# Accelerating the convergence of Newton's method for nonlinear elliptic PDEs using Fourier neural operators

Joubine Aghili, Emmanuel Franck, Romain Hild, Victor Michel-Dansac, Vincent Vigon

▶ **To cite this version:**

Joubine Aghili, Emmanuel Franck, Romain Hild, Victor Michel-Dansac, Vincent Vigon. Accelerating the convergence of Newton's method for nonlinear elliptic PDEs using Fourier neural operators. 2024. hal-04440076v1

**HAL Id: hal-04440076**
**https://hal.science/hal-04440076v1**

Preprint submitted on 5 Feb 2024 (v1), last revised 1 Mar 2024 (v2)

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

# Accelerating the convergence of Newton's method for nonlinear elliptic PDEs using Fourier neural operators

Joubine Aghili[1,2], Emmanuel Franck[2], Romain Hild[3], Victor Michel-Dansac[2], and Vincent Vigon[1,2]

[1]IRMA, Université de Strasbourg, CNRS UMR 7501, 7 rue René Descartes, 67084 Strasbourg, France
[2]Université de Strasbourg, CNRS, Inria, IRMA, F-67000 Strasbourg, France
[3]GE Healthcare, 2 Rue Marie Hamm, 67000 Strasbourg, France

February 5, 2024

**Abstract**

It is well-known that Newton's method, especially when applied to large problems such as the discretization of nonlinear partial differential equations (PDEs), can have trouble converging if the initial guess is too far from the solution. This work focuses on accelerating this convergence, in the context of the discretization of nonlinear elliptic PDEs. We first provide a quick review of existing methods, and justify our choice of learning an initial guess with a Fourier neural operator (FNO). This choice was motivated by the mesh-independence of such operators, whose training and evaluation can be performed on grids with different resolutions. The FNO is trained by minimizing, on generated data, loss functions based on the PDE discretization. Numerical results, in one and two dimensions, show that the proposed initial guess accelerates the convergence of Newton's method by a large margin compared to a naive initial guess, especially for highly nonlinear or anisotropic problems.

## 1 Introduction

The broad context of this work is the resolution of systems of nonlinear equations using Newton's method. Many applications (nonlinear elliptic partial differential equations (PDEs), implicit time-stepping for nonlinear diffusion or hyperbolic equations, ...) require inverting large nonlinear systems. Throughout this document, such systems will be denoted by

$$F(u) = 0. \tag{1.1}$$

In (1.1), $F : \mathbb{R}^{N_h} \to \mathbb{R}^{N_h}$ is a known nonlinear function and $u \in \mathbb{R}^{N_h}$ is the unknown vector. The integer $N_h$ represents the size of the vector $u$; in the case of a nonlinear system arising from the discretization of a PDE, $N_h$ would be the number of degrees of freedom of the discretization.

For smaller problems, Newton's method is often used. It consists in linearizing the system around a known state $u_0 \in \mathbb{R}^{N_h}$, to obtain an affine approximation of $F$ in a neighborhood of $u_0$:

$$F(u_0) + F'(u_0)(u - u_0) \approx 0.$$

This equation is nothing but a linear system with unknown $u$, whose matrix is the Jacobian matrix of $F$. To apply Newton's method, this linear system is then repeatedly solved using the previous iteration as a reference state. This leads to the following iterative process, where $u^{(k)}$ is expected to tend towards $u$ as $k$ goes to infinity.

**Algorithm 1.** Newton's algorithm to solve (1.1).

1. initialization step: set $u^{(0)} = u_0$,
2. main loop: for $k \geq 0$,
   (2.a) solve the linear system $F(u^{(k)}) + F'(u^{(k)})\delta^{(k+1)} = 0$ for $\delta^{(k+1)}$,
   (2.b) update $u^{(k+1)} = u^{(k)} + \delta^{(k+1)}$.

For larger systems, such as the ones stemming from the discretization of a PDE, it is usual to use Jacobian-Free Newton-Krylov (JFNK) methods [25] or inexact Newton methods [16]. In these methods, the linear systems are solved using an iterative method like the generalized minimal residual method (GMRES) or the conjugate gradient method (CG). This means replacing step (2.a) of Algorithm 1 with an iterative linear solver. Such iterative methods only compute matrix-vector products with the Jacobian matrix. Consequently, it is not necessary to construct and store the Jacobian matrix, but it is sufficient to use an approximation of the matrix-vector product, defined for $v \in R^{N_h}$ by

$$F'(u)v \approx \frac{F(u + \varepsilon v) - F(u)}{\varepsilon},$$

and $\varepsilon$ is correctly chosen. When the linearization error is large, the linear solver does not need to converge to machine precision. Therefore, in the JFNK method, the linear solver threshold is adapted to the nonlinear residual. In practice, the linear solver is stopped as soon as the following criterion is satisfied:

$$\|F(u^{(k)}) + F'(u^{(k)})\delta^{(k+1)}\| \leq \eta^{(k)}\|F(u^{(k)})\|,$$

where the threshold $\eta^{(k)}$ depends on the nonlinear convergence, see [15, 2, 20].

This JFNK method is, however, not always sufficient. Indeed, if the problem is strongly nonlinear, the method can converge very slowly, and even sometimes fail to converge. Additionally, for multiscale PDEs like the magnetohydrodynamics (MHD) equations [17] or anisotropic elliptic equations [10, 11], the Jacobian matrix can be ill-conditioned, which makes the linear step difficult to solve. For this second point, linear preconditioning [8] can be applied to the method, but could fail to improve the nonlinear convergence. There exist several approaches to accelerate this convergence; we summarize the most relevant ones in Section 1.1, and describe the context of this work in Section 1.2.

## 1.1 Accelerating the convergence of Newton's method

In this paragraph, we give a brief overview of several approaches dedicated to the acceleration of Newton's method. We refer the reader to [25] for a more complete review. Namely, we present line search in Section 1.1.1, adaptive inexact Newton methods in Section 1.1.2, nonlinear preconditioning in Section 1.1.3, and initial guess prediction in Section 1.1.4.

### 1.1.1 Line search

A classical approach to accelerate the convergence of the JFNK method is to use line search [6, 25, 38]. This helps to globalize the JFNK method, i.e. to relax the choice of the initial guess $u_0$. The idea is to compute a descent direction $\delta^{(k+1)}$ from the classical Newton linear solve, i.e. step (2.a) of Algorithm 1. Then, at each iteration of the Newton solver, the linear search consists in iteratively finding a large enough real number $\lambda \leq 1$ such that

$$\|F(u^{(k)} + \lambda\delta^{(k+1)})\| \leq \|F(u^{(k)})\|,$$

and then replacing the classical Newton update $u^{(k+1)} = u^{(k)} + \delta^{(k+1)}$ (corresponding to step (2.b) of Algorithm 1) with $u^{(k+1)} = u^{(k)} + \lambda\delta^{(k+1)}$. To stop the iterative process of the line search, there exist many criteria, among which the Armijo or Wolfe conditions [44] are most often used.

### 1.1.2 Adaptive inexact Newton methods

In the specific case of nonlinear systems stemming from a finite element discretization of a PDE, there exists a strategy using a posteriori error estimates. This method was first proposed in [16] for nonlinear elliptic equations, and then extended to multiphase flows in porous media in [12, 13]. First, an a posteriori estimate is constructed using the finite element method to discretize the PDE. This estimate is then employed to adaptively change the linear and nonlinear convergence criteria, while simultaneously refining the mesh. Using such estimates reduces the global cost of the method for a given accuracy. While efficient, this class of methods suffers from a lack of generality, since the a posteriori estimate has to be written for the problem and the numerical method under consideration. Additionally, if the initial guess $u_0$ is too far from the exact solution, then convergence issues may remain.

### 1.1.3 Nonlinear preconditioning

Another approach is the use of nonlinear preconditioners. It is an extension of classical linear preconditioning. Consider a strongly nonlinear system $F(u) = 0$. Nonlinear preconditioning consists in finding a nonlinear operator $G$ such that $G \circ F \approx A$, with $A$ a linear operator. In addition, if $G \approx F^{-1}$, then $A \approx \mathrm{Id}$, which accelerates the convergence. Applying nonlinear preconditioning boils down to solving another (almost linear) nonlinear system

$$G(F(u)) = G(0),$$

where a fast convergence is expected, since the operator $G \circ F$ is close to being linear. The main difficulty of this approach, in addition to finding $G$, is ensuring that the Jacobian matrix of $G \circ F$ (or an approximation thereof) is nonsingular and easily computable, to ensure that the linear solve, step (2.a) of Algorithm 1, has a satisfactory convergence.

In [7, 14], the authors propose nonlinear Schwarz-based preconditioning for Newton's method in the context of domain decomposition. It is an extension of a method commonly used to precondition linear problems. In [41], the authors design a physics-based nonlinear preconditioner for potentially discontinuous, time-independent PDEs. Their approach is based on detecting and eliminating strongly nonlinear or discontinuous regions.

### 1.1.4 Initial guess prediction

Like any iterative algorithm, Newton's method requires an initial guess $u_0$ close enough to the actual solution $u$. An alternative approach to accelerate the convergence of the method is the creation of an initial guess $u_0$, ensuring that $u_0$ is close enough to $u$. To that end, an immediate idea is to construct an operator $G$ which approximates $F^{-1}$, and to use $G(0)$ as an initial guess of Newton's method. Compared to nonlinear preconditioning, this approach may be more restrictive on $G$ but eliminates the nonsingular requirement on the Jacobian matrix of $G \circ F$.

As explained above, a first generic possibility to construct this initial guess is to involve a nonlinear preconditioner. A second one, for nonlinear systems stemming from PDE discretizations, is to use discretization- or physics-based criteria. For time-dependent problems, one can use the previous time step, which was often (not always) sufficient to make Newton's method converge (albeit potentially slowly). One can also use the solution to a linearization of the equation at the current time step, see [9]. However, these guesses cannot be used for elliptic equations. Another strategy is to use the solution of a simpler equation as initial guess. For example, one can solve a (linear) Stokes problem to obtain an initial guess for nonlinear PDEs such as the Navier-Stokes or MHD equations, see [24].

Several approaches based on machine learning have also been proposed in order to predict an initial guess. For instance, see [22] for more general iterative methods, [37] for nonlinear elasticity

problems, [34] for applications in high Reynolds number compressible flows or [36] to accelerate the simulation of chemical reactions in the context of a coupling with fluid dynamics.

The main idea behind the current work is similar, but makes use of different neural networks with a more robust learning approach, as explained in Section 2.

### 1.2 Target problem and chosen methodology

In this work, we propose to accelerate the convergence of the JFNK method by constructing a suitable initial guess. Indeed, it is often harder to obtain a suitable initial guess for nonlinear elliptic PDEs compared to time-dependent PDEs. The main reason for this is that there is no time stepping, and thus the solution at the previous time step can no longer be used as an initial guess. This work focuses on elliptic problems since there usually is no information on the good initial guess to choose. Note that the proposed method can also be used to improve the initial guess for time-dependent problems.

For the sake of simplicity in the notation, all continuous quantities are denoted with capital letters, and discrete ones with lowercase letters; all continuous operators are denoted with calligraphic letters, and discrete ones with capital letters.

In this paper, we consider both one-dimensional (1D) and two-dimensional (2D) elliptic problems, described below. They are governed by similar equations, but with slightly different parameters, and thus we write both problems separately.

The 1D problem is governed by the following elliptic equation on $\Omega \subset \mathbb{R}$, with unknown $U : \Omega \to \mathbb{R}$:

$$\begin{cases} U(x) - \alpha_0 \, \partial_x \left( K(x) \, |U(x)|^p \, \partial_x U(x) \right) = \Phi(x), & \forall x \in \Omega, \\ U(x) = 0, & \forall x \in \partial\Omega, \end{cases} \tag{1.2}$$

with $\Phi \in L^2(\Omega)$ and $K \in \mathcal{C}^0(\Omega, \mathbb{R})$ two given functions, $p$ even, and $\alpha_0 > 0$. We assume that there exists $K_0 > 0$ such that, for all $x \in \mathbb{R}$, $K(x) > K_0(1 + x^2)$. For simplicity, homogeneous Dirichlet boundary conditions are prescribed on $\partial\Omega$.

The 2D problem is a general anisotropic nonlinear diffusion equation on $\Omega \subset \mathbb{R}^2$, with unknown $U : \Omega \to \mathbb{R}$ and governed by:

$$\begin{cases} U(x) - \nabla \cdot \left( K(x) \, |U(x)|^p \, \nabla U(x) \right) = \Phi(x), & \forall x \in \Omega, \\ U(x) = 0, & \forall x \in \partial\Omega, \end{cases} \tag{1.3}$$

with $\Phi \in L^2(\Omega)$ and $K \in \mathcal{C}^0(\Omega, \mathcal{M}_2(\mathbb{R}))$ two given functions, and $p$ an even number; homogeneous Dirichlet boundary conditions are once again prescribed on $\partial\Omega$. We assume that there exists $K_0 > 0$ such that, for all $x \in \mathbb{R}^2$, $\langle x, K(x)x \rangle > K_0 |x|^2$, i.e. that $K(x)$ is coercive for all $x \in \mathbb{R}^2$. Such problems are common in physical applications; it appears for example in Tokamak simulations [21] or simulations of multiphase flows in porous media [1]. For $p = 0$, a classical linear anisotropic diffusion equation is obtained. In this case, the asymptotic limit is ill-posed [10, 11], which may highly degrade the linear conditioning.

Both problems (1.2) and (1.3) can actually be rewritten under a more compact form. Indeed, assume that the solution $U$ belongs to some Hilbert space $\mathcal{U} \subset L^2(\Omega)$, and define the function

$$A = (\Phi, K) \in \mathcal{A} \subset L^2(\Omega) \times \mathcal{C}^0(\Omega, \mathcal{M}_d(\mathbb{R})),$$

where $d$ is the dimension of the space domain $\Omega$. Therefore, $A$ corresponds to the data of the physical model. Therefore, (1.2) and (1.3) can be rewritten as

$$\mathcal{F}(U; A) = 0, \tag{1.4}$$

where $\mathcal{F} : \mathcal{U} \times \mathcal{A} \to L^2(\Omega)$ is the residual operator, with natural boundary conditions for simplicity. In the remainder of this work, we consider $A$ as a set of "parameters" of the physical model. In reality, $A$ corresponds to a set of functions parameterizing the physical model.

Equipped with the definition (1.4) of the generic PDE under consideration, we now proceed to define a procedure to obtain a suitable initial guess for Newton's method applied to a discretization of (1.4). To that end, in Section 2, we describe the whole process of learning this initial guess. Namely, we discuss the choice of the neural network, of the loss function, and of the data generation process. Then, in Section 3, numerical results are given in 1D and 2D to validate our approach. A conclusion ends the paper in Section 4.

## 2 Learning an initial guess

Like mentioned before, in this work, we propose to construct a method able to produce a good initial guess for the JFNK method. We discretize the generic PDE (1.4) using a classical finite difference scheme with $N_h$ mesh points. We obtain the following system of nonlinear equations:

$$F(u; a) = 0, \tag{2.1}$$

with $u \in \mathbb{R}^{N_h}$ the discretization of the unknown function $U$, and $a$ a discretization of $A$, i.e. a discretization of $\Phi$ and $K$. Analogously to $A = \{\Phi, K\}$, we write $a = \{\varphi, k\}$, with $\varphi \in \mathbb{R}^{N_h}$ a discretization of $\Phi$ and $k \in \mathbb{R}^{n_c - 1}$ a discretization of $K$, where $n_c = 2$ in 1D for $d = 1$ and $n_c = 5$ in 2D for $d = 2$. Therefore, $F : \mathbb{R}^{N_h} \times \mathbb{R}^{n_c N_h} \to \mathbb{R}^{N_h}$ is a discretization of the operator $\mathcal{F}$. Our objective is then to construct an operator $G^+ : \mathbb{R}^{n_c N_h} \to \mathbb{R}^{N_h}$ such that

$$F(G^+(a); a) \approx 0. \tag{2.2}$$

Note that, if (2.2) were an equality, $G^+$ would be the pseudo-inverse of $F$ with respect to its first variable: we therefore seek to approximate this pseudo-inverse. This function $G^+$ can be viewed as a discrete operator mapping the data on the mesh to the solution on the mesh, contrary to the nonlinear preconditioners mentioned in Section 1.1.3, which approximate the inverse of the continuous operator. Since $G^+$ is a function from $\mathbb{R}^{n_c N_h}$ to $\mathbb{R}^{N_h}$, it quickly becomes high-dimensional when the mesh is refined.

For this reason, we propose to use a neural network with trainable weights $\theta$ to construct a function $G^+_\theta$ approximating the inverse of $F$, like in (2.2). Indeed, for the last ten years, neural networks have shown their ability to outperform other methods for high-dimensional regression problems. Such operators constructed via neural networks are called neural operators; the reader is directed to [26] for a description of a unified framework for operator learning.

Operator learning comes in two flavors: the discrete case, where a map between discretizations of functions is built, and the continuous case, where the map is between the actual functions themselves. On the one hand, for the discrete case, the most natural approach is to use a convolutional neural network (CNN) to construct $G^+_\theta$ on a structured grid. Such neural networks have been successfully implemented to learn mappings between function in various applications, see e.g. [4, 19, 35, 39, 18, 32]. The main drawback of this approach is that the mesh used for the learning process partially restricts the domain of applicability of the discrete operator. On the other hand, in the continuous approach, the operator is actually constructed between functions, mapping $\Phi$ and $K$ to $U$. Several approaches have been proposed to construct such operators, see for instance [3, 28, 33], which have been unified in a general framework in [26].

The approach best suited to our case seems to be Fourier neural operators (FNOs), introduced in [29]. The idea behind FNOs is to learn a convolution in the Fourier domain instead of the space domain. It makes it possible to obtain mesh- and discretization-independent operators. We will use

an FNO to approximate the continuous operator, before applying any discretization to obtain an approximation of the pseudo-inverse of $F$.

This section is organized as follows: firstly, the loss functions are introduced in Section 2.1, Secondly, we discuss the generation of training, validation and testing data in Section 2.2. Thirdly, the structure of FNOs is briefly sketched in Section 2.3. Fourthly, the hyperparameters of the network are stated, and determined with a grid search, in Section 2.4.

### 2.1 Discretization-Informed loss function

Before introducing the full learning process and the specific network architecture considered in this work, we start by discussing and motivate the loss functions to be minimized. Indeed, we have to correctly manage the interplay between data, physics and discretization. For the sake of simplicity, we consider the generic PDE (1.4), governed by $\mathcal{F}(U; A) = 0$.

To define our loss functions, let us start by temporarily remaining at the continuous level. The best pseudo-inverse operator $\mathcal{G}^+ : \mathcal{A} \to \mathcal{U}$ would satisfy

$$\left\|U - \mathcal{G}^+(A)\right\|^2_{L^2(\Omega)} = 0, \tag{2.3}$$

for all $U$ and $A$ such that $\mathcal{F}(U; A) = 0$. This is nothing saying that $\mathcal{G}^+(A) = U$ in the $L^2$ sense. Similarly, we can replace the $L^2$ norm in the equation above by the $H^1_0$ norm, to obtain

$$\left\|\nabla U - \nabla \mathcal{G}^+(A)\right\|^2_{L^2(\Omega)} = 0. \tag{2.4}$$

In addition, since $\mathcal{G}^+$ is such that $\mathcal{F}(\mathcal{G}^+(A); A) = 0$, it also satisfies

$$\left\|\mathcal{F}(\mathcal{G}^+(A); A)\right\|^2_{L^2(\Omega)} = 0. \tag{2.5}$$

Now, we look for an approximation $G^+_\theta$ of this ideal operator $\mathcal{G}^+$ among a given set of parameterized operators. To that end, we will build it such that it minimizes a weighted sum of three different discrete loss functions, consistent with the three equations (2.3), (2.4) and (2.5). Recall that the discretized PDE reads $F(u; a) = 0$. From now on, we assume that we have at our disposal $N_{\text{data}}$ data points $(u_j, a_j)_{j\in\{1,\dots,N_{\text{data}}\}}$ such that

$$\forall j \in \{1, \dots, N_{\text{data}}\}, \quad u_j \in \mathbb{R}^{N_h}, \quad a_j \in \mathbb{R}^{n_c N_h}, \quad \text{and } F(u_j; a_j) = 0.$$

Generating this data is the object of Section 2.2.

First, we wish to minimize, for all $j$, the $L^2$ error between the prediction $G^+_\theta(a_j)$ of the neural network and the solution $u_j$ of the discretized PDE. This is the discrete analogue of (2.3), which reads

$$\mathcal{L}^{L^2}_{\text{data}}(\theta) = \frac{1}{N_{\text{data}}} \sum_{j=1}^{N_{\text{data}}} \left\|u_j - G^+_\theta(a_j)\right\|^2. \tag{2.6}$$

This is a classical loss function used in supervised learning.

Second, minimizing the $H^1$ error instead of the $L^2$ error to provide a discrete analogue of (2.4) requires defining a discrete approximation of the gradient. For simplicity, we use a centered discretization of the first derivative, which we denote by $D$. This leads to the following loss function:

$$\mathcal{L}^{H^1}_{\text{data}}(\theta) = \frac{1}{N_{\text{data}}} \sum_{j=1}^{N_{\text{data}}} \left\|D\big(u_j - G^+_\theta(a_j)\big)\right\|^2. \tag{2.7}$$

Finally, what is usually done in the context of Physics-Informed Neural Operators (PINOs) is to take the PDE into account in the loss function, see [43, 31]. Therefore, learning such operators mix

classical operator learning and Physics-Informed Neural Networks (PINNs) [40, 23]. This would lead to a loss function analogue to (2.5), where $G_\theta^+$ would directly approximate $\mathcal{G}^+$.

However, in our case, the prediction $G_\theta^+(A)$ of the neural network is to be used as an initial guess for Newton's method. Finding $G_\theta^+$ close to $\mathcal{G}^+$ (in some sense) has no guarantee of minimizing the residual of the numerical scheme. Indeed, it is entirely possible that $G_\theta^+$, viewed as an approximation of the continuous operator $\mathcal{G}^+$, would fail to minimize the discrete residual $F(G_\theta^+(a_j); a_j)$ for some $j$, leading to a poor initial guess for Newton's method. For this reason, we design another loss function, based on the PDE discretization $F$, which we call *discretization-informed*. This loss function takes the discretization $F$ into account rather than the continuous PDE $\mathcal{F}$, and is defined by

$$\mathcal{L}_{\text{dis}}(\theta) = \frac{1}{N_{\text{data}}} \sum_{j=1}^{N_{\text{data}}} \left\| F(G_\theta^+(a_j); a_j) \right\|^2. \tag{2.8}$$

## 2.2 Data generation

Now that the loss functions have been defined, we describe how to generate training and validation data. In this paragraph, we assume that the nonlinearity $p \in \mathbb{N}$ is fixed. Moreover, we denote by $E$ the discrete elliptic operator, defined such that $E(u; k) = \varphi$. Note that $E$ is simply another formulation of $F$: indeed, $E(u; k) = \varphi$ is equivalent to $F(u; a) = 0$, with $a = (k, \varphi)$.

The idea behind our data generation is to randomly generate the solution $U$ and the function $K$, and use this information to compute the associated source term $\Phi$. For this strategy to be efficient, we need to have a rough idea of the family of solutions we wish to approximate. This strategy's advantage, in contrast to generating $\Phi$, is that it avoids having to solve the PDE for data generation. Data generation is summarized in Algorithm 2. It depends on the choice of a random generator, whose definition is the objective of the remainder of this paragraph.

**Algorithm 2.** Data generation

**Input:** $p \in \mathbb{N}$, random generator for $U$ and $K$, $N_{\text{data}}$ number of data points
**Output:** input and output datasets $\mathcal{X}$ and $\mathcal{Y}$

1: **for** $i \in \{1, \dots, N_{\text{data}}\}$ **do**
2: randomly generate $U$ and $K$
3: project $U$ and $K$ onto the discrete space to obtain $u$ and $k$
4: set $\varphi = E(u; k)$
5: add $X_i = (k, \varphi)$ to the input dataset $\mathcal{X}$ and $Y_i = u$ to the output dataset $\mathcal{Y}$
6: **end for**

To complete Algorithm 2, a random generator to build interesting functions $U$ and $K$ has to be designed. This problem depends on the chosen application. Here, even though it is an important question, we do not discuss how to construct a representative dataset for one given application. Instead, we propose the following generic generator, which serves as a proof of concept:

$$\Gamma(x) = 0.5 + \sum_{i=0}^{n} \mathcal{N}_{\mu_i, \sigma_i}(x), \tag{2.9}$$

where $\mathcal{N}_{\mu,\sigma}$ is the probability density function of the normal distribution, with mean $\mu$ and variance $\sigma^2$. The integer $n$ is randomly chosen between 0 and 5, and thus the generated function $\Gamma$ corresponds to a sum of $n+1$ Gaussian functions. Each variance $\sigma_i^2$ is uniformly sampled in $[0.025, 0.07]$, and each mean $\mu_i$ is uniformly sampled in a ball of size 0.25, centered at the midpoint of the space domain $\Omega$. This generator is then used to generate both the solution $u$ and the space function $K$.

In principle, the method could work for other function families. However, the larger the family, the poorer the performance. The definition of this generator completes Algorithm 2. It is then used to define three datasets:

- the training dataset, which includes several fixed mesh sizes;
- the validation dataset, used to compare the model to data and to select the hyperparameters, is constructed in the same way but using additional mesh sizes;
- the test dataset, used to evaluate the performance of the model, is constructed like the validation dataset.

### 2.3 Neural network structure

As mentioned before, we elect to use an FNO in both cases (1.2) and (1.3). For the sake of completeness, we recall the main ideas behind FNOs, and represent the architecture, following [29], on Figure 1. The FNO is made of an extrapolation layer, which starts by lifting the input to a higher dimension, followed by several Fourier layers, and ends with a projection layer, projecting the output back to the original dimension. Such a neural network is exactly equivalent to a fully convolutional one: the input $X$ is transformed via a succession of convolutions and non-linear activation functions, but convolutions are performed via Fast Fourier Transform:

$$\texttt{FFT}^{-1}\Big(\widehat{R}\big(\texttt{FFT}(X)\big)\Big),$$

where $\widehat{R}$ is a trainable vector with a small support $[0, m]$ (where e.g. $m = 20$), which multiplicatively modifies the low frequencies (modes) of the signal. We can imagine that $\widehat{R}$ is itself the Fourier transform of a (never computed) kernel $R$; so our convolution is equivalent to the classical one $X \star R$. In classical convolutional networks, the kernels $R$ have a very small support (e.g. 3), while in the case of FNOs, the kernel have a full spatial support. For the full description of the model, we refer the reader to [29].

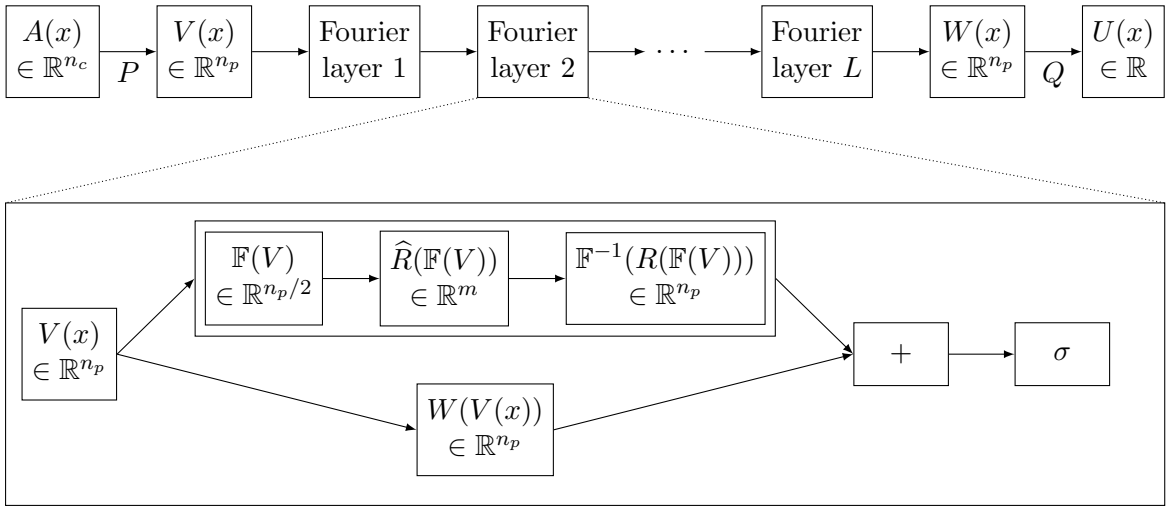



Figure 1: Sketch of an FNO, adapted from [29]. The network has $n_c$ input channels ($n_c = 2$ in 1D and $n_c = 5$ in 2D), which are then extrapolated to $n_p$ dimensions by the extrapolation layer $P$; $L$ Fourier layers follow, whose result is projected back to $\mathbb{R}$ to give an approximation of the PDE solution. In the Fourier layers, $\mathbb{F}$ denotes the fast Fourier transform FFT.

The network always takes as inputs $\Phi$ and $K$; in 1D, this corresponds to two channels, and to five channels in 2D (since $K$ is matrix-valued in this case). The output is always the real-valued

solution $U$. To obtain $G_\theta^+$, this neural network is then batched on the $N_h$ discretization points, to obtain a map from $\mathbb{R}^{n_c N_h}$ to $\mathbb{R}^{N_h}$.

This architecture involves several hyperparameters, written on Figure 1: the number $L$ of Fourier layers, the number $m$ of Fourier modes to keep, and the width $n_p$ (corresponding to the higher dimension after extrapolation). To choose these hyperparameters, we perform a grid search, described below.

## 2.4 Hyperparameter choice

So far, this section has been dedicated to the construction of the neural network $G_\theta^+$ to provide a relevant initial guess, and of the loss functions to be minimized during training. The training process itself is rather standard, based on gradient methods and back-propagation, as is usual for neural networks. However, this process depends on quite a lot of hyperparameters, given in Section 2.4.1; it is crucial to choose them carefully to obtain a good performance. To that end, we perform a grid search, described in Section 2.4.2, to find the best hyperparameters.

### 2.4.1 Hyperparameters

As a first step, we recall all the hyperparameters involved in the learning process. We split them into two categories: the hyperparameters used for the training of the neural network, and the hyperparameters of the neural network itself. For each hyperparameter, we also give the possible values explored in the grid search.

| Training hyperparameters | Set of values for the grid search |
|---|---|
| $\ell_0$ (Initial learning rate) | $10^{-2}, 10^{-3}, 10^{-4}$ |
| $\gamma$ (Exponential decay) | $0.98, 0.99, 1$ |
| $N_b$ (Batch size) | $64, 128, 256, 512$ |
| $\omega$ (Loss function weight) | $0, 0.5, 1$ |

Table 1: Hyperparameters used for the training of the neural network, and values explored in the grid search.

First, the training hyperparameters are collected in Table 1. The first hyperparameter, $\ell_0$, is the initial learning rate of the ADAM optimizer. The learning rate is then exponentially decayed, with rate $\gamma$. The parameter $N_b$ is the batch size, i.e. the number of samples used to compute the loss functions. The parameter $\omega$ corresponds to the weight between the loss functions. The choice of the final loss function will be discussed in more detail in Section 3. For the moment, suffice it to say that the loss function combines the three loss functions (2.8)–(2.6)–(2.7) differently in 1D, for (1.2), and in 2D, for (1.3). In 1D, we use a combination between the $L^2$ data loss function (2.8) and the discretization-informed loss function (2.7); the resulting loss function is given by

$$\mathcal{L}_{1\mathrm{D}}(\theta) = \omega \mathcal{L}^{L^2}_{\mathrm{data}}(\theta) + (1-\omega) 10^{-4} \mathcal{L}_{\mathrm{dis}}(\theta). \tag{2.10}$$

In 2D, the combination is between the $L^2$ data loss function (2.8) and the $H^1$ data loss function (2.6), to obtain the following loss function:

$$\mathcal{L}_{2\mathrm{D}}(\theta) = \omega \mathcal{L}^{L^2}_{\mathrm{data}}(\theta) + (1-\omega) 10^{-2} \mathcal{L}^{H^1}_{\mathrm{data}}(\theta). \tag{2.11}$$

The factors $10^{-4}$ in (2.10) and $10^{-2}$ in (2.11) are introduced to normalize the discretization-informed and the $H^1$ loss functions, which are always larger than the $L^2$ loss function. Indeed, recall that the

generator (2.9) generates functions based on the probability density function of a normal distribution. Therefore, their derivatives with respect to $x$, and especially their second derivatives, have a norm greater than that of the functions themselves. We have added the normalizing factors to account for this discrepancy.

| Network hyperparameters | Set of values for the grid search |
|---|---|
| $L$ (Fourier layers) | $2, 3, 4, 5$ |
| $m$ (Fourier modes) | $10, 20, 30, 40$ |
| $n_p$ (FNO width) | $10, 20, 30, 50$ |

Table 2: Hyperparameters used for the neural network itself, and values explored in the grid search.

Second, the hyperparameters of the neural network are summarized in Table 2. We refer the reader to Section 2.3 and Figure 1 for an explanation of these hyperparameters.

### 2.4.2 Grid search

The main idea behind grid search is to run the training process for all possible combinations of hyperparameters, and to select the best combination. We denote by $\boldsymbol{\mu}$ the set of hyperparameters of our problem:

$$\boldsymbol{\mu} = \{\ell_0, \gamma, N_b, \omega, L, m, p\}.$$

Grid search can be formalized as the following optimization problem:

$$\boldsymbol{\mu}_{\text{opt}} = \underset{\boldsymbol{\mu}}{\text{argmin}}\, S(\boldsymbol{\mu}),$$

with $S$ a score function to be defined and $\boldsymbol{\mu}_{\text{opt}}$ the optimal parameters. To avoid overfitting and increase the generalization ability of our model, this score function will be computed on validation data, which is generated following Section 2.2. We denote by $N_{\text{val}}$ the number of validation data. As we will see upon defining a suitable score function, this score function will not necessarily be differentiable with respect to $\boldsymbol{\mu}$. This is why we choose to use a grid search rather than a gradient-based optimization algorithm.

The classical choice in the machine learning community is to set the score function equal to parts of the loss function of the problem, given here by (2.10) or (2.11). We thus define two score functions: $S_{\text{data}}$, based on the $L^2$ loss function, and $S_{\text{dis}}$, based on the discretization-informed loss function. Their evaluations are summarized in Algorithm 3.

**Algorithm 3.** Evaluation of $S_{\text{data}}(\boldsymbol{\mu})$ and $S_{\text{dis}}(\boldsymbol{\mu})$

**Input:** set of parameters $\boldsymbol{\mu}$, $n_{\text{epoch}}$ number of training epochs, $N_{\text{val}}$ number of validation data, $(X_i, Y_i)_{i\in\{1,\dots,N_{\text{val}}\}}$ validation data

**Output:** scores $S_{\text{data}}(\boldsymbol{\mu})$ and $S_{\text{dis}}(\boldsymbol{\mu})$

1: train $G_\theta^+$ during $n_{\text{epoch}}$ epochs with hyperparameters $\boldsymbol{\mu}$

2: compute the scores $S_{\text{data}}(\boldsymbol{\mu}) = \sum_{i=1}^{N_{\text{val}}} \|G_\theta^+(X_i) - Y_i\|_2^2$ and $S_{\text{dis}}(\boldsymbol{\mu}) = \sum_{i=1}^{N_{\text{val}}} \|F(G_\theta^+(X_i))\|_2^2$

However, using such a score function implies the underlying assumption that the better the neural network approximation of the solution, the faster the convergence of Newton's method. Even though

this assumption seems natural, it remains debatable. Indeed, there could be cases where the model provides a good approximation of the PDE solution in the $L^2$ norm, but local errors or irregularities could slow down the convergence of Newton's method if using this approximation as an initial guess. Since our end goal is to accelerate the convergence of Newton's method, another natural choice for the score function would be to base it on the actual number of iterations. Computing such a score function amounts to running Newton's algorithm with the neural network prediction and with the naive initial guess, and comparing the number of iterations. The score is then defined as the average gain in iterations between the two approaches. As a consequence, we suggest computing this new score function $S_{\text{iter}}(\boldsymbol{\mu})$ via Algorithm 4.

**Algorithm 4.** Evaluation of $S_{\text{iter}}(\boldsymbol{\mu})$

**Input:** set of hyperparameters $\boldsymbol{\mu}$, $n_{\text{epoch}}$ number of training epochs, $N_{\text{val}}$ number of validation data, $(X_i)_{i\in\{1,\dots,N_{\text{val}}\}}$ validation data
**Output:** score $S_{\text{iter}}(\boldsymbol{\mu})$

1: train $G_\theta^+$ during $n_{\text{epoch}}$ epochs with hyperparameters $\boldsymbol{\mu}$
2: **for** $i \in \{1, \dots, N_{\text{val}}\}$ **do**
3: compute the initial guess $u_0 = G_\theta^+(X_i)$ using the trained neural network
4: solve the problem with a JFNK solver, with initial guess $u_0$, and denote by $k_i$ the number of iterations needed to reach convergence
5: solve the problem with a JFNK solver, with the naive initial guess $(1, 1, \dots, 1)$, and denote by $\tilde{k}_i$ the number of iterations needed to reach convergence
6: **end for**
7: compute the score $S_{\text{iter}}(\boldsymbol{\mu}) = \dfrac{1}{N_{\text{val}}} \displaystyle\sum_{i=1}^{N_{\text{val}}} \frac{\tilde{k}_i}{k_i}$

To evaluate the best values of each hyperparameter, we evaluated the three scores on several mesh resolutions. For the loss-based scores $S_{\text{data}}$ and $S_{\text{dis}}$, we used $N_{\text{val}} = 1000$ validation data for each mesh resolution. The iteration-based score $S_{\text{iter}}$, however, is more expensive to compute since Newton's method has to be run up to convergence. Therefore, to compute this score, performed $N_{\text{val}} = 5$ Newton resolutions per mesh resolution.

On Figure 2, we display the results of the grid search on the 1D problem (1.2), only for the batch size hyperparameter $N_b$ for simplicity. From left to right, we display scores $S_{\text{data}}$, $S_{\text{dis}}$ and $S_{\text{iter}}$, with respect to the number $N_h$ of points, for different values of the batch size $N_b$. The left and center panels show that a batch size of $N_b = 512$ is not efficient, while the other batch sizes yield comparable scores. Moving on to the right panel, we observe that $N_b = 64$ is the best choice to reduce the number of iterations of Newton's method. Consequently, on the basis of these indicators, we choose $N_b = 64$.

All in all, the hyperparameters are chosen by performing the grid search and favoring the results associated with $S_{\text{iter}}$, which corresponds to our final objective. Note that, in most cases, all three scores agree on the best value of each hyperparameter. The final choice of all hyperparameters is summarized in Table 3.

## 3 Numerical results

Equipped with the values of the hyperparameters collected in Table 3, we now present some numerical results to validate our approach. Namely, we compare the convergence of Newton's method using a naive, constant initial guess and using our FNO as a predicted initial guess. We first tackle the 1D case in Section 3.1, and move on to the 2D case in Section 3.2. In each case, we start by displaying the

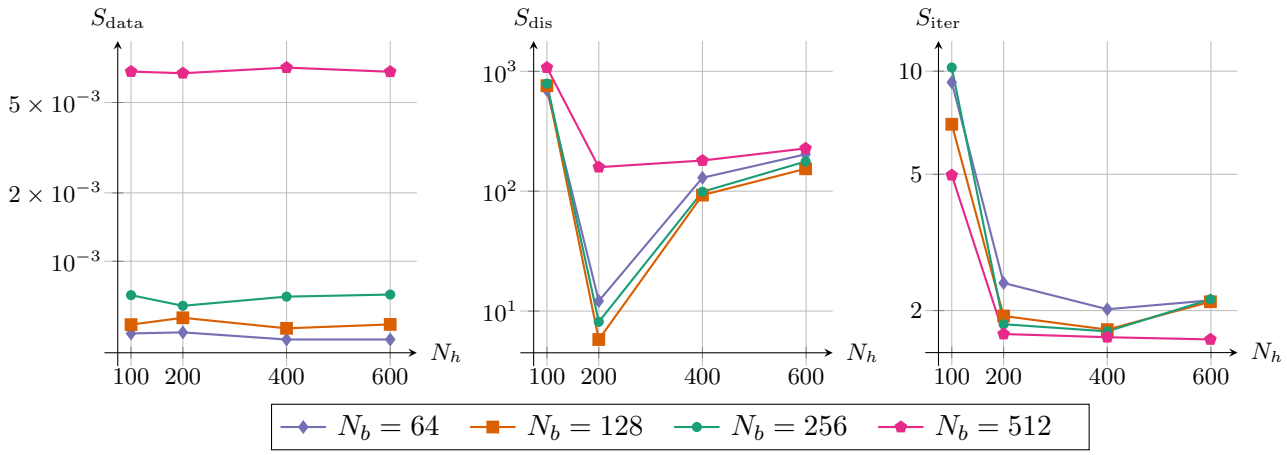



Figure 2: Score with respect to the number of points in the grid, for different values of the batch size $N_b$. From left to right, we display scores $S_{\text{data}}$, $S_{\text{dis}}$ and $S_{\text{iter}}$. Scores $S_{\text{data}}$ and $S_{\text{dis}}$ should be as small as possible; score $S_{\text{iter}}$ should be as large as possible.

| Hyperparameters | Chosen value |
|---|---|
| $\ell_0$ (Initial learning rate) | $10^{-3}$ |
| $\gamma$ (Exponential decay) | 0.99 |
| $N_b$ (Batch size) | 64 |
| $\omega$ (Loss function weight) | 0.5 |
| $L$ (Fourier layers) | 4 |
| $m$ (Fourier modes) | 30 |
| $n_p$ (FNO width) | 30 |

Table 3: Summary of the hyperparameters; for each hyperparameter, we give the best value obtained through grid search.

results of the FNO prediction, to make sure they are close to the exact solution. Then, we compare both types of initial guesses.

Throughout this section, we use the `newton_krylov` solver from the `scipy` library. It is already a very efficient solver, with built-in advanced optimizations. It is a Jacobian-free Newton-Krylov method, where linear solves are performed using the LGMRES method and where the convergence is enhanced with an Armijo line search method. Such a method is typical for large-scale problems, for instance ones stemming from PDE discretizations.

### 3.1 Results in one space dimension

The first batch of experiment we run are based on the 1D PDE (1.2), with the nonlinearity $p$ fixed to 4. The coefficient $\alpha_0$ is there to tune the strength of the nonlinearity: larger values of $\alpha_0$ lead to a prevalence of the nonlinear part, and thus makes the JFNK method slower to converge. We discretize this equation with a classical finite difference scheme.

For each experiment, we fix a value of $\alpha_0$ to represent the difficulty of the problem. Also, the model is trained on meshes with 200 and 400 points. A goal of these experiments is to check the ability of the FNO to predict a solution for mesh resolutions different from the ones used for training.

We first discuss the choice of the loss function in Section 3.1.1, and then check the relevance of the initial guess provided by the FNO for a small nonlinearity $\alpha_0 = 2$ in Section 3.1.2 and for larger nonlinearities $\alpha_0 \in \{5, 8\}$ in Section 3.1.3.

### 3.1.1 Choice of the loss function

In this paragraph, we discuss the choice of the loss function by checking the ability of the FNO to approximate the solution to (1.2), and to its discretized version (2.1). Since the goal is to compare the loss functions, we train a smaller FNO, with $L = 4$, $m = 20$ and $p = 20$. Moreover, we take $\alpha_0 = 5$ to have a strong nonlinearity.

We compare two training strategies: the first one uses only the $L^2$ data loss function (2.8), while the second one uses both the $L^2$ data loss function (2.8) and the discretization-informed loss function (2.7). In both cases, the networks are trained until the mean squared error (MSE) reaches around $2 \times 10^{-4}$. These two strategies are compared in Figures 3 and 4, where 4 random examples are displayed. In each example, functions $\Phi$ and $K$ are generated following Section 2.2, and the FNO is used to predict the resulting solution $u$. In each figure, the first three panels from left to right display the functions $\Phi$, $K$, $u$ (predicted in blue and exact in orange). The rightmost panel compares the residuals of the discretized PDE computed with the exact solution (blue line) and the predicted solution (orange line).

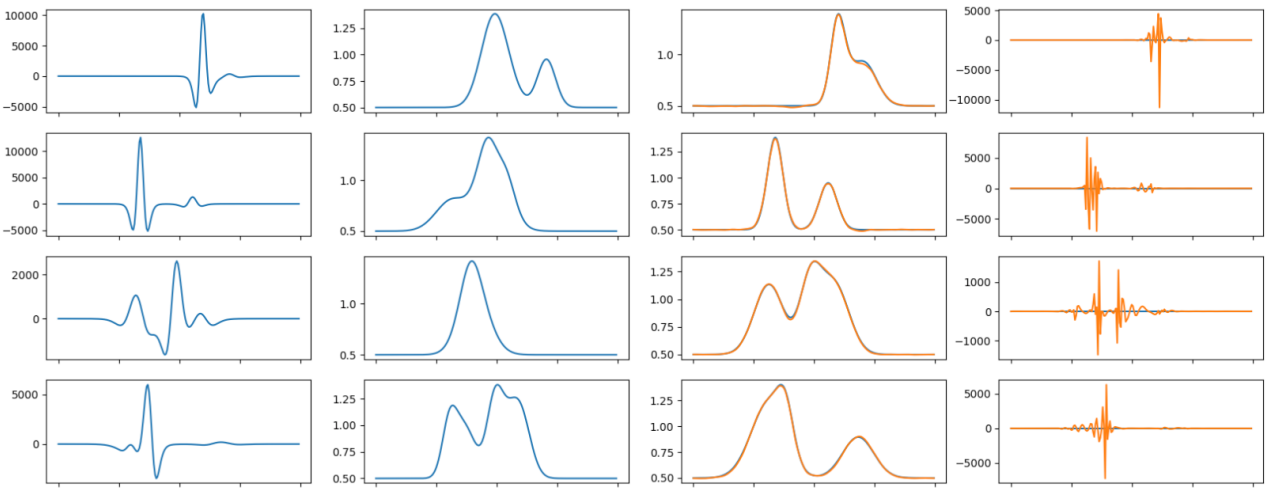


Figure 3: Examples obtained after training the FNO with the loss function $\mathcal{L}^{L^2}_{\text{data}}$ from (2.6) only. From top to bottom, each series of graphs correspond to a different example. In the first three columns, from left to right, we display $\Phi(x)$, $K(x)$, the solution $U(x)$ (blue line) and its prediction (orange line). The rightmost column compares the residuals of the discretized PDE computed with the exact solution (blue line) and the predicted solution (orange line).

In the case of Figure 3, only the $L^2$ data loss function is used; training time is below 10 minutes on a cloud architecture with shared GPUs. We note that the FNO is able to give a very good prediction of the solution $U$. However, we also remark that the residual obtained by plugging this solution into the numerical scheme becomes quite large. In light of the PDE (1.2), since the third column shows that $U$ is correctly approximated by the FNO, the high amplitude of the residual in the fourth column can only be due to a poor approximation of the discrete derivatives of $U$ (and especially of its second derivative).

Figure 4 shows the results obtained for an FNO trained with both the $L^2$ data and discretization-informed loss function; training time is below 15 minutes, still on a cloud architecture with shared GPUs. On the one hand, we observe that the network yields a good approximation of the solution $U$,

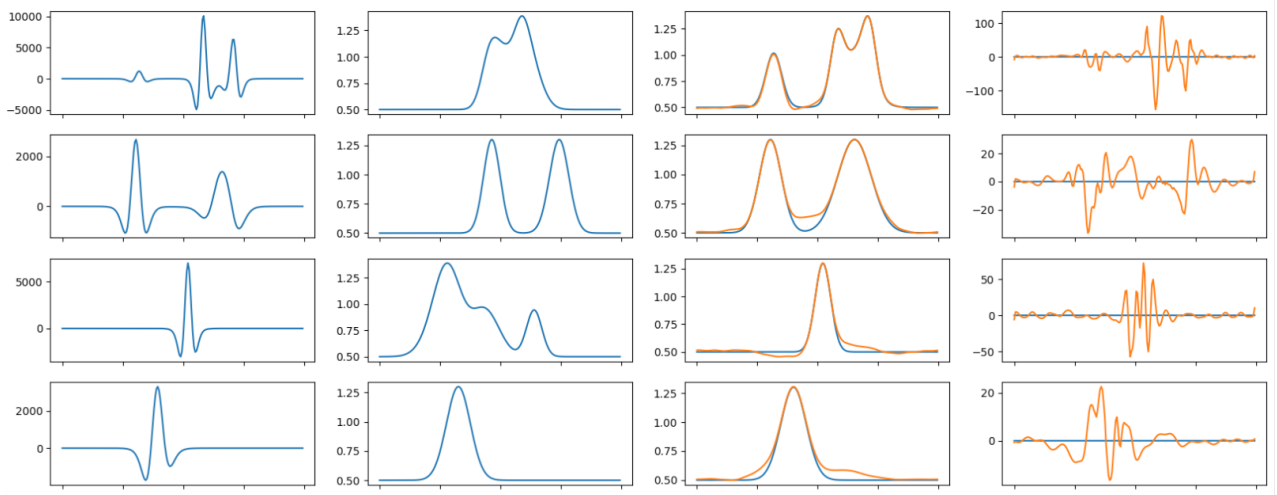


Figure 4: Examples obtained after training the FNO with the loss function $\mathcal{L}^{L^2}_{\text{data}}$ from (2.6) and the discretization-informed loss function $\mathcal{L}_{\text{dis}}$ from (2.8). From top to bottom, the four graphs correspond to a different example. In the first three columns, from left to right, we display $\Phi(x)$, $K(x)$, the solution $U(x)$ (blue line) and its prediction (orange line). The rightmost column compares the residuals of the discretized PDE computed with the exact solution (blue line) and the predicted solution (orange line).

but that it performs noticeably worse than in the previous case. On the other hand, the residual of the scheme is much smaller, which will be beneficial for the convergence of Newton's method. In fact, by minimizing the discretization residual, we have obtained a better reconstruction of the residual, and therefore of the discrete derivatives of $U$. In practice, we will use this second training strategy for the final results, because it produces a better initial guess for the JFNK method.

### 3.1.2 Results for $\alpha_0 = 2$

We start by testing the approach with a weak nonlinearity ($\alpha_0 = 2$). In this case, Newton's method is able to converge quite quickly with the naive initial guess. Nevertheless, we expect our predicted initial guess to outperform the naive one by a significant margin.

Recall that training is done on meshes with 200 and 400 points; validation is performed by running Algorithm 4 on meshes with varying sizes (100, 200, 400 and 600 points). The training time is around 15 minutes on a cloud architecture with shared GPUs. The tolerance for the convergence of Newton's method is fixed to $10^{-6}$. The results are displayed on Figure 5. Namely, in the left panels, we display the gains in number of iterations; in the right panel, the gains in CPU time is displayed. In each case, we perform 50 runs, corresponding to 50 random choices of $\Phi$ and $K$. The gains in number of iterations are nothing but the score $S_{\text{iter}}$ defined in Algorithm 4, while the gains in CPU time (denoted by $S_{\text{CPU}}$) are computed in the same way but comparing the CPU time of the two runs instead of the number of iterations. Instead of the raw gains, we report the gain percentage:

$$G_{\text{iter}} = (S_{\text{iter}} - 1) \times 100 \quad \text{and} \quad G_{\text{CPU}} = (S_{\text{CPU}} - 1) \times 100. \tag{3.1}$$

On the top panels of Figure 5, corresponding to gains on a mesh with 100 points, we observe that the average gain when using the initial guess predicted by the FNO is around 1300% in terms of number of iterations and 500% in terms of CPU time. In every example, we note that the gain in the number of iterations is consistently over 100%. This means that initializing Newton's method with the prediction allows it to converge in at most half the number of iterations compared to using the naive initial guess. This gain in iterations translates into a (lower) gain in CPU time in most

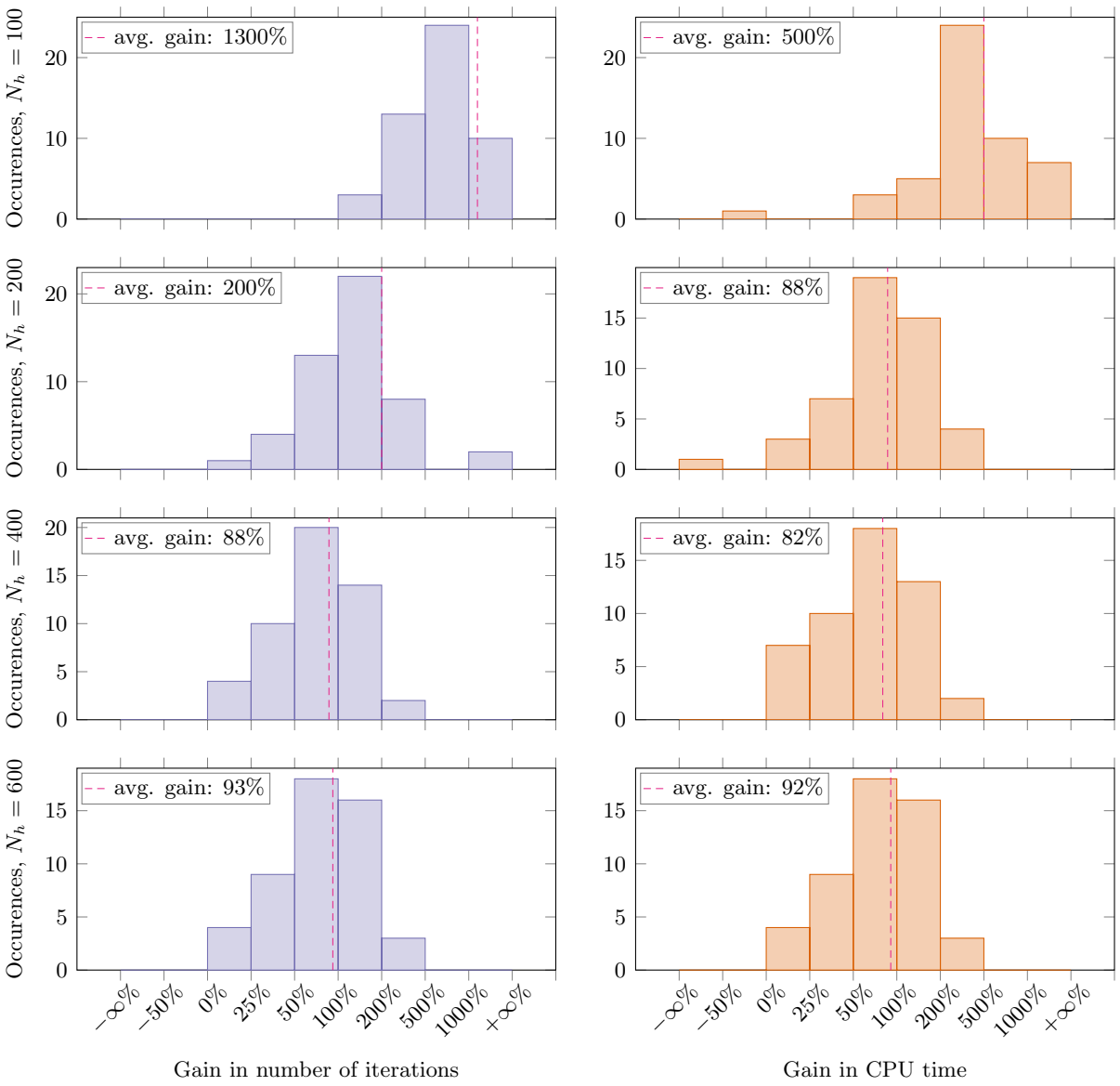



Figure 5: Statistics of the gains $G_{\text{iter}}$ in number of iterations (left panels) and $G_{\text{CPU}}$ in CPU time (right panels) for the 1D problem (1.2) with $\alpha_0 = 2$. From top to bottom, we display the results for meshes with 100, 200, 400 and 600 points. The gains are grouped in bins of varying sizes, and the number of occurrences in each bin is displayed as a histogram.

cases. However, in one case out of fifty (i.e. 2.5% of the time), the total CPU time is actually larger when using the predicted initial guess. This increase in computation time is due to the fact that each iteration of the JFNK method is quite fast (since the mesh is coarse), which increases the relative weight of the network call in the total CPU time. We expect that, the finer the mesh, the closer the gains in number of iterations and in CPU time will be.

Indeed, these expectations are confirmed in the bottom three panels of Figure 5, where the results are displayed, from top to bottom, for meshes with 200, 400 and 600 points. Our predicted initial guess enables a consistent gain in number of iterations and in CPU time, even on cases which were not part of the training database. Moreover, we indeed observe that the gains in number of iterations becomes closer to the gains in CPU time as the mesh becomes finer, since the cost of the FNO call gradually becomes irrelevant.

An important observation is that the method is less effective on finer meshes than on coarser ones, with gains five to ten times lower on fine meshes. This can be attributed to the specific way having a good initial guess enhances Newton's method. Indeed, Newton's method is known to converge in two phases, see e.g. [5]. In the second phase, Newton's method starts converging, i.e. the residual decreases with the iterations. This second phase only starts when the residual is small enough, after a first phase where Newton's method explores the space of solutions. This first phase corresponds to a plateau when graphing the residual with respect to the iteration number. It should be noted that the convergence phase is longer for harder problems, e.g. on finer meshes. Our prediction jumpstarts the convergence phrase by providing an initial guess which already corresponds to a small residual. This makes it possible to reduce, or even skip, the plateau in the convergence curves. This explains the smaller gains for finer meshes: the convergence phase, which is not changed by our approach, represents a larger part of the total number of iterations. We expect this behavior to be reproduced for larger values of $\alpha_0$, which also correspond to harder problems.

#### 3.1.3 Results for $\alpha_0 = 5$ and $\alpha_0 = 8$

To confirm our claims from the previous section, we now treat the same problem with larger values of $\alpha_0$. For large values of $\alpha_0$, the nonlinear part takes precedence over the linear part in the elliptic PDE. Therefore, the problem is harder, and the classical JFNK method will take longer to converge (if it manages to converge at all). In this section, we run 25 experiments for $\alpha_0 = 5$ and $\alpha_0 = 8$, each corresponding to a random choice of $\Phi$ and $K$. The training time is around 20 minutes, slightly increased compared to the more linear case.

The results for $\alpha_0 = 5$ are displayed on Figure 6. We observe a broadly similar behavior compared to the previous case: except in one case, the predicted initial guess consistently outperforms the naive guess in both iteration number and computation time. Compared to the previous $\alpha_0 = 2$ case, the gains are even larger, roughly twice as large in all cases. As expected, since the PDE is harder to solve, the classical JFNK method remains stuck longer in the plateau phase for lack of a suitable initial guess. Our prediction allows Newton's method to start with a lower residual, which skips the plateau phase (or at least greatly reduces its duration).

These results are confirmed by the experiments with $\alpha_0 = 8$. Since this case is harder to deal with, we increase the width of the FNO to 40 (from 30), and decrease the learning rate to $7.5 \times 10^{-4}$ (from $10^{-3}$). Since the distribution of the gains is similar to the $\alpha_0 = 5$ case, we do not display the histograms for the sake of brevity. Instead, we collect the values of the average CPU time gains in Table 4. We also do not report the gains in number of iterations, since they are larger than the gains in computation time.

| mesh size | $\alpha_0 = 2$ (50 examples) | $\alpha_0 = 5$ (25 examples) | $\alpha_0 = 8$ (25 examples) |
|---|---|---|---|
| 100 points | +500% | +1800% | +5000% |
| 200 points | +88% | +230% | +600% |
| 400 points | +82% | +150% | +230% |
| 600 points | +92% | +220% | +250% |

Table 4: Average gains in CPU time when using the predicted initial guess rather than the naive one, for different values of $\alpha_0$.

In Table 4, we observe that the mean gains in CPU time are larger for coarser meshes and larger values of $\alpha_0$. This is consistent with our previous observations. Overall, in each situation under consideration, our prediction never fails to reduce the number of iterations. There is a small

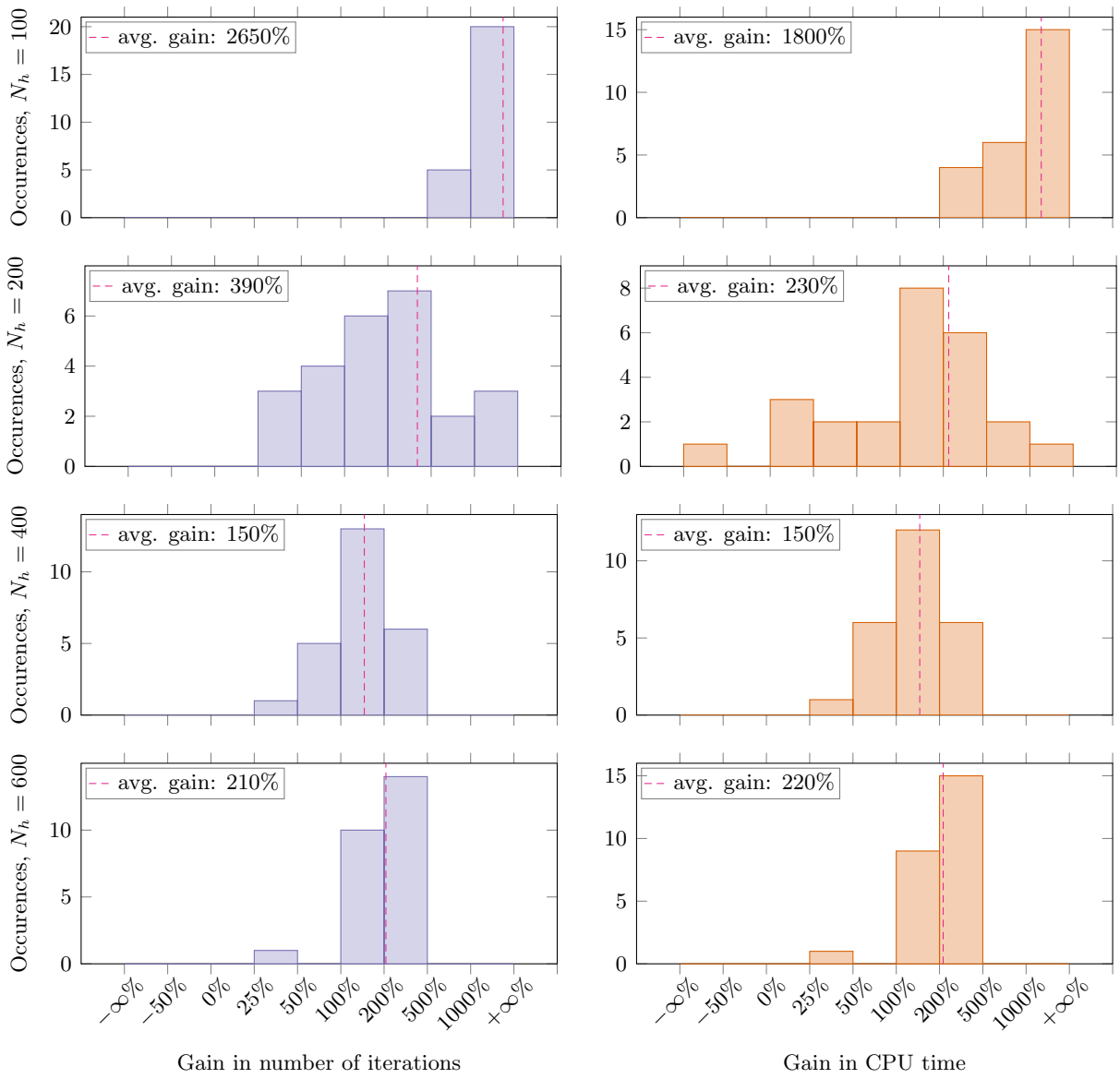



Figure 6: Statistics of the gains $G_{\text{iter}}$ in number of iterations (left panels) and $G_{\text{CPU}}$ in CPU time (right panels) for the 1D problem (1.2) with $\alpha_0 = 5$. From top to bottom, we display the results for meshes with 100, 200, 400 and 600 points. The gains are grouped in bins of varying sizes, and the number of occurrences in each bin is displayed as a histogram.

percentage (around 2%) of simulations where the CPU time is increased by using our prediction. This could be remedied by a larger network or a longer training time.

We have also studied the dependency of the Newton convergence with respect to the naive initial guess. In such elliptic problems, we usually do not have any prior on the solution, so changing initial guesses merely amounts to changing the constant value used as initial guess. We tried values of 0.5, 0.8, 1.2 and 1.5; however, the JFNK method only converged for constant initial guesses of 0.8 and 1. Therefore, it seems necessary to take a constant value close to the average of the true solution. This shows the strong dependency of Newton's method on the initial guess, and further highlights the interest of our approach.

### 3.2 Results in two space dimensions

We now tackle the extension of the 1D case (1.2) to two space dimensions, namely the elliptic system (1.3), with $p = 2$. It is discretized using a classical finite difference scheme for anisotropic equations, see the review paper [42]. The test case is similar to the one we formulated in 1D. The diffusion matrix $K$ is obtained by first generating a function $\delta$ as in 1D, and then generating a random, constant symmetric positive matrix $B$; the resulting matrix $K$ is given for all $x \in \Omega$ by $K(x) = \delta(x)B$. This enables us to control the anisotropy of the problem through the ratio of the eigenvalues of $B$.

Just like before, we first display the capability of our network to predict the solution in Section 3.2.1. Then, in Section 3.2.2, we compare the performance of Newton's method with the naive initial guess and with our predicted initial guess. This time, the initial learning rate is set to $\ell_0 = 8 \times 10^{-4}$ to account for the increased difficulty of the problem. Moreover, in this case, we replaced the discretization-informed loss function (2.7) with the $H^1$ loss function (2.6). Indeed, the value of the discretization-informed loss function depends on the mesh size, which prevented us from finding good hyperparameters, even with a grid search. Since the $H^1$ loss function also gave good results in 1D, we decided to use it instead of the discretization-informed one.

#### 3.2.1 FNO prediction

We first display, in Figure 7, some random examples of functions $U$ predicted by the trained FNO. They show that the FNO is able to predict a reasonable approximation of the solution $U$ for variable matrices $K(x)$ (corresponding to mostly isotropic diffusion in the top three examples, and to anisotropic diffusion in the bottom one). We also observe that the FNO does not produce a perfect approximation of the constant parts of the solution.

#### 3.2.2 Using the FNO prediction as initial guess

Equipped with a suitable prediction, we now study the improvement in the convergence of Newton's method when using it as an initial guess rather than a naive guess. This time, Newton's method is said to converge if the error has reached the tolerance of $10^{-5}$. The FNO was trained on meshes with $60^2$ and $70^2$ points, and the validation is performed on meshes with $40^2$, $60^2$, $80^2$ and $100^2$ points.

In Table 5, we give the improvement ratios for several mesh sizes, in terms of number of iterations and CPU times. This table has been produced by averaging out the results of 25 simulations. To save space, we do not give the detailed breakdown of the results, and we give the raw multiplicative gains $S_{\text{iter}}$ and $S_{\text{CPU}}$ rather than their percentage form (3.1).

| | $S_{\text{iter}}$ | | | $S_{\text{CPU}}$ | | |
|---|---|---|---|---|---|---|
| $N_h$ | min | avg | max | min | avg | max |
| $40^2$ | 3.76 | 4.78 | 5.86 | 1.73 | 2.42 | 2.98 |
| $60^2$ | 2.69 | 3.29 | 3.95 | 1.62 | 1.94 | 2.31 |
| $80^2$ | 1.96 | 2.44 | 3.02 | 1.36 | 1.66 | 2.05 |
| $100^2$ | 1.63 | 2.18 | 2.85 | 1.22 | 1.58 | 1.83 |

Table 5: Total gain factor, in CPU time and number of iterations, obtained by using the network rather than the naive initial guess. We observe that the minimum gain is always greater than 1, in terms of CPU time or number of iterations.

These 2D results are somewhat similar to the 1D ones, even if an exact comparison is hard to perform since there are more computation points in 2D than in 1D. The main takeaway is that, in

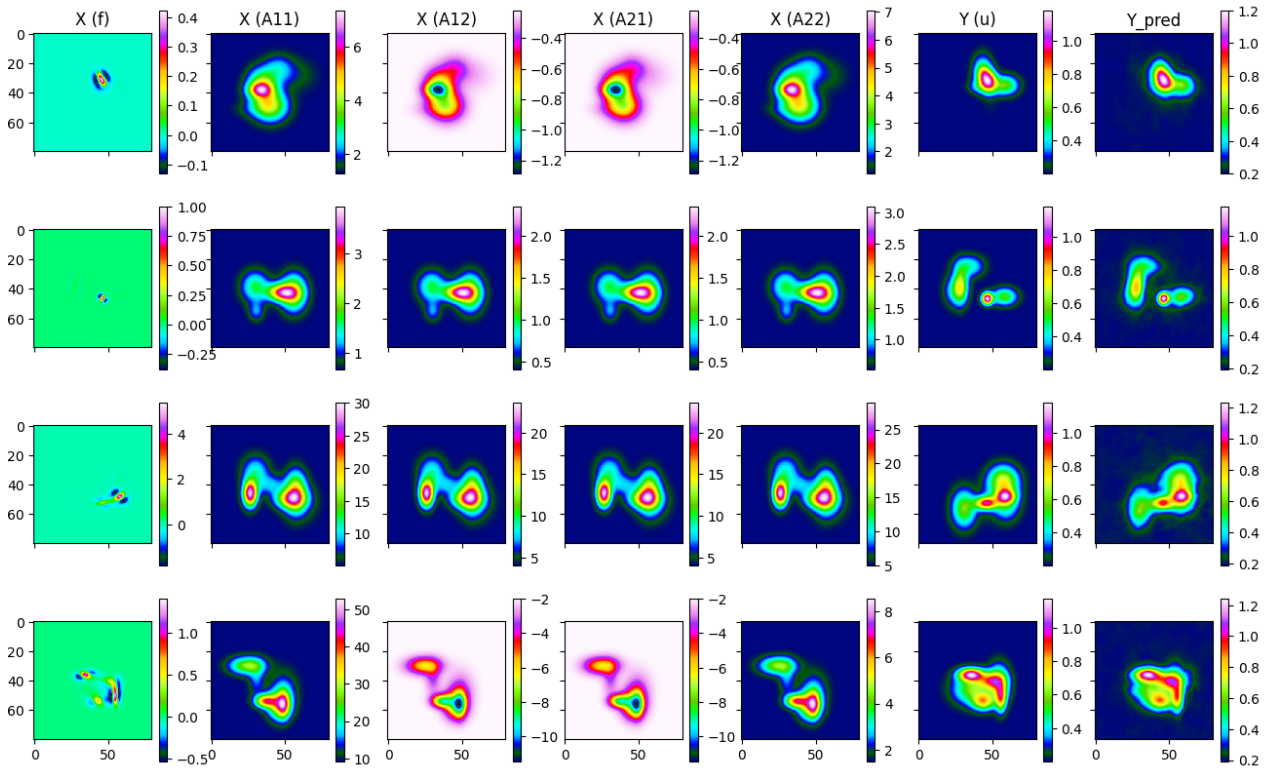



Figure 7: From top to bottom, we display four random examples of generated functions for (1.3), as well as their predictions by the FNO. From left to right, we display the normalized values of $\Phi(x)$, the values of the four coordinates of the matrix $K$, denoted by $K_{11}(x)$, $K_{12}(x)$, $K_{21}(x)$ and $K_{22}(x)$, the reference solution $U(x)$, and the prediction of the FNO. In each case, we observe a good match between the prediction (rightmost column) and the reference solution (column left of the rightmost one).

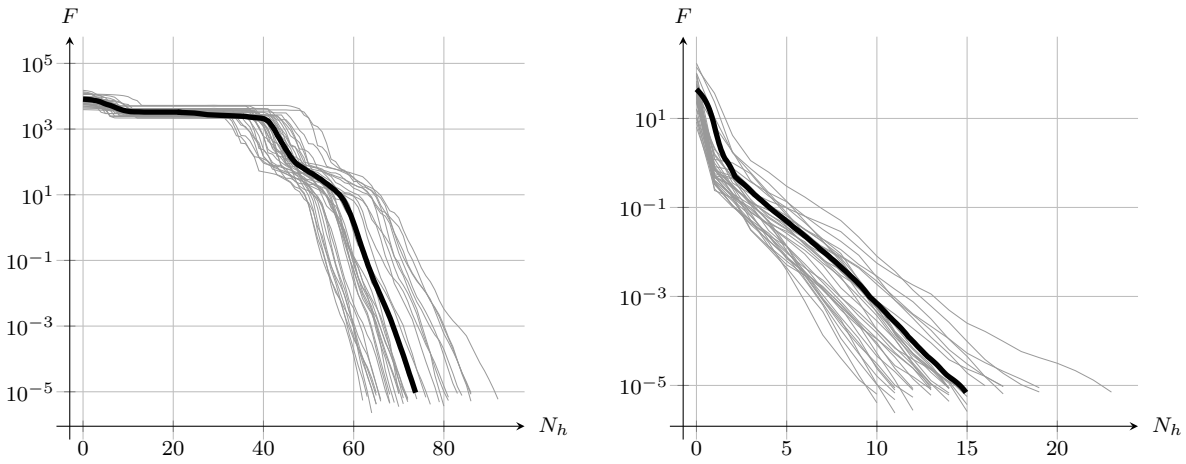
every case, our initialization makes it possible to see consistent gains in both number of iterations and CPU time. Furthermore, it can be seen that the difference between the gain in number of iterations and the gain in computation time is larger than in 1D. This is due to the two-phase convergence of Newton's method; the remainder of this section is dedicated to a finer study of both these phases and of how our prediction affects them.

| | $F \geq 10$ | | | $1 \leq F < 10$ | | | $0.1 \leq F < 1$ | | | $F \leq 0.1$ | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $N_h$ | min | avg | max | min | avg | max | min | avg | max | min | avg | max |
| $40^2$ | 24.50 | 54.19 | $\infty$ | 15.25 | 49.12 | 69.00 | 9.14 | 16.59 | 23.33 | 0.88 | 1.08 | 1.30 |
| $60^2$ | 33.50 | 65.45 | $\infty$ | 13.17 | 21.72 | 33.50 | 5.93 | 9.17 | 13.60 | 0.95 | 1.11 | 1.24 |
| $80^2$ | 16.00 | 36.91 | 73.00 | 7.60 | 12.72 | 16.60 | 3.81 | 5.83 | 8.00 | 0.89 | 1.07 | 1.28 |
| $100^2$ | 13.57 | 27.67 | 69.00 | 6.14 | 9.85 | 15.80 | 3.31 | 4.77 | 7.08 | 0.89 | 1.12 | 1.49 |

Table 6: Minimum, average and maximum gains in the number of iterations required to reach a certain value of the residual $F$ in Newton's method. A value of $\infty$ means that the predicted initial condition was already below the given threshold. We observe that the gains are mostly obtained for large values of the residual.

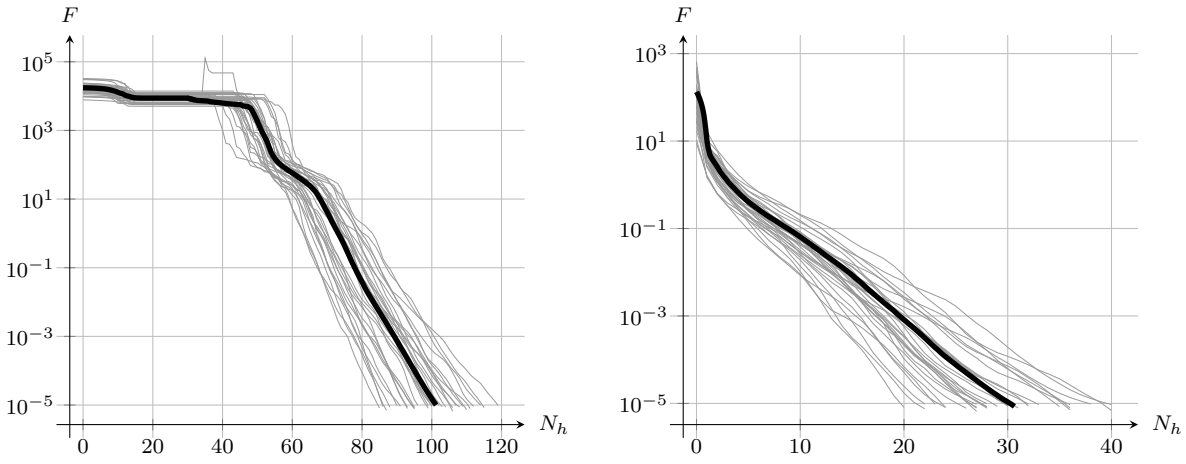
First, in Table 6, we collect the gains in number of iterations required to reach some value of

Newton's residual $F$. Most of the gains are obtained on the first phase of JFNK convergence, i.e. to help the residual reach a small enough value to trigger the convergence phase. This means that, compared with a conventional initialization, using the initial condition from the FNO successfully avoids the first few iterations of Newton's method, which exist to bring the residual to a small enough value. As a consequence, once the residual is small enough, e.g. smaller than 0.1, the new initialization no longer makes any difference as Newton's method has already entered its convergence phase. This illustrates why the gain is lower for finer meshes. In fact, the number of iterations below a residual of 0.1 is proportionally greater as the mesh is finer. These two phases (plateau and convergence), and the fact that using the predicted guess bypasses the plateau phase, are clearly visible on Figures 8 to 11, where we observe that reaching a residual of e.g. 10 is much faster when starting with the prediction rather than the naive guess.



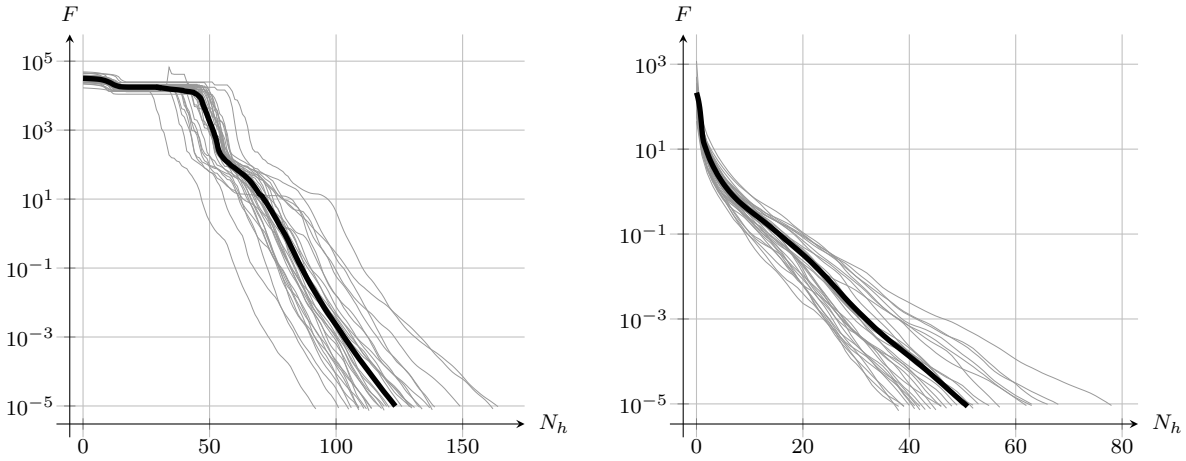

(a) classical Newton's method (b) enhanced Newton's method

Figure 8: Residual $F$ of the discretized PDE with respect to the Newton iterations for the classical Newton's method (left panel) and the enhanced Newton's method (right panel), for $N_h = 40^2$ points. The thin lines correspond to the different examples, while the thick line is an average over all examples.



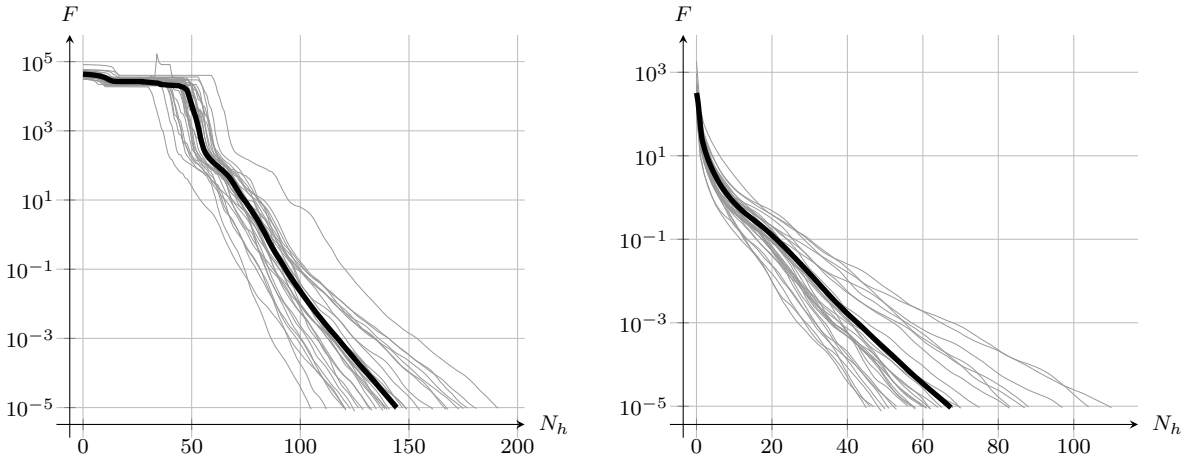

(a) classical Newton's method (b) enhanced Newton's method

Figure 9: Residual $F$ of the discretized PDE with respect to the Newton iterations for the classical Newton's method (left panel) and the enhanced Newton's method (right panel), for $N_h = 60^2$ points. The thin lines correspond to the different examples, while the thick line is an average over all examples.




(a) classical Newton's method (b) enhanced Newton's method

Figure 10: Residual $F$ of the discretized PDE with respect to the Newton iterations for the classical Newton's method (left panel) and the enhanced Newton's method (right panel), for $N_h = 80^2$ points. The thin lines correspond to the different examples, while the thick line is an average over all examples.




(a) classical Newton's method (b) enhanced Newton's method

Figure 11: Residual $F$ of the discretized PDE with respect to the Newton iterations for the classical Newton's method (left panel) and the enhanced Newton's method (right panel), for $N_h = 100^2$ points. The thin lines correspond to the different examples, while the thick line is an average over all examples.

This distribution of average gains also explains the more limited gains in computation time in 2D. Indeed, the initial iterations, when the residual is around 10, take a shorter time to complete than the subsequent ones. This is due to the fact the solution is close to being constant, resulting in a quick linear solve (during the linear stage of the JFNK). In contrast, the later iterations, where the residual is around 1, involve a solution further from being constant, thus making the associated linear solve more challenging. This means that, in 2D, the iterations saved by using the prediction are the fastest iterations to compute, which explains why the iteration gain is larger than the computation time gain. The difference with 1D is that, for some reason we do not quite grasp, the JFNK method converges much faster in 2D once the residual reaches around 1. In 1D, the residual needed to be closer to 0.1 to trigger the convergence phase. All in all, these observations explain why the gain in CPU time is lower than the gain in number of iterations in 2D, even though they were comparable

in 1D.

# 4 Conclusion

In this work, we tested the ability of neural operators, such as Fourier Neural Operators (FNOs), to predict a good initial guess for Newton's method applied to nonlinear elliptic partial differential equations (PDEs). To that end, the FNO was trained to predict the PDE solution on a large family of right-hand sides and diffusion coefficients, in one and two space dimensions. The solution predicted by the FNO was then used as initial guess in an iterative JFNK method. To increase the accuracy of the prediction, we used a discretization-informed loss function (containing information about the residual of the JFNK method applied to a discretization of the PDE) in addition to classical data-driven loss functions. A grid search algorithm was implemented to select the hyperparameters. Instead of using the prediction error as a selection tool in the grid search, we used a score function based on the gains, in terms of total number of iterations of the JFNK method, of using the FNO prediction rather than a naive initial guess. In all 1D and 2D test cases, the total number of iterations decreases when using the predicted guess, even in the case of strong nonlinearities or anisotropy. In the worst case, the average number of iterations was reduced by 82%; in the best one, the average number of iterations was reduced by 5000%. The initialization mainly speeds up the plateau phase of the JFNK convergence, rather than its convergence phase. As a consequence, the results namely depend on the grid: the finer the grid, the greater the number of iterations in the convergence phase, and the lesser the gains obtained with our initial guess. In any case, the approach saves time, especially as training is quick. The offline phase of this method (training the FNO) can be seen as the generation of a predictor, which will then be corrected online by the application of Newton's method, to accelerate the whole process.

In the future, an interesting extension will be tackle unstructured meshes. To that end, a possibility would be to use Geometry-Informed Neural Operators (GINOs), recently proposed in [30] for arbitrary geometries. Another possible direction, on the application side, concerns time-dependent strongly nonlinear problems, such as the MHD equations in Tokamaks, see [17], or fluid flow in porous media, for which a hybrid Newton's method was designed in [27].

# References


[1] J. Aghili, K. Brenner, J. Hennicker, R. Masson, and L. Trenty. Two-phase Discrete Fracture Matrix models with linear and nonlinear transmission conditions. *Int. J. Geomath.*, 10(1), 2019.

[2] H.-B. An, Z.-Y. Mo, and X.-P. Liu. A choice of forcing terms in inexact Newton method. *J. Comput. Appl. Math.*, 200(1):47–60, 2007.

[3] A. Anandkumar, K. Azizzadenesheli, K. Bhattacharya, N. Kovachki, Z. Li, B. Liu, and A. Stuart. Neural Operator: Graph Kernel Network for Partial Differential Equations. In *ICLR 2020 Workshop on Integration of Deep Neural Models and Differential Equations*, 2019.

[4] L. Bois, E. Franck, L. Navoret, and V. Vigon. A neural network closure for the Euler-Poisson system based on kinetic simulations. *Kinet. Relat. Models*, 15(1):49, 2022.

[5] K. Brenner. *On Global and Monotone Convergence of the Preconditioned Newton's Method for Some Mildly Nonlinear Systems*, pages 85–92. Springer Nature Switzerland, 2024.

[6] P. N. Brown and Y. Saad. Hybrid Krylov Methods for Nonlinear Systems of Equations. *SIAM J. Sci. Stat. Comput*, 11(3):450–481, 1990.

[7] S.-C. Cai and D. E. Keyes. Nonlinearly Preconditioned Inexact Newton Algorithms. *SIAM J. Sci. Comput.*, 24(1):183–200, 2002.

[8] G. Chen, L. Chacón, C. A. Leibs, D. A. Knoll, and W. Taitano. Fluid preconditioning for Newton–Krylov-based, fully implicit, electrostatic particle-in-cell simulations. *J. Comput. Phys.*, 258:555–567, 2014.

[9] H. Choi, S. D. Kim, and B.-C. Shin. Choice of an initial guess for Newton's method to solve nonlinear differential equations. *Comput. Math. Appl.*, 117:69–73, 2022.

[10] P. Degond, A. Lozinski, J. Narski, and C. Negulescu. An asymptotic-preserving method for highly anisotropic elliptic equations based on a Micro–Macro decomposition. *J. Comput. Phys.*, 231(7):2724–2740, 2012.

[11] F. Deluzet and J. Narski. A Two Field Iterated Asymptotic-Preserving Method for Highly Anisotropic Elliptic Equations. *Multiscale Model. Sim.*, 17(1):434–459, 2019.

[12] D. A. Di Pietro, É. Flauraud, M. Vohralík, and S. Yousef. A posteriori error estimates, stopping criteria, and adaptivity for multiphase compositional Darcy flows in porous media. *J. Comput. Phys.*, 276:163–187, 2014.

[13] D. A. Di Pietro, M. Vohralík, and S. Yousef. An a posteriori-based, fully adaptive algorithm with adaptive stopping criteria and mesh refinement for thermal multiphase compositional flows in porous media. *Comput. Math. Appl.*, 68(12):2331–2347, 2014.

[14] V. Dolean, M. J. Gander, W. Kheriji, F. Kwok, and R. Masson. Nonlinear Preconditioning: How to Use a Nonlinear Schwarz Method to Precondition Newton's Method. *SIAM J. Sci. Comput.*, 38(6):A3357–A3380, 2016.

[15] S. C. Eisenstat and H. F. Walker. Choosing the forcing terms in an inexact newton method. *SIAM J. Sci. Comput.*, 17(1):16–32, 1996.

[16] A. Ern and M. Vohralík. Adaptive Inexact Newton Methods with A Posteriori Stopping Criteria for Nonlinear Diffusion PDEs. *SIAM J. Sci. Comput.*, 35(4):A1761–A1791, 2013.

[17] E. Franck, M. Hölzl, A. Lessig, and E. Sonnendrücker. Energy conservation and numerical stability for the reduced MHD models of the non-linear JOREK code. *ESAIM: M2AN*, 49(5):1331–1365, 2015.

[18] J. N. Fuhg, A. Karmarkar, T. Kadeethum, H. Yoon, and N. Bouklas. Deep convolutional Ritz method: parametric PDE surrogates without labeled data. *Appl. Math. Mech. (English Ed.)*, 44(7):1151–1174, 2023.

[19] M. Geist, P. Petersen, M. Raslan, R. Schneider, and G. Kutyniok. Numerical Solution of the Parametric Diffusion Equation by Deep Neural Networks. *J. Sci. Comput.*, 88(1), 2021.

[20] M. A. Gomes-Ruggiero, V. L. R. Lopes, and J. V. Toledo-Benavides. A globally convergent inexact Newton method with a new choice for the forcing term. *Ann. Oper. Res.*, 157(1):193–205, 2007.

[21] M. et al Hoelzl. The JOREK non-linear extended MHD code and applications to large-scale instabilities and their control in magnetically confined fusion plasmas. *Nucl. Fusion*, 61(6):065001, 2021.

[22] J. Huang, H. Wang, and H. Yang. Int-Deep: A deep learning initialized iterative method for nonlinear problems. *J. Comput. Phys.*, 419:109675, 2020.

[23] G. E. Karniadakis, I. G. Kevrekidis, L. Lu, P. Perdikaris, S. Wang, and L. Yang. Physics-informed machine learning. *Nat. Rev. Phys.*, 3(6):422–440, 2021.

[24] S. D. Kim, E. Lee, and W. Choi. Newton's algorithm for magnetohydrodynamic equations with the initial guess from Stokes-like problem. *J. Comput. Appl. Math.*, 309:1–10, 2017.

[25] D. A. Knoll and D. E. Keyes. Jacobian-free Newton–Krylov methods: a survey of approaches and applications. *Journal of Computational Physics*, 193(2):357–397, 2004.

[26] N. Kovachki, Z. Li, B. Liu, K. Azizzadenesheli, K. Bhattacharya, A. Stuart, and A. Anandkumar. Neural Operator: Learning Maps Between Function Spaces With Applications to PDEs. *J. Mach. Learn. Res.*, 24:1–97, 2023.

[27] A. Lechevallier, S. Desroziers, T. Faney, É. Flauraud, and F. Nataf. Hybrid Newton method for the acceleration of well event handling in the simulation of CO2 storage using supervised learning. working paper or preprint, 2023.

[28] Z. Li, N. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart, and A. Anandkumar. Multipole Graph Neural Operator for Parametric Partial Differential Equations. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS'20, Red Hook, NY, USA, 2020. Curran Associates Inc.

[29] Z. Li, N. B. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart, and A. Anandkumar. Fourier Neural Operator for Parametric Partial Differential Equations. In *International Conference on Learning Representations*, 2021.

[30] Z. Li, N. B. Kovachki, C. Choy, B. Li, J. Kossaifi, S. P. Otta, M. A. Nabian, M. Stadler, C. Hundt, K. Azizzadenesheli, and A. Anandkumar. Geometry-Informed Neural Operator for Large-Scale 3D PDEs. *arXiv preprint arXiv:2309.00583*, 2023.

[31] Z. Li, H. Zheng, N. Kovachki, D. Jin, H. Chen, B. Liu, K. Azizzadenesheli, and A. Anandkumar. Physics-Informed Neural Operator for Learning Partial Differential Equations. *arXiv preprint arXiv:2111.03794*, 2023.

[32] Z. Long, Y. Lu, X. Ma, and B. Dong. PDE-net: Learning PDEs from data. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 3208–3216. PMLR, 2018.

[33] 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. *Nat. Mach. Intell.*, 3(3):218–229, 2021.

[34] L. Luo and X.-C. Cai. PIN$^{\updownarrow}$ : Preconditioned Inexact Newton with Learning Capability for Nonlinear System of Equations. *SIAM J. Sci. Comput.*, 45(2):A849–A871, 2023.

[35] R. Maulik, B. Lusch, and P. Balaprakash. Reduced-order modeling of advection-dominated systems with recurrent neural networks and convolutional autoencoders. *Phys. Fluids*, 33(3), 2021.

[36] P. Novello, G. Poëtte, D. Lugato, S. Peluchon, and P. M. Congedo. Accelerating hypersonic reentry simulations using deep learning-based hybridization (with guarantees). *J. Comput. Phys.*, 498:112700, 2024.

[37] A. Odot, R. Haferssas, and S. Cotin. DeepPhysics: A physics aware deep learning framework for real-time simulation. *Int. J. Numer. Meth. Eng.*, 123(10):2381–2398, 2022.

[38] R. P. Pawlowski, J. N. Shadid, J. P. Simonis, and H. F. Walker. Globalization Techniques for Newton–Krylov Methods and Applications to the Fully Coupled Solution of the Navier–Stokes Equations. *SIAM Rev.*, 48(4):700–721, 2006.

[39] J. Qu, W. Cai, and Y. Zhao. Learning time-dependent PDEs with a linear and nonlinear separate convolutional neural network. *J. Comput. Phys.*, 453:110928, 2022.

[40] 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. *J. Comput. Phys.*, 378:686–707, 2019.

[41] H. Tang, S. Wang, C. Yin, Y. Di, Y.-S. Wu, and Y. Wang. Fully-Coupled Multi-Physical Simulation with Physics-Based Nonlinearity-Elimination Preconditioned Inexact Newton Method for Enhanced Oil Recovery. *Commun. Comput. Phys.*, 25(1), 2019.

[42] B. van Es, B. Koren, and H. J. de Blank. Finite-difference schemes for anisotropic diffusion. *J. Comput. Phys.*, 272:526–549, 2014.

[43] S. Wang, H. Wang, and P. Perdikaris. Learning the solution operator of parametric partial differential equations with physics-informed DeepONets. *Sci. Adv.*, 7(40), 2021.

[44] Ph. Wolfe. Convergence Conditions for Ascent Methods. *SIAM Rev.*, 11(2):226–235, 1969.