Student: Quoc Tuan Vinh, Ngo
ID: 704526

CS-E4820 – Machine Learning: Advanced Probabilistic Methods
Homework Assignment 6

Problem 1. *"Deriving VB for a simple model, part 1"*

From the problem, we already have that:

$$\tau \sim Beta(\alpha_0, \alpha_0), thus\ \log p(\tau) = \log \frac{\tau^{\alpha_0 - 1}(1 - \tau)^{\alpha_0 - 1}}{const}$$

and:

$$p(\mathbf{z}|\tau) = \prod_{n=1}^{N} \tau^{z_{n2}}(1-\tau)^{z_{n1}}$$

Thus:

$$\begin{aligned}
\log q^*(\tau) = E_{z,\theta}(\log p(x, \mathbf{z}, \tau, \theta)) &= \log p(\tau) + E_z(\log p(\mathbf{z}|\tau)) + const \\
&= \log \frac{\tau^{\alpha_0 - 1}(1 - \tau)^{\alpha_0 - 1}}{const} + E_z\left(\log \prod_{n=1}^{N} \tau^{z_{n2}}(1-\tau)^{z_{n1}}\right) + const \\
&= (\alpha_0 - 1)\log(\tau) + (\alpha_0 - 1)\log(1-\tau) + \log(\tau)\sum_{n=1}^{N} E_z(z_{n2}) \\
&+ \log(1-\tau)\sum_{n=1}^{N} E_z(z_{n1}) + const \\
&= \log(\tau)\left((\alpha_0 - 1) + \sum_{n=1}^{N} E_z(z_{n2})\right) \\
&+ \log(1-\tau)\left((\alpha_0 - 1) + \sum_{n=1}^{N} E_z(z_{n1})\right) + const \\
&= \log(\tau)\left(\alpha_0 - 1 + \sum_{n=1}^{N} r_{n2}\right) + \log(1-\tau)\left(\alpha_0 - 1 + \sum_{n=1}^{N} r_{n1}\right) + const \\
&= \log(\tau)(\alpha_0 - 1 + N_2) + \log(1-\tau)(\alpha_0 - 1 + N_1) + const
\end{aligned}$$

$$where\ E_z(z_{nk}) = r_{nk}\ , and\ N_k = \sum_{n=1}^{N} r_{nk}$$

We exponentiate and recognize the exponentiated form as $q^*(\tau) = Beta(\tau\,|N_2 + \alpha_0,\ N_1 + \alpha_0)$

Problem 2. *"Deriving VB for a simple model, part 2"*

From the problem, we already have that:

$$\theta \sim N(0, \beta_0^{-1}), thus \log p(\theta) = \log(\frac{\beta_0^{\frac{1}{2}}}{\sqrt{2\pi}} \times e^{-\frac{1}{2}\theta^2\beta_0})$$

and

$$p(x|\boldsymbol{z}, \theta) = \prod_{n=1}^{N} N(x_n|0,1)^{z_{n1}} \, N(x_n|\theta, 1)^{z_{n2}}$$

Thus:

$$\begin{aligned}
\log q^*(\theta) &= E_{z,\tau}(\log p(x, \boldsymbol{z}, \tau, \theta)) = \log p(\theta) + E_z(\log p(x|\boldsymbol{z}, \theta)) + const \\
&= \log(\frac{\beta_0^{\frac{1}{2}}}{\sqrt{2\pi}} \times e^{-\frac{1}{2}\theta^2\beta_0}) + E_z\left(\log \prod_{n=1}^{N} N(x_n|0,1)^{z_{n1}} \, N(x_n|\theta, 1)^{z_{n2}}\right) \\
&+ const \\
&= -\frac{1}{2}\,\theta^2\beta_0 + \sum_{n=1}^{N} E_z(z_{n1}) \log N(x_n|0,1) + \sum_{n=1}^{N} E_z(z_{n2}) \log N(x_n|\theta, 1) \\
&+ const
\end{aligned}$$

Since $\sum_{n=1}^{N} E_z(z_{n1}) \log N(x_n|0,1)$ does not contain θ, we can write as:

$$\begin{aligned}
\log q^*(\theta) &= -\frac{1}{2}\,\theta^2\beta_0 + \sum_{n=1}^{N} E_z(z_{n2}) \log N(x_n|\theta, 1) + const \\
&= -\frac{1}{2}\,\theta^2\beta_0 + \sum_{n=1}^{N}\left(E_z(z_{n2})\left(-\frac{1}{2}\log(2\pi) - \frac{1}{2}(x_n - \theta)^2\right)\right) + const \\
&= -\frac{1}{2}\,\theta^2\beta_0 + \sum_{n=1}^{N}\left(E_z(z_{n2})\left(-\frac{1}{2}(x_n^2 - 2x_n\theta + \theta^2)\right)\right) + const \\
&= -\frac{1}{2}\left[\theta\left(\beta_0 + \sum_{n=1}^{N} E_z(z_{n2})\right)\theta\right] + \theta\sum_{n=1}^{N} E_z(z_{n2})\, x_n + const
\end{aligned}$$

Applying completing the square method, we have:

$$\begin{aligned}
\log q^*(\theta) &= -\frac{1}{2}\left[\theta - \beta_2^{-1} \sum_{n=1}^{N} E_z(z_{n2})x_n\right]^2 \times \beta_2 + const \\
&= -\frac{1}{2}[\theta - m_2]^2 \times \beta_2 + const
\end{aligned}$$

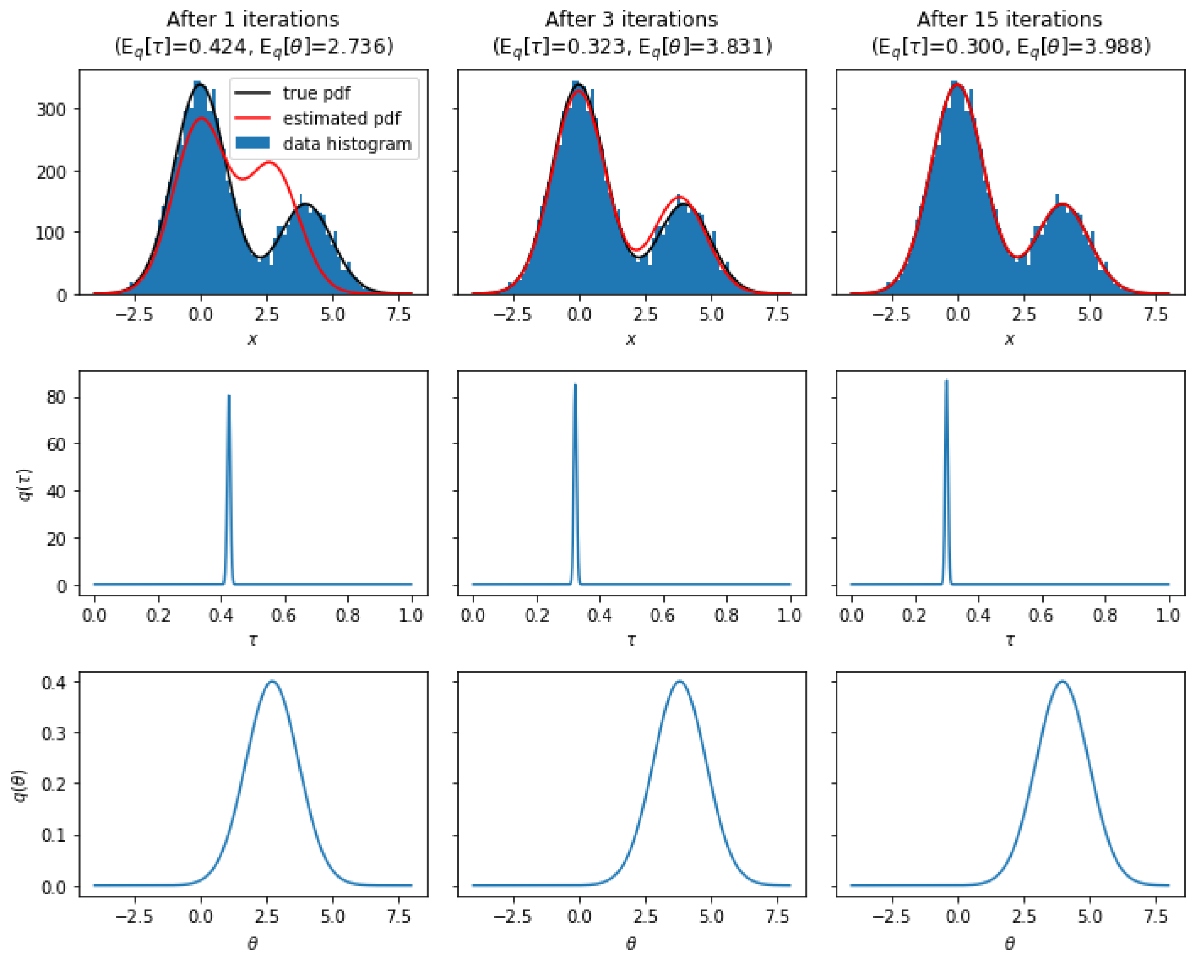
where $\beta_2 = \beta_0 + \sum_{n=1}^{N} E_z(z_{n2}) = \beta_0 + N_2$

and $m_2 = \beta_2^{-1} \sum_{n=1}^{N} E_z(z_{n2})x_n = \beta_2^{-1} \sum_{n=1}^{N} r_{n2}x_n = \beta_2^{-1} \sum_{n=1}^{N} r_{n2}x_n = \beta_2^{-1} N_2 \overline{x_2}$

and $\overline{x_2} = \frac{1}{N_2} \sum_{n=1}^{N} r_{n2}x_n$

We exponentiate and recognize the exponentiated form as $q^*(\theta) = N(\theta|m_2, \beta_2^{-1})$

Plotting for both question 1 and 2:




```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import norm, beta

np.random.seed(123123123)

# Simulate data
theta_true = 4
tau_true = 0.3
n_samples = 10000
z = (np.random.rand(n_samples) < tau_true)  # True with probability tau_true
x = np.random.randn(n_samples) + z * theta_true

# Parameters of the prior distributions.
alpha0 = 0.5
beta0 = 0.2

# The number of iterations
n_iter = 15

# Some initial value for the things that will be updated
E_log_tau = -0.7   # E(log(tau))
E_log_tau_c = -0.7  # E(log(1-tau))
E_log_var = 4 * np.ones(n_samples)  # E((x_n-theta)^2)
r2 = 0.5 * np.ones(n_samples)  # Responsibilities of the second cluster.

# init the plot
iters_to_plot = [0, 2, 14]
```

```
fig, ax = plt.subplots(3, len(iters_to_plot), figsize=(10, 8), sharex='row', sharey='row')
col = 0 # plot column

for i in range(n_iter):

    # Updated of responsibilites, factor q(z)
    log_rho1 = E_log_tau_c - 0.5 * np.log(2 * np.pi) - 0.5 * (x ** 2)
    log_rho2 = E_log_tau - 0.5 * np.log(2 * np.pi) - 0.5 * E_log_var
    max_log_rho = np.maximum(log_rho1, log_rho2) # Normalize to avoid numerical problems when
exponentiating.
    rho1 = np.exp(log_rho1 - max_log_rho)
    rho2 = np.exp(log_rho2 - max_log_rho)
    r2 = rho2 / (rho1 + rho2)
    r1 = 1 - r2

    N1 = np.sum(r1)
    N2 = np.sum(r2)

    # Update of factor q(tau)
    from scipy.special import psi # digamma function
    E_log_tau = psi(N2 + alpha0) - psi(N1 + N2 + 2*alpha0)# EXERCISE
    E_log_tau_c = psi(N1 + alpha0) - psi(N1 + N2 + 2*alpha0)# EXERCISE

    # Current estimate of tau
    tau_est = (N2 + alpha0) / (N1 + N2 + 2*alpha0)# EXERCISE: mean of q(tau)

    # Update of factor q(theta)
    beta2 = beta0 + N2
    x2_ = 1 / N2 * np.sum(r2*x)
    m2 = 1/beta2 * N2 * x2_
    E_log_var = (x - m2)**2 + 1/beta2# EXERCISE

    # Current estimate theta
    theta_est = m2# EXERCISE: mean of q(theta)

    # plotting
    if i in iters_to_plot:
        # plot estimated data distribution
        xgrid = np.linspace(-4, 8, 100)
        ax[0,col].hist(x, xgrid, label="data histogram")
        pdf_true = (1-tau_true) * norm.pdf(xgrid, 0, 1) + tau_true * norm.pdf(xgrid, theta_true, 1)
        pdf_est = (1-tau_est) * norm.pdf(xgrid, 0, 1) + tau_est * norm.pdf(xgrid, theta_est, 1)
        ax[0,col].plot(xgrid, pdf_true * n_samples * (xgrid[1]-xgrid[0]), 'k', label="true pdf")
        ax[0,col].plot(xgrid, pdf_est * n_samples * (xgrid[1]-xgrid[0]), 'r', label="estimated pdf")
        if i == 0:
            ax[0,i].legend()
        ax[0,col].set_title(("After %d iterations\n" +
                             "($\\mathrm{E}_q[\\tau]$=%.3f, $\\mathrm{E}_q[\\theta]$=%.3f)") %
                             (i + 1, tau_est, theta_est))
        ax[0,col].set_xlabel("$x$")

        # plot marginal distribution of tau
        tau = np.linspace(0, 1.0, 1000)
        q_tau = beta.pdf(tau, N2 + alpha0, N1 + alpha0)
        ax[1,col].plot(tau, q_tau)
        ax[1,col].set_xlabel("$\\tau$")

        # plot marginal distribution of theta
        theta = np.linspace(-4.0, 8.0, 1000)
        q_theta = norm.pdf(theta, m2, 1.0)
        ax[2,col].plot(theta, q_theta)
```

```
        ax[2,col].set_xlabel("$\\theta$")
        col = col + 1

# finalize the plot
ax[1,0].set_ylabel("$q(\\tau)$")
ax[2,0].set_ylabel("$q(\\theta)$")
plt.tight_layout()
plt.show()
```

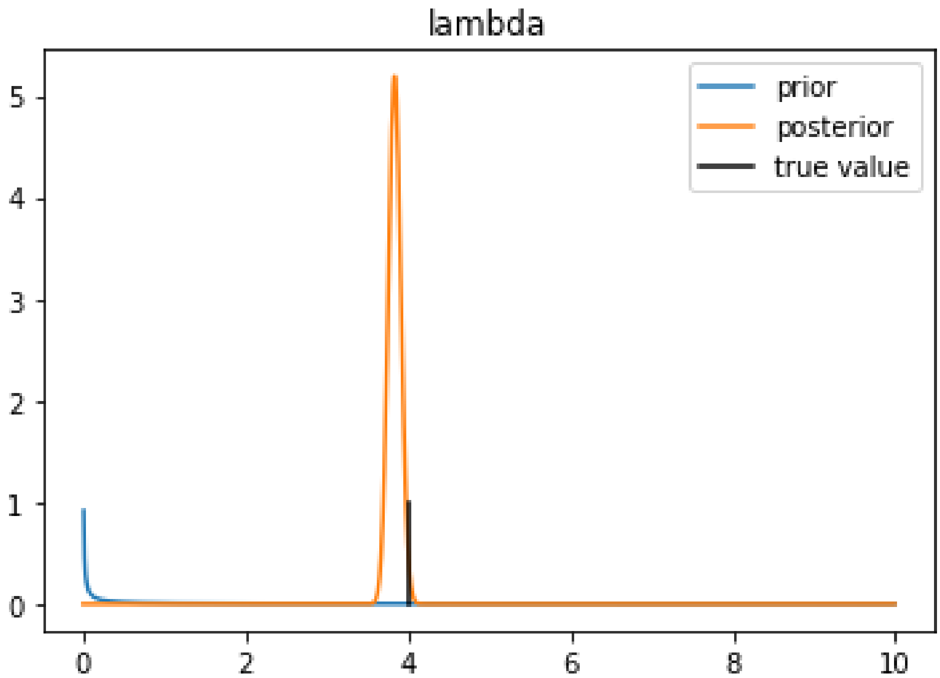
Problem 3. *"KL Divergence"*

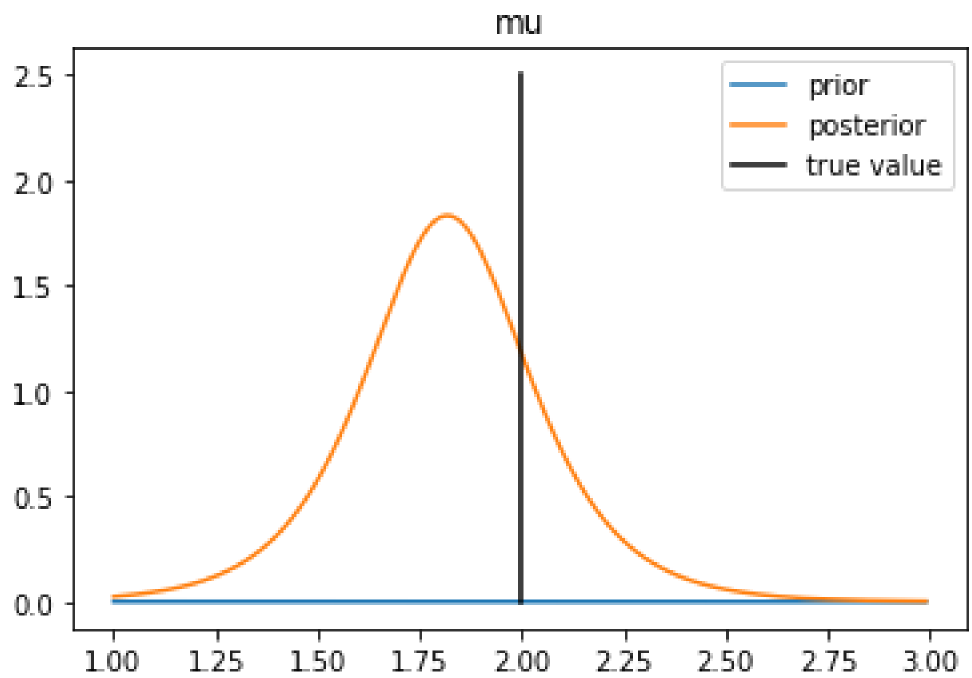
In this case, we can calculate KL-Divergence either by numpy sum or numpy integration function trapz. Both methods are applied and result in similar outcomes.

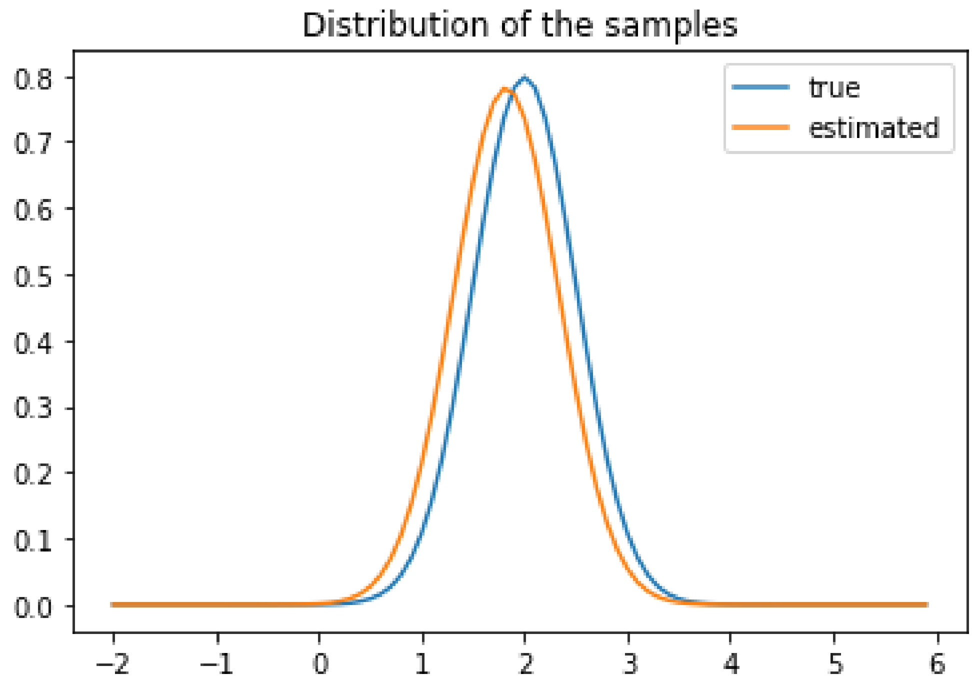
Since the question does not specify explicitly whether $KL_{p||q}$ or $KL_{q||p}$, so both approaches are calculated. Applying the formula:

$$KL_{q||p} = -\int q(\boldsymbol{Z}) \ln\left\{\frac{p(\boldsymbol{Z})}{q(\boldsymbol{Z})}\right\} d\boldsymbol{Z} \quad \& \quad KL_{p||q} = -\int p(\boldsymbol{Z}) \ln\left\{\frac{q(\boldsymbol{Z})}{p(\boldsymbol{Z})}\right\} d\boldsymbol{Z}$$

The results are shown as below.

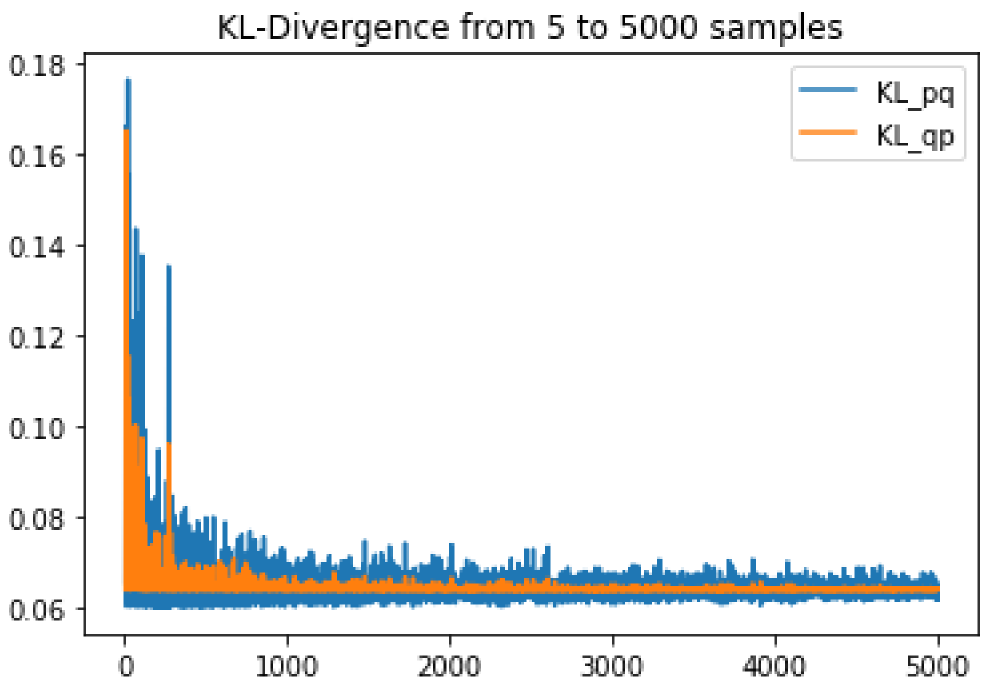







```
Samples of 5:
Use Sum:
KL_pq:  0.0615466323942
KL_qp:  0.0641509202025
Use Numpy Integration Trapz:
KL_pq:  0.0615466323942
KL_qp:  0.0641509202024

Range from 5 to 5000:
```




```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import gamma, norm

# Initialize the random number generator
np.random.seed(24)

# SIMULATE THE TRUE DATA SET
num_samples = 5
mu_true = 2       # mean of the true distribution
lambda_true = 4   # precision of the true distribution
sigma_true = 1 / np.sqrt(lambda_true)    # standard deviation
lambda_range = np.arange(0, 10, 0.01)
full_dist_range = np.arange(-2, 6, 0.1)
a0 = 0.01
b0 = 0.01
mu0 = 0
beta0 = 0.001
data_set = np.random.normal(mu_true, sigma_true, num_samples)
sample_mean = np.mean(data_set)
sample_var = np.var(data_set)
# PLOT THE PRIOR AND THE POSTERIOR DISTRIBUTIONS

# Plot distribution of lambda, the precision
lambda_range = np.arange(0, 10, 0.01)
prior_lambda_pdf = gamma.pdf(lambda_range, a0, scale=1/b0)
posterior_lambda_pdf = gamma.pdf(lambda_range, a_n, scale=1/b_n)

plt.plot(lambda_range, prior_lambda_pdf, label="prior")
plt.plot(lambda_range, posterior_lambda_pdf, label="posterior")
plt.plot([lambda_true,lambda_true], [0,1], "k-", label="true value")
plt.title('lambda')
plt.legend()
plt.show()

plt.plot(mu_range, prior_mu_pdf, label="prior")
```

```
plt.plot(mu_range, posterior_mu_pdf, label="posterior")
plt.plot([mu_true,mu_true],[0,2.5], "k-", label="true value")
plt.title('mu')
plt.legend()
plt.show()


# PLOT THE TRUE AND ESTIMATED DISTRIBUTIONS OF THE SAMPLES

# We estimate the parameters with the mean of the posterior distribution
mu_hat = np.sum(posterior_mu_pdf * mu_range) / np.sum(posterior_mu_pdf)
lambda_hat = np.sum(posterior_lambda_pdf * lambda_range) / np.sum(posterior_lambda_pdf)

full_dist_range = np.arange(-2, 6, 0.1)
true_pdf = norm.pdf(full_dist_range, mu_true, sigma_true)
estimated_pdf = norm.pdf(full_dist_range, mu_hat, 1 / np.sqrt(lambda_hat))

plt.plot(full_dist_range, true_pdf, label="true")
plt.plot(full_dist_range, estimated_pdf, label="estimated")
plt.title('Distribution of the samples')
plt.legend()
plt.show()

print('Samples of 5:')
print('Use Sum:')
KL_pq = np.sum(true_pdf * np.log(true_pdf/estimated_pdf)) /np.sum(true_pdf)
KL_qp = np.sum(estimated_pdf * np.log(estimated_pdf/true_pdf))/np.sum(estimated_pdf)

print('KL_pq: ',KL_pq)
print('KL_qp: ',KL_qp)

print('Use Numpy Integration Trapz:')
KL_pq2 = np.trapz( y= (true_pdf * np.log(true_pdf/estimated_pdf)), x = full_dist_range)
KL_qp2 = np.trapz( y= (estimated_pdf * np.log(estimated_pdf/true_pdf)), x = full_dist_range)

print('KL_pq: ',KL_pq2)
print('KL_qp: ',KL_qp2)

##########################################################################

print('\nRange from 5 to 5000:')
KL_pq_list = []
KL_qp_list = []
for i in range(5,5000,1):
   num_samples = i
   data_set = np.random.normal(mu_true, sigma_true, num_samples)

   sample_mean = np.mean(data_set)
   sample_var = np.var(data_set)

   mu_n = (mu0 * beta0 + num_samples * sample_mean) / (beta0 + num_samples)
   beta_n = beta0 + num_samples
   a_n = a0 + num_samples / 2
   b_n = b0 + (num_samples * sample_var + (beta0 * num_samples * (sample_mean - mu0) ** 2)
        / (beta0 + num_samples)) / 2

   # Plot distribution of lambda, the precision
   prior_lambda_pdf = gamma.pdf(lambda_range, a0, scale=1/b0)
   posterior_lambda_pdf = gamma.pdf(lambda_range, a_n, scale=1/b_n)
   mu_hat = np.sum(posterior_mu_pdf * mu_range) / np.sum(posterior_mu_pdf)
   lambda_hat = np.sum(posterior_lambda_pdf * lambda_range) / np.sum(posterior_lambda_pdf)
```

```
    true_pdf = norm.pdf(full_dist_range, mu_true, sigma_true)
    estimated_pdf = norm.pdf(full_dist_range, mu_hat, 1 / np.sqrt(lambda_hat))

    KL_pq = np.sum(true_pdf * np.log(true_pdf/estimated_pdf)) /np.sum(true_pdf)
    KL_pq_list.append(KL_pq)

    KL_qp = np.sum(estimated_pdf * np.log(estimated_pdf/true_pdf))/np.sum(estimated_pdf)
    KL_qp_list.append(KL_qp)


plt.title("KL-Divergence from 5 to 5000 samples")
plt.plot(range(5,5000,1),KL_pq_list, label="KL_pq")
plt.plot(range(5,5000,1),KL_qp_list, label="KL_qp")
plt.legend()
plt.show()
```

Problem 4. *"Variational approximation for a simple distribution"*

From the probabilities from the question, we can calculate the true joint distribution of those variables as below:

$p(x_1, x_2)$	$x_1 = 0$	$x_1 = 1$
$x_2 = 0$	(0.4 x 0.5) = 0.2	(0.6 x 0.9) = 0.54
$x_2 = 1$	(0.4 x 0.5) = 0.2	(0.6 x 0.1) = 0.06

Nevertheless, we need to estimate $q(x_1, x_2) = q_1(x_1)\, q_2(x_1)$ that best approximates the joint $p(x_1, x_2)$. Since x_1, x_2 are both binary random variables, we can use Bernoulli distribution to derive the KL-Divergence for them.

To do so, let us assume $q_1(x = 1) = a;\ q_2(x = 1) = b$. Hence, we can have $q_1(x = 0) = 1 - a;\ q_2(x = 0) = 1 - b$.

Thus far, we can calculate a similar table as above for $q(x_1, x_2)$:

$q(x_1, x_2)$	$x_1 = 0$	$x_1 = 1$
$x_2 = 0$	(1 – a)(1 – b)	a(1 – b)
$x_2 = 1$	(1 – a)b	ab

Apply the formula for $KL_{p||q}$:

$$KL_{p||q} = -\int p(\boldsymbol{Z}) \log\left\{\frac{q(\boldsymbol{Z})}{p(\boldsymbol{Z})}\right\} d\boldsymbol{Z}$$

With p(Z) = $p(x_1, x_2)$ and q(Z) = $q(x_1, x_2)$, and x_1, x_2 are discreet, we can have:

$$KL_{p||q} = -\left(\sum_Z p(\boldsymbol{Z}) \ln\left\{\frac{q(\boldsymbol{Z})}{p(\boldsymbol{Z})}\right\}\right) = -\left(0.2 \log\frac{(1-a)(1-b)}{0.2} + 0.54 \log\frac{a(1-b)}{0.54} + 0.2 \log\frac{(1-a)b}{0.2} + 0.06 \log\frac{ab}{0.06}\right)$$