Computer-Aided Design 160 (2023) 103520

Contents lists available at ScienceDirect

Computer-Aided Design

journal homepage: www.elsevier.com/locate/cad

A Framework for Physics-Informed Deep Learning Over Freeform Domains

Francesco Mezzadri [a], Joshua Gasick [b], Xiaoping Qian [b,*]

[a] Department of Engineering "Enzo Ferrari", University of Modena and Reggio Emilia, via P. Vivarelli 10/1, Building 26, I-41125, Modena, Italy
[b] Department of Mechanical Engineering, University of Wisconsin-Madison, 1513 University Avenue, Madison, WI, 53706 - 1572, USA

ARTICLE INFO

Article history:
Received 25 October 2022
Received in revised form 28 February 2023
Accepted 29 March 2023

Keywords:
Physics-informed deep learning
Neural network
NURBS
Computer-aided design
Partial differential equations

ABSTRACT

Deep learning is a popular approach for approximating the solutions to partial differential equations (PDEs) over different material parameters and boundary conditions. However, no work has yet been reported on learning PDE solutions over changing shapes of the underlying domain.

We present a framework to train neural networks (NN) and physics-informed neural networks (PINNs) to learn the solutions to PDEs defined over varying freeform domains. This is made possible through our adoption of a parametric non-uniform rational B-Spline (NURBS) representation of the underlying physical shape. Distinct physical domains are mapped to a common parametric space via NURBS parameterization. In our approach, we formulate NNs and PINNs that learn the solutions to PDEs as a function of the shape of the domain itself through shape parameters.

Under this formulation, the loss function is based on an unchanging parametric domain that maps to a variable physical domain. Residual computation in PINNs is made possible through the Jacobian of the mapping.

Numerical results show that our networks can be trained to predict the solutions to a PDE defined over an entire set of shapes. We focus on the linear elasticity PDE and show how we can build a surrogate model that is able to predict displacements and stresses over a variety of freeform domains. To assess the efficacy of all networks in this work, data efficiency, network accuracy, and the capacity of networks to extrapolate are considered and compared between NNs and PINNs. The comparison includes cases where little training data is available. Transfer learning and applications to shape optimization are analyzed as well. A selection of the used codes and datasets is provided at https://github.com/fmezzadri/shape_parameterized.git.

1. Introduction

Computational approaches based on machine learning and neural networks have been widely studied in recent years. Indeed, the growing availability of computational power and a proliferation of data have made it possible to apply neural networks to increasingly complex problems. Further, the generality and relative simplicity of neural networks have cemented them as powerful tools in diverse fields. To meet the demands of many of these environments, neural networks have been adapted, reformulated, and re-structured. For instance, many recent works have focused on optimizing hyperparameters to improve learning [1–3]. Other researchers have proposed alternative network architectures and loss formulations to address difficulties in training.

In this context, physics-informed neural networks (PINNs) have been recently introduced [4] to solve forward and inverse problems involving partial differential equations (PDEs). These networks are characterized by the incorporation of an additional loss term that represents the residual of the neural network-predicted solution with respect to the governing equations themselves. The total loss of a typical physics-informed neural network is then composed of two parts: a data-based term and a residual- or physics-based term. The data term captures the extent to which the PINN predicted solution interpolates the data whereas the physics term identifies the adherence of the predicted solution to some underlying physical equations. The explicit inclusion of the residual of the PDE in the loss acts as a regularization mechanism and can lead to more physically consistent solution fields. Although the training can be an issue in some cases [5], PINNs have proven to be an effective tool especially for inverse problems and surrogate modeling. Variations of the formulation of PINNs have been proposed, such as PINNs designed to satisfy

* Corresponding author.
E-mail addresses: francesco.mezzadri@unimore.it (F. Mezzadri), qian@engr.wisc.edu (X. Qian).

https://doi.org/10.1016/j.cad.2023.103520
0010-4485/

conservation laws [6], PINNs that decompose a long-time problem into many independent short-time problems [7], and PINNs for fractional derivatives [8].

Physics-informed neural networks have also been applied to a variety of physical problems. Examples include the Navier–Stokes equation [9], the Reynolds equation [10], high-speed flows [11], the Chen–Lee–Liu equations [12], and the equations of linear elasticity. With respect to the final of these, PINNs were trained to serve as surrogate models for problems in solid mechanics in [13]. For example, Haghighat et al. in [13] trained PINNs to learn the solution to the equations of linear elasticity for a problem over a simple, square domain as a function of the physical Lamè parameter μ. It was also empirically shown that the trained PINNs could accurately predict the solution of the PDE for wide ranges of μ, including those which were not considered during training.

While [13] and other works [14,15] have shown the usefulness of training PINNs and NNs to model the solutions to PDEs as a function of varying material properties (PDE coefficients), little research has been done to extend this approach to model solutions as a function of varying domain shapes. Further, the necessity to formulate neural networks that can work with freeform domains is called for in the literature. For instance, the support of accurate descriptions of complex curvilinear boundaries is mentioned as a possible future development of the DeepXDE tool [16]. To address these needs, in this paper, we propose a formulation for NNs and PINNs that can predict PDE solutions as a function of domain shape. In our formulation, the loss functions of neural networks are based on an unchanging parametric domain that maps to a variable physical domain. Residual computation in PINNs is made possible by the use the Jacobian of the mapping. The Jacobian is needed only for physics-informed loss computation during the training, and is not needed for PINN-based solution query.

Fig. 1 summarizes our approach. In the figure, a generic physical domain Ω with coordinates (x, y) is mapped on a parametric domain $\hat{\Omega}$. The parametric coordinates are denoted by (ξ, η). In our approach, the mapping between Ω and $\hat{\Omega}$ is constructed through non-uniform rational B-splines (NURBS) [17]. Our choice of NURBS parameterization of shapes is due to several reasons. First, NURBS are a common shape representation widely used in CAD systems and represent shape changes compactly. Second, NURBS parameterization of the physical domain is the basis of a popular analysis approach, isogeometric analysis [18]. Finally, due to the use of a common parametric domain, NURBS allow for convenient querying of point-wise data-based errors or physics-based residuals over varying physical domains. The geometric mapping between the domains is based on the coordinates of a set of control points, which we denote as P_{ij}. As represented in Fig. 1, a set of collocation points is then selected in $\hat{\Omega}$. Working in the parametric domain, the parametric coordinates of the collocation points will not change even when the physical domain is modified. Under the assumption that we have access to the solution $u^*(\xi, \eta)$ of a PDE at these collocation points for one or more training shapes, we can train a neural network that is a surrogate model of the solution of the PDE on different domains. The inputs of such neural network are the parametric coordinates (ξ, η) along with the shape parameters that characterize the shape of the domains. The training data is constituted by the solution $u^*(\xi, \eta)$ of the PDE in the training domains. Since we employ a NURBS parameterization, we can conveniently generate the data by isogeometric analysis (IGA) [18]. With this information, a NN loss function (completely based on the training data) or a PINN loss function (where physics-based terms are also included) can be formulated. The trained neural network can predict a solution $u(\xi, \eta)$ of the PDE defined on any domain when the network is inputted with a set of coordinates (ξ, η) and with the shape parameters, such as NURBS control points or geometric parameters such as axes of ellipses.

To demonstrate the efficacy of this approach, linear elasticity problems defined over domains of varying complexities are addressed. Background information on elasticity, neural network and NURBS are introduced in Section 2. Neural networks for parameterized shapes are then introduced in Section 3. Section 4 contains several numerical experiments. The comparative efficiencies and accuracies of NNs and PINNs are also assessed in Section 4, alongside an analysis of the role of the continuity and the degree of the parameterization on learning. The accuracy of the two types of neural network when little data is available is also compared and analyzed. Transfer learning and applications to shape optimization are finally considered as well. In Section 5, concluding remarks are made and an outline for future work is described.

2. Introduction to elasticity PDEs, neural networks, and NURBS

In this section, a model for linear elasticity, a mathematical summary of the structure of neural networks and physics-informed neural networks, and an overview of NURBS representation are discussed.

2.1. The linear elasticity model

Throughout this paper, the 2D equations of linear elasticity are studied under the plane stress assumption. The constitutive equations are:

$$\sigma_{xx} = (\epsilon_{xx} + \nu\epsilon_{yy})\frac{E}{1-\nu^2}, \tag{1}$$

$$\sigma_{yy} = (\nu\epsilon_{xx} + \epsilon_{yy})\frac{E}{1-\nu^2}, \tag{2}$$

$$\sigma_{xy} = \frac{E}{(1+\nu)}\epsilon_{xy}, \tag{3}$$

where E denotes the Young's stiffness modulus, ν denotes the Poisson's ratio of the material, σ_{ij}, $i, j \in \{x, y\}$, is the stress tensor and ϵ_{ij} is the strain tensor. By the kinematic relations, this last term can be written as

$$\epsilon_{ij} = \frac{1}{2}(u_{i,j} + u_{j,i}), \tag{4}$$

for $i, j \in \{x, y\}$, where u_i denotes the displacements and the comma in the subscripts indicates a partial derivative.

Finally, in general, the linear elasticity model includes a momentum balance

$$\sigma_{ij,j} + f_i = 0, \tag{5}$$

where f_i, $i \in \{x, y\}$, denotes the body forces.

2.2. Neural networks

A neural network is a mathematical tool that can be trained to represent the relationship between data or datasets. For example, a neural network can be trained to associate images with classes, or to predict the solution to a PDE based on a set of spatial coordinates.

A fundamental feature of neural networks is that they are composed of different layers of neurons. Fig. 2 shows a simple example of a neural network where there are two inputs x_1, x_2, two hidden (or intermediate) layers with two neurons each and one output u. The neurons of the hidden layer are named $a_j^{(k)}$, which is the jth neuron of the kth hidden layer. Finally, n_k denotes the cardinality of the weights $w_{ij}^{(k)}$ of the kth layer, $b_j^{(k)}$ is a bias, f_k

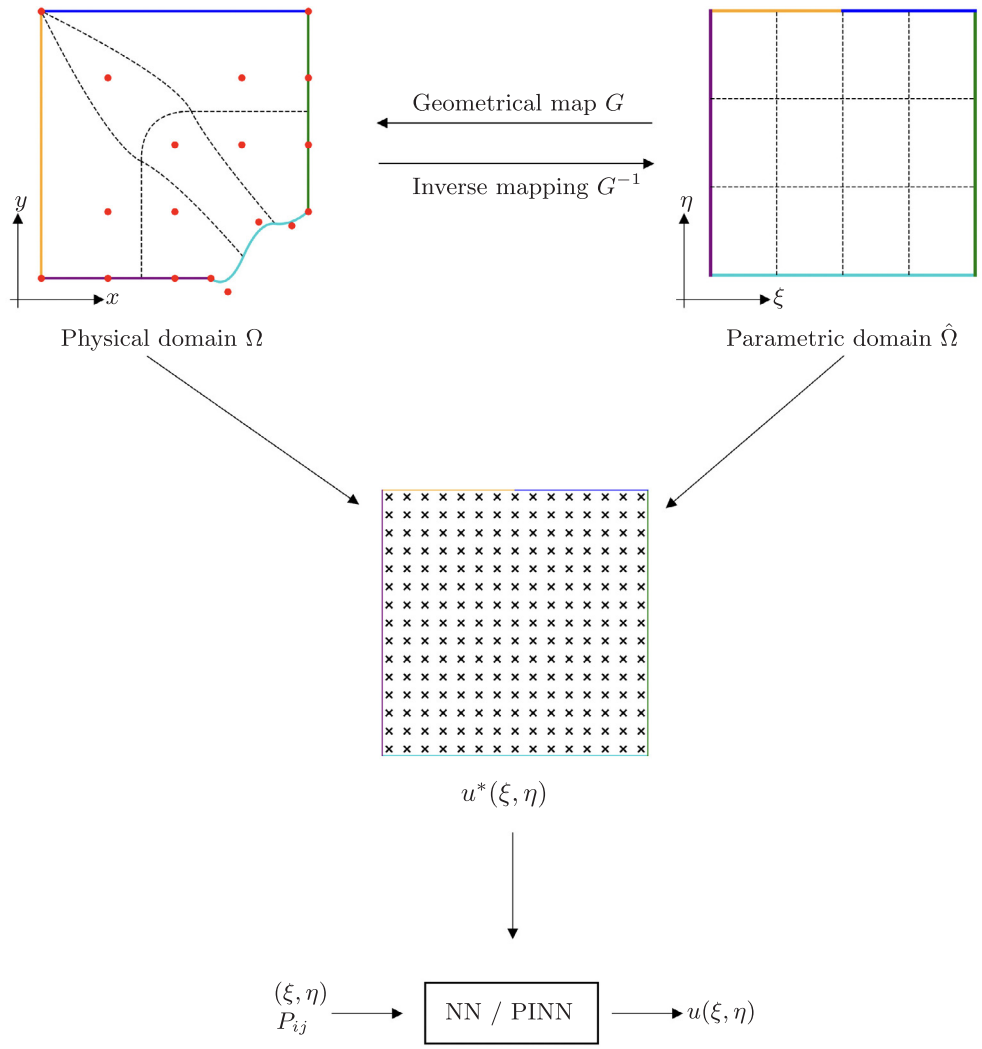


Fig. 1. NURBS-based domain parameterization for physics-informed deep learning over freeform shapes. Different physical domains are parameterized to a common domain $\hat{\Omega}$ by means of the control points P_{ij} of a NURBS representation. The solution u^* of a PDE defined on the freeform shape is then evaluated in a set of collocation points in the parametric domain. Finally, the data is passed to a neural network for training. The trained network can make predictions u of the solution u^* even when the shape of the domain is changed.

denotes an activation function, u^* denotes the expected solution, and $|\square|$ denotes the mean square error, which, for a general output vector u of n components, is

$$|u - u^*| = \frac{1}{n}\sum_{i=1}^{n}(u_i - u_i^*)^2. \tag{6}$$

The weights and the biases are variables which can be adjusted or 'trained' such that the neural network predicts a certain output when given a certain input. During training, the network predicts on some inputs for which the correct output is known. The output predicted by the network is then compared with the expected output by means of the loss function $\mathcal{L}$. The training of the weights is achieved by solving an unconstrained minimization problem to reduce the loss, or, equivalently, to improve the accuracy of the network:

$$\min_{\boldsymbol{b},\boldsymbol{w}} \mathcal{L}. \tag{7}$$

During each iteration of the optimization algorithm, the weights are then tuned by means of a "backpropagation" procedure. If there exists a continuous, bounded relationship between the input and output data, a neural network that is an arbitrarily good model of this input–output relationship can be formed and trained [19]. For more information on the neural networks the reader is referred to [20].

2.3. Physics-informed neural network for linear elasticity

Consider a PDE of the form

$$\mathcal{D}u = b, \tag{8}$$

where $\mathcal{D}$ denotes a partial differential operator that acts on a distribution u in some domain Ω with boundary $\partial\Omega$. The solution to the PDE can be modeled by neural networks for forward or inverse problems.

Physics-informed deep learning [4] supplements generic neural network training by including a PDE-specific residual term in the loss function. A PINN loss function can thus be written as follows, where a superscript '$*$' denotes given information/data:

$$\mathcal{L}_{PINN} = |u - u^*|_{\Omega} + |\mathcal{D}u - b^*| + |u - u_0^*|_{t=0} + |u - u^*|_{\partial\Omega}. \tag{9}$$

The physics-informed component of the loss is the term $|\mathcal{D}u - b^*|$ in Eq. (9). The terms $|u - u_0^*|_{t=0}$ and $|u - u^*|_{\partial\Omega}$ represent the enforcement of initial and boundary conditions, respectively. Finally, the term $|u - u^*|_{\Omega}$ represents a data-driven contribution

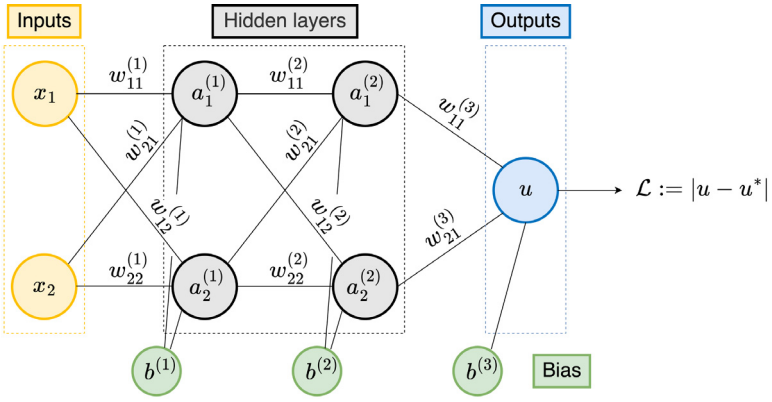


Fig. 2. Outline of the mathematical setting of a generic neural network.

that is based on the knowledge of u in some points of the interior of the domain. Naturally, not all these data-based terms are always present. For instance, if a PINN is used to forward solve a differential problem, the only available data will be at the boundary and at the initial time. Conversely, in inverse problems and surrogate modeling, the solution to the PDE may be known at some points on the interior of the domain but potentially not at the boundaries.

2.4. Parameterization by NURBS

Let $\{B_{i,p}\}$, $i = 0, \dots, n$, be the B-spline function of degree p defined on the knot vector

$$k_\xi = \{\xi_0, \xi_1, \dots, \xi_{n+p+1}\}.$$

Such B-spline functions can be defined recursively. In particular, for the generic ith B-spline function, we have

$$\begin{aligned} B_{i,p}(\xi) &= \frac{(\xi - \xi_i)B_{i,p-1}(\xi)}{\xi_{i+p} - \xi_i} + \frac{(\xi_{i+p+1} - \xi)B_{i+1,p-1}(\xi)}{\xi_{i+p+1} - \xi_{i+1}}, \\ B_{0,p}(\xi) &= \begin{cases} 1 & \xi_i \le \xi \le \xi_{i+1}, \\ 0 & \text{otherwise.} \end{cases} \end{aligned} \tag{10}$$

The derivative of a B-Spline basis function, $B'_{i,p}(\xi)$, is:

$$B'_{i,p}(\xi) = \frac{p}{\xi_{i+p} - \xi_i} B_{i,p-1}(\xi) - \frac{p}{\xi_{i+p+1} - \xi_{i+1}} B_{i+1,p-1}(\xi). \tag{11}$$

A NURBS curve of degree p can be defined as

$$\boldsymbol{x}(\xi) = \frac{\sum_{i=0}^{n} B_{i,p}(\xi) w_i P_i}{\sum_{j=0}^{n} B_{j,p}(\xi) w_j}, \tag{12}$$

where $\{P_i\}$, $i = 0, \dots, n$, represents the coordinates of a set of control points, w_i, $i = 0, \dots, n$, is the corresponding weight, and $\xi \in [0, 1]$ is the parametric variable.

Eq. (12) can written more concisely using NURBS basis functions:

$$\boldsymbol{x}(\xi) = \sum_{i=0}^{n} R_{i,p}(\xi) P_i. \tag{13}$$

In Eq. (13), $R_{i,p}(\xi)$ is defined according to:

$$R_{i,p}(\xi) = \frac{B_{i,p}(\xi) w_i}{\sum_{j=0}^{n} B_{j,p}(\xi) w_j}. \tag{14}$$

The derivative of a NURBS curve is simply:

$$\boldsymbol{x}_\xi(\xi) = \sum_{i=0}^{n} R'_{i,p}(\xi) P_i. \tag{15}$$

In Eq. (15), the NURBS basis function derivative is:

$$R'_{i,p}(\xi) = \frac{B'_{i,p}(\xi) w_i}{\sum_{j=0}^{n} B_{j,p}(\xi) w_j} - \frac{B_{i,p}(\xi) w_i \left(\sum_{j=0}^{n} B'_{j,p}(\xi) w_j\right)}{\left(\sum_{j=0}^{n} B_{j,p}(\xi) w_j\right)^2}. \tag{16}$$

Similarly, it is possible to define a NURBS surface of degree p in the first parametric direction ξ and of degree q in the second parametric direction η as

$$\boldsymbol{x}(\xi, \eta) = \sum_{i=0}^{n} \sum_{j=0}^{m} R_{i,p}(\xi) R_{j,q}(\eta) P_{ij}, \tag{17}$$

where the control points $\{P_{ij}\}$ form a $\{(n+1) \times (m+1)\}$ control net and are associated to the weights $\{w_{ij}\}$. Finally, $\{R_{i,p}(\xi)\}$ and $\{R_{j,q}(\eta)\}$ are the NURBS basis functions based on B-Spline basis functions, which themselves are defined on the knot vectors $k_\xi = \{\xi_0, \xi_1, \dots, \xi_{n+p+1}\}$ and $k_\eta = \{\eta_0, \eta_1, \dots, \eta_{m+q+1}\}$.

The parametric derivatives of a NURBS surface are:

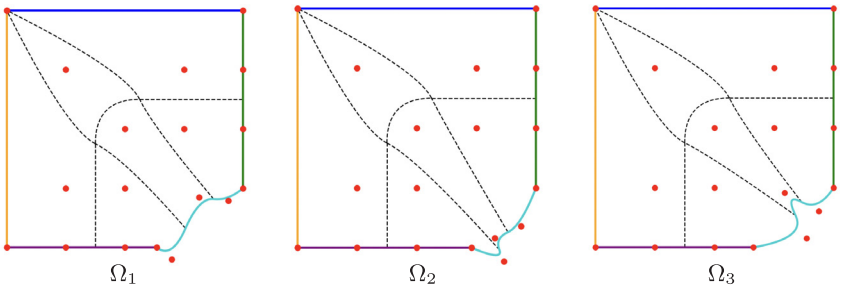
$$\boldsymbol{x}_\xi(\xi) = \sum_{i=0}^{n} \sum_{j=0}^{m} R'_{i,p}(\xi) R_{j,q}(\eta) P_{ij}, \tag{18a}$$

$$\boldsymbol{x}_\eta(\eta) = \sum_{i=0}^{n} \sum_{j=0}^{m} R_{i,p}(\xi) R'_{j,q}(\eta) P_{ij}. \tag{18b}$$

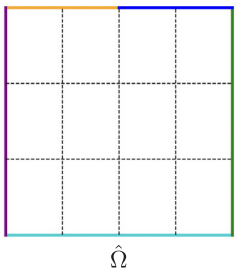
For more information on NURBS, the reader is referred to [17].

The control point and weight arrays define the shape of the NURBS representation. One of the most attractive properties of NURBS parameterization is that NURBS can represent complex shapes compactly with just a few control points.

Consider, for instance, a problem where the objective is to determine the optimal shape to give to a hole at the center of a plate. The profile (and, indeed, the entire geometry) can be represented using a bi-variate NURBS. For examples of different possible hole profiles that can be represented with NURBS curves, see Fig. 3(a), where only one-quarter of the plate is considered for reasons of symmetry. Through a NURBS representation, a wide range of possible plate-hole designs can be represented and each



(a) Example of three shapes Ω_1, Ω_2, and Ω_3 in the physical domain.



(b) Identical parametric domains $\hat{\Omega}$ of Ω_1, Ω_2, and Ω_3.

Fig. 3. Different shapes are mapped to the same parametric domain.

of these is mapped to the same parametric domain. This idea is shown in Fig. 3. The only alteration required to represent a geometry shown in Fig. 3(a) to obtain the geometry shown in Fig. 3(b) is the adjustment of a few of the control point coordinates in the array P_{ij}.

2.5. Data generation via isogeometric analysis

Isogeometric analysis (IGA) is a computational analysis tool used to approximate the solution to PDEs [18]. In contrast to the finite element method, IGA utilizes CAGD principles to represent the geometry and solution. Typically, linear combinations of NURBS basis functions are used in both capacities. Widely discussed and notable advantages of IGA in a general setting include its ability to exactly represent geometries and to obtain higher order continuity in the representation of the solution.

For the purposes of this paper, IGA is perhaps the most natural data collection method. Indeed, training data for distinct shapes can easily be generated. In this regard, data generation follows the following steps:

1. given a set of control points, use NURBS parameterization build a map between physical shapes and a parametric domain. Choose a set of collocation points in the parametric domain;
2. generate a shape by adjusting the control point coordinates;
3. apply IGA to compute the solution at the control point coordinates
4. repeat points 2–3 for new shapes. The data for each shape is thus generated simply by modifying the coordinates of the control points and re-applying an otherwise identical implementation of IGA

Sampling each solution at the same parametric coordinates also introduces a certain level of standardization to the data collection process despite the freeform nature of the domains of interest. These features simplify the training and testing of the NNs and PINNs discussed in this paper.

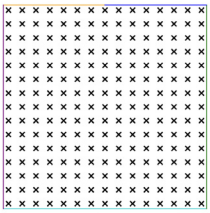
3. Shape-parameterized NNs and PINNs

In this section, we present our shape-parameterized NNs and PINNs. The proposed framework is based on the following key elements:

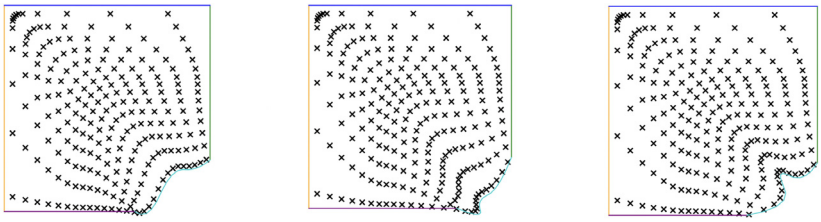
- The inputs of the neural network are a set of parametric coordinates (ξ, η) and a set of shape parameters $\boldsymbol{p}$ that characterize the shape of the domain. In general, the shape parameters are the coordinates of the control points P_{ij} of a NURBS parameterization. Other shape parameters, such as the length and width of a geometric feature, can also be conveniently used as inputs when they can characterize the shapes of the dataset of interest. See, for example, the rectangular and the circular beam problems in Section 4.1.
- The output of the neural network is a prediction of the solution of the PDE. In the case of the linear elasticity PDE, the outputs are the displacements and the stresses evaluated in the parametric domain. We denote such quantities by $\hat{u}_x$, $\hat{u}_y$ and by $\hat{\sigma}_{xx}$, $\hat{\sigma}_{yy}$, $\hat{\sigma}_{xy}$ for displacements and stresses, respectively.
- The data used for training are displacements and stresses as evaluated at a set of collocation points in the parametric domain. We denote the training displacements by $\hat{u}_x^*$, $\hat{u}_y^*$ and the training stresses by $\hat{\sigma}_{xx}^*$, $\hat{\sigma}_{yy}^*$, and $\hat{\sigma}_{xy}^*$.
- The loss functions are denoted by $\mathcal{L}_{NN}$ and $\mathcal{L}_{PINN}$ depending on whether we consider a purely data-based loss or a physics-informed loss, respectively.
- Additional parameters are required to compute the physics-informed loss $\mathcal{L}_{PINN}$. In particular, the material parameters E, ν (which represent the Young's modulus and the Poisson's ratio, respectively) are needed. In addition, the inverse Jacobian J^{-1} of the mapping between the physical and parametric domains is needed. This will be discussed later in this section.

3.1. Formulation for shape-parameterized neural networks

We want to formulate a neural network that can predict the solution of a PDE on a set of freeform domains. The first step is to



(a) Grid of equally spaced collocation points $\hat{c}_p$ in the parametric domain $\hat{\Omega}$



(b) Position of the collocation points in the physical domains Ω_1, Ω_2, Ω_3.

Fig. 4. A set of collocation points with equal spacing in the parametric domain, and corresponding position of the points in a few physical domains.

use a NURBS parameterization to map a set of physical domains of interest to the same parametric domain $\hat{\Omega}$. When the physical domain changes, the shape parameters $\boldsymbol{p}$ are modified, but $\hat{\Omega}$ remains the same.

Next, a set of collocation points is selected in the parametric domain $\hat{\Omega}$. This set is denoted by $\hat{c}_p$ and constitutes the points of $\hat{\Omega}$ where we want the neural network to evaluate the solution. Notice that, working on the parametric domain, $\hat{c}_p$ remain the same irrespective of the physical domain. Fig. 4 illustrates this concept considering, for instance, a set of evenly spaced collocation points in the parametric domain. Fig. 4 demonstrates that collocation points only need to be specified in one parametric domain. The mapping between each physical domain and the parametric domain captured by the NURBS representation eliminates the need to formulate a method for specifying collocation points in each individual and potentially unique physical domain. The shape parameters $\boldsymbol{p}$ are used to characterize different physical domains on the same parametric domain.

Therefore, the aim of the neural network becomes to predict the solution of the PDE on $\hat{c}_p$ for any choice of the shape parameters $\boldsymbol{p}$. To do this, the inputs of the neural network are formulated to be the parametric coordinates (ξ, η) of the collocation points $\hat{c}_p$ and the shape parameters $\boldsymbol{p}$. In the considered model problem of the linear elasticity PDE, the outputs are predictions of the displacements u_x, u_y and of the stresses $\sigma_{xx}, \sigma_{yy}, \sigma_{xy}$. Notice that we are interested in predicting displacements and stresses along the physical directions (x, y), but we evaluate them at the parametric coordinates $(\xi, \eta) \in \hat{\Omega}$. Therefore, for compactness of notation, we denote the predictions of the network evaluated in the parametric domain as

$$\begin{aligned} &\hat{u}_x := u_x(\xi, \eta, \boldsymbol{p}), \qquad \hat{u}_y := u_y(\xi, \eta, \boldsymbol{p}), \\ &\hat{\epsilon}_{xx} := \epsilon_{xx}(\xi, \eta, \boldsymbol{p}), \qquad \hat{\epsilon}_{yy} := \epsilon_{yy}(\xi, \eta, \boldsymbol{p}), \qquad \hat{\epsilon}_{xy} := \epsilon_{xy}(\xi, \eta, \boldsymbol{p}), \\ &\hat{\sigma}_{xx} := \sigma_{xx}(\xi, \eta, \boldsymbol{p}), \qquad \hat{\sigma}_{yy} := \sigma_{yy}(\xi, \eta, \boldsymbol{p}), \qquad \hat{\sigma}_{xy} := \sigma_{xy}(\xi, \eta, \boldsymbol{p}), \end{aligned} \tag{19}$$

where $(\xi, \eta) \in \hat{\Omega}$ with $\xi = \xi(x, y)$ and $\eta = \eta(x, y)$ in accordance with the mapping of the parameterization. These displacements, deformations and stresses are the quantities of interest of the considered elasticity problem. Several network settings can be chosen to compute them. For instance, we could formulate a neural network (or two independent networks) that outputs the displacements $\hat{u}_x$ and $\hat{u}_y$. Deformations and stresses could then be calculated by differentiating the displacements. In alternative, in the mixed formulation (see, e.g., [13]), either $(\hat{u}_x, \hat{u}_y, \hat{\epsilon}_{xx}, \hat{\epsilon}_{yy}, \hat{\epsilon}_{xy})$ or $(\hat{u}_x, \hat{u}_y, \hat{\sigma}_{xx}, \hat{\sigma}_{yy}, \hat{\sigma}_{xy})$ can be considered as outputs of one or multiple independent networks. We resort to this latter formulation, producing the five outputs $(\hat{u}_x, \hat{u}_y, \hat{\sigma}_{xx}, \hat{\sigma}_{yy}, \hat{\sigma}_{xy})$ via five separate networks. While, in principle, a sufficiently wide network would allow to compute all the quantities of interest by a single neural network, the use of separate networks simplifies the work of the optimizer [13]. This structure also avoids the conflation of weights and biases used to represent different solution fields. For example, by assigning one network to learn each solution distribution, the specific weights and biases that define the displacement fields do not interfere with the learning of the weights and biases used to define the stress fields.

To train the network on multiple domains, data must be available at least on a subset of the collocation points for several choices of $\boldsymbol{p}$. We denote the training data as $\hat{u}_x^*, \hat{u}_y^*, \hat{\sigma}_{xx}^*, \hat{\sigma}_{yy}^*, \hat{\sigma}_{xy}^*$. The training data can be either available from the boundary conditions of the PDE (in case of forward solution of the differential problem) or can be provided in the entire domain (in case of inverse problems and surrogate modeling).

Finally, the loss function must be defined. For a non physics-informed neural network, the loss is entirely composed of the difference between the network-predicted solution distribution at some point and the true solution value at the same point. As such, the loss $\mathcal{L}_{NN}$ can be formulated according to

$$\begin{aligned} \mathcal{L}_{NN} = &|\hat{u}_x - \hat{u}_x^*| + |\hat{u}_y - \hat{u}_y^*| + |\hat{\sigma}_{xx} - \hat{\sigma}_{xx}^*| \\ &+ |\hat{\sigma}_{yy} - \hat{\sigma}_{yy}^*| + |\hat{\sigma}_{xy} - \hat{\sigma}_{xy}^*|, \end{aligned} \tag{20}$$

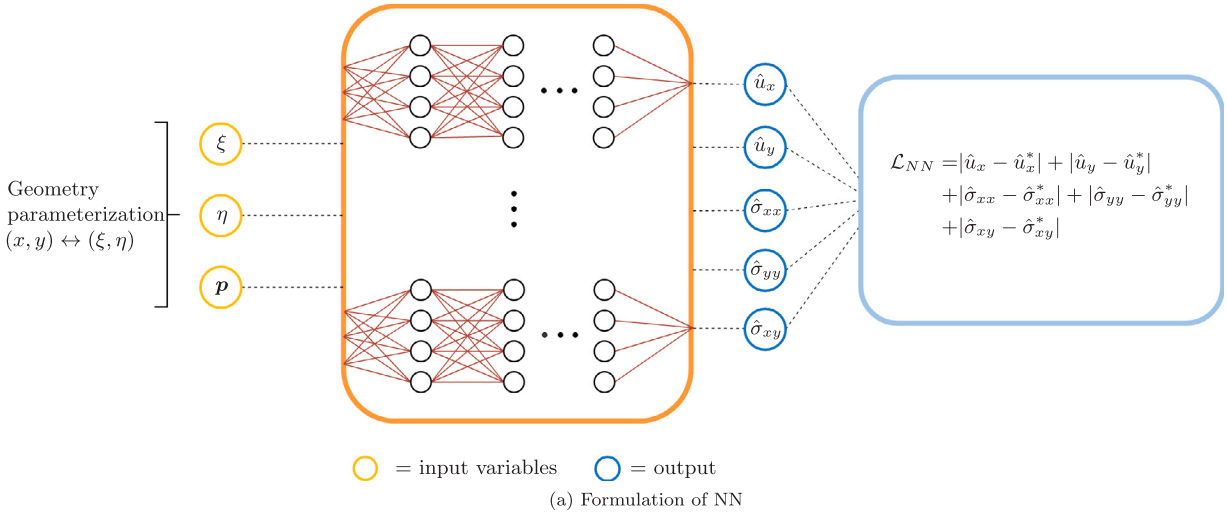
In contrast, the loss of a physics-informed neural network includes residual terms as well, as mentioned in Section 2. Such loss function contains three different types of terms: network outputs, training data, and terms that are computed by taking derivatives of the outputs with respect to physical coordinates. Under the assumption of plane stress, the general loss function of a PINN for linear elasticity becomes as in Eq. (21), where the different terms are highlighted in different colors, see Box I.

The presence of derivatives with respect to physical coordinates is the reason why computing $\mathcal{L}_{PINN}$ requires the inverse Jacobian of the mapping between the physical and parametric domains. This is discussed further in Section 3.2.

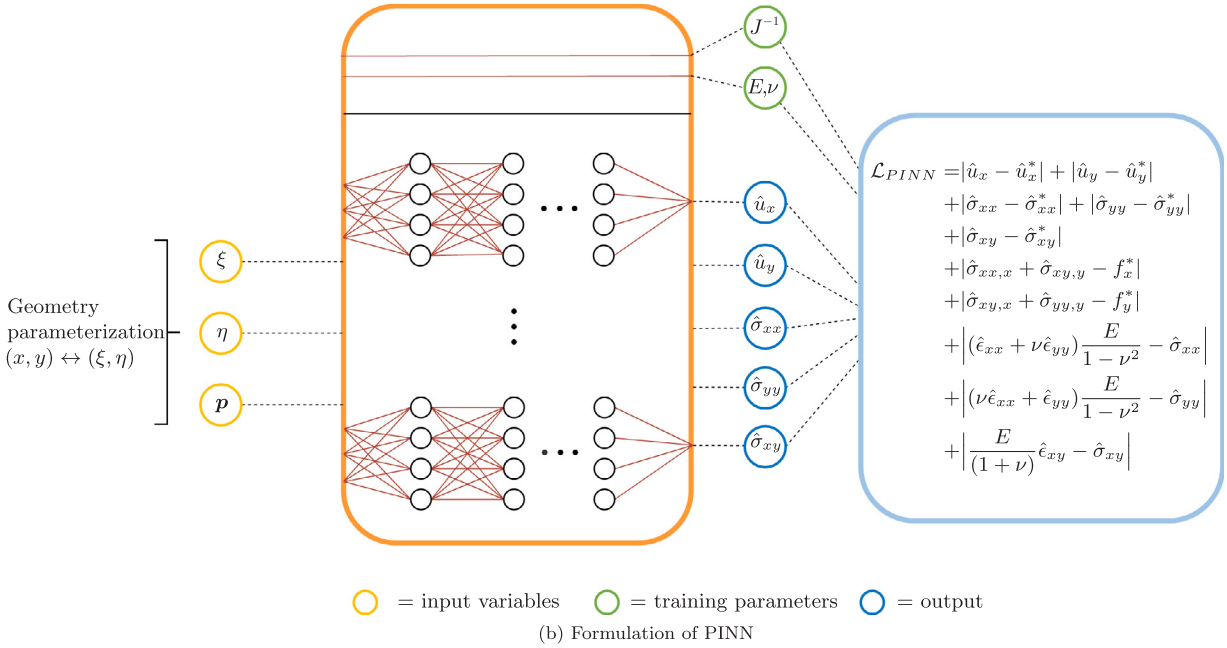
$$\mathcal{L}_{PINN} = |\hat{u}_x - \hat{u}_x^*| + |\hat{u}_y - \hat{u}_y^*| + |\hat{\sigma}_{xx} - \hat{\sigma}_{xx}^*| + |\hat{\sigma}_{yy} - \hat{\sigma}_{yy}^*| + |\hat{\sigma}_{xy} - \hat{\sigma}_{xy}^*|$$
$$+ |\hat{\sigma}_{xx,x} + \hat{\sigma}_{xy,y} - f_x^*| + |\hat{\sigma}_{xy,x} + \hat{\sigma}_{yy,y} - f_y^*| + \left|\left(\hat{\epsilon}_{xx} + \nu\,\hat{\epsilon}_{yy}\right)\frac{E}{1-\nu^2} - \hat{\sigma}_{xx}\right| \tag{21}$$
$$+ \left|\left(\nu\,\hat{\epsilon}_{xx} + \hat{\epsilon}_{yy}\right)\frac{E}{1-\nu^2} - \hat{\sigma}_{yy}\right| + \left|\frac{E}{(1+\nu)}\hat{\epsilon}_{xy} - \hat{\sigma}_{xy}\right|$$

= network outputs = training data = derivatives of output with respect to physical coordinates

Box I.



(a) Formulation of NN



(b) Formulation of PINN

Fig. 5. Shape parameterized NN and PINN.

The scheme of the shape-parameterized NNs and PINNs is represented in Fig. 5. The neural networks receive parametric coordinates values ξ and η, in addition to the shape parameters $\boldsymbol{p}$, which specify the exact form of a geometry. Then, they predict physical displacement $\hat{u}_x$, $\hat{u}_y$, and stress distribution values $\hat{\sigma}_{xx}$, $\hat{\sigma}_{xy}$ and $\hat{\sigma}_{yy}$. To evaluate the PINN loss function, additional

information is required, like the physical information E and ν, and the inverse Jacobian at the specific values of (ξ, η) of the collocation points.

3.2. Role of the inverse Jacobian in shape-parameterized PINN training

The previous section introduced a framework for a shape-parameterized PINN. However, exactly how the strain terms and the derivatives of the stresses in Eq. (21) can be obtained was not explicitly described. In particular, consider the terms highlighted in green in Eq. (21). The major difficulty in computing such terms is that, as Eq. (19) established, these distributions are modeled as functions of the parametric, not the physical, domain. As a result, only the derivatives with respect to the parametric coordinates can be obtained through automatic differentiation [21]. Thus, in this section the definition and necessity of the inverse Jacobian is discussed.

To obtain the physical derivatives of the displacements, the chain rule is invoked. For instance, $\hat{\epsilon}_{xx}$ can be computed as

$$\hat{\epsilon}_{xx} = \frac{\partial \hat{u}_x}{\partial x} = \frac{\partial u_x(\xi(x,y),\eta(x,y))}{\partial x} = \frac{\partial u_x(\xi,\eta)}{\partial \xi}\frac{\partial \xi}{\partial x} + \frac{\partial u_x(\xi,\eta)}{\partial \eta}\frac{\partial \eta}{\partial y}. \tag{22}$$

Automatic differentiation in neural networks is used to obtain the parametric derivatives $\partial u_x(\xi,\eta)/\partial\xi$ and $\partial u_x(\xi,\eta)/\partial\eta$ in Eq. (22). However, the derivatives of the parametric coordinates with respect to the physical coordinates $\frac{\partial \xi}{\partial x}$ and $\frac{\partial \eta}{\partial y}$ are geometry specific, and must be supplied to the PINN during training as there is no way to discover such relationships without external data. In practice, this means importing/computing the entries of

$$J^{-1} = \begin{pmatrix} \frac{\partial \xi}{\partial x} & \frac{\partial \xi}{\partial y} \\ \frac{\partial \eta}{\partial x} & \frac{\partial \eta}{\partial y} \end{pmatrix} \tag{23}$$

at the collocation points for all the training geometries. The entities shown in Eq. (23) can be obtained in this case by inverting the Jacobian of the two-dimensional, bivariate mapping:

$$J = \begin{pmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial x}{\partial \eta} \\ \frac{\partial y}{\partial \xi} & \frac{\partial y}{\partial \eta} \end{pmatrix}. \tag{24}$$

Each of the components of the Jacobian can be computed using one of the relationships shown in Section 2 during the introduction of the NURBS basis (i.e. Eq. (18)). Similar considerations apply to the derivatives of the stresses.

It should be noted that J^{-1} is needed only during the training of a shape-parameterized PINN. This is because physical derivatives of network-predicted distributions must be computed only to compute the loss. When we apply the trained network to make predictions, instead, the PINN outputs the physical distributions directly. It can also be observed from Eq. (20) that evaluating the loss of a non physics-informed neural network requires no Jacobian information.

Finally, we remark that, for simplicity and with no loss of generality, in the following we neglect the residual terms that include the derivatives of the stresses because body forces are not present in our test problems.

3.3. Applications: from surrogate modeling to transfer learning

The proposed procedure enables several attractive applications. For instance, we may know the solution of the PDE in the entire domain for a set of training geometries and we may want to find the relationship that expresses how the solution changes when the domain is changed. In this context, the shape-informed neural network is used to build a surrogate model of the PDE solution as the shape of the domain changes. This is the situation that we are going to consider in the following.

Nonetheless, our formulation is general and we can apply the shape-parameterized networks also under different circumstances. For instance, we may know the solution only at the boundary conditions and want to use the shape-parameterized PINN to forward solve the PDE on freeform domains. In this context, a particularly interesting application consists in *transfer learning*. Indeed, it is known that PINN is not competitive as a PDE forward solver [13] and is often characterized by slow convergence and other issues. Nonetheless, using the shape parameterization, we may exploit the information learnt by the network on a given shape to accelerate the convergence of the training on a different shape. In this setting, we first train a shape-parameterized PINN on a given domain. Then, assume that we want to predict the solution of the PDE on a different domain (sufficiently similar to the first one) where we do not have any data beyond the boundary conditions. In this case, we can aid the convergence of the training of the shape-parameterized PINN on the second shape by initializing the training using the optimized weights and biases obtained on the first shape. This is the situation that we consider when we study transfer learning in Section 4.8.

In these situations, the computational advantage of NNs and PINNs with respect to a direct application of IGA or FEA descends from the fact that a trained neural network learns the relationship that links inputs and outputs. Thus, in surrogate modeling, the training phase can be computationally onerous but the trained shape-parameterized neural network can then immediately predict the PDE solution on a variety of domains. In transfer learning, we aid the solution of the PDE on a domain by the knowledge of what the neural network has learnt on another domain.

4. Numerical experiments

In this section, we present several numerical experiments. We consider problems with 1 to 20 shape parameters and we compare PINN and NN in different situations.

4.1. Setting of the numerical experiments

As preliminary test cases, two distinct one-parameter problems are considered to demonstrate the efficacy of the shape-parameterized neural networks. First, the ability of NNs and PINNs to predict the displacement and stress profiles of a loaded rectangular beam is studied. A similar analysis is then performed with regard to a loaded quarter-circle beam. The geometry of the problems is parameterized by NURBS and the training data is generated by IGA. All the shapes of the datasets can be characterized by simple geometric parameters, which we use as inputs of the shape-parameterized neural networks. In particular, the beam length, L, is used as shape parameter of the rectangular beam problem, and the inner radius a is used for the circular beam problem.

For the rectangular beam problem, the fixed geometric and mechanical parameters are $P = 1$, $c = 0.75$, $E = 4/3$, $\nu = 1/3$. The thickness is assumed to be unitary. The NURBS representation of all cantilever beams is bi-quadratic, and is (C^1, C^0) continuous in the (ξ, η) directions. For the circular beam problem, the fixed geometric and mechanical parameters are $P = 10$, $b = 10$, $E = 1000$, and $\nu = 0.3$. The NURBS representation is bi-quadratic and is C^1 continuous in both parametric directions.

To generate data for the cantilever beam problem, the geometric shape variable L is varied uniformly between $L = 2$ and $L = 10$. For each value of L, a coarse representation of the geometry is

formed. Knot bisection is then performed six and four times in the ξ and η directions, respectively, yielding a refined representation of the domain with 2048 elements. This discretization forms the basis for IGA, which is used to obtain the NURBS coefficients which define the geometries that are the best representations of the true solution profiles of displacements and stresses. These geometries are sampled uniformly on the interior of the parametric domain at 100^2 locations. This is the labeled data used to train neural networks and PINNs for this problem. Similarly for the quarter-circle problem, a is varied uniformly between $a = 4$ and $a = 8$. For each value of a, a coarse representation of the domain is formed. Five knot bisections are performed in both parametric directions to obtain a 1024-element discrete representation of the domain. In the same manner as described above, 10,000 labeled data points are generated for each value of a through IGA.

Additionally, a more complex plate-hole problem is considered. In this case, the analytical solution is not available and both the training and the validation data is computed by IGA. The shape of the domain is described by multiple shape parameters, depending on the exact analysis that is being performed. In particular, we first consider cases where the shape is described by the coordinates of 5 control points (corresponding to 8 shape parameters for each shape). The dataset used 3 knot bisections in the ξ and η directions to obtain a discrete representation of the domain made of 384 elements. A total of 468 control points was used. Then, in later experiments we used 11 control points (corresponding to 20 shape parameters for each shape) to analyze the effect of the continuity of the NURBS representation and of the degree of the basis function on the accuracy of NN and PINN.

For all numerical examples, one neural network is used to represent each solution profile among $u_x, u_y, \sigma_{xx}, \sigma_{xy}, \sigma_{yy}$. Each network has 5 hidden layers with 20 neurons per layer. Hyperbolic tangent activation functions are used for all experiments.

In our implementation, each term of the total loss is adaptively weighted to avoid numerical difficulties produced by gradient pathology. In particular, it is known that the terms with larger derivatives tend to dominate the total loss gradient (e.g., see [22]). This can be problematic in PINNs [23]. In order to reduce this effect, each term of the loss can be weighted, i.e. the total loss is viewed as

$$\mathcal{L} = \sum_{i=1}^{n_t} \lambda_i \mathcal{L}_i,$$

where $\mathcal{L}_i$ denotes the ith individual term of the loss, λ_i is the weight of the ith term, and n_t is the number of terms of the loss. The weight λ_i is generally computed either by a gradient scaling algorithm (*GradNorm*, see [24]) or based on a neural tangent kernel (NTK, see [5]). We choose this latter approach, adjusting the NTK weights λ_i dynamically every 100 epochs. In particular, the NTK can be viewed as a kernel that describes the evolution during training of a neural network of infinite width. As regards the NTK of PINN in particular, [5] proved that it "converges to a deterministic kernel and remains constant during training via gradient descent with an infinitesimally small learning rate". The subsequent analysis of PINNs via their limiting NTK showed that different loss components have significant discrepancy in the convergence rate. Adaptively weighting the loss terms based on the NTK of PINN was then proposed as a strategy to calibrate the convergence rate of the total training error. The reader is referred to [5] for more information.

All the experiments are performed using Tensorflow [25]. The nonlinear minimization problems or neural network trainings are addressed using the Adam optimizer [26]. To simplify the setting up of the network and the formulation of the adaptive weights, the SciANN [27] library is used in all experiments. In particular, SciANN is a Keras/Tensorflow wrapper that automates the construction of PINNs while inheriting all Keras functionalities. SciANN functions can be used to abstractly define variables, inputs, outputs, etc. of the network, and different training settings (including the use of adaptive weights) can be chosen as options of the training command of the SciANN model.

4.2. Physics informed learning for one shape parameter problems

The boundary conditions and loadings for the rectangular and the circular beam problems are shown in Fig. 6 for some choices of L and a. The formal statement of the problems and their analytical solution can be found, for instance, in [28,29]. Examples of the NURBS parameterizations are reported in Appendix.

In this subsection, we train a PINN on these problems using L and a as shape parameters. A comparison with a shape parameterized NN is later performed as well.

4.2.1. Training and validation of PINN

In this subsection, we demonstrate that a shape-parameterized PINN can accurately learn the solution profiles to the one shape parameter problems introduced in Section 4.1. For the rectangular beam problem, the network was trained using the PINN loss function of Eq. (21) and labeled data and collocation points from four different cantilever beam geometries defined by $L = 2, L = 3, L = 5$, and $L = 10$. With respect to the quarter-circle problem, this consisted of training a network using the same loss function using labeled data and collocation points from six geometries, defined by $a = 4.202$, $a = 5.4545$, $a = 6.2626$, $a = 7.0303$, $a = 7.5556$, and $a = 8$. The network was trained for 500 epochs in both cases using a batch size of 100. The learning rate l_r is initialized at $l_r = 0.001$ and halved every 50 epochs that do not produce a decrease of the loss. Fig. 7 summarizes the convergence curve of the networks for both the cantilever beam (Fig. 7(a)) and for the quarter-circle problem (Fig. 7(b)).

A visual comparison of the analytical and PINN-predicted solution profiles for the quarter-circle problem are shown in Fig. 8. In Fig. 8, the network predicts on the geometry associated with $a = 5.697$, which is not one of the geometries in the training dataset. Analogous results are obtained with other choices of a and for the rectangular beam problem.

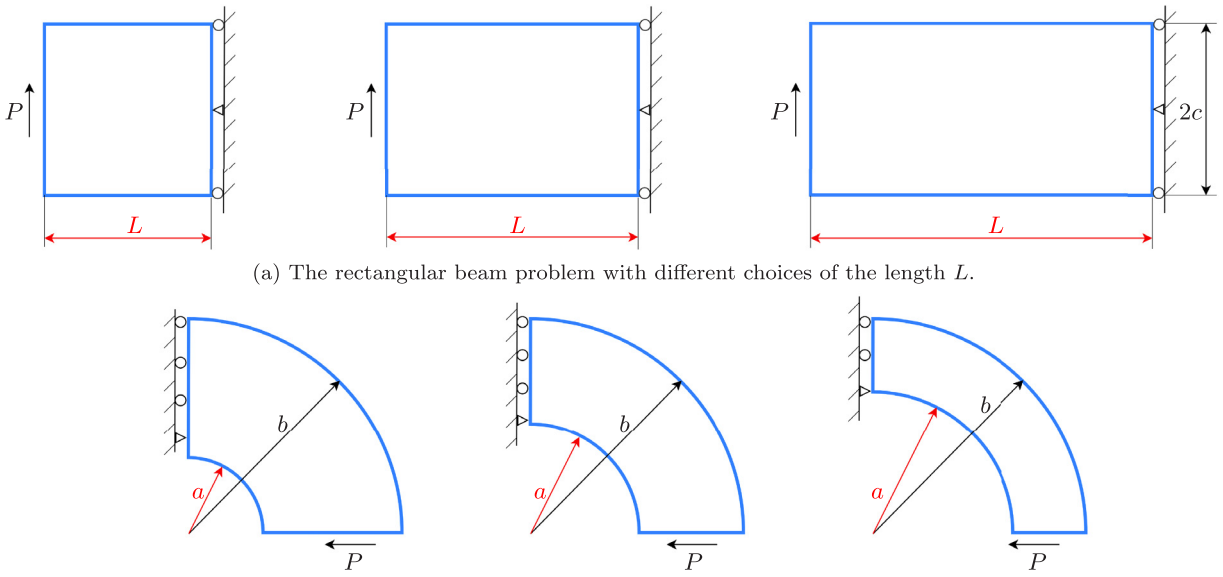
Knowledge of the analytical solution (see [29]) of the problems allows us to easily make quantitative evaluations as well. Indeed, we can compare the predictions of the network against the expected solution for any choice of L and of a. For compactness of notation, in the following analysis $e(\square)$ is used to denote the MSE on the prediction $\square$. For instance,

$$e(u_x) = |u_x - u_x^*|. \tag{25}$$

Fig. 9 plots the mean square error as defined in Eq. (6) for each of the solution profiles for varying L or a in the case of the cantilever beam and quarter-circle problems, respectively. The position of the training geometries is identified by a dashed red line.

It can be seen that the error on displacements and stresses is small at the training geometries and in their neighborhood. However, the error may become large between training geometries, especially when they are far apart, such as for the cantilever beam problem (see Fig. 9(a)). Using additional training values of L and a would likely reduce the increases in error in between training geometries shown in Fig. 9, as we later demonstrate considering nine training geometries.

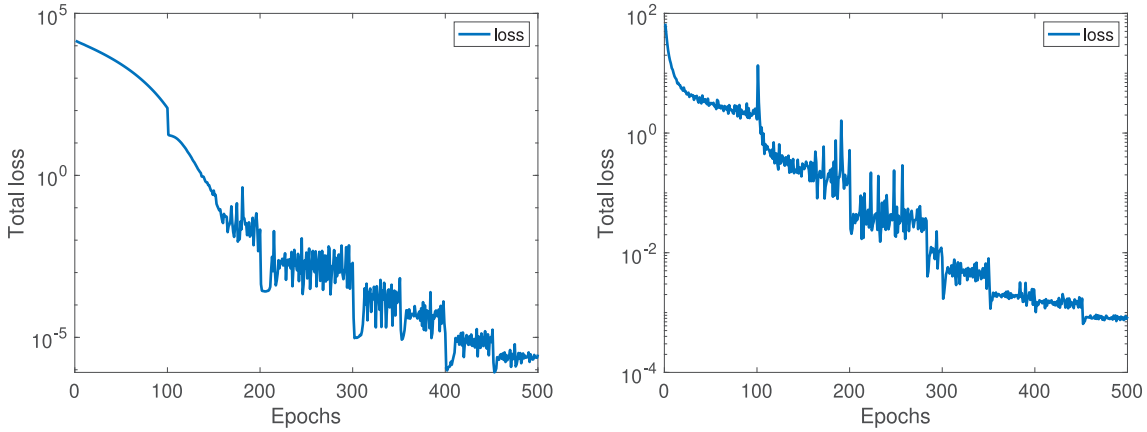
It is also worth noticing that the error is affected by the complexity of the analytical solution. For instance, the analytical stresses for the rectangular beam problem are quite simple (see, for instance, [28,29]). We even have $\sigma_{yy}^* = 0$ everywhere in the domain, for all shapes. Fig. 9(a) shows that the PINN is able to predict σ_{yy} with an accuracy of about 10^{-6} and σ_{xy} with an



(a) The rectangular beam problem with different choices of the length L.

(b) The circular beam problem with different choices of the radius a of the inner circle.

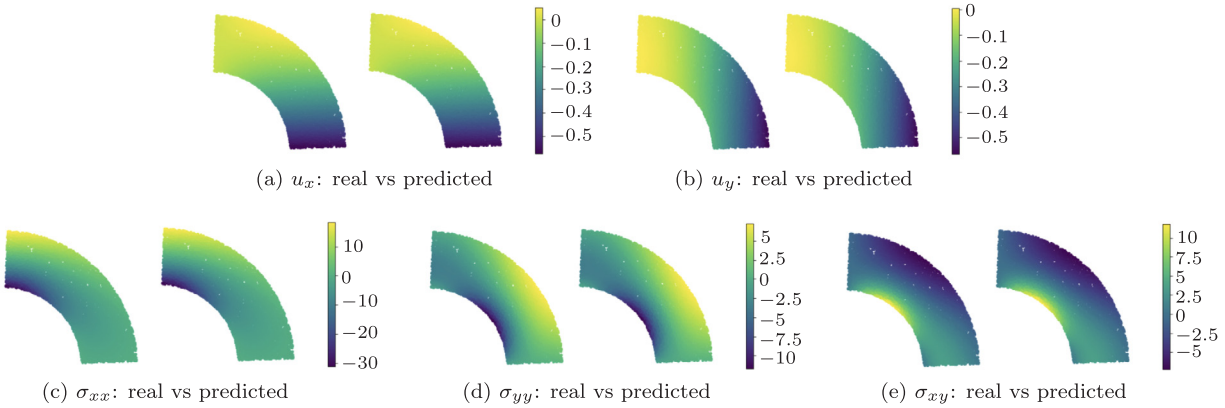
Fig. 6. Test problems with known analytical solution. Variable shape parameters are marked in red. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)



(a) Loss of PINN for the rectangular beam problem

(b) Loss of PINN for the circular beam problem

Fig. 7. Convergence curve of the loss function of PINN for the problems in Fig. 6.



(a) u_x: real vs predicted

(b) u_y: real vs predicted

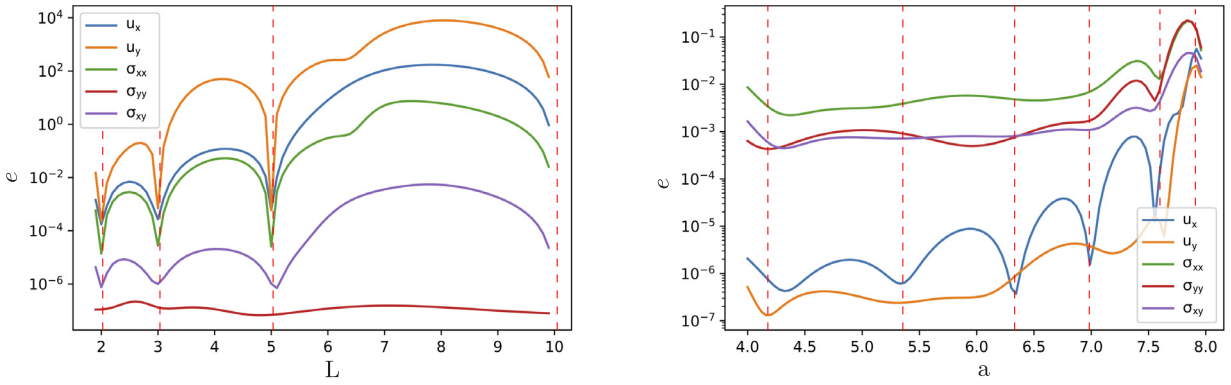
(c) σ_{xx}: real vs predicted

(d) σ_{yy}: real vs predicted

(e) σ_{xy}: real vs predicted

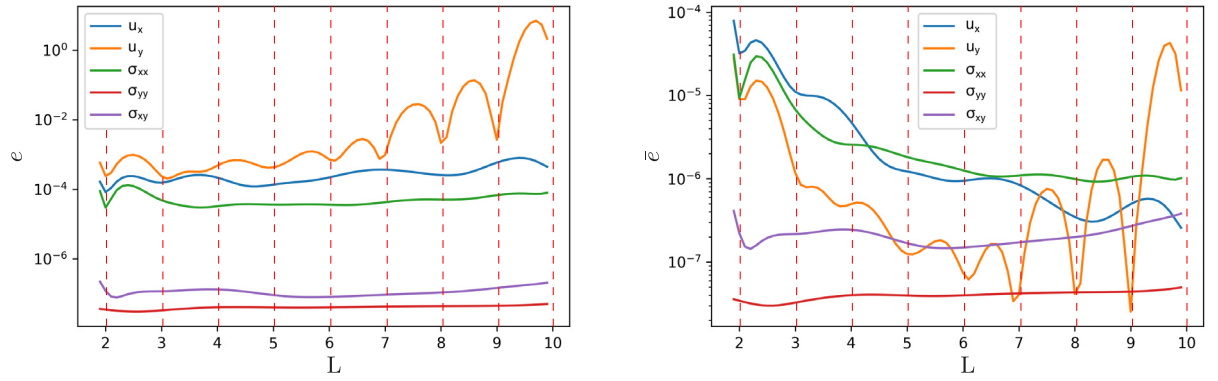
Fig. 8. Real (left) and PINN predicted (right) displacements and stresses for the circular beam problem with $a = 5.697$.

accuracy of at least 10^{-3} in the entire range of a when as few as four training geometries are used. Displacements, on the other hand, contain relations that depend on the square and on the cube of L. Hence, as L increases, the magnitude of the solution field increases, and small inaccuracies tend to produce large absolute errors. Because of this, larger values of the shape parameter



(a) MSE rectangular beam from 4 training geometries (b) MSE circular beam from 6 training geometries

Fig. 9. Mean square error on PINN-predicted displacements and stresses of the rectangular and of the circular beam problems.



(a) MSE of PINN for the rectangular beam (b) Normalized MSE of PINN for the rectangular beam

Fig. 10. Absolute and normalized MSE on displacements and stresses of the rectangular beam problem, when PINN is trained using 9 training geometries.

L for the cantilever beam problem is associated with greater extrapolation error for the displacement profiles, as can be observed from Fig. 9(a). Analogous observations can be made for the quarter-circle problem: as the shape parameter a increases, the displacement tends to increase, thus increasing the predictive error. This also suggests that the choice of the geometries used for the training may affect the accuracy of the network. For instance, it is reasonable to use training geometries closer to one another in ranges of the shape parameter where the solution changes quickly. Instead, it is reasonable to use fewer geometries in intervals of the shape parameter where the solution changes more slowly.

To verify that the accuracy of the network can be increased by using additional training geometries, the number of training values of L is increased from four to nine. The results are shown in Fig. 10, where we used equally spaced training lengths at $L = 2, L = 3, \ldots, L = 10$.

Fig. 10(b) reports the absolute MSE on displacements and stresses, denoted by e for compactness of notation. It can be noticed that increasing the number of training geometries from four to nine is enough to decrease the maximum error over the entire range of lengths by four orders of magnitude. This demonstrates that PINN can learn from the shape parameters, and that a limited number of training geometries can ensure high accuracy on a large range of shapes.

To account for the fact that the solution profile magnitude tends to increase as the shape parameter L increases, Fig. 10(b) shows a normalized measure of the error, where the MSE for all points plotted in Fig. 10(a) is divided by the mean value of the squared solution field. Considering, for instance, a field u_x of n_c components, we compute the mean magnitude of the squared solution field as

$$\bar{u}_x^* = \frac{\sum_{i=1}^{n_c} (u_{x_i}^*)^2}{n_c} \tag{26}$$

and the normalized MSE is

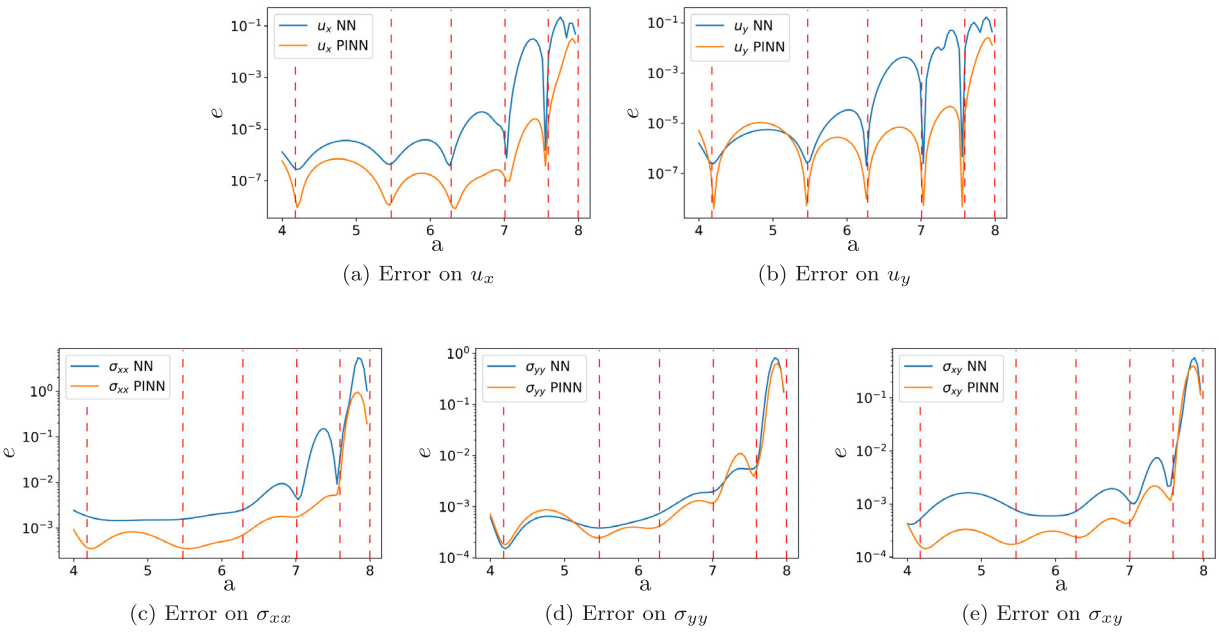
$$\bar{e}(u_x) = \frac{|u_x - u_x^*|}{\bar{u}_x^*}. \tag{27}$$

It can be seen from Fig. 10(b) that the normalized MSE $\bar{e}$ remains below 10^{-4} for all values of L.

4.2.2. Comparison of the accuracy of PINN and NN

While the previous subsection focused on validating shape parameterized PINNs, here we compare the accuracy of PINN and NN. In particular, the aim is to understand whether PINN has any advantage over standard NN they are trained to predict PDE solutions over changing domain shapes. In this regard, Fig. 11 shows the error on displacements and stresses for the circular beam problem after a training of 1500 epochs.

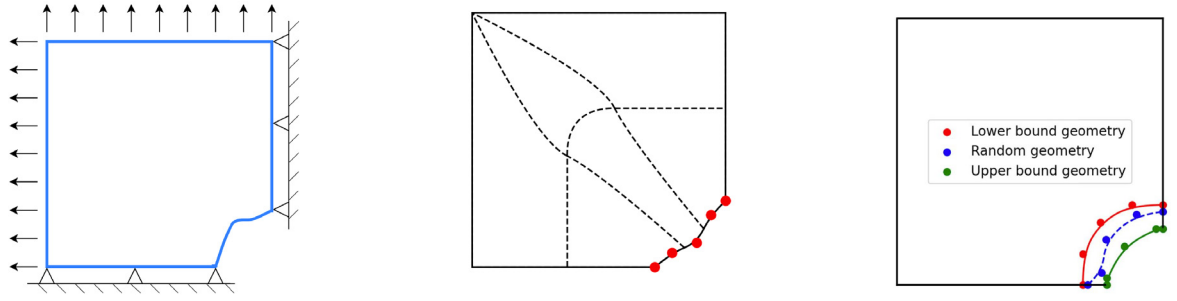
It can be seen from Fig. 11 that the PINN is generally a better predictor of solution profiles compared to the traditional NN. More precisely, Figs. 11(a) and 11(b) illustrate that PINN predictions of the displacement profiles tend to be 1–3 orders of magnitude more accurate than traditional NNs. The difference is less pronounced for stress profiles (Figs. 11(c)–11(e)), but still noticeable for σ_{xx} and σ_{xy}. In all cases, however, the better performances of the PINN seem to be linked to smaller error at the training shapes, rather than to better extrapolation capabilities. Indeed, the error of the PINN is smaller than that of the NN only



(a) Error on u_x (b) Error on u_y

(c) Error on σ_{xx} (d) Error on σ_{yy} (e) Error on σ_{xy}

Fig. 11. Comparison of accuracy of a PINN and NN for quarter-circle problem.



(a) The plate-hole problem. (b) Plate-hole shape with movable boundary parameterized by 5 control points. (c) Highlight of the freeform geometry change.

Fig. 12. The plate-hole problem.

when e of PINN is smaller than e of NN at the training shapes. As we get farther from a training shape, Fig. 11 shows that the error tends to increase at a similar rate in PINN and NN.

This is reasonable because the PDE residual does not contain any shape information. Therefore, when the shape is changed, the error is affected by the choice of the network only indirectly, in case PINN or NN is more accurate at the training geometries.

For the rectangular beam problem, the behavior is similar to the one described above. In this case, the difference between PINN and NN is even smaller, and NN even tends to be better than PINN at predicting the stresses.

4.3. Training and validation of PINN for the plate-hole problem

The previous section showed the capabilities of shape parameterized PINNs and NNs for simple problems with a single shape variable. In this section, a more complex plate-hole problem is considered. See Fig. 12(a) for a representation of an example domain for this problem. As shown in Fig. 12(b), in the following experiments, the changing boundary is represented by the position of 5 movable control points.

As was done with the one-parameter problems, the ability of PINN to simply learn the solutions is verified first. To do this, a PINN was trained to learn the solution profiles using the labeled data and collocation points of 100 different geometries. Fig. 13 reports the convergence of the loss function in a training of 1200 epochs. The starting learning rate is $l_r = 0.001$, which is gradually decreased to $l_r = 0.0005$ at iteration 300, $l_r = 0.0002$ at iteration 700, and $l_r = 0.0001$ at iteration 1100. A visual comparison of the PINN predictions and the expected solution profile is reported in Fig. 14.

As regards a more quantitative evaluation of the accuracy on the validation geometry, for completeness, we hereafter report the actual values of the error on displacements and stresses for the shape of Fig. 14:

$e(u_x) = 2.6\text{E-}6 \quad e(u_y) = 2.5\text{E-}6 \quad e(\sigma_{xx}) = 2.1\text{E-}4$

$e(\sigma_{yy}) = 2.6\text{E-}4 \quad e(\sigma_{xy}) = 1.1\text{E-}4.$

Detailed quantitative analyses are performed in following subsections on the effects of the number of training geometries, continuity, and basis degree.

4.4. Comparison between PINNs and NNs for the plate-hole problem

The role of incorporating the physical information in the loss function can be evaluated by comparing the accuracy and the data efficiency of a PINN and a NN. This comparison is performed in the context of the plate-hole problem.

To study the accuracy of the neural networks, by e_d we here denote the sum of the data-based mean square errors for a given

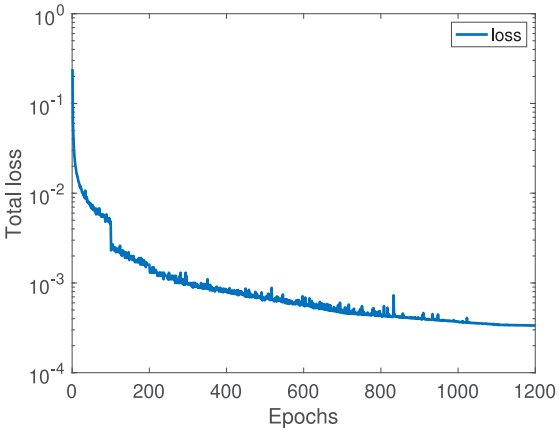


Fig. 13. Loss function of PINN for the plate-hole problem.

geometry, i.e.

$$e_d := |\hat{u}_x - \hat{u}_x^*| + |\hat{u}_y - \hat{u}_y^*| + |\hat{\sigma}_{xx} - \hat{\sigma}_{xx}^*| + |\hat{\sigma}_{yy} - \hat{\sigma}_{yy}^*| + |\hat{\sigma}_{xy} - \hat{\sigma}_{xy}^*|. \tag{28}$$

Similarly, e_r represents the sum of all the residual terms of the loss, which, for a problem without body forces, is

$$e_r := \left|(\hat{\epsilon}_{xx} + \nu\hat{\epsilon}_{yy})\frac{E}{1-\nu^2} - \hat{\sigma}_{xx}\right| + \left|(\nu\hat{\epsilon}_{xx} + \hat{\epsilon}_{yy})\frac{E}{1-\nu^2} - \hat{\sigma}_{yy}\right| + \left|\frac{E}{(1+\nu)}\hat{\epsilon}_{xy} - \hat{\sigma}_{xy}\right|. \tag{29}$$

These values are computed for each shape of the dataset. The normalized quantities $\bar{e}_d$ and $\bar{e}_r$ are obtained by dividing, respectively, e_d and e_r by the mean of the squared displacements and stress fields. Note that this was also done in [30] for the purpose of normalizing the results with respect to the magnitude of the solution fields. Therefore, the normalization implies dividing e_d and e_r corresponding to the kth geometry of the dataset by

$$\frac{1}{5n}\sum_{i=1}^{n_c}\left((\hat{u}_{x_i}^*)^2 + (\hat{u}_{y_i}^*)^2 + (\hat{\sigma}_{xx_i}^*)^2 + (\hat{\sigma}_{yy_i}^*)^2 + (\hat{\sigma}_{xy_i}^*)^2\right) \tag{30}$$

of the kth geometry, where n is the number of collocation points where displacements and stresses are evaluated.

In the following analysis, a dataset of 2500 geometries is used and both neural networks are trained using the same 100 geometries. Figs. 15(a) and 15(b) report $\bar{e}_d$ and $\bar{e}_r$ for the training and validation geometries, respectively.

Figs. 15(a)–15(b) show that PINN predictions have residuals that are roughly one order of magnitude lower than NN predictions. This applies to both training and validation geometries and it occurs because of the PINN's use of a residual term in the loss function, which implies that a soft zero-residual condition is explicitly enforced by PINN. Hence, the residual is explicitly reduced by PINN. The data accuracy of PINN and NN is similar. For the training subset, Fig. 15(a) shows that the data error is in a particularly small range, with $\bar{e}_d \approx \mathcal{O}(10^{-4})$ in both cases.

Fig. 15(b) also demonstrates that our approach can effectively predict PDE solutions on freeform shapes. Indeed, the predictions in Fig. 15(b) occur on 2400 geometries that the networks did not see during training. Since the data error is $\mathcal{O}(10^{-3}) - \mathcal{O}(10^{-4})$ and the residual is either $\mathcal{O}(10^{-2})$ or $\mathcal{O}(10^{-4})$ depending on if a NN or PINN is used (respectively), it can be concluded that the shape parameterized networks are accurate.

Figs. 15(c) and 15(d) report the value of the mean e_r in the training and validation subsets as the number of geometries used to train the NN and PINN is varied. In both subsets, the PINN needs fewer training geometries than NN to achieve the same residual accuracy. More specifically, the PINN achieves a mean normalized residual of $\mathcal{O}(10^{-2})$ with about 20 training geometries. With the NN, about 100 training geometries are required to achieve the same residual accuracy. Thus, Figs. 15(c)–15(d) establish the importance of including a physics-based term in the loss function to achieve better residual accuracy with the same number of training geometries.

Assuming that we want a network with a given residual accuracy, PINNs training can be more computationally efficient than traditional NNs. To show this, Table 1 compares the (wall clock) time-to-train a PINN and a NN for one epoch on a local machine for an increasing number of training geometries.

Table 1 shows that training PINN for one epoch is more expensive than running one training epoch of NN when the same number of geometries is used. This is expected, since the additional loss terms in the PINN necessitate that additional operations be executed during training. However, the better data efficiency of PINN can make it more computationally efficient. Based on estimates derived from Figs. 15(c)–15(d), it is possible to train a PINN using just 20–25 geometries to achieve the same residual accuracy as a NN trained on 100 geometries. In this setting, one epoch of PINN on 25 geometries would take around one third of the computational time of NN trained on 100 geometries. Therefore, while PINNs may be more expensive to train than traditional NNs under an epoch-by-epoch comparison, they are more efficient than NNs in some cases because of their increased data efficiency.

4.5. Point-wise error of shape-parameterized PINNs and NNs

In the above analysis, we have analyzed average measures of the error. This was made necessary not only by the consideration of 2D fields, but also by the fact that we needed to provide a measure of the error on hundreds of different domains. Average measures of the error were then necessary to represent compactly the accuracy of the entire neural networks, as in Fig. 15.

However, averaged results do not provide much information on the local accuracy of the neural networks in each domain. Therefore, for completeness, we here consider the same setting of Section 4.4 and we provide the actual plots of the errors for two of the considered domain. In particular, we report:

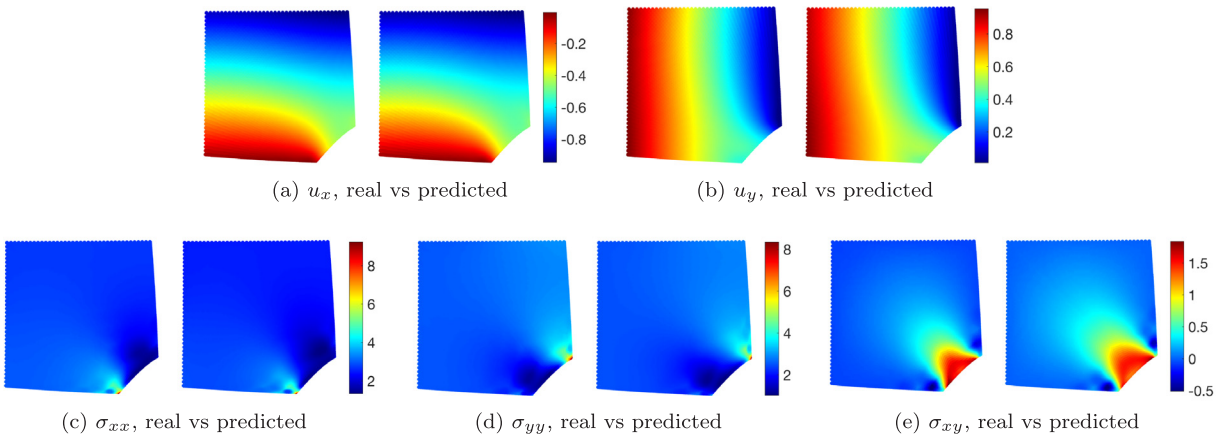
- the error for a random training domain;
- the error for a random validation domain.

Such errors for the shape-parameterized PINNs and NNs are shown in Figs. 16 and 17.

Consistently with Fig. 15, the magnitude of data errors is similar in PINNs and NNs. Furthermore, the error has limited variations in the domain: for instance, in both Figs. 16 and 17, the errors on the displacements are in the order of 10^{-3} everywhere. Similarly, the errors on stresses are in the order of 10^{-2} everywhere. These results suggest that the average errors considered in previous subsections do not hide significantly large local errors within each domain.

4.6. Effect of continuity and degree on the accuracy

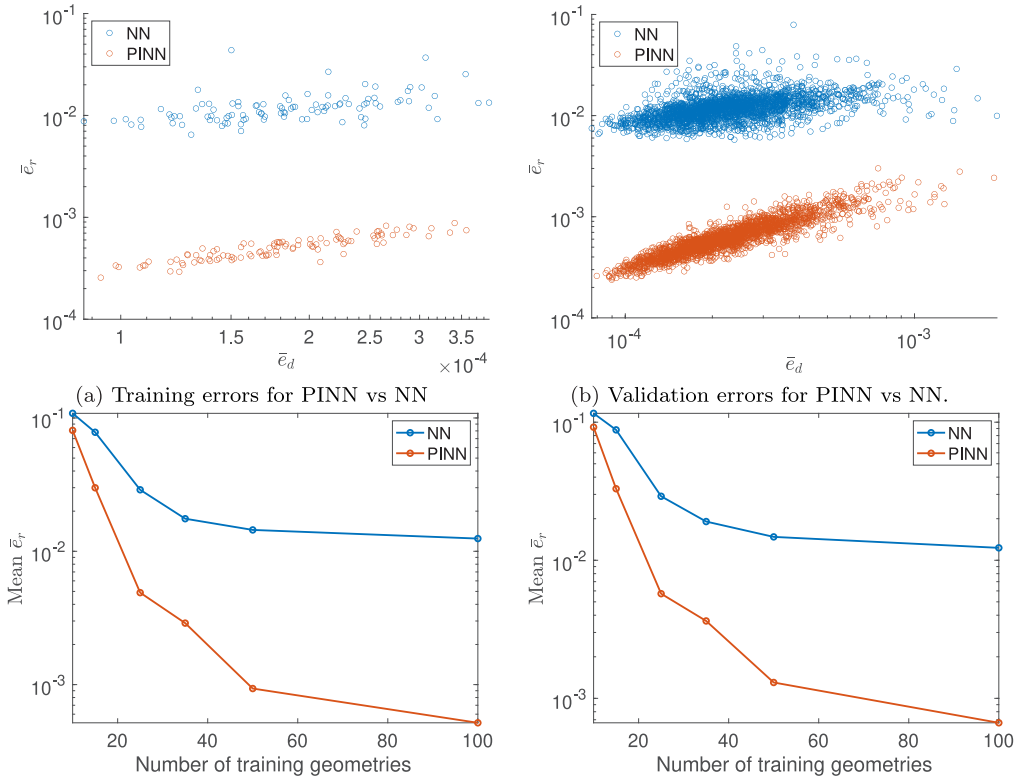
As described at the beginning of Section 4, we generated the training data through IGA. In this subsection, the role of the continuity of the NURBS representation of the geometry and the degree of the basis functions are investigated. In this regard, by labels like "$(C^0, p = 1)$" we denote networks that have been trained on datasets that have been generated using C^0 continuous



(a) u_x, real vs predicted (b) u_y, real vs predicted

(c) σ_{xx}, real vs predicted (d) σ_{yy}, real vs predicted (e) σ_{xy}, real vs predicted

Fig. 14. Qualitative comparison between real (left) and PINN predicted (right) displacements and stresses for a random geometry of the plate-hole dataset. The shape was not used for training.



(a) Training errors for PINN vs NN (b) Validation errors for PINN vs NN.

(c) Data efficiency for PDE residual - training subset (d) Data efficiency for PDE residual - validation subset

Fig. 15. Data efficiency of PINN and NN for the plate-hole problem.

Table 1
Times to run one epoch of PINN and NN using different numbers of training geometries.

Number training geometries	Time 1 epoch (train), NN [s]	Time 1 epoch (train), PINN [s]
10	3	4
15	5	6
25	8	10
35	11	15
50	15	21
100	30	41

meshes and basis functions of degree $p = q = 1$. The plate-hole problem is re-analyzed using 11 control points (corresponding to 20 shape variables), which is the minimum number of shape variables that can be used to study continuities as high as C^2 and basis function degrees as high as $p = q = 5$. A schematic of the plate-hole problem when 11 control points are used is shown in Fig. 18. The increased number of shape variables is expected to lead to more dispersed data than those in Fig. 15, but it also

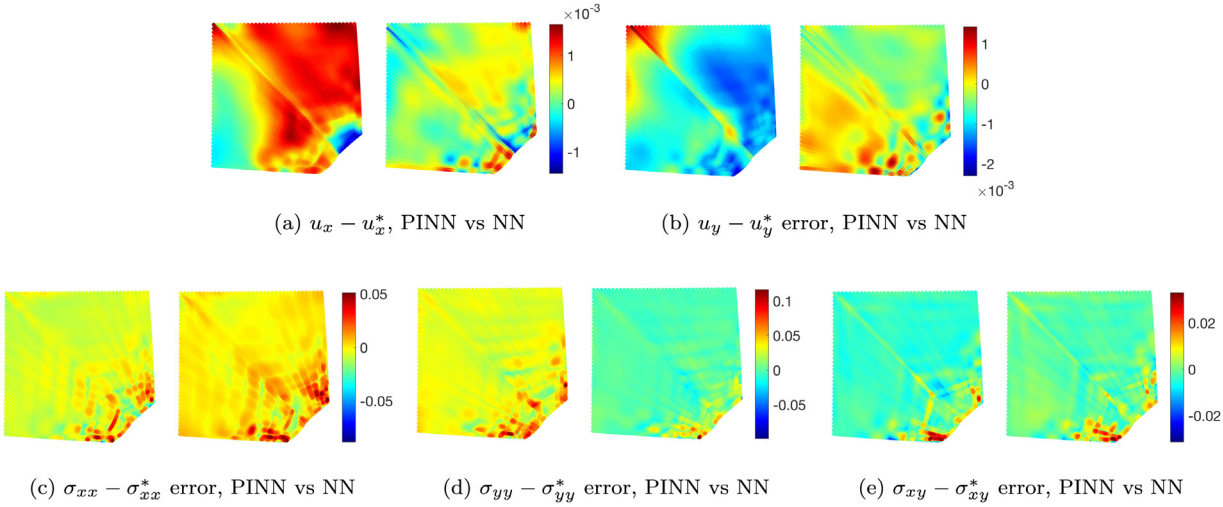


Fig. 16. Point-wise error for a training geometry of the plate-hole dataset.

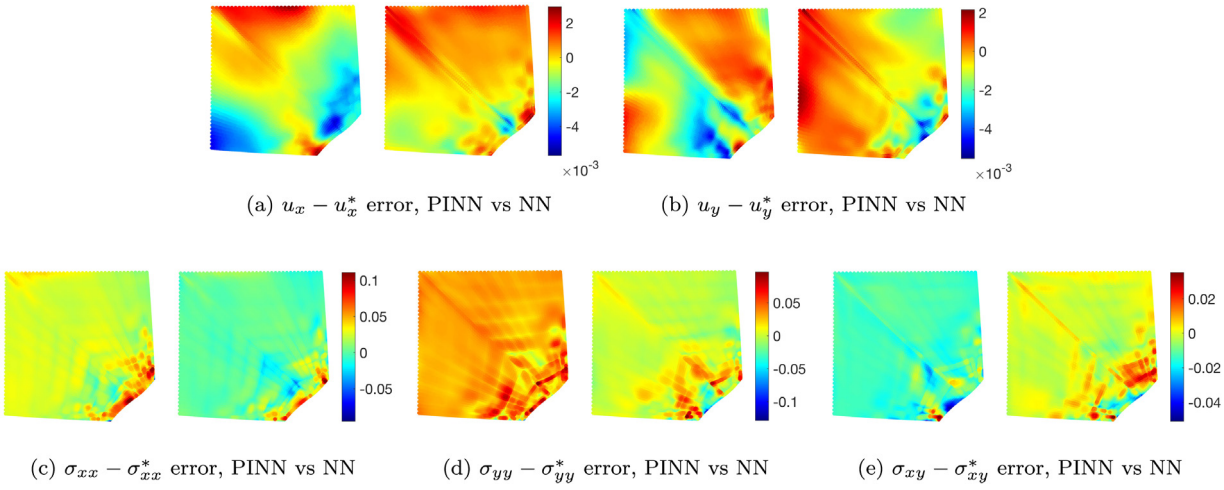


Fig. 17. Point-wise error for a validation geometry of the plate-hole dataset..

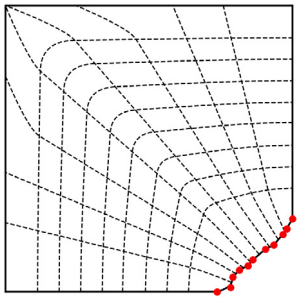


Fig. 18. The plate-hole problem with 11 control points.

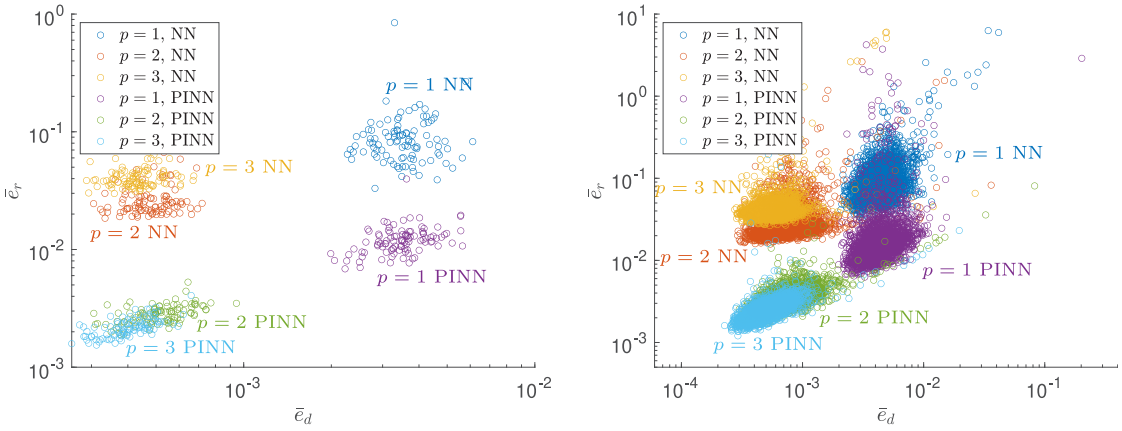
allows us to evaluate the behavior of the network when many shape parameters are used.

Fig. 19(a) shows the scatter plots of $\bar{e}_d$ and $\bar{e}_r$ of NN and PINN for the 100 training shapes only when the degree of the basis functions is changed. Fig. 19(b) reports the same results for the 2400 validation shapes of the dataset. Thus, Fig. 19(b) provides an error analysis that includes hundreds of shapes that were not used for training.

Fig. 19 shows that PINNs achieve an MSE residual that is roughly one order of magnitude smaller than NNs. This further demonstrates that PINNs produce more self-consistent solution fields. Data MSE is, instead, roughly the same in NNs and PINNs. Fig. 19 also shows that the accuracy of the network is affected by the degree of the basis functions. Interestingly, we observe the same behavior of the error both on the training (Fig. 19(a)) and on the validation subsets (Fig. 19(b)).

In particular, Fig. 19 shows that the neural networks based on the NURBS parameterization with $(C^0, p = 1)$ have less consistent predictive accuracy than counterparts trained on data generated using higher order basis functions. For instance, it can be seen in Fig. 19(b) that a few of the predictions of the PINN trained on data generated using linear basis functions have a large residual and data error. Fig. 19(b) also illustrates that, in this example, PINNs becomes steadily more accurate when the basis degree used to generate data is increased, albeit at a decreasing rate. Similar comments can be made for the NN predictions shown in Fig. 19(b). However, since NN does not incorporate the residual of the PDE, the residual accuracy is not strictly linked to the data accuracy. This explains why, in Fig. 19, the NN based on the NURBS parameterization with $(C^0, p = 2)$ characteristics appear to have better residual accuracy compared to the NN trained on data generated from IGA that has $(C^0, p = 3)$ characteristics.

This behavior suggests that $(C^0, p = 1)$ data is only partially adequate to generate training data for the network. The low-order



(a) MSE on the training subset for different degrees of the basis functions and fixed continuity C^0.

(b) MSE on the validation subset for different degrees of the basis functions and fixed continuity C^0.

Fig. 19. Effect of the degree of the basis functions on the accuracy of NN and PINN.

setting may lead to stress concentration or numerical difficulties that hinder the learning process. When the basis degree used to generate data is sufficiently high so as to avoid these issues, further increasing the degree leads to smaller and smaller improvements in the predictive accuracy of the trained PINN.

Considering the effect of both the type of network and of the degree of the basis, Figs. 19(a) and 19(b) show that between $(C^0, p = 1)$-NN and $(C^0, p = 3)$-PINN there is an improvement of about 2 orders of magnitude in the residual accuracy and of one order of magnitude in the data accuracy.

Finally, Fig. 19 also demonstrates that the proposed formulation for shape-informed neural networks can handle problems that use many shape parameters. Indeed, the networks used in this subsection received 20 shape parameters as inputs, which could be used to represent relatively complex shapes with a NURBS curve. Despite the relatively large number of inputs compared to a traditional (non shape-informed) PINN, both data and residual accuracy of PINN are $\mathcal{O}(10^{-3})-\mathcal{O}(10^{-4})$ when continuity and degree are sufficiently high.

4.7. Data requirements of PINN and NN

In this subsection, the role of the availability of data is studied. In particular, we are interested in providing some remarks on the data requirements of the shape-parameterized neural network and on their differences when fewer training data are available. In this context, we also aim at verifying whether PINN and NN still have comparable data accuracy (as noticed in Sections 4.2.2 and 4.6) when only small amount of data is available.

First, we analyze the shape-parameterized PINN and NN can be trained when data is only available at the boundaries of the domain. In this regard, we consider the circular beam problem with a single training geometry at $a = 4.241$. We assume that the solution is known along the entire boundary of the domain, but no data is available in the interior. Figs. 20(a) and 20(a) show the convergence of the loss function of the training of NN and PINN, respectively. Fig. 20(c) shows the MSE on the entire domain of displacements and stresses of the NN predictions at the training geometry and for nearby shapes. Fig. 20(d) provides the same information for PINN. Finally, Fig. 20(e) visually compares the solution u_x^* at the training geometry $a = 4.241$ with PINN and NN predictions.

Figs. 20(c) and 20(d) show that both NN and PINN have large errors in the domain when the network was trained using only boundary data. This is especially evident for the displacements.

Table 2
MSE of NN and PINN predictions at the training geometry $a = 4.241$ when the neural networks are trained with a limited amount of data on the interior of the domain.

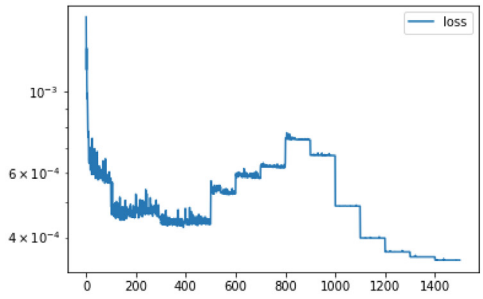
	NN	PINN
$e(u_x)$	1.19E−4	5.47E−7
$e(u_y)$	1.21E−6	1.88E−7
$e(\sigma_{xx})$	0.062	0.05
$e(\sigma_{yy})$	0.159	0.080
$e(\sigma_{xy})$	0.124	0.042

Notwithstanding this, Fig. 20(e) shows that PINN may still qualitatively predict the solution (in this case, the displacement u_x): indeed, although some differences are noticeable between u_x^* and the PINN's prediction, the overall solution fields are similar. This is due to the presence of the residual term in the PINN loss, which contributes to improving the predictions even when limited amount of data is available. The fields predicted by NN is, instead, completely different from the expected solution. This is reasonable, as there is no relationship that can be used by the network to correctly predict the solution in the interior of the domain. The difference between PINN and NN is apparent even from the loss convergence: indeed, in PINN (Fig. 20(b)) the loss converges slowly, as it is trying to reduce the residual and fit the data. In NN, (Fig. 20(a)) the convergence is, instead, almost immediate, as it is easy for the neural network to fit the small amount of provided data. At the same time, however, this does not allow to achieve predictive capabilities of the solution in the interior of the domain.

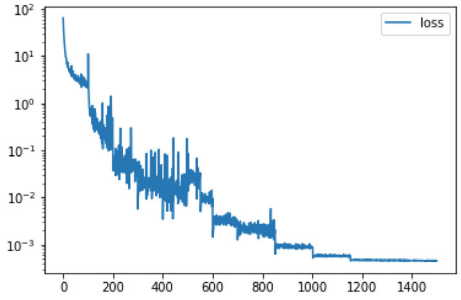
In order to better explore the data requirements of the two networks, we now provide some training data in the interior of the domain as well. The network, nonetheless, will still be required to make predictions on the entire grid or 100×100 collocation points.

In particular, we assume that the training data is available only in the collocation points at $\xi = 0.05$, $\xi = 0.35$, $\xi = 0.65$, and $\xi = 0.95$. Since we are using a grid of 100×100 uniformly distributed collocation points, considering 4 locations of ξ means that, during the training, only $4 \times 100 = 400$ collocation points out of 10,000 are used as available labeled data. The training MSE of displacements and stresses for PINN and NN are compared in Table 2 for a network trained on a single geometry of the circular beam problem with radius of the inner circle $a = 4.241$.

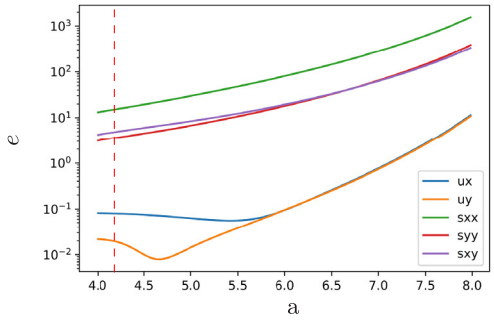
Table 2 shows that PINN errors are significantly smaller than NN errors at training. In particular, $e(u_x)$ is smaller by more than two orders of magnitude when PINN is used instead of NN. Errors



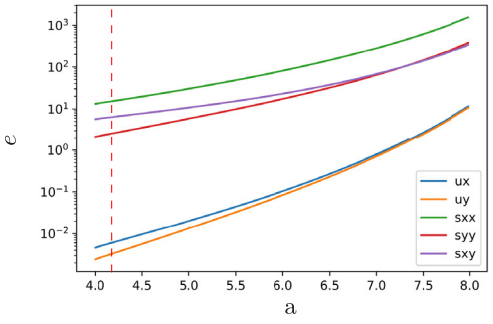
(a) Loss function of NN training.



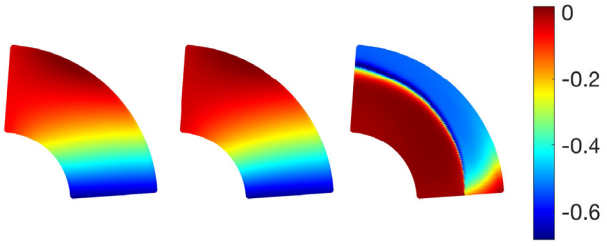
(b) Loss function of PINN training.



(c) Accuracy of NN as a varies



(d) Accuracy of PINN as a varies



(e) From left to right: visual comparison between u_x^*, PINN-predicted u_x, NN-predicted u_x for the training geometry.

Fig. 20. Analysis of the accuracy of PINN and NN when training data is provided only at the boundaries of the domain.

on stresses are all smaller in the PINN as well. This increased accuracy at the training shape propagates to nearby shapes. This is shown in Fig. 21, which represents the mean square errors on displacements and stresses when the shape parameter a is changed. In the figure, the position of the only training geometry used in this experiment is identified by the dashed red line. It can be noticed that the error is smallest at the training geometry in both PINN and NN. However, PINN error is generally much smaller than NN error, and this difference propagates in a neighborhood of the training value of a. When we get farther from the training geometry, both networks become similarly inaccurate, as it could be expected from the previous analysis.

If we use more than one training shape, the advantage of shape parameterized PINN with little data becomes even more apparent. Using, for instance, 6 training shapes, the displacements can be predicted with a significantly higher accuracy by PINN than by NN. This is shown in Fig. 22, where we notice that PINN is significantly more, up to 2 orders of magnitude more, accurate than NN in the entire range of geometries that we are considering.

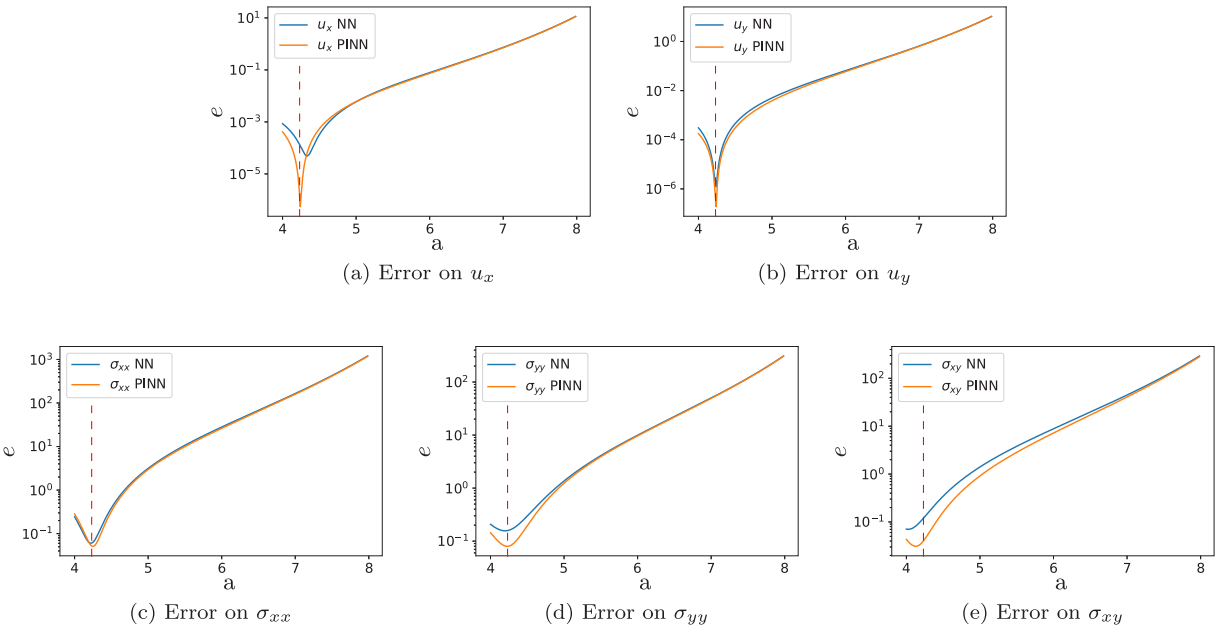
The advantage of data accuracy of PINN prediction over NN prediction for small amount of data can be ascribed to the PDE residual term used in the PINN loss function.

4.8. Transfer learning

One important feature of neural networks trained to predict the solutions to parameterized PDEs is their ability to 'transfer learn' or use what was learned in one scenario to accelerate learning for a 'nearby' scenario.

To study this situation, the plate-hole problem is reconsidered. However, an additional constraint is placed on the plate-hole profiles: all profiles are exact ellipses. Because of this choice, there are two shape parameters which are the lengths of the two semi-axes of the ellipse, a_1 and a_2, as shown in Fig. 23(a).

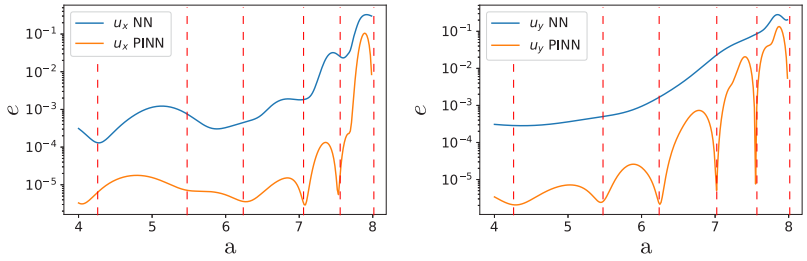
To study transfer learning, a PINN is trained to learn the solution profiles for the case of $a_1 = a_2 = 5$. In the reference, or non transfer learning, case a second PINN is then formed and trained to learn the solution profiles for $a_1 = 5, a_2 = 5.6$ starting from random hyperparameters. For the transfer learning case, a PINN is trained to learn the solution profiles for $a_1 = a_2 = 5$, and then the same PINN is trained to learn the solution profiles for $a_1 = 5, a_2 = 5.6$. In both cases, the second PINN is trained using only labeled data on the boundaries, i.e. no interior labeled data is used. The training is run for a maximum of 2000 epochs starting from a learning rate $l_r = 0.0025$, which is halved every 50 epochs that do not produce a sufficient reduction of the loss. Fig. 23(b) reports the convergence curves of the loss for these trainings.



(a) Error on u_x (b) Error on u_y

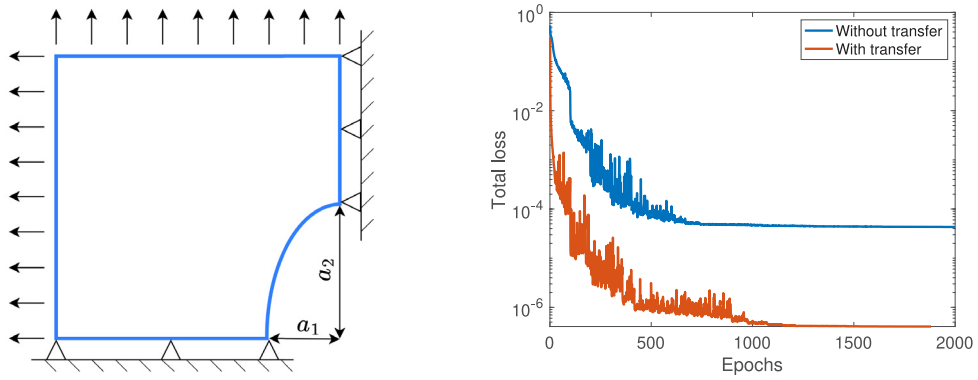
(c) Error on σ_{xx} (d) Error on σ_{yy} (e) Error on σ_{xy}

Fig. 21. Comparison of accuracy of a PINN and NN over varying radii a of the inner circle for the quarter-circle problem trained on a single geometry with limited training data.



(a) Error on u_x (b) Error on u_y

Fig. 22. Comparison of accuracy of a PINN and NN over varying radii a for the quarter-circle problem trained on 6 training shapes with limited training data.



(a) The plate-hole problem with elliptical hole.

(b) Convergence of PINN with and without transfer learning from a nearby shape.

Fig. 23. Definition of the plate-hole problem with elliptical hole and convergence of PINN with and without transfer learning from a nearby shape.

Fig. 23(b) shows that the value of the loss is smaller at all epochs when we use the information of $a_1 = a_2 = 5$ to train the network on the new shape. The difference in accuracy is significant: Fig. 23(b) shows that the value of the loss function with transfer learning is consistently smaller by about two orders of magnitude with respect to the non-transfer learning loss function. This also leads to a faster convergence, as less epochs are needed to achieve the same loss function when

transfer learning is used. This shows that a shape-informed neural network can transfer what it has learned about the solution profiles on one geometry to another, nearby, geometry.

4.9. Shape optimization using shape-parameterized surrogate models

In this section, we show that the shape-parameterized neural networks are attractive to solve shape optimization problems. In this regard, consider the problem that consists in determining the optimal shape to give to the plate-hole so that the compliance is minimized under a volume constraint. The solution of this problem is known: in particular, the optimal shape for an infinitesimal hole is circular.

To address this problem, a shape-parameterized neural network is here trained on 100 different shapes characterized by the same volume $V = 9600$ (the entire plate is 100×100; hence, the volume of a square plate would be 10000). The shape-parameterized NN is used for simplicity, as the previous analysis showed that the data accuracy of PINN and NN is similar for the plate-hole problem. The shapes of the dataset were generated by IGA using 5 control points (8 shape parameters) for the NURBS parameterization.

For this problem, data generation consisted of a three-step process.

1. Create a plate-hole geometry by randomly relocating the control points which influence the hole profile shape, $\mathbf{p}_h^0$. The volume of this geometry may not be close to V, the desired volume.
2. Solve the following equality-constrained optimization problem to adjust the original geometry's control points, $\mathbf{p}_h^0$, to obtain a new set of control points, $\mathbf{p}_h$, which correspond to a geometry with a volume close to the desired volume, V:

$$\min_{\mathbf{p}_h} \ \|\mathbf{p}_h - \mathbf{p}_h^0\|_2^2 \tag{31a}$$

$$|\Omega| = V \tag{31b}$$

3. Use IGA to compute the displacement profiles associated with the new geometry, and sample the displacement and stress fields uniformly in the parametric domain at 100^2 points.

The motivation for the exact formulation of Eq. (31) is twofold. First, a plate-hole geometry with a random hole profile will have a volume that may not be near the desired volume V. To increase the relevance of the training data used by the neural network which will be used for shape optimization, it is desirable to find a way to generate geometries with specific volumes near the desired target volume. Solving Eq. (31) achieves this. Second, it is desirable to generate representative samples of the types of inputs which the network is expected to make predictions based on for the purposes of training. In the context of this problem, this corresponds to randomly sampling the design space. A representative geometry should be created by assigning random values to $\mathbf{p}_h$ between a lower and upper bound in such a way that the volume of this geometry is V. However, it is difficult to analytically represent the space of plate-hole geometries with volumes exactly equal to V. Instead, we generate a random geometry, then find the minimal (with respect to the l_2-norm) adjustment of the control points of this geometry such that it has a volume of V.

After the training, the optimal shape to give to the hole is sought by running 50 iterations of a constrained variation of a trust-region quasi-Newton method on the surrogate model constituted by the trained neural network. The volume constraint $\bar{V} \leq 9600$ is here imposed. The convergence of the optimization and the optimized profile are reported in Fig. 24. In all cases, the optimization required just a few seconds to be run. The IGA values of compliance and volume are reported in Table 3.

Table 3
Values of the compliance and of the volume for the considered shape optimization problem with $\bar{V} \leq 9600$.

	Initial	Optimized	IGA solution [31]	Relative error
Compliance	484.55	467.24	466.57	0.14%
Volume	9465.36	9599.47	9600	0.0055%

Table 4
Values of the compliance and of the volume for the considered shape optimization problem with $\bar{V} \leq 9700$ based on training data of $V = 9600$.

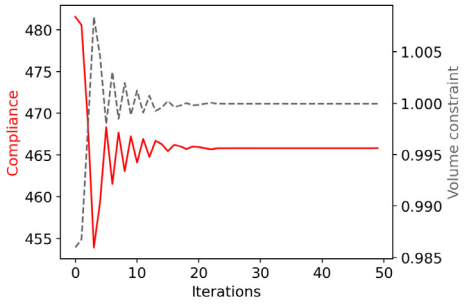
	Initial	Optimized	IGA solution	Relative error
Compliance	486.62	454.85	453.65	0.26%
Volume	9465.15	9699.86	9700	0.0014%

The optimization converges and the optimized profiles resembles the circle that represents the theoretically optimal hole profile. Although Fig. 24(b) shows a visible difference between the optimized and analytical hole profiles, the volume constraint is respected and the optimized compliance is just larger than the theoretical one reported in Table 3. Hence, the optimization works successfully on the surrogate model constituted by the trained neural network. The difference between the optimized and the analytical profile can be explained by the fact that the optimized profile already has a close-to-optimal compliance and further improvements would require a more accurate training of the neural network. Therefore, the results of Fig. 24 and Table 3 suggest that the shape-parameterized neural networks are attractive for applications in shape optimization and in the meta-optimization of geometry.

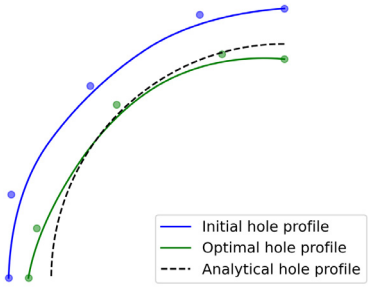
In this regard, it is interesting to evaluate what happens when the volume constraint no longer coincides with the volume of the shapes used to train the neural network. In particular, Fig. 25 shows the results of the optimization when $\bar{V} \leq 9700$ is imposed. The optimization is run on the same surrogate model as above, which was created by considering shapes of volume $V = 9600$. The corresponding values of the compliance and of the volume are reported in Table 4. The deviation from the circle and the relative error on the compliance are more significant than in the $\bar{V} \leq 9600$ case. However, this was expected, as the neural network was trained with shapes of volume $V = 9600$. Furthermore, Table 4 shows that the relative error remains in the order of 10^{-3}.

If the volume constraints is set farther away from the volume of the training shapes, the shape optimization can no longer be performed accurately without acting on the neural network. For instance, consider a shape optimization problem with $\bar{V} \leq 9000$. Such volume is outside of the data normalization range used in the previous experiments and a direct application of the previously trained network would mean asking the network to predict the PDE solution for negative shape variable inputs, which were not seen during training. Therefore, the shape-parameterized neural network was re-trained to analyze this case. In particular, the behavior of the neural network is here studied when the training is performed in the following three situations:

- the training dataset is formed by 100 shapes of volume $V = 9000$. The training is run for 1200 epochs. We denote such case by "9000";
- the training dataset is formed by 100 shapes of volume $V = 9000$, and the training is run for just 100 epochs. We denote such case by "9000_{100}";

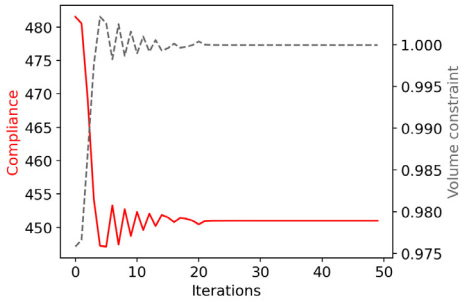


(a) Convergence of compliance and volume.

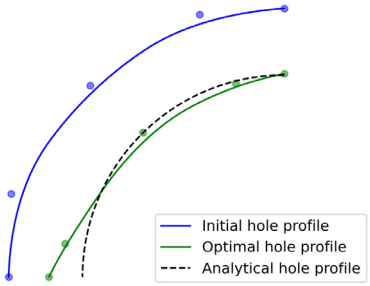


(b) Profiles of initial and optimized solutions.

Fig. 24. Results of the shape optimization problem with $\bar{V} \leq 9600$.



(a) Convergence of compliance and volume.



(b) Profiles of initial and optimized solutions.

Fig. 25. Results of the shape optimization problem with $\bar{V} \leq 9700$ based on training data of $V = 9600$.

Table 5
Values of the compliance and of the volume for the considered shape optimization problem with $\bar{V} \leq 9000$, from different initial shapes. Different training settings of the neural network and different starting iterates of the shape optimization are considered. In all cases, the exact IGA solution is 551.11.

Initial compliance, volume	Case	Final volume	Optimized compliance	Relative error
560.73, 8937.23	9000	8999.80	551.28	0.031%
	9000_{100}	8999.74	551.51	0.073%
	9000T	8999.93	551.22	0.020%
	$9000T_{100}$	8999.69	551.26	0.027%
580.70, 8810.79	9000	8736.99	597.01	8.3%
	9000_{100}	8782.03	585.91	6.3%
	9000T	8999.85	551.70	0.11%
	$9000T_{100}$	8999.44	552.80	0.31%

- the training dataset is formed by 100 shapes of volume $V = 9000$ and transfer learning is used. In particular, the training is initialized from the final weights of a previous training on a dataset made of 100 shapes of volume $V = 9600$. We denote such case by "9000T";
- the training is conducted as in the case "9000T", but the procedure is stopped after just 100 epochs. We denote such case by "$9000T_{100}$".

The results of the optimization using different starting iterates are reported in Table 5, where the relative errors are referred to the compliance.

When the starting iterate is sufficiently close to the solution (top row in Table 5), it shows that the value of the optimal compliance can be computed quite accurately in all four cases. However, when the starting iterate is set farther away from the solution (bottom row in the table), the shape optimization exhibits large errors in the "9000" and "9000_{100}" cases. The optimization is here unsuccessful. However, when the shape optimization is run on neural networks that use transfer learning, the results are still sufficiently accurate. This is particularly remarkable in the "$9000T_{100}$" case, where only 100 training epochs were performed on shapes of volume $V = 9000$. This suggests that transfer learning not only can be beneficial to the accuracy of the optimization on larger sets of shapes, but it is also efficient, requiring just few epochs of re-training.

Finally, we have trained a shape-parameterized neural network using a training dataset made of 100 shapes of volume $V = 9000$ and of 100 shapes of volume $V = 9600$. Then, we have run the shape optimization setting $\bar{V} = 9000$, $\bar{V} = 9400$, and $\bar{V} = 9600$ for the volume constraint. In these experiments, when $\bar{V}$ is set to a value that is represented in the training dataset, the shape optimization is run using only the training shapes of such volume. Indeed, performing the shape optimization considering the entire training dataset can uselessly complicate the optimization procedure. Instead, when we consider a volume that is not present in the training dataset (such as $\bar{V} = 9400$) we are trying to extrapolate information to new volumes. Hence, in this case, the shape optimization is performed considering all the shapes of the training dataset. The results of these experiments are reported in Table 6.

Table 6 shows that the shape optimization could be successfully run not only for $\bar{V} = 9000$ and $\bar{V} = 9600$, but also for $\bar{V} = 9400$. In this context, it is particularly interesting to notice that

Table 6
Values of the compliance and of the volume for the considered shape optimization problem with different choices of $\bar{V}$. The neural network is trained on a dataset containing shapes of volumes $V = 9000$ and $V = 9600$.

$\bar{V}$	Initial compliance	Final volume	Optimized compliance	IGA solution	Relative error
9600	510.15	9599.39	466.64	466.57	0.015%
9400	518.34	9399.99	493.83	493.36	0.095%
9000	577.42	8685.07	551.23	551.11	0.022%

- in previous experiments (see Table 5) we were not able to successfully solve the shape optimization problem with $\bar{V} = 9000$ without using transfer learning;
- no shape of volume $V = 9400$ was here used in the training dataset.

Hence, datasets made of shapes of different volumes can produce more general and robust surrogate models of the PDE solution on different domains. Furthermore, they can help increase the accuracy of the predictions when the volume constraint is changed. The main downside is that the training needs here to be performed on larger datasets, which is computationally onerous. Therefore, unless a very general surrogate model is needed, partial re-training by transfer-learning can be an efficient alternative, as earlier shown in Table 5.

5. Conclusions

We have presented a framework for physics-informed deep learning over freeform domains. In our framework, shapes are represented via NURBS and are parameterized to a common parametric domain. The shape parameters can be the actual coordinates of the control points or parameters with more intuitive geometric meaning, such as the major and minor radii of an ellipse. NNs and PINNs are used to predict PDE solutions over the parametric domain. Learning is conducted via minimizing a loss function where data error and PDE residuals are computed at collocation points in the parametric domain. For training PINNs, which requires the spatial derivative of physical quantities in enforcing physical equilibrium equations, the Jacobian of the geometric mapping is used to assist the evaluation of the spatial derivative of the PDE solutions.

We have formulated PINN and NN networks in the framework of linear elasticity and we have performed the training using data computed by IGA. We have analyzed the convergence of the training and we have shown that the trained neural networks can predict displacements and stresses in shapes that were not included in the training dataset. In this context, we have also shown that the mean square residual of the elasticity PDE using PINN's predictions is roughly one order of magnitude smaller than when we use a traditional NN without physics information. This has effects on the efficiency of the two networks. In particular, we found that, for a plate-hole problem, NN trained using 100 geometries produces a PDE residual that is roughly as accurate as PINN's trained on 20 geometries.

Moreover, we have also considered the effect of the accuracy of the training data. In particular, we have trained the network using IGA data obtained using different choices of mesh continuity and degree of the basis functions. We have noticed that the degree p of the basis functions can have a significant effect on the accuracy, especially when it is small. Indeed, we have noticed that, in our problems, the data and residual accuracy of PINN improves by roughly one order of magnitude if we use cubic data in place of linear data. The effect of continuity is smaller, but is noticeable on the training dataset.

Finally, we have noticed that PINN predictions of the PDE solution are more accurate than NN when little training data is available. In particular, PINN can make more accurate predictions than NN on the training shapes when the solution is known in just a small subset of the collocation points. This behavior can be attributed to the PDE residual term in the PINN loss function. We have observed that the increased accuracy of PINN at training tends to propagate to predictions made on nearby shapes. The effects of transfer learning and the applications of the shape-parameterized neural networks to shape optimization were also briefly discussed.

Our study suggests that our NURBS based framework can be used to train NNs and PINNs to build surrogate models for shape-parameterized PDE solutions. Our formulation for shape-parameterized NNs and PINNs can be further developed in several directions. First, they can be applied to a variety of physical problems beyond linear elasticity. Second, the resulting NNs and PINNs can be used in shape-dependent applications such as shape optimization, where PDE-based sensitivity analysis [31] can be replaced with the gradient from neural networks through automatic differentiation. Furthermore, the results on transfer learning could be expanded to consider how and whether significantly different shapes can accelerate the training of one another. This could allow to devise strategies to warm-start the training of PINN in general. Investigating examples that are challenging for the NURBS representation (such as shapes with sharp corners or high degree of curvature) is an interesting future work as well. Finally, although our numerical implementation uses IGA data, FEA data can also be used for training when domain parameterization is constructed and each FE mesh node is mapped to a parametric point.

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

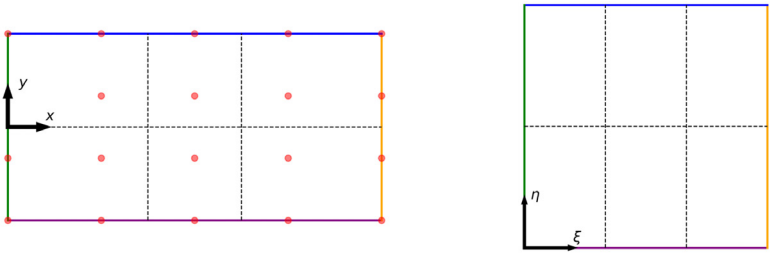
Data will be made available on request.

Acknowledgments

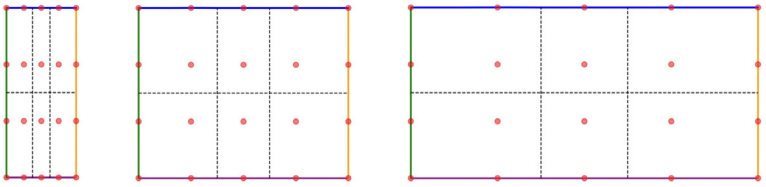
This work is supported in part by ONR, USA grant N00014-18-1-2685 by National Science Foundation, USA grant 1941206 and by National Science Foundation, USA grant 2219931. J. Gasick would like to acknowledge the support by the National Science Foundation Graduate Research Fellowship Program, USA under Grant No. DGE-1747503. F. Mezzadri would like to acknowledge the support of Maria Cristina Murari, who made available the computational resources to train the neural networks via the Scientific Computing Unit of the University of Modena and Reggio Emilia. Finally, all authors would like to thank the Reviewers for their comments, which improved the quality of the paper.

Appendix. Examples of NURBS discretizations

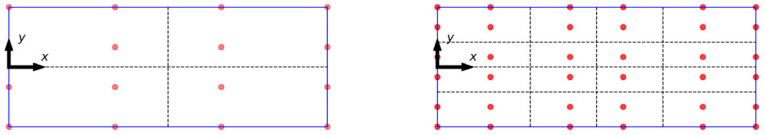
Fig. 26 displays example NURBS discretizations of each geometry shown in Fig. 6. In Fig. 26, p, q are the degrees of the NURBS basis functions in the parametric coordinate directions ξ and η.



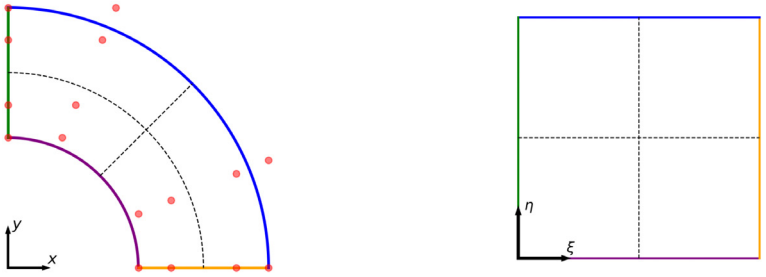
(a) Example of the relationship between the physical and the parametric domains for the rectangular beam problem, with $p = q = 2$, $k_\xi = \{0, 0, 0, 1/3, 2/3, 1, 1, 1\}$, $k_\eta = \{0, 0, 0, 1/2, 1, 1, 1\}$, $N_{p,\xi} = 5$, $N_{p,\eta} = 4$.



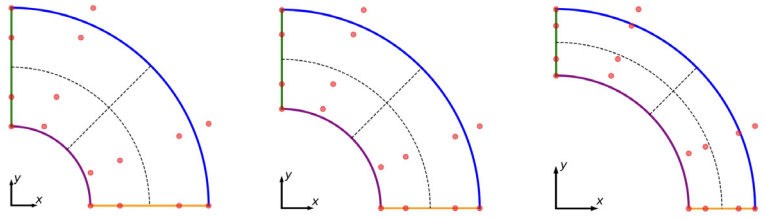
(b) Position of the control points for different lengths of the rectangular beam.



(c) Example of mesh refinement, $p = q = 2$. Left: $k_\xi = k_\eta = \{0, 0, 0, 1/2, 1, 1, 1\}$, $N_{p,\xi} = N_{p,\eta} = 4$. Right: refined mesh, $k_\xi = k_\eta = \{0, 0, 0, 1/4, 1/2, 3/4, 1, 1, 1\}$, $N_{p,\xi} = N_{p,\eta} = 6$.



(d) Example of the relationship between the physical and the parametric domains for the circular beam problem, with $p = q = 2$, $k_\xi = k_\eta = \{0, 0, 0, 1/2, 1, 1, 1\}$, $N_{p,\xi} = N_{p,\eta} = 4$.



(e) Position of the control points for different inner radii of the circular beam.

Fig. 26. NURBS parameterization of the rectangular and circular beam problems.

The knot vectors in the ξ and η parametric directions are referred to as k_ξ and k_η, respectively. The number of control points in the ξ and η directions are $N_{p,\xi}$ and $N_{p,\eta}$, respectively. Figs. 26(b) and 26(e) demonstrate how the position of the control points changes when the length of the beam or the radius of the inner circle are changed. Although more than one control point changes position when L or a is varied, all changes to control point positions can be uniquely specified by these geometric shape parameters. As a result, L and a can be considered as the shape parameters for these problems.

References

[1] Bergstra J, Bardenet R, Bengio Y, Kégl B. Algorithms for hyper-parameter optimization. In: Advances in neural information processing systems. 2011, p. 2546–54.

[2] Yang L, Shami A. On hyperparameter optimization of machine learning algorithms: Theory and practice. Neurocomputing 2020;415:295–316.

[3] Franchini G, Ruggiero V, Porta F, Zanni L. Neural architecture search via standard machine learning methodologies. Math Eng 2023;5(1):1–21.

[4] Raissi M, Perdikaris P, Karniadakis GE. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. J Comput Phys 2019;378:686–707.

[5] Wang S, Yu X, Perdikaris P. When and why PINNs fail to train: A neural tangent kernel perspective. J Comput Phys 2022;449(110768):1–28.

[6] Jagtap AD, Kharazmi E, Karniadakis GE. Conservative physics-informed neural networks on discrete domains for conservation laws: Applications to forward and inverse problems. Comput Methods Appl Mech Engrg 2020;365(113028):1–27.

[7] Meng X, Li Z, Zhang D, Karniadakis GE. PPINN: Parareal physics-informed neural network for time-dependent PDEs. Comput Methods Appl Mech Engrg 2020;370(113250):1–16.

[8] Pang G, Lu L, Karniadakis GE. fPINNs: Fractional physics-informed neural networks. SIAM J Sci Comp 2019;41(4):2603–26.

[9] Jin X, Cai S, Li H, Karniadakis GE. NSFnets (Navier-Stokes flow nets): Physics-informed neural networks for the incompressible Navier-Stokes equations. J Comput Phys 2021;426(109951):1–26.

[10] Li L, Li Y, Du Q, Liu T, Xie Y. ReF-nets: Physics-informed neural network for Reynolds equation of gas bearing. Comput Methods Appl Mech Engrg 2022;391(114524):1–26.

[11] Mao Z, Jagtap AD, Karniadakis GE. Physics-informed neural networks for high-speed flows. Comput Methods Appl Mech Engrg 2020;360(112789):1–26.

[12] Peng W, Pu J, Chen Y. PINN deep learning method for the Chen-Lee-Liu equation: Rogue wave on the periodic background. Commun Nonlinear Sci Numer Simul 2022;105(106067):1–15.

[13] Haghighat E, Raissi M, Moure A, Gomez H, Juanes R. A physics-informed deep learning framework for inversion and surrogate modeling in solid mechanics. Comput Methods Appl Mech Engrg 2021;379:113741.

[14] Wang Y, Liao Z, Shi S, Wang Z, Poh LH. Data-driven structural design optimization for petal-shaped auxetics using isogeometric analysis. CMES Comp Model Eng Sci 2020;122(2):433–58.

[15] Fuchi KW, Wolf EM, Makhija DS, Wukie NA, Schrock CR, Beran PS. Investigation of analysis and gradient-based design optimization using neural networks. In: SME 2020 conference on smart materials, adaptive structures and intelligent systems. 2020, http://dx.doi.org/10.1115/SMASIS2020-2241.

[16] Lu L, Meng X, Mao Z, Karniadakis GE. DeepXDE: A deep learning library for solving differential equations. SIAM Rev 2021;63(1):208–28.

[17] Piegl L, Tiller W. The NURBS book. New York: Springer-Verlag; 1997.

[18] Hughes TJR, Cottrell JA, Bazilevs Y. Isogeometric analysis: CAD, finite elements, NURBS, exact geometry and mesh refinement. Comput Methods Appl Mech Engrg 2005;194:4135–95.

[19] Hornik K, Stinchcombe M, White H. Multilayer feedforward networks are universal approximators. Neural Netw 1989;2(5):359–66.

[20] Gurney K. An introduction to neural networks. Philadelphia, PA, USA: Taylor & Francis, Inc.; 1997.

[21] Baydin AG, Pearlmutter BA, Radul AA, Siskind JM. Automatic differentiation in machine learning: A survey. J Mach Learn Res 2018;18:1–43.

[22] Wang S, Teng Y, Perdikaris P. Understanding and mitigating gradient flow pathologies in physics-informed neural networks. SIAM J Sci Comput 2021;43(5):A3055–81.

[23] Haghighat E, Amini D, Juanes R. Physics-informed neural network simulation of multiphase poroelasticity using stress-split sequential training. Comput Methods Appl Mech Engrg 2022;397:115141.

[24] Chen Z, Badrinarayanan V, Lee C-Y, Rabinovich A. Gradnorm: Gradient normalization for adaptive loss balancing in deep multitask networks. In: International conference on machine learning. PMLR; 2018, p. 794–803.

[25] Abadi M, Agarwal A, Barham P, Brevdo E, et al. TensorFlow: Large-scale machine learning on heterogeneous systems. 2015, Software available from tensorflow.org. URL https://www.tensorflow.org/.

[26] Kingma DP, Ba J. Adam: A method for stochastic optimization. In: 3rd international conference on learning representations. 2014.

[27] Haghighat E, Juanes R. SciANN: A keras/TensorFlow wrapper for scientific computations and physics-informed deep learning using artificial neural networks. Comput Methods Appl Mech Engrg 2021;373:113552.

[28] Zienkiewicz OC, Taylor RL, Zhu JZ. The finite element method: Its basis and fundamentals. 7th ed.. Butterworth-Heinemann; 2013.

[29] Timoshenko S, Goodier JN. Theory of elasticity. McGraw-Hill; 1970.

[30] Chen M, Lupoiu R, Mao C, Huang D-H, Jiang J, Lalanne P, et al. WaveY-Net: Physics-augmented deep learning for high-speed electromagnetic simulation and optimization. 2022, http://dx.doi.org/10.48550/ARXIV.2203.01248, URL https://arxiv.org/abs/2203.01248.

[31] Qian X. Full analytical sensitivities in NURBS based isogeometric shape optimization. Comput Methods Appl Mech Engrg 2010;199:2059–71.