Exercises 8

Gaussian processes in numpy

Exercise 8.1 Sample from GP prior
In this exercises we will write the code needed to draw and plot samples of f from a Gaussian process prior with squared exponential (or, equivalently, RBF) kernel

$$f \sim \mathcal{GP}(m, \kappa), \qquad \text{whith } m(x) = 0 \text{ and } \kappa(x, x') = \sigma_f^2 e^{-\frac{1}{2\ell^2}\|x-x'\|^2}$$

To implement this, we choose a vector of m test input points $\mathbf{x}_*$. We will choose $\mathbf{x}_*$ to contain sufficiently many points, such that it will *appear* as a continuous line on the screen. We then evaluate the $m \times m$ covariance matrix $\kappa(\mathbf{x}_*, \mathbf{x}_*)$ and thereafter generate samples from the multivariate normal distribution

$$f(\mathbf{x}_*) \sim \mathcal{N}\left(m(\mathbf{x}_*),\ \kappa(\mathbf{x}_*, \mathbf{x}_*)\right).$$

(a) Use `numpy.linspace` to construct a vector $\mathbf{x}_*$ with $m = 101$ elements equally spaced from -5 to 5.

(b) Construct a mean vector $m(\mathbf{x}_*)$ with 101 elements all equal to zero and the 101×101 covariance matrix $\kappa(x_*, x_*)$. The expression for $\kappa(\cdot, \cdot)$ is given above. Let the hyperparameters be $\ell = 2$ and $\sigma_f^2 = 1$.

(c) Use `scipy.stats.multivariate_normal` (you might need to use the option `allow_singular=True`) to draw 25 samples $f^{(1)}(\mathbf{x}_*), \dots f^{(25)}(\mathbf{x}_*)$ from the multivariate normal distribution $f(\mathbf{x}_*) \sim \mathcal{N}\left(m(\mathbf{x}_*),\ \kappa(\mathbf{x}_*, \mathbf{x}_*)\right)$.

(d) Plot the samples $f^{(1)}(\mathbf{x}_*), \dots f^{(25)}(\mathbf{x}_*)$ versus the input vector $\mathbf{x}_*$.

(e) Try another value of ℓ and repeat (b)-(d). How do the two plots differ, and why?

Exercise 8.2 GP posterior
In this exercise we will perform Gaussian process regression. That means, based on the n observations $\mathcal{D} = \{x_i, f(x_i)\}_{i=1}^n$ and the prior belief $f \sim \mathcal{GP}(0, \kappa(x, x'))$, we want to find the posterior $p(f|\mathcal{D})$. (In the previous problem, we were only concerned with the prior $p(f)$, not conditioned on having observed the data $\mathcal{D}$.) We consider the same Gaussian process prior (same mean $m(x)$ and $\kappa(x, x')$ and hyperparameters) as in the previous exercise.

(a) Construct two vectors $\mathbf{x} = [-4, -3, -1, 0, 2]^\mathsf{T}$ and $\mathbf{f} = [-2, 0, 1, 2, -1]^\mathsf{T}$, which will be our training data (that is, $n = 5$).

(b) Keep $\mathbf{x}_*$ as in the previous problem. In addition to the $m \times m$ matrix $\kappa(\mathbf{x}_*, \mathbf{x}_*)$, now also compute the $n \times m$ matrix $\kappa(\mathbf{x}, \mathbf{x}_*)$ and the $n \times n$ matrix $\kappa(\mathbf{x}, \mathbf{x})$. Hint: You might find it useful to define a function that returns $\kappa(x, x')$, taking x and x' as arguments.

(c) Use the training data $\mathbf{x}$, $\mathbf{y}$ and the matrices constructed in (b) to compute the posterior mean $\boldsymbol{\mu}_{\text{posterior}}$ and the posterior covariance $\mathbf{K}_{\text{posterior}}$ for $\mathbf{x}_*$, by using the equations for conditional multivariate normal distributions.

(d) In a similar manner as in (c) and (d) in the previous problem, draw 25 samples from the multivariate distribution $f(\mathbf{x}_*) \sim \mathcal{N}\left(\boldsymbol{\mu}_{\text{posterior}},\ \mathbf{K}_{\text{posterior}}\right)$ and plot these samples ($f^{(j)}(\mathbf{x}_*)$ vs. $\mathbf{x}_*$) together with the posterior mean ($\boldsymbol{\mu}_{\text{posterior}}$ vs. $\mathbf{x}_*$) and the actual measurements ($\mathbf{y}$ vs. $\mathbf{x}$). How do the samples in this plot differ from the prior samples in the previous problem?

(e) Instead of plotting samples, plot a credibility region. Here, a credibility region is based on the (marginal) posterior variance. The 68% credibility region, for example, is the area between $\boldsymbol{\mu}_{\text{posterior}} - \sqrt{\mathbf{K}^d_{\text{posterior}}}$ and $\boldsymbol{\mu}_{\text{posterior}} + \sqrt{\mathbf{K}^d_{\text{posterior}}}$, where $\mathbf{K}^d_{\text{posterior}}$ is a vector with the diagonal elements of $\mathbf{K}_{\text{posterior}}$. What is the connection between the credibility regions and the samples you drew previously?

(f) Now, consider the setting where the measurements are corrupted with noise, $y_i = f(\mathbf{x}_i) + \varepsilon$, $\varepsilon \sim \mathcal{N}\left(0,\ \sigma^2\right)$. Use $\sigma = 0.1$ and repeat (c)-(e) with this modification of the model. What is the difference in comparison to the previous plot? What is the interpretation?

(g) Explore what happens with another length scale ℓ.

Exercise 8.3 Other covariance functions/kernels

The squared exponential kernel/covariance function gives samples which are smooth and infinitely continuously differentiable. Other kernels make other assumptions. Now try the previous problems using the exponential kernel instead,

$$\kappa(x, x') = e^{-\frac{1}{\ell}|x-x'|}.$$

Exercise 8.4 Learning hyperparameters

Until now, we have made GP regression using predefined hyperparameters, such as the lengthscale ℓ and noise variance σ^2. In this exercise, we will estimate ℓ and σ^2 from the data by maximizing the marginal likelihood. The logarithm of the marginal likelihood for a Gaussian process observed with Gaussian noise is

$$p\{\mathbf{y}|\mathbf{x}, \ell\} = -\frac{1}{2}\mathbf{y}^\mathsf{T}\mathbf{K}_y^{-1}\mathbf{y} - \frac{1}{2}\log|\mathbf{K}_y| - \frac{n}{2}\log 2\pi$$

where $\mathbf{K}_y = \kappa(\mathbf{x}, \mathbf{x}) + \sigma^2\mathbf{I}$.

(a) Write a function that takes $\mathbf{x}$, $\mathbf{y}$, ℓ and σ^2 as inputs and produces the marginal likelihood as output for the squared exponential covariance function.

(b) Consider the same data as before. Use $\sigma^2 = 0$ and compute the marginal likelihood for values of ℓ between 0.1 and 1 and plot it. What seems to be the maximal value of the marginal likelihood o this interval? Do GP regression based on this value of ℓ.

Exercise 8.5 Learning hyperparameters II

In this exercise we investigate a setting where the marginal likelihood has multiple local minima.

(a) Now, consider the following data

$$\mathbf{x} = \begin{bmatrix}-5 & -3 & 0 & 0.1 & 1 & 4.9 & 5\end{bmatrix}^\mathsf{T}, \qquad \mathbf{y} = \begin{bmatrix}0 & -0.5 & 1 & 0.7 & 0 & 1 & 0.7\end{bmatrix}^\mathsf{T}$$

and compute the log marginal likelihood for both ℓ and σ. Use a logarithmic 2D-grid for values of ℓ spanning from 10^{-1} to 10^2 and for σ^2 spanning from 10^{-2} to 10^0. Visualize the marginal likelihood on that grid with a contour plot.

(b) Find the hyperparameters ℓ and σ^2 that correspond to the maximal marginal likelihood. Perform GP regression on the data using these hyperparameters.

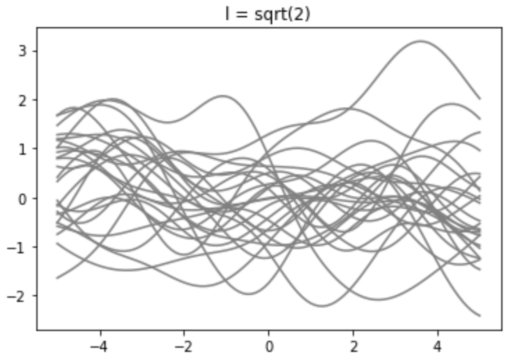
(c) Perform GP regression for the hyperparameters that correspond to other possible local optima of the marginal likelihood. What differences do you see in your posterior?

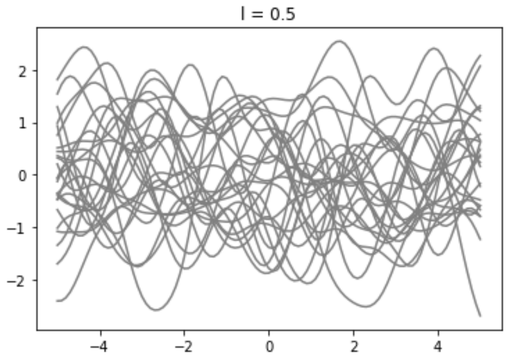
Solutions 8

Gaussian processes in numpy

Solution to Exercise 8.1

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

m = 101
xs = np.linspace(-5,5,m) # Test input vector
mxs = np.zeros(m) # Zero mean vector

l = np.sqrt(2) # hyperparameters
sf2 = 1
Kss = sf2*np.exp(-1/(2*l**2)*np.abs(xs[:,np.newaxis]-xs[:,np.newaxis].T)**2) # Covariance matrix

s = 25 # Draw samples from the prior
fs = multivariate_normal(mean=mxs,cov=Kss,allow_singular=True).rvs(s).T

plt.plot(xs,fs,'gray') # Plot the samples
plt.title('l = sqrt(2)')
plt.show()

# Use another length scale
l = 0.5
Kss = sf2*np.exp(-1/(2*l)*np.abs(xs[:,np.newaxis]-xs[:,np.newaxis].T)**2)
fs = multivariate_normal(mean=mxs,cov=Kss,allow_singular=True).rvs(s).T
plt.plot(xs,fs,'gray')
plt.title('l = 0.5')
plt.show()
```





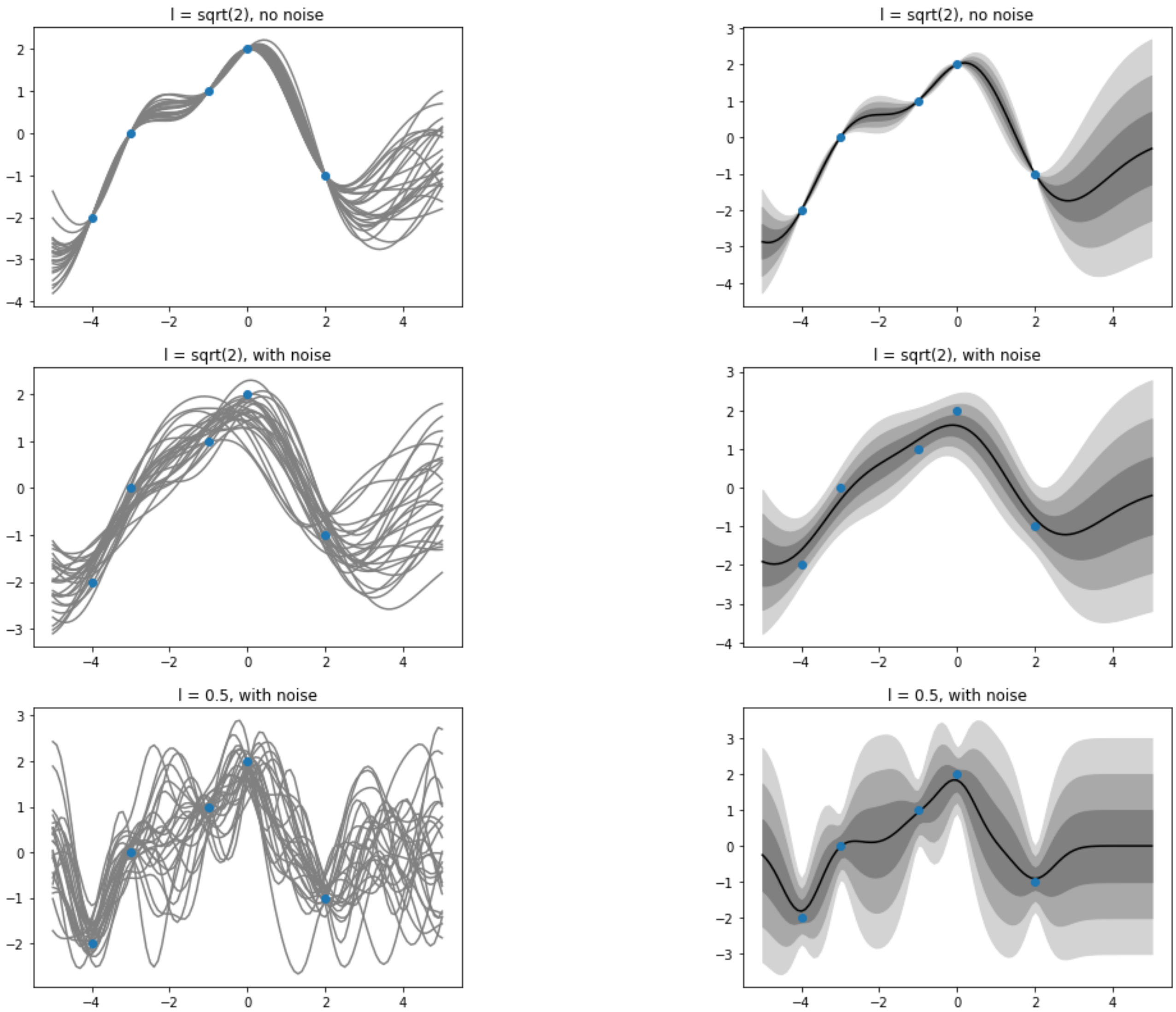
Solution to Exercise 8.2 (d) All posterior samples passes through the observed data points (the prior samples do not necessarily do that). This is natural, since the posterior distribution of $f(\mathbf{x}_*)$ is conditioned on the observations, and must the posterior distribution pass through them.

(e) The 68% credibility region, for example, contains 68% of the posterior samples.

(f) The posterior distribution does not pass exactly through all observations any more, but the observations are not to some extent "explained" as noise.

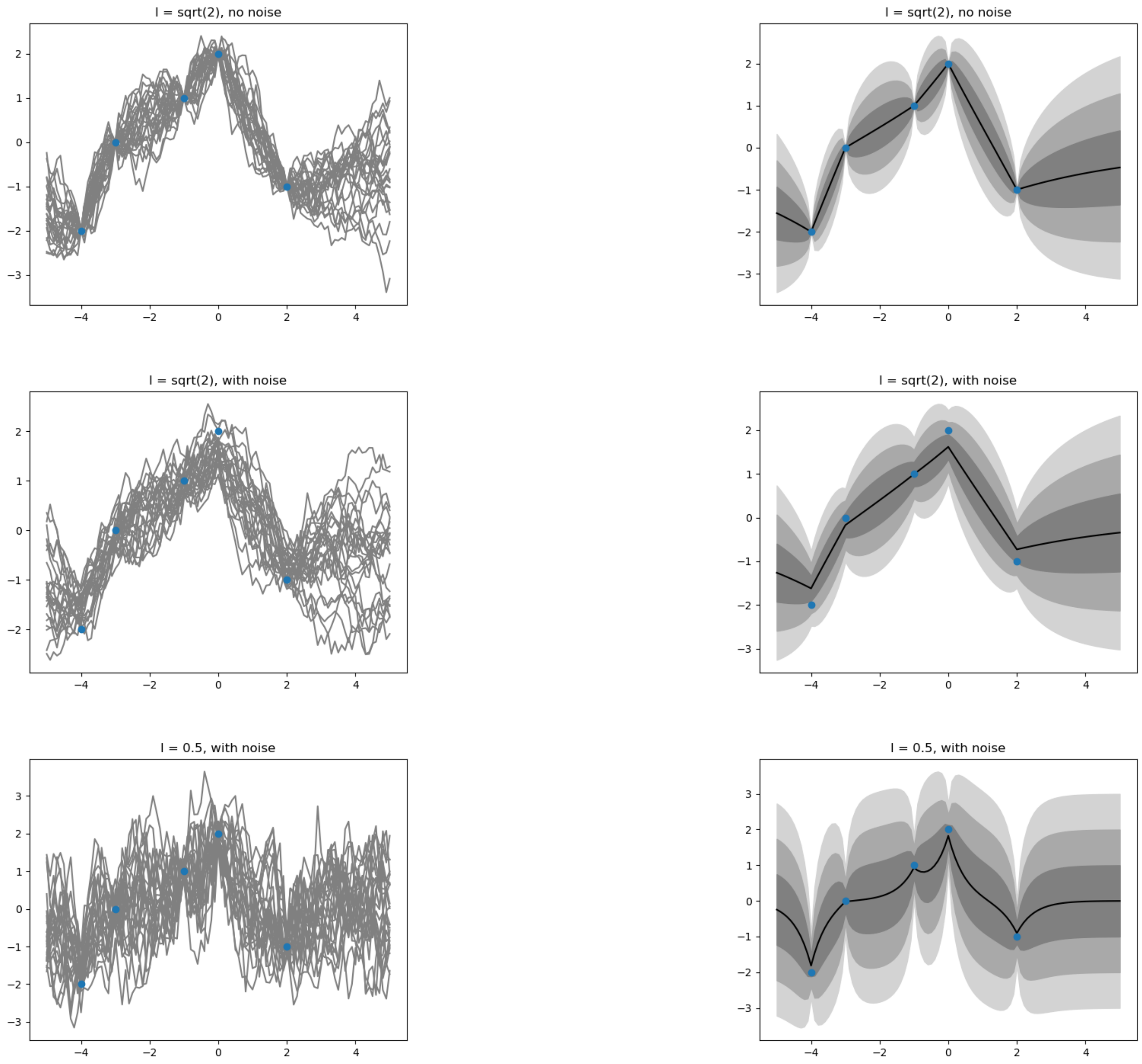
```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import multivariate_normal

n = 5
x=np.array([-4,-3,-1,0,2]) # Observed inputs
f=np.array([-2,0,1,2,-1]) # Observed function values

m = 101
xs = np.linspace(-5,5,m) # Test input vector

l = np.sqrt(2) # hyperparameters
sf2 = 1
def k(x,xp,l):
    return sf2*np.exp(-1/(2*l**2)*np.abs(x[:,np.newaxis]-xp[:,np.newaxis].T)**2)
Kss =  k(xs,xs,l)
Ks =  k(x,xs,l)
K = k(x,x,l)

mu_post = (Ks.T@np.linalg.inv(K))@f
K_post = Kss - Ks.T@np.linalg.inv(K)@Ks

s = 25 # Draw samples from the posterior
fs = multivariate_normal(mean=mu_post,cov=K_post,allow_singular=True).rvs(s).T

plt.plot(xs,fs,'gray') # Plot the samples
plt.scatter(x,f,zorder=3)
plt.title('l = sqrt(2), no noise')
plt.show()

plt.plot(xs,mu_post,'black') # Plot credibility regions
plt.fill_between(xs,mu_post + 3*np.sqrt(np.diag(K_post)),mu_post - 3*np.sqrt(np.diag(K_post)),color=
    'lightgray')
plt.fill_between(xs,mu_post + 2*np.sqrt(np.diag(K_post)),mu_post - 2*np.sqrt(np.diag(K_post)),color=
    'darkgray')
plt.fill_between(xs,mu_post + 1*np.sqrt(np.diag(K_post)),mu_post - 1*np.sqrt(np.diag(K_post)),color=
    'gray')
plt.scatter(x,f,zorder=3)
plt.title('l = sqrt(2), no noise')
plt.show()

# Include measurement noise
K = k(x,x,l) + 0.1*np.eye(n)
mu_post = (Ks.T@np.linalg.inv(K))@(f)
K_post = Kss - Ks.T@np.linalg.inv(K)@Ks
fs = multivariate_normal(mean=mu_post,cov=K_post,allow_singular=True).rvs(s).T

plt.plot(xs,fs,'gray') # Plot the samples
plt.scatter(x,f,zorder=3)
plt.title('l = sqrt(2), with noise')
plt.show()

plt.plot(xs,mu_post,'black') # Plot credibility regions
plt.fill_between(xs,mu_post + 3*np.sqrt(np.diag(K_post)),mu_post - 3*np.sqrt(np.diag(K_post)),color=
    'lightgray')
plt.fill_between(xs,mu_post + 2*np.sqrt(np.diag(K_post)),mu_post - 2*np.sqrt(np.diag(K_post)),color=
    'darkgray')
plt.fill_between(xs,mu_post + 1*np.sqrt(np.diag(K_post)),mu_post - 1*np.sqrt(np.diag(K_post)),color=
    'gray')
plt.scatter(x,f,zorder=3)
plt.title('l = sqrt(2), with noise')
plt.show()
```

```
# Try another length scale
l = 0.5
Kss =  k(xs,xs,l)
Ks =  k(x,xs,l)
K = k(x,x,l) + 0.1*np.eye(n)
mu_post = (Ks.T@np.linalg.inv(K))@(f)
K_post = Kss - Ks.T@np.linalg.inv(K)@Ks
fs = multivariate_normal(mean=mu_post,cov=K_post,allow_singular=True).rvs(s).T
plt.plot(xs,fs,'gray')
plt.scatter(x,f,zorder=3)
plt.title('l = 0.5, with noise')
plt.show()
plt.plot(xs,mu_post,'black')
plt.fill_between(xs,mu_post + 3*np.sqrt(np.diag(K_post)),mu_post - 3*np.sqrt(np.diag(K_post)),color=
    'lightgray')
plt.fill_between(xs,mu_post + 2*np.sqrt(np.diag(K_post)),mu_post - 2*np.sqrt(np.diag(K_post)),color=
    'darkgray')
plt.fill_between(xs,mu_post + 1*np.sqrt(np.diag(K_post)),mu_post - 1*np.sqrt(np.diag(K_post)),color=
    'gray')
plt.scatter(x,f,zorder=3)
plt.title('l = 0.5, with noise')
plt.show()
```



Solution to Exercise 8.3
Change the kernel definition in the code for exercise 8.2 with the following

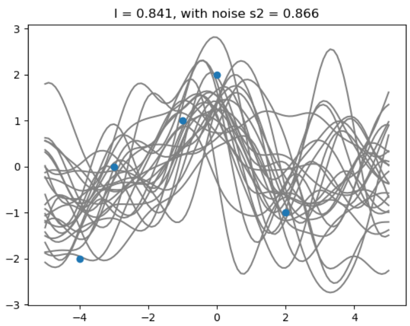
```
def k(x,xp,l):
    return sf2*np.exp(-1/(2*l**2)*np.abs(x[:,np.newaxis]-xp[:,np.newaxis].T))
```

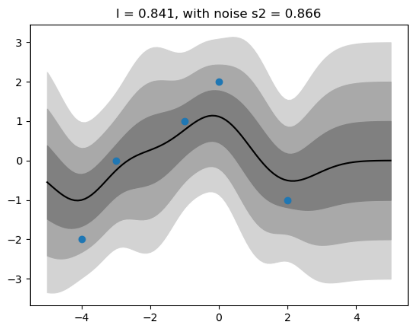


Solution to Exercise 8.4 (a) We use `scipy.optimize.minimize`

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import multivariate_normal
from scipy.optimize import minimize

n = 5
x=np.array([-4,-3,-1,0,2]) # Observed inputs
f=np.array([-2,0,1,2,-1]) # Observed function values

m = 101
xs = np.linspace(-5,5,m) # Test input vector


# Kernel function
def k(x, xp, l, sf2=1):
    return sf2*np.exp(-1/(2*l**2)*np.abs(x[:,np.newaxis]-xp[:,np.newaxis].T)**2)

```

```python
# marginal likelihood
def neg_log_marg_lik(y, x, l, s2): #
    K = k(x, x, l) + s2*np.eye(len(x))
    return 0.5*(y.T@np.linalg.inv(K)@y + np.log(np.linalg.det(K)) + len(x)*np.log(2*np.pi))

# Optimize the function
def optimal_hyperparameters(y, x, l=np.sqrt(2), s2=0):
    result = minimize(lambda hyper: neg_log_marg_lik(y, x, hyper[0], hyper[1]), [l, s2])
    return result.x[0], result.x[1]

# Create optimal kernel
l, s2 = optimal_hyperparameters(f, x)
K = k(x, x, l=l) + s2*np.eye(n)
Ks = k(x, xs, l=l)
Kss = k(xs, xs, l=l)

mu_post = (Ks.T@np.linalg.inv(K))@f
K_post = Kss - Ks.T@np.linalg.inv(K)@Ks

s = 25 # Draw samples from the posterior
fs = multivariate_normal(mean=mu_post,cov=K_post,allow_singular=True).rvs(s).T

mu_post = (Ks.T@np.linalg.inv(K))@(f)
K_post = Kss - Ks.T@np.linalg.inv(K)@Ks
fs = multivariate_normal(mean=mu_post,cov=K_post,allow_singular=True).rvs(s).T

plt.plot(xs,fs,'gray') # Plot the samples
plt.scatter(x,f,zorder=3)
plt.title(f'l = {l:.3f}, with noise s2 = {s2:.3f}')
plt.show()

plt.plot(xs,mu_post,'black') # Plot credibility regions
plt.fill_between(xs,mu_post + 3*np.sqrt(np.diag(K_post)),mu_post - 3*np.sqrt(np.diag(K_post)), color='lightgray')
plt.fill_between(xs,mu_post + 2*np.sqrt(np.diag(K_post)),mu_post - 2*np.sqrt(np.diag(K_post)), color='darkgray')
plt.fill_between(xs,mu_post + 1*np.sqrt(np.diag(K_post)),mu_post - 1*np.sqrt(np.diag(K_post)), color='gray')
plt.scatter(x,f,zorder=3)
plt.title(f'l = {l:.3f}, with noise s2 = {s2:.3f}')
plt.show()

# Same data, kernel, and marginal likelihood function as before
```

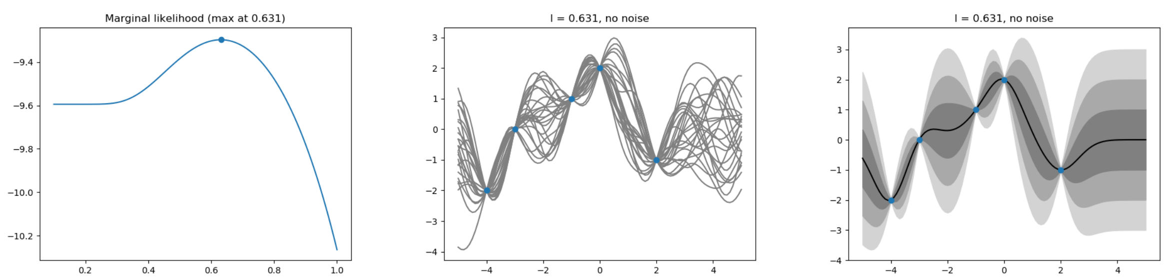




(b)

```python
# Same data, kernel, and marginal likelihood function as before

# Create grid for l
nl = 101
cl = np.linspace(0.1, 1, nl)
marg_lik = np.zeros(nl)

# Compute marginal likelihood over the grid
for i in range(nl):
    marg_lik[i] = -neg_log_marg_lik(f, x, l=cl[i], s2=0)
```

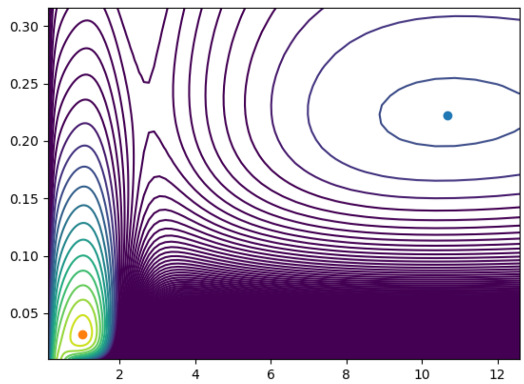
```

idx = np.argmax(marg_lik)
lmax = cl[idx]

plt.plot(cl, marg_lik)
plt.scatter(cl[idx], marg_lik[idx])
plt.title(f'Marginal likelihood (max at {lmax:.3f})')
plt.show()

# Create optimal kernel
K = k(x, x, l=lmax)
Ks = k(x, xs, l=lmax)
Kss = k(xs, xs, l=lmax)

mu_post = (Ks.T@np.linalg.inv(K))@f
K_post = Kss - Ks.T@np.linalg.inv(K)@Ks

s = 25 # Draw samples from the posterior
fs = multivariate_normal(mean=mu_post,cov=K_post,allow_singular=True).rvs(s).T

mu_post = (Ks.T@np.linalg.inv(K))@(f)
K_post = Kss - Ks.T@np.linalg.inv(K)@Ks
fs = multivariate_normal(mean=mu_post,cov=K_post,allow_singular=True).rvs(s).T

plt.plot(xs,fs,'gray') # Plot the samples
plt.scatter(x,f,zorder=3)
plt.title(f'l = {lmax:.3f}, no noise')
plt.show()

plt.plot(xs,mu_post,'black') # Plot credibility regions
# Add a small value to prevent negative variances
plt.fill_between(xs,mu_post + 3*np.sqrt(np.diag(K_post) + 0.0001),mu_post - 3*np.sqrt(np.diag(K_post)+0.0001),color='lightgray')
plt.fill_between(xs,mu_post + 2*np.sqrt(np.diag(K_post) + 0.0001),mu_post - 2*np.sqrt(np.diag(K_post)+0.0001),color='darkgray')
plt.fill_between(xs,mu_post + 1*np.sqrt(np.diag(K_post) + 0.0001),mu_post - 1*np.sqrt(np.diag(K_post)+0.0001),color='gray')
plt.scatter(x,f,zorder=3)
plt.title(f'l = {lmax:.3f}, no noise')
plt.show()
```



Solution to Exercise 8.5 (a)

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import multivariate_normal
from scipy.optimize import minimize

x=np.array([-5,-3,0,0.1,1,4.9,5])
y=np.array([0,-0.5,1,0.7,0,1,0.7])
n = len(x)

m = 101
xs = np.linspace(-6,6,m) # Test input vector


# Kernel function
def k(x, xp, l, sf2=1):
```

```
    return sf2*np.exp(-1/(2*l**2)*np.abs(x[:,np.newaxis]-xp[:,np.newaxis].T)**2)

# marginal likelihood
def neg_log_marg_lik(y, x, l, s2): #
    K = k(x, x, l) + s2*np.eye(len(x))
    return 0.5*(y.T@np.linalg.inv(K)@y + np.log(np.linalg.det(K)) + len(x)*np.log(2*np.pi))

nl = 100
ns = 100
cl = np.logspace(-1, 1.1, nl)
cs = np.logspace(-2, -0.5, ns)

L, S = np.meshgrid(cl, cs)
Z = np.zeros((nl, ns))

for i in range(nl):
    for j in range(ns):
        Z[i,j] = - neg_log_marg_lik(y, x, L[i,j], S[i,j])


plt.contour(L, S, Z, 500, vmin=-7)
plt.show()
```

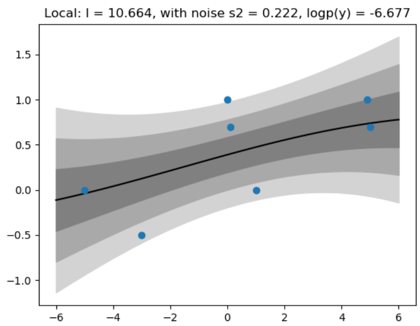


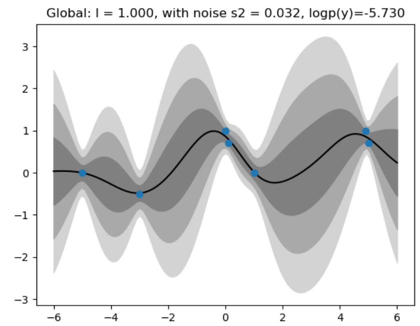
(b)

```
# Local maximum
local_max = minimize(lambda hyper: neg_log_marg_lik(y, x, hyper[0], hyper[1]), [10, 0.2])
print(local_max.message)

# Create first optimal kernel
l, s2 = local_max.x
K = k(x, x, l=l) + s2*np.eye(n)
Ks = k(x, xs, l=l)
Kss = k(xs, xs, l=l)

mu_post = (Ks.T@np.linalg.inv(K))@y
K_post = Kss - Ks.T@np.linalg.inv(K)@Ks

plt.plot(xs,mu_post,'black') # Plot credibility regions
plt.fill_between(xs,mu_post + 3*np.sqrt(np.diag(K_post)),mu_post - 3*np.sqrt(np.diag(K_post)),
    color='lightgray')
plt.fill_between(xs,mu_post + 2*np.sqrt(np.diag(K_post)),mu_post - 2*np.sqrt(np.diag(K_post)),
    color='darkgray')
plt.fill_between(xs,mu_post + 1*np.sqrt(np.diag(K_post)),mu_post - 1*np.sqrt(np.diag(K_post)),
    color='gray')
plt.scatter(x,y,zorder=3)
plt.title(f'Local: l = {l:.3f}, with noise s2 = {s2:.3f}, logp(y) = {-local_max.fun:.3f}')
plt.show()
```

(c)

```
# Constrained optimization to find the global maximum
from scipy.optimize import LinearConstraint
global_max = minimize(lambda hyper: neg_log_marg_lik(y, x, hyper[0], hyper[1]), [1, 0.002],
    constraints=[LinearConstraint(np.eye(2), [0,0], [1, 0.05])])
print(global_max.message)

# Create second optimal kernel
l, s2 = global_max.x
K = k(x, x, l=l) + s2*np.eye(n)
Ks = k(x, xs, l=l)
Kss = k(xs, xs, l=l)

mu_post = (Ks.T@np.linalg.inv(K))@y
K_post = Kss - Ks.T@np.linalg.inv(K)@Ks

plt.plot(xs,mu_post,'black') # Plot credibility regions
plt.fill_between(xs,mu_post + 3*np.sqrt(np.diag(K_post)),mu_post - 3*np.sqrt(np.diag(K_post)),
    color='lightgray')
plt.fill_between(xs,mu_post + 2*np.sqrt(np.diag(K_post)),mu_post - 2*np.sqrt(np.diag(K_post)),
    color='darkgray')
plt.fill_between(xs,mu_post + 1*np.sqrt(np.diag(K_post)),mu_post - 1*np.sqrt(np.diag(K_post)),
    color='gray')
plt.scatter(x,y,zorder=3)
plt.title(f'Global: l = {l:.3f}, with noise s2 = {s2:.3f}, logp(y)={-global_max.fun:.3f}')
plt.show()
```





Bibliography

[1] David Barber. *Bayesian reasoning and machine learning*. Cambridge University Press, 2012.

[2] Christopher M Bishop. *Pattern recognition and machine learning*. springer, 2006.

[3] Kevin B Korb and Ann E Nicholson. *Bayesian artificial intelligence*. CRC press, 2010.

[4] Steffen L Lauritzen and David J Spiegelhalter. Local computations with probabilities on graphical structures and their application to expert systems. *Journal of the Royal Statistical Society: Series B (Methodological)*, 50(2):157–194, 1988.

[5] Kevin P Murphy. *Machine learning: a probabilistic perspective*. MIT press, 2012.