# PROBABILISTIC MACHINE LEARNING - FINAL PROJECT

**Christian M. Jensen**
vdj579

**Frederik Kraglund**
fpk297

All code can be accessed at `https://github.com/Kraglund101/PML_exam`.

## 1 Diffusion-based generative models

### 1.1 Background

Throughout this course, we have explored methods for generating unconditional images using diffusion models. Conditional image generation can be achieved through two primary approaches: Classifier-Guided Diffusion [Dhariwal and Nichol, 2021] and Classifier-Free Guided Diffusion [Ho and Salimans, 2022]. Below, we provide an overview of these methodologies and their underlying principles.

#### 1.1.1 Classifier-Guided Diffusion

Classifier-Guided Diffusion (CGD) integrates class-specific information into the sampling process via a trained classifier $f_\phi(y \mid \mathbf{x}_t, t)$. Given an unconditional denoising diffusion model $p_\theta(\mathbf{x})$ parameterized through $\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)$, the score function can be expressed as:

$$\nabla_{\mathbf{x}_t} \log p_\theta(\mathbf{x}_t) = -\frac{1}{\sqrt{1-\bar{\alpha}_t}} \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t).$$

To incorporate class-specific information, the score function for $p(\mathbf{x}_t, y)$ becomes:

$$\begin{aligned}\nabla_{\mathbf{x}_t} \log p(\mathbf{x}_t, y) &= \nabla_{\mathbf{x}_t} \log p(\mathbf{x}_t) + \nabla_{\mathbf{x}_t} \log p(y \mid \mathbf{x}_t) \\ &= -\frac{1}{\sqrt{1-\bar{\alpha}_t}} \left( \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t) - \sqrt{1-\bar{\alpha}_t} \nabla_{\mathbf{x}_t} p(y \mid \mathbf{x}_t) \right).\end{aligned}$$

This yields a modified predictor:

$$\bar{\boldsymbol{\epsilon}}_\theta(\mathbf{x}_t, t, y) = \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t) - \sqrt{1-\bar{\alpha}_t} \nabla_{\mathbf{x}_t} p(y \mid \mathbf{x}_t).$$

Introducing a scaling parameter $w$ for guidance strength gives:

$$\bar{\boldsymbol{\epsilon}}_\theta(\mathbf{x}_t, t, y) = \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t) - \sqrt{1-\bar{\alpha}_t} w \nabla_{\mathbf{x}_t} p(y \mid \mathbf{x}_t). \tag{1}$$

$$\approx \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t) - \sqrt{1-\bar{\alpha}_t} w \nabla_{\mathbf{x}_t} f_\phi(y \mid \mathbf{x}_t, t). \tag{2}$$

where the classifier $f_\phi(y \mid \mathbf{x}_t, t)$ can be used to approximate $p(y \mid \mathbf{x}_t)$.
This approach emphasizes regions where the classifier assigns high confidence to the correct label, enhancing class-specific fidelity by adjusting guidance strength.

#### 1.1.2 Classifier-Free Guided Diffusion

Classifier-Free Guided Diffusion (CFG) simplifies the training process by omitting the need for a separate classifier. Instead, guidance is incorporated directly into the diffusion model. The term $\nabla_{\mathbf{x}_t} \log p(y \mid \mathbf{x}t)$ can be expanded as:

$$\begin{aligned}\nabla_{\mathbf{x}_t} \log p(y \mid \mathbf{x}_t) &= \nabla_{\mathbf{x}_t} \log p(\mathbf{x}_t \mid y) - \nabla_{\mathbf{x}_t} \log p(\mathbf{x}_t) \\ &= -\frac{1}{\sqrt{1-\bar{\alpha}_t}} (\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t, y) - \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t))\end{aligned}$$

where $p_\theta(\mathbf{x}|y)$ is parameterized through $\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t, y)$. Substituting this result into Equation 1 yields:

$$\begin{aligned}\bar{\boldsymbol{\epsilon}}_\theta(\mathbf{x}_t, t, y) &= \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t) + w(\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t, y) - \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)) \\ &= w\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t, y) - (w-1)\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)\end{aligned} \tag{3}$$

Although slightly differently from Ho and Salimans [2022], the resulting formulation is functionally equivalent. As noted by the authors, applying classifier guidance with weight $w + 1$ to an unconditional model yields the same result as applying guidance with weight $w$ to a conditional model.
Implementation details for both methods can be seen in Appendix A.

### 1.2 Experiments

We evaluate our models using the Fréchet Inception Distance (FID) [Heusel et al., 2018] and Inception Score (IS) [Salimans et al., 2016], which measure the diversity and perceptual quality of generated images. Due to computational constraints, metrics were calculated using 1,000 samples instead of the typical 50,000. For IS, a classifier trained on MNIST (98.4% test accuracy) was used.
Another approach for comparing models is through their likelihood, a key metric for assessing generative models' performance and their ability to capture the underlying data distribution. However, computing likelihoods for diffusion models is challenging due to the intractability of marginalizing over all intermediate latent variables $x_1, x_2, \ldots, x_T$. To overcome this, researchers typically optimize the Evidence Lower Bound (ELBO) or adopt simplified objectives Ho et al. [2020]. While informative, these approximations do not fully capture the true likelihood, leading to discrepancies between likelihood-based scores and perceptual image quality. Furthermore, improved likelihood scores do not always correlate with better image quality Theis et al. [2016]. For these reasons, we do not include likelihood as a metric.

#### 1.2.1 Results

Table 1 summarizes the performance of the baseline and guided diffusion models at various guidance scales $w$.

| **Model** | **FID (↓)** | **IS (↑)** |
|---|---|---|
| Baseline | 22.81 | 7.94 |
| Classifier-Free / Classifier-Guided | | |
| $w = 0$ | 19.94 / 20.26 | 9.12 / 9.07 |
| $w = 0.1$ | 20.03 / 21.13 | 9.35 / 9.32 |
| $w = 0.2$ | 20.02 / 20.33 | 9.53 / 9.49 |
| $w = 0.3$ | 19.63 / 22.02 | 9.69 / 9.49 |
| $w = 0.4$ | 19.17 / 21.34 | 9.68 / 9.49 |
| $w = 0.5$ | 19.91 / 21.25 | 9.81 / 9.57 |
| $w = 0.6$ | 19.22 / 20.65 | 9.82 / 9.57 |
| $w = 0.7$ | 18.68 / 20.95 | 9.89 / 9.62 |
| $w = 0.8$ | **18.63** / 20.73 | 9.92 / 9.63 |
| $w = 0.9$ | 19.81 / 21.70 | 9.93 / 9.61 |
| $w = 1.0$ | 19.52 / 21.52 | 9.96 / 9.60 |
| $w = 2.0$ | 21.78 / 22.16 | **9.99** / 9.74 |
| $w = 3.0$ | 25.80 / 20.61 | **9.99** / 9.75 |
| $w = 4.0$ | 29.76 / 23.75 | 9.98 / 9.75 |

Table 1: FID and IS results for the baseline and guided diffusion models. Best results are highlighted in bold.

The results demonstrate that CFG generally outperforms CDG, consistent with Ho and Salimans [2022]. Remarkably, the unguided setting ($w = 0$) achieves FID performance comparable to the best classifier-free model and surpasses all CDG configurations.
Following Dhariwal and Nichol [2021] and Ho and Salimans [2022], we find that increasing the guidance scale ($w$) controllably enhances IS for both models. The relationships between IS, FID, and $w$ are shown in Fig. 1.

We observe no significant trade-off in terms of IS or FID for either model. Increasing the guidance scale, $w$, consistently improves the IS, which is theoretically capped at 10 for the MNIST dataset. Samples generated for digits 1 through 9, including unconditional baseline samples, are shown in Appendix C. These samples demonstrate that both guidance strategies enhance performance compared to the baseline and purely conditional models, which often produce ambiguous digits. However, neither guidance strategy nor scale seems to consistently generate high quality digits.

The CFG approach generally achieves greater consistency than CDG, aligning with observed results. CFG also benefits from a simpler training process, and follows optimization directions distinct from classifier gradients, making it less vulnerable to adversarial attacks [Ho and Salimans, 2022]. However, a drawback of CFG is slower inference due to two forward passes through the diffusion model. In contrast, CDG leverages smaller, computationally faster classifiers.

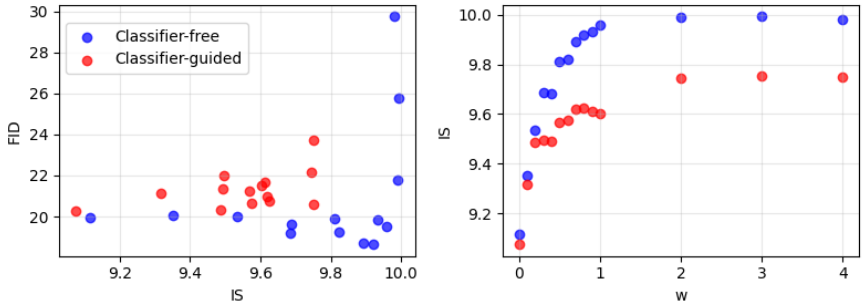



Figure 1: IS/FID and w/IS curves for both guidance strategies.

## 2 Function fitting with Constraints

### 2.1 Introduction to the task

We are tasked with generating 20 datasets following the functional form $y_i = g(x_i) + \epsilon$, where $\epsilon \sim \mathcal{N}(0, 0.01)$ and $x_i = \frac{i-1}{\ell-1}$ for $i = 1, \ldots, \ell$ with $\ell = 30$. The training set consists of 20 data points, with the remaining 10 points designated as test data. Our objective is to fit this data using a probabilistic model $p(y, \theta|X)$ based on Gaussian Processes (GPs). Specifically, the posterior distribution is formulated as the product of the likelihood, $p(y|\theta, X)$, and the prior $p(\theta)$.

### 2.2 Construction of $p(y, \theta|X)$ and $p(\theta)$

To gain insights into an appropriate kernel structure, we simulated a single instance of the training data to analyze its general characteristics. The resulting visualization is presented in Fig. 14. From this figure, we observed a distinct cyclic pattern, which informed our selection of the periodic kernel. The periodic kernel is defined as follows:

$$K_p(x, x') = \sigma_P^2 \exp\left(-\frac{2\sin^2\left(\frac{\pi\|x-x'\|}{P}\right)}{\ell_p^2}\right),$$

where $\sigma_P^2$ is the signal variance, $P$ represents the period, and $\ell_p$ is the length scale of the kernel.

Additionally, the data exhibits an overall upward trend, which could suggest the inclusion of a linear kernel. However, towards the end of the data, a sharp decline is observed, which may be better captured by a polynomial or radial basis function (RBF) kernel. Due to its flexibility and capacity to model non-linear variations effectively, we opted for the RBF kernel, defined as:

$$K_r(x, x') = \sigma_r^2 \exp\left(-\frac{\|x - x'\|^2}{2\ell_r^2}\right),$$

where $\sigma_r^2$ denotes the signal variance, and $\ell_r$ is the length scale.

Given the observed characteristics of the data, we hypothesized that an additive combination of the periodic and RBF kernels would effectively capture both the cyclic pattern and the overall trend. Therefore, the final kernel function was chosen as:

$$K(x, x') = \sigma_P^2 \exp\left(-\frac{2\sin^2\left(\frac{\pi\|x-x'\|}{P}\right)}{\ell_p^2}\right) + \sigma_r^2 \exp\left(-\frac{\|x - x'\|^2}{2\ell_r^2}\right).$$

This combined kernel accounts for both the cyclic nature and the nonlinear trends present in the data.

We instantiated the kernel using model parameters $\theta = (\sigma^2_{\text{noise}}, \sigma_P^2, P, \ell_p, \sigma_r^2, \ell_r) \in \mathbb{R}^6$, obtained from the prior distribution $p(\theta)$. Defining meaningful priors over the kernel parameters is challenging, particularly for noise variance. As the data does not appear overly noisy but could vary due to chance, we assumed $\sigma^2_{\text{noise}} \sim \mathcal{U}(0.005, 0.05)$, implying that the noise associated with each observation is within $\pm 0.141$ to $\pm 0.447$ with 95% probability.

We observe approximately six cycles within the interval $[0, 1]$, suggesting a period of roughly $\frac{1}{6} \approx 0.167$. To accommodate variation around this estimate, we set a tight prior $P \sim \mathcal{U}(0.15, 0.20)$.

For the remaining parameters, we assumed $\mathcal{U}(0.001, 2.0)$, in order to reflect minimal assumptions given the complexity of the kernel's joint interactions. Since the parameters are assumed to be independent, the joint prior distribution factorizes as:

$$p(\theta) = p(\sigma^2_{noise})p(\sigma^2_p)p(P)p(\ell_p)p(\sigma^2_r)p(\ell_r) \tag{4}$$

The complete probabilistic model is constructed as:

$$p(\theta, y|X) = p(y|X, \theta)p(\theta) \tag{5}$$

$$p(y|X, \theta) = \mathcal{N}(0, K + \sigma^2_{noise}I) \tag{6}$$

### 2.3 MAP approach

For the Maximum A Posteriori (MAP) estimation, a grid search was performed over the specified prior ranges. Each interval was discretized into 5 evenly spaced points, resulting in $6^5 = 7,776$ distinct parameter configurations. For each training dataset, the parameter set that maximized the posterior likelihood was selected. The results were analyzed by examining the worst, average, and best fits, with a focus on the posterior distribution, $\mathcal{N}(\mu_{\text{test}}, \sigma^2_{\text{test}})$, and its ability to predict test points while quantifying uncertainty.

The findings, illustrated in Figure 15 in the appendix, indicate the following:

- **Worst Case:** The model successfully captured most test points within the uncertainty band of $\pm 1.95$ standard deviations, despite deviations in the fit.
- **Average Case:** The predictions closely aligned with the test data, with only a few points lying in the outer uncertainty band.
- **Best Case:** The predictions almost perfectly overlapped with the ground truth, demonstrating the model's ability to capture the underlying data distribution in an ideal scenario.

These results highlight the effectiveness of the MAP approach in different scenarios and its robustness in quantifying uncertainty across varying levels of fit quality.

### 2.4 MCMC approach

We initialized our Markov Chain Monte Carlo (MCMC) procedure with 100 warm-up steps, drawing a total of 500 posterior samples uniformly distributed across 10 independent chains. To compute the approximate posterior predictive likelihood for the test data, we averaged the log-likelihood of the test observations under the posterior samples, as follows:

$$\log p(y_{\text{test}}|x_{\text{test}}, \mathcal{D}) \approx \frac{1}{n}\sum_{i=1}^{n} \log p(y_{\text{test}}|x_{\text{test}}, \theta^{(i)}, \mathcal{D})$$

where $n$ denotes the total number of posterior samples.

The quality of MCMC sampling was evaluated by aggregating results across 20 independent runs. Summary statistics are presented in Appendix Table 2, complemented by trace plots (Fig. 16) and posterior marginal distributions (Fig. 17). Convergence diagnostics indicate strong mixing for most parameters, with Gelman-Rubin statistics ($\hat{R}$) close to the theoretical optimum of 1.0, averaging approximately $\hat{R} = 1.03$. Effective sample sizes for these parameters are robust, further supporting the adequacy of sampling. However, significant convergence issues were observed for the noise variance parameter ($\sigma^2_{\text{noise}}$). Specifically, the Gelman-Rubin diagnostic for this parameter was $\hat{R} = 1.41$, indicating poor mixing across chains. Furthermore, the effective sample size for $\sigma^2_{\text{noise}}$ is notably low, with only 8.6% of samples in the bulk distribution being effective samples—suggesting that the chains are not adequately exploring the target posterior distribution. Given the complex dependencies in the model structure, these convergence issues for $\sigma^2_{\text{noise}}$ cast doubt on the reliability of the joint posterior samples.

### 2.5 Comparison between approaches

As shown in Fig. 2, the MAP for our GP significantly outperforms MCMC on this particular dataset in terms of posterior likelihood. MCMC struggles to decipher between signal and noise, consistently overestimating the noise parameter, likely due to the limited training dataset size of 20 points in combination with a relatively large test set. This overestimation creates a cascade effect, with the high noise values leading to systematically lower likelihoods across all experiments.

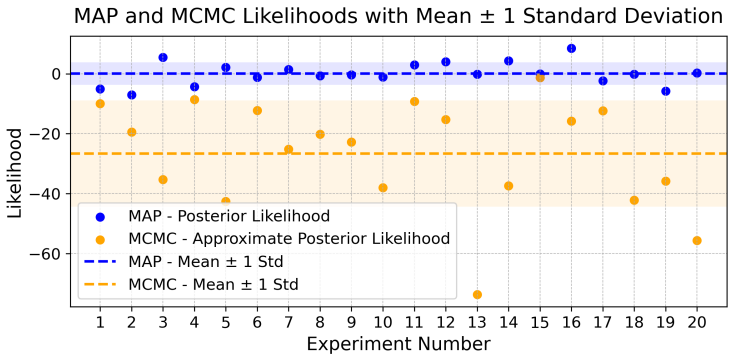



Figure 2: Comparison between the parameter estimates between MAP and MCMC quantified in terms posterior and approximate posterior likelihood

## 3 Learning with Integral Constraints

### 3.1 Derivation of $(\hat{q}, f) \mid X$

We have

$$q \approx \hat{q} = \sum_{i=1}^{\ell} w_i f(x_i), \quad w_i = \begin{cases} \frac{1}{2\ell-2}, & i \in \{1, \ell\} \\ \frac{1}{\ell-1}, & \text{otherwise.} \end{cases}$$

This can be rewritten as the linear transformation $\hat{q} = \mathbf{w}^T \mathbf{f}$, where $\mathbf{f} = [f(x_1), \ldots, f(x_\ell)]^T$ and $\mathbf{w} = [w_1, \ldots, w_\ell]^T$. Since $f \sim \mathcal{GP}(0, k(\cdot, \cdot))$, at grid points $X = \{x_1, \ldots, x_\ell\}$, $\mathbf{f} \mid \mathbf{X} \sim \mathcal{N}(\mathbf{0}, \mathbf{K})$, where $\mathbf{K}_{ij} = k(x_i, x_j)$.

For a linear transformation $\mathbf{Ag}$ where $\mathbf{g} \sim \mathcal{N}(\mathbf{m}, \mathbf{C})$, it holds that $\mathbf{Ag} \sim \mathcal{N}(\mathbf{Am}, \mathbf{ACA}^T)$. Thus, $\hat{q} \mid X \sim \mathcal{N}(0, \mathbf{w}^T \mathbf{K} \mathbf{w})$, and

$$\begin{bmatrix} \mathbf{f} \\ \hat{q} \end{bmatrix} \mid X \sim \mathcal{N}\left( \begin{bmatrix} \mathbf{0} \\ 0 \end{bmatrix}, \begin{bmatrix} \mathbf{K} & \mathbf{Kw} \\ \mathbf{w}^T \mathbf{K} & \mathbf{w}^T \mathbf{Kw} \end{bmatrix} \right).$$

### 3.2 Derivation of $f \mid X, \hat{q}$

For

$$\begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \end{bmatrix} \mid X \sim \mathcal{N}\left( \begin{bmatrix} \boldsymbol{\mu}_1 \\ \boldsymbol{\mu}_2 \end{bmatrix}, \begin{bmatrix} \boldsymbol{\Sigma}_{11} & \boldsymbol{\Sigma}_{12} \\ \boldsymbol{\Sigma}_{21} & \boldsymbol{\Sigma}_{22} \end{bmatrix} \right),$$

it follows that $\mathbf{x}_1 \mid \mathbf{x}_2 = \mathbf{a} \sim \mathcal{N}(\overline{\boldsymbol{\mu}}, \overline{\boldsymbol{\Sigma}})$, where

$$\overline{\boldsymbol{\mu}} = \boldsymbol{\mu}_1 + \boldsymbol{\Sigma}_{12} \boldsymbol{\Sigma}_{22}^{-1} (\mathbf{a} - \boldsymbol{\mu}_2), \quad \overline{\boldsymbol{\Sigma}} = \boldsymbol{\Sigma}_{11} - \boldsymbol{\Sigma}_{12} \boldsymbol{\Sigma}_{22}^{-1} \boldsymbol{\Sigma}_{21}.$$

Applying this we have $\mathbf{f} \mid X, \hat{q} \sim \mathcal{N}(\overline{\boldsymbol{\mu}}, \overline{\boldsymbol{\Sigma}})$ where

$$\overline{\boldsymbol{\mu}} = \mathbf{Kw}(\mathbf{w}^T \mathbf{Kw})^{-1} \hat{q}, \quad \overline{\boldsymbol{\Sigma}} = \mathbf{K} - \mathbf{Kw}(\mathbf{w}^T \mathbf{Kw})^{-1} \mathbf{w}^T \mathbf{K}.$$

The matrix $\mathbf{K}$ is determined by the kernel function $k(\cdot, \cdot)$. For $k$ to be a universal kernel, it must be positive definite for any set of distinct inputs $X = \{x_1, x_2, \ldots, x_\ell\}$. This implies that $\mathbf{K}$ is positive definite and thus has full rank as long as the points $x_i$ are distinct, ensuring $\mathbf{w}^\top \mathbf{Kw} > 0$.

The term $\mathbf{Kw}(\mathbf{w}^\top \mathbf{Kw})^{-1} \mathbf{w}^\top \mathbf{K}$ is a rank-1 matrix, as it is the outer product of $\mathbf{Kw}$ and itself scaled by $(\mathbf{w}^\top \mathbf{Kw})^{-1}$. Since $\mathbf{Kw}$ lies in the column space of $\mathbf{K}$, subtracting this term reduces the rank of the covariance matrix $\overline{\boldsymbol{\Sigma}}$ by 1, resulting in $\text{rank}(\overline{\boldsymbol{\Sigma}}) = \ell - 1$.

Intuitively, when conditioning on the integral approximation $\hat{q}$, we have imposed a linear constraint on $\mathbf{f}$. This constraint effectively removes one degree of freedom, resulting in a covariance matrix with rank $l - 1$.

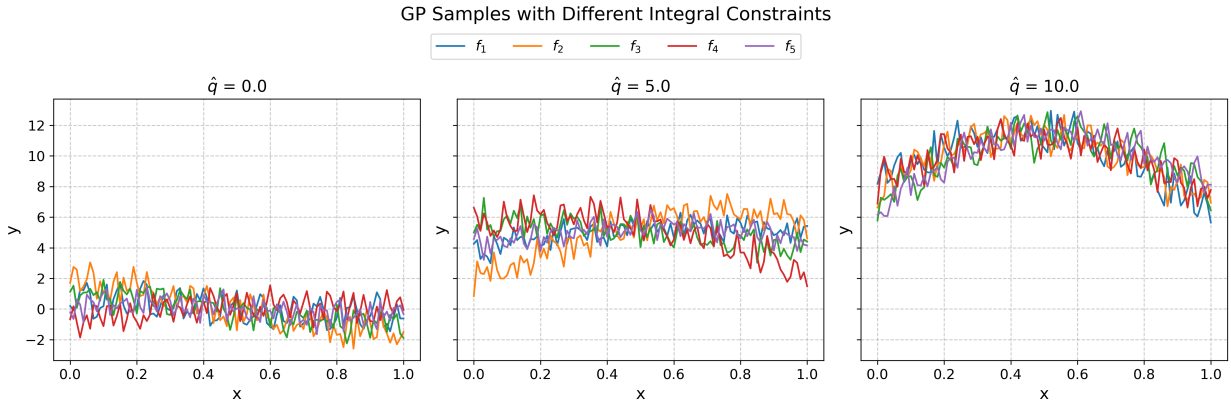



Figure 3: Plot of five samples from $f \mid X, \hat{q}$ for $\ell = 101$ and $\hat{q} \in \{0, 5, 10\}$

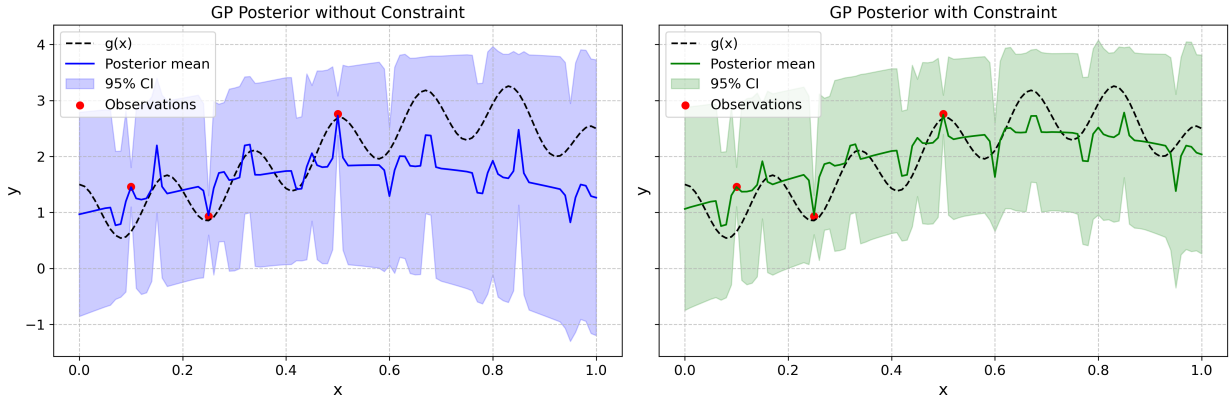



Figure 4: Plot of $f \mid \hat{q}, \mathcal{D}$ and $f \mid \mathcal{D}$ for $\ell = 101$ and $\hat{q} = 2$

### 3.3 Visualization and comparison of samples from $f \mid X, \hat{q}$

We draw 5 samples from $f_i \sim f \mid X, \hat{q}$ for each $\hat{q} \in \{0, 5, 10\}$, where each sample satisfies the constraint of integrating to $\hat{q}$ over our discretization points $X$ using the trapezoidal rule. The samples can be seen in Fig. 3. The integral constraint has two visible effects on our samples. First, we observe a positive correlation between the magnitude of $\hat{q}$ and $f_i(x_i)$ values. Secondly, we notice that the functional form becomes increasingly non-linear as the magnitude of $\hat{q}$ increases. This phenomenon is essentially explained by the fact that when the kernel is less globally constrained, it has more freedom to express its natural covariance structure.

### 3.4 Visualization and comparison of posteriors $f \mid \hat{q}, \mathcal{D}$ and $f \mid \mathcal{D}$

We now examine the posterior distributions conditioned on a sparse training dataset $\mathcal{D}$ consisting of 3 observations, comparing the cases with and without the integral constraint $\hat{q} = 2$, shown in Fig. 4 . The sparsity of $\mathcal{D}$ creates a challenging inference scenario. We observe the GP posterior with the integral constraint $\hat{q} = 2$ achieves notably better alignment with the ground truth generator $g$ compared to the unconstrained posterior. This improved performance stems from the fact that $g$ itself integrates to 2, making the integral constraint an informative prior that effectively reduces the space of possible functions to those satisfying this global property. The constraint thus serves as a powerful regularizer, incorporating known global behavior of $g$ that cannot be inferred from the sparse local observations alone.

## References

Prafulla Dhariwal and Alex Nichol. Diffusion models beat gans on image synthesis, 2021. URL https://arxiv.org/abs/2105.05233.

Jonathan Ho and Tim Salimans. Classifier-free diffusion guidance, 2022. URL https://arxiv.org/abs/2207.12598.

Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium, 2018. URL https://arxiv.org/abs/1706.08500.

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

Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models, 2020. URL https://arxiv.org/abs/2006.11239.

Lucas Theis, Aäron van den Oord, and Matthias Bethge. A note on the evaluation of generative models, 2016. URL https://arxiv.org/abs/1511.01844.

## A Implementation details

Instead of training a separate unconditional diffusion model, we will parameterize both $\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t, y)$ and $\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)$ through a single neural network where $\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t) = \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t, y = \varnothing)$ as in Ho and Salimans [2022]. We do this by embedding $y$ and adding this to the timestep embedding in the existing U-Net. We train the unconditional and conditional model by randomly setting $\mathbf{y}$ to $\varnothing$ with probability $p_{uncond} = 0.1$.
In the derivations of classifier-guided diffusion we assumed an unconditional diffusion model to get the same result as Dhariwal and Nichol [2021]. However, as the authors note, it is also possible to use a conditional diffusion model instead. In fact Dhariwal and Nichol [2021] get the best results using a conditional model. We will therefore use the same network for both classifier-guided and classifier-free diffusion for a fair comparison between the methods. We train our model with $T = 1000$ for 100 epochs with a learning rate of $1e^{-3}$ and a batch size of 256. The rest of the parameters are identical to the template. During sampling we will if a classifier is supplied use eq. 2. If not we will use eq. 3 with $w + 1$ due to the use a conditional model for the classifier-guided approach.
The classifier used is trained on noisy images with the same noise schedule as the diffusion model. The model consist of the existing U-Net with an added linear layer. We use the same parameters for training this classifier as we did with the diffusion model for simplicity. Our classifier achieves an accuracy of 97.1% on a set of withheld noisy images.
The main changes to the code template can be seen in Appendix B.

## B Code changes

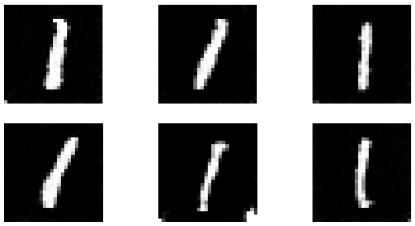
The changes to code is highlighted with yellow. In addition to the highlighted code parts of the code have been refactored slightly for better readability.

```
class ScoreNet(nn.Module):
    def __init__(
        self,
        marginal_prob_std,
        channels=[32, 64, 128, 256],
        embed_dim=256,
        num_classes=None
    ):
        ...
        self.num_classes = num_classes
        if self.num_classes is not None:
            self.c_embed = nn.Embedding(num_classes, embed_dim)

    def forward(self, x, t, c=None):
        embed = self.act(self.t_embed(t))

        if self.num_classes is not None and c is not None:
            embed = embed + self.c_embed(c)
```

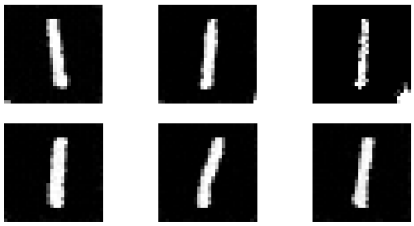
```
        # Encoding path
        ...

        # Decoding path
        ...

        # Normalize output
        ...


class ClassifierFreeDDPM(nn.Module):
    def __init__(
        self, network: nn.Module, T: int, beta_1: float, beta_T: float, drop_prob: float
    ):

        super(ClassifierFreeDDPM, self).__init__()

        self.drop_prob = drop_prob

        self._network = network
        self.network = lambda x, t, c: (
            self._network(x.reshape(-1, 1, 28, 28), (t.squeeze() / T), c)
        ).reshape(-1, 28 * 28)

        ...

    def forward(self, x0: torch.Tensor, c: torch.Tensor) -> torch.Tensor:

        t = torch.randint(1, self.T, (x0.shape[0], 1)).to(x0.device)

        epsilon = torch.randn_like(x0)

        mean = torch.sqrt(self.alpha_bar[t]) * x0
        std = torch.sqrt(1 - self.alpha_bar[t])
        xt = mean + std * epsilon

        if random.random() < self.drop_prob:
            c = None

        return nn.MSELoss()(epsilon, self.network(xt, t, c))

    def reverse_diffusion(
        self, xt: torch.Tensor, t: torch.Tensor, noise: torch.Tensor, eps: torch.Tensor
    ) -> torch.Tensor:

        mean = (
            1.0
            / torch.sqrt(self.alpha[t])
            * (xt - (self.beta[t]) / torch.sqrt(1 - self.alpha_bar[t]) * eps)
        )
        std = torch.where(
            t > 0,
            torch.sqrt(
                ((1 - self.alpha_bar[t - 1]) / (1 - self.alpha_bar[t])) * self.beta[t]
            ),
            0,
        )

        return mean + std * noise
```

```
    def get_gradient(
        self, xt: torch.Tensor, t: torch.Tensor, c: torch.Tensor, classifier: nn.Module
    ) -> torch.Tensor:
        with torch.enable_grad():
            x_in = xt.clone().requires_grad_(True)

            logits = classifier(x_in.reshape(-1, 1, 28, 28), t.squeeze() / self.T)
            log_probs = F.log_softmax(logits, dim=-1)

            selected_logits = log_probs[range(len(c)), c]

            gradient = torch.autograd.grad(selected_logits.sum(), x_in)[0]

        return gradient

    @torch.no_grad()
    def sample(
        self,
        shape: tuple,
        w: float,
        c: torch.Tensor,
        classifier: nn.Module | None = None,
    ) -> torch.Tensor:
        xT = torch.randn(shape).to(self.beta.device)

        xt = xT
        for i in range(self.T, 0, -1):
            # Convert 0 to a tensor of zeros when i=1
            noise = torch.randn_like(xT) if i > 1 else torch.zeros_like(xT)
            t = torch.tensor(i).expand(xt.shape[0], 1).to(self.beta.device)

            if classifier is not None:
                eps = self.network(xt, t, c) + w * self.get_gradient(xt, t, c, classifier) * torch.sqrt(1 - sel
            else:
                eps = (1 + w) * self.network(xt, t, c) - w * self.network(xt, t, None)

            xt = self.reverse_diffusion(xt, t, noise, eps)

        return xt

...
```

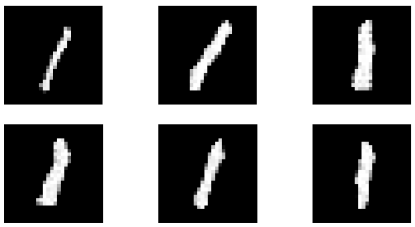
## C Visualization of samples



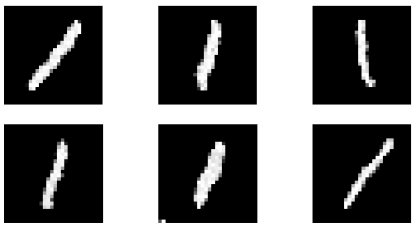
(a) Classifier-Free ($w = 0.8$)



(b) Classifier-Free ($w = 3$)

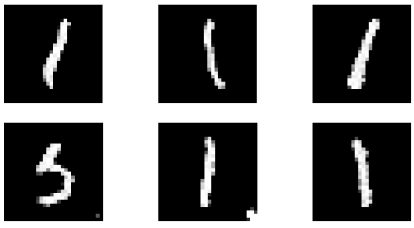


(c) Classifier-Guided ($w = 0.8$)



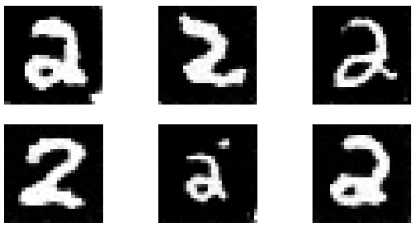
(d) Classifier-Guided ($w = 3$)

(e) Baseline



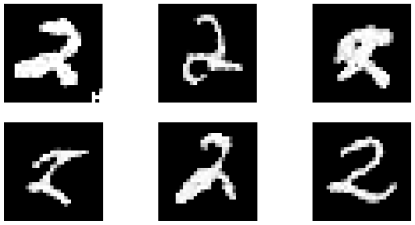
(f) Conditional ($w = 0$)

Figure 5: Comparison of different generations for digit 1

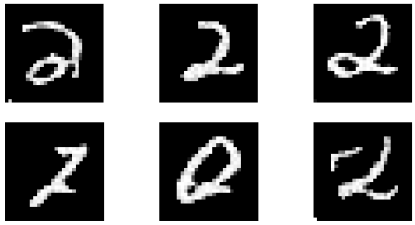


(a) Classifier-Free ($w = 0.8$)

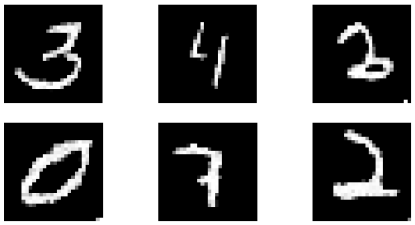
(b) Classifier-Free ($w = 3$)



(c) Classifier-Guided ($w = 0.8$)



(d) Classifier-Guided ($w = 3$)

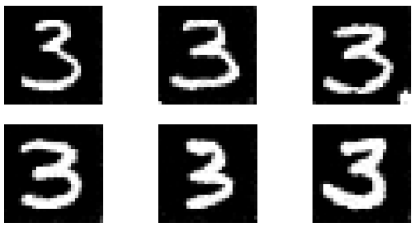


(e) Baseline

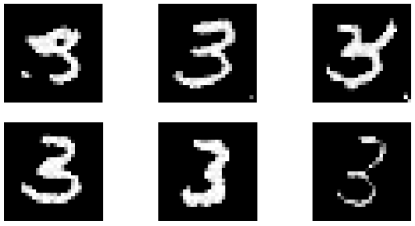
(f) Conditional ($w = 0$)

Figure 6: Comparison of different generations for digit 2

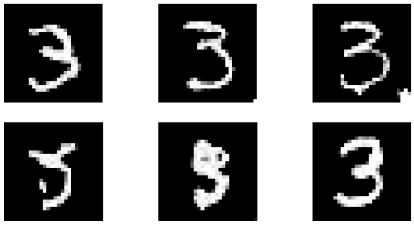
(a) Classifier-Free ($w = 0.8$)



(b) Classifier-Free ($w = 3$)

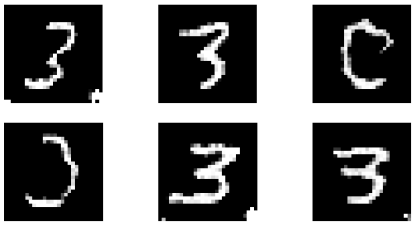


(c) Classifier-Guided ($w = 0.8$)



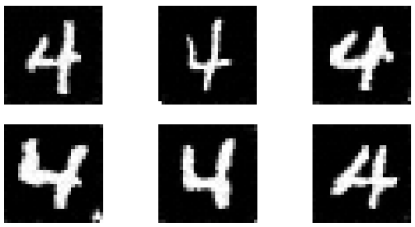
(d) Classifier-Guided ($w = 3$)

(e) Baseline

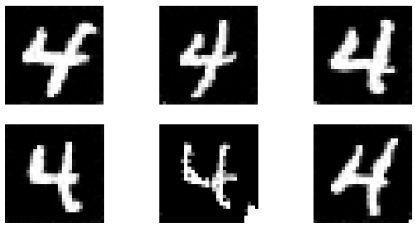


(f) Conditional ($w = 0$)

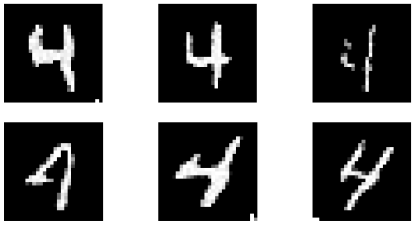
Figure 7: Comparison of different generations for digit 3



(a) Classifier-Free ($w = 0.8$)

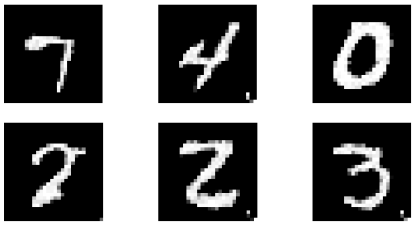


(b) Classifier-Free ($w = 3$)

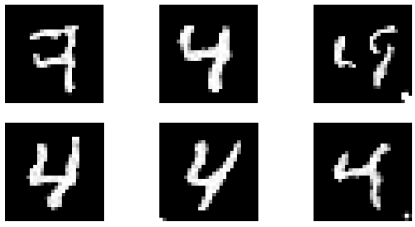


(c) Classifier-Guided ($w = 0.8$)

(d) Classifier-Guided ($w = 3$)

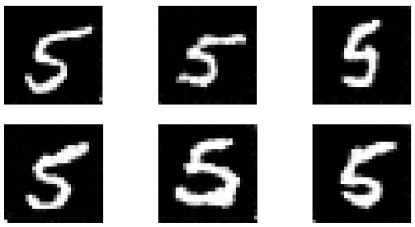


(e) Baseline

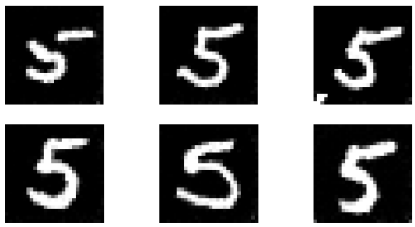


(f) Conditional ($w = 0$)

Figure 8: Comparison of different generations for digit 4



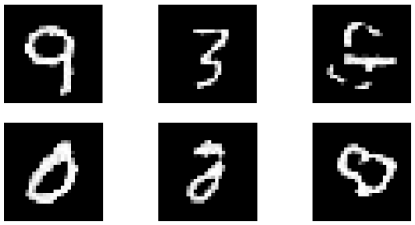
(a) Classifier-Free ($w = 0.8$)



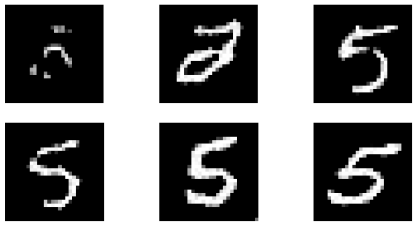
(b) Classifier-Free ($w = 3$)

(c) Classifier-Guided ($w = 0.8$)

(d) Classifier-Guided ($w = 3$)



(e) Baseline



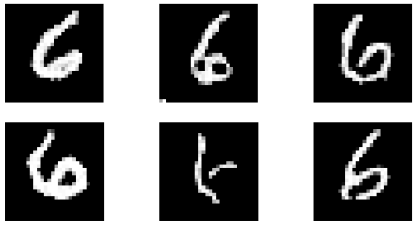
(f) Conditional ($w = 0$)

Figure 9: Comparison of different generations for digit 5

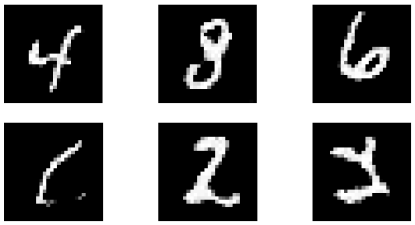
(a) Classifier-Free ($w = 0.8$)

(b) Classifier-Free ($w = 3$)

(c) Classifier-Guided ($w = 0.8$)



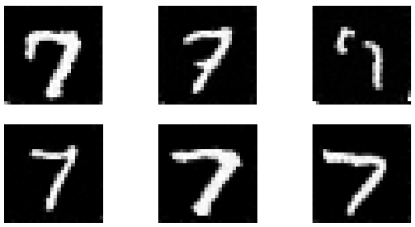
(d) Classifier-Guided ($w = 3$)



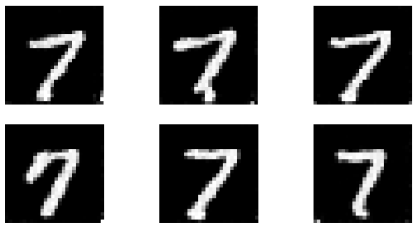
(e) Baseline

(f) Conditional ($w = 0$)

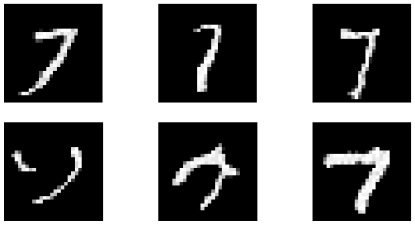
Figure 10: Comparison of different generations for digit 6



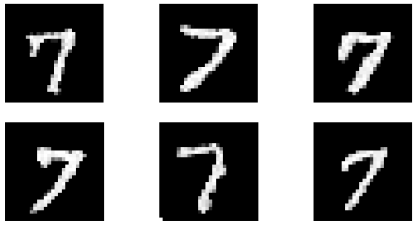
(a) Classifier-Free ($w = 0.8$)



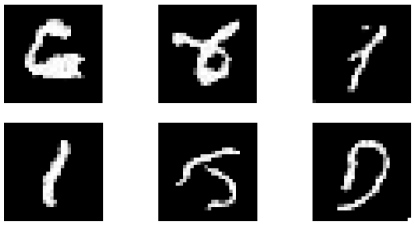
(b) Classifier-Free ($w = 3$)



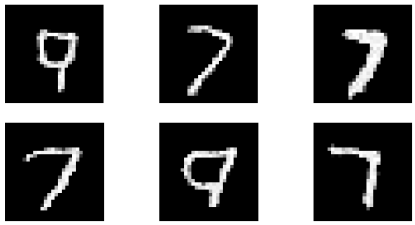
(c) Classifier-Guided ($w = 0.8$)



(d) Classifier-Guided ($w = 3$)

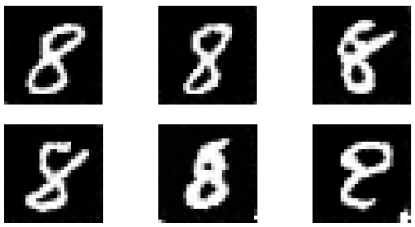


(e) Baseline

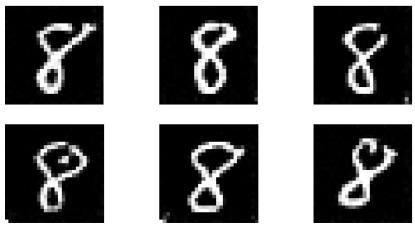


(f) Conditional ($w = 0$)

Figure 11: Comparison of different generations for digit 7



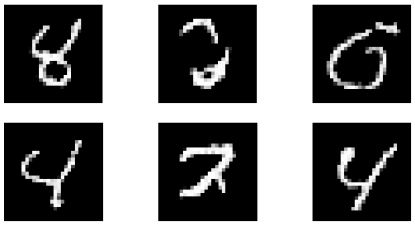
(a) Classifier-Free ($w = 0.8$)



(b) Classifier-Free ($w = 3$)

(c) Classifier-Guided ($w = 0.8$)

(d) Classifier-Guided ($w = 3$)



(e) Baseline

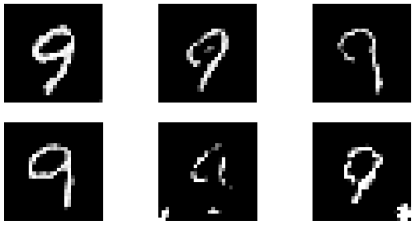
(f) Conditional ($w = 0$)

Figure 12: Comparison of different generations for digit 8

(a) Classifier-Free ($w = 0.8$)



(b) Classifier-Free ($w = 3$)



(c) Classifier-Guided ($w = 0.8$)

(d) Classifier-Guided ($w = 3$)

(e) Baseline

(f) Conditional ($w = 0$)

Figure 13: Comparison of different generations for digit 9

# D B.1

| Parameter | Mean | SD | HDI 3% | HDI 97% | MCSE Mean | MCSE SD | ESS Bulk | ESS Tail | $\hat{R}$ |
|---|---|---|---|---|---|---|---|---|---|
| Lengthscale | 1.007 | 0.576 | 0.089 | 1.962 | 0.005 | 0.004 | 13102.0 | 7057.0 | 1.03 |
| Noise Variance | 0.040 | 0.012 | 0.014 | 0.050 | 0.001 | 0.000 | 430.0 | 1150.0 | 1.46 |
| Period | 0.175 | 0.014 | 0.152 | 0.199 | 0.000 | 0.000 | 15177.0 | 8116.0 | 1.03 |
| Periodic Amplitude | 0.994 | 0.574 | 0.035 | 1.910 | 0.005 | 0.003 | 13904.0 | 8084.0 | 1.03 |
| RBF Lengthscale | 1.002 | 0.572 | 0.093 | 1.959 | 0.004 | 0.003 | 15444.0 | 8530.0 | 1.03 |
| RBF Variance | 0.996 | 0.587 | 0.011 | 1.890 | 0.005 | 0.004 | 13967.0 | 7623.0 | 1.03 |

Table 2: Summary statistics for the parameters.

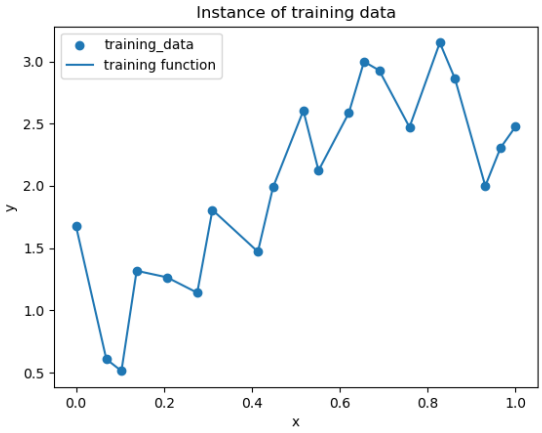
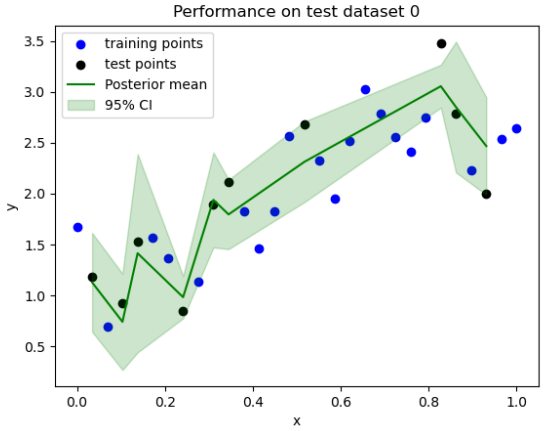
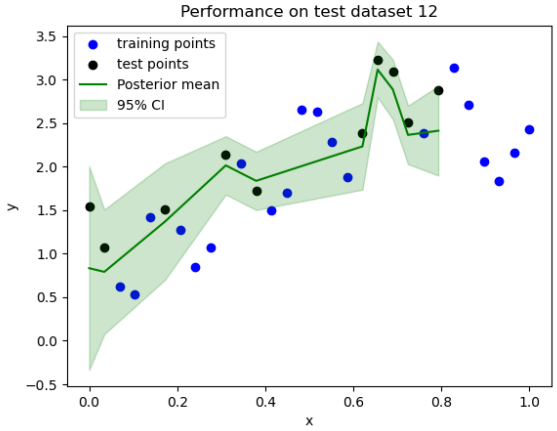


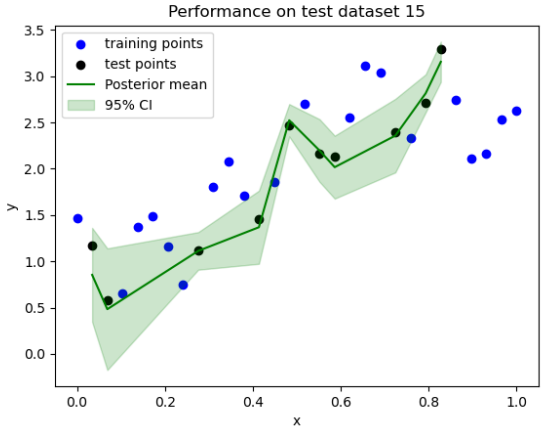

Figure 14: Random training instance




(a) Worst case




(b) Average case




(c) Best case

Figure 15: Illustration of sample quality for MAP approach

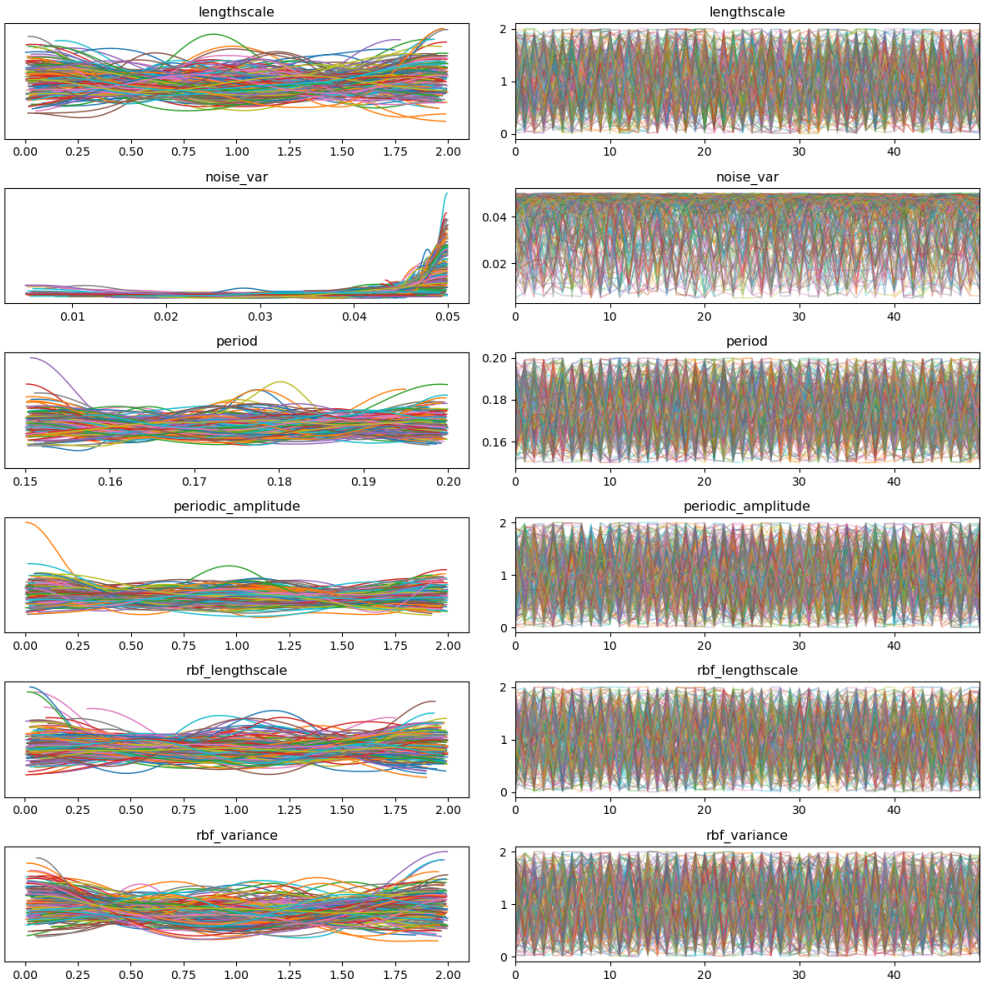



Figure 16: Illustration of how the traces mix across all runs and chains

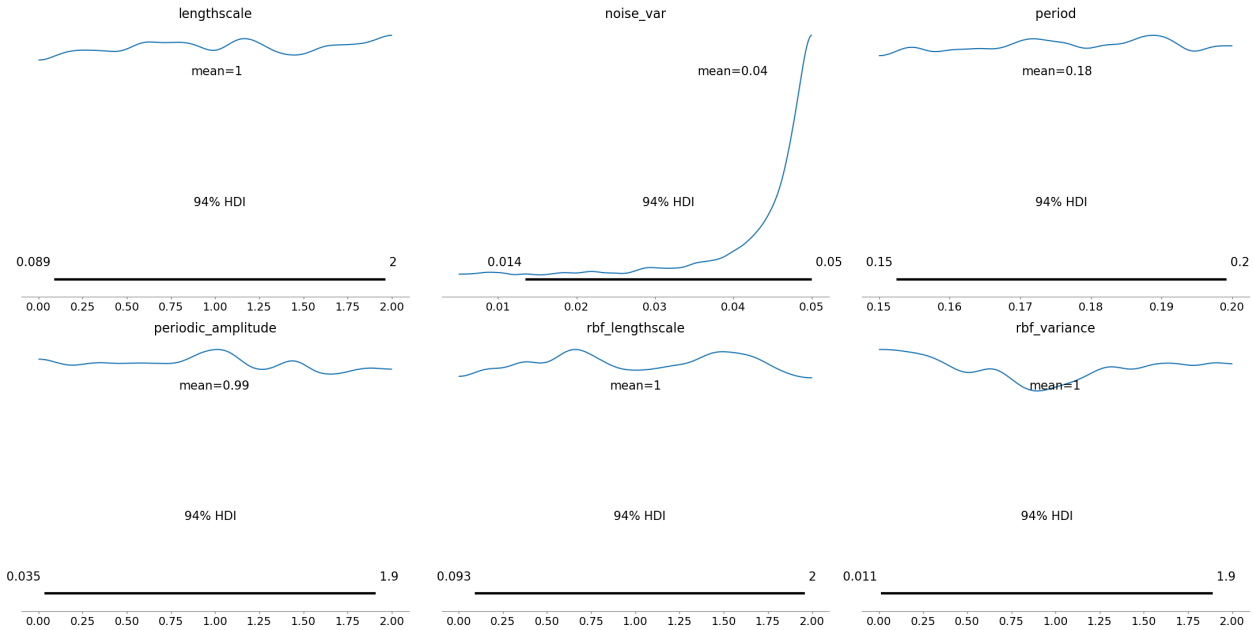



Figure 17: Approximation of the posterior marginal distributions