# 02477 Bayesian Machine Learning Exam F24 - solution

Rev 1.1

# Part 1: Linear Gaussian systems

## Question 1.1

Since $p(\mathbf{x}_1)$ and $p(\mathbf{x_2}|\mathbf{x}_1)$ constitute a linear Gaussian system, the distribution $p(\mathbf{x}_1|\mathbf{x}_2)$ will also be Gaussian, i.e.

$$p(\mathbf{x}_1|\mathbf{x}_2) = \mathcal{N}(\mathbf{x}_1|\mathbf{m}_{1|2}, \mathbf{V}_{1|2}),$$

and we can use the equations from Section 3.3 in Murphy1 to compute the mean and covariance:

$$\begin{aligned}\mathbf{V}_{1|2} &= \left(\mathbf{\Sigma}^{-1} + \mathbf{A^T\Sigma A}\right)^{-1},\\ \mathbf{m}_{1|2} &= \mathbf{V}_{1|2}\mathbf{A}^T\mathbf{\Sigma}^{-1}\mathbf{x}_2.\end{aligned}$$

## Question 1.2

First, we compute $p(\mathbf{x}_2)$ using the sum rule and eq. (3.38) in Murphy1 to get:

$$\begin{aligned}p(\mathbf{x}_2) &= \int p(\mathbf{x}_2|\mathbf{x}_1)p(\mathbf{x}_1)\mathrm{d}\mathbf{x}_1\\ &= \int \mathcal{N}(\mathbf{x}_2|\mathbf{A}\mathbf{x_1}, \mathbf{\Sigma})\mathcal{N}(\mathbf{x}_1|\mathbf{0}, \mathbf{\Sigma})\mathrm{d}\mathbf{x}_1\\ &= \mathcal{N}(\mathbf{x}_2|\mathbf{0}, \Sigma + \mathbf{A\Sigma A}^T),\end{aligned}$$

and then use the sum rule a second time to compute $p(\mathbf{x}_3)$:

$$\begin{aligned}p(\mathbf{x}_3) &= \int p(\mathbf{x}_3|\mathbf{x}_2)p(\mathbf{x}_2)\mathrm{d}\mathbf{x}_2\\ &= \int \mathcal{N}(\mathbf{x}_3|\mathbf{A}\mathbf{x_2}, \mathbf{\Sigma})\mathcal{N}(\mathbf{x}_2|\mathbf{0}, \Sigma + \mathbf{A\Sigma A}^T)\mathrm{d}\mathbf{x}_2\\ &= \mathcal{N}(\mathbf{x}_3|\mathbf{0}, \mathbf{\Sigma} + \mathbf{A\Sigma A}^T + \mathbf{AA\Sigma A}^T\mathbf{A}^T),\end{aligned}$$

where we again use eq. (3.38) from Murphy1 in the last step.

## Question 1.3

The conditional distribution is given by

$$p(\mathbf{x}_3|\mathbf{x}_1) = \frac{p(\mathbf{x}_3, \mathbf{x}_1)}{p(\mathbf{x}_1)}.$$

Since we already now the prior of $\mathbf{x}_1$, we only need to compute the joint disitribution of $\mathbf{x}_1$ and $\mathbf{x}_3$.

This can be obtained by marginalizing the joint distribution $p(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3)$ wrt. $\mathbf{x}_2$:

$$\begin{aligned}p(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3) &= p(\mathbf{x}_3|\mathbf{x}_2)p(\mathbf{x}_2|\mathbf{x}_1)p(\mathbf{x}_1)\\ &= \mathcal{N}(\mathbf{x}_3|\mathbf{A}\mathbf{x_2}, \mathbf{\Sigma})\mathcal{N}(\mathbf{x}_2|\mathbf{A}\mathbf{x_1}, \mathbf{\Sigma})\mathcal{N}(\mathbf{x}_1|\mathbf{0}, \mathbf{\Sigma})\end{aligned}$$

to get

$$\begin{aligned}p(\mathbf{x}_1, \mathbf{x}_3) &= \int \mathcal{N}(\mathbf{x}_3|\mathbf{A}\mathbf{x_2}, \mathbf{\Sigma})\mathcal{N}(\mathbf{x}_2|\mathbf{A}\mathbf{x_1}, \mathbf{\Sigma})\mathrm{d}\mathbf{x}_2\mathcal{N}(\mathbf{x}_1|\mathbf{0}, \mathbf{\Sigma})\\ &= \mathcal{N}(\mathbf{x}_3|\mathbf{A}^2\mathbf{x_1}, \mathbf{\Sigma} + \mathbf{A\Sigma A}^T)\mathcal{N}(\mathbf{x}_1|\mathbf{0}, \mathbf{\Sigma}),\end{aligned}$$

where we again used eq. (3.38) from Murphy1. Hence,

$$p(\mathbf{x}_3|\mathbf{x}_1) = \mathcal{N}(\mathbf{x}_3|\mathbf{A}^2\mathbf{x_1}, \mathbf{\Sigma} + \mathbf{A\Sigma A}^T).$$

---

# Part 2: Gaussian process regression

## Question 2.1

From eq. (4) in the exam set, we can see that $y_n$ is distributed according to $y_n \sim \mathcal{N}(y_n|f(x_n), \sigma^2)$ and hence, the likelihood is Gaussian.

More specifically, the likelihood for a single observation is $p(y_n|f_n) = \mathcal{N}(y_n|f(x_n), \sigma^2)$, and equivalently, $p(\mathbf{y}|\mathbf{f}) = \mathcal{N}(\mathbf{y}|\mathbf{f}, \sigma^2\mathbf{I})$ for the full dataset.

## Question 2.2

The prior distribution of $f^* = f(x^*)$ for any $x^*$ is given by $p(f^*) = \mathcal{N}(f^*|0, k_1(x^*, x^*)) = \mathcal{N}(f^*|0, \kappa^2)$.

Hence, the prior predictive distribution is given by

$$p(y^*|x^* = 1) = \mathcal{N}(y^*|0, \kappa^2 + \sigma^2) = \mathcal{N}(y^*|0, 0.53)$$

```
import numpy as np
kappa = 0.7
ell = 0.5*np.sqrt(2)
sigma = 0.2

print(f'p(y^*|x^*=1) = N(y^*|0, {kappa**2 + sigma**2:3.2f})')
```

```
p(y^*|x^*=1) = N(y^*|0, 0.53)
```

## Question 2.3

Since $x^* = 227$ is far away from all inputs in the training set (in terms of number of lengthscales $\ell$), the covariance between $f(x^*)$ and $\mathbf{f} = \begin{bmatrix} f(x_1) & \dots & f(x_N) \end{bmatrix}$ will be zero, and therefore, the posterior distribution of $f^*$ will be equation to the prior distribution of $f^*$, and hence, the posterior predictive distribution for $x^* = 227$ will be identical to the prior predictive distribution. That is,

$$p(y^*|x^* = 227) = \mathcal{N}(y^*|0, \kappa^2 + \sigma^2) = \mathcal{N}(y^*|0, 0.53)$$

Here you could also compute the posterior predictive distribution for $x^* = 227$ to arrive at the same answer.

# Question 2.4

Gaussian process regression with a Gaussian likelihood is a conjugate model and therefore, we can compute the marginal likelihood analytical, which is given by:

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

where $\mathbf{K}$ is the covariance matrix for the latent function values $\mathbf{f}$, i.e. $\operatorname{cov}(\mathbf{f}, \mathbf{f}) = \mathbf{K}$ and $\sigma$ is the standard deviation of the noise.

We can evaluate the log PDF of the distribution above directly, or we can use the code from the course (week5):

```
from week5 import squared_exponential
from week5 import StationaryIsotropicKernel
from week5 import GaussianProcessRegression
```

```
# data
x = np.array([-2.17, 1.99, 0.57, -3.01, -1.16, 3.30, -4.85, -0.86])[:, None]
y = np.array([0.88, 0.46, -0.06, 0.98, 0.45, 0.88, -0.66, 0.05])[:, None]

# using the code from the course
gp = GaussianProcessRegression(x, y, StationaryIsotropicKernel(squared_exponential))
print(f'log marginal likelihood: {gp.log_marginal_likelihood(kappa, ell, sigma):3.2f} (course code)')

# or by evaluating a multivariate normal directly:
from scipy.stats import multivariate_normal as mvn
kern =  StationaryIsotropicKernel(squared_exponential)
C = kern.contruct_kernel(x, x, kappa=kappa, lengthscale=ell) + sigma**2*np.identity(8)
print(f'log marginal likelihood: {mvn.logpdf(y.ravel(), np.zeros(8), C):3.2f} (direct evaluation)')
```

```
log marginal likelihood: -6.65 (course code)
log marginal likelihood: -6.65 (direct evaluation)
```

Therefore,

$$\log p(y|\kappa, \ell, \sigma) = \log \mathcal{N}(\mathbf{y}|\mathbf{0}, \mathbf{K} + \sigma^2\mathbf{I}) \approx -6.65.$$

## Question 2.5

The equations for the posterior distribution of $f^* = f(x^*)$ for Gaussian process regression is given by:

$$
\begin{aligned}
p(f^*|\mathbf{y}, \mathbf{x}^*) &= \mathcal{N}\left(f^*|\mu_{f^*|\mathbf{y}}, \sigma^2_{f^*|\mathbf{y}}\right), \\
\mu_{f^*|\mathbf{y}} &= \mathbf{k}\left(\mathbf{K} + \sigma^2\mathbf{I}\right)^{-1}\mathbf{y}, \\
\sigma^2_{f^*|\mathbf{y}} &= k_1(\mathbf{x}^*, \mathbf{x}^*) - \mathbf{k}\left(\mathbf{K} + \sigma^2\mathbf{I}\right)^{-1}\mathbf{k}^T,
\end{aligned}
$$

where $\mathbf{k}$ is the covariance between $f^*$ and $\mathbf{f}$ and $\mathbf{K}$ is the prior covariance matrix for $\mathbf{f}$.

This equations can be evaluated directly or using the code from the course

```
xstar = np.array([[1]])
gp.set_hyperparameters(kappa, ell, sigma)
mean_fstar, var_fstar = gp.predict_f(xstar)
print(f'p(f^*|y, x^*=1) = N(f^*|{mean_fstar[0,0]:3.2f}, {var_fstar[0,0]:3.2f})')
```

```
p(f^*|y, x^*=1) = N(f^*|0.07, 0.14)
```

Hence, the posterior distribution for $f^*$ for $x^* = 1$ is given by

$$
p(f^*|\mathbf{y}, \mathbf{x}^*) = \mathcal{N}\left(f^*|0.07, 0.14\right)
$$

The posterior predictive for $y^* = f^* + \epsilon$ can be obtained using `gp.predict_y` or simply by adding the variance of the noise to the variance of the posterior of $f^*$:

$$
p(y^*|\mathbf{y}, \mathbf{x}^*) = \mathcal{N}\left(y^*|0.07, 0.14 + \sigma^2\right) = \mathcal{N}\left(y^*|0.07, 0.18\right)
$$

```
print(f'p(y^*|y, x^*=1) = N(y^*|{mean_fstar[0, 0]:3.2f}, {var_fstar[0,0] + sigma**2:3.2f})')
```

```
p(y^*|y, x^*=1) = N(y^*|0.07, 0.18)
```

## Question 2.6

The second term is the expression for $k_2$ cannot be written as a function of $x - x'$ and hence, the kernel function $k_2$ **cannot** be stationary, and since, it is not stationary, it **cannot** be isotropic either.

## Question 2.7

Inserting the hyperparameter values into the kernel expression yields

$$
k_2(x, x') = \left(1 + \frac{||x - x'||}{2(\frac{1}{\sqrt{2}})^2}\right)^{-1} + xx' = \left(1 + ||x - x'||\right)^{-1} + xx'
$$

Next, we evaluate the kernel function $k_2$ for $x^* = -1$ and $\mathbf{x}$:

```
k2 = lambda x, xprime: 1./(1 + np.abs(x - xprime)) + x*xprime

xstar = np.array(-1)
for xn in x:
    print(f'{np.array2string(k2(xn, xstar), precision=2)}', end=', ')
```

```
[2.63], [-1.74], [-0.18], [3.34], [2.02], [-3.11], [5.06], [1.74],
```

Hence, the covariance between $\mathbf{f}$ and $f^*$ is

$$
\text{cov}(\mathbf{f}, f^*) = \begin{bmatrix} 2.63 & -1.74 & -0.18 & 3.34 & 2.02 & -3.11 & 5.06 & 1.74 \end{bmatrix}.
$$

---

# Part 3: Binary classification

## Question 3.1

There are several accepted answers for this questions. For example:

- If $\hat{\mathbf{w}}_{\text{MAP}}$ is a MAP estimator, i.e. $\hat{\mathbf{w}}_{\text{MAP}} = \arg\max_{\mathbf{w}} \log p(\mathbf{w}|\mathbf{y}) = \arg\max_{\mathbf{w}} \log p(\mathbf{w}, \mathbf{y})$, then we know that the gradient of the log joint must be zero (or numerically zero). So one could verify that the gradient of the log joint is indeed zero evaluated at $\hat{\mathbf{w}}_{\text{MAP}}$.
- One could also consider verifying that the value of the log joint increases in iteration of the maximization process
- Or verify that $\hat{\mathbf{w}}_{\text{MAP}}$ does indeed maximize $\log p(\mathbf{w}, \mathbf{y})$.

# Question 3.2

First, we observe that the model is linear model, so we can write the model as follows

$$f(x) = \mathbf{w}^T \phi(x)$$

for $\phi(x) = \begin{bmatrix} 1 \\ x \\ x^2 \end{bmatrix}$.

The plug-in approximation states that we should simply plug-in the point estimator $\hat{\mathbf{w}}_{\text{MAP}}$ into the likelihood and evaluate it for $x^* = -3$. That is,

$$p(y^*|\mathbf{y}, x^* = -3) \approx \text{Ber}(y^*|\sigma(\hat{\mathbf{w}}_{\text{MAP}}^T \phi(\mathbf{x}^*))) \approx \text{Ber}(y^*|0.91), \tag{2}$$

where $\phi(x) = \begin{bmatrix} 1 \\ x \\ x^2 \end{bmatrix}$

```
# data and non-linear transformation
xstar = -3
phi = lambda x: np.array([1, x, x**2])

# point estimator and inverse link function
w_MAP = np.array([2.647, -1.688, -0.596])
sigmoid = lambda x: 1./(1 + np.exp(-x))

# plug-in approximation
p = sigmoid(w_MAP.dot(phi(xstar)))
print(f'p(y^*|y, x^*=-3) = {p:3.2f} (using plug-in approximation)')
```

```
p(y^*|y, x^*=-3) = 0.91 (using plug-in approximation)
```

# Question 3.3

From eq. (7) in the exam set, we see that the approximate posterior mean for $w_0$ is $2.647$ (first entry in $\mathbf{w}_{\text{MAP}}$) and the approximate the posterior variance of $w_0$ is $3$ (The first element of the diagonal of $\mathbf{S}$, i.e. $\mathbf{S}_{11}$) and hence, the posterior standard deviation is $\sqrt{3}$. We can then compute a 90\% posterior credibility interval as

$$p(-0.20 \leq w_0 \leq 5.50|\mathbf{y}) = 0.9$$

The numbers can be computed in sevaral ways, e.g. via scipy.stats.norm, Monte Carlo or by recalling that a 90%-interval for Gaussian can be obtained as the mean plus/minus $1.645$ times the standard deviations from introductory statistics. All are perfectly valid solutions.

```
# via scipy.stats.norm
from scipy.stats import norm
lower, upper = norm.interval(0.9, loc=2.647, scale=np.sqrt(3))
print(f'[{lower:3.2f}, {upper:3.2f}] (via scipy.stats.norm)')

# Monte Carlo
w0_samples = np.random.normal(2.647, np.sqrt(3), size=10**6)
lower, upper = np.percentile(w0_samples, [5, 95])
print(f'[{lower:3.2f}, {upper:3.2f}] (Monte Carlo estimation)')

# Using knowledge fra introductory stats
z = 1.645
lower, upper = 2.647 - z*np.sqrt(3), 2.647 + z*np.sqrt(3)
print(f'[{lower:3.2f}, {upper:3.2f}] (via properties of the Gaussian)')
```

```
[-0.20, 5.50] (via scipy.stats.norm)
[-0.20, 5.50] (Monte Carlo estimation)
[-0.20, 5.50] (via properties of the Gaussian)
```

# Question 3.4

The random variable $f^* = \mathbf{w}^T \phi(\mathbf{x}^*)$ is an affine transformation of a Gaussian random variable and therefore, we have

$$f^* = \mathbf{w}^T \phi(\mathbf{x}^*) \sim \mathcal{N}(\hat{\mathbf{w}}_{\text{MAP}}^T \phi(\mathbf{x}^*), \phi(\mathbf{x}^*) \mathbf{S} \phi(\mathbf{x}^*)^T),$$

which evaluates to

$$p(f^*|\mathbf{y}, x^* = -3) = \mathcal{N}(f^*|2.35, 5.25).$$

with posterior mean $\mu_* = 2.35$ and the posterior variance equal to $\sigma_*^2 = 5.25$.

The probit approximation of the posterior predictive distribution then yields

$$p(y^*|\mathbf{y}, x^*) \approx \text{Ber}(y^*|p^*) = \text{Ber}(y^*|0.80), \quad \text{where} \quad p^* = \Phi\left(\frac{\mu_*}{\sqrt{\frac{8}{\pi} + \sigma_*^2}}\right) = 0.80,$$

where $\Phi(\cdot)$ is the CDF of the standard normal distribution.

```
w_MAP = np.array([2.647, -1.688, -0.596])
S = np.array([[3, -0.39, -0.3], [-0.39, 1.55, 0.37], [-0.3, 0.37, 0.14]])
xstar = -3

# compute p(f^*|y, x^*)
fstar_mean = w_MAP.dot(phi(xstar))
fstar_var = phi(xstar).T@S@phi(xstar)
print(f'p(f^*|y) = N(f^*|{fstar_mean:3.2f}, {fstar_var:3.2f})')

# compute p(y^*|y, x^*)
Phi = norm.cdf
p_ystar1 = Phi(fstar_mean/np.sqrt(8/np.pi + fstar_var))
print(f'p(y^* = 1|y, x^*=-3) = {p_ystar1:3.2f} (probit approximation)')
```

```
p(f^*|y) = N(f^*|2.35, 5.25)
p(y^* = 1|y, x^*=-3) = 0.80 (probit approximation)
```

## Queestion 3.5

The expected utility for $y^* = 0$ is given by

$$\mathbb{E}_{p(y^*|\mathbf{y},x^*)}\left[\mathcal{U}(y^*, \hat{y})\right] = p(y^* = 0|\mathbf{y}, x^*)\mathcal{U}(0, \hat{y}) + p(y^* = 1|\mathbf{y}, x^*)\mathcal{U}(1, \hat{y})$$

Hence, the expected utility for $\hat{y} = 0$ is

$$\mathbb{E}_{p(y^*|\mathbf{y},x^*)}\left[\mathcal{U}(y^*, 0)\right] = (1 - 0.129) \cdot \mathcal{U}(0, 0) + 0.129 \cdot \mathcal{U}(1, 0) = 0.87 \cdot 2 + 0.129 \cdot 1 = 1.87$$

and for $\hat{y} = 1$:

$$\mathbb{E}_{p(y^*|\mathbf{y},x^*)}\left[\mathcal{U}(y^*, 1)\right] = (1 - 0.129) \cdot \mathcal{U}(0, 1) + 0.129 \cdot \mathcal{U}(1, 1) = 0.87 \cdot 1 + 0.129 \cdot 2 = 1.13$$

```
p = 0.129
U = np.array([[2, 1], [1, 2]])
expected_utility0 = (1-p)*U[0,0] + p*U[1, 0]
expected_utility1 = (1-p)*U[0,1] + p*U[1, 1]

print(f'expected utility for hat y = 0: {expected_utility0:3.2f}')
print(f'expected utility for hat y = 1: {expected_utility1:3.2f}')
```

```
expected utility for hat y = 0: 1.87
expected utility for hat y = 1: 1.13
```

Hence, the optimal decision is $\hat{y} = 0$ because it maximizes the expected utility.

---

# Part 4: Variational inference

# Question 4.1

The entropy of a mean-field Gaussian family with dimension $D = 2$ is given by

$$\mathcal{H}\left[q\right] = \frac{1}{2}\sum_{i=1}^{D}\ln(2\pi e v_i) = \frac{1}{2}[\ln(2\pi e) + \ln(2\pi e)] = \ln(2\pi e) \approx 2.84,$$

where $e$ is the base of the natural logarithm.

```
m1, m2 = -1, 1
v1, v2 = 1, 1

H = np.log(2*np.pi*np.exp(1))
print(f'Entropy of q: {H:3.2f}')
```

```
Entropy of q: 2.84
```

## Question 4.2

We have

$$
\begin{aligned}
&\mathbb{E}_{q(w_1,w_2)}\left[\log p(y|w_1,w_2)\right] = \mathbb{E}_{q(w_1,w_2)}\left[-\frac{1}{2}\log(2\pi\sigma^2) - \frac{1}{2\sigma^2}(y-w_1w_2)^2\right] \\
&= -\frac{1}{2}\log(2\pi\sigma^2) - \frac{1}{2\sigma^2}\mathbb{E}_{q(w_1,w_2)}\left[(y-w_1w_2)^2\right] \qquad \text{(Using linearity)} \\
&= -\frac{1}{2}\log(2\pi\sigma^2) - \frac{1}{2\sigma^2}\mathbb{E}_{q(w_1,w_2)}\left[y^2 + w_1^2w_2^2 - 2yw_1w_2\right] \qquad \text{(Expanding)} \\
&= -\frac{1}{2}\log(2\pi\sigma^2) - \frac{1}{2\sigma^2}\left[y^2 + \mathbb{E}_{q(w_1,w_2)}\left[w_1^2w_2^2\right] - 2y\mathbb{E}_{q(w_1,w_2)}\left[w_1w_2\right]\right] \qquad \text{(Uisng linearity again)} \\
&= -\frac{1}{2}\log(2\pi\sigma^2) - \frac{1}{2\sigma^2}\left[y^2 + \mathbb{E}_{q(w_1,w_2)}\left[w_1^2\right]\mathbb{E}_{q(w_1,w_2)}\left[w_2^2\right] - 2y\mathbb{E}_{q(w_1,w_2)}\left[w_1\right]\mathbb{E}_{q(w_1,w_2)}\left[w_2\right]\right] \qquad \text{(Using the mean-field assumption)} \\
&= -\frac{1}{2}\log(2\pi\sigma^2) - \frac{1}{2\sigma^2}\left[y^2 + (m_1^2+v_1)(m_2^2+v_2) - 2ym_1m_2\right] \qquad \text{(Moments of Gaussians)},
\end{aligned}
$$

where we have used the for a Gaussian $\mathcal{N}(w|m,v)$, the first moment (mean) is $\mathbb{E}\left[w\right] = m$ and the second moment is $\mathbb{E}\left[w^2\right] = m^2 + v$ and $\mathbb{E}\left[w_1w_2\right] = \mathbb{E}\left[w_1\right]\mathbb{E}\left[w_2\right]$, when $w_1$ and $w_2$ are statistically independent.

If we prefer, we can simplify it further:

$$
\begin{aligned}
\mathbb{E}_{q(w_1,w_2)}\left[\log p(y|w_1,w_2)\right] &= -\frac{1}{2}\log(2\pi\sigma^2) - \frac{1}{2\sigma^2}\left[y^2 + m_1^2m_2^2 + m_1^2v_2 + v_1m_2^2 + v_1v_2 - 2ym_1m_2\right] \\
&= -\frac{1}{2}\log(2\pi\sigma^2) - \frac{1}{2\sigma^2}(y-m_1m_2)^2 - \frac{1}{2\sigma^2}\left[m_1^2v_2 + v_1m_2^2 + v_1v_2\right] \\
&= \log\mathcal{N}(y|m_1,m_2,\sigma^2) - \frac{1}{2\sigma^2}\left[m_1^2v_2 + v_1m_2^2 + v_1v_2\right]
\end{aligned}
$$

Evaluating it yields $\mathbb{E}_{q(w_1,w_2)}\left[\log p(y|w_1,w_2)\right] \approx -4.42$.

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

y = 1
sigma2 = 1

expected_loglik = log_npdf(y, m1*m2, sigma2) - 1/(2*sigma2)*(m1**2 * v2 + v1*m2**2 + v1*v1)
print(f'Expected loglik: {expected_loglik:3.2f}')
```

```
Expected loglik: -4.42
```

We can verify the result via simulation:

```
S = 1000000
w = np.random.normal(np.array([m1, m2]), np.sqrt(np.array([v1, v2])), size=(S, 2))
loglik = lambda w: log_npdf(y, w[:, 0]*w[:, 1], sigma2 )
print(f'Expected loglik: {np.mean(loglik(w)):3.2f} (via Monte Carlo)')
```

```
Expected loglik: -4.42 (via Monte Carlo)
```

## Question 4.3

The covariance between $w_1$ and $w_2$ is zero by construction for all members of the variational family due to the mean-field assumption. Hence, the covariance between $w_1$ and $w_2$ wrt. the optimal approximation $q^*$ will also be 0.

---

# Part 5

## Question 5.1

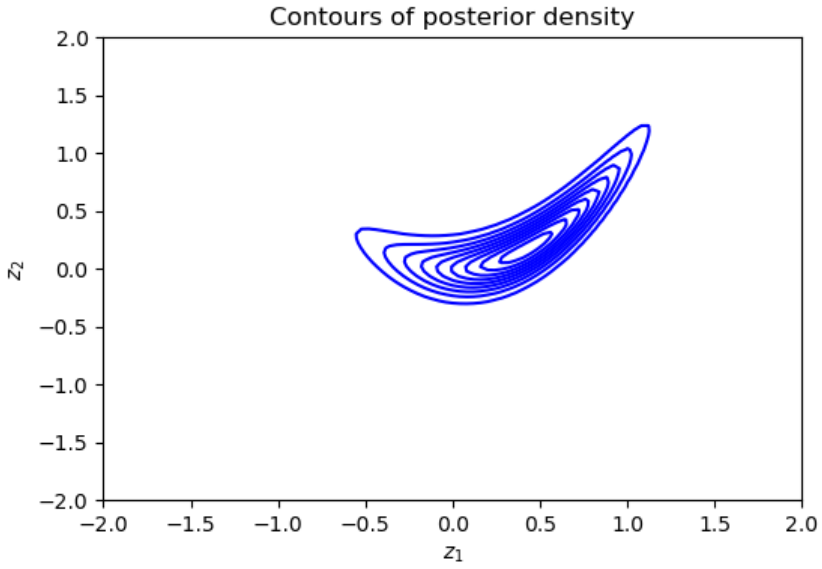
First we implement the log posterior (our target) and then we visualize it using the `Grid2D` -class from week2

```
import matplotlib.pyplot as plt
from week2 import Grid2D

z1s = np.linspace(-2, 2, 101)
z2s = np.linspace(-2, 2, 100)

log_target = lambda z1, z2:  -(1-z1)**2 - 20*(z2 - z1**2)**2   - z1**2 - z2**2
grid = Grid2D(z1s, z2s, log_target)

fig, ax = plt.subplots(1, 1, figsize=(6, 4))
grid.plot_contours(ax, f=np.exp)
ax.set(xlabel='$z_1$', ylabel='$z_2$', title='Contours of posterior density');
```



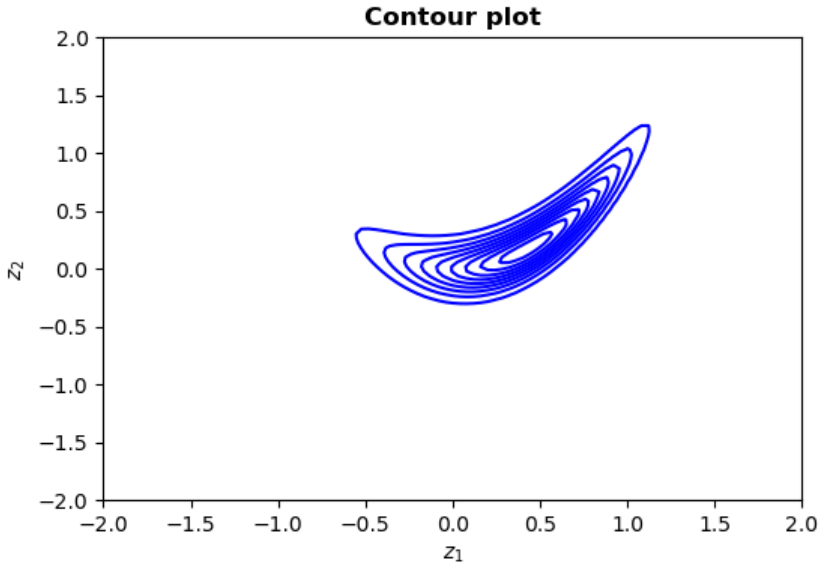

or we can evaluate it and plot it directly:

```
import matplotlib.pyplot as plt

Z = np.zeros((len(z1s), len(z2s)))
for i, z1 in enumerate(z1s):
    for j, z2 in enumerate(z2s):
        Z[i,j] = log_target(z1, z2)


fig, ax = plt.subplots(1, 1, figsize=(6, 4))
ax.contour(z1s, z2s, np.exp(Z).T, 10, colors='b')
ax.set(xlabel='$z_1$', ylabel='$z_2$')
ax.set_title('Contour plot', fontweight='bold')
```

```
Text(0.5, 1.0, 'Contour plot')
```




# Question 5.2

Using the implementation of the Metropolis algorithm from the exercise of week 9 yields

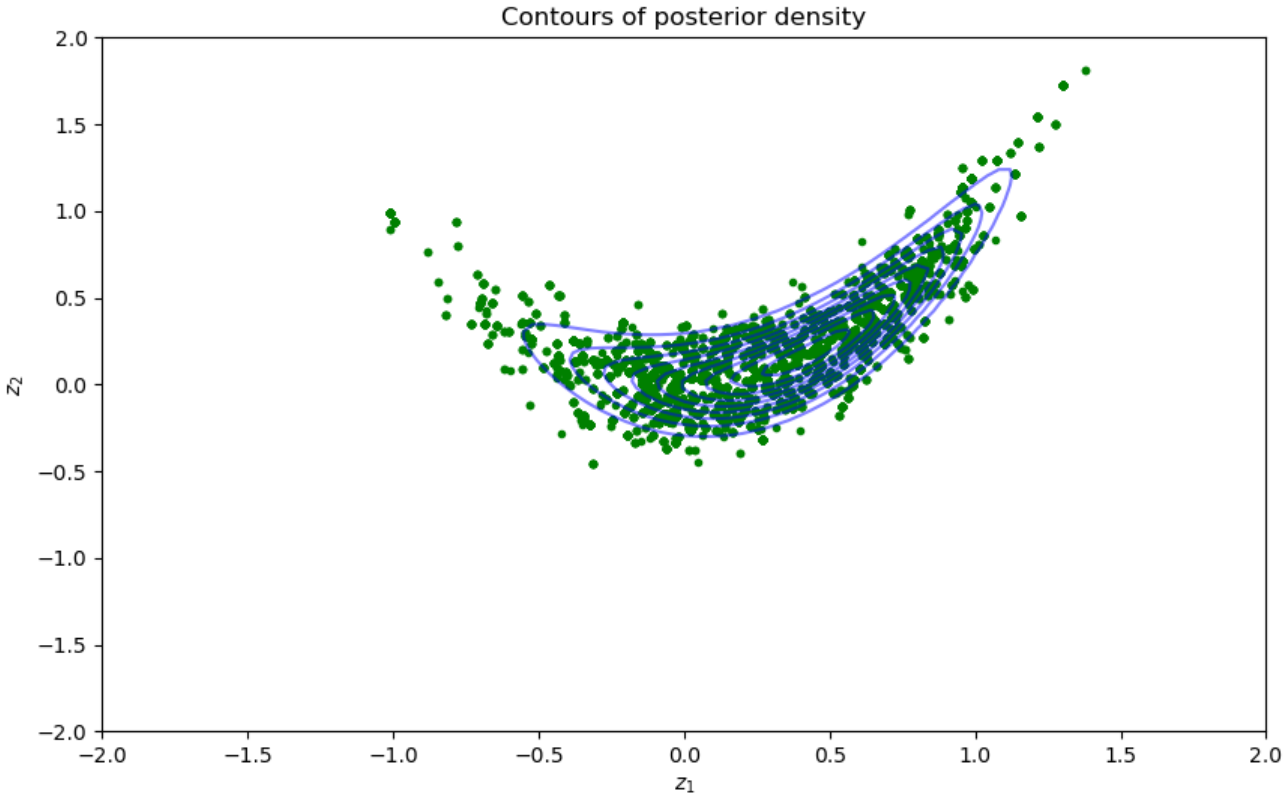
```
from week9 import metropolis
thetas, acc = metropolis(log_joint= lambda theta: log_target(theta[0], theta[1]),
                         num_params=2,
                         tau=1,
                         num_iter=10000,
                         seed=1,
                         theta_init=[0, 1.5])
print(f'Accept rate: {acc:3.2f}')

# discard warm-up
thetas = thetas[1000:, :]
```

```
Accept rate: 0.09
```

Let's do a sanity check by plotting the samples on top of the contours:

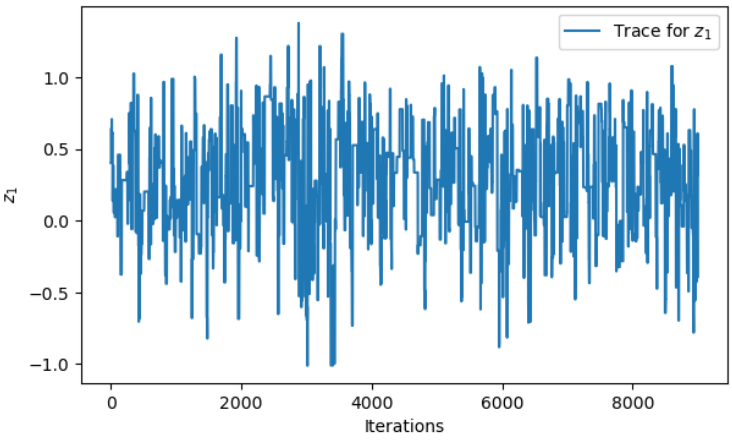
```
fig, ax = plt.subplots(1, 1, figsize=(10, 6))
ax.plot(thetas[:, 0], thetas[:, 1], 'g.')
grid.plot_contours(ax, f=np.exp, alpha=0.5)
ax.set(xlabel='$z_1$', ylabel='$z_2$', title='Contours of posterior density');
```

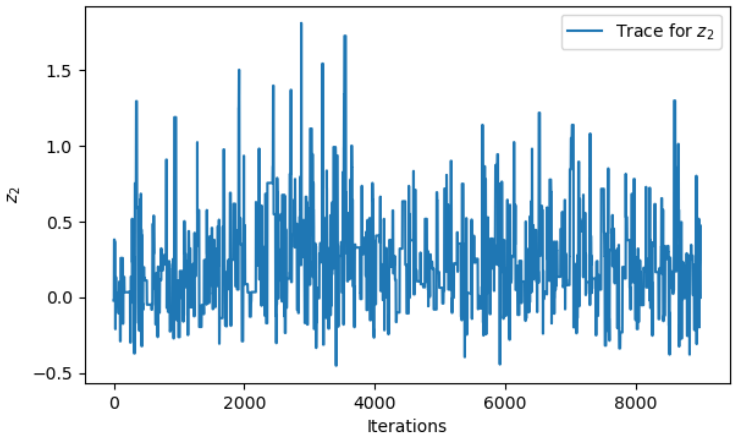



Next, we plot the traces:

```
fig, ax = plt.subplots(1, 2, figsize=(15, 4))
ax[0].plot(thetas[:, 0], label='Trace for $z_1$')
ax[1].plot(thetas[:, 1], label='Trace for $z_2$')
ax[0].set(xlabel='Iterations', ylabel='$z_1$')
ax[1].set(xlabel='Iterations', ylabel='$z_2$')
ax[0].legend()
ax[1].legend()
```

```
<matplotlib.legend.Legend at 0x7f5a4966adf0>
```




# Question 5.3

The posterior mean of $\sin(z_1 z_2)$ can be estimated using Monte Carlo as follows

$$\mathbb{E}\left[\sin(z_1 z_2)|\mathcal{D}\right] \approx \frac{1}{S}\sum_{i=1}^{S} \sin(z_1^{(i)} z_2^{(i)}) \approx 0.13,$$

where $z_1^{(i)}, z_2^{(i)} \sim p(z_1, z_2|\mathcal{D})$ are posterior samples and $S$ is the number of samples.

```
z1, z2 = thetas[:, 0], thetas[:, 1]
mean_sin = np.mean(np.sin(z1*z2), 0)
print(f'E[sin(z1z2|D)] = {mean_sin:3.2f}')
```

```
E[sin(z1z2|D)] = 0.13
```

And if you used samples from the multivariate normal, then the result would be

```
z_normal = np.random.multivariate_normal(np.array([1, 2]), np.array([[1, 0.3], [0.3, 1]]), size=10**4)
mean_sin = np.mean(np.sin(z_normal[:, 0]*z_normal[:, 1]), 0)
print(f'E[sin(z1z2|D)] = {mean_sin:3.2f} (using multivariate normal samples)')
```

```
E[sin(z1z2|D)] = 0.15 (using multivariate normal samples)
```

# Question 5.4

The posterior probability of $z_1 > z_2$ given $\mathcal{D}$ can be estimated using Monte Carlo samples as follows

$$p(z_1 > z_2|\mathcal{D}) \approx \frac{1}{S}\sum_{i=1}^{S}\mathbb{I}\left[z_1^{(i)} > z_2^{(i)}\right] \approx 0.68$$

```
post_prob = np.mean(z1 > z2)
print(f'p[z1 > z2|D)] = {post_prob:3.2f}')
```

```
p[z1 > z2|D)] = 0.68
```

And if you used samples from the multivariate normal, then the result would be

```
post_prob = np.mean(z_normal[:, 0] > z_normal[:, 1])
print(f'p[z1 > z2|D)] = {post_prob:3.2f}')
```

```
p[z1 > z2|D)] = 0.20
```

---