02477 Bayesian Machine Learning Exam F24R - solution

Rev 1.0

```
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
import seaborn as snb

snb.set_style('darkgrid')
```

Part 1: Multi-class classification

Question 1.1

The prior distribution is identified as $p(\boldsymbol{W}) = \prod_{ij} \mathcal{N}(W_{ij}|0, \alpha^{-1})$ and the likelihood is given by $p(y_n|\boldsymbol{f}_n) = \text{Categorical}(\text{softmax}(\boldsymbol{f}_n))$, or alternatively, $p(y_n|\boldsymbol{W}) = \text{Categorical}(\text{softmax}(\boldsymbol{W}\phi(\boldsymbol{x}_n)))$.

Question 1.2

```
# MAP estimate
Wmap = np.array([[-0.5, -2], [3.0, 0.0], [1.0, 1.0]])

# point for prediction
xstar = -1
phi = lambda x: np.array([1, x])
phi_star = phi(xstar)

# apply linear model and softmax
softmax = lambda x: np.exp(x)/np.sum(np.exp(x))
p_plugin = softmax(Wmap@phi_star)


print('Wmap matrix for plug-in approximation')
print(np.array2string(Wmap, precision=2))
print('')


print('Approximate posterior predictive distribution for x^* using plugin approximation')
for i in range(3):
    print(f'p(y**={i+1}|x^*) = {p_plugin[i]:3.2f}')
```

```
Wmap matrix for plug-in approximation
[[-0.5 -2. ]
 [ 3.   0. ]
 [ 1.   1. ]]

Approximate posterior predictive distribution for x^* using plugin approximation
p(y**=1|x^*) = 0.18
p(y**=2|x^*) = 0.79
p(y**=3|x^*) = 0.04
```

Question 1.3

```python
W1 = np.array([[-0.15, -1.92], [3.2, 0.45], [1.37, 0.80]])
W2 = np.array([[-0.31, -2.03], [2.98, 0.08], [1.03, 1.29]])
W3 = np.array([[-0.35, -1.98], [3.09, 0.07], [1.3, 0.96]])

print('Posterior samples of W')
print(np.array2string(W1, precision=2))
print('')
print(np.array2string(W2, precision=2))
print('')
print(np.array2string(W3, precision=2))
print('')

# computer MC estimate
W_samples = [W1, W2, W3]
p_samples = [softmax(Wi@phi_star) for Wi in W_samples]
p_mc = np.mean(p_samples, axis=0)

print('Approximate posterior predictive distribution for x^* using MC estimation')
for i in range(3):
    print(f'p(y**={i+1}|x^*) = {p_mc[i]:3.2f}')
```

```
Posterior samples of W
[[-0.15 -1.92]
 [ 3.2   0.45]
 [ 1.37  0.8 ]]

[[-0.31 -2.03]
 [ 2.98  0.08]
 [ 1.03  1.29]]

[[-0.35 -1.98]
 [ 3.09  0.07]
 [ 1.3   0.96]]

Approximate posterior predictive distribution for x^* using MC estimation
p(y**=1|x^*) = 0.22
p(y**=2|x^*) = 0.72
p(y**=3|x^*) = 0.05
```

Question 1.4

```python
# pred. dist
p = np.array([0.00, 0.27, 0.73])
```

```
def entropy(p):
    idx = p > 0
    return -np.sum(p[idx]*np.log(p[idx]))

confidence = lambda x: np.max(x)


print(f'Confidence is: {confidence(p):3.2f}')
print(f'Entropy (natural log) is: {entropy(p):3.2f}')
```

```
Confidence is: 0.73
Entropy (natural log) is: 0.58
```

Question 1.5

When α is **increased**, the prior variance α^{-1} is **decreased** and hence the prior becomes tighter around $W_{ij} = 0$. Consequently, the weights will be regularized stronger towards zero and thus the coefficients of MAP estimate are expected to be numerically smaller. Alternatively, we can argue that the posterior is a compromise between the prior and posterior and if the prior concentrates around zero, the mass of the posterior will around shift towards $W_{ij} = 0$.

Part 2: Gaussian process regression

```
# data
x = np.array([-2, 0, 2])[:, None]
y = np.array([-2.01, 1.41, 0.23])[:, None]
```

Question 2.1

By comparing the expression for the kernel k_1 to the standard formulation of the squared exponential, we can identify the magnitude of the kernel is $\kappa = \sqrt{2}$ and the lengthscale to $\ell = 2$.

Question 2.2

```
def squared_exponential(tau, kappa, lengthscale):
    return kappa**2*np.exp(-0.5*tau**2/lengthscale**2)

class StationaryIsotropicKernel(object):

    def __init__(self, kernel_fun, kappa=1., lengthscale=1.0):
        """
            the argument kernel_fun must be a function of three arguments kernel_fun(||tau||, kappa, lengthscale), e.g.
            squared_exponential = lambda tau, kappa, lengthscale: kappa**2*np.exp(-0.5*tau**2/lengthscale**2)
        """
        self.kernel_fun = kernel_fun
        self.kappa = kappa
        self.lengthscale = lengthscale

    def contruct_kernel(self, X1, X2, kappa=None, lengthscale=None, jitter=1e-8):
        """ compute and returns the NxM kernel matrix between the two sets of input X1 (shape NxD) and X2 (MxD) using the stationary and isotropic covariance function speci

        arguments:
```

```python
            X1              -- NxD matrix
            X2              -- MxD matrix
            kappa           -- magnitude (positive scalar)
            lengthscale     -- characteristic lengthscale (positive scalar)
            jitter          -- non-negative scalar

        returns
            K               -- NxM matrix
        """

        # extract dimensions
        N, M = X1.shape[0], X2.shape[0]

        # prep hyperparameters
        kappa = self.kappa if kappa is None else kappa
        lengthscale = self.lengthscale if lengthscale is None else lengthscale

        # compute all the pairwise distances efficiently
        dists = np.sqrt(np.sum((np.expand_dims(X1, 1) - np.expand_dims(X2, 0))**2, axis=-1))

        # squared exponential covariance function
        K = self.kernel_fun(dists, kappa, lengthscale)

        # add jitter to diagonal for numerical stability
        if len(X1) == len(X2) and np.allclose(X1, X2):
            K = K + jitter*np.identity(len(X1))

        assert K.shape == (N, M), f"The shape of K appears wrong. Expected shape ({N}, {M}), but the actual shape was {K.shape}. Please check your code. "
        return K


class GaussianProcessRegression(object):

    def __init__(self, X, y, kernel, kappa=1., lengthscale=1., sigma=1/2, jitter=1e-8):
        """
        Arguments:
            X                -- NxD input points
            y                -- Nx1 observed values
            kernel           -- must be instance of the StationaryIsotropicKernel class
            jitter           -- non-negative scaler
            kappa            -- magnitude (positive scalar)
            lengthscale      -- characteristic lengthscale (positive scalar)
            sigma            -- noise std. dev. (positive scalar)
        """
        self.X = X
        self.y = y
        self.N = len(X)
        self.kernel = kernel
        self.jitter = jitter
        self.set_hyperparameters(kappa, lengthscale, sigma)

    def set_hyperparameters(self, kappa, lengthscale, sigma):
        self.kappa = kappa
        self.lengthscale = lengthscale
        self.sigma = sigma

    def predict_f(self, Xstar):
        """ returns the posterior distribution of f^* evaluated at each of the points in x^* conditioned on (X, y)
        Arguments:
        Xstar            -- PxD prediction points

        returns:
        mu               -- Px1 mean vector
        Sigma            -- PxP covariance matrix
        """

        # prepare relevant matrices
        k = self.kernel.contruct_kernel(Xstar, self.X, self.kappa, self.lengthscale, jitter=self.jitter)
        K = self.kernel.contruct_kernel(self.X, self.X, self.kappa, self.lengthscale, jitter=self.jitter)
        Kstar = self.kernel.contruct_kernel(Xstar, Xstar, self.kappa, self.lengthscale, jitter=self.jitter)

        # Compute C matrix
        C = K + self.sigma**2*np.identity(len(self.X))

        # computer mean and Sigma
        mu = np.dot(k, np.linalg.solve(C, self.y))
        Sigma = Kstar - np.dot(k, np.linalg.solve(C, k.T))

        # sanity check for dimensions
        assert (mu.shape == (len(Xstar), 1)), f"The shape of the posterior mu seems wrong. Expected ({len(Xstar)}, 1), but actual shape was {mu.shape}. Please check impleme
        assert (Sigma.shape == (len(Xstar), len(Xstar))), f"The shape of the posterior Sigma seems wrong. Expected ({len(Xstar)}, {len(Xstar)}), but actual shape was {Sigma

        return mu, Sigma
```

```
# specify hyperparameters
ell = 2
kappa = np.sqrt(2)
kernel = StationaryIsotropicKernel(squared_exponential, kappa, ell)

K = kernel.contruct_kernel(x, x)
print('Prior mean\n', np.zeros(len(x)), '\n')
print('Prior covariance\n', np.array2string(K, precision=2))

# The prior covariance could also be computed using a double for loop over k(x, x')
```

```
Prior mean
 [0. 0. 0.]

Prior covariance
 [[2.   1.21 0.27]
 [1.21 2.   1.21]
 [0.27 1.21 2.  ]]
```

The prior distribution for $\boldsymbol{f}$ is given by $p(\boldsymbol{f}|\boldsymbol{x}) = \mathcal{N}(\boldsymbol{f}|\mathbf{0}, \boldsymbol{K})$, where

$$\boldsymbol{K} = \begin{bmatrix} 2 & 1.21 & 0.27 \\ 1.21 & 2 & 1.21 \\ 0.27 & 1.21 & 2 \end{bmatrix}$$

Question 2.3

```
# specify hyperparameters
ell = 2
kappa = np.sqrt(2)
sigma = 0.5
model = GaussianProcessRegression(x, y, kernel, kappa=kappa, lengthscale=ell, sigma=sigma)

mu_f, Sigma_f = model.predict_f(x)

print('posterior mean of f')
print(np.array2string(mu_f, precision=2))
print()
print('posterior covariance of f')
print(np.array2string(Sigma_f, precision=2))
```

```
posterior mean of f
[[-1.53]
 [ 0.89]
 [ 0.43]]

posterior covariance of f
[[ 0.21  0.03 -0.01]
 [ 0.03  0.19  0.03]
 [-0.01  0.03  0.21]]
```

The analytical posterior distribution of $\boldsymbol{f}$ given $\boldsymbol{y}$ is $p(\boldsymbol{f}|\boldsymbol{y}) = \mathcal{N}(\boldsymbol{y}|\boldsymbol{m}, \boldsymbol{\Sigma})$, where the posterior mean $\boldsymbol{m}$ is given by

$$\boldsymbol{m} = \boldsymbol{K}(\boldsymbol{K} + \sigma^2\boldsymbol{I})^{-1}\boldsymbol{y} = \begin{bmatrix} -1.53 \\ 0.89 \\ 0.43 \end{bmatrix}$$

and the posterior variance

$$\boldsymbol{K} = \boldsymbol{K} - \boldsymbol{K}(\boldsymbol{K} + \sigma^2\boldsymbol{I})^{-1}\boldsymbol{K} = \begin{bmatrix} 0.21 & 0.03 & -0.01 \\ 0.03 & 0.19 & 0.03 \\ -0.01 & 0.03 & 0.21 \end{bmatrix},$$

where $\boldsymbol{K}_{ij} = k(x_i, x_j)$ is the prior covariance between $f(x_i)$ and $f(x_j)$. The expressions for $\boldsymbol{m}$ and $\boldsymbol{K}$ can either be derived from the standard GP regression equations for $\boldsymbol{x}^* = \boldsymbol{x}$ or via the equations for linear Gaussian systems in Section 3.3 in Murphy1.

Question 2.4

The analytical prior variance of $f(x)$ under k_2 is given by

$$\mathrm{Var}\left[f(x)\right] = k_2(x, x) = \exp(-\frac{1}{2}||0 - 0||_2) + 2 = 1 + 2 = 3$$

for all $x \in \mathbb{R}$.

Part 3: A mixture model

```python
# data and hyperparameters
X = np.array([[1, 0.5], [-1, 1]])
y = np.array([1, 0])
m = np.array([1, 1])
tau2 = 1.
sigma2 = 1.
```

Question 3.1

The model $p(\mathbf{y}|\boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{y}|\boldsymbol{X}\boldsymbol{\theta}, \sigma^2\boldsymbol{I})$ is a linear model with a Gaussian likelihood. Hence, we can compute the maximum likelihood solution using the normal equations

$$\hat{\boldsymbol{\theta}}_{\text{MLE}} = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{y} = \begin{bmatrix} 0.67 \\ 0.67 \end{bmatrix}$$

```python
# solution to the normal equations
theta_MLE = np.linalg.solve(X.T@X, X.T@y)
print('theta MLE = ', np.array2string(theta_MLE, precision=2))
```

```
theta MLE =  [0.67 0.67]
```

Question 3.2

We evaluate the prior density, $p(\boldsymbol{\theta})$, at $\boldsymbol{\theta} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$ to get:

$$\begin{aligned} p\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix}\right) &= \frac{1}{2}\mathcal{N}\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix} | -\boldsymbol{m}, \tau^2\boldsymbol{I}\right) + \frac{1}{2}\mathcal{N}\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix} |\boldsymbol{m}, \tau^2\boldsymbol{I}\right) \\ &= \frac{1}{2}\mathcal{N}\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix} | -\begin{bmatrix} 1 \\ 1 \end{bmatrix}, \boldsymbol{I}\right) + \frac{1}{2}\mathcal{N}\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix} | \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \boldsymbol{I}\right) \\ &\approx 0.06 \end{aligned}$$

```python
from scipy.stats import multivariate_normal as mvn
prior = lambda theta: 0.5*mvn.pdf(theta, -m, tau2*np.identity(2)) + 0.5*mvn.pdf(theta, m, tau2*np.identity(2))
print(f'The prior evaluated at theta = [0, 0] is {prior(np.array([0,0])):3.2f}')
```

```
The prior evaluated at theta = [0, 0] is 0.06
```

Question 3.3

The marginal likelihood is given by sum rule:

$$
\begin{aligned}
p(\mathbf{y}) &= \int p(\boldsymbol{y}|\boldsymbol{\theta})p(\theta)\mathrm{d}\boldsymbol{\theta} \\
&= \int \mathcal{N}(\boldsymbol{y}|\boldsymbol{X\theta}, \sigma^2\boldsymbol{I}) \left[\frac{1}{2}\mathcal{N}(\boldsymbol{\theta}| -\boldsymbol{m}, \tau^2\boldsymbol{I}) + \frac{1}{2}\mathcal{N}(\boldsymbol{\theta}|\boldsymbol{m}, \tau^2\boldsymbol{I})\right] \mathrm{d}\boldsymbol{\theta} \\
&= \frac{1}{2}\int \mathcal{N}(\boldsymbol{y}|\boldsymbol{X\theta}, \sigma^2\boldsymbol{I})\mathcal{N}(\boldsymbol{\theta}| -\boldsymbol{m}, \tau^2\boldsymbol{I})\mathrm{d}\boldsymbol{\theta} + \frac{1}{2}\int \mathcal{N}(\boldsymbol{y}|\boldsymbol{X\theta}, \sigma^2\boldsymbol{I})\mathcal{N}(\boldsymbol{\theta}|\boldsymbol{m}, \tau^2\boldsymbol{I})\mathrm{d}\boldsymbol{\theta}
\end{aligned}
$$

Both integrals are now recognized as a marginalization of a linear Gaussian system, which can be solved via the equations in Section 3.3 in Murphy1 to give

$$
p(\mathbf{y}) = \frac{1}{2}\mathcal{N}(\boldsymbol{y}| -\boldsymbol{Xm}, \tau^2\boldsymbol{XX}^T + \sigma^2\boldsymbol{I}) + \frac{1}{2}\mathcal{N}(\boldsymbol{y}|\boldsymbol{Xm}, \tau^2\boldsymbol{XX}^T + \sigma^2\boldsymbol{I})
$$

Question 3.4

Bayes rule yields

$$
p(\boldsymbol{\theta}|\boldsymbol{y}) = \frac{p(\boldsymbol{y}|\boldsymbol{\theta})p(\boldsymbol{\theta})}{p(\boldsymbol{y})}
$$

and evaluating for $\boldsymbol{\theta} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$ yields

$$
\begin{aligned}
p\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix} |\boldsymbol{y}\right) &= \frac{\mathcal{N}\left(\boldsymbol{y}|\boldsymbol{X}\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \sigma^2\boldsymbol{I}\right)\left[\frac{1}{2}\mathcal{N}\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix} | -\boldsymbol{m}, \tau^2\boldsymbol{I}\right) + \frac{1}{2}\mathcal{N}\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix} |\boldsymbol{m}, \tau^2\boldsymbol{I}\right)\right]}{\frac{1}{2}\mathcal{N}(\boldsymbol{y}| -\boldsymbol{Xm}, \tau^2\boldsymbol{XX}^T + \sigma^2\boldsymbol{I}) + \frac{1}{2}\mathcal{N}(\boldsymbol{y}|\boldsymbol{Xm}, \tau^2\boldsymbol{XX}^T + \sigma^2\boldsymbol{I})} \\
&\approx \frac{0.10 \cdot 0.06}{0.04} \approx 0.15
\end{aligned}
$$

```python
theta0 = np.array([0,0])
C = tau2*X@X.T + sigma2*np.identity(2)

evidence = 0.5*mvn.pdf(y, -X@m, C) + 0.5*mvn.pdf(y, X@m, C)
lik_term = mvn.pdf(y, X@theta0, sigma2*np.eye(2))
prior_term = prior(theta0)
post_term = lik_term*prior_term/evidence

print(f'p(y|theta) = {lik_term:3.2f}')
print(f'p(theta) = {prior_term:3.2f}')
print(f'p(y) = {evidence:3.2f}')
print(f'p(theta|y) = {post_term:3.2f}')
```

```
p(y|theta) = 0.10
p(theta) = 0.06
p(y) = 0.04
p(theta|y) = 0.15
```

Part 4: A generalized linear model

Question 4.1

Based on the plots we identify the MLE as approximately

$$\boldsymbol{w}_{\{\text{MLE}\}} \approx \begin{bmatrix} 0.5 \\ -0.2 \end{bmatrix}$$

And the MAP as approximately

$$\boldsymbol{w}_{\{\text{MAP}\}} \approx \begin{bmatrix} -0.25 \\ -0.25 \end{bmatrix}$$

Question 4.2

For $x^* = 0$, the prior mean becomes

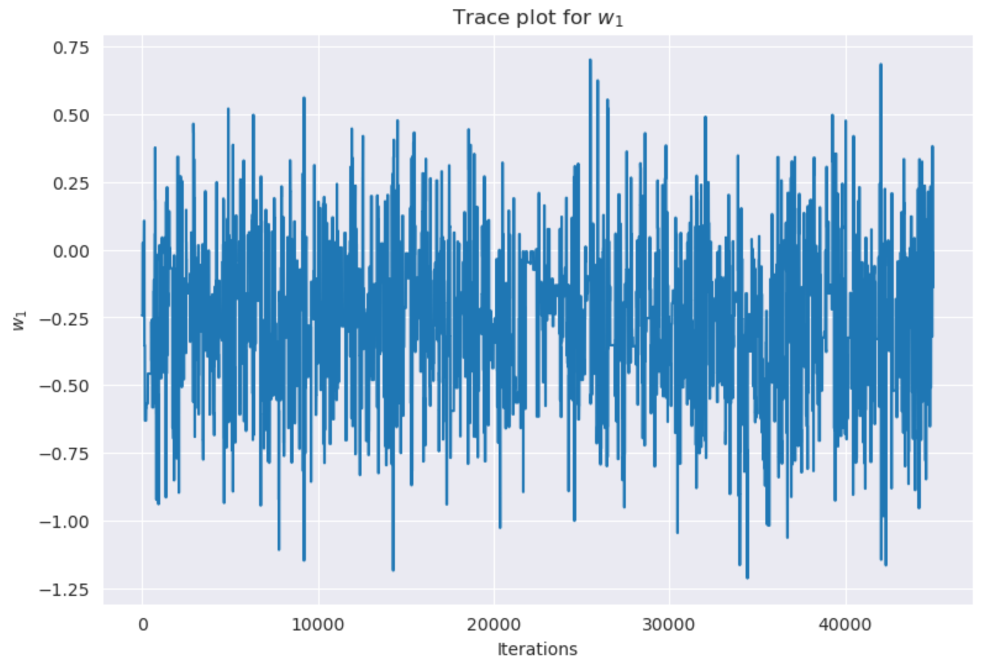
$$\mathbb{E}\left[\mu(0)\right] = \mathbb{E}\left[\exp(3 + w_1 0 + w_2 0^2)\right] = \mathbb{E}\left[\exp(3)\right] = \exp(3)$$

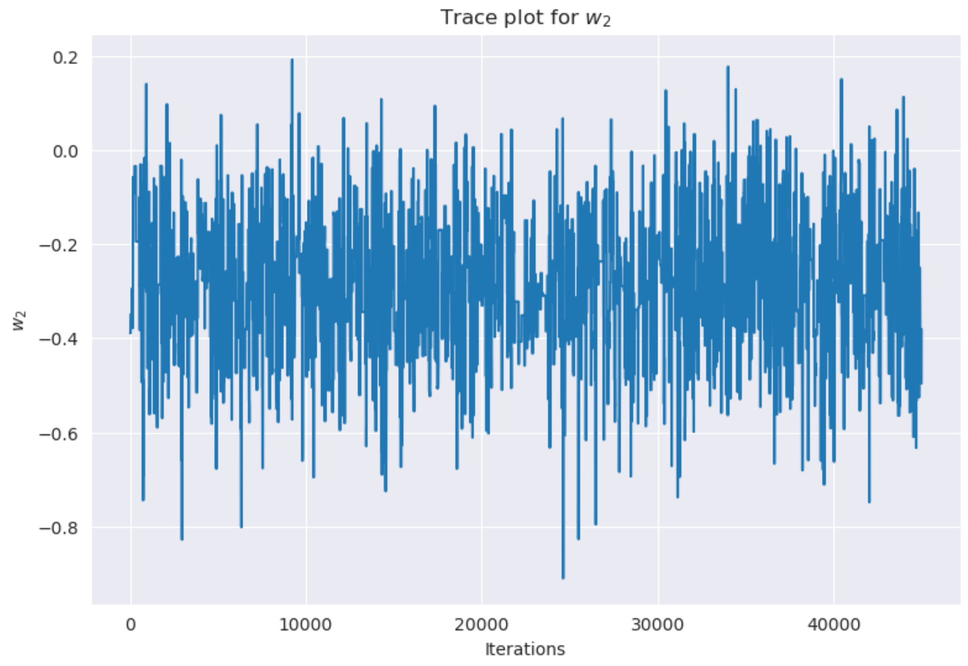
Note that the argument that $\mathbb{E}[\mu(0)]$ is zero because the prior on $\boldsymbol{w}$ is zero in not correct.

Question 4.3

```
def metropolis(log_joint, num_params, tau, num_iter, theta_init=None, seed=None):
    """ Runs a Metropolis-Hastings sampler

        Arguments:
        log_joint:          function for evaluating the log joint distribution
        num_params:         number of parameters of the joint distribution (integer)
        tau:                standard deviation of the Gaussian proposal distribution (positive real)
        num_iter:           number of iterations (interger)
        theta_init:             vector of initial values (np.array with shape (num_params) or None)
        seed:               seed (integer or None)

        returns
        thetas                  np.array with MCMC samples (np.array with shape (num_iter+1, num_params))
        accept_rate         acceptance rate (non_negative scalar)
    """

    if seed is not None:
        np.random.seed(seed)

    if theta_init is None:
        theta_init = np.zeros((num_params))

    # prepare lists
    thetas = [theta_init]
    accepts = []
    log_p_x = log_joint(theta_init)
```

```python
    for k in range(num_iter):

        # get the last value for x and generate new proposal candidate
        x_cur = thetas[-1]
        x_star = x_cur + np.random.normal(0, tau, size=(num_params))

        # evaluate the log density for the candidate sample
        log_p_x_star = log_joint(x_star)

        # compute acceptance probability
        log_r = log_p_x_star - log_p_x
        A = min(1, np.exp(log_r))

        # accept new candidate with probability A
        if np.random.uniform() < A:
            x_next = x_star
            log_p_x = log_p_x_star
            accepts.append(1)
        else:
            x_next = x_cur
            accepts.append(0)

        thetas.append(x_next)


    thetas = np.stack(thetas)
    return thetas, np.mean(accepts)


# data
x = np.array([1., 2., 3.])
y = np.array([10, 4, 1])
alpha = 8

# design matrix
X = np.column_stack((x, x**2))

from scipy.stats import poisson

# implement log joint
def log_joint(w):

    mu = np.exp(3 + X@w)
    log_prior = mvn.logpdf(w, np.zeros(2), 1/alpha*np.identity(2))
    log_lik = poisson.logpmf(y, mu).sum()

    return log_prior + log_lik


samples, accept = metropolis(log_joint, num_params=2, tau=1, num_iter=50000, theta_init=np.array([0, 0]))

# remove warm-up
samples = samples[5000:, :]

fig, ax = plt.subplots(1, 2, figsize=(20, 6))
for i in range(2):
    ax[i].plot(samples[:, i])
    ax[i].set(xlabel='Iterations', ylabel=f'$w_{i+1}$', title=f'Trace plot for $w_{i+1}$')
```





Question 4.4

```
print(f'p(w_1 > 0|y) = {np.mean(samples[:, 0] > 0):3.2f}')
```

```
p(w_1 > 0|y) = 0.16
```

Questions 4.5

```
xstar = 1.5
mu_samples = np.exp(3 + samples[:, 0]*xstar + samples[:, 1]*xstar**2)
print(f'p(mu^* > 7|y) = {np.mean(mu_samples > 7):3.2f}')
```

```
p(mu^* > 7|y) = 0.57
```

Question 4.6

```
y_samples = np.random.poisson(mu_samples)
np.percentile(mu_samples, [5, 95])
```

```
array([ 5.00137416, 10.46926495])
```

Part 5: A non-linear Gaussian model

Question 5.1

We have that

$$\mathbb{E}\left[y\right] = \frac{1}{S}\sum_{i=1}^{S} y^{(i)} \approx 1.67,$$

where $y^{(i)}|w^{(i)} \sim \mathcal{N}(e^{w^{(i)}}, 1)$ and $w^{(i)} \sim \mathcal{N}(0, 1)$.

Alternatively, we could also reason that $y = e^w + \epsilon$, where $\epsilon \sim \mathcal{N}(0, \sigma^2)$ and $w \sim \mathcal{N}(0, 1)$. The prior mean of y can therefore also be computed estimated as

$$\mathbb{E}\left[e^w + \epsilon\right] = \mathbb{E}\left[e^w\right] \approx \frac{1}{S}\sum_{i=1}^{S} e^{w^{(i)}} \approx 1.57,$$

for $w^{(i)} \sim \mathcal{N}(0, 1)$. The specific numerical result may vary a bit depending on the random seed.

```
np.random.seed(0)

# ancestral sampling
S = 1000
w_samples = np.random.normal(0, 1, size=S)
y_samples = np.random.normal(np.exp(w_samples), 1, size=S)
print(f'Prior mean of y = {np.mean(y_samples):3.2f}')
```

```
Prior mean of y = 1.57
```

Questions 5.2

We have

$$\log p(y, w) = \log \mathcal{N}(y|e^w, 1) + \log \mathcal{N}(w|0, 1)$$

and thus the log joint $y = 5$ and $w = w_{\text{MAP}}$ becomes

$$\log p(5, w{\text{MAP}}) = \log \mathcal{N}(5|e^{w{\text{WMAP}}}, 1) + \log \mathcal{N}(w_{\text{WMAP}}|0, 1) \approx -3.59

```
# log pdf of univariate Gaussian
log_npdf = lambda x, m, v: -(x-m)**2/(2*v) - 0.5*np.log(2*np.pi*v)

# model
log_prior = lambda w: log_npdf(w, 0, 1)
log_lik = lambda w: log_npdf(5, np.exp(w), 1)
log_joint = lambda w: log_prior(w) + log_lik(w)

w_map = 1.293404

print(f'Log joint at w_map: {log_joint(w_map):3.2f}')
```

```
Log joint at w_map: -3.59
```

Question 5.3

The logarithm of the joint disitribution is

$$\begin{aligned}\log p(y,w) &= \log \mathcal{N}(y|e^w, 1) + \log \mathcal{N}(w|0,1)\\ &= -\frac{1}{2}(y-e^w)^2 - \frac{1}{2}w^2 + \text{const}\end{aligned}$$

Computing the first derivative wrt. w yields

$$\begin{aligned}\frac{\partial}{\partial w}\log p(y,w) &= -\frac{1}{2}\frac{\partial}{\partial w}(y-e^w)^2 - \frac{1}{2}\frac{\partial}{\partial w}w^2 + \text{const}\\ &= (y-e^w)e^w - w \qquad \text{(via chain rule)}\\ &= ye^w - e^{2w} - w\end{aligned}$$

The second derivative becomes

$$\frac{\partial^2}{\partial w^2}\log p(y,w) = y\frac{\partial}{\partial w}e^w - \frac{\partial}{\partial w}e^{2w} - \frac{\partial}{\partial w}w = ye^w - 2e^{2w} - 1$$

The second derivative at $w = w_{\text{MAP}}$ is then

$$H = ye^{w_{\text{MAP}}} - 2e^{2w_{\text{MAP}}} - 1 \approx -9.34$$

Since the Laplace approximation is given by $p(w|y) \approx q(w) = \mathcal{N}(w|w_{\text{MAP}}, -H^{-1})$, the approximate posterior mean is $\mathbb{E}[w|y] = w_{\text{MAP}} \approx 1.29$ and the approximate posterior variance is $\mathbb{V}[w|y] = -\frac{1}{9.34} \approx 0.11$

```python
y = 5


H = y*np.exp(w_map) - 2*np.exp(2*w_map) - 1
posterior_mean = w_map
posterior_var = -1./H

print(f'The posterior mean is: {posterior_mean:3.2f}')
print(f'The posterior variance is: {posterior_var:3.2f}')
```

```
The posterior mean is: 1.29
The posterior variance is: 0.11
```

Question 5.4

Since the Laplace approximations assumes a Gaussian distribution, and because Gaussians are symmetric, the approximate posterior probability of the event $p > w_{\text{MAP}}$ will be 0.5 by construction.