

Solving inverse problems in physics by optimizing a discrete loss is much more faster and accurate without neural networks

Petr Karnakov

Harvard University

Sergey Litvinov

Harvard University

Petros Koumoutsakos (✉ petros@seas.harvard.edu)

Harvard University <https://orcid.org/0000-0001-8337-2122>

Article

Keywords:

Posted Date: November 8th, 2022

DOI: <https://doi.org/10.21203/rs.3.rs-2243537/v1>

License:   This work is licensed under a Creative Commons Attribution 4.0 International License.

[Read Full License](#)

Additional Declarations: There is **NO** Competing Interest.

Solving inverse problems in physics by optimizing a discrete loss is much more faster and accurate without neural networks

Petr Karnakov^a, Sergey Litvinov^a, and Petros Koumoutsakos^{a*}

^aComputational Science and Engineering Laboratory, Harvard John A. Paulson School of Engineering and Applied Sciences, Cambridge, MA, USA

*corresponding author: petros@seas.harvard.edu

November 5, 2022

Abstract

In recent years there is a wealth of activities for solving inverse problems in physics described by partial differential equations (PDEs) using neural networks (NNs). In these efforts, the solution of the PDEs are expressed as NNs that are 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, classical PDE approximations that are facilitated by machine learning tools. The framework of Optimizing a Discrete Loss (ODIL), minimizes a cost function for discrete approximations of the PDEs using gradient-based and Gauss-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 broadly accessible machine learning tools for automatic differentiation. 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 up to five orders of magnitude in computational speed while having better accuracy and convergence rates. We evaluate the method on various forward and inverse problems involving linear and nonlinear partial differential equations including the Navier-Stokes equations for flow reconstruction problems. The present framework interfaces numerical methods and machine learning and presents a powerful tool for solving effectively challenging inverse problems across scientific domains.

1 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 [18, 34, 16] and the discrete problems are solved to determine quantities of interest given initial and/or boundary conditions. In recent years 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 [6, 5, 11, ?, 14]. Solving inverse problems entails formidable challenges as the PDEs need to be solved while observing physical and geometrical constraints as well as noisy data[4]. Such problems are encountered in several fields of science and engineering and they have been handled by methods such as PDE-constrained optimization [13], data assimilation [19], optical flow in computer vision [8], and system identification [20]. 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[14], albeit with certain limitations [4].

Physics informed learning approaches were introduced by Lagaris et al. [17] that represented the unknown fields by neural networks trained by minimizing a loss function that involves the governing equations. Related approaches involving neural networks were also applied for back-tracking of an N -body system [26] and near-wall turbulent flow modeling [22]. At the time of their introduction these methods did not offer significant advantages for solving forward or inverse problems, in particular due to their computational cost. Twenty years later, the method introduced by Lagaris et al. [17] was revived by Raissi et al. [27] 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 classical numerical methods for solving classical well-posed problems involving PDEs but they are positioned as a universal and convenient tool for solving

ill-posed and inverse problems[14]. 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 classical methods for solving forward and inverse problems. The authors in [27] admit that conventional numerical solvers are more efficient for classical problems, a direct comparison with such solvers reveals important drawbacks of the neural network approach.

The computational cost of PINNs, is mainly attributed to the fact that evaluating the solution at one point is proportional to the number of weights, leading to quadratic complexity. In contrast to neural networks, conventional grid-based methods employ local approximations that entail a linear cost. Furthermore, as each weight of the neural network affects the solution at all points, the method is often not consistent with the locality of the original differential equations. In PINNs, even for linear problems, the solution is approximated by nonlinear neural networks. Consequently, the gradient of the residual with respect to the weights at 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 a cost. 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 [23].

Here we present a framework that is based on classical 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 classical 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 Gauss-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 classical 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 [13], as the penalty method for linear problems [30], and is related to the 4D-VAR problem in data assimilation [19]. Here we present a unifying framework for these ideas to solve forward and inverse problems with missing and noisy data and facilitate their deployment by adopting machine learning tools. Adopting modern machine learning tools, such as automatic differentiation, renders the 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.

2 Methods

In the following, we demonstrate the ODIL framework by first deploying it for the one-dimensional wave equation $u_{tt} = u_{xx}$ that is discretized with finite differences. Here, instead of solving this equation by marching in time from known initial conditions, we rewrite the problem as minimization of the loss function

$$L(u) = \sum_{(i,n) \in \Omega_h} \left(\frac{u_i^{n+1} - 2u_i^n + u_i^{n-1}}{\Delta t^2} - \frac{u_{i+1}^n - 2u_i^n + u_{i-1}^n}{\Delta x^2} \right)^2 + \sum_{(i,n) \in \Gamma_h} (u_i^n - g_i^n)^2,$$

where the unknown parameter is the solution itself, a discrete field u_i^n on a Cartesian grid consisting of points. The loss function includes the residual of the discrete equation in grid points Ω_h and imposes boundary and initial

conditions $u = g$ in grid points Γ_h . To solve this problem with a gradient-based method, such as Adam [15] or L-BFGS-B [33], we only need to compute the gradient of the loss which is also a discrete field $\partial L / \partial u$. To apply the Gauss-Newton method, we assume that the loss function is a sum of quadratic terms such as $L(u) = \|\mathbf{F}[u]\|_2^2 + \|\mathbf{G}[u]\|_2^2$ with discrete operators $\mathbf{F}[u]$ and $\mathbf{G}[u]$ and linearize the operators about the current solution u^s to obtain a quadratic loss

$$L^s(u^{s+1}) = \|\mathbf{F}^s + \nabla \mathbf{F}^s(u^{s+1} - u^s)\|_2^2 + \|\mathbf{G}^s + \nabla \mathbf{G}^s(u^{s+1} - u^s)\|_2^2,$$

where $\mathbf{F}^s$ and $\mathbf{G}^s$ denote $\mathbf{F}[u^s]$ and $\mathbf{G}[u^s]$, while $\nabla \mathbf{F}^s$ and $\nabla \mathbf{G}^s$ denote their Jacobians at point u^s . A minimum of this function provides the solution u^{s+1} at the next iteration and satisfies a linear system

$$\left((\nabla \mathbf{F}^s)^T \nabla \mathbf{F}^s + (\nabla \mathbf{G}^s)^T \nabla \mathbf{G}^s \right) (u^{s+1} - u^s) + (\nabla \mathbf{F}^s)^T \mathbf{F}^s + (\nabla \mathbf{G}^s)^T \mathbf{G}^s = 0.$$

Cases with terms other than $\mathbf{F}$ and $\mathbf{G}$ are handled similarly.

We further assume that $\mathbf{F}[u]$ and $\mathbf{G}[u]$ at each grid point depend only on the value of u in the neighboring points. For example, for the considered problem we assume the following structure

$$\mathbf{F}[u]_i^n = F(u_i^{n-1}, u_{i-1}^n, u_i^n, u_{i+1}^n, u_i^{n+1})$$

with a scalar function $F(u_1, u_2, u_3, u_4, u_5) : \mathbb{R}^5 \rightarrow \mathbb{R}$. This makes the Jacobian $\nabla \mathbf{F}$ a sparse matrix with non-zero components

$$\begin{aligned} \nabla \mathbf{F}[u]_{i,i}^{n,n-1} &= \frac{\partial F}{\partial u_1}(u_i^{n-1}, u_{i-1}^n, u_i^n, u_{i+1}^n, u_i^{n+1}) \\ \nabla \mathbf{F}[u]_{i,i-1}^{n,n} &= \frac{\partial F}{\partial u_2}(u_i^{n-1}, u_{i-1}^n, u_i^n, u_{i+1}^n, u_i^{n+1}) \\ \nabla \mathbf{F}[u]_{i,i}^{n,n} &= \frac{\partial F}{\partial u_3}(u_i^{n-1}, u_{i-1}^n, u_i^n, u_{i+1}^n, u_i^{n+1}) \\ \nabla \mathbf{F}[u]_{i,i+1}^{n,n} &= \frac{\partial F}{\partial u_4}(u_i^{n-1}, u_{i-1}^n, u_i^n, u_{i+1}^n, u_i^{n+1}) \\ \nabla \mathbf{F}[u]_{i,i}^{n,n+1} &= \frac{\partial F}{\partial u_5}(u_i^{n-1}, u_{i-1}^n, u_i^n, u_{i+1}^n, u_i^{n+1}) \end{aligned}$$

The derivatives of F are obtained using automatic differentiation in TensorFlow [1].

To solve the resulting linear system we use either a direct [31] or multigrid sparse linear solver [3].

We note that efficient application of the Gauss-Newton method for our framework is limited to cases where the linearized operator is sparse. For example, if the discrete operator includes neural networks to represent an unknown function, Jacobians of the operator with respect to the weights are dense matrices. However, the optimization problem in such cases can be solved by gradient-based methods.

3 Results

We evaluate our method on the wave equation in comparison with PINN. Finally, we apply the method to forward and inverse problems for the Navier-Stokes and advection equations.

3.1 Wave equation. Accuracy and cost of PINN

We apply ODIL to the solution of the initial-value problem for the wave equation $u_{tt} = u_{xx}$ and compare the results of ODIL with those of PINN in terms of accuracy and computational cost. Both methods reduce the problem to minimization of a loss function. ODIL represents the solution on a uniform grid and constructs the loss from a second-order finite volume discretization of the equation, while PINN represents the solution as a fully connected neural network and evaluates the residuals of the original differential equation exactly on a finite set of collocation points [27]. The initial conditions for u and u_t and Dirichlet conditions on the boundaries are specified from an exact solution $u(x, t) = \frac{1}{10} \sum_{k=1}^5 (\cos(x - t + 0.5)\pi k + \cos(x + t + 0.5)\pi k)$ in a rectangular domain $x \in (-1, 1)$, $t \in (0, 1)$. For a given number of parameters N , ODIL represents the solution on a $\sqrt{N} \times \sqrt{N}$ grid, while PINN consists of two equally sized hidden layers with tanh activation functions and N being the total number of weights. The number of collocation points for PINN is fixed and amounts to 8192 points inside the domain and 768 points for the initial and boundary conditions. Fig. 1 shows examples of the solutions obtained using both methods as well as measurements of their accuracy and cost depending on the number of parameters. The measurements for PINN include 25 samples for the random initial weights and positions of the collocation points. In the case of ODIL, solutions from two optimization methods L-BFGS-B

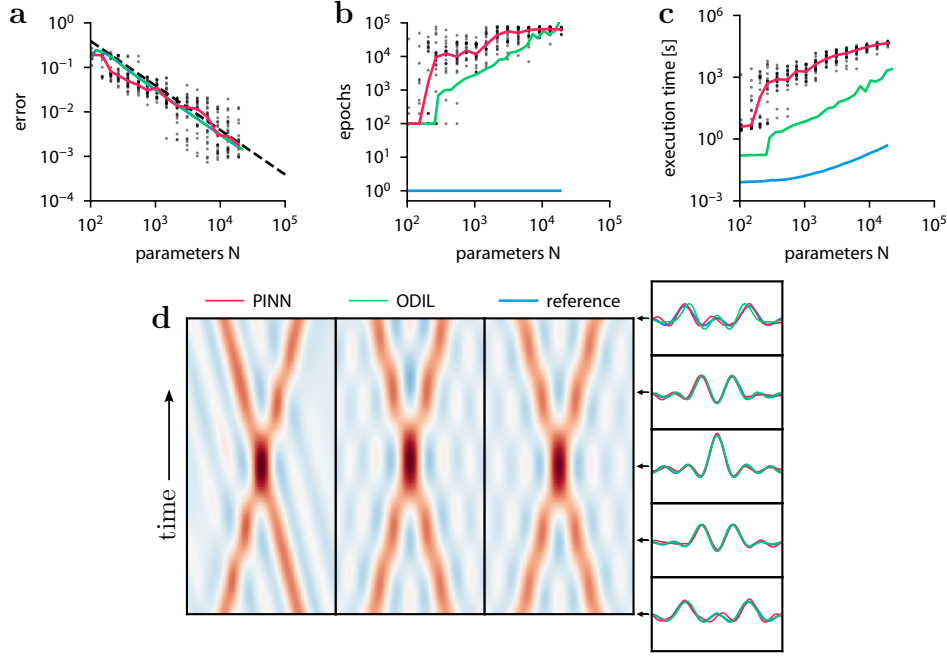


Figure 1: Wave equation solved using PINN and ODIL. (a,b,c) L_2 -error (a), number of epochs (b), and execution time (c) of various methods: PINN optimized with L-BFGS-B —, ODIL optimized with L-BFGS-B —, ODIL optimized with Gauss-Newton —, and line with slope $1/N$ ---. The dots correspond to samples of the initial weights and collocation points of PINN while solids lines show the median value. The number of parameters is the number of weights for PINN and the number of grid points for ODIL. (d) Examples of solutions obtained using PINN with two hidden layers of 25 neurons after 12000 epochs, ODIL on a 25×25 grid (interpolated to a 128×128 grid), and exact reference solution. The cross sections show solutions by PINN optimized with L-BFGS-B —, ODIL optimized with Gauss-Newton —, and reference solution —.

and Gauss-Newton coincide. The execution time is a product of the number of optimization epochs required to reach a 150% of the error obtained after 80000 epochs and an average execution time over the last 100 epochs. The measurements are performed on Intel Xeon E5-2690 v3 CPUs. Both methods demonstrate similar accuracy. The error of ODIL scales as $1/N$ corresponding to a second-order accurate discretization. The error of PINN is significantly scattered, although the median error also scales as $1/N$. The execution times, however, are drastically different. With $N \approx 19000$ parameters, optimization of PINN using L-BFGS-B takes 13 hours, optimization of ODIL using L-BFGS-B takes 35 minutes, and Gauss-Newton with a direct linear solver [31] takes only 0.5 seconds, which is about 100 000 times faster than PINN.

3.2 Lid-driven cavity. Forward problem

The lid-driven cavity problem is a standard test case [12] for numerical methods for the steady-state Navier-Stokes equations in two dimensions

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

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 use a finite volume discretization on a uniform Cartesian grid based on the SIMPLE method [25, 7] with the Rhie-Chow interpolation [28] to prevent oscillations in the pressure field and the deferred correction approach that treats high-order discretization explicitly and low-order discretization implicitly to obtain an operator with a compact stencil.

Fig. 2 shows the streamlines for values of Re between 100 and 3200 obtained on a 128×128 grid, as well as a convergence history of various optimization algorithms on a 64×64 grid. The L_2 error is computed relative to the solution of the discrete problem. The number of iterations it takes for L-BFGS-B to reach an error of 10^{-3} at $\text{Re} = 400$ is in the order of 10000 which is comparable to the number of iterations typically needed for training PINN for flow problems [27]. Adam has only reached an error of 0.05 even after 50 000 iterations. Conversely,

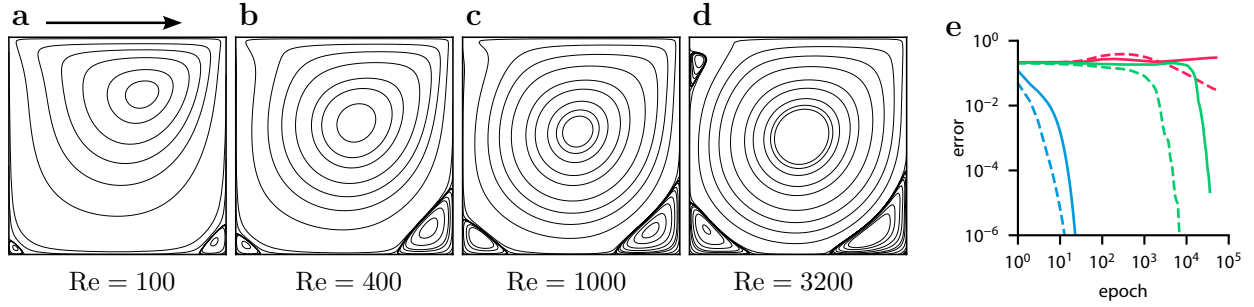


Figure 2: Lid-driven cavity problem solved with ODIL. (a,b,c,d) Streamlines at various Re. (e) History of the L_2 -error from various optimization algorithms: Adam ---, L-BFGS-B ---, and Gauss-Newton --- at Re = 100; Adam —, L-BFGS-B —, and Gauss-Newton — at Re = 400.

Gauss-Newton method takes about 10 iterations to reach an error of 10^{-6} , and the number of iterations does not significantly change between Re = 100 and 400. The learning rate for Adam is set to 0.01 and L-BFGS-B is used with 50 terms for the limited memory matrix and 50 line search steps. The measured execution time per iteration amounts to 0.036 s for Adam, 0.016 s for L-BFGS-B, and 0.8 s for Gauss-Newton.

3.3 Lid-driven cavity. Flow reconstruction

Here we solve the same equations (1) as in the previous case, but instead of the no-slip we only impose free-slip boundary conditions and additionally require that the velocity field equals a reference velocity field in a finite set of points. The loss function is a sum of three types of terms: residuals of equations, terms to impose known velocity values, and a regularization term $k_{\text{reg}}^2 (\|u_{xx}\|^2 + \|u_{yy}\|^2 + \|v_{xx}\|^2 + \|v_{yy}\|^2)$ with $k_{\text{reg}} = 10^{-4}$. The problem is solved on a 128×128 grid and the reference velocity field is obtained from the forward problem. Fig. 3 shows the results of reconstruction from 32 points. Surprisingly, this small number of points is sufficient to reproduce features of the flow.

3.4 Measure of flow complexity

Based on the flow reconstruction from a finite set of points, 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 quantity $E(K) = \min_{|X|=K} \|\mathbf{u}_{\text{ref}} - \mathbf{u}(X)\|$ as the best reconstruction error given K points, 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 $K_{\min}(\varepsilon) = \{K \mid E(K) < \varepsilon\}$.

To illustrate the 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. The problem is solved on a 64×64 grid, the velocity at the boundaries is extrapolated from the inside with second order. The error is defined through the L_1 norm. The loss function includes the same regularization term as in 3.3 with $k_{\text{reg}} = 10^{-3}$. Fig. 4 shows the values of $E(K)$ and $K_{\min}(0.05)$. 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.

3.5 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 $c(\mathbf{x}, t)$ satisfies the advection equation $\frac{\partial c}{\partial t} + \mathbf{u} \cdot \nabla c = 0$ in a unit domain, and the initial $c(\mathbf{x}, 0) = c_0(\mathbf{x})$ and final $c(\mathbf{x}, 1) = c_1(\mathbf{x})$ profiles are known. The discrete problem is solved in space and time on a $64 \times 64 \times 64$ 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. 5. The inferred 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 [8, 21] which is important in computer vision. Furthermore, reconstructing the velocity field from a concentration field can assist experimental measurements.

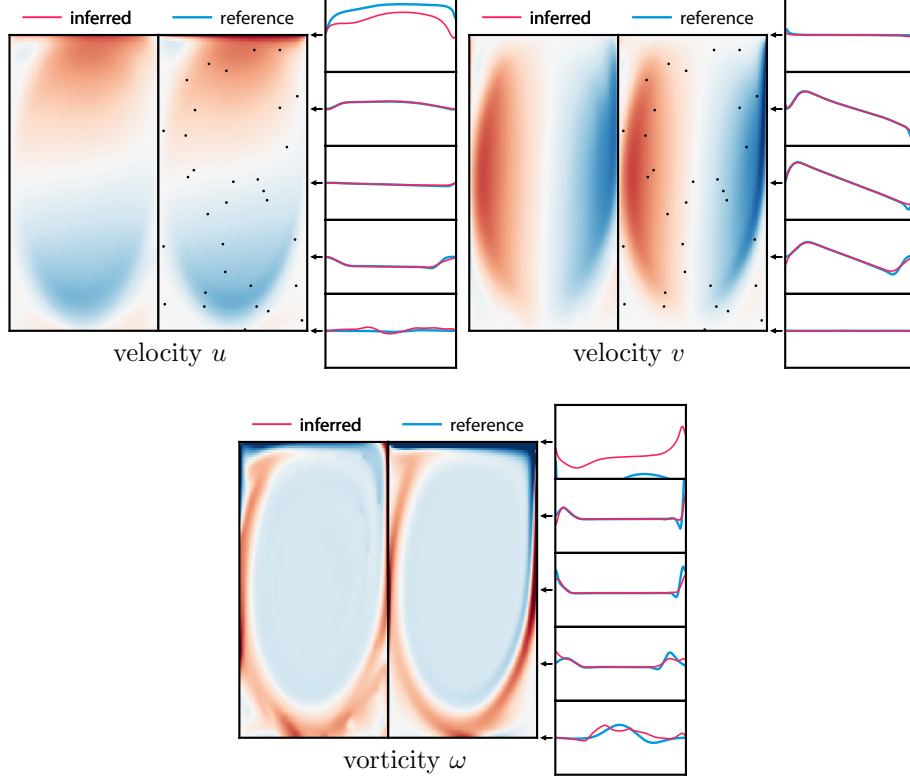


Figure 3: Reconstruction of the lid-driven cavity flow at $Re = 3200$ from velocity in a finite set of points. Inferred and reference velocity components and vorticity. The reference velocity vector is imposed in 32 points (dots), and the Navier-Stokes equations are imposed everywhere with free-slip boundary conditions.

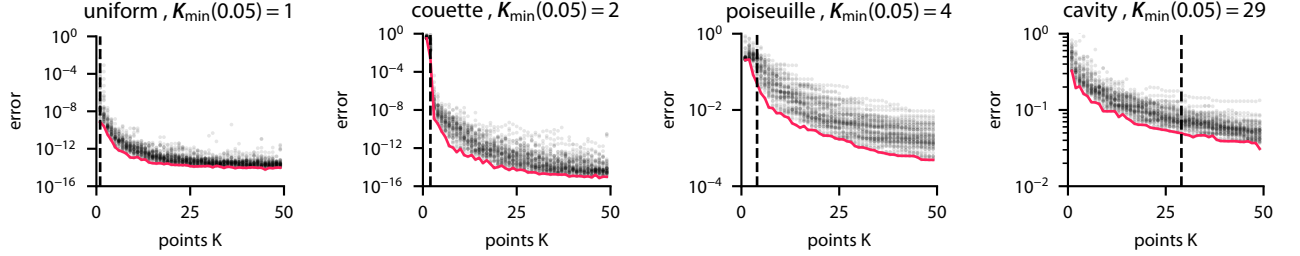


Figure 4: Fluid flow reconstruction error depending on the number of reconstruction points for various flows: uniform velocity, Couette flow, Poiseuille flow, and lid-driven cavity. The dots correspond to samples of sets of K points and the minimal value over them is an estimate of $E(K)$ —.

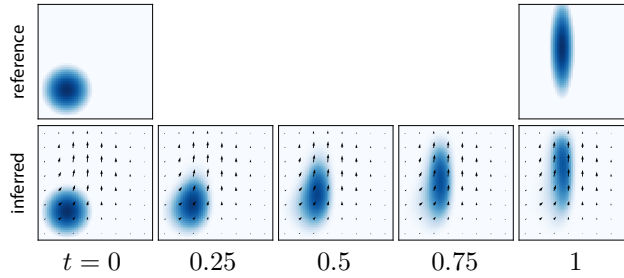


Figure 5: Inferring the velocity field from two snapshots of a tracer. The tracer satisfies the advection equation with an unknown steady velocity field, and the reference field is imposed at the first and final time instant. Arrows show the inferred velocity field.

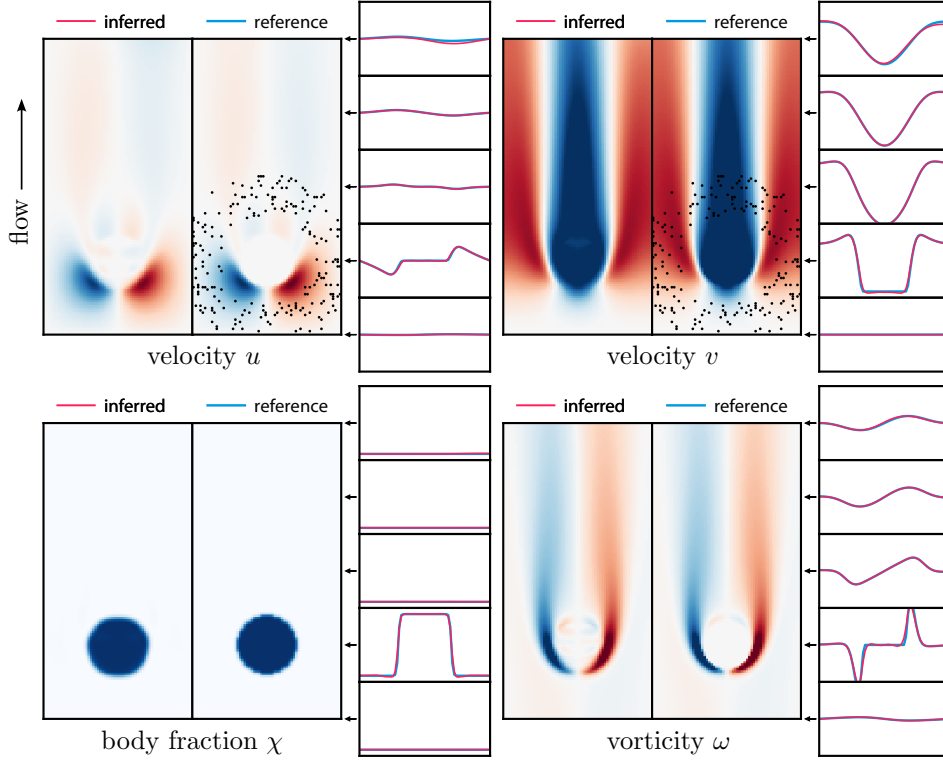


Figure 6: Body shape inferred from flow velocity on a finite set of points. The reference shape is a circle. Inferred and reference fields: x - and y -components of velocity and body volume fraction χ . The reference velocity vector is imposed in points shown with black dots.

3.6 Inferring body shape from velocity

Here we consider an inverse problem of inferring the shape of a body from 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 [2]

$$\begin{aligned} u_x + v_y &= 0, \\ (1 - \chi)(uu_x + vv_y + p_x - \frac{D}{\text{Re}}(u_{xx} + u_{yy})) + \lambda\chi u &= 0, \\ (1 - \chi)(uv_x + vv_y + p_y - \frac{D}{\text{Re}}(v_{xx} + v_{yy})) + \lambda\chi v &= 0, \end{aligned} \quad (2)$$

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 $[0, 1] \times [0, 3]$ with the inlet conditions $u = 0$ and $v = 1$ at $y = 0$, outlet at $y = 3$, and free-slip walls at $x = 0$ and $x = 1$. The discretization follows Section 3.2. The forward problem is to find the velocities u and v and pressure p given equations (2) 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 (2) with boundary conditions and values of u and v on a finite set of points.

To solve the inverse problem using ODIL, we formulate it as a minimization of the loss function in terms of the unknown fields: velocities u and v , pressure p , and body fraction χ . The loss function is a sum of the residuals of equations (2), terms to impose reference data, velocity smoothing term $k_1^2(\|u_{xx} + u_{yy}\|^2 + \|v_{xx} + v_{yy}\|^2)$, body fraction smoothing term $k_2^2\|\chi_{xx} + \chi_{yy}\|^2$, and a term $k_3^2\|1 - (2\chi - 1)^2\|$ to prioritize fields with values 0 and 1. We use annealing for the regularization parameters k_1 and k_2 , i.e. they decay during the optimization as $k_1 = k_2 = 0.01/(1 + s)$, where s is the iteration number. The remaining regularization parameter is fixed at $k_3 = 0.2$ and the penalization parameter is set to $\lambda = 1$. The problem is solved on a 64×192 grid and the reference data is obtained from the forward problem. Figs. 6 and 7 show the results of inference for two different bodies: a circle and an ellipse. The characteristic length of the body is taken to be $D = 0.4$, the Reynolds number is $\text{Re} = 60$. The optimization takes 100 iterations and completes in about 3 minutes on one CPU core. In both cases, ODIL recovers the reference body shape from velocity data on a finite set of 201 points.

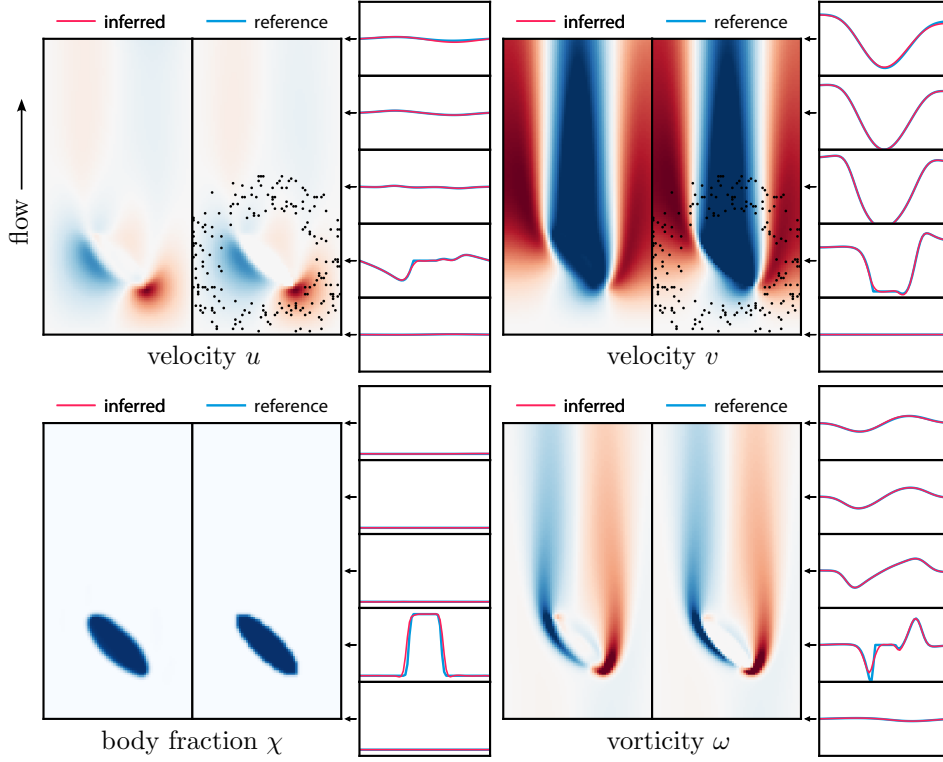


Figure 7: Body shape inferred from flow velocity on a finite set of points. The reference shape is an ellipse. Inferred and reference fields: x - and y -components of velocity and body volume fraction χ . The reference velocity vector is imposed in points shown with black dots.

4 Conclusions

We introduce the Optimizing DIcrete Loss (ODIL) framework for solving forward and inverse problems for PDEs by casting their discretization as an optimization problem and applying optimization techniques that are widely available in machine learning software libraries. The concept of casting the PDE as is closely related to the neural network formulations proposed by Lagaris et al. [17] 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” [32].

We remark that for high-dimensional problems, the error of PINNs typically scales as a square root of the number of training points regardless of the space dimensionality [23]. In ODIL, the representation on a grid 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 [29, 10, 24] 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 has proved successful in the field of view synthesis in computer graphics and outperformed fully-connected networks by two orders of magnitude [9].

The present results suggest that ODIL, and even more the ideas it entails, can serve as inspiration and be well suited for solving challenging data assimilation and inverse problems such as weather prediction, complex biological flows, and other scientific and industrial simulations.

4.1 Code availability

The software implementation of the method is available at <https://github.com/odil22/odil> along with examples and instructions to reproduce the results.

5 Supplementary Information

In order to demonstrate the simplicity and effectiveness of of ODIL we discuss two simple applications: (i) solving an algebraic equation and (ii) a one-dimensional finite-difference equation.

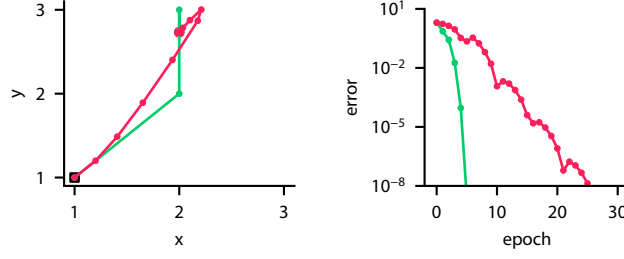


Figure 8: Algebraic equation with two unknowns solved using various optimization algorithms. Trajectory (left) and history of the L_2 error (right) of gradient descent — and Gauss-Newton —.

5.1 Algebraic equation

We consider an algebraic equation $\mathbf{F}(\mathbf{x}) = \mathbf{0}$ with two unknowns:

$$\mathbf{F}(\mathbf{x}) = \begin{pmatrix} x - 2 \\ x - y^2 + xy \end{pmatrix}, \quad \mathbf{x} = \begin{pmatrix} x \\ y \end{pmatrix}.$$

We solve it in the context of ODIL using as optimization algorithms, the gradient descent and Gauss-Newton method. ODIL reformulates this problem as minimization of a loss function

$$L(\mathbf{x}) = \frac{1}{2} \mathbf{F}(\mathbf{x}) \cdot \mathbf{F}(\mathbf{x}).$$

One iteration of the gradient descent method is

$$\mathbf{x}^{s+1} = \mathbf{x}^s - \eta (\nabla \mathbf{F}^s)^T \mathbf{F}^s,$$

and one iteration of the Gauss-Newton method is

$$\nabla \mathbf{F}^s (\mathbf{x}^{s+1} - \mathbf{x}^s) = \mathbf{F}^s,$$

where $\mathbf{F}^s = \mathbf{F}(\mathbf{x}^s)$, $\nabla \mathbf{F}^s$ is the Jacobian of $\mathbf{F}$ at point $\mathbf{x}^s$, and $\eta = 0.2$ is a constant parameter.

Fig. 8 shows the trajectory and convergence history of both algorithms started from an initial guess $\mathbf{x}^0 = (1, 1)^T$. The Gauss-Newton method converges faster and maintains the exact solution $x = 2$ of the first equation throughout the optimization.

5.2 One-dimensional finite difference equation

Here we examine the performance of two optimization algorithms, namely the gradient descent and the Gauss-Newton method, to a boundary value problem for a one-dimensional finite difference equation. The unknown field is u_i for $i = 1, \dots, N$ and the problem is stated as

$$\begin{aligned} u_1 &= 1 \\ u_i - u_{i-1} &= 0, \quad i = 2, \dots, N. \end{aligned}$$

The exact solution to this problem is a uniform field $u_i = 1$. ODIL reformulates this problem as minimization of the following loss function

$$L(u) = \sum_{i=2}^N (u_i - u_{i-1})^2 + (u_1 - 1)^2,$$

which has the gradient

$$\frac{1}{2} \frac{\partial L}{\partial u_j} = \sum_{i=2}^N (u_i - u_{i-1})(\delta_{i,j} - \delta_{i-1,j}) + (u_1 - 1)\delta_{1,j} = \begin{cases} -u_2 + 2u_1 - 1, & j = 1 \\ -u_{j-1} + 2u_j - u_{j+1}, & j = 2, \dots, N-1 \\ u_N - u_{N-1}, & j = N \end{cases}$$

where $\delta_{i,j} = 1$ if $i = j$ and 0 otherwise. The gradient $\frac{\partial L}{\partial u_i}$ at each point i only depends on the values in its immediate neighbors u_{i-1} , u_i , u_{i+1} . Therefore, one iteration of the gradient descent method

$$u_i^{s+1} = u_i^s - \eta \frac{\partial L}{\partial u_i}$$

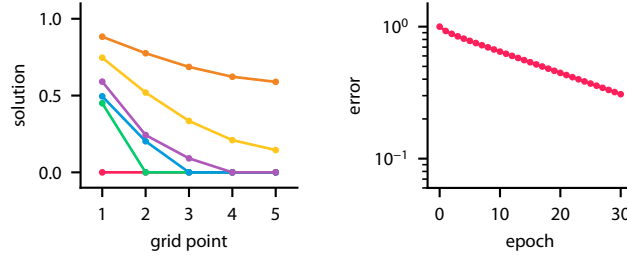


Figure 9: Gradient descent applied to the one-dimensional finite difference equation on a grid of $N = 5$ points. Left: Solution at epochs 0 —, 1 —, 2 —, 3 —, 10 —, and 30 —. Right: History of the L_2 error.

can propagate the information from the boundary $u_1 = 1$ by at most one point. On the other hand, one iteration of the Gauss-Newton method solves a linear system

$$\sum_{j=1}^N \frac{\partial^2 L}{\partial u_i \partial u_j} (u_j^{s+1} - u_j^s) = \frac{\partial L}{\partial u_i}, \quad i = 1, \dots, N$$

and, since the original problem is linear, finds the exact solution from any initial guess. Fig. 9 shows the solution for $N = 5$ at various iterations and convergence history of the gradient descent started from a zero initial guess.

References

- [1] ABADI, M., AGARWAL, A., BARHAM, P., BREVDO, E., CHEN, Z., CITRO, C., CORRADO, G. S., DAVIS, A., DEAN, J., DEVIN, M., GHEMAWAT, S., GOODFELLOW, I., HARP, A., IRVING, G., ISARD, M., JIA, Y., JOZEFOWICZ, R., KAISER, L., KUDLUR, M., LEVENBERG, J., MANÉ, D., MONGA, R., MOORE, S., MURRAY, D., OLAH, C., SCHUSTER, M., SHLENS, J., STEINER, B., SUTSKEVER, I., TALWAR, K., TUCKER, P., VANHOUCKE, V., VASUDEVAN, V., VIÉGAS, F., VINYALS, O., WARDEN, P., WATTENBERG, M., WICKE, M., YU, Y., AND ZHENG, X. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.
- [2] ANGOT, P., BRUNEAU, C.-H., AND FABRIE, P. A penalization method to take into account obstacles in incompressible viscous flows. *Numerische Mathematik* 81, 4 (1999), 497–520.
- [3] BELL, N., OLSON, L. N., AND SCHRODER, J. PyAMG: Algebraic multigrid solvers in python. *Journal of Open Source Software* 7, 72 (2022), 4142.
- [4] BOCHE, H., FONO, A., AND KUTYNIOK, G. Limitations of deep learning for inverse problems on digital hardware, 2022.
- [5] BRUNTON, S. L., AND KUTZ, J. N. *Data-Driven Science and Engineering: Machine Learning, Dynamical Systems, and Control*. Cambridge University Press, 2019.
- [6] CUI, T., MARZOUK, Y. M., AND WILLCOX, K. E. Data-driven model reduction for the bayesian solution of inverse problems. *International Journal for Numerical Methods in Engineering* 102, 5 (2015), 966–990.
- [7] FERZIGER, J. H., AND PERIC, M. *Computational methods for fluid dynamics*. Springer Science & Business Media, 2012.
- [8] FLEET, D., AND WEISS, Y. Optical flow estimation. In *Handbook of mathematical models in computer vision*. Springer, 2006, pp. 237–257.
- [9] FRIDOVICH-KEIL, S., YU, A., TANCIK, M., CHEN, Q., RECHT, B., AND KANAZAWA, A. Plenoxels: Radiance fields without neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2022), pp. 5501–5510.
- [10] GARCKE, J. Sparse grids in a nutshell. In *Sparse grids and applications*. Springer, 2012, pp. 57–80.
- [11] GHATTAS, O., AND WILLCOX, K. Learning physics-based models from data: perspectives from inverse problems and model reduction. *Acta Numerica* 30 (2021), 445–554.
- [12] GHIA, U., GHIA, K. N., AND SHIN, C. High-Re solutions for incompressible flow using the Navier-Stokes equations and a multigrid method. *Journal of computational physics* 48, 3 (1982), 387–411.

- [13] GUNZBURGER, M. D. *Perspectives in flow control and optimization*. SIAM, 2002.
- [14] KARNIADAKIS, G. E., KEVREKIDIS, I. G., LU, L., PERDIKARIS, P., WANG, S., AND YANG, L. Physics-informed machine learning. *Nature Reviews Physics* 3, 6 (2021), 422–440.
- [15] KINGMA, D. P., AND BA, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).
- [16] KOUMOUTSAKOS, P. Multiscale flow simulations using particles. *Annu. Rev. Fluid Mech.* 37 (2005), 457–487.
- [17] LAGARIS, I. E., LIKAS, A., AND FOTIADIS, D. I. Artificial neural networks for solving ordinary and partial differential equations. *IEEE transactions on neural networks* 9, 5 (1998), 987–1000.
- [18] LEVEQUE, R. J. *Finite difference methods for ordinary and partial differential equations: steady-state and time-dependent problems*. SIAM, 2007.
- [19] LEWIS, J. M., LAKSHMIVARAHAN, S., AND DHALL, S. *Dynamic data assimilation: a least squares approach*, vol. 13. Cambridge University Press, 2006.
- [20] LJUNG, L. *System Identification (2nd Ed.): Theory for the User*. Prentice Hall PTR, USA, 1999.
- [21] MANG, A., GHOLAMI, A., DAVATZIKOS, C., AND BIROS, G. Pde-constrained optimization in medical image analysis. *Optimization and Engineering* 19, 3 (2018), 765–812.
- [22] MILANO, M., AND KOUMOUTSAKOS, P. Neural network modeling for near wall turbulent flow. *Journal of Computational Physics* 182, 1 (2002), 1–26.
- [23] MISHRA, S., AND MOLINARO, R. 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 (2022), 981–1022.
- [24] NOBILE, F., TEMPONE, R., AND WEBSTER, C. G. A sparse grid stochastic collocation method for partial differential equations with random input data. *SIAM Journal on Numerical Analysis* 46, 5 (2008), 2309–2345.
- [25] PATANKAR, S. V., AND SPALDING, D. B. A calculation procedure for heat, mass and momentum transfer in three-dimensional parabolic flows. In *Numerical Prediction of Flow, Heat Transfer, Turbulence and Combustion*. Elsevier, 1983, pp. 54–73.
- [26] QUITO JR, M., MONTEROLA, C., AND SALOMA, C. Solving n-body problems with neural networks. *Physical review letters* 86, 21 (2001), 4741.
- [27] RAISSI, M., PERDIKARIS, P., AND KARNIADAKIS, G. E. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational physics* 378 (2019), 686–707.
- [28] RHIE, C. M., AND CHOW, W.-L. Numerical study of the turbulent flow past an airfoil with trailing edge separation. *AIAA journal* 21, 11 (1983), 1525–1532.
- [29] SMOLYAK, S. A. Quadrature and interpolation formulas for tensor products of certain classes of functions. In *Doklady Akademii Nauk* (1963), vol. 148, Russian Academy of Sciences, pp. 1042–1045.
- [30] VAN LEEUWEN, T., AND HERRMANN, F. J. A penalty method for pde-constrained optimization in inverse problems. *Inverse Problems* 32, 1 (2015), 015007.
- [31] VIRTANEN, P., GOMMERS, R., OLIPHANT, T. E., HABERLAND, M., REDDY, T., COURNAPEAU, D., BUROVSKI, E., PETERSON, P., WECKESSER, W., BRIGHT, J., VAN DER WALT, S. J., BRETT, M., WILSON, J., MILLMAN, K. J., MAYOROV, N., NELSON, A. R. J., JONES, E., KERN, R., LARSON, E., CAREY, C. J., POLAT, İ., FENG, Y., MOORE, E. W., VANDERPLAS, J., LAXALDE, D., PERKTOLD, J., CIMRMAN, R., HENRIKSEN, I., QUINTERO, E. A., HARRIS, C. R., ARCHIBALD, A. M., RIBEIRO, A. H., PEDREGOSA, F., VAN MULBREGT, P., AND SCI-PY 1.0 CONTRIBUTORS. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods* 17 (2020), 261–272.
- [32] WANG, S., SANKARAN, S., AND PERDIKARIS, P. Respecting causality is all you need for training physics-informed neural networks. *arXiv preprint arXiv:2203.07404* (2022).

- [33] ZHU, C., BYRD, R. H., LU, P., AND NOCEDAL, J. Algorithm 778: L-BFGS-B: Fortran subroutines for large-scale bound-constrained optimization. *ACM Transactions on mathematical software (TOMS)* 23, 4 (1997), 550–560.
- [34] ZIENKIEWICZ, O. C., TAYLOR, R. L., AND ZHU, J. Z. *The finite element method: its basis and fundamentals*. Elsevier, 2005.