PNAS Nexus, 2023, pp. 1–17

doi: DOI HERE
Advance Access Publication Date: Day Month Year
Paper

PAPER

Solving inverse problems in physics by optimizing a discrete loss: fast and accurate learning without neural networks

Petr Karnakov,[a] Sergey Litvinov[a] and Petros Koumoutsakos[a*]

[a] Computational Science and Engineering Laboratory, Harvard John A. Paulson School of Engineering and Applied Sciences, 29 Oxford St, 02138, MA, United States
*To whom correspondence should be addressed: petros@seas.harvard.edu
FOR PUBLISHER ONLY Received on Date Month Year; accepted on Date Month Year

Abstract

In recent years, advances in computing hardware and computational methods, have prompted a wealth of activities for solving inverse problems in physics. These problems are often described by systems of partial differential equations (PDEs). The advent of machine learning has re-invigorated the interest in solving inverse problems using neural networks (NNs). In these efforts, the solution of the PDEs are expressed as NNs trained through the minimization of a loss function involving the PDE. Here we show how to accelerate this approach by five orders of magnitude by deploying, instead of NNs, conventional PDE approximations. The framework of Optimizing a Discrete Loss (ODIL) minimizes a cost function for discrete approximations of the PDEs using gradient-based and Newton methods. The framework relies on grid-based discretizations of PDEs and inherits their accuracy, convergence, and conservation properties. The implementation of the method is facilitated by adopting machine learning tools for automatic differentiation. We also propose a multigrid technique to accelerate the convergence of gradient-based optimizers. We present applications to PDE-constrained optimization, optical flow, system identification, and data assimilation. We compare ODIL with the popular method of Physics Informed Neural Networks (PINNs) and show that it outperforms it by several orders of magnitude in computational speed while having better accuracy and convergence rates. We evaluate ODIL on inverse problems involving linear and nonlinear PDEs including the Navier-Stokes equations for flow reconstruction problems. ODIL bridges numerical methods and machine learning and presents a powerful tool for solving challenging, inverse problems across scientific domains.

Key words: partial differential equations, inverse problems, physics-informed neural networks, computational fluid dynamics

Significance statement
Inverse problems are fundamental in science and engineering, often revealing solutions unattainable through direct methods. Inspired by efforts to solve such problems using physics neural networks (PINNs), we present the framework of Optimizing a Discrete Loss (ODIL). ODIL deploys conventional discretizations on a grid instead of neural networks. ODIL is a drop-in replacement for PINNs in many scientific and engineering problems, while being several orders of magnitude more efficient in terms of computational cost and achieving higher accuracy.

Introduction

The numerical solution of physical models expressed as partial differential equations (PDEs) is indispensable across all areas of science, engineering and medicine. Commonly solutions are presented for the so called forward problems where the PDEs are discretized on grids or particles [38, 41, 78] and the discrete problems are solved to determine quantities of interest given initial and/or boundary conditions. In recent years the the advent of data science has prompted intense interest in solving PDEs in the context of "inverse problems" where the parameters and even the structure of PDE models are inferred from data [14, 17, 27, 35]. Solving inverse problems in science and engineering entails formidable challenges as the PDEs need to be solved while observing physical and geometrical constraints as well as noisy data [12]. Such problems are encountered in several fields of science and engineering and they have been handled by methods such as PDE-constrained optimization [31], data assimilation [42], optical flow in computer vision [24], and system identification [46]. As there is an ever increasing number of problems that involve noisy or missing data there is a strong need to develop efficient methods for solving such problems and machine learning has emerged as a potent modality [5, 35, 73], albeit with certain limitations [12].

Physics-informed learning using neural networks (NNs) for partial and ordinary differential equations were introduced three decades ago, by Dissanayake et al. [20], van Milligen et al. [71], and Lagaris et al. [40]. In these works the unknown fields were represented by NNs with weights obtained by minimizing a loss function composed by the residuals of the governing equations. Related efforts used NNs for modeling nonlinear dynamical systems [29, 65], back-tracking of an N-body system [61] and (possibly the first ever) deep NNs for reconstructing the flow field in the near-wall region of a turbulent flow using wall only information [51]. At the time of their introduction these methods were not favored for solving forward or inverse problems, in particular due to their computational cost and the speed of available computing architectures. Twenty years later, the method was revived by Raissi et al. [63] who used modern machine learning methods and software (such as deep neural networks, automatic differentiation, and TensorFlow) to carry out its operations. The method has been popularized by the term physics-informed neural network (PINN).

PINNs have created a lot of enthusiasm in the scientific community as they offered an alternative to solving complex physical problems using convenient representations at a significantly reduced cost of implementation. However, it is broadly accepted that PINNs cannot match conventional numerical methods for solving well-posed forward problems involving PDEs but they are positioned as a convenient tool for solving ill-posed and inverse problems and data-driven modeling [35]. The proper assessment of this claim hindered by the fact that to date there is no baseline for assessing the capabilities of PINNs with respect to conventional methods for solving forward and inverse problems. The authors in [63] admit that conventional numerical solvers are more efficient for forward problems, a direct comparison with such solvers reveals important drawbacks of the neural network approach [6].

The overall computational cost of PINNs originates from the cost of evaluating the solution in each collocation point. Represented by a fully connected network, the solution in each point depends on all weights of the neural network, therefore the cost of evaluating it in one point is proportional to the number of weights. In contrast to neural networks, conventional grid-based methods represent the solution with local approximations resulting in a constant cost per point. Consequently, the gradient of the residual with respect to the weights in one point is a dense vector and the Hessian of the loss function a dense matrix. This makes higher order optimization methods such as Newton's method unsuitable for training NNs. Furthermore, the convenience of representation and ease of formalism by PINNs comes with limitations. While PINN evaluates the differential operator exactly on a set of collocation points, it makes no allowance for physically and numerically motivated adjustments that speed up convergence and ensure discrete conservation such as deferred correction, upwind schemes, and discrete conservation laws as represented, for example, in the finite volume method. Furthermore, decades of development and analysis that went into conventional solvers enables understanding, prediction, and control of their convergence and stability properties. Such information is not available with neural networks and recent works aim to remedy this situation [39, 53]. Application of PINN to equations with high-order derivatives is also limited [49, 55] as the computational cost of nested automatic differentiation is exponential in the order of differentiation [8, 10], while in the case of finite differences, the cost is linear.

More recently, the architectures deployed in PINNs have undergone significant enhancements, including the introduction of positional encodings, exact enforcement of boundary conditions, incorporation of adaptive activation functions, and improving the treatment of constraints and invariances [7, 21, 32, 50, 68]. While these techniques significantly improve the performance in certain cases, they do not guarantee a universal improvement in accuracy or training speed. When it comes to studies involving PINNs, it is generally not advisable to start with these techniques, considering the numerous hyperparameters that already require tuning. For instance, there are cases where enforcing exact boundary conditions can lead to worse performance. In such scenarios, a more effective approach may involve blending the boundary conditions into the approximation, even if it diverges from the reference solution. The same principle applies to positional encodings, which entail adding sine waves to the input. While this approach clearly boosts performance in scenarios like the wave equation with periodic boundary conditions, where the solution is a linear combination of sine waves, it may not yield the same benefits for more general conditions. Nevertheless, in our experimentation with PINNs, we applied several well-known algorithmic modifications. Our observations did not indicate any noteworthy enhancements, consistent with similar experiences by other groups (for example, [15, 22, 30]). Moreover, the effects of these modifications were similar to the variations observed when changing the random seed (see appendix). Our observations align with a recent paper authored by one of the co-creators of PINNs, wherein the attempt to solve the $\mathrm{Re} = 2000$ lid-driven cavity flow problem not only was unsuccessful but also resulted in several non-physical, pseudo-solutions [76].

Here we present a framework that is based on conventional numerical methods that overcomes many of the challenges encountered by PINNs. We show that this valuable approach can be accelerated by five orders of magnitude by deploying conventional numerical algorithms that are facilitated by machine learning tools. We cast the framework as Optimizing a Discrete Loss (ODIL) by combining discrete formulations of PDEs with modern machine learning tools to extend their scope to ill-posed and inverse problems.

The framework of Optimizing a Discrete Loss (ODIL), minimizes a cost function for the discrete forms of PDEs using gradient-based and Newton methods. We demonstrate that ODIL is effective in solving inverse problems for equations with missing parameters and with limited data. This framework has two key aspects. First, the discretization itself, that defines the accuracy, stability and consistency of the method. Second, the optimization algorithm to solve the discrete problem. Our method uses efficient computational tools that are readily available for both of these aspects. The discretization properties are inherited from the conventional numerical methods building upon the advances in this field over multiple decades. Since the sparsity of the problem is preserved, the optimization algorithm can use a Hessian and achieve a quadratic converge rate, which remains out of reach for gradient-based training of neural networks. We remark that solving the discrete equations as a minimization problem is not a new idea. It has been developed in various forms in different domains, including the discretize-then-differentiate approach in the context of PDE-constrained optimization [31], as the penalty method [70], and is related to the 4D-VAR problem in variational data assimilation [42, 66], in addition to regularization techniques [33]. Differentiable solvers [2, 43, 44, 74] serve as a tool for computing the gradients of the solution with respect to parameters of the problem. They can be combined with a gradient-based optimization method to fit the solution to known data and infer the parameters. However, they assume that the forward problem is well-posed, which is not always the case as demonstrated by the "notorious test problem" in optimal control [11, 13] discussed in Supplementary Fig. S4.

Another related approach is the the neural bootstrapping method (NBM) [54, 55]. NBM represents the solution with a neural network as in PINN, but evaluates conventional discretization schemes on a set of random collocation points using a local stencil around each point. The authors have applied this approach to elliptic problems with discontinuities. Although NBM does not reduce the cost of evaluating the solution in each point introduced by a fully-connected NN, it can be less computationally expensive than PINN if the equations contain high-order derivatives. The cost of nested automatic differentiation in PINN grows exponentially with the order of the derivative, but NBM uses finite differences for which the cost is linear. While the authors do not provide a direct comparison with conventional solvers in terms of the accuracy and computational cost, they use small neural networks (e.g. 1000 parameters) and evaluate the loss on a grid with significantly more cells (e.g. 128^3 cells), suggesting that NBM is more efficient than conventional solvers in terms of the number of required degrees of freedom. However, the presented test cases involve smooth functions with only low-frequency modes (e.g. products of sinusoidals). Given the phenomenon of spectral bias [62] of NNs, more complex solutions will require larger networks. Moreover, the authors do not demonstrate applications of NBM to inverse problems, which is the primary domain of ODIL. In a related work, blended inverse-PDE networks (BiPDE) [60] incorporate a differentiable PDE solver as a layer in a NN with the goal of solving inverse problems for parameterized PDEs. BiPDE is an autoencoder that takes a solution field, passes it through an encoder to the unknown hidden parameters, and uses the parameters as input for the PDE solver layer, which outputs the solution. The autoencoder is trained on data obtained by solving the forward problem with random parameters sampled from a given distribution. The trained encoder can be used for finding the parameters from a known solution field, possibly perturbed by noise. We remark that, unlike BiPDE, ODIL includes both the residual of the discretized PDEs and known data into the loss function and obtains the solution and the parameters by minimizing the total loss.

In the following we present a potent computational framework for solving forward and inverse problems with missing and noisy data and facilitate its deployment by adopting machine learning tools. Adopting modern machine learning tools, such as automatic differentiation, renders the implementation of ODIL as convenient as applying gradient-based methods in PINNs, providing a baseline for the comparison of these methods. We present a number of solutions to prototypical and benchmark problems to evaluate the performance of ODIL. Our results indicate that ODIL outperforms PINNs by up to five orders of magnitude in computational speed and with better accuracy while covering the same class of inverse and ill-posed problems. Finally, we note that ODIL is an example of bridging numerical methods and machine learning. The successful fusion of these two domains with complementary potentials can create further developments of computational methods for challenging scientific and engineering problems described by PDEs.

Methods

Formulation of ODIL

Here we formulate the ODIL framework to solve forward and inverse problems for PDEs. Using a given discretization of the PDEs, we construct a loss function from the residuals of the discretization, data terms, and regularization terms. The applications in this paper involve discretizations on a uniform Cartesian grid, but this general formulation is applicable to other cases, e.g. an unstructured triangular grid. Let C denote the cell indices of a grid in a d-dimensional space $\mathbb{R}^d$ with the number of cells $N_C = |C|$. Consider a system of N_F discrete equations, N_u unknown fields, and N_θ parameters

$$F^{(i)}[\mathbf{u}, \theta] = 0, \quad i = 1, \dots, N_F, \tag{1}$$

where $F^{(i)} : (\mathbb{R}^C)^{N_u} \times \mathbb{R}^{N_\theta} \to \mathbb{R}^C$, $i = 1, \dots, N_F$ are the residual operators of the discrete equations, $\mathbf{u} = (u^{(1)}, \dots, u^{(N_u)})$, are the unknown discrete fields $u^{(j)} \in \mathbb{R}^C$, and $\theta \in \mathbb{R}^{N_\theta}$ is the vector of parameters. These discrete equations can represent discretizations of the governing PDEs (e.g. a finite difference scheme), equality constraints to impose known data (e.g. measurements of the solution in a finite set of points), and regularization terms (e.g. discrete Laplace operator as a smoothness constraint). The parameters θ include all unknowns that are not discrete fields (e.g. weights of neural networks used to represent nonlinear terms). The relevant boundary and initial conditions can either be incorporated into the discretization of the PDEs or added as extra equality constraints. The ODIL framework solves the unconstrained minimization problem

$$\min_{\mathbf{u}, \theta} \mathcal{L}(\mathbf{u}, \theta) \tag{2}$$

with the loss function

$$\mathcal{L}(\mathbf{u}, \theta) = \sum_{i=1}^{N_F} \left[\frac{1}{N_C} \sum_{c \in C} \left(F_c^{(i)}[\mathbf{u}, \theta] \right)^2 \right], \tag{3}$$

where $\cdot_c$ denotes the value of a field in cell $c \in C$.

We consider two approaches to solving this minimization problem. The first approach is to use a standard gradient-based

method, such as Adam [37] or L-BFGS-B [77], with the gradients of $\mathcal{L}(\mathbf{u}, \theta)$ computed using automatic differentiation [1]. One important enhancement of this approach is the multigrid decomposition technique to accelerate the convergence, which is described further below.

The second approach is the Gauss-Newton method, which iteratively solves the problem by updating the solution based on linearization of the residual operators about the current solution. Let $(\mathbf{u}^s, \theta^s)$ denote the solution at iteration s. Define the linear approximations to the operators $F^{(i)}$ about the point $(\mathbf{u}^s, \theta^s)$

$$\tilde{F}^{(i),s}[\mathbf{u}, \theta] = F^{(i)}[\mathbf{u}^s, \theta^s] + \left.\frac{\partial F^{(i)}}{\partial \mathbf{u}}\right|^s [\mathbf{u} - \mathbf{u}^s] + \left.\frac{\partial F^{(i)}}{\partial \theta}\right|^s [\theta - \theta^s], \quad (4)$$

where $\cdot|^s$ indicates that the Jacobian is evaluated at $(\mathbf{u}^s, \theta^s)$. Using the linearized operators, define the approximate loss function

$$\mathcal{L}^s(\mathbf{u}, \theta) = \sum_{i=1}^{N_F} \left[\frac{1}{N_C} \sum_{c \in C} \left(\tilde{F}_c^{(i),s}[\mathbf{u}, \theta] \right)^2 \right], \quad (5)$$

which is now a quadratic function of $\mathbf{u}$ and θ. A minimum of this function provides the solution at the next iteration

$$(\mathbf{u}^{s+1}, \theta^{s+1}) = \arg\min_{(\mathbf{u}, \theta)} \mathcal{L}^s(\mathbf{u}, \theta). \quad (6)$$

On the other hand, a minimum of $\mathcal{L}^s$ satisfies the optimality conditions

$$\begin{aligned} \nabla_{\mathbf{u}} \mathcal{L}^s(\mathbf{u}^{s+1}, \theta^{s+1}) &= 0, \\ \nabla_{\theta} \mathcal{L}^s(\mathbf{u}^{s+1}, \theta^{s+1}) &= 0, \end{aligned} \quad (7)$$

which is a system of linear equations. To derive the system, we combine all unknowns in one column vector

$$\mathbf{U} = (\mathbf{u}, \theta) \in \mathbb{R}^{N_u N_C + N_\theta}, \quad (8)$$

all residuals in one column vector

$$\mathbf{F} = (F^{(1)}, \dots, F^{(N_F)}) \in \mathbb{R}^{N_F N_C}, \quad (9)$$

and the corresponding derivatives in one matrix

$$\mathbf{A} = \begin{bmatrix} \frac{\partial F^{(1)}}{\partial \mathbf{u}} & \frac{\partial F^{(1)}}{\partial \theta} \\ \vdots & \vdots \\ \frac{\partial F^{(N_F)}}{\partial \mathbf{u}} & \frac{\partial F^{(N_F)}}{\partial \theta} \end{bmatrix} \in \mathbb{R}^{(N_F N_C) \times (N_u N_C + N_\theta)}. \quad (10)$$

In this notation, the linearized operators (4) take the form

$$\tilde{\mathbf{F}}^s[\mathbf{U}] = \mathbf{F}^s + \mathbf{A}^s(\mathbf{U} - \mathbf{U}^s), \quad (11)$$

where $\mathbf{F}^s = \mathbf{F}[\mathbf{U}^s]$. The approximate loss function (5) becomes

$$\mathcal{L}^s(\mathbf{U}) = \frac{1}{N_C} \left\| \tilde{\mathbf{F}}^s[\mathbf{U}] \right\|_2^2 = \frac{1}{N_C} \left\| \mathbf{F}^s + \mathbf{A}^s(\mathbf{U} - \mathbf{U}^s) \right\|_2^2. \quad (12)$$

Finally, the optimality conditions (7) are now equivalent to

$$(\mathbf{A}^s)^* \mathbf{A}^s (\mathbf{U}^{s+1} - \mathbf{U}^s) = -(\mathbf{A}^s)^* \mathbf{F}^s. \quad (13)$$

Solving this linear system for $\mathbf{U}^{s+1}$ provides the iteration update. To solve the resulting sparse linear system at each iteration we use a direct method [18, 72] or an algebraic multigrid method [9]. Iterative methods require an initial guess to start the iteration. Unless stated otherwise, we use a zero initial guess for the discrete fields in all presented applications. We note that if the discretization is linear and the corresponding optimization problem is quadratic, the Gauss-Newton method converges to the solution after one iteration from any initial guess. If a problem involves a neural network, the initial weights are sampled from a uniform distribution and the initial biases are zero. For brevity, we refer to the above Gauss-Newton method as simply Newton's method throughout the paper.

Calculating sparse Jacobians

Discrete equations (1) that result from approximations of PDEs are often local, i.e. the residual in each cell depends on values of unknown fields in a limited number of neighboring cells. In such cases, the Jacobians with respect to the unknown fields in (10) are sparse. To calculate them, we intend to use automatic differentiation of scalar functions. Coloring techniques [16, 58] can be used to estimate a sparse Jacobian using multiple evaluations of a Jacobian-vector product. However, we use a more explicit approach that directly tracks the dependencies on the neighboring cells. For simplicity, assume that $N_u = 1$. Cases with $N_u > 1$ are handled by extension. To calculate the Jacobian of a residual operator $F : \mathbb{R}^C \times \mathbb{R}^{N_\theta} \to \mathbb{R}^C$, we assume that the operator explicitly depends on the field values in K neighboring cells

$$F[u, \theta] = g[S^{(1)}[u], \dots, S^{(K)}[u], \theta], \quad (14)$$

where $g : (\mathbb{R}^C)^K \times \mathbb{R}^{N_\theta} \to \mathbb{R}^C$ is an element-wise operator of its field arguments, i.e. its value $g_c[v^{(1)}, \dots, v^{(K)}, \theta]$ in each cell $c \in C$ only depends on $(v_c^{(1)}, \dots, v_c^{(K)}, \theta)$. Here $S^{(k)} : \mathbb{R}^C \to \mathbb{R}^C$, $k = 1, \dots, K$ are shift operators selecting values from the neighboring cells or possibly the identity operator. On a Cartesian grid, a discrete field is a d-dimensional array and a shift operator simply shifts the array. On an unstructured triangular grid, a shift operator selects adjacent triangles. Then we introduce a scalar function

$$\mathcal{G}(v^{(1)}, \dots, v^{(K)}, \theta) = \sum_{c \in C} g_c(v^{(1)}, \dots, v^{(K)}, \theta) \quad (15)$$

and calculate the desired Jacobian as

$$\frac{\partial F}{\partial u} = \sum_{k=1}^{K} \frac{\partial \mathcal{G}}{\partial v^{(k)}} \odot S^{(k)}, \quad (16)$$

where the derivatives of $\mathcal{G}$ are evaluated at $v^{(k)} = S^{(k)}[u]$ and $\odot$ is the element-wise product. Note that (16) expresses the Jacobian through K derivatives of the scalar function $\mathcal{G}$ as intended.

Multigrid decomposition

Here we propose a multigrid technique to accelerate the convergence of optimization methods for problems that involve discrete fields on a grid. Multigrid methods are generally accepted as the fastest numerical methods for solving elliptic differential equations [69]. A standard multigrid method consists of the following parts: a hierarchy of grids including the original grid and coarser grids, discretizations of the problem on each grid, interpolation operators to finer levels, and restriction operators to coarser levels. The method iteratively updates the solution on each level, interpolates the update to finer levels, and restricts the residuals to coarser levels.

As discussed above, the optimization problem in ODIL can be solved with Newton's method which involves a linear system at each step, so the multigrid method can be applied directly to that linear system. However, then we need to construct the linear system explicitly and rely on a general multigrid solver which may have limited efficiency especially on GPUs [56]. Furthermore, if the equation involves unknown parameters, such as parameters of a neural network, as demonstrated on the problem of inferring conductivity from temperature, in Results, the corresponding Jacobian matrix is dense, which makes Newton's method impractical.

Instead of Newton's method, we can solve the problem with a general optimization algorithm, such as Adam or L-BFGS. However, as we demonstrate on the lid-driven cavity flow in Results, they require orders of magnitude more iterations than Newton's method. To accelerate the convergence, we propose the following technique. Consider a uniform grid with $N_1 = N$ points in each direction. Introduce a hierarchy of successively coarser grids of size $N_i = N/2^{i-1}$ for $i = 1, \dots, L$, where L is the total number of levels. Define the multigrid decomposition operator as

$$M_L[u_1, \dots, u_L] = u_1 + T_1 u_2 + \cdots + T_1 T_2 \dots T_{L-1} u_L, \quad (17)$$

where each u_i is a field on grid N_i, and each T_i is an interpolation operator from grid N_{i+1} to the finer grid N_i. The multigrid decomposition of a discrete field u on a grid of size N reads

$$u = M_L[u_1, \dots, u_L]. \quad (18)$$

Note that this representation is over-parameterized and therefore not unique. The total number of scalar parameters increases from N^d of the original field u to $N_1^d + \cdots + N_L^d$ for the representation $u_1, \dots, u_L$. We define the interpolation operators T_i using linear interpolation [69]. The rest of the framework remains the same, including the discretized PDEs and the optimization algorithm. Gradients of the resulting loss function can be computed using automatic differentiation. This technique addresses the issue of locality of gradient-based optimizers by extending the domain of dependence of each scalar parameter so that information can propagate through the grid faster. Supplementary Fig. S3 demonstrates the speedup on the lid-driven cavity flow. We refer to [34] for further details and examples related to the multigrid decomposition technique. All applications of ODIL with a gradient-based optimizer presented below in Results use this technique. The only exception is the wave equation, where we intend to compare ODIL and PINN using standard optimization methods without enhancements.

Results

Wave equation. Accuracy and cost of PINN

We apply ODIL and PINN to solve an initial-value problem for the wave equation in two dimensions and compare both methods in terms of accuracy and computational cost. The problem consists of the wave equation

$$u_{tt} = u_{xx} \quad (19)$$

with the initial conditions and boundary conditions

$$\begin{aligned} u(x,0) &= U(x,0) \\ u_t(x,0) &= U_t(x,0) \\ u(-1,t) &= U(-1,t) \\ u(1,t) &= U(1,t) \end{aligned} \quad (20)$$

in a rectangular domain $(x,t) \in \Omega = [-1,1] \times [0,1]$, where $u(x,t)$ is the unknown function and $U(x,t)$ is the exact solution

$$U(x,t) = \frac{1}{10} \sum_{k=1}^{5} \big(\cos(x - t + 0.5)\pi k + \cos(x + t + 0.5)\pi k \big). \quad (21)$$

Both methods reduce the problem to minimization of a loss function. ODIL represents the solution on a uniform Cartesian grid of size $N_x \times N_t$ with cell centers $x_i = -1 + (i+0.5)\Delta x$, $i = 1, \dots, N_x$ and $t^n = (n + 0.5)\Delta t$, $n = 1, \dots, N_t$, where $\Delta x = 2/N_x$ and $\Delta t = 1/N_t$. A second-order finite volume discretization yields the following residual operator

$$F_i^n[u] = \frac{u_i^n - 2u_i^{n-1} + u_i^{n-2}}{\Delta t^2} - \frac{u_{i+1}^{n-1} - 2u_i^{n-1} + u_{i-1}^{n-1}}{\Delta x^2} \quad (22)$$

for $n > 2$ and

$$\begin{aligned} F_i^2[u] &= \frac{u_i^2 - u_i^1}{\Delta t^2} - \frac{U_t(x_i,0)}{\Delta t} - \frac{u_{i+1}^1 - 2u_i^1 + u_{i-1}^1}{\Delta x^2} \\ F_i^1[u] &= U(x_i,0) + U_t(x_i,0) - u_i^1 \end{aligned} \quad (23)$$

for $n = 1, 2$ derived using the initial conditions. The boundary conditions enter the discretization through values in the halo cells set to

$$\begin{aligned} u_0^n &= \frac{1}{3}(u_2^n - 6u_1^n + 8U(-1,t^n)) \\ u_{N_x+1}^n &= \frac{1}{3}(u_{N_x-1}^n - 6u_{N_x}^n + 8U(1,t^n)) \end{aligned} \quad (24)$$

using second-order extrapolation. ODIL solves the minimization problem for the loss function (3) computed from this residual operator. We choose $N_x = N_t$ and define the number of parameters of ODIL as $N = N_x N_t$. PINN represents the solution as a fully-connected neural network and evaluates the residuals of the differential equation exactly on a finite set of collocation points [63]. The network consists of two equally sized hidden layers with tanh activation functions. We define the number of parameters N of PINN as the total number of weights. The number of collocation points for PINN amounts to 8192 points inside the domain and 768 points for the initial and boundary conditions. The optimization problem is solved with L-BFGS-B or Newton for ODIL and with L-BFGS-B for PINN. Here we do not use the multigrid decomposition technique of ODIL.

Figure 1 presents examples of the solutions obtained using both methods and also shows how the error and execution time of both methods depend on the number of parameters. The results of PINN include 25 runs with random initial weights and positions of the collocation points. In the case of ODIL, solutions from two optimization methods, L-BFGS-B and Newton, coincide. For each run of PINN or ODIL with L-BFGS-B, we determine the best error achieved during 80000 epochs and estimate the required number of epochs as the number of epochs to achieve 150% of the best error. Then we estimate the execution time per epoch as the mean over the last 100 epochs for L-BFGS-B and one epoch for Newton's method, excluding the time spent on startup, building the computational graph, and output. The measurements are performed on one CPU core of Intel Xeon E5-2690 v3 and exclude the startup time spent on building the computational graph required on the first epoch. We report the execution time as the product of the required number of epochs and the execution time per epoch. Both methods demonstrate similar accuracy. The error of ODIL scales as

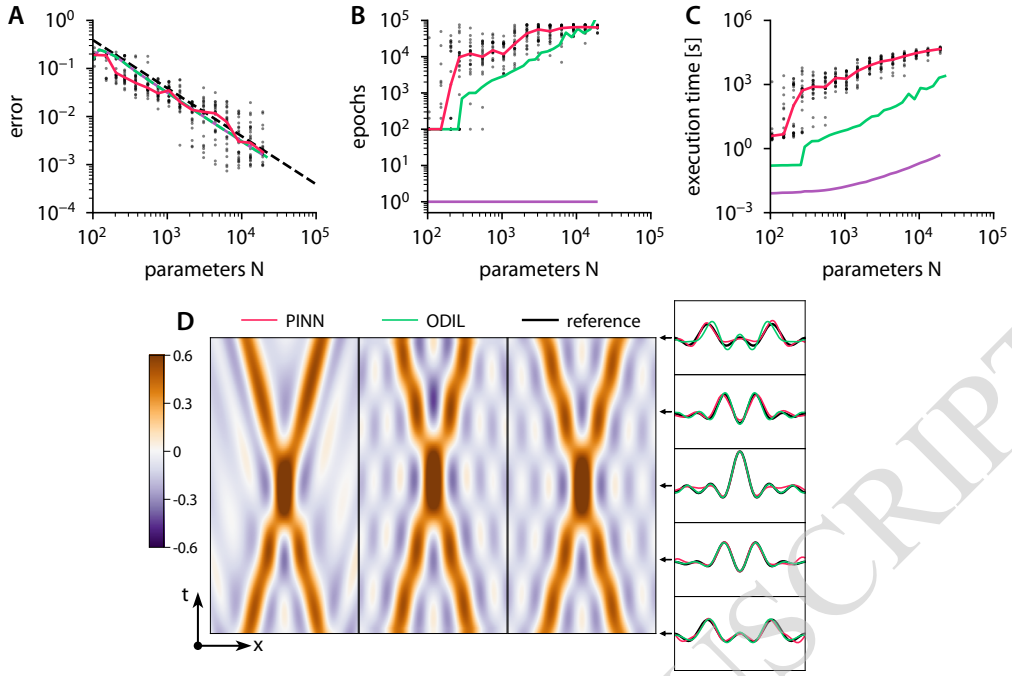

Figure 1. Wave equation solved using PINN and ODIL. A, B, C) Root-mean-square error relative to the exact reference solution (A), required number of epochs (B), and execution time on one CPU core (C) of various methods: PINN with L-BFGS-B ——, ODIL with L-BFGS-B ——, ODIL with Newton ——, and line with slope $1/N$ – – –·. For PINN, the dots show samples with random initial weights and collocation points and solids lines show the median value. The number of parameters is the number of weights for PINN and the number of grid cells for ODIL. D) Solution obtained using PINN with L-BFGS-B —— with two hidden layers of 25 neurons, ODIL with Newton on a grid of 25 × 25 cells ——, and exact reference solution ——.

$1/N$ corresponding to a second-order accurate discretization. The error of PINN is significantly scattered, but the median error also scales as $1/N$. In addition, we refer to Supplementary Fig. S5 that shows the effect of weights initialization and demonstrates how PINN behaves on an asymmetric solution. The execution times of the methods, however, are drastically different. With $N \approx 19000$ parameters, PINN with L-BFGS-B takes 13 hours, ODIL with L-BFGS-B takes 35 minutes, and ODIL with Newton takes only 0.5 seconds, which is about 100000 times faster than PINN. We note that running the same test on a GPU would reduce the timings of PINN since it creates a significant workload for a GPU. However, the timings obtained here on one CPU core serve as an estimate of the overall computational cost of each method, i.e. the number of required floating point operations.

Poisson equation in two dimensions

We apply ODIL and PINN to solve the boundary-value problem for the Poisson equation

$$u_{xx} + u_{yy} = f(x, y) \tag{25}$$

with zero Dirichlet boundary conditions in a unit domain $(x, y) \in [0, 1]^2$. The exact reference solution is an oscillatory function

$$u(x, y) = \sin(\pi(kx)^2) \sin \pi y \tag{26}$$

with $k = 2$ or $k = 4$ and the right-hand side $f(x, y)$ is found by substituting the exact solution into the equation, i.e. using the method of manufactured solutions. ODIL solves the problem on a grid of 64 × 64 cells. PINN uses a neural network of size 32 × 32 × 32 with 400 points on the boundaries and 1000 collocation points inside the domain, which are resampled every 100 iterations. Both methods use the Adam optimizer with the learning rate set to 0.001 for PINN and 0.005 for ODIL. To implement the PINN method, we use the DeepXDE framework [47].

Figure 2 shows results of both methods compared to the reference solution along with the convergence history. The error is measured relative to the exact solution. In the case $k = 2$, both methods recover the reference solution achieving an error of 0.02 for PINN and 0.002 for ODIL. However, PINN takes about 100 times more iterations to achieve the same error. In the case $k = 4$, PINN fails to recover the reference solution after 500000 epochs, while ODIL achieves an error of 0.1.

Velocity from tracer

We consider the evolution of a tracer field governed by the advection equation in two dimensions. The problem is to find a velocity field $\mathbf{u}(x, y)$ given that the tracer field $c(x, y, t)$ satisfies the advection equation

$$\frac{\partial c}{\partial t} + \mathbf{u} \cdot \nabla c = 0 \tag{27}$$

in a unit domain, and the initial $c(x, y, 0) = c_0(x, y)$ and final $c(x, y, 1) = c_1(x, y)$ profiles are known. The discrete problem is solved in space and time on a 64 × 64 × 64 grid. The loss function includes a discretization of the equation with a first-order upwind scheme, terms to impose known initial and final profiles of the tracer, and regularization terms $\|10^{-3}\nabla^2\mathbf{u}\|^2$ and $\|\mathbf{u}_t\|^2$ to prioritize velocity fields that are smooth and stationary. The results of inference are shown in fig. 3. The inferred

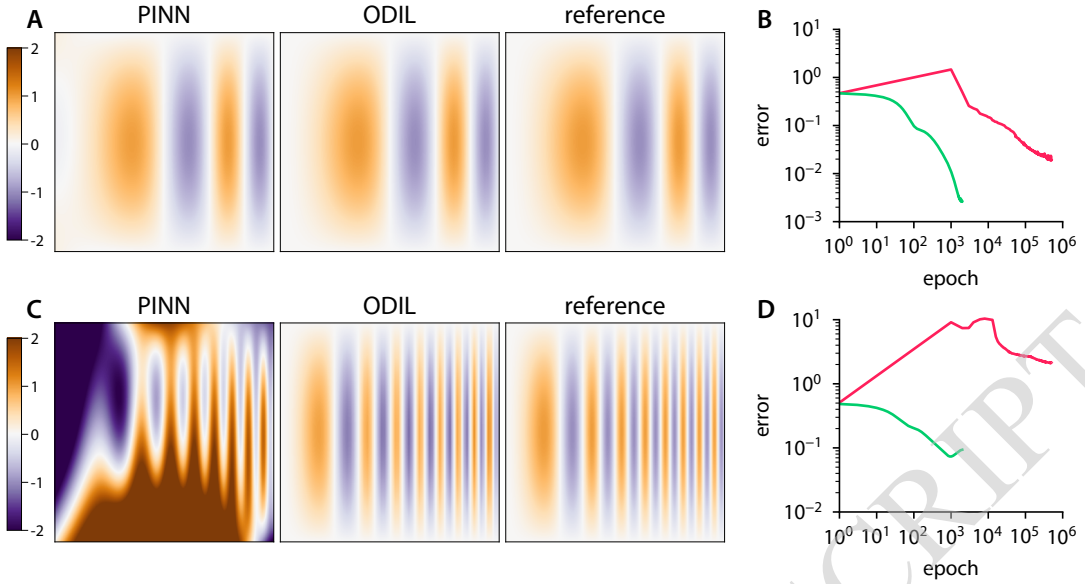

Figure 2. Poisson equation solved using PINN and ODIL. The exact reference solution is an oscillatory function $u(x, y) = \sin(\pi(kx)^2)\sin \pi y$. (A, B) Solution and history of root-mean-square error obtained using PINN with Adam and ODIL with Adam compared to the exact reference solution for $k = 2$. (C, D) Same for $k = 4$.

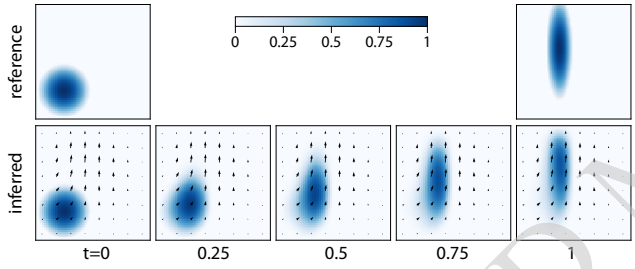

Figure 3. Inferring the velocity field from two snapshots of a tracer. The tracer satisfies the advection equation with an unknown velocity, and the reference tracer is imposed at the first and final time instants. Arrows show the inferred velocity field.

velocity field stretches the initial tracer representing a circle to match the final profile.

Inferring the velocity field from tracers is an example of the optical flow problem [24] in computer vision and the image registration problem [48]in medical imaging. Furthermore, reconstructing the velocity field from a concentration field can assist experimental measurements.

Inferring conductivity from temperature

Here we consider an inverse problem of inferring a conductivity function from temperature measurements. We solve the problem in a unit domain $(x, t) \in [0, 1]^2$. The problem is to find a nonlinear conductivity function $k(u)$ and temperature field $u(x, t)$ that satisfies the heat equation

$$u_t - (k(u)u_x)_x = 0 \tag{28}$$

with zero Dirichlet boundary conditions $u(0, t) = u(1, t) = 0$ and initial conditions $u(x, 0) = U(x)$, where $U(x) = g(x) - g(0)$ and $g(x) = e^{-50(x-0.5)^2}$. In addition, the temperature field needs to take known values $u(x_j, t_j) = \tilde{U}_j$ in a finite set of measurement points $(\tilde{x}_j, \tilde{t}_j)$ for $j = 1, \ldots, N_{\text{data}}$ with $N_{\text{data}} = 200$. We discretize the equation on a uniform grid of $N_x \times N_t$ cells with a second-order Crank-Nicolson scheme such that the residual operator takes the form

$$F_i^n[u, k] = \frac{u_i^n - u_i^{n-1}}{\Delta t} - \frac{k(u_{i+1/2}^{n-1/2})(u_{i+1}^{n-1/2} - u_i^{n-1/2}) - k(u_{i-1/2}^{n-1/2})(u_i^{n-1/2} - u_{i-1}^{n-1/2})}{\Delta x^2} \tag{29}$$

for $i = 1, \ldots, N_x$ and $n = 1, \ldots, N_t$, where $u_{i+1/2} = (u_{i+1} + u_i)/2$, $u^{n-1/2} = (u^n + u^{n-1})/2$, and $k : \mathbb{R} \to \mathbb{R}$ is a conductivity function. The initial and boundary conditions enter the discretization through values in the halo cells defined as

$$\begin{aligned} u_i^0 &= \frac{1}{3}(u_i^2 - 6u_i^1 + 8U(x_i)) \\ u_0^n &= \frac{1}{3}(u_2^n - 6u_1^n) \\ u_{N_x+1}^n &= \frac{1}{3}(u_{N_x-1}^n - 6u_{N_x}^n) \end{aligned} \tag{30}$$

using second-order extrapolation. The loss function for the inverse problem consists of the residual and terms to impose the temperature measurements

$$\mathcal{L}(u, \theta) = \frac{1}{N_x N_t}\sum_{i=1}^{N_x}\sum_{n=1}^{N_t}\left(F_i^n(u, k_\theta)\right)^2 + \frac{w_{\text{data}}^2}{N_{\text{data}}}\sum_{i=1}^{N_x}\sum_{n=1}^{N_t}\left((u_i^n - \tilde{U}_i^n)\chi_i^n\right)^2 \tag{31}$$

where $w_{\text{data}} = 2$. We assume that the set of measurement points $(\tilde{x}_j, \tilde{t}_j)$ is a subset of the cell centers. If a cell (i, n) contains a measurement point j, then we set $\tilde{U}_i^n = \tilde{U}_j$ and define a mask field $\chi_i^n = 1$, otherwise $\chi_i^n = 0$. For Newton's method, the loss

function additionally includes a damping term for the weights of the neural network

$$\mathcal{L}^{\text{Newton}}(u,\theta) = \mathcal{L}(u,\theta) + \frac{w_\theta^2}{N_\theta}\sum_{i=1}^{N_\theta}(\theta_i - \theta_i^*)^2 \tag{32}$$

where $w_\theta = 1$, θ is a vector of all weights, and θ^* is a vector with the same weights but "frozen" so they are ignored in the linearization of the problem. This regularization makes the matrix in the linear system (13) nonsingular, but does not affect the solution if the iteration converges.

To generate the temperature measurements and the reference solution, we specify the reference conductivity function as

$$k(u) = 0.02\, e^{-20(u-0.5)^2} \tag{33}$$

and solve the forward problem using ODIL on a grid of 256×256 cells. Then we solve the inverse problem using PINN and ODIL and compare the results. To represent the unknown conductivity function $k(u)$, we use a fully-connected neural network of size $1\times5\times5\times1$, i.e. one input u, two hidden layers with five neurons in each layer and tanh activations, and its output q passed through $k = 0.1/(1+e^{-q})$ to ensure that the conductivity is non-negative. To represent the temperature field $u(x,t)$, we use a fully-connected neural network of size $2\times32\times32\times32\times32\times1$ in PINN and a uniform grid of 64×64 cells in ODIL. The number of collocation points for PINN is 4096 points inside the domain and 384 for the initial and boundary conditions. To solve the optimization problem with PINN and ODIL with use Adam with the learning rate set to 0.001, and also use Newton's method with ODIL. Figure 4 shows the convergence history, the inferred temperature and conductivity. Both PINN and ODIL infer similar temperature fields and conductivity functions. The convergence history includes the root-mean-square error in the temperature field relative to the reference solution and the root-mean-square error in the conductivity function in the range $u \in (0,1)$ with both quantities normalized by their maximum values in the reference solution. ODIL with Newton demonstrates the fastest convergence and takes about 25 iterations to achieve an error of 5% in the conductivity function. ODIL with Adam takes 13000 iterations to achieve the same error. In contrast, PINN converges slower achieving the same error after 55000 iterations. The execution time on one CPU core amounts to about 250 ms per iteration for PINN, 3.2 ms per iteration for ODIL with Adam, and 5500 ms per iteration for ODIL with Newton. On a GPU, PINN takes 3.5 ms per iteration. Consistent with our previous observations, ODIL takes fewer iterations than PINN and each iteration is cheaper, which results in two-three orders of magnitude lower computational cost overall. Figure 5 shows the results if the temperature measurements are perturbed by Gaussian noise with standard deviation $\sigma = 0.1$. The error is computed relative to the unperturbed reference solution. All methods achive lower accuracy of recovering the conductivity function, ODIL with Newton is affected more by the added noise as the inferred conductivity has a larger error than that of ODIL and PINN optimized with Adam.

Lid-driven cavity. Forward problem

The lid-driven cavity problem is a standard test case [28] for numerical methods for the steady-state Navier-Stokes equations in two dimensions

$$\begin{aligned} u_x + v_y &= 0,\\ uu_x + vu_y &= -p_x + \tfrac{1}{\text{Re}}(u_{xx} + u_{yy}),\\ uv_x + vv_y &= -p_y + \tfrac{1}{\text{Re}}(v_{xx} + v_{yy}), \end{aligned} \tag{34}$$

where $u(x,y)$ and $v(x,y)$ are the two velocity components and $p(x,y)$ is the pressure. The problem is solved in a unit domain with no-slip boundary conditions. The upper boundary is moving to the right at a unit velocity while the other boundaries are stagnant. We solve this problem using ODIL and PINN. In ODIL, we use a finite volume discretization on a uniform Cartesian grid with the second-order upwind scheme for the momentum equations, Rhie-Chow interpolation [64] to prevent oscillations in the pressure field, and the deferred correction approach [23] that treats high-order discretization explicitly and low-order discretization implicitly to obtain an operator with a compact stencil and therefore reduce the size of the linear system for Newton's method. In PINN, we use a fully-connected network of size $2\times64\times64\times64\times64\times64\times3$, with 400 collocation points on the boundaries, and inside the domain there are 10000 points which are resampled every 100 epochs. The implementation of PINN is based on the DeepXDE framework [47]. We refer to Supplementary Fig. S6 and Supplementary Fig. S7 for a hyper-parameter study for PINN.

Figure 6 shows the streamlines for values of Re between 100 and 3200 obtained using ODIL with Newton on a 128×128 grid, as well as a convergence history of PINN with L-BFGS-B, ODIL with Adam, and ODIL with Newton. The learning rate of Adam is 10^{-3}. The root-mean-square error in velocity u in ODIL is computed relative to the solution after 100 epochs with Newton. The error of PINN is measured relative to the solution after 500000 epochs with L-BFGS-B. To get an idea about the convergence speed of each method, we measure how many epochs it takes to reach an error of 1%. At Re = 100, ODIL with Adam takes 5500 epochs and PINN takes 18000 epochs. At Re = 1000, ODIL with Adam takes 6050 epochs and PINN formally takes 420000 epochs, but that indicates that the PINN optimization hasn't converged after 500000 epochs. In contrast, ODIL with Newton takes 3 iterations at Re = 100 and 8 iterations at Re = 1000, showing that Newton's method converges much faster, but at the cost of more expensive iterations. ODIL with Newton takes 2200 ms per epoch, ODIL with Adam takes 8 ms on one CPU core, and PINN takes 30 ms on a GPU Nvidia A100. The results shown in Figure 6 as contour plots of vorticity and profiles of vertical velocity agree well with reference numerical data [28] for both methods at Re = 100, but PINN at Re = 1000 fails to reproduce the reference data.

Lid-driven cavity. Flow reconstruction

Here we solve the same equations (34) as in the forward problem, but we impose the known velocity values $\mathbf{u}(\mathbf{x}_j) = \tilde{\mathbf{u}}_j$ in a finite set of measurement points $\mathbf{x}_i$ for $j = 1,\dots,N_{\text{data}}$ with $N_{\text{data}} = 100$. However, we remove the no-slip boundary conditions and replace them with linear extrapolation, turning this problem into a flow reconstruction problem. The loss function is the same as in the forward problem, but the known velocity values are imposed through reparameterization of the velocity field, which takes the known value in cells containing a measurement point, and an unknown value otherwise. The problem is solved on a grid of 128×128 cells and the reference velocity field is obtained from the forward problem. Figure 7 shows the results of reconstruction from 100 points.

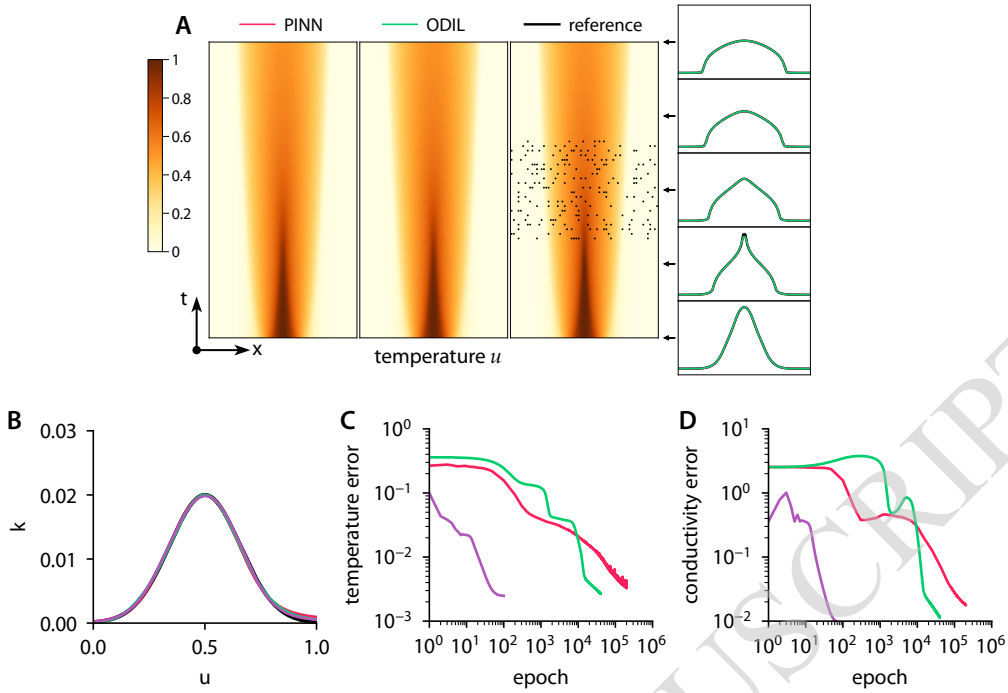

Figure 4. Inferring conductivity from temperature measurements using ODIL and PINN. Temperature field (A) and conductivity function (B) inferred by PINN with Adam ——, ODIL with Adam ——, and ODIL with Newton —— compared to reference ——. The reference solution is from forward problem solved using ODIL on a finer grid. The temperature measurements in 200 points (black dots) are taken from the reference solution. History of root-mean-square error in temperature (C) and conductivity (D).

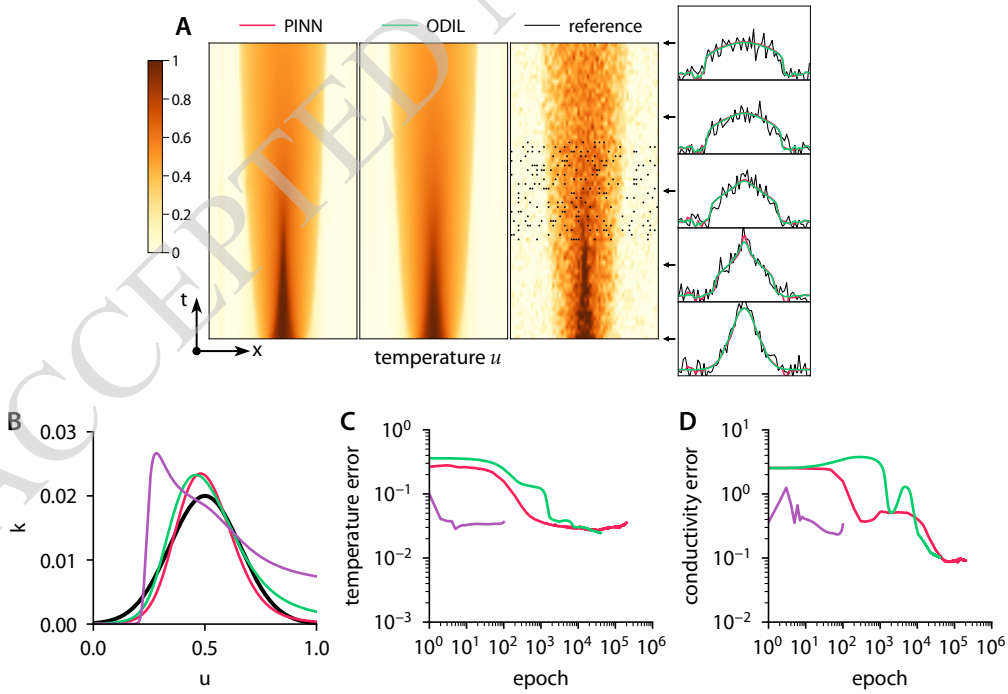

Figure 5. Inferring conductivity from noisy temperature measurements using ODIL and PINN. Temperature field (A) and conductivity function (B) inferred by PINN with Adam ——, ODIL with Adam ——, and ODIL with Newton —— compared to reference ——. The reference solution is from forward problem solved using ODIL on a finer grid. The temperature measurements in 200 points (black dots) are taken from the reference solution perturbed by Gaussian noise with $\sigma = 0.1$. History of root-mean-square error in temperature (C) and conductivity (D) relative to unperturbed reference solution.

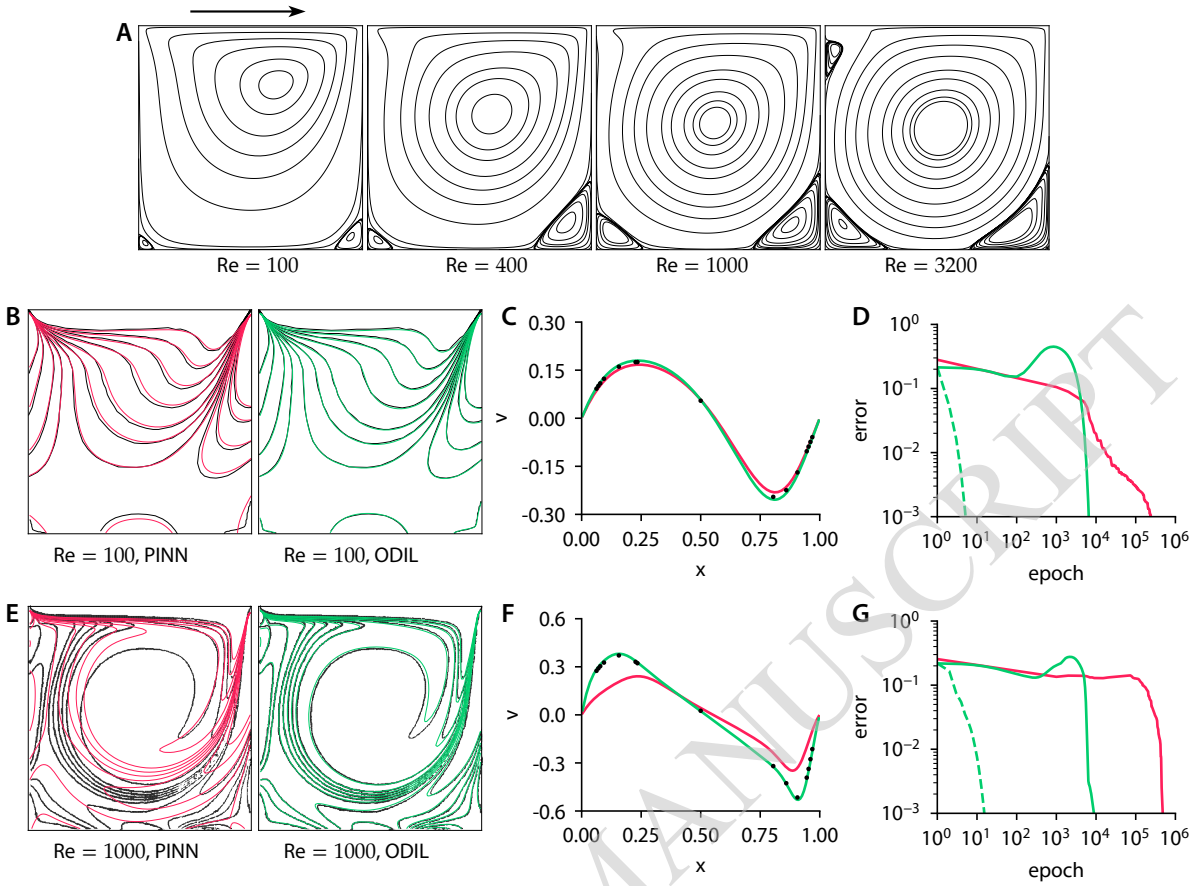

Figure 6. Lid-driven cavity problem solved using PINN and ODIL. A) Streamlines for Re = 100, 400, 1000, and 3200 obtained using ODIL. The arrow indicates the direction of upper wall. B, C, D) Results for Re = 100 obtained using PINN with L-BFGS-B ——— and ODIL with Newton ———. E, F, G) Same for Re = 1000. B, E) Contours of vorticity compared to reference [28] ———. C, F) Profiles of velocity v along the centerline $y = 0.5$ compared to reference [28] ●. D, G) History of root-mean-square error in velocity u obtained using PINN with L-BFGS-B ———, ODIL with Newton - - - -, and ODIL with Adam ———. The error of PINN is relative to the solution after 500000 epochs with L-BFGS-B. The error of ODIL is relative to the solution after 100 epochs with Newton.

Measure of flow complexity

Based on the flow reconstruction from velocity measurements using ODIL in the previous case, we introduce a measure of flow complexity. Consider a velocity field $\mathbf{u}_{\text{ref}}$ that satisfies the Navier-Stokes equations. First we define the minimal reconstruction error given K points

$$E_{\min}(K) = \min_{|X|=K} \|\mathbf{u}_{\text{ref}} - \mathbf{u}(X)\|, \tag{35}$$

where $\mathbf{u}(X)$ is the velocity field reconstructed from points X and the minimum is taken over all sets of K points. Then we define a measure of flow complexity for a given accuracy $\varepsilon > 0$ as the minimal number of points required to achieve that accuracy

$$K_{\min}(\varepsilon) = \{K \mid E_{\min}(K) < \varepsilon\}. \tag{36}$$

To illustrate this complexity measure, we consider four types of flow: uniform velocity, Couette flow (linear profile), Poiseuille flow (parabolic profile), and the flow in a lid-driven cavity. The reference velocity fields in the first three cases are chosen once at an arbitrary orientation and such that the maximum velocity magnitude is unity. As in the previous case, we do not impose any boundary conditions for the velocity, replacing them with linear extrapolation. However, to generate the samples faster, we use a smaller $N \times N$ grid with $N = 64$ and optimize using Newton's method. Newton's method with ODIL requires adding a regularization term, for which we choose a discretization of

$$k_{\text{reg}}^2\left(\|u_{xx}\|^2 + \|u_{yy}\|^2 + \|v_{xx}\|^2 + \|v_{yy}\|^2\right) \tag{37}$$

with $k_{\text{reg}} = 10^{-3}$. Figure 8 shows the values of $E(K)$ and $K_{\min}(0.05)$. The error is the mean absolute difference between the inferred and reference solutions summed over both velocity components. As expected, the uniform flow takes 1 point to reconstruct while adding higher order terms to the velocity increases the number of points. Finally, 29 points are required to reach an error of 0.05 for the lid-driven cavity flow.

Inferring body shape from velocity in two dimensions

Here we consider a two-dimensional inverse problem of inferring the shape of a body from measurements of the flow velocity around the body. The model consists of the steady-state Navier-Stokes equations with penalization terms to impose the no-slip conditions on the body [3]

$$\begin{aligned} u_x + v_y &= 0, \\ (1-\chi)\left(uu_x + vu_y + p_x - \tfrac{D}{\text{Re}}(u_{xx} + u_{yy})\right) + \lambda\chi u &= 0, \\ (1-\chi)\left(uv_x + vv_y + p_y - \tfrac{D}{\text{Re}}(v_{xx} + v_{yy})\right) + \lambda\chi v &= 0, \end{aligned} \tag{38}$$

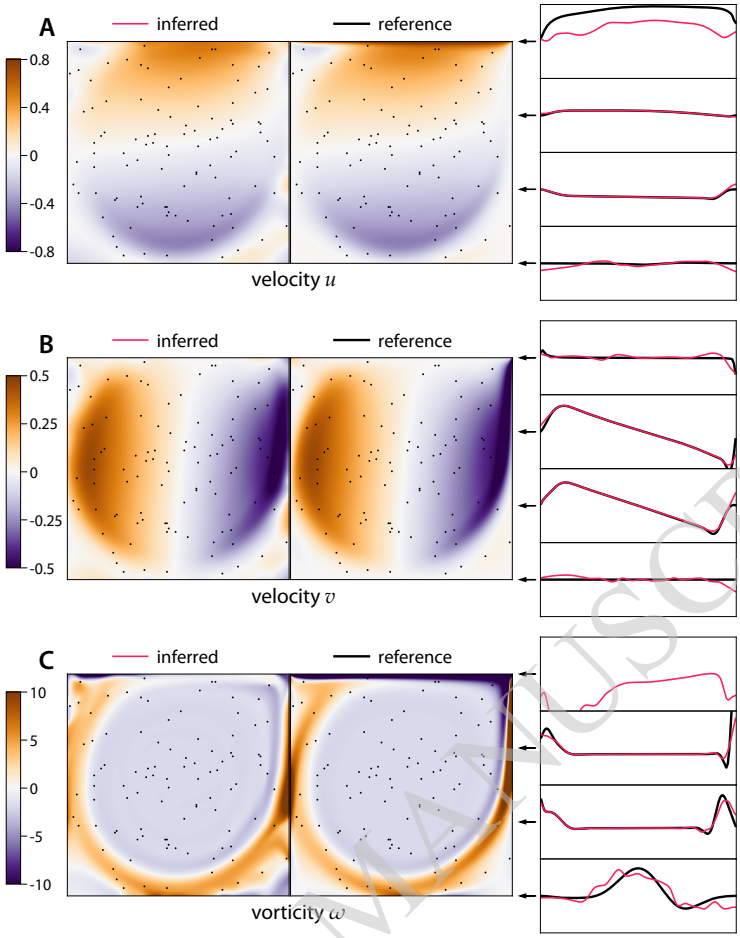

Figure 7. Lid-driven cavity flow at Re = 3200 reconstructed using ODIL from velocity measurements in 100 points (black dots) without boundary conditions for velocity. Inferred and reference velocity u (A), velocity v (B), and vorticity ω (C).

where λ is a penalization parameter and D is a characteristic length of the body. The shape of the body is described by the body fraction $\chi(x, y)$ which takes values $\chi = 1$ inside the body and $\chi = 0$ outside. The problem is solved in the domain $[0, 2] \times [0, 1]$ with the inlet conditions $u = 1$ and $v = 0$ at $x = 0$, outlet at $x = 2$, and free-slip walls at $y = 0$ and $y = 1$. The discretization of the Navier-Stokes equations follows the one used for the lid-driven cavity problem. The forward problem is to find the velocities u and v and pressure p given equations (38) with boundary conditions and a prescribed body fraction χ. The inverse problem is to find the velocities u and v, pressure p, and body fraction χ given equations (38) with boundary conditions and values of u and v in a set of measurement points.

To solve the inverse problem using ODIL, we formulate it as minimization of the loss function in terms of the unknown fields: velocities u and v, pressure p, and a level-set function φ. The problem is solved on a $2N \times N$ grid with $N = 64$. The known velocity values are imposed through reparameterization of the velocity field, which takes the known value in cells containing a measurement point, and an unknown value otherwise. The loss function consists of the residuals of the governing equations (38) and a term with the eikonal equation for the level-set function

$$k_\phi^2 \left\| |\nabla \varphi|^2 - 1 \right\|_2^2 \tag{39}$$

discretized using an upwind scheme [59]. The parameter decays as $k_\phi = \max(1, 10 \cdot (0.5)^{s/1000})$ with epoch s, and the penalization parameter is set to $\lambda = 10$. The level-set function determines the body fraction as

$$\chi = \max(0, \min(1, \tfrac{1}{2} + \tfrac{\phi}{4\Delta x})). \tag{40}$$

This approach with the level-set function ensures that the interface is diffused over a constant number of cells. The reference data is obtained from the forward problem. The chosen grid is sufficiently fine to resolve the flow based on the grid refinement study in Supplementary Fig. S11. Figures 9 to 11 show the results of inference for three different bodies: a circle, an ellipse, and a non-convex body. The characteristic length of the body is $D = 0.4$ and the Reynolds number is Re = 60. To solve the optimization problem, we use Adam with a learning rate of 0.001. The initial guess is $u = 1$, $v = 0$, $p = 0$, and the level-set function

$$\phi(x, y) = 0.02 - \sqrt{(x - 0.5)^2 + (y - 0.5)^2} \tag{41}$$

describing a small circle. The optimization takes 20000 iterations and completes in about 3 minutes on one CPU core. ODIL recovers the reference shape from velocity measurements in 100 points if the shape is convex. However, we observe that the method does not infer non-convex shapes and instead finds a convex shape that matches the velocity measurements.

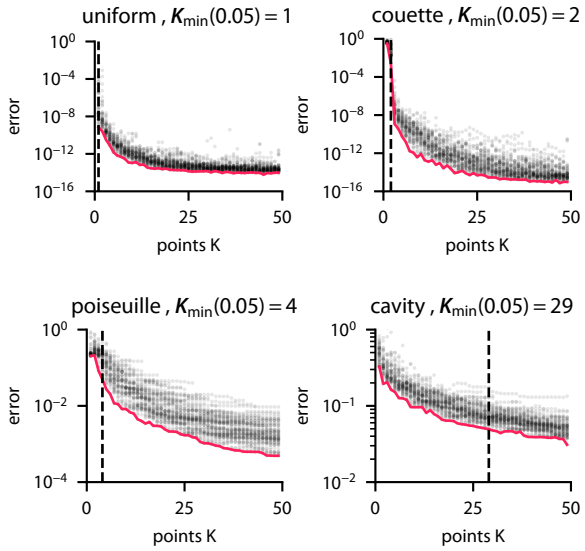

Figure 8. Fluid flow reconstruction error depending on the number of measurement points for various flows: uniform, Couette flow, Poiseuille flow, and lid-driven cavity. The dots show samples of random sets of K points and the minimum over them is an estimate of the minimal reconstruction error $E_{\min}(K)$ ——.

Inferring body shape from velocity in three dimensions

Here we consider a three-dimensional inverse problem of inferring the shape of a body from measurements of the flow velocity around the body. The model consists of the steady-state Navier-Stokes equations with penalization terms to impose the no-slip conditions on the body [3]

$$\begin{aligned} \nabla \cdot \mathbf{u} &= 0, \\ (1-\chi)\big((\mathbf{u}\cdot\nabla)\mathbf{u} + \nabla p - \tfrac{D}{\mathrm{Re}}\nabla^2\mathbf{u}\big) + \lambda\chi\mathbf{u} &= 0, \end{aligned} \tag{42}$$

where λ is a penalization parameter and D is a characteristic length of the body. The shape of the body is described by the body fraction $\chi(\mathbf{x})$ which takes values $\chi = 1$ inside the body and $\chi = 0$ outside. The problem is solved in the domain $[0,2]\times[0,1]\times[0,1]$ with the inlet condition $\mathbf{u} = (1,0,0)$ at $x = 0$, outlet condition $p = 0$ at $x = 2$, and free-slip walls on the other boundaries. The discretization of the Navier-Stokes equations follows the one used for the lid-driven cavity problem. The forward problem is to find the velocity $\mathbf{u}$ and pressure p that satisfy equations (42) given a prescribed body fraction χ. The inverse problem is to find the velocity $\mathbf{u}$, pressure p, and body fraction χ that satisfy equations (42) such that the velocity field takes known values $\mathbf{u}(\mathbf{x}_i) = \mathbf{u}_i$ in a finite set of N measurement points $\mathbf{x}_i$ for $i = 1,\dots,N$.

To solve the inverse problem using ODIL, we formulate it as minimization of the loss function in terms of the unknown fields: velocity $\mathbf{u}$, pressure p, and body fraction $\hat{\chi}$. Here $\hat{\chi}$ is a transformed body fraction defined as $\chi = 1/(1+e^{-(\hat{\chi}+5)})$, so that during the optimization the body fraction χ only takes values between 0 and 1. The loss function is a sum of the residuals of equations (42) and terms to impose the reference data. We do not include any regularization terms. The penalization parameter is set to $\lambda = 1$. The problem is solved on a $129\times65\times65$ grid and the reference data is obtained from the forward problem. The characteristic length of the body is $D = 0.4$ and the Reynolds number is $\mathrm{Re} = 60$. To solve the optimization problem, we use L-BFGS [45] implemented in TensorFlow Probability [19]. The initial guess is $\mathbf{u} = (1,0,0)$ for the velocity, $p = 0$ for the pressure, and $\hat{\chi} = 0$ for the transformed body fraction. According to the above transformation, the corresponding initial guess for the body fraction is $\chi = 1/(1+e^5)$. We terminate the algorithm after 10000 epochs for the forward problem and 20000 epochs for the inverse problem.

Figures 12 and 13 show the results of the inference from 684 measurement points for two different bodies: a sphere and a hemisphere. The convergence history includes the velocity error and the body fraction error which are defined relative to the solution of the forward problem. In both cases, ODIL recovers a body shape that qualitatively agrees with the reference, although the relative error in the body fraction field amounts to 50%, so the inferred body volume is larger. On a GPU Nvidia A100, the forward problem with a sphere takes 53 minutes in total and 320 ms per epoch, while the inverse problem takes 122 minutes in total and 366 ms per epoch. We note that solving the same inverse problem on a finer grid of $257\times129\times129$ cells takes 132 minutes in total and 400 ms per epoch. Therefore, an eightfold increase in the number of grid cells will lead to a minor additional cost in the execution time of 8% since the GPU operates more efficiently with larger arrays.

Conclusion

We introduce the Optimizing DIscrete Loss (ODIL) framework for solving inverse problems for PDEs by casting their discretization as an optimization problem and applying optimization techniques that are widely available in machine learning software. The concept of casting the PDE as is closely related to the neural network formulations proposed by [20, 40, 71] and recently revived as PINNs. However, the fact that we use the discrete approximation of the equations allows for ODIL to be orders of magnitude more efficient in terms of computational cost and accuracy compared to the PINN for which complex flow problems "remain elusive" [75].

We remark that the error of PINN typically scales as a square root of the number of training points regardless of the space dimensionality [53], which can be beneficial for high-dimensional problems [36, 52]. However, a recent attempt [22] to apply PINN for simulating an open quantum system has proven unsuccessful, outperformed by the Q-Flow method based on normalizing flows proposed therein. In ODIL, the grid-based discretization may suffer from the curse of dimensionality as the error of a p-order accurate method with N grid points scales as $\mathcal{O}(N^{p/d})$ in a d-dimensional space. To solve high-dimensional problems with ODIL, we suggest the use of use of sparse grids [26, 57, 67] that introduce a hierarchical basis so that the error scales as $\mathcal{O}(N^p(\log N)^d)$ with the number of space dimensions. Another approach is to use a grid in some of the dimensions (e.g. spatial dimensions) and discretize the remaining dimensions with neural networks or basis functions such as spherical harmonics. The latter approach has outperformed fully-connected networks by two orders of magnitude on problems of view synthesis in computer graphics [25]. The treatment of complex geometries with PINNs is equivalent to treatments deployed in particle methods, using globally interpolating kernels [38]. as the collocation points can be positioned arbitrarily. For ODIL, we have demonstrated here the use of

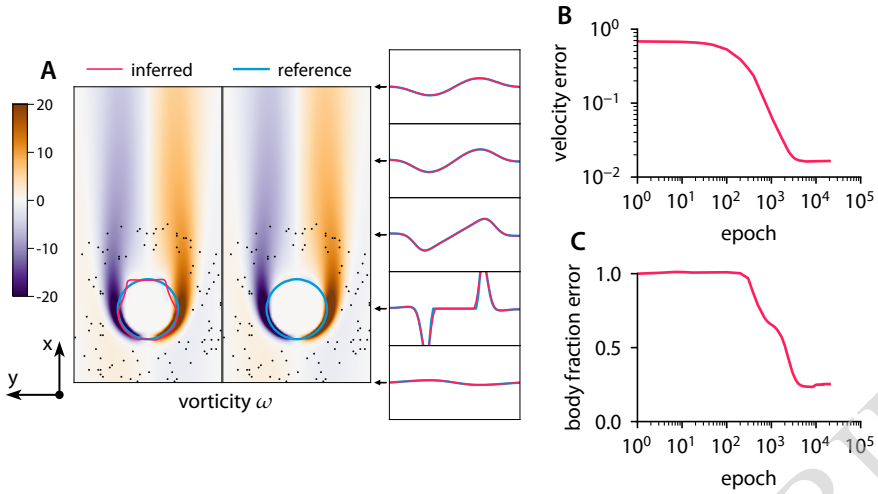

Figure 9. Inferring body shape from velocity measurements for a flow past a circle at Re = 60 using ODIL. The reference solution is from the forward problem solved using ODIL. A) Inferred ——— and reference ——— vorticity ω overlapped by contours of body fraction $\chi = 0.5$. The velocity measurements are imposed in 100 points (black dots) chosen at a distance from the body. B) History of root-mean-square in velocity u. C) History of root-mean-square in body fraction χ normalized by the mean of reference. See Supplementary Fig. S13 for visualizations of velocity u, velocity v, body fraction χ, and level-set function ϕ.

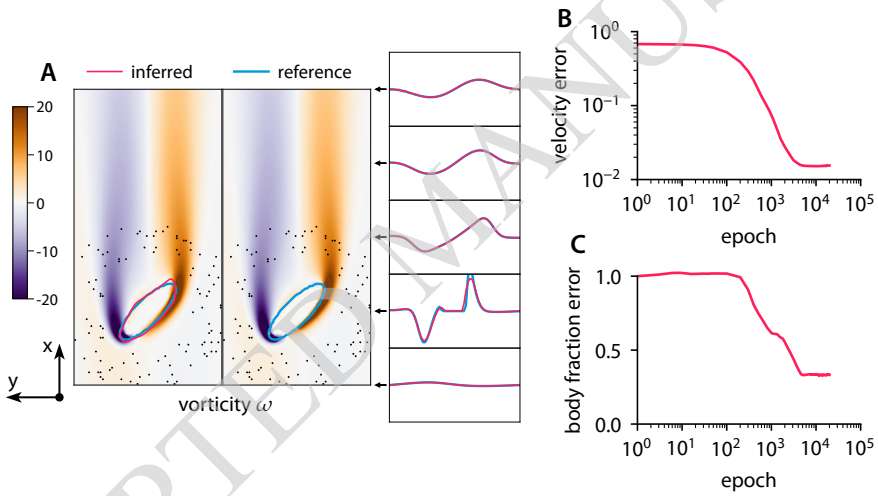

Figure 10. Inferring body shape from velocity measurements for a flow past an ellipse at Re = 60 using ODIL. The reference solution is from the forward problem solved using ODIL. A) Inferred ——— and reference ——— vorticity ω overlapped by contours of body fraction $\chi = 0.5$. The velocity measurements are imposed in 100 points (black dots) chosen at a distance from the body. B) History of root-mean-square in velocity u. C) History of root-mean-square in body fraction χ normalized by the mean of reference. See Supplementary Fig. S14 for visualizations of velocity u, velocity v, body fraction χ, and level-set function ϕ.

the penalization method for the flow past a body. Another possibility is to use an unstructured grid fitted to the complex boundaries or even ideas from particle methods.

We have presented the applications of ODIL to several forward and inverse problems involving PDEs, including the Poisson equation, the heat equation, the advection equation, and the Navier-Stokes equations. In most cases the discrete loss has been formulated using second-order accurate finite volume discretizations and incorporate exactly the boundary and initial conditions through extrapolation. We emphasize that the primary scope of ODIL is in solving ill-posed and inverse problems. For forward, and in particular time dependent problems, a multitude of classical solvers based on marching in time are far more efficient than PINNs and ODIL since they do not require storing all the time steps as unknowns.

The ideas and results presented in this paper suggest that ODIL, and even more the ideas it entails, can serve as inspiration for advancing the solution of inverse problems. We remark that while ODIL is shown to vastly outperform PINNs, its development was motivated by the widespread use of PINNs and their treatment of equations and data. We believe that there are many promising venues at such interfaces (and competitions) of machine learning and scientific computing for solving inverse problems using physics and data informed methodologies. Ongoing works using ODIL include tracking the evolution of brain tumors from medical images [4], the derivation of material properties for specific heat conduction and the identification of submerged obstacles from surface information.

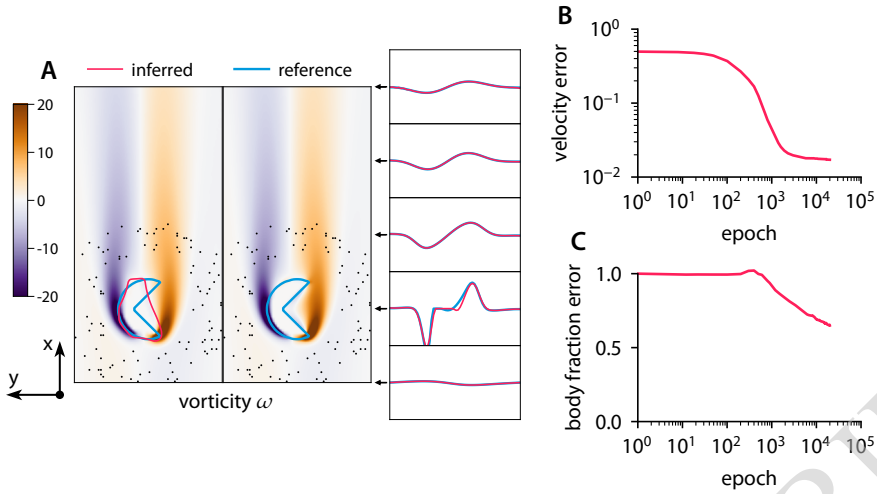

Figure 11. Inferring body shape from velocity measurements for a flow past a non-convex body at Re = 60 using ODIL. The reference solution is from the forward problem solved using ODIL. A) Inferred and reference vorticity ω overlapped by contours of body fraction $\chi = 0.5$. The velocity measurements are imposed in 100 points (black dots) chosen at a distance from the body. B) History of root-mean-square in velocity u. C) History of root-mean-square in body fraction χ normalized by the mean of reference. See Supplementary Fig. S15 for visualizations of velocity u, velocity v, body fraction χ, and level-set function ϕ.

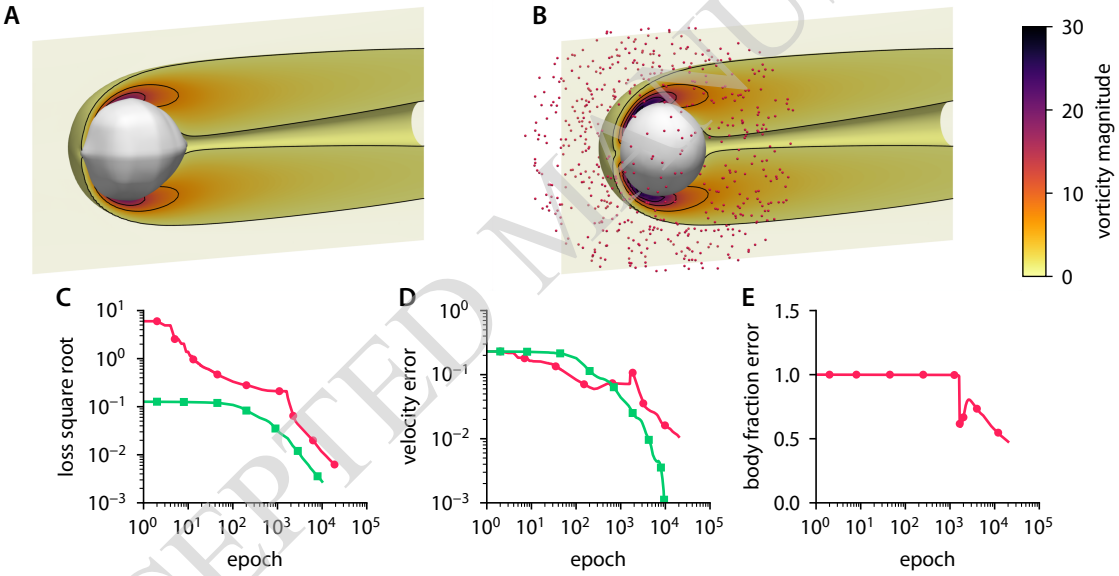

Figure 12. Inferring body shape from velocity measurements for a flow past a sphere using ODIL. Inferred (A) and reference (B) body shape and contours of vorticity magnitude. The velocity measurements are imposed in 684 points (red dots) chosen at a distance from the body. History of the square root of the loss (C), root-mean-square error in x-velocity (D), and body fraction χ normalized by the mean of reference (E) for the inverse problem and forward problem . The error is relative to the solution of the forward problem after 10000 epochs.

Acknowledgments

We thank Professors Michael P. Brenner (Harvard University), Georges-Henri Cottet (retired) and Thomas Y. Hou (Caltech) for several insightful discussions. We are also grateful to the five anonymous Reviewers for their multiple, valuable suggestions that helped us to improve this paper.

Funding

This work is supported in part by funds from The European High Performance Computing Joint Undertaking (EuroHPC) Grant DCoMEX (956201-H2020-JTI-EuroHPC-2019-1). P. Koumoutsakos acknowledges support by the AFOSR program (FA9550-21-1-0058).

Supplementary Material

Supplementary material is available at PNAS Nexus online.

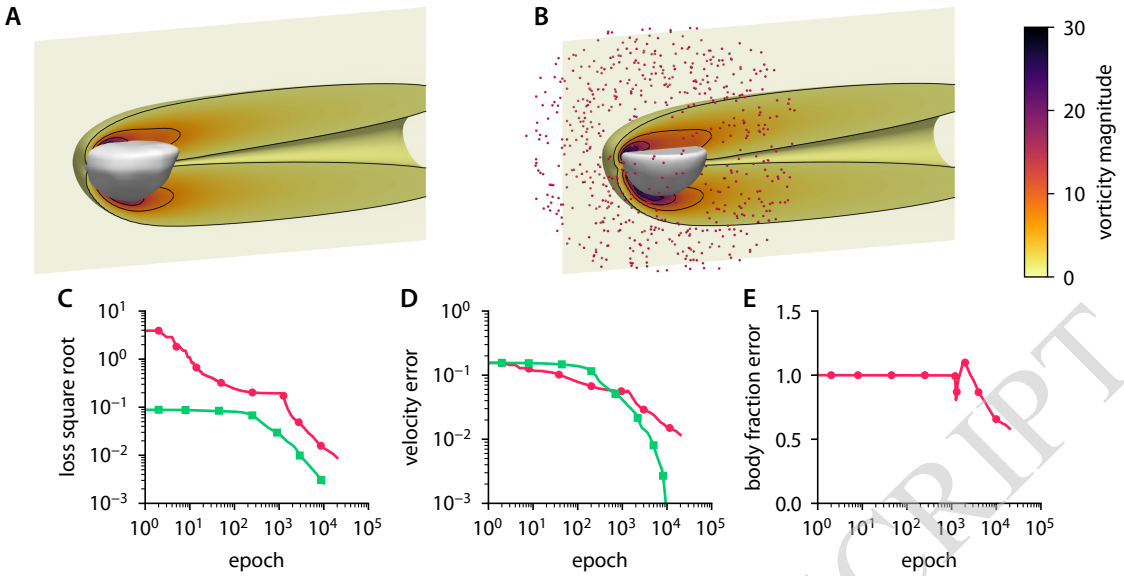

Figure 13. Inferring body shape from velocity measurements for a flow past a hemisphere using ODIL. Inferred (A) and reference (B) body shape and contours of vorticity magnitude. The velocity measurements are imposed in 684 points (red dots) chosen at a distance from the body. History of the square root of the loss (C), root-mean-square error in x-velocity (D), and body fraction χ normalized by the mean of reference (E) for the inverse problem —●— and forward problem —■—. The error is relative to the solution of the forward problem after 10000 epochs.

Author contributions statement

P.Karnakov: methodology, software implementation of ODIL, visualization, writing (original draft). S.L.: methodology, software implementation of PINN, validation, writing (review and editing). P.Koumoutsakos: conceptualization, methodology, supervision, writing (review and editing).

Preprints

A preprint of this article is published at arXiv:2205.04611.

Data availability

The source code of the implementation of the method is available at `https://github.com/cselab/odil` along with examples and instructions to reproduce the results.

References

1. M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.
2. B. Amos and J. Z. Kolter. Optnet: Differentiable optimization as a layer in neural networks. In *International Conference on Machine Learning*, pages 136–145. PMLR, 2017.
3. P. Angot, C.-H. Bruneau, and P. Fabrie. A penalization method to take into account obstacles in incompressible viscous flows. *Numerische Mathematik*, 81(4):497–520, 1999.
4. M. Balcerak, I. Ezhov, P. Karnakov, S. Litvinov, P. Koumoutsakos, J. Weidner, R. Z. Zhang, J. S. Lowengrub, B. Wiestler, and B. Menze. Individualizing glioma radiotherapy planning by optimization of a data and physics informed discrete loss. *arXiv preprint arXiv:2312.05063*, 2023.
5. Y. Bar-Sinai, S. Hoyer, J. Hickey, and M. P. Brenner. Learning data-driven discretizations for partial differential equations. *Proceedings of the National Academy of Sciences*, 116(31):15344–15349, 2019.
6. S. Basir and I. Senocak. Critical investigation of failure modes in physics-informed neural networks. In *AIAA SCITECH 2022 Forum*, page 2353, 2022.
7. S. Basir and I. Senocak. Physics and equality constrained artificial neural networks: Application to forward and inverse problems with multi-fidelity data fusion. *Journal of Computational Physics*, 463:111301, 2022.
8. 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.
9. N. Bell, L. N. Olson, and J. Schroder. PyAMG: Algebraic multigrid solvers in python. *Journal of Open Source Software*, 7(72):4142, 2022.
10. J. Bettencourt, M. J. Johnson, and D. Duvenaud. Taylor-mode automatic differentiation for higher-order derivatives in JAX. In *Program Transformations for ML Workshop at NeurIPS 2019*, 2019.
11. J. T. Betts. *Practical methods for optimal control and estimation using nonlinear programming*. SIAM, 2010.
12. H. Boche, A. Fono, and G. Kutyniok. Limitations of deep learning for inverse problems on digital hardware, 2022.

13. H. G. Bock. Recent advances in parameteridentification techniques for ode. In *Numerical Treatment of Inverse Problems in Differential and Integral Equations: Proceedings of an International Workshop, Heidelberg, Fed. Rep. of Germany, August 30—September 3, 1982*, pages 95–121. Springer, 1983.
14. S. L. Brunton and J. N. Kutz. *Data-Driven Science and Engineering: Machine Learning, Dynamical Systems, and Control*. Cambridge University Press, 2019.
15. P.-Y. Chuang and L. A. Barba. Experience report of physics-informed neural networks in fluid simulations: pitfalls and frustration. *arXiv preprint arXiv:2205.14249*, 2022.
16. T. F. Coleman and J. J. Moré. Estimation of sparse jacobian matrices and graph coloring problems. *SIAM journal on Numerical Analysis*, 20(1):187–209, 1983.
17. T. Cui, Y. M. Marzouk, and K. E. Willcox. Data-driven model reduction for the bayesian solution of inverse problems. *International Journal for Numerical Methods in Engineering*, 102(5):966–990, 2015.
18. J. W. Demmel, S. C. Eisenstat, J. R. Gilbert, X. S. Li, and J. W. H. Liu. A supernodal approach to sparse partial pivoting. *SIAM J. Matrix Analysis and Applications*, 20(3):720–755, 1999.
19. J. V. Dillon, I. Langmore, D. Tran, E. Brevdo, S. Vasudevan, D. Moore, B. Patton, A. Alemi, M. Hoffman, and R. A. Saurous. Tensorflow distributions. *arXiv preprint arXiv:1711.10604*, 2017.
20. M. Dissanayake and N. Phan-Thien. Neural-network-based approximations for solving partial differential equations. *communications in Numerical Methods in Engineering*, 10(3):195–201, 1994.
21. S. Dong and N. Ni. A method for representing periodic functions and enforcing exactly periodic boundary conditions with deep neural networks. *Journal of Computational Physics*, 435:110242, 2021.
22. O. M. Dugan, P. Y. Lu, R. Dangovski, D. Luo, and M. Soljacic. Q-flow: generative modeling for differential equations of open quantum dynamics with normalizing flows. In *International Conference on Machine Learning*, pages 8879–8901. PMLR, 2023.
23. J. H. Ferziger and M. Peric. *Computational methods for fluid dynamics*. Springer Science & Business Media, 2012.
24. D. Fleet and Y. Weiss. Optical flow estimation. In *Handbook of mathematical models in computer vision*, pages 237–257. Springer, 2006.
25. S. Fridovich-Keil, A. Yu, M. Tancik, Q. Chen, B. Recht, and A. Kanazawa. Plenoxels: Radiance fields without neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5501–5510, 2022.
26. J. Garcke. Sparse grids in a nutshell. In *Sparse grids and applications*, pages 57–80. Springer, 2012.
27. O. Ghattas and K. Willcox. Learning physics-based models from data: perspectives from inverse problems and model reduction. *Acta Numerica*, 30:445–554, 2021.
28. U. Ghia, K. N. Ghia, and C. Shin. High-Re solutions for incompressible flow using the Navier-Stokes equations and a multigrid method. *Journal of computational physics*, 48(3):387–411, 1982.
29. N. Gicquel, J. Anderson, and I. Kevrekidis. Noninvertibility and resonance in discrete-time neural networks for time-series processing. *Physics Letters A*, 238(1):8–18, 1998.
30. T. G. Grossmann, U. J. Komorowska, J. Latz, and C.-B. Schönlieb. Can physics-informed neural networks beat the finite element method? *arXiv preprint arXiv:2302.04107*, 2023.
31. M. D. Gunzburger. *Perspectives in flow control and optimization*. SIAM, 2002.
32. A. D. Jagtap, K. Kawaguchi, and G. E. Karniadakis. Adaptive activation functions accelerate convergence in deep and physics-informed neural networks. *Journal of Computational Physics*, 404:109136, 2020.
33. B. Kaltenbacher. Regularization based on all-at-once formulations for inverse problems. *SIAM Journal on Numerical Analysis*, 54(4):2594–2618, 2016.
34. P. Karnakov, S. Litvinov, and P. Koumoutsakos. Flow reconstruction by multiresolution optimization of a discrete loss with automatic differentiation. *The European Physical Journal E*, 46(7):59, Jul 2023.
35. G. E. Karniadakis, I. G. Kevrekidis, L. Lu, P. Perdikaris, S. Wang, and L. Yang. Physics-informed machine learning. *Nature Reviews Physics*, 3(6):422–440, 2021.
36. S. Kawaguchi and T. Murakami. Physics-informed neural networks for solving the boltzmann equation of the electron velocity distribution function in weakly ionized plasmas. *Japanese Journal of Applied Physics*, 61(8):086002, 2022.
37. D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.
38. P. Koumoutsakos. Multiscale flow simulations using particles. *Annu. Rev. Fluid Mech.*, 37:457–487, 2005.
39. A. Krishnapriyan, A. Gholami, S. Zhe, R. Kirby, and M. W. Mahoney. Characterizing possible failure modes in physics-informed neural networks. *Advances in Neural Information Processing Systems*, 34:26548–26560, 2021.
40. I. E. Lagaris, A. Likas, and D. I. Fotiadis. Artificial neural networks for solving ordinary and partial differential equations. *IEEE transactions on neural networks*, 9(5):987–1000, 1998.
41. R. J. LeVeque. *Finite difference methods for ordinary and partial differential equations: steady-state and time-dependent problems*. SIAM, 2007.
42. J. M. Lewis, S. Lakshmivarahan, and S. Dhall. *Dynamic data assimilation: a least squares approach*, volume 13. Cambridge University Press, 2006.
43. J. Liang, M. Lin, and V. Koltun. Differentiable cloth simulation for inverse problems. *Advances in Neural Information Processing Systems*, 32, 2019.
44. B. List, L.-W. Chen, and N. Thuerey. Learned turbulence modelling with differentiable fluid solvers: physics-based loss functions and optimisation horizons. *Journal of Fluid Mechanics*, 949:A25, 2022.
45. D. C. Liu and J. Nocedal. On the limited memory bfgs method for large scale optimization. *Mathematical programming*, 45(1-3):503–528, 1989.
46. L. Ljung. *System Identification (2nd Ed.): Theory for the User*. Prentice Hall PTR, USA, 1999.
47. 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.
48. A. Mang, A. Gholami, C. Davatzikos, and G. Biros. Pde-constrained optimization in medical image analysis. *Optimization and Engineering*, 19(3):765–812, 2018.
49. R. Mattey and S. Ghosh. A novel sequential method to train physics informed neural networks for Allen Cahn and Cahn Hilliard equations. *Computer Methods in Applied Mechanics and Engineering*, 390:114474, 2022.

50. N. McGreivy and A. Hakim. Invariant preservation in machine learned pde solvers via error correction. *arXiv preprint arXiv:2303.16110*, 2023.
51. M. Milano and P. Koumoutsakos. Neural network modeling for near wall turbulent flow. *Journal of Computational Physics*, 182(1):1–26, 2002.
52. S. Mishra and R. Molinaro. Physics informed neural networks for simulating radiative transfer. *Journal of Quantitative Spectroscopy and Radiative Transfer*, 270:107705, 2021.
53. 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.
54. P. Mistani, S. Pakravan, R. Ilango, S. Choudhry, and F. Gibou. Neuro-symbolic partial differential equation solver. In *Neural Information Processing Systems, Machine Learning and the Physical Sciences Workshop*, 2022.
55. P. A. Mistani, S. Pakravan, R. Ilango, and F. Gibou. JAX-DIPS: Neural bootstrapping of finite discretization methods and application to elliptic problems with discontinuities. *Journal of Computational Physics*, 493:112480, 2023.
56. M. Naumov, M. Arsaev, P. Castonguay, J. Cohen, J. Demouth, J. Eaton, S. Layton, N. Markovskiy, I. Reguly, N. Sakharnykh, et al. Amgx: A library for gpu accelerated algebraic multigrid and preconditioned iterative methods. *SIAM Journal on Scientific Computing*, 37(5):S602–S626, 2015.
57. F. Nobile, R. Tempone, and C. G. Webster. A sparse grid stochastic collocation method for partial differential equations with random input data. *SIAM Journal on Numerical Analysis*, 46(5):2309–2345, 2008.
58. J. Nocedal and S. J. Wright. *Numerical optimization*. Springer, 1999.
59. S. Osher and J. A. Sethian. Fronts propagating with curvature-dependent speed: Algorithms based on hamilton-jacobi formulations. *Journal of computational physics*, 79(1):12–49, 1988.
60. S. Pakravan, P. A. Mistani, M. A. Aragon-Calvo, and F. Gibou. Solving inverse-pde problems with physics-aware neural networks. *Journal of Computational Physics*, 440:110414, 2021.
61. M. Quito Jr, C. Monterola, and C. Saloma. Solving n-body problems with neural networks. *Physical review letters*, 86(21):4741, 2001.
62. 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 *International Conference on Machine Learning*, pages 5301–5310. PMLR, 2019.
63. 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.
64. C. M. Rhie and W.-L. Chow. Numerical study of the turbulent flow past an airfoil with trailing edge separation. *AIAA journal*, 21(11):1525–1532, 1983.
65. R. Rico-Martinez and I. G. Kevrekidis. Continuous time modeling of nonlinear systems: A neural network-based approach. In *IEEE International Conference on Neural Networks*, pages 1522–1525. IEEE, 1993.
66. M. Schlottbom and J.-F. Pietschmann. Data-driven gradient flows. *Electronic Transactions on Numerical Analysis*, 57:193–215, 2022.
67. S. A. Smolyak. Quadrature and interpolation formulas for tensor products of certain classes of functions. In *Doklady Akademii Nauk*, volume 148, pages 1042–1045. Russian Academy of Sciences, 1963.
68. N. Sukumar and A. Srivastava. Exact imposition of boundary conditions with distance functions in physics-informed deep neural networks. *Computer Methods in Applied Mechanics and Engineering*, 389:114333, 2022.
69. U. Trottenberg, C. W. Oosterlee, and A. Schuller. *Multigrid*. Elsevier, 2000.
70. T. van Leeuwen and F. J. Herrmann. A penalty method for pde-constrained optimization in inverse problems. *Inverse Problems*, 32(1):015007, 2015.
71. B. P. van Milligen, V. Tribaldos, and J. Jiménez. Neural network differential equation and plasma equilibrium solver. *Physical review letters*, 75(20):3594, 1995.
72. P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020.
73. P. R. Vlachas, J. Pathak, B. R. Hunt, T. P. Sapsis, M. Girvan, E. Ott, and P. Koumoutsakos. Backpropagation algorithms and reservoir computing in recurrent neural networks for the forecasting of complex spatiotemporal dynamics. *Neural Networks*, 126:191–217, 2020.
74. N. Wandel, M. Weinmann, and R. Klein. Learning incompressible fluid dynamics from scratch - towards fast, differentiable fluid models that generalize. In *International Conference on Learning Representations*, 2021.
75. S. Wang, S. Sankaran, and P. Perdikaris. Respecting causality is all you need for training physics-informed neural networks. *arXiv preprint arXiv:2203.07404*, 2022.
76. Z. Wang, X. Meng, X. Jiang, H. Xiang, and G. E. Karniadakis. Solution multiplicity and effects of data and eddy viscosity on navier-stokes solutions inferred by physics-informed neural networks. *arXiv preprint arXiv:2309.06010*, 2023.
77. C. Zhu, R. H. Byrd, P. Lu, and J. Nocedal. Algorithm 778: L-BFGS-B: Fortran subroutines for large-scale bound-constrained optimization. *ACM Transactions on mathematical software (TOMS)*, 23(4):550–560, 1997.
78. O. C. Zienkiewicz, R. L. Taylor, and J. Z. Zhu. *The finite element method: its basis and fundamentals*. Elsevier, 2005.