KERNEL METHODS FOR SOLVING PARTIAL DIFFERENTIAL EQUATIONS

CADE BALLEW AND ALEX HSU

Department of Applied Mathematics, University of Washington, Seattle, WA
ballew@uw.edu, owlx@uw.edu

Abstract. We discuss and implement kernel methods for solving PDEs based on the framework of [3] with the modification of working in the feature space, learning the coefficients in the kernel machine rather than point evaluations. This makes it conceptually clear that we are performing spectral collocation in the RKHS associated to our choice of kernel. We find that such methods are widely applicable, including to problems in which classical numerical methods encounter difficulties, such as those with irregular geometries. The use of modern autodifferentiation tools and jit compilers ameliorates difficulties involved in computing derivatives, both in the kernel matrix, and in solving the variational problems involved.

1. Introduction and Overview

Partial differential equations (PDEs) play a fundamental role in describing numerous phenomena in science and engineering, ranging from fluid dynamics and heat transfer to electromagnetism and quantum mechanics. Solving PDEs accurately and efficiently is essential for gaining insights into complex systems and making informed decisions. Traditional numerical methods, such as finite difference, finite element, and spectral methods, have been widely employed for PDE solving; however, these methods often face challenges in dealing with high-dimensional problems, irregular geometries, and complex boundary conditions.

Kernel methods, which have gained significant attention in machine learning and data analysis, offer a promising alternative for solving PDEs numerically. Inspired by the principles of kernel functions and reproducing kernel Hilbert spaces (RKHS), kernel methods provide a powerful framework for function approximation and interpolation. They are particularly advantageous in handling large datasets, non-uniformly sampled data, and irregular domains. In recent years, researchers have successfully applied kernel methods to various PDE problems, demonstrating their effectiveness in achieving accurate and efficient solutions.

In this paper, we adapt and implement a framework for numerically solving PDEs with kernel methods that was originally outlined for linear PDEs in [6] and extended to nonlinear PDEs in [3]. We explore the behavior of these methods on various equations and domains as well as a technique for solving and learning a PDE simultaneously.

In Section 2, we outline the theoretical background necessary for building numerical kernel methods for solving PDEs. In Section 3, we discuss the implementation of the methods outlined in Section 2. In Section 4, we present numerical results of this implementation, demonstrating that it can be used to numerically solve both linear and nonlinear PDEs and

Date: June 9, 2023.

solve and learn PDEs simultaneously. In Section 5, we dsicuss the efficacy of these methods and potential future work. Code used to generate the plots in this paper can be found at [1].

2. Theoretical Background

Let $d \geq 1$ and $\Omega \in \mathbb{R}^d$ be a bounded open domain with a Lipschitz boundary $\partial\Omega$. We are interested in numerical solutions to PDEs of the form

$$\begin{aligned} -\Delta u(x) + \tau(u(x)) &= f(x), \quad x \in \Omega, \\ u(x) &= g(x), \quad x \in \partial\Omega, \end{aligned} \tag{1}$$

where Δ denotes the Laplace operator and $f : \Omega \to \mathbb{R}$, $g : \partial\Omega \to \mathbb{R}$, and $\tau : \mathbb{R} \to \mathbb{R}$ are given continuous functions selected so that (1) has a unique classical solution.

As an initial simplification, we set the (potentially) nonlinear term $\tau = 0$ and assume homogenous Dirichlet boundary conditions, obtaining a Poisson equation of the form

$$\begin{aligned} -\Delta u(x) &= f(x), \quad x \in \Omega, \\ u(x) &= 0, \quad x \in \partial\Omega. \end{aligned} \tag{2}$$

To construct a kernel-based method for numerically solving (2), we assume that we are given a PDS kernel K with RKHS $\mathcal{H}$[1]. Consider M collocation points distributed such that $X_I = \{x_1, \ldots, x_m\} \in \Omega$, $X_B = \{x_{m+1}, \ldots, x_M\} \in \Omega$, and define the function $K_{\Delta,\Delta}$ to be the kernel function K with the negative Laplace operator applied to both inputs and the function K_Δ to be K with the negative Laplace operator applied to the first argument. We define

$$\mathbf{K} = \begin{pmatrix} K_{\Delta,\Delta}(X_I, X_I) & K_\Delta(X_I, X_B) \\ K_\Delta(X_I, X_B)^T & K(X_B, X_B) \end{pmatrix}, \tag{3}$$

where the input of a set represents the matrix of the function evaluated at each of its elements. For instance, $(K_{\Delta,\Delta}(X_I, X_I))_{jk} = K_{\Delta,\Delta}(x_j, x_k)$ for $j, k = 1, \ldots, m$. The upper left block corresponds to the contribution of Laplacian of the kernel applied to the interior points to the Laplacian of the solution u at the interior points. The upper right block corresponds to the contribution of the boundary kernel points to the Laplacian of the solution u on the interior. The lower left block corresponds to the contribution to the Laplacian of the kernel at the interior points applied to the boundary, and the bottom right block corresponds to contribution of the boundary terms to themselves.

An approximate solution to (2) is obtained by solving the linear system

$$\mathbf{K}\alpha = \begin{pmatrix} f(X_I) \\ \mathbf{0} \end{pmatrix}, \tag{4}$$

for the coefficient vector $\alpha \in \mathbb{R}^M$ where $\mathbf{0} \in \mathbb{R}^{M-m}$ is the zero vector. The upper $f(X_I)$ block of the vector corresponds to the forcing on the RHS, and the zero block corresponds to the homogeneous Dirichlet boundary conditions. In practice, this system is very close to singular, leading us to introduce a nugget term; we discuss this in Section 3. Given the coefficient vector α, the approximate solution function is recovered as

$$u^*(x) = (K_\Delta(x, X_I), K_\Delta(x, X_B))^T \alpha.$$

[1]Throughout our computations, all kernels are taken to be the RBF kernel.

In the more general frame, a solution to the (potentially) nonlinear PDE (1) is approximated by a solution to the minimization problem

$$\begin{cases} \min_{u\in\mathcal{H}} \|u\|_{\mathcal{H}} \\ \text{s.t.} -\Delta u(x_j) + \tau(u(x_j)) = f(x_j), & x_j \in X_I, \\ \quad u(x_j) = g(x_j), & x_j \in X_B. \end{cases} \tag{5}$$

Applying the representer theorem to (5) and relaxing equality constraints yields the optimization problem

$$\min_{\alpha\in\mathbb{R}^M} \| -\Delta u(X_I) + \tau(u(X_I)) - f(X_I)\|^2 + \|u(X_B) - g(X_B)\|^2 + \lambda\alpha^T\mathbf{K}\alpha, \tag{6}$$

where $\lambda > 0$ is a parameter, $-\Delta u(X_I)$ and $\tau(u(X_I))$ are extracted from $\mathbf{K}\alpha$, and $\mathbf{K}$ is the joint kernel matrix generated by $\delta_y \circ \Delta$, and δ_y applied to the interior points and only δ_y on the boundary. We discuss solving (6) numerically in Section 3.

Finally, we provide an extension to this framework for simultaneously learning the RHS function f and solving the PDE (2). We do this by parametrizing both the solution u and the RHS function f using kernels and solving the problem

$$\min_{\alpha,\beta} \| -\Delta u_\alpha(X_I) - f_\beta(X_I)\|^2 + \|u_\alpha(X_B) - g(X_B)\|^2 + \lambda_u\alpha^T\mathbf{K}_u\alpha + \lambda_f\beta^T\mathbf{K}_f\beta,$$

where the α, β subscripts are to denote which function the variables are associated with, $\lambda_u, \lambda_f > 0$ are parameters, $\mathbf{K}_u$ is the kernel matrix corresponding to u defined in equation 3, $\mathbf{K}_f$ is the kernel matrix applied to the interior points corresponding to the function f, and α, β are found over the space of real vectors of appropriate dimension. Then, $u(x)$ is defined similarly to the solution of problem 2, and $f(x) := K_f(x, X_I)\beta$. Since this is a quadratic objective, it can be reduced to solving a (large) linear system.

3. Algorithm Implementation and Development

We use the Jax Python package [2] extensively throughout this work to compute derivatives of kernels and for the outer nonlinear optimization loops. We use 2000 randomly sampled interior points and 1200 boundary points in all of the problems. We set our nugget term to be a multiple of the diagonal of the kernel matrices, wherever applicable. In general, this multiple is taken to be 10^{-10}. To solve the nonlinear problem, we employ the Levenberg–Marquardt algorithm [4, 5]. Additionally, for generating very high accuracy solutions in the linear case, we employ iterative refinement, taking the residual from solving the linear system (4) with a nugget term, solving a secondary linear system with the residual on the right hand side, and adding the coefficients obtained in the two stages together. See Figure 1 for an example solution.

4. Computational Results

As a first example, we consider numerically solving the linear problem (2) on $[0, 1]^2$ with $f(x_1, x_2) = 13\pi^2 \sin(3\pi x_1)\sin(2\pi x_2)$. In Figure 2, we compare the computed numerical solution with the exact solution given by $u(x_1, x_2) = \sin(3\pi x_1)\sin(2\pi x_2)$. In Figure 3, we plot the max norm error for the problem (2) with a different RHS f for various numbers of collocation points M. We observe that this error decreases as M increases, so our method is, in fact, converging. To demonstrate the efficacy of this method on a variety of domains

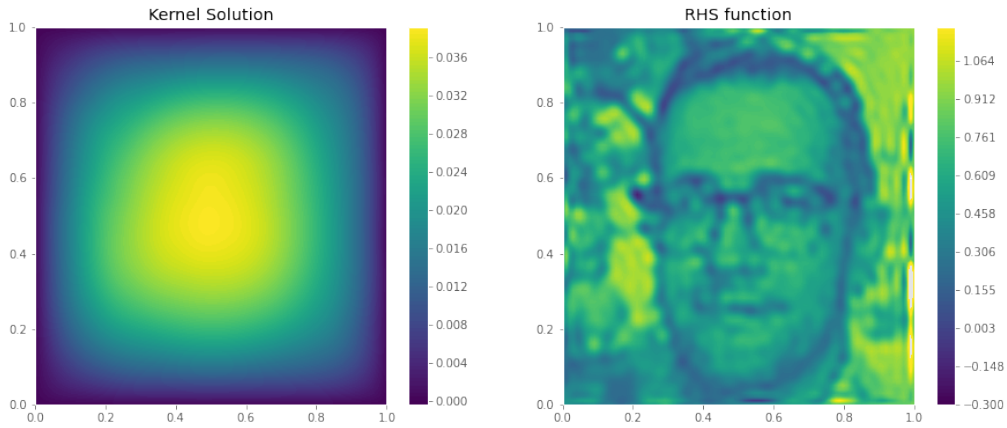

FIGURE 1. Solution to a Poisson equation with 2500 uniformly spaced interior and 1200 boundary collocation points computed using iterative refinement. The artifacts in the image on the right are a result of the interpolation and are unrelated to the kernel solver.

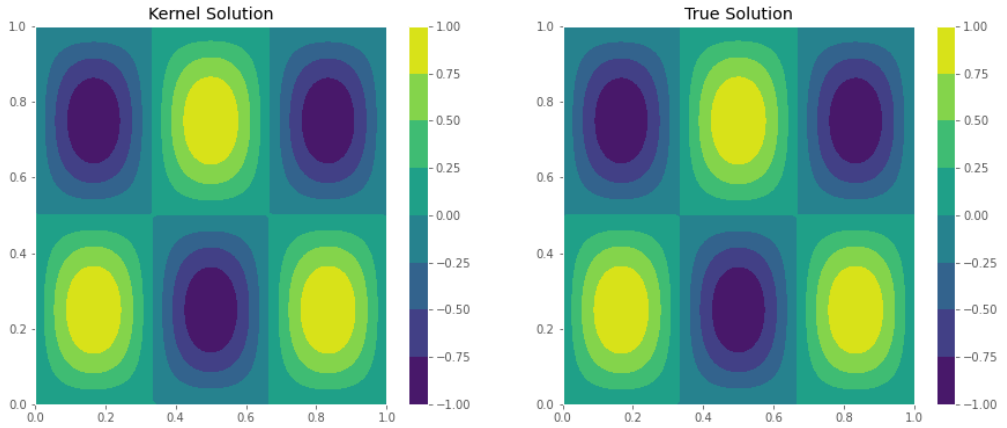

FIGURE 2. Plots of the kernel approximation (left) and true solution (right) of the Poisson equation in two dimensions (on the horizontal and vertical axes) with RHS function $f(x_1, x_2) = 13\pi^2 \sin(3\pi x_1)\sin(2\pi x_2)$. The max norm residual is 6.87e-05, and the max norm error is 7.16e-08.

Ω, we include a plot of a numerical solution to the Poisson equation on a triangular domain in Figure 4.

To incorporate nonlinearities, we consider the PDE (1) with $\tau(u) = u^2$ and $g(x) = 0$. In order to test accuracy, we prescribe the solution $u_{true}(x_1, x_2) = x_1(1-x_1)x_2(1-x_2)\cos(5\pi((0.5-x_1)^2+(0.3x_2)^2))$ and set $f(x_1, x_2) = -\Delta(u_{true}) + u_{true}^2$. In Figure 5, we plot the true solution to (1) with these functions and the forcing function f. In Figure 6, we plot the numerical solution obtained by solving the optimization problem (6) against the true solution to (1) from Figure 5. In Figure 7, we plot the error between this numerical solution and the true

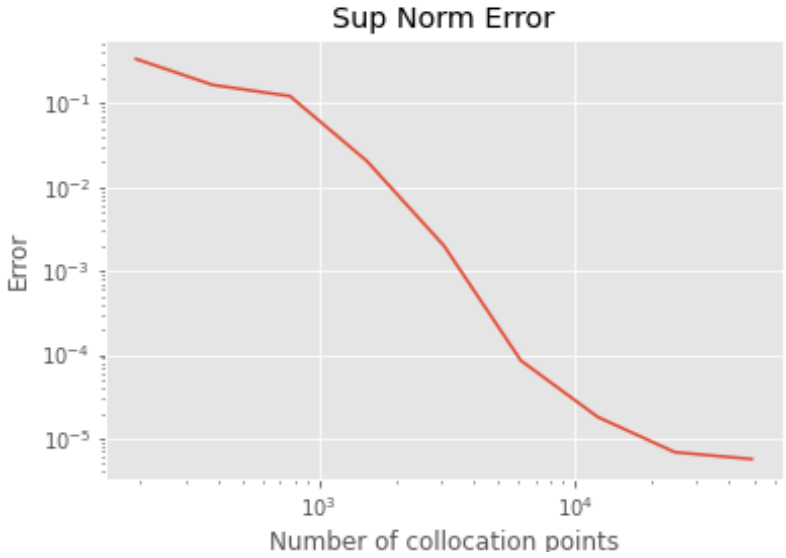

FIGURE 3. Plot of the max norm error of the kernel approximation to the Poisson equation in two dimensions with true solution $u_{true}(x_1, x_2) = x_1(1 - x_1)x_2(1 - x_2)\cos(5\pi((0.5 - x_1)^2 + (0.3x_2)^2))$ for various numbers of collocation points.

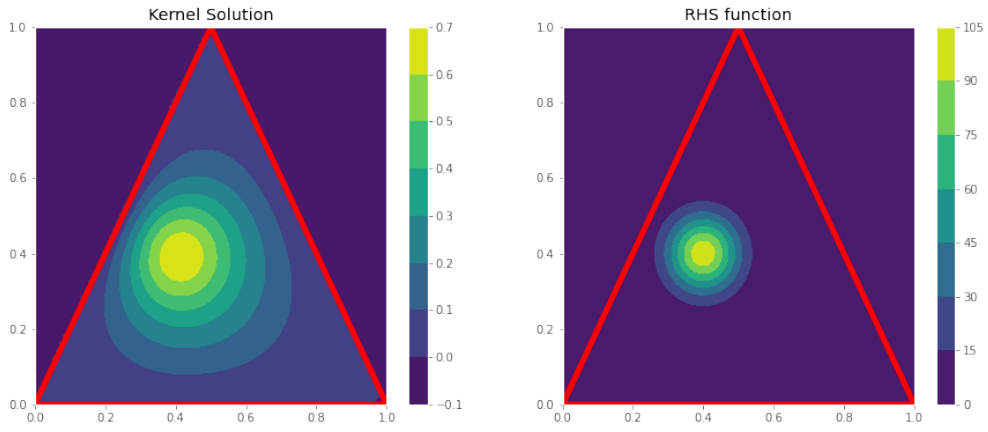

FIGURE 4. Plot of the kernel approximation to the Poisson equation on a triangular domain (left) for a given RHS function (right).

solution. We observe that this error is quite a bit larger than in the linear case, but this is to be expected since nonlinear PDEs are fundamentally harder problems.

Finally, we exhibit solving and learning PDEs simultaneously. Specifically, we are given data sampled from the solution to a Poisson equation (2) where the RHS function f is treated as an unknown, as pictured in Figure 8. In Figure 9, we plot the learned solution to (2) against the true solution from which the data are sampled. In Figure 10, we plot the learned approximation to the RHS function f against the true RHS used to generate the data.

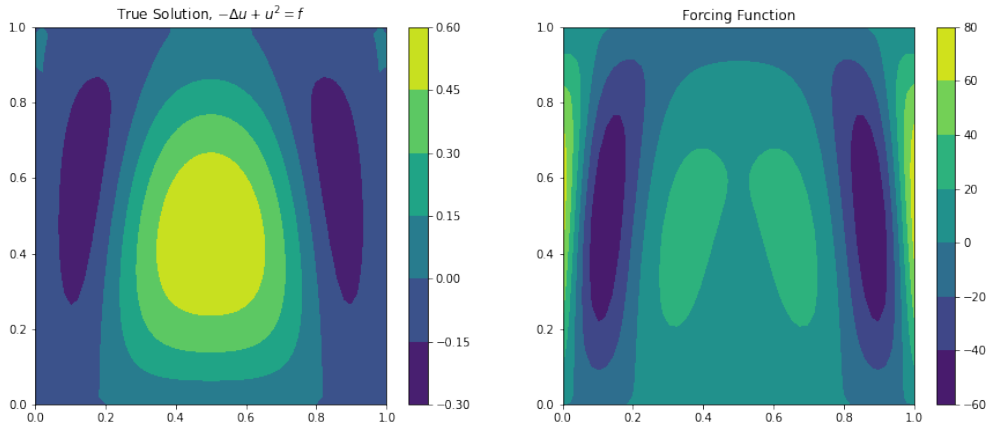

FIGURE 5. Plots of the solution of the PDE $-\Delta u + u^2 = f$ on $[0,1]^2$ (left) and the RHS function f (right).

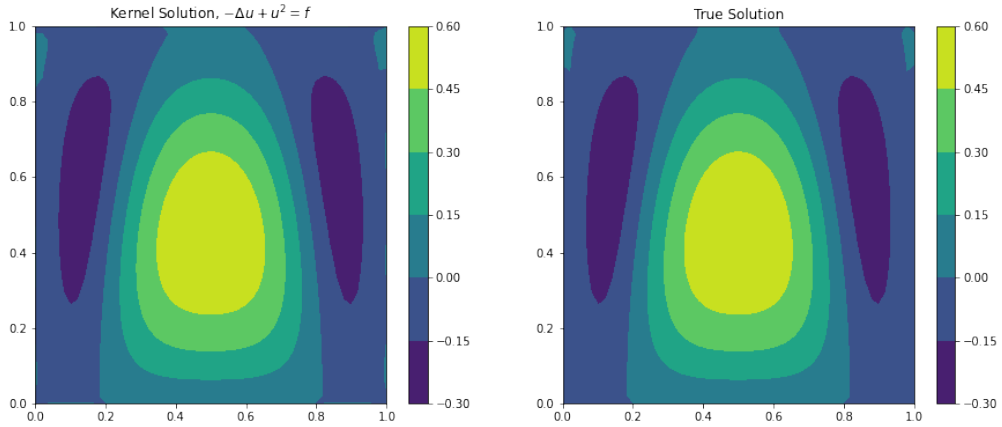

FIGURE 6. Plots of the kernel approximation to the PDE $-\Delta u + u^2 = f$ on $[0,1]^2$ (left) and the true solution to this PDE (right).

5. Summary and Conclusions

In this work, we have outlined and implemented kernel methods for numerically solving both linear and nonlinear PDEs and a method for learning and solving a PDE simultaneously. Our results indicate that such methods are quite robust in their applicability to a variety of problems. For example, while PDEs with irregular geometries can pose difficulties for classical numerical methods, when kernel methods are applied, they only impose the added difficulty of sampling points on an irregular domain. We note that our implementation did not achieve the high level of accuracy that we would hope for in the nonlinear case, only getting the first three digits correct. This could likely be improved by further tuning the hyperparameters in the optimization, kernels, and nugget terms. Additionally, the accuracy

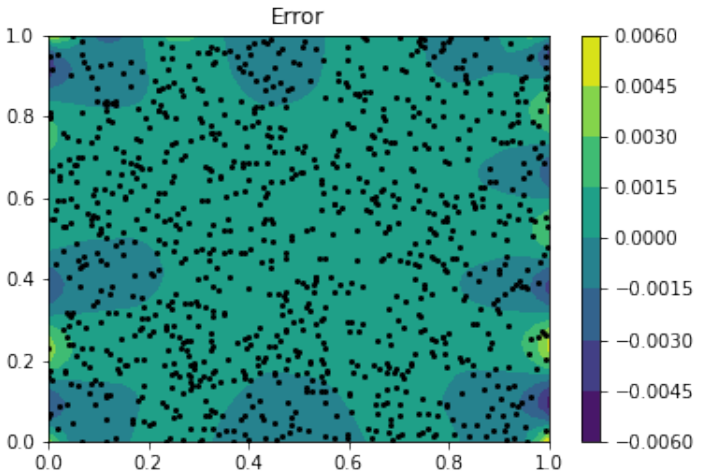

FIGURE 7. A plot of the difference between the kernel approximation to the PDE $-\Delta u + u^2 = f$ on $[0,1]^2$ and the true solution to this PDE.

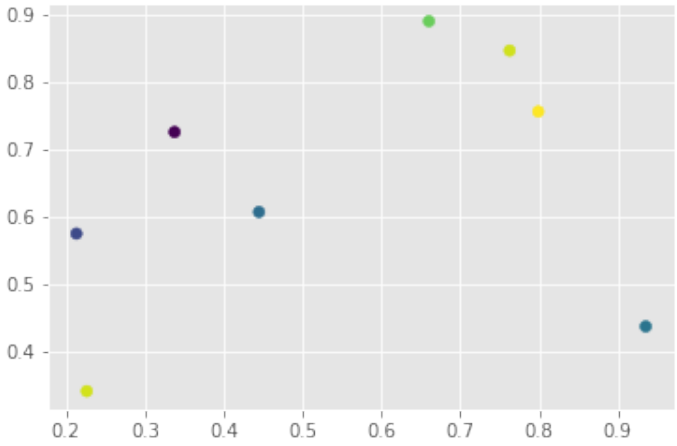

FIGURE 8. 8 randomly sampled points in $[0,1]^2$ of the solution to (2) on this domain for some RHS f.

throughout could be improved by more carefully solving the linear systems, and in the nonlinear case, by employing a method that doesn't explicitly form the normal equations since as this squares the condition number of the system that is solved. Future work would exclude our framework for simultaneously solving and learning the RHS to nonlinear problems and learning more structural parameters.

ACKNOWLEDGEMENTS

We acknowledge Bamdad Hosseini's attempts to scare Cade into starting earlier—he was successful, but not for the reasons he originally intended.

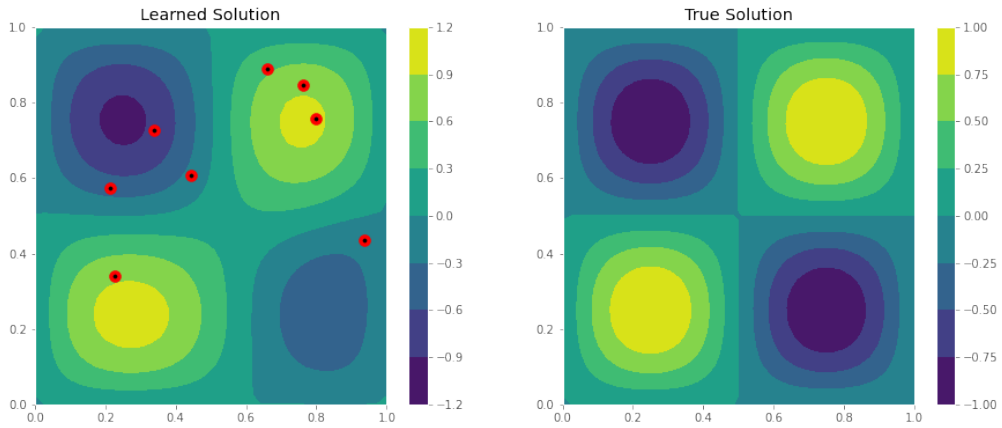

FIGURE 9. Plots of a numerical solution to (2) on $[0,1]^2$ for an unknown RHS f (left) learned from 8 data points (red) and the true solution to this PDE (right).

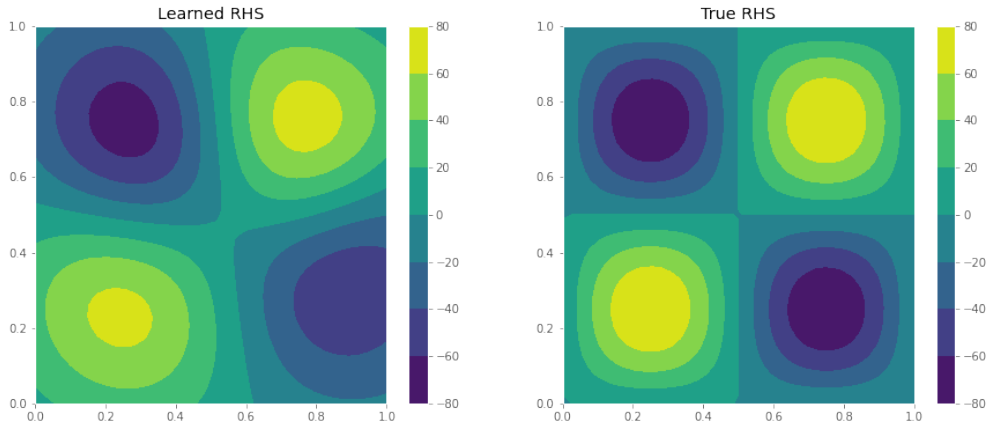

FIGURE 10. Plots of a learned approximation to the RHS function f (left) from sampling 8 points of the solution to the PDE (2) on $[0,1]^2$ and the true function f (right).

REFERENCES

[1] C. Ballew and A. Hsu. https://github.com/cade-b/kernel-pdes, 2023.
[2] 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.
[3] Y. Chen, B. Hosseini, H. Owhadi, and A. M. Stuart. Solving and learning nonlinear PDEs with Gaussian processes. *Journal of Computational Physics*, 447:110668, 2021.
[4] K. Levenberg. A Method for the Solution of Certain Non-Linear Problems in Least Squares. *Quarterly of Applied Mathematics*, 2(2):164–168, 1944.
[5] D. W. Marquardt. An Algorithm for Least-Squares Estimation of Nonlinear Parameters. *Journal of the Society for Industrial and Applied Mathematics*, 11(2):431–441, 1963.

[6] H. Owhadi. Bayesian Numerical Homogenization. *Multiscale Modeling & Simulation*, 13(3):812–828, 2015.