

SDS 383D: Homework 3

Giorgio Paulon

March 20, 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

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

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

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

We can write

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

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

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

so that

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

(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

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

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

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

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.

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

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^* at some new point x^* , denoted $\hat{f}(x^*)$. Assume for the moment that $f(x)$ is linear, and that y and x have already had their means subtracted, in which case $y_i = \beta x_i + \epsilon_i$.

Return to your least-squares estimator for multiple regression. Show that for the one-predictor case, your prediction $\hat{y}^* = f(x^*) = \hat{\beta}x^*$ may be expressed as a linear smoother of the following form:

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

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

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

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

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

In the case of Least squares, the estimator is

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

where

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

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^* and x_i affects the weights $w(x_i, x^*)$. 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^* and x_i . The weight $w(x_i, x^*)$ is negative if x_i and x^* lie to the opposite sides of \bar{x} .

(B) A kernel function $K(x)$ is a smooth function satisfying

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

A very simple example is the uniform kernel,

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

Another common example is the Gaussian kernel:

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

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

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

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, \dots, 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.

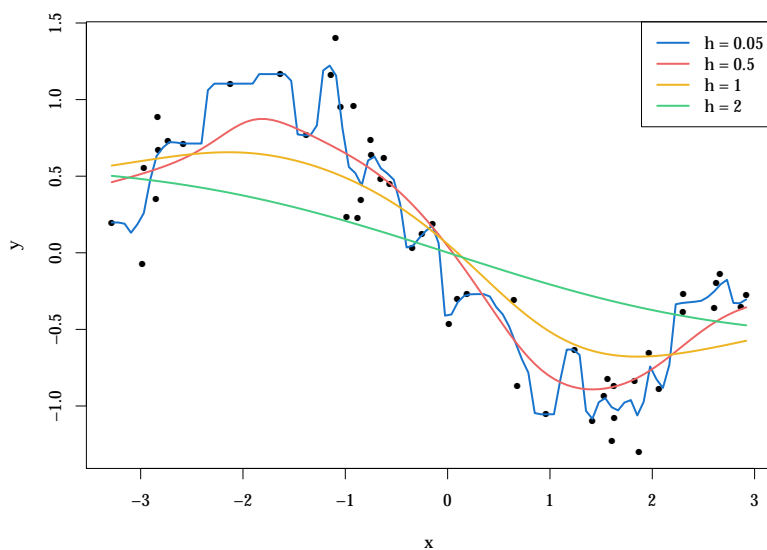


Figure 1: Gaussian kernel regression for the sampled black points. Different values of the bandwidth λ result in different smoothed curves.

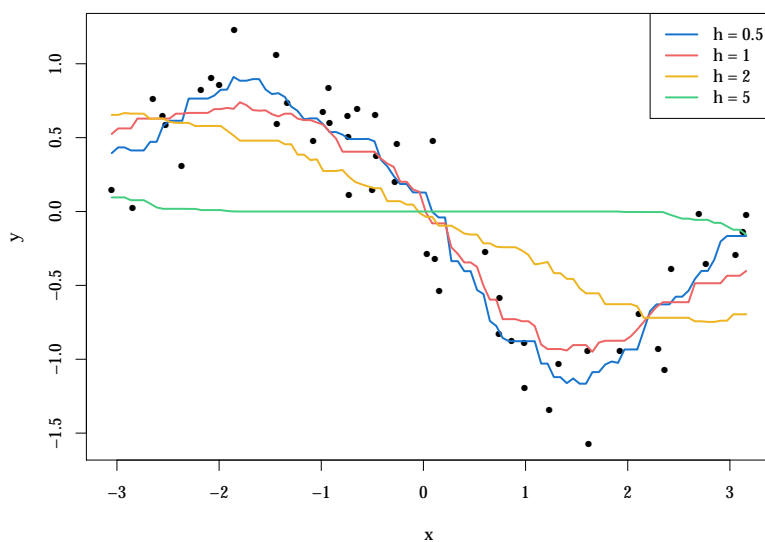


Figure 2: Uniform kernel regression for the sampled black points. Different values of the bandwidth λ result in different smoothed curves.

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:

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

where (y_i^*, x_i^*) are the points in the test set, and \hat{y}_i^* 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!

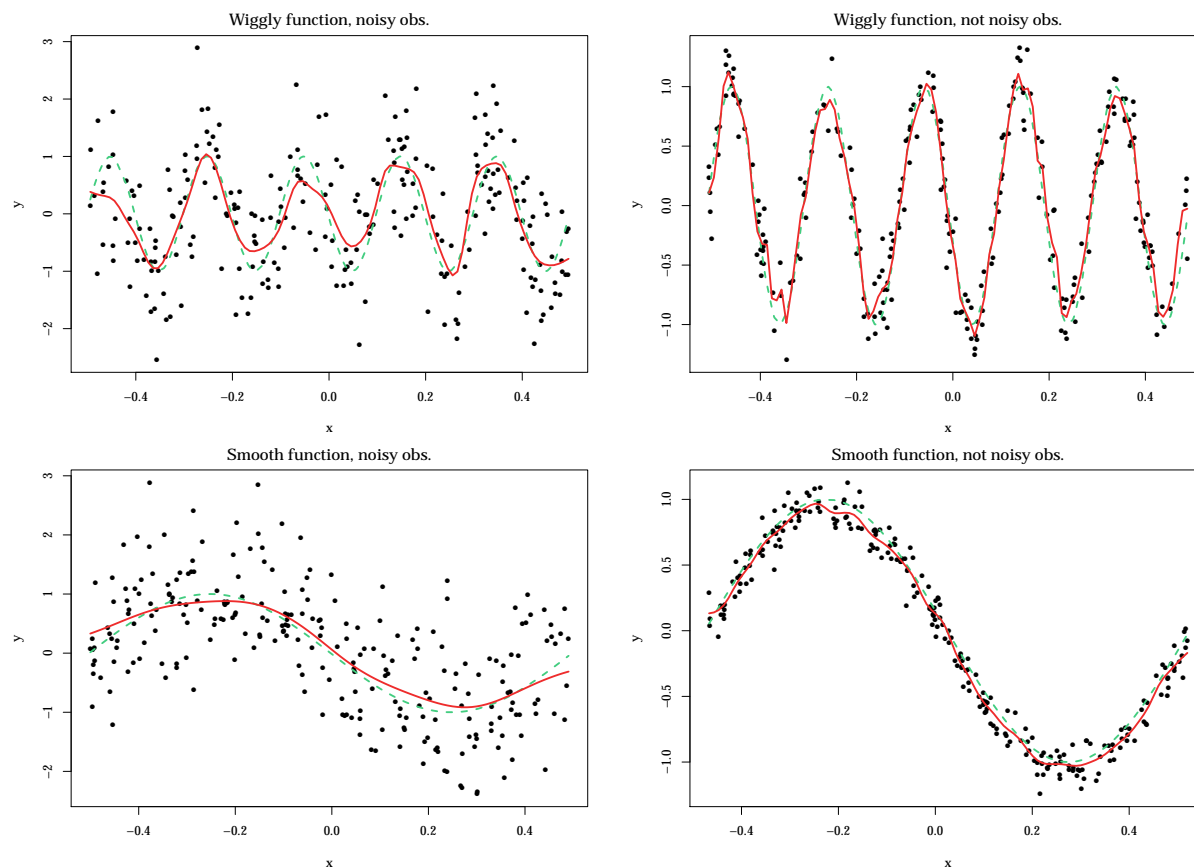


Figure 3: Four different smoothing problems. The smoothed version of the data is represented in red solid line, the “true” density generating the data is overlapped in green dashed line. The wiggly function is $f(x) = \sin(10\pi x)$, the smooth one is $g(x) = \sin(2\pi x)$. The noisy observations have 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 i th 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,

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

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

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

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

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

where $w^*(x, x_i) = \frac{K\left(\frac{x_i - x}{h}\right)}{\sum_{i=1}^n K\left(\frac{x_i - x}{h}\right)}$ 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

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

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

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

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 $\hat{\mathbf{a}}$, 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 $\hat{\mathbf{a}}_x$, so that $g_x(x_1, \dots, x_n; \hat{\mathbf{a}}) = \hat{f}(x)$.

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

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

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

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

and the corresponding estimated function at the target point x is

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

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

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

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

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

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

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

Thus,

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

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 \mathbf{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 \mathbf{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 $\mathbf{X}^T Q \mathbf{X}$ has expectation

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

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

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

and simplify things as much as possible. Roughly under what circumstances will this estimator be nearly unbiased for large n ? Note: the quantity $2\text{tr}(H) - \text{tr}(H^T H)$ 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 $E[\mathbf{y}] = f(\mathbf{x})$ and $\text{Cov}[\mathbf{y}] = \sigma^2 \mathcal{I}_n$, we can write

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

Therefore

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

which is nearly unbiased for large n if $f(\mathbf{x}) \approx Hf(\mathbf{x})$, that is, if the “true” and unknown function $f(\mathbf{x})$ 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 $\text{tr}(H) = p$. Thus,

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

In general, $n - \text{tr}(H^T H) - 2\text{tr}(H)$ 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.

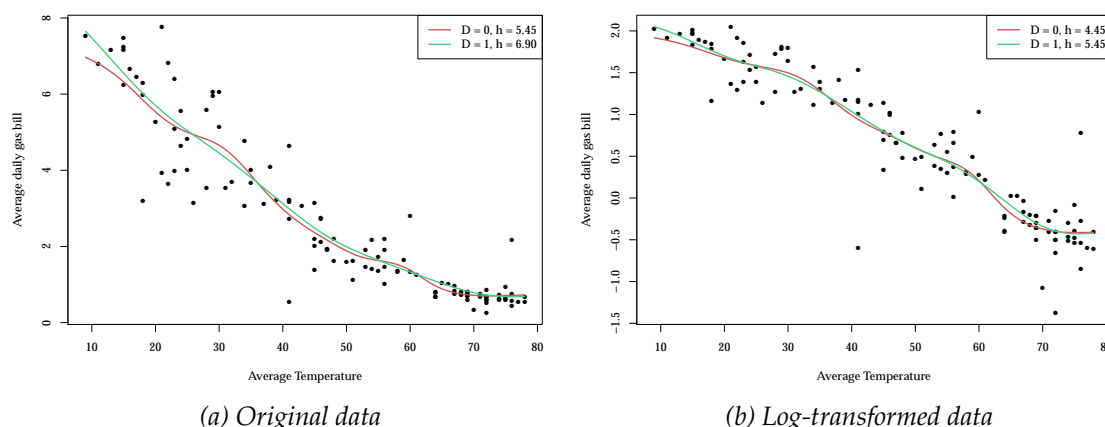


Figure 4

- (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. $\text{Var}(\epsilon_i) = E[\epsilon_i^2]$. 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.

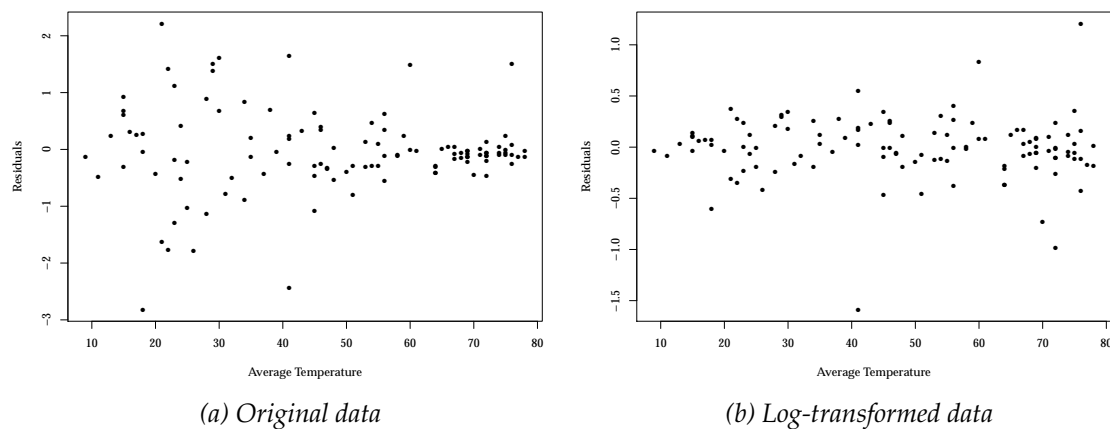


Figure 5: Residuals of the model applied to both the original data and the log-transformed data.

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.

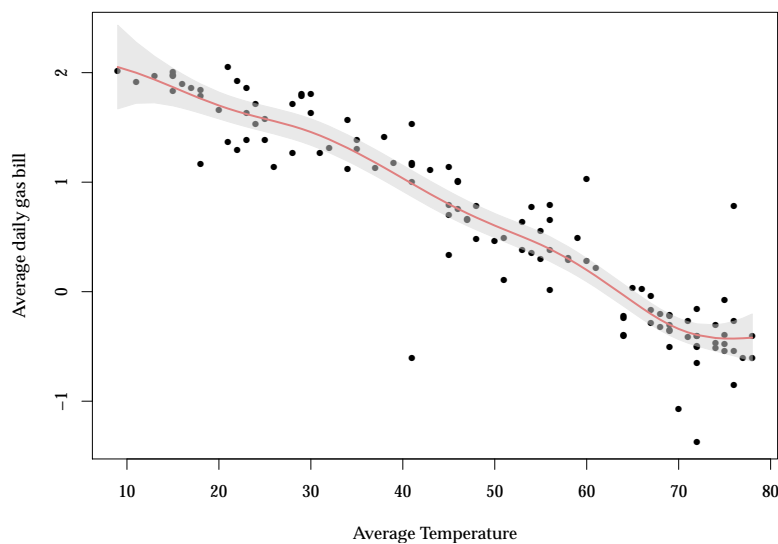


Figure 6: Local linear smoother for the utilities data. The optimal value of the bandwidth has been chosen via LOOCV. The gray shaded area represents the 95% confidence intervals.

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 $x_1, \dots, x_N \in \mathcal{X}$, the random vector $[f(x_1), \dots, f(x_N)]^T$ 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 $f \sim \text{GP}(m, C)$ for some mean function $m : \mathcal{X} \rightarrow \mathbb{R}$ and a covariance function $C : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}^+$. These functions define the moments of all finite-dimensional marginals of the process, in the sense that

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

for all $x_1, x_2 \in \mathcal{X}$. More generally, the random vector $[f(x_1), \dots, f(x_N)]^T$ 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, \dots, 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: $f(x_1), \dots, f(x_N)$.

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 $5/2$:

$$C_{M5/2}(x_1, x_2) = \tau_1^2 \left\{ 1 + \frac{\sqrt{5}d}{b} + \frac{5d^2}{3b^2} \right\} \exp \left(\frac{-\sqrt{5}d}{b} \right) + \tau_2^2 \delta(x_1, x_2),$$

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.

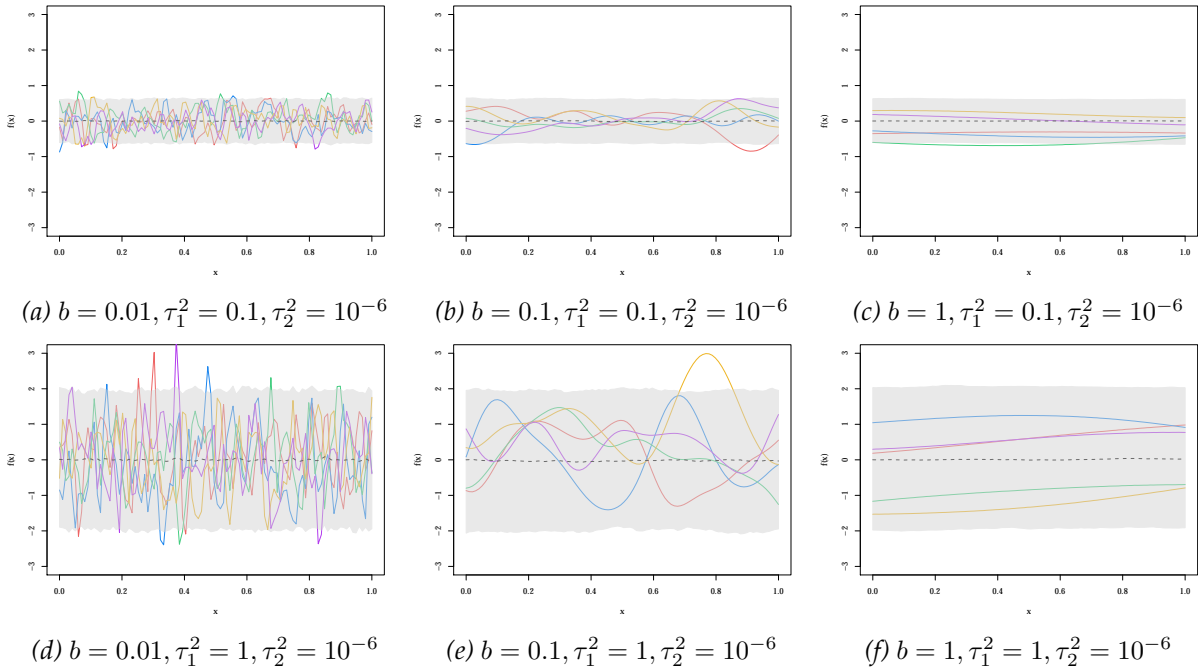


Figure 7: Sample of 5 functions from a Gaussian process with squared exponential covariance function. The mean $m(x) = 0$ is represented in dashed black line, the 95% credible interval is shadowed in gray.

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^* ? 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 $p(f(x^*)|x^*, \mathbf{x}, \mathbf{f}(\mathbf{x}))$ the requested predictive distribution for the value of the gaussian process at a new point.

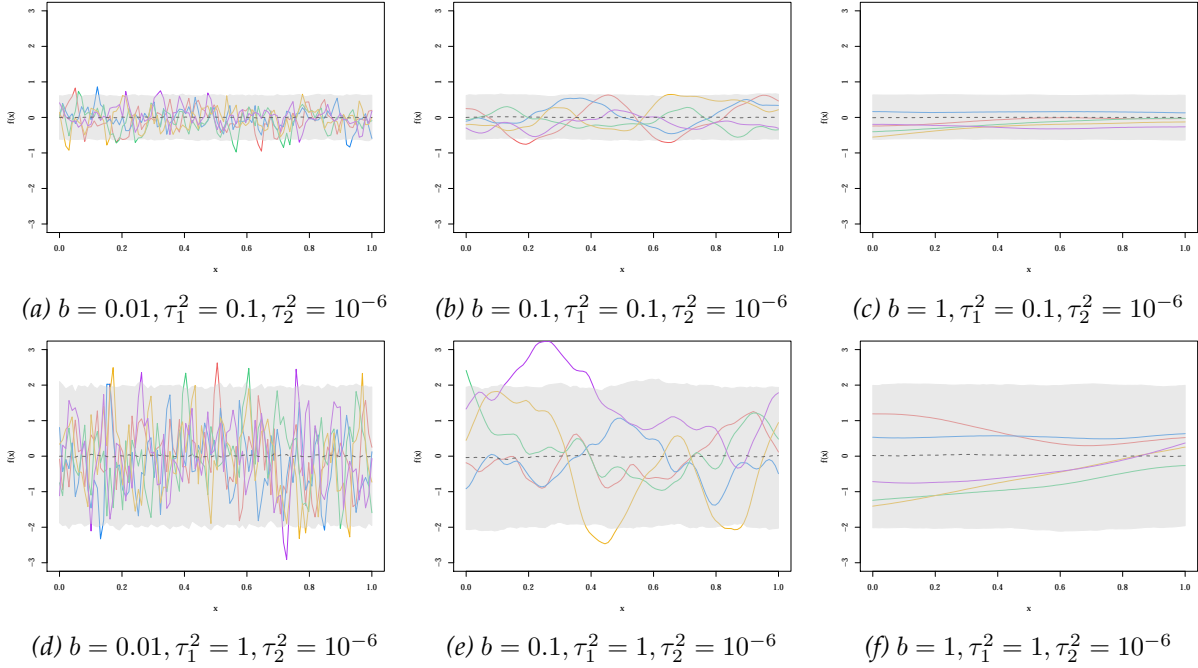


Figure 8: Sample of 5 functions from a Gaussian process with Matern 5/2 exponential covariance function. The mean $m(x) = 0$ is represented in dashed black line, the 95% credible interval is shadowed in gray.

We know that, by definition of Gaussian process,

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

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

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

where

$$\Sigma_{12} = (C(x^*, x_1), \dots, C(x^*, x_N))$$

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

(C) Prove the following lemma.

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

$$\boldsymbol{\epsilon} \sim N(\mathbf{0}, \Sigma)$$

$$\boldsymbol{\theta} \sim N(\boldsymbol{m}, V).$$

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.

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

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

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

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

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

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

where

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

Therefore,

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

- (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 $f(x^*)$ at some new point x^* 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

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

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 $\epsilon_i \stackrel{\text{iid}}{\sim} N(0, \sigma^2)$. Therefore

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

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

$$f(x^*) | \mathbf{y} \sim N \left(C(\mathbf{x}, \mathbf{x}^*) [C(\mathbf{x}, \mathbf{x}) + \sigma^2 \mathcal{I}]^{-1} \mathbf{y}; C(\mathbf{x}^*, \mathbf{x}^*) - C(\mathbf{x}, \mathbf{x}^*) [C(\mathbf{x}, \mathbf{x}) + \sigma^2 \mathcal{I}]^{-1} C(\mathbf{x}^*, \mathbf{x}) \right)$$

The posterior mean can be written as

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

where

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

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 $E[f(x^*) | \mathbf{y}, \tau_1^2, \tau_2^2, b]$ is still linear but $E[f(x^*) | \mathbf{y}]$ 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 $\mathbf{y} = (y_1, \dots, y_N)$ 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

$$\begin{aligned} Y_i | f(x_i), x_i, \sigma^2 &\stackrel{\text{ind}}{\sim} N(f(x_i), \sigma^2) \\ f &\sim GP(0, C) \end{aligned}$$

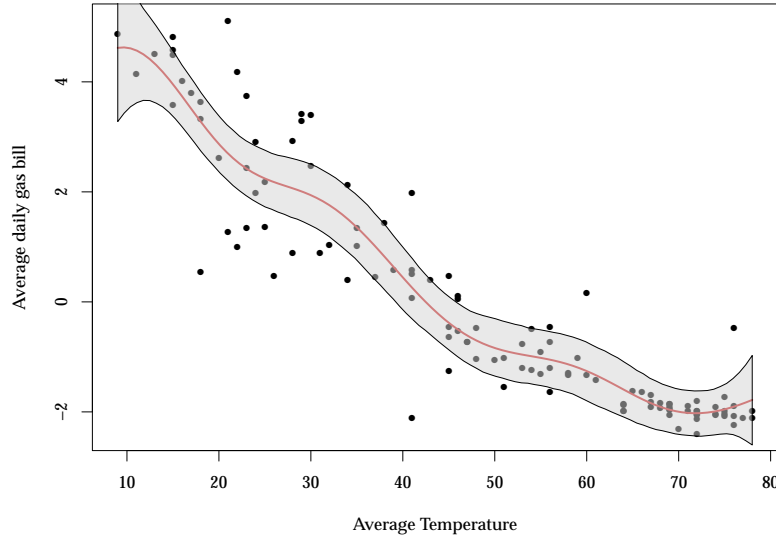


Figure 9: Fit of the utilities data with a Gaussian process, using a non optimal choice of the parameters. The pointwise 95% confidence interval is overlapped in the gray area.

we can compute the marginal likelihood by integrating out the random function \mathbf{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 \mathbf{x} , which we will denote with \mathbf{f} . Hence we have (the conditioning on σ^2 and \mathbf{x} is implicit),

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

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

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

where

$$\begin{aligned} \mathbf{m}^* &= \left(\frac{\mathcal{I}}{\sigma^2} + C^{-1} \right)^{-1} \frac{\mathcal{I}}{\sigma^2} \mathbf{y} \\ v &= -\frac{1}{2} \mathbf{y}^T \frac{\mathcal{I}}{\sigma^2} \left(\frac{\mathcal{I}}{\sigma^2} + C^{-1} \right)^{-1} \frac{\mathcal{I}}{\sigma^2} \mathbf{y}. \end{aligned}$$

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

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

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

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

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

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

- (E) Return to the utilities or ethanol data sets. Fix $\tau_2^2 = 0$, and evaluate the log of the marginal likelihood function $p(\mathbf{y} | \tau_1^2, b)$ 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 \mathbf{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

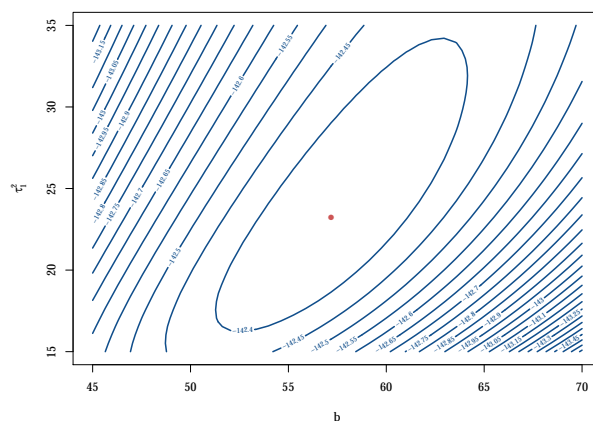


Figure 10: Marginal log-likelihood evaluated on a fine grid of points for the two parameters (b, τ_1^2) . The optimal value is the red point.

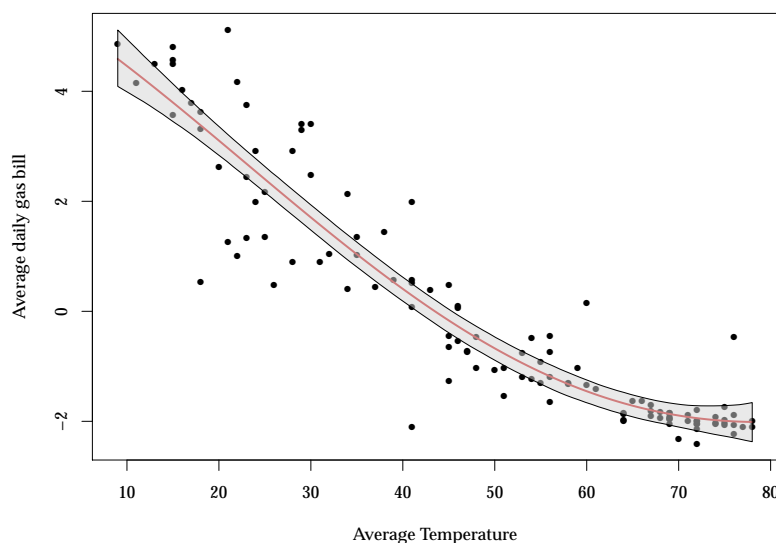


Figure 11: Fit of the utilities data with a Gaussian process, using the optimal choice of the parameters. The pointwise 95% confidence interval is overlapped in the gray area.

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.

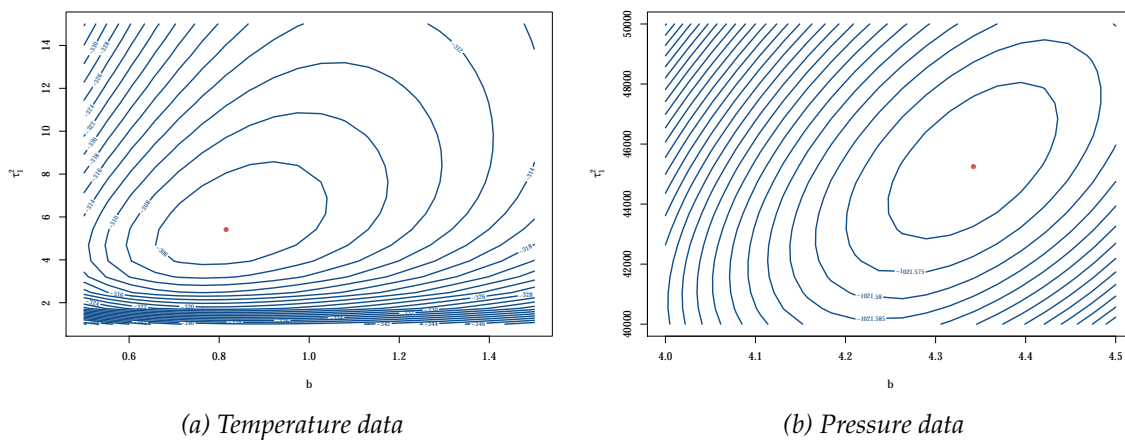


Figure 12: Marginal log-likelihood as a function of the two parameters (b, τ_1^2) for the two different responses: temperature and pressure

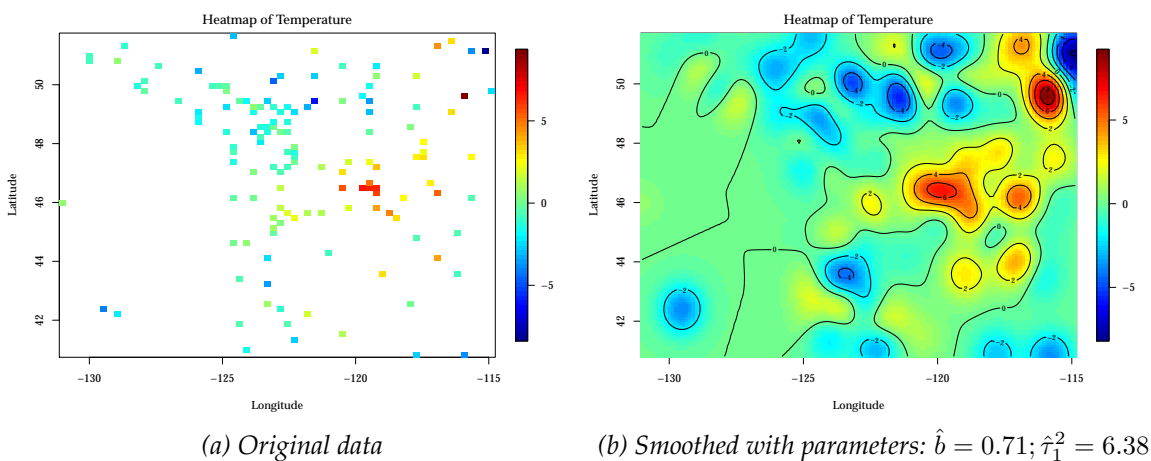


Figure 13: Smoothing of the temperature data

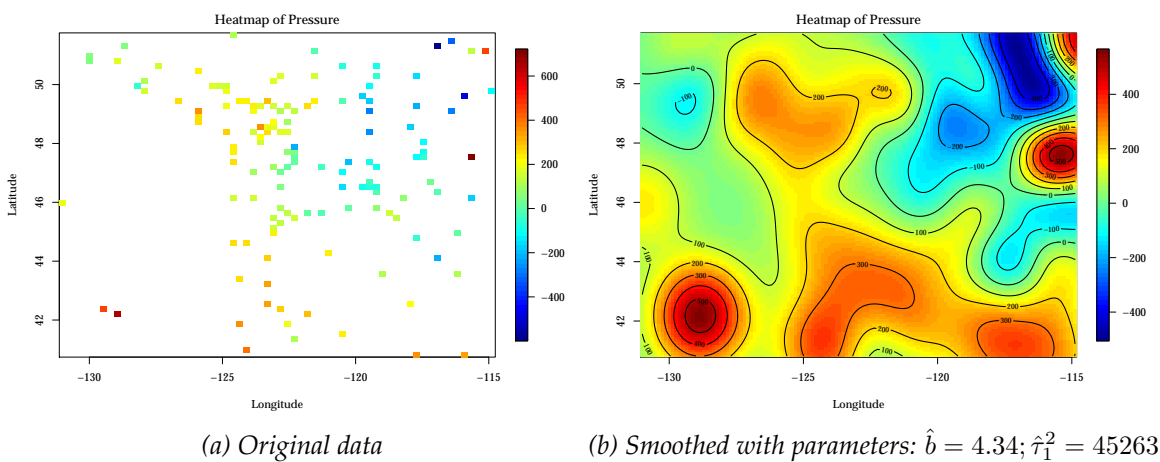


Figure 14: Smoothing of the pressure data

Appendix A

R code

```
1 gauss.kernel <- function(x) {
2   # Computes the Gaussian kernel function
3   # -----
4   # Args:
5   #   - x: distance between the two points, scaled by h
6   # Returns:
7   #   - K: value of the Gaussian kernel
8   # -----
9   K <- (1/sqrt(2*pi)) * exp(-(1/2) * x^2)
10
11   return (K)
12 }
13
14
15 unif.kernel <- function(x) {
16   # Computes the Uniform kernel function
17   # -----
18   # Args:
19   #   - distance: distance between the two points, scaled by h
20   # Returns:
21   #   - K: value of the Gaussian kernel
22   # -----
23   K <- (1/2) * as.numeric(abs(x) <= 1)
24
25   return (K)
26 }
27
28
29 weights <- function(x, xnew, h, type = 'gaussian'){
30   # Computes the weights for the new observation xnew given the previous ones x
31   # -----
32   # Args:
33   #   - x: vector containing the points
34   #   - xnew: new point for which we compute the weights
35   #   - h: bandwidth
36   # Returns:
37   #   - w: vector of weights for the new observation xnew
38   # -----
39   n <- length(x)
40   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)))

```

```

98 MSE <- array(NA, dim = k)
99 for (i in 1:k){
100   y.new[i,] <- linsmooth(x.train, y.train, x.test, hgrid[i])
101   MSE[i] <- sum((y.new[i,] - y.test)^2)
102 }
103
104 return (list('y.pred' = y.new, 'MSE' = MSE))
105 }
106
107
108 polregr <- function(x, y, xnew, D, h, type = 'gaussian'){
109   # Computes the predicted values of the linear polynomial regression for the
110   # values xnew
111   # -----
112   # Args:
113   #   - x: vector of the data to fit the linear smoother
114   #   - y: response vector to fit the linear smoother
115   #   - xnew: grid on which we wish to evaluate the linear smoother
116   #   - D: maximum degree of the polynomial
117   #   - h: bandwidth
118   # Returns:
119   #   - ynew: values of the linear smoother on the grid xnew
120   #   - H: the projection matrix, i.e. yhat = H %*% y
121   # -----
122   n <- length(x)
123   nstar <- length(xnew)
124
125   # Initialization of the data structures
126   ynew <- array(NA, dim = nstar)
127   H <- array(NA, dim = c(nstar, n))
128   Rx <- array(NA, dim = c(n, D+1))
129   Rx[,1] <- rep(1, n) # Initialization of the R matrix
130   for (i in 1:length(ynew)){ # loop through the observations to predict
131     # Compute the weights
132     W <- diag(weights(x, xnew[i], h, type))
133
134     # We compute the Rx matrix for the target point xnew[i]
135     if (D >= 1){
136       Rx[,2:(D+1)] <- rep(x - xnew[i], D)
137       for (d in 1:(D+1))
138         Rx[,d] <- (Rx[,d])^(d-1)
139     }
140
141     # We solve the WLS problem
142     RtW <- crossprod(Rx, W) # precaching
143     b <- solve(RtW %*% Rx) %*% RtW # solution of the WLS problem
144
145     # Compute the projection matrix for the ith data point
146     H[i,] <- b[1,]
147
148     # Remember: at the target point, the estimated function is a_0^hat
149     ynew[i] <- (b %*% y)[1]
150   }
151   return(list('ynew' = ynew, 'H' = H))
152 }
153
154

```

```

155 cv.polregr <- function(x, y, D, hgrid, type = 'gaussian'){
156   # Computes the LOOCV error for the linear polynomial regression for several values
157   # of the bandwidth h
158   # -----
159   # Args:
160   #   - x: vector of the data to fit the linear smoother
161   #   - y: response vector to fit the linear smoother
162   #   - D: maximum degree of the polynomial
163   #   - hgrid: grid of values for the bandwidth
164   # Returns:
165   #   - LOOCV: leave-one-out cross validation error
166   # -----
167   k <- length(hgrid)
168   n <- length(x)
169   H <- array(NA, dim = c(n, n))
170
171   LOOCV <- array(NA, dim = k)
172   for (h in 1:k){
173     # We compute the H matrix for each possible value of the bandwidth
174     H <- polregr(x, y, x, D, hgrid[h])$H
175
176     # We use the LOOCV lemma
177     LOOCV[h] <- sum((y - H %*% y)^2 / (1 - diag(H)^2)
178   }
179
180   return (LOOCV)
181 }

```

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

```

1 #####
2 ### CURVE FITTING BY LINEAR SMOOTHING ###
3 #####
4
5 source('SDS383D/HW3/smoothing.R')
6
7 # pdf('./SDS383D/HW3/Notes/Img/h.pdf', width = 8.5, height = 6)
8
9 # Data generation
10 n <- 50
11 x <- runif(n, 0, 2*pi)
12 eps <- rnorm(n, 0, 0.25)
13 y <- sin(x) + eps
14
15 # Centering step
16 x <- x - mean(x)
17 y <- y - mean(y)
18
19 par(mar=c(4,4,2,2), family = 'Palatino', cex = 1.1)
20 plot(x, y, pch = 16, cex = 0.8, xlab = 'x', ylab = 'y')
21
22 h <- 1
23
24 xnew <- seq(min(x), max(x), length.out = 100)
25 ynew <- linsmooth(x, y, xnew, h)
26
27 lines(xnew, ynew, col = 'firebrick3', lwd = 2)

```

```

28
29 # pdf('./SDS383D/HW3/Notes/Img/l12.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,
68       noisy obs.')
69 lines(sort(x.train), sin(10*pi*(sort(x.train) + mx)), lwd = 2.5, col = 'seagreen3', lty = 2)
70
71 hgrid <- exp(seq(-10, 1, length.out = 50))
72 cv.lin <- cv.linsmooth(x.train, y.train, x.test, y.test, hgrid)
73 h.opt <- hgrid[which.min(cv.lin$MSE)]
74
75 xgrid <- seq(min(x.train), max(x.train), length.out = 100)
76 y.pred <- linsmooth(x.train, y.train, xgrid, h.opt)
77 lines(xgrid, y.pred, lwd = 2.5, col = 'firebrick2')
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)

```

```

139 y <- sin(2*pi*x) + eps
140
141 # Centering step
142 mx <- mean(x); x <- x - mx
143 y <- y - mean(y)
144
145 x.train <- x[1:(n/2)]
146 y.train <- y[1:(n/2)]
147 x.test <- x[(n/2 + 1):n]
148 y.test <- y[(n/2 + 1):n]
149
150 par(mar=c(4,4,2,2), family = 'Palatino', cex = 1.1)
151 plot(x.train, y.train, pch = 16, cex = 0.8, xlab = 'x', ylab = 'y', main = 'Smooth function, not
    noisy obs.')
152 lines(sort(x.train), sin(2*pi*(sort(x.train) + mx)), lwd = 2.5, col = 'seagreen3', lty = 2)
153
154 hgrid <- exp(seq(-10, 1, length.out = 50))
155 cv.lin <- cv.linsmooth(x.train, y.train, x.test, y.test, hgrid)
156 h.opt <- hgrid[which.min(cv.lin$MSE)]
157
158 xgrid <- seq(min(x.train), max(x.train), length.out = 100)
159 y.pred <- linsmooth(x.train, y.train, xgrid, h.opt)
160 lines(xgrid, y.pred, lwd = 2.5, col = 'firebrick2')
161
162
163 #####
164 ### LOCAL POLYNOMIAL REGRESSION ###
165 #####
166
167 rm(list=ls())
168
169 source('SDS383D/HW3/smoothing.R')
170
171
172 utilities <- read.csv(file = 'SDS383D-master/data/utilities.csv')
173
174 y <- log(utilities[,2]/utilities[,3])
175 x <- utilities[,1]
176
177 n <- length(y)
178
179 # # Centering step
180 # x <- x - mean(x)
181 # y <- y - mean(y)
182
183
184 par(mar=c(4,4,2,2), family = 'Palatino', cex = 1.1)
185 plot(x, y, pch = 16, cex = 0.8, xlab = 'Average Temperature', ylab = 'Average daily gas bill')
186
187
188 # Try with different values of h and with different bandwidths
189 h <- 1
190 D <- 1
191 xnew <- seq(min(x), max(x), length.out = 100)
192 ynew <- polregr(x, y, xnew, D, h)$ynew
193 lines(xnew, ynew, col = 'seagreen2', lwd = 2)
194 ynew <- linsmooth(x, y, xnew, h)

```

```

195 lines(xnew, ynew, col = 'darkorchid2', lwd = 2)
196
197 # We now pick the optimal h via LOOCV
198 # pdf('./SDS383D/HW3/Notes/Img/prova2.pdf', width = 8.5, height = 6)
199 par(mar=c(4,4,2,2), family = 'Palatino', cex = 1.1)
200 plot(x, y, pch = 16, cex = 0.8, xlab = 'Average Temperature', ylab = 'Average daily gas bill')
201
202 D <- 1
203 hgrid <- seq(1, 10, length.out = 100)
204 LOOCV <- cv.polregr(x, y, D, hgrid)
205 h.opt <- hgrid[which.min(LOOCV)]
206
207 xgrid <- seq(min(x), max(x), length.out = 100)
208 fit <- polregr(x, y, xgrid, D, h.opt)
209 y.pred <- fit$ynew
210 H <- fit$H
211 lines(xgrid, y.pred, lwd = 2, col = 'seagreen3')
212
213
214 # Inspect the residuals of this optimal model
215 yhat <- polregr(x, y, x, D, h.opt)$ynew
216 par(mar=c(4,4,2,2), family = 'Palatino', cex = 1.1)
217 plot(x, y - yhat, pch = 16, cex = 0.8, xlab = 'Average Temperature', ylab = 'Residuals')
218
219
220 # Let us now construct confidence intervals for the regression line
221 fit <- polregr(x, y, x, D, h.opt)
222 y.pred <- fit$ynew
223 H <- fit$H
224 sigma2hat <- sum((y - as.numeric(H %*% y))^2) / (n - 2 * sum(diag(H)) + sum(diag(crossprod(H))))
225 sigma2x <- rowSums(H^2)[order(x)]
226
227 par(mar=c(4,4,2,2), family = 'Palatino', cex = 1.1)
228 plot(x, y, pch = 16, cex = 0.8, xlab = 'Average Temperature', ylab = 'Average daily gas bill',
      ylim = c(min(y), 2.4))
229 lines(sort(x), y.pred[order(x)], lwd = 2, col = 'firebrick2')
230 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.

```

1 my.dist <- function(x, y){
2   # Function that computes the distance matrix D, where D_{ij} is x_i - y_j
3   # -----
4   # Args:
5   #   - x: first vector of points
6   #   - y: second vector of points
7   # Returns:
8   #   - D: matrix of pairwise distances
9   # -----
10  n <- length(x)
11  m <- length(y)
12
13  D <- array(NA, dim = c(n, m))
14
15  # In order to speed up the code, we vectorize the distance computations: we create

```

```

16 # two matrices, X and Y, by replicating the vectors x and y. The only operation we do
17 # is the difference between the two matrices.
18 X <- matrix(rep(x, m), nrow = n, ncol = m, byrow = F)
19 Y <- matrix(rep(y, n), nrow = n, ncol = m, byrow = T)
20
21 D <- abs(X - Y)
22
23 return (D)
24 }
25
26
27 C.squaredexp <- function(x, y, b, tau1sq, tau2sq){
28   # Function that computes the squared exponential covariance function
29   # -----
30   # Args:
31   #   - x: first vector of points
32   #   - y: second vector of points
33   #   - b, tau1sq, tau2sq: parameters of the SE covariance function
34   # Returns:
35   #   - CSE: squared exponential covariance matrix
36   # -----
37   n <- length(x)
38   m <- length(y)
39   CSE <- array(NA, dim = c(n, m))
40
41   d <- my.dist(x, y)
42   CSE <- tau1sq * exp(-(1/2)*(d/b)^2)
43   diag(CSE) <- diag(CSE) + tau2sq
44
45   return(CSE)
46 }
47
48
49 C.matern52 <- function(x, y, b, tau1sq, tau2sq){
50   # Function that computes the Matern 5/2 covariance function
51   # -----
52   # Args:
53   #   - x: first vector of points
54   #   - y: second vector of points
55   #   - b, tau1sq, tau2sq: parameters of the M52 covariance function
56   # Returns:
57   #   - CM52: Matern 5/2 covariance matrix
58   n <- length(x)
59   m <- length(y)
60   CM52 <- array(NA, dim = c(n, m))
61
62   d <- my.dist(x, y)
63   CM52 <- tau1sq * (1 + sqrt(5)*d/b + (5*d^2)/(3*b^2)) * exp(- sqrt(5) * d/b)
64   diag(CM52) <- diag(CM52) + tau2sq
65
66   return(CM52)
67 }
68
69
70 rgaussproc <- function(x, m, C){
71   # Samples one trajectory from a Gaussian Process
72   # -----

```

```

73 # Args:
74 #   - x: vector where evaluating the process
75 #   - m: mean function for the process evaluated at x
76 #   - C: covariance function of the process evaluated at x
77 # Returns:
78 #   - values of the process at the target points x
79 # -----
80 return (as.numeric(rmvnorm(1, mean = m, sigma = C)))
81 }
82
83
84 pred.GP <- function(x, y, xstar, b, taulsq, tau2sq, sigma2, type = 'squaredexp'){
85   # Computes the log-marginal likelihood of the Gaussian Process for a given set of
86   # parameters
87   # -----
88   # Args:
89   #   - x: vector where we observe the process
90   #   - y: observed values for y
91   #   - xstar: values where to predict mean and variance of f(xstar)
92   #   - b, taulsq, tau2sq: parameters for the Gaussian Process
93   #   - sigma2: noise (unknown) of the data
94   # Returns:
95   #   - pred.mean: predicted mean of f(xstar)
96   #   - pred.var: predicted variance of f(xstar)
97   # -----
98   n <- length(x)
99
100  if (type == 'squaredexp'){
101    Cxx <- C.squaredexp(x, x, b, taulsq, tau2sq)
102    Czx <- C.squaredexp(xstar, x, b, taulsq, tau2sq)
103    Czz <- C.squaredexp(xstar, xstar, b, taulsq, tau2sq)
104  }
105  else if (type == 'matern'){
106    Cxx <- C.matern52(x, x, b, taulsq, tau2sq)
107    Czx <- C.matern52(xstar, x, b, taulsq, tau2sq)
108    Czz <- C.matern52(xstar, xstar, b, taulsq, tau2sq)
109  }
110  else{
111    stop ('Select a valid Covariance function family')
112  }
113
114  Vinv = solve(Cxx + diag(sigma2, n))
115
116  pred.mean <- Czx %*% Vinv %*% y
117  pred.var <- Czz - Czx %*% Vinv %*% t(Czx)
118
119  return (list('pred.mean' = pred.mean, 'pred.var' = pred.var))
120 }
121
122
123 marginal.likelihood <- function(x, y, b, taulsq, tau2sq, sigma2, type = 'squaredexp'){
124   # Computes the log-marginal likelihood of the Gaussian Process for a given set of
125   # parameters
126   # -----
127   # Args:
128   #   - x: vector where we observe the process
129   #   - y: observed values for y

```

```

130 # - b, tau1sq, tau2sq: parameters for the Gaussian Process
131 # - sigma2: noise (unknown) of the data
132 # Returns:
133 # - the value of the log-marginal likelihood
134 # -----
135 n <- length(x)
136
137 if (type == 'squaredexp')
138   C <- C.squaredexp(x, x, b, tau1sq, tau2sq)
139 else if (type == 'matern')
140   C <- C.matern52(x, x, b, tau1sq, tau2sq)
141 else
142   stop('Select a valid Covariance function family')
143
144 V = C + diag(sigma2, n)
145
146 return (dmvnorm(y, mean = rep(0, n), sigma = V, log=TRUE))
147 }

```

Listing A.3: Gaussian Process functions.

```

1 my.dist <- function(x, y, b1, b2){
2   # Function that computes the distance matrix D, where D_{ij} is x_i - y_j
3   # -----
4   # Args:
5   # - x: first vector of points
6   # - y: second vector of points
7   # Returns:
8   # - D: matrix of pairwise distances
9   # -----
10  n <- dim(x)[1]
11  m <- dim(y)[1]
12
13  D <- array(NA, dim = c(n, m))
14
15  X1 <- matrix(rep(x[,1], m), nrow = n, ncol = m, byrow = F)
16  Y1 <- matrix(rep(y[,1], n), nrow = n, ncol = m, byrow = T)
17  X2 <- matrix(rep(x[,2], m), nrow = n, ncol = m, byrow = F)
18  Y2 <- matrix(rep(y[,2], n), nrow = n, ncol = m, byrow = T)
19
20  D <- sqrt( ((X1 - Y1) / b1)^2 + ((X2 - Y2) / b2)^2 )
21
22  return (D)
23 }
24
25
26 C.squaredexp <- function(x, y, b, tau1sq, tau2sq, b1, b2){
27   # Function that computes the squared exponential covariance function
28   # -----
29   # Args:
30   # - x: first vector of points
31   # - y: second vector of points
32   # - b, tau1sq, tau2sq: parameters of the SE covariance function
33   # Returns:
34   # - CSE: squared exponential covariance matrix
35   # -----
36  n <- dim(x)[1]

```

```

37 m <- dim(y)[1]
38 CSE <- array(NA, dim = c(n, m))
39
40 d <- my.dist(x, y, b1, b2)
41 CSE <- tau1sq * exp(-(1/2)*(d/b)^2)
42 diag(CSE) <- diag(CSE) + tau2sq
43
44 return(CSE)
45 }
46
47
48 C.matern52 <- function(x, y, b, tau1sq, tau2sq, b1, b2){
49   # Function that computes the Matern 5/2 covariance function
50   # -----
51   # Args:
52   #   - x: first vector of points
53   #   - y: second vector of points
54   #   - b, tau1sq, tau2sq: parameters of the M52 covariance function
55   # Returns:
56   #   - CM52: Matern 5/2 covariance matrix
57   n <- length(x)
58   m <- length(y)
59   CM52 <- array(NA, dim = c(n, m))
60
61   d <- my.dist(x, y, b1, b2)
62   CM52 <- tau1sq * (1 + sqrt(5)*d/b + (5*d^2)/(3*b^2)) * exp(- sqrt(5) * d/b)
63   diag(CM52) <- diag(CM52) + tau2sq
64
65   return(CM52)
66 }
67
68
69 rgaussproc <- function(x, m, C){
70   # Samples one trajectory from a Gaussian Process
71   # -----
72   # Args:
73   #   - x: vector where evaluating the process
74   #   - m: mean function for the process evaluated at x
75   #   - C: covariance function of the process evaluated at x
76   # Returns:
77   #   - values of the process at the target points x
78   # -----
79   return (as.numeric(rmvnorm(1, mean = m, sigma = C)))
80 }
81
82
83 pred.GP <- function(x, y, xstar, b, tau1sq, tau2sq, sigma2, b1, b2, type = 'squaredexp'){
84   # Computes the log-marginal likelihood of the Gaussian Process for a given set of
85   # parameters
86   # -----
87   # Args:
88   #   - x: vector where we observe the process
89   #   - y: observed values for y
90   #   - xstar: values where to predict mean and variance of f(xstar)
91   #   - b, tau1sq, tau2sq: parameters for the Gaussian Process
92   #   - sigma2: noise (unknown) of the data
93   # Returns:

```

```

94 # - pred.mean: predicted mean of f(xstar)
95 # - pred.var: predicted variance of f(xstar)
96 # -----
97 n <- dim(X)[1]
98
99 if (type == 'squaredexp'){
100   Cxx <- C.squaredexp(x, x, b, tau1sq, tau2sq, b1, b2)
101   Czx <- C.squaredexp(xstar, x, b, tau1sq, tau2sq, b1, b2)
102   Czz <- C.squaredexp(xstar, xstar, b, tau1sq, tau2sq, b1, b2)
103 }
104 else if (type == 'matern'){
105   Cxx <- C.matern52(x, x, b, tau1sq, tau2sq, b1, b2)
106   Czx <- C.matern52(xstar, x, b, tau1sq, tau2sq, b1, b2)
107   Czz <- C.matern52(xstar, xstar, b, tau1sq, tau2sq, b1, b2)
108 }
109 else{
110   stop ('Select a valid Covariance function family')
111 }
112
113 Vinv = solve(Cxx + diag(sigma2, n))
114
115 pred.mean <- Czx %*% Vinv %*% y
116 pred.var <- Czz - Czx %*% Vinv %*% t(Czx)
117
118 return (pred.mean)
119 }
120
121
122 marginal.likelihood <- function(x, y, b, tau1sq, tau2sq, sigma2, b1, b2, type = 'squaredexp'){
123   # Computes the log-marginal likelihood of the Gaussian Process for a given set of
124   # parameters
125   # -----
126   # Args:
127   # - x: vector where we observe the process
128   # - y: observed values for y
129   # - b, tau1sq, tau2sq: parameters for the Gaussian Process
130   # - sigma2: noise (unknown) of the data
131   # Returns:
132   # - the value of the log-marginal likelihood
133   # -----
134   n <- dim(x)[1]
135
136   if (type == 'squaredexp'){
137     C <- C.squaredexp(x, x, b, tau1sq, tau2sq, b1, b2)
138   } else if (type == 'matern'){
139     C <- C.matern52(x, x, b, tau1sq, tau2sq, b1, b2)
140   } else{
141     stop ('Select a valid Covariance function family')
142   }
143
144   V = C + diag(sigma2, n)
145
146   return (dmvnorm(y, mean = rep(0, n), sigma = V, log=TRUE))
147 }

```

Listing A.4: Spatial smoothing functions.

```

1 # =====

```



```

2  # ==== Basics on Gaussian Processes Smoothing ====
3  # =====
4
5  source('SDS383D/HW3/gaussianprocess.R')
6
7  b <- 1
8  tau1sq <- 1
9  tau2sq <- 1E-6
10 # Role of the parameters:
11 #   - b: controls how much distant points are correlated. Small values of b produce a non
12 #       correlated process (more noisy), large values of b produce a process with high
13 #       correlation and, subsequently, smoother functions.
14 #   - tau1sq: scaling factor that controls the amplitude of the covariances.
15 #   - tau2sq: scaling factor that controls the amplitude of the variances, that is,
16 #       how noisy each point is.
17
18
19 colors <- c('indianred2', 'dodgerblue2', 'seagreen3', 'goldenrod2', 'darkorchid2')
20 #pdf('./SDS383D/HW3/Notes/Img/gaussianexp6.pdf', width = 8.5, height = 6)
21 # Squared Exponential covariance function
22 par(mar=c(4,4,1,2), family = 'Palatino', cex = 1.1)
23 xgrid <- seq(0, 1, length.out = 100)
24 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)')
25 for (i in 2:5){
26   lines(xgrid, rgaussproc(xgrid, m = rep(0, length(xgrid)), C = C.squaredexp(xgrid, xgrid, b,
    tau1sq, tau2sq)), col = colors[i], lwd = 2)
27 }
28
29 Npred <- 250
30 pred <- array(NA, dim = c(Npred, length(xgrid)))
31 for (i in 1:Npred){
32   pred[i,] <- rgaussproc(xgrid, m = rep(0, length(xgrid)), C = C.squaredexp(xgrid, xgrid, b,
    tau1sq, tau2sq))
33 }
34 pred.quant <- as.matrix(apply(pred, 2, quantile, prob = c(0.025, 0.5, 0.975)))
35 lines(xgrid, pred.quant[2,], lty = 2, lwd = 2)
36 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))
37 dev.off()
38
39 #pdf('./SDS383D/HW3/Notes/Img/gaussianmatern6.pdf', width = 8.5, height = 6)
40 # Matern covariance function
41 par(mar=c(4,4,1,2), family = 'Palatino', cex = 1.1)
42 xgrid <- seq(0, 1, length.out = 100)
43 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)')
44 for (i in 2:5){
45   lines(xgrid, rgaussproc(xgrid, m = rep(0, length(xgrid)), C = C.matern52(xgrid, xgrid, b,
    tau1sq, tau2sq)), col = colors[i], lwd = 2)
46 }
47 # We notice that, in general, using the same parameters as before the Matern covariance
48 # function yields to a more noisy Gaussian process.
49
50 Npred <- 250
51 pred <- array(NA, dim = c(Npred, length(xgrid)))
52 for (i in 1:Npred){

```

```

53   pred[i,] <- rgaussproc(xgrid, m = rep(0, length(xgrid)), C = C.matern52(xgrid, xgrid, b, tau1sq
    , tau2sq))
54 }
55 pred.quant <- as.matrix(apply(pred, 2, quantile, prob = c(0.025, 0.5, 0.975)))
56 lines(xgrid, pred.quant[2,], lty = 2, lwd = 2)
57 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))
58 dev.off()
59
60
61
62 # =====
63 # === Nonparametric Regression ===
64 # =====
65
66 rm(list = ls())
67
68 source('SDS383D/HW3/gaussianprocess.R')
69
70 utilities <- read.csv(file = 'SDS383D-master/data/utilities.csv')
71
72 y <- utilities[,2]/utilities[,3]
73 x <- utilities[,1]
74
75 # Centering Step
76 y <- y - mean(y)
77
78 n <- length(y)
79 sigma2 <- 1
80
81 #pdf('./SDS383D/HW3/Notes/Img/gaussianfit.pdf', width = 8.5, height = 6)
82 par(mar=c(4,4,2,2), family = 'Palatino', cex = 1.1)
83 plot(x, y, pch = 16, cex = 0.8, xlab = 'Average Temperature', ylab = 'Average daily gas bill')
84
85 b <- 10
86 tau1sq <- 10
87 tau2sq <- 1E-6
88
89 xgrid <- seq(min(x), max(x), length.out = 100)
90
91 pred <- pred.GP(x, y, xgrid, b, tau1sq, tau2sq, sigma2, type = 'matern')
92 lines(xgrid, pred$pred.mean, pch = 16, lwd = 2, col = 'firebrick3')
93
94 Npred <- 500
95 pred.grid <- array(NA, dim = c(Npred, length(xgrid)))
96 for (i in 1:Npred){
97   pred.grid[i,] <- rgaussproc(xgrid, m = pred$pred.mean, C = pred$pred.var)
98 }
99 pred.quant <- as.matrix(apply(pred.grid, 2, quantile, prob = c(0.025, 0.5, 0.975)))
100 polygon(c(xgrid, rev(xgrid)), c(pred.quant[1,], rev(pred.quant[3,])), col = rgb
    (0.83,0.83,0.83,0.5), lwd = 0.1)
101
102
103 # Pick the hyperparameters by marginal likelihood
104 pred <- pred.GP(x, y, x, b, tau1sq, tau2sq, sigma2, type = 'matern')
105 sigma2 <- as.numeric(var(y - pred$pred.mean))
106

```

```

107 tau2sq <- 1E-6
108
109 D <- 20
110 b.grid <- seq(90, 110, length.out = D)
111 tau1sq.grid <- seq(30, 40, length.out = D)
112 sigma2.grid <- seq(0.01, 5, length.out = D)
113
114 marg.loglik = array(NA, dim = c(D, D, D))
115 for(i in 1:D){
116   for(j in 1:D){
117     for(k in 1:D){
118       marg.loglik[i,j,k] <- marginal.likelihood(x, y, b.grid[i], tau1sq.grid[j], tau2sq, sigma2.
119         grid[k], type = 'matern')
120     }
121   }
122   cat('Iteration: ', i, "\n")
123 }
124 b.opt <- b.grid[which(marg.loglik == max(marg.loglik), arr.ind = T)[1]]
125 tau1sq.opt <- tau1sq.grid[which(marg.loglik == max(marg.loglik), arr.ind = T)[2]]
126 sigma2.opt <- sigma2.grid[which(marg.loglik == max(marg.loglik), arr.ind = T)[3]]
127
128 # Check if we are looking in the correct region
129 par(mar=c(4,4.5,2,2), family = 'Palatino')
130 contour(b.grid, tau1sq.grid, marg.loglik[, , which(marg.loglik == max(marg.loglik), arr.ind = T)
131   [3]], nlevels=20, xlab = 'b', ylab = bquote(tau[1]^2), col = 'dodgerblue4', lwd = 2)
132
133 points(b.opt, tau1sq.opt, pch = 16, col = 'indianred3')
134
135 par(mar=c(4,4,2,2), family = 'Palatino', cex = 1.1)
136 plot(x, y, pch = 16, cex = 0.8, xlab = 'Average Temperature', ylab = 'Average daily gas bill')
137
138 pred <- pred.GP(x, y, xgrid, b.opt, tau1sq.opt, tau2sq, sigma2.opt, type = 'matern')
139 lines(xgrid, pred$pred.mean, pch = 16, lwd = 2, col = 'firebrick3')
140
141 Npred <- 250
142 pred.grid <- array(NA, dim = c(Npred, length(xgrid)))
143 for (i in 1:Npred){
144   pred.grid[i,] <- rgaussproc(xgrid, m = pred$pred.mean, C = pred$pred.var)
145 }
146 pred.quant <- as.matrix(apply(pred.grid, 2, quantile, prob = c(0.025, 0.5, 0.975)))
147 polygon(c(xgrid, rev(xgrid)), c(pred.quant[1,], rev(pred.quant[3,])), col = rgb
148   (0.83,0.83,0.83,0.5), lwd = 0.1)
149
150 # =====
151 # === Spatial Smoothing ===
152 # =====
153 rm(list = ls())
154
155 source('SDS383D/HW3/spatialsmoothing.R')
156 library(fields)
157 library(ggmap)
158 library(RColorBrewer)
159 cols <- rev(colorRampPalette(brewer.pal(10, "RdYlBu"))(100))
160

```

```

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

```

```

215 tau2sq <- 1E-6 # this is fixed
216 D <- 20
217 b.grid <- seq(0.5, 1.5, length.out = D)
218 #b.grid <- seq(1, 1.6, length.out = D)
219 tau1sq.grid <- seq(1, 15, length.out = D)
220 #tau1sq.grid <- seq(50000, 80000, length.out = D)
221
222 marg.loglik = array(NA, dim = c(D, D))
223 for(i in 1:D){
224   for(j in 1:D){
225     marg.loglik[i,j] <- marginal.likelihood(X, z, b.grid[i], tau1sq.grid[j], tau2sq, sigma2, b1
226       , b2, type = 'matern')
227   }
228   cat('Iteration: ', i, "\n")
229 }
230 par(mar=c(4,4.5,2,2), family = 'Palatino')
231 contour(b.grid, tau1sq.grid, marg.loglik, nlevels=20, xlab = 'b', ylab = bquote(tau[1]^2), col =
232   'dodgerblue4', lwd = 2)
233
234 b.opt <- b.grid[which(marg.loglik == max(marg.loglik), arr.ind = T)[1]]
235 tau1sq.opt <- tau1sq.grid[which(marg.loglik == max(marg.loglik), arr.ind = T)[2]]
236 points(b.opt, tau1sq.opt, pch = 16, col = 'indianred3')
237
238 # Fit the final model (fine grid, optimal hyperparameters)
239 D <- 64
240 longrid <- seq(min(weather$lon), max(weather$lon), length.out = D)
241 latgrid <- seq(min(weather$lat), max(weather$lat), length.out = D)
242 grid <- expand.grid(longrid, latgrid)
243
244 pred <- pred.GP(X, z, grid, b.opt, tau1sq.opt, tau2sq, sigma2, b1, b2, type = 'matern')
245
246 #load(file = './SDS383D/HW3/pred_press.Rdata')
247 #load(file = './SDS383D/HW3/pred_temp.Rdata')
248
249 #pdf('./SDS383D/HW3/Notes/Img/tempsmooth.pdf', width = 8.5, height = 6)
250 par(mar=c(4,4,2,2), family = 'Palatino')
251 quilt.plot(grid[,1], grid[,2], pred, xlab = 'Longitude', ylab = 'Latitude', main = 'Heatmap of
252   Temperature', nx = D, ny = D)
253
254 contour(x = longrid, y = latgrid, matrix(pred, D, D, byrow = F), nlevels = 10, add = T, lwd =
255   1.5, labcex = 0.7)
256 dev.off()
257
258 pred.values <- data.frame(cbind(grid, pred))
259 names(pred.values) <- c('lon', 'lat', 'pred')
260 #pdf('./SDS383D/HW3/Notes/Img/presssmoothmap.pdf', width = 8.5, height = 6)
261 ggmap(bw.map) +
262   geom_tile(data = pred.values, aes(x = lon, y = lat, fill = pred), alpha = 0.7) +
263   scale_fill_distiller(palette = 'RdYlBu') +
264   geom_contour(data = pred.values, aes(x = lon, y = lat, z = pred, colour = ..level..), size = 1)
265   +
266   scale_colour_gradientn(colors = cols) +
267   geom_point(data = weather, aes(x = lon, y = lat, fill = temperature), colour = 'gray', size =
268     2, pch = 21)
269 dev.off()

```

```

266 # =====
267 # ==== Spatial Smoothing ====
268 # =====
269
270 rm(list = ls())
271
272 source('SDS383D/HW3/spatialsmoothing.R')
273 library(fields)
274
275 weather <- read.csv(file = 'SDS383D-master/data/weather.csv')
276
277
278 #pdf('./SDS383D/HW3/Notes/Img/pressure.pdf', width = 8.5, height = 6)
279 par(mar=c(4,4,2,2), family = 'Palatino')
280 quilt.plot(weather$lon, weather$lat, weather$pressure, xlab = 'Longitude', ylab = 'Latitude',
281           main = 'Heatmap of Pressure')
282 dev.off()
283 #pdf('./SDS383D/HW3/Notes/Img/temperature.pdf', width = 8.5, height = 6)
284 par(mar=c(4,4,2,2), family = 'Palatino')
285 quilt.plot(weather$lon, weather$lat, weather$temperature, xlab = 'Longitude', ylab = 'Latitude',
286           main = 'Heatmap of Temperature')
287 dev.off()
288
289 # Rename the data
290 X <- cbind(weather$lon, weather$lat)
291 z <- weather$pressure
292 w <- weather$temperature
293
294 plot(w, z, pch = 16, cex = 0.8)
295 fit <- lm(z ~ w)
296 abline(a = fit$coefficients[1], b = fit$coefficients[2], lwd = 2, col = 'indianred3')
297 res <- z - fit$coefficients[1] - fit$coefficients[2] * w
298
299 hist(res, col = 'gray', border = 'white', nclass = 20)
300
301 # First guess for sigma:
302 # - temperature data range from -9 to 9 (use sigma2 = 1)
303 # - pressure data range from -600 to 700 (use sigma2 = 10000)
304 sigma2 <- 1000
305
306 # Plot for a guess of the parameters on a coarse grid
307 b <- 2
308 tau1sq <- 10000
309 tau2sq <- 1E-6
310
311 D <- 64
312 longgrid <- seq(min(weather$lon), max(weather$lon), length.out = D)
313 latgrid <- seq(min(weather$lat), max(weather$lat), length.out = D)
314 grid <- expand.grid(longgrid, latgrid)
315
316 pred <- pred.GP(X, res, grid, b, tau1sq, tau2sq, sigma2)
317 quilt.plot(grid[,1], grid[,2], pred, nx = D, ny = D)
318 quilt.plot(weather$lon, weather$lat, res, nx = D, ny = D)
319
320 # Estimate the variance of the model
321 pred <- pred.GP(X, res, X, b, tau1sq, tau2sq, sigma2)

```

```

321 sigma2 <- as.numeric(var(z - pred)) # estimate the variance of the model
322
323 # Pick the hyperparameters by marginal likelihood
324 tau2sq <- 1E-6 # this is fixed
325 D <- 40
326 #b.grid <- seq(0.5, 1.5, length.out = D)
327 b.grid <- seq(1.3, 1.4, length.out = D)
328 #taulsq.grid <- seq(1, 15, length.out = D)
329 taulsq.grid <- seq(40000, 55000, length.out = D)
330
331 marg.loglik = array(NA, dim = c(D, D))
332 for(i in 1:D){
333   for(j in 1:D){
334     marg.loglik[i,j] <- marginal.likelihood(X, z, b.grid[i], taulsq.grid[j], tau2sq, sigma2)
335   }
336   cat('Iteration: ', i, "\n")
337 }
338
339 par(mar=c(4,4.5,2,2), family = 'Palatino')
340 contour(b.grid, taulsq.grid, marg.loglik, nlevels=20, xlab = 'b', ylab = bquote(tau[1]^2), col =
341   'dodgerblue4', lwd = 2)
342
343 b.opt <- b.grid[which(marg.loglik == max(marg.loglik), arr.ind = T)[1]]
344 taulsq.opt <- taulsq.grid[which(marg.loglik == max(marg.loglik), arr.ind = T)[2]]
345 points(b.opt, taulsq.opt, pch = 16, col = 'indianred3')
346
347 # Fit the final model (fine grid, optimal hyperparameters)
348 D <- 64
349 longrid <- seq(min(weather$lon), max(weather$lon), length.out = D)
350 latgrid <- seq(min(weather$lat), max(weather$lat), length.out = D)
351 grid <- expand.grid(longrid, latgrid)
352
353 pred <- pred.GP(X, res, grid, b.opt, taulsq.opt, tau2sq, sigma2)
354
355 #load(file = './SDS383D/HW3/pred_press.Rdata')
356 #load(file = './SDS383D/HW3/pred_temp.Rdata')
357
358 #pdf('./SDS383D/HW3/Notes/Img/tempsmooth.pdf', width = 8.5, height = 6)
359 par(mar=c(4,4,2,2), family = 'Palatino')
360 quilt.plot(grid[,1], grid[,2], pred, xlab = 'Longitude', ylab = 'Latitude', main = 'Heatmap of
361   Temperature', nx = D, ny = D)
362
363 contour(x = longrid, y = latgrid, matrix(pred, D, D, byrow = F), nlevels = 10, add = T, lwd =
364   1.5, labcex = 0.7)
365
366 dev.off()
367
368 pred.mat <- matrix(pred, D, D, byrow = F)
369 # Define colors
370 nbcol = 100
371 color = rev(rainbow(nbcol, start = 0/6, end = 4.5/6))
372 zcol = cut(pred.mat, nbcol)
373 # Plot 3D
374 open3d()
375 persp3d(longrid, latgrid, pred.mat, theta=50, phi=25, expand=0.75, col=color[zcol],
376   ticktype="detailed", xlab="Longitude", ylab="Latitude", zlab="", axes=TRUE, alpha = 0.7)
377 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.