Gaussian Processes for Uncertainty Quantification over PDE solutions

Alex Ali [*1] **Eshwar Kancherla** [*1] **Kevin Shi** [*1]

Abstract

PDE solvers such as physics-informed neural networks (PINNs), finite difference methods, and finite element methods can offer approximate and accurate solutions to a wide class of partial differential equations (PDEs). However, these solutions are analogous to point-estimates, ignoring the role of aleatoric (inherent randomness) as well as epistemic (model-based) uncertainty which can significantly degrade model performance. As a highly flexible model class, Gaussian Processes (GPs) offer a promising avenue to uncertainty quantification (UQ) by either directly modeling a distribution over PDE solutions or quantifying uncertainty over existing PDE solutions. We benchmark several existing approaches to solving PDEs, combine traditional and deep-learning based solutions for PDEs with GPs, and combine neural networks and Gaussian Processes as a novel method for UQ.

1. Introduction

Imagine a world where weekly weather forecasts were consistently and horribly wrong - where confident predictions of sunny skies were invariably proven wrong by unexpected downpours. The earth's atmosphere epitomizes a chaotic system where minute variations in initial conditions can trigger dramatic meteorological shifts. In estimating this uncertainty, modern forecasts employ ensembles of nearly 50 predictions with varying initial conditions to probabilistically estimate uncertain outcomes (Palmer, 2019). This approach mirrors a fundamental question in computational physics: how do we effectively quantify uncertainty in complex systems governed by nonlinear partial differential equations (PDEs)?

Partial differential equations, especially non-linear ones, are notoriously difficult to solve. In fact, most real-world PDEs lack analytic solutions (Borthwick, 2018) so we turn to the laundry list of approximate PDE solvers instead.

Classical methods like the finite difference and finite element method have been applied to PDEs since at the 1930s. Thomée (2001) gives an account of the development of finite difference and finite element methods for much of the 20th century. Generally, these methods discretize the domain into a mesh, allowing for numerical approximations to the PDE over a set of fixed, collocation points. These methods have desirable convergence properties as well as extensive bodies of work on error bounds (Ainsworth, 2010) and convergence rates (Ye et al., 2020). However, these methods also scale poorly to higher-dimensional PDEs as computational cost scales exponentially with the dimension of the domain, victims of the notorious curse of dimensionality (Bruna et al., 2024).

Raissi et al. (2019) recently introduced physics-informed neural networks (PINNs) which leverage deep learning techniques to estimate solutions to general nonlinear PDEs. Specifically, PINNs parametrize a surrogate solution to any given PDE as a deep neural network (DNN), taking advantage of automatic differentiation to weakly constrain the DNN to respect various physical laws. This knowledge of physical laws also serves to constrain the hypothesis space of possible solutions - one could view a PINN as a DNN with soft inductive biases towards physical solutions. While PINNs lack the strong theoretical guarantees of traditional PDE solvers, they can be more attractive as they can be tailored to avoid the curse of dimensionality (Hu et al., 2024).

Together, PINNs and classical mesh-based methods offer approximate solutions to an extremely wide class of PDEs. However, these solutions are analogous to point estimates and lack uncertainty quantification estimates which is especially important when real-world data with inherent aleatoric uncertainty is used.

Gaussian Processes are a flexible and interpretable model class which directly model a distribution over functions. The main advantage of such methods is uncertainty quantification which is measured by the posterior distribution of the GP at the collocation points. Chen et al. (2021) uses Gaussian Process (GP) methods to solve PDEs, a method which we reproduce with variation on how to effectively enforce boundary constraints.

[*]Equal contribution [1]New York University, USA. Correspondence to: Alex Ali <aa10253@nyu.edu>, Eshwar Kancherla <ek4178@nyu.edu>, Kevin Shi <kys2020@nyu.edu>.

In this paper, we examine different methods for uncertainty quantification over a variety of boundary-value problems with Dirichlet boundary conditions with particular focus on GPs. In terms of novelty, we examine different ways of combining deep neural networks with GPs such as parameterizing the mean function of the GP with a PINN, taking indirect inspiration from deep kernel learning (Wilson et al., 2015b). However, we found that implementing deep kernel learning to directly solve PDEs was a highly non-trivial task, potentially requiring more research.

Future directions for research may include directly modeling solutions for PDEs through deep kernel learning as well as integrating Gaussian Processes into an adaptive learning algorithm where wider confidence regions correspond to greater uncertainty, informing where to sample more collocation points. Our code is available at: https://github.com/alexali04/NeuralPDE.

2. Related Work

We review GP-based and standard methods for quantifying uncertainty over PDE solutions.

2.1. GPs for PDEs

Gaussian Processes are a highly interpretable and flexible model class which can be used to approximate solutions to linear and nonlinear PDEs. Following notational conventions used in Wilson et al. (2015a), we briefly review predictive equations for a Gaussian Process. Later, when reviewing our method, we will go over the more involved process of fitting a GP to a PDE. Given a mean function m and kernel k, a Gaussian Process $\mathcal{GP}(m, k)$ models the function values of a set of input points as having a joint Gaussian distribution. Specifically, for input points $\mathbf{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$, the function values $\mathbf{f}$ are modeled as:

$$\mathbf{f} = f(\mathbf{X}) \sim \mathcal{GP}(m(\mathbf{X}), K_{XX}) \tag{1}$$

Observe that the function values here have a joint Gaussian distribution. By sampling again from this distribution, we can draw a different function. Additionally, the multi-variate Gaussian naturally gives rise to a 95% confidence interval which we can compare against error estimates for traditional methods.

An element of the covariance matrix is equal to the kernel evaluated at two input points, i.e. $[K_{XX}]_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$. Closed form solutions for the covariance matrix and mean vector of the *predictive distribution* conditioned on the training points exist allowing GPs to perform both interpolation and extrapolation. See Rasmussen and Williams (2005) for a thorough discussion of GPs. By sampling from a GP multiple times, we can generate a distribution over solutions, making them a natural choice for solving PDEs as seen in 1. Conditioning the Gaussian Process on hyperparameters (possibly learned through marginal likelihood) and training data, we can construct a predictive distribution over functions. Specifically, where σ^2 is the noise variance estimate, X_* are the predictive input points, and the mean is 0,

$$\mathbf{f}_* \mid X_*, X, \mathbf{y}, \sigma^2 \sim \mathcal{N}(E[\mathbf{f}_*], K_{XX}) \tag{2}$$

$$E[\mathbf{f}_*] = K_{X_*X}[K_{XX} + \sigma^2 I]^{-1}\mathbf{y} \tag{3}$$

$$\text{Cov}(\mathbf{f}_*) = K_{X_*X_*} - K_{X_*X}[K_{XX} + \sigma^2 I]^{-1}K_{XX_*} \tag{4}$$

GPs with user-specified kernels are widespread in estimating PDE solutions. Chen et al. (2021) approximate the solution to nonlinear PDEs as a maximum a posteriori (MAP) estimate of a Gaussian Process conditioned on solving the PDE at a few collocation points. Additionally, Pförtner et al. (2022) propose physics-informed Gaussian Process regression for solving linear PDEs and interpret several classical methods such as finite element methods as specific instances of the GP view. Mora et al. (2024) also bridges the gap between deep learning and GPs by using deep kernel learning to solve PDEs.

Deep Kernel learning learns a good representation of the input points in some high-dimensional embedding space before passing the inputs through a kernel (Wilson et al., 2015b). For inputs $\mathbf{x}_i, \mathbf{x_j}$ and a neural network NN with parameters θ, the deep kernel is $k_{\text{DK}} = k(\text{NN}_\theta(\mathbf{x}_i), \text{NN}_\theta(\mathbf{x}_j))$. This has been used for solving PDEs before. Wang et al. (2022b) uses physics-informed deep kernel learning where the posterior of a GP is used as a surrogate solution alongside generative components.

2.2. PINN for Solving PDEs

The use of DNNs to solve Boundary Value Problem (BVP) PDEs was first introduced in Berg and Nyström (2018), using the capabilities such as auto differentiation (Paszke et al., 2019). The general approach is to take the DNN as the solution to the PDE and minimize the PDE residual and the error of prediction of the DNN at the boundary. More specifically, let u be the DNN, and $\mathcal{L}$ be the PDE, $g(x)$ be the boundary condition

$$\hat{\theta} = \min_\theta \mathcal{L}[u](x_d) + ||u(x_b) - g(x)||^2 \tag{5}$$

Where x_d are randomly sampled points in the domin and x_b are the randomly sampled points in the boundary.

2.3. Standard UQ for PDE Solutions

Apart from Gaussian Processes, there are several other techniques for quantifying uncertainty over approximate PDE

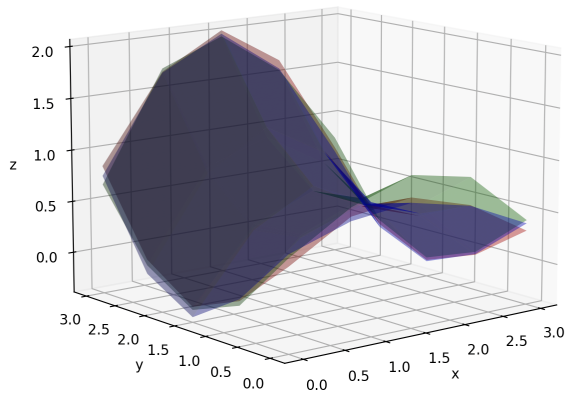

Figure 1. Three solutions are drawn from a Gaussian Process fitted to the noisy output of a PINN trained on the Laplacian PDE with Dirichlet boundary conditions. Specifically, the boundary conditions are: $g(x,3) = 2, g(x,0) = 0, g(0,y), g(3,y) = 0$. The figure illustrates how a GP can give a distribution over PDE solutions

solutions. See Dongre and Hora (2023) for a comprehensive survey of different UQ methods for Neural PDE solutions.

Polynomial Chaos Expansion. One widely adopted technique for UQ for generic complex systems is polynomial chaos expansion (PCE) which propagates uncertainty in the input parameters to the output of the system. The output is decomposed into a series expansion of orthogonal polynomials where the coefficients of the orthogonal polynomials represent the random input variables. This approach offers a probability density over model outputs but suffers from the curse of dimensionality where the number of coefficients in the series needed to effectively represent the output increases drastically with the dimensionality of the input (Yao et al., 2023). This method has been applied specifically to PDEs (Novák et al., 2024).

$$Y = \sum_{i \in \mathbb{N}} c_i \Psi_i(\mathbf{X})$$

Deep Ensembles are the offsprings of the weather forecast systems mentioned in the introduction. Multiple DNNs (an ensemble) are trained with different initializations and hyperparameters. Then, predictions from the individual models are aggregated to give a mean prediction and the variance of the predictions . This approach has been used in conjunction with active learning for UQ in PDEs (Pestourie et al., 2020).

3. Methodology

We explore variations on using a GP to fit the errors of a DNN solution for the PDE, and methods to fit non linear GP to PDEs.

3.1. GP Trained on PINN

In this method we directly use the GP to fit the DNN solution on the same set of sample points used to train the GP. Additionally, we set the heteroskastic noise for each of the training point as the residual error of the PDE observed at that point. The resulting predictive distribution would be used to generate the UQ over the DNN predictions. For simplicity, let us assume that the output function is in $\mathbb{R}$. Let X be the set of sample points used to train the DNN as shown in 5. Then we fit the GP with kernel K using the equation 4 and compute the standard deviation against each input point.

3.2. GP Trained on PINN Errors for Linear PDEs

The previous method ignores the residual and boundary errors that arise when computing the PDE. We can incorporate both these losses by creating a joint distribution of two GPs, the first models the PDE residual and the second to model the boundary error. An obvious disadvantage is that this method is limited to linear PDEs, as any non linearity would not result in a GP. The joint distribution can be modeled as,

$$\begin{bmatrix} \mathcal{L}[u] \\ u \end{bmatrix} \sim \mathcal{N}\left(0, \begin{bmatrix} \mathcal{L}_x\mathcal{L}_y K(X_d, Y_d) & \mathcal{L}_x K(X_d, Y_d) \\ \mathcal{L}_x K(X_d, Y_d)^\top & K(X_b, Y_b) \end{bmatrix}\right)$$

Where, X_d are the sample points in the domain, X_b are the sample points in the boundary, $\mathcal{L}$ is the linear partial differential equation, K is the GP kernel.

3.3. Solving Elliptic PDEs Using GP

We describe a simplified version of the method in (Chen et al., 2021) with extensions for parameter training.

Consider the two dimensional elliptic PDE given by

$$\begin{aligned} \Delta u &= f(\mathbf{x}) && \mathbf{x} \in \Omega \\ u(\mathbf{x}) &= g(\mathbf{x}) && \mathbf{x} \in \partial\Omega \end{aligned}$$

where, $u : \mathbb{R}^2 \to \mathbb{R}$, Δ is the Laplace operator, $\Omega \subset \mathbb{R}^2$ is the domain of the function u and $\partial\Omega \subset \Omega$ is the domain of the boundary, f is the value of the PDE in the interior, and g is the value of u at the boundary. If we assume that $u \sim \mathcal{GP}(0, K)$, where K is an appropriate kernel in which the solution for the PDE is feasible, then $\Delta u \sim \mathcal{GP}(0, K')$ as Δ is a linear operator, where $K'(x,y) = \Delta_x\Delta_y K(x,y)$, $\Delta_x K(x,y)$ is the Laplacian with respect to the first argument of K and $\Delta_y K(x,y)$ is the Laplacian with respect to the second argument.

Our objective is now to form a predictive GP conditioned on the two GPs Δu and u. Unlike GP regression, where we use observations of target values to form the predictive distribution, our objective is to find the set of observations that satisfy both GP conditions. We can achieve this by maximizing the joint marginal likelihood of the two GPs at a set of sample points, known as collocation points. More formally, we define the joint distribution as

$$\begin{bmatrix} \Delta u \\ u \end{bmatrix} \sim \mathcal{N}\left(0, \begin{bmatrix} \Delta_x \Delta_y K(X_d, Y_d) & \Delta_x K(X_d, Y_d) \\ \Delta_x K(X_d, Y_d)^\top & K(X_{db}, Y_{db}) \end{bmatrix}\right)$$

where, $X_d = Y_d = (\mathbf{x}_1, ..., \mathbf{x}_M) \in \Omega$ are sample points in the interior, $X_{db} = Y_{db} = (\mathbf{x}_1, ..., \mathbf{x}_M, \mathbf{x}_{M+1}, ..., \mathbf{x}_{M+M_d})$, with $\mathbf{x}_{M+1}, ..., \mathbf{x}_{M+M_d} \in \partial\Omega$ are sample points in the interior and the boundary. We denote the resulting covariance matrix as Θ. Let $\mathbf{z} = \{z_i\}_{i=1}^{M+M_b}$ be the solution to u at the sample points. We can derive the objective function to maximize the negative log marginal likelihood as

$$\mathbf{z}^* = \arg\max_{\mathbf{z}} -\begin{pmatrix} \mathbf{f} & \mathbf{z} & \mathbf{g} \end{pmatrix} \Theta^{-1} \begin{pmatrix} \mathbf{f} \\ \mathbf{z} \\ \mathbf{g} \end{pmatrix} - \log|\Theta|$$

$$\Leftrightarrow \mathbf{z}^* = \arg\min_{\mathbf{z}} \begin{pmatrix} \mathbf{f} & \mathbf{z} & \mathbf{g} \end{pmatrix} \Theta^{-1} \begin{pmatrix} \mathbf{f} \\ \mathbf{z} \\ \mathbf{g} \end{pmatrix}$$

where $\mathbf{f}$ is the value of f at the interior collocation points, X_d, and $\mathbf{g}$ is the value of g at the boundary collocation points. Since $\Delta u = f$, we can directly substitute $\mathbf{f}$ to eliminate additional parameters. In our experiments, we have used the Newton-Raphson method to solve this optimization problem.

To learn the parameters θ of the kernel we can use any standard gradient based method to minimize the log marginal likelihood. Formally,

$$\hat{\theta} = \arg\min_{\theta} \left(\min_{\mathbf{z}} \begin{pmatrix} \mathbf{f} & \mathbf{z} \end{pmatrix} \Theta^{-1} \begin{pmatrix} \mathbf{f} \\ \mathbf{z} \end{pmatrix} \right) + \log|\Theta| \quad (6)$$

We compute the predictive mean and variance from the joint distribution

$$\begin{bmatrix} \mathbf{u}_* \\ \mathbf{f} \\ \mathbf{z} \end{bmatrix} \sim \mathcal{N}(0, \mathbf{K}) \quad (7)$$

$$\mathbf{K} = \begin{bmatrix} K(X_*, X_*) & \Delta_y K(X_*, X_d) & K(X_*, X_d) \\ \Delta_y K(X_*, X_d)^\top & \Theta & \\ K(X_*, X_d) & & \end{bmatrix} \quad (8)$$

using the standard GP expressions for mean and covariance (Rasmussen and Williams, 2005).

3.4. Deep Kernel Learning

We propose to follow the deep kernel methodology from (Wilson et al., 2015b) with different neural network architectures. However, in our implementation we found that naively solving the optimization problem in (6) is computationally prohibitive, even when the number of sample points is as small as 100. Using approximations from (Wilson et al., 2015a) might be promising.
Ultimately, when fitting a GP with a PINN as the mean function, we did not decide to pursue DKL, as the neural network serves a similar function to the deep kernel.

4. Experiments

4.1. Solution Using GP

We solve the two dimensional steady state heat transfer equation, also knows as the Laplace equation with a non-continuous boundary value. The equation is given as

$$\begin{aligned} &\Delta u = 0, \\ &u(x = 0, y) = u(x = 1, y) = u(x, y = 0) = 1, \\ &u(x, y = 1) = 0 \end{aligned} \quad (9)$$

where $x, y \in [0, 1]$. We use the radial basis function (RBF) kernel and fit 900 interior points, 120 boundary points. The result generated for a uniform grid of 10000 points, which is the same for the true numerical solution. The results of the experiments are shown in figures 2 and the corresponding uncertainty bound computed as four standard deviations at each point (the variance for each point is the corresponding diagonal element of the covariance matrix given in 3). We notice that the uncertainty bound is not comparable to the error matrix and would need further investigation.

We noticed that if the boundary value is changed to 100 instead of 1, the GP is unable to fit the PDE illustrating the limitations of the RBF kernel used. This might also be an argument for using Deep Kernels.

In Figure 3, we see that the error on the predictions does not line up with the covariance matrix, as we would expect some sort of relationship between the two, which currently is an unsolved problem for us that we will need to work out. Further, we see in Figure 4 that our GP does not make good predictions for PDE solutions; however, it is interesting that a general shape can be traced that roughly matches the finite difference solution, despite the fact that the values are completely wrong. This result was somewhat expected given discussion on the results from (Gulian et al., 2020), and we hope to resolve this issue by implementing their solution.

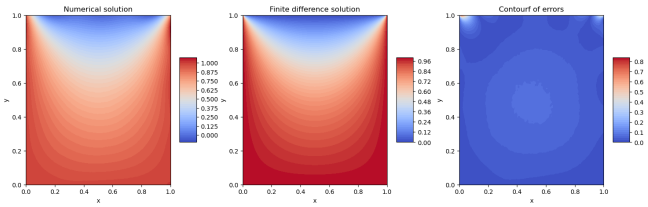

Figure 2. Left: results of fitting Yifan's method to the Laplacian; middle: the finite difference solution (in this case, the true solution); right: contour of errors between the two solutions.

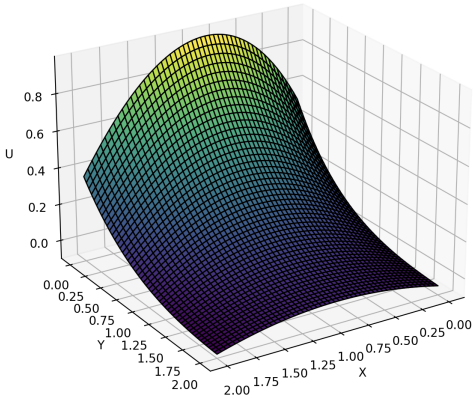

Figure 5. Results of fitting laplace equation using PINN.

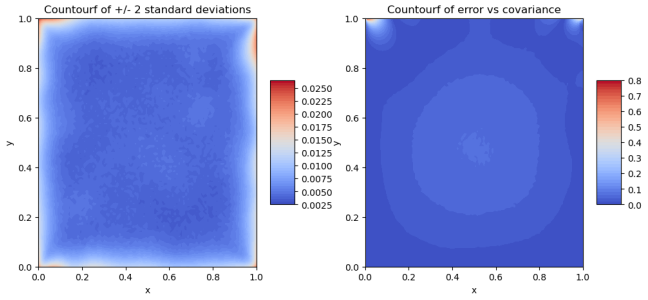

Figure 3. Left: contour of 4 standard deviations at each point; right: the difference between the error contour and standard deviations.

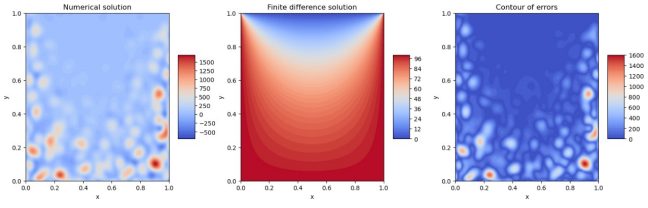

Figure 4. Results of fitting Yifan's method to the Laplacian, along with the finite difference solution (in this case, the true solution) with the boundary condition value being 100 instead of 1.

4.2. Solution Using PINN

The boundary value problem using a multilayer perceptron and following the methods form (Berg and Nyström, 2018) to compute the PDE from 9. The result is shown in 5. We use this as the base PINN in the remaining subsections.

4.3. Mean Function as PINN

We then model the mean function as a PINN in order to obtain uncertainty over the PINN predictions. We can do this using either solutions obtained from finite difference methods, which we will treat as the true solution, or without them, which would be more desirable. However, using the finite difference solution can serve as a good baseline for testing our novel method for uncertainty quantification that does not use finite difference solutions.

4.4. Using Finite Difference

Consider a solution obtained from the finite difference (FD) method where the function $u(x, y) = \text{FD}(x, y)$. Additionally, we can also fit a GP to produce sample functions $u(x, y) \sim GP(\text{PINN}(x, y), K_{\mathcal{D}})$, using the PINN as the prior mean. We investigate modeling the residual error of the finite-difference solution from the PINN, i.e. $u(x, y) - \text{PINN}(x, y) \sim GP(0, K)$.

When making predictions, we can simply add the prior mean onto the predictive mean of this GP to obtain samples from $u(x, y)$, taking into account the prior assumption that the prior mean is equal to the PINN solutions. It is equivalently simpler to produce confidence regions without adding back the prior mean, which gives us a confidence interval on the difference between the finite difference solution and the

PINN solution.

To test this methodology, we experiment with two prior mean functions using GPyTorch: ZeroMean() and ConstantMean(). Gaussian Processes famously exhibit regression to the mean as distance from observed point drops in extrapolation scenarios - for kernels like the squared exponential kernel, we would expect this drop to be rather dramatic due to the exponential decrease in correlation as a function of distance in input points. For our specific example, we believe our data should probably revert to 0 as we move outside of the specified boundary as the boundary condition is nonzero in exactly one of the four segments constraining the domain. Honestly representing our beliefs, we would expect our prior mean to be ZeroMean().

As we can see from figures 8 and 9, using constant prior mean allows the GP to make predictions (by adding the PINN solution and Difference Surface together) that match closer to the finite difference solution; however, we suspect this is because using constant mean makes the GP simply fit to the finite difference points without the assumption that the prior mean is the PINN, which is not super interesting. This is confirmed when looking at the Contour of four standard deviations plot, the 95% percent confidence interval, which is very flat except at the boundaries, where the RBF kernel does not expect the temperature to sharply increase.

On the other hand, using zero mean results in more interesting behavior in figures 10 and 11. We can see that the Difference Surface summed with the PINN solution does not perfectly reconstruct the finite difference solution, and further that the confidence interval is larger at various points in the interior as well. We believe that this indicates that using zero mean properly models the uncertainty over the PINN solutions.

However, there is still higher uncertainty at the boundaries, which isn't particularly desirable since the 'bottom' of the Difference Surface plot in figure 10 has the highest accuracy. This is likely due to undesirable smoothing behavior of the RBF kernel, which naturally motivates future experimentation with various other kernel functions.

Finally, we checked how close the lower and upper bounds of the confidence region approached the predictive mean by calculating $\min(\text{pred\_mean} - \text{lower\_bound})$ and $\min(\text{upper\_bound} - \text{pred\_mean})$ for each method, where they ultimately did not differ much. For zero prior mean and constant prior mean, we obtained $0.1184, 0.1184$ and $0.1185, 0.1185$ respectively.

4.5. GP Trained on PINN

We fit a GP with Radial Basis Function (RBF) kernel to the PINN for solving equation 9. The results are shown in 6. We observe that the confidence interval is non zero unlike the other predictions, and broadly captures the error at different points.

4.6. GP Trained on PINN Errors for Linear PDEs

We fit a GP with a RBF kernel jointly distributed according to the method mentioned in section 3.2 to solve the 9. We chose the RBF kernel as it is infinitely differentiable, and use JAX (Bradbury et al., 2018) to compute the Laplace operator on the kernel. The joint covariance matrix is preconditioned by adding a small offset and the linear solves are computed using the Cholesky decomposition. The resulting graphs are shown in 7. Observing the close match between the GP error prediction and the countour of errors, the GP appears to have fit to the PDE. The confidence interval to some extent supports this observation.

5. Discussion

There have been attempts to model PDE solutions with GPs beforehand, notably from (Wang et al., 2022a), which discusses a similar idea to use Deep Kernel Learning to solve PDEs. Our proposed method using DKL differs from the method discussed in this paper, as we propose an approach more closely resembling PINNs where we fully know the form of the differential equation, while the method proposed in this paper is far more general. We find the physics informed deep kernel learning method discussed in this paper to be a promising way to quantify uncertainty over PDE solutions.

We may also want to tackle more complex PDEs, as linear PDE solvers have well known time complexity, many of which can be solved in linear time complexity. Therefore, it would be good to test our model on non-linear or chaotic PDEs, where our model may be of more practical use.

However, we believe these initial experiments on quantifying uncertainty over PINN predictions of PDE solutions are an important step in creating PINN PDE solvers that can be trusted. We have developed several methods to produce confidence intervals on the error of PINN predictions, which allows the model to assess how incorrect its predictions are, and how confident it is in that assessment. We believe this method novel, as other methods solving PDEs using GPs model uncertainty over PDE solutions directly, while our method places uncertainty over trained PINN point estimates, allowing us to model uncertainty over what is essentially a black box model.

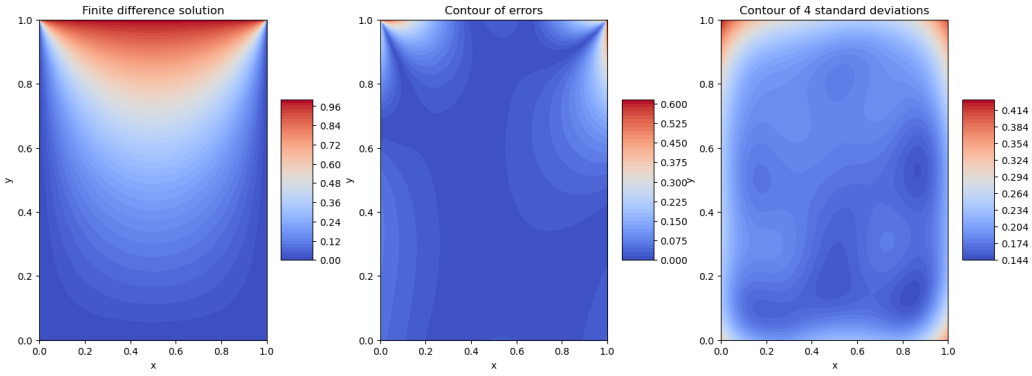

Figure 6. Left: the finite difference solution. Middle: error between the PINN solution and the finite difference solution. Right: the predicted 95% confidence interval of the GP trained on the PINN with the residual error as the heteroscedastic noise.

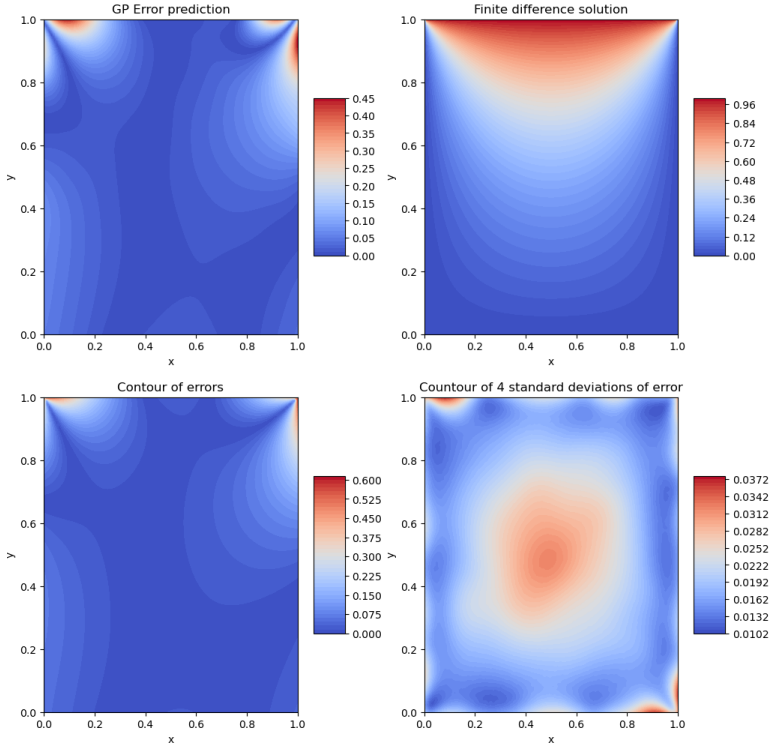

Figure 7. Top left: the GP prediction of the PINN error. Top right: the finite difference solution. Bottom left: error between the PINN solution and the finite difference solution. Bottom right: the predicted 95% confidence interval of the GP trained on the PINN errors.

6. Acknowledgments

We would like to sincerely acknowledge Professor Andrew G. Wilson and Andres Potapczynski from the NYU Center for Data Science for their invaluable advice on how to effectively formulate the paper as well as technical advice on deep kernel learning and Gaussian Processes. We would also like to acknowledge Clark Miyamoto from the NYU Physics Department for similarly valuable advice on partial differential equations.

References

M. Ainsworth. A framework for obtaining guaranteed error bounds for finite element approximations. *Journal of Computational and Applied Mathematics*, 234(9):2618–2632, 2010. ISSN 0377-0427. doi: https://doi.org/10.1016/j.cam.2010.01.037. URL https://www.sciencedirect.com/science/article/pii/S0377042710000415. Third International Workshop on Analysis and Numerical Approximation of Singular Problems [IWANASP08].

J. Berg and K. Nyström. A unified deep artificial neural network approach to partial differential equations in complex geometries. *Neurocomputing*, 317:28–41, Nov. 2018. ISSN 0925-2312. doi: 10.1016/j.neucom.2018.06.056. URL http://dx.doi.org/10.1016/j.neucom.2018.06.056.

D. Borthwick. *Introduction to Partial Differential Equations first edition*. Universitext. Springer, 2018. doi: doi: 10.1007/978-3-319-48936-0.

J. Bradbury, R. Frostig, P. Hawkins, M. J. Johnson, C. Leary, D. Maclaurin, G. Necula, A. Paszke, J. VanderPlas, S. Wanderman-Milne, and Q. Zhang. JAX: composable transformations of Python+NumPy programs, 2018. URL http://github.com/jax-ml/jax.

J. Bruna, B. Peherstorfer, and E. Vanden-Eijnden. Neural galerkin schemes with active learning for high-dimensional evolution equations. *Journal of Computational Physics*, 496:112588, Jan. 2024. ISSN 0021-9991. doi: 10.1016/j.jcp.2023.112588. URL http://dx.doi.org/10.1016/j.jcp.2023.112588.

Y. Chen, B. Hosseini, H. Owhadi, and A. M. Stuart. Solving and learning nonlinear pdes with gaussian processes, 2021. URL https://arxiv.org/abs/2103.12959.

V. Dongre and G. S. Hora. Evaluating uncertainty quantification approaches for neural pdes in scientific applications, 2023. URL https://arxiv.org/abs/2311.04457.

M. Gulian, A. Frankel, and L. Swiler. Gaussian process regression constrained by boundary value problems, 2020. URL https://arxiv.org/abs/2012.11857.

Z. Hu, K. Shukla, G. E. Karniadakis, and K. Kawaguchi. Tackling the curse of dimensionality with physics-informed neural networks. *Neural Networks*, 176:106369, 2024. ISSN 0893-6080. doi: https://doi.org/10.1016/j.neunet.2024.106369. URL https://www.sciencedirect.com/science/article/pii/S0893608024002934.

C. Mora, A. Yousefpour, S. Hosseinmardi, and R. Bostanabad. Integrating kernel methods and deep neural networks for solving PDEs. In *ICLR 2024 Workshop on AI4DifferentialEquations In Science*, 2024. URL https://openreview.net/forum?id=5bRRVmYhVS.

L. Novák, H. Sharma, and M. D. Shields. Physics-informed polynomial chaos expansions. *Journal of Computational Physics*, 506:112926, 2024. ISSN 0021-9991. doi: https://doi.org/10.1016/j.jcp.2024.112926. URL https://www.sciencedirect.com/science/article/pii/S002199912400175X.

T. N. Palmer. Stochastic weather and climate models. *Nat. Rev. Phys.*, 1(7):463–471, May 2019.

A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. Pytorch: An imperative style, high-performance deep learning library, 2019. URL https://arxiv.org/abs/1912.01703.

R. Pestourie, Y. Mroueh, T. V. Nguyen, P. Das, and S. G. Johnson. Active learning of deep surrogates for pdes: application to metasurface design. *npj Computational Materials*, 6(1):164, Oct 2020. ISSN 2057-3960. doi: 10.1038/s41524-020-00431-2. URL https://doi.org/10.1038/s41524-020-00431-2.

M. Pförtner, I. Steinwart, P. Hennig, and J. Wenger. Physics-informed gaussian process regression generalizes linear pde solvers, 2022. URL https://arxiv.org/abs/2212.12474.

M. Raissi, P. Perdikaris, and G. 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. ISSN 0021-9991. doi: https://doi.org/10.1016/j.jcp.2018.10.045. URL https://www.sciencedirect.com/science/article/pii/S0021999118307125.

C. E. Rasmussen and C. K. I. Williams. *Gaussian Processes for Machine Learning*. The MIT Press, 11 2005. ISBN 9780262256834. doi: 10.7551/mitpress/3206.001.0001. URL https://doi.org/10.7551/mitpress/3206.001.0001.

V. Thomée. From finite differences to finite elements: A short history of numerical analysis of partial differential equations. *Journal of Computational and Applied Mathematics*, 128(1):1–54, 2001. ISSN 0377-0427. doi: https://doi.org/10.1016/S0377-0427(00)00507-0. URL https://www.sciencedirect.com/science/article/pii/S0377042700005070. Numerical Analysis 2000. Vol. VII: Partial Differential Equations.

Z. Wang, W. Xing, R. Kirby, and S. Zhe. Physics informed deep kernel learning, 2022a. URL https://arxiv.org/abs/2006.04976.

Z. Wang, W. Xing, R. Kirby, and S. Zhe. Physics informed deep kernel learning, 2022b. URL https://arxiv.org/abs/2006.04976.

A. G. Wilson, C. Dann, and H. Nickisch. Thoughts on massively scalable gaussian processes, 2015a. URL https://arxiv.org/abs/1511.01870.

A. G. Wilson, Z. Hu, R. Salakhutdinov, and E. P. Xing. Deep kernel learning, 2015b. URL https://arxiv.org/abs/1511.02222.

W. Yao, X. Zheng, J. Zhang, N. Wang, and G. Tang. Deep adaptive arbitrary polynomial chaos expansion: A mini-data-driven semi-supervised method for uncertainty quantification. *Reliability Engineering and System Safety*, 229:108813, 2023. ISSN 0951-8320. doi: https://doi.org/10.1016/j.ress.2022.108813. URL https://www.sciencedirect.com/science/article/pii/S095183202200432X.

C. Ye, H. Dong, and J. Cui. Convergence rate of multiscale finite element method for various boundary problems. *Journal of Computational and Applied Mathematics*, 374:112754, 2020. ISSN 0377-0427. doi: https://doi.org/10.1016/j.cam.2020.112754. URL https://www.sciencedirect.com/science/article/pii/S0377042720300455.

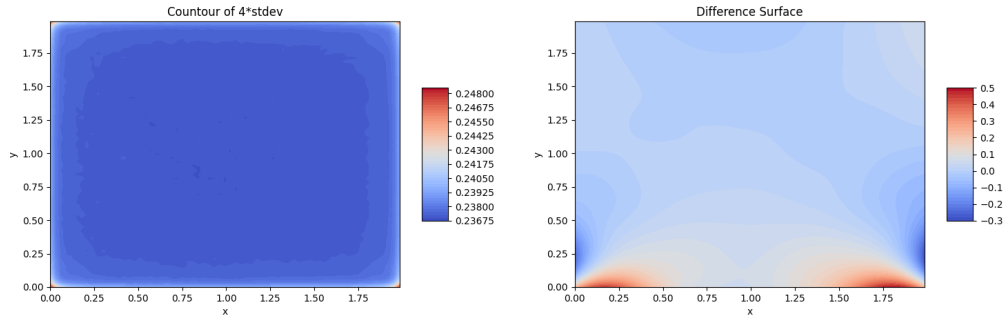

Figure 8. Results using constant mean.

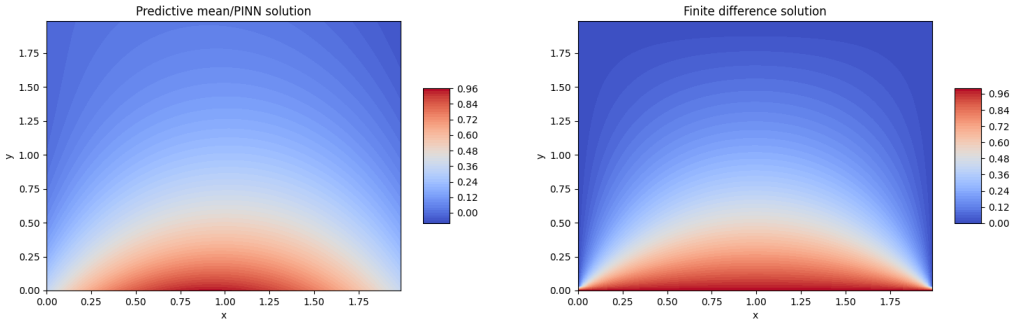

Figure 9. Results using constant mean.

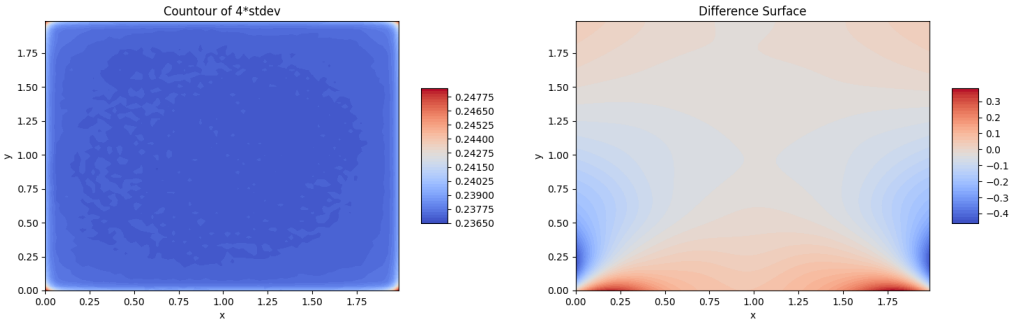

Figure 10. Results using zero mean.

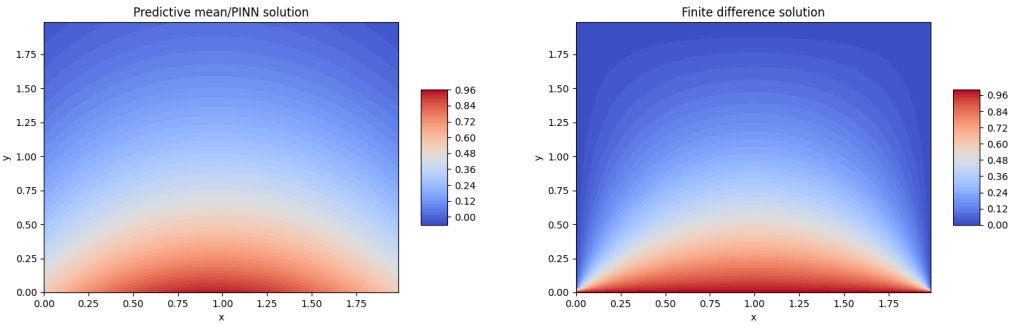

Figure 11. Results using zero mean.