Probabilistic Decision Making VU, WS25
Assignment 3
Variational Inference, Markov Chain Monte Carlo

Thomas Wedenig
thomas.wedenig@tugraz.at

Teaching Assistant:	Leon Tiefenböck
Points to achieve:	35 Points
Deadline:	09.01.2026 23:59 (strict, no late submissions allowed)
Hand-in procedure:	You can work in groups of **at most two people**. **Exactly one** team member uploads 8 files to TeachCenter: **The report (.pdf)**, and all 7 Python files. The first page of the report must be the **cover letter**. Do not rename the Python files. Do not upload the `data` and `figures` folders. Do not upload a folder. Do not zip the files. We do not accept submissions via means other than TeachCenter.
Plagiarism:	If detected, we grade *all involved parties* with "Ungültig aufgrund von Täuschung"

General Remarks

Your submission will be graded based on:

- Correctness (Is your code doing what it should be doing? Is your derivation correct?)
- The depth of your interpretations (Usually, only a couple of lines are needed.)
- The quality of your plots (Is everything clearly readable/interpretable? Are axes labeled? …)

Remarks:

- All results (i.e., plots, results of computations) should be included in your PDF report.
- Report all intermediate steps in pen & paper exercises—only presenting the final solution is insufficient.
- Your submission must run with Python 3.11.13 and the package versions listed in `requirements.txt`.
  - Check TeachCenter for instructions to setup a conda environment.
- Do not use any external packages except for the ones listed in `requirements.txt`.
- **Do not modify the function signatures** of the provided functions.
  - i.e., do not edit the function names and inputs
- Do not use Large Language Models (LLMs) to generate any part of your solution.
  - Evident LLM usage will be treated as plagiarism.

Failure to adhere to these rules may result in point deductions.

Task 1 — Variational Inference [14 Points]

In this task, we consider a simple Bayesian inference problem with a latent variable model.

$$p(\mathbf{z}, \mathbf{x}) = p(\mathbf{z})p(\mathbf{x} \mid \mathbf{z})$$

where $\mathbf{z} = (z_1, z_2)^\top \in \mathbb{R}^2$ is the latent variable and $\mathbf{x} \in \mathbb{R}^3$ is the observed data. For example, assume a physical process we can measure ($\mathbf{x}$), but the underlying parameters of the system $\mathbf{z}$ are unknown.

Assume that the likelihood $p(\mathbf{x} \mid \mathbf{z})$ is known due to domain knowlege (e.g., a known physical process) and that we have domain experts that can design a sensible prior $p(\mathbf{z})$. In our case, the experts propose

- $p(\mathbf{z}) = \mathcal{N}(\mathbf{z}; \mathbf{0}, \Sigma)$ with
$$\Sigma := \begin{bmatrix} 0.5 & 0.35 \\ 0.35 & 1.0 \end{bmatrix}$$
- $p(\mathbf{x} \mid \mathbf{z}) = \mathcal{N}(\mathbf{x}; f(\mathbf{z}), \sigma_x^2 I)$ where the measurement error is $\sigma_x = 0.7$ and the non-linear function $f : \mathbb{R}^2 \to \mathbb{R}^3$ is defined as
$$f(\mathbf{z}) := (z_1^2 - 4,\ z_1 z_2,\ z_2^2)^\top$$

Given a measurement $\mathbf{x} \in \mathbb{R}^3$, we are interested in the posterior $p(\mathbf{z} \mid \mathbf{x})$. Since we cannot compute this in closed form, we will approximate the posterior in various ways.

Task 1.1 [1.5 points] In `model.py`, implement all missing methods in the class `LatentVariableModel`. At initialization, we pass `prior_mean`, `prior_cov`, and `sigma_x`. Implement

- The function f defined above[1]
- `log_p_z`, which computes the prior density $\log p(\mathbf{z})$
- `log_p_x_given_z`, which computes the likelihood $\log p(\mathbf{x} \mid \mathbf{z})$
- `log_joint`, which computes the joint $\log p(\mathbf{x}, \mathbf{z})$

Note that the method `plot_posterior` plots the 2D posterior $p(\mathbf{z} \mid \mathbf{x} = \texttt{x\_obs})$ by numerically evaluating the joint on a grid of values for $\mathbf{z}$ (with fixed $\mathbf{x} = \texttt{x\_obs}$), and normalizing it. Moreover, if passed parameters `q_mean`, `q_cov`, it will also plot the contour lines of a Gaussian with these parameters.

Task 1.2 [12.5 points] We will now approximate the posterior using *Variational Inference (VI)*. For a fixed $\mathbf{x}$, we will approximate the posterior $p(\mathbf{z} \mid \mathbf{x})$ (which is now just a function of $\mathbf{z}$) with a parameterized *variational distribution* $q_{\boldsymbol{\phi}}(\mathbf{z})$. For this task, we will pick

$$q_{\boldsymbol{\phi}}(\mathbf{z}) := \mathcal{N}(\mathbf{z}; \boldsymbol{\mu}_q, \Sigma_q)$$

with parameters $\boldsymbol{\phi} := (\boldsymbol{\mu}_q, \Sigma_q)$ where Σ_q is a *diagonal* covariance matrix $\Sigma_q := \mathrm{diag}((\sigma_1^2, \sigma_2^2)^\top)$.

1. For a fixed $\mathbf{x}$, write down the *Evidence Lower Bound* $\mathrm{ELBO}(\boldsymbol{\phi})$ as an expectation of $q_{\boldsymbol{\phi}}(\mathbf{z})$. What's the "evidence" and why is it called "Evidence Lower Bound"?

2. For this fixed $\mathbf{x}$, we want that $q_{\boldsymbol{\phi}}(\mathbf{z}) \approx p(\mathbf{z} \mid \mathbf{x})$. Why is maximizing the ELBO w.r.t. $\boldsymbol{\phi}$ a reasonable objective here? Hint: Consider the gap
$$\log p(\mathbf{x}) - \mathrm{ELBO}(\boldsymbol{\phi})$$
and write it down as a well-known divergence.

[1] This will be a static method as it does not need access to any of the fields in the class.

3. In `vi.py`, implement `elbo`, which should return a Monte Carlo estimate of the ELBO. Use the "reparameterization trick" to draw M samples from $q_{\boldsymbol{\phi}}(\mathbf{z})$ using `eps_samples`.

4. In `vi.py`, implement `vi_fit`, which should optimize the variational parameters $\boldsymbol{\phi}$ by maximizing the ELBO using vanilla gradient ascent (for `num_steps` steps and fixed learning rate `lr`). Make sure to draw fresh noise `eps_samples` in each iteration of your optimization loop. Initialize $\boldsymbol{\phi}$ in a reasonable way and track the ELBO in your optimization loop.

5. Why do we choose to optimize $\log \sigma_1, \log \sigma_2$ instead of e.g. σ_1^2, σ_2^2 directly?

6. You are given a dataset $\mathcal{X}_{\text{test}} := \{\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(N_{\text{test}})}\}$ with $N_{\text{test}} = 100$. For the first 5 data points in this set, use the `plot_posterior` method to plot both the true posterior and the corresponding variational surrogate. Also use the already imported `plot_loss` function to plot the loss function. Use reasonable settings for your hyperparameters (`num_steps`, `lr`, M). For all 5 data points, after optimizing $\boldsymbol{\phi}$ until convergence, recompute the final ELBO that this variational $q_{\boldsymbol{\phi}}$ achieves with $M_{\text{eval}} = 100,000$ Monte Carlo samples (which is a lot more than what we use during optimization) and show it in the title of the respective plot. Include these 2 plots for each of the 5 data points in your report (10 plots in total). For each plot, briefly comment on the approximation quality.

7. What happens to $q_{\boldsymbol{\phi}}(\mathbf{z})$ if the true posterior $p(\mathbf{z} \mid \mathbf{x})$ is multimodal and the modes are disconnected by (somewhat large) regions of low density (e.g., in the first test example)? Write down the mathematical definition of the divergence we minimize in VI and use this formula to explain this behaviour.

8. This divergence is not symmetric, i.e., we get a different value when swapping the two arguments. Assume we were able to minimize this divergence in the other direction (i.e., we swap the arguments): Would $q_{\boldsymbol{\phi}}(\mathbf{z})$ still behave in the same way when the true posterior has disconnected modes? Justify your reasoning with the formula of the divergence with swapped arguments.

9. In `vi_numpyro.py`, we will use `numpyro`[2] to perform variational inference with a *full-covariance* Gaussian variational distribution:

$$q_{\boldsymbol{\phi}}(\mathbf{z}) = \mathcal{N}(\mathbf{z}; \boldsymbol{\mu}_q, \Sigma_q), \qquad \boldsymbol{\mu}_q := (\mu_1, \mu_2)^\top, \qquad \Sigma_q := \begin{bmatrix} \sigma_1^2 & \Sigma_{12} \\ \Sigma_{12} & \sigma_2^2 \end{bmatrix}$$

   with parameters $\boldsymbol{\phi} := (\boldsymbol{\mu}_q, \Sigma_q)$ where Σ_q is a full-covariance matrix. Implement `numpyro_model` in `vi_numpyro.py`, which should define the model $p(\mathbf{z}, \mathbf{x})$ as a `numpyro` model. Then implement `numpyro_guide_full_cov`, which should define the guide $q_{\boldsymbol{\phi}}(\mathbf{z})$ as a `numpyro` model. The given function `vi_fit_full_cov` will then use these two functions to perform VI by minimizing the negative ELBO w.r.t. $\boldsymbol{\phi}$ using the `Adam` optimizer.

10. What is the practical issue with using full-covariance Gaussians in VI? How well does this method scale with the dimensionality D of $\mathbf{z} \in \mathbb{R}^D$?

11. Pick sensible hyperparameters and run `vi_fit_full_cov` for the first 5 test data points and create the same posterior and loss plots as above (10 plots in total), again recomputing the ELBO with $M_{\text{eval}} = 100,000$ Monte Carlo steps: For this, you can use the provided `trace_elbo_full_cov` function, since your `elbo` implementation in `vi.py` will only support diagonal covariance Gaussians. How sensitive are your results w.r.t. initialization of $\boldsymbol{\phi}$ in the `numpyro` guide? Comment on this in your report.

[2] `https://num.pyro.ai`

12. For a particular $\mathbf{x}$, the domain experts are interested in the *posterior expectation* of a particular polynomial:

$$\gamma_p := \mathbb{E}_{\mathbf{z}\sim p(\mathbf{z}|\mathbf{x})}\left[g(\mathbf{z})\right], \qquad g(\mathbf{z}) := z_1^2 + z_2^2 + z_1 z_2 + z_1 + z_2$$

Implement `g` in `gamma.py`. We will *approximate* $\gamma_p(\mathbf{x})$ by replacing the true posterior $p(\mathbf{z} \mid \mathbf{x})$ with the variational posterior $q_{\boldsymbol{\phi}}(\mathbf{z})$:

$$\gamma_q := \mathbb{E}_{\mathbf{z}\sim q_{\boldsymbol{\phi}}(\mathbf{z})}\left[g(\mathbf{z})\right]$$

Since q is Gaussian, we can compute γ_q analytically (no Monte Carlo needed): Write down the analytical expression for γ_q as a simple function of $\boldsymbol{\mu}_q = (\mu_1, \mu_2)^\top$ and the full-covariance matrix Σ_q. Use the fact that $\mathbb{E}_q[z_1 z_2] = \Sigma_{12} + \mu_1\mu_2$ and recall that $\mathrm{Var}\left[z_i\right] = \sigma_i^2 = \mathbb{E}[z_i^2] - \mathbb{E}[z_i]^2$. Implement `estimate_gamma_with_vi_posterior` in `gamma.py`, which should compute γ_q using the analytical expression. In `gamma.py`, there is a function that computes γ_p numerically[3] using a fixed grid of points. Include a table in your report that shows the numerically computed "true" γ_p and the approximations γ_q for the first 5 test data points for both (1) the diagonal and (2) the full-covariance VI cases. Comment on the approximation quality.

[3]Note that this computation would of course be intractable for high dimensions, and is only possible in this toy setting.

Task 2 – Amortized Variational Inference [7 Points]

Up until this point, we were given a single $\mathbf{x}$ and then ran our optimization procedure to obtain varational parameters $\boldsymbol{\phi} = (\mu_1, \mu_2, \log \sigma_1, \log \sigma_2)^\top \in \mathbb{R}^4$. Instead, we can train a *neural network*[4] $e_{\boldsymbol{\theta}} : \mathbb{R}^3 \to \mathbb{R}^4$ that tries to directly *learn* the map from $\mathbf{x}$ to $\boldsymbol{\phi}$. This is called *Amortized Variational Inference* (AVI). Note that this is now the first time we will access the training data $\mathcal{X}_{\text{train}} := \{\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(N_{\text{train}})}\}$ with $N_{\text{train}} = 900$.

1. In `avi.py`, a small neural network is already implemented for you (`Encoder`). In this file, implement `amortized_elbo_single`, which takes a single $\mathbf{x}$ and computes a M-sample Monte Carlo estimate of $\text{ELBO}(\boldsymbol{\phi})$, where $\boldsymbol{\phi}$ is the output of the network.

2. We will train the model by maximizing the *average* ELBO over a batch of training data $\mathcal{X}$. Implement `amortized_elbo_batch`, which takes a batch $\mathcal{B} \subset \mathcal{X}$ and computes the average ELBO over the batch.

3. Pick sensible hyperparameters and train the network by minimizing the loss (i.e., the negative ELBO). Plot the loss curve over traning iterations and include it in your report.

4. For a fixed $\mathbf{x}$, consider the so-called *amortization gap*

$$\text{AG}(\mathbf{x}) := \text{ELBO}(\boldsymbol{\phi}_{\text{VI}}) - \text{ELBO}(\boldsymbol{\phi}_{\text{AVI}})$$

   where $\boldsymbol{\phi}_{\text{AVI}} = e_{\boldsymbol{\theta}}(\mathbf{x})$ is the output of the encoder network and $\boldsymbol{\phi}_{\text{VI}}$ is the output of the non-amortized VI optimization procedure. Assume that for some $\mathbf{x}$, $\boldsymbol{\phi}^*_{\text{VI}}$ denotes the global optimum of the ELBO for the non-amortized VI case. Can the amortization gap $\text{ELBO}(\boldsymbol{\phi}^*_{\text{VI}}) - \text{ELBO}(\boldsymbol{\phi}_{\text{AVI}})$ be negative in this case? Justify your reasoning.

5. Create a plot which shows 3 subplots, which all show the true posterior and the corresponding variational surrogate using (1) amortized VI, (2) non-amortized VI with a diagonal covariance, and (3) non-amortized VI with a full-covariance. Create such a plot for each of the first 5 test data points. Recompute $\text{ELBO}(\boldsymbol{\phi}_{\text{AVI}})$ and the amortization gap for both the diagonal and full-covariance VI cases with $M_{\text{eval}} = 100,000$ Monte Carlo samples and display them in the plot. Include all 5 plots in your report, comment on the approximation quality, and discuss the ELBO and amortization gap values. Can it happen that the observations differ from the theoretical scenario we assumed previously ($\boldsymbol{\phi}^*_{\text{VI}}$)? If so, why? How sensitive is the amortization gap to the hyperparameters and initialization of the non-amortized VI parameters?

6. Compute the amortization gap for all test data points for both the diagonal and full-covariance VI cases. Given this set of numbers, report mean, standard deviation, minimum, and maximum in your report and comment on it. What is the computational bottleneck here?

7. What is the practical advantage of amortized VI over non-amortized VI? What are the limitations of amortized VI?

8. How does the setup in Task 2 differ from a *Variational Autoencoder* (VAE) and how are they related?

[4] In the context of VAEs, such a network is called an *encoder* network.

Task 3 – Markov Chain Monte Carlo [14 Points]

Next, we will use a different technique to approximate the Bayesian posterior, namely through *sampling* via *Markov Chain Monte Carlo* (MCMC). In this section, we will overload q to now denote the proposal distribution for the MCMC samplers (which is unrelated to the variational distributions we considered earlier). We will use all these algorithms to sample from the posterior $p(\mathbf{z} \mid \mathbf{x})$ for a fixed $\mathbf{x}$.

1. In `mcmc.py`, implement the *Random Walk Metropolis-Hastings (MH)* algorithm to sample from the posterior $p(\mathbf{z} \mid \mathbf{x})$ for a fixed $\mathbf{x}$. We will use an isotropic Gaussian proposal distribution:

   $$q(\mathbf{z}' \mid \mathbf{z}) = \mathcal{N}(\mathbf{z}'; \mathbf{z}, \sigma^2 I)$$

   where $\mathbf{z}'$ is the proposed point, $\mathbf{z}$ is the current point, and σ is a hyperparameter of the sampler. Implement `metropolis_hastings_random_walk`: This function runs the algorithm to sample from multiple *chains* using multiple initial points. Discard the first `burn_in` points in every chain. Return the samples of shape `(num_chains, num_samples, 2)` and return the *acceptance rate*, which is the average acceptance probability α after the burn-in phase (also averaged over all chains). In your report, state how you compute the acceptance probability α.

2. Next, we will implement the *Metropolis-Adjusted Langevin Algorithm* (MALA), which is an instance of the Metropolis-Hastings algorithm where the proposal distribution is a Gaussian with a momentum term:

   $$q\left(\mathbf{z}' \mid \mathbf{z}\right) = \mathcal{N}\left(\mathbf{z}'; \mathbf{z} + \frac{\eta}{2} \nabla_{\mathbf{z}} \log p\left(\mathbf{z} \mid \mathbf{x}\right),\ \eta I\right)$$

   where η is the `step_size` (a hyperparameter of the sampler). Explain in your report how we can compute the gradient $\nabla_{\mathbf{z}} \log p\left(\mathbf{z} \mid \mathbf{x}\right)$, even though we do not have access to the posterior $p(\mathbf{z} \mid \mathbf{x})$. Implement `mala`, which runs MALA for multuple chains. Again, discard the first `burn_in` points in every chain and return the samples and the *acceptance rate* (averaged over all chains). In your report, state how you compute the acceptance probability α.

3. Use `numpyro`'s implementation of the No U-Turn Sampler (NUTS), which is a variant of Hamiltonian Monte Carlo (HMC). Implement `nuts`, which calls the NUTS sampler and returns the samples for all chains.

4. For the first 5 test data points, run all 3 MCMC methods and create a plot of the posterior, overlaid with the samples you obtained from the MCMC methods. Include these 5 plots in your report (where each plot shows the posterior and the samples from all 3 methods).

5. After running them for sufficiently long, will these MCMC methods return *i.i.d.* samples from the posterior $p(\mathbf{z} \mid \mathbf{x})$? Justify your reasoning.

6. Read the provided function `compute_diagnostics`, which consumes the samples from all chains and computes (1) the Gelman-Rubin $\hat{R}$ statistic, and (2) the *Effective Sample Size* (ESS) for the samples. Explain in your report how these quantities are computed, how we can interpret them, and how they can be used to diagnose the quality of the MCMC samples.

7. For each of the 3 MCMC methods, plot the Autocorrelation Function (ACF) using the provided function `plot_acfs`, which will also display the $\hat{R}$ and ESS values for the 2 dimensions of $\mathbf{z}$. Carefully read the code and explain in your report how the ACF is computed. Include this plot in your report for the first 5 test data points. For each method, interpret and comment on the ACFs, $\hat{R}$ and the ESS values.

8. Play around with the hyperparameters σ and η for the MH and MALA algorithms and observe how the ACFs, $\hat{R}$ and ESS values change for different values of these hyperparameters. For each of the 5 test data points, create one more `plot_acfs` plot that uses different hyperparameters that change the ACFs substantially. Include them in your report and briefly discuss your observations. Pick good hyperparameters for all 3 methods and include them in your report. You are also allowed to use different hyperparameters for each of the 5 test points if you think this is necessary.

9. Implement `estimate_gamma_with_mcmc_samples` in `gamma.py`, which uses the MCMC samples to estimate γ_p. Use this function in `mcmc.py` to estimate γ_p for the first 5 training data points (with all 3 methods) by picking a sensible value for `num_samples` (which should be the same for all methods, and large enough to get a good approximation). Add the approximations to the table in your report. When/why do they differ substantially from the VI estimates?
10. Is this estimator biased or unbiased? What happens to the bias as `num_samples` $\to \infty$? What happens to the variance as `num_samples` $\to \infty$? Justify your reasoning.

Task 4 – Peer Review [0 Points]

Each group member must write a single paragraph outlining their opinion on the work distribution within the group. Did every group member contribute equally? Did you split up tasks in a fair manner, or jointly worked through the exercises? Do you think that some members of your group deserve a different grade from others?