Bayesian Numerical Integration with Neural Networks (Supplementary Material)

Katharina Ott[1, 2] **Michael Tiemann**[1] **Philipp Hennig**[2, 3] **François-Xavier Briol**[4, 5]

[1]Bosch Center for Artificial Intelligence, Renningen, Germany
[2]University of Tübingen, Tübingen, Germany
[3]MPI for Intelligent Systems, Tübingen, Germany
[4]Department of Statistical Science, University College London, London, United Kingdom
[5]The Alan Turing Institute, London, United Kingdom

1 EXPERIMENTAL DETAILS

Below we give implementation details for all the datasets used and provide additional experimental results.

For our implementation of the BSN and the experiments described in the main text we make use of the following packages: `PyTorch` [Paszke et al., 2017], `emukit` [Paleyes et al., 2019], `GPyTorch` [Gardner et al., 2018], `laplace-torch` [Daxberger et al., 2021], `PyWake`[Pedersen et al., 2019], and `Matplotlib` [Hunter, 2007].

1.1 IMPACT OF ARCHITECTURE DESIGN

We provide additional discussion concerning the choice of activation function, choice of sampling strategy and choice of optimizer.

1.1.1 Choice of Optimizer

We compare different optimizers for the BSN and a standard neural network, where for the standard neural network we use the same architecture as for u_{θ_u}. We use the 1-dimensional wind farm dataset with $n = 320$ data points. We choose this dataset, due to the complicated structure of the score function of a mixture of Gaussians. For the experiment we consider three optimizers, i.e., Adam [Kingma and Ba, 2015], L-BFGS [Liu and Nocedal, 1989] and the Hessian-free optimizer [Martens, 2010]. For Adam, we use mini-batching with a batch size of 32 and full-batch training. For the Hessian-free optimizer and L-BFGS we only consider full-batch training. For Adam, we use 10000 iterations, for the Hessian-free optimizer 1000 iterations, and for L-BFGS we use automatic stopping based on the strong Wolfe conditions. We compare the loss for all training methods. We also use CELU and RELU activation functions, where RELU is included as it is the standard activation function for neural networks.

Training a standard neural network with RELUs work significantly better, than using CELUs both in terms of the loss reached at the end of training and in terms of runtime (see Figure 1 and Figure 2). Using RELUs does not work for the BSN, as the gradients of u_{θ_u} lead to discontinuities.

Training of the BSN using Adam is considerably slower than the training progress of the standard neural network. We find that for CELU activation function, using (approximate) second order methods leads to a large improvement both in terms of speed and loss. The success of the second order methods might be due to a narrow loss landscape, i.e., a larger spread in the eigenvalue spectrum of the curvature. Therefore, we also examine the condition number of the Hessian, and we find that the BSN has a slightly higher condition number than the standard neural network (we do not report the condition number for RELUs as it cannot be computed numerically). Given its short runtime and good optimization results, we choose L-BFGS for all our experiments.

Accepted for the 39th Conference on Uncertainty in Artificial Intelligence (UAI 2023).

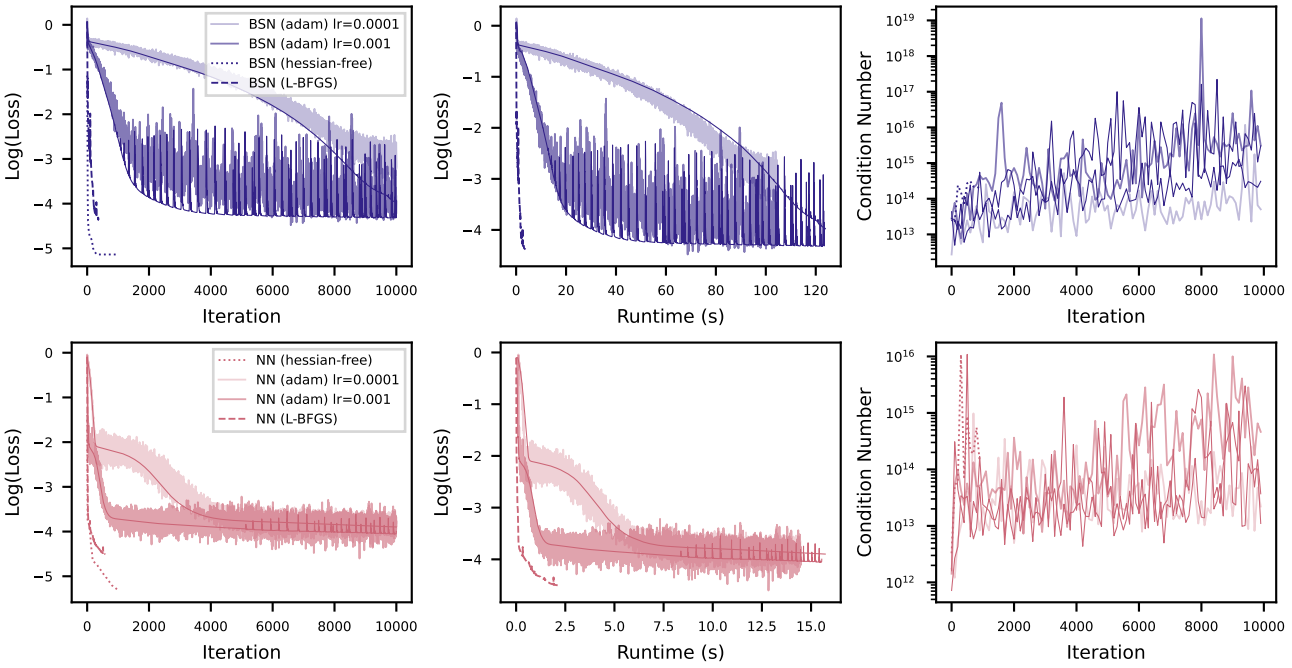

Figure 1: Regression performance of a plain neural network (red) and a BSN (blue) using CELU activations. Loss (*left*), and condition number (*left*) as a function of the iteration. *Centre:* loss as a function of the runtime. Thin dark lines correspond to training with full-batch Adam. Runtime of the Hessian-free optimizer not plotted, due to its long runtime.

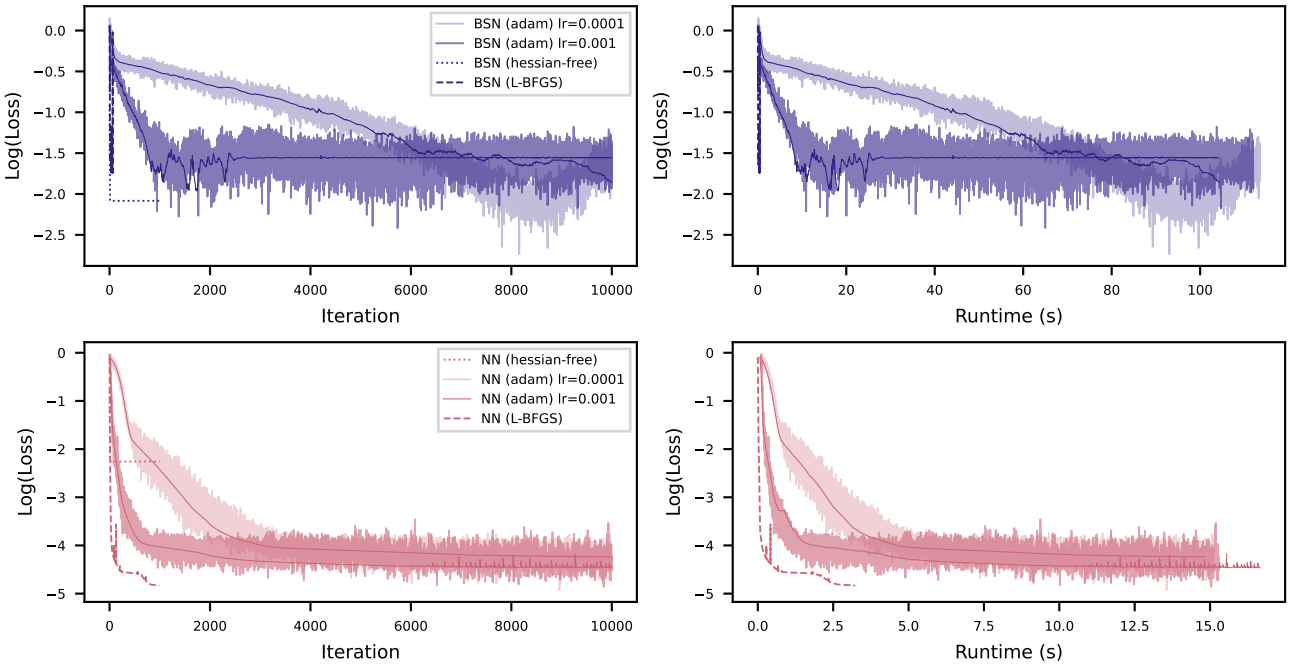

Figure 2: Regression performance of a plain neural network (red) and a BSN (blue) using RELU activations. Loss (*left*) as a function of the iteration. *Centre:* loss as a function of the runtime. Thin dark lines correspond to training with full-batch Adam. Runtime of the Hessian-free optimizer not plotted, due to its long runtime.

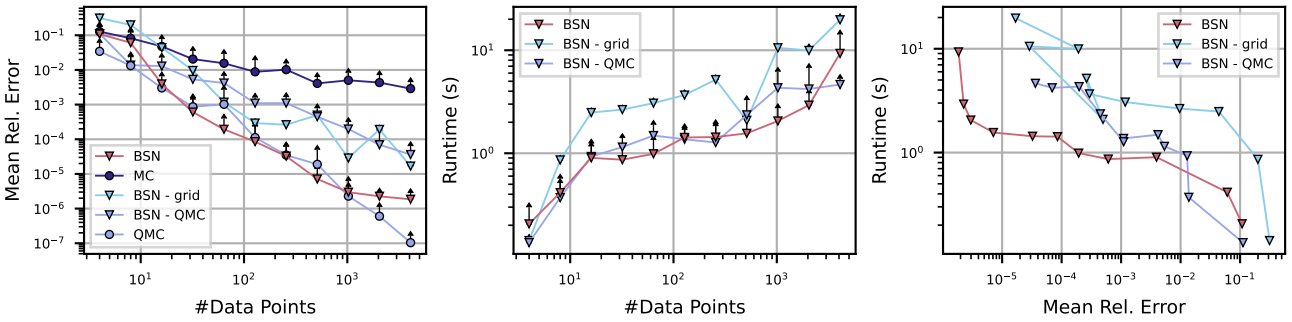

Figure 3: Comparing different sampling schemes on the continuous Genz dataset in $d = 1$. The BSN is trained on MC-sampled points, QMC-sampled points and on a regular grid. Mean relative integration error (*left*), and runtime (*centre*), (based on 5 repetitions) as a function of n. *Right:* Run time in seconds as a function of mean relative integration error.

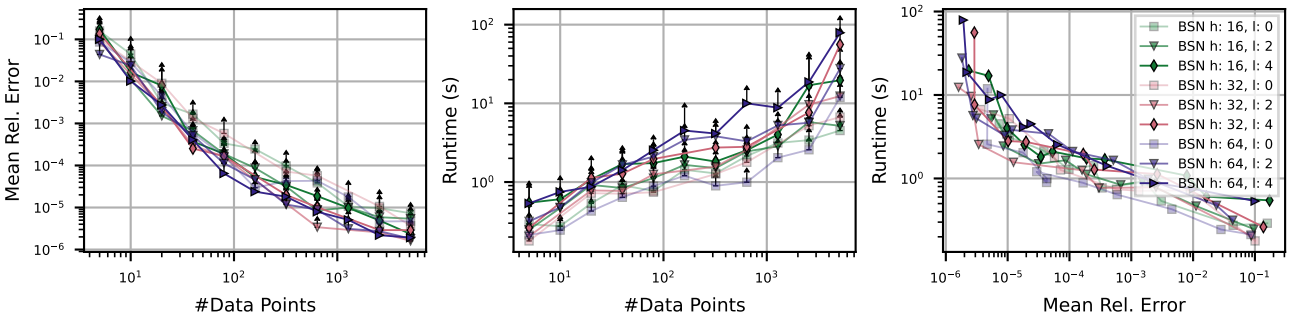

Figure 4: Testing different architectures on the 1-dimensional continuous Genz dataset. Mean relative integration error (*left*), and run time (*centre*), (based on 5 repetitions) as a function of n. *Right:* Run time in seconds as a function of mean relative integration error.

1.1.2 Sampling Strategies

For our experiments in the main text, we choose the data points by sampling from π, i.e., $x_i \sim \pi$. Here we consider two additional sampling strategies:

- Using a quasi-Monte Carlo (QMC) sequence. We use `SciPy`'s [Virtanen et al., 2020] implementation of QMC based on the Sobol sequence [Sobol', 1967].
- Linearly spaced points in a hypercube (called *grid* in Figure 3). Here we consider the hypercube $[-5\sigma_\pi, 5\sigma_\pi]^d$, where $\pi(x) = \mathcal{N}(x|0, \sigma_\pi)$.

Figure 3 shows the result of the different sampling strategies in $d = 1$. The BSN performs better using MC samples then using QMC samples and grid points. The low performance of the latter is expected, since too few points a placed in regions with a high probability mass.

1.1.3 Choice of Architecture

We consider a basic architecture of the following form:

$$u_{\theta_u} = \text{Linear}(d, h) \circ CELU(\circ\text{Linear}(h, h) \circ CELU)^l \circ \text{Linear}(h, d),$$

where h are the number of hidden units and l are the number of hidden layers. Figure 4 shows the performance of different architectures on the 1-dimensional continuous Genz dataset. All architectures perform similar but the architecture with $l = 2$ and $h = 32$ reaches the lowest error the fastest for large n. Hence, we use this architecture for our experiments.

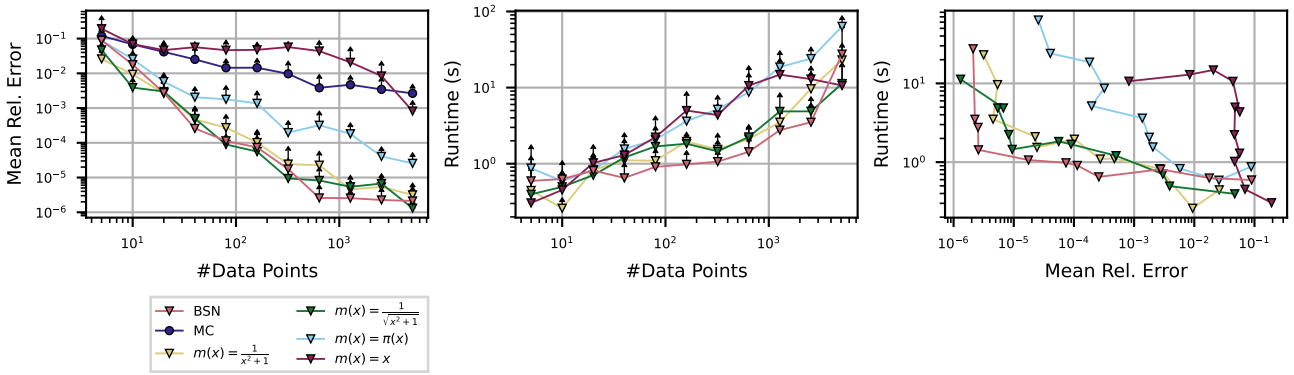

Figure 5: *Continuous Genz dataset in* $d = 1$ *with different* $m(x)$. Mean relative integration error (*left*), run time (*center*) (based on 5 repetitions) as a function of n. *Right:* Run time in seconds as a function of mean relative integration error.

1.1.4 Choice of $m(x)$

For most of our experiments we set $m(x) = I_d$ in Equation (3). This might not necessarily be the best choice for a given task, but finding a function m that works well is hard. We test different m on the 1-dimensional Continuous Genz function:

- $m(x) = \frac{I_d}{||x||_2^2+1}$ - $m(x)$ goes to zero for $x \to \pm\infty$
- $m(x) = \frac{I_d}{\sqrt{||x||_2^2+1}}$ - $m(x)$ goes to zero for $x \to \pm\infty$ and cancels the $\nabla_x \log \pi(x)$ term for large x.
- $m(x) = I_d \pi(x)$ - in cases where π is a normal distribution, this function also goes to zero for $x \to \pm\infty$.
- $m(x) = \text{diag}x$ - example of a function having negative effect.

The results of comparing these different m are shown in Figure 5. On this test problem, none of the proposed m significantly outperforms the choice $m(x) = I_d$, with some performing significantly worse.

1.1.5 Choice of GP Kernel

As a benchmark we use BQ with an RBF kernel for all our experiments. The reason for this choice of kernel is the closed form availability of posterior mean and covariance when π is a normal distribution. Here we add an experiment using a Matern 1/2 kernel. For this choice of kernel the posterior mean is only available in $d = 1$, hence we conduct the experiment on the 1 dimensional Genz dataset (see Section 2.3 for the expression of the kernel mean embedding). The corresponding results a found in Figure 6. Once again, we do not observe a significant difference in performance, except for the continuous Genz dataset.

1.2 GENZ BENCHMARK

In our experiments we use the Genz integrand family dataset. Here we include a short description of each dataset, plus additional experiments on the 2-dimensional version of each dataset. In our experiments we integrate the Genz function against a standard normal $\pi(x) = \mathcal{N}(x|0, 1)$. This requires the transformation of the inputs to the original Genz functions f, which are to be integrated against $[0, 1]^d$. Therefore, we compute $\Pi_\pi[f \circ c]$ where $c(x) = \frac{1}{2}\left(1 + \text{erf}\left(\frac{x}{\sqrt{2}}\right)\right)$ is the cumulative density function of the standard normal. We give the form of f below.

Continuous Genz dataset The integrand is given by

$$f(x) = \exp\left(-\sum_{k=1}^{d} a_k |x_k - u_k|\right)$$

with parameters $a_k = 1.3$ and $u_k = 0.55$. See Figure 7 for results on a 2 dimensional version of this dataset.

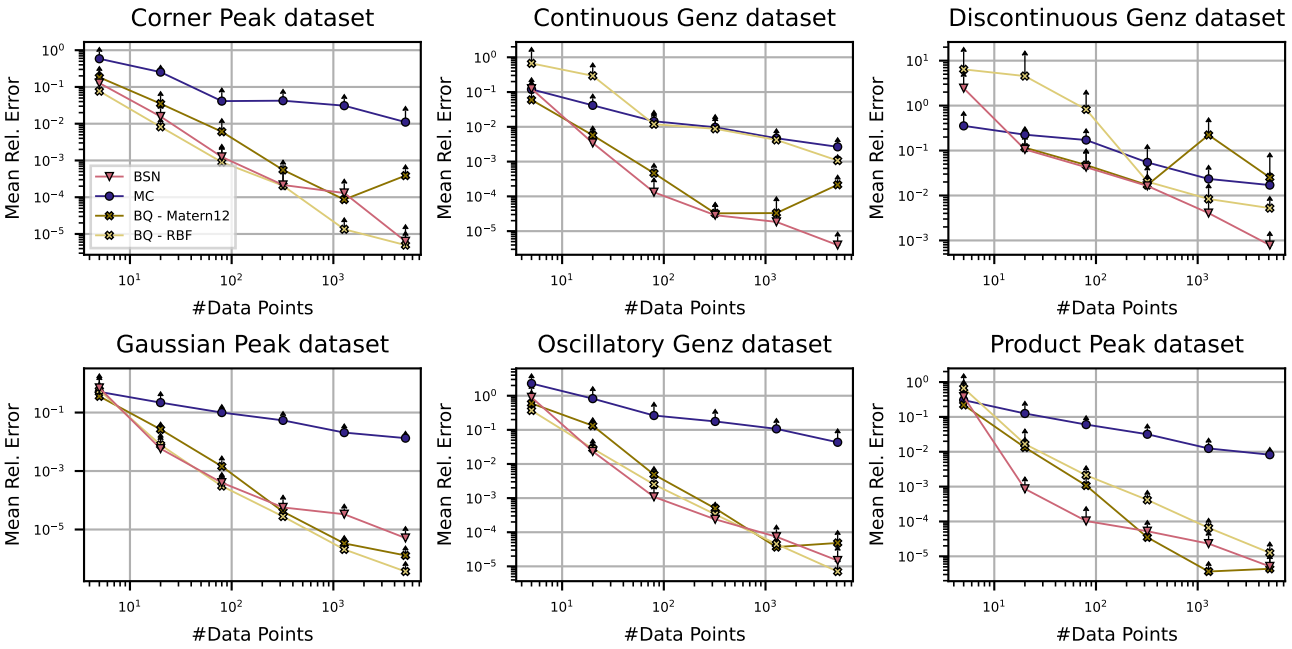

Figure 6: *BQ with Matern 1/2 kernel on the Genz family in* $d = 1$. Mean relative integration error (based on 5 repetitions) as a function of n.

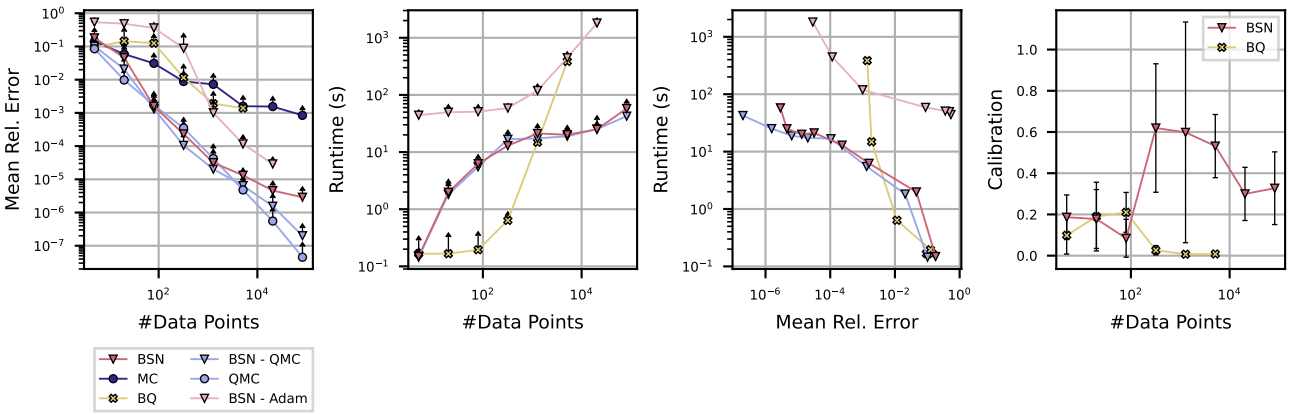

Figure 7: *Continuous Genz dataset in* $d = 2$. Mean relative integration error (*left*), run time (*centre-left*), and calibration (*right*) (based on 5 repetitions) as a function of n. *Center-right:* Run time in seconds as a function of mean relative integration error.

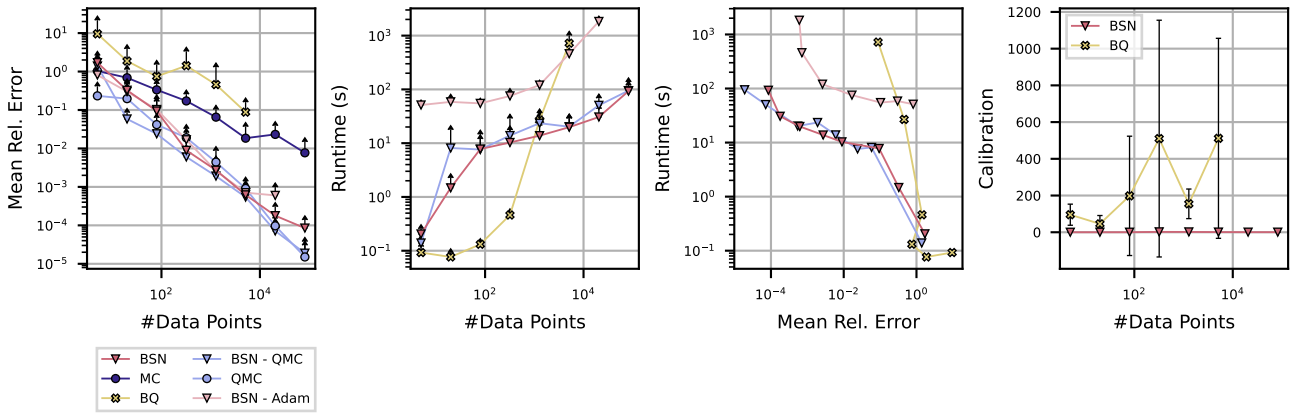

Figure 8: *Corner Peak dataset in* $d = 2$. Mean relative integration error (*left*), run time (*centre-left*), and calibration (*right*) (based on 5 repetitions) as a function of n. *Center-right:* Run time in seconds as a function of mean relative integration error.

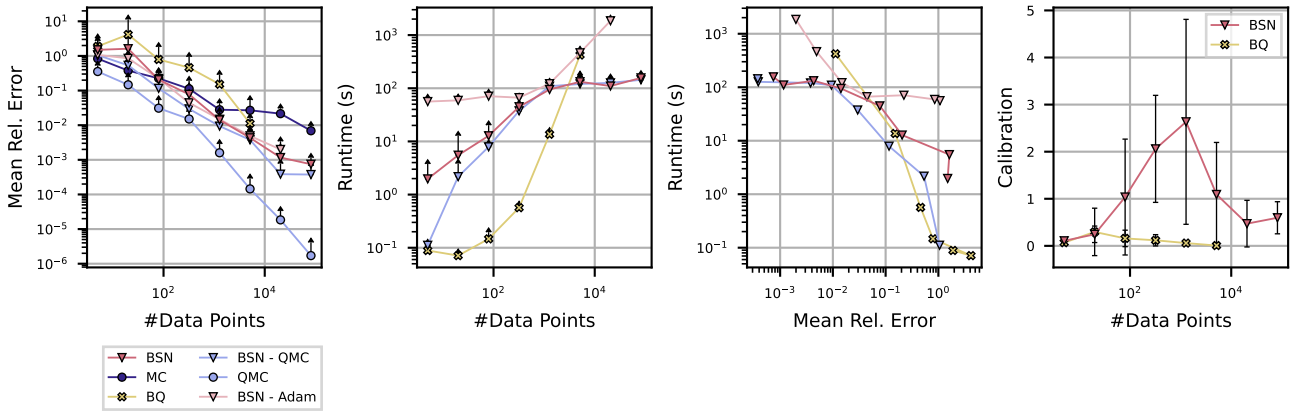

Figure 9: *Discontinuous Genz dataset in* $d = 2$. Mean relative integration error (*left*), run time (*centre-left*), and calibration (*right*) (based on 5 repetitions) as a function of n. *Center-right:* Run time in seconds as a function of mean relative integration error.

Corner Peak dataset The integrand is given by

$$f(x) = \left(1 + \sum_{k=1}^{d} a_k x_k\right)^{-(d+1)}$$

with parameters $a_k = 5$. See Figure 8 for results on a 2 dimensional version of this dataset.

Discontinuous Genz dataset The integrand is given by

$$f(x) = \begin{cases} 0, & \text{if } x_k > u_k \text{ for any k} \\ \exp\left(\sum_{k=1}^{d} a_k x_k\right) \end{cases}$$

with parameters $a_k = 5$ and $u_k = 0.5$. See Figure 9 for results on a 2 dimensional version of this dataset.

Gaussian peak dataset The integrand is given by

$$f(x) = \exp\left(-\sum_{k=1}^{d} a_k^2 (x_k - u_k)^2\right)$$

with parameters $a_k = 5$ and $u_k = 0.5$ See Figure 10 for results on a 2 dimensional version of this dataset.

Product peak dataset The integrand is given by

$$f(x) = \prod_{k=1}^{d} \frac{1}{\left(a_k^{-2} + (x_k - u_k)^2\right)}$$

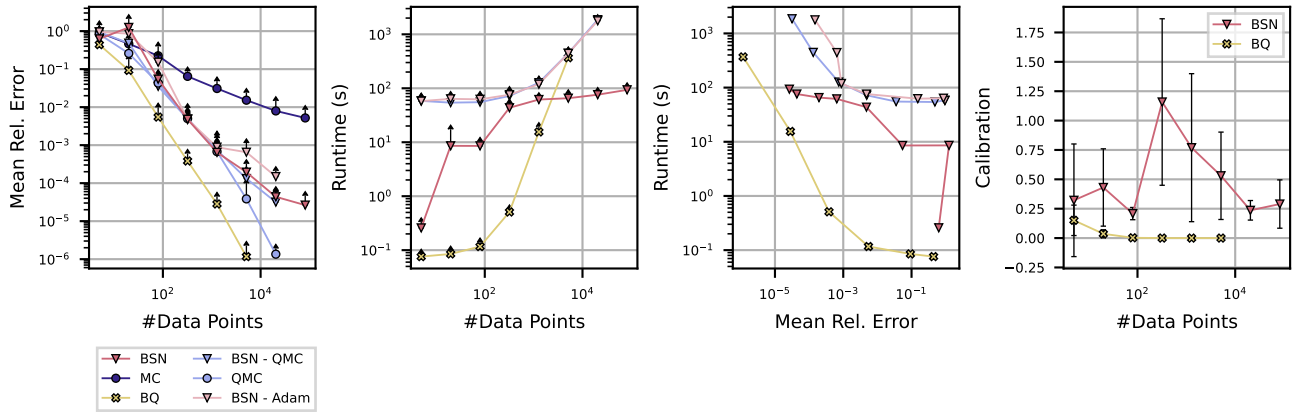

Figure 10: *Gaussian peak dataset in* $d = 2$. Mean relative integration error (*left*), run time (*centre-left*), and calibration (*right*) (based on 5 repetitions) as a function of n. *Center-right:* Run time in seconds as a function of mean relative integration error.

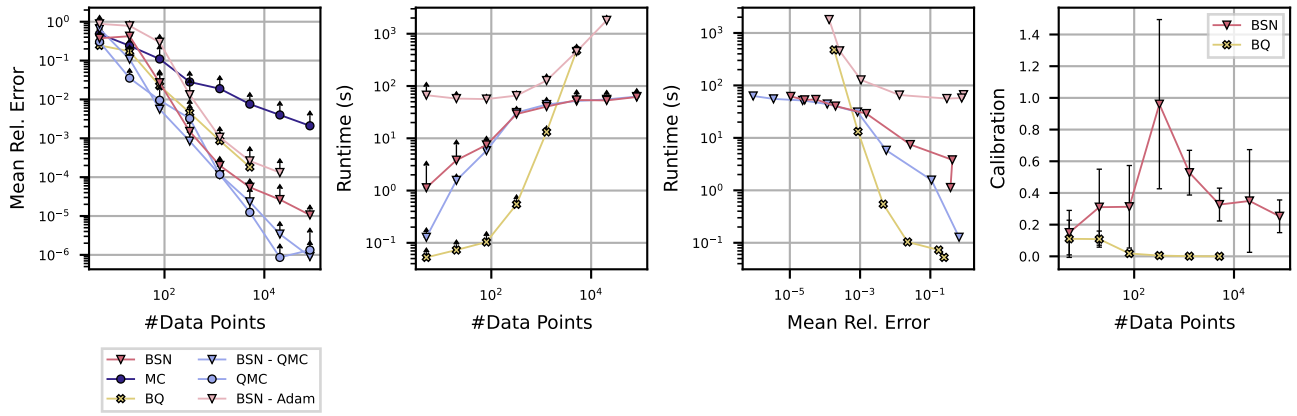

Figure 11: *Product peak dataset in* $d = 2$. Mean relative integration error (*left*), run time (*centre-left*), and calibration (*right*) (based on 5 repetitions) as a function of n. *Center-right:* Run time in seconds as a function of mean relative integration error.

with parameters $a_k = 5$ and $u_k = 0.5$ See Figure 11 for results on a 2 dimensional version of this dataset.

Oscillatory Genz dataset The integrand is given by

$$f(x) = \cos\left(2\pi u + \sum_{k+1}^{d} a_k x_k\right)$$

with parameters $a_k = 5$ and $u = 0.5$ See Figure 12 for results on a 2 dimensional version of this dataset.

1.3 GOODWIN OSCILLATOR

Goodwin oscillator [Goodwin, 1965] describes how the feedback loop between mRNA transcription and protein expression can lead to oscillatory dynamics in a cell. We here consider the case with no intermediate protein species. The experimental setup is based on earlier work by [Riabiz et al., 2022, Chen et al., 2019, Calderhead and Girolami, 2009, Oates et al., 2016].

The Goodwin oscillator with no intermediate protein species is given by:

$$\frac{du_1}{dt} = \frac{a_1}{1+a_2 u_2^{\rho}} - \alpha u_1$$
$$\frac{du_2}{dt} = k_1 u_1 - \alpha u_2,$$

where u_1 corresponds to the concentration of mRNA and u_2 to the concentration of the corresponding protein product. We set $\rho = 10$.

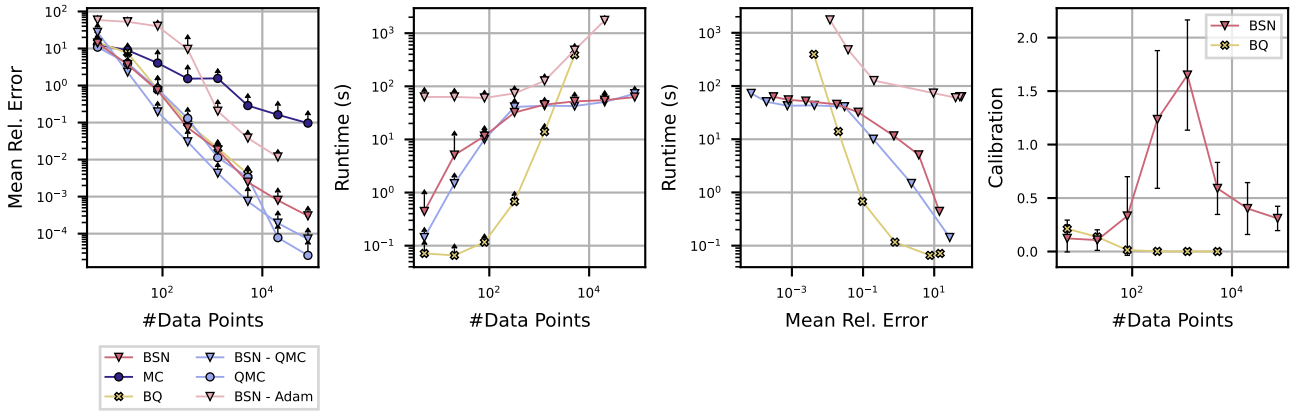

Figure 12: *Oscillatory Genz dataset in* $d = 2$. Mean relative integration error (*left*), run time (*centre-left*), and calibration (*right*) (based on 5 repetitions) as a function of n. *Center-right:* Run time in seconds as a function of mean relative integration error.

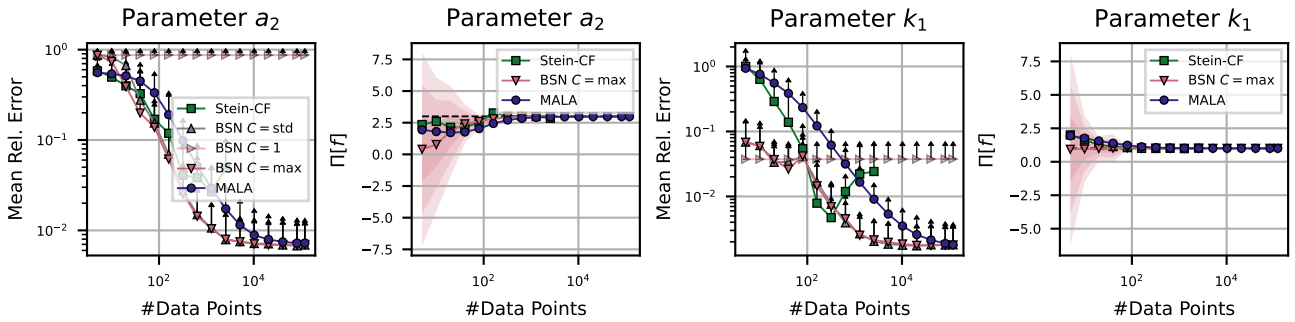

Figure 13: *Posterior expectations for the parameters of a Goodwin ODE*. Mean relative integration error (*left* and *centre-right*), and uncertainty estimates (*centre-left* and *right*) (based on 5 repetitions) as a function of n.

As initial conditions we set $u_0 = (0, 0)$. To generate the ground truth dataset, we set $a_1 = 1$, $a_2 = 3$, $k_1 = 1$ and $\alpha = 0.5$. We use a measurement noise of $\sigma = (0.1, 0.05)$. Data was collected for 2400 time points in $t \in [1, 25]$, leading to the following expression for the likelihood:

$$p(y|x) \propto \exp\left(-\frac{1}{2\sigma_1^2}\sum_{k=1}^{2400}||y_{1,k} - u_1(t_k)||_2^2 - \frac{1}{2\sigma_2^2}\sum_{k=1}^{2400}||y_{2,k} - u_2(t_k)||_2^2\right)$$

We use an `JAX`'s implementation of Dopri5(4) to solve the ODE. We use automatic differentiation implemented in `JAX` to compute derivatives of the likelihood with respect to the parameters. To avoid parameters becoming negative, we use log-transformed parameters $w = \log(x)$ for the parameter inference via MCMC. We place a standard normal prior on the log-transformed parameters w. For this dataset we choose MALA, based on the successful application and extensive study of this MCMC algorithm in previous work [Riabiz et al., 2022, Chen et al., 2019, Calderhead and Girolami, 2009, Oates et al., 2016]. For each dataset we run five chains, where the initial conditions for each chain are sampled from the prior. Each chain is run with a step size of $h = 0.0033$ for 500000 steps. We use thinning with a step size of 20. This results in datasets of $n = 125000$. We did not use any warm-up on this dataset to keep the choice consistent for all dataset sizes.

Figure 13 shows the results for the remaining two parameters not shown in the main text.

1.4 WIND FARM MODELLING

For the wind farm model in our experiments, we assume we have a large-scale wind farm with equally spaced turbines on a two-dimensional grid and an ambient turbulence intensity. For each turbine, we use a wake deficit model by Niayifar and Porté-Agel [2016]. We put the following distributions on parameters for the wind farm simulation

- **Turbine resistance coefficient:** Gaussian distribution with mean $\mu = 1.33$ and variance $\sigma^2 = 0.1$.

- **Coefficient describing the wake expansion:** Gaussian distribution left-truncated at 0 with mean $\mu = 0.38$ and variance $\sigma = 0.001$.
- **Second coefficient describing the wake expansion:** Gaussian distribution left-truncated at 0 with mean $\mu = 4e-3$ and variance $\sigma^2 = 1e-8$.
- **Turbulence intensity:** Gaussian distribution left-truncated at 0 with mean $\mu = 0.1$ and variance $\sigma^2 = 0.003$
- **Wind direction:** Mixture of Gaussian distributions truncated so as to have support on $[0, 45]$ with means $\mu_1 = 0, \mu_2 = 22.5, \mu_3 = 33.75$ and variances $\sigma_1^2 = 50, \sigma_2^2 = 40, \sigma_3^2 = 8$.
- **Hub heights:** Gaussian distribution left-truncated at 0 with mean $\mu = 100$ and variance $\sigma^2 = 0.5$.
- **Hub diameter:** Gaussian distribution left-truncated at 0 with mean $\mu = 100$ and variance $\sigma^2 = 0.1$.

These distributions were chosen to have scales which might realistically represent uncertainty for their input, but if applying our method in practice these would have to be elicited from wind-farm experts. Note that the BSNs could be applied to much more complex distributions so-long as the density of Π can be evaluated pointwise up to some normalization constant.

Our code is based on the code estimating the local turbine thrust coefficient Kirby et al. [2022] using a low-order wake model provided here: `https://github.com/AndrewKirby2/ctstar_statistical_model`. This code is based on the `PyWake` package [Pedersen et al., 2019].

2 BAYESIAN QUADRATURE

We now provide a short introduction to BQ and the derivation of the kernel mean embedding for truncated Gaussians.

2.1 INTRODUCTION TO BAYESIAN QUADRATURE

Recall that we are interest in approximating the integral $\Pi[f] = \int_{\mathcal{X}} f(x)\pi(x)dx$. BQ works by placing a $\mathcal{GP}(m, k)$ on f, i.e. a GP with mean function $m : \mathcal{X} \to \mathbb{R}$ and covariance functions (or kernel) $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$. Then, given observation $\{x_i, f(x_i)\}_{i=1}^n$, we can compute the posterior mean and variance on the value of $\Pi[f]$ as

$$\begin{aligned}
\mathbb{E}\left[\Pi[f]\right] &= \int_{\mathcal{X}} \left(m(x) + k(x, x_{1:n})k(x_{1:n}, x_{1:n})^{-1}(f(x_{1:n}) - m(x_{1:n}))\right)\pi(x)dx \\
&= \Pi[m] + \Pi[k(\cdot, x_{1:n})]k(x_{1:n}, x_{1:n})^{-1}(f(x_{1:n}) - m(x_{1:n})), \\
\mathbb{V}\left[\Pi[f]\right] &= \int_{\mathcal{X}}\int_{\mathcal{X}} \left(k(x, x') - k(x, x_{1:n})k(x_{1:n}, x_{1:n})^{-1}k(x_{1:n}, x')\right)\pi(x)\pi(x')dxdx' \\
&= \Pi\bar{\Pi}[k] + \Pi[k(\cdot, x_{1:n})]k(x_{1:n}, x_{1:n})^{-1}\Pi[k(x_{1:n}, \cdot)],
\end{aligned}$$

where $f(x_{1:n}) \in \mathbb{R}^n$ with $[f(x_{1:n})]_i = f(x_i)$, $m(x_{1:n}) \in \mathbb{R}^n$ with $[m(x_{1:n})]_i = m(x_i)$, $k(x_{1:n}, x)^\top = k(x, x_{1:n}) \in \mathbb{R}^n$ with $[k(x, x_{1:n})]_i = k(x, x_i)$, $k(x_{1:n}, x_{1:n}) \in \mathbb{R}^{n\times n}$ with $[k(x_{1:n}, x_{1:n})]_{ij} = k(x_i, x_j)$ for all i, j in $\{1, \ldots, n\}$. Finally, $\Pi\bar{\Pi}[k] = \int_{\mathcal{X}}\int_{\mathcal{X}} k(x, x')\pi(x)\pi(x')dxdx'$.

Clearly, the expressions above can only be used if $\Pi[k(\cdot, x)]$, called the kernel mean embedding, and $\Pi\bar{\Pi}[k]$, called the initial error, are known in closed-form. This is only possible for some combinations of distribution π and covariance function k. For example, if $\mathcal{X} = \mathbb{R}^d$, π is a Gaussian and k is the RBF-kernel, then the expressions above can be computed analytically. A more challenging case is that of truncated Gaussian distributions. In the next section, we show that the kernel mean embedding can be derived in that case.

2.2 KERNEL MEAN EMBEDDING FOR TRUNCATED GAUSSIANS

For truncated Gaussian distributions and the RBF kernel, we can compute the posterior mean but not the posterior variance. Here we consider the 1-dimensional case with $\mathcal{X} = [a, b]$ which can be extended to the d-dimensinal case for isotropic Gaussians. We provide the expression for the kernel mean embedding: $\Pi[k(\cdot, x)] = \int_{\mathcal{X}} k(x', x)\pi(x')dx'$. We consider the case when π is a truncated Gaussian and introduce the following notation:

$$\pi(x) = \frac{\phi(x, \mu, \sigma)}{\Phi(\frac{b-\mu}{\sigma}) - \Phi(\frac{a-\mu}{\sigma})}$$

where $\phi(x) = (\sqrt{2\pi}\sigma)^{-1}\exp(-(x-\mu)^2/2\sigma^2)$ and $\Phi(x) = \frac{1}{2}(1 + \text{erf}(x/\sqrt{2}))$.

We use Z to denote the normalization constant

$$Z(a,b,\mu,\sigma) = \Phi\left(\tfrac{b-\mu}{\sigma}\right) - \Phi\left(\tfrac{a-\mu}{\sigma}\right)$$

We rewrite the RBF kernel using the above identities $k(x,x') = \exp\left(-(x-x')^2/2l^2\right) = l\sqrt{2\pi}\phi(x,x',l)$. We can now express the kernel mean embedding as:

$$\Pi[k(\cdot,x)] = \int_a^b l\sqrt{2\pi}\phi(x,x',l)\tfrac{\phi(x',\mu,\sigma)}{Z(a,b,\mu,\sigma)}dx' = Cl\sqrt{2\pi}\int_a^b \tfrac{\phi(x',\tilde{\mu},\tilde{\sigma})}{Z(a,b,\mu,\sigma)}dx' = l\sqrt{2\pi}C\tfrac{Z(a,b,\tilde{\mu},\tilde{\sigma})}{Z(a,b,\mu,\sigma)},$$

where For truncated Gaussian distributions and the RBF kernel, we can compute the posterior mean but not the posterior variance. Here we consider the 1-dimensional case with $\mathcal{X} = [a,b]$ which can be extended to the d-dimensinal case for isotropic Gaussians. We provide the expression for the kernel mean embedding: $\Pi[k(\cdot,x)] = \int_{\mathcal{X}} k(x',x)\pi(x')dx'$. We consider the case when π is a truncated Gaussian and introduce the following notation:

$$\pi(x) = \tfrac{\phi(x,\mu,\sigma)}{\Phi(\frac{b-\mu}{\sigma})-\Phi(\frac{a-\mu}{\sigma})}$$

where $\phi(x) = (\sqrt{2\pi}\sigma)^{-1}\exp(-(x-\mu)^2/2\sigma^2)$ and $\Phi(x) = \frac{1}{2}(1+\text{erf}(x/\sqrt{2}))$.

We use Z to denote the normalization constant

$$Z(a,b,\mu,\sigma) = \Phi\left(\tfrac{b-\mu}{\sigma}\right) - \Phi\left(\tfrac{a-\mu}{\sigma}\right)$$

We rewrite the RBF kernel using the above identities $k(x,x') = \exp\left(-(x-x')^2/2l^2\right) = l\sqrt{2\pi}\phi(x,x',l)$. We can now express the kernel mean embedding as:

$$\Pi[k(\cdot,x)] = \int_a^b l\sqrt{2\pi}\phi(x,x',l)\tfrac{\phi(x',\mu,\sigma)}{Z(a,b,\mu,\sigma)}dx' = Cl\sqrt{2\pi}\int_a^b \tfrac{\phi(x',\tilde{\mu},\tilde{\sigma})}{Z(a,b,\mu,\sigma)}dx' = l\sqrt{2\pi}C\tfrac{Z(a,b,\tilde{\mu},\tilde{\sigma})}{Z(a,b,\mu,\sigma)},$$

where

$$\tilde{\mu} = \tfrac{\mu l^2 + x\sigma^2}{\sigma^2+l^2}, \qquad \tilde{\sigma} = \sqrt{\tfrac{\sigma^2 l^2}{\sigma^2+l^2}}, \qquad C = \tfrac{1}{\sqrt{2\pi(\sigma^2+l^2)}}\exp\left(\tfrac{(\mu-x)^2}{2(\sigma^2+l^2)}\right).$$

2.3 KERNEL MEAN EMBEDDING FOR MATERN 1/2 KERNEL

For Gaussian distributions and the Matern 1/2 kernel, we can compute the posterior mean but only in $d = 1$. We provide the expression for the kernel mean embedding: $\Pi[k(\cdot,x)] = \int_{\mathbb{R}} k(x',x)\pi(x')dx'$, where $\pi(x) = \mathcal{N}(0,1)$ is a standard normal and $k(x',x) = \exp(|x-x'|/l)$ is the Matern 1/2 kernel.

$$\Pi[k(\cdot,x)] = \tfrac{1}{2}\exp\left(\tfrac{2xl+1}{2l^2}\right)\text{erfc}\left(\tfrac{x+\frac{1}{l}}{\sqrt{2}}\right) + \tfrac{1}{2}\exp\left(\tfrac{1-2xl}{2l^2}\right)\left(\text{erf}\left(\tfrac{x-\frac{1}{l}}{\sqrt{2}}\right)+1\right)$$

3 ADDITIONAL BACKGROUND: LAPLACE APPROXIMATION

The Laplace approximation constructs a second-order Taylor approximation around the maximum of the posterior, i.e., the mode of the posterior, which amounts to a Gaussian approximate of the posterior around the MAP (maximum a-posteriori) estimate. Here we provide a detailed introduction.

We want to compute a posterior for the parameters of our model

$$\begin{aligned} p(\theta|\mathcal{D}) &= \tfrac{p(\mathcal{D}|\theta)p(\theta)}{Z}, \quad \text{where} \\ Z &= \int p(\mathcal{D}|\theta)p(\theta)d\theta. \end{aligned} \tag{1}$$

Here, the integral for the normalization constant Z is usually not tractable, and we will have to resort to some approximation to compute it. We provide the expressions for negative log prior

$$-\log p(\theta) = \tfrac{1}{2\sigma_0^2}\|\theta\|_2^2 - \tfrac{p+1}{2}\log\pi\sigma_0^2 \tag{2}$$

and the negative log likelihood

$$-\log p(\mathcal{D}|\theta) = \frac{1}{2\sigma^2}\sum_{i=1}^{n}\|f(x_i) - g_\theta(x_i)\|_2^2 - \frac{n}{2}\log \pi\sigma^2, \tag{3}$$

where σ is the dataset noise. By comparing (3) and (2) to the mean square loss with weight decay

$$\begin{aligned} l_{\text{tot}}(\theta) &= l(\theta) + \lambda\|\theta\|_2^2 \\ l(\theta) &= \frac{1}{n}\sum_{i=1}^{n}\|f(x_i) - g_\theta(x_i)\|_2^2 \end{aligned}$$

we note $l \propto -\log p(\mathcal{D}|\theta)$ and $\lambda\|\theta\|_2^2 \propto -\log p(\theta)$. Hence, the minimum of the loss correspond the maximum of the posterior, i.e. $\theta_{\text{MAP}} = \text{argmin}_\theta l_{\text{tot}}(\theta) = \text{argmin}_\theta -\log p(\mathcal{D}|\theta) - \log p(\theta) = \text{argmin}_\theta -\log p(\theta|\mathcal{D})$. We denote $L(\theta) = \log p(\mathcal{D}|\theta) + \log p(\theta)$, and rewrite Equation (1)

$$p(\theta|\mathcal{D}) = \frac{e^{L(\theta)}}{Z}.$$

To find a suitable approximation for the posterior we, we use as Taylor series expansion of L around θ_{MAP}

$$L(\theta) \approx L(\theta_{\text{MAP}}) + (\theta - \theta_{\text{MAP}})^\top \nabla_\theta L(\theta_{\text{MAP}}) + \frac{1}{2}(\theta - \theta_{\text{MAP}})^\top \nabla_\theta^2 L(\theta_{\text{MAP}})(\theta - \theta_{\text{MAP}})$$

The second term is equal to zero by definition of θ_{MAP}. Hence, we arrive at a Gaussian approximation of the posterior $q(\theta)$ for the form

$$q_{\text{Laplace}}(\theta) = \mathcal{N}\left(\theta \mid \theta_{\text{MAP}}, \Sigma\right),$$

where Σ is proportional to the inverse Hessian of the loss l_{tot} evaluated at θ_{MAP}:

$$\begin{aligned} \Sigma^{-1} &= \nabla_\theta^2 L|_{\theta=\theta_{\text{MAP}}} = \left(-\nabla_\theta^2 \log p(\mathcal{D}|\theta) - \nabla_\theta^2 \log p(\theta)\right)|_{\theta=\theta_{\text{MAP}}} \\ &= H + \sigma_0^{-2} I_{p+1}. \end{aligned}$$

Since computing H is computationally expensive, we use the GGN Approximation to compute it.

3.1 GGN APPROXIMATION

For the Laplace approximation we are interested in computing the Hessian of the log likelihood $-\log p(\mathcal{D}|\theta) \propto \frac{1}{2\sigma^2}\sum_{i=1}^{n}\|f(x_i) - g_\theta(x_i)\|_2^2$. The GGN approximation is a positive-semi-definite approximation of the full Hessian H, i.e.,

$$\begin{aligned} H &= \nabla_\theta^2\left(-\log p(\mathcal{D}|\theta)\right) \\ &= \nabla_\theta^2 \frac{1}{2\sigma^2}\sum_{i=1}^{n}\|f(x_i) - g_\theta(x_i)\|_2^2 \\ &= \frac{1}{2\sigma^2}\sum_{i=1}^{n}\left[\left(\nabla_\theta g_\theta(x_i)|_{\theta=\theta_{\text{MAP}}}\right)^\top 2\nabla_\theta g_\theta(x_i)|_{\theta=\theta_{\text{MAP}}} + \nabla_\theta^2 g_\theta(x_i)|_{\theta=\theta_{\text{MAP}}}\left(\nabla_g\|f(x_i) - g\|_2^2|_{g=g_{\theta_{\text{MAP}}}(x_i)}\right)^2\right] \\ &= H_{\text{GGN}} + R, \end{aligned}$$

where σ is the dataset noise as in Eq. (3). The GGN approximation is given by

$$H_{\text{GGN}} = \frac{1}{\sigma^2}\sum_{i=1}^{n} J(x_i)^\top J(x_i)$$

where $J(x_i) = \nabla_\theta g_\theta(x_i)|_{\theta=\theta_{\text{MAP}}}$.

References

B. Calderhead and M. Girolami. Estimating Bayes factors via thermodynamic integration and population MCMC. *Computational Statistics and Data Analysis*, 53(12):4028–4045, 2009.

W. Y. Chen, A. Barp, F-X. Briol, J. Gorham, M. Girolami, L. Mackey, and C. J. Oates. Stein point Markov chain Monte Carlo. In *International Conference on Machine Learning*, pages 1011–1021, 2019.

E. Daxberger, A. Kristiadi, A. Immer, R. Eschenhagen, M. Bauer, and P. Hennig. Laplace redux - effortless Bayesian deep learning. In *Neural Information Processing Systems*, pages 20089–20103, 2021.

J. R. Gardner, G. Pleiss, D. Bindel, K. Q. Weinberger, and A. G. Wilson. Gpytorch: Blackbox matrix-matrix gaussian process inference with gpu acceleration. In *Advances in Neural Information Processing Systems*, 2018.

B. C. Goodwin. Oscillatory behavior in enzymatic control processes. *Advances in Enzyme Regulation*, 1965.

J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95, 2007. doi: 10.1109/MCSE.2007.55.

D. P. Kingma and J. L. Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2015.

A. Kirby, T. Nishino, and T. D. Dunstan. Two-scale interaction of wake and blockage effects in large wind farms. *arXiv:2207.03148*, 2022.

Dong C Liu and Jorge Nocedal. On the limited memory bfgs method for large scale optimization. *Mathematical programming*, 45(1):503–528, 1989.

J. Martens. Deep learning via Hessian-free optimization. In *International Conference on Machine Learning*, pages 735–742, 2010.

A. Niayifar and F. Porté-Agel. Analytical modeling of wind farms: A new approach for power prediction. *Energies*, 9(9): 1–13, 2016.

C. J. Oates, T. Papamarkou, and M. Girolami. The controlled thermodynamic integral for Bayesian model comparison. *Journal of the American Statistical Association*, 111(514):634–645, 2016.

A. Paleyes, M. Pullin, M. Mahsereci, N. Lawrence, and J. González. Emulation of physical processes with emukit. In *Second Workshop on Machine Learning and the Physical Sciences, NeurIPS*, 2019.

A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer. Automatic differentiation in pytorch. 2017.

M. M. Pedersen, P. van der Laan, M. Friis-Møller, J. Rinker, and P-E. Réthoré. Dtuwindenergy/pywake: Pywake. Feb 2019. doi: 10.5281/zenodo.2562662.

M. Riabiz, W. Chen, J. Cockayne, P. Swietach, S. A. Niederer, L. Mackey, and C. J. Oates. Optimal thinning of MCMC output. *Journal of the Royal Statistical Society Series B (Statistical Methodology), to appear.*, 2022.

I.M Sobol'. On the distribution of points in a cube and the approximate evaluation of integrals. *USSR Computational Mathematics and Mathematical Physics*, 7(4):86–112, 1967.

Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020. doi: 10.1038/s41592-019-0686-2.