SDS 383D, Exercises 3: Linear smoothing and Gaussian processes

Jan-Michael Cabrera

March 27, 2019

Curve fitting by linear smoothing

(A) Linear smoothing 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 $y_i = \beta x_i + \epsilon_i$.

Show that for the one-predictor case, your prediction $\hat{y}^* = \hat{f}(x^*) = \hat{\beta}x^*$ may be expressed as:

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

We can express $\hat{\beta}$ in matrix notation as,

$$\hat{\beta} = (X^T X)^{-1} X^T Y.$$

Plugging this in to find our predictor, $\hat{y}^*$ we get,

$$\begin{aligned} \hat{y}^* &= x^* (X^T X)^{-1} X^T Y \\ &= \frac{X^T Y}{X^T X} \\ &= \frac{\sum_{i=1}^{n} x_i x^* y_i}{\sum_{i=1}^{n} x_i^2} \end{aligned}$$

The result is of the form $\hat{f}(x^*) = \sum_{i=1}^{n} w(x_i, x^*) y_i$ with the weights expressed as,

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

This resultant *smoother* has constant weights and produces predictions that lie on the line with slope $\hat{\beta}$. If the data is binned and locality preserved, this estimate is similar to a running line average.

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

With *K-nearest-neighbor smoothing* $\hat{y}^*$ is essentially the arithmetic mean of the K y_i's nearest x^*.

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

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

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

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

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.)

```
from __future__ import division
import numpy as np
import matplotlib.pyplot as plt
import sys
sys.path.append('../../scripts/')
from smoothers import kernel_smoother
```

```
# Create noisy data, y_i = sin(x_i) + e_i
x = np.linspace(0, 2*np.pi, num = 20)
y = np.sin(x) + np.random.normal(scale=0.2, size=x.shape)

# Create vector for fitting purposes
x_star = np.linspace(0, 2*np.pi)

# Instantiate array of bandwidths
h = np.array([0.25, 1.0])
```

```
# Instantiate a list to append kernel_smoother objects
H = []

# Iterates through array of bandwidths and passes the feature vector, response vector, and ↩
    bandwidth to kernel_smoother object
for i in range(len(h)):
    H.append(kernel_smoother(x, y, x_star, h=h[i]))
```

```
# Plots data
plt.figure()
plt.plot(x, y, '.k', label='Noisy response')
plt.plot(x, np.sin(x), '--k', label='True function')
# Iterates over the smoother objects and plots functions for the uniform and gaussian kernels
for i in range(len(h)):
    plt.plot(x_star, H[i].predictor(kernel='uniform'), label='Uniform Kernel, h='+str(h[i]))
    plt.plot(x_star, H[i].predictor(kernel='gaussian'), label='Gaussian Kernel, h='+str(h[i]))
plt.legend(loc=0)
# plt.show()
plt.savefig('figures/kernel_smoother.pdf')
plt.close()
```

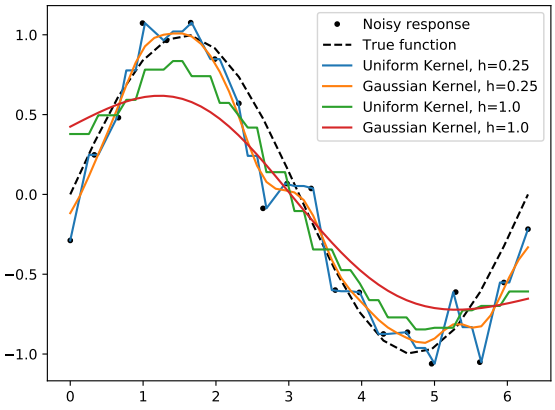


Figure 1: Uniform and Gaussian Kernel smoothing for different bandwidths

Cross validation

(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.

```
from __future__ import division
import numpy as np
import sys
sys.path.append('../../scripts/')
from smoothers import kernel_smoother
```

```
# Function for testing smoothers
def func(x):
    return np.sin(x)

# Initialize random seed
np.random.seed(3)

# Initialize x-vector
x = np.linspace(0, 2*np.pi, num=20)

# Training data of the form, y_i = f(x_i) + \epsilon_i
y_training = func(x) + np.random.normal(scale=0.2, size=x.shape)

# Testing data of the form, y_i = f(x_i) + \epsilon_i
y_test = func(x) + np.random.normal(scale=0.2, size=x.shape)
```

```
# Initialize list for storing kernel_smoother objects
H =[]

# Array of differing bandwidths to evaluate
h = np.array([0.2, 0.3, 0.4, 0.5, 0.6])

# Initialize MSE vector
mse = np.zeros(len(h))
```

```
# Iterates over bandwidth array and finds approximate MSE for each
for i in range(len(h)):
    H.append(kernel_smoother(x, y_training, x, h=h[i]))
    H[i].predictor()
    mse[i] = H[i].MSE(y_test)

print(mse)
```

```
jancabrera@phoenix$ python cross_validation_a.py
[0.07662882 0.0815833  0.0892882  0.10014996 0.11362704]
```

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

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

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

```
from __future__ import division
import numpy as np
import matplotlib.pyplot as plt
import sys
sys.path.append('../../scripts/')
from smoothers import kernel_smoother
```

```
# Function for testing smoothers
def func(x, period = 1):
    return np.sin(x*2*np.pi*period)

# Initialize random seed
np.random.seed(3)

# Initialize x-vector
x = np.linspace(0, 1, num=100)

# Noise level array, low and high
noise = np.array([0.05, 0.25])
noise_label = ['Low', 'High']

# Period of function, high period (wiggly), low period (smooth)
period = np.array([3, 0.5])
period_label = ['Wiggly', 'Smooth']

### Initialize lists for storing the different responses
# Training responses
Y_training = []
# Testing responses
Y_test = []
# True response
T = []
```

```
# Iterates over noise and period arrays to assign data to lists
for n in range(len(noise)):
    for p in range(len(period)):
        Y_training.append(func(x, period=period[p]) + np.random.normal(scale=noise[n], size=x.↩
            shape))
        Y_test.append(func(x, period=period[p]) + np.random.normal(scale=noise[n], size=x.↩
            shape))
        T.append(func(x, period=period[p]))
```

```
# Generates plots
i = 0
for n in range(len(noise)):
    for p in range(len(period)):

        # Instantiate kernel_smoother object
        y_smooth = kernel_smoother(x, Y_training[i], x)
        # Perform initial curve fit
        y_smooth.predictor()

        # Optimize bandwidth given the test data
        y_smooth.optimize_h(Y_test[i])

        # Perform curve fit with optimized bandwidth
        y_smooth.predictor()

        # Predicted values for opimized bandwidth
        y_pred = y_smooth.y_star
```

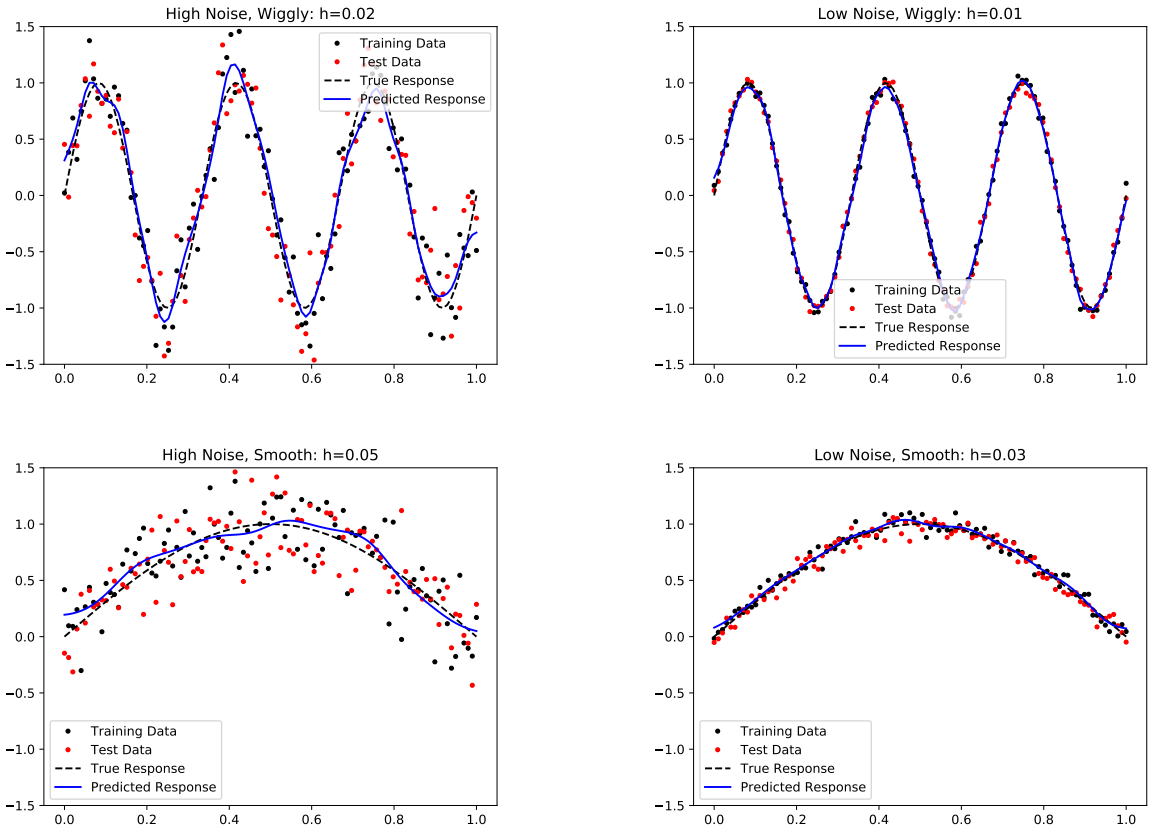


Figure 2: Bandwidths for various functions and noise levels

```
        h_star = y_smooth.h

        plt.figure()
        plt.title(noise_label[n]+' Noise, '+ period_label[p] + ': h={:.2f}'.format(h_star[0]))
        plt.plot(x, Y_training[i], '.k', label='Training Data')
        plt.plot(x, Y_test[i], '.r', label='Test Data')
        plt.plot(x, T[i], '--k', label='True Response')
        plt.plot(x, y_pred, '-b', label='Predicted Response')
        plt.legend(loc=0)
        # plt.show()
        plt.savefig('figures/cross_validation_'+noise_label[n]+'_'+period_label[p]+'.pdf')
        plt.close()
        i += 1
```

Local polynomial regression

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

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

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

calculus, it is easy to see that if the unnormalized weights are

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

then the solution is exactly the kernel-regression estimator.

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

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

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

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

where $w_i \equiv w(x_i, x)$ are the kernel weights defined just above, normalized to sum to one. Derive a concise (matrix) form of the weight vector $\hat{a}$, and by extension, the local function estimate $\hat{f}(x)$ at the target value x. Life will be easier if you define the matrix X_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.

We first begin by plugging in $g_x(x_i; a)$ into the formula for $\hat{a}$, expanding terms and collecting them in matrix form.

$$\begin{aligned}
\hat{a} &= \arg\min_{R^{D+1}} \sum_{i=1}^{n} w_i \left\{ y_i - \left[a_0 + \sum_{k=1}^{D} a_j (x_i - x)^k \right] \right\}^2 \\
&= \arg\min_{R^{D+1}} \sum_{i=1}^{n} w_i \left\{ y_i - a_0 - a_1(x_i - x) - a_2(x_i - x)^2 - \ldots - a_D(x_i - x)^D \right\}^2 \\
&= \arg\min_{R^{D+1}} (y - Ra)^T W (y - Ra)
\end{aligned}$$

We are then interested in minimizing the expression. This is done by setting the derivative with respect to a of the expression to zero. The term in the brackets are expanded.

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

Using the hint from exercise one that $\frac{\partial (z^T A z)}{dz} = (A + A^T) z$, and $\frac{\partial b^T z}{dz} = b$.

$$\begin{aligned}
0 &= -2 y^T W R + 2 R^T W R a \\
y^T W R &= R^T W R a
\end{aligned}$$

We then evaluate the expression and solve for a.

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

Notice that this has the same form as the linear case:

$$\hat{a} = Hy$$

With the hat matrix being, $H = (R^T W R)^{-1} R^T W$. The approximate estimate at our target value is then $\hat{f}(x) = \begin{bmatrix} 1 & 0 & \dots & 0 \end{bmatrix} \hat{a}$

(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

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

From $\hat{f}(x) = \begin{bmatrix} 1 & 0 & \dots & 0 \end{bmatrix} \hat{a}$, we expand the matrices for the $D = 1$ case and multiply the matrices together.

$$\begin{aligned} \hat{f}(x) &= \begin{bmatrix} 1 & 0 \end{bmatrix} \left\{ \begin{bmatrix} (x_1 - x)^0 & (x_1 - x) \\ \vdots & \vdots \\ (x_n - x)^0 & (x_n - x) \end{bmatrix}^T \begin{bmatrix} w_1 & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & w_n \end{bmatrix} \begin{bmatrix} (x_1 - x)^0 & (x_1 - x) \\ \vdots & \vdots \\ (x_n - x)^0 & (x_n - x) \end{bmatrix} \right\}^{-1} \begin{bmatrix} (x_1 - x)^0 & (x_1 - x) \\ \vdots & \vdots \\ (x_n - x)^0 & (x_n - x) \end{bmatrix}^T \begin{bmatrix} w_1 & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & w_n \end{bmatrix} \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix} \\ &= \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} \sum_{i=1}^n w_i & \sum_{i=1}^n w_i (x_i - x) \\ \sum_{i=1}^n w_i (x_i - x) & \sum_{i=1}^n w_i (x_i - x)^2 \end{bmatrix}^{-1} \begin{bmatrix} \sum_{i=1}^n w_i y_i \\ \sum_{i=1}^n (x_i - x) w_i y_i \end{bmatrix} \end{aligned}$$

We note that for a two-by-two matrix the inverse can be found as

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix}^{-1} = \frac{1}{ad - bc} \begin{bmatrix} d & -b \\ -c & a \end{bmatrix}$$

Evaluating the inverse and multiplying the resultant matrices together we obtain the following.

$$\begin{aligned} \hat{f}(x) &= \left[\sum_{i=1}^n w_i \sum_{i=1}^n w_i (x_i - x)^2 - \left(\sum_{i=1}^n w_i (x_i - x) \right)^2 \right]^{-1} \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} \sum_{i=1}^n w_i (x_i - x)^2 & -\sum_{i=1}^n w_i (x_i - x) \\ -\sum_{i=1}^n w_i (x_i - x) & \sum_{i=1}^n w_i \end{bmatrix} \begin{bmatrix} \sum_{i=1}^n w_i y_i \\ \sum_{i=1}^n (x_i - x) w_i y_i \end{bmatrix} \\ &= \left[\sum_{i=1}^n w_i \sum_{i=1}^n w_i (x_i - x)^2 - \left(\sum_{i=1}^n w_i (x_i - x) \right)^2 \right]^{-1} \left[\sum_{i=1}^n w_i (x_i - x)^2 \sum_{i=1}^n w_i y_i - \sum_{i=1}^n w_i (x_i - x) \sum_{i=1}^n w_i y_i (x_i - x) \right] \end{aligned}$$

With $s_1 = \sum_{i=1}^n w_i (x_i - x)$ and $s_2 = \sum_{i=1}^n w_i (x_i - x)^2$ we obtain the form we were seeking:

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

(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.

We start by taking the expectation, noting that $\mathbf{e}_1 = \begin{bmatrix} 1 & 0 & \dots \end{bmatrix}$

$$\begin{aligned} \mathrm{E}\left[\hat{f}(x)\right] &= \mathrm{E}\left[\mathbf{e}_1 H y\right] \\ &= \mathbf{e}_1 H \mathrm{E}[y] \\ &= \mathbf{e}_1 H f(x) \end{aligned}$$

Derivation of the variance of $\hat{f}(x)$ follows a similar logic, using proofs from Exercise 1,

$$\begin{aligned} \mathrm{var}[\hat{f}(x)] &= \mathrm{var}(\mathbf{e}_1 H y) \\ &= \{\mathbf{e}_1 H\}^T \mathrm{var}(y) \{\mathbf{e}_1 H\} \\ &= \sigma^2 \{\mathbf{e}_1 H\}^T \{\mathbf{e}_1 H\} \end{aligned}$$

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

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

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

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

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

Taking the expectation and reducing it to where only the random variable is present results in the following,

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

We then use the quadratic form of the expectation to reduce values in the numerator:

$$\begin{aligned} \mathrm{E}[y^T y] &= \mathrm{tr}(\Sigma) + f(x)^T f(x) \\ &= n\sigma^2 + f(x)^T f(x) \\ \mathrm{E}[y^T H y] &= \mathrm{tr}(H\Sigma) + f(x)^T H f(x) \\ &= \sigma^2 \mathrm{tr}(H) + f(x)^T H f(x) \\ \mathrm{E}[y^T H^T H y] &= \mathrm{tr}(H^T H \Sigma) + f(x)^T H^T H f(x) \\ &= \sigma^2 \mathrm{tr}(H^T H) + f(x)^T H^T H f(x) \end{aligned}$$

Re-introducing these terms into the expression, collecting like terms, and reducing results in the expectation of the variance reducing to,

$$\begin{aligned}
\mathrm{E}\left[\hat{\sigma}^2\right] &= \frac{n\sigma^2 + f(x)^T f(x) - 2\sigma^2 \mathrm{tr}(H) - 2f(x)^T H f(x) + \sigma^2 \mathrm{tr}(H^T H) + f(x)^T H^T H f(x)}{n - 2\mathrm{tr}(H) + \mathrm{tr}(H^T H)} \\
&= \frac{\sigma^2[n - 2\mathrm{tr}(H) + \mathrm{tr}(H^T H)] + f(x)^T f(x) - 2f(x)^T H f(x) + f(x)^T H^T H f(x)}{n - 2\mathrm{tr}(H) + \mathrm{tr}(H^T H)} \\
&= \sigma^2 + \frac{\|f(x) - Hf(x)\|_2^2}{n - 2\mathrm{tr}(H) + \mathrm{tr}(H^T H)}
\end{aligned}$$

We see that that the it is unbiased when the difference, $f(x) - Hf(x)$ is small.

(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.

```
from __future__ import division
import numpy as np
import matplotlib.pyplot as plt
import sys
sys.path.append('../../scripts/')
from smoothers import kernel_smoother
import pandas as pd
```

```
# Import data
data = pd.read_csv('../../data/utilities.csv', delimiter=',')

# Parse data
X = data['temp']
Y = np.log(data['gasbill']/data['billingdays'])

# Sort X data for plotting purposes
x_star = X.drop_duplicates().sort_values().values

# Instantiate smoother object (Default is a Gaussian kernel, order one)
model = kernel_smoother(X, Y, X, h=2)
# Run smoother on the data
model.local_general()
# Optimize the bandwidth using LOOCV
h = model.LOOCV_optimization()
```

```
# Instantiate second model with sorted x-data
optimal = kernel_smoother(X, Y, x_star, h=h)
# Run model
optimal.local_general()

# Plot fit
plt.figure()
plt.plot(X, Y, '.k')
plt.plot(x_star, optimal.y_star, '-b')
plt.xlabel('temperature ($^{\circ}$ F)')
plt.ylabel('normalized gassbill')
# plt.show()
plt.savefig('figures/utilities_fit.pdf')
plt.close()
```

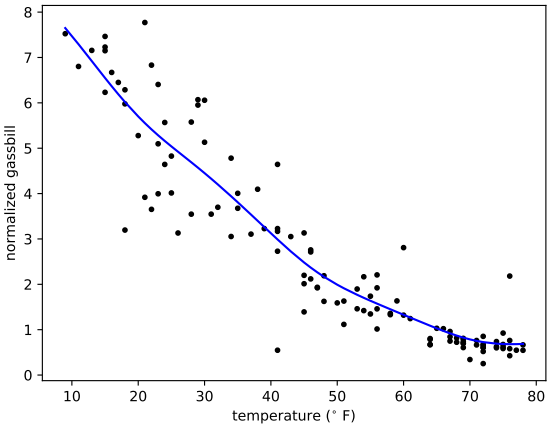


Figure 3: Fit using LOOCV, $h = 6.87$

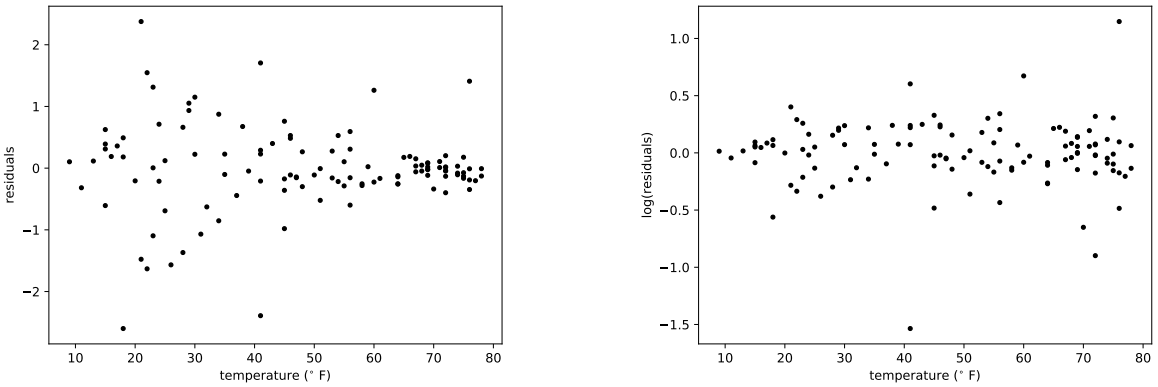


Figure 4: Residuals and log-residuals

(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?

```
# Plot residuals
plt.figure()
plt.plot(X, model.residuals, '.k')
plt.xlabel('temperature ($^{\circ}$ F)')
plt.ylabel('residuals')
# plt.show()
plt.savefig('figures/utilities_residuals.pdf')
plt.close()

# Plot residuals
plt.figure()
plt.plot(X, model.residuals, '.k')
plt.xlabel('temperature ($^{\circ}$ F)')
plt.ylabel('log(residuals)')
# plt.show()
plt.savefig('figures/utilities_residuals_log.pdf')
plt.close()
```

It appears that in the log space, the residuals appear to be relatively homoscedastic.

(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.

```
# Instantiate second model with sorted x-data
optimal = kernel_smoother(X, Y, x_star, h=h)
# Run model
optimal.local_general()

# Establish intervel from model
interval = 1.96*model.sigma

# Sort the interval data for plotting purposes
I = pd.DataFrame(np.transpose([X.values, interval]))
I = I.sort_values(0).drop_duplicates()

# Establish upper and lower bounds for plotting
upper = optimal.y_star + I[1]
lower = optimal.y_star - I[1]
```

```
# Plot fit and confidence interval
plt.figure()
plt.plot(X, Y, '.k')
plt.plot(x_star, optimal.y_star, '-b')
plt.plot(x_star, upper, '-g')
plt.plot(x_star, lower, '-g')
plt.xlabel('temperature ($^{\circ}$ F)')
plt.ylabel('normalized gassbill')
# plt.show()
plt.savefig('figures/utilities_fit_bounds.pdf')
plt.close()
```

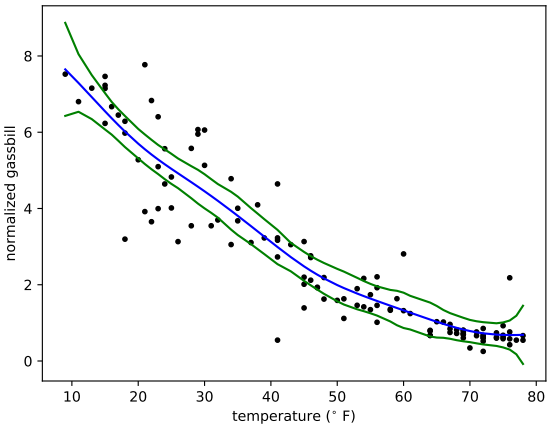


Figure 5: Fit using LOOCV, $h = 6.87$

Guassian 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 $f \sim \mathrm{GP}(m, C)$ 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

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

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 Matern(5/2) 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.

```
from __future__ import division
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import multivariate_normal
```

```
def kronecker_delta(x_1, x_2):
    """
    Parameters
    ----------
        x_1, x_2: floats (scalars)
```

```
    Returns
    ----------
        delta: 0 or 1
            .. math:: \\delta(x_1, x_2)
    """
    if x_1 == x_2:
        delta = 1
    else:
        delta = 0
    return delta

def matern_52(x, hyperparams):
    """
    Parameters
    ----------
        x: float (vector)
            Vector of points

        hyperparams: float (tuple)
            Hyperparameters for a Gaussian process

    Returns
    ----------
        C: float (matrix)
            Returns a Matern (5,2) square covariance matrix of size(x)
            .. math:: C_{5,2}(x_1, x_2) = \\tau_1^2 [1 + \\sqrt{5} d / b + (5/3) (d/b)^2 ] e↩
                ^{-\\sqrt{5} (d/b)} + \\tau_2^2 \\delta(x_1, x_2)
    """

    # Unpack hypereparameters
    b, tau_1_squared, tau_2_squared = hyperparams

    # Initialize covariance matrix
    C = np.zeros((x.shape[0], x.shape[0]))

    # Evaluate (i,j) components of covariance matrix
    for i in range(x.shape[0]):
        for j in range(x.shape[0]):
            d = np.abs(x[i] - x[j])
            C[i][j] = tau_1_squared*(1 + np.sqrt(5)*(d/b) + (5/3)*(d/b)**2)*np.exp(-np.sqrt(5)↩
                *(d/b)) + tau_2_squared*kronecker_delta(x[i], x[j])

    return C
```

```
# Initialize random seed
np.random.seed(3)

# Randomly draw values from a uniform distribution and sort from lowest to highest
x = np.sort(np.random.uniform(size=100))

# Create hyperparameters
b = np.logspace(-1, 1, num=4)
tau_1_squared = np.logspace(-1, 0, num=4)
tau_2_squared = np.logspace(-6, 0, num=4)
```

```
#### Plot varied b
plt.figure()
plt.title('$\\tau_1^2$={:.2f}'.format(tau_1_squared[0])+'; $\\tau_2^2$={:.6f}'.format(↩
    tau_2_squared[0]))
# Iterates over vector of b values and plots them on the same plot
for i in range(len(b)):
    np.random.seed(3)
    hyperparams = b[i], tau_1_squared[0], tau_2_squared[0]
    # Calculates covariance given x and current hyperparameters
    cov = matern_52(x, hyperparams)
```

```
    # Generates random sample from a multivariate normal
    fx = multivariate_normal.rvs(mean=np.zeros(x.shape[0]), cov=cov)
    plt.plot(x, fx, label='b={:.2f}'.format(b[i]))
plt.legend(loc=0)
plt.ylim([-3, 3])
# plt.show()
plt.savefig('figures/matern_52_varied_b.pdf')
plt.close()
```

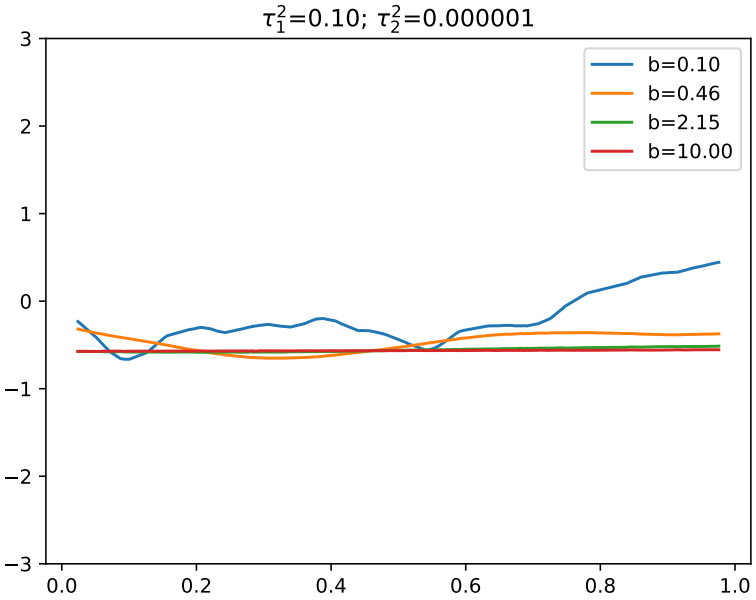


Figure 6: Varied b for Matern(5,2) covariance function

```
#### Plot varied tau_1^2
plt.figure()
plt.title('b={:.2f}'.format(b[i])+'; $\\tau_2^2$={:.6f}'.format(tau_2_squared[0]))
# Iterates over vector of tau_1^2 values and plots them on the same plot
for i in range(len(tau_1_squared)):
    np.random.seed(3)
    hyperparams = b[0], tau_1_squared[i], tau_2_squared[0]
    # Calculates covariance given x and current hyperparameters
    cov = matern_52(x, hyperparams)
    # Generates random sample from a multivariate normal
    fx = multivariate_normal.rvs(mean=np.zeros(x.shape[0]), cov=cov)
    plt.plot(x, fx, label='$\\tau_1^2$={:.2f}'.format(tau_1_squared[i]))
plt.legend(loc=0)
plt.ylim([-3, 3])
# plt.show()
plt.savefig('figures/matern_52_varied_tau_1.pdf')
plt.close()
```

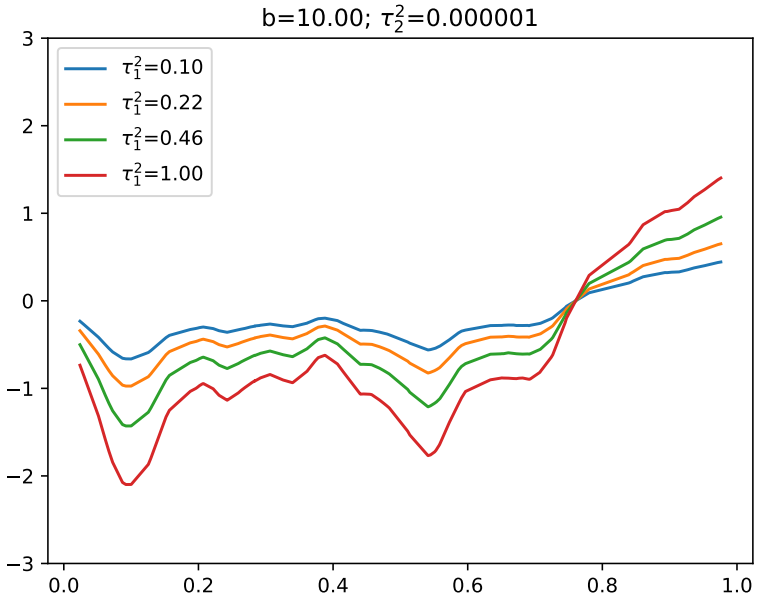


Figure 7: Varied τ_1^2 for Matern(5,2) covariance function

```
#### Plot varied tau_2^2
plt.figure()
plt.title('b={:.2f}'.format(b[i])+'; $\\tau_1^2$={:.2f}'.format(tau_1_squared[0]))
# Iterates over vector of tau_2^2 values and plots them on the same plot
for i in range(len(tau_1_squared)):
    np.random.seed(3)
    hyperparams = b[0], tau_1_squared[0], tau_2_squared[i]
    # Calculates covariance given x and current hyperparameters
    cov = matern_52(x, hyperparams)
    # Generates random sample from a multivariate normal
    fx = multivariate_normal.rvs(mean=np.zeros(x.shape[0]), cov=cov)
    plt.plot(x, fx, label='$\\tau_2^2$={:.6f}'.format(tau_2_squared[i]))
plt.legend(loc=0)
plt.ylim([-3, 3])
# plt.show()
plt.savefig('figures/matern_52_varied_tau_2.pdf')
plt.close()
```

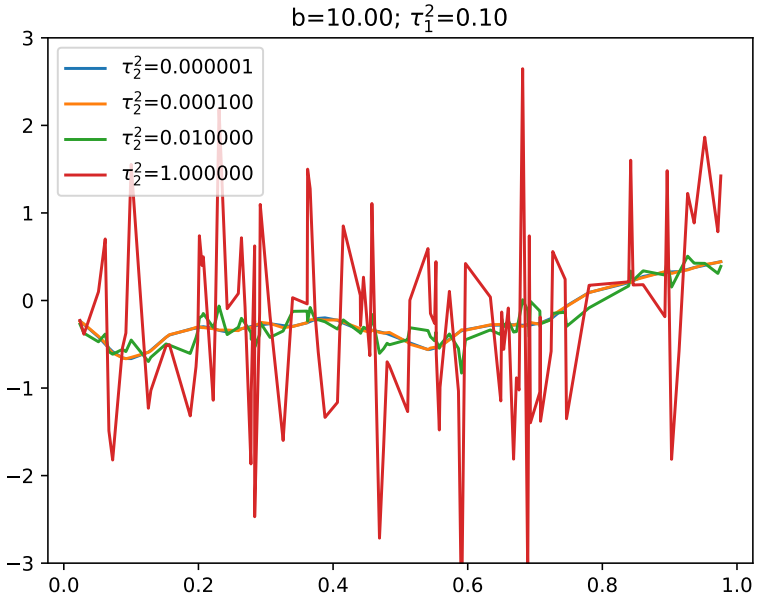


Figure 8: Varied τ_2^2 for Matern(5,2) covariance function

```
##### Plot b vs tau_1^2
fig, ax = plt.subplots(b.shape[0], tau_1_squared.shape[0], sharex='col', sharey='row')
fig.subplots_adjust(hspace=0.3, wspace=0.2)
fig.suptitle('$\\tau_2^2$={:.6f}'.format(tau_2_squared[0]))
fig.set_size_inches(8,4.5)
for i in range(b.shape[0]):
    for j in range(tau_1_squared.shape[0]):
        np.random.seed(3)
        ax[i, j].set_title('b={:.2f}'.format(b[i]) + '; $\\tau_1^2$={:.2f}'.format(↩
            tau_1_squared[j]), fontsize=8)
        hyperparams = b[i], tau_1_squared[j], tau_2_squared[0]
        # Calculates covariance given x and current hyperparameters
        cov = matern_52(x, hyperparams)
        # Generates random sample from a multivariate normal
        fx = multivariate_normal.rvs(mean=np.zeros(x.shape[0]), cov=cov)
        ax[i, j].plot(x, fx, '-k')
# plt.show()
plt.savefig('figures/matern_52_b_vs_tau_1.pdf')
```

- b: Controls the smoothness of the resulting function. As b increases, the smoothness increases until it approaches a constant.
- τ_1^2: appears to control how much the resulting function varies from the mean at a given point.
- τ_2^2: seems to increase the noise as its value increases.

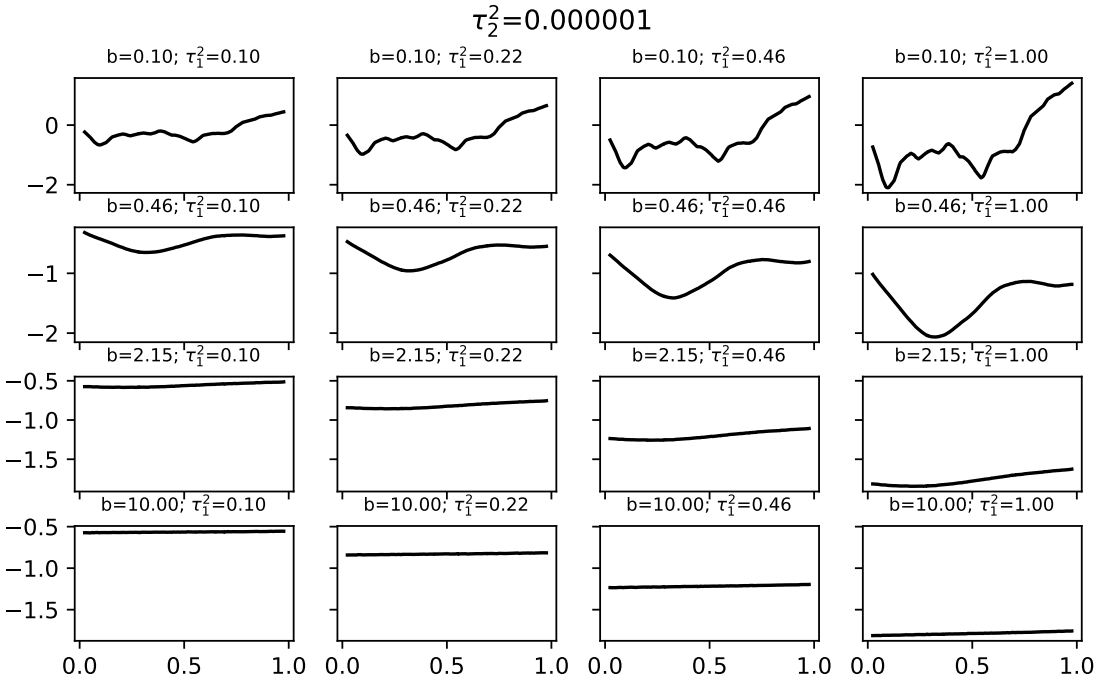


Figure 9: Varied b and τ_1^2 for Matern(5,2) covariance function

(B) Suppose you observe the value of a Gaussian process $f \sim \mathrm{GP}(m, C)$ at points $x_1, \ldots, x_N$. 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.

The gaussian process can be written as,

$$[f(x_1), ..., f(x_N)]^T \sim \mathrm{N}(m, C)$$
$$\mathrm{N}(m, C) = \mathrm{N}\left(\begin{bmatrix} m_1 \\ \vdots \\ m_n \end{bmatrix}, \begin{bmatrix} C_{11} & \ldots & C_{1n} \\ \vdots & \ddots & \vdots \\ C_{n1} & \ldots & C_{nn} \end{bmatrix} \right)$$

We then construct the joint distribution considering the new point x^*,

$$[f(x_1), ..., f(x_N), f(x^*)]^T \sim \mathrm{N}\left(\begin{bmatrix} m_1 \\ \vdots \\ m_n \\ m^* \end{bmatrix}, \begin{bmatrix} C_{11} & \ldots & C_{1n} & C_{1*} \\ \vdots & \ddots & \vdots & \vdots \\ C_{n1} & \ldots & C_{nn} & C_{n*} \\ C_{*1} & \ldots & C_{*n} & C_{**} \end{bmatrix} \right)$$
$$\sim \mathrm{N}\left(\begin{bmatrix} \mathbf{m} \\ m^* \end{bmatrix}, \begin{bmatrix} C(\mathbf{x}, \mathbf{x}) & C(\mathbf{x}^*\mathbf{x}) \\ C(\mathbf{x}^*\mathbf{x})^T & C(\mathbf{x}^*, \mathbf{x}^*) \end{bmatrix} \right)$$

From this joint distribution we can find the conditional distribution for the function at x^* using the following derived in exercises 1:

$$x_1 | x_2 \sim \mathrm{N}(\mu_1 + \Sigma_{22}^{-1}\Sigma_{12}^T(x_2 - \mu_2), \Sigma_{11} - \Sigma_{12}\Sigma_{22}^{-1}\Sigma_{12}^T)$$

The resulting function is then,

$$f(x^*)|f(x_1), ..., f(x_N) \sim \mathrm{N}\left(m^* + C(\mathbf{x}, \mathbf{x})^{-1}C(\mathbf{x}^*\mathbf{x})^T(f(x_1), ..., f(x_N) - \mathbf{m})\right), C(\mathbf{x}^*, \mathbf{x}^*) - C(\mathbf{x}^*\mathbf{x})C(\mathbf{x}, \mathbf{x})^{-1}C(\mathbf{x}^*\mathbf{x})^T$$

Note, it is common to let the mean functions equal zero so that $f \sim \mathrm{GP}(0, C)$.

(C) Prove the following lemma.

Lemma 1 *Suppose that the joint distribution of two vectors y and θ has the following properties: (1) the conditional distribution for y given θ is multivariate normal, $(y \mid \theta) \sim N(R\theta, \Sigma)$; and (2) the marginal distribution of θ is multivariate normal, $\theta \sim N(m, V)$. Assume that R, Σ, m, and V are all constants. Then the joint distribution of y and θ is multivariate normal.*

The model for y can be written as

$$y = R\theta + \epsilon$$

where

$$y \sim \mathrm{N}(R\theta, \Sigma)$$
$$\epsilon \sim \mathrm{N}(m, V)$$

We can then show that $y = R\theta + \epsilon$ and $\theta = \theta + 0$ can be written

$$\begin{bmatrix} y \\ \theta \end{bmatrix} = \begin{bmatrix} R \\ I \end{bmatrix} \theta + \begin{bmatrix} I \\ 0 \end{bmatrix} \epsilon$$

This is an affine transformation of multivariate normals, hence it is also multivariate normal.

Nonparameteric 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: $f \sim \mathrm{GP}(0, C)$ for some covariance function C. Let $x_1, \ldots, x_N$ denote the previously observed x points. Derive the posterior distribution for the random vector $[f(x_1), \ldots, f(x_N)]^T$, given the corresponding outcomes $y_1, \ldots, y_N$, assuming that you know σ^2.

Our prior is,

$$p(f) \propto \exp\left[-\frac{1}{2}(f)^T C^{-1}(f)\right]$$

And the likelihood function is,

$$p(y|f) \propto \exp\left[-\frac{1}{2}(y-f)^T \sigma^2 I(y-f)\right]$$

We then find the posterior distribution,

$$\begin{aligned} p(f|y) &\propto p(f)p(y|f) \\ &\propto \exp\left[-\frac{1}{2}(f)^T C^{-1}(f)\right] \exp\left[-\frac{1}{2}(y-f)^T \sigma^2 I(y-f)\right] \end{aligned}$$

Note that from exercises 1 we had the following,

$$\theta|x_1, ..., x_n \sim \mathrm{N}\left(\frac{\frac{n}{\sigma^2}\bar{x} + \frac{1}{v}m}{\frac{n}{\sigma^2}+\frac{1}{v}}, \left[\frac{n}{\sigma^2}+\frac{1}{v}\right]^{-1}\right)$$

Following a similar logic we get,

$$f|y \sim \mathrm{N}\left(\frac{\frac{1}{\sigma^2}y}{C^{-1}+\frac{1}{\sigma^2}I}, \left[\frac{1}{\sigma^2}I + C^{-1}\right]^{-1}\right)$$

(B) As before, suppose we observe data $y_i = f(x_i) + \epsilon_i$, $\epsilon_i \sim N(0,\sigma^2)$, for $i = 1, \ldots, N$. 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 $E\{f(x^\star) \mid y_1, \ldots, y_N\}$ is a linear smoother, and derive expressions both for the smoothing weights and the posterior variance of $f(x^\star)$.

$$\begin{bmatrix} y \\ f^* \end{bmatrix} \sim \mathrm{N}\left(0, \begin{bmatrix} C(x,x)+\sigma^2 I & C(x^*,x)^T \\ C(x^*,x) & C(x^*,x^*) \end{bmatrix}\right)$$

$$x_1|x_2 \sim \mathrm{N}(\mu_1 + \Sigma_{22}^{-1}\Sigma_{12}^T(x_2-\mu_2), \Sigma_{11} - \Sigma_{12}\Sigma_{22}^{-1}\Sigma_{12}^T)$$

$$f^*|y \sim \mathrm{N}\left(0 + (C(x,x)+\sigma^2 I)^{-1}C(x^*,x), \quad C(x^*,x^*) - C(x^*,x)[C(x,x)+\sigma^2 I]^{-1}C(x^*,x)^T\right)$$

$$\mathrm{E}[f^*|y] = \sum_{i=1}^{n} w_i y_i = w^T y$$

$$w = C(x^*,x)[C(x,x)+\sigma^2 I]^{-1}$$

$$\mathrm{var}[f^*|y] = C(x^*,x^*) - C(x^*,x)(C(x,x)+\sigma^2 I)^{-1}C(x^*,x)^T$$

(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.

```
from __future__ import division
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import multivariate_normal
import sys
sys.path.append('../../scripts/')
from gaussian_process import gaussian_process
import pandas as pd
```

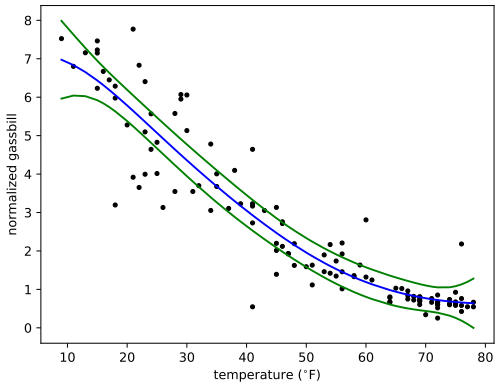
```
# Import data
data = pd.read_csv('../../data/utilities.csv', delimiter=',')

# Parse data
X = data['temp']
Y = data['gasbill']/data['billingdays']

# Sort X data for plotting purposes
x_star = X.drop_duplicates().sort_values().values
```

```
# Set hyperparameters
b = 30
tau_1_squared = 10
tau_2_squared = 10**-6

# Pack hyperparameters for passing to model
hyperparams = b, tau_1_squared, tau_2_squared

# Create a guassian process object from data and prediction vector
GP = gaussian_process(X, hyperparams, y=Y, x_star=x_star, cov='squared_exponential')
```

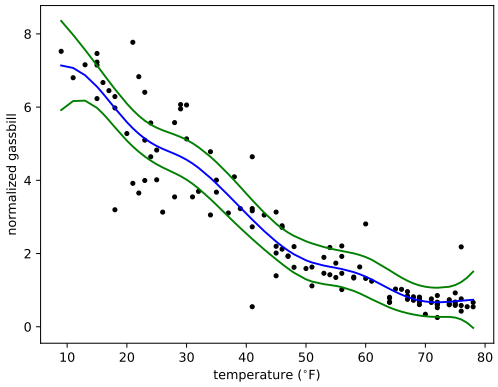
```
# Create a guassian process object from data and prediction vector
GP = gaussian_process(X, hyperparams, y=Y, x_star=x_star, cov='squared_exponential')

var = 1

# Run the GP smoother with the approximated variance
y_star, variance = GP.smoother(variance = var)

# Calculate credible interval
upper = y_star + np.sqrt(variance)*1.96
lower = y_star - np.sqrt(variance)*1.96
```

```
# Plot fit and credible interval
plt.figure()
plt.plot(X, Y, '.k')
plt.plot(x_star, y_star, '-b')
plt.plot(x_star, upper, '-g')
plt.plot(x_star, lower, '-g')
plt.xlabel('temperature ($^{\circ}$F)')
plt.ylabel('normalized gassbill')
# plt.show()
plt.savefig('figures/utilities_fit_gp_squared_exponential.pdf')
plt.close()
```

Note here that the code was re-run using the Matern (5,2) covariance function.



(a) Squared exponential



(b) Matern (5,2)

Figure 10: Gaussian process fit to utilities data using squared exponential and Matern(5,2) covariance functions

(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_2^1, range b, and nugget τ_2^2. Derive an expression for the marginal distribution of $y = (y_1 \ldots, y_N)$ in terms of (τ_1^2, b, τ_2^2), integrating out the random function f. This is called a marginal likelihood.

The marginal likelihood can be found by integrating out f from the joint distribution $p(y, f)$:

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

Like terms are gathered in the exponential,

$$\begin{aligned} &f^T C^{-1} f + (y-f)^T \left(\frac{1}{\sigma^2}I\right)(y-f) = \\ &f^T C^{-1} f + y^T \left(\frac{1}{\sigma^2}I\right) y - 2f^T \left(\frac{1}{\sigma^2}I\right) y + f^T \left(\frac{1}{\sigma^2}I\right) f = \\ &f^T \left(C^{-1} + \frac{1}{\sigma^2}I\right) f - 2f^T \left(\frac{1}{\sigma^2}I\right) y + y^T \left(\frac{1}{\sigma^2}I\right) y \end{aligned}$$

From this we aim to complete the square for f noting that $x^T Ax + x^T b + c = (x-h)^T A(x-h) + k$ with $h = -\frac{1}{2}A^{-1}b$ and $k = c - \frac{1}{4}b^T A^{-1} b$.

$$f^T Af + f^T b + c = (f-h)^T A(f-h) + k$$

The corresponding terms for the above are,

$$\begin{aligned} A &= \left(C^{-1} + \frac{1}{\sigma^2}I\right) \\ b &= -2\left(\frac{1}{\sigma^2}I\right) y \\ h &= -\frac{1}{2}A^{-1}b = \left(C^{-1} + \frac{1}{\sigma^2}I\right)^{-1} \left(\frac{1}{\sigma^2}I\right) y \\ k &= c - \frac{1}{4}b^T A^{-1} b = y^T \left(\frac{1}{\sigma^2}I\right) y - y^T \left(\frac{1}{\sigma^2}I\right) \left(C^{-1} + \frac{1}{\sigma^2}I\right)^{-1} \left(\frac{1}{\sigma^2}I\right) y \end{aligned}$$

We reintroduce this back into the exponential term. The term $\int \exp\left[-\frac{1}{2}(f-h)^T A(f-h)\right] df$ integrates to a constant not dependent on y.

$$\begin{aligned} p(y) &\propto \exp\left[-\frac{1}{2}k\right] \int \exp\left[-\frac{1}{2}(f-h)^T A(f-h)\right] df \\ &\propto \exp\left[-\frac{1}{2}\left\{y^T \left(\frac{1}{\sigma^2}I\right) y - y^T \left(\frac{1}{\sigma^2}I\right) \left(C^{-1} + \frac{1}{\sigma^2}I\right)^{-1} \left(\frac{1}{\sigma^2}I\right) y\right\}\right] \\ &\propto \exp\left[-\frac{1}{2}y^T \left\{\left(\frac{1}{\sigma^2}I\right) - \left(\frac{1}{\sigma^2}I\right) \left(C^{-1} + \frac{1}{\sigma^2}I\right)^{-1} \left(\frac{1}{\sigma^2}I\right)\right\} y\right] \end{aligned}$$

If we let $B = \left(\frac{1}{\sigma^2} I\right)$ and $D = C^{-1}$, we can reduce the term in the parenthesis using the following identity,

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

Which reduces to,

$$\left(\frac{1}{\sigma^2} I\right) - \left(\frac{1}{\sigma^2} I\right)\left(\frac{1}{\sigma^2} I + C^{-1}\right)^{-1}\left(\frac{1}{\sigma^2} I\right) = \left(\left(\frac{1}{\sigma^2} I\right)^{-1} + C\right)^{-1}$$

Note that with $\left(\frac{1}{\sigma^2} I\right)^{-1} = \sigma^2 I$ the marginal for $p(y)$ becomes,

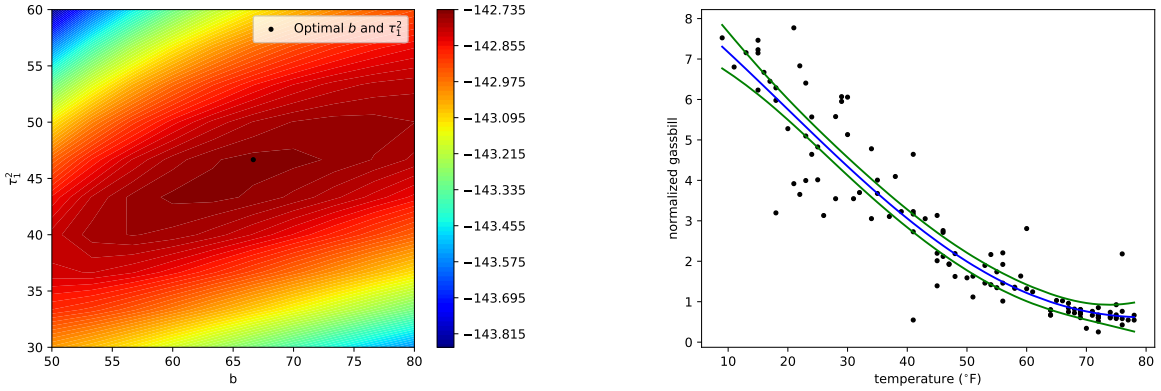
$$p(y) \propto \exp\left[-\frac{1}{2} y^T \left(\sigma^2 I + C\right)^{-1} y\right]$$

$$y \sim \mathrm{N}\left(0, \sigma^2 I + C\right)$$

(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.

```
from __future__ import division
import numpy as np
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt
from matplotlib import cm
from matplotlib.ticker import LinearLocator, FormatStrFormatter
from scipy.stats import multivariate_normal
import sys
sys.path.append('../../scripts/')
from gaussian_process import gaussian_process
from gaussian_process import covariance_functions
import pandas as pd
```

```
# Import data
data = pd.read_csv('../../data/utilities.csv', delimiter=',')

# Parse data
X = data['temp']
Y = data['gasbill']/data['billingdays']

# Sort X data for plotting purposes
x_star = X.drop_duplicates().sort_values().values

# Set hyperparameters
b = 20
tau_1_squared = 10
tau_2_squared = 10**-6

# Pack hyperparameters for passing to model
hyperparams = b, tau_1_squared, tau_2_squared

# Create a guassian process object from data and prediction vector
GP = gaussian_process(X, hyperparams, y=Y, x_star=x_star, cov='squared_exponential')
```



(a) Optimal b and τ_1^2

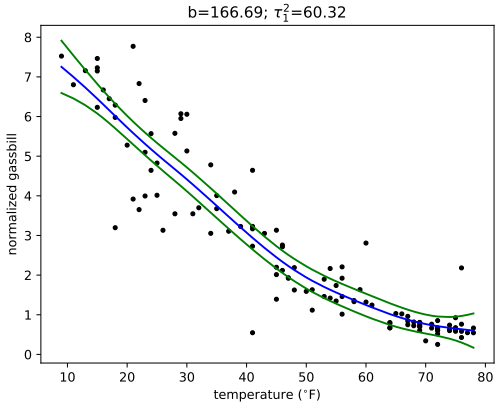
(b) Optimal fit

Figure 11: Gaussian process fit to utilities data using squared exponential with optimal $b = 67$, $\tau_1^2 = 47$

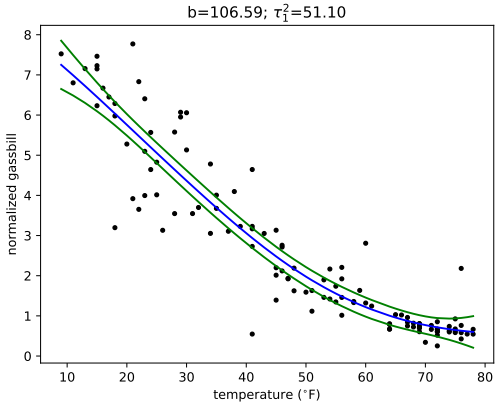
```
# Create arrays for evaluating and saving the hyperparameters
b = np.linspace(50, 80, num=10)
tau_1_squared = np.linspace(30, 60, num=10)
Z = np.zeros((len(b), len(tau_1_squared)))

# Approximate the variance by fitting a GP once and evaluating the residual sum of squared errors
variance = GP.approx_var()

# Loops over each b and tau_1_squared and evaluates the log of the marginal likelihood
for i in range(len(b)):
    for j in range(len(tau_1_squared)):
        hyperparams = b[i], tau_1_squared[j], tau_2_squared
        Z[i][j] = GP.log_marginal_likelihood(hyperparams, variance=variance)
```

```
# Chooses the maximum value from the array of caclulated log marginal likelihoods
ind = np.unravel_index(np.argmax(Z, axis=None), Z.shape)

# Optimal values from the above
b_hat = b[ind[0]]
tau_1_hat = tau_1_squared[ind[1]]

# Prints the optimal values (these are evaluated on the dataset using a different script)
print(b_hat, tau_1_hat)

# For plotting purposes
b, tau_1_squared = np.meshgrid(b, tau_1_squared)
```

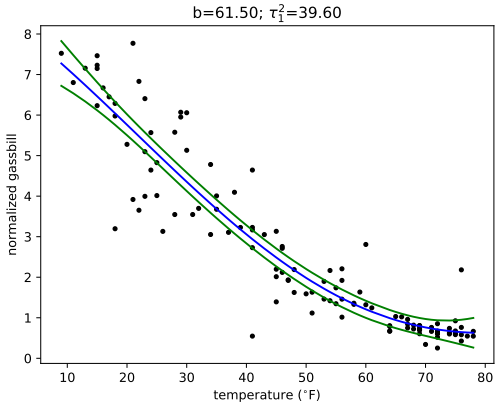
```
# Plots contour of log marginal likelihood vs b and tau_1_squared
plt.figure()
plt.contourf(b, tau_1_squared, Z, 100, cmap='jet')
plt.plot(b_hat, tau_1_hat, '.k', label='Optimal $b$ and $\\tau_1^2$')
plt.xlabel('b')
plt.ylabel('$\\tau_1^2$')
plt.colorbar()
plt.legend(loc=0)
# plt.show()
plt.savefig('figures/optimal_b_tau_squared_exponential.pdf')
```



(a) Matern (3,2)



(b) Matern (5,2)



(c) Squared exponential

Figure 12: Gaussian process fit to utilities data using BFGS optimization for hyperparameters

(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 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.

```
from __future__ import division
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import multivariate_normal
import sys
from mpl_toolkits import mplot3d
sys.path.append('../../scripts/')
from gaussian_process import gaussian_process
from gaussian_process import covariance_functions
import pandas as pd
from matplotlib.mlab import griddata
```

```
# Import data
data = pd.read_csv('../../data/weather.csv', delimiter=',')

# Parse data into Y and X components
pressure = data['pressure'] # Y_1
temperature = data['temperature'] # Y_2
longitude = data['lon']
latitude = data['lat']
X = np.transpose(np.array((longitude, latitude))) # X
```

```
### Build up array to evaluate accross a grid of values
# Specifies bounds and number of points to divide the bounds into
num = 30
lon = np.linspace(longitude.min(), longitude.max(), num=num)
lat = np.linspace(latitude.min(), latitude.max(), num=num)

# Builds the grid from the previous lines (num x num Array)
lon_1, lat_1 = np.meshgrid(lon, lat)

# Converts the arrays into vectors (num^2 x 1)
X1 = np.ravel(lon_1)
X2 = np.ravel(lat_1)

# Builds an array from unraveled vectors (num^2 x 2)
x_star = np.array((X1, X2)).T
```

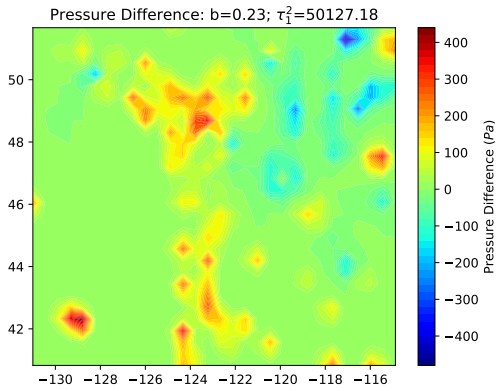
```
#### Pressure GP
# Initialize hyperparameters and fit a GP to observed data
```

```
p_hyperparams = 0.23, 50127.18, 10**-6
P = gaussian_process(X, p_hyperparams, y=pressure, x_star=x_star)
P.optimize_lml() # optimizes b, and tau_1_squared
P_star, P_var = P.smoother()

# Reshapes outputs of smoother for plotting purposes
P_star = P_star.reshape(lon_1.shape)
P_var = P_var.reshape(lon_1.shape)
```

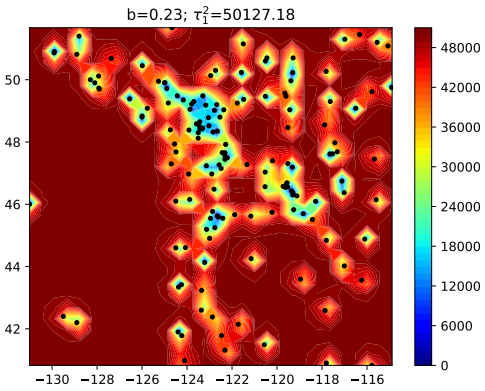
```
#### Temperature GP
# Initialize hyperparameters and fit a GP to observed data
t_hyperparams = 0.99, 6.07, 10**-6
T = gaussian_process(X, t_hyperparams, y=temperature, x_star=x_star)
T.optimize_lml() # optimizes b, and tau_1_squared
T_star, T_var = T.smoother()

# Reshapes outputs of smoother for plotting purposes
T_star = T_star.reshape(lon_1.shape)
T_var = T_var.reshape(lon_1.shape)
```

```
levels = 50

#### Pressure
plt.figure()
plt.title('Pressure Difference: b={:.2f}'.format(P.hyperparams[0])+'; $\\tau_1^2$={:.2f}'.↩
    format(P.hyperparams[1]))
CS = plt.contourf(lon, lat, P_star, levels, cmap='jet') #, vmax=pressure.max(), vmin=pressure.↩
    min())
plt.colorbar(label='Pressure Difference ($Pa$)')
# plt.show()
plt.savefig('figures/weather_pressure.pdf')

#### Pressure variance
plt.figure()
plt.title('b={:.2f}'.format(P.hyperparams[0])+'; $\\tau_1^2$={:.2f}'.format(P.hyperparams[1]))
CS = plt.contourf(lon, lat, P_var, levels, cmap='jet', vmax=P_var.max(), vmin=P_var.min())
plt.plot(longitude, latitude, '.k')
plt.colorbar()
# plt.show()
plt.savefig('figures/weather_pressure_var.pdf')
```

```
#### Temperature
plt.figure()
plt.title('Temperature Difference: b={:.2f}'.format(T.hyperparams[0])+'; $\\tau_1^2$={:.2f}'.↩
    format(T.hyperparams[1]))
CS = plt.contourf(lon, lat, T_star, levels, cmap='jet') #, vmax=temperature.max(), vmin=↩
    temperature.min())
plt.colorbar(label='Temperature Difference ($^{\circ}$C)')
# plt.show()
plt.savefig('figures/weather_temperature.pdf')

#### Temperature variance
plt.figure()
plt.title('b={:.2f}'.format(T.hyperparams[0])+'; $\\tau_1^2$={:.2f}'.format(T.hyperparams[1]))
CS = plt.contourf(lon, lat, T_var, levels, cmap='jet', vmax=T_var.max(), vmin=T_var.min())
plt.plot(longitude, latitude, '.k')
plt.colorbar()
# plt.show()
plt.savefig('figures/weather_temperature_var.pdf')
```

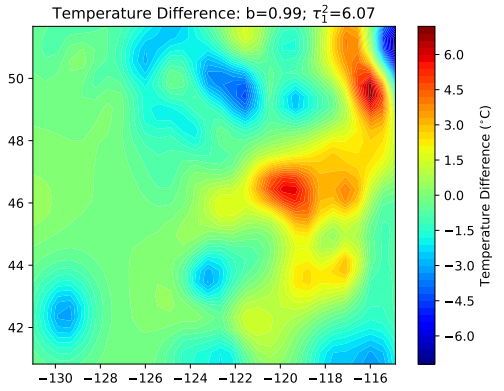


(a) Smoothed pressure

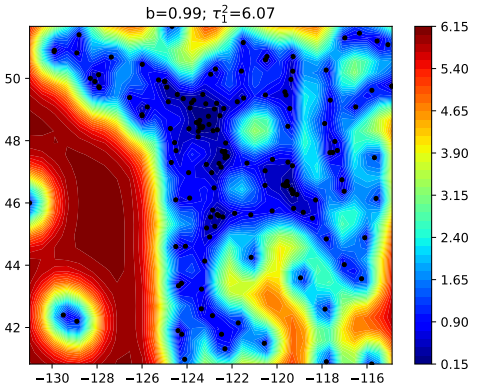


(b) Posterior variance and weather station loacation

Figure 13: Gaussian process fit to weather data using Matern (5,2)



(a) Smoothed temperature



(b) Posterior variance and weather station loacation

Figure 14: Gaussian process fit to weather data using Matern (5,2)