ====
# ============================

rm(list = ls())

source('SDS383D/HW3/spatialsmoothing.R')
library(fields)

weather <- read.csv(file = 'SDS383D-master/data/weather.csv')


#pdf('./SDS383D/HW3/Notes/Img/pressure.pdf', width = 8.5, height = 6)
par(mar=c(4,4,2,2), family = 'Palatino')
quilt.plot(weather$lon, weather$lat, weather$pressure, xlab = 'Longitude', ylab = 'Latitude',
    main = 'Heatmap of Pressure')
dev.off()
#pdf('./SDS383D/HW3/Notes/Img/temperature.pdf', width = 8.5, height = 6)
par(mar=c(4,4,2,2), family = 'Palatino')
quilt.plot(weather$lon, weather$lat, weather$temperature, xlab = 'Longitude', ylab = 'Latitude',
    main = 'Heatmap of Temperature')
dev.off()

# Rename the data
X <- cbind(weather$lon, weather$lat)
z <- weather$pressure
w <- weather$temperature

plot(w, z, pch = 16, cex = 0.8)
fit <- lm(z ~ w)
abline(a = fit$coefficients[1], b = fit$coefficients[2], lwd = 2, col = 'indianred3')
res <- z - fit$coefficients[1] - fit$coefficients[2] * w

hist(res, col = 'gray', border = 'white', nclass = 20)

# First guess for sigma:
# - temperature data range from -9 to 9 (use sigma2 = 1)
# - pressure data range from -600 to 700 (use sigma2 = 10000)
sigma2 <- 1000

# Plot for a guess of the parameters on a coarse grid
b <- 2
tau1sq <- 10000
tau2sq <- 1E-6

D <- 64
longrid <- seq(min(weather$lon), max(weather$lon), length.out = D)
latgrid <- seq(min(weather$lat), max(weather$lat), length.out = D)
grid <- expand.grid(longrid, latgrid)

pred <- pred.GP(X, res, grid, b, tau1sq, tau2sq, sigma2)
quilt.plot(grid[,1], grid[,2], pred, nx = D, ny = D)
quilt.plot(weather$lon, weather$lat, res, nx = D, ny = D)


# Estimate the variance of the model
pred <- pred.GP(X, res, X, b, tau1sq, tau2sq, sigma2)
```

```
sigma2 <- as.numeric(var(z - pred)) # estimate the variance of the model

# Pick the hyperparameters by marginal likelihood
tau2sq <- 1E-6 # this is fixed
D <- 40
#b.grid <- seq(0.5, 1.5, length.out = D)
b.grid <- seq(1.3, 1.4, length.out = D)
#tau1sq.grid <- seq(1, 15, length.out = D)
tau1sq.grid <- seq(40000, 55000, length.out = D)

marg.loglik = array(NA, dim = c(D, D))
for(i in 1:D){
  for(j in 1:D){
    marg.loglik[i,j] <- marginal.likelihood(X, z, b.grid[i], tau1sq.grid[j], tau2sq, sigma2)
  }
  cat('Iteration: ', i, "\n")
}

par(mar=c(4,4.5,2,2), family = 'Palatino')
contour(b.grid, tau1sq.grid, marg.loglik, nlevels=20, xlab = 'b', ylab = bquote(tau[1]^2), col =
    'dodgerblue4', lwd = 2)

b.opt <- b.grid[which(marg.loglik == max(marg.loglik), arr.ind = T)[1]]
tau1sq.opt <- tau1sq.grid[which(marg.loglik == max(marg.loglik), arr.ind = T)[2]]
points(b.opt, tau1sq.opt, pch = 16, col = 'indianred3')

# Fit the final model (fine grid, optimal hyperparameters)
D <- 64
longrid <- seq(min(weather$lon), max(weather$lon), length.out = D)
latgrid <- seq(min(weather$lat), max(weather$lat), length.out = D)
grid <- expand.grid(longrid, latgrid)

pred <- pred.GP(X, res, grid, b.opt, tau1sq.opt, tau2sq, sigma2)

#load(file = './SDS383D/HW3/pred_press.Rdata')
#load(file = './SDS383D/HW3/pred_temp.Rdata')

#pdf('./SDS383D/HW3/Notes/Img/tempsmooth.pdf', width = 8.5, height = 6)
par(mar=c(4,4,2,2), family = 'Palatino')
quilt.plot(grid[,1], grid[,2], pred, xlab = 'Longitude', ylab = 'Latitude', main = 'Heatmap of
    Temperature', nx = D, ny = D)
contour(x = longrid, y = latgrid, matrix(pred, D, D, byrow = F), nlevels = 10, add = T, lwd =
    1.5, labcex = 0.7)
dev.off()



pred.mat <- matrix(pred, D, D, byrow = F)
# Define colors
nbcol = 100
color = rev(rainbow(nbcol, start = 0/6, end = 4.5/6))
zcol  = cut(pred.mat, nbcol)
# Plot 3D
open3d()
persp3d(longrid, latgrid, pred.mat, theta=50, phi=25, expand=0.75, col=color[zcol],
        ticktype="detailed", xlab="Longitude", ylab="Latitude", zlab="", axes=TRUE, alpha = 0.7)
zcol  = cut(z, nbcol)
```

```
375 points3d(X[,1], X[,2], z, col = color[zcol], cex = 1.5)
```

Listing A.5: Script to call the gaussian process and spatial smoothing functions.