Finite Basis Physics-Informed Neural Networks as a Schwarz Domain Decomposition Method

Dolean, Victorita; Heinlein, Alexander; Mishra, Siddhartha; Moseley, Ben

DOI
10.1007/978-3-031-50769-4_19

Publication date
2024

Document Version
Final published version

Published in
Domain Decomposition Methods in Science and Engineering XXVII

Finite Basis Physics-Informed Neural Networks as a Schwarz Domain Decomposition Method

Victorita Dolean, Alexander Heinlein, Siddhartha Mishra, and Ben Moseley

1 Introduction

The success and advancement of machine learning (ML) in fields such as image recognition and natural language processing has lead to the development of novel methods for the solution of problems in physics and engineering. However, algorithms developed in traditional fields of ML usually require a large amount of data, which are difficult to obtain from measurements and/or traditional numerical simulations. Furthermore, such algorithms can be difficult to interpret and can struggle to generalize. To overcome these issues, a new research paradigm has emerged, known as scientific machine learning (SciML) [1, 7], which aims to more tightly combine ML with scientific principles to provide more powerful algorithms.

One such approach are physics-informed neural networks (PINNs) [4, 10], which are designed to approximate the solution to the boundary value problem

$$\begin{aligned} \mathcal{N}[u](\mathbf{x}) &= f(\mathbf{x}), \quad \mathbf{x} \in \Omega \subset \mathbb{R}^d, \\ \mathcal{B}_k[u](\mathbf{x}) &= g_k(\mathbf{x}), \quad \mathbf{x} \in \Gamma_k \subset \partial\Omega \end{aligned} \tag{1}$$

where $\mathcal{N}[u](\mathbf{x})$ is a differential operator, u is the solution and $\mathcal{B}_k(\cdot)$ is a set of boundary conditions, such that the solution u is uniquely determined. Note that boundary conditions are to be understood in a broad sense and the $\mathbf{x}$ variable can also include time. In particular, we do not distinguish between initial and boundary conditions.

Victorita Dolean
University of Strathclyde, 26, Richmond Street, G1 1XH Glasgow, UK,
e-mail: work@victoritadolean.com

Alexander Heinlein
Delft University of Technology, Faculty of Electrical Engineering Mathematics & Computer Science, Delft Institute of Applied Mathematics, Mekelweg 4, 2628 CD Delft, Netherlands,
e-mail: a.heinlein@tudelft.nl

Siddhartha Mishra, Ben Moseley
ETH Zürich, Computational and Applied Mathematics Laboratory / ETH AI Center, Rämistrasse 101, 8092 Zürich, Switzerland

Z. Dostál et al. (eds.), *Domain Decomposition Methods in Science and Engineering XXVII*, Lecture Notes in Computational Science and Engineering 149,
https://doi.org/10.1007/978-3-031-50769-4_19

The approximation to the solution of (1) is given by a neural network $u(\mathbf{x}, \boldsymbol{\theta})$ (for the sake of simplicity we use the same notation for the solution of the PDE and the neural network) where $\boldsymbol{\theta}$ is a vector of all the parameters of the neural network (i.e., its weights and biases). The network is trained via the loss function

$$\mathcal{L}(\boldsymbol{\theta}) = \underbrace{\frac{\lambda_I}{N_I}\sum_{i=1}^{N_I}(\mathcal{N}[u](\mathbf{x}_i, \boldsymbol{\theta}) - f(\mathbf{x}_i))^2}_{\mathcal{L}_{\text{PDE}}} + \underbrace{\sum_{k=1}^{N_k}\frac{\lambda_B^k}{N_B^k}\sum_{j=1}^{N_B^k}(\mathcal{B}_k[u](\mathbf{x}_j^k, \boldsymbol{\theta}) - g_k(\mathbf{x}_j^k))^2}_{\mathcal{L}_{\text{BC}}}. \tag{2}$$

Here, $\{\mathbf{x}_i\}_{i=1}^{N_I}$ is a set of collocation points sampled in the interior of the domain, $\{\mathbf{x}_j^k\}_{j=1}^{N_B^k}$ is a set of points sampled along each boundary condition, and λ_I and λ_B^k are well-chosen scalar hyperparameters which ensure that the terms in the loss function are well balanced. Intuitively, one can see that the PDE loss tries to ensure that the solution learned by the network obeys the underlying PDE whilst the boundary loss tries to ensure it obeys the boundary conditions.

In practice, the presence of the boundary loss in eq. (2) often slows down training as it can compete with the PDE term [12]. In a slightly different formulation, boundary conditions can instead be enforced exactly as hard constraints by using the neural network as part of a solution ansatz Cu where C is a constraining operator which enforces that the solution explicitly satisfies the boundary conditions [4, 8]. This turns the optimization problem into an unconstrained one, and only the PDE loss from eq. (2) is required to train the PINN. For example, suppose we want to enforce that $u(0) = 0$ when solving a one-dimensional ODE, then the ansatz and constraining operator can be chosen as $[Cu](x, \boldsymbol{\theta}) = \tanh(x)u(x, \boldsymbol{\theta})$. The rationale behind this is that the function $\tanh(x)$ is null at 0, forcing the boundary condition to be obeyed, but non-zero away from 0, allowing the network to learn the solution away from the boundary condition.

Whilst PINNs have proven to be successful for solving many different types of differential equations, they often struggle to scale to problems with larger domains and more complex, multi-scale solutions [8, 11]. This is in part due to the spectral bias of neural networks [9] (their tendency to learn higher frequencies much slower than lower frequencies), and the increasing size of the underlying PINN optimization function. One way to alleviate these scaling issues is to combine PINNs with a domain decomposition method (DDM); by taking a divide-and-conquer approach, one hopes that the large, global optimization problem can be turned into a series of smaller and easier localized problems. In particular, [8] proposed finite basis physics-informed neural networks (FBPINNs) where the global PINN is partitioned into many local networks that are trained to approximate the solution on an overlapping domain decomposition. Related approaches are the deep domain decomposition method (D3M) [5] and the deep-learning-based domain decomposition method (DeepDDM) [6], which combine overlapping Schwarz domain decomposition methods with a PINN-based discretization. Other earlier works on the use

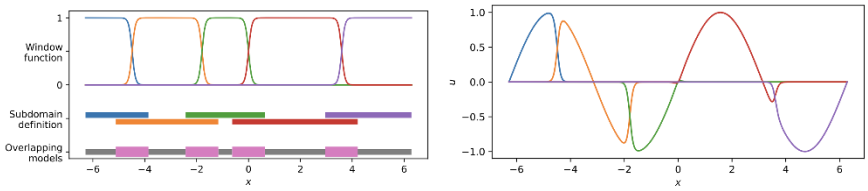

Fig. 1 Local FBPINN subdomains and window functions w_j (left), local solutions u_j (right)

of machine learning and domain decomposition methods include the prediction of the geometrical location of constraints in adaptive FETI-DP and BDDC methods; see [2]. For an overview of the combination of domain decomposition methods and machine learning, see [3].

In this work, we build upon FBPINNs by showing how Schwarz-like additive, multiplicative and hybrid iterative training strategies for FBPINNs can be developed. We present numerical experiments on the influence of these training strategies on convergence and accuracy. We propose and evaluate a preliminary implementation of a coarse space correction for FBPINNs, to further improve their efficiency.

2 Finite basis physics-informed neural networks (FBPINNs)

First we briefly present the FBPINN method introduced by [8] from a DDM perspective. The FBPINN method can be seen as a network architecture that allows for a localization of the network training. Therefore, let us consider a set of collocation points $X = \{\mathbf{x}_i\}_{i=1}^{N}$ in the global domain Ω and a decomposition into overlapping domains $\Omega = \cup_{j=1}^{J} \Omega_j$ inducing a decomposition into subsets of collocation points $X_j = \{\mathbf{x}_i^j\}_{i=1}^{N_j}$, $j = 1, \ldots, J$. As usual in overlapping Schwarz methods, $X = \cup_{j=1}^{J} X_j$ is not disjoint. For each subdomain Ω_j, we denote $\mathcal{N}_j$ the index set of neighboring subdomains, $\Omega_j^\circ = \cup_{l=1}^{\mathcal{N}_j} \Omega_l \cap \Omega_j$ the overlapping subset of Ω_j, and $\Omega_j^{\text{int}} = \Omega_j \setminus \Omega_j^\circ$, the interior part of the domain; let X_j° and X_j^{int} be the corresponding sets of collocation points, and $X^\circ = \cup_{j=1}^{J} X_j^\circ$ and $X^{\text{int}} = \cup_{j=1}^{J} X_j^{\text{int}}$. We now define the global network u as the sum of local networks $u_j(\mathbf{x}, \boldsymbol{\theta}_j)$ weighted by window functions ω_j: $u = \sum_{j, \mathbf{x}_i \in \Omega_j} \omega_j u_j$. Here, the local networks have individual network parameters $\boldsymbol{\theta}_j$, and of course, they could simply be evaluated everywhere in $\mathbb{R}^d$. In order to restrict them to their corresponding overlapping subdomains, we multiply them with the window functions, which have the properties $\text{supp}(\omega_i) \subset \Omega_i$ and $\Omega \subset \cup_{j=1}^{J} \text{supp}(\omega_i)$; the specific definition of ω_i employed here can be found in [8, eq. (14)]. See Fig. 1 for a graphical representation of the overlapping subdomains, their overlapping and interior sets, window functions, and local solutions for a simple one-dimensional example. If we insert the expression for u into eq. (2), we see that the loss function can be written as:

$$\mathcal{L}(\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_J) = \frac{1}{N} \sum_{i=1}^{N} \left(\mathcal{N}[C \sum_{j, \mathbf{x}_i \in X_j} \omega_j u_j](\mathbf{x}_i, \boldsymbol{\theta}_j) - f(\mathbf{x}_i) \right)^2 . \tag{3}$$

The contribution to the global loss function can be split into a part coming from interior points and another one from points in the overlap, respectively:

$$\begin{aligned} \mathcal{L}(\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_J) = & \underbrace{\frac{1}{N} \sum_{\mathbf{x} \in X^{\text{int}}} \left(\mathcal{N}[C \sum_{l, \mathbf{x} \in X_l} \omega_l u_l](\mathbf{x}, \boldsymbol{\theta}_l) - f(\mathbf{x}) \right)^2}_{=: \mathcal{L}^{\text{int}}(\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_J)} \\ & + \frac{1}{N} \sum_{\mathbf{x} \in X^{\circ}} \left(\mathcal{N}[C \sum_{l, \mathbf{x} \in X_l} \omega_l u_l](\mathbf{x}, \boldsymbol{\theta}_l) - f(\mathbf{x}) \right)^2 . \end{aligned} \tag{4}$$

Note also that, since $X_i^{\text{int}} \cap X_j^{\text{int}} = \emptyset$ for $i \neq j$, the interior contribution can be simplified as follows:

$$\mathcal{L}^{\text{int}}(\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_J) = \frac{1}{N} \sum_{j=1}^{J} \sum_{\mathbf{x}_i \in X_j^{\text{int}}} \left(\mathcal{N}[C \omega_j u_j](\mathbf{x}_i, \boldsymbol{\theta}_j) - f(\mathbf{x}_i) \right)^2 .$$

In [8], the authors introduce the notion of *scheduling* which is related to the degree of parallelism one can consider in Schwarz domain decomposition methods. For example in the well-know alternating Schwarz method, local solves take place sequentially, in an alternating manner, with data being exchanged at the interfaces. In the case of the parallel Schwarz method, local solutions are computed simultaneously, but subdomains only have access to interface data at the previous iteration. As is well-known in DDMs, the alternating method convergences in fewer iterations than the parallel method, whereas the second methods allows the concurrent computation of the local solutions; hence, the parallel Schwarz method is often more efficient in a parallel implementation.

In the case of many subdomains, one can define a so-called coloring strategy, i.e., subdomains with the same color are computed in parallel and different colors are processed sequentially. Here, we will consider any possible coloring scheme, allowing for arbitrary combinations of additive and multiplicative coupling. In particular, let us split the set of subdomain indices as follows $\{1, \ldots, J\} = \mathcal{A} \cup \mathcal{I}$, such that subdomains Ω_j, $j \in \mathcal{A}$ are allocated the same 'color' which is different than those of the subdomains Ω_j, $j \in \mathcal{I}$. In the case of training FBPINNs, the notion of coloring is replaced by that of *scheduling*, that is, subdomains indexed in $\mathcal{A}$ are considered to be *active* at a given iteration and those indexed in $\mathcal{I}$ are *inactive*. The case when $\mathcal{I} = \emptyset$ corresponds to the fully parallel Schwarz method, whereas the case where only one subdomain is *active* at a time corresponds to a fully alternating Schwarz iteration. Denoting a subdomain Ω_j as *inactive* corresponds to fixing $\boldsymbol{\theta}_j$ during the optimization of $\mathcal{L}(\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_J)$.

Algorithm 1 FBPINN training step for each subdomain

if $j \in \mathcal{A}$ (Ω_j is an active domain) **then**

Perform p iterations of gradient descent on $\boldsymbol{\theta}_j^k$ ($\boldsymbol{\theta}_i^k$ where $i \neq j$ are kept fixed):

$$\boldsymbol{\theta}_j^{k+l} = \boldsymbol{\theta}_j^{k+l-1} - \lambda \nabla_{\boldsymbol{\theta}_j} \mathcal{L}(\boldsymbol{\theta}_1^k, \ldots, \boldsymbol{\theta}_{j-1}^k, \boldsymbol{\theta}_j^{k+l-1}, \boldsymbol{\theta}_{j+1}^k, \ldots, \boldsymbol{\theta}_J^k), l = 1, \ldots, p.$$

Update the solution in the overlapping regions (communicate with neighbours):

$$\forall \mathbf{x} \in \Omega_j^\circ,\ u(\mathbf{x}, \boldsymbol{\theta}_j^{k+p}) \leftarrow \sum_{l, \mathbf{x} \in \Omega_l} \omega_l u_l(\mathbf{x}, \boldsymbol{\theta}_l^{k+p}).$$

end if

The FBPINN training algorithm follows the 'coloring' strategy described above. Let us denote by $\boldsymbol{\theta}_j^k$ the parameter values at the k-th training step, and to simplify the presentation, we focus on the case of a first order gradient-based optimizer. If we start from an initial guess $\boldsymbol{\theta}_j^0$, then the training step for each subdomain is given by Algorithm 1. Once all active subdomains have completed one training step, the set $\mathcal{A}$ and $\mathcal{I}$ are updated. This whole procedure is repeated until any stopping criterion, such as a maximum number of iterations or a tolerance for the loss, is met.

Let us note that:

- The gradient updates can be performed in parallel and are fully localized even if the loss function is global; only in the update step are network solutions and network gradients transferred between neighboring subdomains.
- It is not necessary to perform communication in the overlaps (here in orange) at every iteration of gradient descent, but rather every p iterations for a better computational efficiency. The overall convergence can also be affected; cf. Fig.2.
- Unlike in classical domain decomposition methods, in our approach, the global problem is not decomposed into local problems, which can be solved independently. Instead, we always compute gradient updates with respect to the global loss function, and the domain decomposition and hence the localization enters through the window functions in the definition of the architecture of global network.

To illustrate the behavior of this algorithm we will consider a scaling study for its flexible training strategy. In particular, we fix the number of global collocation points and investigate the influence of changing the number of subdomains and the value of p on convergence when solving the simple 1D ODE $\frac{du}{dx} = \cos \omega x$, $u(0) = 0$, with $\omega = 15$. All subdomains are kept active all of the time, and all other FBPINN design choices are kept the same, including window function and local network architecture per subdomain. We only consider the case of relatively large overlap of 70% of the subdomain width, but the results are qualitatively the same for other sizes of overlap. As discussed in [8] and as in classical overlapping Schwarz methods, performance generally improves when increasing the size of the overlap; a systematic investigation is still open.

In Figure 2, we display the convergence of the loss function when the communication between subdomains takes place every $p \in \{1, 10, 100, 1000\}$ epochs. We ob-

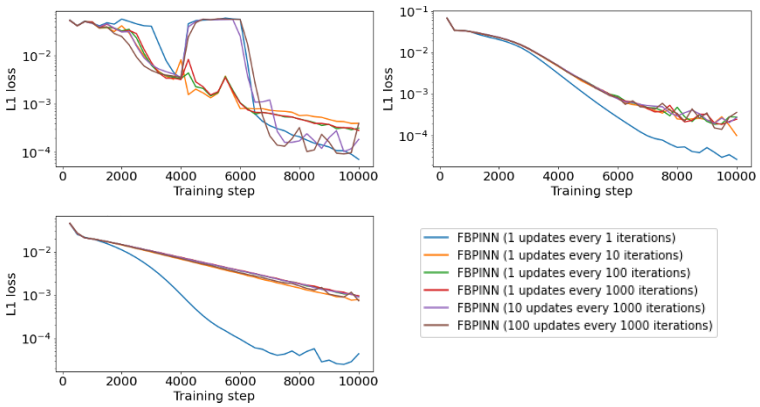

Fig. 2 FBPINN convergence for decompositions into 8 (upper left), 16 (upper right) and 32 (lower left) subdomains and different p values. Each network has 2 layers and 16 hidden units per layer. A total of 3,000 collocation points regularly sampled over the domain are used. For each decomposition, subdomains are regularly spaced, with an overlap of 70% of the subdomain width.

serve that the case of 8 subdomains is rather special since convergence appears rather unstable and there is no option that performs clearly best. As we increase the number of subdomains to 16 and 32 we observe an expected behavior, that is, the convergence rate improves if we communicate solutions and gradients in the overlaps every iteration. Moreover, when increasing the number of subdomains, naturally the global training performs less well, which is well known in domain decomposition as lack of scalability; we observe this behavior for all values of p. This is expected because the method above corresponds to a one-level method (meaning only neighboring subdomains communicate and there is no global exchange of information). Surprisingly, we do not see any clear difference in the convergence depending on p, that is, depending on how often we communicate. Since, in a parallel setting, it is computationally more efficient to communicate less, the results seems to indicate that, if we do not communicate in each step, it is beneficial to communicate as little as possible.

3 Coarse correction

Coarse spaces are instrumental in DDMs, as they ensure the robustness of a given method with respect to the number of subdomains as well as other problem-specific parameters, such as physical properties like frequency for wave problems or conductivity for diffusion type problems. Coarse spaces are often defined based on geometrical information (like a coarser mesh) but more sophisticated coarse spaces can be constructed using spectral information of underlying local problems. When training PINNs, it is not immediately clear how to define a coarse space, that is, a coarse network model, nor how to choose the number of collocation points and parameters of the coarse model. In what follows, we propose and evaluate a preliminary implementation of a coarse space correction for FBPINNs.

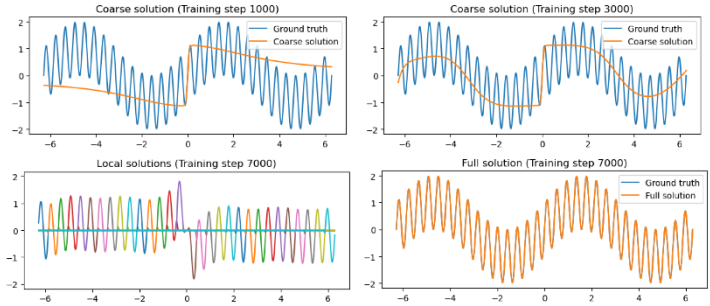

Fig. 3 Coarse correction for FBPINNs. Each subdomain network and the coarse network has 2 layers and 16 hidden units per layer. A total of 500 collocation points regularly sampled over the domain are used to train the coarse network, and 3,000 for the local networks. For the local networks, the subdomains are regularly spaced, with an overlap of 70% of the subdomain width.

In particular, we exploit the spectral bias of neural networks in order to build a coarse correction. This is the well-studied phenomenon that they tend to learn higher frequencies much slower than lower frequencies [9], and similar effects are observed for PINNs [8, 11]. Indeed, this effect is what motivated the use of domain decomposition in FBPINNs. More precisely, we first train a small but global network for enough epochs to learn the low frequency component of the solution; in particular, we employ a coarse network with the same architecture as a single local network. Then, local subdomains are added to approximate missing higher frequency components. The resulting FBPINN solution is given by $u = u_g + \sum_{j,\mathbf{x}_i \in \Omega_j} \omega_j u_j$, where $u_g(\mathbf{x}, \boldsymbol{\theta}_g)$ is the coarse network and $u_j(\mathbf{x}, \boldsymbol{\theta}_j)$ are the local networks. Because of spectral bias, low frequencies are first learned by the coarse network, and a relatively small network is sufficient to approximate the low frequencies. Then the local networks only need to learn the remaining higher frequencies. Since the local models only have to learn a local part of the solution, relatively small local network models are also sufficient.

We will apply these ideas on the simple 1D ODE, $\frac{du}{dx} = \omega_1 \cos(\omega_1 x) + \omega_2 \cos(\omega_2 x)$, $u(0) = 0$, where two frequencies are present in the solution, $u(x) = \sin(\omega_1 x) + \sin(\omega_2 x)$. For our test case, we choose $\omega_1 = 1$ as a lower frequency and $\omega_2 = 15$ as the higher frequency and we decompose the global domain into 30 overlapping subdomains; see Fig. 3. We note, as shown in [8] for an ODE with a single high frequency, solving such a problem with a single PINN requires a high network complexity and large number of iterations. First, we train the global coarse network, u_g, until the lower frequency is learned. We illustrate this progressive process in Fig. 3 where we see that we need roughly 3,000 epochs to identify the lower frequency. Here, we have chosen the number of epochs by hand based on the accuracy of the coarse solution, but in the future, we will work on automating the training of the coarse network. Then, the coarse network is fixed and the local networks are trained to approximate the remaining component of the solution, with all local networks kept active at each training step. As can be seen in Fig. 3, using our proposed approach, the coarse network approximates the coarse component of the solution, and the local subdomain networks approximate the high frequency components on the local subdomains.

4 Conclusions

In this work, we provide first insights on how to incorporate techniques from classical Schwarz domain decomposition methods into the FBPINN method. We show that its algorithmic components can be translated in the language of domain decomposition methods, and the well-established notions of additive, multiplicative and, hybrid Schwarz iterations can be identified through the notion of the flexible scheduling strategies introduced in [8]. Finally, we start exploring the notion of coarse space for FBPINNs. In particular, we train a coarse network to approximate the low frequency components of the solution and then continue by training local networks to approximate the remaining high frequency components. These ideas can be extended in a straightforward way to other, more complex boundary value problems.

References

1. Baker, N., Alexander, F., Bremer, T., Hagberg, A., Kevrekidis, Y., Najm, H., Parashar, M., Patra, A., Sethian, J., Wild, S., Willcox, K., and Lee, S. Workshop Report on Basic Research Needs for Scientific Machine Learning: Core Technologies for Artificial Intelligence. Tech. rep., USDOE Office of Science (SC) (United States) (2019).
2. Heinlein, A., Klawonn, A., Lanser, M., and Weber, J. Machine learning in adaptive domain decomposition methods - predicting the geometric location of constraints. *SIAM Journal on Scientific Computing* **41**(6), A3887–A3912 (2019).
3. Heinlein, A., Klawonn, A., Lanser, M., and Weber, J. Combining machine learning and domain decomposition methods for the solution of partial differential equations – a review. *GAMM-Mitteilungen* **44**(1), e202100001 (2021).
4. Lagaris, I. E., Likas, A., and Fotiadis, D. I. Artificial neural networks for solving ordinary and partial differential equations. *IEEE Transactions on Neural Networks* **9**(5), 987–1000 (1998). 9705023.
5. Li, K., Tang, K., Wu, T., and Liao, Q. D3M: A deep domain decomposition method for partial differential equations. *IEEE Access* **8**, 5283–5294 (2019).
6. Li, W., Xiang, X., and Xu, Y. Deep domain decomposition method: Elliptic problems. In: *Mathematical and Scientific Machine Learning*, 269–286. PMLR (2020).
7. Moseley, B. *Physics-informed machine learning: from concepts to real-world applications*. Ph.D. thesis, University of Oxford (2022).
8. Moseley, B., Markham, A., and Nissen-Meyer, T. Finite Basis Physics-Informed Neural Networks (FBPINNs): a scalable domain decomposition approach for solving differential equations. *arXiv preprint arXiv:2107.07871* (2021).
9. Rahaman, N., Baratin, A., Arpit, D., Draxler, F., Lin, M., Hamprecht, F., Bengio, Y., and Courville, A. On the spectral bias of neural networks. In: Chaudhuri, K. and Salakhutdinov, R. (eds.), *Proceedings of the 36th International Conference on Machine Learning*, *Proceedings of Machine Learning Research*, vol. 97, 5301–5310. PMLR (2019).
10. Raissi, M., Perdikaris, P., and Karniadakis, G. E. 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).
11. Wang, S., Wang, H., and Perdikaris, P. On the eigenvector bias of Fourier feature networks: From regression to solving multi-scale PDEs with physics-informed neural networks. *Computer Methods in Applied Mechanics and Engineering* **384**, 113938 (2021). 2012.10047.
12. Wang, S., Yu, X., and Perdikaris, P. When and why PINNs fail to train: A neural tangent kernel perspective. *Journal of Computational Physics* **449**, 110768 (2022).