Copyright © 2019 Tech Science Press CMC, vol.59, no.2, pp.433-456, 2019

A Deep Collocation Method for the Bending Analysis of Kirchhoff Plate

Hongwei Guo[3], Xiaoying Zhuang[3, 4, 5] and Timon Rabczuk[1, 2, *]

Abstract: In this paper, a deep collocation method (DCM) for thin plate bending problems is proposed. This method takes advantage of computational graphs and backpropagation algorithms involved in deep learning. Besides, the proposed DCM is based on a feedforward deep neural network (DNN) and differs from most previous applications of deep learning for mechanical problems. First, batches of randomly distributed collocation points are initially generated inside the domain and along the boundaries. A loss function is built with the aim that the governing partial differential equations (PDEs) of Kirchhoff plate bending problems, and the boundary/initial conditions are minimised at those collocation points. A combination of optimizers is adopted in the backpropagation process to minimize the loss function so as to obtain the optimal hyperparameters. In Kirchhoff plate bending problems, the C^1 continuity requirement poses significant difficulties in traditional mesh-based methods. This can be solved by the proposed DCM, which uses a deep neural network to approximate the continuous transversal deflection, and is proved to be suitable to the bending analysis of Kirchhoff plate of various geometries.

Keywords: Deep learning, collocation method, Kirchhoff plate, higher-order PDEs.

1 Introduction

Thin plates are widely employed as basic structural components in engineering fields [Ventsel and Krauthammer (2001)], which combines light weight, efficient load-carrying capacity, economy with technological effectiveness. Their mechanical behaviours have long been studied by various methods such as finite element method [Bathe (2006); Hughes (2012); Zhuang, Huang, Zhu et al. (2013)], boundary element method [Katsikadelis (2016); Brebbia and Walker (2016)], meshfree method [Nguyen, Rabczuk, Bordas et al. (2008)], isogeometric analysis [Nguyen, Anitescu, Bordas et al. (2015)], and numerical manifold method [Zheng, Liu and Ge (2013); Guo and Zheng (2018); Guo,

[1] Division of Computational Mechanics, Ton Duc Thang University, Ho Chi Minh City, Viet Nam.

[2] Faculty of Civil Engineering, Ton Duc Thang University, Ho Chi Minh City, Viet Nam.

[3] Institute of Continuum Mechanics, Leibniz Universität Hannover, Hannover, Germany.

[4] Department of Geotechnical Engineering, Tongji University, Shanghai, China.

[5] Key Laboratory of Geotechnical and Underground Engineering of Ministry of Education, Tongji University, Shanghai, China.

[*] Corresponding Author: Timon Rabczuk. Email: timon.rabczuk@tdtu.edu.vn.

Zheng and Zhuang (2019)]. The Kirchhoff bending problem is a classical fourth-order problem, its mechanical behaviour is described by fourth-order partial differential equation which poses difficulties to construct a shape function to be globally C^1 continuous but piecewise C^2 continuous, namely, H^2 regular, for those mesh-based numerical method. However, according to the universal approximation theorem, see Cybenko [Cybenko (1989)] and Hornic [Hornik (1991)], any continuous function can be approximated arbitrarily well by a feedforward neural network, even with a single hidden layer, which offers a new possibility of analyzing Kirchhoff plate bending problems.

Deep learning was first brought up as a new branch of machine learning in the realm of artificial intelligence in 2006, which uses deep neural networks to learn features of data with high-level of abstractions [LeCun, Bengio and Hinton (2015)]. The deep neural networks adopt artificial neural network architectures with various hidden layers, which exponentially reduce the computational cost and amount of training data in some applications [Al-Aradi, Correia, Naiff et al. (2018a)]. The major two desirable traits of deep learning lie in the nonlinear processing in multiple hidden layers in supervised or unsupervised learning [Vargas, Mosavi and Ruiz (2018)]. Several types of deep neural networks such as convolutional neural networks (CNN) and recurrent/recursive neural networks (RNN) [Patterson and Gibson (2017)] have been created, which further boost the application of deep learning in image processing [Yang, MacEachren, Mitra et al. (2018)], object detection [Zhao, Zheng, Xu et al. (2019)], speech recognition [Nassif, Shahin, Attili et al. (2019)] and many other domains including genomics [Yue and Wang (2018)] and even finance [Fischer and Krauss (2018)].

As a matter of fact, artificial neural networks (ANN) which are main tools in deep learning have been around since the 1940's [McCulloch and Pitts (1943)] but have not performed well until recently. They only became a major part of machine learning in the past few decades due to strides in computing techniques and explosive growth in date collection and availability, especially the arrival of backpropagation technique and advance in deep neural networks. However, based on the function approximation capabilities of feed forward neural networks, ANN were adopted to solve partial differential equations (PDEs) [Lagaris, Likas and Fotiadis (1998); Lagaris, Likas and Papageorgiou (2000); McFall and Mahan (2009)], which results in a solution that can be described by a closed analytical form. Due to vanishing gradients, neural networks with many hidden layers require a long time for training. However, pretraining, which sets the initial values of connection weights and biases, with back propagation algorithm are now proposed to solve this problem more efficiently. More recently, with improved theory incorporating unsupervised pre-training, stacks of auto-encoder variants, and deep belief nets, deep learning has become a central and popular branch in research and applications.

Some researchers employed deep learning for the solution of PDEs. Mills et al. deployed a deep conventional neural network to solve schrödinger equation, which directly learned the mapping between potential and energy [Mills (2017)]. E et al. applied deep learning-based numerical methods for high-dimensional parabolic PDEs and back-forward stochastic differential equations, which was proven to be efficient and accurate for 100-dimensional nonlinear PDEs [E, Han and Jentzen (2017); Han, Jentzen, and E (2018)]. Also, E and Yu proposed a Deep Ritz method for solving variational problems arising from partial differential equations [E and Yu (2018)]. Raissi et al. applied the probabilistic machine learning in solving linear and nonlinear differential equations using Gaussian Processes and later introduced a data-driven Numerical Gaussian Processes to solve time-dependent and nonlinear PDEs, which circumvented the need for spatial discretization [Raissi, Perdikaris and Karniadakis (2017a); Raissi and Karniadakis (2018); Raissi, Perdikaris and Karniadakis (2018)]. Later, Raissi et al. [Raissi, Perdikaris and Karniadakis (2017b, 2017c); Raissi, Perdikaris and Karniadakis (2019)] introduced a physical informed neural networks for supervised learning of nonlinear partial differential equations from Burger's equations to Navier-Stokes equations. Two distinct models were tailored for spatio-temporal datasets: continuous time and discrete time models. They also applied a deep neural networks in solving coupled forward-backward stochastic differential equations and their corresponding high-dimensional PDEs [Raissi (2018)]. Beck et al. [Beck, Becker, Grohs et al. (2018); Beck, E and Jentzen (2019)] studied the deep learning in solving stochastic differential equations and Kolmogorov equations. Nabian and Meidani studied the presentation of high-dimensional random partial differential equations with a feed-forward fully-connected deep neural networks [Nabian and Meidani (2018a, 2018b)]. Based on the physics informed deep neural networks, Tartakovsky et al. studied the estimation of parameters and unknown physics in PDE models [Tartakovsky, Marrero, Perdikaris et al. (2018)]. Qin et al. applied the deep residual network and observation data to approximate unknown governing differential equations [Qin, Wu and Xiu (2018)]. Sirignano et al. [Sirignano and Spiliopoulos (2018)] gave a theoretic motivation of using deep neural networks as PDE approximators, which converged as the number of hidden layers tend to infinity. Based on this, a deep Galerkin method was tested to solve PDEs including high-dimensional ones. Berg et al. [Berg and Nyström (2018)] proposed a unified deep neural network approach to approximate solutions to PDEs and then used deep learning to discover PDEs hidden in complex data sets from measurement data [Berg and Nyström (2019)]. In general, a deep feed-forward neural networks can serve as a suitable solution approximators, especially for high-dimensional PDEs with complex domains.

Some researchers studied the surrogate of FEM by deep learning, which mainly trains the deep neural networks from datasets obtained from FEM. For instance, Liang et al. [Liang, Liu, Martin et al. (2017); Liang, Liu, Martin et al. (2018)] investigated the relationship between geometric features of aorta and FEM-predicted ascending aortic aneurysm rupture risk and then a deep learning was used to estimate the stress distribution of the aorta. In this research, we will not confine deep learning application within FEM datasets. Rather, the deflection of Kirchhoff plate is first approximated with deep physical informed feedforward neural networks with hyperbolic tangent activation functions and trained by minimizing a loss function related to the governing equation of Kirchhoff bending problems and related boundary conditions hence avoiding a FEM discretization entirely. The training data for deep neural networks are obtained by randomly distributed collocation points from the physical domain of the plate. This deep collocation method can be seen as a truly mesh-free method without the need of background grids.

The paper is organized as follows: First a brief introduction of Kirchhoff plate bending strong form with typical boundary conditions is given. Then we give a short introduction to deep learning. Finally, the deep collocation method with varying hidden layers and neurons are adopted for plates with various shapes and boundary conditions to show the performance of the proposed method.

2 Kirchhoff plate bending

Based on Kirchhoff plate bending theory [Ventsel and Krauthammer (2001)], the relation between lateral deflection $w(x,y)$ of the middle surface $(z=0)$ and rotations about the x, y -axis can be given by:

$$\theta_x = \frac{\partial w}{\partial x}, \quad \theta_y = \frac{\partial w}{\partial y} \tag{1}$$

Under the coordinate system shown in Fig. 1, the displacement field in a thin plate can be expressed as:

$$\begin{aligned} u(x,y,z) &= -z\frac{\partial w}{\partial x}, \\ v(x,y,z) &= -z\frac{\partial w}{\partial y}, \\ w(x,y,z) &= w(x,y). \end{aligned} \tag{2}$$

It is obviously that the transversal deflection of the middle plane of the thin plate can be regard as the field variables of the bending problem of thin plates. The corresponding bending and twist curvatures are the generalized strains:

$$k_x = -\frac{\partial^2 w}{\partial x^2}, \ k_y = -\frac{\partial^2 w}{\partial y^2}, \ k_{xy} = -2\frac{\partial^2 w}{\partial xy}. \tag{3}$$

Therefore, the geometric equations of Kirchhoff bending can be expressed as:

$$\boldsymbol{k} = \begin{Bmatrix} k_x \\ k_y \\ k_{xy} \end{Bmatrix} = -\begin{Bmatrix} \frac{\partial^2 w}{\partial x^2} \\ \frac{\partial^2 w}{\partial y^2} \\ 2\frac{\partial^2 w}{\partial x \partial y} \end{Bmatrix} = \boldsymbol{L}w, \tag{4}$$

with $\boldsymbol{L}$ being the differential operator defined as $\boldsymbol{L} = -\left\{\frac{\partial^2}{\partial x^2} \ \frac{\partial^2}{\partial y^2} \ 2\frac{\partial^2}{\partial x \partial y}\right\}^T$.

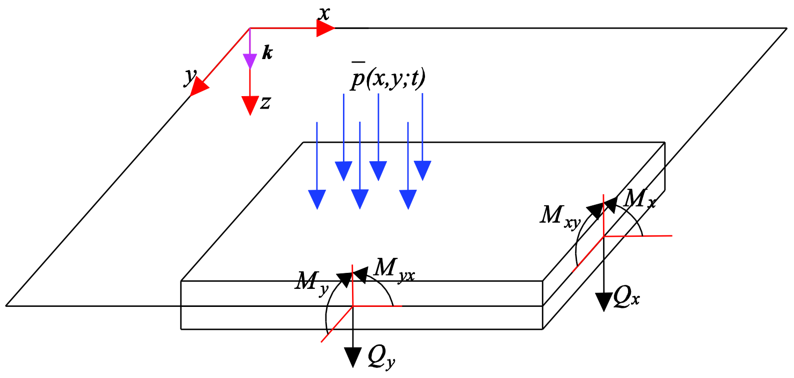


Figure 1: Internal force in the coordinate system

Accordingly, the bending and twisting moments, shown in Fig. 1 can be obtained as:

$$M_x = -D_0\left(\frac{\partial^2 w}{\partial x^2} + \upsilon\frac{\partial^2 w}{\partial y^2}\right),$$
$$M_y = -D_0\left(\upsilon\frac{\partial^2 w}{\partial x^2} + \frac{\partial^2 w}{\partial y^2}\right), \quad (5)$$
$$M_{xy} = M_{yx} = -D_0(1-\nu)\frac{\partial^2 w}{\partial xy}.$$

$D_0 = \frac{Eh^3}{12(1-\nu^2)}$ is the bending rigidity, where E and ν are the Young's modulus and Poisson ratio, and h is the thickness of the thin plate. For isotropic thin plate, the constitutive equation can be expressed in Matrix form:

$$\boldsymbol{M} = \boldsymbol{Dk}. \quad (6)$$

The shear forces can be obtained in terms of the generalized stress components:

$$Q_x = \frac{\partial M_x}{\partial x} + \frac{\partial M_{xy}}{\partial y},\ Q_y = \frac{\partial M_{xy}}{\partial x} + \frac{\partial M_y}{\partial y}. \quad (7)$$

The differential equation for the deflections for thin plate based on Kirchhoff's assumptions can be expressed by transversal deflection as:

$$\nabla^2(\nabla^2 w) = \nabla^4 w = \frac{p}{D}, \quad (8)$$

where $\nabla^4() = \frac{\partial^4}{\partial x^4} + \frac{\partial^4}{\partial x^2 y^2} + \frac{\partial^4}{\partial y^4}$ is commonly called biharmonic operator.

Consequently, the Kirchhoff plate bending problems can be boiled down to a fourth order PDE problem, which pose difficulty for tradition mesh-based method in constructing a shape function to be H^2 regular. Moreover, the boundary conditions of Kirchhoff plate taken into consideration in this paper can be generally classified into three parts, namely,

$$\partial\Omega = \Gamma_1 + \Gamma_2 + \Gamma_3, \quad (9)$$

For clamped edge boundary Γ_1, $w = \widetilde{w}$, $\frac{\partial w}{\partial n} = \tilde{\theta}_n$. $\widetilde{w}$ and $\tilde{\theta}_n$ are functions of arc length.

For simply supported edge boundary Γ_2, $w = \widetilde{w}$, $M_n = \widetilde{M}_n$. $\widetilde{M}_n$ is the function of arc length, too.

For free edge boundary Γ_3, $M_n = \widetilde{M}_n$, $\frac{\partial M_{ns}}{\partial s} + Q_n = \tilde{q}$, where $\tilde{q}$ is a load exerted along this boundary.

It should be noted that $\boldsymbol{n}$ and $\boldsymbol{s}$ here refer to the normal and tangent directions along the boundaries.

3 Deep collocation method for solving Kirchhoff plate bending

3.1 Feed forward neural network

The basic architecture of a fully connected feedforward neural network is shown in Fig. 2. It comprises of multiple layers: input layer, one or more hidden layers and output layer. Each layer consists of one or more nodes called neurons, shown in the Fig. 2 by small coloured circles, which are the basic units of computation. For an interconnected structure, every two neurons in neighbouring layers have a connection, which is represented by a connection weight, see Fig. 2, where the weight between neuron k in hidden layer $l-1$ and neuron j in hidden layer l is denoted by ω_{jk}^{l}. No connection exists among neurons in the same layer as well as in the non-neighbouring layers. Input data, defined from x_1 to x_N, flow through this neural network via connections between neurons, starting from the input layer, through hidden layer $l-1$, l, to the output layer, which eventually outputs data from y_1 to y_M. The feedforward neural network defines a mapping FNN: $R^N \to R^M$.

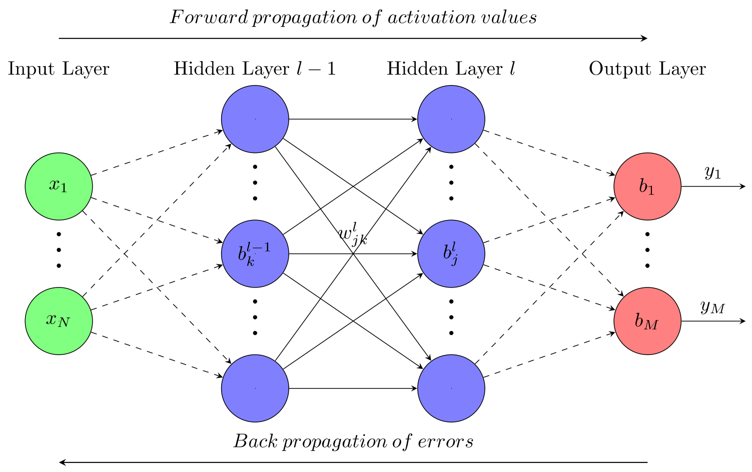


Figure 2: Architecture of a fully connected feedforward back-propagation neural network

However, it should be noted that the number of neurons on each hidden layers and number of hidden layers can be arbitrarily chosen and are invariably determined through a trial and error procedure. It has also been concluded that any continuous function can be approximated with any desired precision by a feed forward neural network with even a single hidden layer [Funahashi (1989); Hornik, Stinchcombe and White (1989)].

On each neuron in the feed forward neural network, a bias is supplied including neurons in the output layer except the neurons in the input layer, which is defined by b_j^l for bias of neuron j in layer l. Furthermore, the activation function is defined for an output of each neuron in order to introduce a non-linearity into the neural network and make the back-propagation possible where gradients are supplied along with an error to update weights and biases. The activation function in layer l will be denoted by σ here. There are many activation functions such as sigmoid function, hyperbolic tangent function *Tanh*, Rectified linear units *Re*lu, to name a few. Suggestions upon their choice can be found in [Hayou, Doucet and Rousseau (2018)]. Hence, for the value on each neuron in the hidden layers and output layer adds the weighted sum of values of output values from the previous layer with corresponding connection weights to basis on the neuron. An intermediate quantity for neuron j on hidden layer l is defined as

$$a_j^l = \sum_k \omega_{jk}^l y_k^{l-1} + b_j^l, \tag{10}$$

and its output is given by the activation of the above weighted input

$$y_j^l = \sigma(a_j^l) = \sigma\left(\sum_k \omega_{jk}^l y_k^{l-1} + b_j^l\right), \tag{11}$$

where y_k^{l-1} is the output from the previous layer.

When Eq. (11) is applied to compute y_j^l, the intermediate quantity a_j^l is calculated 'along the way'. This quantity turns out to be useful and named here as weighted input to neuron j on hidden layer l. Eq. (10) can be written in a compact matrix form, which calculates the weighted inputs for all neurons on certain layers efficiently, obtaining:

$$\boldsymbol{a} = \boldsymbol{W}^l \boldsymbol{y}^{l-1} + \boldsymbol{b}^l, \tag{12}$$

Accordingly, from Eq. (12) $\boldsymbol{y}^l = \sigma(\boldsymbol{a})$, where activation functions are applied elementwise. A feedforward network thus defines a function $f(\boldsymbol{x};\theta)$ depending on the input data $\boldsymbol{x}$ and parametrized by θ consisting of weights and biases in each layer. The defined function provides an efficient way to approximate unknown field variables.

3.2 Backpropagation

Backpropagation $(backward\,propagation)$ is an important and computationally efficient mathematical tool to compute gradients in deep learning [Nielsen (2018)]. Essentially, backpropagation is based on recursively applying the chain rule and deciding which computations can be run in parallel from computational graphs. In our problem, the governing equation contains fourth order partial derivatives of field variable $w(\boldsymbol{x})$ approximated by the deep neural networks $f(\boldsymbol{x};\theta)$. For the approximation defined by $f(\boldsymbol{x};\theta)$, in order to find the weights and biases, a loss function $\boldsymbol{L}(f,w)$ is defined to be minimized [Janocha and Czarnecki (2017)]. The backpropagation algorithm for computing the gradient of this loss function $\boldsymbol{L}(f,w)$ can be defined as follows [Nielsen (2018)]:

- **Input:** Input dataset $x^1,\cdots,x^n$, prepare activation y^1 for input layer;
- **Feedforward:** For each layer $l = 2,3,\cdots,L$, compute $a^l = \sum_k W^l y^{l-1} + b^l$, and $\sigma(a^l)$;
- **Output error:** Compute the error $\delta^L = \nabla_{y^L}\boldsymbol{L} \odot \sigma_L'(a^L)$, σ_L' measures how fast σ changes at a^L;
- **Backpropagation error:** For each $l = L-1, L-2,\cdots,2$, compute $\delta^l = \left(\left(W^{l+1}\right)^T \delta^{l+1}\right) \odot \sigma_l'(a^l)$;
- **Output**: The gradient of the loss function is given by $\frac{\partial \boldsymbol{L}}{\partial \omega_{jk}^l} = y_k^{l-1}\delta_j^l$ and $\frac{\partial \boldsymbol{L}}{\partial b_j^l} = \delta_j^l$.

Here, $\odot$ denotes the Hadamard product.

There are lists of deep learning tools to setup the training such as Pytorch or Tensorflow. The former inputs a numerical value and then computes the derivatives at this node, while the latter computers the derivatives of a symbolic variable, then stores the derivative operations into new nodes added to the graph for later use. Obviously, the latter is more advantageous in computing higher-order derivatives, which can be computed from its extended graph by running backpropagation repeatedly. Therefore, since the fourth-order derivatives of field variables is needed to be computed, the Tensorflow framework is adopted [Al-Aradi, Correia, Naiff et al. (2018b)].

3.3 Formulation of deep collocation method

The formulation of a deep collocation in solving Kirchhoff plate bending problems is introducted in this section. Collocation method is a widely used method seeking numerical solutions for ordinary, partial differential and integral equations [Atluri (2005)]. It is a popular method for trajectory optimization in control theory. A set of randomly distributed points (also known as collocation points) is often deployed to represent a desired trajectory that minimizes the loss function while satisfying a set of constraints. The collocation method tends to be relatively insensitive to instabilities (such as blowing/vanishing gradients with neural networks) and is a viable way to train the deep neural networks [Agrawal].

Eqs. (8), (9)-the Kirchhoff plate bending problem-can be boiled down to the solution of a fourth order biharmonic equations with boundary constraints. Thus we first discretize the physical domain with collocation points denoted by $\boldsymbol{x}_\Omega = \left(x_1, \cdots, x_{N_\Omega}\right)^T$. Another set of collocation points is employed to discretize the boundary conditions denoted by $\boldsymbol{x}_\Gamma = \left(x_1, \cdots, x_{N_\Gamma}\right)^T$. Then the transversal deflection w is approximated with the aforementioned deep feedforward neural network $w^h(\boldsymbol{x};\theta)$. A loss function can thus be constructed to find the optimal hyperparameters by minimizing governing equation with boundary conditions approximated by $w^h(\boldsymbol{x};\theta)$. The mean squared error loss form is adopted here.

Substituting $w^h(\boldsymbol{x}_\Omega;\theta)$ into Eq. (8), we obtain

$$G(\boldsymbol{x}_\Omega;\theta) = \nabla^4 w^h(\boldsymbol{x}_\Omega;\theta) = \frac{p}{D}, \tag{13}$$

which results in a physical informed deep neural network $G(\boldsymbol{x}_\Omega;\theta)$.

The boundary conditions illustrated in Section 2 can also be expressed by the neural network approximation $w^h(\boldsymbol{x}_\Omega;\theta)$ as:

On Γ_1, we have

$$w^h\left(\boldsymbol{x}_{\Gamma_1};\theta\right) = \widetilde{w},\ \frac{\partial w^h(\boldsymbol{x}_{\Gamma_1};\theta)}{\partial \mathrm{n}} = \tilde{\theta}_n. \tag{14}$$

On Γ_2,

$$w^h\left(\boldsymbol{x}_{\Gamma_2};\theta\right) = \widetilde{w},\ M_n\left(\boldsymbol{x}_{\Gamma_2};\theta\right) = \widetilde{M}_n. \tag{15}$$

where $M_n\left(\boldsymbol{x}_{\Gamma_2};\theta\right)$ can be obtained from Eq. (5) by combing $w^h\left(\boldsymbol{x}_{\Gamma_1};\theta\right)$.

On Γ_3,

$$M_n\left(\boldsymbol{x}_{\Gamma_3};\theta\right) = \widetilde{M}_n,\ \frac{\partial M_{ns}(\boldsymbol{x}_{\Gamma_3};\theta)}{\partial s} + Q_n\left(\boldsymbol{x}_{\Gamma_3};\theta\right) = \tilde{q}, \tag{16}$$

where $M_{ns}\left(\boldsymbol{x}_{\Gamma_3};\theta\right)$ can be obtained from Eq. (5) and $Q_n\left(\boldsymbol{x}_{\Gamma_3};\theta\right)$ can be obtained from Eq. (7) by combing $w^h\left(\boldsymbol{x}_{\Gamma_3};\theta\right)$.

It should be noted that $\boldsymbol{n}$, $\boldsymbol{s}$ here refer to the normal and tangent directions along the boundaries. Note the induced physical informed neural network $G(\boldsymbol{x};\theta)$, $M_n(\boldsymbol{x};\theta)$, $M_{ns}(\boldsymbol{x};\theta)$, $Q_n(\boldsymbol{x};\theta)$ share the same parameters as $w^h(\boldsymbol{x};\theta)$. Considering the generated collocation points in domain and on boundaries, they can all be learned by minimizing the mean square error loss function:

$$L(\theta) = MSE = MSE_{\mathrm{G}} + MSE_{\Gamma_1} + MSE_{\Gamma_2} + MSE_{\Gamma_3}, \tag{17}$$

with

$$MSE_{\mathrm{G}} = \frac{1}{N_\Omega}\sum_{i=1}^{N_\Omega}\|G(\boldsymbol{x}_\Omega;\theta)\|^2 = \frac{1}{N_\Omega}\sum_{i=1}^{N_\Omega}\left\|\nabla^4 w^h(\boldsymbol{x}_\Omega;\theta) - \frac{p}{D}\right\|^2,$$

$$MSE_{\Gamma_1} = \frac{1}{N_{\Gamma_1}}\sum_{i=1}^{N_{\Gamma_1}}\left\|w^h(\boldsymbol{x}_{\Gamma_1};\theta) - \widetilde{w}\right\|^2 + \frac{1}{N_{\Gamma_1}}\sum_{i=1}^{N_{\Gamma_1}}\left\|\frac{\partial w^h(x_{\Gamma_1};\theta)}{\partial \mathrm{n}} - \widetilde{\theta}_n\right\|^2,$$

$$MSE_{\Gamma_2} = \frac{1}{N_{\Gamma_2}}\sum_{i=1}^{N_{\Gamma_2}}\left\|w^h(\boldsymbol{x}_{\Gamma_2};\theta) - \widetilde{w}\right\|^2 + \frac{1}{N_{\Gamma_2}}\sum_{i=1}^{N_{\Gamma_2}}\left\|M_n(\boldsymbol{x}_{\Gamma_2};\theta) - \widetilde{M}_n\right\|^2, \tag{18}$$

$$MSE_{\Gamma_3} = \frac{1}{N_{\Gamma_3}}\sum_{i=1}^{N_{\Gamma_3}}\left\|M_n(\boldsymbol{x}_{\Gamma_3};\theta) - \widetilde{M}_n\right\|^2 + \frac{1}{N_{\Gamma_3}}\sum_{i=1}^{N_{\Gamma_3}}\left\|\frac{\partial M_{ns}(x_{\Gamma_3};\theta)}{\partial \mathrm{s}} + Q_n(\boldsymbol{x}_{\Gamma_3};\theta) - \widetilde{q}\right\|^2.$$

where $\boldsymbol{x}_\Omega \in R^N$, $\theta \in R^K$are the neural network parameters. $L(\theta) = 0$, $w^h(\boldsymbol{x};\theta)$ is then a solution to transversal deflection. Our goal is to find the a set of parameters θ that the approximated deflection $w^h(\boldsymbol{x};\theta)$ minimizes the loss $L(\theta)$. If $L(\theta)$ is a very small value, the approximation $w^h(\boldsymbol{x};\theta)$ is very closely satisfying governing equations and boundary conditions, namely

$$w^h = \underset{\theta\in R^K}{argmin}\, L(\theta), \tag{19}$$

The solution of thin plate bending problems by deep collocation method can be reduced to an optimization problem. In the deep learning Tensorflow framework, a variety of optimizers are available. One of the most widely used optimization methods is the Adam optimization algorithm [Kingma and Ba (2015)], which is also adopted in the numerical study in this paper. The idea is to take a descent step at collocation point $\boldsymbol{x}_i$ with Adam-based learning rates α_i,

$$\theta_{i+1} = \theta_{i+1} + \alpha_i \nabla_\theta L(\boldsymbol{x}_i;\theta), \tag{20}$$

and then the process in Eq. (20) is repeated until a convergence criterion is satisfied.

4 Numerical examples

In this section, several numerical examples on plate bending problems with various shapes and boundary conditions are studied. A combined optimizer suggested by Berg et al. [Berg and Nyström (2018)] is adopted using L-BFGS optimizer [Liu and Nocedal (1989)] first and in linear search where BFGS may fail, an Adam optimizer is then applied with a very small learning rate. For all numerical examples, predicted maximum transverse with increasing layers are studied in order to show the convergence of deep collocation method in solving the plate bending problem.

4.1 Simply-supported square plate

A simply-supported square plate under a sinusoidal distribution of transverse loading is first studied. The distributed load is given by.

$$p = \frac{p_0}{D} sin\left(\frac{\pi x}{a}\right) sin\left(\frac{\pi y}{b}\right), \tag{21}$$

Here, a, b indicate the length of the plate; D denotes the flexural stiffness of the plate depending on the plate thickness and material properties. The exact solution for this problem is given by

$$w = \frac{p_0}{\pi^4 D\left(\frac{1}{a^2}+\frac{1}{b^2}\right)^2} sin\left(\frac{\pi x}{a}\right) sin\left(\frac{\pi y}{b}\right), \tag{22}$$

w represents the transverse plate deflection. For this numerical example, we first generate 1000 randomly distributed collocation points in the physical domain depicted in Fig. 3. We thoroughly studied the influence of deep neural network with a varying number of hidden layer and neurons on the maximum deflection at the center of the plate, which is shown in Tab. 1. The numerical results are compared with the exact solution. It is clear that the results predicted by more hidden layers are more desirable, especially for neural networks with three hidden layers. To better reflect the deflection vector in the whole physical domain, the contour plot, contour error plot of deflection for increasing hidden layers with 50 neurons are shown in Fig. 5, Fig. 6, Fig. 7.

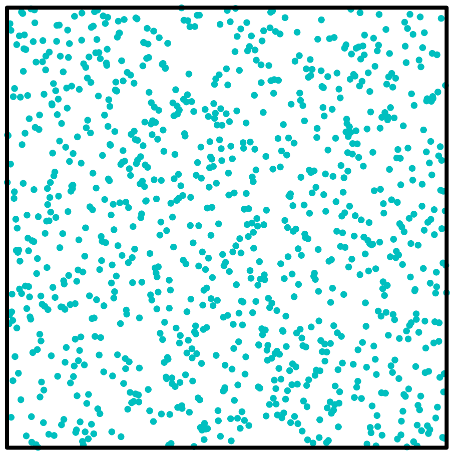


Figure 3: Collocation points discretize the square domain

We employed a varying number of hidden layers from 1 to 4 and in each layer and the number of neurons varies from 20 to 60, see Tab. 1. We calculated the corresponding maximum transversal deflection at the center of the square plate. The L^2 relative error of deflection vector at all predicted points is shown in Fig. 4 for each case. Even the neural network with only one single hidden layer with 20 neurons gives very accurate results. With increasing neurons and hidden layers, the results converge to the exact solution and the results are very accurate even with a few neurons and a single hidden layer. In Fig. 4, all three hidden layer types get very accurate results. Though the single layer with 20 neurons is the most accurate in all three types with 20 neurons, the magnitude of all is 10^{-4}. As the number of hidden layers and neurons increases, the relative error flattens.

Table 1: Maximum deflection predicted by deep collocation method

Simply-supported Square Plate	Predicted Maximum Deflection	Exact Maximum Deflection
1 hidden layers, 20 neurons	2.566529	
1 hidden layers, 30 neurons	2.566556	
1 hidden layers, 40 neurons	2.566541	
1 hidden layers, 50 neurons	2.566576	
1 hidden layers, 60 neurons	2.566509	
2 hidden layers, 20 neurons	2.566501	
2 hidden layers, 30 neurons	2.566429	
2 hidden layers, 40 neurons	2.566329	2.566496
2 hidden layers, 50 neurons	2.566442	
2 hidden layers, 60 neurons	2.566420	
3 hidden layers, 20 neurons	2.566366	
3 hidden layers, 30 neurons	2.566518	
3 hidden layers, 40 neurons	2.566434	
3 hidden layers, 50 neurons	2.566468	
3 hidden layers, 60 neurons	2.566549	

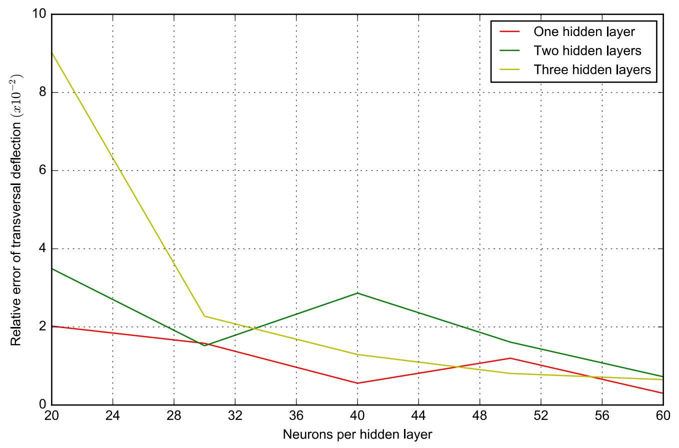
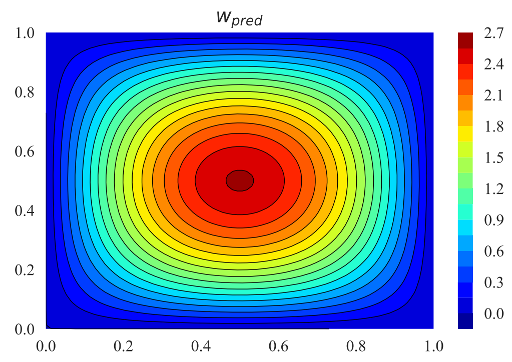
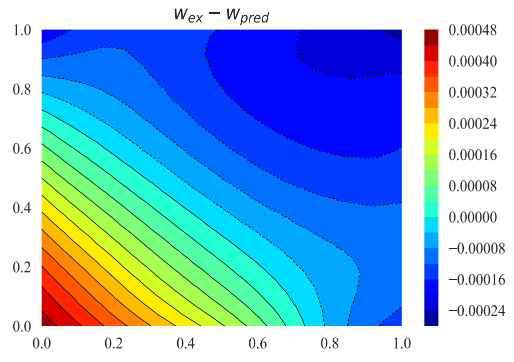


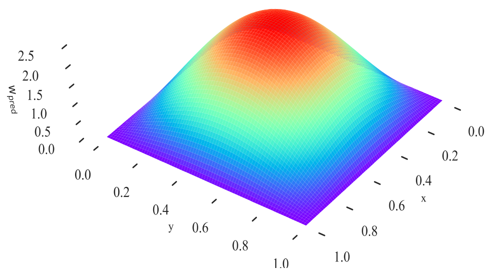
Figure 4: The relative error of deflection with varying hidden layers and neurons



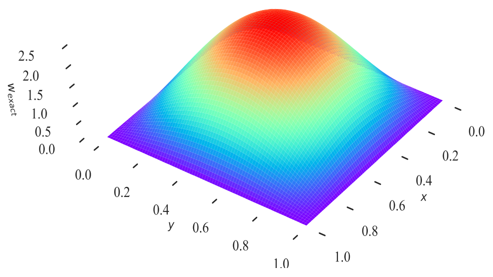
(a). Predicted deflection contour



(b). Deflection error contour

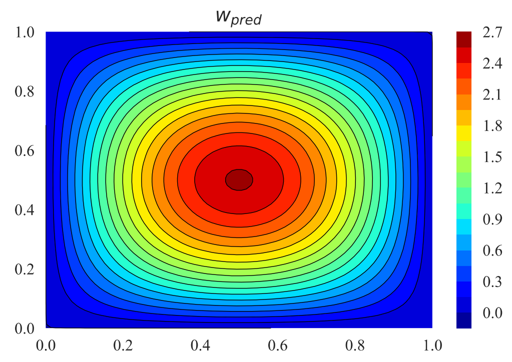


(c). Predicted deflection contour

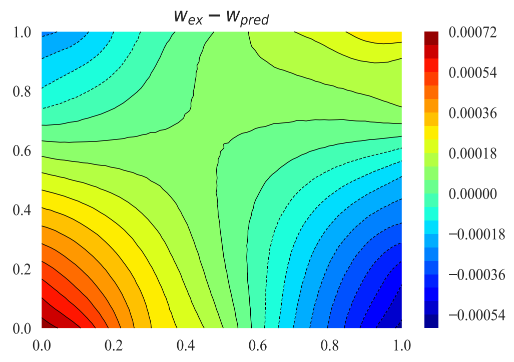


(d). Deflection error contour

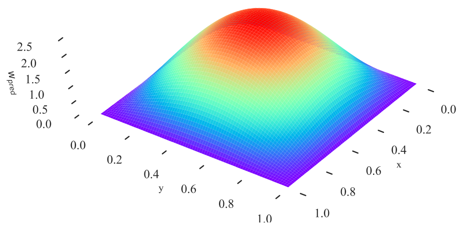
Figure 5: (a) Predicted deflection contour (b) Deflection error contour (c) Predicted deflection (d) Exact deflection of the simply-supported square plate with 1 hidden layer and 50 neurons with varying hidden layers and neurons



(a). Predicted deflection contour



(b). Deflection error contour


(c). Predicted deflection contour

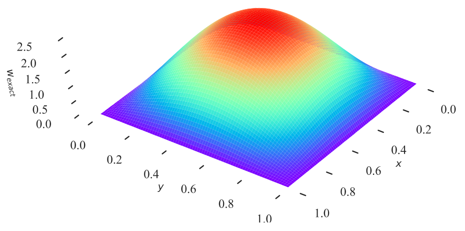
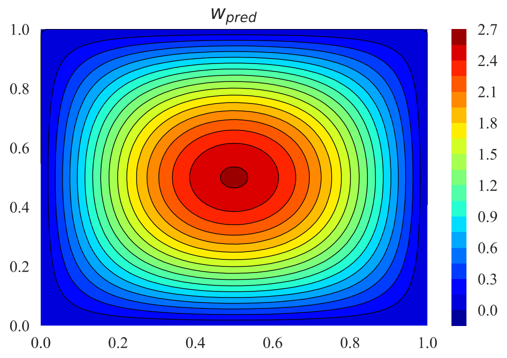
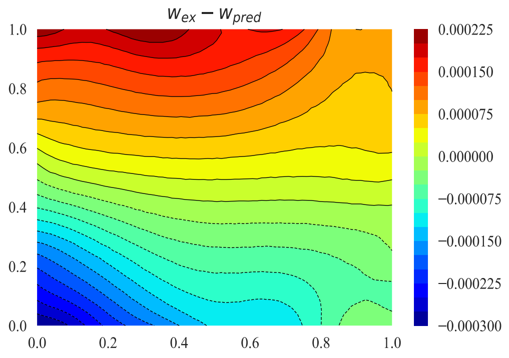

(d). Deflection error contour

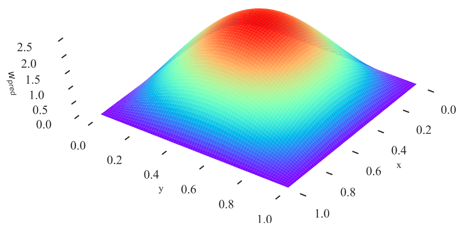
Figure 6: (a) Predicted deflection contour (b) Deflection error contour (c) Predicted deflection (d) Exact deflection of the simply-supported square plate with 2 hidden layers and 50 neurons with varying hidden layers and neurons



(a). Predicted deflection contour



(b). Deflection error contour


(c). Predicted deflection contour

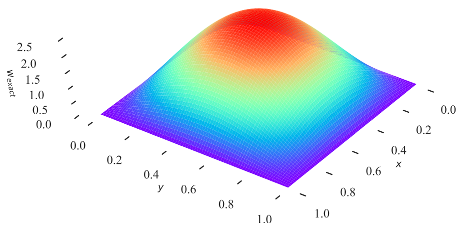

(d). Deflection error contour

Figure 7: (a) Predicted deflection contour (b) Deflection error contour (c) Predicted deflection (d) Exact deflection of the simply-supported square plate with 3 hidden layers and 50 neurons with varying hidden layers and neurons

From Fig. 5, Fig. 6, Fig. 7, we can observe that the deflection obtained by the deep collocation method agrees well with the exact solutions. As the hidden layer number increases, the numerical results converge to the exact solutions in the whole square plate.

The predicted plate deformation agrees well with the exact deformation. The advantages of neural networks with hidden layers are not conspicuously reflected in this numerical example, as the next numerical example shows more clearly.

4.2 Clamped square plate

Next, a clamped square plate under a uniformly distributed transverse loading is analyzed with deep collocation method. No exact solution for the deflection in the whole plate is available. Therefore, a solution obtained by the Galerkin method [Khan, Tiwari and Ali (2012)] is adopted as a comparison:

$$\begin{bmatrix} a_{11} \\ a_{12} \\ a_{21} \\ a_{22} \end{bmatrix} = \frac{b^4 p}{D} \begin{bmatrix} 0.318682766 \\ 0.038459815 \\ 0.038459815 \\ 0.008281438 \end{bmatrix}, \tag{23}$$

$$\begin{aligned} w = \frac{b^4 p}{D} \Big\{ & a_{11} \left(1-\frac{x}{a}\right)^2 \left(1-\frac{y}{b}\right)^2 \left(\frac{x}{a}\right)^2 \left(\frac{y}{b}\right)^2 + \\ & a_{12} \left(1-\frac{x}{a}\right)^2 \left(\frac{y}{b}-\frac{y^2}{b^2}\right)^2 \left(\frac{x}{a}\right)^2 \left(\frac{y}{b}\right)^2 \Big\} + \\ \frac{b^4 p}{D} \Big\{ & a_{21} \left(\frac{x}{a}-\frac{x^2}{a^2}\right)^2 \left(1-\frac{y}{b}\right)^2 \left(\frac{x}{a}\right)^2 \left(\frac{y}{b}\right)^2 + \\ & a_{22} \left(\frac{x}{a}-\frac{x^2}{a^2}\right)^2 \left(\frac{y}{b}-\frac{y^2}{b^2}\right)^2 \left(\frac{x}{a}\right)^2 \left(\frac{y}{b}\right)^2 \Big\}, \end{aligned} \tag{24}$$

For the maximum transversal deflection at the center of an isotropic square plate, the Ritz method gives the maximum deflection at the center as $w_{max} = 0.00133 \frac{b^4 p}{D}$ [Khan, Tiwari and Ali (2012)], and Timoshenko et al. [Timoshenko and Woinowsky-Krieger (1959)] gave an exact solution $w_{max} = 0.00126 \frac{b^4 p}{D}$. Here, D denotes the flexural stiffness of the plate and depends on the plate thickness and material properties; a, b indicate the length dimension of the plate. 1000 randomly generated collocation points as in Fig. 3 are used to discretize the clamped square plate.

For this clamped case, a deep feedforward neural network with increasing layers and neurons is studied in order to validate the convergence of this scheme. First, the maximum central deflection shown in Tab. 2 is calculated for different number of layers and neurons and compared with aforementioned Ritz method, Galerkin method and exact solution by Timoshenko. The results of our deep collocation agree best with the exact solution. However, for neural networks with a single hidden layer, the results are less accurate, even when 60 neurons are used. As the number of neurons increases, the results are indeed more accurate for the neural network with single hidden layer. This can be observed for the other two hidden layer types. Additionally, as the number of hidden layer increases, the results are significantly more accurate than the single hidden layer neural network results.

Table 2: Maximum deflection predicted by deep collocation method

Clamped Square Plate	Predicted Maximum Deflection	Galerkin method	Ritz method	Exact solution
1 hidden layers, 20 neurons	0.860568	1.321993	1.330000	1.260000
1 hidden layers, 30 neurons	1.152175			
1 hidden layers, 40 neurons	1.195843			
1 hidden layers, 50 neurons	1.249870			
2 hidden layers, 20 neurons	1.257226			
2 hidden layers, 30 neurons	1.257759			
2 hidden layers, 40 neurons	1.265145			
2 hidden layers, 50 neurons	1.261974			
3 hidden layers, 20 neurons	1.261780			
3 hidden layers, 30 neurons	1.260374			
3 hidden layers, 40 neurons	1.261743			
3 hidden layers, 50 neurons	1.258893			

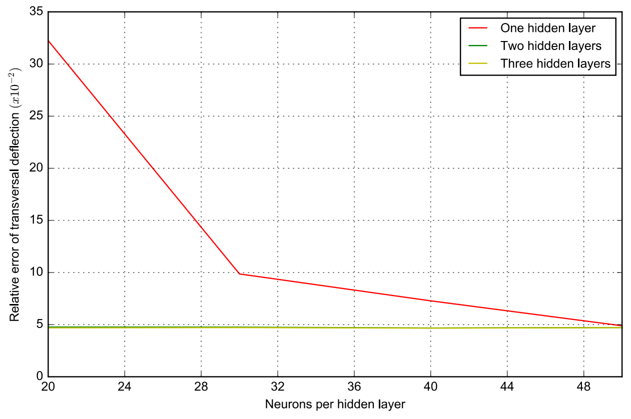
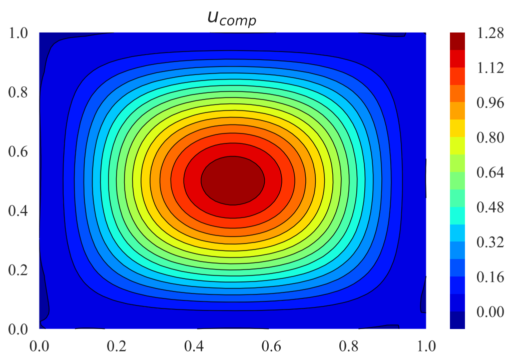
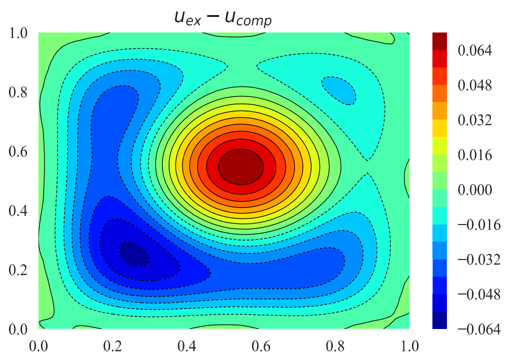


Figure 8: The relative error of deflection with varying hidden layers and neurons

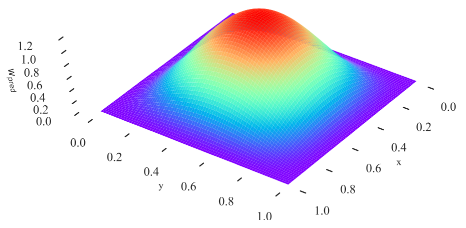
The relative error with the analytical solution with different hidden layers and different neurons is shown in Fig. 8. The magnitude of the relative error of the deflection for this numerical example is 10^{-2}, see also Tab. 2. With increasing number of hidden layers, the two flat relative error curves coincide and converge to the exact solution.



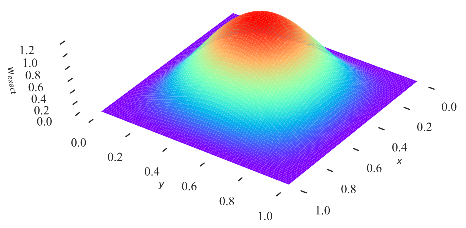
(a). Predicted deflection contour



(b). Deflection error contour



(c). Predicted deflection contour



(d). Deflection error contour

Figure 9: (a) Predicted deflection contour (b) Deflection error contour (c) Predicted deflection (d) Exact deflection of the clamped square plate with 3 hidden layers and 50 neurons with varying hidden layers and neurons

Finally, the deflection contour, relative error contour and deformed deflection of the middle surface for the deep neural network with three layers and 50 neurons is illustrated in Fig. 9.

4.3 Clamped circular plate

A clamped circular plate with radius R under a uniformly distributed force is employed in the domain of the circular plate. This problem has an exact solution given by Timoshenko et al. [Timoshenko and Woinowsky-Krieger (1959)]:

$$w = \frac{p_0}{64D}\left(R^2 - (x^2 + y^2)\right)^2 \tag{25}$$

D denoting the flexural stiffness of the plate. The maximum deflection at the central of the circular plate with varying hidden layers and neurons is summarized in Tab. 3 and compared with the exact solution. The predicted maximum deflection becomes more accurate with increasing number of neurons and hidden layers. The relative error for deflection of clamped circular plate with increasing hidden layers and neurons is depicted in Fig. 11. As hidden layer number increases, the relative error curves become flatter and converges to the exact solution. All neural networks perform well with a relative error magnitude of 10^{-4}.

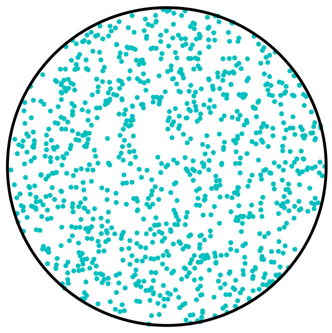


Figure 10: Collocation points discretize the circular domain

Table 3: Maximum deflection predicted by deep collocation method

Clamped Circular Plate	Predicted Maximum Deflection	Exact solution
1 hidden layers, 30 neurons	15.5958	
1 hidden layers, 40 neurons	15.5685	
1 hidden layers, 50 neurons	15.6201	
2 hidden layers, 30 neurons	15.6251	
2 hidden layers, 40 neurons	15.6264	15.6250
2 hidden layers, 50 neurons	15.6224	
3 hidden layers, 30 neurons	15.6269	
3 hidden layers, 40 neurons	15.6247	
3 hidden layers, 50 neurons	15.6229	

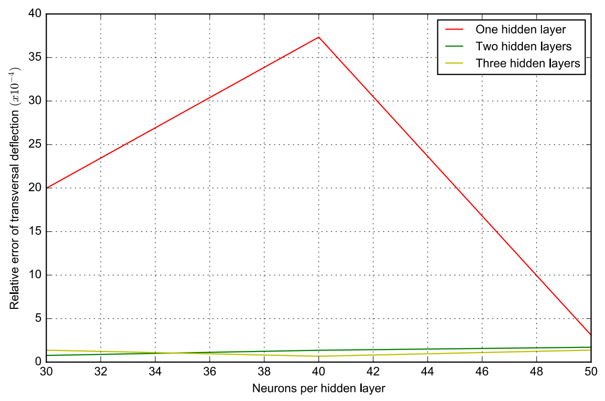
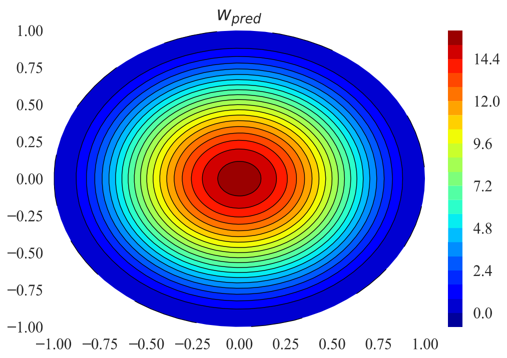
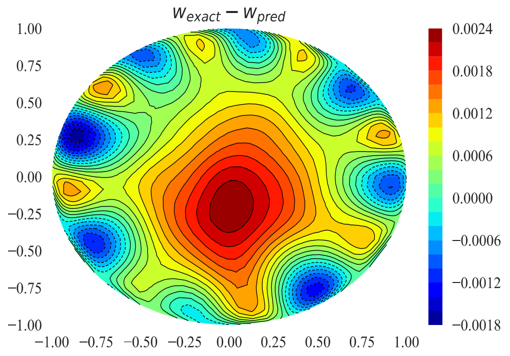


Figure 11: The relative error of deflection with varying hidden layers and neurons

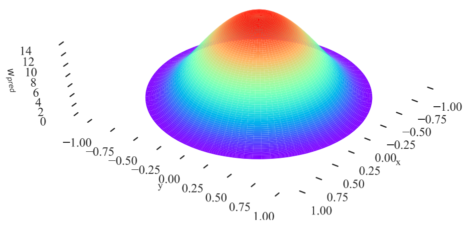
The deformation contour, deflection error contour, predicted and exact deformation figures are displayed in Fig. 12. The deflection of this circular plate agrees well with the exact solution.



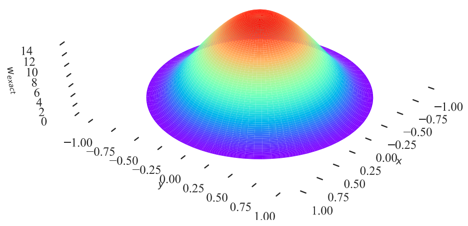
(a). Predicted deflection contour



(b). Deflection error contour



(c). Predicted deflection contour



(d). Deflection error contour

Figure 12: (a) Predicted deflection contour (b) Deflection error contour (c) Predicted deflection (d) Exact deflection of the clamped circular plate with 3 hidden layers and 50 neurons with varying hidden layers and neurons

4.4 Simply-supported square plate on Winkler foundation

The last example is a simply-supported square plate resting on Winkler foundation, which assumes that the foundation's reaction $p(x, y)$ can be described by $p(x, y) = kw$, k being a constant called $foundation\,modulus$. For a plate on a continuous Winkler foundation, the governing Eq. (8) can be written as:

$$\nabla^2(\nabla^2 w) = \nabla^4 w = \frac{(p-q)}{D} = \frac{(p-kw)}{D}. \qquad (26)$$

The analytical solution for this numerical example is [Timoshenko and Woinowsky-Krieger (1959)]:

$$w = \frac{16p}{\pi^2} \sum_{m=1,3,5,\cdots}^{\infty} \sum_{n=1,3,5,\cdots}^{\infty} \frac{sin\frac{m\pi x}{a} sin\frac{n\pi y}{b}}{mn\left[\pi^4 D\left(\frac{m^2}{a^2}+\frac{n^2}{b^2}\right)^2 + k\right]}, \qquad (27)$$

For this numerical example, the same arrangement of collocation points is depicted in Fig. 3. Neural networks with different neurons and depth are applied in the calculation. Tab. 3 lists the maximum deflection at the central point in all cases. Good agreement can be observed in this numerical example as well. As hidden layer and neuron number grows, the maximum deflection becomes more accurate approaching the analytical serial solution for even two hidden layers. The relative error shown in Fig. 13 better depicts the advantages of deep neural network than shallow wide neural network. More hidden layers, with more neurons yield flatting of the relative error. Various contour plots are shown in Fig. 14 and compared with the analytical solution.

Table 4: Maximum deflection predicted by deep collocation method

Square Plate on Winkler foundation	Predicted Maximum Deflection	Exact solution
1 hidden layers, 30 neurons	0.33999	
1 hidden layers, 40 neurons	0.35689	
1 hidden layers, 50 neurons	0.32168	
2 hidden layers, 30 neurons	0.32248	
2 hidden layers, 40 neurons	0.32176	0.32137
2 hidden layers, 50 neurons	0.32168	
3 hidden layers, 30 neurons	0.32216	
3 hidden layers, 40 neurons	0.32172	
3 hidden layers, 50 neurons	0.32181	

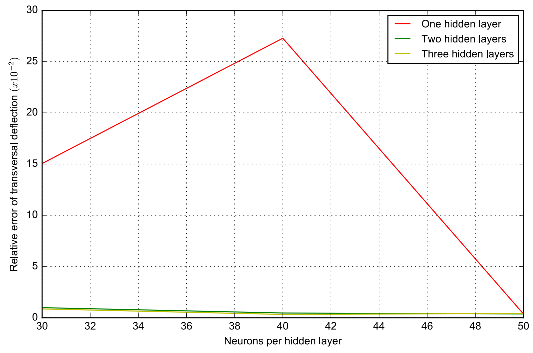
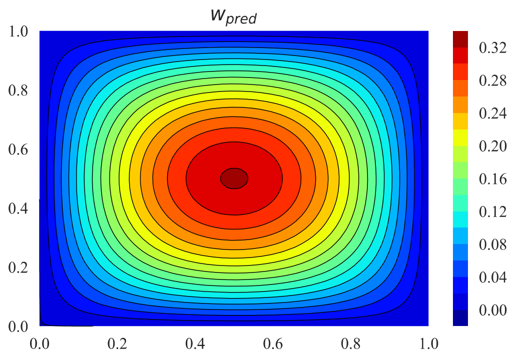
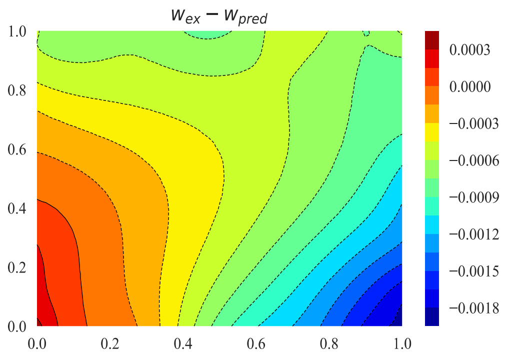


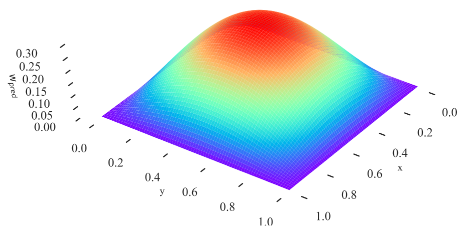
Figure 13: The relative error of deflection with varying hidden layers and neurons



(a). Predicted deflection contour



(b). Deflection error contour


(c). Predicted deflection contour

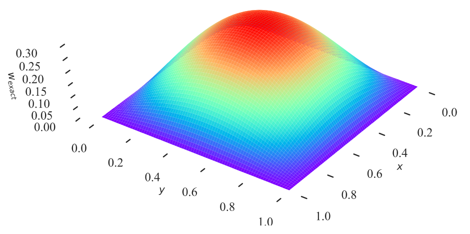

(d). Deflection error contour

Figure 12: (a) Predicted deflection contour (b) Deflection error contour (c) Predicted deflection (d) Exact deflection of the simply-supported plate on Winkler foundation with 3 hidden layers and 50 neurons with varying hidden layers and neurons

5 Conclusions

We propose a deep collocation method to study the bending analysis of Kirchhoff plates of various shapes, loads and boundary conditions. The governing equation of this problem is a fourth order partial differential equation (biharmonic equation). The proposed deep collocation method can be considered as truly "meshfree" and can be used to approximate any continuous function, which is very suitable for the analysis of thin plate bending problems. The deep collocation method is very simple in implementation, which can be further applied in a wide variety of engineering problems.

Moreover, the deep collocation method with randomly distributed collocations and deep neural networks perform very well with MSE loss function minimized by the combined L-BFGS and Adam optimizer. Accurate results are obtained even for a single layer and 20 neurons. However, as the hidden layers and neurons on each layer increase, results gain in accuracy and converge to the exact and analytical solution. Most importantly, once those deep neural networks are trained, they can be used to evaluate the solution at any desired points with minimal additional computation time.

However, there are still several issues for the deep neural network based method such as the influence of choosing other neural network types, activation functions, loss function forms, weight/bias initialization, and optimizers on the accuracy and efficiency of this deep collocation method, which will be studied in our future research.

References

Agrawal, P. Collocation based approach for training recurrent neural networks. https://people.eecs.berkeley.edu/~pulkitag/collocation-report.pdf.

Al-Aradi, A.; Correia, A.; Naiff, D.; Jardim, G.; Saporito, Y. (2018): Solving nonlinear and high-dimensional partial differential equations via deep Learning. arXiv:1811.08782.

Atluri, S. N. (2005): *Methods of Computer Modeling in Engineering & the Sciences*. Tech Science Press.

Bathe, K. J. (2006): *Finite Element Procedures*. Klaus-Jurgen Bathe.

Beck, C.; Becker, S.; Grohs, P.; Jaafari, N.; Jentzen, A. (2018): Solving stochastic differential equations and Kolmogorov equations by means of deep learning. arXiv:1806.00421.

Beck, C.; Weinan, E.; Jentzen, A. (2019): Machine learning approximation algorithms for high-dimensional fully nonlinear partial differential equations and second-order backward stochastic differential equations. *Journal of Nonlinear Science*, vol. 18, pp. 1-57.

Berg, J.; Nyström, K. (2018): A unified deep artificial neural network approach to partial differential equations in complex geometries. *Neurocomputing*, vol. 317, pp. 28-41.

Berg, J.; Nyström, K. (2019): Data-driven discovery of PDEs in complex datasets. *Journal of Computational Physics*, vol. 384, pp. 239-252.

Brebbia, C. A.; Walker, S. (2016): *Boundary Element Techniques in Engineering*. Elsevier.

Cybenko, G. (1989): Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, vol. 2, no. 4, pp. 303-314.

E, W. N.; Han, J.; Jentzen, A. (2017): Deep learning-based numerical methods for high-dimensional parabolic partial differential equations and backward stochastic differential equations. *Communications in Mathematics and Statistics*, vol. 5, no. 4, pp. 349-380.

E, W. N.; Bing, Y. (2018): The deep ritz method: A deep learning-based numerical algorithm for solving variational problems. *Communications in Mathematics and Statistics*, vol. 6, no. 1, pp. 1-12.

Fischer, T.; Krauss, C (2018): Deep learning with long short-term memory networks for financial market predictions. *European Journal of Operational Research*, vol. 270, no. 2, pp. 654-669.

Funahashi, K. I. (1989): On the approximate realization of continuous mappings by neural networks. *Neural Networks*, vol. 2, no. 3, pp. 183-192.

Guo, H.; Zheng, H. (2018): The linear analysis of thin shell problems using the numerical manifold method. *Thin-Walled Structures*, vol. 124, pp. 366-383.

Guo, H.; Zheng, H.; Zhuang, X. (2019): Numerical manifold method for vibration analysis of Kirchhoff's plates of arbitrary geometry. *Applied Mathematical Modelling*, vol. 66, pp. 695-727.

Han, J.; Jentzen, A.; E, W. N. (2018): Solving high-dimensional partial differential equations using deep learning. *Proceedings of the National Academy of Sciences*, vol. 115, no. 34, pp. 8505-8510.

Hayou, S.; Doucet, A.; Rousseau, J. (2018): On the selection of initialization and activation function for deep neural networks. arXiv:1805.08266.

Hornik, K. (1991): Approximation capabilities of multilayer feedforward networks. *Neural Networks*, vol. 4, no. 2, pp. 251-257.

Hornik, K.; Stinchcombe, M.; White, H. (1989): Multilayer feedforward networks are universal approximators. *Neural Networks*, vol. 2, no. 5, pp. 359-366.

Hughes, T. J. (2012): *The Finite Element Method: Linear Static and Dynamic Finite Element Analysis*. Courier Corporation.

Janocha, K.; Czarnecki, W. M. (2017): On loss functions for deep neural networks in classification. arXiv:1702.05659.

Katsikadelis, J. T. (2016): *The Boundary Element Method for Engineers and Scientists: Theory and Applications*. Academic Press.

Khan, Y.; Tiwari, P.; Ali, R. (2012): Application of variational methods to a rectangular clamped plate problem. *Computers & Mathematics with Applications*, vol. 63, no. 4, pp. 862-869.

Kingma, D. P.; Ba, J. (2015): Adam: a method for stochastic optimization. arXiv:1412.6980.

Lagaris, I. E.; Likas, A.; Fotiadis, D. I. (1998): Artificial neural networks for solving ordinary and partial differential equations. *IEEE Transactions on Neural Networks*, vol. 9, no. 5, 987-1000.

Lagaris, I. E.; Likas, A. C.; Papageorgiou, D. G. (2000): Neural-network methods for boundary value problems with irregular boundaries. *IEEE Transactions on Neural Networks*, vol. 11, no. 5, pp. 1041-1049.

LeCun, Y.; Bengio, Y.; Hinton, G. (2015): Deep learning. *Nature*, vol. 521, no. 7553, pp. 436.

Liang, L.; Liu, M.; Martin, C.; Elefteriades, J. A.; Sun, W. (2017): A machine learning approach to investigate the relationship between shape features and numerically predicted risk of ascending aortic aneurysm. *Biomechanics and Modeling in Mechanobiology*, vol. 16, no. 5, pp. 1519-1533.

Liang, L.; Liu, M.; Martin, C.; Sun, W. ((2018): A deep learning approach to estimate stress distribution: a fast and accurate surrogate of finite-element analysis. *Journal of the Royal Society Interface*, vol. 15, no. 138, pp. 20170844.

Liu, D. C.; Nocedal, J. (1989): On the limited memory BFGS method for large scale optimization. *Mathematical Programming*, vol. 45, no. 1-3, pp. 503-528.

McCulloch, W. S.; Pitts, W. (1943): A logical calculus of the ideas immanent in nervous activity. *The Bulletin of Mathematical Biophysics*, vol. 5, no. 4, pp. 115-133.

McFall, K. S.; Mahan, J. R. (2009): Artificial neural network method for solution of boundary value problems with exact satisfaction of arbitrary boundary conditions. *IEEE Transactions on Neural Networks*, vol. 20, no. 8, pp. 1221-1233.

Mills, K. (2017): Deep learning and the Schrödinger equation. *Physical Review A*, vol. 96, no. 4, 042113.

Nabian, M. A.; Meidani, H. (2018): A deep neural network surrogate for high-dimensional random partial differential equations. arXiv:1806.02957.

Nabian, M. A.; Meidani, H. (2018): Physics-informed regularization of deep neural networks. arXiv:1810.05547.

Nassif, A. B.; Shahin, I.; Attili, I.; Azzeh, M.; Shaalan, K. (2019): Speech recognition using deep neural networks: a systematic review. *IEEE Access*, vol. 7, pp. 19143-19165.

Nguyen, V. P.; Anitescu, C.; Bordas, S. P.; Rabczuk, T. (2015): Isogeometric analysis: an overview and computer implementation aspects. *Mathematics and Computers in Simulation*, vol. 117, pp. 89-116.

Nguyen, V. P.; Rabczuk, T.; Bordas, S.; Duflot, M. (2008): Meshless methods: A review and computer implementation aspects. *Mathematics and Computers in Simulation*, vol. 79, no. 3, pp. 763-813.

Nielsen, M. A. (2015): *Neural Networks and Deep Learning*, volume 25. USA: Determination Press.

Patterson, J; Gibson, A. (2017): *Deep Learning: A Practitioner's Approach*. O'Reilly Media, Inc.

Qin, T.; Wu, K.; Xiu, D. (2018): Data driven governing equations approximation using deep neural networks. arXiv:1811.05537.

Raissi, M. (2018): Forward-backward stochastic neural networks: deep learning of high-dimensional partial differential equations. arXiv:1804.07010.

Raissi, M.; Karniadakis, G. E. (2018): Hidden physics models: machine learning of nonlinear partial differential equations. *Journal of Computational Physics*, vol. 357, pp. 125-141.

Raissi, M.; Perdikaris, P.; Karniadakis, G. E. (2018): Numerical Gaussian processes for time-dependent and nonlinear partial differential equations. *SIAM Journal on Scientific Computing*, vol. 40, no. 1, pp. A172-A198.

Raissi, M.; Perdikaris, P.; Karniadakis, G. E. (2017): Machine learning of linear differential equations using Gaussian processes. *Journal of Computational Physics*, vol. 348, pp. 683-693.

Raissi, M.; Perdikaris, P.; Karniadakis, G. E. (2017): Physics informed deep learning (part i): data-driven solutions of nonlinear partial differential equations. arXiv:1711.10561.

Raissi, M.; Perdikaris, P.; Karniadakis, G. E. (2017): Physics informed deep learning (part ii): Data-driven discovery of nonlinear partial differential equations. arXiv:1711.10566.

Raissi, M.; Perdikaris, P.; Karniadakis, G. E. (2019): 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.

Sirignano, J.; Spiliopoulos, K. (2018): DGM: a deep learning algorithm for solving partial differential equations. *Journal of Computational Physics*, vol. 375, pp. 1339-1364.

Tartakovsky, A. M.; Marrero, C. O.; Tartakovsky, D.; Barajas-Solano, D. (2018): Learning parameters and constitutive relationships with physics informed deep neural networks. arXiv:1808.03398.

Timoshenko, S. P.; Woinowsky-Krieger, S. (1959): *Theory of Plates and Shells*. McGraw-hill.

Vargas, R.; Mosavi, A.; Ruiz, R. (2018): Deep learning: a review. *Advances in Intelligent Systems and Computing*.

Ventsel, E.; Krauthammer, T. (2001): *Thin Plates and Shells: Theory: Analysis, and Applications*. CRC Press.

Yang, L.; MacEachren, A.; Mitra, P.; Onorati, T. (2018): Visually-enabled active deep learning for (geo) text and image classification: a review. *ISPRS International Journal of Geo-Information*, vol. 7, no. 2, pp. 65.

Yue, T.; Wang, H. (2018): Deep learning for genomics: a concise overview. arXiv:1802.00810.

Zhao, Z. Q.; Zheng, P.; Xu, S. T.; Wu, X. (2019): Object detection with deep learning: a review. *IEEE Transactions on Neural Networks and Learning Systems*.

Zheng, H.; Liu, Z.; Ge, X. (2013): Numerical manifold space of Hermitian form and application to Kirchhoff's thin plate problems. *International Journal for Numerical Methods in Engineering*, vol. 95, no. 9, pp. 721-739.

Zhuang, X. Y.; Huang, R. Q.; Zhu, H. H. (2013): A new and simple locking-free triangular thick plate element using independent shear degrees of freedom. *Finite element in Analysis and Design*, vol. 75, pp. 1-7.