Bayesian Numerical Integration with Neural Networks

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

Abstract

Bayesian probabilistic numerical methods for numerical integration offer significant advantages over their non-Bayesian counterparts: they can encode prior information about the integrand, and can quantify uncertainty over estimates of an integral. However, the most popular algorithm in this class, Bayesian quadrature, is based on Gaussian process models and is therefore associated with a high computational cost. To improve scalability, we propose an alternative approach based on Bayesian neural networks which we call *Bayesian Stein networks*. The key ingredients are a neural network architecture based on Stein operators, and an approximation of the Bayesian posterior based on the Laplace approximation. We show that this leads to orders of magnitude speed-ups on the popular Genz functions benchmark, and on challenging problems arising in the Bayesian analysis of dynamical systems, and the prediction of energy production for a large-scale wind farm.

1 INTRODUCTION

Integration is a core task in probabilistic machine learning. It is required to perform operations such as marginalizing out random variables, or computing normalization constants, predictive distributions, and posterior expectations. In this paper, we consider the computation of the integral of some function $f : \mathcal{X} \to \mathbb{R}$, where $\mathcal{X} \subseteq \mathbb{R}^d$, against some distribution Π with (Lebesgue) density $\pi : \mathcal{X} \to \mathbb{R}$:

$$\Pi[f] = \int_{\mathcal{X}} f(x)\pi(x)dx, \tag{1}$$

where we assume we have access to evaluations $\{f(x_i)\}_{i=1}^n$ at a set of points $\{x_i\}_{i=1}^n \subseteq \mathcal{X}$. The problem is particularly challenging if f and π are multi-modal and/or are very input-sensitive in different regions of the support. A plethora of methods exist for tackling this task; the most common are Monte Carlo (MC) methods, which are sampling-based methods that have been studied extensively in theory and practice [Robert and Casella, 2000, Owen, 2013]. This subsumes naive Monte Carlo, Markov chain Monte Carlo (MCMC) and quasi-Monte Carlo (QMC). Sampling is (at least asymptotically, for MCMC) unbiased and thus a gold standard, but precisely for this reason, it can only converge with stochastic rate, and thus requires a large number of samples n, both for accuracy and uncertainty quantification.

This is a challenge if evaluations of f or samples from π are expensive. The former ("expensive f") emerges regularly in climate simulations or other large physical models. Section 5 provides an example with a wind farm model – a field where state-of-the-art models require hundreds of hours of CPU for a single evaluation [Kirby et al., 2022, 2023]. The latter ("expensive sampling") occurs when π is a posterior distribution for a complex model conditioned on a large amount of data. Section 5 illustrates this through an example of Bayesian inference in dynamical system.

In such scenarios, probabilistic numerical methods (PNMs) [Hennig et al., 2015, Cockayne et al., 2019, Oates and Sullivan, 2019, Wenger et al., 2021, Hennig et al., 2022], and in particular Bayesian approaches, perform particularly well. For numerical integration, the principle behind Bayesian PNMs is to encode prior information about the integrand f, then condition on evaluations of f to obtain a posterior distribution over $\Pi[f]$. These methods are well suited for computationally expensive problems since informative priors can be used to encode properties of the problem and to reduce the number of evaluations needed. In addition, the posterior quantifies uncertainty for any finite value of n.

The most popular Bayesian PNM for integration is Bayesian Quadrature (BQ) [O'Hagan, 1991, Diaconis, 1988, Rasmussen and Ghahramani, 2002, Briol et al., 2019], a method that places a Gaussian Process (GP) [Rasmussen and Williams, 2006] prior on f. With this convenient choice of prior, the posterior on $\Pi[f]$ is a univariate Gaussian, whose

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

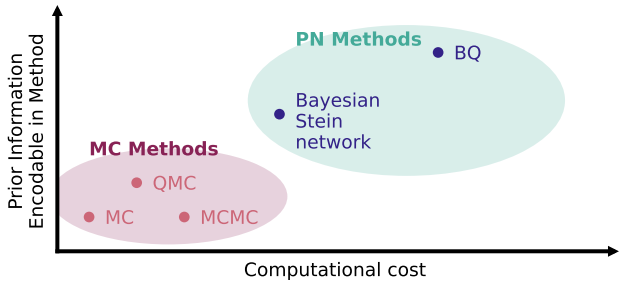

Figure 1: Integration methods can be compared at a high-level in terms of their computational cost and ability to include prior information. In both respects, BSNs provide a compromise in-between MC and BQ.

mean and variance can be computed in closed form for certain combinations of prior covariance and distribution. However, for high-dimensional problems where large amounts of data are necessary, the computational cost of GPs, cubic in n, can render BQ too computationally expensive. Fast BQ methods have been proposed to resolve this issue [Karvonen and Särkkä, 2018, Jagadeeswaran and Hickernell, 2019], but these usually work for a limited range of π or $\{x_i\}_{i=1}^n$, and therefore do not provide a widely applicable solution.

This raises the question of whether an alternative probabilistic model could be used in place of a GP within probabilistic integration. Bayesian neural networks (BNNs) are an obvious candidate, as they are known to work well in high dimensions and with large n. Unfortunately, their application to integration tasks is not straightforward since, in contrast to the GP case, analytical integration of the posterior mean of a BNN is usually intractable. This is a significant challenge which has so far prevented their use for probabilistic numerics. We resolve this challenge by proposing the concept of *Bayesian Stein (neural) networks* (BSNs), a novel BNN architecture based on a final layer constructed through a Stein operator [Anastasiou et al., 2021]. Such choice of architecture is designed specifically so that the resulting BNN is analytically integrable (see Section 3), and hence at our disposal for numerical integration.

Given these approaches—MC, BQ, BSNs—a natural question remains: "How should we select a method for a given integration task?". We provide an empirical answer to this question in Section 5, where we consider a popular benchmark dataset, compute posterior expectations arising in the Bayesian treatment of dynamical systems, and estimate the expected power output of a wind farm.

Our conclusions are summarized in Figure 1 and presented below. If sampling π and evaluating f is computationally cheap, so one can obtain a very large number of data points relative to the complexity of the problem, then MC methods are likely the best choice. But if n is very limited due to our computational budget, then BQ is likely a better option. BSNs excel in the intermediate region where n is such that BQ becomes prohibitively expensive but MC is not accurate enough. The architecture of neural networks, plus sophisticated deep learning software libraries, make training of (small) neural networks memory efficient and fast. However, achieving good accuracy at low training cost requires special care during training for the Stein architecture. Finding a good training setup is a main contribution of this work, outlined in Section 4.

For all integration methods, estimates from scarce data are imperfect, so uncertainty estimates are crucial. Bayesian deep learning provides this functionality. Full Bayesian inference is costly even for small neural networks, but we show that a lightweight Laplace approximation [MacKay, 1992, Ritter et al., 2018] can provide good approximate uncertainty for the Stein network.

2 RELATED WORK

BQ is the method most closely related to our proposed approach and the approach is fully detailed in Appendix 2. Bayesian PN methods based on alternative priors have also been proposed. These include Bayesian additive regression tree priors [Zhu et al., 2020], multi-output Gaussian process priors [Xi et al., 2018, Gessner et al., 2019], and Dirichlet process priors [Oates et al., 2017b]. These priors each provide different advantages, such as the ability to model highly discontinuous functions, vector-valued functions, or modelling probability distributions respectively. Unfortunately none of these approaches significantly improve scalability, the main goal of our paper.

The use of (non-Bayesian) neural networks for integration was previously proposed by Lloyd et al. [2020]. However, their method is only applicable for uniform π and shallow networks. Wan et al. [2020], Si et al. [2021] propose to use a Langevin Stein operator applied to a neural network to find good control variates for variance reduction in Monte Carlo approximations (based on an earlier construction by Oates et al. [2017a]). In contrast to their work, we use the neural network to directly compute $\Pi[f]$, and our neural network follows Bayesian semantics and can be used to quantify uncertainty. This requires a different network architecture and an efficient posterior inference algorithm.

3 BAYESIAN STEIN NETWORKS

We now describe BSNs. This requires introducing Stein operators, BNNs, and Laplace approximations.

Stein Neural Networks Stein operators are a technical construction originating in probability theory, but have recently been used as a computational tool [Anastasiou et al., 2021]. Building on this line of work, we will use Stein oper-

ators to construct the final layer of our BNNs. The reason for this is simple: given some function u (with possibly unknown mean) and a distribution π, a Stein operator can map u to a mean zero function under π. This final layer therefore allows us to construct flexible BNNs with the powerful property that *any draw from the posterior will have a known mean under* π. We now highlight this procedure in detail.

We call $\mathcal{S}$ a *Stein Operator* if for any suitably regular continuously differentiable $u : \mathbb{R}^d \to \mathbb{R}^d$, the following holds

$$\Pi\left[\mathcal{S}[u]\right] = 0. \tag{2}$$

Suppose $\mathcal{X} = \mathbb{R}^d$, π is continuously differentiable on $\mathcal{X}$, such that $\nabla_x \log \pi$ is well-defined ($[\nabla_x \log \pi(x)]_i = \partial \log \pi(x)/\partial x_i$ for all $i \in \{1, \dots, d\}$). One example of an operator fulfilling (2) is the diffusion Stein operator [Gorham et al., 2019, Barp et al., 2019]:

$$\begin{aligned}\mathcal{S}_m[u](x) := &\left(m(x)^\top \nabla_x \log \pi(x)\right)^\top u(x) \\ &+ \nabla_x \cdot \left(m(x)u(x)\right),\end{aligned} \tag{3}$$

where $\nabla_x \cdot u(x) = \sum_{i=1}^d \partial u_i(x)/\partial x_i$, and $m : \mathbb{R}^d \to \mathbb{R}^{d\times d}$ is an invertible matrix-valued function. This operator only requires access to $\nabla_x \log \pi(x_i)$, and can thus be used even if the normalization constant of π is unknown. This is an advantage if π is itself a posterior distribution. In such settings, samples from π can be obtained via MCMC, but the distribution π itself cannot be evaluated directly.

To construct BSNs, we use an architecture based on a continuously differentiable deep neural network $u_{\theta_u} : \mathcal{X} \to \mathbb{R}^d$, where $\theta_u \in \Theta_u \subseteq \mathbb{R}^p$, combined with a final layer taking the form of a Stein operator (that we call a *Stein layer*). More precisely, we consider an architecture $g_\theta : \mathcal{X} \to \mathbb{R}$:

$$g_\theta(x) := \mathcal{S}_m\left[u_{\theta_u}\right](x) + \theta_0. \tag{4}$$

We call this neural network a *Stein neural network* following [Wan et al., 2020, Si et al., 2021, Sun et al., 2023], but note that we use the more general diffusion Stein operators $\mathcal{S}_m$ [Gorham et al., 2019, Barp et al., 2019]. Previous cases can be recovered with $m(x) = I_d$, where I_d is a d-dimensional identity matrix, however we will demonstrate in Section 5 that alternative choices for m can significantly improve the performance of our method.

The parameter $\theta = \{\theta_0, \theta_u\} \in \Theta \subseteq \mathbb{R}^{p+1}$ denotes the weights of the neural network g_θ. Thanks to our choice of architecture, (2) holds and we have:

$$\Pi\left[g_\theta\right] = \theta_0. \tag{5}$$

The last layer of g_θ directly tracks the integral of the network, which is the key property for our purpose: by training such a network g_θ on data from f so that $g_\theta \approx f$, we are simultaneously constructing a good approximation of the integral $\Pi[g_\theta] \approx \Pi[f]$ (see Figure 2 for a summary).

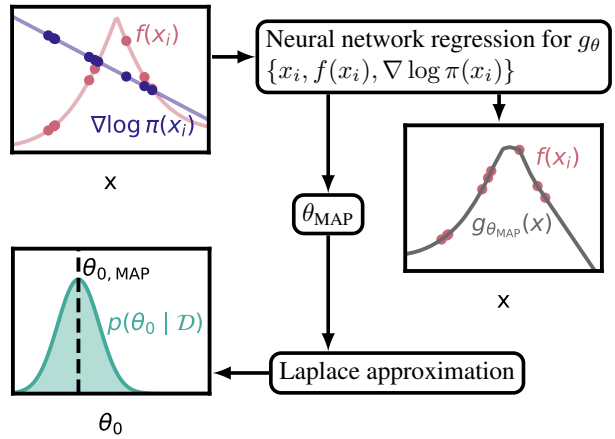

Figure 2: *Visualization of BSNs.* The BSN prior is conditioned on $\{x_i, f(x_i), \nabla \log \pi(x_i)\}_{i=1}^n$ to obtain a Bayesian posterior on θ_0. This posterior quantifies our uncertainty about $\Pi[f]$. For computational reasons, this posterior is approximated the Laplace approximation around the MAP estimate $\theta_{0,\text{MAP}}$.

Uncertainty Estimates for Stein Neural Networks In the context of Bayesian PNM, proposing a BNN architecture is not enough: we are also interested in *tractable uncertainty estimates over* $\Pi[f]$. We show how to obtain this through the Laplace approximation and a suitable choice of prior, but further details are available in Appendix 3.

The specific architecture of the BSN model means that all the uncertainty on $\Pi[f]$ is represented by the Bayesian posterior on θ_0. This can be obtained through a standard application of Bayes' theorem $p(\theta|\mathcal{D}) \propto p(\mathcal{D}|\theta)p(\theta)$ where in our case the dataset is $\mathcal{D} = \{x_i, f(x_i), \nabla_x \log \pi(x_i)\}_{i=1}^n$, and $p(\theta)$ denotes our prior, $p(\theta|\mathcal{D})$ the posterior and $p(\mathcal{D}|\theta)$ the likelihood. The posterior on θ_0 is then the marginal of $p(\theta|\mathcal{D})$. Bayesian inference for deep networks provides uncertainty estimates [Neal, 1996, Mackay, 1995] through $p(\theta|\mathcal{D})$, but this posterior is intractable in general. MCMC is a prominent tool for approximating $p(\theta|\mathcal{D})$, but using it within an integration method would be circular and reintroduce the spectre of high computational cost [Izmailov et al., 2021]. Other popular approximate inference schemes include variational inference [Graves, 2011, Blundell et al., 2015, Hinton and van Camp, 1993] and ensemble methods [Lakshminarayanan et al., 2017]. Although cheaper, the cost associated with this can still be significant.

We instead opt for the arguably most lightweight approach available for BNNs: the Laplace approximation [MacKay, 1992, Ritter et al., 2018]. It is a simple and computationally cheap method, but yet provides competitive uncertainty estimates [Daxberger et al., 2021]. The Laplace approximation constructs a second-order Taylor approximation around the mode of the posterior, which amounts to a Gaussian approximate of the posterior around the MAP (maximum a-

posteriori) estimate. This can be criticized from a Bayesian standpoint as the MAP estimate and the posterior mean of the weights do not necessarily coincide. However, the MAP estimate is the quantity that is usually tuned in deep learning and is also cheap as it only has to be computed once. To be more precise, our approximation of the posterior is implemented in two steps: a Laplace approximation, and an approximation of the corresponding Hessian.

For the first step, we train the network g_θ by minimizing the mean squared error loss with weight decay regularizer, given for $\lambda > 0$ by:

$$\begin{aligned} l_{\text{tot}}(\theta) &= l(\theta) + \lambda \|\theta\|_2^2 \\ \text{where } l(\theta) &= \tfrac{1}{n} \textstyle\sum_{i=1}^n \|f(x_i) - g_\theta(x_i)\|_2^2 \end{aligned} \tag{6}$$

We notice that $l \propto -\log p(\mathcal{D}|\theta)$ and $\lambda\|\theta\|_2^2 \propto -\log p(\theta)$ whenever we take the prior to be $p(\theta) = \mathcal{N}(\theta \mid 0, \sigma_0^2 I_{p+1})$ (σ_0 is related to λ through a known constant see Appendix 3). As a result, the minimum of the loss above is indeed a MAP estimate: $\theta_{\text{MAP}} = \text{argmin}_\theta l_{\text{tot}}(\theta)$.

Of course, *any* Bayesian treatment of neural networks requires a prior $p(\theta)$. The choice is important since the prior encodes the model class, but there is currently no widely accepted choice. Our choice above was motivated by the fact that for the Laplace approximation, only isotropic Gaussian priors are currently feasible [MacKay, 1992, Ritter et al., 2018, Daxberger et al., 2021]. Fortuin et al. [2022] suggest that such priors are undesirable, but Wilson and Izmailov [2020] argue to the contrary: despite their simplicity, such priors still induce sufficiently complex distributions over functions. Daxberger et al. [2021] note that it is often beneficial to tune σ_0 for inference.

Once the MAP has been identified, we can construct our Laplace approximation using a Taylor approximation (up to second order) of the log-posterior $\log p(\theta \mid \mathcal{D})$ around that point. This results in a Gaussian approximation of the posterior distribution: $q_{\text{Laplace}}(\theta) = \mathcal{N}(\theta \mid \theta_{\text{MAP}}, \Sigma)$, where Σ is proportional to the inverse Hessian of the loss l_{tot}:

$$\begin{aligned} \Sigma^{-1} &= -\nabla^2 \log p(\mathcal{D}|\theta) - \nabla^2 \log p(\theta) \\ &= H + \sigma_0^{-2} I_{p+1}, \quad \text{where } H \propto \nabla_\theta^2 l(\theta_{\text{MAP}}) \end{aligned}$$

Our second step consists of an approximation of the Hessian. This is necessary since it is often infeasible to calculate H due to the large computational cost when p is large. As a result, we use a positive definite approximation called the Generalized-Gauss-Newton (GGN; [Schraudolph, 2002]) approximation:

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

where $J(x_i) = \nabla_\theta g_\theta(x_i)|_{\theta=\theta_{\text{MAP}}}$ and σ is the dataset noise. This gives us another approximation of the posterior that we denote $q_{\text{GGN-Laplace}}(\theta)$ obtain through $\Sigma_{\text{GGN}}^{-1} = H_{\text{GGN}} + \sigma_0^{-2} I_{p+1}$. Hence, we can extract an approximation of the posterior on the network's prediction of the integral $\Pi[f]$ using Eq. (5):

$$q_{\text{GGN-Laplace}}(\theta_0) = \mathcal{N}\left(\theta_0 | \theta_{0,\text{MAP}}, (\Sigma_{\text{GGN}})_{0,0}\right).$$

4 ARCHITECTURE

Due to their specific architecture, naive attempts to train BSNs can lead to unsatisfactory results. Below, as a key contribution, we provide architectural considerations that we have found to significantly improve the conditioning of the loss and lead to better training.

Choice of Activation Function We require u_{θ_u} to be continuously differentiable on $\mathcal{X}$, which imposes restrictions on the activation functions of the BSN. A sufficient condition is for these activation functions to be themselves continuously differentiable. This excludes the popular RELU activation functions, but includes the CELU ('Continuously Differentiable Exponential Linear Units' [Barron, 2017]; $\text{CELU}(x) = \max(0, x) + \min(0, \exp(x) - 1)$), its continuous extension. It also includes the tanh ($\tanh(x) = (\exp(x) - \exp(-x))/(\exp(x) + \exp(-x))$), Gaussian ($\text{Gauss}(x) = \exp(-x^2)$), and sigmoid ($\text{sigm}(x) = 1/(1 + \exp(-x))$), TanhShrink ($\text{TanhShrink}(x) = x - \tanh(x)$) activations. We compared activation functions (see Figure 4 below) and found the CELU to give marginally superior performance on test problems. Based on its good performance, we use CELU activations for all experiments.

Choice of Optimization Procedure Optimization for BSNs is challenging due to the unique network architecture. For one, the architecture contains gradients of the Stein layer, which are harder to train than standard activation functions. This is because $\nabla_x \log \pi$ can be arbitrarily complicated depending on π. We find that the training of g_θ with Adam [Kingma and Ba, 2015] is considerably slower compared with training u_{θ_u} (see the Appendix 1.1.1). We suspect that this is due to the loss landscape of the BSN being more narrow (i.e., having a larger spread in curvature eigenvalue spectrum) than that of u_{θ_u}. A second order method should alleviate this issue. Hence, we train the BSN with L-BFGS (an approximate second order method) and the *Hessian-free* optimizer [Martens, 2010] (a conjugate gradient based second order method). And indeed, (approximate) second order optimization reaches much better performance (for an extended discussion see the Appendix 1.1.1).

We therefore used L-BFGS throughout all subsequent experiments. Such quasi-Newton methods have fallen out of fashion in deep learning because they are not stable to noise. In our experiments, we train on the full dataset, so noise is not an issue. We accomplish better (i.e., lower loss) and faster convergence (both in iterations and compute time) with this method compared to gradient descent and its variants. Note that this approach is only feasible for relatively

small (in number of weights p) network architectures, as it requires storing the gradient history for the approximate Hessian in memory. When training on the entire dataset (i.e., no mini-batching), we observe significant speed-up from using GPUs when n is large ($\approx 10^4$).

Choice of $m(x)$ For most of the experiments we set $m(x) = I_d$, but in general other choices for m are possible. We test a set of different choices ($m(x) = I_d/(||x||_2^2 + 1)$, $m(x) = I_d/\sqrt{||x||_2^2 + 1}$, $m(x) = I_d\pi(x)$, $m(x) = \mathrm{diag}(x)$), but find that none of these perform significantly better than $m(x) = I_d$ (see Appendix 1.1.4 for more details).

Choice of Point Set BSNs can be implemented regardless of the choice of $\{x_i\}_{i=1}^n$, but we expect better performance when $\{x_i\}_{i=1}^n$ cover regions of high probability under π. A simple solution is to use independent samples from π; this will be our default choice. When independent sampling is not possible, we can use MCMC instead, so long as π can be evaluated up to some normalization constant. Alternatives also include grid of points or QMC point sets (see the Appendix 1.1.2 for a comparison of different point sets), but these are usually only helpful when $\mathcal{X}$ is a hypercube and π is uniform. Alternatively, one could also use active learning (see Gunter et al. [2014], Briol et al. [2015] for corresponding approaches for BQ) based on the Laplace approximation of the uncertainty, but this may not perform well for large d, and we did not explore the idea further.

Stein Architecture for Bounded Domains The architecture outlined in Section 3 is only valid on the open integration domain $\mathcal{X} = \mathbb{R}^d$. For bounded $\mathcal{X} \subset \mathbb{R}^d$, it is incorrect because $\Pi[\mathcal{S}_m[u]] = 0$ is not necessarily true. This can be guaranteed by adding a layer before the Stein layer. For example, let $\tilde{u}_{\theta_u}(x) = u_{\theta_u}(x)\delta(x)$, where $\delta(x)$ is a smooth function (so that $\tilde{u}_{\theta_u}$ is continuously differentiable) going to zero on the boundary of $\mathcal{X}$. Then, $\pi(\cdot)\tilde{u}_{\theta_u}(\cdot)$ is zero on the boundary of $\mathcal{X}$, and as a result $\Pi[\mathcal{S}[\tilde{u}_{\theta_u}]] = 0$. When $\mathcal{X} = (a, b) \subset \mathbb{R}$, one such function is given by $\delta(x) = (x - a)(b - x)$, and we will use this example where necessary in our experiments. Beyond bounded $\mathcal{X}$, the architecture can also be adapted to manifold or discrete $\mathcal{X}$; see Barp et al. [2022] and Shi et al. [2022] respectively.

5 EXPERIMENTS

We consider three main experiments: the Genz functions benchmark, a parameter inference problem for a dynamical system called Goodwin Oscillator, and an example describing the energy output of a wind farm. We compare BSNs to the following approaches:

- Monte Carlo methods. When independent sampling from π can be used (i.e. for the Genz benchmark and the wind farm experiments) we use MC. When this is not possible, we use instead an MCMC method called Metropolis-Adjusted Langevin algorithm [MALA; Roberts and Tweedie, 1996].
- A BQ implementation based on `emukit` [Paleyes et al., 2019], with an RBF covariance function $k(x, y) = \lambda \exp(-\|x - y\|_2^2/l^2)$ for some $l, \lambda > 0$. We use log-likelihood maximization to choose l and set the GP prior mean to 0, as we do not have any prior knowledge about the value of the integral. In Appendix 1.1.5 we conduct an additional experiment using the Matern 1/2 Kernel. However, for this kernel, the posterior mean is only available in $d = 1$.
- A control functional estimator based on Stein's method (Stein-CF) as described in Oates et al. [2019] for the experiments on the Genz dataset and the Goodwin oscillator. The approach can be thought of as a kernel interpolant alternative to our neural network. We use $m(x) = I_d$ and an RBF kernel. We use log-likelihood maximization to set the kernel hyperparameters.

To implement the Laplace approximation, we use `laplace-torch` library [Daxberger et al., 2021]. Across all experiments we employ the same fully connected architecture for u_{θ_u}, where each hidden layer has 32 units, and we use 2 hidden layers (see the Appendix 1.1.3 for more details).

Genz Benchmark We first consider the Genz family of integrands [Genz, 1984], as a test ground (see Appendix 1.2 for detailed definitions). This benchmark, consisting of six integrands with known integrals, was proposed to highlight the performance of numerical integration methods on challenging tasks including discontinuities, peaks and oscillations. Each integrand has a parameter which can be used to increase the dimensionality d of the domain. We follow the implementation of Si et al. [2021], where the test functions are transformed to be supported on $\mathcal{X} = \mathbb{R}^d$ and integrated against a multivariate standard Gaussian distribution π. Since these functions are very cheap, we do not expect BSN or BQ to be competitive with MC methods in terms of runtime, but we use this experiment to showcase the performance of BSNs for challenging integrands and compare methods for fixed n.

In Table 1, we first consider the case $d = 2$ and $n = 5120$. BSN and BQ both outperform MC by several orders of magnitude in terms of mean relative integration error. Notably, BSN is significantly better than BQ for the discontinuous Genz function, indicating that the neural network is able to adapt to rapidly changing functions. For the Gaussian Genz function, BQ outperforms the BSN due to the fact that the prior is more informative. Both methods lead to a significant improvement over MC, but we can run the BSN at higher number of data points n than BQ (see Appendix 1.2).

Table 1: *Performance on Genz integral family in $d = 2$.* Mean relative integration error and standard deviation (based on 5 repetitions) using $n = 5120$.

	Mean Absolute Error		
Integrand	**MC**	**BQ**	**BSN**
Continuous Genz	1.59e-03 ± 0.90e-03	1.40e-03 ± 0.09e-03	**1.11e-05 ± 0.55e-05**
Discontinuous Genz	2.69e-02 ± 2.64e-02	1.12e-02 ± 0.50e-02	**2.56e-03 ± 1.94e-03**
Gaussian peak	1.52e-02 ± 8.85e-03	**1.17e-06 ± 1.11e-06**	1.83e-04 ± 1.35e-04
Corner peak	1.85e-02 ± 1.85e-02	**2.49e-04 ± 1.53e-04**	6.00e-04 ± 5.39e-04
Oscillatory Genz	2.88e-01 ± 1.75e-01	4.13e-03 ± 0.89e-03	**1.34e-03 ± 0.97e-03**
Product peak	7.59e-03 ± 4.11e-03	1.82e-04 ± 0.42e-04	**1.42e-04 ± 0.76e-04**

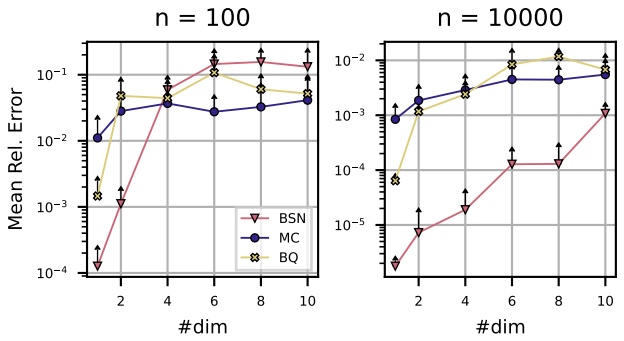

Figure 3: *Continuous Genz function*. We compare methods as a function of d for $n = 100$ (*left*) and $n = 10000$ (*right*)(mean and standard deviation based on 5 repetitions).

We then considered the impact of dimensionality on MC, BQ, and BSN in Figure 3. We focus on the Continuous Genz function for simplicity. If too few evaluations n are available, the Stein network cannot approximate f well, but with a sufficiently large n (i.e. $n \approx 10^2$ in $d = 1$ and $n \approx 10^4$ in $d = 10$), BSN significantly outperforms MC and BQ.

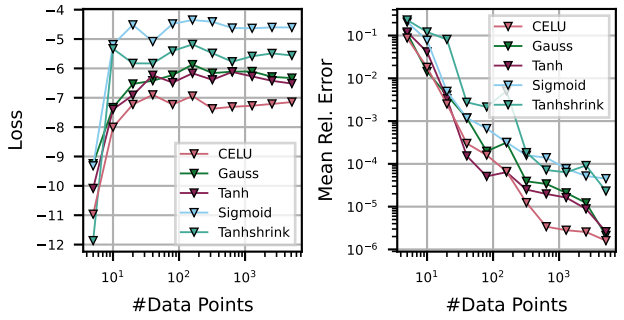

Figure 4: *Impact of the choice of activation function for the Continuous Genz function.* Loss l (*left*) and mean relative integration error (*right*) (mean based on 5 repetitions) as a function of n.

We also considered the impact of the choice of activation functions for u_{θ_u} in Figure 4. Again, we focus on the Continuous Genz integrand, but limit ourselves to $d = 1$. We consider a diverse set of activation functions (described in Section 4), all continuously differentiable as required for the final Stein layer. We find that the CELU activation leads to the best results on the Continuous Genz dataset, but other activation functions like the tanh and Gaussian activations also perform well.

Finally, we have a deeper look at the Continuous Genz function in $d = 20$ in Figure 5. We observe that a large enough n ($n \approx 10^4$) is necessary for the interpolation capabilities of the model to significantly improve performance. In those cases, the BSN achieves significantly better performance than MC-sampling. We note that MC sampling is cheap on the Genz benchmark dataset, and this benchmark is only used as a test bed to vary the complexity of our integrands, so we only compare the MC to the other methods in terms of sample efficiency. Both BQ and Stein-CF do not achieve good performance and are too expensive (in runtime and in memory) to run for large n. The BSN can perform well even for much larger datasets (we ran it up to $n \approx 10^6$).

We can compare the BSN and BQ not only in runtime but also in terms of memory requirements. However, computing accurate memory requirements in python can be difficult as common python libraries use for example `C++` backends. The memory requirements of these non-python backends is commonly not taken into account using the built-in memory profiler. So instead, we use the profiler of our cluster, which outputs the maximum memory required by the program. Figure 6 shows that the BSN memory requirements increase more slowly than for BQ. Both kernel based methods (BQ and CF) surpass our allotted memory limit of 20 GB.

To evaluate the uncertainty estimates provided by the GGN-Laplace approach, we calculate their calibration γ. The calibration is given by the ratio between relative integration error e_{abs}, and the standard deviation σ_{θ_0} of the GGN-Laplace approximation of the posterior on θ_0: $\gamma = e_{\text{abs}}/\sigma_{\theta_0}$. Similarly, for BQ, σ_{θ_0} is the posterior standard deviation on $\Pi[f]$. A calibration fluctuating around one indicates a well calibrated model, and a large calibration suggests a model that is overconfident, rendering its uncertainty estimates unreliable. The GGN-Laplace approach as well as BQ lead to

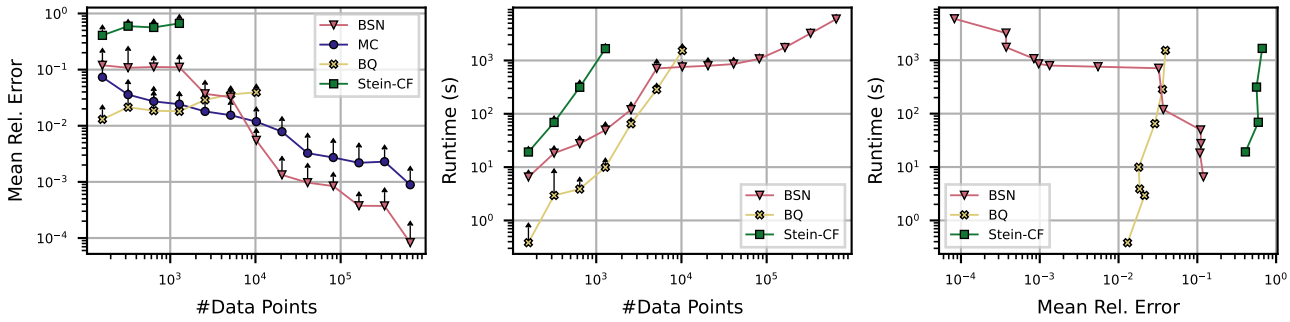

Figure 5: *Continuous Genz function in* $d = 20$. Mean relative integration error (*left*), and run time (*center*) (mean and standard deviation based on 5 repetitions) as a function of n. *Right:* Run time in seconds as a function of mean relative integration error.

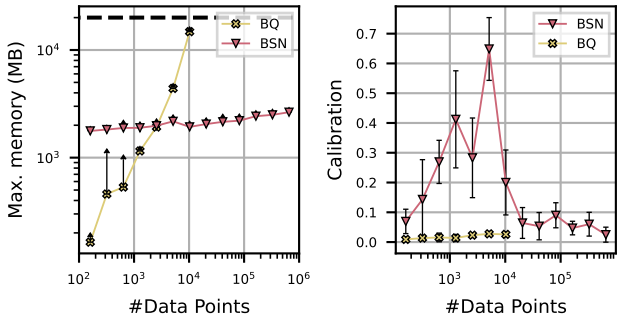

Figure 6: *Continuous Genz function in* $d = 20$. Memory requirements (*left*) and calibration (*right*) (mean and standard deviation based on 5 repetitions) as a function of n.

uncertainty estimates which are underconfident (although less so for the BSN), especially in the high data regime (see Figure 6). Underconfident predictions are still useful in that they provide a prudent assessment of our uncertainty.

Bayesian Inference for the Goodwin Oscillator A challenging computational task in Bayesian inference is posterior inference for parameters of dynamical systems (see for example Calderhead and Girolami [2011]). The challenge is due to the large computational cost of posterior sampling, which is incurred due to the need to solve systems of differential equations numerically at a high-level of accuracy. In addition, large datasets can further increase the computational cost, making the task a prime candidate for BSNs. For this experiment, we consider parameter inference in a dynamical system called the Goodwin oscillator [Goodwin, 1965]. This model describes how the feedback loop between mRNA transcription and protein expression can lead to oscillatory dynamics in a cell. It is a common benchmark for MC methods [Calderhead and Girolami, 2009, Oates et al., 2016, Riabiz et al., 2022].

We analyse the setting with no intermediate protein species, leading to a system with $d = 4$ parameters: $x = (a_1, a_2, k, \alpha) \in \mathbb{R}^4_+$. Given a posterior distribution π, we want to compute the posterior mean $\Pi[f]$ of each of the ODE parameters, i.e., $f(x) = x$. For this experiment, the posterior distribution is conditioned on a synthetic dataset of 2400 observations generated for some known parameter values. Our exact experimental setup is based on [Chen et al., 2019], and we refer to the Appendix 1.3 for a detailed description.

The posterior density π is only available in unnormalized form, and we therefore use MALA for sampling. This is relatively expensive: sampling $n = 1000$ realizations takes around 30 seconds, which is on the same timescale as network inference ($\sim$ 1 min). For ODE problems requiring more complex solvers or settings with a large dataset, the sampling time might increase even further.

In this setting, $\nabla_x \log \pi(x)$ can take very large values, which makes training the BSN harder. We find that $m(x) = I_d/C$ for $C \in \mathbb{R}$ can considerably improve the performance. We considered two choices for the constant C:

- the standard deviation of $\{\nabla_x \log \pi(x_i)\}_{i=1}^n$ (called $C = \text{std}$ in Figure 7).
- the largest score value: $C = \max_{i=1,\dots,n} \nabla_x \log \pi(x_i)$ ($C = \max$ in Figure 7).

Figure 7 compares the performance of the proposed regularizations. Both choices work well, in contrast to using no regularization at all (i.e. $C = 1$). We find that the BSN either matches the performance of MALA (for parameter α) or surpasses it (parameter a_1). The Stein-CF performs well but struggles in the high data regime due to unstable hyperparameter optimization. The results for a_2 and k are presented in Appendix 1.3. The saturation in reached accuracy for both the BSN and MALA can be attributed to the noisy likelihood evaluations.

Before concluding, we emphasize that BSN is the only available Bayesian PNM here. This is because π is unnormalized and BQ would require an additional step of computing the normalization constant, which would lead to additional runtime and likely incur additional numerical error.

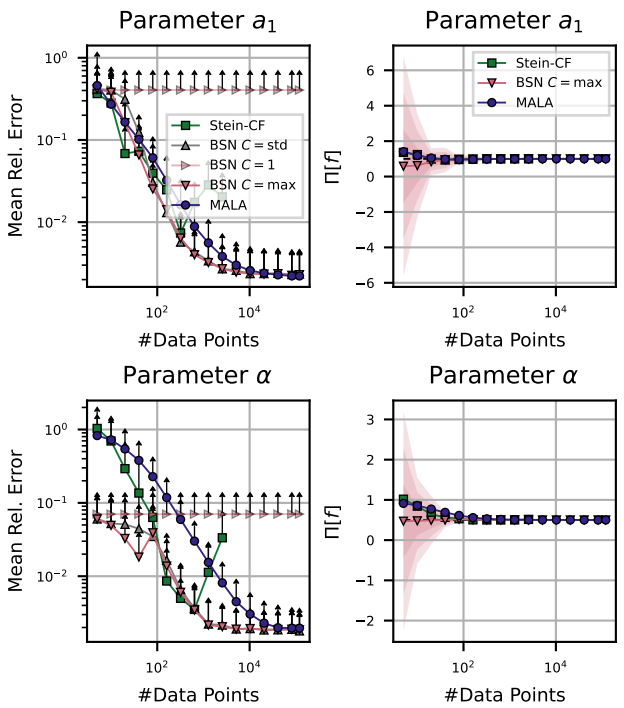

Figure 7: *Posterior expectations for the parameters of a Goodwin ODE*. Mean relative integration error and standard deviation (*top-left* and *bottom-left*), and uncertainty estimates (*top-right* and *bottom-right*) (based on 5 repetitions) as a function of n.

Expected Local Turbine Thrust Coefficient for Wind Farm Layout Design The energy produced by a wind farm depends on factors including the distance between turbines, the direction of the wind, and the wake produced by individual turbines. To understand this phenomenon, fluid dynamic simulations can be used to estimate a local turbine thrust coefficient (which we denote f), which largely determines energy production [Nishino, 2016]. Since a number of these factors are unknown, it is common practice to represent uncertainty through a distribution (denoted π), and calculate the *expected* local turbine thrust coefficient $\Pi[f]$.

A particular challenge here is the cost of evaluating f. For the model we are using (a low-order wake model from Niayifar and Porté-Agel [2016]), each evaluation of f takes approximately 130 seconds, but more accurate models [Kirby et al., 2022] can take up to tens of hours per evaluation. However, it is well known that f is a smooth function of the inputs, which makes Bayesian PNMs, such as BSNs, prime candidates for the task.

The input to our model f are the wind direction, the turbulence intensity, as well as a number of parameters representing the design of the wind farm (including parameters impacting the distance between turbines, and turbine-specific parameters such as the turbine resistance coefficient, the turbine hub height and diameter, and parameters describe the turbine wake). The distribution π consists of independent distributions (either mixtures of Gaussians, or a truncated Gaussian) on each input to the wake model. The Appendix 1.4 provides full details on the wind farm dataset.

The results are presented in Figure 8. Since the ground truth is unknown for this problem, we ran BSN on a dataset which is 5 times larger than what is plotted in order to get a benchmark value. We compared the runtime of all methods including sampling, where we assume that all the points were sampled sequentially (corresponding to running the experiment on a single CPU). The additional runtime of both BQ and the BSN is negligible compared to the initial sampling time. Both methods achieve a much lower mean relative integration error compared to sampling, clearly demonstrating the power of Bayesian PNM methods for problems involving expensive integrands.

On this dataset BQ cannot be used to compute uncertainty estimates, because we cannot integrate the kernel twice in closed form for truncated Gaussians. However, the uncertainty estimates computed with the Laplace approximation for the BSN accurately capture deviations from the ground truth value (shown in Figure 8).

6 LIMITATIONS AND DISCUSSION

The primary advantage of BSNs is in terms of scalability, but they also suffer from some limitations, discussed below.

Firstly, in contrast to GPs where prior knowledge (such as periodicity or smoothness) about f can be encoded via a kernel, selecting good functional priors for BNNs can be challenging. In general, the BSN encodes that the integrand f is smooth, as the outputs of the BSN are smooth functions (the BSN is composition of smooth functions). Our experiments show that simple prior choices are often sufficient to achieve good results for moderately hard problems. More advanced options [Sun et al., 2019, Pearce et al., 2019] could be considered, but this would require novel Laplace approximations. In cases the BNN prior is misspecified (i.e., the function f is not contained in the prior), we can always increase the number of parameters of the neural network.

Secondly, our experiments suggest convergence with large n. Although we did not analyse this convergence from a theoretical viewpoint, we note that Si et al. [2021, Proposition 1 and 2] can be used to prove consistency of the BSN posterior mean to the true value of the integral. Currently, we do not have any results for the convergence rates, but this could be an interesting direction for future research (for example, Belomestny et al. [2023] provides a rate for a related approach). This is in contrast with the GP case where convergence results are highly developed [Kanagawa et al., 2020, Kanagawa and Hennig, 2019, Karvonen et al., 2020, Wynne et al., 2021].

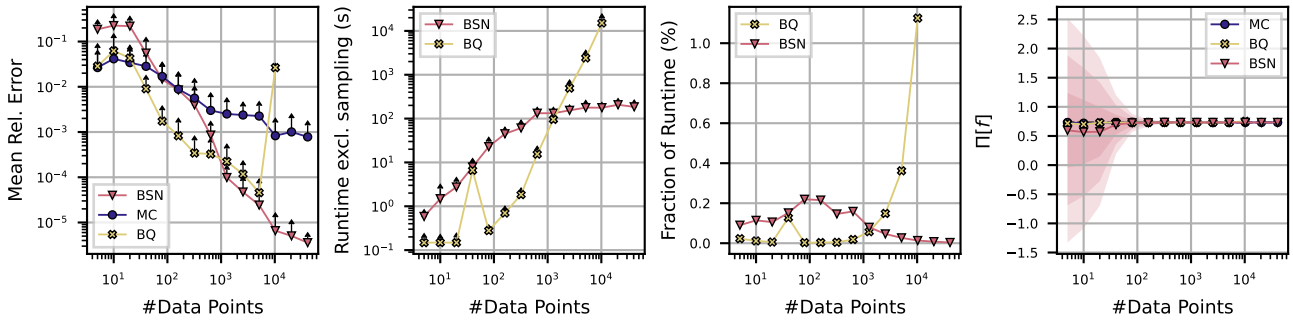

Figure 8: *Wind farm model*. Mean relative integration error (*left*), and run time (*centre-left*) (mean and standard deviation based on 5 repetitions) as a function of n. *Center-right:* Fraction of runtime BSN and BQ contribute to the total runtime which includes the runtime of the wind farm simulation. *Right:* Uncertainty estimates provided by the Laplace approximation.

Thirdly, computational cost is highly dependent on the complexity of the deep network u_{θ_u}. Using standard matrix multiplication, neural network training is linear in the number of (non-bias) parameters p, the number of training samples n, and the number of gradient iterations i, i.e, $O(pni)$. Across all our experiments we used the same architecture for u_{θ_u} independent of n, but we expect that the complexity of the network will need to increase significantly when high accuracy is required. In such cases, we expect that mini-batching and first order optimization could improve *scalability*, but would likely incur new issues with *stability*.

7 CONCLUSION

We have introduced a way to leverage the function approximation abilities of deep BNNs specifically for integration through the application of a Stein operator. Employing a Laplace approximation provides uncertainty quantification of good quality in this architecture. We have noted that significant work is required to stabilize the training process to this end: both the architecture and the training method must be adapted to the non-standard form of the loss.

BSNs perform consistently well across experiments, both in accuracy and in runtime, and are thus an interesting alternative to BQ, especially for the intermediate regime between very small sample size (where traditional BQ works well), and very large sample numbers (where classic MC methods continue to be the preferred solution). Our experiments on a variety of applications also highlight some functional strengths of the BSN approach. In particular, it can deal flexibly with a wide range of integration densities, including cases in which the density is known in unnormalized form.

Acknowledgements

The authors would like to thank Zhuo Sun, Lukas Tatzel, Frank Schneider and Andrew Kirby for helpful discussions. We would also like to thank Andrew Kirby for sharing code to run the wind-farm experiments, which is available at https://github.com/AndrewKirby2/ctstar_statistical_model. Part of this work was initiated by Dagstuhl Seminar 21432 "Probabilistic Numerical Methods - From Theory to Implementation." PH gratefully acknowledges financial support by the European Research Council through ERC StG Action 757275 / PANAMA; the DFG Cluster of Excellence “Machine Learning - New Perspectives for Science”, EXC 2064/1, project number 390727645; the German Federal Ministry of Education and Research (BMBF) through the Tübingen AI Center (FKZ: 01IS18039A); and funds from the Ministry of Science, Research and Arts of the State of Baden-Württemberg. FXB was supported by the Lloyd’s Register Foundation Programme on Data-Centric Engineering and The Alan Turing Institute under the EPSRC grant [EP/N510129/1], and through an Amazon Research Award on “Transfer Learning for Numerical Integration in Expensive Machine Learning Systems”.

References

A. Anastasiou, A. Barp, F-X. Briol, B. Ebner, R. E. Gaunt, F. Ghaderinezhad, J. Gorham, A. Gretton, C. Ley, Q. Liu, L. Mackey, C. J. Oates, G. Reinert, and Y. Swan. Stein’s method meets statistics: A review of some recent developments. *arXiv:2105.03481*, 2021.

A. Barp, F.-X. Briol, A. B. Duncan, M. Girolami, and L. Mackey. Minimum Stein discrepancy estimators. In *Neural Information Processing Systems*, pages 12964–12976, 2019.

A. Barp, C. J. Oates, E. Porcu, and M. Girolami. A Riemannian–Stein kernel method. *Bernoulli, (to appear), arXiv:1810.04946*, 2022.

Jonathan T Barron. Continuously differentiable exponential linear units. *arXiv preprint arXiv:1704.07483*, 2017.

D. Belomestny, A. Goldman, A. Naumov, and S. Sam-

sonov. Theoretical guarantees for neural control variates in MCMC. *arXiv:2304.01111*, 2023.

C. Blundell, J. Cornebise, K. Kavukcuoglu, and D. Wierstra. Weight uncertainty in neural network. In *International Conference on Machine Learning*, 2015.

F-X. Briol, C. J. Oates, M. Girolami, and M. A. Osborne. Frank-Wolfe Bayesian quadrature: Probabilistic integration with theoretical guarantees. In *Neural Information Processing Systems*, pages 1162–1170, 2015.

F-X. Briol, C. J. Oates, M. Girolami, M. A. Osborne, and D. Sejdinovic. Probabilistic integration: a role in statistical computation? *Statistical Science*, 34(1):1–22, 2019.

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

B. Calderhead and M. Girolami. Statistical analysis of nonlinear dynamical systems using differential geometric sampling methods. *Journal of Royal Society: Interface Focus*, 1:821–835, 2011.

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.

J. Cockayne, C. Oates, T. Sullivan, and M. Girolami. Bayesian probabilistic numerical methods. *SIAM Review*, 61(4):756–789, 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.

P. Diaconis. Bayesian numerical analysis. *Statistical Decision Theory and Related Topics IV*, pages 163–175, 1988.

V. Fortuin, A. Garriga-Alonso, S. W. Ober, F. Wenzel, G. Ratsch, R. E. Turner, M. van der Wilk, and L. Aitchison. Bayesian neural network priors revisited. In *International Conference on Learning Representations*, 2022.

A. Genz. Testing multidimensional integration routines. In *Proc. of international conference on Tools, methods and languages for scientific and engineering computation*, pages 81–94, 1984.

A. Gessner, J. Gonzalez, and M. Mahsereci. Active multi-information source Bayesian quadrature. In *Uncertainty in Artificial Intelligence*, 2019.

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

J. Gorham, A. Duncan, L. Mackey, and S. Vollmer. Measuring sample quality with diffusions. *Annals of Applied Probability*, 29(5):2884–2928, 2019.

A. Graves. Practical variational inference for neural networks. In *Neural Information Processing Systems*, 2011.

T. Gunter, R. Garnett, M. Osborne, P. Hennig, and S. Roberts. Sampling for inference in probabilistic models with fast Bayesian quadrature. In *Neural Information Processing Systems*, pages 2789–2797, 2014.

P. Hennig, M. A. Osborne, and M. Girolami. Probabilistic numerics and uncertainty in computations. *Journal of the Royal Society A*, 471(2179), 2015.

P. Hennig, M. A. Osborne, and H. Kersting. *Probabilistic Numerics: Computation as Machine Learning*. Cambridge University Press, 2022.

G. E. Hinton and D. van Camp. Keeping the neural networks simple by minimizing the description length of the weights. In *Conference on Computational Learning Theory*, page 5–13, 1993.

P. Izmailov, S. Vikram, M. D. Hoffman, and A. G. Wilson. What are Bayesian neural network posteriors really like? In *International Conference on Machine Learning*, 2021.

R. Jagadeeswaran and F. J. Hickernell. Fast automatic Bayesian cubature using lattice sampling. *Statistics and Computing*, 29(6):1215–1229, 2019.

M. Kanagawa and P. Hennig. Convergence guarantees for adaptive Bayesian quadrature methods. In *Neural Information Processing Systems*, pages 6237–6248, 2019.

M. Kanagawa, B. K. Sriperumbudur, and K. Fukumizu. Convergence analysis of deterministic kernel-based quadrature rules in misspecified settings. *Foundations of Computational Mathematics*, 20:155–194, 2020.

T. Karvonen and S. Särkkä. Fully symmetric kernel quadrature. *SIAM Journal on Scientific Computing*, 40(2):697–720, 2018.

T. Karvonen, C. J. Oates, and M. Girolami. Integration in reproducing kernel Hilbert spaces of Gaussian kernels. *Mathematics of Computation*, 90(331):2209–2233, 2020.

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.

A. Kirby, F-X. Briol, T. D. Dunstan, and T. Nishino. Data-driven modelling of turbine wake interactions and flow resistance in large wind farms. *arXiv:2301.01699*, 2023.

B. Lakshminarayanan, A. Pritzel, and C. Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. In *Neural Information Processing Systems*, 2017.

S. Lloyd, R. A. Irani, and M. Ahmadi. Using neural networks for fast numerical integration and optimization. *IEEE Access*, 8:84519–84531, 2020.

D. J. C. MacKay. A practical Bayesian framework for backpropagation networks. *Neural computation*, 4(3): 448–472, 1992.

D. J. C. Mackay. Probable networks and plausible predictions — a review of practical bayesian methods for supervised neural networks. *Network: Computation in Neural Systems*, 1995.

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

R. M. Neal. *Bayesian learning for neural networks*. Springer Verlag, 1996.

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

T. Nishino. Two-scale momentum theory for very large wind farms. *Journal of Physics: Conference Series*, 753 (3), 2016.

C. J. Oates and T. J. Sullivan. A modern retrospective on probabilistic numerics. *Statistics and Computing*, 29: 1335–1351, 2019.

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.

C. J. Oates, M. Girolami, and N. Chopin. Control functionals for Monte Carlo integration. *Journal of the Royal Statistical Society B: Statistical Methodology*, 79(3):695–718, 2017a.

C. J. Oates, S. Niederer, A. Lee, F-X. Briol, and M. Girolami. Probabilistic models for integration error in the assessment of functional cardiac models. In *Neural Information Processing Systems*, pages 110–118, 2017b.

C. J. Oates, J. Cockayne, F-X. Briol, and M. Girolami. Convergence rates for a class of estimators based on stein's method. *Bernoulli*, 25(2):1141–1159, 2019.

A. O'Hagan. Bayes–hermite quadrature. *Journal of Statistical Planning and Inference*, 29, 1991.

A. B. Owen. *Monte Carlo Theory, Methods and Examples*. 2013.

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.

T. Pearce, R. Tsuchida, M. Zaki, A. Brintrup, and A. Neely. Expressive priors in Bayesian neural networks: Kernel combinations and periodic functions. In *Conference on Uncertainty in Artificial Intelligence*, 2019.

C Rasmussen and Z Ghahramani. Bayesian Monte Carlo. In *Neural Information Processing Systems*, pages 489–496, 2002.

C. Rasmussen and C. Williams. *Gaussian Processes for Machine Learning*. MIT Press, 2006.

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.

H. Ritter, A. Botev, and D. Barber. A scalable Laplace approximation for neural networks. In *International Conference on Learning Representations*, 2018.

C. P. Robert and G. Casella. *Monte Carlo Statistical Methods*. Springer, 2000.

G. O. Roberts and R. L. Tweedie. Exponential convergence of langevin distributions and their discrete approximations. *Bernoulli*, pages 341–363, 1996.

Nicol N. Schraudolph. Fast curvature matrix-vector products for second-order gradient descent. *Neural Computation*, 14(7):1723–1738, 2002.

J. Shi, Y. Zhou, J. Hwang, M. K. Titsias, and L. Mackey. Gradient estimation with discrete Stein operators. In *Neural Information Processing Systems*, 2022.

S. Si, C. J. Oates, A. B. Duncan, L. Carin, and F-X. Briol. Scalable control variates for Monte Carlo methods via stochastic optimization. *Proceedings of the 14th Conference on Monte Carlo and Quasi-Monte Carlo Methods. arXiv:2006.07487*, 2021.

S. Sun, G. Zhang, J. Shi, and R. Grosse. Functional variational Bayesian neural networks. In *International Conference on Learning Representations*, 2019.

Z. Sun, C. J. Oates, and F-X. Briol. Meta-learning control variates: Variance reduction with limited data. *arXiv:2303.04756, to appear at UAI*, 2023.

R. Wan, M. Zhong, H. Xiong, and Z. Zhu. Neural control variates for monte carlo variance reduction. In *Machine Learning and Knowledge Discovery in Databases*. Springer International Publishing, 2020.

J. Wenger, N. Krämer, M. Pförtner, J. Schmidt, N. Bosch, N. Effenberger, J. Zenn, A. Gessner, T. Karvonen, F-X. Briol, M. Mahsereci, and P. Hennig. ProbNum: Probabilistic numerics in Python. *arXiv:2112.02100*, 2021.

A. G. Wilson and P. Izmailov. Bayesian deep learning and a probabilistic perspective of generalization. In *Neural Information Processing Systems*, 2020.

G. Wynne, F.-X. Briol, and M. Girolami. Convergence guarantees for Gaussian process means with misspecified likelihoods and smoothness. *Journal of Machine Learning Research*, 22(123):1–40, 2021.

X. Xi, F-X. Briol, and M. Girolami. Bayesian quadrature for multiple related integrals. In *35th International Conference on Machine Learning*, volume 12, pages 8533–8564, 2018.

H. Zhu, X. Liu, R. Kang, Z. Shen, S. Flaxman, and F-X. Briol. Bayesian probabilistic numerical integration with tree-based models. In *Neural Information Processing Systems*, pages 5837–5849, 2020.