SAND2025-11868R

SANDIA REPORT
SAND2025-238208
Printed September 2025

Machine learning models for PDE constrained optimization

Brian Chen

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185
Livermore, California 94550

ABSTRACT

Partial differential equation (PDE)-constrained optimization problems arise in a variety of scientific and engineering applications, such as topology optimization, electrodynamics, fluid dynamics, and structural dynamics. However, these problems are often challenging and computationally expensive to solve, due to the need to solve the PDEs within the optimization loop. One approach to reducing the computational cost of these methods while providing convergence guarantees is through inexact trust region methods [14, 15]; this method uses lower fidelity solutions of the PDE at early stages of the optimization and adjusts the required accuracy of inexact PDE solvers as the optimization progresses. In this work, we explore the use of machine learning based surrogate models with these inexact trust region methods. We first demonstrate the potential of this approach by using Gaussian processes as the surrogate model and test this on a simple PDE-constrained optimization problem. We then document explorations into improving the computational costs of evolutional deep neural network / neural Galerkin methods [6, 2], with the eventual goal of using these methods with the inexact trust region algorithms. We are able to speed up these approaches, albeit at the cost of lower accuracy.

This page intentionally left blank.

CONTENTS

This page intentionally left blank.

LIST OF FIGURES

This page intentionally left blank.

LIST OF TABLES

This page intentionally left blank.

1. INTRODUCTION

The solution of large scale optimization problems where the physics is modeled via partial differential equations (PDEs) is an important problem for a variety of scientific and engineering applications. Examples include topology optimization, electrodynamics, fluid dynamics, and structural dynamics. These problems can be computationally expensive, as the optimization procedure often requires solving PDEs many times in the inner loop of the optimization.

One approach to reducing the computational costs of PDE-constrained optimization is to use surrogate models, which provide faster but less accurate solutions of the PDE. Ideally, such an approach would still come with convergence guarantees. Our approach will be based on the inexact optimization methods developed in [14, 15], which provide provable guarantees on convergence by adjusting the required accuracy of the surrogate models as the optimization procedure proceeds. While [14, 15] consider sparse grid discretizations of the PDE, our motivation for this work is to develop and use machine learning based surrogate models with these inexact optimization schemes.

For use within these optimization schemes, the following are required of the surrogate models:

- Some method of estimating the accuracy of the machine learning surrogate model.
- Some method of improving the accuracy of these models to within some prescribed level of error.

For these surrogate models to be useful, we of course also want these methods to be computationally inexpensive. In this project, we explored two approaches: Gaussian processes (GPs) [22] and evolutional deep neural network / neural Galerkin methods [6, 2].

Gaussian process regression models functions as stochastic processes where the function's values at any finite set of points is assumed to be drawn from a multivariate normal distribution. Given some observed data (X,Y) and test points X_*, conditioning on the observed data provides a predicted distribution on the function values at the test points. This predicted distribution provides a means of estimating the uncertainty of the model at the test points. Furthermore, these models can be refined by adding additional data to (X,Y). These properties make GPs well-suited for the inexact optimization methods described above.

Our approach will be to use two separate GPs - one to model the objective function and one to model the gradient of the objective function with respect to the control variables. When the error estimate of the GP at a point x is above some prescribed tolerance, we can then evaluate the objective or gradient at x and use that data to update the GP.

We test this approach on a one dimensional Possion control problem, as a proof of concept for our overall approach of using machine learning based surrogate models with inexact trust region methods. We find that using the GP as a surrogate model does lead to improved optimization performance relative to using the exact objective and gradient information, in the case that Hessian information is unavailable.

While GPs are a promising approach, we note that scalability to problems in higher dimensions can be challenging. This is due to both potentially needing more data to accurately represent functions in high dimensions and potentially needing a larger control vector. Inference with GPs scales cubically with number of data points. In addition, we use multioutput GPs to model the gradient; inference also scales cubically with the size of the output vector. Given these challenges, a second thrust of this project was to explore the use of deep learning based surrogate models, with the hope that these scale better with dimension size.

We document efforts into improving the computational cost and accuracy of evolutional deep neural network / neural Galerkin methods, an approach to modeling time-dependent PDEs with neural networks. An advantage of these approaches is that while an initial optimization step is needed to match the initial condition of the time-dependent PDE, the solution at times $t > 0$ is determined via time stepping. The draw of these approaches is that they avoid a complicated optimization procedure that has to fit both the initial condition as well as a PDE residual. Furthermore, the time-stepping procedure allows one to measure a residual at every time step, giving a measure of the inaccuracy incurred at every time step.

However, there are several challenges for these methods. In particular, one challenge is finding parameters θ_0 such that the neural network u_{θ_0} is close to the initial condition. Achieving very high accuracy of this initial fit is important for the overall performance of the method, but can be computationally expensive. Since we are interested in using these models as surrogate models in place of traditional numerical solvers, reducing the computational time needed for these approaches is important. Secondly, while a residual error can be estimated at every time step, we are not aware of theory on the overall accuracy of neural Galerkin methods.

In this project, we mainly focus on the first challenge. We test a couple of approaches for improving the computational time and accuracy of matching the initial condition: using Fourier neural networks and adding a corrective residual term to the initial neural network. These methods are tested on the heat equation and a heat equation with a source term, both on unbounded space. While the Fourier neural networks may not be suitable for these unbounded domain problems, we found that adding a corrective term does improve training time, but at the cost of increased error. For the two dimensional problems, we are able to achieve moderate levels of accuracy (10^{-3} to 10^{-4} relative L^2 error). Given the accuracy and computational costs of traditional numerical methods, we hypothesize that neural Galerkin approaches may be more comparatively advantageous for higher dimensional PDEs.

2. METHOD

This section quickly summarizes the inexact trust region methods developed in [14, 15], which provide provable convergence guarantees. Next, we discuss Gaussian processes, which provide error estimates and an ability to update the model. This capability makes GPs a convenient class of models to use with the inexact trust region methods. Finally, we discuss evolutional deep neural networks / neural Galerkin methods and work towards improving the computational costs of these approaches.

2.1. Inexact Trust Region Methods

Developing optimization methods that utilize inexact function and gradient evaluations, while still providing optimization guarantees, has been an active area of research. The particular approach we work with in this project are the inexact trust region methods developed in [14, 15]. At early stages of optimization, these methods utilize lower fidelity solutions of the PDE. However, as the optimization continues, the required fidelity of the solution increases. This can be achieved by using, for instance, adaptive finite elements, where the mesh is adaptively refined to obtain the desired levels of accuracy.

These inexact trust region methods require that the model for the objective and gradient can produce sufficiently accurate estimates. Let f be the objective function, J be the surrogate model, k be the iteration number of the optimization procedure, and z_k be the current iterate. At each iteration of a trust region method, a trial step s_k is determined by minimizing a model function m_k (often a quadratic) in a "trust region" around the current iterate z_k. This step is then accepted or rejected depending on if the step yields a large enough decrease in the objective function; the size of the trust region is also adjusted.

The inexact trust region method requires that the computed reduction using the surrogate model J (not to be confused with the trust region model m_k) in taking the trial step s_k

$$\mathrm{cred}_k = J(z_k) - J(z_k + s_k) \tag{2.1}$$

is close to the actual reduction in the objective function

$$\mathrm{ared}_k = f(z_k) - f(z_k + s_k). \tag{2.2}$$

In particular, the condition from (4.8) in [14] is given by

$$|\mathrm{ared}_k - \mathrm{cred}_k| \leq \kappa_{obj} \left(\eta \min \left\{\mathrm{pred}_k, r_k\right\}\right)^{1/\omega}, \tag{2.3}$$

where η is a fixed parameter constrained by the settings of the trust region algorithm, r_k is a sequence of non-negative numbers such that $\lim_{k\to\infty} r_k = 0$, $\kappa_{obj} > 0$ and $\omega \in (0,1)$ are fixed constants, and pred_k is the predicted reduction of the trust region model:

$$\text{pred}_k = m_k(0) - m_k(s_k), \tag{2.4}$$

The condition on the accuracy of the gradient g_k is given by (4.3) in [14]:

$$\|g_k - \nabla f(z_k)\| \leq \kappa_{grad} \min\{\|g_k\|, \delta_k\}, \tag{2.5}$$

where δ_k is the radius of the trust region and $\kappa_{grad} > 0$ is a fixed constant.

As such, we will need the ability to estimate the error in our machine learning based surrogate models, as well as an ability to refine the models to be accurate within the tolerances specified by (2.3) and (2.5).

With this framework in mind, we explored two machine learning based surrogate models for use with these inexact trust region methods: Gaussian processes and evolutional deep neural network / neural Galerkin approaches.

2.2. Gaussian Process Regression

Gaussian process regression is a commonly used regression model, where a function f is modeled as a Gaussian process; that is, the function's values at any set of points is assumed to be distributed according to a multivariate normal distribution. The Gaussian process is specified by its mean function $m(x)$ and covariance function $k(x,x')$. Common forms for the mean function include the zero mean function or a linear combination of basis functions. Common forms for the covariance function include the squared exponential covariance function, given by

$$k(x,x') = \exp\left(-\frac{\|x-x'\|_2^2}{2l^2}\right), \tag{2.6}$$

and the Matérn class of kernels given by

$$k(x,x') = \frac{2^{1-\nu}}{\Gamma(\nu)}\left(\frac{\sqrt{2\nu}\|x-x'\|_2}{l}\right)^{\nu} K_\nu\left(\frac{\sqrt{2\nu}\|x-x'\|_2}{l}\right). \tag{2.7}$$

The parameter $l > 0$ in these covariance functions is often referred to as the lengthscale parameter. For the Matérn class of kernels, $\nu > 0$ is an additional parameter, K_ν is a modified Bessel function, and Γ is the gamma function. A reference for these covariance functions and Gaussian process regression in general is [22].

Given a set of points X at which the function f has been evaluated and a set of test points X_*, the Gaussian process prior yields a joint distribution between the vector of function values Y and Y_* at the points X and X_*:

$$\begin{bmatrix} Y \\ Y_* \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} M(X) \\ M(X_*) \end{bmatrix}, \begin{bmatrix} K(X,X) & K(X,X_*) \\ K(X_*,X) & K(X_*,X_*) \end{bmatrix}\right). \tag{2.8}$$

If A and B are a sets of points, we use $K(A, B)$ to denote the matrix $[k(a_i, b_j)]_{i,j}$ and $M(A)$ to denote the vector $[m(a_i)]_i$. Given the observed values Y, the predicted distribution of Y_* is obtained by conditioning this distribution on the observations Y, yielding:

$$\begin{aligned} Y_*|X_*, X, Y \sim \mathcal{N}(&K(X_*, X)K(X, X)^{-1}Y, && (2.9)\\ &K(X_*, X_*) - K(X_*, X)K(X, X)^{-1}K(X, X_*)) && (2.10) \end{aligned}$$

See Section 2.2 of [22] for a reference.

2.2.1. GPs within Inexact Trust Region Methods

Gaussian processes nicely fit into the inexact trust region framework. First, note that the Gaussian process predicts a distribution for the value of f at a point x. As such, we can use this distribution to estimate the uncertainty in the prediction. In particular, from (2.10), we can calculate the covariance matrix for the predicted output. In our experiments, we will use two times the sum of the standard deviations as our error estimate.

Secondly, updating the model with new data is trivial, as it only involves appending the new data to data matrices X and Y and possibly updating hyperparameters for the GP. Note that GPs interpolate between known data points (assuming no noise in the outputs); it can be shown that the posterior distribution at a point x where x is in the observed data X is a normal distribution with mean equal to the observed y and zero covariance matrix. Thus, when we need to update the GP model to be below some given error tolerance at a point x, evaluating the function at x and adding this data point to (X, Y) will reduce the estimated error at x to zero.

2.2.2. Experiments

The approach is tested on the following PDE-constrained optimization example:

$$\begin{aligned} \min_z \quad & \left(\frac{1}{2}\|u - w\|_2^2 + \frac{\alpha}{2}\|z\|^2\right) \\ \text{subject to} \quad & -\frac{\partial}{\partial x}\left(e^z \frac{\partial u}{\partial x}\right) = \begin{cases} 1, & 0 \le x < 0.5 \\ 2, & 0.5 \le x \le 1 \end{cases} \\ & u(0) = u(1) = 1, \end{aligned} \tag{2.11}$$

where $w(x) = x(1 - x)$.

2.2.3. Implementation

In this project, we developed a general interface that can use surrogate models defined in Python with the inexact trust region methods implemented in the Rapid Optimization Library [10] (ROL), a high performance C++ library for numerical optimization. This interface is built on the PyROL interface first developed in LDRD 229308 “Fractional-Order Models for Sea Ice (FOMSI)” and continued in LDRD 233068 "SNOML: Stochastic Nonsmooth Optimization for Machine Learning." The advantage of such an approach is that many machine learning models are written in Python, whereas ROL is a high performance optimization library that provides many optimization methods that adaptively control inexact function and gradient evaluations.

The Gaussian processes are implemented in Python, using the Botorch [1] package. The inexact trust region methods are implemented in ROL; the two are connected via the interface described above.

2.3. Neural Galerkin Methods

In this project, we have also explored the evolutional deep neural network / neural Galerkin method [6, 2] as a potential surrogate model for modeling time dependent PDEs:

$$\begin{aligned} \frac{\partial}{\partial t} u(x,t) &= \mathcal{F}(u,x,t) \\ u(x,0) &= g(x) \end{aligned} \tag{2.12}$$

The approach models the solution of the time dependent PDE with a time indexed sequence of parameters θ_t such that the neural network u_{θ_t} with these parameters approximates the solution at time t. This is done by first finding parameters θ_0 such that $u_{\theta_0} \approx g$. Then, one can derive the following system of differential equations:

$$M(\theta(t))\theta'(t) = V(t, \theta(t)), \tag{2.13}$$

where

$$M_{ij} := \int \frac{\partial}{\partial \theta_i} u_\theta(x) \frac{\partial}{\partial \theta_j} u_\theta(x) dx \tag{2.14}$$

$$V_i := \int \left(\frac{\partial}{\partial \theta_i} u_\theta(x) \right) \mathcal{F}(u_\theta(x), x, t) dx \tag{2.15}$$

Any ordinary differential equation solver can then be used to update θ and arrive at a sequence of parameters θ_t such that θ_t approximates the solution at time t.

Compared to physics-informed neural networks (PINN) [21], the neural Galerkin method simplifies the optimization procedure as the solution at times $t > 0$ is determined by time-stepping via the analytical equations given in (2.13), (2.14), and (2.15). Furthermore, this approach allows the following residual term to be measured at every time step, thus giving a measure of accuracy of each time step:

$$\left\| \mathcal{F}(u_{\theta_t}(x), x, t)\delta t - \left(u_{\theta_{t+\delta t}(x) - u_{\theta_t}(x)} \right) \right\|$$

However, some challenges with using neural Galerkin methods with inexact trust region methods are:

- Estimation and reduction of the errors in neural Galerkin methods.
- Computational cost of matching the initial condition g.

In this project, we focus mostly on the problem of reducing the computational cost of fitting θ_0 such that $u_{\theta_0} \approx g$. Developing theory around the accuracy of these methods is left to future work.

We explore two approaches. We explore using randomized Fourier neural networks, trained via the method described in [3]. The approach uses a Markov Chain Monte Carlo sampling procedure to iteratively update the network, which avoids the need for gradient based optimization and maintains error control. The second approach we consider is using standard feedforward neural networks, but adding a corrective term to the network to exactly fit the initial condition g, up to machine precision.

2.3.1. Randomized Fourier Neural Networks

In our project, we explored the use of randomized Fourier neural networks [19, 20, 11, 12, 3]. These networks employ complex-exponential activation functions. Rather than training the network via gradient based methods, we instead use the approach taken in [3]. This approach uses a Markov Chain Monte Carlo sampling approach to iteratively train layers of the network and provides a theoretical approximation rate as the number of layers is increased. Such an approach is promising given the goal of matching the initial condition quickly and with a very high degree of accuracy.

2.3.2. Adding corrective residual term

Another approach we explore is simply adding a corrective term to a feedforward neural network that has been only "partially" initialized. In other words, we take a reduced number of optimization steps in finding θ_0. We can then add on a corrective term as follows:

$$v_\theta = u_\theta + \alpha(-u_{\theta_0} + g), \tag{2.16}$$

where α is set to 1. We note that, by construction, v_θ will match the initial condition exactly (up to machine precision). In our experiments, we try to improve the computational costs and accuracy of the initial fit by adding this corrective term to a network that has been trained using fewer iterations of a gradient based optimization procedure.

Concretely, we find parameters θ_0 by minimizing the loss term

$$\left\|u_{\theta_0}(x) - g(x)\right\|_2^2 dx + \sum_{i=1}^{d} \left\|\frac{\partial}{\partial x_i} u_{\theta_0}(x) - \frac{\partial}{\partial x_i} g(x)\right\|_2^2, \tag{2.17}$$

where d is the dimension of the state space. This loss term is estimated by taking samples uniformly from a fixed domain and minimized using the Adam optimizer [13], with a cosine decay learning rate schedule [16]. In our experiments, we compare the case where the optimization is run for 10^6 iterations, to the case where the optimization is run for 10^5 iterations, with the corrective term added.

2.3.3. Higher dimensional PDEs

We also test neural Galerkin methods on higher dimensional PDEs. Since the method requires estimating the integrals in (2.14) and (2.15), additional care has to be taken to accurately estimate these integrals in high dimensions. To do this, we use a normalizing flow architecture [5, 8] to model u. Since these architectures model probability distributions and allow sampling according to these distributions, we can use these distributions to perform importance sampling for estimating (2.14) and (2.15). The use of normalizing flows for the neural Galerkin method has been explored in [23, 24]. In contrast to [23], we remove the need for the underlying state variable to be a probability density by also modeling a scale factor. In contrast to [24], we model u on the continuum rather than on a grid.

2.3.4. Experiments

We test these two approaches on the heat equation. First we consider the linear heat equation with a Gaussian initial condition, in unbounded space.

$$\begin{aligned} \frac{\partial u}{\partial t} &= \sum_{i=1}^{d} \frac{\partial^2 u}{\partial x_i^2} \\ u(x,0) &= \frac{1}{(2\pi)^{d/2}} \exp\left(\frac{1}{2}\sum_{i=1}^{d} x_i^2\right) \end{aligned} \tag{2.18}$$

We also consider the heat equation with an added time-varying source term $f(x,t)$

$$\begin{aligned} \frac{\partial u}{\partial t} &= \sum_{i=1}^{d} \frac{\partial^2 u}{\partial x_i^2} + f(x,t) \\ u(x,0) &= \frac{1}{(2\pi)^{d/2}} \exp\left(\frac{1}{2}\sum_{i=1}^{d} x_i^2\right) \end{aligned} \tag{2.19}$$

The source term $f(x,t)$ is chosen so that the solution to the PDE is given by

$$u(x,t) = \frac{1}{2(4\pi(t+0.5))^{d/2}} \left[\exp\left(-\frac{\sum_i (x_i + \lambda \sin(t\pi))^2}{4t+2}\right) + \exp\left(-\frac{\sum_i (x_i - \lambda \sin(t\pi))^2}{4t+2}\right)\right] \tag{2.20}$$

Letting

$$f^{+}(x,t) = \frac{1}{(4\pi(t+0.5))^{d/2}} \exp\left(-\frac{\sum_i (x_i + \lambda \sin(t\pi))^2}{4t+2}\right) \tag{2.21}$$

$$f^{-}(x,t) = \frac{1}{(4\pi(t+0.5))^{d/2}} \exp\left(-\frac{\sum_i (x_i - \lambda \sin(t\pi))^2}{4t+2}\right), \tag{2.22}$$

$f(x,t)$ is given by:

$$f(x,t) = \frac{\lambda\pi\cos(\pi t)}{2(2t+1)} \left[-f^{+}(x,t)\sum_i (x_i + \lambda\sin(\pi t)) + f^{-}(x,t)\sum_i (x - \lambda\sin(\pi t))\right] \tag{2.23}$$

2.3.5. *Training and Architecture Details*

For the two dimensional examples, we train the networks to match the initial condition g using samples from $[-10, 10]^2$. The randomized Fourier neural networks (Section 2.3.1) are trained using a fixed sample of $100,000$ samples from $[-10, 10]^2$. The training procedure is described in [3]. The network is trained with a maximum depth of 20 and width 10, leading to a total of 810 parameters.

The feedforward networks (Section 2.3.2) are trained by minimizing (2.17) via the Adam optimizer [13] with a cosine learning rate schedule [16] and an initial learning rate of 0.001. The loss term (2.17) is estimated using $10,000$ randomly sampled points from $[-10, 10]^2$. We test two cases - one where this optimizer is ran for $1,000,000$ iterations, and another where the optimizer is ran for $100,000$ iterations with the correction term (2.16) added. The networks use tanh activations, with 10 hidden layers and 10 nodes in each hidden layer. The network has 921 parameters.

For the five dimensional experiments (Section 2.3.3), we use a RealNVP architecture [5] for the normalizing flow. The normalizing flows are trained using the same approach as the feedforward network, except with samples drawn from a five dimensional standard normal distribution. Note that for our examples, the initial condition is the probability density function of a standard normal distribution. The RealNVP architecture uses a standard normal distribution as the base density. We use 8 coupling layers, where each coupling layer has one hidden layer with width 10, yielding a network with 721 parameters.

After fitting the initial condition g, time stepping is performed using an explicit RK4 time stepper to estimate the solution to (2.13). For the two dimensional examples, the quantities (2.14) and (2.15), used in (2.13), are estimated using $10,000$ samples uniformly sampled from $[-10, 10]^2$. In the five dimensional case, (2.14) and (2.15) are estimated using samples drawn from the normalizing flow. For these experiments, we run RK4 for 1000 time steps, with a time step $\delta t = .001$.

2.3.6. *Implementation*

The neural Galerkin method and feedforward networks are implemented in Pytorch [18]. Training for the randomized Fourier neural networks to match the initial condition g is performed using code from the authors of [3]. The Fourier neural networks are then converted into Pytorch models, before being used with the neural Galerkin method.

This page intentionally left blank.

3. RESULTS AND DISCUSSION

We first discuss the results of applying GPs in the framework of inexact trust region methods for a one dimensional PDE constrained optimization problem. After demonstrating the potential of using machine learning based surrogate models with inexact trust region methods with this example, we delve into experiments with neural Galerkin methods. The neural Galerkin experiments test some approaches towards improving the computational costs and accuracy of fitting a network to an initial condition, which is a computationally expensive bottleneck of these approaches.

3.1. Gaussian Process results

We test our approach of using Gaussian processes within inexact trust region methods on the following PDE-constrained optimization problem, as a proof of concept:

$$\begin{aligned} \min_{z} \quad & \left(\frac{1}{2}\|u-w\|_2^2+\frac{\alpha}{2}\|z\|^2\right) \\ \text{subject to} \quad & -\frac{\partial}{\partial x}\left(e^z\frac{\partial u}{\partial x}\right)=\begin{cases}1, & 0\le x<0.5\\ 2, & 0.5\le x\le 1\end{cases} \\ & u(0)=u(1)=1, \end{aligned} \tag{3.1}$$

where $w(x)=x(1-x)$.

The control variable z is discretized on 32 points. As described earlier, two Gaussian processes are used to model the objective function and its derivative with respect to z. The Hessian of the objective with respect to the control is modeled by taking the Jacobian J_{GP} of the Gaussian process modeling the gradient and symmetrizing the Jacobian: $J=\frac{1}{2}\left(J_{GP}+J_{GP}^T\right)$.

Plots of the objective function versus number of iterations for the inexact trust region method with Gaussian processes and the exact trust region method are shown in Figure 3-1. Note that we compare against both the case where the Hessians are estimated with L-BFGS and the case where the exact Hessian is calculated. When only the objective and gradients are available, the use of Gaussian processes with the inexact trust region method does improve optimization. On the other hand, if Hessian information is available, the exact trust region method does converge faster than the inexact trust region method with GPs. We note, however, that the Hessian is not made available to the inexact trust region method with GPs.

This example serves primarily as a proof of concept for the overall method of using machine learning-based surrogate models with inexact trust region methods. Although Gaussian processes

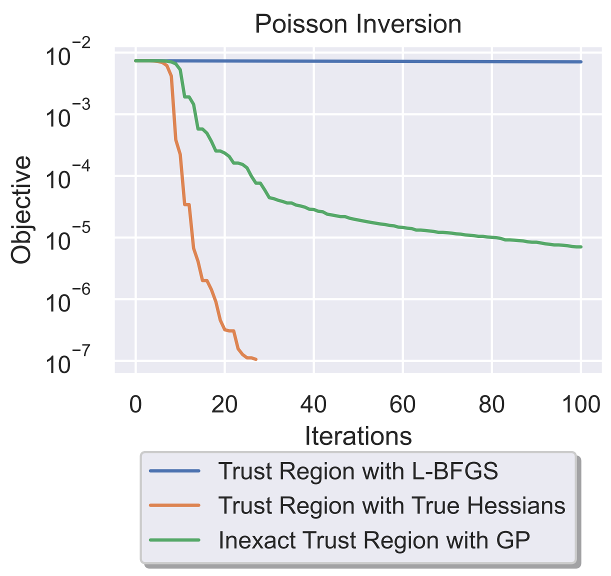


Figure 3-1. Comparison of the exact trust region method against the inexact trust region method with GP surrogates on the PDE-constrained optimization problem (3.1). When exact Hessians are not available, the inexact trust region method with the GP surrogate models led to improved optimization over the trust region method using exact objective and gradient evaluations with L-BFGS estimated Hessians.

provide a convenient way to estimate the error and update the model, we note that there are challenges to scaling up Gaussian processes. Two scalability issues are as follows:

- Modeling problems in higher dimensions may require more data points to accurately model the desired function. The evaluation cost of GPs scales cubically with the number of data points. See, e.g., [22].
- In the approach described, we model the gradient of the objective with respect to the control using a multioutput GP. The evaluation cost of multioutput GPs scales cubically with the size of the output vector [7, 4, 17], which poses a challenge if the control vector is large.

Due to these scalability challenges, we have also in parallel explored the use of deep learning architectures as surrogate models for PDEs.

3.2. Neural Galerkin results

A second thrust of the project was to explore neural Galerkin methods for time-dependent PDEs. A large computational cost in neural Galerkin methods is finding parameters θ_0 such that the neural network with parameters θ_0 approximates the initial condition $g(x)$, that is, $u_{\theta_0} \approx g$. To that end, we tested two approaches for fitting the initial condition:

- Random fourier neural networks
- Adding a corrective term to the network

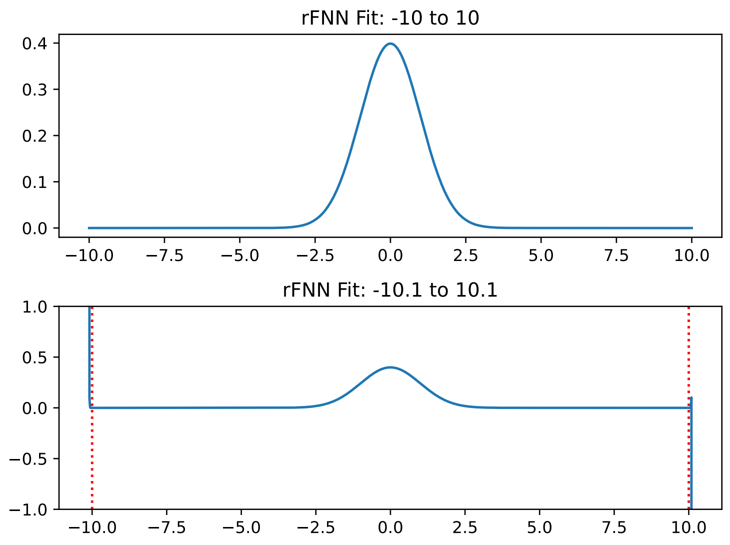


Figure 3-2. While the rFNN training process is able accurately match the initial condition on the interval on which it is trained, the rFNN does overfit outside of this interval.

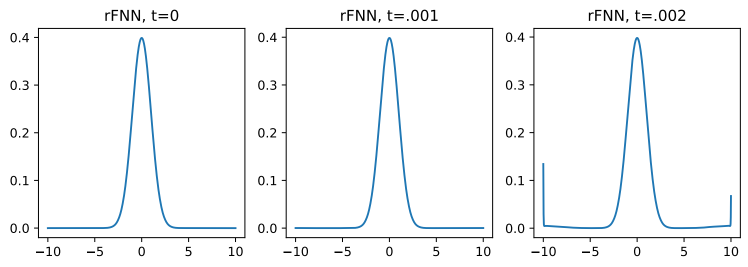


Figure 3-3. The neural Galerkin method with rFNNs yielded poor accuracies for the unbounded domain problems we considered. This seems to be due to the overfitting of the rFNN, as illustrated in Figure 3-2.

3.2.1. Random Fourier Neural Networks

We tested random Fourier neural networks, trained via a Markov Chain Monte Carlo sampling approach developed in [3]. Such a training procedure has several nice properties, including a theoretical approximation rate as the number of layers is increased.

However, for the unbounded domain problems we considered, the network overfits points outside the domain for which the network is trained. As shown in Figure 3-2, while the network is very accurate on the training domain $[-10, 10]$, the errors are large outside of this domain.

The errors outside of $[-10, 10]$ seem to negatively impact the neural Galerkin solution. In Figure 3-3, we plot the neural Galerkin solution using the rFNN after a few time steps.

Additional work in properly handling the artificial boundary will likely improve this approach. These networks may also be useful on problems with bounded spatial domains. Given the theoretical properties of rFNNs and the ability to avoid an expensive gradient based optimization procedure, this remains a promising approach as a neural network architecture for neural Galerkin methods.

Time (s)	Finding θ_0	Time stepping	Total
Neural Galerkin	8345	421	8766
Neural Galerkin with Residual	837	655	1492

Table 3-1. Timings for fitting the initial condition and for time stepping for the heat equation example (2.18). Training the network to fit the initial condition with fewer iterations and adding the corrective residual term 2.16 does reduce the time needed to fit the initial condition, but does increase the time stepping costs. Overall, this yields about a 83% reduction in overall end-to-end computational time.

3.2.2. Adding a corrective term

We test the neural Galerkin method using a neural network where the initial parameters θ_0 are found after running the Adam optimizer for 10^6 iterations, compared to a network with parameters θ_0 found after 10^5 iterations with the corrective residual term (2.16) added. In this section, we will refer to the former as "Neural Galerkin" and the latter as "Neural Galerkin with Residual."

3.2.2.1. Heat equation

We first test on the heat equation in two dimensions, given by (2.18). The training costs to find θ_0 and the time stepping cost are shown in Table 3-1. As noted previously, the initial step of fitting the neural network to the initial condition can be computationally expensive. By training the network for 90% fewer iterations and adding on a corrective term, we are able to predictably reduce the initial training time by about 90%.

A plot of the solutions compared to the ground truth are shown in Figure 3-4. The neural Galerkin solutions are qualitatively similar to each other and the ground truth.

As shown in Figure 3-5, while adding on this residual term leads the network to fit the initial condition exactly up to machine precision (by construction), this does lead to a higher error at later times.

3.2.2.2. Heat equation with source term

We repeat the experiment with the heat equation with source term example (2.20). The time-varying source term is chosen such that the solution exhibits two modes that separate and come back together, as shown in Figure 3-4.

As before, training the network for fewer iterations and adding the corrective residual term reduces the total computational time by about 83%, as shown in Table 3-2.

The neural Galerkin solutions are plotted against the ground truth in Figure 3-6. As before, the neural Galerkin solutions are qualitatively similar to the ground truth. In particular, these methods are able to capture the behavior of the Gaussian separating into two Gaussians before coming back together.

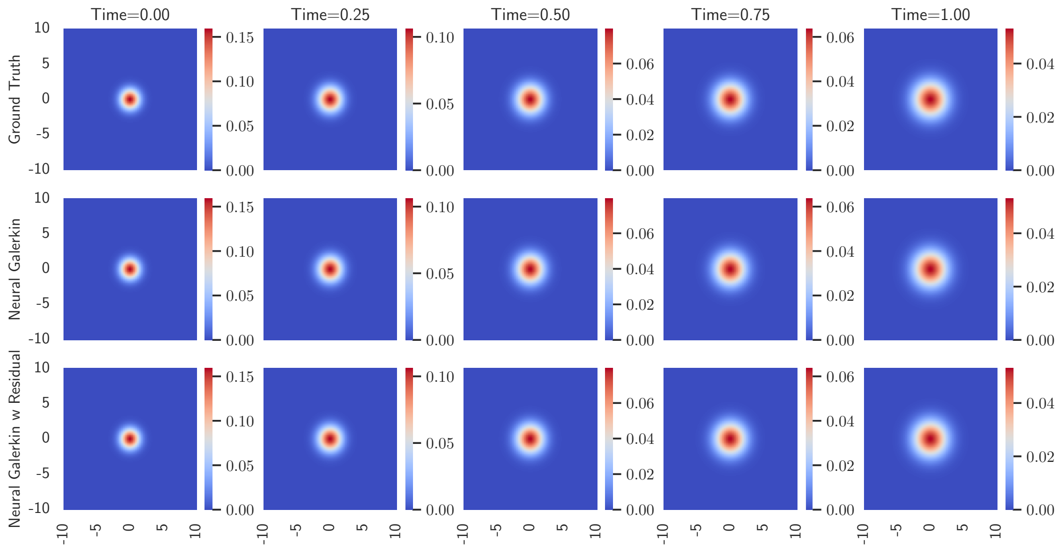


Figure 3-4. Neural Galerkin solutions and the ground truth for the 2D heat equation example (2.18). The neural Galerkin solutions qualitatively line up with the ground truth.

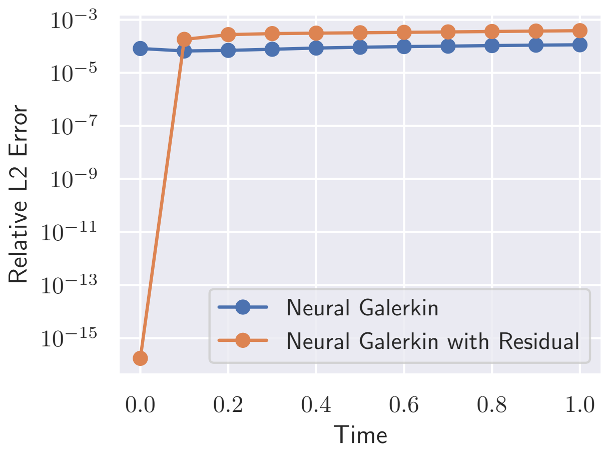


Figure 3-5. Relative L^2 accuracy of the neural Galerkin method, when using the fully initialized network ("Neural Galerkin") compared to the partially initialized network with residual term ("Neural Galerkin with Residual"), for the heat equation example (2.18). While using the partially initialized model does lead to improved computational speeds, and does match the initial condition exactly, this does lead to lower accuracy.

Time (s)	Finding θ_0	Time stepping	Total
Neural Galerkin	8345	496	8841
Neural Galerkin with Residual	837	649	1486

Table 3-2. Timings for fitting the initial condition and for time stepping for the heat equation with source term example (2.19). As in Table 3-1, training the network for fewer iterations and adding the corrective residual term reduces end-to-end computational time by about 83%.

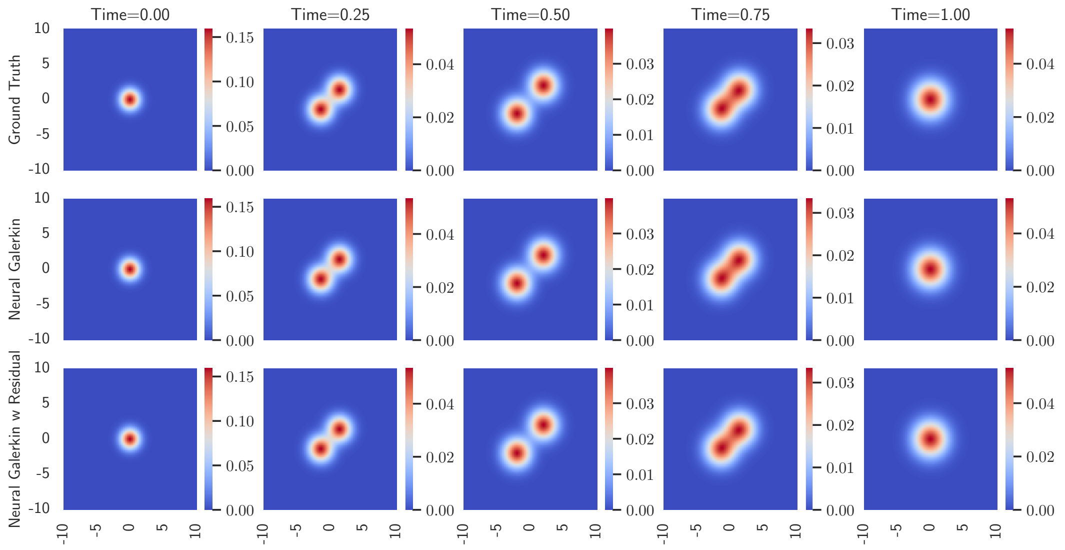


Figure 3-6. Neural Galerkin solutions and the ground truth for the 2D heat equation with source term example (2.19)**. The neural Galerkin solutions qualitatively line up with the ground truth.**

Relative L^2 accuracies of the approaches are shown in Figure 3-7. In this example, we were able to achieve faster computational times while maintaining a similar level of accuracy.

3.2.3. Higher dimensional problems

In the two dimensional examples, we were able to achieve moderate levels of accuracy, on the order of 10^{-3} to 10^{-4}. However, for these problems in two dimensions, traditional numerical methods work well, are well-studied, and come with theoretical guarantees on accuracy. As such, neural Galerkin methods and other deep learning approaches may be more beneficial for high dimensional problems [2, 9], where traditional numerical methods can struggle due to the curse of dimensionality.

We test the neural Galerkin approach for (2.18) and (2.19) in five dimensions. As described in Section 2.3.3, we use a normalizing flow architecture. The relative L^2 accuracies are shown in Figure 3-8. While we are able to achieve accurate results for the heat equation (on the order of 10^{-5}), we were only able to achieve errors on the order of 10^{-1} for the heat equation with source term. Further research into improving the accuracy of these methods, especially in high dimensions, would be needed to apply these methods for high-dimensional PDEs.

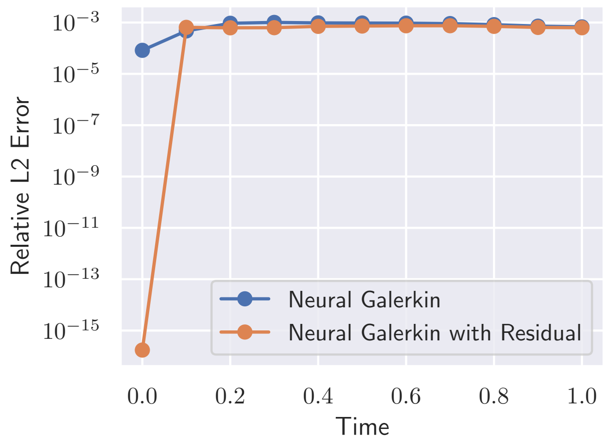
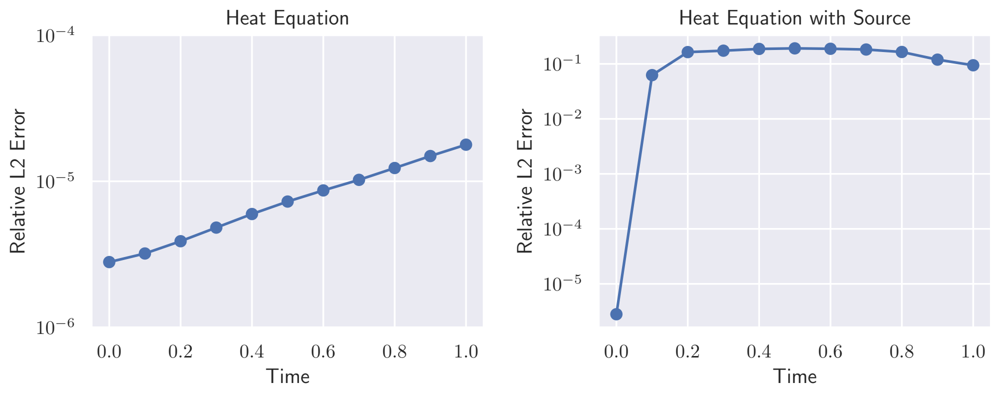


Figure 3-7. Relative L^2 accuracy of the neural Galerkin method, when using the fully initialized network ("Neural Galerkin") compared to the partially initialized network with residual term ("Neural Galerkin with Residual"), for the heat equation with source term example (2.19). For this example, we are able to achieve improved computational speeds for the initial fit, while yielding about the same level of accuracy at times $t > 0$.



(a) Heat equation **(b) Heat equation with source term**

Figure 3-8. Relative L^2 accuracies of the neural Galerkin method for the five dimensional heat equation (2.18) and heat equation with source term examples (2.19). We achieve accurate results for the heat equation example, with relative errors on the order of 10^{-5}, but not for the heat equation with source example, with relative errors on the order of 10^{-1}.

This page intentionally left blank.

4. ANTICIPATED OUTCOMES AND IMPACTS

During the lifetime of this project, this work was presented both internally and externally. This work was presented at the internal 1400 Postdoctoral and Early Career Seminar in April 2025, as well as at the public Sandia Machine Learning / Deep Learning workshop in August 2025.

This project led to the development of a general framework to use surrogate models defined in Python with the Rapid Optimization Library (ROL), a high-performance C++ library for numerical optimization developed at Sandia. This framework builds on the "PyROL" interface connecting Python with ROL, first developed in the "Fractional-Order Models for Sea Ice (FOMSI)" LDRD [229308] and further developed in the ongoing LDRD "SNOML: Stochastic Nonsmooth Optimization for Machine Learning" [233068]. The framework was designed with the goal of being agnostic to the type of surrogate model being used; while we tested our approach on Gaussian processes and neural networks, other types of surrogate models such as polynomial interpolation, could also be used with this framework.

This work has also led to a collaboration with the Rapid Optimization Library team (Bobby Baraldi, Aurya Javeed, Drew Kouri, Denis Ridzal, Greg von Winckel, and Radoslav Vuchkov (all in 01463)) in writing a software paper, detailing the capabilities of the package and providing user-friendly examples written in Python using the aforementioned PyROL interface. An outcome of this project has been contributions to this upcoming paper, in particular the machine learning and inexact trust region example sections.

This project has received follow-on funding to complete our investigation of using machine learning surrogate models with inexact optimization procedures. This funding comes from the Advanced Scientific Computing Research (ASCR) Early Career Project (PI: Drew Kouri [01463]) entitled "Adaptive and Fault-Tolerant Algorithms for Data-Driven Optimization, Design, and Learning".

The authors will also seek out collaborations across the laboratory, especially if results on high dimensional partial differential equations like the Vlasov equation and Boltzmann equation are promising. These equations have applications to plasma physics and thermodynamics problems. We have initiated conversations with people across the laboratory to understand their use cases of machine learning-based surrogate models for PDEs, including Sharlotte Kramer [01528] and Reese Jones [08363]. In terms of improvements to the neural Galerkin method, we have initialized conversations with Jonas Actor [01441].

As described in this manuscript, this project has explored two different surrogate model approaches. Both methods are promising, and we anticipate fruitful research directions in both. For Gaussian processes:

- Although Gaussian processes show promise in the low dimensional example problem (a Poisson inversion problem in one dimension) considered in this manuscript, scaling up

Gaussian processes to high dimensional problems is challenging. This stems from two difficulties:

  - Higher dimensional problems will likely require more data to model accurately, and the computational costs of GPs scale cubically in the number of data points.
  - In our work, we model the gradient of the control with a multioutput GP. The computational costs of a multioutput GP scale cubically with the number of outputs. For higher dimensional problems, we will likely need to consider larger control vectors (e.g., consider a control function discretized on a grid) and thus a larger gradient vector.

  Many approaches have been developed to improve the scalability of GPs, often by introducing approximations of the "full" GP. An area of future research will be testing these approaches with the inexact optimization framework and investigating the impact these approximations have on the optimization procedure.

On the other hand, for neural Galerkin methods, two areas of future research are:

- High-dimensional PDEs like the Vlasov equation and Boltzmann equation are challenging. It will be interesting to explore if the neural Galerkin method can yield reasonably accurate results for these difficult PDEs in high dimensions. If these methods do hold promise for these PDEs, the next step will be then to use these methods as a subroutine in PDE-constrained optimization problems for plasma physics and thermodynamics applications.
- For neural Galerkin methods to be used with the inexact trust region methods in this manuscript, we require error estimates of these methods. We conjecture that the time stepping procedure of the approach as well as the measurable residual at each time step will allow for error estimates to be developed, but this is left as an avenue for future research.

5. CONCLUSION

Our work sought to use machine learning based surrogate models with inexact trust region algorithms, with the aim of reducing the computational cost of solving PDE-constrained optimization problems while maintaining convergence guarantees. We first test Gaussian processes, since these models provide a means of estimating the error in predictions as well as an easy way to update the model to reduce this error below a given threshold; these two capabilities are needed to use the inexact trust region methods developed in [14, 15]. We test the inexact trust region method with Gaussian processes on a one-dimensional Poisson inversion problem, demonstrating the potential for using machine learning based surrogate models with inexact trust region methods.

We also investigated deep learning based methods for modeling time-dependent PDEs, in particular the evolutional deep neural network / neural Galerkin methods. These methods were appealing due to the solution at times $t > 0$ being determined via time-stepping, thus potentially reducing the computational cost of these approaches. Furthermore, these methods allow for measuring a residual error at every time step. Fitting the neural networks to an initial condition is the only optimization step required with these methods and is computationally expensive, so we explored approaches to reducing this computational cost. We test these approaches on a heat equation example and a heat equation example with a source term added. While we are able to reduce this initial cost, this led to reduced accuracy in one of the examples (and similar accuracies in the other). Overall, we achieved moderate levels of accuracy (10^{-3} to 10^{-4}) for these two dimensional problems.

Some future areas of research include:

- Improving the scalability of GPs: Multioutput GPs scale cubically in both the amount of data as well as the number of outputs. This serves as a challenge for our approach for higher dimensional problems, as more data will likely be needed to model high dimensional problems accurately. Furthermore, since we model the gradient of the control with a multioutput GP, larger control vectors may also prove challenging.
- Error estimates for neural Galerkin methods: Can error estimates be developed by leveraging the ability to measure a residual error at every time step?
- Higher dimensional PDEs: Can neural Galerkin methods be used to solve high- dimensional PDEs like the Vlasov equation or Boltzmann equation, and if so, can they be used to speed up PDE constrained optimization?

This page intentionally left blank.

REFERENCES

[1] Maximilian Balandat, Brian Karrer, Daniel R. Jiang, Samuel Daulton, Benjamin Letham, Andrew Gordon Wilson, and Eytan Bakshy. BoTorch: A Framework for Efficient Monte-Carlo Bayesian Optimization. In *Advances in Neural Information Processing Systems 33*, 2020.

[2] Joan Bruna, Benjamin Peherstorfer, and Eric Vanden-Eijnden. Neural galerkin schemes with active learning for high-dimensional evolution equations. *Journal of Computational Physics*, 496:112588, 2024.

[3] Owen Davis, Gianluca Geraci, and Mohammad Motamed. Deep learning without global optimization by random fourier neural networks. *SIAM Journal on Scientific Computing*, 47(2):C265–C290, 2025.

[4] Filip de Roos, Alexandra Gessner, and Philipp Hennig. High-dimensional gaussian process inference with derivatives. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 2535–2545. PMLR, 18–24 Jul 2021.

[5] Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density estimation using real NVP. In *International Conference on Learning Representations*, 2017.

[6] Yifan Du and Tamer A. Zaki. Evolutional deep neural network. *Phys. Rev. E*, 104:045303, Oct 2021.

[7] David Eriksson, Kun Dong, Eric Lee, David Bindel, and Andrew G Wilson. Scaling gaussian process regression with derivatives. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

[8] Mathieu Germain, Karol Gregor, Iain Murray, and Hugo Larochelle. Made: Masked autoencoder for distribution estimation. In Francis Bach and David Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 881–889, Lille, France, 07–09 Jul 2015. PMLR.

[9] Zheyuan Hu, Khemraj Shukla, George Em Karniadakis, and Kenji Kawaguchi. Tackling the curse of dimensionality with physics-informed neural networks. *Neural Networks*, 176:106369, 2024.

[10] A. Javeed, D.P. Kouri, D. Ridzal, and G. Von Winckel. Get rol-ing: An introduction to sandia's rapid optimization library. In *7th International Conference on Continuous Optimization*, 2022.

[11] Aku Kammonen, Jonas Kiessling, Petr Plecháč, Mattias Sandberg, and Anders Szepessy. Adaptive random fourier features with metropolis sampling. *Foundations of Data Science*, 2(3):309–332, 2020.

[12] Aku Kammonen, Jonas Kiessling, Petr Plecháč, Mattias Sandberg, Anders Szepessy, and Raul Tempone. Smaller generalization error derived for a deep residual neural network compared with shallow networks. *IMA Journal of Numerical Analysis*, 43(5):2585–2632, 09 2022.

[13] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[14] D. P. Kouri, M. Heinkenschloss, D. Ridzal, and B. G. van Bloemen Waanders. A trust-region algorithm with adaptive stochastic collocation for pde optimization under uncertainty. *SIAM Journal on Scientific Computing*, 35(4):A1847–A1879, 2013.

[15] D. P. Kouri, M. Heinkenschloss, D. Ridzal, and B. G. van Bloemen Waanders. Inexact objective function evaluations in a trust-region algorithm for pde-constrained optimization under uncertainty. *SIAM Journal on Scientific Computing*, 36(6):A3011–A3029, 2014.

[16] Ilya Loshchilov and Frank Hutter. SGDR: Stochastic gradient descent with warm restarts. In *International Conference on Learning Representations*, 2017.

[17] Misha Padidar, Xinran Zhu, Leo Huang, Jacob Gardner, and David Bindel. Scaling gaussian processes with derivative information using variational inference. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 6442–6453. Curran Associates, Inc., 2021.

[18] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[19] Ali Rahimi and Benjamin Recht. Random features for large-scale kernel machines. In J. Platt, D. Koller, Y. Singer, and S. Roweis, editors, *Advances in Neural Information Processing Systems*, volume 20. Curran Associates, Inc., 2007.

[20] Ali Rahimi and Benjamin Recht. Weighted sums of random kitchen sinks: Replacing minimization with randomization in learning. In D. Koller, D. Schuurmans, Y. Bengio, and L. Bottou, editors, *Advances in Neural Information Processing Systems*, volume 21. Curran Associates, Inc., 2008.

[21] M. Raissi, P. Perdikaris, and G.E. Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378:686–707, 2019.

[22] Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning*. The MIT Press, 11 2005.

[23] Moritz Reh and Martin Gärttner. Variational monte carlo approach to partial differential equations with neural networks. *Machine Learning: Science and Technology*, 3(4):04LT02, dec 2022.

[24] Tianchen Zhao, Chuhao Sun, Asaf Cohen, James Stokes, and Shravan Veerapaneni. Quantum-inspired variational algorithms for partial differential equations: application to financial derivative pricing. *Quantitative Finance*, 24(1):1–11, 2024.

DISTRIBUTION

Email—Internal

Name	Org.	Sandia Email Address
David Littlewood	1444	djlittl@sandia.gov
Drew Kouri	1463	dpkouri@sandia.gov
Sophia Lefantzi	1463	slefant@sandia.gov
Technical Library	1911	sanddocs@sandia.gov

Hardcopy—Internal

Number of Copies	Name	Org.	Mailstop
1	L. Martin, LDRD Office	1910	0359

This page intentionally left blank.

Sandia National Laboratories