Learning Reduced-Order Mappings between Functions

An Investigation of Suitable Inputs and Outputs

Bo Bakker[1]

Supervisor(s): dr. David Tax[1]**, Mahdi Naderibeni**[1]

[1]EEMCS, Delft University of Technology, The Netherlands

A Thesis Submitted to EEMCS Faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering
January 28, 2024

Name of the student: Bo Bakker
Final project course: CSE3000 Research Project
Thesis committee: dr. David Tax, Mahdi Naderibeni, Nergis Tömen

Abstract

Data-driven approaches are a promising new addition to the list of available strategies for solving Partial Differential Equations (PDEs). One such approach, the Principal Component Analysis-based Neural Network PDE solver, can be used to learn a mapping between two function spaces, corresponding to a PDE. However, the practical limitations of this approach are unclear. This paper seeks to investigate for which types of inputs and outputs this type of solver gives useful results. Using a dataset with inputs sampled from Gaussian Random Fields with different parameters, and outputs for Poisson's equation and the Heat equation, obtained by using a Finite Element solver, neural networks are trained, and their performance is evaluated. The method performs adequately for the chosen inputs, and patterns are found in the resulting error, which differ for each set of input parameters. Thus, for these equations, it seems that this method performs differently for different input distributions, but further research is necessary to investigate if these patterns will hold for other equations.

1 Introduction

Many tasks in science and engineering require one to solve Partial Differential Equations. There are countless methods for solving PDEs, often designed for particular types of equations, and considerable computational resources are expended to obtain solutions. For large and complicated problems, this usually entails the use of large (and expensive) computer clusters, which many people do not have access to.

Often, it is necessary to evaluate a PDE for many similar but slightly different inputs, and it would be convenient if we could exploit previously solved instances of problems, solved with similar inputs, to speed up the process of solving a new problem. In a series of previous publications [1], [2], [3], it was shown that one can do this.

This novel type of PDE solver, the Principal Component Analysis-based Neural Network (PCA-NN) solver, works as follows: One starts with a discretization, usually an image, representing the function one wishes to use as input. One applies PCA to this input, and provides some fixed number of PCA components to the input layer of a Neural Network, which attempts to predict some fixed number of PCA components of the output. The output can then be reconstructed from the predicted PCA components.

The network itself is trained on a set of input-output pairs, which are obtained by using a conventional PDE solver. PCA is applied to the training data, and a fixed number of PCA components is chosen for the input and output layers of the network. The network is then trained on the PCA components of the inputs and outputs in the training set.

However, while this approach is obviously attractive, since it is both fast and data-driven[1], it is not clear what the limitations of this approach are. In previous research [1], [2], [3], the data has been designed to evaluate the performance of this approach on only a few types of inputs, and has not included inputs designed to test the limits of these methods.

The goal of the paper is to take a first step towards answering the question *"What are the limitations on the types of inputs and outputs PCA-NN solvers can provide adequate solutions for?"*

In order to do this, we generate different types input data, and use this input data to perform numerical experiments. Using the results of these experiments, we attempt to find patterns in the performance of PCA-NN solvers.

[1]There is no need to specify explicitly which PDE one wishes to solve, since this information is implicit in the training data.

Previous Work

Previous research has been done on PCA-NN solvers. One of the first papers to be published in this area was [1], which not only demonstrated the practical viability of this approach, but also provided a substantial amount of theoretical elucidation of how and why this method works. The approach taken in [1] is fully data-driven, in that the solver has no explicit knowledge of the underlying PDE. The only thing provided is a set of inputs and outputs of the mapping to be learned.

Of course, the PCA-based approach is not the only one available. Other methods, like the method outlined in [4], can be used for similar purposes. This method, the Fourier Neural Operator solver, has many of the same advantages as the approach taken in [1]. In [5], a general description of this type of method is provided, and a number of different Neural Operator solvers are compared.

Another paper, which compares the performance of PCA-NN solvers to other similar methods, is [2]. Here, PCA-NN solvers are compared to a number of other solvers, including Fourier Neural Operator solvers. Each method is evaluated on four separate equations, and the training and testing error of each method is compared.

There has been previous work in a similar direction to the one taken here. An example of this is [3], which provides a large dataset with data for a number of different equations. This allows one to compare the performance of a given method when varying a parameter of a PDE[2]. However, the distributions of the inputs are all still quite similar, and so this dataset, on its own, cannot be used to perform the numerical experiments we are trying to perform.

2 Our Contribution

In previous works, the datasets are designed to evaluate the performance of a given method [1] or compare different methods [3], [2]. The datasets used in these papers have inputs drawn from Gaussian Random Fields (GRFs)[3].

While these datasets enable one to evaluate if a given method can work in principle, none of them are suitable to

[2]For example, the dataset contains data for the Navier-Stokes equation, used to simulate the behaviour of fluids, with different values of viscosity.

[3]Gaussian Random Fields are families of random variables representing points in space, subject to a covariance function, which determines the joint variability of any two points of the GRF. See [6, Chapter 7] for a more detailed introduction to random fields.

evaluate the types of input for which a given method provides usable results. As such, we have created datasets with inputs drawn from GRFs with a number of different covariance functions, and we compare the performance of our PCA-NN solver on each of these.

Using these results, we may attempt to draw conclusions about the suitability of PCA-NN solvers for use on a given type of problem. If the solution of a given equation with a given covariance function (or family of covariance functions) is close to the reference solution, we may conclude that this equation can be solved by a PCA-NN solver for similar input distributions. If we do this with a large enough set of covariance functions, we can try to find patterns, and start to predict which types of input this method is suitable for in practice.

In order to enable the comparison of the results we obtain with those obtained in earlier work, we attempt to replicate (as closely as possible) the solvers used in [1] and [2]. We use similarly sized inputs, and use neural networks of similar type and size.

3 Methodology

The approach we take is essentially the same as the approach taken in [1], the primary difference being the chosen equations, as well as the training and testing data used.

First, it is necessary to choose a set of PDEs (and boundary conditions) to simulate using conventional methods in order to generate the training and testing data.

Equation 1 corresponds to Poisson's Equation[4]. Here, we attempt to have the network learn the mapping from f to u.

$$\begin{aligned} \Delta u(\mathbf{x}) &= f(\mathbf{x}) \qquad && \mathbf{x} \in \Omega \\ u(x_0, x_1) &= 1 - x_1 && \mathbf{x} \in \partial\Omega \end{aligned} \tag{1}$$

Equation 2 corresponds to the Heat Equation. Here, we attempt to have the network learn the mapping from u_0 to $u|_{t=1}$.

$$\begin{aligned} \frac{\partial u(\mathbf{x}, t)}{\partial t} - \Delta u(\mathbf{x}, t) &= 0 \qquad && \mathbf{x} \in \Omega \\ u(x_0, x_1, t) &= 1 - x_1 && \mathbf{x} \in \partial\Omega \\ u(\mathbf{x}, 0) &= u_0(\mathbf{x}) && \mathbf{x} \in \Omega \end{aligned} \tag{2}$$

The domain Ω (in this case the unit square) and the boundary conditions are the same for both equations.

We solve these equations with GRFs as inputs. We not only solve these equations for GRFs with different covariance functions, but we also vary two parameters, variance and correlation length, of these covariance functions.

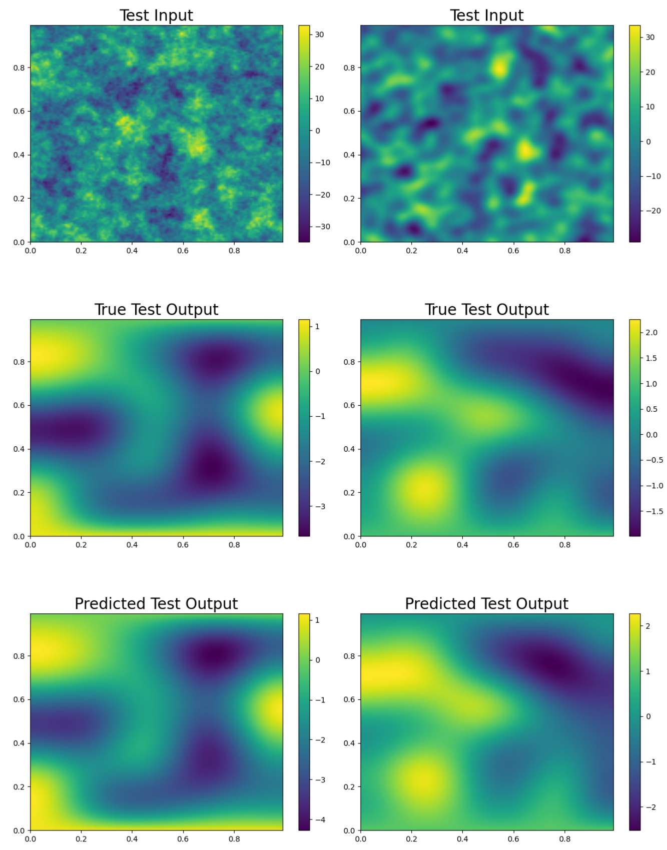
After generating the training and testing data, we apply PCA to the input-output pairs corresponding to each set of parameters, and use the result of this to train one network for each. We evaluate the average error of these networks, both for the data used for training, and for unseen testing data.

4 Experimental Setup and Results

We generate a set of input-output pairs, where the inputs are 101x101 pixel images sampled from a GRF with a given covariance function, generated using the parafields library [7]. The covariance functions used are Gaussian, Exponential, and Separable Exponential. Examples of inputs drawn from GRFs with these covariance functions are shown in figure 2.

In order to generate the outputs, we use the FEniCS finite element solver [8], [9], [10]. For equation 1, the input corresponds to f, and the mapping we attempt to learn is $f \to u$. For equation 2, the input corresponds to u_0, and the mapping we attempt to learn is $u_0 \to u|_{t=1}$. As such, we solve each of these equations with their given input using FEniCS, and store 2000 input-output pairs per covariance function,

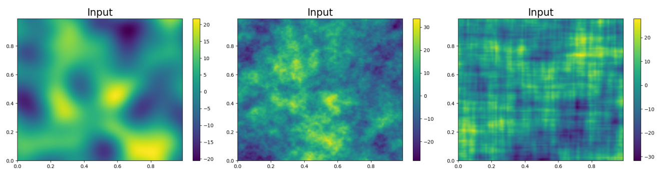
[4]Here, as in equation 2, Δ represents the Laplace operator, with $\Delta f = \nabla \cdot \nabla f$.



(a) Left column: Exponential Covariance

(b) Right column: Gaussian Covariance

Figure 1: Inputs and Outputs for the Heat Equation (Eq. 2) with a correlation length of 0.05 and a variance of 100



(a) Gaussian (b) Exponential (c) Sep. Exp.

Figure 2: Inputs generated with different covariance functions with a correlation length of 0.15 and a variance of 100

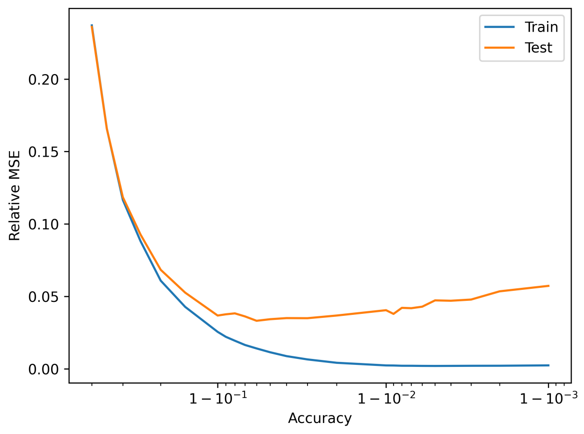


Figure 3: Training and testing error when varying the accuracy percentage for Poisson's equation, with Gaussian covariance, a correlation length of 0.1, and a variance of 100. In this particular case, the first output component alone already gives an accuracy of around 60%.

and per equation. Thus, with two equations, three covariance functions, and three values each for variance and correlation length, we have 54 sets of 2000 input-output pairs, which we use to train 54 different neural networks.

We divide each set of input-output pairs into a training set and a test set of equal size, and we apply PCA to the inputs (and outputs) of the training set, and select numbers of PCA components such that 99% of the variance of the input and output is accounted for[5]. This accuracy value was determined by investigating the error when it is varied, and 99% was determined to be sufficient in all cases that were tried. The results for one such case are shown in figure 3.

We use a network similar to the one used in [1], with input and output layers sized according to the number of PCA components of the input and output, five hidden layers of 1000, 1000, 2000, 1000 and 1000 neurons, and the SELU activation function. The network is trained, using the Adam optimizer, for 500 epochs, with a batch size of 100, an initial learning rate of 0.001, a step size of 100 epochs, and a gamma of 0.5.

After training, we evaluate the performance of the PCA and neural network on our unseen testing data. The results for two different networks are displayed in figure 1, both for the heat equation, but with different covariance functions.

The training and testing error of these experiments for equation 1 and 2 are displayed in table 1 and 2, respectively. The top value is the training error, whereas the bottom value is the testing error. The variance parameter determines the scale of the covariance function, with a larger variance resulting in a larger range of output values. The correlation length parameter determines the scale of the pattern, with a smaller correlation length giving a more fine-grained input, and a larger correlation length resulting in larger blobs.

[5]In order to allow us to compare the performance of PCA-NN on many different distributions of inputs and outputs, it is necessary to hold the amount of detail captured by PCA fixed.

		Correlation Length		
Cov.	Var.	0.15	0.10	0.05
Gauss.	100	3.30×10^{-3}	2.46×10^{-3}	3.13×10^{-3}
		4.19×10^{-2}	4.10×10^{-2}	4.85×10^{-2}
	10	2.10×10^{-3}	1.44×10^{-3}	1.09×10^{-3}
		1.35×10^{-2}	1.31×10^{-2}	1.51×10^{-2}
	1	1.79×10^{-3}	1.31×10^{-3}	7.81×10^{-4}
		4.39×10^{-3}	4.31×10^{-3}	5.01×10^{-3}
Exp.	100	6.71×10^{-3}	5.30×10^{-3}	4.23×10^{-3}
		3.47×10^{-1}	1.71×10^{-1}	9.86×10^{-2}
	10	2.21×10^{-3}	1.90×10^{-3}	1.30×10^{-3}
		9.68×10^{-2}	7.63×10^{-2}	2.93×10^{-2}
	1	1.10×10^{-3}	9.70×10^{-4}	8.05×10^{-4}
		3.25×10^{-2}	1.93×10^{-2}	7.44×10^{-3}
Sep.	100	5.70×10^{-3}	4.92×10^{-3}	3.48×10^{-3}
		2.55×10^{-1}	1.84×10^{-1}	5.31×10^{-2}
	10	2.10×10^{-3}	1.67×10^{-3}	1.21×10^{-3}
		1.05×10^{-1}	6.88×10^{-2}	2.08×10^{-2}
	1	1.06×10^{-3}	9.23×10^{-4}	7.56×10^{-4}
		3.12×10^{-2}	2.36×10^{-2}	5.34×10^{-3}

Table 1: Relative MSE for Poisson's Equation (Eq. 1)
Top Value: Training Error, Bottom Value: Testing Error

		Correlation Length		
Cov.	Var.	0.15	0.10	0.05
Gauss.	100	1.22×10^{-2}	1.15×10^{-2}	1.19×10^{-2}
		1.02×10^{-1}	1.29×10^{-1}	2.59×10^{-1}
	10	1.15×10^{-2}	1.25×10^{-2}	1.03×10^{-2}
		1.01×10^{-1}	1.24×10^{-1}	2.23×10^{-1}
	1	1.12×10^{-2}	1.05×10^{-2}	7.85×10^{-3}
		8.69×10^{-2}	9.43×10^{-2}	1.26×10^{-1}
Exp.	100	1.09×10^{-2}	1.09×10^{-2}	1.16×10^{-2}
		3.45×10^{-1}	2.32×10^{-1}	1.75×10^{-1}
	10	1.05×10^{-2}	1.07×10^{-2}	1.08×10^{-2}
		3.01×10^{-1}	3.12×10^{-1}	1.60×10^{-1}
	1	9.80×10^{-3}	9.33×10^{-3}	8.89×10^{-3}
		1.64×10^{-1}	1.29×10^{-1}	1.01×10^{-1}
Sep.	100	1.12×10^{-2}	1.16×10^{-2}	1.09×10^{-2}
		1.93×10^{-1}	1.80×10^{-1}	2.14×10^{-1}
	10	1.09×10^{-2}	1.05×10^{-2}	1.13×10^{-2}
		2.87×10^{-1}	1.76×10^{-1}	1.69×10^{-1}
	1	9.97×10^{-3}	8.92×10^{-3}	7.69×10^{-3}
		1.54×10^{-1}	1.32×10^{-1}	9.98×10^{-2}

Table 2: Relative MSE for the Heat Equation (Eq. 2)
Top Value: Training Error, Bottom Value: Testing Error

		Correlation Length		
Covariance Function	Variance	0.15	0.10	0.05
Gaussian	100	169	335	829
		58	106	267
	10	169	335	828
		42	73	164
	1	169	336	829
		24	37	64
Exponential	100	971	972	976
		164	206	290
	10	971	972	976
		87	109	151
	1	971	972	976
		31	38	49
Separable Exponential	100	958	965	974
		166	219	315
	10	958	965	974
		90	116	165
	1	958	965	974
		33	41	52

Table 3: PCA components for Poisson's Equation (Eq. 1)
Top Value: Input, Bottom Value: Output

		Correlation Length		
Covariance Function	Variance	0.15	0.10	0.05
Gaussian	100	169	336	828
		30	34	36
	10	169	336	828
		30	33	36
	1	169	336	829
		29	32	34
Exponential	100	971	972	976
		29	31	34
	10	971	972	976
		29	31	34
	1	971	972	976
		28	30	32
Separable Exponential	100	958	965	974
		30	32	35
	10	958	965	974
		30	32	34
	1	958	965	974
		29	31	33

Table 4: PCA components for the Heat Equation (Eq. 2)
Top Value: Input, Bottom Value: Output

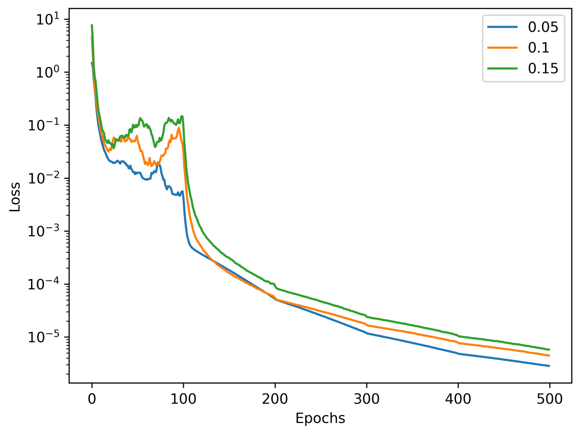


Figure 4: Training loss for Poisson's Equation, with Gaussian covariance, and a variance of 1

The number of PCA components for the input and output (i.e. the size of the input and output layers) for equation 1 and 2 are displayed in table 3 and 4, respectively. The top value is the number of PCA components for the input, whereas the bottom value is the number of PCA components for the output, which represent an accuracy value of 0.99 in both cases.

The training loss for three networks is shown in figure 4, where we can clearly see the effect of stepping down the learning rate every 100 epochs.

5 Responsible Research

For the type of research performed here, there are usually a number of ethical issues to consider, such as reproducibility, data access, privacy, and wider societal impact.

In order to enable other researchers to reproduce our results, we have ensured that the methodology and experimental setup are adequately described. Furthermore, the most important implementation details used to acquire these particular results, such as hyperparameters, are provided as well.

Since all of the data used in this research is generated using the parafields library [7], there is no need to upload the actual dataset, since an equivalent dataset can simply be generated by anyone attempting to replicate our results. For the same reason, there are no issues with data privacy, since we are not using any personal data at all.

As for the issue of wider societal impact, PDE solvers can be used for many purposes, not all of which are benign. For example, since most physical phenomena can be modeled with PDEs, easy access to such solvers can enable one to more easily develop weapons, since one can predict the behavior of a design before actually constructing it. However, since conventional PDE solvers are already widely available to anyone with the skills to use them, the impact of this novel type of method (let alone this particular research), will most likely be minimal.

6 Discussion

The results for each equation are different, and somewhat unexpected. The results for Poisson's equation (in table 1) seem to indicate roughly equal performance for smaller correlation lengths for Gaussian covariance, and better performance for smaller correlation lengths for Exponential and Separable Exponential covariance. This is precisely the opposite of what one would naively expect, since smaller correlation lengths represent "rougher" inputs, and in table 3, we see that more PCA components are needed to capture the same amount of detail. On the other hand, the results for variance are more or less as expected, with smaller variance resulting in better performance, presumably due to better numerical precision for smaller inputs.

As for the performance of Poisson's equation for a given type of covariance function, it appears that Gaussian covariance outperforms both Exponential and Separable Exponential covariance in absolute terms.

The results for the Heat equation (in table 2) are also unexpected, with testing error increasing for smaller correlation lengths for Gaussian covariance, and decreasing for smaller correlation lengths for Exponential and Separable Exponential covariance. Again, the results for variance are as expected, with error decreasing with decreasing variance.

However, the Heat equation seems to be remarkably tolerant to different input distributions, since the performance for each covariance functions (when holding variance and correlation length constant) is remarkably similar across the board, and the differences for different covariance functions are smaller than those seen for Poisson's equation.

The main difference between Poisson's equation (an elliptic PDE) and the Heat equation (a parabolic PDE) when comparing covariance functions appears to be how it responds to GRFs with Gaussian covariance. Also, in table 4, we see that for the output the number of PCA components needed to reach an accuracy of 0.99 is relatively constant when varying correlation length and variance, whereas for the outputs in table 3 the number of PCA components varies significantly.

From this data alone, it is difficult to draw any firm conclusions about the types of input PCA-NN methods can handle. It seems that solvers for different equations do perform differently on different inputs, and, more importantly, that the error may increase significantly with slight variations in certain parameters (e.g. a twofold increase in correlation length can result in a nearly fourfold increase in error), but the differences are not particularly pronounced, at least for the chosen parameters.

However, what we can conclude with certainty is that PCA-NN seems to perform well on all of these inputs, which serves to give us confidence that for these equations and these types of input, PCA-NN can learn mappings corresponding to these equations, and provides useful solutions.

7 Conclusions and Future Work

We have attempted to discover the types of inputs and outputs that PCA-NN solvers are suited for. To do this, we have generated datasets with inputs drawn from a number of different Gaussian Random Fields (with different parameters). As a result, we have discovered a number of patterns in the way PCA-NN solvers respond to different types of inputs, but since the performance of the method is adequate for every type of input (with the error in tables 1 and 2 never exceeding 0.35, and usually being much lower), it is not obvious how to extrapolate from these results.

As such, we have a number of recommendations for further research in this direction. Firstly, it would be interesting to see if some of these patterns hold for other covariance functions. Secondly, since there was a noticeable difference between the performance of Poisson's equation and the Heat equation, it would be interesting to see if one could discover similar patters for Hyperbolic equations, like the Wave equation. Thirdly, since we have only used only one Elliptic and one Parabolic equation, it is not clear if the patterns discovered here hold for Elliptic and Parabolic equations in general, or only these specific equations. Fourthly, since the number of PCA components was varied and the amount of detail captured was held constant, it may be useful to investigate the same (or similar) inputs as used here, but with the number of PCA components held constant and the amount of detail captured varied.

Of course, there are also other possible directions for empirical investigations into PCA-NN (and similar methods, like FNO), like the performance for out-of-distribution inputs, the performance of PCA-NN with more physically realistic input distributions, and the performance of these methods when varying the size of the neural network.

References

[1] K. Bhattacharya, B. Hosseini, N. B. Kovachki, and A. M. Stuart, "Model reduction and neural networks for parametric pdes," 2021.

[2] M. V. de Hoop, D. Z. Huang, E. Qian, and A. M. Stuart, "The cost-accuracy trade-off in operator learning with neural networks," 2022.

[3] M. Takamoto, T. Praditia, R. Leiteritz, D. MacKinlay, F. Alesiani, D. Pflüger, and M. Niepert, "Pdebench: An extensive benchmark for scientific machine learning," 2023.

[4] Z. Li, N. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart, and A. Anandkumar, "Fourier neural operator for parametric partial differential equations," 2021.

[5] N. Kovachki, Z. Li, B. Liu, K. Azizzadenesheli, K. Bhattacharya, A. Stuart, and A. Anandkumar, "Neural operator: Learning maps between function spaces," 2023.

[6] G. J. Lord, C. E. Powell, and T. Shardlow, *An Introduction to Computational Stochastic PDEs*. Cambridge Texts in Applied Mathematics, Cambridge University Press, 2014.

[7] D. Kempf, O. Klein, R. Kutri, R. Scheichl, and P. Bastian, "parafields: A generator for distributed, stationary gaussian processes," *Journal of Open Source Software*, vol. 8, no. 92, p. 5735, 2023.

[8] M. W. Scroggs, J. S. Dokken, C. N. Richardson, and G. N. Wells, “Construction of arbitrary order finite element degree-of-freedom maps on polygonal and polyhedral cell meshes,” *ACM Transactions on Mathematical Software*, vol. 48, no. 2, pp. 18:1–18:23, 2022.

[9] M. W. Scroggs, I. A. Baratta, C. N. Richardson, and G. N. Wells, “Basix: a runtime finite element basis evaluation library,” *Journal of Open Source Software*, vol. 7, no. 73, p. 3982, 2022.

[10] M. S. Alnaes, A. Logg, K. B. Oelgaard, M. E. Rognes, and G. N. Wells, “Unified form language: A domain-specific language for weak formulations of partial differential equations,” 2013.