Contents

Parametric deep energy approach for elasticity accounting for strain gradient effects

Vien Minh Nguyen-Thanh[a], Cosmin Anitescu[b], Naif Alajlan[c], Timon Rabczuk[c,**], Xiaoying Zhuang[a,*]

[a] *Chair of Computational Science and Simulation Technology, Department of Mathematics and Physics, Leibniz Universität Hannover, Germany.*
[b] *Institute of Structural Mechanics, Bauhaus-Universität Weimar, 99423, Weimar, Germany.*
[c] *Department of Computer Engineering, College of Computer and Information Sciences, King Saud University, Riyadh, Saudi Arabia.*

Abstract

In this work, we present a Parametric Deep Energy Method (P-DEM) for elasticity problems accounting for strain gradient effects. The approach is based on physics-informed neural networks (PINNs) for the solution of the underlying potential energy. Therefore, a cost function related to the potential energy is subsequently minimized. P-DEM does not need any classical discretization and requires only a definition of the potential energy, which simplifies the implementation. Instead of training the model in the physical space, we define a parametric/reference space similar to isoparametric finite elements, which is in our example a unit square. The inputs are naturally normalized preventing the vanishing gradient problem and leading to much faster convergence compared to the original DEM. Forward-backward mapping is established by means of NURBS basis functions. Another advantage of this approach is that Gauss quadrature can be employed to approximate the total potential energy, which is the loss function calculated in the parametric domain. Backpropagation available in PyTorch with automatic differentiation is performed to calculate the gradients of the loss function with respect to the weights and biases. Once the network is trained, a numerical solution can be obtained in the reference domain and then is mapped back to the physical domain. The performance of the method is demonstrated through various numerical benchmark problems in elasticity and compared to analytical solutions. We also consider strain gradient elasticity, which poses challenges to conventional finite elements due to the requirement for C^1 continuity.

Keywords: Neural Networks (NN), Physics-informed Neural Networks (PINNs), Partial Differential Equations (PDEs), Deep Energy Method, elasticity, strain gradient elasticity

1. Introduction

Many problems in physics and engineering can be described through partial differential equations (PDEs). Due to the high complexity of such problems, analytical solutions can rarely be obtained and numerical methods are usually employed to find approximate solutions. Popular numerical methods include the Finite Element Method (FEM) [1, 2, 3], meshfree methods [4, 5, 6] and Isogeometric Analysis [7, 8, 9], among many others. These methods are based on the weak form and 'transform' the underlying boundary value problem (BVP) into a linear (or linearized) system of equations, which can be solved by 'standard' solvers. The mathematical properties of the numerical method guarantee the convergence of the approximate solution. Although the aforementioned methods have been successfully applied to numerous complex problems, several challenges remain for certain problems, e.g. problems with high dimensionality, coupled problems, or ill-posed

*Corresponding author: zhuang@iop.uni-hannover.de
**Corresponding author: timon.rabczuk@uni-weimar.de
Email addresses: minh.nguyen@iop.uni-hannover.de (Vien Minh Nguyen-Thanh), cosmin.anitescu@uni-weimar.de (Cosmin Anitescu), timon.rabczuk@uni-weimar.de (Timon Rabczuk), zhuang@iop.uni-hannover.de (Xiaoying Zhuang)

problems, to name a few.

Recently, the machine learning-based solution of PDE has become a hot topic and attracted lots of attention [10, 11, 12, 13, 14, 15, 16, 17, 18, 19]; it has nevertheless already started a long time ago. In 1998, Lagaris et al. [20] proposed an interesting alternative approach based on machine learning to solve PDEs, which did not gain much attention until today, probably due to developments such as automatic differentiation, deep machine learning algorithms, and advances in GPU technology. The key idea is to minimize a loss function, which is the residuum in the domain and on the boundaries of the underlying PDEs at specific collocation points, with respect to weights and biases. That process is referred to as training a network. Once the optimal biases and weights of the neural network have been obtained, the solution of the problem is readily established. Based on this idea, Raissi et al. [21, 17, 16] introduced the Physics-informed neural networks (PINNs) to find the solutions of linear and nonlinear PDEs. Sirignano & Spiliopoulos [22] extended the method to problems with high dimensionality. Berg & Nyström [23] applied PINNs to problems with complex geometries. Guo et al. [24] applied the method to Kirchhoff plates exploiting the higher-order continuity of a neural network (NN). For second-order BVPs, Anitescu et al. [25] developed an adaptive collocation approach; they solved both forward and inverse problems with the same framework. Recently, Han Gao et al. [26] have developed the physics-informed geometry-adaptive for solving PDEs on an irregular domain by using the rasterization technique so that a convolutional neural network (CNN) can be exploited on that grid. While all the above approaches are based on the strong form, Weinan & Yu [27] proposed a method to approximate trial functions utilizing deep networks to solve the weak formulation of Poisson's equation.

In computational mechanics, Wessels et al. [28] developed an updated Lagrangian method for the solution of the incompressible free surface flow subject to the inviscid Euler equations, and named it as the Neural Particle Method. Samaniego et al. [29] presented a Deep Energy Method (DEM) with applications to elasticity, plates, phase-field models for fractures, and piezoelectricity. Instead of relating the cost function to the underlying BVP, it is expressed in terms of the potential energy of the engineering/mechanics problems, which reduces the requirements on the differentiability of the basis functions and automatically fulfills zero von Neumann boundary conditions. This approach is particularly relevant for engineering problems that are directly based on energy and has the advantage that it automatically fulfills underlying physical laws. Nguyen-Thanh et al. [30] subsequently applied the DEM to hyperelasticity. On the downside, this approach introduces integration errors and not all governing equations can be cast in an energy-minimization framework.

In the line of work in [30, 29], we propose a Parametric Deep Energy Method (P-DEM) and prove that it is capable of solving problems in elasticity and strain gradient elasticity. The latter would be difficult for conventional FEM due to the required C^1 continuity. Instead of training the network in the physical domain, we train the neural network in a reference domain, so the inputs are naturally normalized preventing the vanishing gradient problem. Unlike the work of Han Gao et al. [26], where the authors employed the elliptic coordinate transformation, in our work the mapping from a physical domain to the reference domain is accomplished with the NURBS basis function commonly used in IGA. This provides simultaneously the quadrature points to approximate the total potential energy. Since the loss function contains functionals, the accuracy of the solution relies on methods that numerically evaluate the integrals. Backpropagation is used for computing the gradients of the loss function [31] while optimizers such as Adam [32] or L-BFGS [33] yield solutions close to the global minimum. With the optimal network parameters, the solutions are predicted by our trained network. The robustness of the proposed method is demonstrated through various elasticity problems with different geometries. Furthermore, a problem characterized by a third-order PDE presenting strain gradient effects in micro materials is studied.

The outline of the paper is as follows: Section 2 gives the mathematical formulation of the physical system. In Section 3, the neural network and the DEM are briefly discussed. Afterwards, training in the reference domain and P-DEM are explained. In Section 4, some numerical examples in elasticity with different geometries are examined. We also study a strain gradient elasticity benchmark problem to demonstrate the robustness and efficiency of the proposed method. Some concluding remarks are given in Section 5.

2. Mathematical formulation of a physical system

Without loss of generality, we consider a parametrized PDE expressed as

$$\begin{aligned} &\mathcal{L}_{\boldsymbol{X}}(\boldsymbol{u};\boldsymbol{\lambda}) = f(\boldsymbol{X}), \quad && \boldsymbol{X} \in \Omega_{\boldsymbol{X}}, \\ &\mathcal{B}_k(\boldsymbol{u}) = g_k(\boldsymbol{X}), && \boldsymbol{X} \in \partial\Omega^k_{\boldsymbol{X}}, \quad k = 1, 2, ..., n_b, \end{aligned} \tag{1}$$

in which $\mathcal{L}_{\boldsymbol{X}}(\cdot)$ is the differential operator defined on the physical domain, $\boldsymbol{u}$ is the primary variable, $\boldsymbol{\lambda}$ are the model parameters, $\mathcal{B}(\cdot)_k$ is the boundary condition of the k^{th} boundary where the prescribed condition is given by the function $g_k(\boldsymbol{X})$, and $f(\boldsymbol{X})$ is the load/source term. For an isothermal static system, the temporal variable and temperature field, which introduces another internal variable to describe the dissipation of material, are neglected for the sake of simplicity. The system of differential equations with the solution specified at boundaries is usually called a boundary value problem (BVP). This is also referred to be the strong form of the system. Instead of solving directly the PDE (1), the solution of the system can be attained by finding the stationary point if one can reformulate the system in form of the total potential energy taking advantage of the variational method. In engineering, this method is named the *principle of the minimum of total potential energy* or its generalization for transient considerations, the *Hamilton's principle of continuum.* It is also called the weak form of the system and the basis of e.g. the finite-element method.

Without loss of generality, let us assume that an energy form of the PDE in equation (1) exists and has the form

$$\Pi = U - W, \tag{2}$$

where U denotes the internal energy and W indicates the external work. Then, we will solve problem

$$\min_{\boldsymbol{u}\in\mathcal{H}} \Pi(\boldsymbol{u}), \tag{3}$$

where $\mathcal{H}$ is the space of admissible functions (trial functions), $\boldsymbol{u}$ is constrained by the boundary conditions and Π is the total potential energy of the system. To find the stationary value, we take the variation of Π yielding

$$\delta\Pi(\boldsymbol{u}) = \frac{d}{d\eta}\Pi(\boldsymbol{u} + \eta\delta\boldsymbol{u})\big|_{\eta=0} = 0, \tag{4}$$

which has to hold for any admissible $\delta\boldsymbol{u}$ satisfying the boundary conditions and the fields equations. It is also known as a test function. Therefore, the equivalence of the strong form and the weak form is ensured. Ultimately, the variation can be written as

$$\delta\Pi = \delta U - \delta W. \tag{5}$$

This variation can also be obtained by the *principle of virtual work.* Therein, virtual displacements or test functions fulfilling specific properties, i.e. arbitrary, infinitesimal, satisfying the boundary conditions and the conditions of the fields, are multiplied with the strong form and then integrated over the volume.

3. Deep Energy Method based on PINN and a new invariant

3.1. Neural networks

The objective of a neural network is to approximate a continuous function that maps an input to its corresponding output based on the universal approximation theorem [34]. A feedforward fully-connected neural network is constructed by connecting neurons between layers. The first layer (0^{th}) is the input layer, the last layer (L^{th}) is the output layer, and the layers in between are hidden layers (l^{th}). If a network has one hidden layer, it is called a shallow network. If a network has more than one hidden layer, it is often called a deep neural network. Neurons in each layer are fully connected to other neurons of neighboring layers. The number

of neurons and layers are the hyperparameters among others, that need to be tuned, which in turn depends on the specific application. The mapping from the input to the output is denoted as $\hat{z} : \mathbb{R}^n \to \mathbb{R}^m / \boldsymbol{X} \Rightarrow \hat{z}(\boldsymbol{X})$, where n and m define the dimension of the input and output of a network, respectively. An activation function denoted by $\sigma : \mathbb{R} \to \mathbb{R}$ is used to activate each neuron. It returns a value, which is operated by the activation function acting on a linear mapping of the output of the neurons in the previous layer through a weight matrix $\boldsymbol{w}$ and a bias vector $\boldsymbol{b}$. This can be written in index notation as

$$\hat{z}_k^l = \underbrace{\sigma\left(\sum_{j=1}^{n_{l-1}} w_{kj}^l \, \hat{z}_j^{l-1} + b_k^l\right)}_{\sigma(z_k^l)}, \qquad 0 < l \leq L, \tag{6}$$

or compact notation as

$$\hat{\boldsymbol{z}}^l = \underbrace{\boldsymbol{\sigma}\left(\boldsymbol{w}^l \cdot \hat{\boldsymbol{z}}^{l-1} + \boldsymbol{b}^l\right)}_{\boldsymbol{\sigma}(\boldsymbol{z}^l)}, \qquad 0 < l \leq L, \tag{7}$$

where j is the j^{th} neuron and n_{l-1} is the number of neurons at the $(l-1)^{th}$ layer. It is necessary to find the optimal network parameters (weights and biases) so that the outputs are the best approximation of the actual solutions. The residual between the prediction and target solution is represented by an error function, which is usually called the loss function or objective function, denoted by $\mathcal{L}(\hat{\boldsymbol{z}}^L; \boldsymbol{\theta})$ or $\mathcal{L}(\boldsymbol{\theta})$. The network parameters are obtained by simply minimizing the loss function

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}), \tag{8}$$

which in turn requires the gradient of the loss function with respect to the network parameters $\frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}} = \left[\frac{\partial \mathcal{L}}{\partial \boldsymbol{w}}; \frac{\partial \mathcal{L}}{\partial \boldsymbol{b}}\right]^T$. These partial derivatives can be efficiently computed by automatic symbolic differentiation integrated into machine-learning frameworks, e.g. PyTorch, or TensorFlow. It is based on a computational graph whose optimizer and compiler can run parallel on a CPU or GPU so that it handles the derivatives in an optimized way.

The learning process of a network can be summarized by the following steps [1]

1. Input $[\boldsymbol{X}] = [\boldsymbol{X}_1; ...; \boldsymbol{X}_n]^T$ where n is the number of space coordinates: set the corresponding $\hat{\boldsymbol{z}}^0([\boldsymbol{X}]) = [\boldsymbol{X}]$ for the input layer.
2. Initialize weights $\boldsymbol{w}^l$ and biases $\boldsymbol{b}^l$ for all layers.
3. Feedforward: For each $l := 1, 2, ..., L$ compute $\boldsymbol{z}^l = \boldsymbol{w}^l \cdot \hat{\boldsymbol{z}}^{l-1} + \boldsymbol{b}^l$ and $\hat{\boldsymbol{z}}^l = \boldsymbol{\sigma}(\boldsymbol{z}^l)$.
4. Backpropagation (see Figure 1):
   - Compute the gradient of loss function w.r.t the output $\boldsymbol{\delta}^L = \mathcal{L}' \cdot (\boldsymbol{\sigma}^L)'$ where $(\boldsymbol{\sigma}^L)' = \frac{\partial \boldsymbol{\sigma}(\boldsymbol{z}^L)}{\partial \boldsymbol{z}^L}$
   - For each $\alpha := l, l-1, ..., 1$ compute $\boldsymbol{\delta}^\alpha = \boldsymbol{\delta}^{\alpha+1} \cdot \ \boldsymbol{w}^{\alpha+1} \cdot (\boldsymbol{\sigma}^\alpha)'$.
5. Gradient: The gradient of loss function is given by $\frac{\partial \mathcal{L}}{\partial \boldsymbol{w}^\alpha} = \boldsymbol{\delta}^\alpha \cdot \hat{\boldsymbol{z}}^{\alpha-1}$ and $\frac{\partial \mathcal{L}}{\partial \boldsymbol{b}^\alpha} = \boldsymbol{\delta}^\alpha$.
6. Update new weights and biases based on an optimization method in every layer, e.g. in stochastic gradient descent (SGD)
   - For each $l := L, ..., 2, 1$: $\boldsymbol{w}_{new}^l = \boldsymbol{w}^l - \gamma \frac{\partial \mathcal{L}}{\partial \boldsymbol{w}^l}$ and $\boldsymbol{b}_{new}^l = \boldsymbol{b}^l - \gamma \frac{\partial \mathcal{L}}{\partial \boldsymbol{b}^l}$ where γ is the learning rate.

[1] A detailed discussion can be found in [35].

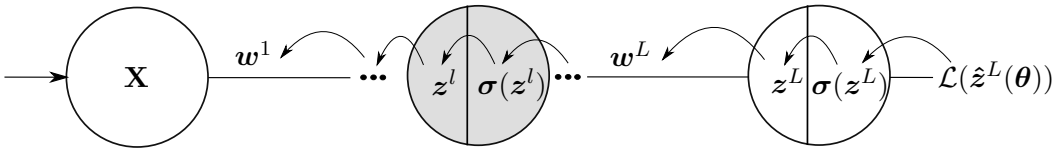


Figure 1: A schematic scheme of backpropagation.

One of the disadvantages of a neural network is the nonconvexity of the loss function if the network is constructed by connecting neurons whose values are evaluated by nonlinear activation functions. Therefore, the minimization turns into a non-convex optimization problem [36]. In most cases, we use a gradient-based method such as Adam optimizer, or a quasi-Newton method such as L-BFGS optimizer, to find the local minima [37].

3.2. Deep Energy Method (DEM)

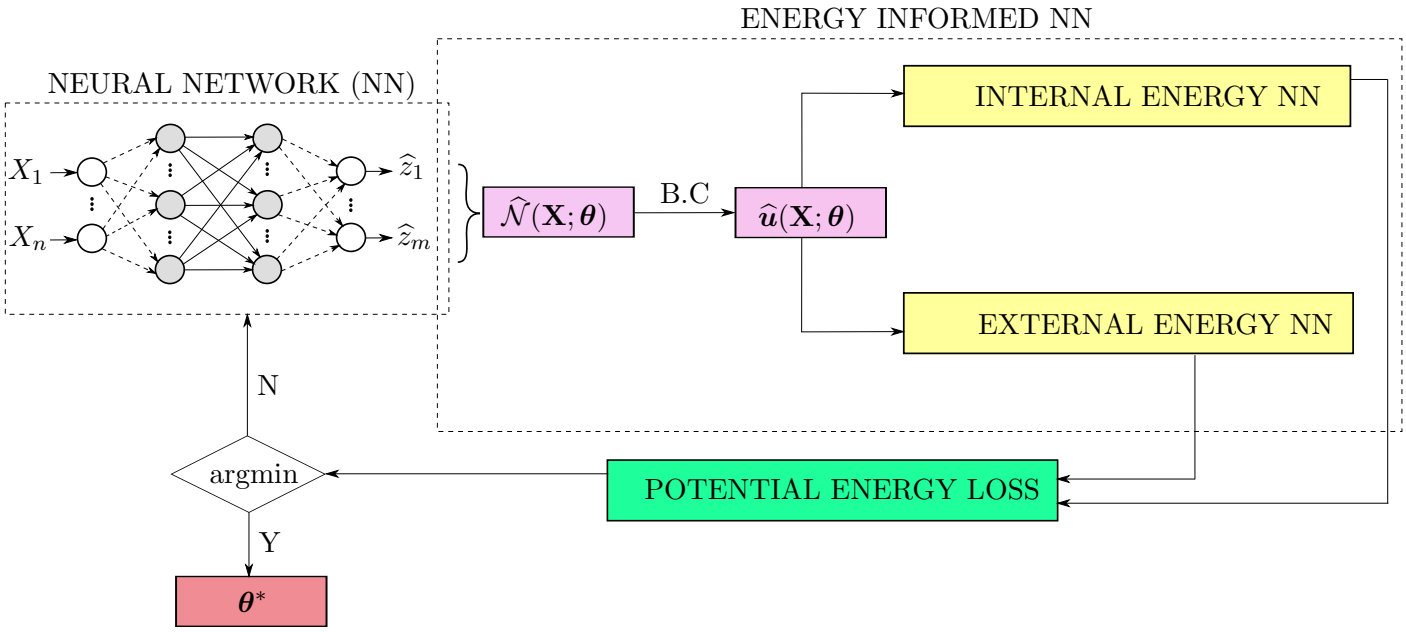


Figure 2: The schematic process of a Deep Energy Method.

The idea of the DEM is to minimize the potential energy (2) with the help of artificial neural networks as shown in Figure 2. Therefore, we need to cast the minimization into the optimization problem in the context of machine learning. As such, the definition of a loss function is essential, which is simply the potential energy of the system, i.e.

$$\underbrace{\mathcal{L}(\hat{\boldsymbol{u}};\boldsymbol{\theta})}_{\text{Potential energy loss}} = \underbrace{U(\hat{\boldsymbol{u}};\boldsymbol{\theta})}_{\text{Internal energy NN}} - \underbrace{W(\hat{\boldsymbol{u}};\boldsymbol{\theta})}_{\text{External energy NN}}, \tag{9}$$

in which the hat symbols indicate the solution yielded by the neural network. The trial function $\hat{\boldsymbol{u}}(\boldsymbol{X};\boldsymbol{\theta})$ or shortly $\hat{\boldsymbol{u}}(\boldsymbol{X})$ has to fulfill boundary conditions a-priori. A feedforward neural network constructed by network parameters $\boldsymbol{\theta}$ is employed to establish the trial solution. The displacement computed by the neural

network has a form so that it satisfies the boundary conditions. This can be written as

$$\hat{\boldsymbol{u}}(\boldsymbol{X}) = \boldsymbol{A}(\boldsymbol{X}) + \boldsymbol{B}(\boldsymbol{X}, \hat{\boldsymbol{u}}^L(\boldsymbol{X}; \boldsymbol{\theta})), \tag{10}$$

where $\hat{\boldsymbol{u}}^L(\boldsymbol{X}; \boldsymbol{\theta})$ is the unconstrained displacement (the output displacement of neural networks) with network parameters $\boldsymbol{\theta}$. The term $\boldsymbol{A}(\cdot)$ refers to a smooth extension of the boundary data and $\boldsymbol{B}(\cdot)$ is a smooth distance function. Both are chosen in such a way that $\hat{\boldsymbol{u}}(\boldsymbol{X})$ must satisfy the boundary conditions at certain points. A discussion on how to choose these terms can be found in the works of Lagaris [20] and Berg & Nyström [23]. By applying this procedure we can replace the original constrained optimization with an unconstrained optimization problem which is much easier to handle. The unconstrained optimization problem now is

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \ \mathcal{L}(\hat{\boldsymbol{u}}; \boldsymbol{\theta}), \tag{11}$$

which is solved with L-BFGS. The derivatives of the loss function $\mathcal{L}(\hat{\boldsymbol{u}}; \boldsymbol{\theta})$ with respect to the weights and biases are computed by backpropagation.

Let us summarize some key features of the proposed DEM:

- The minimization of the loss function is similar to the classical principle of minimum potential energy of finding the stationary points. It deals directly with the minimization of the energy Π, circumventing the need to derive the weak form from the energy of the system explicitly.
- The NN solutions $\hat{\boldsymbol{u}}$ ensure the loss $\mathcal{L}(\hat{\boldsymbol{u}}; \boldsymbol{\theta})$ fulfills the equilibrium state when the potential energy is minimized.
- The energy form has derivatives of a lower order than the strong form, and therefore less computational effort is required.
- Traction-free boundary conditions are automatically fulfilled.
- Compared to the finite-element approaches, the implementation is simpler because it only requires the definition of the potential energy and bypasses discretization and assembly steps.

3.3. Parametric Deep Energy Method (P-DEM)

The underlying problem and related integrals are formulated in the physical domain. We adopt an idea from isoparametric elements and convert the formulation from the physical domain to a parametric domain, that is – for our problems – a unit square with defined natural coordinates. Therefore, the potential strain energy of the system can be rewritten as

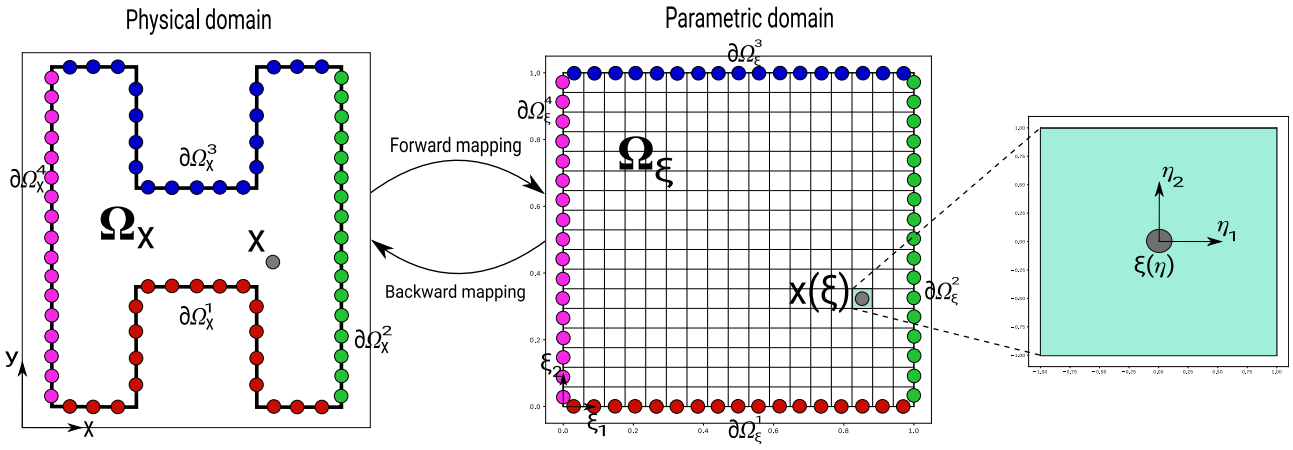


Figure 3: P-DEM - Parametric Deep Energy Method based on the Physics-informed Neural Networks. The physical domain which is the letter H is mapped to the reference domain $\Omega_{\boldsymbol{\xi}}$ (parametric domain). The formulation of total potential energy is minimized in the reference domain. The approximation of integrals in the potential energy of the systems is done by a numerical integration on $\Omega_{\boldsymbol{\xi}}$.

$$U = \int_{\Omega_{\boldsymbol{X}}} \Psi(\boldsymbol{X})\, dV_X = \int_{\Omega_{\boldsymbol{\xi}}} \Psi(\boldsymbol{\xi}) |\boldsymbol{J}_1|\, dV_{\boldsymbol{\xi}}, \tag{12}$$

in which the coordinate transformation is done with the determinant of Jacobian denoted by $|\boldsymbol{J}_1| = |\partial \boldsymbol{X}/\partial \boldsymbol{\xi}|$. The Jacobian matrix is defined as

$$\boldsymbol{J}_1 = \begin{bmatrix} \frac{\partial X_1}{\partial \xi_1} & \frac{\partial X_2}{\partial \xi_1} \\ \\ \frac{\partial X_1}{\partial \xi_2} & \frac{\partial X_2}{\partial \xi_2} \end{bmatrix}. \tag{13}$$

The components of the Jacobian matrix can be calculated by a finite difference approach or any numerical procedure. Due to the similar concept of the reference domain which is exactly the parametric domain used in IGA [38, 39, 40], we exploit the concept of IGA with the help of non-uniform rational B-splines (NURBS) as the basis functions to perform the interpolation so that the Jacobian can be computed easily. First, the knot vector denoted by $\Xi = \{\xi_1, \xi_2, ..., \xi_{n+p+1}\}$ is defined as a set of non-decreasing real numbers, where p is the polynomial degree of the basis and n is the number of the basis functions. Second, a univariate B-spline basis function can be defined using a recursive relation. We start with a basis of piecewise constant functions when the polynomial degree is 0:

$$N_{i,0}(\xi) = \begin{cases} 1, & \text{if} \quad \xi_i \leq \xi < \xi_{i+1} \\ 0, & \text{otherwise.} \end{cases} \tag{14}$$

If $\xi_i = \xi_{i+1}$, then $N_{i,0} = 0$. For $p > 0$,

$$N_{i,p}(\xi) = \frac{\xi - \xi_i}{\xi_{i+p} - \xi_i} N_{i,p-1}(\xi) + \frac{\xi_{i+p+1} - \xi}{\xi_{i+p+1} - \xi_{i+1}} N_{i+1,p-1}(\xi). \tag{15}$$

Third, the NURBS basis function in 1D is defined as follows

$$R_i^p(\xi) = \frac{N_{i,p}(\xi)\, \tilde{w}_i}{\sum_{\hat{i}}^{n} N_{\hat{i}p}(\xi)\, \tilde{w}_{\hat{i}}}, \tag{16}$$

where $\tilde{w}_i$ is the weight corresponding to the i^{th} B-spline function $N_{i,p}(\xi)$. The NURBS basis function in 2D for the parametric domain (ξ_1, ξ_2) and the associated polynomial degrees $(p,\, q)$ is now defined as

$$R^{p,q}_{i,j}(\xi_1,\xi_2) = \frac{N_{i,p}(\xi_1) M_{j,q}(\xi_2)\tilde{w}_{i,j}}{\sum_{\hat{i}=1}^{n}\sum_{\hat{j}=1}^{m} N_{\hat{i},p}(\xi_1) M_{\hat{j},q}(\xi_2)\tilde{w}_{\hat{i},\hat{j}}}. \tag{17}$$

Finally, the backward mapping can be performed by the interpolation as

$$\boldsymbol{X}(\xi_1,\xi_2) = \sum_{i=1}^{n}\sum_{j=1}^{m} R^{p,q}_{i,j}(\xi_1,\xi_2)\boldsymbol{B}_{i,j}, \tag{18}$$

where $\boldsymbol{B}_{i,j}$ are the corresponding control points defined in the physical space.

The calculation of the integrals of the strain energy in (13) plays the prominent contribution in yielding a good solution of the DEM. The internal energy U can be evaluated by Monte-Carlo integration in which the data points are randomly generated over the entire domain. However, due to the perfect shape of the reference domain, more precise integration schemes like the trapezoidal rule, Simpson rule, or Gaussian quadrature are better options. Here, we use the Gauss-Legendre quadrature to approximate the internal potential energy of the system. Therefore, the reference domain is discretized into smaller elements and the integral of the strain energy is calculated by a summation of smaller integrals in which each integral is approximated by the strain energy evaluated at each quadrature point. Finally, the potential strain energy in equation (13) is approximated by

$$U = \sum_{e=1}^{N} U^e = \sum_{e=1}^{N}\int_{\Omega^e_{\boldsymbol{\xi}}} \Psi^e(\boldsymbol{\xi})|\boldsymbol{J}^e_1| dV^e_{\boldsymbol{\xi}}, \tag{19}$$

where N is the number of elements, and

$$U^e = \int_{\Omega_{\boldsymbol{\eta}}} \Psi(\boldsymbol{\xi}(\boldsymbol{\eta}))|\boldsymbol{J}_1(\boldsymbol{\eta})||\boldsymbol{J}_2(\boldsymbol{\eta})| dV_{\boldsymbol{\eta}} \approx \sum_{i}^{n_x^{GP}}\sum_{j}^{n_y^{GP}} \Psi(\boldsymbol{\xi}(\eta_1^i,\eta_2^j))|\boldsymbol{J}_1(\eta_1^i,\eta_2^j)||\boldsymbol{J}_2(\eta_1^i,\eta_2^j)|\, \bar{w}^i\, \bar{w}^j, \tag{20}$$

in which n_x^{GP} and n_y^{GP} refer to the number of quadrature points being used to approximate the integral in x and y direction, respectively; $\bar{w}^i$ and $\bar{w}^j$ are the weights of corresponding Gauss points; and the determinant of the second Jacobian matrix is expressed by $|\boldsymbol{J}_2| = |\partial\boldsymbol{\xi}/\partial\boldsymbol{\eta}|$. The transformation from the physical domain to (parametric) reference domain, is performed by a change of coordinates, from physical coordinates $\boldsymbol{X}$ to natural coordinates $\boldsymbol{\xi}$. It can be obtained by introducing the Jacobian matrix and its inverse. The necessary derivatives with respect to physical coordinates can be expressed by the chain rule

$$\begin{bmatrix} \frac{\partial}{\partial X_1} \\ \\ \frac{\partial}{\partial X_2} \end{bmatrix} = \begin{bmatrix} \frac{\partial \xi_1}{\partial X_1} & \frac{\partial \xi_2}{\partial X_1} \\ \\ \frac{\partial \xi_1}{\partial X_2} & \frac{\partial \xi_2}{\partial X_2} \end{bmatrix} \begin{bmatrix} \frac{\partial}{\partial \xi_1} \\ \\ \frac{\partial}{\partial \xi_2} \end{bmatrix} \quad \text{or} \quad \frac{\partial}{\partial \boldsymbol{X}} = \boldsymbol{J}^{-1}(\boldsymbol{\xi})\frac{\partial}{\partial \boldsymbol{\xi}}. \tag{21}$$

By means of coefficient comparison and the definition of inverse of a matrix, we can get the identities

$$\begin{aligned} \frac{\partial \xi_1}{\partial X_1} &= \frac{1}{|\boldsymbol{J}(\boldsymbol{\xi})|}\frac{\partial X_2}{\partial \xi_2}, & \frac{\partial \xi_2}{\partial X_1} &= -\frac{1}{|\boldsymbol{J}(\boldsymbol{\xi})|}\frac{\partial X_2}{\partial \xi_1}, \\ \frac{\partial \xi_1}{\partial X_2} &= -\frac{1}{|\boldsymbol{J}(\boldsymbol{\xi})|}\frac{\partial X_1}{\partial \xi_2}, & \frac{\partial \xi_2}{\partial X_2} &= \frac{1}{|\boldsymbol{J}(\boldsymbol{\xi})|}\frac{\partial X_1}{\partial \xi_1}. \end{aligned} \tag{22}$$

One advantage of this approach is that the input values are naturally normalized due to the coordinates of training points only distributing across the unit domain. Therefore, the convergence of the loss function

tends to approach the optimal value faster than the conventional training on the physical domain. Another benefit of the approach is that the training on the reference domain can be exploited for the further training of different problems due to the same geometry on the reference domain (see Figure 4).

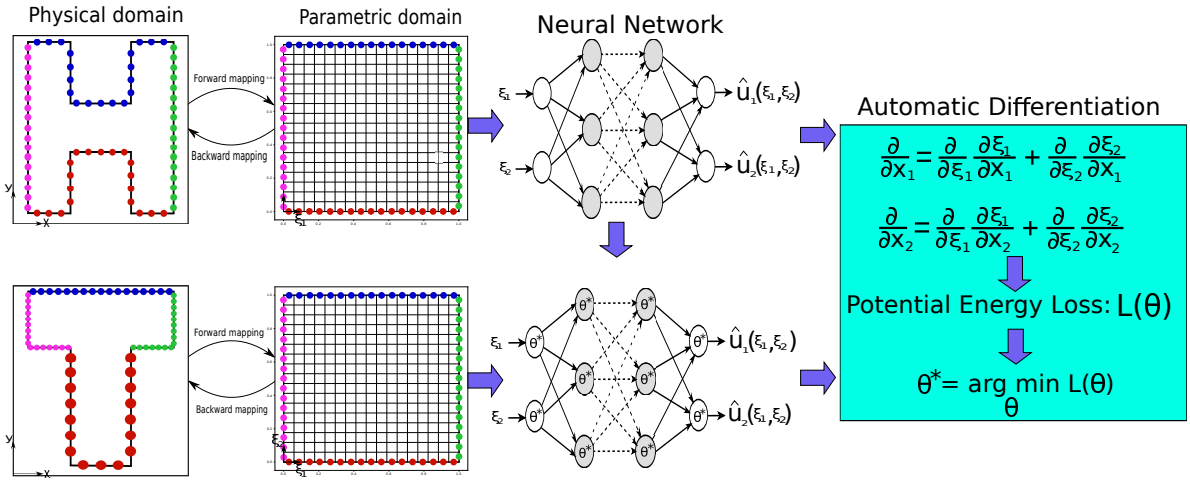


Figure 4: Various problems with different geometries can be solved by P-DEM. The formulation of potential energy loss is calculated by the automatic differentiation integrated in PyTorch.

Note that the concept of IGA here is only for the geometry representation and not directly used for the field approximation. The increase of the control points in IGA or choices of basis function shall not significantly affect the training process and hence not dominate the accuracy of P-DEM, though more accurate geometric representation may be realized by adding more Gauss points where the residual is large [41].

4. Applications to mechanical problems

4.1. Elasticity

Let us consider a body made of a homogeneous, isotropic, linearly elastic material. The domain is denoted by Ω and described by a set of particles or material points $\boldsymbol{X}$. Its boundary is indicated by $\partial\Omega$. The body is subject to prescribed displacements on the Dirichlet boundary $\partial\Omega_u$, and surface loads $\bar{\boldsymbol{t}}$ on the Neumann boundary $\partial\Omega_t$ (see Figure 5) with $\partial\Omega_u \bigcup \partial\Omega_t = \partial\Omega$ and $\partial\Omega_u \cap \partial\Omega_t = \emptyset$. The governing equations and associated boundary conditions in elastostatics are given as

$$\text{Equilibrium:} \quad \nabla \cdot \boldsymbol{\sigma}(\boldsymbol{X}) + \boldsymbol{f}_b(\boldsymbol{X}) = \boldsymbol{0} \quad \forall \boldsymbol{X} \in \Omega, \tag{23}$$

$$\text{Dirichlet boundary:} \quad \boldsymbol{u}(\boldsymbol{X}) = \bar{\boldsymbol{u}}(\boldsymbol{X}) \quad \forall \boldsymbol{X} \in \partial\Omega_u, \tag{24}$$

$$\text{Neumann boundary:} \quad \boldsymbol{\sigma}(\boldsymbol{X}) \cdot \boldsymbol{n} = \bar{\boldsymbol{t}}(\boldsymbol{X}) \quad \forall \boldsymbol{X} \in \partial\Omega_t, \tag{25}$$

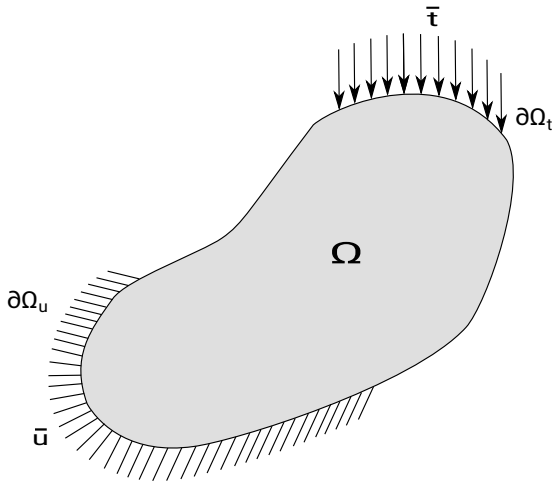


Figure 5: A body Ω and its boundaries.

where $\nabla \cdot \boldsymbol{\sigma}(\boldsymbol{X})$ indicates the divergence operator acting on the Cauchy stress tensor. The primal variables, i.e. the displacements, are denoted by $\boldsymbol{u}$. The outward normal unit vector is $\boldsymbol{n}$ and the body force is $\boldsymbol{f}_b$. The Cauchy stress tensor is related to the linear strain tensor through the constitutive equation (Hooke's law). The constitutive equation is derived from an elastic strain energy density. Indeed, by taking the derivative of the strain energy w.r.t the strain, we obtain the stress

$$\boldsymbol{\sigma} = \frac{\partial \Psi(\boldsymbol{\epsilon})}{\partial \boldsymbol{\epsilon}}, \tag{26}$$

where the strain energy density is denoted by $\Psi(\boldsymbol{\epsilon}) = \mu(\boldsymbol{\epsilon} : \boldsymbol{\epsilon}) + \frac{1}{2}\lambda\,(\boldsymbol{\epsilon} : \mathbf{1})^2$. Taking derivatives of the stress tensor w.r.t the strain yields the fourth order elasticity $\mathbb{C}$

$$\mathbb{C} = \frac{\partial \boldsymbol{\sigma}}{\partial \boldsymbol{\epsilon}} = \frac{\partial^2 \Psi(\boldsymbol{\epsilon})}{\partial \boldsymbol{\epsilon} \partial \boldsymbol{\epsilon}}. \tag{27}$$

which satisfies both major and minor symmetry properties, $C_{ijkl} = C_{jikl} = C_{jilk} = C_{ijlk}$; μ and λ are two Lamé's constants. According to equation (26), the stress tensor is obtained as

$$\boldsymbol{\sigma} = 2\,\mu\,\boldsymbol{\epsilon} + \lambda\,\mathrm{tr}(\boldsymbol{\epsilon})\mathbf{1}. \tag{28}$$

The strain-displacement relations are given by

$$\boldsymbol{\epsilon} = \frac{1}{2}\left(\nabla \boldsymbol{u} + \nabla \boldsymbol{u}^T\right). \tag{29}$$

The potential energy of the system is given by

$$\Pi = U - W = \int_\Omega \Psi(\boldsymbol{\epsilon})\, dV - \int_\Omega \boldsymbol{f}_b \cdot \boldsymbol{u}\, dV - \int_{\partial\Omega_t} \bar{\boldsymbol{t}} \cdot \boldsymbol{u}\, dA, \tag{30}$$

where the strain energy of the body refers to the first term and the external energy is determined by the remaining terms

$$U = \int_\Omega \Psi(\boldsymbol{\epsilon})\, dV, \tag{31}$$

$$W = \int_\Omega \boldsymbol{f}_b \cdot \boldsymbol{u}\, dV + \int_{\partial\Omega_t} \bar{\boldsymbol{t}} \cdot \boldsymbol{u}\, dA. \tag{32}$$

In the spirit of the DEM, we employ the total potential energy as the loss function yielding

$$\mathcal{L}(\hat{\boldsymbol{u}};\boldsymbol{\theta}) = U(\hat{\boldsymbol{u}};\boldsymbol{\theta}) - W(\hat{\boldsymbol{u}};\boldsymbol{\theta}). \tag{33}$$

We adopt the feed-forward neural network to build a surrogate model without the labeled data to perform the optimization with regard to the defined loss function so that the optimal neural network parameters can be obtained. The idea of DEM integrated with the parametric training can now be applied to the elasticity problems. The components of strain can be expressed as follows based on strain-displacement relation and the equation (22)

$$\begin{aligned}
\epsilon_{11} &= \frac{\partial u_1}{\partial X_1} = \frac{\partial u_1}{\partial \xi_1}\frac{\partial \xi_1}{\partial X_1} + \frac{\partial u_1}{\partial \xi_2}\frac{\partial \xi_2}{\partial X_1} = \quad \frac{1}{|\boldsymbol{J}(\boldsymbol{\xi})|}\left(\frac{\partial u_1}{\partial \xi_1}\frac{\partial X_2}{\partial \xi_2} - \frac{\partial u_1}{\partial \xi_2}\frac{\partial X_2}{\partial \xi_1}\right) \\
\epsilon_{22} &= \frac{\partial u_2}{\partial X_2} = \frac{\partial u_2}{\partial \xi_1}\frac{\partial \xi_1}{\partial X_2} + \frac{\partial u_2}{\partial \xi_2}\frac{\partial \xi_2}{\partial X_2} = -\frac{1}{|\boldsymbol{J}(\boldsymbol{\xi})|}\left(\frac{\partial u_2}{\partial \xi_1}\frac{\partial X_1}{\partial \xi_2} - \frac{\partial u_2}{\partial \xi_2}\frac{\partial X_1}{\partial \xi_1}\right) \\
\epsilon_{21} = \epsilon_{12} &= \frac{1}{2}\left(\frac{\partial u_1}{\partial X_2} + \frac{\partial u_2}{\partial X_1}\right) = \frac{1}{2}\left(\frac{\partial u_1}{\partial \xi_1}\frac{\partial \xi_1}{\partial X_2} + \frac{\partial u_1}{\partial \xi_2}\frac{\partial \xi_2}{\partial X_2} + \frac{\partial u_2}{\partial \xi_1}\frac{\partial \xi_1}{\partial X_1} + \frac{\partial u_2}{\partial \xi_2}\frac{\partial \xi_2}{\partial X_1}\right) \\
&= -\frac{1}{2}\frac{1}{|\boldsymbol{J}(\boldsymbol{\xi})|}\left(\frac{\partial u_1}{\partial \xi_1}\frac{\partial X_1}{\partial \xi_2} - \frac{\partial u_1}{\partial \xi_2}\frac{\partial X_1}{\partial \xi_1} - \frac{\partial u_2}{\partial \xi_1}\frac{\partial X_2}{\partial \xi_2} + \frac{\partial u_2}{\partial \xi_2}\frac{\partial X_2}{\partial \xi_1}\right).
\end{aligned} \tag{34}$$

4.1.1. Timoshenko beam

Let us consider a beam of length $L = 8\text{m}$, height $D = 2\text{m}$, subject to a parabolic loading $P = 2\text{N}$ at the right edge. The Dirichlet boundary condition is prescribed at the left edge as shown in Figure 6.

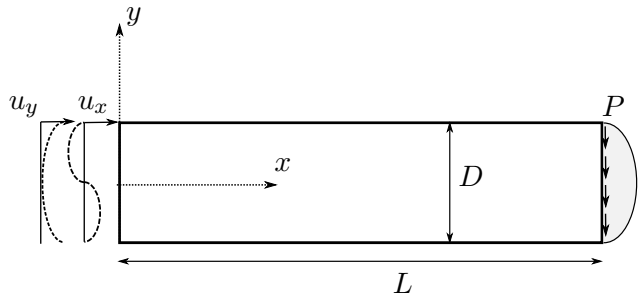


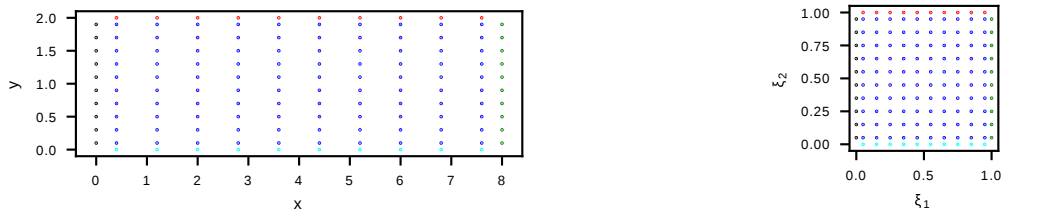
Figure 6: Timoshenko beam: geometry and boundary conditions

The beam is considered in plane stress condition with parameters $E = 10^3\text{N/m}^2$, $\nu = 0.25$. The analytical

stress and displacement solutions given by Timoshenko and Goodier [42] are expressed as

$$
\begin{aligned}
\sigma_{xx} &= \frac{P(L-x)y}{I}, \\
\sigma_{yy} &= 0, \\
\tau_{xy} &= -\frac{P}{2I}\left(\frac{D^2}{4} - y^2\right), \\
u_x &= \frac{Py}{6EI}\left[(6L-3x)x + (2+\nu)\left(y^2 - \frac{D^2}{4}\right)\right], \\
u_y &= -\frac{P}{6EI}\left[3\nu y^2(L-x) + (4+5\nu)\frac{D^2 x}{4} + (3L-x)x^2\right],
\end{aligned}
$$

where $I = D^3/12$ is the moment of inertia (unit thickness is assumed).



(a) Data in physical domain

(b) Data in reference domain

Figure 7: Data distribution in physical domain and reference domain.

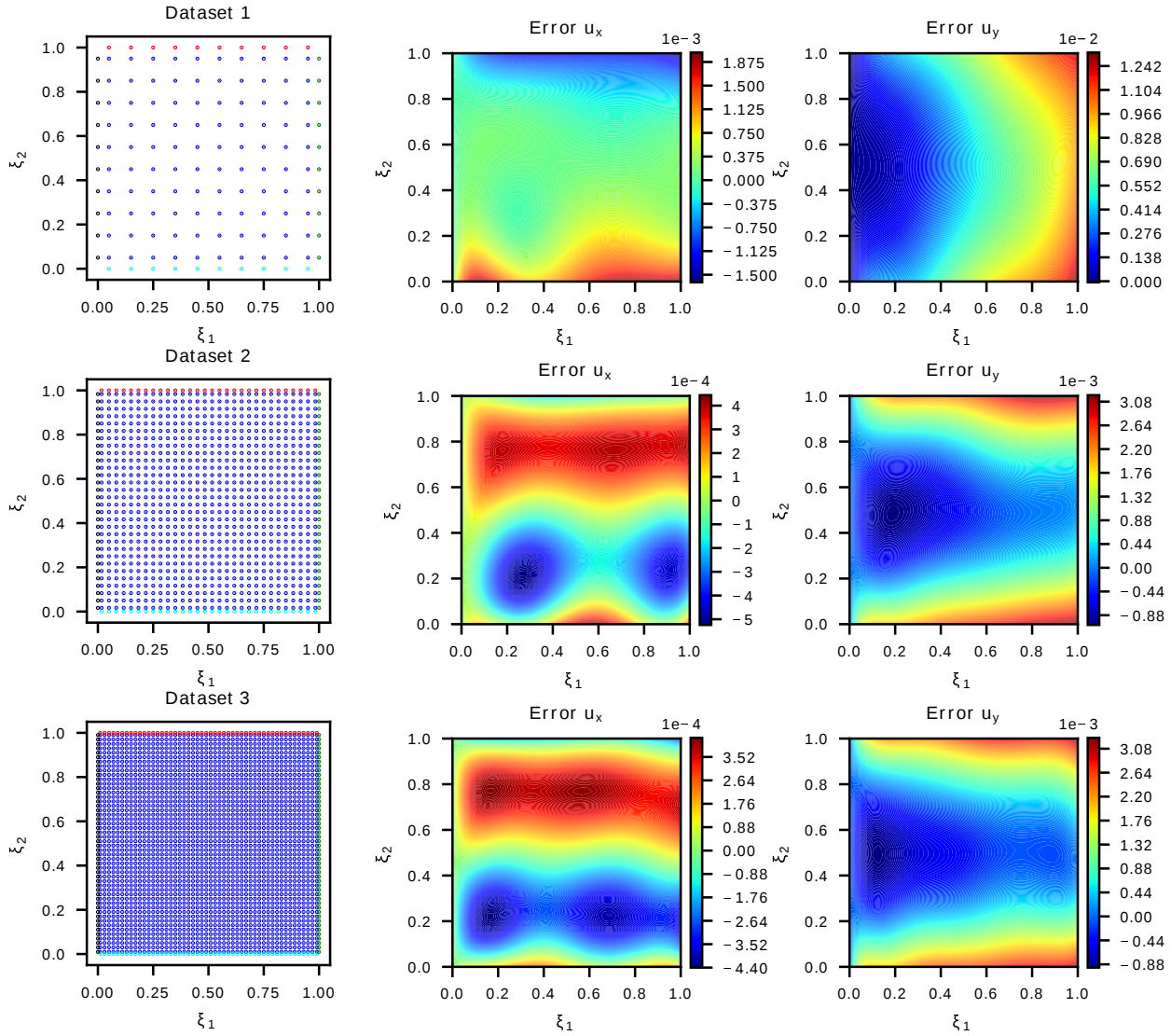


Figure 8: The displacement error in x and y direction with respect to the three types of data grid (using *tanh* activation function).

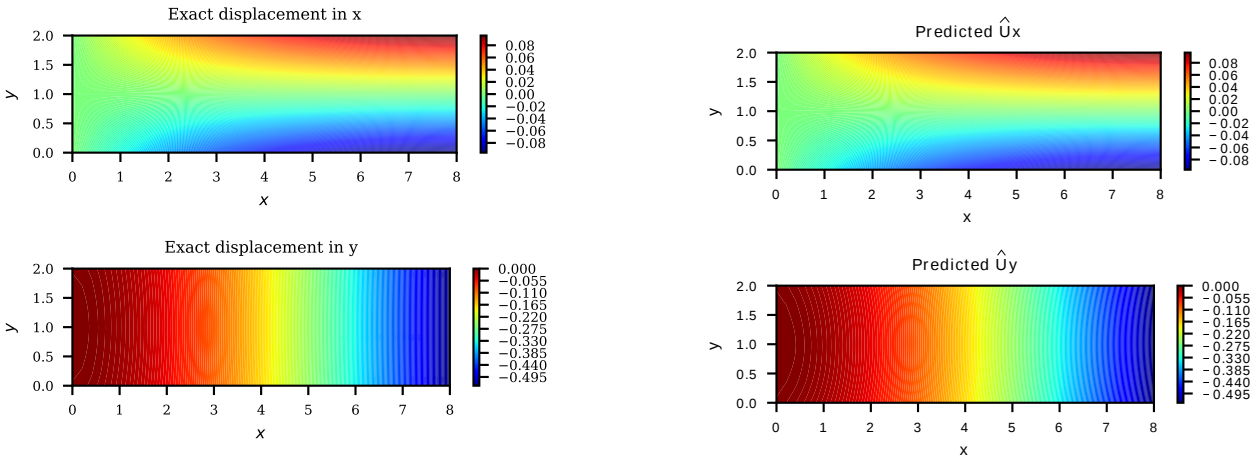


Figure 9: The exact solutions and neural network solutions using *tanh* activation function and Dataset 1.

L^2 **enorm**	**Energy enorm**	**Dataset**	**Training time**	**Epochs**	**Activation**
0.021061	0.100201	10 x 10	42.1673	60	sigmoid
0.004063	0.097791	30 x 30	105.3095	100	sigmoid
0.003526	0.089176	50 x 50	262.5227	120	sigmoid
0.023661	0.094829	10 x 10	43.2374	60	tanh
0.004261	0.080955	30 x 30	110.2322	100	tanh
0.004257	0.076556	50 x 50	272.9136	120	tanh
0.021296	0.098104	10 x 10	44.3976	60	CELU
0.003601	0.094199	30 x 30	103.6346	100	CELU
Nan	Nan	50 x 50	288.4305	120	CELU

Table 1: L^2 error norm and H^1 error seminorm comparison in 3 cases: grid 10x10, grid 30x30, grid 50x50. Also, the results with different activation functions are compared.

The number of data, neurons, hidden layers, and other hyperparameters need to be carefully chosen and the procedure involves some manual search. Although there is no rule of thumb to choose such neural network structures, the selection of the width and depth of a network and other hyperparameters can be done in the spirit of optimization. In fact, these hyperparameters can be tuned automatically by Bayesian optimization [43], genetic algorithms [44], artificial neural networks [45], random search [46], or grid search [47]. However, this is out of scope of our work. In this example, we will study the influence of some hyperparameters. We generate three datasets: (1) 10×10 equidistant grid points for the internal domain and 10 points along each edge for the boundary, (2) 30×30 equidistant grid points for the internal domain, and 30 points along each edge for the boundary, (3) 50×50 equidistant grid points for the internal domain and 50 points along each edge for the boundary. The optimizer L-BFGS with a learning rate 0.1 is used for the training. Because the number of data used for training is different, different epoch numbers are set for the corresponding dataset, i.e. 60, 100, and 120. In all testcases, we use the same network structure: 2 - 30 - 30 - 30 - 2 but different activation functions, i.e. *sigmoid*, *tanh*, and *CELU*. The weights of the neural network are initialized with values sampled from a xavier uniform distribution and the biases are all set to be zero. The L^2 error norm and the energy error norm (H_0^1) defined as follows are used to verify the accuracy of the model

$$||e||_{L^2} = \frac{||\boldsymbol{u}_{ex} - \hat{\boldsymbol{u}}||_{L^2}}{||\boldsymbol{u}_{ex}||_{L^2}} = \frac{\sqrt{\int_\Omega (\boldsymbol{u}_{ex} - \hat{\boldsymbol{u}})^2 \, dV}}{\sqrt{\int_\Omega \boldsymbol{u}_{ex}^2 \, dV}}, \tag{35}$$

$$||e||_{H_0^1} = \frac{||\boldsymbol{u}_{ex} - \hat{\boldsymbol{u}}||_{H_0^1}}{||\boldsymbol{u}_{ex}||_{H_0^1}} = \frac{\sqrt{\frac{1}{2}\int_\Omega (\boldsymbol{\epsilon}^{ex} - \hat{\boldsymbol{\epsilon}}) \, \mathbb{C} \, (\boldsymbol{\epsilon}^{ex} - \hat{\boldsymbol{\epsilon}}) \, dV}}{\sqrt{\frac{1}{2}\int_\Omega \boldsymbol{\epsilon}^{ex} \, \mathbb{C} \, \boldsymbol{\epsilon}^{ex} \, dV}}. \tag{36}$$

According to [30], the NN solution constrained by boundary conditions in the reference domain has the form

$$\hat{u}_x(\xi_1, \xi_2) = \xi_1 \, \hat{u}_x^L(\xi_1, \xi_2) + u_x(\xi_1, \xi_2), \tag{37}$$

$$\hat{u}_y(\xi_1, \xi_2) = \xi_1 \, \hat{u}_y^L(\xi_1, \xi_2) + u_y(\xi_1, \xi_2). \tag{38}$$

An example of mapping from the physical domain to the reference domain using Dataset 1 is illustrated in Figure 7. The approximate displacement agrees well with the exact solution as we can see in Figure 9. In Figure 8 the point-wise error in terms of displacement of three datasets shows that the more points are trained,

the smaller error we obtain. More precisely, as shown in Table 1, the errors in all cases are getting smaller as long as we use more data points. The number of integration points employed in calculating the total potential energy plays a vital factor in order to obtain higher precision. For Dataset 3, the errors decrease to 0.3% for L^2 error norm and 7% for energy error norm. However, more training points would cost more training time in all cases. Besides, we can see the *sigmoid* or *tanh* activation functions can predict accurate solutions in all cases and give roughly the same errors while the result with *CELU* is not stable. This is because *CELU* has a linear part and its derivative would be constant in [0, 1]. Using Dataset 2 and *tanh* activation function returns the optimal solution if we desire to achieve the balance between computational time and accuracy.

4.1.2. Annulus problem

Let us consider a thin disk subject to an internal and/or an external pressure. We assume plane stress conditions. The internal pressure p_i and external pressure p_o act on two surfaces of the disk having the internal radius r_i and outer radius r_o as described in Figure 10. Due to the special geometry and symmetry, the cylindrical coordinate system (r, θ, z) associated with the base vectors $\{\boldsymbol{e}_r, \boldsymbol{e}_\theta, \boldsymbol{e}_z\}$ is used to formulate the analytical solution. The base vectors are expressed as

$$\boldsymbol{e}_r = \begin{pmatrix} \cos\theta \\ \sin\theta \\ 0 \end{pmatrix}; \quad \boldsymbol{e}_\theta = \begin{pmatrix} -\sin\theta \\ \cos\theta \\ 0 \end{pmatrix}; \quad \boldsymbol{e}_z = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}. \tag{39}$$

The exact solutions [42] described in cylindrical coordinate is given as

$$\begin{aligned} \sigma_{rr} &= \frac{p_i\, r_i^2}{r_o^2 - r_i^2}(1 - \frac{r_o^2}{r^2}), \\ \sigma_{\theta\theta} &= \frac{p_i\, r_i^2}{r_o^2 - r_i^2}(1 + \frac{r_o^2}{r^2}), \\ \sigma_{r\theta} &= 0, \\ u(r) &= \frac{r_i^2\, p_i\, r}{E(r_o^2 - r_i^2)} \left(1 - \nu + \left(\frac{r_o}{r}\right)^2 (1 + \nu)\right), \end{aligned} \tag{40}$$

and the displacement solution described in Cartesian space is written as

$$\boldsymbol{u} = u(r)\, \boldsymbol{e}_r. \tag{41}$$

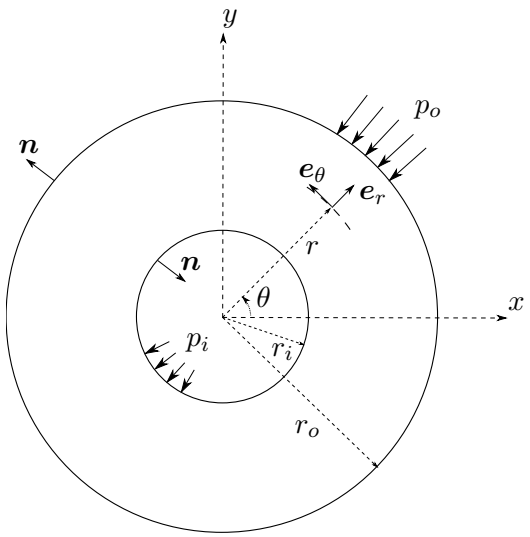


Figure 10: Pressurized annulus.

The stress components in case of plane stress state in elasticity are calculated through the constitutive relation (28) as follows

$$\begin{bmatrix}\sigma_{11}\\ \sigma_{22}\\ \sigma_{12}\end{bmatrix} = \frac{E}{1-\nu^2}\begin{bmatrix}1 & \nu & 0\\ & 1 & 0\\ \text{sym} & & (1-\nu)/2\end{bmatrix}\begin{bmatrix}\epsilon_{11}\\ \epsilon_{22}\\ 2\epsilon_{12}\end{bmatrix}. \tag{42}$$

Because of the geometrical symmetry of the pressurized annulus, we study a quarter of the problem. The material parameters are $E = 135\text{GPa}$, $\nu = 0.3$ and geometry parameters $r_i = 1\mu\text{m}$, $r_o = 5\mu\text{m}$; the pressures are $p_o = 10\text{MPa}$ and $p_o = 0\text{MPa}$.

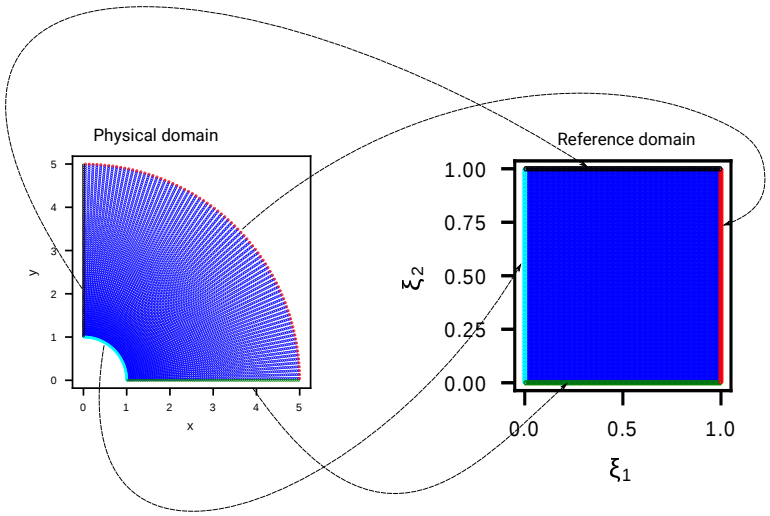


Figure 11: A quarter of the annulus is mapped to the reference domain. Its boundaries in the physical domain are mapped to the corresponding edges in the reference domain.

No	Dataset	Network	Optimizer	Epochs	Learning rate	Activation
1	50 x 50	2 - 30 - 30 - 30 - 2	L-BFGS	100	0.01	tanh
2	80 x 80	2 - 30 - 30 - 30 - 2	L-BFGS	100	0.1	tanh

Table 2: Hyper-parameter setting for the annulus problem.

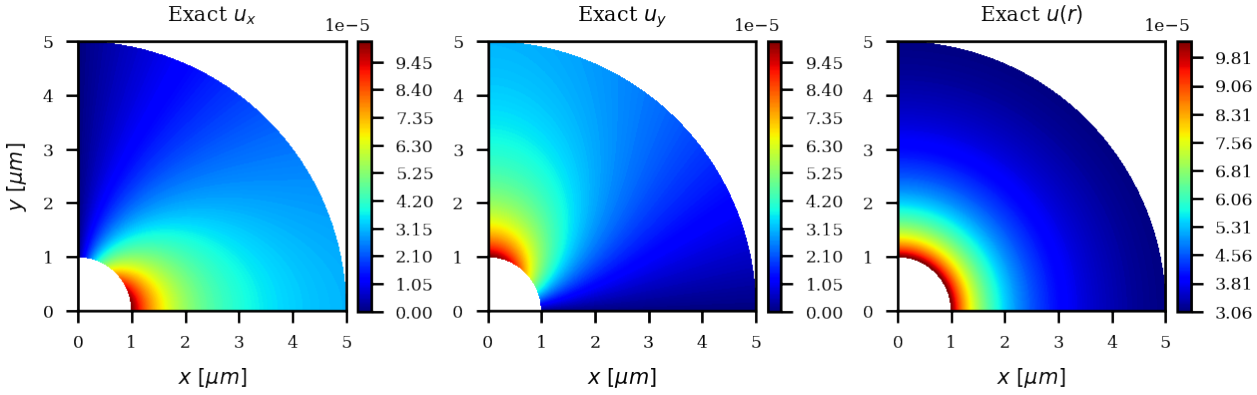


Figure 12: Exact displacement solutions.

In this example, we generate two datasets: (1) 50×50 equidistant grid points for the internal domain and 50 points along each edge for the boundary, (2) 80×80 equidistant grid points for the internal domain, and 80 points along each edge for the boundary. The list of hyperparameters is described in Table 2. The reason to choose such different learning rates is that with a small dataset, a large learning rate will cause the explosion. The mapping from the physical domain to the corresponding reference domain using Dataset 2 is demonstrated in Figure 11. Moreover, solutions of the neural network constrained by boundary conditions are expressed as

$$\begin{aligned} \hat{u}_x(\xi_1, \xi_2) &= (\xi_2 - 1)\, \hat{u}_x^L, \\ \hat{u}_y(\xi_1, \xi_2) &= \xi_2\, \hat{u}_y^L. \end{aligned} \tag{43}$$

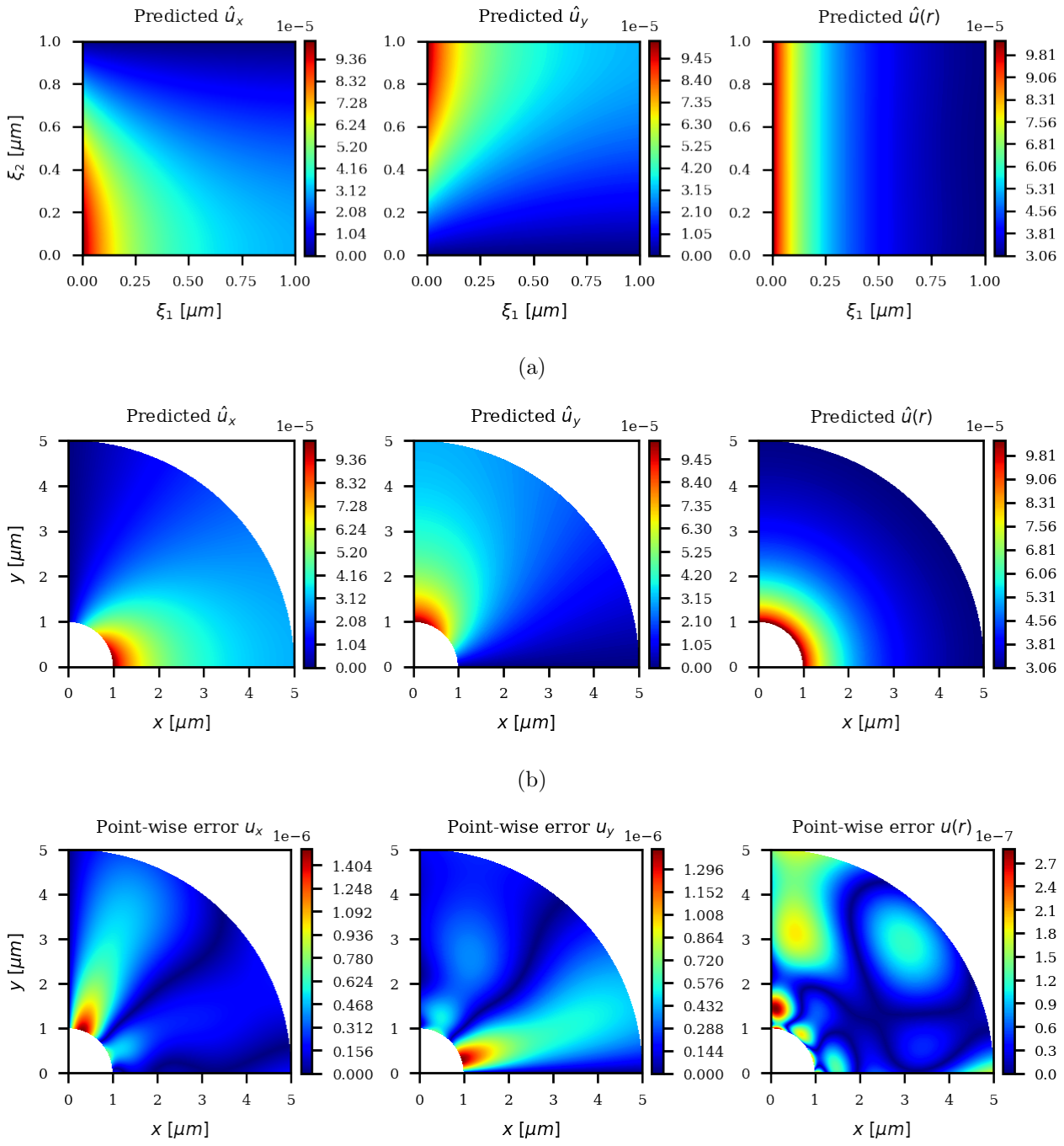
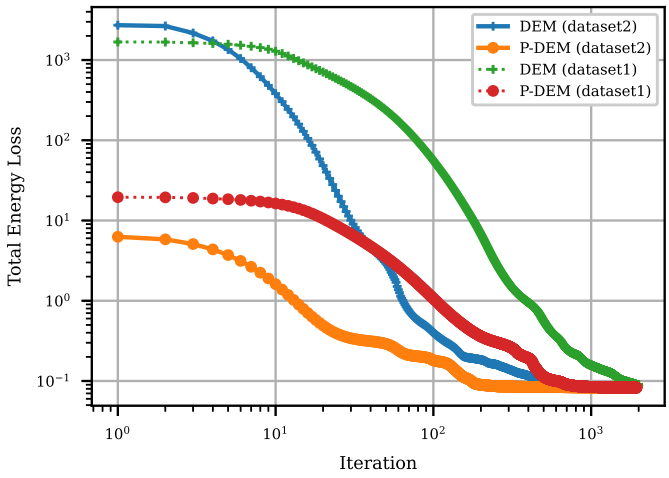


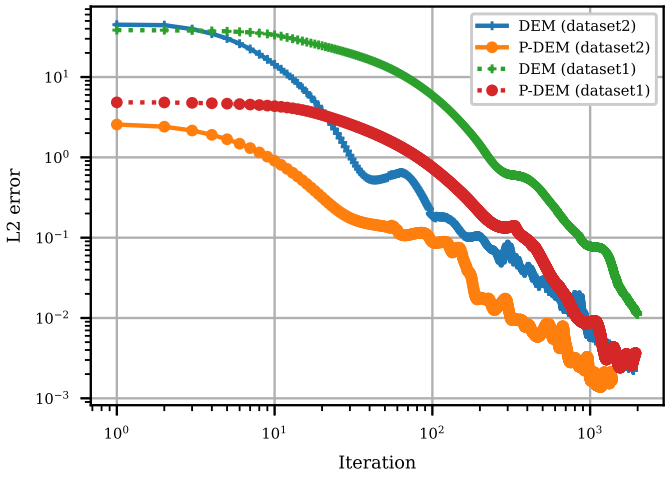
Figure 13: Displacements in x, y and radial direction predicted by P-DEM in reference domain using Dataset 2 (a). The results in reference domain are mapped back to the physical domain (b). The comparison between exact solution and predicted solution is shown (c) measured in point-wise error ($|u - \hat{u}|$).

The predicted displacements in the parametric space training with Dataset 2 are plotted in Figure 13a. The solutions are then mapped to the corresponding positions on the physical domain as illustrated in Figure 13b. As we can see from Figures 13b and 12, the predicted solutions agree well with the exact solutions. The point-wise errors between the exact and approximate solution in Figure 13c are relatively small. We also

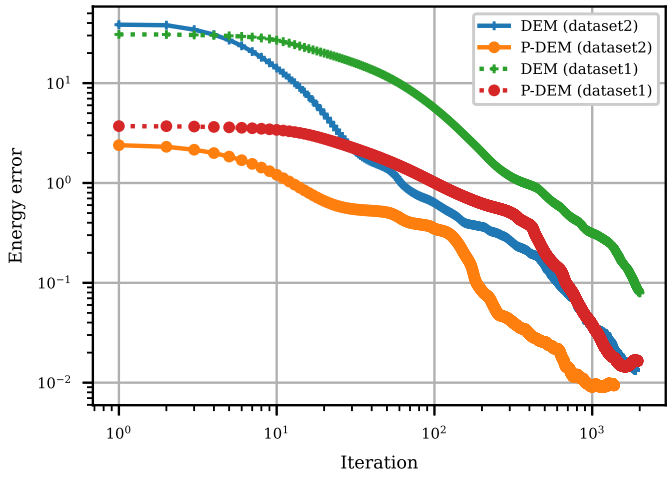
compare the performance of P-DEM and DEM. As shown in Figure 14a, although both P-DEM and DEM converge to the optimal value below 10^{-1} after 200 iterations, P-DEM always reaches this value before DEM in all cases. Moreover, the total energy loss of P-DEM and DEM training with more points (Dataset 2) is always smaller than that of training with fewer points (Dataset 1). To measure the accuracy of P-DEM, we compare the errors in L^2 error norm and energy error norm yielded by P-DEM with DEM. The results in Figures 13b and 13c show that P-DEM's curves always go deeper than DEM. That means P-DEM obtains smaller error than DEM, and thus more accurate than DEM if we use the same settings such as dataset, network architecture, hyperparameters. In all cases, the more data points we used to train, the better results we will obtain. P-DEM is more efficient than DEM at least for this specific application because the data are automatically scaled within the unit domain, and it prevents the gradient vanishing issue that might cause slow convergence as long as *tanh* or *sigmoid* is used as an activation function. Figure 15 shows the point-wise error on the physical domain, and L^2 error norm and energy error norm are given in Table 3. P-DEM has improved the results since the errors of P-DEM measured in L^2 norm and energy norm are much smaller than the errors of DEM. In particular, while DEM achieves an accuracy of about 1% for L^2 error norm and 8% for energy error norm with Dataset 1, P-DEM achieves a higher accuracy of about 0.4% and 2% for the corresponding error measures. With Dataset 2, while DEM obtains only around 0.3% for L^2 error norm and 1% for energy norm, P-DEM obtains smaller errors of about 0.2% and 0.9%, respectively.



(a)

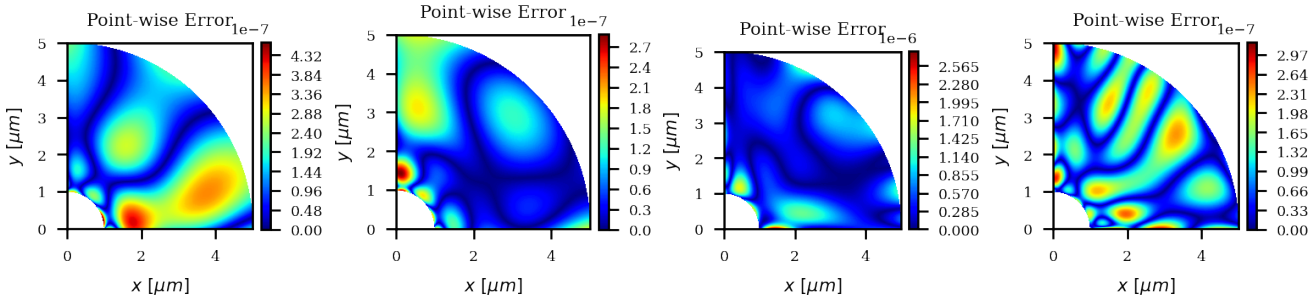


(b)



(c)

Figure 14: The convergence of the total energy loss of both P-DEM and DEM in two training datasets (a). The comparison between P-DEM and DEM in terms of L^2 error norm (b) and energy error norm (c) in two training datasets.



(a) P-DEM (DS1) (b) P-DEM (DS2) (c) DEM (DS1) (d) DEM (DS2)

Figure 15: The point-wise error $|u(r) - \hat{u}(r)|$ plots of both P-DEM and DEM in two datasets.

Dataset	Type	L^2 enorm	Energy enorm
DS1	DEM	0.011733	0.080316
DS1	P-DEM	0.004260	0.016827
DS2	DEM	0.002636	0.013338
DS2	P-DEM	0.001872	0.009401

Table 3: The L^2 and energy error norm of P-DEM and DEM comparison.

4.1.3. Plate with a circular hole

Let us consider an infinite plate with a circular hole subject to a uniform traction at infinity. The problem is assumed in the plane stress state. We model a finite region of the plate as shown in Figure 16 and apply traction boundary conditions according to the analytical solution. The parameters in this problem are: E = 10^3N/m^2, $\mu = 0.3$, L = 4m, R = 1m, $T_x = 10$N.

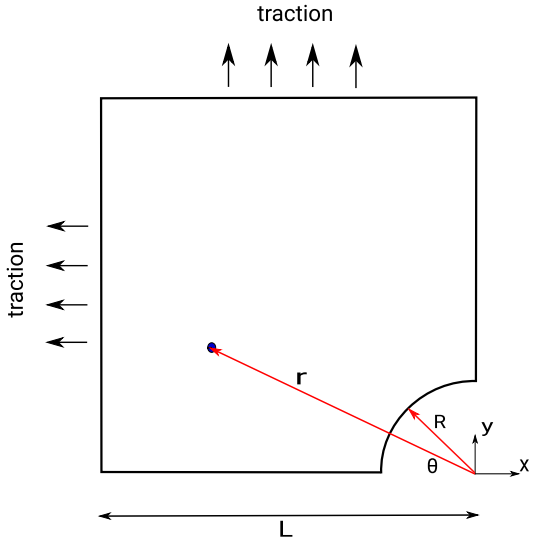


Figure 16: Plate with a circular hole.

The exact solutions in polar coordinates are given as

$$
\begin{aligned}
u(r,\theta) &= \frac{1+\nu}{E}T_x\left(\frac{1}{1+\nu}r\cos\theta + 2\frac{R^2}{(1+\nu)r}\cos\theta + \frac{R^2}{2r}\cos 3\theta - \frac{R^4}{2r^3}\cos 3\theta\right)\\
v(r,\theta) &= \frac{1+\nu}{E}T_x\left(\frac{-\nu}{1+\nu}r\sin\theta - (1-\nu)\frac{R^2}{(1+\nu)r}\sin\theta + \frac{R^2}{2r}\sin 3\theta - \frac{R^4}{2r^3}\sin 3\theta\right)\\
\sigma_{rr}(r,\theta) &= \frac{T_x}{2}\left(1-\frac{R^2}{r^2}\right) + \frac{T_x}{2}\left(1+3\frac{R^4}{r^4} - 4\frac{R^2}{r^2}\right)cos2\theta\\
\sigma_{\theta\theta}(r,\theta) &= \frac{T_x}{2}(1+\frac{R^2}{r^2}) - \frac{T_x}{2}(1+3\frac{R^4}{r^4})\cos 2\theta\\
\sigma_{r\theta}(r,\theta) &= -\frac{T_x}{2}\left(1+2\frac{R^2}{r^2} - 3\frac{R^4}{r^4}\right)\sin 2\theta
\end{aligned}
\tag{44}
$$

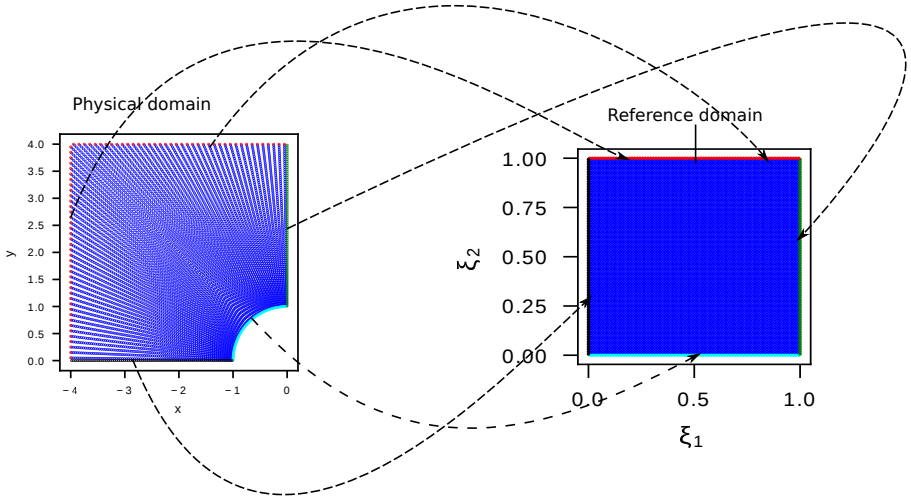


Figure 17: A quarter of a plate with a circular hole is mapped to the reference domain. Its boundaries in the physical domain are mapped to the corresponding edges in the reference domain.

Again, the same neural network structure (2 - 30 - 30 - 30 - 2) is employed in this example. We use the L-BFGS optimizer with the learning rate is set to be 0.5 and the network is trained for 350 epochs. An equidistant grid points 50×50 distributed inside the domain and 50 points equally located along each edge are used for the training network. A mapping from the physical domain to the reference domain is done with the help of *NURBS* functions as shown in Figure 17. Note that due to the mapping from the physical domain to the reference domain, solutions of our neural network constrained by boundary conditions must be modified corresponding to the mapping

$$\begin{aligned} \hat{u}_x(\xi_1, \xi_2) &= (\xi_1 - 1)\, \hat{u}_x^L, \\ \hat{u}_y(\xi_1, \xi_2) &= \xi_1\, \hat{u}_y^L. \end{aligned} \tag{45}$$

The displacement solutions in x and y directions are shown in Figure 18. Solutions obtained by P-DEM presented in a unit square domain are returned to the physical domain. As we can see, the approximate solutions are in good agreement with the exact solutions in this case and the error is relatively small. Furthermore, we also test the result by using different network structures with an increasing number of hidden layers. As shown in Table 4, although using four hidden layers we can obtain the best result in terms of energy error norm, using three hidden layers is sufficient for a 2D problem to achieve the optimal solution, 0.1% for L^2 error norm and 4.3% for energy error norm, while the shallow network gives around 5.8% and 10% for L^2 and energy error norm, respectively. This numerical result agrees with the approximation theorem for PDEs (Section 7 in [22]).

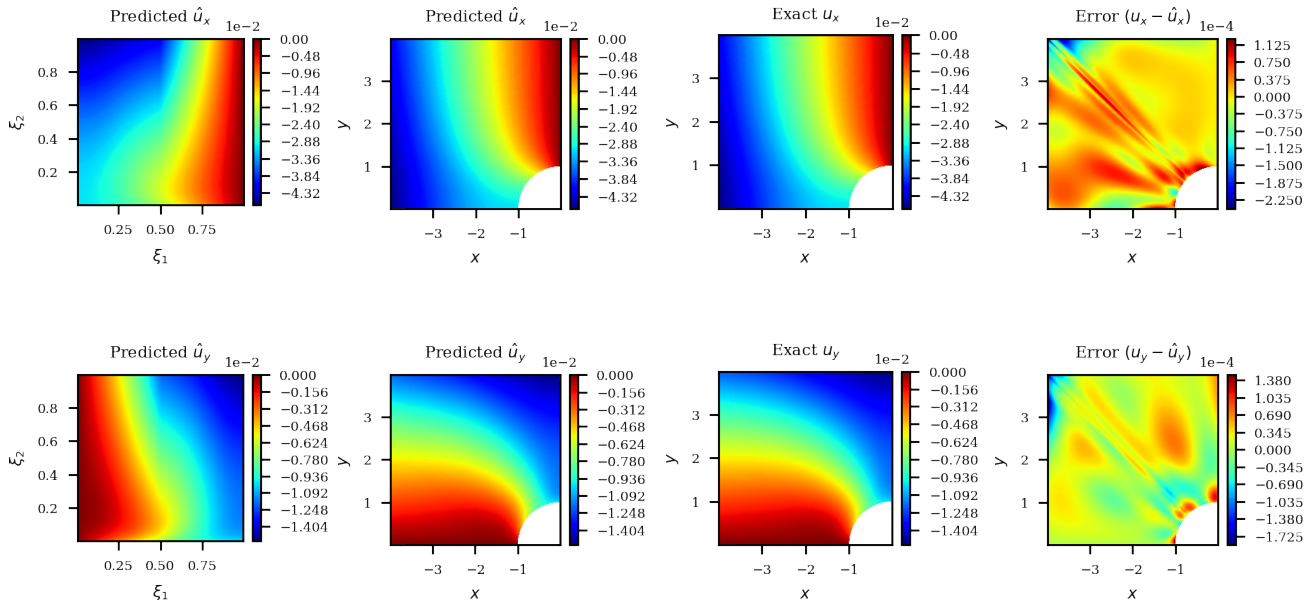


Figure 18: Displacements in x, y directions predicted by P-DEM in reference domain. The results in reference domain are mapped back to the physical domain. The exact solutions are provided for the comparison.

Hidden layers	**Network**	L^2 **enorm**	**Energy enorm**
1	2 - 30 - 2	0.058442	0.102506
2	2 - 30 - 30 - 2	0.013185	0.048588
3	2 - 30 - 30 - 30 - 2	0.001752	0.043349
4	2 - 30 - 30 - 30 - 30 -2	0.008157	0.026548

Table 4: Solutions measured in L^2 error norm and energy norm with respect to the number of hidden layer.

4.2. Strain gradient elasticity

Higher-order strain gradient theories are commonly used to describe materials at small scales that exhibit significant microstructural size effects. Strain gradient theories have been proposed e.g. by Mindlin [48], Casal [49, 50], Vardoulakis & Sulem [51], Vardoulakis et al. [52], Exadaktylos & Vardoulakis [53] or Exadaktylos [54, 55]. The work of Altan & Aifantis [56] simplifies the theory of Mindlin & Eshel [57]. In fact, only three material parameters describe the nonlocal effect in elasticity, i.e. the two Lamé's constants and a strain gradient coefficient. Despite its simplicity, a complete variational formulation had not been yet established until 2007 [58]. We adopt the model of Altan & Aifantis [56] and the analytical solutions derived in [58] to demonstrate the capabilities of the proposed P-DEM.

Let us consider an isotropic, gradient-dependent, linearly elastic material. The strain energy density Ψ can be expressed by

$$\Psi = \Psi(\epsilon_{ij}, \epsilon_{ij,k}) = \frac{1}{2}\,\lambda\,\epsilon_{ii}\,\epsilon_{jj} + \mu\,\epsilon_{ij}\,\epsilon_{ij} + c\,(\frac{1}{2}\,\lambda\,\epsilon_{ii,k}\,\epsilon_{jj,k} + \mu\,\epsilon_{ij,k}\,\epsilon_{ij,k}), \tag{46}$$

where λ and μ are the Lamé constants, c is the additional material length scale parameter, or so-called the strain gradient coefficient. The components of the classical infinitesimal strain tensor ϵ_{ij} are given as

$$\epsilon_{ij} = \frac{1}{2}(u_{i,j} + u_{j,i}). \tag{47}$$

Then, the strain energy U of the body reads

$$U = \int_\Omega \Psi \, dV = \frac{1}{2} \int_\Omega (\tau_{ij} \, \epsilon_{ij} + \mu_{ijk} \, \kappa_{ijk}) \, dV, \tag{48}$$

where the components of the Cauchy stress, the double stress, and the strain gradient are denoted by τ_{ij}, μ_{ijk}, and κ_{ijk}, respectively. Like classical strain theory, the Cauchy stress is defined as

$$\tau_{ij} = \frac{\partial \Psi}{\partial \epsilon_{ij}} = \lambda \, \epsilon_{ll} \, \delta_{ij} + 2 \, \mu \, \epsilon_{ij} = \tau_{ij}. \tag{49}$$

Taking the derivative of the strain energy w.r.t the strain gradient, we obtain the double stress tensor

$$\mu_{ijk} = \frac{\partial \Psi}{\partial \kappa_{ijk}} = c \, (\lambda \, \epsilon_{ll} \, \delta_{ij} + 2 \, \mu \, \epsilon_{ij})_{,k} = c \, \tau_{ij,k} = \mu_{jik}, \tag{50}$$

in which the strain gradient is defined as

$$\kappa_{ijk} = \epsilon_{ij,k} = \frac{1}{2} \, (u_{i,jk} + u_{j,ik}). \tag{51}$$

The work done by the external loads is given as

$$W = \int_\Omega f_i \, u_i \, dV + \int_{\partial\Omega} (t_i \, u_i + q_i Du_i) \, dA, \tag{52}$$

where f_i, and t_i are the components of the body force and traction, respectively. The new term q_i is the double stress traction, and Du_i is the normal (directional) derivative of the displacement u_i defined as

$$Du_i = n_l \, u_{i,l} \tag{53}$$

with n_l being the components of the outward unit normal vector on the boundary $\partial\Omega$ of the structure. The total potential energy density Π is obtained by combining equations (48) and (52), which results in

$$\Pi = U - W = \frac{1}{2} \int_\Omega (\tau_{ij} \, \epsilon_{ij} + \mu_{ijk} \, \kappa_{ijk}) \, dV - \int_\Omega f_i \, u_i \, dV - \int_{\partial\Omega} (t_i \, u_i + q_i \, Du_i) \, dA. \tag{54}$$

The additional couple of power conjugate, double stress μ_{ijk} and strain gradient κ_{ijk}, will not be taken into account in the computation if the strain gradient coefficient c is zero. Obviously, the equation contains the second-order derivative of displacement with respect to the spatial coordinates, and thus it requires C^1 continuity. Conventional FEM based on Lagrange polynomials is only C^0, whereas the activation functions of the neural networks in the P-DEM easily fulfill the C^1 requirement. Equation (54) also yields the strong form from the weak form based on its equality if we apply the principle of minimum total potential energy ($\delta\Pi = 0$) in order to find the stationary point as described in section 2. It leads to the governing equations given by

$$\sigma_{ij,j} + f_i = 0 \quad \text{in } \Omega \tag{55}$$

where $\boldsymbol{\sigma} = \sigma_{ij}\boldsymbol{e}_i \otimes \boldsymbol{e}_j$ are the components of the total stress defined as

$$\sigma_{ij} = \tau_{ij} - \mu_{ijk,k}. \tag{56}$$

The boundary conditions are given by

$$\underbrace{\sigma_{ij}n_j - (\mu_{ijk}n_k)_{,j} + (\mu_{ijk}n_k n_l)_{,l}n_j = \overline{t}_i}_{\text{Surface traction B.C.}} \quad \text{or} \quad \underbrace{u_i = \overline{u}_i}_{\text{Displacement B.C.}} \quad \text{on} \quad \partial\Omega \tag{57}$$

$$\underbrace{\mu_{ijk}n_j n_k = \overline{q}_i}_{\text{Higher-order traction B.C.}} \quad \text{or} \quad \underbrace{u_{i,l}n_l = \overline{u}_{i,l}n_l}_{\text{Displacement gradients B.C.}} \quad \text{on} \quad \partial\Omega \tag{58}$$

Let us consider the same problem from section 4.1.2 in plane strain state but now employing strain-gradient theory (see Figure 10). Again, the problem can be formulated in cylindrical coordinates. In [58], the analytical solution is given as

$$u(r) = A\,r + \frac{B}{r} + CI_1\left(\frac{1}{\sqrt{c}}r\right) + DK_1\left(\frac{1}{\sqrt{c}}r\right), \tag{59}$$

in which $I_1(\cdot)$ and $K_1(\cdot)$ are the modified Bessel functions of the first and second kinds of order 1, respectively. The solution shows that the radial displacement explicitly depends on the strain gradient coefficient c. With a combination of boundary conditions, four constants A, B, C, D can be determined by solving a system of these algebraic equations

$$\begin{aligned}
&(-2\lambda-2\mu)\,A + \left(\frac{2\mu}{r_i^2} + \frac{4c\mu}{r_i^4}\right)B + \left[-\frac{2\sqrt{c}\mu}{r_i^2}I_0\left(\frac{1}{\sqrt{c}}r_i\right) - \left(\frac{\lambda}{r_i} - \frac{4c\mu}{r_i^3}\right)I_1\left(\frac{1}{\sqrt{c}}r_i\right)\right]C \\
&\qquad + \left[\frac{2\sqrt{c}\mu}{r_i^2}K_0\left(\frac{1}{\sqrt{c}}r_i\right) - \left(\frac{\lambda}{r_i} - \frac{4c\mu}{r_i^3}\right)K_1\left(\frac{1}{\sqrt{c}}r_i\right)\right]D = p_i, \qquad (60)\\
&(-2\lambda-2\mu)\,A + \left(\frac{2\mu}{r_o^2} + \frac{4c\mu}{r_o^4}\right)B + \left[-\frac{2\sqrt{c}\mu}{r_o^2}I_0\left(\frac{1}{\sqrt{c}}r_o\right) - \left(\frac{\lambda}{r_o} - \frac{4c\mu}{r_o^3}\right)I_1\left(\frac{1}{\sqrt{c}}r_o\right)\right]C \\
&\qquad + \left[\frac{2\sqrt{c}\mu}{r_o^2}K_0\left(\frac{1}{\sqrt{c}}r_o\right) - \left(\frac{\lambda}{r_o} - \frac{4c\mu}{r_o^3}\right)K_1\left(\frac{1}{\sqrt{c}}r_o\right)\right]D = p_o, \qquad (61)\\
&\frac{4\mu}{r_i^3}B + \left[-\frac{2\mu}{\sqrt{c}r_i}I_0\left(\frac{1}{\sqrt{c}}r_i\right) + \left(\frac{\lambda+2\mu}{c} + \frac{4\mu}{r_i^2}\right)I_1\left(\frac{1}{\sqrt{c}}r_i\right)\right]C \\
&+ \left[\frac{2\mu}{\sqrt{c}r_i}K_0\left(\frac{1}{\sqrt{c}}r_i\right) + \left(\frac{\lambda+2\mu}{c} + \frac{4\mu}{r_i^2}\right)K_1\left(\frac{1}{\sqrt{c}}r_i\right)\right]D = 0, \qquad (62)\\
&\frac{4\mu}{r_o^3}B + \left[-\frac{2\mu}{\sqrt{c}r_o}I_0\left(\frac{1}{\sqrt{c}}r_o\right) + \left(\frac{\lambda+2\mu}{c} + \frac{4\mu}{r_o^2}\right)I_1\left(\frac{1}{\sqrt{c}}r_o\right)\right]C \\
&+ \left[\frac{2\mu}{\sqrt{c}r_o}K_0\left(\frac{1}{\sqrt{c}}r_o\right) + \left(\frac{\lambda+2\mu}{c} + \frac{4\mu}{r_o^2}\right)K_1\left(\frac{1}{\sqrt{c}}r_o\right)\right]D = 0. \qquad (63)
\end{aligned}$$

The strain field is given by

$$\epsilon(r) = \frac{du}{dr}\boldsymbol{e}_r \otimes \boldsymbol{e}_r + \frac{u}{r}\boldsymbol{e}_\theta \otimes \boldsymbol{e}_\theta. \tag{64}$$

The converting tensors between Cartesian and Cylindrical-polar bases are obtained by the following transformation

$$\begin{bmatrix} \epsilon_{rr} & \epsilon_{r\theta} & \epsilon_{rz} \\ \epsilon_{\theta r} & \epsilon_{\theta\theta} & \epsilon_{\theta z} \\ \epsilon_{zr} & \epsilon_{z\theta} & \epsilon_{zz} \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \epsilon_{xx} & \epsilon_{xy} & \epsilon_{xz} \\ \epsilon_{yx} & \epsilon_{yy} & \epsilon_{yz} \\ \epsilon_{zx} & \epsilon_{zy} & \epsilon_{zz} \end{bmatrix} \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}. \tag{65}$$

The material properties are: Young's modulus $E = 135$GPa, Poisson's ratio $\nu = 0.3$, and different strain gradient coefficient, i.e. $c = 0.25\mu\text{m}^2$, $c = 0.01\mu\text{m}^2$ and $c = 0.05\mu\text{m}^2$. The other parameters are adopted from

the example in the previous subsection, i.e. $r_i = 1\mu m$, $r_o = 5\mu m$, $p_i = 10MPa$, $p_o = 0MPa$.

For the hyperparameter settings, we use two feed-forward neural networks which are connected to the same input as schematically depicted in Figure 20. The first NN has the structure 2-30-50-50-30-2 where the output layer has two neurons for the displacement components, while the second NN has the structure 2-30-50-50-30-3 whose last layer has three neurons for the strain components. The reason for such structure is that displacement and strain have different lengthscale ratios, i.e. one has a unit of measurement while the other one is unitless; sharing the same ansatz functions, which can be considered as the activation functions, might lead to a suboptimal approximation. Besides, in order to calculate strain gradients the derivative of strain with respect to parametric coordinates needs to be evaluated, i.e. $\partial\boldsymbol{\epsilon}/\partial\boldsymbol{X} = \partial\boldsymbol{\epsilon}/\partial\boldsymbol{\xi} \cdot \partial\boldsymbol{\xi}/\partial\boldsymbol{X}$, using only displacements as the output of the network results in the tedious derivation and computation of this term. Instead, adding more neurons for output strains will help us to fully exploit the potential of our network because it utilizes the automatic differentiation of the computational graph integrated in the machine learning framework. Note that we still calculate the components of strain based on the primary output displacements like in elasticity, denoted by $\hat{\epsilon}^*$, and the additional output strains, denoted by $\hat{\epsilon}$, are enforced by a regularization term, i.e. $||\hat{\epsilon}^* - \hat{\epsilon}||$. Therefore, the compatibility conditions are still satisfied. The components of strain gradient are explicitly expressed as

$$
\begin{aligned}
\kappa_{111} = \epsilon_{11,1} &= \frac{\partial \epsilon_{11}}{\partial X_1} = \frac{\partial \epsilon_{11}}{\partial \xi_1}\frac{\partial \xi_1}{\partial X_1} + \frac{\partial \epsilon_{11}}{\partial \xi_2}\frac{\partial \xi_2}{\partial X_1}, \\
\kappa_{112} = \epsilon_{11,2} &= \frac{\partial \epsilon_{11}}{\partial X_2} = \frac{\partial \epsilon_{11}}{\partial \xi_1}\frac{\partial \xi_1}{\partial X_2} + \frac{\partial \epsilon_{11}}{\partial \xi_2}\frac{\partial \xi_2}{\partial X_2}, \\
\kappa_{121} = \epsilon_{12,1} &= \frac{\partial \epsilon_{12}}{\partial X_1} = \frac{\partial \epsilon_{12}}{\partial \xi_1}\frac{\partial \xi_1}{\partial X_1} + \frac{\partial \epsilon_{12}}{\partial \xi_2}\frac{\partial \xi_2}{\partial X_1}, \\
\kappa_{122} = \epsilon_{12,2} &= \frac{\partial \epsilon_{12}}{\partial X_2} = \frac{\partial \epsilon_{12}}{\partial \xi_1}\frac{\partial \xi_1}{\partial X_2} + \frac{\partial \epsilon_{12}}{\partial \xi_2}\frac{\partial \xi_2}{\partial X_2}, \\
\kappa_{221} = \epsilon_{22,1} &= \frac{\partial \epsilon_{22}}{\partial X_1} = \frac{\partial \epsilon_{22}}{\partial \xi_1}\frac{\partial \xi_1}{\partial X_1} + \frac{\partial \epsilon_{22}}{\partial \xi_2}\frac{\partial \xi_2}{\partial X_1}, \\
\kappa_{222} = \epsilon_{22,2} &= \frac{\partial \epsilon_{22}}{\partial X_2} = \frac{\partial \epsilon_{22}}{\partial \xi_1}\frac{\partial \xi_1}{\partial X_2} + \frac{\partial \epsilon_{22}}{\partial \xi_2}\frac{\partial \xi_2}{\partial X_2}, \\
\kappa_{211} &= \kappa_{121}, \\
\kappa_{212} &= \kappa_{122},
\end{aligned}
\tag{66}
$$

where the derivatives of the strains with respect to the natural coordinates can be obtained by utilizing the automatic differentiation and computational graph, which are already embedded in the PyTorch framework. As shown in Figure 19, the problem can be solved by defining an energy form written as a loss function with the symmetry boundary conditions for displacements and displacement gradients imposed on the left and bottom edge. The boundary conditions at the left edge read

$$u_1(0, x_2) = 0, \tag{67}$$
$$u_{1,2}(0, x_2) = 0, \tag{68}$$
$$u_{2,1}(0, x_2) = 0, \tag{69}$$

and the boundary conditions at the bottom edge are

$$u_2(x_1, 0) = 0, \tag{70}$$
$$u_{1,2}(x_1, 0) = 0, \tag{71}$$
$$u_{2,1}(x_1, 0) = 0. \tag{72}$$

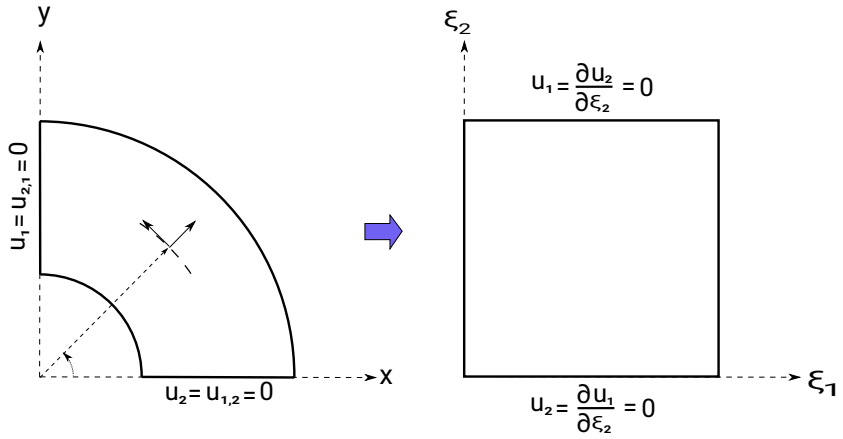


Figure 19: A quarter of the domain is mapped to the parametric space.

The loss function in this case can be fully expressed as

$$\mathcal{L}(\hat{\boldsymbol{u}}; \boldsymbol{\theta}) = U(\hat{\boldsymbol{u}}; \boldsymbol{\theta}) - W(\hat{\boldsymbol{u}}; \boldsymbol{\theta}) + \gamma_1(BC_1 + BC_2) + \gamma_2(e_1 + e_2 + e_3), \tag{73}$$

where U and W are the internal energy loss and external energy loss, respectively, as defined in equations (48, 52); BC_1 and BC_2 are the boundary conditions written as mean square error loss of the left and the bottom edge; e_1, e_2, e_3 are the mean square error constrains for the difference of the strain components yielded by automatic differentiation $\hat{\epsilon}^*$ and the strain components given by the second fully-connected network $\hat{\epsilon}$; γ_1 and γ_2 are two penalty factors and they are chosen as 12000 and 25000. The displacement boundary conditions are preconstrained as in the elasticity example

$$\hat{u}_1(\xi_1, \xi_2) = (\xi_2 - 1)\, \hat{u}_1^L, \tag{74}$$
$$\hat{u}_2(\xi_1, \xi_2) = \xi_2\, \hat{u}_2^L.$$

The displacement gradient conditions are constrained as

$$BC_1 = \frac{1}{N_{left}} \sum_{i=1}^{N_{left}} ||\frac{\partial \hat{u}_2}{\partial \xi_2} \frac{\partial \xi_2}{\partial x_1} - 0||^2, \tag{75}$$

$$BC_2 = \frac{1}{N_{bottom}} \sum_{i=1}^{N_{bottom}} ||\frac{\partial \hat{u}_1}{\partial \xi_2} \frac{\partial \xi_2}{\partial x_2} - 0||^2, \tag{76}$$

$$\tag{77}$$

where N_{left}, N_{bottom} are the number of points on the left boundary and the bottom boundary, respectively, and the residual strains written in terms of additional losses are

$$e_1 = \frac{1}{N_\Omega}\sum_{i=1}^{N_\Omega} ||\hat{\epsilon}^*_{11}(\boldsymbol{x}_\Omega;\boldsymbol{\theta}) - \hat{\epsilon}_{11}(\boldsymbol{x}_\Omega;\boldsymbol{\theta})||^2, \tag{78}$$

$$e_2 = \frac{1}{N_\Omega}\sum_{i=1}^{N_\Omega} ||\hat{\epsilon}^*_{22}(\boldsymbol{x}_\Omega;\boldsymbol{\theta}) - \hat{\epsilon}_{22}(\boldsymbol{x}_\Omega;\boldsymbol{\theta})||^2, \tag{79}$$

$$e_3 = \frac{1}{N_\Omega}\sum_{i=1}^{N_\Omega} ||\hat{\epsilon}^*_{12}(\boldsymbol{x}_\Omega;\boldsymbol{\theta}) - \hat{\epsilon}_{12}(\boldsymbol{x}_\Omega;\boldsymbol{\theta})||^2, \tag{80}$$

N_Ω being the total points in the domain. The weights of the neural network are first initialized with values sampled from a normal distribution and the biases are all set to be zero. A grid of 2500 collocation points (50×50) in the domain and 100 points on each boundary are generated as the feeding data and they are used for the domain and boundary integration, respectively. We use *tanh* activation function to evaluate neural values in the hidden layers. The L-BFGS optimizer is employed to search for the network parameters. The neural network is trained for 120 epochs with learning rate $lr = 0.05$.

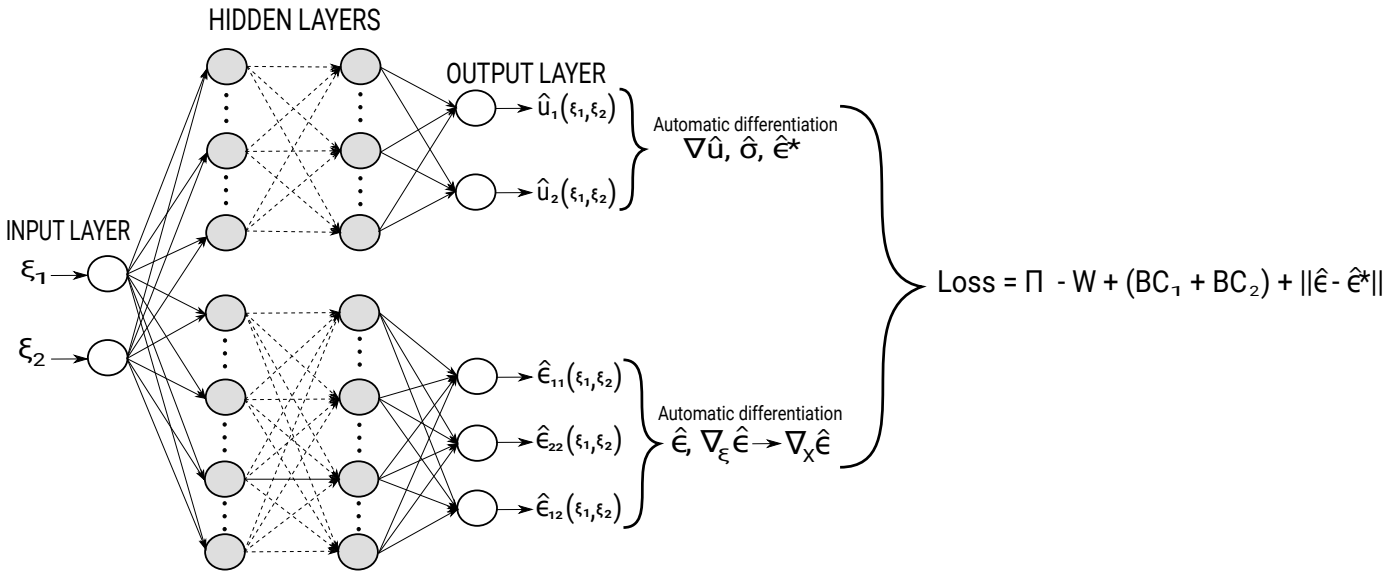
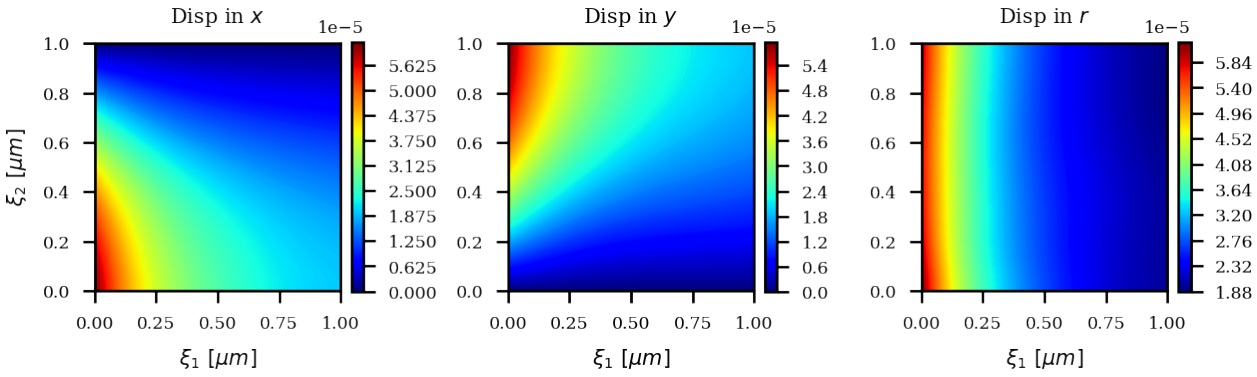
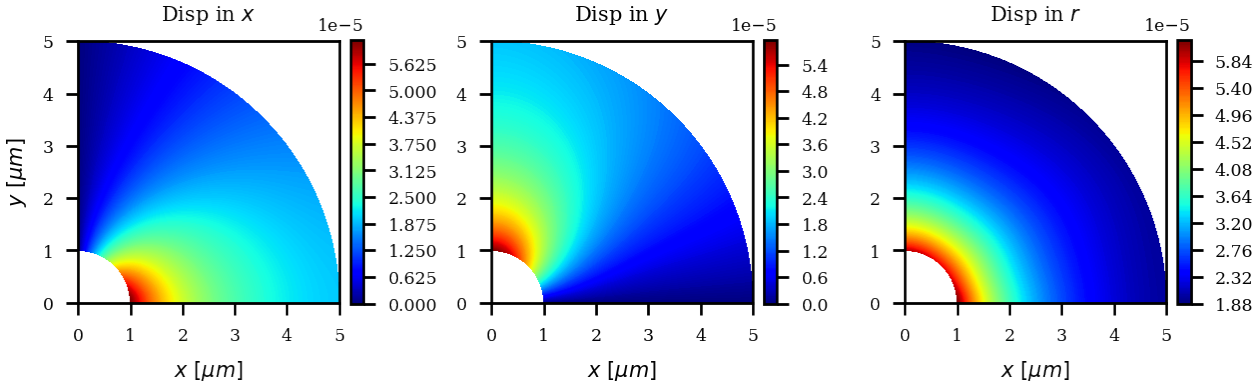


Figure 20: The designed network for the gradient strain elasticity problem.

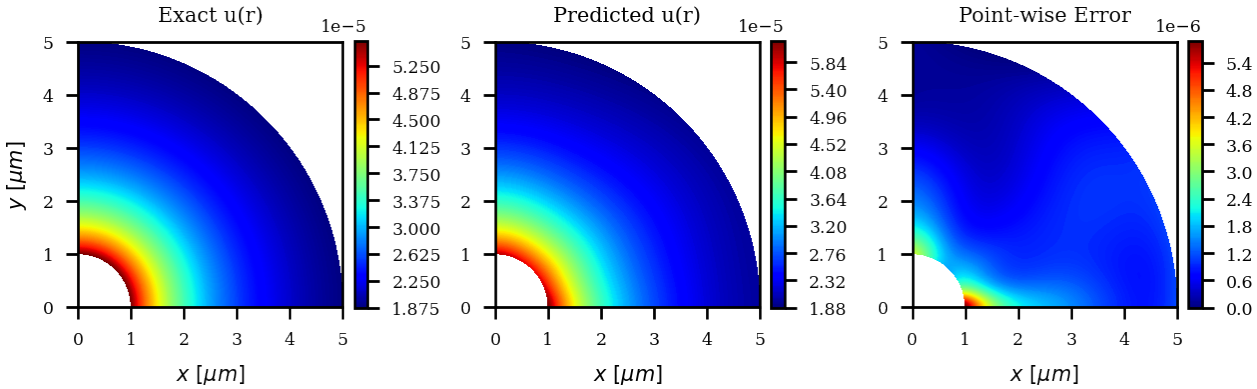
We randomly generate 100 points in the domain and on the boundaries to compare the displacements, radial and circumferential strains between the P-DEM and exact solution. As shown in Figure 22, the approximate solutions agree with the analytical solutions in terms of displacement and strain. Apparently, the significant difference of solutions when we set up the problem with different strain gradient coefficients can be recognized. The magnitude of both displacement and strain of the solutions with high strain gradients are always smaller than the ones with small strain gradients. Furthermore, the differences between the solutions with small strain gradient coefficients ($c = 0.01$ and $c = 0.05$) are relatively small, but are significant when the coefficient is larger ($c = 0.25$). This indicates that the classical solution in elasticity derived in section 4.1.2 may not be accurate for materials at smaller scales due to the microstructural effects.



(a)

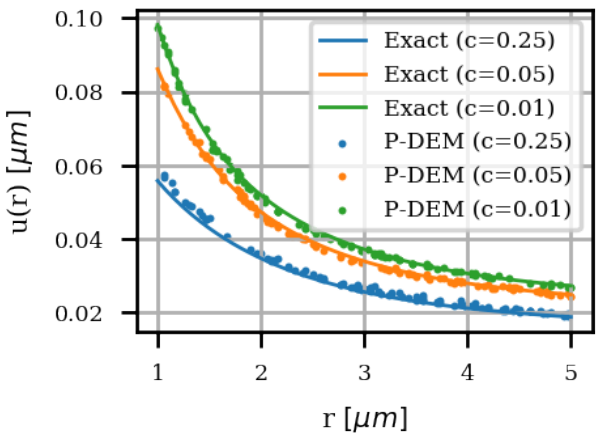


(b)

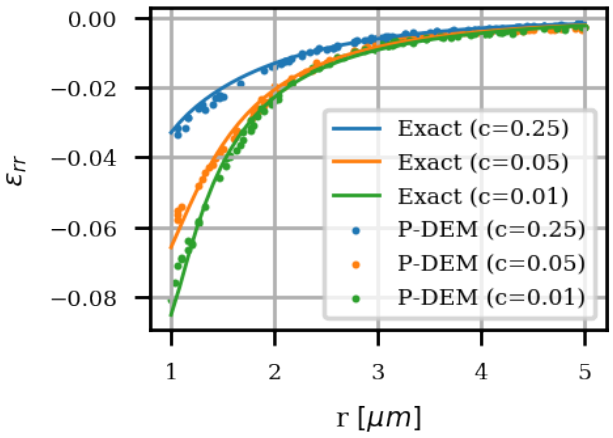


(c)

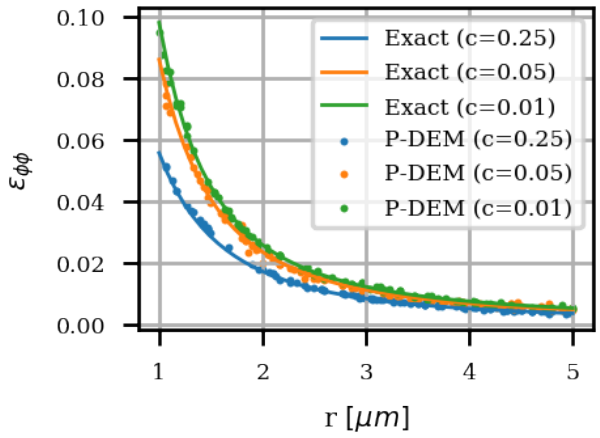
Figure 21: Displacements in x, y directions and radial displacement predicted by our neural network in reference domain (a). The results in reference domain are mapped back to the physical domain (b). The comparison between exact solution and predicted solution is shown with strain gradient coefficient $c = 0.25$ (c).



(a)



(b)



(c)

Figure 22: The comparison between the exact solution and numerical solution for the radial displacement (a), radial strain (b), and circumferential strain (c) with different strain gradient coefficients.

5. Conclusions

We have proposed the P-DEM which is an extension of the original DEM and applied it to problems in elasticity and strain-gradient elasticity. Instead of training the DNN in the physical domain, we train the network in a parameter domain. The correspondence between two domains associated with the forward and backward mapping is achieved through NURBS basis functions commonly used in IGA. In this context, Gauss quadrature is employed to calculate the potential energy that is considered as the loss function. Although it requires some effort in dealing with mapping using NURBS to create a reference domain, it brings as a result some benefits to the training due to the unit square domain. One advantage of the presented P-DEM over the original DEM is that the inputs are automatically normalized within the natural domain, and therefore it prevents the gradient vanishing that might appear when the hyperbolic tangent is used as an activation function. Another advantage that could be exploited is using transfer learning for other problems having different geometries[2]. The robustness of the method is demonstrated through several numerical examples in linear elasticity with different geometries. Furthermore, we presented an example for strain gradient elasticity

[2]The preliminary work of this application can be found in the Appendix

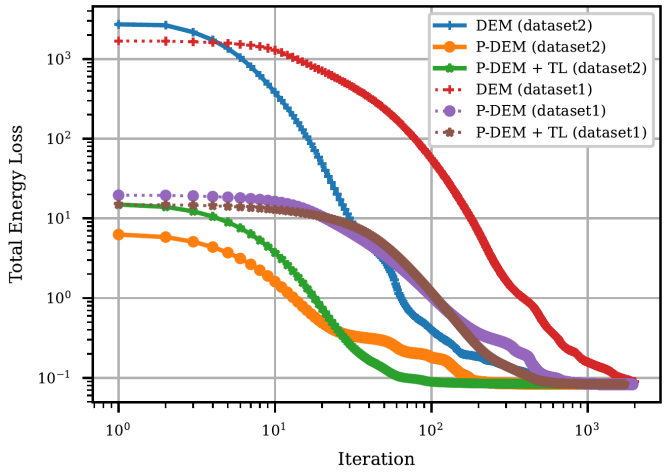
whose strong form is defined by a third-order PDE, which cannot easily be solved in FEM due to the C^1-continuity requirement. The numerical results show that P-DEM achieves the optimal value of loss energy faster than DEM and yields accurate results in presented benchmark problems.

6. Appendix

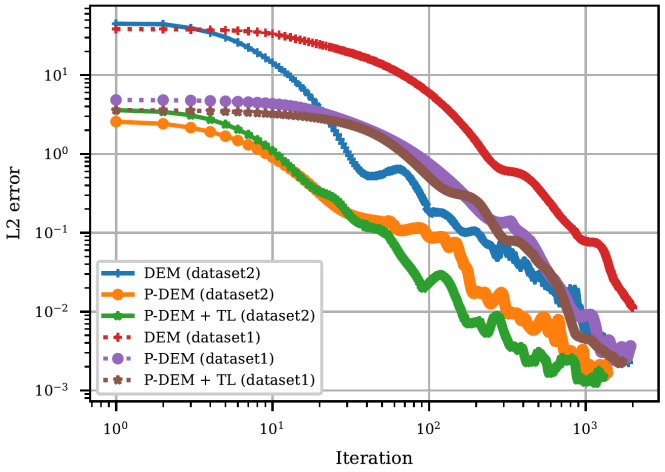
6.1. P-DEM with Transfer Learning

Due to the same training geometry on the reference domain, it is in some cases feasible to reuse the trained parameters of the previous example for the next problems. *Transfer learning* in this scenario is recognized as the powerful technique to retain valuable knowledge by copying or freezing some parts of the trained neural networks [59]. As discussed in the conclusion, the study of how to solve various problems with the same reference domain and how the domain decomposition could work for the entire complex domain is a promising direction, and that could be done in future work. Although it is not completely developed for all problems, one example of using the network's parameters from the Timoshenko beam as the initialization parameters for the annulus problem in 2D elasticity is given to verify its capability.

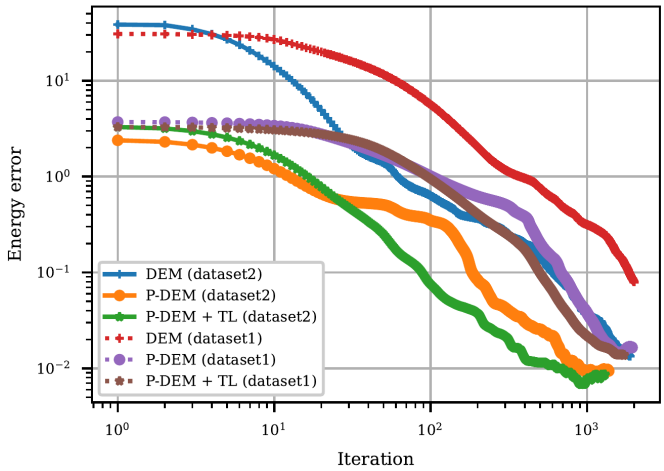
In this example, we use the same settings as in section 4.1.2 and we transfer all weights and biases from the network of the 2D beam to the network of the annulus. Here, we compare the performance of our network in two cases, with and without using transfer learning. As shown in Figure 23, there is a difference in total energy loss, L^2 error norm, and energy error norm of P-DEM with transfer learning and P-DEM without transfer learning. With transfer learning using Dataset 2, the minimal loss value is obtained after around 100 training steps, while training without transfer learning reaches the same level after 300 steps. Similarly, we can see in Figures 23b, 23c the P-DEM combined transfer learning always gives smaller errors than without using transfer learning. In fact, the smallest errors obtained by P-DEM using Dataset 2 without transfer learning are about 0.0018 and 0.0094 as shown in Table 3 for L^2 and energy error norm, respectively, while using transfer learning P-DEM can give better results, 0.0016 and 0.0086, for the corresponding errors as shown in Table 5. The contour plots (see Figure 24) show the point-wise error using transfer learning.



(a)

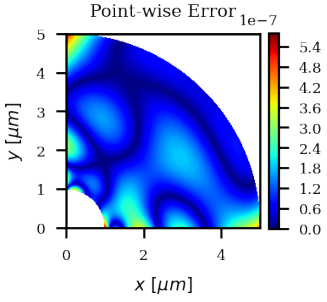


(b)

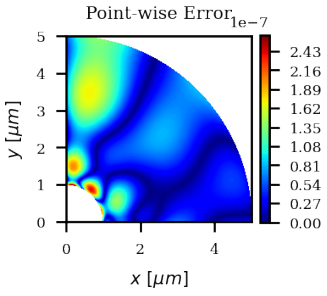


(c)

Figure 23: The convergence of the total energy loss of P-DEM, P-DEM with TL, and DEM in two training datasets(a). The comparison between P-DEM, P-DEM with TL, and DEM in terms of L^2 error norm (b) and energy error norm (c) in two training datasets.



(a) P-DEM + TL (DS1)



(b) P-DEM + TL (DS2)

Figure 24: The point-wise error $|u(r) - \hat{u}(r)|$ plots of P-DEM combined transfer learning in two testcases.

Dataset	Type	L^2 enorm	Energy enorm
DS1	P-DEM + TL	0.002584	0.013786
DS2	P-DEM + TL	0.001661	0.008631

Table 5: The L^2 error norm and energy error norm of both P-DEM combined TL in two testcases.

6.2. Supplementary code

The code of this work will be available at https://github.com/MinhNguyenIKM/parametric-deep-energy-method upon publication.

7. Acknowledgment

The first and last authors owe the gratitude to the sponsorship from Sofja Kovalevskaja Programme of Alexander von Humboldt Foundation.

References

[1] K.-J. Bathe, *Finite Element Procedures*. Prentice Hall, 2006.

[2] R. W. Clough, *The Finite Element Method in Plane Stress Analysis*. American Society of Civil Engineers, 1960.

[3] R. W. Clough, "Original formulation of the finite element method," 1989.

[4] T. Belytschko, Y. Y. Lu, and L. Gu, "Element-free galerkin methods," *International Journal for Numerical Methods in Engineering*, vol. 37, no. 2, pp. 229–256, 1994.

[5] J. J. Monaghan, "Smoothed particle hydrodynamics," *Annual Review of Astronomy and Astrophysics*, vol. 30, no. 1, pp. 543–574, 1992.

[6] V. P. Nguyen, T. Rabczuk, S. Bordas, and M. Duflot, "Meshless methods: A review and computer implementation aspects," *Mathematics and Computers in Simulation*, vol. 79, no. 3, pp. 763 – 813, 2008.

[7] T. Hughes, J. Cottrell, and Y. Bazilevs, "Isogeometric analysis: Cad, finite elements, nurbs, exact geometry and mesh refinement," *Computer Methods in Applied Mechanics and Engineering*, vol. 194, no. 39, pp. 4135 – 4195, 2005.

[8] J. Cottrell, T. Hughes, and Y. Bazilevs, *Isogeometric Analysis: Toward Integration of CAD and FEA*. Wiley, 2009.

[9] V. P. Nguyen, C. Anitescu, S. P. Bordas, and T. Rabczuk, "Isogeometric analysis: An overview and computer implementation aspects," *Mathematics and Computers in Simulation*, vol. 117, pp. 89 – 116, 2015.

[10] S. L. Brunton, J. L. Proctor, and J. N. Kutz, "Discovering governing equations from data by sparse identification of nonlinear dynamical systems," *Proceedings of the National Academy of Sciences*, vol. 113, no. 15, pp. 3932–3937, 2016.

[11] J. Han, A. Jentzen, and W. E, "Solving high-dimensional partial differential equations using deep learning," *Proceedings of the National Academy of Sciences*, vol. 115, no. 34, pp. 8505–8510, 2018.

[12] G. Pang, L. Lu, and G. E. Karniadakis, "fpinns: Fractional physics-informed neural networks," 2018.

[13] L. Lu, X. Meng, Z. Mao, and G. E. Karniadakis, "Deepxde: A deep learning library for solving differential equations," 2020.

[14] C. Rao, H. Sun, and Y. Liu, "Physics informed deep learning for computational elastodynamics without labeled data," 2020.

[15] A. D. Jagtap and G. Em Karniadakis, "Extended physics-informed neural networks (xpinns): A generalized space-time domain decomposition based deep learning framework for nonlinear partial differential equations," *Communications in Computational Physics*, vol. 28, no. 5, pp. 2002–2041, 2020.

[16] M. Raissi and G. E. Karniadakis, "Hidden physics models : Machine learning of nonlinear partial differential equations," *Journal of Computational Physics*, vol. 357, pp. 125–141, 2018.

[17] M. Raissi, "Deep hidden physics models: Deep learning of nonlinear partial differential equations," *J. Mach. Learn. Res.*, vol. 19, p. 932–955, Jan. 2018.

[18] E. Kharazmi, Z. Zhang, and G. E. Karniadakis, "hp-vpinns: Variational physics-informed neural networks with domain decomposition," *Computer Methods in Applied Mechanics and Engineering*, vol. 374, p. 113547, 2021.

[19] A. Li, R. Chen, A. B. Farimani, and Y. J. Zhang, "Reaction diffusion system prediction based on convolutional neural network," *Scientific Reports*, vol. 10, pp. 2045–2322, 2020.

[20] I. E. Lagaris, A. Likas, and D. I. Fotiadis, "Artificial neural networks for solving ordinary and partial differential equations," *IEEE Transactions on Neural Networks*, vol. 9, pp. 987–1000, Sep. 1998.

[21] M. Raissi, P. Perdikaris, and G. Karniadakis, "Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations," *Journal of Computational Physics*, vol. 378, pp. 686 – 707, 2019.

[22] J. Sirignano and K. Spiliopoulos, "Dgm: A deep learning algorithm for solving partial differential equations," *Journal of Computational Physics*, vol. 375, pp. 1339 – 1364, 2018.

[23] J. Berg and K. Nyström, "A unified deep artificial neural network approach to partial differential equations in complex geometries," *Neurocomputing*, vol. 317, pp. 28–41, 2018.

[24] H. Guo, X. Zhuang, and T. Rabczuk, "A deep collocation method for the bending analysis of kirchhoff plate," *Computers, Materials and Continua*, vol. 59, pp. 433–456, 2 2019.

[25] C. Anitescu, E. Atroshchenko, N. Alajlan, and T. Rabczuk, "Artificial neural network methods for the solution of second order boundary value problems," *Computers, Materials and Continua*, vol. 59, pp. 345–359, 1 2019.

[26] H. Gao, L. Sun, and J.-X. Wang, "Phygeonet: Physics-informed geometry-adaptive convolutional neural networks for solving parameterized steady-state pdes on irregular domain," *Journal of Computational Physics*, vol. 428, p. 110079, 2021.

[27] E. Weinan and Bing Yu, "The deep ritz method: A deep learning-based numerical algorithm for solving variational problems," *Communications in Mathematics and Statistics*, vol. 6, pp. 1–12, Mar 2018.

[28] H. Wessels, C. Weißenfels, and P. Wriggers, "The neural particle method – an updated lagrangian physics informed neural network for computational fluid dynamics," *Computer Methods in Applied Mechanics and Engineering*, vol. 368, p. 113127, 2020.

[29] E. Samaniego, C. Anitescu, S. Goswami, V. Nguyen-Thanh, H. Guo, K. Hamdia, X. Zhuang, and T. Rabczuk, "An energy approach to the solution of partial differential equations in computational mechanics via machine learning: Concepts, implementation and applications," *Computer Methods in Applied Mechanics and Engineering*, vol. 362, p. 112790, 2020.

[30] V. M. Nguyen-Thanh, X. Zhuang, and T. Rabczuk, "A deep energy method for finite deformation hyperelasticity," *European Journal of Mechanics - A/Solids*, vol. 80, p. 103874, 2020.

[31] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in pytorch," 2017.

[32] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2017.

[33] D. C. Liu and J. Nocedal, "On the limited memory bfgs method for large scale optimization," *Mathematical Programming*, vol. 45, pp. 503–528, Aug 1989.

[34] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural Networks*, vol. 2, no. 5, pp. 359 – 366, 1989.

[35] M. Nielsen, "Neural Networks and Deep Learning," 2018.

[36] P. Jain and P. Kar, "Non-convex optimization for machine learning," *arXiv preprint arXiv:1712.07897*, 2017.

[37] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge university press, 2004.

[38] J. A. Cottrell, T. J. R. Hughes, and Y. Bazilevs, *Isogeometric Analysis*. John Wiley & Sons, Ltd, 2009.

[39] A. Li, A. B. Farimani, and Y. J. Zhang, "Deep learning of material transport in complex neurite networks," 2021.

[40] A. Li, X. Chai, G. Yang, and Y. J. Zhang, "An isogeometric analysis computational platform for material transport simulation in complex neurite networks," *Molecular & Cellular Biomechanics*, vol. 16, no. 2, pp. 123–140, 2019.

[41] S. Goswami, C. Anitescu, and T. Rabczuk, "Adaptive fourth-order phase field analysis using deep energy minimization," *Theoretical and Applied Fracture Mechanics*, vol. 107, p. 102527, 2020.

[42] S. Timoshenko and J. Goodier, *Theory of elasticity*. McGraw-Hill classic textbook reissue series, McGraw-Hill, 1969.

[43] R. Turner, D. Eriksson, M. McCourt, J. Kiili, E. Laaksonen, Z. Xu, and I. Guyon, "Bayesian optimization is superior to random search for machine learning hyperparameter tuning: Analysis of the black-box optimization challenge 2020," 2021.

[44] X. Xiao, M. Yan, S. Basodi, C. Ji, and Y. Pan, "Efficient hyperparameter optimization in deep learning using a variable length genetic algorithm," 2020.

[45] A. Koutsoukas, K. J. Monaghan, X. Li, and J. Huan, "Deep-learning: investigating deep neural networks hyper-parameters and comparison of performance to shallow methods for modeling bioactivity data," *Journal of Cheminformatics*, vol. 9, 2017.

[46] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *J. Mach. Learn. Res.*, vol. 13, pp. 281–305, 2012.

[47] P. Liashchynskyi and P. Liashchynskyi, "Grid search, random search, genetic algorithm: A big comparison for nas," 2019.

[48] R. Mindlin, "Second gradient of strain and surface-tension in linear elasticity," *International Journal of Solids and Structures*, vol. 1, no. 4, pp. 417 – 438, 1965.

[49] P. Casal, "La capillarite interne," *Cahier du Groupe Francais d'Etudes de Rheologie C.N.R.S*, vol. VI, pp. 31 – 37, 1961.

[50] P. Casal, "La theórie du second gradient et la capillarité.," *C.R. Acad. Sci. Paris A*, vol. 274, pp. 1571 – 1574, 1972.

[51] I. Vardoulakis and J. Sulem, *Bifurcation analysis in geomechanics.* London, Blackie Academic & Professional., 1995.

[52] I. Vardoulakis, G. Exadaktylos, and E. Aifantis, "Gradient elasticity with surface energy: mode-iii crack problem," *International Journal of Solids and Structures*, vol. 33, no. 30, pp. 4531 – 4559, 1996.

[53] G. Exadaktylos and I. Vardoulakis, "Surface instability in gradient elasticity with surface energy," *International Journal of Solids and Structures*, vol. 35, no. 18, pp. 2251 – 2281, 1998.

[54] G. Exadaktylos, "Gradient elasticity with surface energy: Mode-i crack problem," *International Journal of Solids and Structures*, vol. 35, no. 5, pp. 421 – 456, 1998.

[55] G. Exadaktylos, "Some basic half-plane problems of the cohesive elasticity theory with surface energy," *Acta Mechanica*, vol. 133, no. 5, pp. 175 – 198, 1999.

[56] B. S. Altan and E. C. Aifantis, "On Some Aspects in the Special Theory of Gradient Elasticity," *Journal of the Mechanical Behavior of Materials*, vol. 8, pp. 231–282, Sept. 1997.

[57] R. Mindlin and N. Eshel, "On first strain-gradient theories in linear elasticity," *International Journal of Solids and Structures*, vol. 4, no. 1, pp. 109 – 124, 1968.

[58] X.-L. Gao and S. Park, "Variational formulation of a simplified strain gradient elasticity theory and its application to a pressurized thick-walled cylinder problem," *International Journal of Solids and Structures*, vol. 44, no. 22, pp. 7486 – 7499, 2007.

[59] F. Zhuang, Z. Qi, K. Duan, D. Xi, Y. Zhu, H. Zhu, H. Xiong, and Q. He, "A comprehensive survey on transfer learning," 2020.