Final PLM project 2025

Group 2

Marcus Winther Dreisler
Department of Chemistry
University of Copenhagen
md@chem.ku.dk

Carolin Christin Heinzler
Department of Computer Science
University of Copenhagen
chei@di.ku.dk

Sebastian Holmby Hansen
Department of Computer Science
University of Copenhagen
seha@di.ku.dk

Julie Maria Johansen
Department of Computer Science
University of Copenhagen
jujo@di.ku.dk

January 17, 2025

Abstract

This report presents our solutions to the exam project for the Probabilistic Machine Learning (PML) course. The project is divided into two main parts: the first explores various diffusion model variants in comparison to a baseline model, while the second focuses on Gaussian processes under constraints.

Relevant code for the programming exercises can be found in following GitHub repository: `https://github.com/mdreisler/PML-hand-in`.

A: Diffusion-based generative models

In the provided Denoising Diffusion Probabilistic Model (DDPM) by Ho et al. [2020], the network predicted the noise ϵ. This DDPM will be called the baseline DDPM, and samples drawn from this model are shown in Figure 1. In the following sections, we will make variations of this DDPM and compare their performances.

A.1: Variations/extensions of the basic MNIST diffusion model

Predicting x_0

From the Ho et al. [2020] paper we also know that it is a possibility to have the network predict x_0, although they found a worse sample quality in doing so. We implemented the x_0-predicting network first by keeping the forward diffusion as $q(x_t|x_0) = x_t = \sqrt{\bar{\alpha}_t}x_0 + 1 - \bar{\alpha}_t\epsilon$ where ϵ is the noise sample $\epsilon \sim \mathcal{N}(0, 1)$. For the reverse diffusion predicting x_0, we began by calculating the mean and standard for $q(x_{t-1}|x_t, x_0) = \mathcal{N}(x_{t-1}; \tilde{\mu}_t(x_t, x_0), \tilde{\beta}_t I)$ by using equation 7 in the paper; $\tilde{\mu}_t(x_t, x_0) = \frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1-\bar{\alpha}_{t-1}}x_0 + \frac{\sqrt{\alpha_t}(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}x_t$ and $\tilde{\beta}_t = \frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}\beta_t$. We further changed the objective function to the mean squared error between the predicted and true x_0. Samples drawn from the DDPM predicting x_0 can be seen in Figure 2

Reducing variance with low-discrepancy sampler

Our estimates for ELBO are quite noisy, so we attempt to reduce this variance first by using the low-discrepancy sampler from Kingma et al. [2021]. This approach samples "Low-discrepancy" time steps instead of random independent sampling by sampling a single uniform random number $u_0 \sim \mathcal{U}[0, 1]$ and then sets the time steps to $t^i = \mod(\frac{u_0+1}{k}, 1)$, where we process a minibatch of k examples. The authors claim that the minibatch of time steps will cover the space in $[0, 1]$ more equally which will reduce the variance of the noisy estimates. Samples drawn from the DDPM variant using low-discrepancy sampler for variance reduction can be seen in Figure 3

Reducing variance with importance sampling

We also attempted to reduce the variance of the training objective by using importance sampling of the time steps *t* as suggested by Nichol and Dhariwal [2021]. We apply importance sampling by using equation 18 in the paper: $L_{vlb} = E_{t\sim p_t}[\frac{L_t}{p_t}]$, where $p_t \propto \sqrt{E[L_t^2]}$ and $\sum p_t = 1$. The idea is to reweigh the loss based on the importance for different time steps. We store an exponential moving average (EMA) of the squared loss for each time step, and the importance sampling probability p_t is proportional to this measure. In the beginning of training, we sample *t* uniformly until we draw 10 samples for each *t*. Reducing the variance using importance sampling makes the model train slower since we are drawing more samples for each *t*. Samples drawn from the DDPM variant using importance sampling for variance reduction can be seen in Figure 4

Continuous-time version of the diffusion model

Transitioning from the discrete DDPM to a continuous-time diffusion model involves shifting from optimizing the ELBO objective to minimizing the MSE between an estimated score and the true data score, the so-called score matching. Our implementation follows that of Song et al. [2020]:

$$\mathcal{L}_{\text{score}} = \mathbb{E}_{t\in[0,T],\mathbf{x}_0,\epsilon}\left[\|s_\theta(\mathbf{x}_t, t) - \nabla_{\mathbf{x}_t}\log p_t(\mathbf{x}_t)\|^2\right]$$

For the VE SDE framework, which we implemented, the noising schedule is governed by a continuous function of t, for which the perturbation kernel is given by

$$\mathcal{N}(x(t); x(0), [\sigma^2(t) - \sigma^2(0)]\mathbf{I})$$

Thus, given a noise schedule via $\sigma(t)$, we may obtain a noisy sample, much like in the case of the discrete diffusion model. The score network takes, as was the case for the denoising network in the discrete case, a noisy sample and time encoding as arguments. Using the Euler-Maruyama method and our network estimate of $\nabla_x \log p_t(x)$, we can generate samples from the learned model via

$$\mathrm{d}x = \left[\mathbf{f}(x, t) - g(t)^2\nabla_x \log p_t(x)\right]\mathrm{d}t + g(t)\mathrm{d}\bar{\mathbf{w}}.$$

Where,

$$\begin{aligned} f(x,t) &: \text{Drift term that depends on the state } x \text{ and time } t. \\ g(t) &: \text{Time-dependent diffusion coefficient.} \\ \mathbf{w} &: \text{The standard Wiener process.} \end{aligned}$$

The results of the continuous-time version of the diffusion model can be seen in Figure 5.

Conditional sampling with classifier guided diffusion

By initially training a classifier $f_\theta(y \mid x_t, t)$ on the MNIST dataset alongside noisy images generated through the diffusion process (for $t \in \{0, \dots, T\}$), we can produce samples from the conditional distribution $p(x \mid y)$, where y represents the label of the desired digit to sample.
We first trained the classifier f_θ and then according to the blogpost of Weng, adapted the sampling process of the trained diffusion model. We sample x_{t-1} (when x_t is given) from the distribution $\mathcal{N}(\mu_\theta(x_t, t) + w \cdot \sigma^2 \cdot \nabla_{x_t} f_\theta(y|x_t, t)|\sigma^2)$, where $w > 0$ is a weight parameter, $\mu_\theta(x_t, t)$ the same mean model (which uses the NN $\epsilon_\theta(x_t, t)$) and $\sigma^2 = \frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t} \cdot \beta_t$ is the same variance as in the unguided model. To generate the samples in Figure 6 we used $w = 8$ and $y = 4$ to generate 10 images of 4s.

A.2: Quantitative comparisons

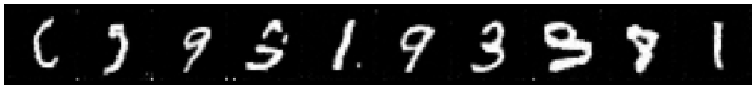
Figure 1: 10 samples drawn from the baseline DDPM predicting noise ϵ.

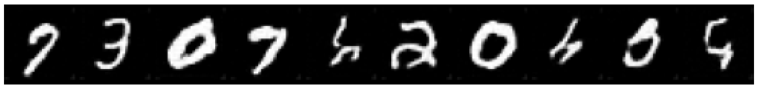
Figure 2: 10 samples drawn from the DDPM variant predicting x_0.

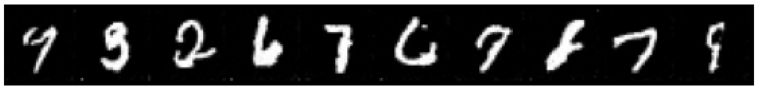
Figure 3: 10 samples drawn from DDPM variant using low-discrepance sampler.

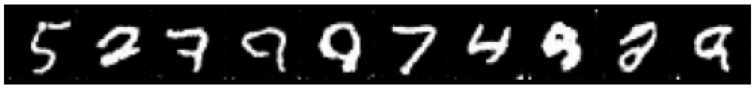
Figure 4: 10 samples drawn from DDPM variant using importance sampling.

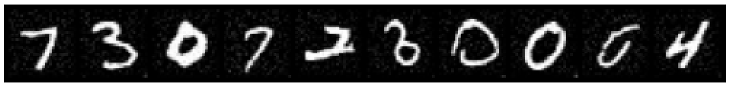
Figure 5: 10 samples from the continuous time diffusion model.

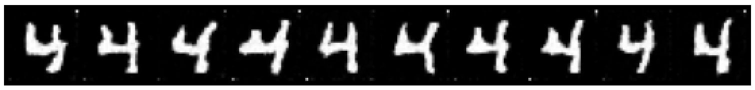
Figure 6: 10 samples from the classifier guided diffusion model.

Metric: FID and Inception score

The inception score introduced by Salimans et al. [2016] is a metric used to evaluate how realistic generated images are in terms of quality and diversity. Inception score is based on a pretrained model called the inception model and is calculated from features extracted from the images. The metric is given by $\exp E_x KL(p(y|x)||p(y))$ containing the KL divergence between the conditional distribution $p(y|x)$ and the marginal distribution $p(y)$. We calculated the inception score using `torchmetrics`' function for inception.[1] The inception score expect RBG images, so we have repeated the gray-scale samples 3 times.

The Fréchet Inception Distance (FID) score is used to assess the quality of generated images by comparing the distribution of feature embeddings from the Inception model in a set of real images to the embedded features of the generated images. The FID score is given by $||\mu - \mu_w|| + \text{trace}(\Sigma + \Sigma_w - 2(\Sigma\Sigma_w)^{\frac{1}{2}}$. We calculated the FID score using `torchmetrics`' function for FID. [2]

Using likelihood as a metric

In generative models, when validating the generated samples, typical performance metrics such as precision are not trivial as in supervised models; hence, the log-likelihood of such unseen data points becomes the only objective measure in model comparison. In a diffusion model the marginal likelihood of the data x_0 is:

$$p(x_0) = \int p(x_{0:T}) dx_{1:T}$$

with

$$p(x_{0:T}) = p(x_T) \prod_{t=1}^{T} p(x_{t-1} \mid x_t)$$

The analytical likelihood therefore involves integrating over the entire sequence of latent variables $x_{1:T}$, which for large T becomes computationally intractable, leading us to use the ELBO on $\log p(x_0)$ instead.

$$\log p(x_0) \geq \mathbb{E}_{q(x_{1:T}|x_0)} \left[\log \frac{p(x_{0:T})}{q(x_{1:T} \mid x_0)} \right].$$

Following the DDPM-procedure from Ho et al. [2020] (eq. 5 therein), this can be expanded to

$$\log p(x_0) \geq \mathbb{E}_q \left[-\log \frac{p(x_T)}{q(x_T \mid x_0)} - \sum_{t>1} \log \frac{p_\theta(x_{t-1} \mid x_t)}{q(x_{t-1} \mid x_t, \mathbf{x}_0)} - \log p_\theta(x_0 \mid x_1) \right]$$

This separates the log likelihood into a KL term and a reconstruction term. Given a trained model, and a real sample for the dataset, we can obtain the necessary components: $q(x_t \mid x_{t-1})$ is known exactly from the forward diffusion and from the trained network we have a learned expression for $p(x_t \mid x_t)$. If we assume that there are enough $t \in T$ to adequately ensure that the x_T is indeed complete noise, then $p(x_T) = q(x_T \mid x_0)$, such that the first term is $\log(1) = 0$.

- In practice, we sample a path $x_{1:T}$ from the forward process $q(x_{1:T} \mid x_0)$.
- We then compute the per-timestep Kullback–Leibler divergence (KL) between $q(x_{t-1} \mid x_t, x_0)$ and $p_\theta(x_{t-1} \mid x_t)$ (middle term eq. 5).
- At the final step ($t = 1$), we add a reconstruction term $-\log p_\theta(x_0 \mid x_1)$ (final term in eq. 5).
- Summing all these terms yields an estimate of $-\log p_\theta(x_0)$, or a lower bound on it (the ELBO).
- To account for the potential issues in comparing likelihood across different datasets, a transformed log likelihood measure, the so-called bits per dimension (bpd), can be obtained by normalizing by the dimensionality of the data.
- In the code, we evaluate this term-by-term for each batch x_0 and then average the resulting bpd (or ELBO) values across a subset of 1000 samples to quantify model performance.

In general, likelihood is used as it is an objective and, at least analytically, well-defined measure, for which a higher likelihood means the model better *explains* or *covers* the distribution of the data. Especially for our diffusion model purposes, it ties directly to how the model class is trained (through maximizing an ELBO). There are, however, limitations to the use of likelihood as a metric, as the exact likelihood can be intractable and approximations are somewhat computationally expensive with no guarantee of it being aligned with certain qualitative goals like sharpness or diversity of samples.

[1]For full documentation see `https://lightning.ai/docs/torchmetrics/stable/image/inception_score.html`

[2]For full documentation see `https://lightning.ai/docs/torchmetrics/stable/image/frechet_inception_distance.html`

Model performance

Different implementations of variations of the MNIST diffusion model are elaborately described in section A1. We employ three different quantitative model assessment scores, as described in section A2, the FID, Inception score, and a likelihood-based comparison, although the likelihood assessment was limited to the first three implementations of model variants. We calculated these scores based on 1000 samples.
The results are shown below in table 1.

Table 1: Performance of model variants

DDPM variation	**Inception score**	**FID**	**NLL (bits/dim)**
Predicting ϵ (baseline)	1.874 ± 0.060	115.1	3.278
Predicting x_0	1.825 ± 0.074	128.0	14.12
Low-discrepancy sampler	1.435 ± 0.040	360.9	3.073
Importance sampling	1.988 ± 0.057	135.0	-
Continuous-time model	1.786 ± 0.069	315.8	-
Classifier guided diffusion[3]	1.152 ± 0.021	422.8	-

B: Function fitting with constraints

B.1: Fitting a standard GP

The chosen model

We first select a suitable model. For this, we choose a kernel that is a product of two kernels; the *RBF* kernel, which we have previously seen in the course, and the *periodic* kernel, as the data seems to have periodic tendencies from the visualization of the data. The periodic kernel on itself cannot deal with variation in the periodicity, and as such we take the product of that and the RBF kernel. The kernels are defined (with *p* being the periodic parameter and *l* the lengthscale) as[4]

$$\begin{aligned} k_{\text{RBF}}(x,z) &= \sigma^2_{RBF} \exp\left(-\frac{0.5|x-z|^2}{l^2_{RBF}}\right) \\ k_{\text{Periodic}}(x,z) &= \sigma^2_{Periodic} \exp\left(-\frac{2\sin^2(\pi(x-z)/p)}{l^2_{Periodic}}\right) \\ k_{\text{Combined}}(x,z) &= k_{\text{RBF}}(x,z) \cdot k_{\text{Periodic}}(x,z) \end{aligned}$$

where k_{Combined} is then the kernel we choose for this assignment, and which we will from now on just refer to as *k*. We have the following priors for the parameters of *k*:

$$\begin{aligned} \text{Variance}_k &= \sigma^2_{RBF} = \sigma^2_{Periodic} = \Gamma(2.0, 2.0) \\ \text{Lengthscale}_k &= l_{RBF} = l_{Periodic} = \text{HalfNormal}(0.0, 2.0) \\ \text{Period}_k &= p = \text{HalfNormal}(0.0, 0.2) \\ \text{Observation noise} &= \text{HalfNormal}(0.0, 0.1) \end{aligned}$$

For all four parameters, we chose prior distributions that only yield non-negative values, as this was a necessity for the given parameters. The variance and lengthscale priors were the same for both the periodic and RBF kernels, but sampled individually to allow more room in the optimization. For the model variance, we chose a Gamma distribution with a quite wide probability mass to capture the possibility of both large and smaller values of variance, and because we did not have much prior intuition on the value. We chose a HalfNormal distribution for the lengthscale parameter, with more probability mass on the lower end of the values, but still with room for higher values of lengthscale (as an alternative we also considered - and tested - LogNormal and Gamma priors, but the HalfNormal allowed us to best model our uncertainty about the parameter value). For the period, we again know that it should be positive, and by

[4]Both kernels are implemented in Pyro as `gp.kernels.RBF` and `gp.kernels.Periodic`, see `https://docs.pyro.ai/en/dev/contrib.gp.html#module-pyro.contrib.gp.kernels` for full documentation. Throughout this assignment we will primarily be using Pyro to model our gaussian processes.

visual inspection it should also be fairly low - so we use the HalfNormal prior to put more probability mass on lower values while ensuring non-negativity. Finally, for the observation noise, we are given the actual value in the assignment text (0.01), but to be 'fair' we pretend we don't know the actual noise. Instead, we use the mentioned HalfNormal prior, again with a fairly low scale and starting from zero.

MAP estimate and NUTS sample quality

We then compute the maximum a-posteriori estimate θ^* using gradient descent, and evaluate the posterior log-likelihood of the test set using the fitted GP (i.e., we compute $\log p(y_{test}|x_{test}, \mathcal{D}, \theta^*)$). The function we used to so is given in Figure 9 in appendix section B1, and a visualization of the fitted GP can be seen in Figure 10 in the same appendix.

Next, we sample from the posterior using NUTS (again using the Pyro implementation) and perform diagnostics using Arviz. The full summary from Arviz is included in appendix B1, and we here focus on the most important results. For the NUTS sampling, we (using Arviz) decided to sample using four chains, with 250 warm-up steps and 500 samples. These hyperparameters seemed to guarantee that all parameters had $\hat{R}$ close to 1, and with what we deemed as sufficient effective sample size. As can be seen in the summary table, all parameters have both bulk and tail effective sample size above 400, which as mentioned should be sufficient. One thing of note - which may imply that further consideration could be put into the priors of our model - is that the tail effective sample size is lower than the bulk effective sample size by a fair margin for all sampled parameters. To further investigate this, we also considered the trace plot of the samples, which can be seen in Figure 11 in appendix B1. From this plot, it seems that all parameters converge for all chains without any notable drift or trends. Thus, after exploring a host of priors and NUTS hyperparameters, the aforementioned was deemed the most successful hyperparameters.

Approximate posterior likelihood

Next, we compute the approximate posterior likelihood for the samples. We do so by drawing 500 samples of each parameter at random from the obtained samples from the posterior. We then compute the predictive mean and covariance for each sample (similar to what we did to find the posterior log-likelihood of the MAP estimate), that is

$$\mu_{pred}^{(s)} = \mathrm{E}[y_{test}|x_{test}, \theta^{(s)}]$$
$$\Sigma_{pred}^{(s)} = \mathrm{Cov}[y_{test}|x_{test}, \theta^{(s)}]$$

where $\theta^{(s)}$ is the sampled parameters. We then input these into a multivariate normal distribution, yielding

$$p(y_{test}|x_{test}, \theta^{(s)}) = \mathcal{N}\left(y_{test}|\mu_{pred}^{(s)}, \Sigma_{pred}^{(s)}\right)$$

To get the approximate posterior log likelihood, we then sum over (for all samples) the log of these likelihoods, as so:

$$\log p(y_{test}|x_{test}, \mathcal{D}) = \frac{1}{S}\sum_{s=1}^{S} \log p(y_{test}|x_{test}, \theta^{(s)})$$

We implement this in Pyro, and the implementation can be seen in our submitted code (GitHub repo) for this task.

Comparison of likelihoods

Finally, we randomly generate 20 different datasets, run the two aforementioned methods, and compare the likelihoods of the model using MAP and approximate posterior likelihood. The descriptive statistics of these 20 runs can be seen in Table 3, while the results of each individual run can be seen in Table 4 (both in appendix B1). In this section, we highlight the most important of these results. First and foremost, we see that the posterior log-likelihood of the test set on the GP fitted to the MAP is generally much higher than the approximate posterior log-likelihood of the MCMC samples, with a mean of 3.695 for MAP in comparison to -19.352 for MCMC. Similarly, the highest (approximate) log-likelihood achieved by the MCMC sampling, -3.609, is significantly below both the mean and median log-likelihoods achieved by the MAP-fitted GP. As such, it is clear that the GP fitted using MAP performs much better than the MCMC sampling method. The exact reasons for this aren't easy to pinpoint, but we hypothesize that this might be a result of the MAP fitting leading to overfitting because of the fairly low number of data points, or perhaps the MCMC sampling's log-likelihood is closer to the true uncertainty of the model.

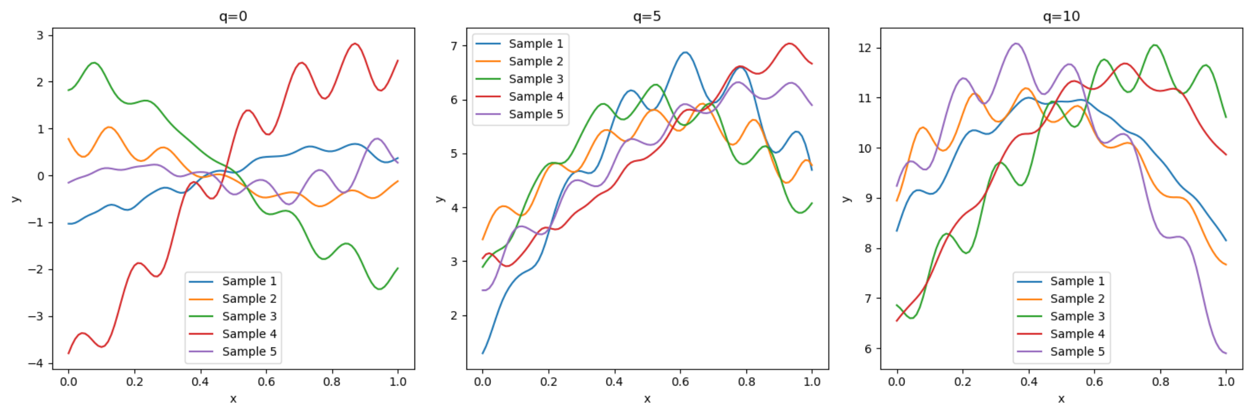

Figure 7: 5 samples from $f|X, \hat{q}$ for $q \in \{0, 5, 10\}$

B.2: Learning with integral constraints

We will denote $w = (w_1, \ldots, w_l) \in \mathbb{R}^l$, $f(X) = (f(x_1), \ldots, f(x_l)) \in \mathbb{R}^l$ and $k(X) = (k(x_i, x_j))_{i,j=1}^{l} \in \mathbb{R}^{l \times l}$. To determine the probability of the random variable $(\hat{q}, f)|X$, we first note that $f|X = f(X) \sim \mathcal{N}(0, k(X))$, as $f \sim \mathcal{GP}(0, k(\cdot, \cdot))$. Furthermore, the vector $(\hat{q}, f)|X$ can be written as a linear transformation of f: we write $(\hat{q}, f)|X = Qf|X$, where $Q = \begin{pmatrix} w \\ \mathbb{1}_l \end{pmatrix} \in \mathbb{R}^{(l+1)\times l}$ the vector w stacked on top of the identity matrix in l-dimensions $\mathbb{1}_l$. Thus, it immediately follows from the linear transformation of Gaussian random variables, that

$$(\hat{q}, f)|X = Qf|X \sim \mathcal{N}(0, Qk(X)Q^T).$$

To derive the distribution of the random variable $f|X, \hat{q}$, we can use the result from the lecture about determining the distribution of $X_2|X_1$, when (X_1, X_2) is a joint normal distribution. In our case, we condition the vector $(\hat{q}, f)|X$ on $\hat{q}$ and we immediately get that $f|X, \hat{q} \sim \mathcal{N}(\mu_{2|1}, \Sigma_{2|1})$, where

$$\mu_{2|1} = \mu_2 + \Sigma_{21}\Sigma_{11}^{-1}(\hat{q} - \mu_1) = \Sigma_{21} \cdot \frac{\hat{q}}{wk(X)w^T} \tag{1}$$

$$\Sigma_{2|1} = \Sigma_{22} - \Sigma_{21}\Sigma_{11}^{-1}\Sigma_{21}^T = k(X) - \Sigma_{21}\Sigma_{21}^T \frac{1}{wk(X)w^T} \tag{2}$$

where we used $\Sigma_{11} = wk(X)w^T$, $\Sigma_{22} = k(X)$ and $\Sigma_{21} = (\langle w, k(x_i, X)\rangle)_{i=1}^{l} \in \mathbb{R}^l$.
The covariance matrix of $f|X, \hat{q}$ cannot have full rank. Assuming that k is a universal kernel, we know that the matrix $k(X)$ must have full rank l (as 0 is no eigenvalue, thus the kernel is only trivial). Through conditioning on $\hat{q} = \langle w, f(X)\rangle$, we restrict the random variables $f(x_1), \ldots, f(x_l)$ to lie on a $l - 1$ dimensional linear subspace (namely their weighted sum being equal to $\hat{q}$), thus $f|X, \hat{q}$ does not have full rank.

In figure 7 we have drawn 5 samples from the Gaussian process f, with the MAP kernel parameters obtained from B1, at 101 datapoints X with the constraint that the random function integrates to $\hat{q} = \{0, 5, 10\}$. We observe that the values the function takes increases with higher $\hat{q}$. The higher the area under the curve, the more similar the functions seem to become (for $\hat{q} = 0$ there is quite different trajectories, compared to $\hat{q} = 10$).

In figure 8 we have plotted the posterior distributions $f|\mathcal{D}, X$ and $f|\mathcal{D}, \hat{q}, X$ and drawn 5 sample functions from them respectively. We use the kernel we have derived in B1 and sample at 101 datapoints X. Note that we condition on the given dataset $\mathcal{D}$, thus we need to use the distribution we have derived in the first two parts of this exercise and condition this distribution on $\mathcal{D}$. We can clearly observe that the variance for x closer to 1 decreases for the integral constrained version. This is what we would expect with the three points given in the interval $[0, 0.5]$, in order to fulfill the integral constraint of $\hat{q} = 2$ there is not that many functions left on the interval $[0.5, 1]$. Furthermore, we observe that in both cases for the given points $\mathcal{D}$ the variance around them decreases, however is not 0 as we observe noisy samples.

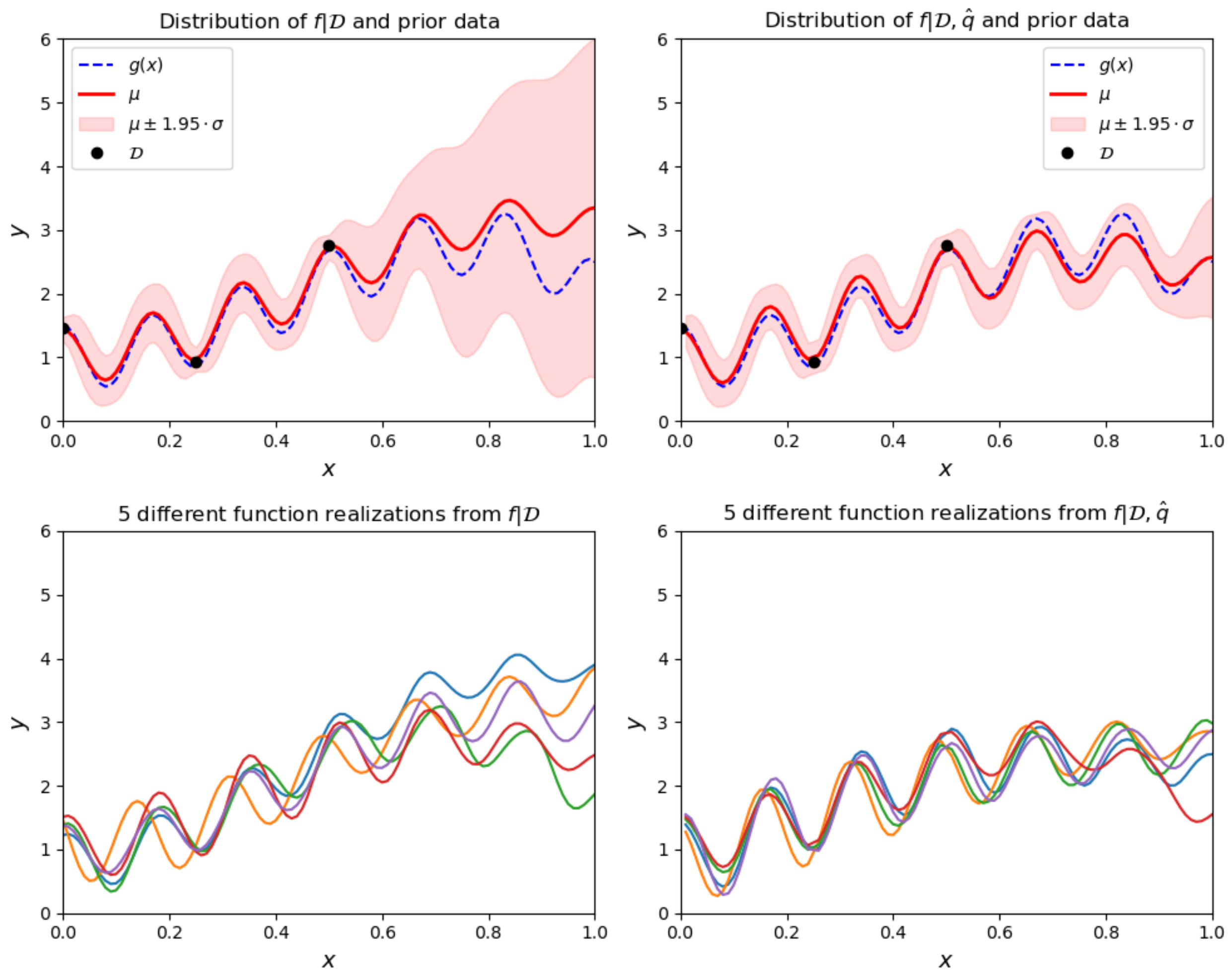

Figure 8: Comparison of distributions of $f|\mathcal{D}, X$ and $f|\mathcal{D}, \hat{q}, X$ and sampling 5 functions respectively.

References

Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in neural information processing systems*, 33:6840–6851, 2020. URL `https://arxiv.org/abs/2006.11239`.

Diederik Kingma, Tim Salimans, Ben Poole, and Jonathan Ho. Variational diffusion models. *Advances in Neural Information Processing Systems*, 34:21696–21707, 2021. URL `https://arxiv.org/abs/2107.00630`.

Alexander Quinn Nichol and Prafulla Dhariwal. Improved denoising diffusion probabilistic models. In *International Conference on Machine Learning*, PMLR, 2021. URL `https://arxiv.org/abs/2102.09672`.

Yang Song, Jascha Sohl-Dickstein, Diederik P. Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. Score-based generative modeling through stochastic differential equations. *CoRR*, abs/2011.13456, 2020. URL `https://arxiv.org/abs/2011.13456`.

Lilian Weng. What are Diffusion Models? URL `https://lilianweng.github.io/posts/2021-07-11-diffusion-models/`.

Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training gans. *arXiv preprint arXiv:1606.03498*, 2016. URL `https://arxiv.org/abs/1606.03498`.

Appendices

B1

```python
    #Corresponds to equation 57 of the PML notes from Oswin.
    def log_likelihood(gpr, x_test, y_test):
        with torch.no_grad():
            predictive_mean, predictive_cov = gpr(x_test, full_cov=True,
                                                  noiseless=False)

            predictive_distribution = dist.MultivariateNormal(predictive_mean,
                                                  predictive_cov)

            log_likelihood = predictive_distribution.log_prob(y_test)

            return log_likelihood.item()
```

Figure 9: Function to calculate the posterior log-likelihood given `gpr` a Pyro GPRegression object (with a fitted kernel)

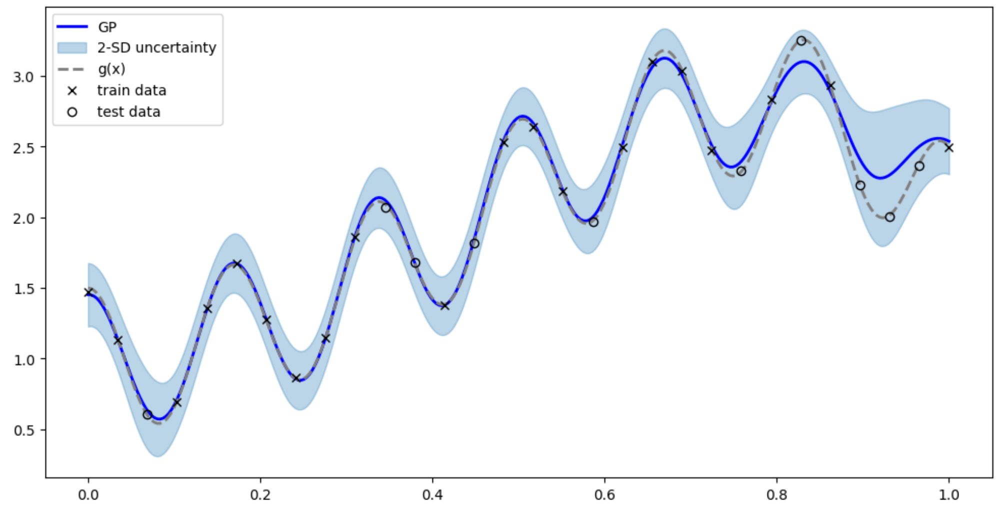

Figure 10: GP with θ^* estimated from MAP, trained using gradient descent. The dotted line indicates the true function $g(x)$, with crosses marking training points and circles marking test points. The blue line shows the fitted GP, while the shaded area shows the 2-standard deviation uncertainty around the prediction. As can be seen, the model is significantly less certain (and further away from the actual y-values) for the last couple of data points, presumably because there is a couple of test points in succession.

Table 2: Summary diagnostics from Arviz on the sampled posterior of the proposed GP, with each row representing a parameter.

	mean	sd	hdi_3%	hdi_97%	mcse_mean	mcse_sd	ess_bulk	ess_tail	r_hat
lengthscale_periodic	1.567	1.2	0.003	3.741	0.034	0.024	855.0	682.0	1.0
lengthscale_rbf	1.576	1.249	0.003	3.809	0.034	0.024	733.0	417.0	1.01
noise	0.081	0.061	0.0	0.191	0.001	0.001	1200.0	755.0	1.0
period	0.162	0.127	0.0	0.392	0.004	0.003	749.0	438.0	1.0
variance_periodic	1.018	0.728	0.026	2.285	0.017	0.012	1546.0	808.0	1.0
variance_rbf	1.007	0.693	0.069	2.301	0.02	0.014	1033.0	837.0	1.0

Table 3: Descriptive statistics of the MAP and NUTS posterior log-likelihoods, including mean and standard deviation. Here, *Min* and *Max* refer to lowest and highest likelihoods observed, respectively, and the quantiles are also shown.

Measure	**MAP**	**NUTS**
Mean	3.695	-19.532
Standard deviation	12.038	16.072
Min	-21.572	-75.567
25%	-6.232	-24.927
50%	7.742	-17.229
75%	13.439	-10.316
Max	19.318	-3.609

Table 4: Full results for 20 random permutations of dataset, with comparison of posterior log-likelihood for the MAP estimate and approximate posterior log-likelihood through NUTS.

Iteration	**MAP**	**NUTS**
1	17.538	-29.558
2	10.663	-29.688
3	16.84	-10.44
4	10.711	-35.826
5	-3.816	-9.945
6	7.639	-5.097
7	-5.84	-17.395
8	-5.567	-5.967
9	7.844	-5.851
10	16.428	-10.84
11	0.447	-3.609
12	15.95	-17.062
13	11.663	-13.508
14	-21.572	-10.444
15	19.318	-18.822
16	-9.667	-18.711
17	-7.408	-27.462
18	-8.369	-24.082
19	12.602	-75.567
20	-11.469	-20.767

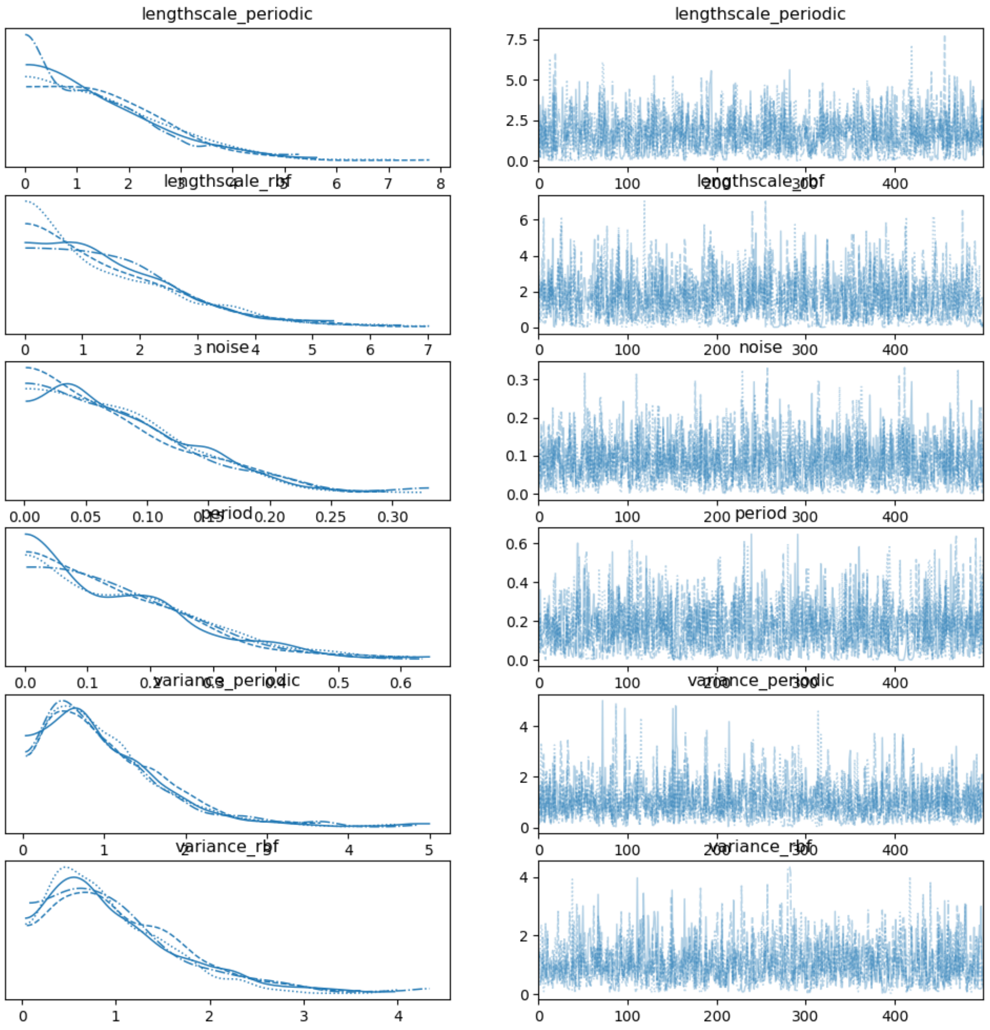

Figure 11: Trace plot of the samples generated using Arviz.