02477 Bayesian Machine Learning 2025: Assignment 3

This is the last 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 1,2 if possible) and hand in via DTU Learn). The assignment is **mandatory**. The deadline is **4th of May 23:59**.

Part 1: Fully Bayesian inference for Gaussian process regression

In this part, we will extend our Gaussian process regression analysis of the bike sharing dataset (from week 5) to fully Bayesian inference on hyperparameter level. Use the code below (Fig. **??**) to load and preprocess the data .

```
# load data from disk
data = jnp.load('./data_exercise5b.npz')
X = data['day']
y = jnp.log(data['bike_count'])

# remove mean and scale to unit variance
ym, ys = jnp.mean(y), jnp.std(y)
y = (y-ym)/ys
```

Figure 1: Code for loading and preprocessing the bike sharing dataset.

The Gaussian process regression model for the dataset $\mathcal{D} = \{(x_n, y_n)\}_{n=1}^N$ is given below:

$$y_n = f(x_n) + \epsilon_n, \tag{1}$$

where $\epsilon_n \sim \mathcal{N}(0, \sigma^2)$ and $f(x) \sim \mathcal{GP}(0, k(x, x'))$ for $k(x, x') = \kappa^2 \exp\left(-\frac{\|x-x'\|_2^2}{2\ell^2}\right)$.

We will impose the following prior distributions on the hyperparameters:

$$\begin{aligned}\kappa &\sim \mathcal{N}_+(0, 1)\\ \ell &\sim \mathcal{N}_+(0, v)\\ \sigma &\sim \mathcal{N}_+(0, 1),\end{aligned}$$

where $v > 0$ is a positive constant (which you will determine in the next task) and $\mathcal{N}_+(m, v)$ is the half-normal distribution. These assumptions lead to the following joint distribution

$$\begin{aligned}p(\boldsymbol{y}, \boldsymbol{f}, \sigma, \kappa, \ell) &= p(\boldsymbol{y}|\boldsymbol{f}, \sigma^2)p(\boldsymbol{f}|\kappa, \ell)p(\kappa)p(\ell)p(\sigma)\\ &= \mathcal{N}(\boldsymbol{y}|\boldsymbol{f}, \sigma^2\boldsymbol{I})\mathcal{N}(\boldsymbol{f}|\boldsymbol{0}, \boldsymbol{K})\mathcal{N}_+(\kappa|0,1)\mathcal{N}_+(\ell|0,v)\mathcal{N}_+(\sigma|0,1).\end{aligned}$$

In this exercise, we want to impose a prior that prevent the lengthscale from becoming too large.

Task 1.1: Choose a value for v such that the prior probability of observing a lengthscale larger than 100 is approximately 1%, i.e. $p(\ell > 100) \approx 0.01$.

Hints: You can do this in several ways, e.g. numerically or analytically

Solution

For Gaussian, we know that approximately 68% of the probability mass fall within one standard deviation from the mean, approximately 95% within two standard deviations and approximately 99.7% within three deviations. Hence, we expect solution for v to be close to 100/3. In fact,

$$\ell \sim \mathcal{N}_+(0, 100^2/3) \Rightarrow P(\ell > 100) \approx 0.003 \tag{2}$$

If we want more precision:

$$p(\ell > 100) = \int_0^x \mathcal{N}_+(\ell|0, v)\mathrm{d}\ell = 0.01 \tag{3}$$

which can be solved via the error function or via the by inverting the CDF of the standardized Gaussian Φ

$$x = \Phi^{-1}(0.995) \approx 2.58,$$

where we use 0.995 to account for the fact that the regular Gaussian is two-sided, whereas the half-Gaussian is one-sided. Hence,

$$v = \left(\frac{100}{2.58}\right)^2 \approx 1507.182 \tag{4}$$

End of solution

Task 1.2: Determine the marginalized distribution $p(\boldsymbol{y}, \sigma, \kappa, \ell)$.

Solution

We have

$$\begin{aligned}
p(\boldsymbol{y}, \sigma, \kappa, \ell) &= \int p(\boldsymbol{y}, \boldsymbol{f}, \sigma, \kappa, \ell)\mathrm{d}\boldsymbol{f} \\
&= \int \mathcal{N}(\boldsymbol{y}|\boldsymbol{f}, \sigma^2\boldsymbol{I})\mathcal{N}(\boldsymbol{f}|\boldsymbol{0}, \boldsymbol{K})\mathcal{N}_+(\kappa|0,1)\mathcal{N}_+(\ell|0,v)\mathcal{N}_+(\sigma|0,1)\mathrm{d}\boldsymbol{f} \\
&= \int \mathcal{N}(\boldsymbol{y}|\boldsymbol{f}, \sigma^2\boldsymbol{I})\mathcal{N}(\boldsymbol{f}|\boldsymbol{0}, \boldsymbol{K})\mathrm{d}\boldsymbol{f}\, \mathcal{N}_+(\kappa|0,1)\mathcal{N}_+(\ell|0,v)\mathcal{N}_+(\sigma|0,1) && \text{(Using linearity)}
\end{aligned}$$

We now recognize the integral $\int \mathcal{N}(\boldsymbol{y}|\boldsymbol{f}, \sigma^2\boldsymbol{I})\mathcal{N}(\boldsymbol{f}|\boldsymbol{0}, \boldsymbol{K})\mathrm{d}\boldsymbol{f}$ as the definition of the model evidence for a GP, and therefore, we know solution is $p(\mathbf{y}) = \mathcal{N}(\boldsymbol{y}|\boldsymbol{f}, \mathbf{K} + \sigma^2\boldsymbol{I})$, e.g. by eq. (18.74) in Murphy2. Hence,

$$p(\boldsymbol{y}, \sigma, \kappa, \ell) = \mathcal{N}(\boldsymbol{y}|\boldsymbol{0}, \boldsymbol{K} + \sigma^2\boldsymbol{I})\mathcal{N}_+(\kappa|0,1)\mathcal{N}_+(\ell|0,v)\mathcal{N}_+(\sigma|0,1)$$

End of solution

The next goal is to approximate the posterior distribution over the hyperparameters, i.e. $p(\kappa, \ell, \sigma|\boldsymbol{y})$, using a Metropolis-sampler. Define $\theta = \{\kappa, \ell, \sigma^2\}$ to be the set of hyperparameters of the model. Since the scale of the hyperparameters are quite different, we will use an anisotropic proposal distribution:

$$q(\boldsymbol{\theta}^*|\boldsymbol{\theta}^{k-1}) = \mathcal{N}(\boldsymbol{\theta}^*|\boldsymbol{\theta}^{k-1}, \boldsymbol{\Sigma}) \quad \text{for} \quad \boldsymbol{\Sigma} = \frac{1}{2}\begin{bmatrix} 1 & 0 & 0 \\ 0 & 100 & 0 \\ 0 & 0 & 0.01 \end{bmatrix}. \tag{5}$$

That is, the proposed step-size will generally be larger for the lengthscale dimension and so on.

Task 1.3: Implement a Metropolis sampler using the proposal distribution in eq. (5) for generating samples from the posterior $p(\kappa, \ell, \sigma|\boldsymbol{y})$. Run 4 chains for 10000 iterations each.

Hint: The half-normal distribution can be implemented as follows

```
from scipy.stats import norm
def log_halfnormal(x):
    return jnp.log(2) + norm.logpdf(x, 0, 1)
```

Hint: The parameters $\kappa, \ell, \sigma > 0$ are strictly positive parameters, and therefore, we need to be careful that our MCMC chain will jump to invalid parameter configuration. One way to achieve this is to implement the log joint function to return `-jnp.inf` *whenever one or more hyperparameters are less than or equal to zero.*

Solution

The first step is the implement a function for evaluating the logarithm of the joint distribution form the previous task:

```
# prepare gp object
gp = GaussianProcessRegression(X_train, y_train, StationaryIsotropicKernel(squared_exponential))

# specify prior params
param_names = ['kappa', 'ell', 'sigma']
prior_std_devs = jnp.array([1, 100/2.58, 1])

# implement log target
def log_target(theta):
    if theta[0] < 0 or theta[1] <0  or theta[2] <0:
        return -jnp.inf

    # prior contribution
    log_prior_kappa = jnp.log(2) + norm.logpdf(theta[0], 0, prior_std_devs[0])
    log_prior_ell = jnp.log(2) + norm.logpdf(theta[1], 0, prior_std_devs[1])
    log_prior_sigma = jnp.log(2) + norm.logpdf(theta[2], 0, prior_std_devs[2])

    # likelihood contribution
    log_lik = gp.log_marginal_likelihood(*theta)

    # sum and return
    return log_lik + log_prior_kappa + log_prior_ell + log_prior_sigma
```

and then we can set up a Metropolis sampler with the specified proposal distribution as follows

```
    # generate initial values from the prior (shape: num_chains x params)
    key = random.PRNGKey(1)
    theta_init = jnp.abs(prior_std_devs*random.normal(key, shape=(4, 3)))

    # run sampler without excluding warmup
    theta_samples, accept_rates = metropolis_multiple_chains(log_target, 3, num_chains, tau=jnp.sqrt(1/
```

End of solution

Task 1.4: Plot the trace for each parameter and report the convergence diagnostics $\hat{R}$ and S_{eff} for each parameter. Discard warm-up samples and report the number of samples discarded.

Solution

We plot the traces for each chain for each parameter

```
    fig, ax = plt.subplots(1, 3, figsize=(20, 5))
    for i in range(3):
        ax[i].plot(theta_samples[:, :, i].T)
        ax[i].set(xlabel='Iterations', ylabel=param_names[i], title=f'Trace for {param_names[i]}')
```

Figure 4 shows the trace for the 4 chains of MCMC. By visual inspection, all chains appears to have (roughly)

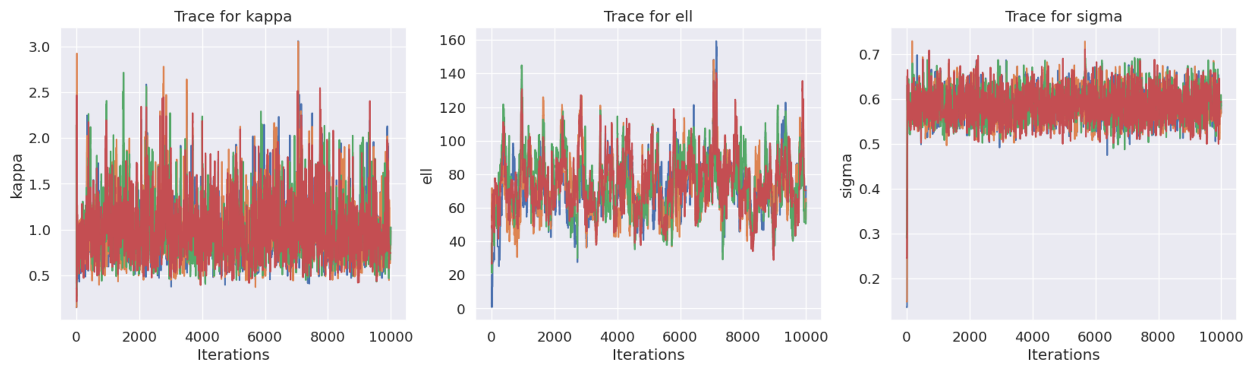

Figure 2: Trace of MCMC plot

mixed after approximately 500 iterations, but to be conservative, we discard the first 2000 samples as warm up samples, but less would probably suffice.

End of solution

Task 1.5: Estimate and report the posterior mean for each hyperparameter. Report the MCSE for each estimate.

Solution

First, we discard the warm-up samples and check the convergence diagnostics

```
# discard warmup
theta_samples = theta_samples[:, 2000:, :]

# compute diagnostics
Rhat = compute_Rhat(theta_samples)
S_eff = compute_effective_sample_size(theta_samples)

# print
for i in range(3):
    print(f'{param_names[i]:10s} Rhat={Rhat[i]:4.3f}, S_eff={S_eff[i]:4.3f}')
```

which yields

$$\hat{R}(\kappa) = 1.01 \qquad S_{\text{eff}}(\kappa) = 1075$$
$$\hat{R}(\ell) = 1.04 \qquad S_{\text{eff}}(\ell) = 259$$
$$\hat{R}(\sigma) = 1.00 \qquad S_{\text{eff}}(\sigma) = 2897$$

where we see that the $\hat{R}$-statistics is below 1.1 for all parameters. Then we merge the chains and evaluate the posterior mean and the MCSEs

```
merged_chains = theta_samples.reshape((-1, 3))
chain_mean = jnp.mean(merged_chains, axis=0)
chain_std = jnp.std(merged_chains, axis=0)

MCSE = 1/jnp.sqrt(S_eff) * chain_std

for i in range(3):
    print(f'{param_names[i]:10s}Posterior mean: {chain_mean[i]:4.2f} (MCSE = {MCSE[i]:4.3f})')
```

which yields

$$\mathbb{E}\left[\kappa|\boldsymbol{y}\right] \approx 1.12 \qquad \text{MCSE}(\kappa) \approx 0.02$$
$$\mathbb{E}\left[\ell|\boldsymbol{y}\right] \approx 69.80 \qquad \text{MCSE}(\ell) \approx 1.13$$
$$\mathbb{E}\left[\sigma|\boldsymbol{y}\right] \approx 0.56 \qquad \text{MCSE}(\sigma) \approx 0.00$$

End of solution

Task 1.6: Estimate a 95% posterior credibility interval for each hyperparameter.

Solution

Using the posterior samples, we compute the requested interval using the `numpy.percentile`-function as follows

```
lower, upper = jnp.percentile(merged_chains, jnp.array([2.5, 97.5]), axis=0)
for i in range(3):
    print(f'{param_names[i]:10s}95% Interval: [{lower[i]:3.2f}, {upper[i]:3.2f}]')
```

which yields

$$p(0.62 < \kappa < 1.93|\boldsymbol{y}) \approx 0.95$$
$$p(44.63 < \ell < 99.63|\boldsymbol{y}) \approx 0.95$$
$$p(0.51 < \sigma < 0.62|\boldsymbol{y}) \approx 0.95$$

End of solution

Part 2: Regression modelling using mixture of experts

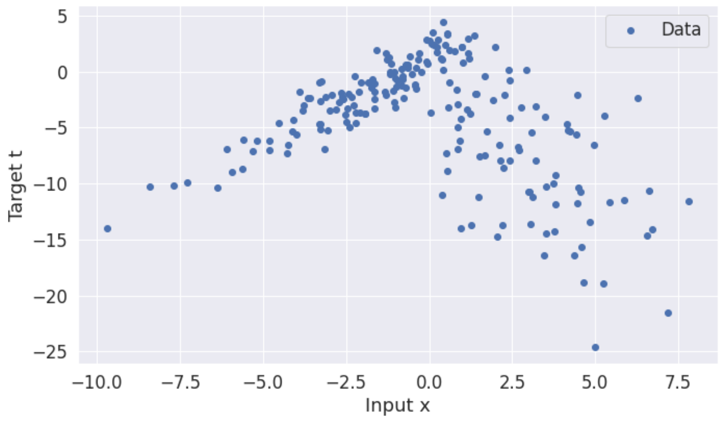

Figure 3: Dataset for regression

Consider the dataset for regression given in Figure 3. It is evident from the plot that there is a strong non-linear dependency between the target variables y_n and the input variables x_n. We also observe that the noise variance seems to depend on the input x. Consequently, assuming $y_n = y(x_n) + e_n$, where e_n is independent and identically distributed (i.i.d) noise would not be appropriate.

In this part, we will work with a so-called *Mixture of experts* (MoE) model for regression. The core idea is to model the dataset using several submodels, which are 'experts' in their own region of the input space. For example, the dataset in Figure 3 could be well-represented using two linear models: a linear model with positive slope and relative low noise variance for the 'left half' of the dataset and another linear model with negative slope and larger noise variance on the 'right half' of the dataset. Our goal is to simultaneously learn these two linear models as well as when to use which model.

To construct this model, we will consider two different linear models and then introduce a latent binary variable for each data point, $z_n \in \{0, 1\}$, to control which of the two linear models that should explain the data point y_n. That is, if $z_n = 0$ we assume y_n should be explained a linear model with parameters $\boldsymbol{w}_0$ and if $z_n = 1$, then y_n should be explained by a linear model with weights $\boldsymbol{w}_1$.

We model each z_n using Bernoulli distributions as follows

$$p(z_n|\pi_n) = \text{Ber}(z_n|\pi_n), \tag{6}$$

where $\pi_n = \sigma(\boldsymbol{v}^T \boldsymbol{x}_n)$ is the probability of $z_n = 1$, $\sigma(\cdot)$ is the logistic sigmoid function and $\boldsymbol{v}$ is a parameter vector to be estimated. That is, we basically model the latent z_n variable using a logistic regression model. We can set up a conditional likelihood as follows

$$p(y_n|x_n, z_n, \boldsymbol{w}_0, \boldsymbol{w}_1, \sigma_0^2, \sigma_1^2) = \begin{cases} \mathcal{N}(y_n|\boldsymbol{w}_1^T \boldsymbol{x}_n, \sigma_1^2) & \text{if} \quad z_n = 1, \\ \mathcal{N}(y_n|\boldsymbol{w}_0^T \boldsymbol{x}_n, \sigma_0^2) & \text{if} \quad z_n = 0, \end{cases} = \mathcal{N}(y_n|\boldsymbol{w}_{z_n}^T \boldsymbol{x}_n, \sigma_{z_n}^2) \tag{7}$$

where we also allow the noise variance to depend on z_n. We complete the model with generic priors

$$\tau, \sigma_0^2, \sigma_1^2 \sim \mathcal{N}_+(0, 1) \tag{8}$$

$$\boldsymbol{w}_0, \boldsymbol{w}_1, \boldsymbol{v} \sim \mathcal{N}(\boldsymbol{0}, \tau^2 \boldsymbol{I}) \tag{9}$$

$$z_n|\boldsymbol{v} \sim \text{Ber}(\sigma(\boldsymbol{v}^T \boldsymbol{x}_n)) \tag{10}$$

$$y_n|z_n \sim \mathcal{N}(\boldsymbol{w}_{z_n}^T \boldsymbol{x}_n, \sigma_{z_n}^2), \tag{11}$$

where $\mathcal{N}_+(0,1)$ is the half-normal distribution. This leads to a joint model of the form

$$p(\boldsymbol{y}, \boldsymbol{w}_1, \boldsymbol{w}_0, \boldsymbol{v}, \boldsymbol{z}, \tau, \sigma_0, \sigma_1) = \left[\prod_{n=1}^{N} \mathcal{N}(y_n|\boldsymbol{w}_{z_n}^T \boldsymbol{x}_n, \sigma_{z_n}^2)\text{Ber}(z_n|\sigma(\boldsymbol{v}^T\boldsymbol{x}_n))\right]\mathcal{N}(\boldsymbol{w}_0|\boldsymbol{0}, \tau^2\boldsymbol{I})$$
$$\mathcal{N}(\boldsymbol{w}_1|\boldsymbol{0}, \tau^2\boldsymbol{I})\mathcal{N}(\boldsymbol{v}|\boldsymbol{0}, \tau^2\boldsymbol{I})\mathcal{N}_+(\tau^2|0,1)\mathcal{N}_+(\sigma_0|0,1)\mathcal{N}_+(\sigma_1|0,1). \tag{12}$$

The purpose is now to fit this model to the dataset in Figure 3 using MCMC. The dataset can be found on DTU Learn as loaded as follows:

```
data = jnp.load('./data_assignment3.npz')
x, y = data['x'], data['t']
```

To avoid handling the intercepts in the linear models explicitly, we will use the convention $\boldsymbol{x_n} = [x_n,\ 1]$.

Task 2.1: Marginalize out each z_n from to joint model in eq. (12) to obtain a joint distribution, where the likelihood for each observation is a mixture of two Gaussian distributions.

Hints: Use the sum rule to marginalize out z_n.

Solution

Denoting $p(\boldsymbol{w}_0, \boldsymbol{w}_1, \boldsymbol{v}, \tau^2, \sigma_0, \sigma_1) = \mathcal{N}(\boldsymbol{w}_0|\boldsymbol{0}, \tau^2\boldsymbol{I})\mathcal{N}(\boldsymbol{w}_1|\boldsymbol{0}, \tau^2\boldsymbol{I})\mathcal{N}(\boldsymbol{v}|\boldsymbol{0}, \tau^2\boldsymbol{I})\mathcal{N}_+(\tau^2|0,1)\mathcal{N}_+(\sigma_0|0,1)\mathcal{N}_+(\sigma_1|0,1)$ and using the sum rule, we get

$$p(\boldsymbol{y}, \boldsymbol{w}_1, \boldsymbol{w}_0, \boldsymbol{v}, \boldsymbol{z}, \tau, \sigma_0, \sigma_1) \overset{(a)}{=} \sum_{\boldsymbol{z}} p(\boldsymbol{y}, \boldsymbol{w}_1, \boldsymbol{w}_0, \boldsymbol{v}, \boldsymbol{z}, \tau, \sigma_0, \sigma_1) \tag{13}$$
$$\overset{(b)}{=} \sum_{\boldsymbol{z}} \left[\prod_{n=1}^{N} \mathcal{N}(y_n|\boldsymbol{w}_{z_n}^T \boldsymbol{x}_n, \sigma_{z_n}^2)\text{Ber}(z_n|\sigma(\boldsymbol{v}^T\boldsymbol{x}_n))\right] p(\boldsymbol{w}_0, \boldsymbol{w}_1, \boldsymbol{v}, \tau^2, \sigma_0, \sigma_1) \tag{14}$$
$$\overset{(c)}{=} \left[\sum_{\boldsymbol{z}} \left[\prod_{n=1}^{N} \mathcal{N}(y_n|\boldsymbol{w}_{z_n}^T \boldsymbol{x}_n, \sigma_{z_n}^2)\text{Ber}(z_n|\sigma(\boldsymbol{v}^T\boldsymbol{x}_n))\right]\right] p(\boldsymbol{w}_0, \boldsymbol{w}_1, \boldsymbol{v}, \tau^2, \sigma_0, \sigma_1) \tag{15}$$
$$\overset{(d)}{=} \left[\prod_{n=1}^{N} \sum_{z_n} \mathcal{N}(y_n|\boldsymbol{w}_{z_n}^T \boldsymbol{x}_n, \sigma_{z_n}^2)\text{Ber}(z_n|\sigma(\boldsymbol{v}^T\boldsymbol{x}_n))\text{d}z_n\right] p(\boldsymbol{w}_0, \boldsymbol{w}_1, \boldsymbol{v}, \tau^2, \sigma_0, \sigma_1) \tag{16}$$
$$\overset{(e)}{=} \left[\prod_{n=1}^{N} (1-\sigma(\boldsymbol{v}^T\boldsymbol{x}_n))\mathcal{N}(y_n|\boldsymbol{w}_0^T \boldsymbol{x}_n, \sigma_0^2) + \sigma(\boldsymbol{v}^T\boldsymbol{x}_n)\mathcal{N}(y_n|\boldsymbol{w}_1^T \boldsymbol{x}_n, \sigma_1^2)\right] p(\boldsymbol{w}_0, \boldsymbol{w}_1, \boldsymbol{v}, \tau^2, \sigma_0, \sigma_1), \tag{17}$$

where in (a) we use the sum rule, in (b) we substitute in the distributions concerning $\boldsymbol{z}$, in (c) we use the fact that $p(\boldsymbol{w}_0, \boldsymbol{w}_1, \boldsymbol{v}, \tau^2, \sigma_0, \sigma_1)$ is independent of $\boldsymbol{z}$, in (d) we use the fact that $\mathbb{E}[xy] = \mathbb{E}[x]\mathbb{E}[y]$ when x, y are independent and finally, in (e) we compute the expectations with respect to each z_n. Therefore,

$$p(\boldsymbol{y}, \boldsymbol{w}_1, \boldsymbol{w}_0, \boldsymbol{v}, \boldsymbol{z}, \tau, \sigma_0, \sigma_1) \tag{18}$$
$$= \left[\prod_{n=1}^{N} (1-\sigma(\boldsymbol{v}^T\boldsymbol{x}_n))\mathcal{N}(y_n|\boldsymbol{w}_0^T \boldsymbol{x}_n, \sigma_0^2) + \sigma(\boldsymbol{v}^T\boldsymbol{x}_n)\mathcal{N}(y_n|\boldsymbol{w}_1^T \boldsymbol{x}_n, \sigma_1^2)\right] \tag{19}$$
$$\mathcal{N}(\boldsymbol{w}_0|\boldsymbol{0}, \tau^2\boldsymbol{I})\mathcal{N}(\boldsymbol{w}_1|\boldsymbol{0}, \tau^2\boldsymbol{I})\mathcal{N}(\boldsymbol{v}|\boldsymbol{0}, \tau^2\boldsymbol{I})\mathcal{N}_+(\tau^2|0,1)\mathcal{N}_+(\sigma_0|0,1)\mathcal{N}_+(\sigma_1|0,1) \tag{20}$$

Task 2.2: Implement a Python function to evaluate the marginalized log joint distribution

Hints: Let θ denote all the parameters of the model, i.e. $\theta = \left\{\boldsymbol{w}_0, \boldsymbol{w}_1, \boldsymbol{v}, \tau, \sigma_0^2, \sigma_1^2\right\}$, then implement a function that takes θ and returns $\ln p(\boldsymbol{y}, \boldsymbol{w}_1, \boldsymbol{w}_0, \boldsymbol{v}, \tau, \sigma_0, \sigma_1)$.

Solution There are many ways to implement the log joint, e.g.

```
from scipy.stats import norm

sigmoid = lambda x: 1./(1 + jnp.exp(-x))

def halfnormal(x):
    if x < 0:
        return 0
    else:
        return 2*norm.pdf(x, 0, 1)

def log_halfnormal(x):
    if x < 0:
        return -jnp.inf
    else:
        return jnp.log(2) + norm.logpdf(x, 0, 1)

def unpack(params):
    w1 = params[:2]
    w2 = params[2:4]
    v = params[4:6]
    tau = params[6]
    sigma1 = params[7]
    sigma2 = params[8]
    return w1, w2, v, tau, sigma1, sigma2

def log_likelihood(X, y, params):
    w1, w2, v, tau, sigma1, sigma2 = unpack(params)

    g = X@v
    f1 = X@w1
    f2 = X@w2

    pi = sigmoid(g)

    if tau > 0 and sigma1 > 0 and sigma2 > 0:
        # could be implemented numerically more robust via logsumexp
        return jnp.log((1-pi)*norm.pdf(y, f1, sigma1) + pi*norm.pdf(y, f2, sigma2)).sum()
    else:
        return -jnp.inf

def log_prior(params):
    w1, w2, v, tau, sigma1, sigma2 = unpack(params)
    log_p = log_halfnormal(tau) + log_halfnormal(sigma2) + log_halfnormal(sigma1)
    log_p += norm.logpdf(w1, 0, tau).sum() + norm.logpdf(w2, 0, tau).sum() + norm.logpdf(v, 0,

    return log_p

# precompute design matrix
N = len(x)
X = jnp.column_stack((x, jnp.ones(N)))

# define log joint
```

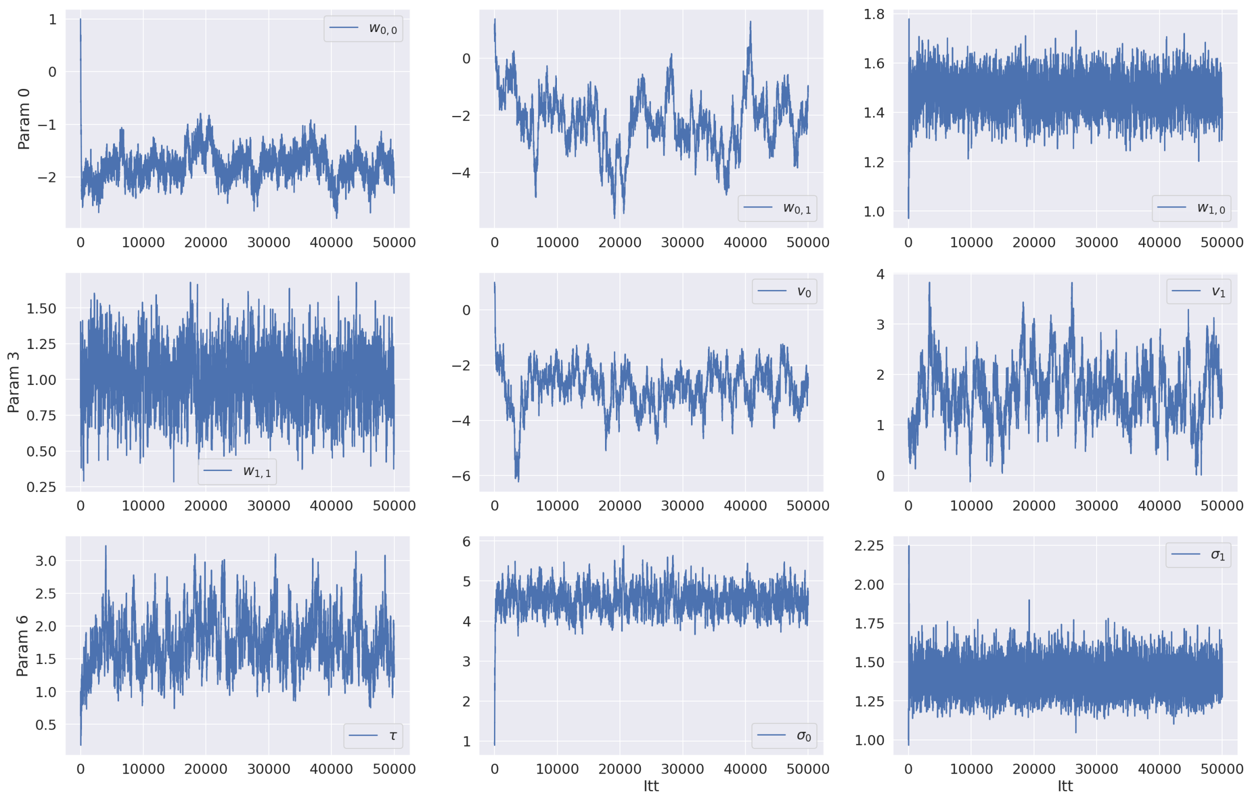

Figure 4: Trace of MCMC run for all 9 parameters.

```
log_joint_data = lambda X, y, params: log_likelihood(X, y, params)  + log_prior(params)
log_joint = lambda p: log_joint_data(X, y, p)
```

Task 2.3: Run a Metropolis-Hasting sampler to infer all parameters. Explain the settings you used (number of iterations, proposal distribution etc).

Hints: Feel free to use the Metropolis-Hasting code from the exercises. Plot the trace for all parameters to assess convergence (convergence diagnostics might be tricky due to the multimodality of the model). Compute the posterior mean of the weights $\boldsymbol{w}_0, \boldsymbol{w}_1, \boldsymbol{v}$*, and plot the resulting function on top of the data as a sanity check. Make sure to initialize the sampler using a valid parameter vector, i.e. positive scale parameters* $\sigma_0, \sigma_1, \tau > 0$.

Solution

```
num_params = 9
tau = 0.1
num_iter = 50000
seed = 123
params = jnp.ones(num_params)
samples, accept = metropolis(log_joint, num_params, tau, num_iter, theta_init=params, seed=123)
```

Figure 4 plots the trace of all 9 parameters of the model obtained using 50000 iterations with a Gaussian proposal distribution with std. dev. 10^{-1}.

Task 2.4: Report the posterior mean and 95% credibility intervals for all parameters.

Solution

We can estimate means and intervals as follows

```
params_samples_after_warmup = samples[25000:, :]
for i in range(num_params):
    # compute mean
    m = jnp.mean(params_samples_after_warmup[:, i])
    # intervals
    lower, upper = jnp.percentile(params_samples_after_warmup[:, i], jnp.array([2.5, 97.5]))
    print(f'{param_names[i]:15s} Mean = {m:+4.3f}\tInterval = [{lower:+4.3f}, {upper:+4.3f}]')
```

which yields

```
w_{0,0}         Mean = +1.471     Interval = [+1.342, +1.601]
w_{0,1}         Mean = +0.984     Interval = [+0.613, +1.369]
w_{1,0}         Mean = -1.735     Interval = [-2.289, -1.251]
w_{1,1}         Mean = -2.502     Interval = [-4.518, -0.536]
v_{0}           Mean = +2.379     Interval = [+1.590, +3.138]
v_{1}           Mean = -1.547     Interval = [-2.735, -0.725]
\tau            Mean = +1.747     Interval = [+1.003, +2.719]
\sigma0         Mean = +1.399     Interval = [+1.237, +1.589]
\sigma1         Mean = +4.500     Interval = [+3.904, +5.150]
```

Task 2.5: For $x \in [-12, 12]$, plot posterior predictive distribution for $p(\pi^*|\boldsymbol{y}, \boldsymbol{x}^*)$, $p(y^*|\boldsymbol{y}, \boldsymbol{x}^*, z^* = 0)$ and $p(y^*|\boldsymbol{y}, x, z^* = 1)$ on top of the data.

Hints: For inspiration on how to plot these distributions using samples, see the exercise on Gibbs sampling for change point detection.

Solution

Given the set of posterior samples, we can first compute the posterior samples of $\pi^*_{(i)}$, $y^*|z^* = 0$, $y^*|z^* = 1$, where (i) indicated the i'th sample:

$$\pi^*_{(i)} = \sigma(\boldsymbol{v}^T_{(i)}\boldsymbol{x}^*) \tag{21}$$

$$y^*_{(i)}|z^* = 0 \sim \mathcal{N}\left(\left(\boldsymbol{w}_0^{(i)}\right)^T \boldsymbol{x}^*, \left(\sigma_0^{(i)}\right)^2\right) \tag{22}$$

$$y^*_{(i)}|z^* = 1 \sim \mathcal{N}\left(\left(\boldsymbol{w}_1^{(i)}\right)^T \boldsymbol{x}^*, \left(\sigma_1^{(i)}\right)^2\right) \tag{23}$$

$$\tag{24}$$

and then compute the means and intervals using the `numpy.percentile`-function.

```
# extract samples for plotting p(pi^*|y, x^*) using pi^* = sigma(v^T x^*)
w0_samples = params_samples_after_warmup[:, 0:2].T
w1_samples = params_samples_after_warmup[:, 2:4].T
v_samples = params_samples_after_warmup[:, 4:6].T
sigma0_samples = params_samples_after_warmup[:, 7]
sigma1_samples = params_samples_after_warmup[:, 8]

# prep inputs
xp = jnp.linspace(-12, 12, 1000)
```

```
Xp = jnp.column_stack((xp, jnp.ones(len(xp))))

# prep keys
key = random.PRNGKey(123)
key1, key2 = random.split(key)

# compute p(pi^*|y, x^*)
pi_star = sigmoid(Xp@v_samples)
pi_star_lower, pi_star_upper = jnp.percentile(pi_star, jnp.array([2.5, 97.5]), axis=1)
pi_star_mean = jnp.mean(pi_star, axis=1)

# compute p(y^*|y, x^*, z^*=0)
y_star_z0 = Xp@w0_samples + sigma0_samples*random.normal(key1, shape=sigma0_samples.shape)
y_star_z0_lower, y_star_z0_upper = jnp.percentile(y_star_z0, jnp.array([2.5, 97.5]), axis=1)
y_star_z0_mean = jnp.mean(y_star_z0, axis=1)

# compute p(y^*|y, x^*, z^*=1)
y_star_z1 = Xp@w1_samples + sigma1_samples*random.normal(key2, shape=sigma1_samples.shape)
y_star_z1_lower, y_star_z1_upper = jnp.percentile(y_star_z1, jnp.array([2.5, 97.5]), axis=1)
y_star_z1_mean = jnp.mean(y_star_z1, axis=1)

# plot
fig, ax = plt.subplots(1, 2, figsize=(25, 10))
ax[0].plot(xp, pi_star_mean, 'b-', label='$p(\\pi^*|\\mathbf{y}, x^*)$')
ax[0].fill_between(xp, pi_star_lower, pi_star_upper, color='b', alpha=0.3)
ax[1].plot(xp, y_star_z0_mean, 'g-', label='$p(y^*|\\mathbf{y}, x^*, z^*=0)$')
ax[1].fill_between(xp, y_star_z0_lower, y_star_z0_upper , color='g', alpha=0.3)
ax[1].plot(xp, y_star_z1_mean, 'r-', label='$p(y^*|\\mathbf{y}, x^*, z^*=1)$')
ax[1].fill_between(xp, y_star_z1_lower, y_star_z1_upper , color='r', alpha=0.3)

for i in range(2):
    ax[i].legend()
    ax[i].set(xlabel='x')
ax[0].plot(x, 0.7 + 0.05*y, 'o', alpha=0.5, label='Shifted and scale data')
ax[1].plot(x, y, 'o', alpha=0.5, label='data')
ax[0].legend()
ax[1].legend()
```

From the plot in Figure 5, we can see that the "change point" of the data roughly matches the position where $\pi^* = 0.5$ as expected (note that the data has been shifted and scale in the left panel to make this more clear). We can also that two conditional linear models appear to fit the left and right most part of the data nicely.

Task 2.6: For $x \in [-12, 12]$, plot posterior predictive distribution for $p(y^*|\boldsymbol{y}, \boldsymbol{x}^*)$

Solution

To compute the posterior predictive distribution, we repeat the same procedure as above, except that we now sample z^* as well.

```
# sample y^* based on posterior samples
def predictive_sample(key, params):
    # split key
    key_z, key_y1, key_y2 = random.split(key, num=3)
```

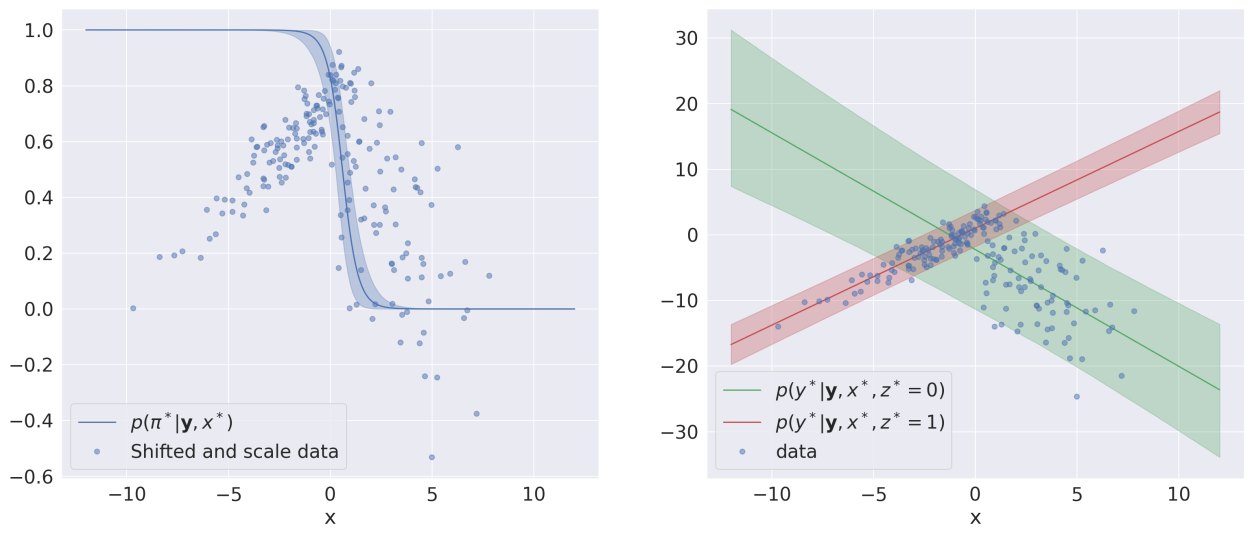

Figure 5: Plots for task 2.5

```
    w1_, w2_, v_, tau_, sigma1_, sigma2_ = unpack(params)

    pi = sigmoid(Xp@v_)
    z = random.binomial(key_z, 1, pi)
    y1 = Xp@w1_ + sigma1_*random.normal(key_y1)
    y2 = Xp@w2_ + sigma2_*random.normal(key_y2)
    return (1-z)*y1 + z*y2

# collect statistics
key = random.PRNGKey(123)
keys = random.split(key, num=len(params_samples_after_warmup))
y_samples = jnp.vstack([predictive_sample(key, p) for key, p in zip(keys, params_samples_after_warmu
y_mean = y_samples.mean(0)
y_lower, y_upper = jnp.percentile(y_samples, jnp.array([2.5, 97.5]), axis=0)
# plot
fig, ax = plt.subplots(1, 1, figsize=(12, 8))
ax.plot(x, y, '.', label='Data')
ax.plot(xp, y_mean, 'b-', label='$p(y^*|\\mathbf{y}, x^*)$')
ax.fill_between(xp, y_lower, y_upper, alpha=0.3, color='b')
ax.set(xlabel='x')
ax.legend()
```

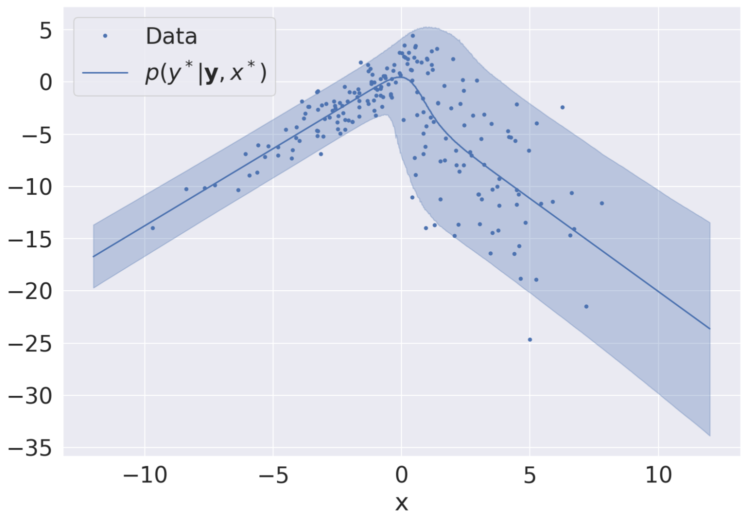

Figure 6: Plot for task 2.6