02477 Bayesian Machine Learning 2025: Assignment 2

This is the second assignment out of three in the Bayesian machine learning course 2025. The assignment is a group work of 3-5 students (please use the same groups as in assignment 1 if possible) and hand in via DTU Learn). The assignment is **mandatory**. The deadline is **6th of April 23:59**. The solution must be handed in as a single **PDF** document.

Part 1: Gaussian processes and covariance functions

In this part, we will study covariance functions for Gaussian process models. Consider the following six covariance functions

$$k_1(x,x') = 2\exp\left(-\frac{(x-x')^2}{2\cdot 0.3^2}\right) \tag{1}$$

$$k_2(x,x') = \exp\left(-\frac{(x-x')^2}{2\cdot 0.1^2}\right) \tag{2}$$

$$k_3(x,x') = 4 + 2xx' \tag{3}$$

$$k_4(x,x') = \exp\left(-2\sin\left(3\pi\cdot|x-x'|\right)^2\right) \tag{4}$$

$$k_5(x,x') = \exp\left(-2\sin\left(3\pi\cdot|x-x'|\right)^2\right) + 4xx' \tag{5}$$

$$k_6(x,x') = \frac{1}{5} + \min(x,x') \tag{6}$$

Some of them should be familiar and some of them might be new to you.

Task 1.1: Determine the analytical marginal prior mean and variance of a Gaussian process, $f_i(x) \sim \mathcal{GP}(0, k_i(x,x'))$, i.e. compute $\mathbb{E}\left[f_i(x)\right]$ and $\mathbb{V}\left[f_i(x)\right]$ for each of the six covariance functions (for $i = 1, 2, \dots, 6$).

Solution

The prior mean for each process is 0 by construction. The marginal prior variance of a Gaussian process is given by $\mathbb{V}\left[f_i(x)\right] = k_i(x,x)$. Therefore, we have

$$\mathbb{V}\left[f_1(x)\right] = 2\exp\left(-\frac{0^2}{2\cdot 0.3^2}\right) = 2 \tag{7}$$

$$\mathbb{V}\left[f_2(x)\right] = \exp\left(-\frac{0^2}{2\cdot 0.1^2}\right) = 1 \tag{8}$$

$$\mathbb{V}\left[f_3(x)\right] = 4 + 2xx = 4 + 2x^2 \tag{9}$$

$$\mathbb{V}\left[f_4(x)\right] = \exp\left(-2\sin\left(3\pi\cdot|0|\right)^2\right) = 1 \tag{10}$$

$$\mathbb{V}\left[f_5(x)\right] = \exp\left(-2\sin\left(3\pi\cdot|0|\right)^2\right) + 4x^2 = 1 + 4x^2 \tag{11}$$

$$\mathbb{V}\left[f_6(x)\right] = \frac{1}{5} + \min(x,x) = \frac{1}{5} + x \tag{12}$$

End of solution

Task 1.2: Which of the six covariance functions are stationary covariance functions?

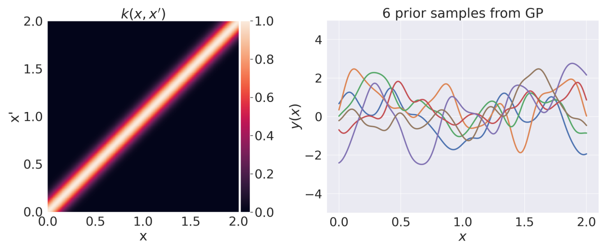
Solution

Covariance functions k_1, k_2, and k_4 are stationary, whereas k_3, k_5 and k_6 are not. For example, if we let $\tau = x - x'$, then we can rewrite k_1 as a function of τ, i.e.

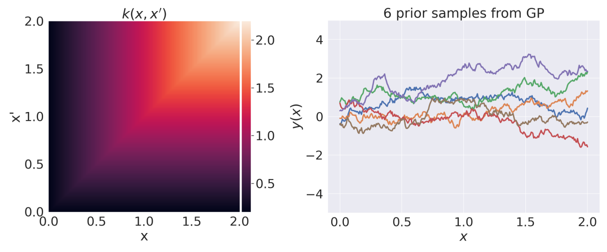
$$k_1(x, x') = k_1(\tau) = 2\exp\left(-\frac{\tau^2}{2 \cdot 0.3^2}\right), \tag{13}$$

which is not possible for the non-stationary functions.

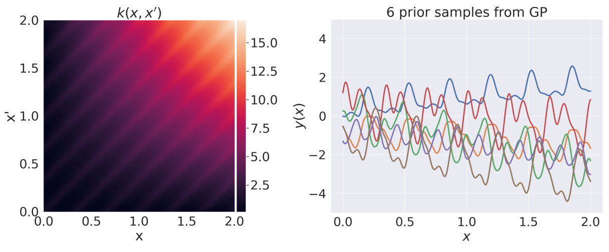
End of solution



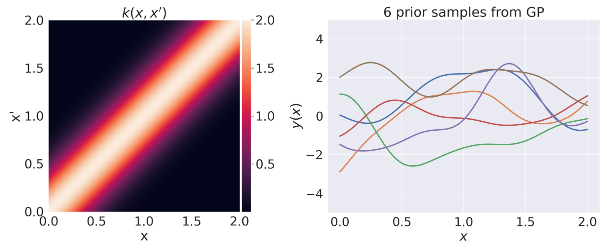
(a)



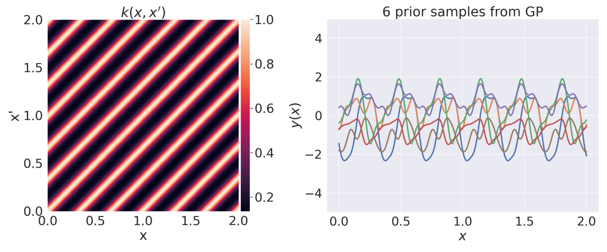
(b)



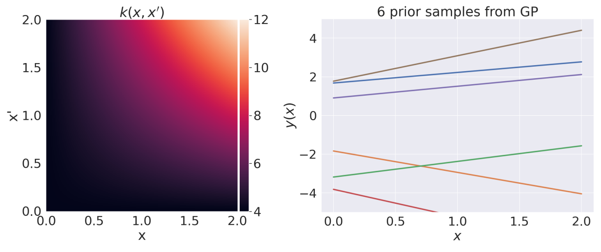
(c)



(d)



(e)



(f)

Figure 1: Covariance matrices and samples from the corresponding Gaussian process prior distribution for the six different covariance functions in randomized order.

Task 1.3: Let $\boldsymbol{X} = \{x_i\}_{i=1}^{100}$ be a sorted set of equidistant points in the interval $[0, 2]$. Figure 1 shows the covariance matrices for function values evaluated at $\boldsymbol{X}$ as well as samples from the corresponding Gaussian process prior for each of the six covariance functions. Match the plots to each of the six covariance functions.

Solution

First, we recognize k_1 and k_2 as squared exponential (exponentiated quadratic) kernels with a (relative) long and short lengthscale, respectively. Therefore, we have that k_1 correspond to (d) and k_2 corresponds to (a) as evidence both from the structure of the covariance matrix as well as the samples functions. Next, we recognize k_3 as the sum of a constant kernel and a linear kernel, which matches panel (f). As a sanity check, we concluded earlier that $\mathbb{V}\left[y_3(x)\right] = 4 + 2x^2$,and therefore $\mathbb{V}\left[y_3(0)\right] = 4$, which is also consistent with panel (f). Observing that kernels k_4 and k_5 both contain a periodic component, we conclude that they must correspond to either (c) or (e). Noting that k_5 also contain a linear component, we can deduce that k_5 matches (c) and k_4 matches (e). Finally, this means that k_6 must correspond to (b).

The matching pairs are summarized below:

Covariance function	Plot
1	d
2	a
3	f
4	e
5	c
6	b

End of solution

Consider now the following kernel function:

$$k_7(x, x') = \kappa_0^2 + \kappa_1^2 x x' + \kappa_2^2 \exp\left(-\frac{||x - x'||_2^2}{2\ell^2}\right), \tag{14}$$

where $\kappa_1, \kappa_2 \geq 0$ and $\ell > 0$ are hyperparameters.

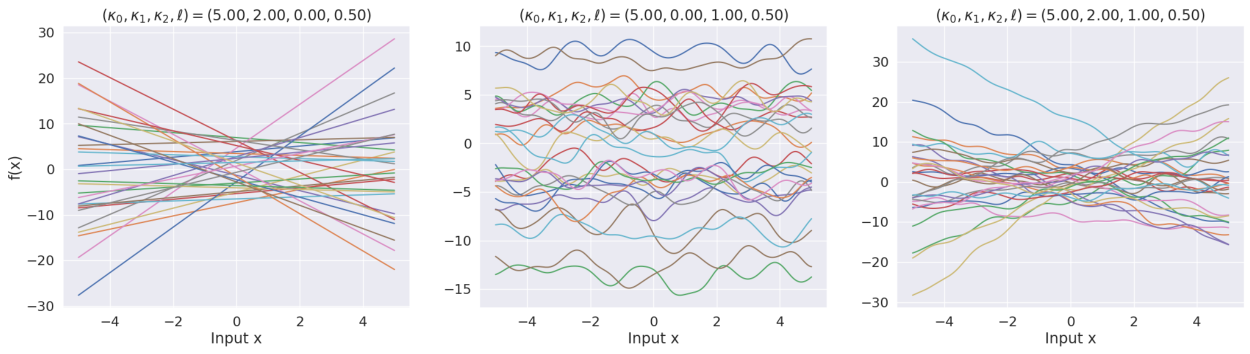
Task 1.4: Implement the kernel function k_7. Generate and plot $S = 30$ realizations (samples) of the process $f(x) \sim \mathcal{GP}(0, k_7(x, x'))$ for $x \in [-6, 6]$ for $(\kappa_0, \kappa_1, \kappa_2, \ell) = (5, 2, 0, \frac{1}{2})$. Repeat with $(\kappa_0, \kappa_1, \kappa_2, \ell) = (5, 0, 1, \frac{1}{2})$ and $(\kappa_0, \kappa_1, \kappa_2, \ell) = (5, 2, 1, \frac{1}{2})$

Hint: You can either adapt the code template from the GP exercises or you can simply implement the relevant equations directly.

Solution

See figure 2.

End of solution



```python
# implemement covariance function
def kernel7(X1, X2, kappa0, kappa1, kappa2, ell):
    # compute squared distances for SE-component
    dists_squared = jnp.sum((jnp.expand_dims(X1, 1) - jnp.expand_dims(X2, 0))**2, axis=-1)
    # sum all components
    K = kappa0**2 + kappa1**2*X1@X2.T + kappa2**2*jnp.exp(-dists_squared/(2*ell**2))
    return K

# settings
num_samples = 30
hyperparameter_values = [  (5, 2, 0., 0.5),
                            (5, 0., 1., 0.5),
                            (5, 2., 1., 0.5), ]

# plot
xp = jnp.linspace(-5, 5, 1000)[:, None]
fig, ax = plt.subplots(1, 3, figsize=(25, 6))
key = random.PRNGKey(1234)
for i, (kappa0, kappa1, kappa2, lengthscale) in enumerate(hyperparameter_values):
    key, subkey = random.split(key)
    # construct mean and covariance
    mu =  jnp.zeros(len(xp))
    K = kernel7(xp, xp, kappa0, kappa1, kappa2, lengthscale)
    # generate samples
    f_samples = generate_samples(subkey, mu, K, num_samples=num_samples, jitter=1e-8)
    # plot samples
    ax[i].plot(xp, f_samples)
    ax[i].set(xlabel='Input x', title=f'$(\\kappa_0, \\kappa_1, \\kappa_2, \\ell) = ({kappa0
    :3.2f}, {kappa1:3.2f}, {kappa2:3.2f}, {lengthscale:3.2f})$')
ax[0].set(ylabel='f(x)')

```

Figure 2: Prior samples for Task 1.4

Part 2: Laplace approximation for a simple neural network

Consider the following small regression data set with $N = 10$ observations $\mathcal{D} = \{(x_n, y_n)\}_{n=1}^N$, where

$$\boldsymbol{x} = \begin{bmatrix}9.589, 7.375, 4.647, 2.501, 2.538, 6.783, 4.294, 5.111, 0.130, & 0.783\end{bmatrix} \tag{15}$$

$$\boldsymbol{y} = \begin{bmatrix}3.032, 3.349, 2.906, 2.126, 1.538, 2.787, 3.078, 2.993, 0.828, -0.331\end{bmatrix} \tag{16}$$

such that the n'th element in $\boldsymbol{x}$ is $x_n \in \mathbb{R}$ and similar for $y_n \in \mathbb{R}$.

We will work with the following very simple two-parameter neural network:

$$f(x) = w_1\sigma(x + w_0), \tag{17}$$

where $\boldsymbol{w} = \begin{bmatrix}w_0 & w_1\end{bmatrix} \in \mathbb{R}^2$ is the parameters of the network and $\sigma : \mathbb{R} \to (0, 1)$ is the logistic sigmoid function. We can construct a non-linear probabilistic model for regression by using the network $f(x)$ as the

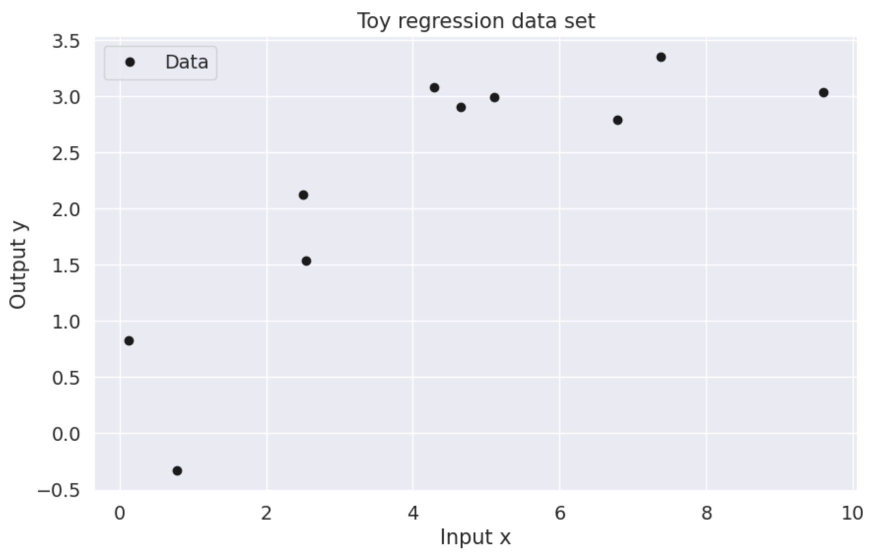


Figure 3: Data for Part 2

mean function of a Gaussian likelihood:

$$p(y_n|x_n, \boldsymbol{w}) = \mathcal{N}(y_n|f(x), \beta^{-1}) = \mathcal{N}(y_n|w_1\sigma(x + w_0), \beta^{-1}), \tag{18}$$

where $\beta^{-1} > 0$ is the noise variance. We impose i.i.d. Gaussian prior distributions on both parameters:

$$w_0 \sim \mathcal{N}(0, \tau^2) \tag{19}$$
$$w_1 \sim \mathcal{N}(0, \tau^2), \tag{20}$$

where $\tau^2 > 0$ is variance of the prior.

Assume $\tau = 2$ and $\beta = 4$.

Task 2.1: Generate $S = 100$ samples from the prior $p(\boldsymbol{w})$ and plot the corresponding functions $f(x)$ for $x \in [0, 10]$ on top of a scatter plot of the data.

Hint: You can find inspiration in the course notebooks in order to make the plots look nice.

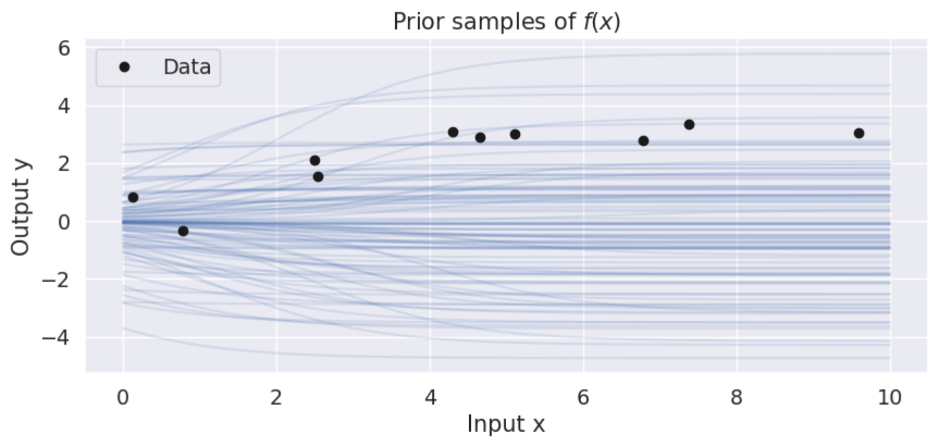
Solution

See Figure 4

End of solution

The joint density for $(\boldsymbol{y}, \boldsymbol{w})$ according to the probabilistic model in eq. (18)-(20) is given

$$p(\boldsymbol{y}, \boldsymbol{w}) = p(\boldsymbol{w}) \prod_{n=1}^{N} p(y_n|x_n, \boldsymbol{w}) = \mathcal{N}(w_0|0, \tau^2)\mathcal{N}(w_1|0, \tau^2) \prod_{n=1}^{N} \mathcal{N}(y_n|w_1\sigma(x_n + w_0), \beta^{-1}) \tag{21}$$



```python
# hyperparameters
beta = 4
tau = 2.

# implement sigmoid and neural network f(x)
sigmoid = lambda x: 1./(1 + jnp.exp(-x))
f = lambda x, w: w[1]*sigmoid(x + w[0])

# number of samples, random seed etc.
S = 100
key = random.PRNGKey(1)
xpred = jnp.linspace(0, 10, 1000)[:, None]

# generate prior samples of the model
w_prior = random.multivariate_normal(key, jnp.zeros(2), tau**2*jnp.identity(2), shape=(S))

# make plot
fig, ax = plt.subplots(1, 1, figsize=(10, 4))
for wi in w_prior:
    ax.plot(xpred, f(xpred, wi), 'b-', alpha=0.15)
ax.plot(xtrain, ytrain, 'ko', label='Data')
ax.set(xlabel='Input x', ylabel='Output y', title='Prior samples of $f(x)$')
ax.legend()

```

Figure 4: Solution for task 2.1

Task 2.2: What prevents us from using the equations in section 3.3 in Murphy1 to compute the exact posterior distribution of the parameters given the data analytically for the system in eq. (21) **?**

Solution

Even though both prior and likelihood are Gaussians, we cannot use the equations from section 3.3 in Murphy1 because of the *non-linear* dependency on w_0 in the mean of the likelihood.

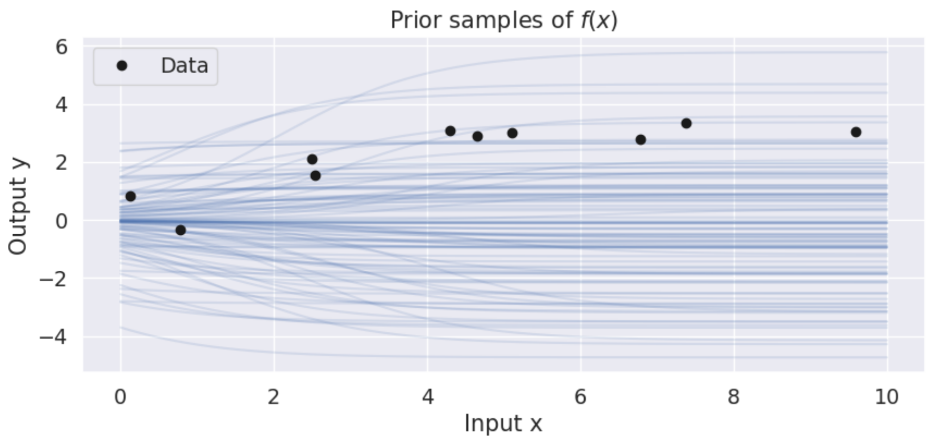
End of solution

Task 2.3: Implement a python for function for evaluating the *logarithm of the joint density* in eq. (21)**. Report the numerical value of $\log p(\boldsymbol{y}, \boldsymbol{w})$ for the dataset given above when $w_0 = w_1 = 0$.**

Solution

Evaluating the logarithm of the joint density at the zero vector yields $\log p(y, \mathbf{0}) = -130.949$. See Figure 5.

End of solution



```python
def log_lik(w):
    ftrain = f(xtrain, w)
    log_lik_y = log_npdf(ytrain, ftrain, 1./beta)
    return log_lik_y.sum(0)[0]

def log_prior(w):
    return log_npdf(w[0], 0, tau**2) + log_npdf(w[1], 0, tau**2)

def log_joint(w):
    return log_prior(w) + log_lik(w)

print(f'Evaluating the log joint at w = [0, 0]: {log_joint(jnp.zeros(2)):4.3f}')

```

Figure 5: Solution for task 2.3

Task 2.4: Determine the expression for the gradient and Hessian of the logarithm of the joint density wrt. $\boldsymbol{w}$.

Hints: 1) The chain rule for differentiation will be handy. 2) Recall the identities $\frac{d}{dx}\sigma(x) = \sigma'(x) = \sigma(x)(1-\sigma(x))$ and $\frac{d^2}{dx^2}\sigma(x) = \sigma''(x) = (1-2\sigma(x))\sigma'(x)$. 3) Start by determining the gradient and then validate your result via the next two tasks and then come and work on the Hessian.

Solution

First, we write out the expression for the log joint density and get rid of terms that are constant wrt. w_0 and/or w_1:

$$\log p(\boldsymbol{y}, \boldsymbol{w}) = \log \mathcal{N}(w_0|0,\tau^2) + \log \mathcal{N}(w_1|0,\tau^2) + \sum_{n=1}^{N} \log \mathcal{N}(y_n|w_1\sigma(x_n+w_0), \beta^{-1}) \tag{22}$$

$$= -\frac{1}{2}\log(2\pi\tau^2) - \frac{1}{2\tau^2}w_0^2 - \frac{1}{2}\log(2\pi\tau^2) - \frac{1}{2\tau^2}w_1^2 + \sum_{n=1}^{N}\left[-\frac{1}{2}\log(2\pi\sigma^2) - \frac{\beta}{2}(y_n - w_1\sigma(x_n+w_0))^2\right] \tag{23}$$

$$= -\frac{1}{2\tau^2}w_0^2 - \frac{1}{2\tau^2}w_1^2 - \frac{\beta}{2}\sum_{n=1}^{N}(y_n - w_1\sigma(x_n+w_0))^2 + \text{constant} \tag{24}$$

Next, we compute the derivative wrt. w_0

$$\frac{\partial}{\partial w_0}\log p(\boldsymbol{y}, \boldsymbol{w}) = -\frac{1}{2\tau^2}\frac{\partial}{\partial w_0}w_0^2 - \frac{1}{2\tau^2}\frac{\partial}{\partial w_0}w_1^2 - \frac{\beta}{2}\sum_{n=1}^{N}\frac{\partial}{\partial w_0}(y_n - w_1\sigma(x_n+w_0))^2 + \frac{\partial}{\partial w_0}\text{constant} \tag{25}$$

$$= -\frac{1}{\tau^2}w_0 - \frac{\beta}{2}\sum_{n=1}^{N}\frac{\partial}{\partial w_0}(y_n - w_1\sigma(x_n+w_0))^2 \tag{26}$$

$$= -\frac{1}{\tau^2}w_0 - \frac{\beta}{2}\sum_{n=1}^{N}2(y_n - w_1\sigma(x_n+w_0))\left[\frac{\partial}{\partial w_0}y_n - w_1\frac{\partial}{\partial w_0}\sigma(x_n+w_0)\right] \tag{27}$$

$$= -\frac{1}{\tau^2}w_0 + \beta\sum_{n=1}^{N}(y_n - w_1\sigma(x_n+w_0))w_1\frac{\partial}{\partial w_0}\sigma(x_n+w_0) \tag{28}$$

$$= -\frac{1}{\tau^2}w_0 + \beta\sum_{n=1}^{N}(y_n - w_1\sigma(x_n+w_0))w_1\sigma(x_n+w_0)(1-\sigma(x_n+w_0)) \tag{29}$$

$$= -\frac{1}{\tau^2}w_0 + \beta\sum_{n=1}^{N}(y_n - f(x_n))w_1\sigma'(x_n+w_0) \tag{30}$$

And the derivative wrt. w_1

$$\frac{\partial}{\partial w_1}\log p(\boldsymbol{y},\boldsymbol{w}) = -\frac{1}{2\tau^2}\frac{\partial}{\partial w_1}w_0^2 - \frac{1}{2\tau^2}\frac{\partial}{\partial w_1}w_1^2 - \frac{\beta}{2}\sum_{n=1}^{N}\frac{\partial}{\partial w_1}(y_n - w_1\sigma(x_n+w_0))^2 + \frac{\partial}{\partial w_1}\text{constant} \tag{31}$$

$$= -\frac{1}{\tau^2}w_1 - \frac{\beta}{2}\sum_{n=1}^{N}2(y_n - w_1\sigma(x_n+w_0))\frac{\partial}{\partial w_1}\left[y_n - w_1\sigma(x_n+w_0)\right] \tag{32}$$

$$= -\frac{1}{\tau^2}w_1 + \beta\sum_{n=1}^{N}(y_n - w_1\sigma(x_n+w_0))\sigma(x_n+w_0) \tag{33}$$

$$= -\frac{1}{\tau^2}w_1 + \beta\sum_{n=1}^{N}(y_n - f(x_n))\sigma(x_n+w_0) \tag{34}$$

Hence, the gradient wrt. $\boldsymbol{w}$ becomes

$$\nabla_{\boldsymbol{w}}\log p(\boldsymbol{y},\boldsymbol{w}) = -\frac{1}{\tau^2}\boldsymbol{w} + \beta\sum_{n=1}^{N}(y_n - f(x_n))\begin{bmatrix} w_1\sigma'(x_n+w_0) \\ \sigma(x_n+w_0)\end{bmatrix} \tag{35}$$

We can now compute the Hessian matrix $\mathcal{H} \in \mathbb{R}^{2\times 2}$ element by element. First,

$$\mathcal{H}_{11} = \frac{\partial^2}{\partial w_0^2}\log p(\boldsymbol{y},\boldsymbol{w}) \tag{36}$$

$$= -\frac{\partial}{\partial w_0}\frac{1}{\tau^2}w_0 + \beta\sum_{n=1}^{N}\frac{\partial}{\partial w_0}(y_n - f(x_n))f(x_n)(1-\sigma(x_n+w_0)) \tag{37}$$

$$= -\frac{1}{\tau^2} + \beta\sum_{n=1}^{N}\left[\frac{\partial}{\partial w_0}(y_n f(x_n)(1-\sigma(x_n+w_0)) - f(x_n)f(x_n)(1-\sigma(x_n+w_0)))\right] \tag{38}$$

$$= -\frac{1}{\tau^2} + \beta\sum_{n=1}^{N}\left[\frac{\partial}{\partial w_0}(y_n w_1\sigma(x_n+w_0)(1-\sigma(x_n+w_0)) - w_1^2\sigma(x_n+w_0)^2(1-\sigma(x_n+w_0)))\right] \tag{39}$$

$$= -\frac{1}{\tau^2} + \beta\sum_{n=1}^{N}\left[\frac{\partial}{\partial w_0}(y_n w_1\sigma'(x_n+w_0) - w_1^2\sigma(x_n+w_0)\sigma'(x_n+w_0))\right] \tag{40}$$

$$= -\frac{1}{\tau^2} + \beta\sum_{n=1}^{N}\left[(y_n w_1\sigma''(x_n+w_0) - w_1^2\left[\sigma'(x_n+w_0)\sigma'(x_n+w_0) + \sigma(x_n+w_0)\sigma''(x_n+w_0)\right])\right] \tag{41}$$

$$= -\frac{1}{\tau^2} + \beta\sum_{n=1}^{N}\left[(y_n w_1 - w_1^2\sigma(x_n+w_0))\sigma''(x_n+w_0) - (w_1\sigma'(x_n+w_0))^2\right]. \tag{42}$$

And

$$\begin{aligned}
\mathcal{H}_{22} &= \frac{\partial^2}{\partial w_1^2} \log p(\boldsymbol{y}, \boldsymbol{w}) && (43)\\
&= -\frac{1}{\tau^2}\frac{\partial}{\partial w_1} w_1 + \beta \sum_{n=1}^{N} \frac{\partial}{\partial w_1}(y_n - f(x_n))\sigma(x_n + w_0) && (44)\\
&= -\frac{1}{\tau^2} + \beta \sum_{n=1}^{N} \frac{\partial}{\partial w_1}(y_n \sigma(x_n + w_0) - f(x_n)\sigma(x_n + w_0)) && (45)\\
&= -\frac{1}{\tau^2} - \beta \sum_{n=1}^{N} \frac{\partial}{\partial w_1} f(x_n)\sigma(x_n + w_0) && (46)\\
&= -\frac{1}{\tau^2} - \beta \sum_{n=1}^{N} \frac{\partial}{\partial w_1} w_1 \sigma(x_n + w_0)\sigma(x_n + w_0) && (47)\\
&= -\frac{1}{\tau^2} - \beta \sum_{n=1}^{N} \sigma(x_n + w_0)^2. && (48)
\end{aligned}$$

Finally, the cross term

$$\begin{aligned}
\mathcal{H}_{12} &= \frac{\partial}{\partial w_0}\frac{\partial}{\partial w_1} \log p(\boldsymbol{y}, \boldsymbol{w}) && (49)\\
&= -\frac{1}{\tau^2}\frac{\partial}{\partial w_0} w_1 + \beta \sum_{n=1}^{N} \frac{\partial}{\partial w_0}(y_n - f(x_n))\sigma(x_n + w_0) && (50)\\
&= \beta \sum_{n=1}^{N} \frac{\partial}{\partial w_0}(y_n \sigma(x_n + w_0) - f(x_n)\sigma(x_n + w_0)) && (51)\\
&= \beta \sum_{n=1}^{N} \left[y_n \frac{\partial}{\partial w_0}\sigma(x_n + w_0) - \frac{\partial}{\partial w_0} f(x_n)\sigma(x_n + w_0) \right] && (52)\\
&= \beta \sum_{n=1}^{N} \left[y_n \sigma'(x_n + w_0) - \frac{\partial}{\partial w_0} w_1 \sigma(x_n + w_0)\sigma(x_n + w_0) \right] && (53)\\
&= \beta \sum_{n=1}^{N} \left[y_n \sigma'(x_n + w_0) - \frac{\partial}{\partial w_0} w_1 \sigma(x_n + w_0)^2 \right] && (54)\\
&= \beta \sum_{n=1}^{N} \left[y_n \sigma'(x_n + w_0) - 2 w_1 \sigma(x_n + w_0) \frac{\partial}{\partial w_0}\sigma(x_n + w_0) \right] && (55)\\
&= \beta \sum_{n=1}^{N} \left[y_n \sigma'(x_n + w_0) - 2 w_1 \sigma(x_n + w_0)\sigma'(x_n + w_0) \right] && (56)\\
&= \beta \sum_{n=1}^{N} (y_n - 2 w_1 \sigma(x_n + w_0))\sigma'(x_n + w_0) && (57)
\end{aligned}$$

End of solution

Task 2.5: Determine the Laplace approximation $\boldsymbol{q}(\boldsymbol{w})$ of the posterior distribution such that $p(\boldsymbol{w}|\boldsymbol{y}) \approx \boldsymbol{q}(\boldsymbol{w})$.

Solution

The Laplace approximation for $p(\boldsymbol{w}|\boldsymbol{y})$ is given by $q(\boldsymbol{w}) = \mathcal{N}(\boldsymbol{w}|\boldsymbol{m}, \boldsymbol{\Sigma})$, where $\boldsymbol{m} = \boldsymbol{w}_{\hat{\text{MAP}}}$ and $\boldsymbol{\Sigma} = -\mathcal{H}^{-1}$ and $\mathcal{H}$ is the negative Hessian of the log joint evaluated at $\boldsymbol{w}_{\hat{\text{MAP}}}$. Since we have the gradient, we can obtain $\boldsymbol{w}_{\text{MAP}}$ is a simple first-order optimization scheme:

$$\boldsymbol{w}^{t+1} = \boldsymbol{w}^t + \alpha g(\boldsymbol{w}^t), \tag{58}$$

where $g(\boldsymbol{w}) = \nabla_{\boldsymbol{w}} \log p(\boldsymbol{y}, \boldsymbol{w})$ and t is the iteration index. Using $\alpha = 2 \cdot 10^{-2}$ and running 200 iterations starting from origo yields

$$\hat{\boldsymbol{w}}_{\text{MAP}} = \begin{bmatrix} -2.204 \\ 3.104 \end{bmatrix} \tag{59}$$

where the squared norm of the gradient is approximately 10^{-17}. Evaluating the Hessian at $\hat{\boldsymbol{w}}_{\text{MAP}}$ yields:

$$\mathcal{H} = \begin{bmatrix} -7.854 & -6.897 \\ -6.897 & -25.143 \end{bmatrix} \tag{60}$$

such that the resulting covariance matrix becomes

$$\boldsymbol{\Sigma} = -\mathcal{H}^{-1} \approx \begin{bmatrix} 0.168 & -0.046 \\ -0.046 & 0.052 \end{bmatrix}. \tag{61}$$

See Figure 6 for the code.

End of solution

```python
# first and second order gradient of logistic sigmoid
grad_sigmoid = lambda x: sigmoid(x)*(1-sigmoid(x))
grad2_sigmoid = lambda x: (1-2*sigmoid(x))*sigmoid(x)*(1-sigmoid(x))

def gradient(w):
    """ implements gradient of log joint """
    # precompute
    fx = model(xtrain, w)
    sigmoid_x = sigmoid(xtrain+w[0])
    grad_sigmoid_x = grad_sigmoid(xtrain+w[0])

    # coordinate wise gradient
    g0 = -w[0]/tau**2 + beta*jnp.sum((ytrain - fx)*w[1]*grad_sigmoid_x)
    g1 = -w[1]/tau**2 + beta*jnp.sum((ytrain - fx)*sigmoid_x)
    return jnp.array([g0, g1])


def hessian(w):
    """ implements Hessian of log joint """
    # precompute
    fx = model(xtrain, w)
    sigmoid_x = sigmoid(xtrain+w[0])
    g1 = grad_sigmoid(xtrain+w[0])
    g2 = grad2_sigmoid(xtrain+w[0])

    # compute each entry of Hessian matrix
    H11 = -1./tau**2 + beta*jnp.sum((ytrain*w[1] - w[1]**2*sigmoid_x)*grad2_sigmoid(xtrain+w
    [0]) - (w[1]*g1)**2)
    H22 = -1./tau**2 - beta*jnp.sum(sigmoid_x**2)
    H12 = beta*jnp.sum((ytrain-2*w[1]*sigmoid_x)*g1)

    return jnp.array([[H11, H12], [H12, H22]])

# optimization loop
w_MAP = jnp.array([0., 0.])
step_size = 2e-2
for itt in range(200):

    # compute gradient
    g = gradient(w_MAP)

    # update
    w_MAP = w_MAP + step_size*g

    # report progress
    if (itt+1) % 50 == 0:
        print(f'Itt {itt:4d}: log joint = {log_joint(w_MAP):4.3f}, ||g||^2 = {jnp.sum(g**2)
    :3.2e}')

print(f'\nCoordinates for w_MAP, i.e. mean for Laplace')
print(np.array2string(w_MAP, precision=3))

print(f'\n Hessian at w_MAP')
H = hessian(w_MAP)
print(np.array2string(H, precision=3))

Sigma = -jnp.linalg.inv(H)
print(f'\n Covariance matrix for Laplace approximation')
print(np.array2string(Sigma, precision=3))

```

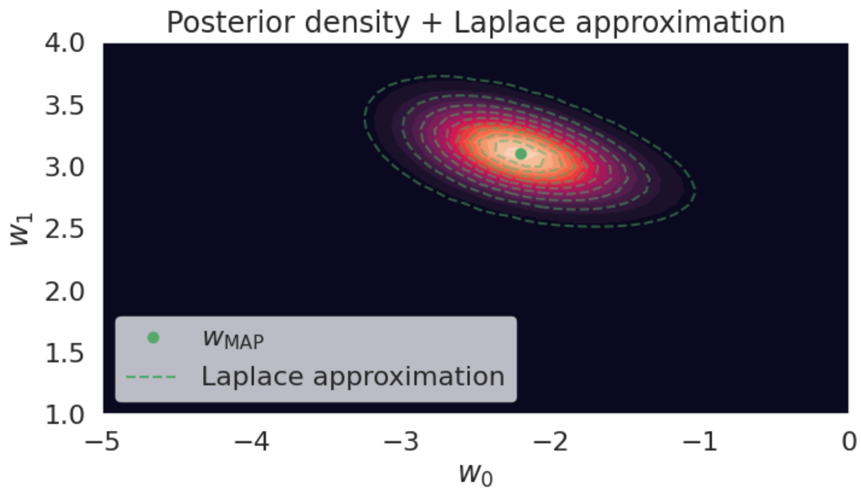
Figure 6: Solution for task 2.5

Task 2.6: Create a 2D contour plot of posterior distribution of the parameters. Plot the contours of the Laplace approximation on top of the plot to validate your results from the previous tasks.

Hints: Find inspiration in the exercise notebook for week 3 & 4 if you are struggling with the plotting code.

Solution

See Figure 7.



```python
from scipy.stats import multivariate_normal

# define grid
w0_grid = jnp.linspace(-5, 5, 100)
w1_grid = jnp.linspace(-5, 5, 101)

# evaluate joint
Z = jnp.zeros((len(w0_grid), len(w1_grid)))
for i in range(len(w0_grid)):
    for j in range(len(w1_grid)):
        wi = jnp.array([w0_grid[i], w1_grid[j]])
        Z = Z.at[i,j].set(log_joint(wi))

# evalaute approximation
q_logpdf = jnp.zeros((len(w0_grid), len(w1_grid)))
for i in range(len(w0_grid)):
    for j in range(len(w1_grid)):
        wi = jnp.array([w0_grid[i], w1_grid[j]])
        q_logpdf = Z.at[i,j].set(multivariate_normal.logpdf(wi, w_MAP, Sigma))

# plot
fig, ax = plt.subplots(1, 1, figsize=(8, 4))
ax.contourf(w0_grid, w1_grid, jnp.exp(Z.T), 20)
ax.plot(w_MAP[0], w_MAP[1], 'go', label='$w_{\\text{MAP}}$')
ax.set(xlabel='$w_0$', ylabel='$w_1$', title='Posterior density + Laplace approximation',
    xlim=(-5, 0), ylim=(1, 4))
ax.contour(w0_grid, w1_grid, jnp.exp(q_logpdf.T), jnp.linspace(1e-6, jnp.exp(log_joint(w_MAP
    )), 10), colors='g', linestyles='--', alpha=0.5)
ax.plot(0, 0, 'g--', label='Laplace approximation')
ax.legend(loc='lower left')

```

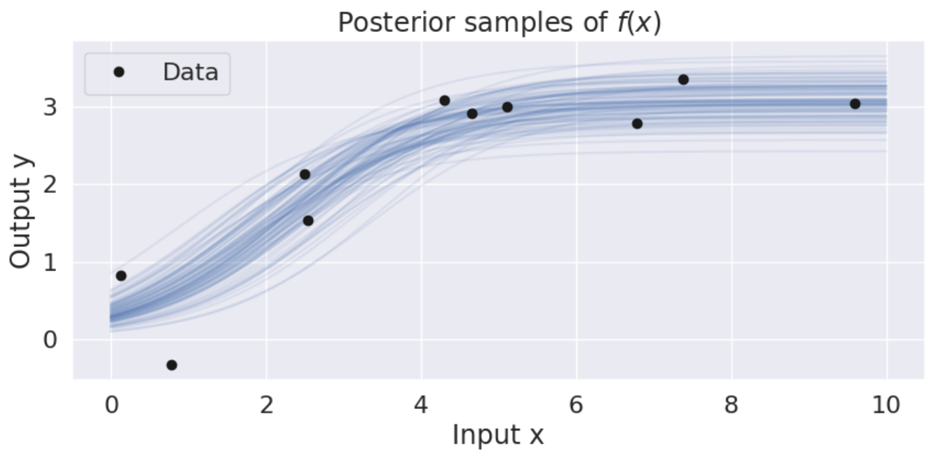
Figure 7: Contour plot of the posterior density and Laplace approximation

End of solution

Task 2.7: Generate and plot $S = 100$ samples from the approximate posterior distribution. Pot the corresponding function $f(x)$ for $x \in [0, 10]$ on top of a scatter plot of the data.

Solution

See Figure 8.



```python
# generate samples
S = 100
w_posterior = random.multivariate_normal(key, w_MAP, Sigma, shape=(S))

# make plot
fig, ax = plt.subplots(1, 1, figsize=(10, 4))
for wi in w_posterior:
    ax.plot(xpred, model(xpred, wi), 'b-', alpha=0.1)

ax.plot(xtrain, ytrain, 'ko', label='Data')
ax.set(xlabel='Input x', ylabel='Output y', title='Posterior samples of $f(x)$')

```

Figure 8: Posterior samples of $f(x)$

End of solution

Task 2.8: Use the Laplace approximation to compute the (approximate) posterior probability for the event $f(8) > 3$.

Solution

We can estimate the requested probability using a Monte Carlo estimator based on samples from the Laplace approximation as follows.

$$\begin{aligned}
p(f(8) > 3|\mathbf{y}) &= \int \mathbb{I}\left[f(8; \boldsymbol{w}) > 3\right] p(\boldsymbol{w}|\boldsymbol{y})\mathrm{d}\boldsymbol{w} \\
&\approx \int \mathbb{I}\left[f(8; \boldsymbol{w}) > 3\right] q(\boldsymbol{w})\mathrm{d}\boldsymbol{w} && \text{(Laplace approx.)} \\
&\approx \frac{1}{S}\sum_{i=1}^{S} \mathbb{I}\left[f(8; \boldsymbol{w}^{(i)}) < 3\right] && \text{(Monte Carlo approx.)} \\
&\approx 0.660\ (\pm 0.015)
\end{aligned}$$

where $\boldsymbol{w}^{(i)} \sim q(\mathbf{w})$.

```python
# generate samples
S = 1000
w_posterior = random.multivariate_normal(key, w_MAP, Sigma, shape=(S))

# compute f(8) for each posterior sample
fstar = jnp.array([model(8, wi) for wi in w_posterior])

# estimate probablity and MCSE
event = fstar > 3
MCSE = jnp.sqrt(jnp.var(event)/S)
print(f'p(f(8)) > 3 = {jnp.mean(event):4.3f} (MCSE={MCSE:4.3f})')

```

Figure 9: Solution task 2.8

End of solution