Available online at www.sciencedirect.com
ScienceDirect
Comput. Methods Appl. Mech. Engrg. 411 (2023) 116042
Computer methods in applied mechanics and engineering
www.elsevier.com/locate/cma

Physics-informed deep learning for simultaneous surrogate modeling and PDE-constrained optimization of an airfoil geometry

Yubiao Sun, Ushnish Sengupta, Matthew Juniper*

Department of Engineering, University of Cambridge, Trumpington Street, Cambridge, CB2 1PZ, UK

Received 1 December 2022; received in revised form 29 March 2023; accepted 30 March 2023
Available online 14 April 2023

Abstract

We use a physics-informed neural network (PINN) to simultaneously model and optimize the flow around an airfoil to maximize its lift to drag ratio. The parameters of the airfoil shape are provided as inputs to the PINN and the multidimensional search space of shape parameters is populated with collocation points to ensure that the Navier–Stokes equations are approximately satisfied throughout. We use the fact that the PINN is automatically differentiable to calculate gradients of the lift-to-drag ratio with respect to the airfoil shape parameters. This allows us to optimize with the L-BFGS gradient-based algorithm, which is more efficient than non-gradient-based algorithms, without deriving an adjoint code. We train the PINN with adaptive sampling of collocation points, such that the accuracy of the solution improves as the solution approaches the optimal point. We demonstrate this on two examples: one that optimizes a single parameter, and another that optimizes eleven parameters. The method is successful and, by comparison with conventional CFD, we find that the velocity and pressure fields have small pointwise errors and that the method converges to optimal parameters. We find that different PINNs converge to slightly different parameters, reflecting the fact that there are many closely-spaced local minima when using stochastic gradient descent. This method can be applied relatively easily to other optimization problems and avoids the difficult process of writing adjoint codes. It is, however, more computationally expensive than adjoint-based optimization. As knowledge about training PINNs improves and hardware dedicated to neural networks becomes faster, this method of simultaneous training and optimization with PINNs could become easier and faster than using adjoint codes.

Keywords: Physics-informed neural network; Shape optimization; Surrogate model

1. Introduction

A partial differential equation (PDE) solver maps from a set of parameters (e.g. airfoil shape, Reynolds number, Mach number) to a set of field variables (e.g. velocity, pressure, temperature), and optionally to a performance measure (e.g. lift-to-drag ratio). For some applications, knowledge of the field variables is sufficient. Often, however, the user would like to optimize the performance measure by changing the parameters, for example when optimizing an airfoil shape for drag minimization [1], acoustic metamaterial geometry for sound wave control [2], acoustic horn shape [3], or current-carrying multi-cables [4]. Optimization could be achieved with a non-gradient-based

* Corresponding author.
E-mail address: mpj1001@cam.ac.uk (M. Juniper).

https://doi.org/10.1016/j.cma.2023.116042

optimization method, but only after a large number of direct evaluations of the performance measure. If there are many parameters, the number of direct evaluations becomes prohibitively large. Gradient-based optimization is more efficient. For this, the gradient of the performance measure is calculated with respect to the field variables, whose gradients are in turn calculated with respect to the parameters. In a single calculation, the chain rule of differentiation then gives the gradient of the performance measure with respect to all parameters. This gradient information is then used to converge efficiently to a local optimum.

The calculation of the gradient either requires a new code to solve for adjoint variables or requires that the original PDE solver be differentiable. Most, if not all, PDE solvers need significant adaptation to be differentiable. In many cases, particularly for commonly used PDE solvers, this adaptation is impractical. Neural networks (NN), on the other hand, are differentiable by design. Usually their performance measure is the discrepancy between a measurement and the NN's prediction, given its current parameters. The gradient of this discrepancy with respect to the parameters is automatically calculated and can then be combined with a gradient descent algorithm in order to reduce this discrepancy. Being universal function approximators, NNs can be trained to be surrogate models for a PDE solver [5]. Further, they can be constructed to calculate the performance measure, together with all its gradients with respect to the parameters. In this paper we investigate whether we can train a neural network to be a surrogate model for a PDE solver, while simultaneously using the fact that the NN is automatically differentiable to solve an optimization problem for a performance measure that is constrained by that PDE. We are therefore not simply replacing a traditional PDE solver with a PINN; we are also using the PINN for a purpose that a traditional solver cannot achieve without significant adaptation. We choose shape optimization of an airfoil because this is particularly challenging to achieve with standard mesh-based PDE solvers. The method we investigate is general and could be applied widely in PDE constrained optimization problems.

1.1. PDE-constrained shape optimization

Many engineering problems require solutions of PDEs on complex geometries, for which there is no analytical solution. Numerical solutions are therefore sought, often using finite difference, finite element, or finite volume methods, which require the computational domain to be meshed. Some meshless methods exist but they have limitations. For example, spectral methods require simple geometries, smooth particle hydrodynamics (SPH) suffers from low convergence rates and a lack of turbulence modeling [6], Element Free Galerkin (EFG) methods need a background mesh to evaluate the stiffness matrix and field variables at nodal locations [7], and the Reproducing Kernel Particle Method (RKPM) is not applicable to time-dependent functions because it require symmetric kernels and is therefore limited to spatial functions and their derivatives [8]. Most PDE solvers therefore use meshes, with the field variables evaluated at points or within regions on the mesh.

Gradient-based optimization methods require the gradients of the field variables and/or performance measure to be calculated with respect to the PDE's parameters. In the continuous adjoint approach (also known as "optimize then discretize"), the gradient of the performance measure with respect to the parameters is calculated by constructing a constrained optimization problem with a Lagrangian functional, and re-arranging it to form the continuous adjoint PDEs [9–11]. These are usually solved on the same mesh as the direct PDEs. The continuous adjoint approach works but requires a new code with new boundary conditions, which tends to have different stability properties to the direct code. In the discrete adjoint approach (also known as "discretize then optimize"), the gradient of the performance measure with respect to the parameters is calculated through successive applications of the chain rule of differentiation, through the intermediate steps in the solution of the direct PDE. One way to calculate the derivatives at the intermediate steps is with symbolic differentiation using software such as Maple, Maxima, Mathematica, and Theano. This creates analytical derivatives which give valuable insight but are inefficient at runtime [12], particularly because the derivative expressions are usually much larger than the original expressions. Alternatively, these gradients can be calculated with automatic differentiation using software such as Tapenade and JaX. This computes numerical values of derivatives using symbolic rules of differentiation, as before. However, it calculates and stores these derivative values only around the solution to the PDE, rather than as general symbolic representations of the derivatives around any solution to the PDE. Automatic differentiation is therefore quicker than symbolic differentiation and ideally suited to gradient-based optimization, for which the only gradient required is that around the solution to the PDE. The discrete adjoint approach works well, but requires a direct code that has been designed to be differentiable.

The challenges of writing differentiable codes on mesh-based PDE solvers are exacerbated when the boundaries change shape because this causes the mesh nodes to move. In mesh-based PDE solvers, the field variables are held at points on the mesh. For accurate derivatives in the discrete adjoint approach, one therefore needs to calculate the gradients of the field variables with respect to the mesh positions, as well as with respect to the parameters. With either approach, one also has to introduce regularization (smoothing) of the boundary in order to avoid oscillations. There are two popular methods to model boundary deformation: free form deformation [13,14], and the level set method [15,16], both of which require regularization. Further, if the boundary deforms significantly, the mesh quality degrades and frequent re-meshing is required [17]. These challenges can be overcome but, again, significant code development is required.

The challenges described above can be avoided by using a mesh-free surrogate model to solve the PDE approximately. One option, which is not chosen here, is to train a surrogate model on the field variables calculated with solutions from a PDE solver at several different parameter values [18]. This surrogate model, which is computationally cheaper than the PDE solver, can then be used for optimization. This, however, requires a large amount of training data. For instance, [19] train a deep neural network with high-fidelity datasets to build a surrogate model for 2D laminar flow around airfoils. Then they optimize the airfoil geometry to minimize drag. A different option, explored in this paper, is to construct a surrogate model that satisfies the PDE at a cloud of points in the domain, and to simultaneously optimize as the model is trained.

1.2. Physics-informed neural networks

Deep neural networks are general models that can be trained to solve scientific computing problems [20,21]. A typical deep neural network consists of a multi-layer stack of simple modules. Each module computes nonlinear input–output mappings and this transformation increases both the selectivity and the invariance of the representation [22]. With multiple nonlinear layers, a deep learning model can approximate complicated functions [23] and, with proper training, any universal function [24]. The optimal parameters of a deep neural network are identified by minimizing a loss function, which quantifies the difference between the model and the expected outputs. Gradient-based optimization is used, which requires the gradients of the loss function with respect to the neural network's parameters. These are computed automatically with the backpropagation algorithm [25], which is more efficient than alternatives such as the finite-difference method [12,26]. Note that backpropagation is a subset of reverse-mode automatic differentiation.

Neural Networks are usually trained on labeled data (i.e., inputs and their expected outputs). Labeled data, whether acquired through experimental measurements or numerical simulations, is costly. On the other hand, a Physics-informed Neural Network (PINN) is trained by minimizing the deviations from the governing equations, initial conditions and boundary conditions at given points, known as collocation or sampling points [27]. This avoids the need for labeled data. A standard PINN has two components: a surrogate/approximator network and a residual block [28]. Typically, the surrogate network is a fully-connected feed-forward neural network. The residual block contains an embedded governing physics equation, whose residues are calculated by taking the output of the surrogate network. The weights and biases of the surrogate network are identified by minimizing the values of the residual block. Fig. 1 shows how the steady-state Navier–Stokes equations are encoded in the residual network.

A PINN performs well at inference or prediction [29] because it can successfully approximate complex, nonlinear, and high-dimensional functions. This provides a gridless alternative to traditional mesh-based numerical solvers and avoids the need for mesh generation, which is usually a significant burden in mesh-based CFD calculations. A PINN can be trained to approximately satisfy strong form PDEs at the collocation points. This PINN can then automatically compute derivatives of the model outputs with respect to the model inputs, which can be used for optimization.

PINNs were introduction by Raissi et al. [30] and have since been extended to solve forward and inverse problems with several different PDEs. As well as solving for the velocity and pressure fields alone [31], PINNs have been used to simulate vortex-induced vibrations [32] and to tackle ill-posed inverse fluid mechanics problems [20]. Furthermore, PINNs have be used to de-noise and obtain super-resolution of 4D flow magnetic resonance imaging [33], as well as to predict near-wall blood flow and quantify wall shear stress using sparse measurement data [34]. More recently, PINNs have been used to simulate turbulence directly with incomplete or noisy boundary conditions [35].

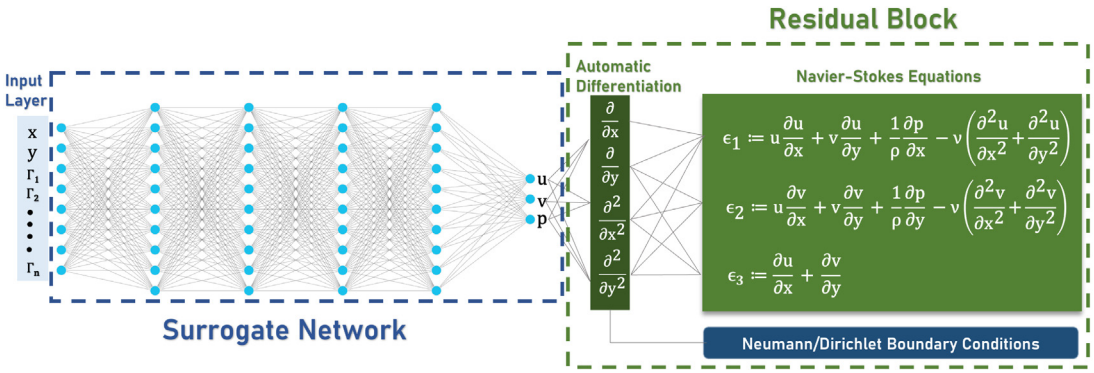


Fig. 1. The network architecture used to construct a PINN-based surrogate model with a set of extra input design parameters $(\Gamma_1, \Gamma_2, \ldots \Gamma_n)$. This network approximates residuals of the steady-state Navier–Stokes equations. The input collocation points (x, y) represent a flexible and mesh free representation of the domain. After training, the model approximates flow variables u, v, p at any new input points in the computational domain, as well as their derivatives with respect to the design parameters.

1.3. PINNs for PDE-constrained shape optimization

In this paper we propose a physics-informed deep learning framework that models steady incompressible flow around multiple sets of airfoils geometries. This framework is combined with quasi-Newton optimization to identify the optimal airfoil parameters for a given objective function. For flow around an airfoil, an objective function typically depends on the aerodynamic properties and the physical location of the boundary. Instead of a mesh, sample points are generated on the airfoil surface and within the computational domain. The loss function is the sum of the mean squared residuals of the governing partial differential equations calculated at these sample points, combined with boundary conditions. During the optimization process, the computational domain changes as the shape of the airfoil changes. Usually, physics-informed neural networks obtain solutions only for a single computational domain with a fixed geometry. This makes them unsuitable for rapid explorations of variable airfoil geometries. To overcome this challenge, we propose a new PINN architecture by constructing a heterogeneous input layer, which takes several design parameters along with collocation points as inputs. This extended input layer equips the PINN with a mechanism to extract geometric features of input computational domains and consequently can output flow fields for various computational domains. After being trained over the set of geometries as it optimizes, this PINN can, if desired, reliably predict the flow fields around varying airfoil shapes.

Although many prior studies have obtained approximate solutions of PDEs using PINNs, they have not used physics-informed deep learning to perform optimization. First, we evaluate the potential of PINNs to approximate the solutions of Navier–Stokes equations (NSEs). Our goal is to develop a new class of physics-based optimization approach, combining deep learning models with standard optimization algorithms such as the limited-memory Broyden–Fletcher–Goldfarb–Shanno optimization (L-BFGS) [36]. Second, we propose a physics-informed training method. To the best of our knowledge, existing PINNs have collocation points as the only inputs. Here we combine collocation points with a set of design parameters and input this composite data into a PINN. The trained PINN is able to model flows around a geometry defined by the design parameters within or close to the training range. The main strength of our PINN is that it represents the flow field for general geometries continuously. This makes the PINN particularly attractive for shape optimization because it avoids the need for separate training for each geometry. In this study, we apply this method to optimization of airfoils at low Reynolds number. The method is general, however, and could be used for other optimization problems such as vehicle shape optimization or process optimization for chemical productions.

In Section 2, we introduce the governing equations and the PINN architecture. In Section 3 we use the PINN to solve the Navier–Stokes equations and to optimize a single-parameter problem: the angle of attack of an airfoil. In Section 4, we apply the proposed approach to solve a multi-parameter shape optimization problem. In Section 5 we conclude and outline future challenges. The Python code used in this paper is available at [37].

2. Methodology

We solve a minimization/maximization problem of an objective function constrained by a set of partial differential equations:

$$\max_{U,D}(\min)\ J(U,D)\ \text{ subject to } g_c(U,D)=0,\ c\in\{1,2,\ldots,C\} \tag{1}$$

where $J(U,D)\in\mathbb{R}$ is an objective function, which will be defined in the next section. The PDE constraint $g_c(U,D)=0$ enforces the state variable U over N-dimensional space $D\in\mathbb{R}^N$, where c represents the number of constraints. The state variable U is a function of a design variable Γ and spatial variables $\boldsymbol{x}$. Design variables $\Gamma\in V$ live in the underlying high-dimensional parameter space $V\subset\mathbb{R}^d$. The spatial domain is labeled $\boldsymbol{x}\in\Omega\subset\mathbb{R}^s$.

The governing PDEs encoded in the PINN are the steady, incompressible Navier–Stokes equations for a Newtonian fluid:

$$\rho\boldsymbol{u}\cdot\nabla\boldsymbol{u}=-\nabla p+\mu\nabla^2\boldsymbol{u} \tag{2a}$$

$$\nabla\cdot\boldsymbol{u}=0\,, \tag{2b}$$

where ρ is the density, μ the dynamic viscosity, $\boldsymbol{u}(x,y)$ is the velocity vector, $p(x,y)$ the pressure field, and $x,y\in\Omega$ the spatial coordinate. Boundary conditions (BCs) can be expressed as

$$\mathcal{B}(\boldsymbol{x},\boldsymbol{u},p)=0,\ \boldsymbol{x}\in\partial\Omega\,, \tag{3}$$

where $\mathcal{B}$ is a precisely known operator determining boundary conditions on the boundary $\partial\Omega$.

In this study, the PINN is employed as a surrogate model to approximate the velocity and pressure fields as a function of space and of the parameters. We construct fully-connected neural networks by stacking layers of artificial neurons in which each layer $\boldsymbol{Z}_i$ performs an affine transformation on the previous layer with nonlinearity imposed via an element-wise activation function σ, typically a nonlinear function such as sigmoid, tanh and ReLu [23].

$$\boldsymbol{Z}_i=\sigma(\boldsymbol{W}_{i-1}^T\boldsymbol{Z}_{i-1}+\boldsymbol{b}_{i-1}), \tag{4}$$

where $\boldsymbol{W}_i$, $\boldsymbol{b}_i$ are the weight matrices and bias vectors for layer i. Our neural network is trained to obtain optimal weights and biases with the Adam algorithm, a variant of stochastic gradient descent. We use a loss function that includes the residuals from both the momentum and continuity equations:

$$\mathcal{L}_{pde}=\|\rho\boldsymbol{u}\cdot\nabla\boldsymbol{u}+\nabla p-\mu\nabla^2\boldsymbol{u}\|_\Omega+\beta\|\nabla\cdot\boldsymbol{u}\|_\Omega\,, \tag{5}$$

where β weights the residue arising from the continuity equation with that from the momentum equation. Boundary conditions are satisfied by minimizing

$$\mathcal{L}_{bc}=\|\mathcal{B}(\boldsymbol{x},\boldsymbol{u},p)\|_{\partial\Omega} \tag{6}$$

The total loss is the sum of the residue of the governing equations and the residue due to deviations from given boundary conditions, weighted by a hyperparameter λ:

$$\mathcal{L}(\boldsymbol{W}_i,\boldsymbol{b}_i)=\mathcal{L}_{pde}+\lambda\mathcal{L}_{bc} \tag{7a}$$

$$\boldsymbol{W}_i^*,\ \boldsymbol{b}_i^*=\underset{\boldsymbol{W}_i,\boldsymbol{b}_i}{\operatorname{argmin}}\,\mathcal{L}(\boldsymbol{W}_i,\boldsymbol{b}_i) \tag{7b}$$

The optimal weights and biases for each layer, $\boldsymbol{W}_i^*$, $\boldsymbol{b}_i^*$, are found by minimizing the total loss $\mathcal{L}$. We quantify losses with the L_2 norm.

The quality, and thus the predictive power, of a PINN depends on its network structure. Fig. 1 shows a diagram of this PINN. A point cloud is established in the domain and on the boundaries. The spatial coordinates of this point cloud are the inputs of the fully-connected neural network, and the outputs are the velocity, (u,v), and pressure, p, at those points. The main characteristics of the PINN are: (i) the input layer has thirteen neurons, two of which represent the x and y coordinates of a collocation point, and the other of which account for the eleven PARSEC design parameters; (ii) the network is fully connected with 20 hidden layers and 50 neurons per layer; each neuron is connected by affine linear maps between neurons in neighboring layers and then with nonlinear activation functions within neurons; (iii) the output layer contains three neurons; (iv) the sigmoid activation function ($\sigma(z)=1/1+e^{-z}$) is used; (v) the Adam optimizer is used; (vi) the loss function consists of the mean squared error of the PDE residual and boundary conditions, regularized with the squared residual evaluated at interior collocation points. The PINN is constructed with the open-source software TensorFlow [38].

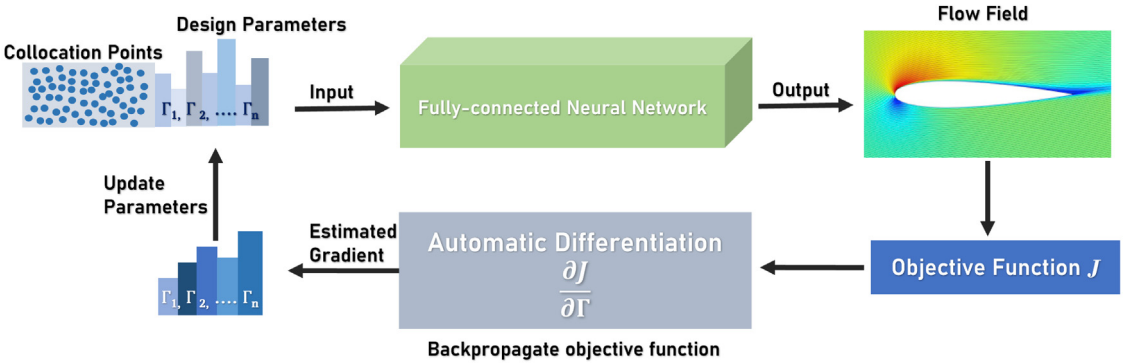


Fig. 2. The procedures of physics-informed training and parameter optimization based on backpropagation. First, an initial guess for the parameters, together with the spatial data points, are used as inputs to the neural network. Second, the flow field is approximated using the forward pass of the PINN to construct a map between the parameter inputs and the flow outputs. Third, the objective function is computed and its gradient with respect to the design parameters is obtained. Fourth, the parameters are updated and the process repeated in order to maximize the objective function.

2.1. Physics-informed training

Fig. 2 shows the key components to construct a surrogate model for adaptive optimization. First, a set of design parameters is chosen from the feasible space. No prior knowledge is assumed, apart from the parameters' lower and upper bounds. Sample points are randomly generated, for which the first two dimensions are for the spatial coordinates of the collocation points and the remaining dimensions are for the design parameters. The rationale behind these high-dimensional points is to save computational cost. Because neural networks interpolate well, we can use PINNs to approximate flow fields over a continuous range of designs. Instead of treating each design separately and sampling spatial points (2D points with x, y coordinates in this study) around each of them, we consider all designs together and train the PINN using data from the entire design space. Thus training data is generated by associating each spatial point with expanded dimensions accounting for design dimensions. Theoretically, each of these sample points represents a possible design. For a given airfoil, we therefore construct a point cloud between its boundary and the boundary of the computational domain. Points on the airfoil surfaces represent the geometry of the no-slip boundaries. Representing multiple airfoil shapes as point clouds instead of meshes can dramatically lower the memory requirement for calculations.

Second, these sample points are used as inputs of the PINN. Minimizing the PDE residuals measured at these sample points during the optimization process enforces the satisfaction of physics constraints, i.e., g_c in Eq. (1). Third, the flow variables (u, v, p) outputted from the surrogate model are used to compute the objective function values. Back-propagation through the PINN is used to calculate the gradient of the objective function (e.g., lift/drag ratio) with respect to the design parameters. In this algorithm, the gradient calculation proceeds backwards through the deep neural network and this backward flow of information facilitates efficient computation of the gradient at each layer [23]. Furthermore, reverse-mode automatic differentiation enables us to compute the gradient of a loss function with respect to the design parameters. This method is N-times more efficient than a finite-difference method, where N represents the number of parameters. Fourth, the design parameters are updated with a quasi-Newton algorithm. Sample points are added in order to increase accuracy as the optimal point is approached. The process is repeated. This means that the initial optimization steps are performed cheaply at low accuracy, while the final steps are performed more expensively at high accuracy. The data is normalized to ensure that the input variables have similar ranges of values (ranging from 0 to 1), which makes the optimization algorithm converge more quickly and accurately. A z-score normalization is used before all inputs are fed into the neural networks for training. By removing the mean and scaling it to unit variance, different input variables share the same distribution of mean = 0 and std = 1. The formula to normalize a point, x, is as follows: $x' = \frac{x-v}{\sigma}$. The obtained z-score represents the number of standard deviations away from the mean. Training on these normalized data is performed on a NVIDIA GeForce RTX 3080 graphics card with 16 Gigabytes of RAM. The Adam optimization algorithm is used to find the optimal neural network parameters [39]. The training process uses minibatch for loss estimates and

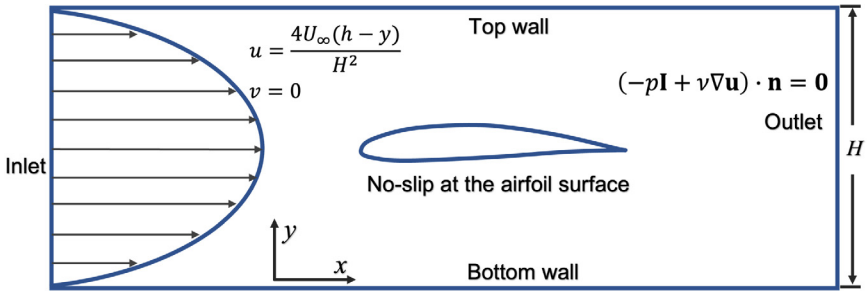


Fig. 3. Boundary conditions for the flow past an airfoil.

gradient computation. The training process starts with a high learning rate of 5.0×10^{-4}, followed by a reduced value of 10^{-6} at a later stage.

2.2. Quasi-Newton optimization

The proposed framework has an embedded surrogate model, which softly enforces the physics constraints, g_c in Eq. (1). The limited-memory BFGS (L-BFGS) gradient-based optimization algorithm is then employed to optimize the design parameters. The L-BFGS uses the first derivative of the objective function J with respect to the parameters in order to estimate the second derivatives (Hessian) (as shown in in Table 2), and thereby speed up optimization. The limited memory form can be implemented particularly efficiently [40]. The steps in this quasi-Newton algorithm are shown in Table 2. A maximum line search step length of 0.02 is employed to control the magnitude of the design variables and to avoid non-physical deformation of airfoil shapes. Optimization process terminates when the stopping criteria (the step or function change is less than the tolerance of 10^{-6}) are satisfied, or the maximum iteration limit of 50 is reached.

3. Single parameter optimization

The method is demonstrated first on laminar ($Re = 20$) flow over a Joukowski airfoil at different angles of attack. The challenge is to maximize the lift-to-drag ratio by optimizing the angle of attack. For this problem, the objective J in Eq. (1) is the lift-to-drag ratio under the constraints of the Navier–Stokes equations. A Joukowski airfoil is generated in the complex plane (z-plane) by applying the Joukowski transform to a circle in the ζ-plane, i.e., $z = \zeta + \frac{1}{\zeta}$. The horizontal spatial coordinates of the generated airfoil are normalized to [0, 1], running from the leading to the trailing edge. It is placed in a rectangular computational domain $\Omega = [-2.0, 1.0] \times [-1.0, 1.0]$. Taking a fluid density of $\rho = 1.0$ kg/m^3 and the dynamic viscosity of $\mu = 0.01$ kg/(m s), the fluid is characterized by the steady Navier–Stokes Eqs. (2). The computational domain and boundary conditions are shown in Fig. 3. A parabolic inlet velocity condition is used:

$$u = \frac{4U_\infty y(H-y)}{H^2}; \quad v = 0; \tag{8}$$

where H is the height of the rectangular domain, (u, v) are the horizontal and vertical velocity components, U_∞ is the maximum velocity of the inlet flow. No-slip conditions are imposed on the airfoil surface. A free boundary condition is imposed on the outflow, which takes the form

$$(-p\mathbf{I} + \nu\nabla\mathbf{u}) \cdot \mathbf{n} = 0, \tag{9}$$

where $\mathbf{I}$ is the 2×2 identity matrix, p is the pressure and $\mathbf{n}$ denotes the outward unit normal to the rightmost boundary. For a maximum velocity of $U_\infty = 0.3$, the parabolic profile results in a mean velocity $\bar{U} = 0.2$.

Fig. 1 shows the architecture of the fully connected network, which contains 10 hidden layers and 50 neurons per layer. The input layer consists of three neurons, taking realizations from the physical domain (i.e., x, y) and a single optimization variable, the angle of attack (AOA). There are three neurons in the output layer, representing flow velocity (u, v) and pressure (p). Many conventional approaches require, for every angle of attack, training

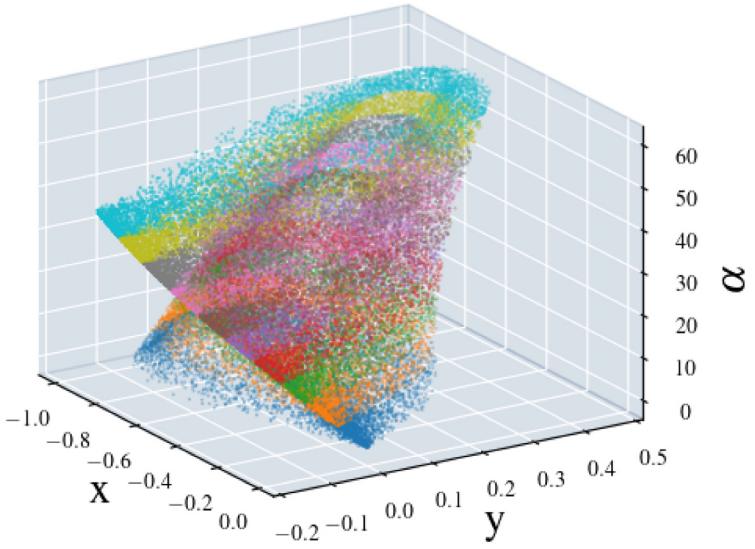


Fig. 4. Sampled collocation points along airfoil surfaces at angle of attack $\alpha = 0 - 60°$. Different colors represent different angles of attack.

with new collocation points in order to find the optimum angle of attack. To explore the design space completely, training would be required at many angles of attack, which would be computationally inefficient.

Instead, here we combine the collocations points and the angles of attack together into a single point cloud to create a heterogeneous exploration space in which the first two dimensions are the collocation points and the third dimension is the angle of attack, shown in Fig. 4. When training the PINN, each batch contains 20,400 randomly-sampled points from this three dimensional space, of which 18000 points are sampled inside the computational domain (including the airfoil surface) and 2400 are boundary points, generated using Latin hypercube sampling at the other boundaries (wall, inlet and outlet). 5000 collocation points are sampled near the airfoil surface to impose stricter constraints on the flow there. Training takes 18 h and an optimization can be completed within 10 min. Building up a surrogate model successively by considering all angles of attack simultaneously greatly reduces the computation time compared with considering each angle of attack separately. The loss history during training is shown in Fig. 5, showing residual data every 100 iterations. This contains the total residual, the PDE residual (5) in the domain, and the PDE residual at the boundaries. Fig. 5 shows that the predicted results obey all boundary conditions well.

The PINN is trained on collocation points over a wide range of angles of attack (AOA= $0-50°$). After training, the PINN can accurately predict the flow fields in this training range, if desired, as illustrated in Fig. 6 for cases of AOA= $10°, 20°, 30°, 40°$. The predicted velocity (u, v) and pressure fields (p) obtained from PINN agree well with the numerical results from FEniCS, an open-source computing platform for solving partial differential equations [41]. In FEniCS, the computational domain is discretized with a mesh of triangular elements. Taylor–Hood elements are employed to build the test and trial function spaces. This trained PINN can predict flows successfully over airfoils at any angle of attack within the training range, not just the four illustrative cases shown in Fig. 6. Further, this figure also shows the discrepancy between the PINN predictions and CFD simulations. These error contours show the pointwise error distribution within the computational domain. Here we calculate the relative error using

$$\text{Relative Error} = \left| \frac{Q_{PINN} - Q_{CFD}}{Q_{CFD} + \epsilon} \right|, \tag{10}$$

where Q is the quantity of interest and $\epsilon = 10^{-4}$ is a small number to avoid having a zero denominator. For the velocity fields, the maximum error is observed at the leading edge of the airfoil for all angles of attack because the velocity field has highest gradients there and the point cloud is not sufficiently fine to resolve it well. This has also been found in the neural network predicted flows around a NACA 0028 airfoil [42]. The PINN used here predicts the velocity fields (maximum error 10^{-2}) more accurately than the pressure field (maximum error 10^{-1}).

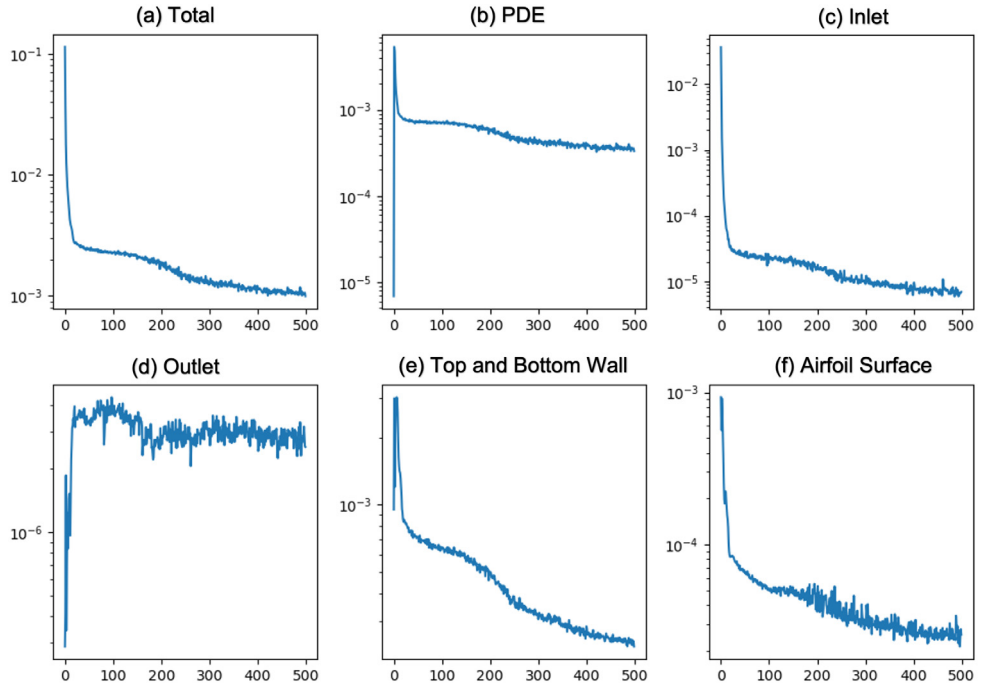


Fig. 5. Different categories of residual as a function of iteration during training: (a) total residuals, (b) PDE, (c) inlet boundary, (d) outlet boundary, (e) top and bottom wall, and (f) airfoil surface. Specific boundary conditions can be found in Fig. 3.

The discrepancy in the pressure field is located in a small region at the trailing edge but, as will be shown next, this has little influence on the lift-to-drag ratio, which is the objective functional for optimization.

For an airfoil with fixed shape at a given Reynolds number, the angle of attack determines the lift-to-drag ratio. In this example, the objective is to maximize the lift-to-drag ratio by changing the angle of attack. The drag and lift forces are calculated from the pressure and velocity fields taken from the PINN predictions. The lift force F_l and drag force F_d are computed as

$$F_l = N\cos\alpha - A\sin\alpha, \quad F_d = N\sin\alpha + A\cos\alpha \tag{11}$$

where

$$N = \int_{top} (-p\cos\theta - \tau\sin\theta)\, ds + \int_{bot} (p\cos\theta - \tau\sin\theta)\, ds, \tag{12}$$

$$A = \int_{top} (-p\sin\theta + \tau\cos\theta)\, ds + \int_{bot} (p\sin\theta + \tau\cos\theta)\, ds. \tag{13}$$

Here A, N are the axial and normal forces, parallel and perpendicular to the airfoil chord line, α represents the angle of attack and τ is the magnitude of the wall shear stress, θ is the angle between the airfoil surface and the chord line, taking a positive value in the clockwise direction. The differential section ds of the perimeter runs from the trailing edge to the leading edge over the top surface (top) and ds runs in the opposite direction over the bottom surface (bot). Using these equations, we can compare the lift force F_l and drag force F_d between PINN predictions and FEniCS simulations, shown in Fig. 7. As required, there is good agreement between the two around the area of interest, which is the angle of attack at which the lift to drag ratio is maximum.

Having confirmed that the PINN can compute the lift to drag ratio sufficiently accurately for our purposes, we restart the training and optimization process, using a gradient descent algorithm to find the AOA that maximizes the lift to drag ratio (see Fig. 8). The gradient of the lift-to-drag ratio with respect to the design parameter is obtained with reverse-mode automatic differentiation, which is a natural feature of a PINN and requires no further coding. Fig. 9 shows the lift-to-drag ratio and its gradient during this optimization process, which starts from AOA=5°. It

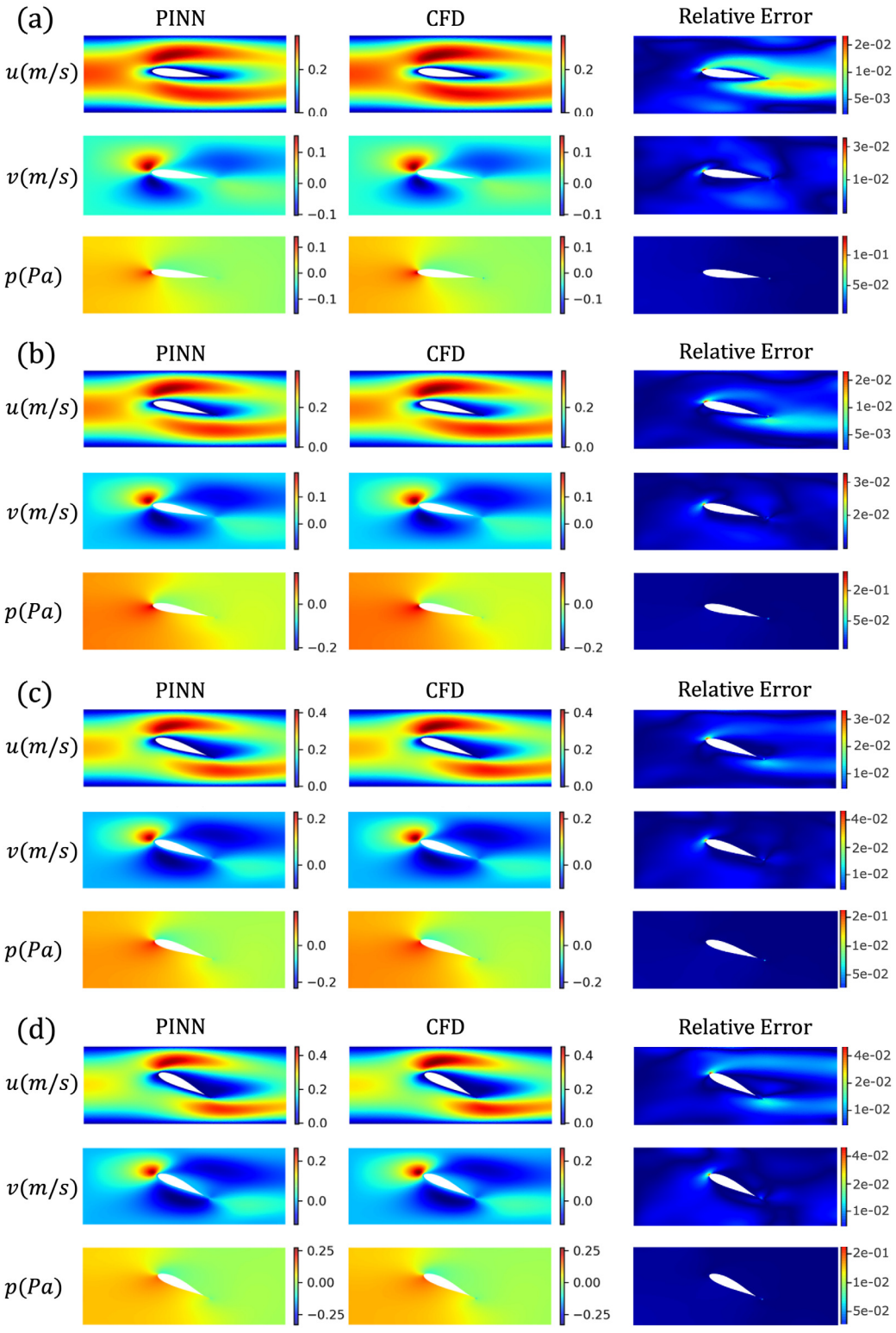


Fig. 6. PINN predictions, CFD simulations, and their discrepancies for the velocity and pressure fields at four angles of attack (AOA). (a) AOA= 10°, (b) AOA= 20°, (c) AOA= 30°, (d) AOA= 40°.

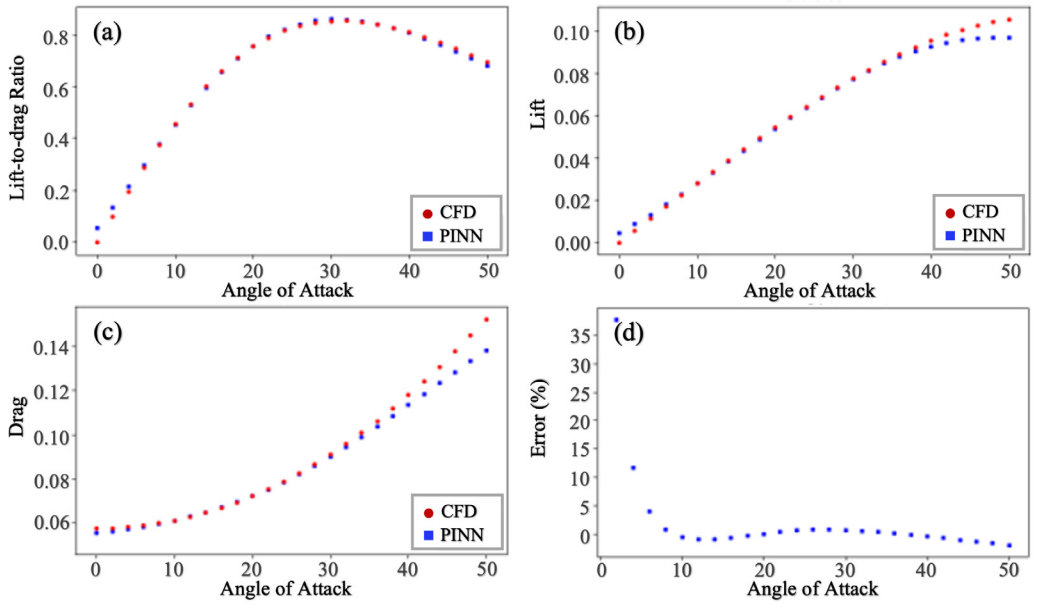


Fig. 7. (a) Lift-to-drag ratio, (b) lift and (c) drag calculated with the PINN (blue squares) and CFD (red dots). (d) The discrepancy between lift-to-drag ratio between the PINN and that computed with CFD.

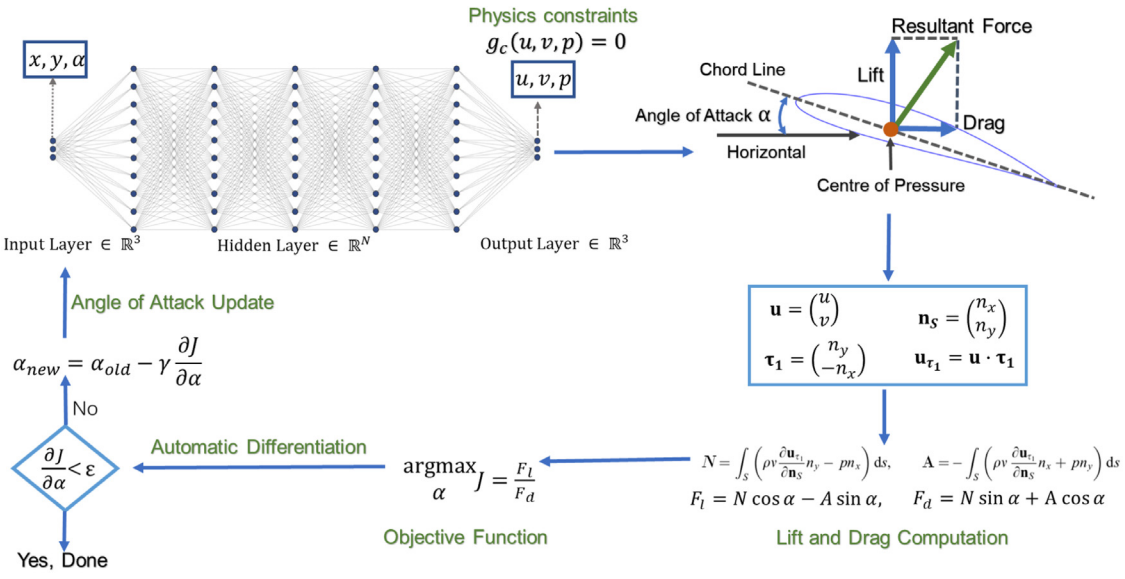


Fig. 8. A schematic diagram of the process used to find the optimal angle of attack.

is evident that the optimization process is successful. The largest lift-to-drag ratio of 0.92 is found at AOA=30.7° where the gradient and reaches its stop criteria, i.e., gradient less than 10^{-4}.

To ensure that the optimization results are reliable, the discrepancy between the PINN results and FEniCS result is calculated again for the optimal AOA. Fig. 10 shows the flow fields from neural network prediction agree well

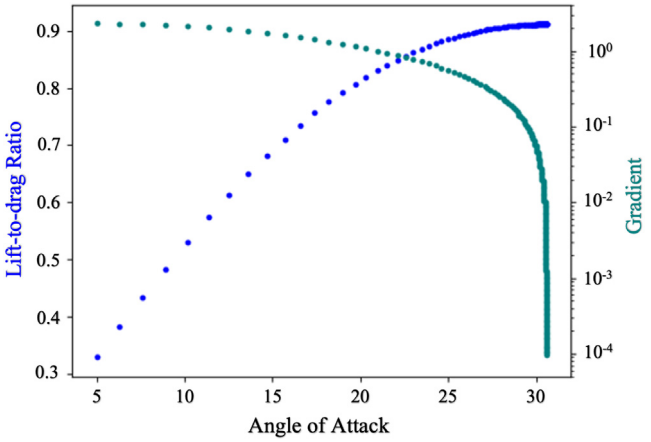


Fig. 9. Lift-to-drag ratio (blue) and its gradient (green) during the simultaneous PINN training and optimization process. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

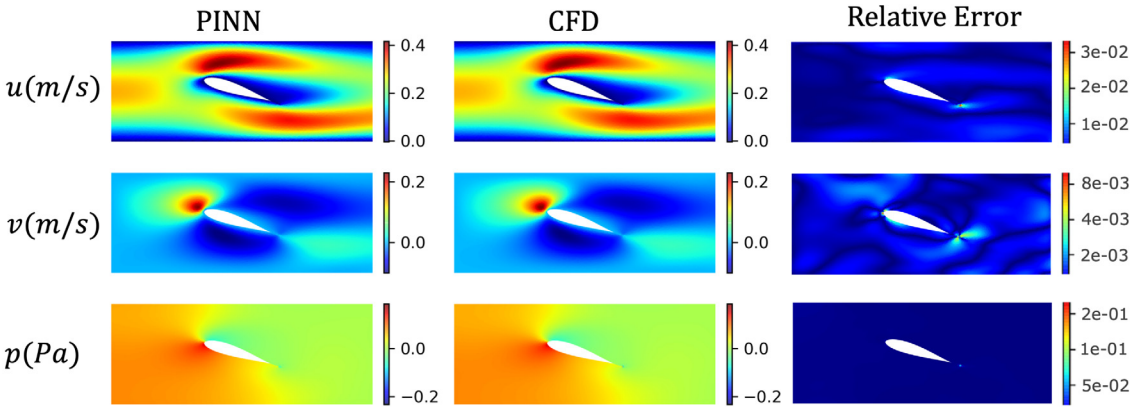


Fig. 10. As for the case of the optimal AOA=30.7°, the PINN predictions agree well with the CFD solutions.

with the CFD results, as evidenced by the small discrepancy. The PINN outputs a flow field immediately for a given AOA and, although approximate, is sufficiently accurate to be useful for this optimization process.

4. Multi-parameter optimization

In this section, the developed neural network framework is applied to a multi-parameter optimization problem: finding the set of 11 PARSEC parameters that maximize the lift-to-drag ratio of an airfoil. The lift-to-drag ratio remains the objective function but the design variables are the PARSEC parameters. Here the PARSEC parameterization translates Cartesian coordinates into 11 parameters ($\Gamma_1, \Gamma_2, \ldots, \Gamma_{11}$) to define an airfoil shape [43,44]. These parameters represent the physical characteristics of a given airfoil, such as leading edge radius, curvature, thickness, maximum thickness abscissa. PARSEC parameterization is a physically intuitive method because it enables a designer to link typical airfoil parameters to the characteristics of a given airfoil, such as leading edge radius, airfoil thickness, and trailing edge angle. Fig. 11 shows the PARSEC parameterization definition. The physical meanings of the symbols are summarized in Table 1. The airfoil geometry characterized by PARSEC parameterization can be defined using the following polynomial functions

$$y_{up} = \sum_{j=1}^{j=6} A_{up\,j} x^{j-\frac{1}{2}}, \quad y_{lw} = \sum_{j=1}^{j=6} A_{lw\,j} x^{j-\frac{1}{2}} \tag{14}$$

Table 1
List of the PARSEC parameters.

PARSEC parameter	Geometry parameter	Definition
Γ_1	r_{le}	Leading edge radius
Γ_2	x_{up}	Upper crest position in horizontal coordinates
Γ_3	y_{up}	Upper crest position in vertical coordinates
Γ_4	γ_{up}	Upper crest curvature
Γ_5	x_{lw}	Lower crest position in horizontal coordinates
Γ_6	y_{lw}	Lower crest position in vertical coordinates
Γ_7	γ_{lw}	Lower crest curvature
Γ_8	y_{te}	Trailing edge offset in vertical sense
Γ_9	Δy_{te}	Trailing edge thickness
Γ_{10}	α_{te}	Trailing edge direction
Γ_{11}	β_{te}	Trailing edge wedge angle

Table 2
Quasi-Newton optimization algorithm.

Inputs:	Objective function J, initial guess $\Gamma_0 \in D$, starting Hessian $B_0 \in \mathbb{R}^{d\times d}$, and tolerance parameter ϵ.
Goals:	Find local minimizer for the optimization problem (1).
Step i:	Given Γ_i, B_i, 1. Identify search direction by solving $B_i p_i = -\nabla J(\Gamma)$; 2. Conduct a line search $\eta_i = \underset{\eta}{\text{argmin}}\, J(\Gamma_i + \eta p_i)$; 3. Obtain new design parameter $\Gamma_{i+1} = \Gamma_i + \eta_i p_i$; 4. Update the approximate Hessian. $B_{i+1} = \varphi(B_i, \Gamma_i, \Gamma_{i+1}, \nabla J(\Gamma_i), \nabla J(\Gamma_{i+1}))$. (16)
Convergence check:	When $\lvert\Gamma_{i+1} - \Gamma_i\rvert < \epsilon$, stop the iteration. Otherwise continue until this condition is satisfied.

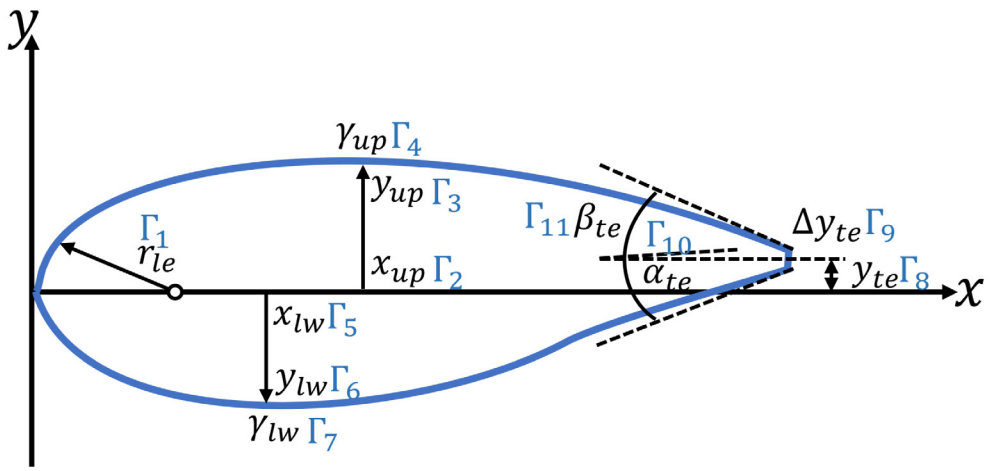


Fig. 11. Diagram of the PARSEC parameters.

where: x is the horizontal coordinate normalized by chord length to become $[0, 1]$; y_{up} and y_{lw} are the vertical coordinates of the upper and lower surface respectively; A_{up} and A_{lw} are the coefficients computed from the eleven PARSEC parameters. Details of the determination of these parameters can be found in [45].

As shown in Fig. 12, each training point has 13 dimensions: two for the x, y coordinates of the collocation points and the remaining 11 dimensions for the PARSEC parameters. In the neural network design, we therefore need to use 13 neurons in the input layer to represent the thirteen-dimensional sample points. The upper and lower limits of the eleven PARSEC parameters used for training are shown in Table 5. A wide range of airfoil shapes can be

Table 3
Parameter optimization procedure.

Inputs:	Physical equations (2), objective function J, training set $\mathcal{S} \subset D$ (randomly chosen), quasi-Newton optimization algorithm described in Table 2.
Goals:	Find global minimizer for the optimization problem (1).
Step 1:	For a given training set $\mathcal{S} = \{\tilde{\Gamma}_i\}$, with $\tilde{\Gamma}_i = \{\Gamma_{i1}, \Gamma_{i2}, \Gamma_{i3}, \ldots, \Gamma_{i11}\}$ for $1 \leq i \leq N$, generate the deep neural network surrogate map.
Step 2:	Draw M random parameter sets $\tilde{\Gamma}_k$. For each $1 \leq k \leq M$, run the optimization algorithm in Table 2 with objective function J_k and starting values $\tilde{\Gamma}_k$. Discard the parameter set with unfavorable objective functions according to the design threshold and retain the good sets for further updates until all cases converge to a set $\bar{\Gamma}_k$ of approximate minimizers to the optimization problem.

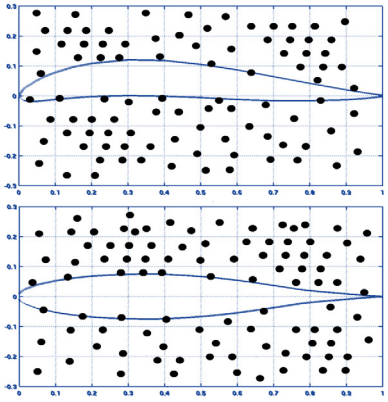
created using this method. The batch size is set to 30,000, including 8000 points on the boundaries. Instead of simply relying on 12,000 training points in the computational domain, we augment the training batch by adding 10,000 refinement points to achieve better accuracy for regions with large errors. This adaptive sampling strategy is used in order to improve the training efficiency. We add additional sample points for every 100 iterations according to the distribution of PDE residuals. The number of added points is proportional to the residual level, which measures the errors in the satisfaction of the Navier–Stokes equations. Consequently, a higher density of training points is used in regions with poor predictions. The regions with large number density carry more weight in the overall loss function. This therefore improves predictions where the residuals were high. This strategy shares some similarities with importance sampling, which has been proved to facilitate training [39]. During network training, points generated from a Gaussian distribution are added to the training dataset. To avoid a GPU memory leak, the size of adaptively added points is capped at 10,000. This adaptive sampling inevitably increases the training cost. 75 h is required to complete the training, while an optimal design can be found in 35 min (see Table 3).

This adaptive sampling strategy facilitates training and improves accuracy of the PINN. The progressive improvement of the flow field predictions can be seen in Fig. 13. The first four columns show the flow fields obtained from PINN predictions as the iterations progress, and the last plots are the numerical results from CFD. In the early stage of training, the PINN gives poor predictions of the flow fields. Fig. 14 shows how, as the training continues, new sample points (pink and blue dots) are created around the existing training points that have high residuals (red dots). Since additional training points are concentrated in areas with large residuals, the error can be reduced efficiently during the optimization process. Further training leads to better satisfaction of the Navier–Stokes equations, and thus a reduced number of added points (Fig. 14(b)). Fig. 15 shows the pointwise residuals in the continuity and momentum equations for two different airfoils with $\Gamma_1 = 0.014$ and $\Gamma_1 = 0.017$, respectively. These residuals are $\sim 10^{-4}$ to 10^{-3}, showing that the pressure and velocity fields predicted by the PINN satisfy these governing equations reasonably well. These small residuals arising from the flow fields for two different airfoil shapes demonstrate that the incorporation of the Navier–Stokes equations into the surrogate model enables the PINN to make physical predictions.

To illustrate the PINN's suitablility for optimization of lift to drag ratio, flows around the airfoils with the range of shapes defined in Table 4 are calculated. Fig. 16 compares the flow fields from the PINN with those from the CFD modeling of four different airfoils. The relatively good predictions assure us that the PINN can output reliable flow fields for arbitrary airfoil shapes, if close to those examined. This is essential for calculation of the objective function during the optimization process. Here the objective function is the lift-to-drag ratio, and the PINN calculates the gradient of this with respect to the PARSEC parameters. The predicted lift and drag forces, compared against the FEniCS calculations, are shown in Fig. 17 for the test airfoils in Table 4. These are predicted to within 1% by the PINN.

Fig. 18 shows the lift to drag ratio during the last 11 iterations of the optimization process, during which the lift to drag ratio reaches its maximum at 0.761. The flow around the optimized airfoil is compared with the CFD results using the finite element method, as shown in Fig. 19. The optimal airfoil has higher thickness and a longer suction surface than the initial guess. The optimization tends to increase the airfoil thickness for a higher lift. This trend is also observed in the optimization of the NREL S809 airfoil at $Re = 2 \times 10^6$ using a genetic algorithm [45].

Training Dataset	Point 1	Point 2	Point 3	Point 4	• • •	Point N
Spatial Coordinates	x_1	x_2	x_3	x_4	• • •	x_N
	y_1	y_2	y_3	y_4	• • •	y_N
Design Parameters	Γ_1^1	Γ_2^1	Γ_3^1	Γ_4^1	• • •	Γ_N^1
	Γ_1^2	Γ_2^2	Γ_3^2	Γ_4^2	• • •	Γ_N^2
	Γ_1^3	Γ_2^3	Γ_3^3	Γ_4^3	• • •	Γ_N^3
	⋮	⋮	⋮	⋮		⋮
	Γ_1^m	Γ_2^m	Γ_3^m	Γ_4^m	• • •	Γ_N^m



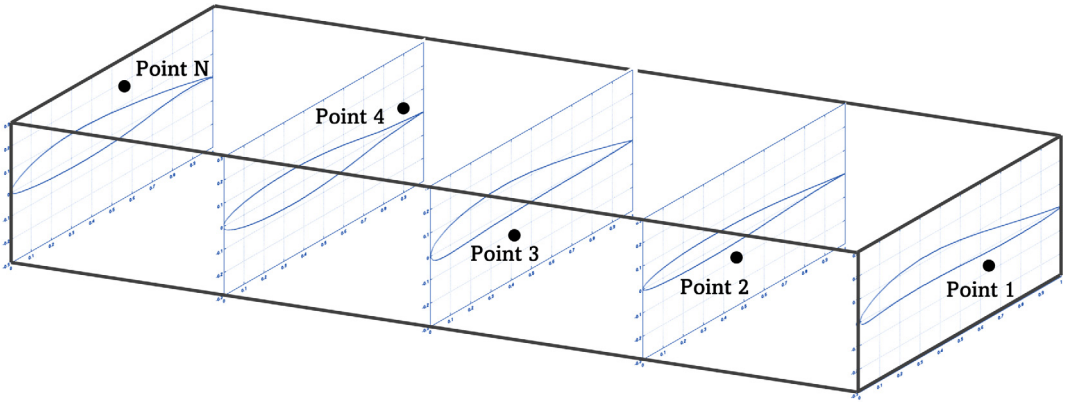


Fig. 12. High-dimensional inputs for the PINN in surrogate modeling. (a) The upper table shows input parameters for each training point, 1 to N, and an illustration of the point cloud in (x, y)-space for two different training points, which correspond to two different airfoils. (b) The lower figure illustrates airfoil shapes at the different training points.

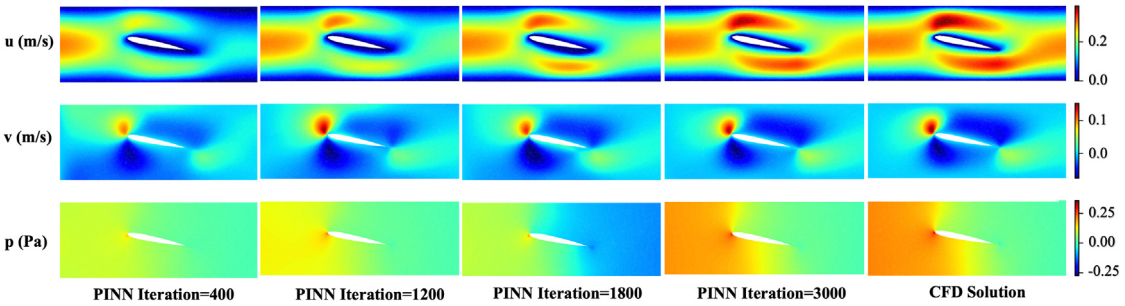


Fig. 13. The flow fields calculated with the PINN as the optimization progresses.

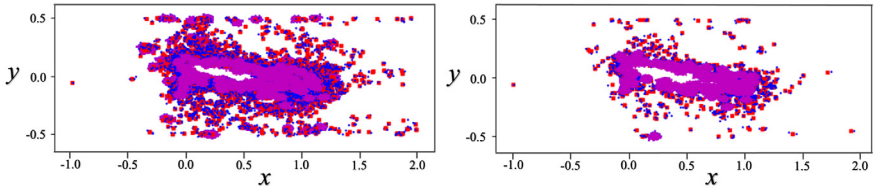


Fig. 14. Illustration of the adaptive sampling strategy during PINN training at (a) 500 iterations and (b) 2000 iterations. Red points denote existing sample points with total residuals above the preset threshold. Blue points denote points added due to residuals in the momentum equation. Pink points denote points added due to residuals in the continuity equation.

Table 4
PARSEC parameters for the test airfoils with Γ_1 as index. These airfoils are shown in Fig. 16 and Fig. 17.

Test Case	Index	Γ_1	Γ_2	Γ_3	Γ_4	Γ_5	Γ_6	Γ_7	Γ_8	Γ_9	Γ_{10}	Γ_{11}
Airfoil 1	0.014	0.014	0.282	0.056	−0.467	0.282	−0.056	0.467	−0.0013	0.001	−2.594	8.6333
Airfoil 2	0.015	0.015	0.303	0.060	−0.500	0.303	−0.060	0.500	0.000	0.001	−2.779	9.250
Airfoil 3	0.017	0.017	0.343	0.068	−0.567	0.343	−0.068	0.567	0.0027	0.001	−3.145	10.483
Airfoil 4	0.018	0.018	0.363	0.072	−0.600	0.363	−0.072	0.600	0.004	0.001	−3.335	11.100

Table 5
The upper and lower limits of the PARSEC parameters.

PARSEC parameter	Geometry description	Lower limit	Upper limit
Γ_1	Leading edge radius	0.012	0.018
Γ_2	Upper crest position in horizontal coordinates	0.242	0.363
Γ_3	Upper crest position in vertical coordinates	0.048	0.072
Γ_4	Upper crest curvature	−0.6	−0.4
Γ_5	Lower crest position in horizontal coordinates	0.242	0.363
Γ_6	Lower crest position in vertical coordinates	−0.072	−0.048
Γ_7	Lower crest curvature	0.4	0.6
Γ_8	Trailing edge offset in vertical sense	−0.004	0.004
Γ_9	Trailing edge thickness	0.001	0.005
Γ_{10}	Trailing edge direction	−3.335	−2.223
Γ_{11}	Trailing edge wedge angle	7.4	11.1

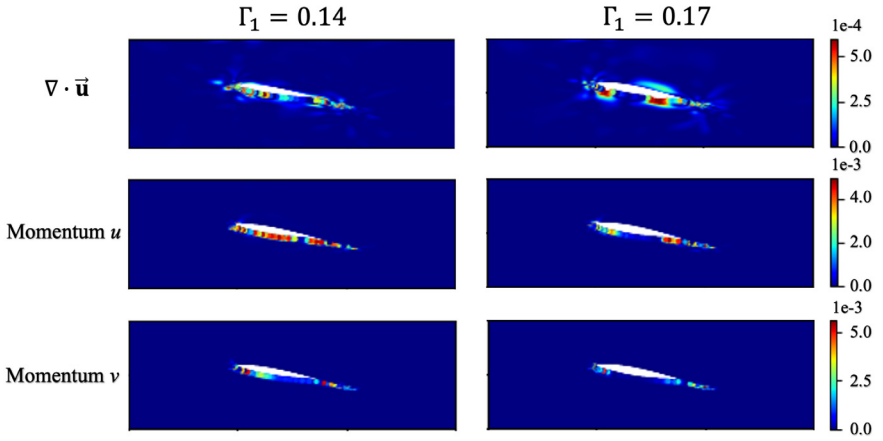


Fig. 15. Distributions of residuals of the continuity and momentum equations for airfoils with (a) $\Gamma_1 = 0.014$ and (b) $\Gamma_1 = 0.017$.

The optimization process starts from an initial guess, updates the PARSEC parameters using computed gradient information, and gradually moves towards a local optimum. Our surrogate model is advantageous and computationally inexpensive because it does not explore solutions in the entire high-dimensional design space. This strategy is especially suitable for high-dimensional problems, or when the intermediate simulations are computationally expensive. For low-dimensional problems, this method can be further improved by adopting subset selection methods, which show promising performance due to the fact that they can select the most important features and identify the corresponding model parameters [18]. Another point to emphasize is that while we are interested in generating a good surrogate model capable of accurately predicting the optimum, the constructed model is still able to make good predictions of flow fields for intermediate cases, as illustrated in Fig. 16.

The results shown above are obtained with a gradient-based optimization from a single starting point. We now perform the same analysis from ten different starting points. These starting points are randomly generated within the design space, shown in Table A.8 in the appendix. The achieved optimization results, i.e., optimal PARSEC parameters and achieved lift-to-drag ratios are shown in Table 6. These ratios are close to the desirable value of

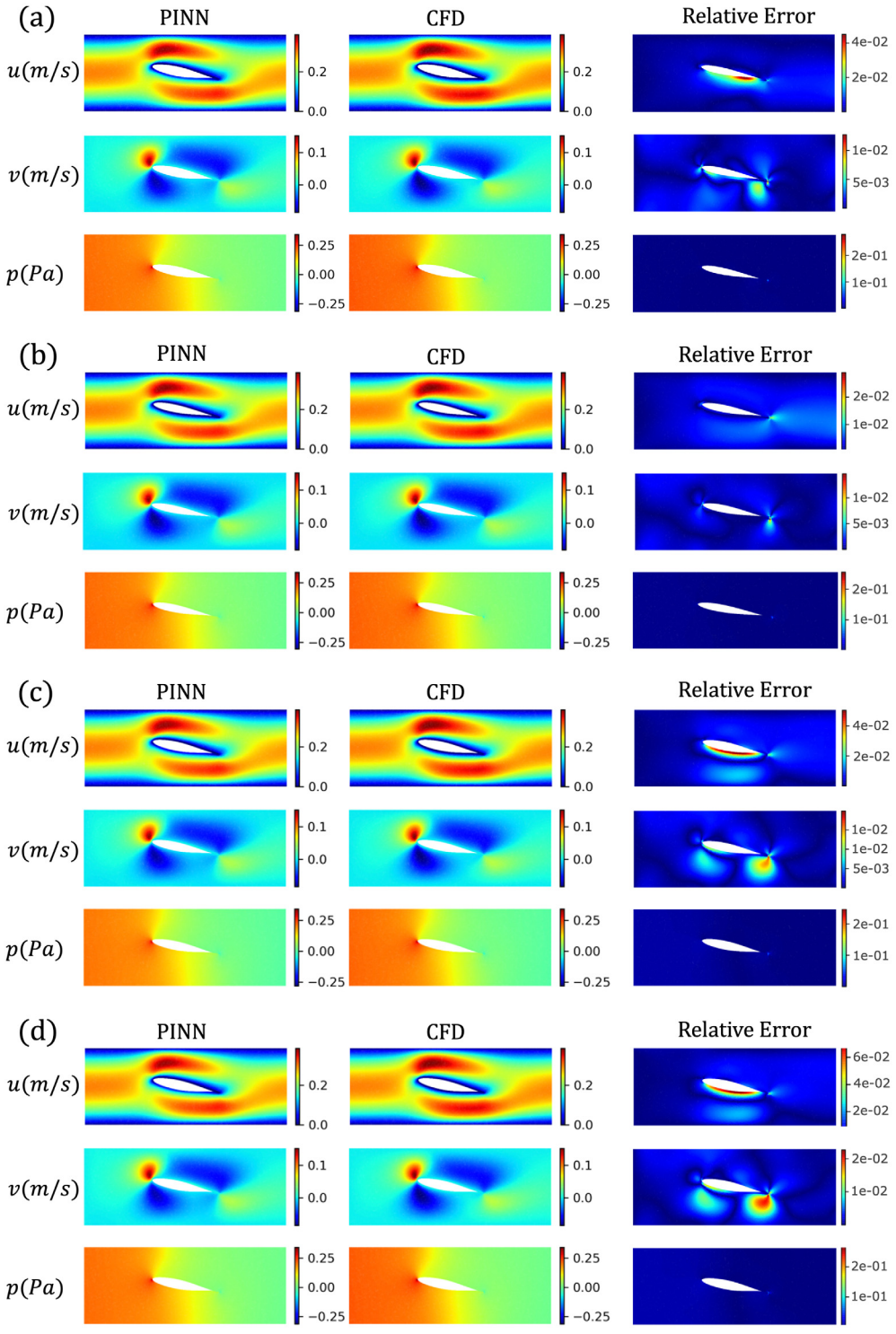


Fig. 16. Comparison between flow fields calculated with the PINN and CFD for various airfoils determined by 11 PARSEC parameters. (a) Airfoil 1 determined by the PARSEC parameter $\Gamma_1 = 0.014$. (b) Airfoil 2 determined by the PARSEC parameter $\Gamma_1 = 0.015$. (c) Airfoil 3 determined by the PARSEC parameter $\Gamma_1 = 0.017$. (d) Airfoil 4 determined by the PARSEC parameter $\Gamma_1 = 0.018$.

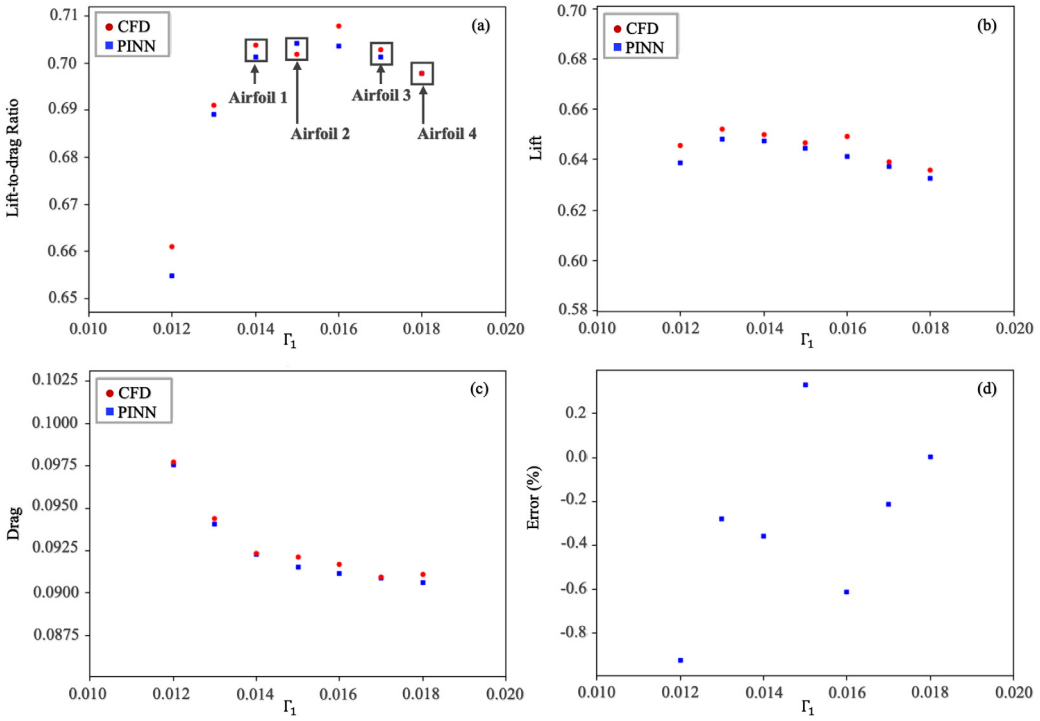


Fig. 17. (a) Lift-to-drag ratio, (b) lift and (c) drag calculated with the PINN (blue squares) and CFD (red dots). (d) The discrepancy between lift-to-drag ratio between the PINN and that computed with CFD. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

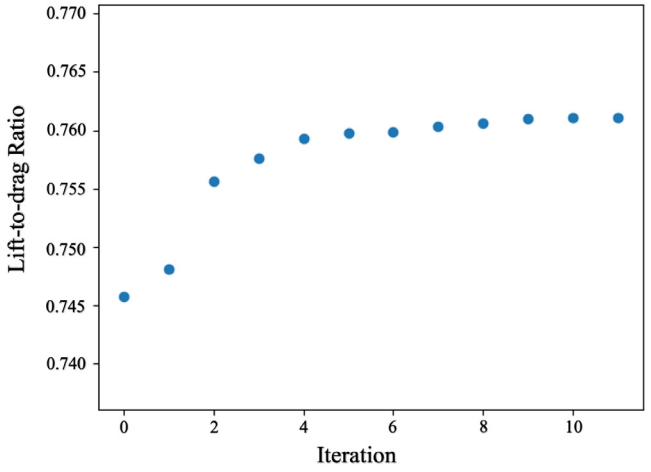


Fig. 18. Lift-to-drag ratio variations during the last few optimization iterations.

0.761 but are not the same, showing that the process has found ten local optima. Nevertheless, most cases yield optimal lift-to-drag ratios between 0.75 and 0.761. This consistency, despite starting from randomly-selected initial PARSEC parameters, indicates that the process has converged to the region of a global optimum and that the small differences in L/D arise from closely-spaced local optima in this region.

For comparison, we also perform CFD simulations for 12,000 airfoils with different PARSEC parameters, some of which are sampled around the optimal parameter set obtained from the PINN, the rest of which are randomly sampled within the design space. Fig. 20 shows the lift-to-drag ratio for these airfoils, as a function of two of

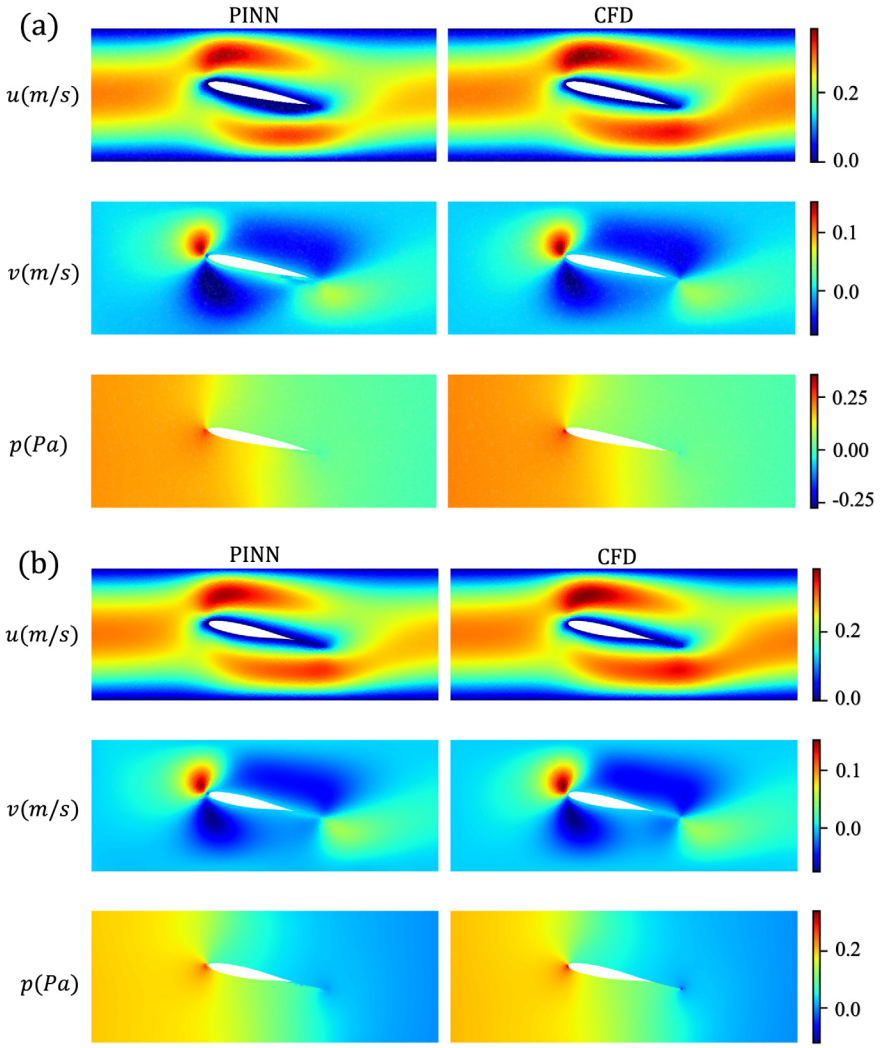


Fig. 19. (**a**) Comparison of flow fields around the initial airfoil shape calculated with the PINN (left) and CFD (right). (**b**) The same comparison for the optimized airfoil.

Table 6
Optimized PARSEC parameters and lift-to-drag ratio starting from the ten random guesses shown in Table A.8.

Γ_1	Γ_2	Γ_3	Γ_4	Γ_5	Γ_6	Γ_7	Γ_8	Γ_9	Γ_{10}	Γ_{11}	Lift-to-drag ratio
0.0167	0.263	0.057	−0.518	0.252	−0.0481	0.517	−0.0043	0.00172	−2.441	10.159	**0.7543**
0.0156	0.253	0.056	−0.516	0.249	−0.0478	0.513	−0.0036	0.00182	−2.475	10.173	**0.7469**
0.0162	0.259	0.055	−0.509	0.258	−0.0478	0.506	−0.0035	0.00168	−2.464	10.192	**0.7526**
0.0155	0.247	0.052	−0.524	0.258	−0.0465	0.518	−0.0030	0.00175	−2.452	10.203	**0.7607**
0.0161	0.262	0.052	−0.522	0.248	−0.0478	0.525	−0.0035	0.00178	−2.469	10.262	**0.7636**
0.0152	0.253	0.056	−0.516	0.258	−0.0472	0.512	−0.0038	0.00171	−2.389	10.170	**0.7491**
0.0160	0.259	0.055	−0.509	0.248	−0.0478	0.506	−0.0029	0.00168	−2.464	10.187	**0.7443**
0.0147	0.262	0.051	−0.518	0.253	−0.0483	0.509	−0.0041	0.00183	−2.441	10.253	**0.7582**
0.0153	0.259	0.053	−0.522	0.258	−0.0465	0.522	−0.0030	0.00183	−2.384	10.212	**0.7595**

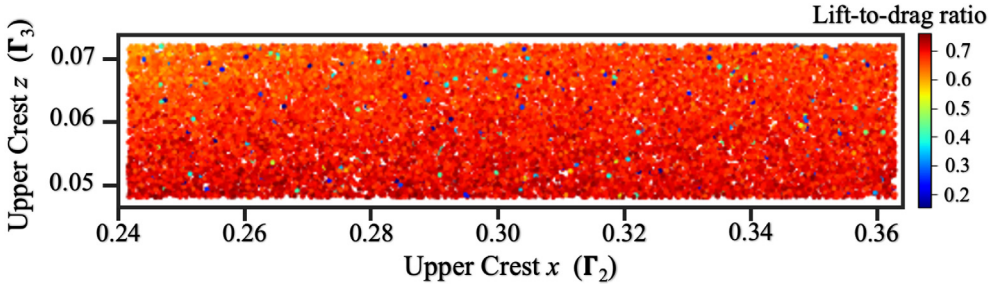


Fig. 20. Lift-to-drag ratio for airfoils characterized by various PARSEC parameters, computed with FEniCS.

Table 7
The optimal PARSEC parameters obtained with the PINN and with FEniCS.

PARSEC parameter	Geometry description	FEniCS	PINN
Γ_1	Leading edge radius	0.0159	0.0152
Γ_2	Upper crest position in horizontal coordinates	0.262	0.257
Γ_3	Upper crest position in vertical coordinates	0.050	0.053
Γ_4	Upper crest curvature	−0.478	−0.514
Γ_5	Lower crest position in horizontal coordinates	0.260	0.251
Γ_6	Lower crest position in vertical coordinates	−0.0484	−0.0486
Γ_7	Lower crest curvature	0.532	0.516
Γ_8	Trailing edge offset in vertical sense	−0.0033	−0.0032
Γ_9	Trailing edge thickness	0.00174	0.00179
Γ_{10}	Trailing edge direction	−2.366	−2.418
Γ_{11}	Trailing edge wedge angle	10.75	10.23
L/D	**Lift-to-drag ratio**	**0.757**	**0.761**

the PARSEC parameters (Γ_2, Γ_3). Most of the candidate airfoils have lift-to-drag ratios larger than 0.6 because the sample points were taking near the optimal PINN design, which gives 0.761. Some airfoils have very poor aerodynamic performance, however, as indicated by lift-to-drag ratios less than 0.4. The maximum value is 0.757, which is close to the PINN optimization result of 0.761. Table 7 compares optimal PARSEC parameters using PINNs with the best found from the 12,000 CFD simulations. The parameters are close to each other, showing that the PINN optimization has performed well. Slight differences are to be expected because the solution method is different.

5. Conclusion

In this paper, we develop and apply a physics informed neural network (PINN) that simultaneously calculates the flow around an airfoil and optimizes the airfoil's parameters to maximize the lift to drag ratio. The framework represents the geometries using mesh-free point clouds in a high dimensional space. This approach can handle complex geometries. We accelerate the training with adaptive sampling, in which additional sample points are added to the regions with large errors in order to improve the accuracy of predictions. We use the L-BFGS algorithm, and provide shape-sensitivities obtained by automatic differentiation of the PINN. We thereby simultaneously converge to a local optimum while improving the accuracy of the solution. Although, as it optimizes, it is trained only on 200,000 sets of design parameters, the PINN can still reliably predict the flow over a wide range of parameters. We demonstrate this method on two examples: one that optimizes a single parameter, and another that optimizes eleven parameters. The method is successful and, by comparison with conventional CFD, we find that the velocity and pressure fields have small pointwise errors and that the method successfully finds the optimal parameters. Nevertheless, we find that different PINNs converge to slightly different parameters, reflecting the fact that there are many closely-spaced local minima when using stochastic gradient descent.

This approach combines two strengths of deep neural networks: the ability to approximate a high dimensional function well, and the automatic differentiability of that function with respect to its inputs. The first strength means that the surrogate model can approximate the flow at any point in parameter space. In principle, it could be trained

Table A.8
Initial guesses of PARSEC parameters.

Initial Guess	Γ_1	Γ_2	Γ_3	Γ_4	Γ_5	Γ_6	Γ_7	Γ_8	Γ_9	Γ_{10}	Γ_{11}
Case 1	0.012	0.252	0.056	−0.447	0.262	−0.066	0.417	−0.0038	0.002	−2.544	8.783
Case 2	0.013	0.323	0.064	−0.520	0.333	−0.056	0.530	0.0023	0.003	−2.782	9.562
Case 3	0.014	0.243	0.048	−0.467	0.314	−0.049	0.557	0.0035	0.001	−3.168	10.234
Case 4	0.015	0.303	0.052	−0.580	0.268	−0.070	0.600	0.0040	0.004	−3.335	11.100
Case 5	0.016	0.316	0.062	−0.573	0.273	−0.064	0.487	0.0017	0.005	−3.035	10.035
Case 6	0.017	0.267	0.066	−0.487	0.323	−0.052	0.592	−0.0025	0.001	−3.265	8.283
Case 7	0.018	0.323	0.071	−0.527	0.286	−0.058	0.535	−0.0012	0.002	−2.745	7.562
Case 8	0.017	0.296	0.054	−0.506	0.257	−0.067	0.512	0.0036	0.003	−3.185	9.423
Case 9	0.015	0.258	0.053	−0.492	0.326	−0.055	0.467	−0.0027	0.005	−2.845	10.863
Case 10	0.014	0.263	0.068	−0.578	0.353	−0.062	0.587	0.0028	0.001	−3.056	9.883

at all points but, because we are interested in simultaneous training and optimization, we train it predominantly around the optimal point. The second strength avoids the need to write an adjoint code. This saves development effort because adjoint codes, whether continuous or discrete, are challenging to construct. The concept in this paper can therefore be applied relatively easily to other optimization problems and is particularly appealing in applications where labeled data is scarce or expensive. Future work is needed to explore how this method would cope with several dozen parameters and to investigate other ways to optimize the loss function of the PINN. As our knowledge of how to efficiently train PINNs increases, and the hardware to train them becomes faster, this method of simultaneous training and optimization could become easier and faster than using adjoint codes.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

The doi of the code is listed in the references

Acknowledgments

We acknowledge funding by the Royal Society, United Kingdom (grant number NIF\R1\192317) for supporting this study. Ushnish Sengupta received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Sklodowska-Curie grant agreement No 766264.

Appendix

See Table A.8.

References

[1] A. Jameson, S. Kim, Reduction of the adjoint gradient formula for aerodynamic shape optimization problems, AIAA J. 41 (11) (2003) 2114–2129, http://dx.doi.org/10.2514/2.6830.

[2] S. Molesky, Z. Lin, A.Y. Piggott, W. Jin, J. Vucković, A.W. Rodriguez, Inverse design in nanophotonics, Nat. Photonics 12 (11) (2018) 659–670, http://dx.doi.org/10.1038/s41566-018-0246-9.

[3] S. Schmidt, E. Wadbro, M. Berggren, Large-scale three-dimensional acoustic horn optimization, SIAM J. Sci. Comput. 38 (6) (2016) B917–B940, http://dx.doi.org/10.1137/15M1021131.

[4] H. Harbrecht, F. Loos, Optimization of current carrying multicables, Comput. Optim. Appl. 63 (1) (2016) 237–271, http://dx.doi.org/10.1007/s10589-015-9764-2.

[5] J. Han, A. Jentzen, E. Weinan, Solving high-dimensional partial differential equations using deep learning, Proc. Natl. Acad. Sci. USA 115 (34) (2018) 8505–8510, http://dx.doi.org/10.1073/pnas.1718942115, arXiv:1707.02568.

[6] J. Li, Y. Guan, G. Wang, G. Wang, H. Zhang, J. Lin, A meshless method for topology optimization of structures under multiple load cases, Structures 25 (2020) 173–179, http://dx.doi.org/10.1016/j.istruc.2020.03.005, https://www.sciencedirect.com/science/article/pii/S2352012420300886.

[7] D. Daxini, J.M. Prajapati, Parametric shape optimization techniques based on Meshless methods : A review, Struct. Multidiscip. Optim. 56 (2017) 1197–1214, http://dx.doi.org/10.1007/s00158-017-1702-8.

[8] E. Madenci, A. Barut, M. Dorduncu, Peridynamic Differential Operator for Numerical Analysis, first ed., Springer Cham, 2019, http://dx.doi.org/10.1007/978-3-030-02647-9.

[9] O. Pironneau, On optimum design in fluid mechanics, J. Fluid Mech. 64 (1) (1974) 97–110.

[10] A. Jameson, Aerodynamic design via control theory, J. Sci. Comput. 3 (3) (1988) 233–260, http://dx.doi.org/10.1007/BF01061285.

[11] T.R. Bewley, Flow control: New challenges for a new Renaissance, Prog. Aerosp. Sci. 37 (1) (2001) 21–58, http://dx.doi.org/10.1016/S0376-0421(00)00016-6.

[12] A. Güne, G. Baydin, B.A. Pearlmutter, J.M. Siskind, Automatic differentiation in machine learning: a survey, J. Mach. Learn. Res. 18 (2018) 1–43.

[13] W.M. Hsu, J.F. Hughes, H. Kaufman, Direct manipulation of free-form deformations, Comput. Graph. (ACM) 26 (2) (1992) 177–184, http://dx.doi.org/10.1145/142920.134036.

[14] C.L. Koo, D.W. Zingg, Comparison of B-spline surface and free-form deformatio geometry control for aerodynamic optimization, AIAA J. 55 (1) (2017) 228–240, http://dx.doi.org/10.2514/1.J055102.

[15] H. Yu, M.P. Juniper, L. Magri, Combined state and parameter estimation in level-set methods, J. Comput. Phys. 399 (2019) 108950, http://dx.doi.org/10.1016/j.jcp.2019.108950, arXiv:1903.00321.

[16] A. Kontogiannis, S. Elgersma, A.J. Sederman, M.P. Juniper, Joint reconstruction and segmentation of noisy velocity images as an inverse Navier–Stokes problem, J. Fluid Mech. 944 (2022) A40, http://dx.doi.org/10.1017/jfm.2022.503.

[17] K. Stein, T.E. Tezduyar, R. Benney, Automatic mesh update with the solid-extension mesh moving technique, Comput. Methods Appl. Mech. Engrg. 193 (21–22) (2004) 2019–2032, http://dx.doi.org/10.1016/j.cma.2003.12.046.

[18] S.H. Kim, F. Boukouvala, Machine learning-based surrogate modeling for data-driven optimization: A comparison of subset selection for regression techniques, Optim. Lett. 14 (2020) 989–1010, http://dx.doi.org/10.1007/s11590-019-01428-7.

[19] L.W. Chen, B.A. Cakal, X. Hu, N. Thuerey, Numerical investigation of minimum drag profiles in laminar flow using deep learning surrogates, J. Fluid Mech. 919 (2021) 1–28, http://dx.doi.org/10.1017/jfm.2021.398, arXiv:2009.14339.

[20] M. Raissi, A. Yazdani, G.E. Karniadakis, Hidden fluid mechanics : A Navier–Stokes informed deep learning framework for assimilating flow visualization data, Science 367 (6481) (2020) 1026–1030, arXiv:arXiv:1808.04327v1.

[21] H. Eivazi, M. Tahani, P. Schlatter, R. Vinuesa, Physics-informed neural networks for solving Reynolds-averaged Navier–Stokes equations, 2021, arXiv (2107.10711). arXiv:2107.10711.

[22] F. Anselmi, L. Rosasco, T. Poggio, On invariance and selectivity in representation learning, Inf. Inference 5 (2) (2016) 134–158, http://dx.doi.org/10.1093/imaiai/iaw009, arXiv:1503.05938.

[23] Y. Lecun, Y. Bengio, G. Hinton, Deep learning, Nature 521 (7553) (2015) 436–444, http://dx.doi.org/10.1038/nature14539.

[24] K. Hornik, M. Stinchcombe, H. White, Multilayer feedforward networks are universal approximators, Neural Netw. 2 (5) (1989) 359–366, http://dx.doi.org/10.1016/0893-6080(89)90020-8.

[25] C.F. Higham, D.J. Higham, Deep learning: An introduction for applied mathematicians, SIAM Rev. 61 (4) (2019) 860–891, http://dx.doi.org/10.1137/18M1165748, arXiv:1801.05894.

[26] M. Bartholomew-Biggs, S. Brown, B. Christianson, L. Dixon, Automatic differentiation of algorithms, J. Comput. Appl. Math. 124 (1–2) (2000) 171–190, http://dx.doi.org/10.1016/S0377-0427(00)00422-2.

[27] S.A. Niaki, E. Haghighat, T. Campbell, A. Poursartip, Physics-informed neural network for modelling the thermochemical curing process of composite-tool systems, Comput. Methods Appl. Mech. Engrg. 384 (2021) 113959, arXiv:arXiv:2011.13511v2.

[28] S. Markidis, The Old and the New: Can Physics-Informed Deep-Learning Replace Traditional Linear Solvers? Front. Big Data 4 (November) (2021) 1–15, http://dx.doi.org/10.3389/fdata.2021.669097, arXiv:2103.09655.

[29] Y. Yang, P. Perdikaris, Adversarial uncertainty quantification in physics-informed neural networks, J. Comput. Phys. 394 (2019) 136–152, http://dx.doi.org/10.1016/j.jcp.2019.05.027, arXiv:1811.04026.

[30] M. Raissi, P. Perdikaris, G.E. Karniadakis, Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations, J. Comput. Phys. 378 (2019) 686–707, http://dx.doi.org/10.1016/j.jcp.2018.10.045.

[31] L. Lu, X. Meng, Z. Mao, G.E. Karniadakis, DeepXDE: A deep learning library for solving differential equations, SIAM Rev. 3 (2021) 208–228, arXiv:1907.04502.

[32] M. Raissi, Z. Wang, M.S. Triantafyllou, G.E. Karniadakis, Deep learning of vortex-induced vibrations, J. Fluid Mech. 861 (2019) 119–137, http://dx.doi.org/10.1017/jfm.2018.872, arXiv:1808.08952.

[33] M.F. Fathi, I. Perez-Raya, A. Baghaie, P. Berg, G. Janiga, A. Arzani, R.M. D'Souza, Super-resolution and denoising of 4D-Flow MRI using physics-Informed deep neural nets, Comput. Methods Programs Biomed. (2020) http://dx.doi.org/10.1016/j.cmpb.2020.105729.

[34] A. Arzani, J.X. Wang, R.M. D'Souza, Uncovering near-wall blood flow from sparse data with physics-informed neural networks, Phys. Fluids 33 (7) (2021) 1–19, http://dx.doi.org/10.1063/5.0055600, arXiv:2104.08249.

[35] X. Jin, S. Cai, H. Li, G.E. Karniadakis, NSFnets (Navier–Stokes flow nets): Physics-informed neural networks for the incompressible Navier–Stokes equations, J. Comput. Phys. 426 (2021) 109951, http://dx.doi.org/10.1016/j.jcp.2020.109951, https://www.sciencedirect.com/science/article/pii/S0021999120307257.

[36] C. Zhu, R.H. Byrd, P. Lu, J. Nocedal, L-BFGS-B: Fortran subroutines for large-scale bound-constrained optimization, ACM Trans. Math. Software 23 (4) (1997) 550–560, http://dx.doi.org/10.1145/279232.279236.

[37] Y. Sun, U. Sengupta, M.P. Juniper, Code supporting current paper, 2023, http://dx.doi.org/10.17863/CAM.95425.

[38] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, R. Jozefowicz, Y. Jia, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, M. Schuster, R. Monga, S. Moore, D. Murray, C. Olah, 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, X. Zheng, TensorFlow: large-scale machine learning on heterogeneous systems, 2015, https://www.tensorflow.org/.

[39] M.A. Nabian, R.J. Gladstone, H. Meidani, Efficient training of physics-informed neural networks via importance sampling, Comput.-Aided Civ. Infrastruct. Eng. 36 (8) (2021) 962–977, http://dx.doi.org/10.1111/mice.12685, arXiv:2104.12325.

[40] R.H. Byrd, P. Lu, J. Nocedal, C. Zhu, A limited memory algorithm for bound constrained optimization, SIAM J. Sci. Comput. 16 (5) (1995) 1190–1208, http://dx.doi.org/10.1137/0916069.

[41] A. Logg, K.-A. Mardal, G. Wells, Automated Solution of Differential Equations By the Finite Element Method, Springer, Berlin, Heidelberg, 2012, http://dx.doi.org/10.1007/978-3-642-23099-8.

[42] A. Kashefi, D. Rempe, L.J. Guibas, A point-cloud deep learning framework for prediction of fluid flow fields on irregular geometries, Phys. Fluids 33 (2) (2021) http://dx.doi.org/10.1063/5.0033376, arXiv:2010.09469.

[43] H. Sobieczky, Geometry Generator for CFD and Applied Aerodynamics, Springer Vienna, Vienna, 1997, pp. 137–157, http://dx.doi.org/10.1007/978-3-7091-2658-5_9.

[44] H. Sobieczky, Parametric Airfoils and Wings, 1999, pp. 71–87, http://dx.doi.org/10.1007/978-3-322-89952-1_4.

[45] P. Della Vecchia, E. Daniele, E. D'Amato, An airfoil shape optimization technique coupling PARSEC parameterization and evolutionary algorithm, Aerosp. Sci. Technol. 32 (1) (2014) 103–110, http://dx.doi.org/10.1016/j.ast.2013.11.006.