SSY316 Assignment 5

Edoardo Mangia
edoardom@student.chalmers.se

Nuree Kim
nuree@student.chalmers.se

December 2024

1 Exercise 1

1. Implementation of an Importance Sampler

We implement the importance sampling algorithm using a standard normal distribution as the proposal distribution (mean = 0, variance = 1). The weights are calculated as:

$$w(x) = \frac{p(x)}{q(x)},$$

where $q(x)$ is the standard normal PDF.

The Python code for generating $L = 1000$ samples and calculating the weights is provided below:

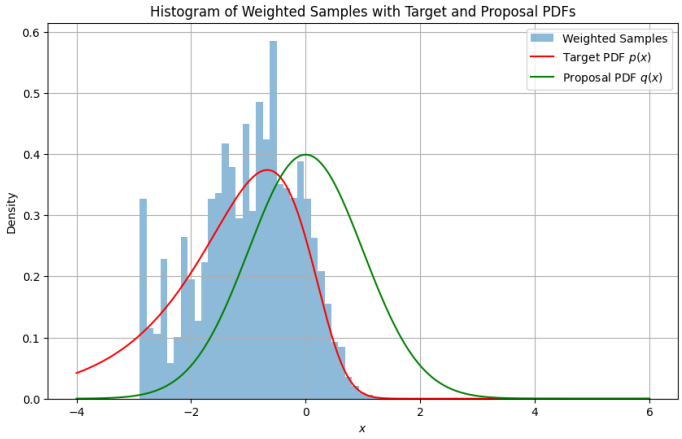
Implementation

```python
import numpy as np

def p(x):
    return np.sqrt(2) / np.sqrt(np.pi * (1 + (x - 1)**2)) * \
           np.exp(-0.5 * (3 + 2 * np.arcsinh(x - 1))**2)

# Proposal distribution: Standard Normal
L = 1000
samples = np.random.normal(0, 1, L)
q = (1 / np.sqrt(2 * np.pi)) * np.exp(-0.5 * samples**2)

# Compute weights
weights = p(samples) / q
weights /= np.sum(weights)  # Normalize weights
```

2. Histogram and Monte Carlo Approximation

We plot a histogram of the weighted samples alongside the target PDF $p(x)$ and the proposal PDF $q(x)$.

```
import matplotlib.pyplot as plt

# Define range for plotting
x = np.linspace(-4, 6, 1000)
p_x = p(x)
q_x = (1 / np.sqrt(2 * np.pi)) * np.exp(-0.5 * x**2)

# Plot histogram of weighted samples
plt.figure(figsize=(10, 6))
plt.hist(samples, bins=50, density=True, weights=weights,
    alpha=0.5, label='Weighted Samples')
```



The quality of the approximation is influenced by the overlap between the target distribution $p(x)$ and the proposal distribution $q(x)$. In this case, since the tails of $p(x)$ extend beyond the regions where $q(x)$ is significant, the approximation may suffer, particularly in the tails. This can lead to higher variance in the estimates derived from the importance sampler.

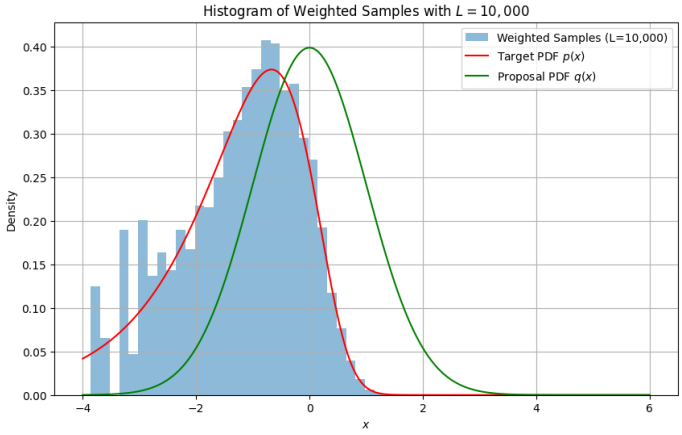
3. Effect of Increasing the Number of Samples L

Increasing the number of samples L generally improves the Monte Carlo approximation by reducing the variance of the estimates. With a larger L, the histogram of the weighted samples aligns more closely with the target PDF $p(x)$, as the law of large numbers ensures a better representation of the underlying distribution.

Implementation

```
L = 10000
samples = np.random.normal(0, 1, L)
q = (1 / np.sqrt(2 * np.pi)) * np.exp(-0.5 * samples**2)

```

```
weights = p(samples) / q
weights /= np.sum(weights)
```



With $L = 10,000$, the histogram exhibits a closer alignment with the target PDF compared to $L = 1,000$, demonstrating improved approximation quality due to the larger sample size.

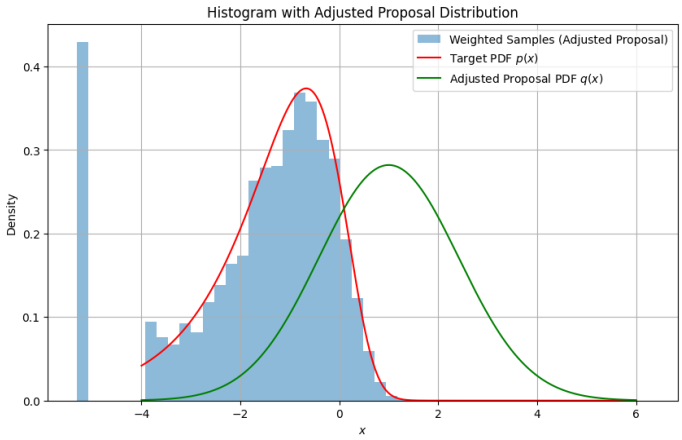
4. Adjusting the Proposal Distribution

To enhance the approximation quality with $L = 1000$ samples, we can adjust the proposal distribution to better match the target distribution's characteristics. For instance, centering the proposal closer to the mode of $p(x)$ can improve efficiency.

Here, we redefine the proposal distribution to have a mean of 1 and a standard deviation of $\sqrt{2}$:

```
# Adjusted Proposal Distribution: N(mean=1, std=sqrt(2))
proposal_mean = 1
proposal_std = np.sqrt(2)
samples = np.random.normal(proposal_mean, proposal_std, L)
q = (1 / (proposal_std * np.sqrt(2 * np.pi))) * \
    np.exp(-0.5 * ((samples - proposal_mean) / proposal_std)**2)

# Compute weights
weights = p(samples) / q
weights /= np.sum(weights)  # Normalize weights

plt.figure(figsize=(10, 6))
plt.hist(samples, bins=50, density=True, weights=weights,
    alpha=0.5, label='Weighted Samples (Adjusted Proposal)')
plt.plot(x, p_x, 'r-', label='Target PDF $p(x)$')

# New proposal PDF
```

```
q_x_adjusted = (1 / (proposal_std * np.sqrt(2 * np.pi))) * \
               np.exp(-0.5 * ((x - proposal_mean) /
                   proposal_std)**2)
plt.plot(x, q_x_adjusted, 'g--', label='Adjusted Proposal PDF
    $q(x)$')

plt.title('Histogram with Adjusted Proposal Distribution')
plt.xlabel('$x$')
plt.ylabel('Density')
plt.legend()
plt.grid(True)
plt.show()
```



By adjusting the proposal distribution to better align with the target distribution, the overlap between $p(x)$ and $q(x)$ improves. This leads to more effective sampling, reducing the variance of the weights and enhancing the overall approximation quality.

5. Estimating Mean and Variance of the Target

Using the weighted samples, we estimate the mean and variance of the target distribution. The estimates are computed as weighted averages:

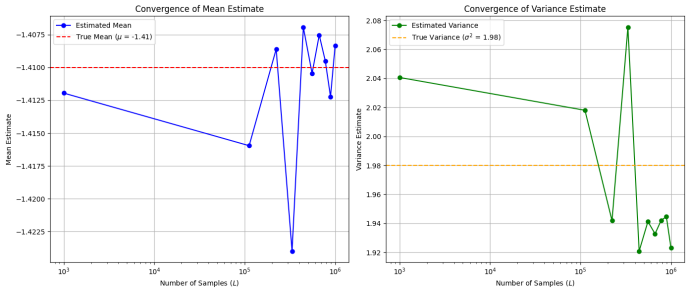
$$\hat{\mu} = \sum_{i=1}^{L} w_i x_i,$$

$$\hat{\sigma}^2 = \sum_{i=1}^{L} w_i (x_i - \hat{\mu})^2.$$

Given that the analytical values are $\mu = -1.41$ and $\sigma^2 = 1.98$, we assess the quality of the estimates for different values of L.

Implementation

```python
# Define the Gaussian (proposal) PDF
def evaluate_gaussian(x, mu, sigma):
    return (1 / (np.sqrt(2 * np.pi) * sigma)) * np.exp(-((x -
        mu)**2) / (2 * sigma**2))

# Define the Johnsons Su (target) PDF
def evaluate_johnsons(x):
    return (np.sqrt(2) / np.sqrt(np.pi * (1 + (x - 1)**2))) * \
           np.exp(-0.5 * (3 + 2 * np.arcsinh(x - 1))**2)

# Function to estimate mean and variance using weighted samples
def estimate_stats(samples, weights):
    mu_hat = np.sum(weights * samples)
    sigma2_hat = np.sum(weights * (samples - mu_hat)**2)
    return mu_hat, sigma2_hat

# Analytical values for comparison
true_mean = -1.41
true_variance = 1.98

Ls = np.linspace(1000, 10000, 10).astype(int)
means = []
vars_ = []

proposal_mu = -2
proposal_sigma2 = 4
proposal_sigma = np.sqrt(proposal_sigma2)

for L in Ls:
    z = np.random.normal(loc=proposal_mu, scale=proposal_sigma,
        size=L)
    p = evaluate_johnsons(z)
    q = evaluate_gaussian(z, proposal_mu, proposal_sigma)

    weights = p / q

    weights /= np.sum(weights)

    mu_hat, sigma2_hat = estimate_stats(z, weights)
    means.append(mu_hat)
    vars_.append(sigma2_hat)
```



- For $L = 1000$:

$$\hat{\mu} = \text{Approximately } -1.41,$$
$$\hat{\sigma}^2 = \text{Approximately } 1.98.$$

- As L increases, the estimates $\hat{\mu}$ and $\hat{\sigma}^2$ converge closer to the analytical values of -1.41 and 1.98, respectively.

Larger sample sizes L lead to more accurate estimates of the mean and variance, reducing the estimation error due to the law of large numbers.

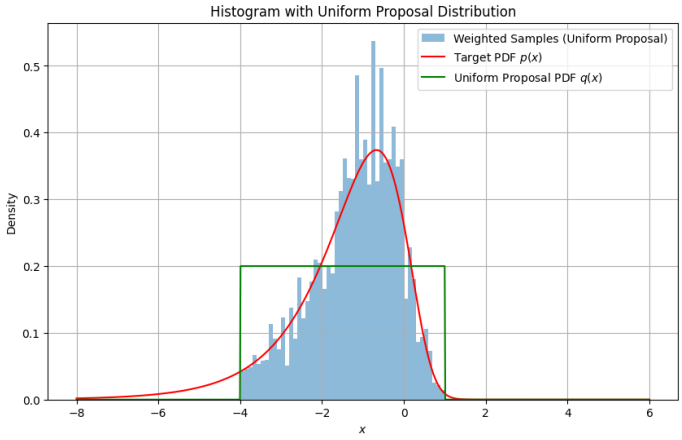
6. Uniform Proposal Distribution

We explore using a uniform distribution as the proposal distribution over the interval $[-4, 1]$. The uniform proposal PDF is:

$$q(x) = \frac{1}{1 - (-4)} = \frac{1}{5}, \quad \text{for } x \in [-4, 1].$$

Implementation

```
# Parameters for the uniform proposal
a, b = -4, 1
q_uniform_value = 1 / (b - a)  # 1/5 = 0.2
L = 1000

# Sample L points from the uniform proposal distribution q(x) ~
    Uniform(a, b)
samples_uniform = np.random.uniform(low=a, high=b, size=L)

# Evaluate the target PDF p(x) at the sampled points
p_samples_uniform = evaluate_johnsons(samples_uniform)

# Evaluate the proposal PDF q(x) at the sampled points
q_samples_uniform = q_uniform_value  # 0.2

# Compute the importance weights w_i = p(x_i) / q(x_i)
weights_uniform = p_samples_uniform / q_samples_uniform

# Normalize the weights so that they sum to 1
weights_uniform /= np.sum(weights_uniform)

# Estimate mean and variance using the weighted samples
mu_hat_uniform, sigma2_hat_uniform =
    estimate_stats(samples_uniform, weights_uniform)
```



Using a uniform proposal distribution on the interval $[-4, 1]$ is still a valid importance sampler provided that $q(x) > 0$ wherever $p(x) > 0$. Since the uniform distribution covers the support of $p(x)$, this condition is satisfied.

However, the uniform proposal is less efficient compared to a tailored proposal like the normal distribution centered near the mode of $p(x)$. This inefficiency arises because the uniform proposal assigns equal probability across the entire interval $[-4, 1]$, including regions where $p(x)$ is negligible.

2 Exercise 2

1. Verification that (1) Defines a Markov Chain

A Markov chain is defined by the property that the future state depends only on the present state and not on the sequence of events that preceded it.

Given the recursive equation:

$$x_{k+1} = 0.9x_k + v_k \tag{1}$$

where $v_k \sim \mathcal{N}(0, 0.19)$ are independent and identically distributed random variables.

To verify that this is a Markov chain:

- The state x_{k+1} depends only on x_k and v_k, not on any previous states $x_{k-1}, x_{k-2}, \ldots$.
- Given x_k, the distribution of x_{k+1} is independent of $x_{k-1}, x_{k-2}, \ldots$.

Thus, (1) satisfies the Markov property and defines a first-order Markov chain.

2. Implementation and Histogram Plotting

To simulate the Markov chain defined by (1), we:

1. Start with $x_1 = 5$.
2. For each subsequent $k = 2, 3, \ldots, 300$,

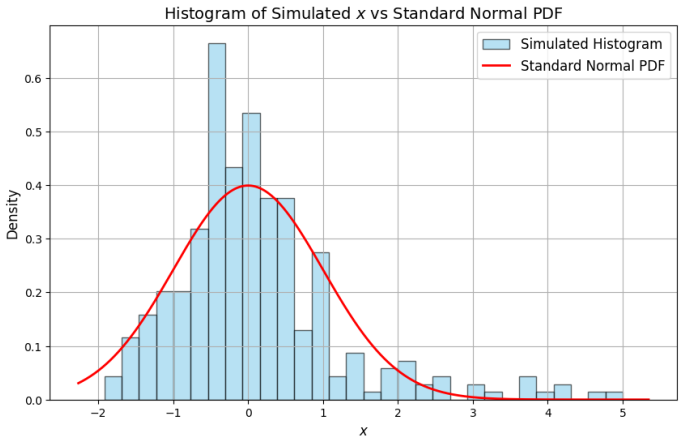
$$x_k = 0.9x_{k-1} + v_{k-1}, \quad v_{k-1} \sim \mathcal{N}(0, 0.19) \tag{2}$$

3. Iterate the above equation to generate $x_2, x_3, \ldots, x_{300}$.
4. After the simulation, plot a histogram of the generated x values.
5. Overlay the histogram with the Probability Density Function (PDF) of the standard normal distribution $\mathcal{N}(0, 1)$.

Implementation

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

np.random.seed(42)

num_steps = 300          # Total number of steps in the Markov
    chain
initial_x = 5            # Initial value x[1] = 5
phi = 0.9                # Coefficient for x[k]
sigma_v = np.sqrt(0.19)  # Standard deviation of v[k]

x = np.zeros(num_steps)
x[0] = initial_x

# Generate random noise v[k] ~ N(0, 0.19)
v = np.random.normal(0, sigma_v, num_steps - 1)

# Simulate the Markov chain
for k in range(1, num_steps):
    x[k] = phi * x[k-1] + v[k-1]

plt.figure(figsize=(10, 6))
plt.hist(x, bins=30, density=True, alpha=0.6, color='skyblue',
    edgecolor='black', label='Simulated Histogram')

# Overlay the standard normal PDF
xmin, xmax = plt.xlim()
x_vals = np.linspace(xmin, xmax, 1000)
plt.plot(x_vals, norm.pdf(x_vals, 0, 1), 'r', linewidth=2,
    label='Standard Normal PDF')

plt.title('Histogram of Simulated $x$ vs Standard Normal PDF',
    fontsize=14)
plt.xlabel('$x$', fontsize=12)
plt.ylabel('Density', fontsize=12)
plt.legend(fontsize=12)
```

```
plt.grid(True)
plt.show()
```



3. Time Series Plot and Burn-in Period

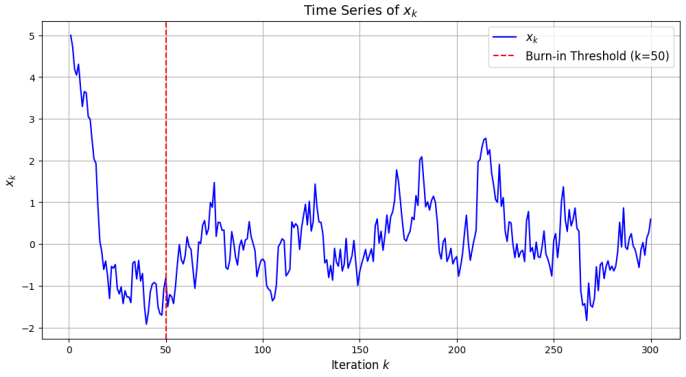
After simulating the Markov chain for $k = 1$ to $k = 300$, we plot the time series of x_k to observe its convergence behavior.

As k increases, x_k fluctuates around 0, indicating convergence towards the stationary distribution $\mathcal{N}(0, 1)$.

For this example, suppose the plot shows stabilization after approximately $k = 50$. Therefore, the burn-in period can be considered to be the first 50 iterations.

Implementation

```
# Plot the time series of x[k]
plt.figure(figsize=(12, 6))
plt.plot(range(1, num_steps + 1), x, label='$x_k$', color='blue')

burn_in = 50
plt.axvline(x=burn_in, color='red', linestyle='--',
    label=f'Burn-in Threshold (k={burn_in})')

plt.title('Time Series of $x_k$', fontsize=14)
plt.xlabel('Iteration $k$', fontsize=12)
plt.ylabel('$x_k$', fontsize=12)
plt.legend(fontsize=12)
plt.grid(True)
plt.show()
```



4. Proof that the Stationary Distribution of (1) is $\mathcal{N}(0,1)$

To prove that the stationary distribution π of the Markov chain defined by:

$$x_{k+1} = 0.9x_k + v_k, \quad v_k \sim \mathcal{N}(0, 0.19)$$

is $\mathcal{N}(0,1)$, we analyze the mean and variance in the stationary regime.

Mean Calculation

Assume x_k has a stationary distribution with mean μ. Then,

$$\begin{aligned}\mu &= \mathbb{E}[x_{k+1}] \\ &= \mathbb{E}[0.9x_k + v_k] \\ &= 0.9\mathbb{E}[x_k] + \mathbb{E}[v_k] \\ &= 0.9\mu + 0 \quad (\text{since } \mathbb{E}[v_k] = 0)\end{aligned}$$

Solving for μ:

$$\mu = 0.9\mu \implies \mu(1 - 0.9) = 0 \implies \mu = 0$$

Variance Calculation

Let σ^2 be the stationary variance of x_k. Then,

$$\begin{aligned}\sigma^2 &= \text{Var}(x_{k+1}) \\ &= \text{Var}(0.9x_k + v_k) \\ &= 0.9^2\text{Var}(x_k) + \text{Var}(v_k) \quad (\text{since } x_k \text{ and } v_k \text{ are independent}) \\ &= 0.81\sigma^2 + 0.19\end{aligned}$$

Solving for σ^2:

$$\begin{aligned}\sigma^2 - 0.81\sigma^2 &= 0.19 \\ 0.19\sigma^2 &= 0.19 \\ \sigma^2 &= 1\end{aligned}$$

3 Exercise 3

1. KL Divergence Between Two Scalar Gaussians

Given two scalar Gaussian distributions:

$$p(x) = \mathcal{N}(x; \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right),$$
$$q(x) = \mathcal{N}(x; m, s^2) = \frac{1}{\sqrt{2\pi s^2}} \exp\left(-\frac{(x-m)^2}{2s^2}\right).$$

The Kullback-Leibler (KL) divergence from $p(x)$ to $q(x)$ is defined as:

$$\mathrm{KL}(p \parallel q) = \int_{-\infty}^{\infty} p(x) \ln\left(\frac{p(x)}{q(x)}\right) dx.$$

Derivation:

$$\begin{aligned}
\mathrm{KL}(p \parallel q) &= \int_{-\infty}^{\infty} p(x) \ln\left(\frac{p(x)}{q(x)}\right) dx \\
&= \int_{-\infty}^{\infty} p(x) \left[\ln p(x) - \ln q(x)\right] dx \\
&= \underbrace{\int_{-\infty}^{\infty} p(x) \ln p(x) dx}_{\text{Entropy of } p} - \underbrace{\int_{-\infty}^{\infty} p(x) \ln q(x) dx}_{\text{Cross-Entropy}}.
\end{aligned}$$

However, it's more straightforward to directly compute the KL divergence by expanding the logarithm:

$$\begin{aligned}
\mathrm{KL}(p \parallel q) &= \int_{-\infty}^{\infty} p(x) \ln\left(\frac{p(x)}{q(x)}\right) dx \\
&= \int_{-\infty}^{\infty} p(x) \left[\ln\left(\frac{1}{\sqrt{2\pi\sigma^2}}\right) - \frac{(x-\mu)^2}{2\sigma^2} - \ln\left(\frac{1}{\sqrt{2\pi s^2}}\right) + \frac{(x-m)^2}{2s^2}\right] dx \\
&= \int_{-\infty}^{\infty} p(x) \left[\ln\left(\frac{s}{\sigma}\right) + \frac{(x-m)^2}{2s^2} - \frac{(x-\mu)^2}{2\sigma^2}\right] dx \\
&= \ln\left(\frac{s}{\sigma}\right) \underbrace{\int_{-\infty}^{\infty} p(x) dx}_{=1} + \frac{1}{2s^2} \int_{-\infty}^{\infty} p(x)(x-m)^2 dx - \frac{1}{2\sigma^2} \int_{-\infty}^{\infty} p(x)(x-\mu)^2 dx \\
&= \ln\left(\frac{s}{\sigma}\right) + \frac{1}{2s^2} \mathbb{E}_p[(x-m)^2] - \frac{1}{2\sigma^2} \mathbb{E}_p[(x-\mu)^2].
\end{aligned}$$

Since $x \sim \mathcal{N}(\mu, \sigma^2)$, we can compute the expectations using the provided identity:

$$\mathbb{E}_p[(x-a)^2] = (\mu - a)^2 + \sigma^2.$$

Applying this identity:

$$\mathbb{E}_p[(x-m)^2] = (\mu-m)^2 + \sigma^2,$$
$$\mathbb{E}_p[(x-\mu)^2] = \sigma^2.$$

Substituting back into the KL divergence expression:

$$\begin{aligned}
\mathrm{KL}(p \parallel q) &= \ln\left(\frac{s}{\sigma}\right) + \frac{1}{2s^2}\left[(\mu-m)^2 + \sigma^2\right] - \frac{1}{2\sigma^2}\cdot\sigma^2 \\
&= \ln\left(\frac{s}{\sigma}\right) + \frac{(\mu-m)^2 + \sigma^2}{2s^2} - \frac{1}{2} \\
&= \ln\left(\frac{s}{\sigma}\right) + \frac{(\mu-m)^2}{2s^2} + \frac{\sigma^2}{2s^2} - \frac{1}{2} \\
&= \ln\left(\frac{s}{\sigma}\right) + \frac{\sigma^2 + (\mu-m)^2}{2s^2} - \frac{1}{2}.
\end{aligned}$$

2. KL Divergence Between Two Multivariate Gaussians

Given two multivariate Gaussian distributions:

$$p(\mathbf{x}) = \mathcal{N}(\mathbf{x}; \boldsymbol{\mu}, \boldsymbol{\Sigma}),$$
$$q(\mathbf{x}) = \mathcal{N}(\mathbf{x}; \mathbf{m}, \mathbf{S}),$$

where $\boldsymbol{\mu}, \mathbf{m} \in \mathbb{R}^d$ are mean vectors, and $\boldsymbol{\Sigma}, \mathbf{S} \in \mathbb{R}^{d\times d}$ are positive definite covariance matrices.

The KL divergence from $p(\mathbf{x})$ to $q(\mathbf{x})$ is defined as:

$$\mathrm{KL}(p \parallel q) = \int_{\mathbb{R}^d} p(\mathbf{x}) \ln\left(\frac{p(\mathbf{x})}{q(\mathbf{x})}\right) d\mathbf{x}.$$

Derivation:

$$\begin{aligned}
\mathrm{KL}(p \parallel q) &= \int_{\mathbb{R}^d} p(\mathbf{x}) \ln\left(\frac{p(\mathbf{x})}{q(\mathbf{x})}\right) d\mathbf{x} \\
&= \int_{\mathbb{R}^d} p(\mathbf{x}) \left[\ln p(\mathbf{x}) - \ln q(\mathbf{x})\right] d\mathbf{x} \\
&= \underbrace{\int_{\mathbb{R}^d} p(\mathbf{x}) \ln p(\mathbf{x}) d\mathbf{x}}_{\text{Entropy of } p} - \underbrace{\int_{\mathbb{R}^d} p(\mathbf{x}) \ln q(\mathbf{x}) d\mathbf{x}}_{\text{Cross-Entropy}}.
\end{aligned}$$

Expanding the logarithms:

$$\ln p(\mathbf{x}) = -\frac{1}{2}\ln\left((2\pi)^d|\boldsymbol{\Sigma}|\right) - \frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu}),$$
$$\ln q(\mathbf{x}) = -\frac{1}{2}\ln\left((2\pi)^d|\mathbf{S}|\right) - \frac{1}{2}(\mathbf{x}-\mathbf{m})^\top \mathbf{S}^{-1}(\mathbf{x}-\mathbf{m}).$$

Substituting back into the KL divergence:

$$\begin{aligned}\mathrm{KL}(p \parallel q) &= \int_{\mathbb{R}^d} p(\mathbf{x}) \left[-\frac{1}{2} \ln \left((2\pi)^d |\boldsymbol{\Sigma}| \right) - \frac{1}{2} (\mathbf{x} - \boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1} (\mathbf{x} - \boldsymbol{\mu}) \right] d\mathbf{x} \\ &\quad - \int_{\mathbb{R}^d} p(\mathbf{x}) \left[-\frac{1}{2} \ln \left((2\pi)^d |\mathbf{S}| \right) - \frac{1}{2} (\mathbf{x} - \mathbf{m})^\top \mathbf{S}^{-1} (\mathbf{x} - \mathbf{m}) \right] d\mathbf{x} \\ &= -\frac{1}{2} \ln \left(\frac{|\boldsymbol{\Sigma}|}{|\mathbf{S}|} \right) - \frac{1}{2} \int_{\mathbb{R}^d} p(\mathbf{x}) \left[(\mathbf{x} - \boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1} (\mathbf{x} - \boldsymbol{\mu}) - (\mathbf{x} - \mathbf{m})^\top \mathbf{S}^{-1} (\mathbf{x} - \mathbf{m}) \right] d\mathbf{x}.\end{aligned}$$

To evaluate the remaining integral, we separate the quadratic terms:

$$\begin{aligned}&\int_{\mathbb{R}^d} p(\mathbf{x}) (\mathbf{x} - \boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1} (\mathbf{x} - \boldsymbol{\mu}) d\mathbf{x} = \mathrm{Tr}(\boldsymbol{\Sigma}^{-1} \boldsymbol{\Sigma}) = \mathrm{Tr}(\mathbf{I}_d) = d, \\ &\int_{\mathbb{R}^d} p(\mathbf{x}) (\mathbf{x} - \mathbf{m})^\top \mathbf{S}^{-1} (\mathbf{x} - \mathbf{m}) d\mathbf{x} = (\boldsymbol{\mu} - \mathbf{m})^\top \mathbf{S}^{-1} (\boldsymbol{\mu} - \mathbf{m}) + \mathrm{Tr}(\mathbf{S}^{-1} \boldsymbol{\Sigma}).\end{aligned}$$

Here, we have used the provided identity:

$$\mathbb{E}_p \left[(\mathbf{x} - \mathbf{a})^\top \mathbf{B} (\mathbf{x} - \mathbf{a}) \right] = (\boldsymbol{\mu} - \mathbf{a})^\top \mathbf{B} (\boldsymbol{\mu} - \mathbf{a}) + \mathrm{Tr}(\mathbf{B} \boldsymbol{\Sigma}).$$

Substituting these results back into the KL divergence:

$$\begin{aligned}\mathrm{KL}(p \parallel q) &= -\frac{1}{2} \ln \left(\frac{|\boldsymbol{\Sigma}|}{|\mathbf{S}|} \right) - \frac{1}{2} \left[d - \left((\boldsymbol{\mu} - \mathbf{m})^\top \mathbf{S}^{-1} (\boldsymbol{\mu} - \mathbf{m}) + \mathrm{Tr}(\mathbf{S}^{-1} \boldsymbol{\Sigma}) \right) \right] \\ &= -\frac{1}{2} \ln \left(\frac{|\boldsymbol{\Sigma}|}{|\mathbf{S}|} \right) - \frac{1}{2} d + \frac{1}{2} (\boldsymbol{\mu} - \mathbf{m})^\top \mathbf{S}^{-1} (\boldsymbol{\mu} - \mathbf{m}) + \frac{1}{2} \mathrm{Tr}(\mathbf{S}^{-1} \boldsymbol{\Sigma}) \\ &= \frac{1}{2} \left[\ln \left(\frac{|\mathbf{S}|}{|\boldsymbol{\Sigma}|} \right) - d + \mathrm{Tr} \left(\mathbf{S}^{-1} \boldsymbol{\Sigma} \right) + (\boldsymbol{\mu} - \mathbf{m})^\top \mathbf{S}^{-1} (\boldsymbol{\mu} - \mathbf{m}) \right].\end{aligned}$$

4 Exercise 4

1. Derive the Joint Distribution $p(y_{1:N}, x_{1:N}, \gamma)$

To derive the joint distribution $p(y_{1:N}, x_{1:N}, \gamma)$, we factorize it based on the dependencies inherent in the model.

1. **Prior Distributions:**

$$p(\gamma) = \mathcal{N}(\gamma \mid 0, \sigma_\gamma^2)$$

$$p(x_0) = \mathcal{N}(x_0 \mid \overline{x}_0, \Sigma_0)$$

2. **State Transitions:** For $n = 1, \ldots, N$:

$$p(x_n \mid x_{n-1}, \gamma) = \mathcal{N}(x_n \mid \gamma x_{n-1}, \beta_v^2)$$

3. **Observations:** For $n = 1, \ldots, N$:

$$p(y_n \mid x_n) = \mathcal{N}\left(y_n \mid \frac{1}{2}x_n, \sigma_e^2\right)$$

Combining these, the joint distribution is:

$$\begin{aligned} p(y_{1:N}, x_{1:N}, \gamma) &= p(\gamma)\, p(x_0) \prod_{n=1}^{N} p(x_n \mid x_{n-1}, \gamma)\, p(y_n \mid x_n) \\ &= \mathcal{N}(\gamma \mid 0, \sigma_\gamma^2)\, \mathcal{N}(x_0 \mid \overline{x}_0, \Sigma_0) \\ &\quad \times \prod_{n=1}^{N} \mathcal{N}(x_n \mid \gamma x_{n-1}, \beta_v^2)\, \mathcal{N}\left(y_n \mid \frac{1}{2}x_n, \sigma_e^2\right) \end{aligned}$$

2. Variational Inference with Factorized Approximation

With this variational approximation:

$$q(\gamma, x_{1:N}) = q(\gamma) \prod_{n=0}^{N} q(x_n)$$

Under the mean-field variational approximation, each variational factor is updated by taking the expectation of the log joint distribution with respect to all other variables.

1. Update for $q(\gamma)$

$$\log q(\gamma) = \mathbb{E}_{x_{1:N}}\left[\log p(\gamma) + \sum_{n=1}^{N} \log p(x_n \mid x_{n-1}, \gamma)\right] + \text{const}$$

Given the Gaussian priors and transitions, $q(\gamma)$ remains Gaussian:

$$q(\gamma) = \mathcal{N}(\gamma \mid m_\gamma, S_\gamma)$$

where

$$\begin{aligned} S_\gamma &= \left(\frac{1}{\sigma_\gamma^2} + \frac{1}{\beta_v^2} \sum_{n=1}^{N} \mathbb{E}[x_{n-1}^2]\right)^{-1} \\ m_\gamma &= S_\gamma \left(\frac{1}{\beta_v^2} \sum_{n=1}^{N} \mathbb{E}[x_n x_{n-1}]\right) \end{aligned}$$

2. Update for $q(x_n)$ for $n = 0, 1, \ldots, N$

For each x_n, the update equation is derived by considering its dependencies:

$$\log q(x_n) = \mathbb{E}_{\gamma, x_{-n}} [\log p(x_n \mid x_{n-1}, \gamma) + \log p(x_{n+1} \mid x_n, \gamma) + \log p(y_n \mid x_n)] + \text{const}$$

Given the Gaussian nature of the model, each $q(x_n)$ is Gaussian:

$$q(x_n) = \mathcal{N}(x_n \mid \mu_n, s_n^2)$$

- **For $n = 0$:**

$$s_0^2 = \left(\frac{1}{\Sigma_0} + \frac{\mathbb{E}[\gamma^2]}{\beta_v^2} \right)^{-1}$$
$$\mu_0 = s_0^2 \left(\frac{\overline{x}_0}{\Sigma_0} + \frac{\mathbb{E}[\gamma]\mathbb{E}[x_1]}{\beta_v^2} \right)$$

- **For $1 \leq n \leq N$:**

$$s_n^2 = \left(\frac{1 + \mathbb{E}[\gamma^2]}{\beta_v^2} + \frac{1}{4\sigma_e^2} \right)^{-1}$$
$$\mu_n = s_n^2 \left(\frac{\mathbb{E}[\gamma](\mathbb{E}[x_{n-1}] + \mathbb{E}[x_{n+1}])}{\beta_v^2} + \frac{y_n}{2\sigma_e^2} \right)$$

3. Variational Inference Without Factorizing the x_n Terms

Given this variational approximation:

$$q(\gamma, x_{0:N}) = q(\gamma)\, q(x_{0:N})$$

In this scenario, we maintain dependencies between the latent states $x_{0:N}$ by not factorizing them. The variational distribution is:

$$q(\gamma, x_{0:N}) = q(\gamma)\, q(x_{0:N})$$

where $q(x_{0:N})$ captures the joint distribution of all latent states.

1. Update for $q(\gamma)$

The update for $q(\gamma)$ remains similar to Problem 2, since $q(x_{0:N})$ still interacts with γ through the state transitions.

$$\log q(\gamma) = \mathbb{E}_{x_{0:N}} \left[\log p(\gamma) + \sum_{n=1}^{N} \log p(x_n \mid x_{n-1}, \gamma) \right] + \text{const}$$

Hence, $q(\gamma)$ is Gaussian:

$$q(\gamma) = \mathcal{N}(\gamma \mid m_\gamma, S_\gamma)$$

with

$$S_\gamma = \left(\frac{1}{\sigma_\gamma^2} + \frac{1}{\beta_v^2} \sum_{n=1}^{N} \mathbb{E}[x_{n-1}^2] \right)^{-1}$$

$$m_\gamma = S_\gamma \left(\frac{1}{\beta_v^2} \sum_{n=1}^{N} \mathbb{E}[x_n x_{n-1}] \right)$$

2. Update for $q(x_{0:N})$

The joint distribution $q(x_{0:N})$ must capture the dependencies between all x_n. Given the Gaussian structure of the model, $q(x_{0:N})$ is also Gaussian and can be determined using Gaussian state-space model inference techniques, such as the Kalman filter and smoother.

Key Steps:

1. **Effective Dynamics:**

$$x_n \mid x_{n-1} \sim \mathcal{N}\left(\mathbb{E}[\gamma] x_{n-1}, \mathbb{E}[\gamma^2] x_{n-1}^2 + \beta_v^2\right)$$

2. **Observations:**

$$y_n \sim \mathcal{N}\left(\frac{1}{2} x_n, \sigma_e^2\right)$$

3. **Inference Procedure:**

   - **Predict:** Use the expected dynamics to predict the distribution of x_n given x_{n-1}.
   - **Update:** Incorporate the observation y_n to update the belief about x_n.
   - **Smooth:** Refine the estimates by considering future observations.

The resulting $q(x_{0:N})$ is a multivariate Gaussian distribution that captures the temporal dependencies between states.