arXiv:2302.04107v1 [math.NA] 8 Feb 2023

CAN PHYSICS-INFORMED NEURAL NETWORKS BEAT THE FINITE ELEMENT METHOD?

Tamara G. Grossmann*, Urszula Julia Komorowska†, Jonas Latz‡ and Carola-Bibiane Schönlieb*

ABSTRACT

Partial differential equations play a fundamental role in the mathematical modelling of many processes and systems in physical, biological and other sciences. To simulate such processes and systems, the solutions of PDEs often need to be approximated numerically. The finite element method, for instance, is a usual standard methodology to do so. The recent success of deep neural networks at various approximation tasks has motivated their use in the numerical solution of PDEs. These so-called physics-informed neural networks and their variants have shown to be able to successfully approximate a large range of partial differential equations. So far, physics-informed neural networks and the finite element method have mainly been studied in isolation of each other. In this work, we compare the methodologies in a systematic computational study. Indeed, we employ both methods to numerically solve various linear and nonlinear partial differential equations: Poisson in 1D, 2D, and 3D, Allen–Cahn in 1D, semilinear Schrödinger in 1D and 2D. We then compare computational costs and approximation accuracies. In terms of solution time and accuracy, physics-informed neural networks have not been able to outperform the finite element method in our study. In some experiments, they were faster at evaluating the solved PDE.

1 Introduction

Partial differential equations (PDEs) are a corner stone of mathematical modelling and possibly applied mathematics itself. They are used to model a multitude of physical [40], biological [63], socioeconomic [11], and financial [24] systems and processes. Beyond classical modelling, PDEs can describe the evolution of certain stochastic processes [53], be used in image reconstruction [54], as well as for filtering [37] and optimal control [5] of dynamical systems.

The underlying idea is always the same: we aim to represent some process through a function that describes the behaviour of the process in space and time. The PDE is then a collection of laws that this function is supposed to satisfy. An obvious questions is, whether these laws are actually sufficient to uniquely specify this function [18]. Once uniqueness or even well-posedness [22] has been established, the question is how to find the function satisfying these laws that will ultimately be the mathematical model.

In many practical situations, it is impossible to find closed-form solutions for arising partial differential equations; they need to be solved numerically. Throughout the last decades, several numerical methods have been proposed and analysed to solve PDEs, especially the *finite element method* (FEM) [13, 25] that may be the standard methodology for a huge class of PDEs. Other techniques are, e.g., the finite difference [27], finite volume [19], and the spectral element [47] method. Due to the simplicity of the underlying approximation, many of these approaches are well-understood from a theoretical perspective: there are

*Department of Applied Mathematics and Theoretical Physics, University of Cambridge, Cambridge, UK tg410@cam.ac.uk

†Department of Computer Science and Technology, University of Cambridge, Cambridge, UK

‡Maxwell Institute for Mathematical Sciences and School of Mathematical and Computer Sciences, Heriot-Watt University, Edinburgh, UK

existing error estimators, as well as convergence and stability guarantees. Moreover, the given discretised problems are often in a form that can easily be solved numerically: relying on large, but sparse linear systems or on Newton solves with good initial values and convergence guarantees. Whilst implementing an FEM solver from scratch can be fairly tedious, several multipurpose computational libraries have appeared throughout the years, such as FEniCS [2] or DUNE [55].

A clear disadvantage of this classical methodology is that it usually relies on a spatial discretisation through, e.g., a spatial grid or a large polynomial basis, and thus, let it suffer from the *curse of dimensionality*: in three space dimensions, it can already be difficult to employ, e.g., the finite element method. In filtering and optimal control, we are easily interested in PDEs occurring in hundreds or thousands of dimensions – here, classical methodology can rarely be employed. In addition, certain non-linear and non-smooth PDEs are difficult to discretise with, e.g., finite elements due to behaviour that needs to be resolved on a very fine grid, general non-smooth behaviour, or singularities. Since FEM is a mesh-based solver, obtaining solutions on certain irregular domains demands tailored approaches that are challenging to design and solve.

In recent year, deep learning approaches have become a promising and popular methodology for the numerical solution of various PDEs. They have the potential to overcome some of the challenges that classical methods are facing. That is, neural networks have the advantage to not generally rely on a grid. By leveraging automatic differentiation [4], they eliminate the need for discretisation. Additionally, neural networks are able to represent more general functions than, e.g., an FEM basis, and they do not suffer from the curse of dimensionality. While the training of a neural network can become computationally demanding, especially when it consists of a non-convex optimisation problem, it is very efficient when evaluating new data points once trained. In this emerging field of deep learning for approximating PDE solutions, the class of approaches closest to the classical methodologies is the one of function approximators [62]. They essentially model the PDE solution by a deep neural network and train the network's parameters to approximate the solution. Such approaches are, for example, the Deep Ritz method [17] or the Deep Galerkin method [58]. A widely used and adapted method of this class are the so-called *physics-informed neural networks* (PINNs) [52] that will be the focus of this paper. Originally published in a two-part instalment [50, 51], Raissi et al. developed the vanilla PINNs approach [52]. The basic idea behind PINNs is to minimise an energy functional that is the residual of the PDE and its initial and boundary conditions. The neural network itself models the solution function $u(t, x)$ given input variables t and x based on the underlying PDE. It has shown great success for many different types of PDEs [44, 59] and has been extended to various specialised cases [28, 30, 64, 14].

While deep learning approaches for PDEs have gained a lot of traction in the last years and are being employed in increasingly more applications, they come with their set of challenges. In this work, we therefore systematically compare the finite element method and physics-informed neural networks in a computational study. Before giving a complete outline of the following, we review recent works on PINNs.

1.1 PINNs: state of the art.

As opposed to the finite element method, the theoretical groundwork for PINNs is rather sparse. The first work on convergence results with respect to the number of training points is by Shin et al. [57]. For linear second-order elliptic and parabolic PDEs, they prove strong convergence in C^0 for i.i.d. sampled training data. Mishra et al. [45] have in turn developed upper bounds on the generalisation error of PINNs given some stability assumptions on the PDE. Focusing on a specific PDE, Ryck et al. [16] investigate the incompressible Navier-Stokes equation and provide an upper bound on the total error. However, their considerations are limited to the case of networks with two hidden layers and $\tanh$ activation functions. Despite the remaining need for more extensive theoretical work, PINNs have been extended into many different directions. Jagtap et al. [28] develop conservative PINNs (cPINNs) to incorporate complex non-regular geometries by decomposing a spatial domain into independent parts and train separate PINNs for each. This work has been generalised to the extended PINNs (XPINNs) [15] to allow space-time domain decomposition that can be applied to any type of PDE and enables parallelisation in training. Another domain decomposition method for PINNs are the hp-VPINNs [31]. This work is based on the variational PINNs [30] that take inspiration from classical approaches for solving PDEs. VPINNs form the loss functional based on the variational form

of the PDE with Legendre polynomials as test functions. Thus, allowing, e.g., for certain non-smoothnesses in PDEs and also for more efficient training. Bayesian PINNs [64] for noisy data employ a Bayesian neural network. Finite Basis Physics-Informed Neural Networks [46] combine the PINN-idea with the finite element method, aiming to reduce the spectral bias in PINNs [49]. Interesting applications of PINNs appear in fluid dynamics [44], electromagnetism [33], elastic material deformation [38] and seismology [59]. For a more extensive review of PINNs, its applications and extended forms, we refer to the survey by Cuomo et al. [14]. Some shortcomings of PINNs are documented by Krishnapriyan et al. [35], who find that PINNs struggle to learn relevant physics in more challenging PDE regimes. However, they simultaneously present ideas to address these issues.

PINNs were created with physical application in mind. Thanks to their flexibility, the main building block can be kept the same across many physical problems with smaller adjustments to the architecture. Implementation of the framework has also become more approachable since the release of dedicated software packages such as DeepXDE [43], NVIDIA Modulus (previously SimNet) [23] or NeuroDiffEq [12].

1.2 Contributions and outline

As mentioned above, the main goal of this work is a systematic comparison of physics-informed neural networks and the finite element method for the solution of partial differential equations. Indeed, we consider

- the elliptic Poisson equation in one, two, and three space-dimensions,
- the parabolic Allen-Cahn equation in one space-dimension, and
- the hyperbolic semilinear Schrödinger equation in one and two space dimensions.

We choose these model problems to cover a large range of classes of partial differential equations. We compare PINNs and FEM in terms of solution time, evaluation time, and accuracy. We employ different finite element bases, as well as a multitude of network architectures. Several improvements over the vanilla PINNs have been discussed in the literature. As those are either still fundamentally based on the same idea or rather represent a mix of a classical approach and PINNs, we focus on vanilla PINNs in the following.

This work is structured as follows. We discuss the PDEs, FEM, and PINNs in Section 2. We introduce our method of comparison in Section 3, before presenting our computational results regarding Poisson, Allen–Cahn, and semilinear Schrödinger equation in Sections 4, 5, and 6, respectively. We discuss our results and conclude the work in Section 7.

2 Mathematical Background

In the following, we first study a very general class of partial differential equations that we formally denote by

$$\mathcal{A}u(x,t) = f(x,t) \quad x \in \Omega,\ t \in [0,T]. \tag{1}$$

Here, the function $u : \overline{\Omega} \times [0,T] \to \mathbb{R}^n$ denotes the solution of the PDE, where $\Omega \subset \mathbb{R}^d$ is open, bounded, and connected and usually represents a spatial domain, whereas $[0,T]$ is the time interval. $\mathcal{A}$ denotes the differential operator acting on u that can also be nonlinear and $f : \Omega \times [0,T] \to \mathbb{R}^n$ is a source term. We denote the corresponding boundary conditions and the initial condition by

$$\mathcal{B}u(x,t) = g(x,t), \quad x \in \partial\Omega,\ t \in [0,T] \tag{2}$$
$$u(x,0) = h(x), \quad x \in \Omega, \tag{3}$$

respectively.

Throughout this work, we consider three different partial differential equations: the *Poisson equation*, the *Allen–Cahn equation*, and the *semilinear Schrödinger equation*. We introduce those PDEs in the following.

Poisson equation. The Poisson equation is a linear elliptic PDE of the form

$$\Delta u(x) = f(x), \qquad x \in \Omega,$$

where $\Delta = \sum_{i=1}^{d} \frac{\partial}{\partial x_i}$ denotes the Laplacian and $f : \overline{\Omega} \to \mathbb{R}$ is a source term. To be well-defined, we need to equip it with boundary conditions, say, of *Dirichlet*-type:

$$u(x) = u_\partial(x), \qquad x \in \partial\Omega,$$

Neumann-type:

$$\partial_{\vec{n}} u(x) = u_\partial(x) \qquad x \in \partial\Omega,$$

or a combination of the two. In each case, we employ the function $u_\partial : \partial\Omega \to \mathbb{R}$ to determine the boundary behaviour. There are several results about the existence of strong and weak solutions of elliptic PDEs, see, e.g. Theorem 3 in Chapter 6.2 in [18]. The Poisson equation models, for instance, the stationary heat distribution in a homogeneous object Ω; here, f describes heat sources and sinks.

Allen–Cahn equation. The Allen–Cahn equation is a nonlinear parabolic PDE of the form

$$\frac{\partial u(x,t)}{\partial t} = \varepsilon \Delta u(x,t) - \frac{2}{\varepsilon} u(t,x)(1-u(t,x))(1-2u(t,x)), \qquad x \in \Omega, t \in [0,T],$$

where $\varepsilon > 0$ and which, of course, is considered together with appropriate boundary conditions and an initial condition. The PDE is semilinear, since the nonlinearity depends on u but not on a second derivative of u. If ε is sufficiently small, the solution of the Allen–Cahn equation approximately partitions the domain Ω into patches where $u \approx 0$ and $u \approx 1$; in-between those patches – at the so-called diffuse interface – it is smooth. These patches can represent binary alloys [1] or, e.g., different segments in an image with pixels in Ω [6]. Allen–Cahn is also a relaxation of the mean-curvature flow to which it converges as $\varepsilon \downarrow 0$ [20].

Semilinear Schrödinger equation. The semilinear Schrödinger equation is a complex-valued nonlinear hyperbolic PDE of the form

$$\mathrm{i}\frac{\partial h(x,t)}{\partial t} = -\Delta h(x,t) - h(x,t)|h(x,t)|^2, \qquad x \in \Omega, t \in [0,T],$$

again subject to appropriate initial and boundary conditions. We have represented the semilinear Schrödinger equation above as it is usual in the literature. This way of writing might be confusing. Thus, we present the PDE again splitting real and imaginary parts. We set $h(x,t) =: u_R(x,t) + \mathrm{i}u_I(x,t)$ and write

$$\begin{aligned}
\frac{\partial u_R(x,t)}{\partial t} &= -\Delta u_I(x,t) - (u_R(x,t)^2 + u_I(x,t)^2) u_I(x,t), \\
\frac{\partial u_I(x,t)}{\partial t} &= \Delta u_R(x,t) + (u_R(x,t)^2 + u_I(x,t)^2) u_R(x,t), \qquad x \in \Omega, t \in [0,T].
\end{aligned}$$

We refer to [61] for an application of the semilinear Schrödinger equation in non-linear optics.

2.1 Finite Element Method

As mentioned before, the finite element method has been the gold standard for the spatial discretisation of a huge class of partial differential equations. We now discuss the basics of the finite element method given the example of an elliptic PDE with Dirichlet boundaries $u_\partial = 0$ and square intregable $f \in \mathcal{L}^2(\Omega)$. We mention non-stationary PDEs and other boundary conditions further below.

When solving an elliptic PDE with the finite element method, we aim to find a weak solution. That means, we try to find u in an appropriate function space U such that for all functions v in an appropriate V, we have

$$\int_\Omega v(x)(\Delta u(x) - f(x))\mathrm{d}x = 0.$$

Applying integration by parts, we obtain the usual weak formulation of the Poisson equation:

$$\int_\Omega \langle \nabla v(x), \nabla u(x) \rangle + v(x) f(x) \mathrm{d}x = 0. \tag{4}$$

An appropriate choice of function spaces U, V in this case is $U = V = H_0^1(\Omega)$, the Sobolev space of square-integrable functions $\Omega \to \mathbb{R}$ that have a weak derivative that is also square-integrable. In finite elements, we now replace $H_0^1(\Omega)$ by a finite-dimensional space H with basis $(\varphi_i)_{i=1}^N$. On this finite dimensional space, we can write the weak equation as

$$-\int_\Omega \langle \nabla \varphi_i(x), \nabla \sum_{j=1}^n a_j \varphi_j(x) \rangle \mathrm{d}x = \int_\Omega \varphi_i(x) f(x) \mathrm{d}x \qquad i = 1, ..., N,$$

which can be rewritten as a usual linear system $Ba = b$, with

$$B = \left(-\int_\Omega \langle \nabla \varphi_i(x), \nabla \varphi_j(x) \rangle \mathrm{d}x \right)_{i,j=1}^N, \qquad b = \left(\int_\Omega \varphi_i(x) f(x) \mathrm{d}x \right)_{i=1}^N.$$

Now, of course, $u(\cdot) \approx \sum_{j=1}^n a_j \varphi_j(\cdot)$. While this approach allows for a large range of bases $(\varphi_i)_{i=1}^N$, we usually speak only of finite elements when choosing specific locally supported, piecewise polynomial functions. As usual differential operators are local, this will usually lead to B having a favourable, sparse structure.

In the discussion above, we consider homogeneous Dirichlet boundary conditions $u_\partial = 0$. Inhomogeneous boundary conditions, in which u_∂ is not constantly 0, can be achieved by first finding a function that satisfies the boundary conditions and then solving an auxiliary homogeneous problem with $u_\partial = 0$. Neumann conditions can be enforced by adding to the bilinear form $\int_{\partial\Omega} v u_\partial \mathrm{d}x$ and choosing functions in $U = V = H^1(\Omega)$ the space of square-integrable functions with square-integrable weak derivatives. Mixed boundary conditions can be enforced in a similar way.

The time-domain of a non-stationary PDE can also be discretised with finite elements, see, e.g., [26]. However, the probably more usual approach is to use finite elements in space and usual ODE solvers in time. To represent Allen–Cahn and semilinear Schrödinger at the same time, we consider some semilinear PDE of the form

$$\frac{\partial u(x,t)}{\partial t} = \Delta u(x,t) + F(u(x,t)), \quad u(x,0) = u_0(x) \qquad (x \in \Omega, t \in [0,T]),$$

which is also subject to boundary conditions. Due to the stiffness of the heat equation, we are required to use implicit techniques, such as the implicit Euler method. In a semi-discrete form, we can write the update step as

$$u_{t+1} = u_t + \delta t \Delta u_{t+1} + \delta t F(u_{t+1}) \qquad (t = 0, 1, \ldots),$$

where $\delta t > 0$ denotes the size of the time steps. In the weak formulation, we obtain

$$\int_\Omega v u_{t+1} \mathrm{d}x = \int_\Omega v u_t - \delta t \langle \nabla v, \nabla u_{t+1} \rangle + \delta t v F(u_{t+1}) \mathrm{d}x \qquad (v \in V, t = 0, 1, \ldots)$$

and can now again use a finite element discretisation in space to obtain an approximation of u_{t+1} represented through a finite-dimensional equation. In the case of Allen–Cahn and semilinear Schrödinger, this equation is non-linear and requires us to repeatedly employ a Newton's method.

We can prevent the additional cost of a Newton-solve and usually still obtain a stable discretisation, by employing a semi-implicit strategy: linear parts of the PDE are solved with an implicit discretisation, non-linear parts are discretised explicitly. In our setting, we obtain

$$u_{t+1} = u_t + \delta t \Delta u_{t+1} + \delta t F(u_t) \qquad (t = 0, 1, \ldots)$$

which requires us to solve the following weak problem

$$\int_\Omega v u_{t+1} + \delta t \langle \nabla v, \nabla u_{t+1} \rangle \mathrm{d}x = \int_\Omega v u_t + \delta t v F(u_t) \mathrm{d}x \qquad (v \in V, t = 0, 1, \ldots)$$

that is fully linear.

For more details on the finite element method, we refer to, e.g., the book by Braess [9] or the classical references Courant [13] and Hrennikoff [25]. ODE integrators, as well as evolution equations, are thoroughly considered by Iserles [27]. Semi-implicit schemes appear, for instance, in the work by Bertozzi and Schönlieb [7].

2.2 Physics-informed Neural Networks

The aim of physics-informed neural networks is to approximate PDE solutions using a deep neural network. They make use of the powerful tool that is automatic differentiation and therefore do not rely on discretisation of the space-time domain but rather on random sampling of the domain.

Let us consider a PDE of the form (1) with suitable boundary and initial conditions (2). The vanilla PINNs approach, as introduced by Raissi et al. [52], uses a fully connected neural network with N_l hidden layers and $N_e(l)$ neurons per layer l. Inputs of the network are the PDE variables (x, t) sampled in the domain $\Omega \times [0, T]$. The neural network acts as the function approximator $u_\theta(x, t)$ of the PDE solution, with θ the network weights that are optimised during training. The PDE is integrated as a soft constraint in the optimisation. That is, the network is trained with the PDE residual, as well as boundary and initial condition residuals as the loss functional:

$$\begin{aligned} Loss(\theta) =& \frac{1}{N_f}\sum_{i=1}^{N_f} \|\mathcal{A}u_\theta(x_f^i, t_f^i) - f(x_f^i, t_f^i))\|^2 \\ &+ \frac{1}{N_g}\sum_{j=1}^{N_g} \|\mathcal{B}u_\theta(x_g^j, t_g^j) - g(x_g^j, t_g^j)\|^2 + \frac{1}{N_h}\sum_{k=1}^{N_h} \|u_\theta(x_h^k, 0) - h(x_h^k)\|^2. \end{aligned}$$

Here, N_f is the number of collocation points $(x_f^i, t_f^i) \in \Omega \times [0, T]$ for $i = 1, \ldots, N_f$ sampled for the PDE residual in the loss. Similarly, $(x_g^j, t_g^j) \in \partial\Omega \times [0, T]$ for $j = 1, \ldots, N_g$ denote the training points on the boundary and $x_k^h \in \Omega$ for $k = 1, \ldots, N_h$ the training data for the initial condition. Additionally, we need data $g(x_g^j, t_g^j)$ for the boundary and $h(x_h^k)$ initial conditions. This can be measured data and does not need to be represented as an analytic function. PINNs is therefore able to incorporate measurements and leverage data-driven information. We follow [52] in using Latin Hybercube Sampling [60], a quasi-random approach for space filling sampling, to obtain the collocation points for training. In our experiments, we re-sample the collocation points in every epoch to get a better coverage of the sampling domain; see also [29]. As mentioned above, the differential operator $\mathcal{A}$ and any derivatives in the boundary condition are evaluated using automatic differentiation [4]. In contrast to numerical methods such as finite differences, automatic differentiation uses the chain rule to backpropagate through the network and evaluate the derivative. It is therefore not dependent on a grid or mesh that discretises the domain. In turn, the sampling method becomes more important. Automatic differentiation gives rise to a significant advantage of PINNs towards other classical numerical methods for solving PDEs, that is, it does not deal with prohibitively small step-sizes and scales well in higher dimensions. In PINNs, the design of the neural network architecture, i.e., the type of network structure, the number of hidden layers and the number of nodes per layer, is flexible and can be adjusted based on the PDE complexity. In the following, we will use fully connected feed-forward dense neural network. We denote the size and numbers of nodes per layers as $[N_e(1), N_e(2), ..., N_e(l)]$. That is, $[5, 5, 1]$, for example, represents a neural network with 2 hidden layers and 5 nodes per layer. The last entry 1 represents the size of the output layer. Lastly, the loss function is typically minimised using first the Adam optimiser [32] for a coarser optimisation and then the second-order quasi-Newton optimiser L-BFGS [41].

3 Method of comparison

In this section, we give an overview of the experimental design and the measures that we consider to compare FEM and PINNs. The main features that we investigate are the time to solve the PDEs and the accuracy of

the results. We therefore ask the questions of which methodology is computationally faster, more accurate, and most efficient. We investigate different types of PDEs with varying complexity and dimensionality to cover a broad spectrum of PDEs and examine if computational speed and accuracy of the two approaches change based on the PDE type.

3.1 Experimental Setup

We now discuss the experimental setup of our computational study. We first discuss ground truth solutions, and then describe the setup for FEM and PINNs and the metrics with which compare them.

Indeed, for a systematic comparison of FEM and PINNs, we need a ground truth solution of the PDEs to compare the solution approximations to and evaluate their accuracy. The first step is therefore to determine these ground truth solutions. For the Poisson equations, we have analytical solutions that we can use to determine the accuracy of the approximation approaches. However, neither the Allen-Cahn equation nor the semilinear Schrödinger equation have analytical solutions. Instead, we use FEM on a very fine mesh to compute a very accurate reference solution that we will use as the ground truth; time-stepping is performed using the implicit Euler method. One of the advantages of FEM is that we have extensive theoretical foundations for the convergence for large classes of partial differential equations, including the Allen–Cahn equation, e.g., [21] and the semilinear Schrödinger equation, e.g. [56]. Thus, a finely meshed finite element solution can therefore guarantee accuracy up to a small error.

Let us now discuss the details and specifications for FEM that we consider in the comparison. As mentioned before, finer meshes will lead to higher accuracy albeit slower computation time. We therefore solve the PDEs for different mesh sizes to see the relationship between computation time and accuracy based on the FEM design. For time-dependent PDEs, that are, the Allen-Cahn and the semilinear Schrödinger equations, we choose a semi-implicit strategy for discretisation as detailed in Subsection 2.1. All FEM solution approximations are implemented using the python toolbox FEniCS [2, 3]. In the case of PINNs, we closely follow the vanilla approach as described in Raissi et al. [52]. That is, we design the loss functional as in Subsection 2.2 and use the Adam optimiser [32] in the first instance and L-BFGS [41] to refine the optimisation. The learning rate is heuristically chosen for optimal results in each PDE. All derivatives in the loss functional are computed using automatic differentiation. The collocation points that make up the input to the neural network are newly sampled for each epoch using Latin Hybercube sampling [60] in the Adam optimisation. This way, we are able to cover more of the sampling space and improve generalisability of the trained network. L-BFGS does not allow for re-sampling between iterations and we therefore only sample the collocation points once before the optimisation again using Latin Hybercube sampling. PINNs allow for a flexible design of the neural network architecture based on the underlying PDE. We therefore run the network training on architectures of different sizes, that is, with varying numbers of layers and nodes. The code for PINN training and evaluation was written with the python neural network library jax [8] and is available on github: `https://github.com/TamaraGrossmann/FEM-vs-PINNs`.

We consider three main features for comparison. That is, we evaluate the FEM and PINNs approaches based on their solution time, evaluation time, and accuracy. The solution time refers to the time it takes for each method to approximate the general solution. We differentiate between solution and evaluation time due to the way FEM and PINNs approximate solutions differently. FEM, typically, solve the PDE on a fixed mesh of interest but it is possible to interpolate that solution to a different mesh. On the other hand, for PINNs a neural network is trained on a set of collocation points. The trained network can then be evaluated on any point in the domain. While training time of PINNs can be rather slow, one of the advantages of neural networks is that once trained the evaluation on new input data tends to be very fast. We therefore consider both times. For FEM this means solution time refers to solving the weak form of the PDE on a fixed mesh. FEM is run on a CPU. Considering PINNs, the solution time refers to the training time of the neural network which is run on a GPU. In turn, the evaluation time in FEM is measured as the time to interpolate the solution on a different mesh. In PINNs, the evaluation time is the time to evaluate the trained neural network on a new set of collocation points. We measure the accuracy of both methods on the same mesh in comparison to the ground truth solutions. For the Allen-Cahn and Schrödinger equations the evaluation mesh is chosen as the fine mesh on which the ground truth solution was derived. The measure for accuracy is the ℓ^2 relative

error. All experiments were run on the same machine that has 12 CPU cores and we used a Quadro P6000 GPU for the neural network training. The codes were run 10 times and the reported solution and evaluation times are the average over the 10 recorded times.

4 Approximating the Poisson equation

Let us first investigate the Poisson equation in one, two, and three space-dimensions. Each of the equations we consider has an analytical solution that we can use to evaluate the accuracy of the FEM and PINN approximation. Comparing the same type of PDE in different dimensions also allows us to draw conclusions about cost and accuracy effects due to the dimensionality of the PDE.

4.1 1D

For the one-dimensional case, we are examining the Poisson equation with a right hand side $f(x)$ as detailed below on a unit interval and employ Dirichlet boundary conditions:

$$\begin{aligned} \Delta u(x) &= (4x^3 - 6x)\exp(-x^2) \quad x \in (0,1) \\ u(0) &= 0 \\ u(1) &= \exp(-1). \end{aligned} \tag{5}$$

This PDE has an analytical solution that can be written as:

$$u_{\text{true}}(x) = x\exp(-x^2) \qquad (x \in [0,1]).$$

A visualisation of the solution to the 1D Poisson equation is shown in Figure 1a.

FEM

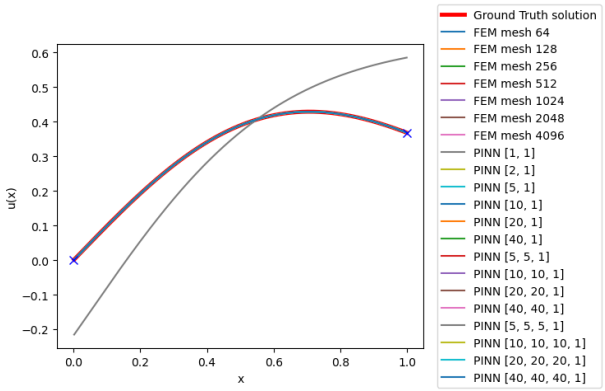
As introduced in Section 2.1, the first step in solving PDEs with the finite element method is deriving the weak formulation of the PDE, which we have done for Poisson equation alright right there. Here, of course $f(x) = (4x^3 - 6x)\exp(-x^2)$. Next, we need to define the finite element mesh. We choose a regular mesh on the domain $[0,1]$ with varying numbers of cells $n \in \{64, 128, 256, 512, 1024, 2048, 4096\}$. Of course, a higher number of cells corresponds to a finer grid on which the PDE is solved and subsequently leads to a more accurate, but also computationally more costly solution. The finite elements we are using are standard linear Lagrange elements, that is, piecewise linear hat functions (P_1). Subsequently, we solve the variational problem in (4) with the Dirichlet boundary conditions specified in (5) using the conjugate gradient method with incomplete LU factorisation as a preconditioner. All specifications are implemented using the python toolbox FEniCS [2, 3] to compute an approximate PDE solution.

PINNs

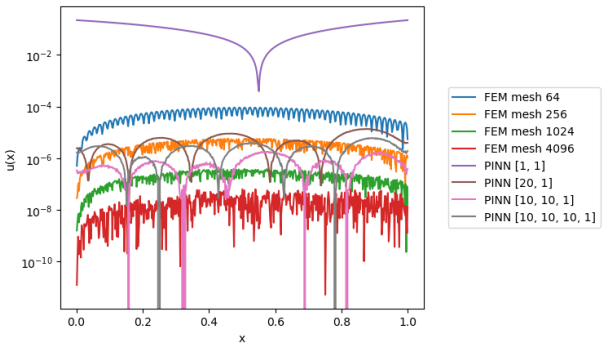
For solving the 1D Poisson equation using PINNs, there are three design parameters that we need to specify before training. The first step is choosing a loss functional. Following the vanilla PINNs approach, we evaluation the goodness of the solution using the discretised ℓ^2-energy or mean squared error over the PDE, boundary and initial conditions. In particular, we define the loss function as:

$$\text{Loss}(\theta) = \frac{1}{N}\sum_{i=1}^{N} \|\Delta u_\theta(x_i) - (4x_i^3 - 6x_i)\exp(-x_i^2)\|_2^2 + \|u_\theta(0)\|_2^2 + \|u_\theta(1) - exp(-1)\|_2^2,$$

with u_θ the neural network, θ the trained weights and $N = 256$ the number of collocation points x_i sampled in each epoch using latin hybercube sampling. The second design parameter is the neural network architecture, that is, the type of neural network, the activation function, and the number of hidden layers and nodes. For the 1D Poisson case, we train feed-forward dense neural networks with $\tanh$ as the activation function. We compare the results on architectures of different sizes. The network architectures we consider for 1D Poisson are $[1,1], [2,1], [5,1], [10,1], [20,1], [40,1], [5,5,1], [10,10,1],$

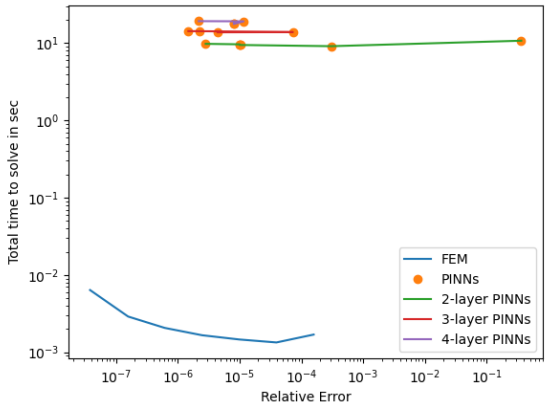


(a) Plot of the PDE ground truth solution and the FEM and PINNs solution approximations.

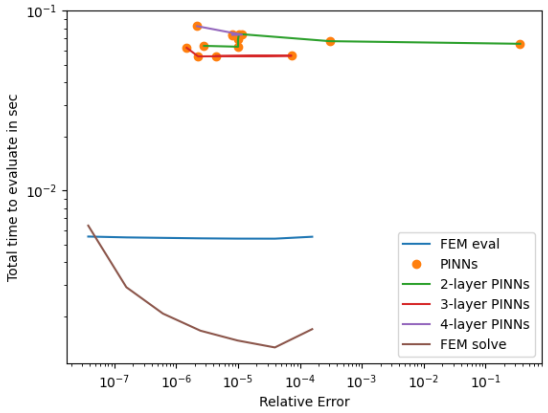


(b) Plot of the difference between some of the FEM / PINNs solution approximations to the ground truth solution on a log scale.

Figure 1: Plot for 1D Poisson equation solution.



(a) Plot of time to solve FEM and train PINN in sec versus ℓ^2 relative error.



(b) Plot of time to interpolate FEM and evaluate PINN in sec versus ℓ^2 relative error.

Figure 2: Plot for 1D Poisson equation of time in sec versus ℓ^2 relative error.

$[20, 20, 1]$, $[40, 40, 1]$, $[5, 5, 5, 1]$, $[10, 10, 10, 1]$, $[20, 20, 20, 1]$, and $[40, 40, 40, 1]$. The loss function is minimised for each network architecture using the Adam optimiser for $15,000$ epochs with a learning rate of $1e-4$ in the first instance. Additionally, we refine the optimisation using L-BFGS.

Results

The resulting PDE solution approximations of the 1D Poisson equation using FEM and PINNs are compared to the analytical solution on a mesh in $[0, 1]$ with 512 mesh points. The ground truth solution of the 1D Poisson equation in (5) and its approximations are displayed in Figure 1a. Likewise, the difference from the approximate solutions to the GT solution are shown in Figure 1b for all mesh sizes of FEM and architectures in PINNs. One of the architectures in the PINNs approximation renders results with large relative error: the PINN with a single hidden layer and one node is not even able to learn the solution in a way in which the boundary conditions are satisfied. However, all other solution approximations differ from the ground truth only marginally.

Let us compare the time it takes to solve or rather approximate the PDE using FEM and PINNs to the relative error produced on a new set of grid points. For FEM the solution time is the time to solve the linear systems

and for PINNs we consider the time to train the neural network. The results are shown in Figure 2a. We can clearly show that overall FEM are faster and more accurate in their solution approximation. While there are some PINN architectures that are able to achieve similar or even lower relative errors than some of the FEM approximations, their training time is 2-3 orders of magnitude higher than in FEM. Considering the evaluation time, that is, the time to interpolate the FEM solution on a different mesh and evaluate the trained PINN on a test set, we can see a similar relationship; see Figure 2b. FEM solution approximations are overall faster and more accurate. Finally, we consider the relationship between the number of layers and the solution time and relative error. We observe that the time to train a PINN is similar relative to the number of layers. However, the accuracy of each network is related to the number of nodes in the layers. That is, we cannot show that networks with more layers generally achieve better results in 1D Poisson.

4.2 2D

Let us now investigate the two-dimensional Poisson equations defined as:

$$\begin{aligned} \Delta u(x,y) &= 2(x^4(3y-2) + x^3(4-6y) + x^2(6y^3 - 12y^2 + 9y - 2) \\ &\quad - 6x(y-1)^2 y + (y-1)^2 y) \qquad (x,y) \in (0,1)^2 \end{aligned} \tag{6}$$

$$\begin{aligned} \partial_{\vec{n}} u(0,y) &= 0 \quad y \in [0,1] \\ \partial_{\vec{n}} u(1,y) &= 0 \quad y \in [0,1] \\ u(x,0) &= 0 \quad x \in [0,1] \\ \partial_{\vec{n}} u(x,1) &= 0 \quad x \in [0,1] \end{aligned} \tag{7}$$

We can solve the PDE (6) with mixed boundary conditions analytically:

$$u_{\text{true}}(x,y) = x^2(x-1)^2 y(y-1)^2.$$

The solution is displayed in Figure 3.

FEM

For the 2D Poisson equation, we again refer to the weak formulation of the Poisson equation in (4) with $f(x,y) = 2(x^4(3y-2)+x^3(4-6y)+x^2(6y^3-12y^2+9y-2)-6x(y-1)^2y+(y-1)^2y)$. The finite element mesh is defined on the unit square $[0,1] \times [0,1]$; it contains $n \times n$ squares wih $n \in \{100, 200, \ldots, 1000\}$. Each square on the mesh is divided into two triangles on which we define again piecewise linear finite elements (P_1). We solve the variational problem (4) with mixed boundary conditions given in (7) using the conjugate gradient method with incomplete LU factorisation preconditioning. The method is implemented using FEniCS.

PINNs

The loss functional for the PINN approximation is the ℓ^2-residual of the PDE (6) and its boundary conditions. For u_θ the neural network with weights θ that are to be trained, the loss reads:

$$\begin{aligned} &\text{Loss}(\theta) \\ &= \frac{1}{N_f} \sum_{i=1}^{N_f} \| \Delta u_\theta(x_f^i, y_f^i) - 2((x_f^i)^4(3y_f^i - 2) + (x_f^i)^3(4 - 6y_f^i) + (x_f^i)^2(6(y_f^i)^3 - 12(y_f^i)^2 + 9y_f^i - 2) \|_2^2 \\ &\quad + \frac{1}{N_g} \sum_{j=1}^{N_g} \left(\left\| \partial_{\vec{n}} u_\theta(0, y_g^j) \right\|_2^2 + \left\| \partial_{\vec{n}} u_\theta(1, y_g^j) \right\|_2^2 + \left\| u_\theta(x_g^j, 0) \right\|_2^2 + \left\| \partial_{\vec{n}} u_\theta(x_g^j, 1) \right\|_2^2 \right). \end{aligned}$$

The collocation points are sampled at every epoch using Latin Hybercube sampling with $N_f = 2000$ and $N_g = 250$. We train a feed-forward dense neural network with $\tanh$ activation function. We consider 11 different architectures for training, these are [20,1], [60,1], [20,20,1], [60,60,1], [20,20,20,1], [60,60,60,1], [20,20,20,20,1], [60,60,60,60,1], [20,20,20,20,20,1], [60,60,60,60,60,1], and [120,120,120,120,120,1]. Just like in 1D Poisson, we use the Adam optimiser for $20,000$ epochs with a learning rate of $1e-3$ to train the network. Subsequently, we refine the optimisation using L-BFGS.

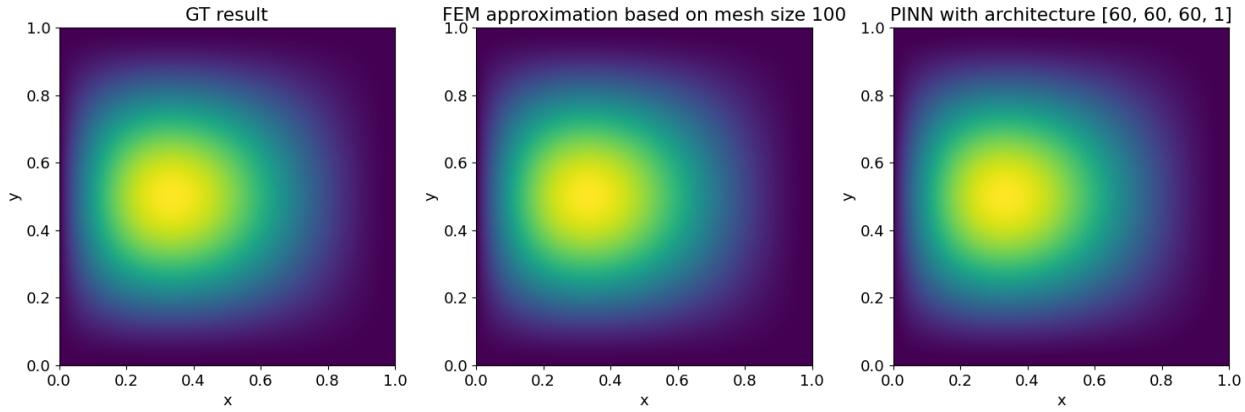
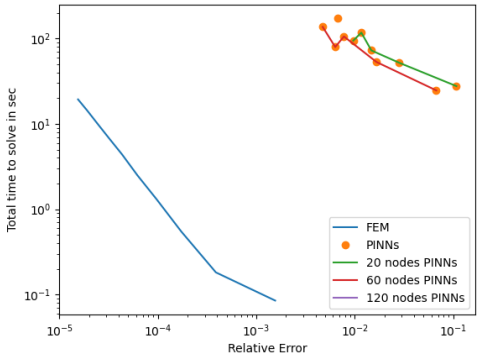
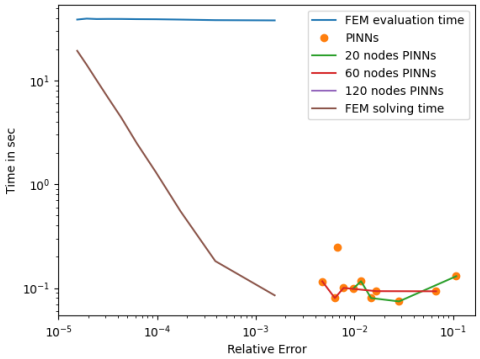


Figure 3: Comparison of the 2D Poisson ground truth solution to examples of the FEM and PINN approximations.



(a) Plot of time to solve FEM and train PINN in sec versus ℓ^2 relative error.



(b) Plot of time to interpolate FEM and evaluate PINN in sec versus relative error. For comparison, the time to solve FEM is also plotted.

Figure 4: Plot for 2D Poisson equation of time in sec versus ℓ^2 relative error.

Results

An example of the resulting solution approximations on a mesh with 2000×2000 cells next to the ground truth solution is show in Figure 3. The time versus accuracy plots for the 2D Poisson equation are displayed in Figure 4a and Figure 4b. Considering the solution time, FEM clearly outperforms all PINN approximations both in accuracy and computation time. Using FEM to obtain the PDE solution is faster by 1-3 orders of magnitude. However, when we look at the evaluation times in Figure 4b, the relationship changes. That is, the time to evaluate a PINN is 2-3 orders of magnitude faster then the interpolation of FEM on a new mesh. We additionally plotted the FEM solution time in the same plot, and the PINN evaluation remains faster, however, by a smaller margin. Interestingly, the solution time of FEM is faster than the FEM evaluation time; this is possibly due to an inefficient interpolation code. Even though the evaluation of the trained neural network gives an improvement in time, the PINN approximations remain to have lower accuracy.

4.3 3D

For the three-dimensional Poisson equation, we choose a PDE on the unit cube with Dirichlet boundary conditions, as follows:

$$\begin{aligned} \Delta u(x,y,z) &= -3\pi^2 \sin(\pi x)\sin(\pi y)\sin(\pi z) \quad (x,y,z) \in (0,1)^3 \\ u(x,y,z) &= 0 \quad (x,y,z) \in \partial(0,1)^3 \end{aligned}$$

The analytical solution of this 3D Poisson equation is displayed in Figure 5 and can be written as:

$$u_{\text{true}}(x,y,z) = \sin(\pi x)\sin(\pi y)\sin(\pi z).$$

FEM

The weak formulation of the 3D Poisson equation is again given in (4) with $f(x,y,z) = -3\pi^2 \sin(\pi x)\sin(\pi y)\sin(\pi z)$. The finite element mesh is defined on a unit cube $[0,1] \times [0,1] \times [0,1]$ consisting of $n \times n \times n$ cubes with $n \in \{16, 32, 64, 128\}$. We subdivide each cube into tetrahedrals and use again piecewise linear finite elements (P_1). The weak problem is solved using the conjugate gradient method and incomplete LU factorisation as the preconditioner.

PINNs

Similar to the one- and two-dimensional Poission cases, we design the loss functional as the PDE and boundary condition residual. That is, we define the loss as follows:

$$\begin{aligned} \text{Loss}(\theta) = &\frac{1}{N_f}\sum_{i=1}^{N_f} \|\Delta u_\theta(x_f^i, y_f^i, z_f^i) + 3\pi^2 \sin(\pi x_f^i)\sin(\pi y_f^i)\sin(\pi z_f^i)\|_2^2 \\ &+ \frac{1}{N_g}\sum_{j=1}^{N_g} \left(\|u_\theta(0, y_g^j, z_g^j)\|_2^2 + \|u_\theta(x_g^j, 0, z_g^j)\|_2^2 + \|u_\theta(x_g^j, y_g^j, 0)\|_2^2 \right) \\ &+ \frac{1}{N_g}\sum_{j=1}^{N_g} \left(\|u_\theta(1, y_g^j, z_g^j)\|_2^2 + \|u_\theta(x_g^j, 1, z_g^j)\|_2^2 + \|u_\theta(x_g^j, y_g^j, 1)\|_2^2 \right). \end{aligned}$$

Here, u_θ denotes the neural network with θ the training parameters. We sample $N_f = 1000$ collocation points in the domain and $N_g = 100$ on the boundary. The neural network is again a feed-forward dense neural network with $\tanh$ activation function. We consider 8 different architectures to train with the following layer and node specifications: [20,20,1], [60,60,1], [20,20,20,1], [60,60,60,1], [20,20,20,20,1], [60,60,60,60,1], [20,20,20,20,20,1], and [60,60,60,60,60,1]. The Adam optimiser with learning rate $1e-3$ is used for 20,000 epochs before employing the L-BFGS optimiser.

Results

The PDE is evaluated on a mesh of $150 \times 150 \times 150$ grid points and example results are shown in Figure 5. The time versus accuracy plots can be found in Figure 6. As expected, the FEM results with slower computation times have lower relative errors as they are solved on finer meshes as shown in Figure 6a. While the PINN approximations are about 1-3 orders of magnitude slower in training time depending on the FEM mesh solution that we compare to, they are able to achieve equal and even higher accuracy in most cases. On the other hand, PINNs outperform FEM when considering the evaluation time as plotted in Figure 6b. The time to interpolate FEM on a new mesh is 2-3 orders of magnitude slower than the evaluation time of PINNs. Additionally, PINNs are able to achieve equal or higher accuracy scores. If we also take the FEM solving time into account, we find again that this is faster than the evaluation time. However, it is slower than the PINNs evaluation. Therefore, a trained PINN has a lower computation time for the evaluation and a similar to lower relative error as compared to both the FEM solution and evaluation times.

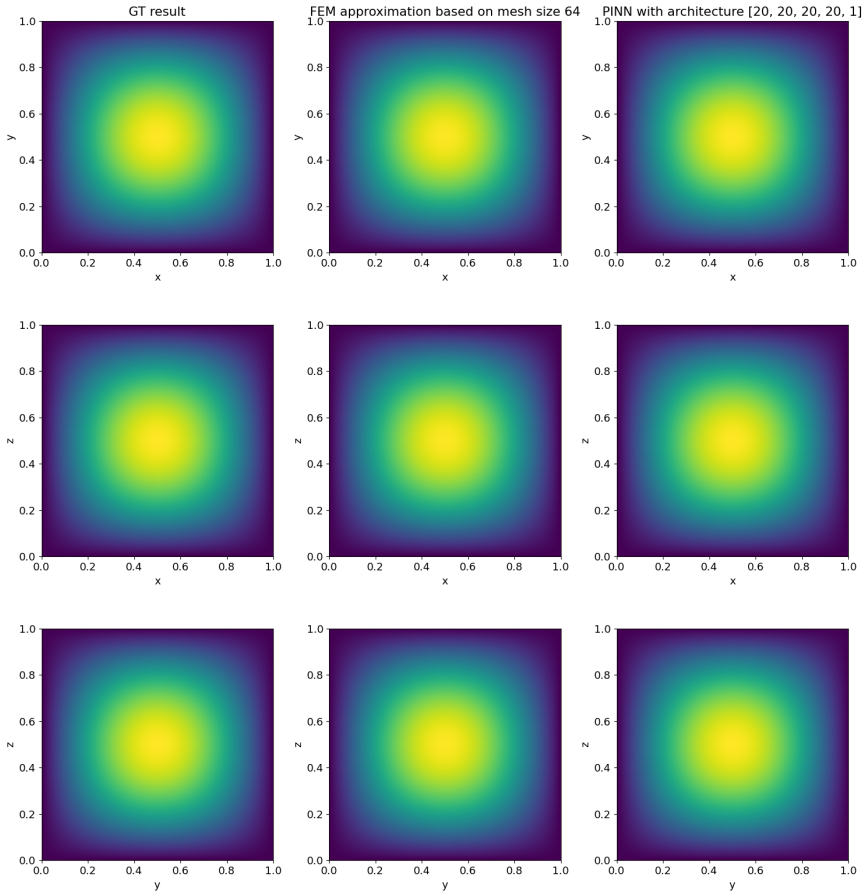


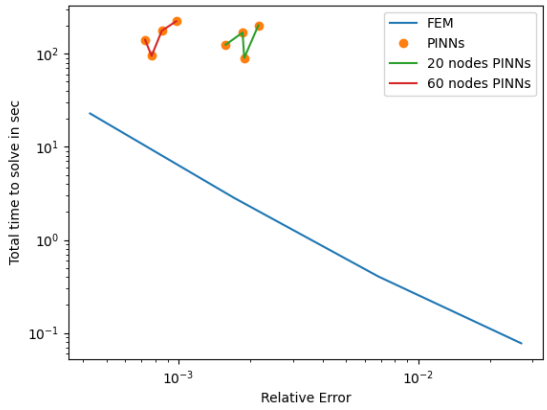
Figure 5: Comparison of the 3D Poisson ground truth solution slices at x,y,z = 0.5 respectively to examples of the FEM and PINN approximations.

5 Approximating the Allen-Cahn equation

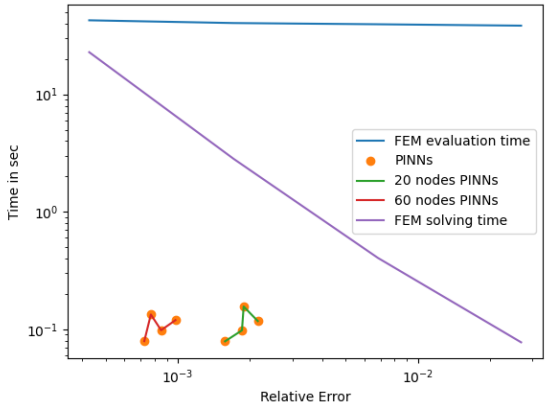
To study the one dimensional Allen-Cahn equation, we consider the following PDE:

$$\begin{aligned} \frac{\partial u(t,x)}{\partial t} &= \epsilon \Delta u - \frac{2}{\epsilon} u(t,x)(1-u(t,x))(1-2u(t,x)), \quad x \in \Omega = [0,1],\ t \in [0,T] \\ u(t,0) &= u(t,1), \quad t \in [0,T] \\ u(0,x) &= \frac{1}{2}\left(\frac{1}{2}\sin(x2\pi) + \frac{1}{2}\sin(x16\pi)\right) + \frac{1}{2}, \quad x \in \Omega, \end{aligned} \tag{8}$$

where $T = 0.05$ and $\epsilon = 0.01$. As mentioned in Section 2, we note that a smaller ϵ will yields close to piecewise constant solutions, whereas solutions with large ϵ will be overall more smooth. The ϵ chosen in this case is due to the inability of PINNs to approximate the Allen-Cahn solution with a smaller ϵ. We have trained the neural network to solve Allen-Cahn with $\epsilon = 0.001$ for various network architectures and with activation functions such as softplus that are typically able to handle discontinuous solutions. However, the PINNs were not able to recover results close to the ground truth solution. We will discuss this case further below. For now, the results refer to $\epsilon = 0.01$.



(a) Plot of time to solve FEM and train PINN in sec versus ℓ^2 relative error.



(b) Plot of time to interpolate FEM and evaluate PINN in sec versus relative error. For comparison, the time to solve FEM is also plotted.

Figure 6: Plot for 3D Poisson equation of time in sec versus ℓ^2 relative error.

FEM

As opposed to the Poisson equation, the Allen-Cahn equation has a time dependency that we need to consider in the discretisation. As mentioned in Section 2.1, we use a semi-implicit Euler strategy to approximate the solution. The weak formulation in a semi-discrete form of the 1D Allen-Cahn equation with Dirichlet $u_\partial = 0$ boundaries can be written as

$$\begin{aligned}\int_0^1 (u_{t+1}(x) - u_t(x))v(x)\mathrm{d}x + \epsilon \int_0^1 \langle \nabla u_{t+1}(x), \nabla v(x)\rangle \mathrm{d}x \\ + \frac{2}{\epsilon}\int_0^1 u_t(x)(1-u_t(x))(1-2u_t(x))v(x)\mathrm{d}x = 0,\end{aligned} \tag{9}$$

for all test functions $v \in H_0^1([0,1])$. The time is discretised with distance of $dt = 1e-3$. For the finite element mesh in space, we choose an interval mesh on the domain $[0,1]$ with varying numbers of cells $n \in \{32, 128, 512, 2048\}$. As in 1D Poisson, we employ piecewise-linear finite element functions (P_1). The non-linear variational problem (9) with periodic boundary conditions is solved using a Newton solver. The FEM approach was implemented using the python toolbox FEniCS.

PINNs

We follow the general PINNs approach and design the loss based on the PDE residual, the boundary residual and the initial condition residual. Additionally, we introduce a weighting of the different terms in the loss functional. We found heuristically this weighting to render the best results.

$$\begin{aligned}\mathrm{Loss}(\theta) = &\frac{1}{N_f}\sum_{i=1}^{N_f} \| \frac{\partial u_\theta(t_f^i, x_f^i)}{\partial t} - \epsilon \Delta u_\theta(t_f^i, x_f^i) + \frac{2}{\epsilon} u_\theta(t_f^i, x_f^i)(1-u_\theta(t_f^i, x_f^i))(1-2u_\theta(t_f^i, x_f^i))\|_2^2 \\ &+ \frac{1}{N_g}\sum_{k=1}^{N_g} \|u_\theta(t_g^j, 0) - u_\theta(t_g^j, 1)\|_2^2 \\ &+ \frac{1000}{N_h}\sum_{k=1}^{N_h} \|u_\theta(0, x_h^k) - \frac{1}{2}\left(\frac{1}{2}\sin(2\pi x_h^k) + \frac{1}{2}\sin(16\pi x_h^k)\right) + \frac{1}{2}\|_2^2,\end{aligned} \tag{10}$$

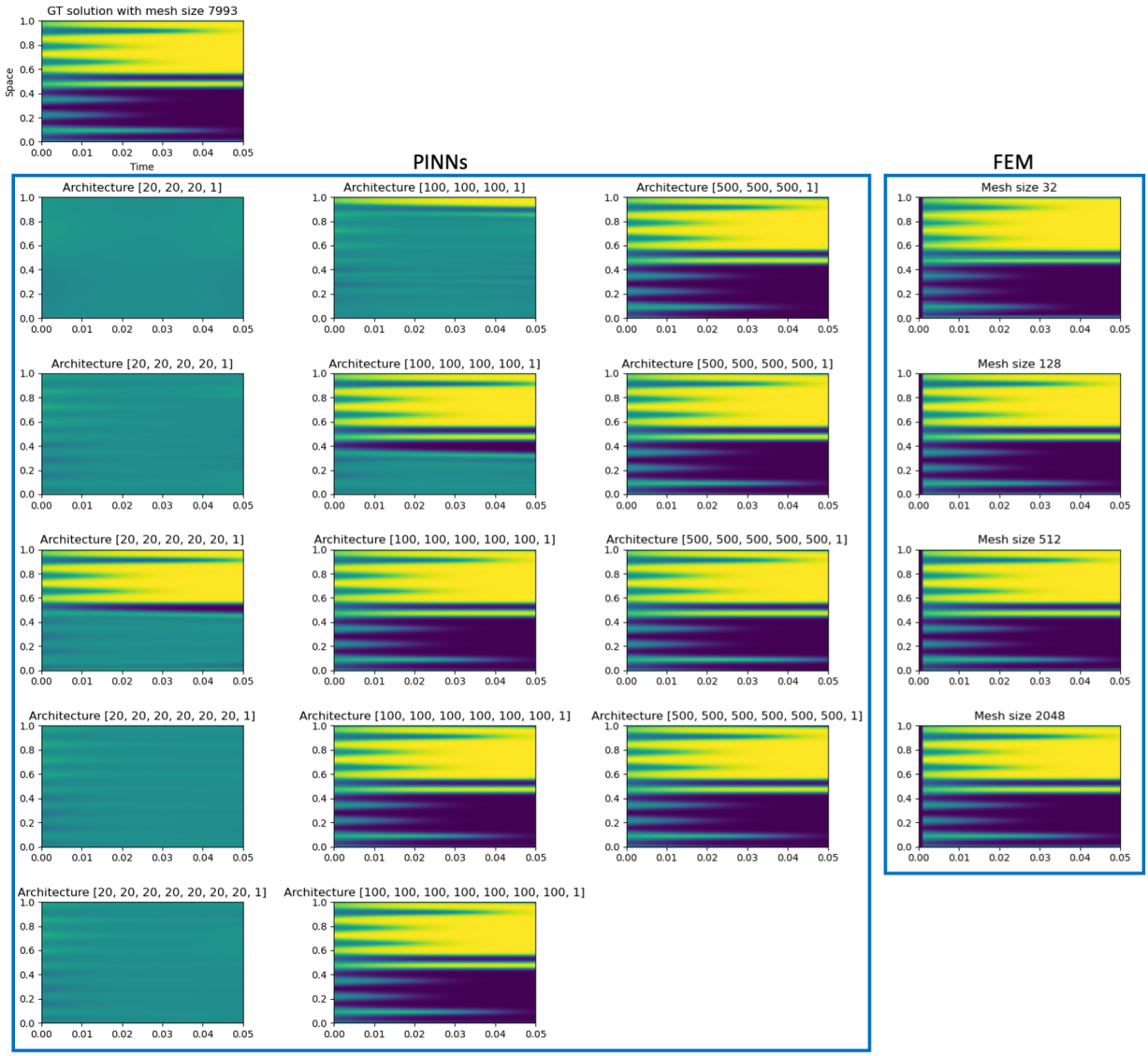
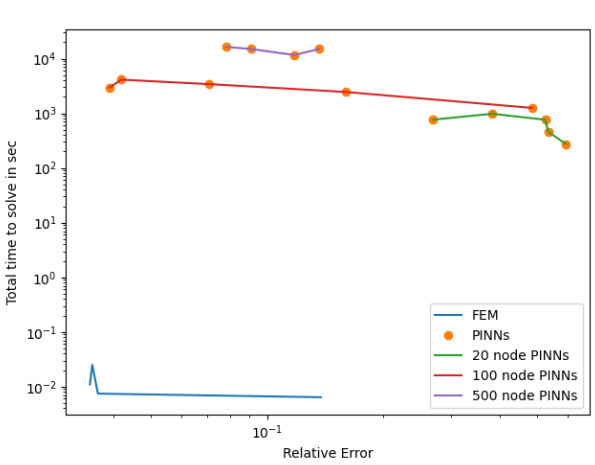


Figure 7: Comparison of the 1D Allen-Cahn solution approximated by FEM and PINNs.

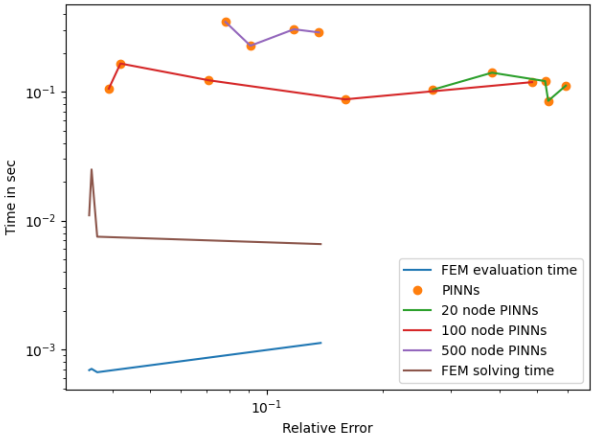
with u_θ the neural network and θ the trained weights. We train the network on $N_f = 20,000$ collocation points $(t_f^i, x_f^i) \in [0, 0.05] \times [0, 1]$ that are sampled using Latin Hybercubes. The training points for the boundary with $N_g = 250$ and for the initial condition with $N_h = 500$ are also sampled in each epoch with Latin Hybercube sampling. The network architecture is a feed-forward dense neural network with a $\tanh$ activation functional. We consider architectures of the following 14 different sizes: [20,20,20,1], [100,100,100,1], [500,500,500,1], [20,20,20,20,1], [100,100,100,100,1], [500,500,500,500,1], [20,20,20,20,20,1], [100,100,100,100,100,1], [500,500,500,500,500,1], [20,20,20,20,20,20,1], [100,100,100,100,100,100,1], [500,500,500,500,500,500,1], [20,20,20,20,20,20,20,1] and [100,100,100,100,100,100,100,1]. For a network of size [500,500,500,500,500,500,500,1] we run out of memory on the GPU available to us. This constitutes the limitation of our training. For the optimisation we first run the Adam optimiser with learning rate $1e-4$ for 7000 epochs over the initial loss alone. We have found the network struggling to learn the initial condition when the optimisation is run on the full loss directly. After the 7000 epochs optimising the initial loss, we run the Adam optimiser for $50,000$ epochs on the full loss function (10). Finally, we refine the optimisation using L-BFGS.

Results

The PDE approximations with FEM and PINNs are compared on the mesh of the ground truth solution spanning $[0, 1]$ with 7993 mesh points and at a time discretisation of $\frac{1}{3}e - 4$. The fine-meshed FEM approximation for the ground truth solution was derived using implicit Euler. The FEM and the PINN approximations compared to the ground truth solution are displayed in Figure 7 for the different mesh sizes and network architectures. FEM is able to recover the solution of the PDE at all mesh sizes. However, closer inspection of the result for a mesh of size 32 shows slight errors along the diffusive interface. On the other hand, the ability of a PINN to approximate the PDE solution well is dependent on the architecture and the number of free parameters or weights that are to be determined. While all architectures with 20 nodes (cf. Figure 7 column 1) are not able to recover the solution whatsoever, networks with 100 nodes per layer are able to be trained for the solution. We can clearly observe a progression based on the number of layers. Finally, neural networks that have 500 nodes per layer are all able to approximate the solution well.



(a) Plot of time to solve FEM and train PINN in sec versus ℓ^2 relative error.



(b) Plot of time to interpolate FEM and evaluate PINN in sec versus ℓ^2 relative error. For comparison, the time to solve FEM is plotted.

Figure 8: Plot for 1D Allen-Cahn equation of time in sec versus ℓ^2 relative error.

Let us compare the solution and evaluation time versus the accuracy for the FEM and PINN approximations as shown in Figure 8. For the solution time, that is the time to solve the PDE using FEM and the time to train the neural network in PINN, FEM is 5-6 orders of magnitude faster than PINNs. This is mainly due to the size of the neural networks. Here, the complexity of the PDE requires larger architectures to be able to capture the solution. Some of the PINN architectures are then able though to achieve similar relative errors to FEM. This is especially true for networks with 100 or 500 nodes per layer as also seen in Figure 7. The evaluation time of the PINN is plotted against the evaluation time of FEM and the solution time of FEM in Figure 8b. The calculation time advantage of FEM evaluation drops to about one order of magnitude as compared to the solution time. While it is to be expected that the evalutation of a neural network is much faster then the training, FEM is still faster in both solution and evaluation time.

Compared to the other PDEs that we are considering in this study, the Allen-Cahn equation needs slight modification, i.e. weighting of the loss and pre-training on only part of the loss functional. This is due to the difficulty we have found the network to have in learning the PDE solution. In addition, we should note that we have also attempted to train a PINN for Allen-Cahn (8) with $\epsilon = 0.001$. The resulting solution – shown in Figure 9 – becomes close to binary after a certain amount of time. We had discussed this effect in Section 2. This renders very large gradients or discontinuities in the solution. We trained PINNs with different activation functions such as softplus or ReLU that are typically able to handle discontinuities. However, all results were insufficient to be considered an approximation. This speaks to the assumption that PINNs in the vanilla form are not well equipped to handle discontinuous solutions; this may also be due to PINNs solving the strong PDE, rather than a weak form. Variations of the vanilla PINNs approach might

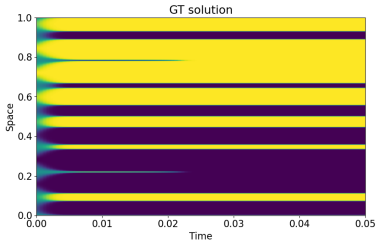


Figure 9: FEM solution of the 1D Allen-Cahn equation for $\epsilon = 0.001$ derived on a mesh with 7993 cells.

be able to obtain satisfactory approximations. However, as we are only considering the vanilla approach, this goes beyond the scope of the paper. We make a note that FEM is able to approximate Allen-Cahn with $\epsilon = 0.001$ albeit we should anticipate the use of a finer mesh to accurate represent the diffuse interface.

6 Approximating the Semilinear Schrödinger equation

Finally, we investigate the semilinear Schrödinger equation in one and two space-dimensions. The specifications of the one-dimensional case are taken from the original PINNs paper of Raissi et al. [52]. Note that the semilinear Schrödinger equation has a complex-valued solution; further increasing the difficulty of its approximation.

6.1 1D

In the one-dimensional case, we consider the semilinear Schrödinger equation with periodic boundary conditions, as follows:

$$\begin{aligned}
\mathrm{i}\frac{\partial h(t,x)}{\partial t} &= -0.5\Delta h(t,x) - |h(t,x)|^2 h(t,x) && x \in [-5,5],\ t \in [0,\pi/2],\\
h(0,x) &= 2\mathrm{sech}(x), && x \in [-5,5],\\
h(t,-5) &= h(t,5), && t \in [0,\pi/2],\\
\frac{\partial h(t,-5)}{\partial x} &= \frac{\partial h(t,5)}{\partial x}, && t \in [0,\pi/2].
\end{aligned}$$

We note that the identical problem had also been considered by [52]. As $h(t,x)$ is a complex valued function, the semilinear Schrödinger equation is solved for $h(t,x) =: u_R(t,x) + \mathrm{i}u_I(t,x)$, with $u_R(t,x)$ the real part and $u_I(t,x)$ the imaginary part. Example results are visualised for $|h(t,x)| = \sqrt{u_R^2(t,x) + u_I^2(t,x)}$ in Figure 10.

FEM

Similar to the 1D Allen-Cahn equation that we considered, we use a semi-implicit Euler strategy to approximate the 1D Schrödinger equation in time. We can then write the weak formulation for the real and imaginary parts of the PDE, assuming Dirichlet $u_\partial = 0$ boundaries, separately as

$$\int_{-5}^{5}(u_{t+1}^I(x) - u_t^I(x))v^R(x)\mathrm{d}x - 0.5\int_{-5}^{5}\langle\nabla u_{t+1}^R(x), \nabla v^R(x)\rangle\mathrm{d}x - |h_t(x)|^2\int_{-5}^{5}u_{t+1}^R(x)v^R(x)\mathrm{d}x = 0,$$
$$\int_{-5}^{5}(u_{t+1}^R(x) - u_t^R(x))v^I(x)\mathrm{d}x + 0.5\int_{-5}^{5}\langle\nabla u_{t+1}^I(x), \nabla v^I(x)\rangle\mathrm{d}x + |h_t(x)|^2\int_{-5}^{5}u_{t+1}^I(x)v^I(x)\mathrm{d}x = 0,$$

for all test functions $v^R, v^I \in H_0^1(\Omega)$. The time is discretised with $dt = 1e-3$ and we define the finite elements on an interval mesh in $[-5,5]$. We consider meshes with $n \in \{32, 128, 512, 2048\}$ cells. We again employ piecewise linear finite element basis functions on those cells (P_1) and use the generalised minimal residual method (gmres) for a solver.

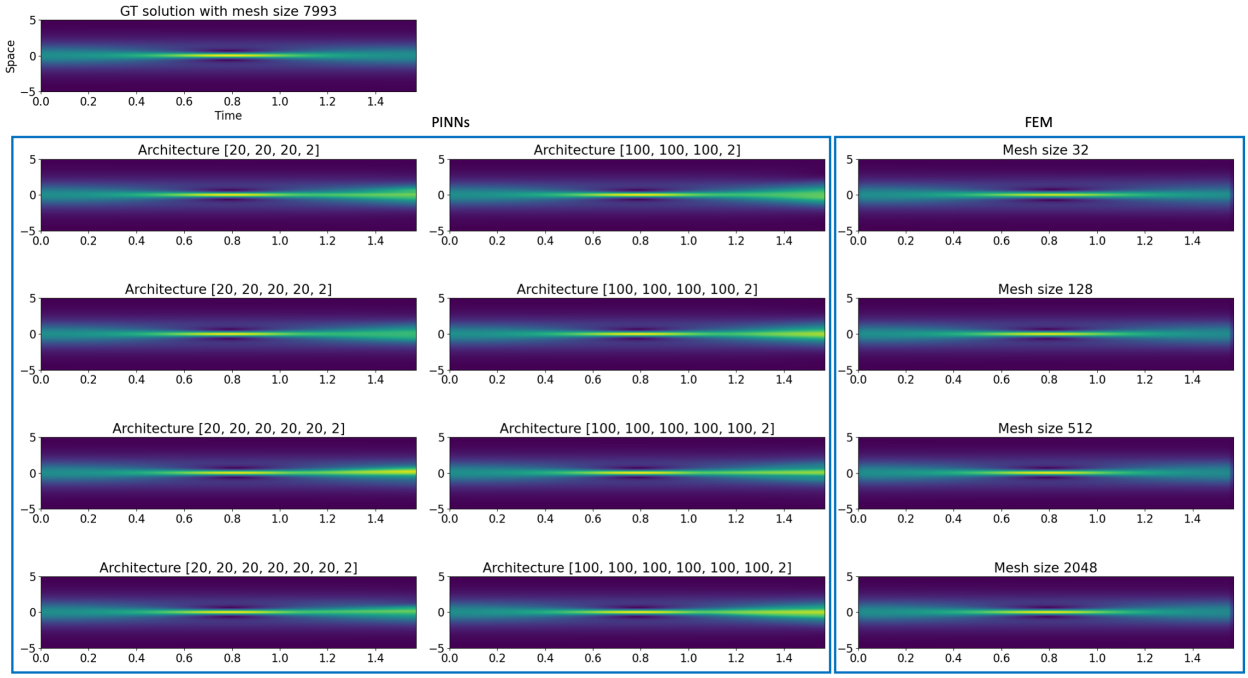


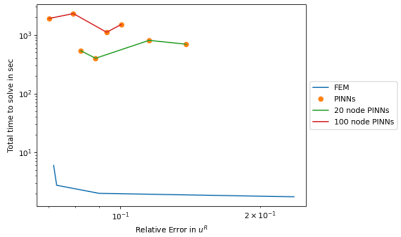
Figure 10: Comparison of the 1D semilinear Schrödinger solution $|h(t,x)| = \sqrt{u_R^2(t,x) + u_I^2(t,x)}$ approximated by FEM and PINNs of different mesh and architecture sizes.

PINNs

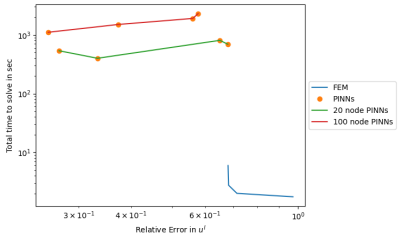
Let us now define the loss functional used in the neural network approximation of the PDE solution. Again, we consider the residual of the PDE, inital condition and boundary conditions as follows:

$$
\begin{aligned}
\text{Loss}(\theta) = \frac{1}{N_f}\sum_{i=1}^{N_f}(\| &\frac{\partial u_\theta^I(t_f^i, x_f^i)}{\partial t} - \epsilon \Delta u_\theta^R(t_f^i, x_f^i) - |h_\theta(t_f^i, x_f^i)|^2 u_\theta^R(t_f^i, x_f^i)\|_2^2 \\
&+ \| \frac{\partial u_\theta^R(t_f^i, x_f^i)}{\partial t} + \epsilon \Delta u_\theta^I(t_f^i, x_f^i) + |h_\theta(t_f^i, x_f^i)|^2 u_\theta^I(t_f^i, x_f^i)\|_2^2) \\
&+ \frac{1}{N_g}\sum_{k=1}^{N_g} \left(\|u_\theta^R(t_g^j, -5) - u_\theta^R(t_g^j, 5)\|_2^2 + \|u_\theta^I(t_g^j, -5) - u_\theta^I(t_g^j, 5)\|_2^2 \right) \\
&+ \frac{1}{N_g}\sum_{k=1}^{N_g} \left(\| \frac{\partial u_\theta^R(t_g^j, -5)}{\partial x} - \frac{\partial u_\theta^R(t_g^j, 5)}{\partial x} \|_2^2 + \| \frac{\partial u_\theta^I(t_g^j, -5)}{\partial x} - \frac{\partial u_\theta^I(t_g^j, 5)}{\partial x} \|_2^2 \right) \\
&+ \frac{1}{N_h}\sum_{k=1}^{N_h} \left(\|u_\theta^R(0, x_h^k) - 2\text{sech}(x_h^k)\|_2^2 + \|u_\theta^I(0, x_h^k)\|_2^2 \right),
\end{aligned}
$$

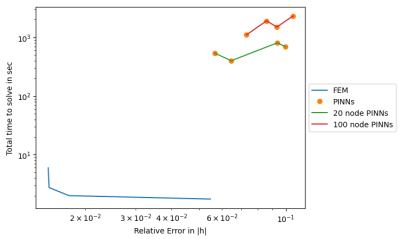
where $h_\theta(t,x) = [u_\theta^R(t,x), u_\theta^I(t,x)]$ the neural network the produces the real and imaginary parts of the PDE solution of the network output with weights θ. The network is trained on $N_f = 20,000$ collocation points $(t_f^i, x_f^i) \in [0, \pi/2] \times [-5, 5]$ and with $N_g = 50$ point on the boundary and $N_h = 50$ points for the initial condition using Latin Hybercube sampling. The network architecture is a feed-forward dense neural network with $\tanh$ activation function. In these specifications, we have followed the original vanilla PINNs paper [52]. We investigate the performance of 8 network architectures, that are: [20,20,20,2], [100,100,100,2], [20,20,20,20,2], [100,100,100,100,2], [20,20,20,20,20,2], [100,100,100,100,100,2], [20,20,20,20,20,20,2],



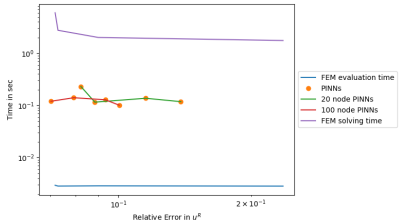
(a) Solution time versus accuracy for the real part of the PDE u^R.



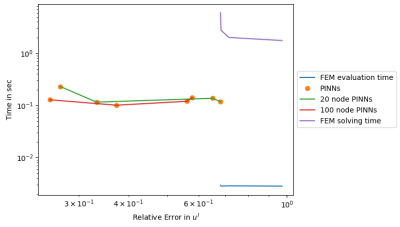
(b) Solution time versus accuracy for the imaginary part of the PDE u^I.



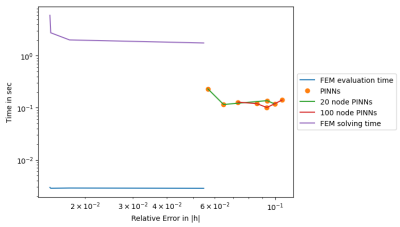
(c) Solution time versus accuracy for the modulus $|h|$.



(d) Evaluation time versus accuracy for the real part of the PDE u^R. For comparison, the time to solve FEM is plotted.



(e) Evaluation time versus accuracy for the imaginary part of the PDE u^I. For comparison, the time to solve FEM is plotted.



(f) Evaluation time versus accuracy for the modulus $|h|$. For comparison, the time to solve FEM is plotted.

Figure 11: Plot for 1D Schrödinger equation of time in sec versus ℓ^2 relative error. Plots are split for the real and imaginary parts of the PDE as well as $|h| = \sqrt{u_R^2 + u_I^2}$.

and [100,100,100,100,100,100,2]. We employ the Adam optimiser for $50,000$ epochs and a learning rate of $1e-4$. Afterwards, we use the L-BFGS optimiser to refine the training results.

Results

The results of the FEM and PINNs approximation are compared on the mesh of the ground truth solution that has a size of 7993 cells and a time discretisation with $dt = \frac{1}{3}e-4$. The resulting approximations for $|h(t,x)| = \sqrt{u_R^2(t,x) + u_I^2(t,x)}$ are displayed in Figure 10. Already visually, we notice that the PINN approximations become slightly less accurate for larger time instance than the FEM approximations. Quantitatively speaking, the time versus accuracy plots give a broader overview of the performance of each method. The plots are shown in Figure 11. Focusing on the modulus $|h|$, FEM both has a lower solution time by 2 orders of magnitude and a lower relative error than any neural network approximation as shown in Figure 11c. Considering the evaluation time for both methods alone, FEM continues to outperform PINNs in time and accuracy. However, if we compare the FEM solving time to the PINNs evaluation time, neural networks are able to evaluate solutions faster by one order of magnitude. Nonetheless, FEM remains to produce results with higher accuracy for $|h|$. The results are inconclusive for u_R and u_I: the FEM solution is considerably faster but less or hardly more accurate than the PINNs solution as shown in Figures 11d and 11e.

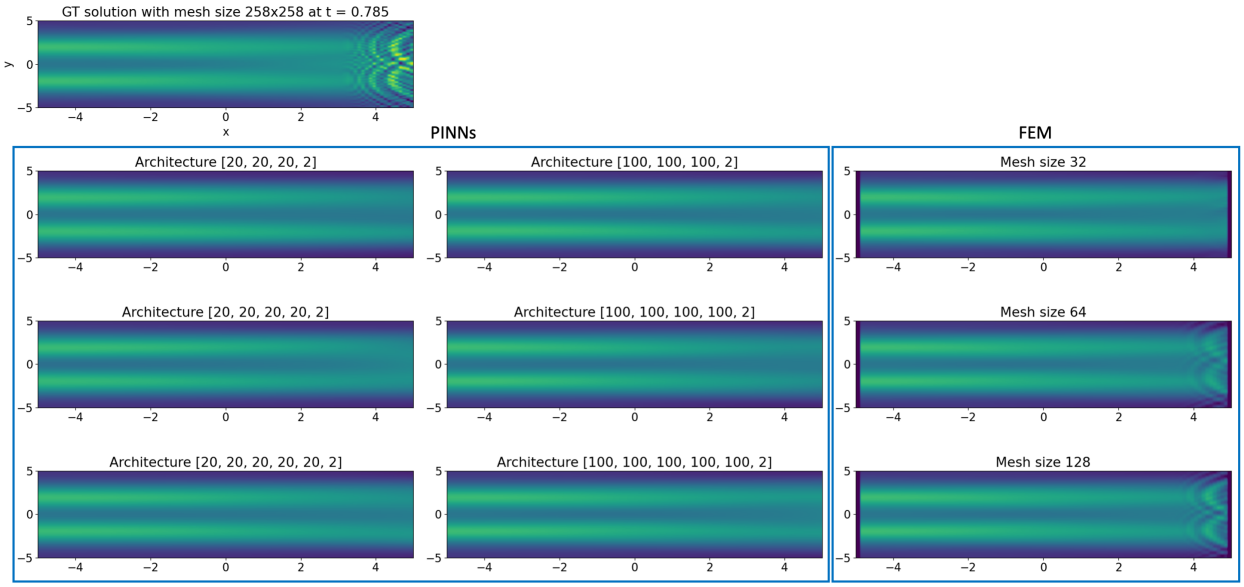


Figure 12: Comparison of the 2D semilinear Schrödinger solution $|h(t,x)| = \sqrt{u_R^2(t,x) + u_I^2(t,x)}$ at time $t = \pi/4$ approximated by FEM and PINNs of different mesh and architecture sizes.

6.2 2D

Let us finally move to the two-dimensional semilinear Schrödinger equation. We consider periodic boundary conditions and define the initial condition as follows:

$$
\begin{aligned}
\mathrm{i}\frac{\partial h(t,x,y)}{\partial t} &= -0.5\Delta h(t,x,y) - |h(t,x,y)|^2 h(t,x,y), && x,y \in [-5,5],\ t \in [0,\pi/2] \\
h(0,x,y) &= \mathrm{sech}(x) + 0.5\mathrm{sech}(y-2) + 0.5\mathrm{sech}(y+2), && x,y \in [-5,5] \\
h(t,-5,y) &= h(t,5,y), && t \in [0,\pi/2],\ y \in [-5,5] \\
h(t,x,-5) &= h(t,x,5), && t \in [0,\pi/2],\ x \in [-5,5] \\
\frac{\partial h(t,-5,y)}{\partial x} &= \frac{\partial h(t,5,y)}{\partial x}, && t \in [0,\pi/2],\ y \in [-5,5] \\
\frac{\partial h(t,x,-5)}{\partial y} &= \frac{\partial h(t,x,5)}{\partial y} && t \in [0,\pi/2],\ x \in [-5,5].
\end{aligned}
$$

The PDE is complex-valued and we approximate the solution for $h(t,x,y) =: u_R(t,x,y) + \mathrm{i}u_I(t,x,y)$. The approximate solutions for the modulus for $|h(t,x,y)| = \sqrt{u_R^2(t,x,y) + u_I^2(t,x,y)}$ are shown in Figure 12.

FEM

Similar to the one-dimensional case of the Schrödinger equation, we split the weak formulation of the PDE (again for Dirichlet $u_\partial = 0$ boundaries) into its real and imaginary parts:

$$\begin{aligned}
&\int_{-5}^{5}\int_{-5}^{5}(u_{t+1}^I(x,y) - u_t^I(x,y))v^R(x,y)\mathrm{d}x\mathrm{d}y - 0.5\int_{-5}^{5}\int_{-5}^{5}\langle\nabla u_{t+1}^R(x,y), \nabla v^R(x,y)\rangle\mathrm{d}x\mathrm{d}y \\
&\qquad - |h_t(x,y)|^2\int_{-5}^{5}\int_{-5}^{5}u_{t+1}^R(x,y)v^R(x,y)\mathrm{d}x\mathrm{d}y = 0, \\
&\int_{-5}^{5}\int_{-5}^{5}(u_{t+1}^R(x,y) - u_t^R(x,y))v^I(x,y)\mathrm{d}x\mathrm{d}y + 0.5\int_{-5}^{5}\int_{-5}^{5}\langle\nabla u_{t+1}^I(x,y), \nabla v^I(x,y)\rangle\mathrm{d}x\mathrm{d}y \\
&\qquad + |h_t(x,y)|^2\int_{-5}^{5}\int_{-5}^{5}u_{t+1}^I(x,y)v^I(x)\mathrm{d}x\mathrm{d}y = 0,
\end{aligned}$$

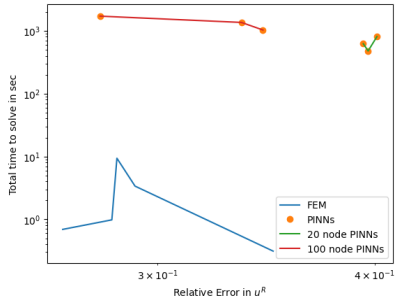
for all test functions $v^R, v^I \in H_0^\Omega$. Time is discretised with $dt = 1e-3$ and the finite elements are defined on a rectangular mesh in the domain $[-5,5]\times[-5,5]$. We consider meshes with $\{(16,16),(32,32),(40,40),(64,64),(128,128)\}$ squares, each again being split into two triangles on which we define piecewise-linear finite element basis functions (P_1). We again use gmres to solve the linear system.

PINNs

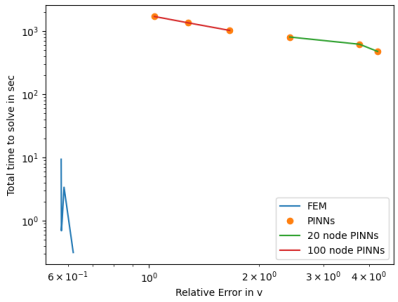
The loss functional for the neural network approximations are defined for both the real and imaginary parts of the PDE:

$$\begin{aligned}
\mathrm{Loss}(\theta) = &\frac{1}{N_f}\sum_{i=1}^{N_f}(\|\frac{\partial u_\theta^I(t_f^i,x_f^i,y_f^i)}{\partial t} - \epsilon\Delta u_\theta^R(t_f^i,x_f^i,y_f^i) - |h_\theta(t_f^i,x_f^i,y_f^i)|^2 u_\theta^R(t_f^i,x_f^i,y_f^i)\|_2^2 \\
&\quad + \|\frac{\partial u_\theta^R(t_f^i,x_f^i,y_f^i)}{\partial t} + \epsilon\Delta u_\theta^I(t_f^i,x_f^i,y_f^i) + |h_\theta(t_f^i,x_f^i,y_f^i)|^2 u_\theta^I(t_f^i,x_f^i,y_f^i)\|_2^2) \\
&+ \frac{1}{N_g}\sum_{k=1}^{N_g}(\|u_\theta^R(t_g^j,-5,y_g^j) - u_\theta^R(t_g^j,5,y_g^j)\|_2^2 + \|u_\theta^I(t_g^j,-5,y_g^j) - u_\theta^I(t_g^j,5,y_g^j)\|_2^2 \\
&\quad + \|u_\theta^R(t_g^j,x_g^j,-5) - u_\theta^R(t_g^j,x_g^j,5)\|_2^2 + \|u_\theta^I(t_g^j,x_g^j,-5) - u_\theta^I(t_g^j,x_g^j,5)\|_2^2) \\
&+ \frac{1}{N_g}\sum_{k=1}^{N_g}\left(\|\frac{\partial u_\theta^R(t_g^j,-5,y_g^j)}{\partial x} - \frac{\partial u_\theta^R(t_g^j,5,y_g^j)}{\partial x}\|_2^2 + \|\frac{\partial u_\theta^I(t_g^j,-5,y_g^j)}{\partial x} - \frac{\partial u_\theta^I(t_g^j,5,y_g^j)}{\partial x}\|_2^2\right) \\
&+ \frac{1}{N_g}\sum_{k=1}^{N_g}\left(\|\frac{\partial u_\theta^R(t_g^j,x_g^j,-5)}{\partial y} - \frac{\partial u_\theta^R(t_g^j,x_g^j,5)}{\partial y}\|_2^2 + \|\frac{\partial u_\theta^I(t_g^j,x_g^j,-5)}{\partial y} - \frac{\partial u_\theta^I(t_g^j,x_g^j,5)}{\partial y}\|_2^2\right) \\
&+ \frac{1}{N_h}\sum_{k=1}^{N_h}\left(\|u_\theta^R(0,x_h^k,y_h^k) - \mathrm{sech}(x_h^k) - 0.5\mathrm{sech}(y_h^k-2) - 0.5\mathrm{sech}(y_h^k+2)\|_2^2\right),
\end{aligned}$$

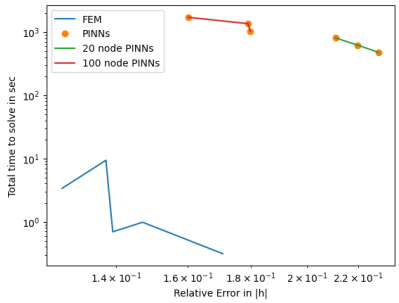
for $h_\theta(t,x,y) = u_\theta^R(t,x,y) + \mathrm{i}u_\theta^I(t,x,y)$ the neural network with an output size 2 for the real and imaginary parts and θ the network weights that are determined by training. The loss functional is optimised based on latin hybercube sampled collocation points $(t_f^i,x_f^i,y_f^i)\in[0,\pi/2]\times[-5,5]\times[-5,5]$ with $N_f = 5,000$ points for the domain, $N_g = 100$ points for the boundary and $N_h = 100$ points for the inital condition. We chose feed-forward dense neural netwrok in the architecture design with $\tanh$ the activation function. The results are compared for 6 different neural network sizes: [20,20,20,2], [100,100,100,2], [20,20,20,20,2], [100,100,100,100,2], [20,20,20,20,20,2], and [100,100,100,100,100,2]. The Adam optimiser is run for $50,000$ epochs with a learning rate of $1e-3$ before employing the L-BFGS optimiser.



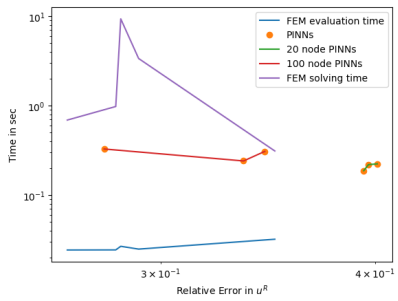
(a) Solution time versus accuracy for the real part of the PDE u^R.



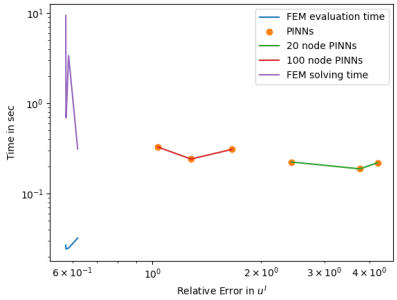
(b) Solution time versus accuracy for the imaginary part of the PDE u^I.



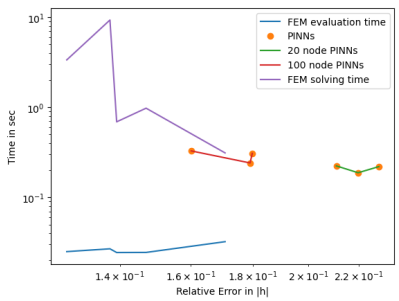
(c) Solution time versus accuracy for the modulus $|h|$.



(d) Evaluation time versus accuracy for the real part of the PDE u^R. For comparison, the time to solve FEM is plotted.



(e) Evaluation time versus accuracy for the imaginary part of the PDE u^I. For comparison, the time to solve FEM is plotted.



(f) Evaluation time versus accuracy for modulus $|h|$. For comparison, the time to solve FEM is plotted.

Figure 13: Plot for 2D Schrödinger equation of time in sec versus ℓ^2 relative error. Plots are split for the real and imaginary parts of the PDE as well as $|h| = \sqrt{u_R^2 + u_I^2}$.

Results

A first look at the visualisation of the results at an example time $t = \pi/4$ in Figure 12 shows that PINNs is having difficulties to recover the wave-type shapes at the right hand side boundary of x. While PINNs are able to reproduce the main features of the PDE solution, the examples shown lack the detailed features containing edges and discontinuities. For FEM, we observe that coarse meshes similarly fail at recovering the fine details. However, with finer meshes, FEM is able to correctly solve the PDE. Considering the quantified results on time versus accuracy in Figure 13, the difference in accuracy between PINNs and FEM become less prudent in the real part. However, for both solving time in Figures 13a-11c and evaluation time in Figures 13d-11f FEM clearly outperforms PINNs by 2-3 and 1 order of magnitude, respectively. Taking the FEM solution time into account, we can see that PINNs slightly outperforms in their evaluation time with similar relative error. Let us also note that the PINN approximation of the imaginary part of the complex-valued PDE solution is significantly less accurate compared to the FE method.

7 Discussion and Conclusions

After having investigated each of the PDEs on its own, let us know discuss and draw conclusions from the results as a whole.

Considering the solution time and accuracy, PINNs are not able to beat the finite element method in our study. Other than the inconclusive results for real and imaginary part in the Schrödinger 1D test, FEM solutions were generally faster at the same accuracy or at a higher accuracy.

After having solved the PDE, PINNs are sometimes faster at the pointwise evaluation of the respective solution: we were only able to show this in the 3D Poisson test. So when needing to evaluate a PDE on a very fine grid, one could consider solving a PINN. Although, in our examples, the solution time of FEM was so much faster that continued solving the PDE with FEM on different adapted grids would likely still be considerably faster than solving and evaluating the PINN.

We were particularly surprised that PINNs had difficulties with the Allen–Cahn equation using a small ε. This might be due to the close-to-singular behaviour at the diffuse interface and the fact that PINNs aim to solve the possibly ill-conditioned strong form of the PDE. We anticipated that PINNs would outperform FEM: the solution of an Allen–Cahn equation has very much the flavour of a classification (see [10]) at which neural networks excel; FEM requires a very fine grid to resolve the diffuse interface. A similar case in which we were surprised that PINNs did not outperform FEM was the Schrödinger 2D examples, where the PDE solution, again, contains very finely structured areas. In both these cases an adaptive PINNs approach or variational PINNs [30] might help. The latter would then also allow activation functions that have weak derivatives.

An aspect of the evaluation time that we have not considered throughout this work is the possibility of solving parameterised PDEs with neural networks, so-called operator approximators. See, for instance, Fourier Neural Operators [39] and DeepONets [42]. Whilst the finite element method requires continued solutions of PDEs when changing the parameters, neural networks can take parameters as additional inputs and be trained throughout all of them. They have been shown to work well as surrogates if the PDE needs to be solved sufficiently often; see also [62]. In a future work, those should be compared to classical methods for parameterised PDEs, such as reduced bases [48] or low-rank tensor methods [34]. Again, the time of the offline phase in which the parametric model is constructed or trained has to be considered carefully.

PINNs were good at the transition into higher dimensions: there is no increment in computational cost from the Poisson equation in 2D and 3D. This hints at the efficiency of PINNs in high-dimensional settings, in which classical techniques (such as FEM) are prohibitively expensive. This has been considered in [36]. In general, PINNs open up many interesting new research directions, especially when employed to solve such high-dimensional PDEs or when combining PDEs and data. The analysis of PINNs is both very challenging and highly interesting. Our study suggests that for certain classes of PDEs for which classical methods are applicable, PINNs are not able to outperform those.

Acknowledgments

Initial research for this work was carried out with support from the Philippa Fawcett internship programme.

TGG and CBS acknowledge the support of the Cantab Capital Institute for the Mathematics of Information and the European Union Horizon 2020 research and innovation programme under the Marie Skodowska-Curie grant agreement No. 777826 NoMADS. TGG additionally acknowledges the support of the EPSRC National Productivity and Investment Fund grant Nr. EP/S515334/1 reference 2089694. JL and CBS acknowledge support from the EPSRC grant EP/S026045/1. CBS acknowledges support from the Philip Leverhulme Prize, the Royal Society Wolfson Fellowship, the EPSRC advanced career fellowship EP/V029428/1, EPSRC grants EP/T003553/1, EP/N014588/1, EP/T017961/1, the Wellcome Trust 215733/Z/19/Z and 221633/Z/20/Z, and the Alan Turing Institute.

We also acknowledge the support of NVIDIA Corporation with the donation of two Quadro P6000 GPU used for this research.

References

[1] S. M. Allen and J. W. Cahn. Ground state structures in ordered binary alloys with second neighbor interactions. *Acta Metallurgica*, 20(3):423–433, 1972.

[2] M. Alnæs, J. Blechta, J. Hake, A. Johansson, B. Kehlet, A. Logg, C. Richardson, J. Ring, M. E. Rognes, and G. N. Wells. The FEniCS Project Version 1.5. *Archive of Numerical Software*, 3, 2015.

[3] K.-A. M. Anders Logg and G. N. Wells. *Automated Solution of Differential Equations by the Finite Element Method*. Springer, 2012.

[4] A. G. Baydin, B. A. Pearlmutter, A. A. Radul, and J. M. Siskind. Automatic differentiation in machine learning: a survey. *Journal of Marchine Learning Research*, 18:1–43, 2018.

[5] R. Bellmann. Dynamic Programming and a new Formalism in the Calculus of Variations. *Proceedings of the National Academy of Sciences*, 40(4):231–235, 1954.

[6] M. Beneš, V. Chalupecký, and K. Mikula. Geometrical image segmentation by the Allen–Cahn equation. *Applied Numerical Mathematics*, 51(2):187–205, 2004.

[7] A. Bertozzi and C.-B. Schönlieb. Unconditionally stable schemes for higher order inpainting. *Communications in Mathematical Sciences*, 9(2):413–457, 2010.

[8] 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.

[9] D. Braess. *Finite Elements: Theory, Fast Solvers, and Applications in Solid Mechanics*. Cambridge University Press, 3 edition, 2007.

[10] J. Budd, Y. van Gennip, and J. Latz. Classification and image processing with a semi-discrete scheme for fidelity forced Allen–Cahn on graphs. *GAMM-Mitteilungen*, 44(1):e202100004, 2021.

[11] M. Burger, L. Caffarelli, and P. A. Markowich. Partial differential equation models in the socioeconomic sciences. *Philos Trans A Math Phys Eng Sci*, 372(2028), Nov. 2014.

[12] F. Chen, D. Sondak, P. Protopapas, M. Mattheakis, S. Liu, D. Agarwal, and M. Di Giovanni. NeuroDiffEq: A Python package for solving differential equations with neural networks. *Journal of Open Source Software*, 5(46):1931, 2020.

[13] R. Courant. Variational methods for the solution of problems of equilibrium and vibrations. *Bulletin of the American Mathematical Society*, 49:1–23, 1943.

[14] S. Cuomo, V. S. Di Cola, F. Giampaolo, G. Rozza, M. Raissi, and F. Piccialli. Scientific Machine Learning Through Physics–Informed Neural Networks: Where we are and What's Next. *Journal of Scientific Computing*, 92(3):88, Jul 2022.

[15] A. D. Jagtap and G. Em Karniadakis. Extended Physics-Informed Neural Networks (XPINNs): A Generalized Space-Time Domain Decomposition Based Deep Learning Framework for Nonlinear Partial Differential Equations. *Communications in Computational Physics*, 28(5):2002–2041, 2020.

[16] T. De Ryck, A. D. Jagtap, and S. Mishra. Error estimates for physics-informed neural networks approximating the Navier–Stokes equations. *IMA Journal of Numerical Analysis*, Jan 2023.

[17] W. E and B. Yu. The Deep Ritz Method: A Deep Learning-Based Numerical Algorithm for Solving Variational Problems. *Communications in Mathematics and Statistics*, 6(1):1–12, 2018.

[18] L. C. Evans. *Partial differential equations*. American Mathematical Society, Providence, R.I., 2010.

[19] R. Eymard, T. Gallouët, and R. Herbin. Finite Volume Methods. In P. Ciarlet and J. Lions, editors, *Handbook of Numerical Analysis*, volume 7, pages 713–1020. Elsevier, 1997.

[20] X. Feng and A. Prohl. Numerical analysis of the Allen-Cahn equation and approximation for mean curvature flows. *Numerische Mathematik*, 94(1):33–65, 2003.

[21] X. Feng and H.-j. Wu. A Posteriori Error Estimates and an Adaptive Finite Element Method for the Allen–Cahn Equation and the Mean Curvature Flow. *Journal of Scientific Computing*, 24(2):121–146, 2005.

[22] J. Hadamard. Sur les problèmes aux dérivées partielles et leur signification physique. *Princeton University Bulletin*, pages 49–52, 1902.

[23] O. Hennigh, S. Narasimhan, M. A. Nabian, A. Subramaniam, K. Tangsali, Z. Fang, M. Rietmann, W. Byeon, and S. Choudhry. Nvidia simnet™: An ai-accelerated multi-physics simulation framework. In M. Paszynski, D. Kranzlmüller, V. V. Krzhizhanovskaya, J. J. Dongarra, and P. M. Sloot, editors, *Computational Science – ICCS 2021*, pages 447–461, Cham, 2021. Springer International Publishing.

[24] S. L. Heston. A Closed-Form Solution for Options with Stochastic Volatility with Applications to Bond and Currency Options. *The Review of Financial Studies*, 6(2):327–343, 04 2015.

[25] A. Hrennikoff. Solution of Problems of Elasticity by the Framework Method. *Journal of Applied Mechanics*, 8(4):A169–A175, 03 2021.

[26] G. M. Hulbert and T. J. Hughes. Space-time finite element methods for second-order hyperbolic equations. *Computer Methods in Applied Mechanics and Engineering*, 84(3):327–348, 1990.

[27] A. Iserles. *A First Course in the Numerical Analysis of Differential Equations*. Cambridge Texts in Applied Mathematics. Cambridge University Press, 2 edition, 2008.

[28] A. D. Jagtap, E. Kharazmi, and G. E. Karniadakis. Conservative physics-informed neural networks on discrete domains for conservation laws: Applications to forward and inverse problems. *Computer Methods in Applied Mechanics and Engineering*, 365:113028, 2020.

[29] K. Jin, J. Latz, C. Liu, and C.-B. Schönlieb. A Continuous-time Stochastic Gradient Descent Method for Continuous Data. *arXiv e-prints*, page arXiv:2112.03754, Dec. 2021.

[30] E. Kharazmi, Z. Zhang, and G. E. Karniadakis. Variational Physics-Informed Neural Networks For Solving Partial Differential Equations. *arXiv preprint arXiv:1912.00873*, 2019.

[31] E. Kharazmi, Z. Zhang, and G. E. Karniadakis. hp-VPINNs: Variational physics-informed neural networks with domain decomposition. *Computer Methods in Applied Mechanics and Engineering*, 374:113547, 2021.

[32] D. P. Kingma and J. Ba. Adam: A Method for Stochastic Optimization. In Y. Bengio and Y. LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[33] A. Kovacs, L. Exl, A. Kornell, J. Fischbacher, M. Hovorka, M. Gusenbauer, L. Breth, H. Oezelt, M. Yano, N. Sakuma, A. Kinoshita, T. Shoji, A. Kato, and T. Schrefl. Conditional physics informed neural networks. *Communications in Nonlinear Science and Numerical Simulation*, 104:106041, 2022.

[34] D. Kressner and C. Tobler. Low-Rank Tensor Krylov Subspace Methods for Parametrized Linear Systems. *SIAM Journal on Matrix Analysis and Applications*, 32(4):1288–1316, 2011.

[35] A. Krishnapriyan, A. Gholami, S. Zhe, R. Kirby, and M. W. Mahoney. Characterizing possible failure modes in physics-informed neural networks. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan, editors, *Advances in Neural Information Processing Systems*, 2021.

[36] Kunisch, Karl and Walter, Daniel. Semiglobal optimal feedback stabilization of autonomous systems via deep neural network approximation*. *ESAIM: COCV*, 27:16, 2021.

[37] H. J. Kushner. On the differential equations satisfied by conditional probablitity densities of markov processes, with applications. *Journal of the Society for Industrial and Applied Mathematics Series A Control*, 2(1):106–119, 1964.

[38] W. Li, M. Z. Bazant, and J. Zhu. A physics-guided neural network framework for elastic plates: Comparison of governing equations-based and energy-based approaches. *Computer Methods in Applied Mechanics and Engineering*, 383:113933, 2021.

[39] Z. Li, N. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart, and A. Anandkumar. Fourier neural operator for parametric partial differential equations. *arXiv preprint arXiv:2010.08895*, 2020.

[40] C. C. Lin and L. A. Segel. *Mathematics Applied to Deterministic Problems in the Natural Sciences*. Society for Industrial and Applied Mathematics, 1988.

[41] D. C. Liu and J. Nocedal. On the Limited Memory BFGS Method for Large Scale Optimization. *Mathematical Programming*, 45:503–528, 1989.

[42] L. Lu, P. Jin, G. Pang, Z. Zhang, and G. E. Karniadakis. Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators. *Nature machine intelligence*, 3(3):218–229, 2021.

[43] L. Lu, X. Meng, Z. Mao, and G. E. Karniadakis. DeepXDE: A deep learning library for solving differential equations. *SIAM Review*, 63(1):208–228, 2021.

[44] Z. Mao, A. D. Jagtap, and G. E. Karniadakis. Physics-informed neural networks for high-speed flows. *Computer Methods in Applied Mechanics and Engineering*, 360:112789, 2020.

[45] S. Mishra and R. Molinaro. Estimates on the generalization error of physics-informed neural networks for approximating a class of inverse problems for PDEs. *IMA Journal of Numerical Analysis*, 42(2):981–1022, 2022.

[46] B. Moseley, A. Markham, and T. Nissen-Meyer. Finite Basis Physics-Informed Neural Networks (FBPINNs): a scalable domain decomposition approach for solving differential equations. *arXiv preprint arXiv:2107.07871*, 2021.

[47] A. T. Patera. A spectral element method for fluid dynamics: Laminar flow in a channel expansion. *Journal of Computational Physics*, 54(3):468–488, 1984.

[48] A. Quarteroni, A. Manzoni, and F. Negri. *Reduced Basis Methods for Partial Differential Equations: An Introduction*. Springer International Publishing, Cham, 2016.

[49] N. Rahaman, A. Baratin, D. Arpit, F. Draxler, M. Lin, F. Hamprecht, Y. Bengio, and A. Courville. On the Spectral Bias of Neural Networks. In K. Chaudhuri and R. Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 5301–5310. PMLR, 09–15 Jun 2019.

[50] M. Raissi, P. Perdikaris, and G. E. Karniadakis. Physics Informed Deep Learning (Part I): Data-driven Solutions of Nonlinear Partial Differential Equations. *arXiv preprint arXiv:1711.10561*, 2017.

[51] M. Raissi, P. Perdikaris, and G. E. Karniadakis. Physics Informed Deep Learning (Part II): Data-driven Discovery of Nonlinear Partial Differential Equations. *arXiv preprint arXiv:1711.10566*, 2017.

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

[53] H. Risken. *The Fokker-Planck Equation: Methods of Solution and Applications*. Springer Berlin Heidelberg, Berlin, Heidelberg, 1996.

[54] L. I. Rudin, S. Osher, and E. Fatemi. Nonlinear total variation based noise removal algorithms. *Physica D: Nonlinear Phenomena*, 60(1):259–268, 1992.

[55] O. Sander. *DUNE — The Distributed and Unified Numerics Environment*. Springer International Publishing, Cham, 2020.

[56] D. Shi, X. Liao, and L. Wang. Superconvergence analysis of conforming finite element method for nonlinear Schrödinger equation. *Applied Mathematics and Computation*, 289:298–310, 2016.

[57] Y. Shin, J. Darbon, and G. E. Karniadakis. On the Convergence of Physics Informed Neural Networks for Linear Second-Order Elliptic and Parabolic Type PDEs. *arXiv preprint arXiv:2004.01806*, 2020.

[58] J. Sirignano and K. Spiliopoulos. DGM: A deep learning algorithm for solving partial differential equations. *Journal of computational physics*, 375:1339–1364, 2018.

[59] J. D. Smith, Z. E. Ross, K. Azizzadenesheli, and J. B. Muir. HypoSVI: Hypocentre inversion with Stein variational inference and physics informed neural networks. *Geophysical Journal International*, 228(1):698–710, 08 2021.

[60] M. Stein. Large Sample Properties of Simulations Using Latin Hypercube Sampling. *Technometrics*, 29(2):143–151, 1987.

[61] W. A. Strauss. The Nonlinear Schrödinger Equation. In G. M. De La Penha and L. A. J. Medeiros, editors, *Contemporary Developments in Continuum Mechanics and Partial Differential Equations*, volume 30 of *North-Holland Mathematics Studies*, pages 452–465. North-Holland, 1978.

[62] D. N. Tanyu, J. Ning, T. Freudenberg, N. Heilenkötter, A. Rademacher, U. Iben, and P. Maass. Deep Learning Methods for Partial Differential Equations and Related Parameter Identification Problems. *arXiv preprint arXiv:2212.03130*, 2022.

[63] C. H. Taubes. *Modeling Differential Equations in Biology*. Cambridge University Press, 2 edition, 2008.

[64] L. Yang, X. Meng, and G. E. Karniadakis. B-PINNs: Bayesian physics-informed neural networks for forward and inverse PDE problems with noisy data. *Journal of Computational Physics*, 425:109913, 2021.