박 사 학 위 논 문

Ph.D. Dissertation

딥러닝을 이용한 2D 사각 유한요소 개발

Deep Learned 2D Quadrilateral Finite Elements

2021

정 재 호 (鄭 在 皓 Jung, Jaeho)

한 국 과 학 기 술 원

Korea Advanced Institute of Science and Technology

박 사 학 위 논 문

딥러닝을 이용한 2D 사각 유한요소 개발

2021

정 재 호

한 국 과 학 기 술 원

기계항공공학부/기계공학과

딥러닝을 이용한 2D 사각 유한요소 개발

정 재 호

위 논문은 한국과학기술원 박사학위논문으로
학위논문 심사위원회의 심사를 통과하였음

2020년 11월 9일

심사위원장 이 필 승 (인)
심 사 위 원 이 익 진 (인)
심 사 위 원 유 승 화 (인)
심 사 위 원 장 인 권 (인)
심 사 위 원 유 용 균 (인)

Deep Learned 2D Quadrilateral Finite Elements

Jaeho Jung

Advisor: Phill-Seung Lee

A dissertation/thesis submitted to the faculty of
Korea Advanced Institute of Science and Technology in
partial fulfillment of the requirements for the degree of
Doctor of Philosophy in Mechanical Engineering

Daejeon, Korea
November 9, 2020

Approved by

Phill-Seung Lee,
Professor of Mechanical Engineering

The study was conducted in accordance with Code of Research Ethics[1)].

1) Declaration of Ethical Conduct in Research: I, as a graduate student of Korea Advanced Institute of Science and Technology, hereby declare that I have not committed any act that may damage the credibility of my research. This includes, but is not limited to, falsification, thesis written by someone else, distortion of research findings, and plagiarism. I confirm that my dissertation contains honest conclusions based on my own careful research under the guidance of my advisor.

DME 정재호. 딥러닝을 이용한 2D 사각 유한요소 개발. 기계공학과. 2021년. 93+vii 쪽. 지도교수: 이필승. (영문 논문)
Jaeho Jung. Deep Learned 2D Quadrilateral Finite Elements. Mechanical Engineering. 2021. 93+vii pages. Advisor: Phill-Seung Lee. (Text in English)

초 록

본 연구에서는 인공지능 기법의 하나인 딥러닝(deep learning)을 이용하여 유한요소의 강성행렬을 생성하는 방법을 제안한다. 첫번째로 제안된 방법은 참조 데이터 모델(reference data model)에서 변형률(strain)을 학습하여 강성행렬을 생성하는 방법이다. 본 방법으로 개발된 요소는 조각 시험(patch test)과 영에너지모드 시험(zero energy mode test)을 실용적 수준에서 통과한다. 두번째로 제안된 방법은 분석해와 딥러닝을 이용한 국부좌표 설정으로 강성행렬을 생성하는 방법이다. 본 방법으로 개발된 유한요소는 조각 시험(patch test)과 영에너지모드 시험(zero energy mode test)을 통과한다. 다양한 수치 예제를 통해, 개발된 요소들의 성능을 조사하고 기존 요소와 비교한다. 이를 통해 개발된 유한요소가 기존요소의 성능을 뛰어넘을 수 있음을 확인하였다.

핵 심 낱 말 유한요소, 솔리드 요소, 강성행렬, 인공지능, 딥러닝

Abstract

In this work, we propose a method that employs deep learning, an artificial intelligence technique, to generate stiffness matrices of finite elements. The first proposed method is to generate a stiffness matrix by training the strain from the reference data model. The elements generated using the first method practically pass the patch tests and the zero energy mode tests. The second proposed method is to generate a stiffness matrix through an analytical strain and setting the local coordinates using deep learning. The elements generated using the second method pass the patch test and zero energy mode test. Through various numerical examples, the performance of the developed elements is investigated and compared with those of existing elements. It was confirmed that the deep learned finite elements can potentially outperform existing finite elements.

Keywords Finite element, Solid element, Stiffness matrix, Artificial intelligence, Deep learning, Neural network

Contents

List of Tables

List of Figures

Chapter 1. Introduction

1.1 Research background

Finite element method (FEM) is widely employed not only in structural analysis but also in almost all fields of engineering such as analyses of electromagnetic fields, flows, heat transfer, fluid-structure interactions [1-8]. In particular, FEM is the most powerful tool for structural analysis. FEM is inevitably used in various stages, from product design to manufacturing. However, the challenge of improving FEM is still being addressed.

An artificial neural network (ANN) is a brain-inspired system consisting of connected artificial neurons and is used to perform tasks based on data without specific rules. After the pioneering works of McCulloch and Pitts [9] and Rosenblatt [10], Hinton et al. [11,12] developed advanced algorithms for training networks that enabled the use of deep learning. Deep learning was used in AlexNet [13], which won the ImageNet Large Scale Visual Recognition Challenge (ILSVRC 2012) owing to its outstanding image recognition ability, and its application has gained momentum. Alpha Go [14] and Alpha Go Zero [15] use deep learning and have drawn worldwide attention for defeating top-ranked professional Go players. Deep learning is increasingly being applied in various fields such as automotive industry, medicine, finance and law.

Deep learning is also being studied for application in the field of numerical analysis, and several attempts have been made to use deep learning to solve partial differential equations [16-18]. Furthermore, deep learning has actively been adopted for computational fluid dynamics [19-24]. Several studies have related deep learning to the FEM. Takeuchi and Kosugi [25] showed that FEM formulations can be expressed as a neural network. Deep learning has been applied to construct surrogate models [26,27] and constitutive models for material nonlinear finite element analysis [28,29]. Oishi and Yagawa [30] utilized deep learning to increase the accuracy of numerical integration when calculating the stiffness matrix of finite elements.

1.2 Research purpose

In this paper, we show how deep learning can be used to generate a stiffness matrix of finite elements. First, we construct a neural network to generate the strain–displacement matrix, which is a key component of the stiffness matrix, corresponding to the geometry and material properties given as input data. For efficient learning, the geometric information is normalized to reduce the amount of training data. Strain values corresponding to a given displacement are obtained from a reference data model discretized with a fine mesh. Using the obtained data, we train a neural network that can generate strain–displacement matrices at Gauss points. Preprocessing is performed to generate the input of the trained network, and post-processing is performed to generate the stiffness matrix from

the output. A correction procedure is also applied to sufficiently represent rigid body motions of the finite elements.

The first proposed method is used to develop 8-node and 4-node quadrilateral plane stress finite elements, which we call “deep learned finite elements.” Basic numerical tests, including the patch and zero-energy mode tests, are carried out. The performance of the developed deep learned finite elements is demonstrated through various numerical problems.

Second, a new 4-node element is presented based on a mode-based formulation using analytical strains and an update method using deep learning. The analytical strain is obtained from rigid body modes, constant strain modes, and bending modes. In the case of bending modes, the strain depends on the determination of the local coordinates for bending modes. The local coordinates are initially set at a small angle and are updated using the displacement obtained from the analysis, elemental geometry, and material properties. The update of local coordinate determined through a deep learning network. According to the elemental geometry, displacement, and material properties, the local coordinates that make the element have the minimum strain energy are obtained through an optimization method. Then, the network is trained using the elemental information as an input data to inference the angle of local coordinate. For efficient learning, the geometric information and displacement are normalized to reduce the amount of training data. The update of local coordinate continues until the strain energy decreases. We call the finite element generated in this method Self-Updated Finite Element (SUFE).

SUFE passes patch test and zero energy mode. Many benchmark problems and numerical examples are applied to SUFE for comparing the results to previously developed elements.

1.3 Organization

This paper is organized as follows:

In Chapter 2, the formulation of the finite element in structural analysis and the problems of the conventional finite element are described.

In Chapter 3, we briefly introduce deep learning and the representative deep learning network structures.

In Chapter 4, we present the method for generating 8- and 4-node quadrilateral finite elements by deep learning, including data generation, network configuration and training, and the construction of the stiffness matrix. The basic test results of the deep learned finite elements, and the performance of the obtained finite elements are reported.

In Chapter 5, we present the method for generating SUFE using deep learning, including data generation, network configuration and training, and the construction of the stiffness matrix applying iteration. The basic test results of the deep learned finite elements, and the performance of the elements are reported.

Finally, in Chapter 6, the conclusions are presented.

Chapter 2. Overview of Finite Element

Finite element method (FEM) needs a discretization of the analysis domain according to the geometry and the properties. The discretized domain is filled with meshes called finite elements, and the performance of each finite element determines the accuracy of the FEM model [39, 40, 43-45, 49-54]. In this chapter, the formulation of the finite element in structural analysis and the problems of the conventional finite element are described.

2.1 Formulation of finite element

The discrete element is composed of nodes as shown in Fig. 2.1. In this chapter, we briefly describe the procedure to generate finite elements based on the *q*-node 2D quadrilateral solid element [1].

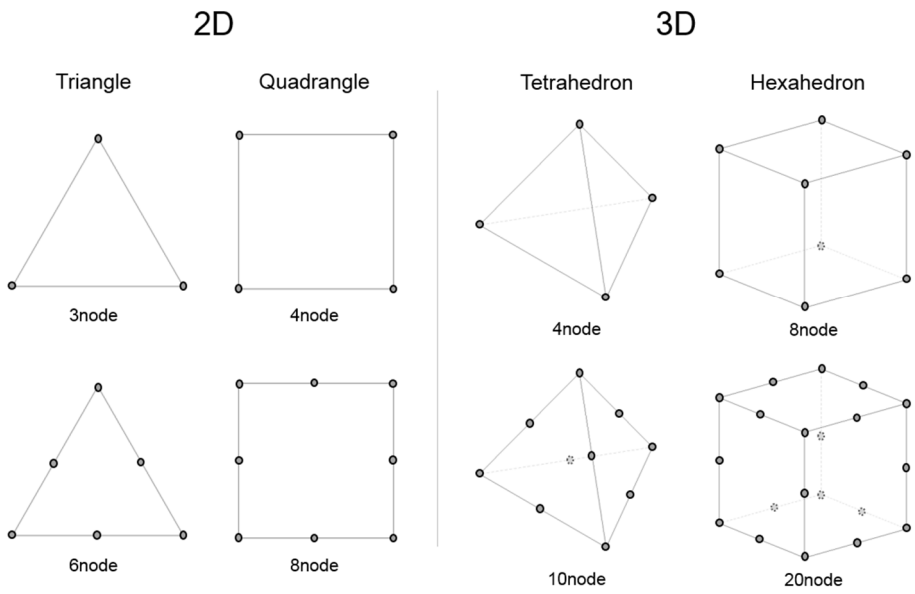

Figure 2.1 Finite element types according to the number of nodes and dimension.

In structural analysis, the nodal displacements determine the displacement and stress fields in the element. The elemental displacement fields are obtained by the shape functions as follows,

$$\mathbf{u} = \sum_{i=1}^{q} h_i \mathbf{u}_i \quad \text{with} \quad \mathbf{u} = \begin{bmatrix} u & v \end{bmatrix}^{\mathrm{T}} \quad \text{and} \quad \mathbf{u}_i = \begin{bmatrix} u_i & v_i \end{bmatrix}^{\mathrm{T}}, \tag{2-1}$$

in which h_i are the i[th] shape function, q is the number of nodal point in the element, u is x-directional displacement, v is y-directional displacement, $\mathbf{u}_i$ is the displacement vector of node i, and i is local index of nodes in the element.

The strain of a structure is obtained by partial differentiation of the displacement as follows,

$$\varepsilon_{xx}=\frac{\partial \mathbf{u}_x}{\partial x},\ \varepsilon_{yy}=\frac{\partial \mathbf{u}_y}{\partial y},\ \gamma_{xy}=\frac{\partial \mathbf{u}_y}{\partial x}+\frac{\partial \mathbf{u}_x}{\partial y}\ \text{ where }\ \mathbf{u}_x=\begin{bmatrix}u_1 & \dots & u_q\end{bmatrix}^{\mathrm{T}},\ \mathbf{u}_y=\begin{bmatrix}v_1 & \dots & v_q\end{bmatrix}^{\mathrm{T}}, \tag{2-2}$$

in which ε_{xx} is x-directional strain, ε_{yy} is y-directional strain, and γ_{xy} is engineering shear strain.

The strain can be formulated as the multiplication of the displacement vector and the strain displacement matrix ($\mathbf{B}$) as follows,

$$\boldsymbol{\varepsilon}=\mathbf{B}\mathbf{u}_e \ \text{ where }\ \boldsymbol{\varepsilon}=\begin{bmatrix}\varepsilon_{xx} & \varepsilon_{yy} & \gamma_{xy}\end{bmatrix}^{\mathrm{T}},\ \mathbf{u}_e=\begin{bmatrix}u_1 & u_2 & \cdots & u_q & v_1 & v_2 & \cdots & v_q\end{bmatrix}^{\mathrm{T}},$$

$$\mathbf{B}=\begin{bmatrix}\frac{\partial h_1}{\partial x} & \dots & \frac{\partial h_q}{\partial x} & 0 & \dots & 0\\ 0 & \dots & 0 & \frac{\partial h_1}{\partial y} & \dots & \frac{\partial h_q}{\partial y}\\ \frac{\partial h_1}{\partial y} & \dots & \frac{\partial h_q}{\partial y} & \frac{\partial h_1}{\partial x} & \dots & \frac{\partial h_q}{\partial x}\end{bmatrix} \tag{2-3}$$

The stiffness matrix of the element is given by

$$\mathbf{K}=\int_V \mathbf{B}^{\mathrm{T}}\mathbf{C}\mathbf{B}dV \tag{2-4}$$

where $\mathbf{C}$ is the material law matrix. The integration of the stiffness matrix can be calculated numerically.

2.2 Problem of conventional finite element

2.2.1 Locking

In certain cases, the displacements calculated by the finite element method are much smaller than they should be, and when this happens, the elements are said to be locking.

Volumetric locking is the most common locking encountered with the elements. The stress in the element can be divided into deviatoric and volumetric components. The volumetric component is obtained by

$$\sigma_{\mathrm{Vol}}=K\varepsilon_{\mathrm{Vol}} \ \text{ with }\ K=\frac{E}{3(1-2\nu)},\ \varepsilon_{\mathrm{Vol}}=-\frac{dV}{V} \tag{2-5}$$

where K is bulk modulus, and $\varepsilon_{\mathrm{Vol}}$ is volumetric strain. Volumetric locking is exhibited by incompressible materials having Poisson's ratio (ν) near to 0.5. As the Poisson's ratio approaches 0.5, the bulk modulus diverges to infinity. It means that the element can be overly stiffened. The element in which the volumetric locking has occurred reduces the deformation and causes an error.

Shear locking is another common locking that increases the elemental stiffness and causes an error in the linear element. Shear locking occurs when elements are subjected to bending. The displacement in pure bending should be as shown in Fig. 2.2(a) and γ_{xy} should be zero. However, due to the limitation of the linear element, the displacement of the upper and lower sides becomes a straight line as shown in Fig. 2.2(b) and γ_{xy} is non-zero. Accordingly, the stiffness of the element becomes larger than they should be.

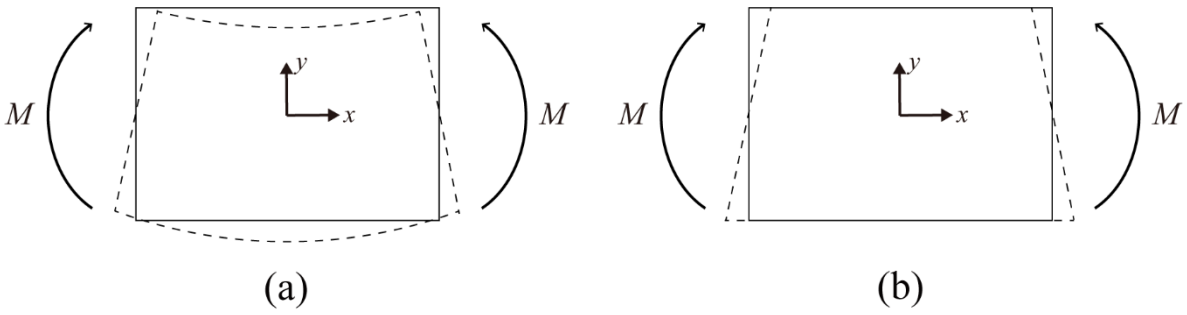

Figure 2.2. The displacements of element under pure bending. (a) Exact displacements. (b) Displacements of the linear element. The solid line is the undeformed shape, and the dotted line is the deformed shape by pure bending.

2.2.2 Limits of shape function

In general finite elements, the displacement fields are calculated by interpolating the shape function. The shape function of the isoparametric element, which is the most commonly used, does not reflect the changes in geometry and material properties (see Chapter 4.1). An error occurs due to this limitation of shape function.

Fig. 2.3 plots the displacement fields using one 2D plane stress element (Fig. 2.3(b)) or 900 2D plane stress elements (Fig. 2.3(c)) by applying the same material properties, geometry, and outer nodal displacement. In FEM, it is known that the higher the density of element is, the more accurate [80]. If the shape function is accurate, the internal displacements should be the same for cases where one element is used and cases where 900 elements are used. However, as shown in Fig. 2.3(d), the difference can be seen. It means that shape function has an error. The general finite element has such an error fundamentally because it approximates the internal displacement in a polynomial form.

The shape function does not consider the material properties. Fig. 2.4 shows the displacement field by changing the Poisson's ratio using 900 2D plane stress elements in the geometry and condition of Fig. 2.3(a). Even with the same geometry, if the Poisson’s ratio is changed, the internal displacement changes as shown in Fig. 2.4. However, if one element is used under the same conditions, this change cannot be detected, because the shape function is not a function of the material properties.

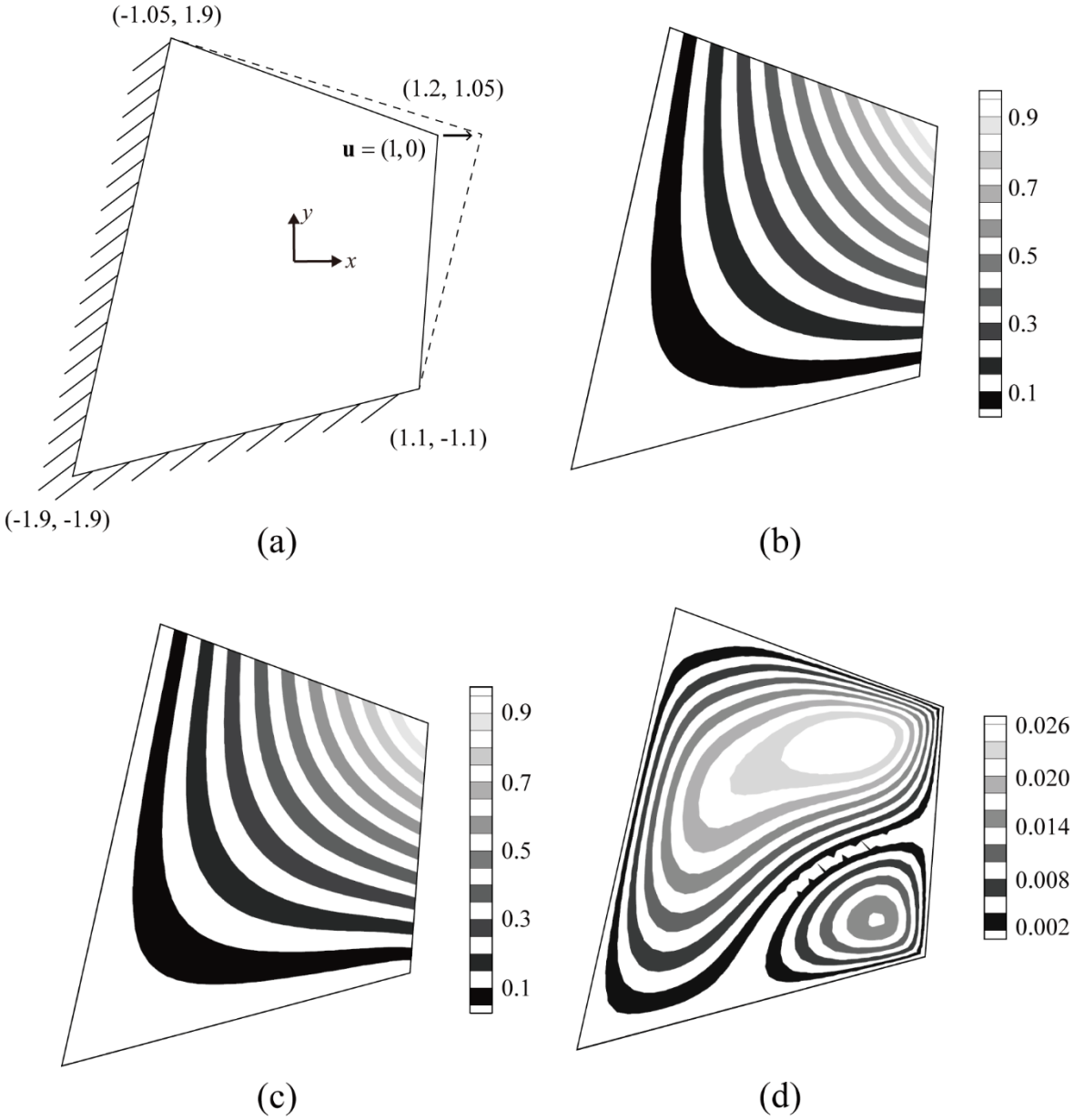

Figure 2.3. The displacement fields contour in (a) the geometry and boundary conditions ($E = 2.0\times10^{11}$, $\nu = 0.3$, thickness $= 1.0$), using (b) one element or (c) 900 elements. (d) The difference of displacements between (b) and (c) is contoured.

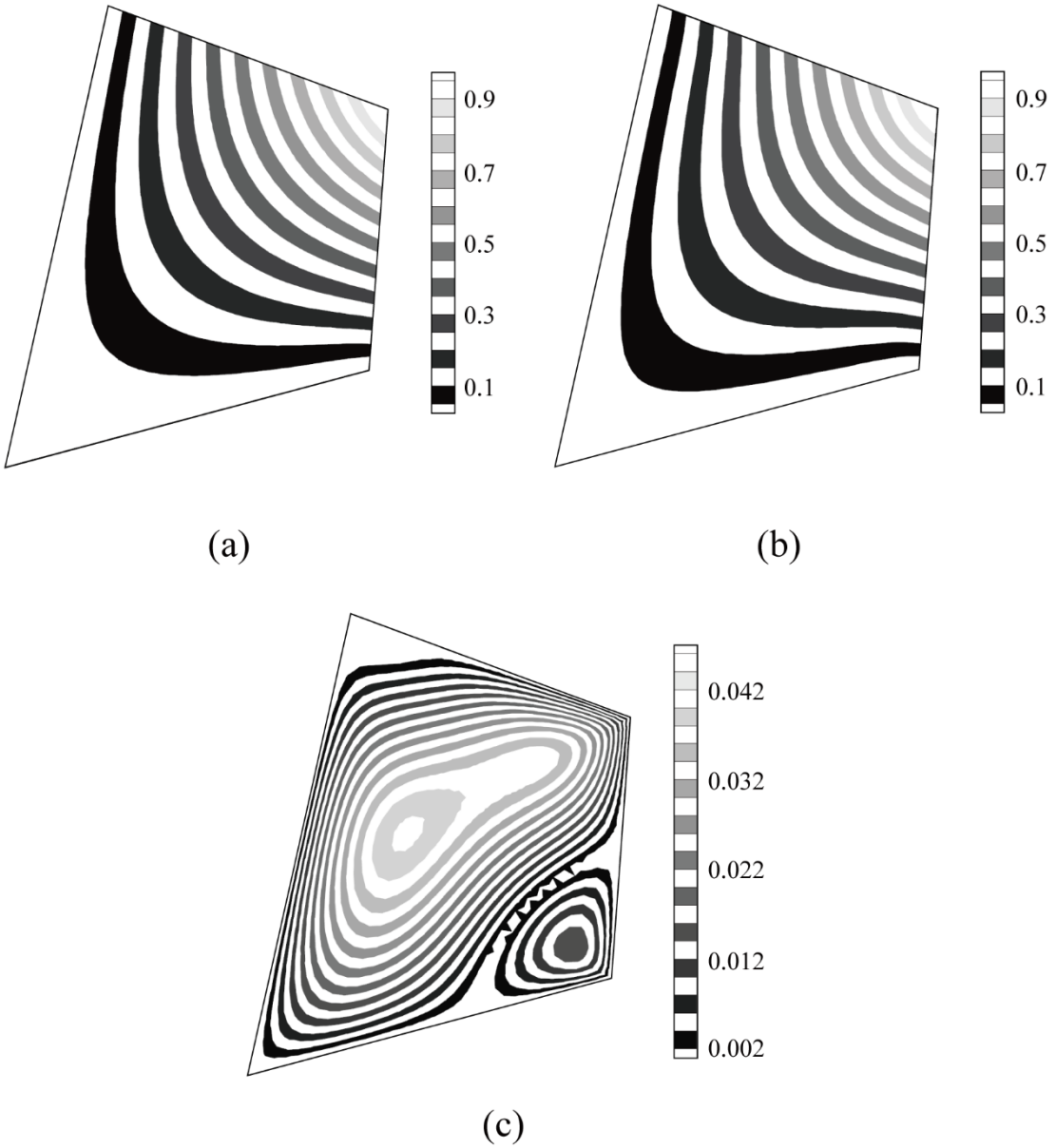

Figure 2.4. The displacement fields contour in the conditions of Fig. 2.3(a) using 900 elements according to (a) $\nu = 0.1$, (b) $\nu = 0.4999$. (c) The difference of displacements between (a) and (b) is contoured.

The shape function does not consider the geometry of element. Fig. 2.5 shows the displacement field by changing the geometry using 900 2D plane stress elements in the same topological condition to Fig. 2.3(a). Even with the same condition, if the geometry is changed, the internal displacement changes as shown in Fig. 2.5. However, if one element is used, this change cannot be detected, because the shape function is not a function of the elemental geometry.

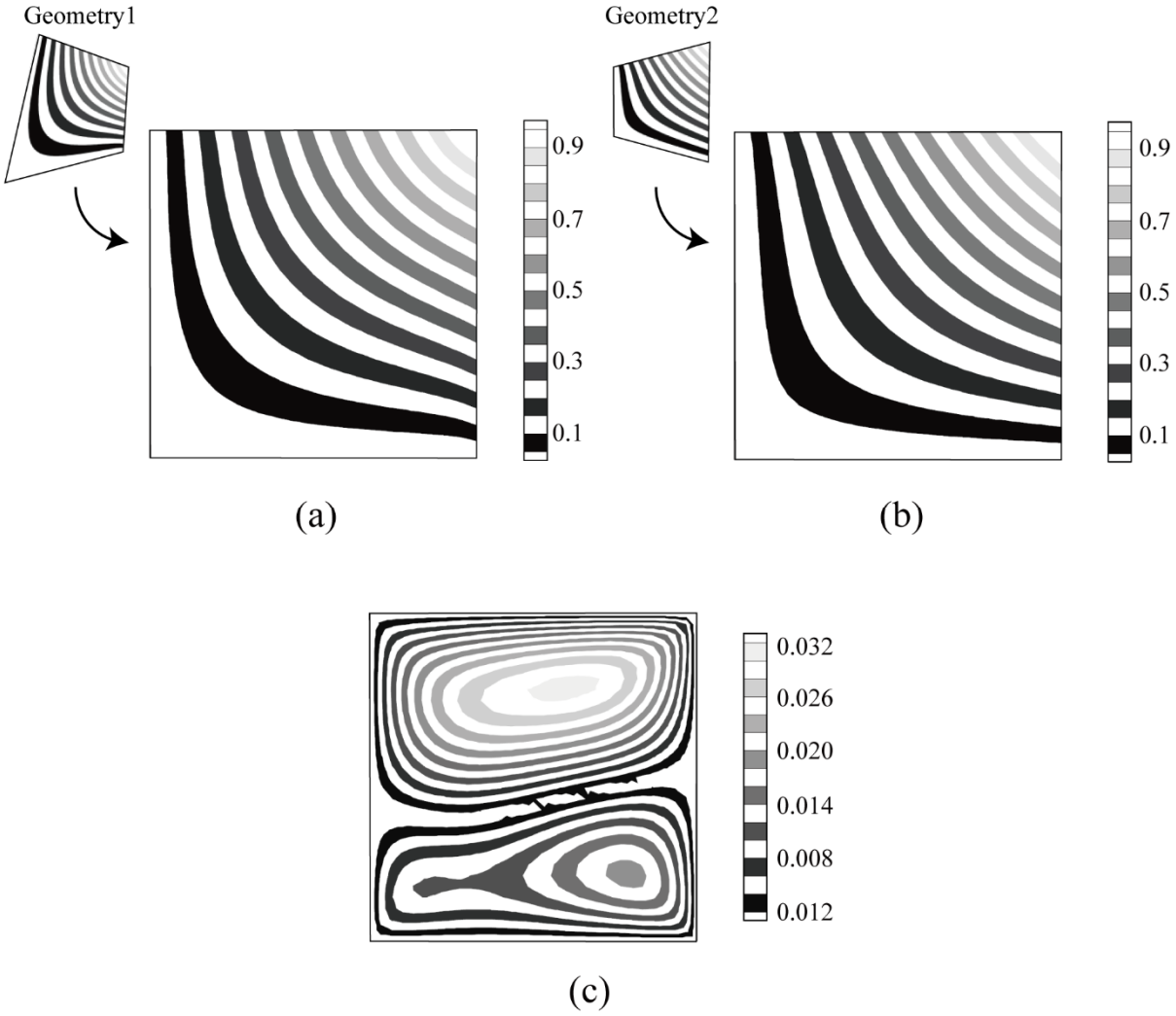

Figure 2.5 The displacement fields mapped from (a) Geometry1 and (b) Geomety2 to regular shape in the same topological conditions of Fig. 2.3(a) using 900 elements. (c) The difference of displacements between (a) and (b) is contoured.

Chapter 3. Overview of Deep Learning

Deep learning is a kind of machine learning. As shown in Fig. 3.1, the artificial neuron is a structure that produces an output when the summation of multiple inputs multiplied by weights input to the activation function. It is a structure created by computationally imitating the neurons in the brain. ANN are made up of multiple layers of artificial neurons. It was inspired by the structure of the brain.

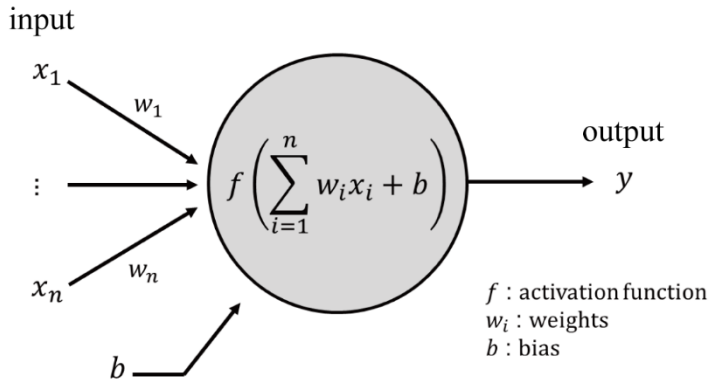

Figure 3.1 Schematic of artificial neuron.

The concept of ANN was first proposed by McCulloch and Pitts [9]. Rosenblatt [10] devised Perceptron that is a trainable ANN with a practical algorithm. After that, backpropagation method in ANN was proposed by Paul Werbos [78, 79]. This method made it possible to train multilayered ANN. As new techniques for training were introduced and the structure of the network became deeper and more diverse according to the training target, the training method of deep ANN was called Deep learning.

In this chapter, we briefly introduce some of the representative deep learning network structures.

3.1 Fully connected network

Fully connected network (FCN) is a general ANN structure and does not require a special input form. FCN is a structure that is completely connected between each layer with 1 input layer, multiple hidden layers, and 1 output layer as shown in Fig. 3.2. A structure in which the output does not enter the input to the previous layer is called a feed forward neural network, and this structure can be used as a universal approximator [32, 33]. In other words, a fully connected feed forward neural network can create functions that produce desired results according to training.

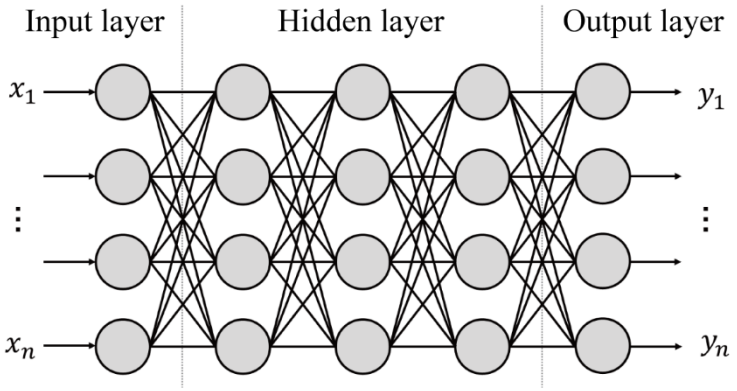

Figure 3.2 Schematic of fully connected network.

3.2 Convolutional neural network

Convolutional neural network (CNN) is a network mainly used for image processing and uses 2D data format as input. CNN uses filter-type weights as shown in Fig. 3.3. Each layer is not completely connected, and the output is calculated through a filter composed of weights. Since all layers are not fully connected, less weight is used compared to FCN. In CNN, a filter is trained to extract features of input data. CNN is based on Fukushima's neocognitron [81, 82], and later, Lecun [83, 84] standardized it and used it to classify handwriting digit images to become the current form of CNN.

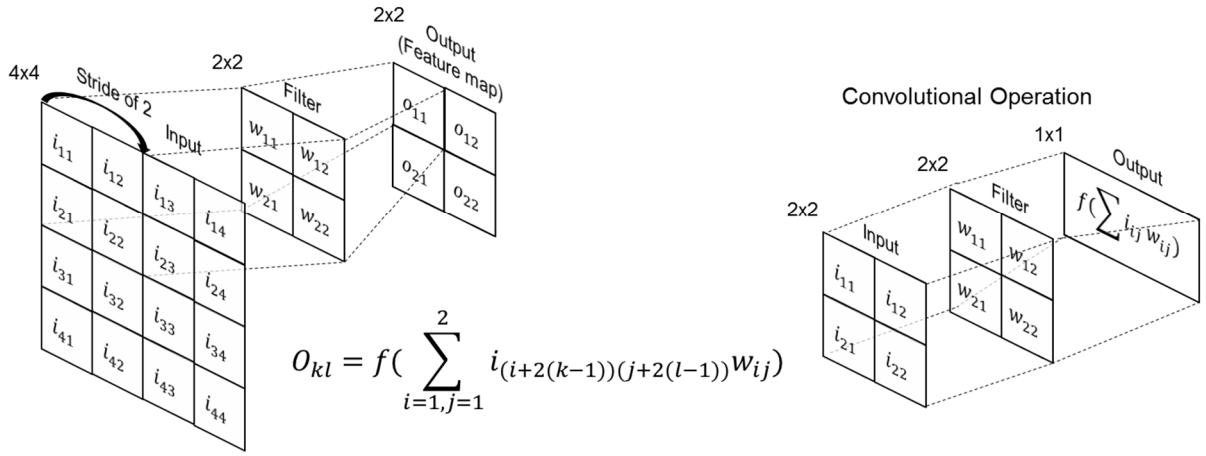

Figure 3.3 Example of convolutional neural network with a stride of 2.

3.3 Recurrent neural network

Recurrent neural network is a network model for handling sequence data. As shown in Fig. 3.4, It is trained to generate output by taking sequence data and part of previous output of network as input. It operates like a recursive function, and the reused output as an input is called the state generated by the previous input and serves as a

memory of the previous data [85, 86].

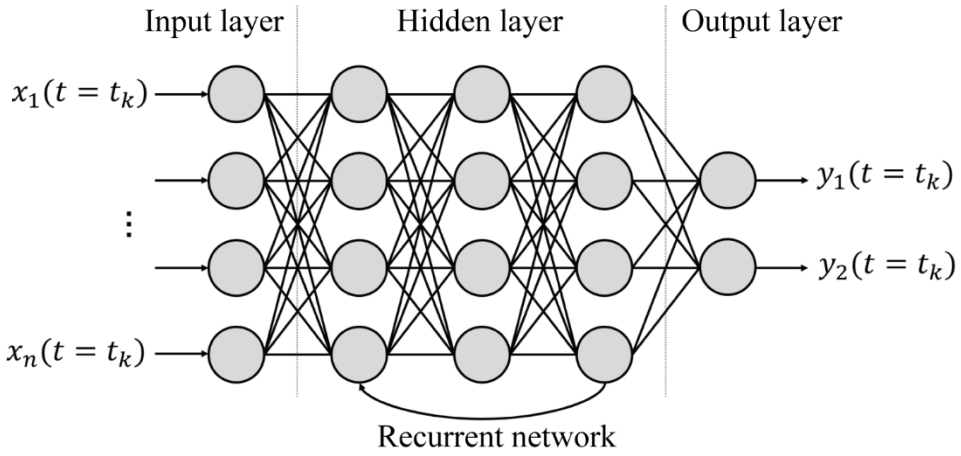

Figure 3.4 Schematic of recurrent neural network.

3.4 Variational auto encoder and Generative adversarial network

Variational auto encoder (VAE) and Generative adversarial network (GAN) are models proposed by Kingma [87] and Goodfellow [88], respectively, and are a kind of generative model. It is used for unsupervised learning. Auto encoder has an encoder network, a decoder network, and a laten variable as shown in Fig. 3.5. VAE, a kind of auto encoder, uses Gaussian distribution to make latent variable as shown in Fig 3.6. It makes the output blurry but natural. GAN is a structure that produces clear and natural results by competitively training a generator and a discriminator. Various modified structures of GAN have been published [89-94].

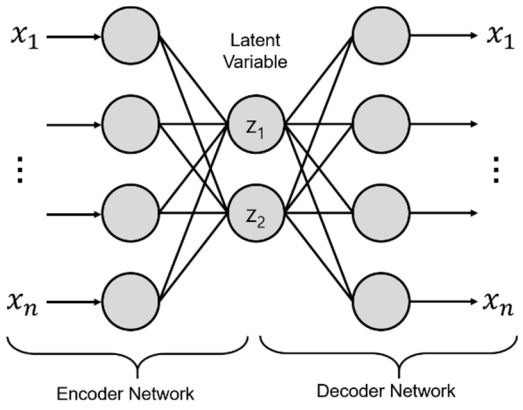

Figure 3.5 Schematic of auto encoder.

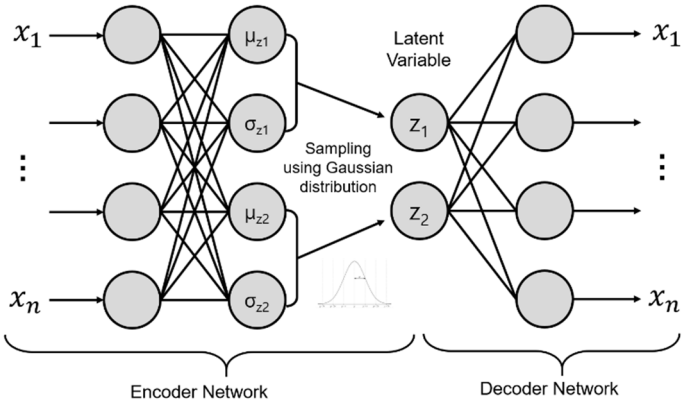

Figure 3.6 Schematic of variational auto encoder.

Chapter 4. Deep Learned Finite Elements

In this chapter, deep learned finite elements is presented. Chapter 4.1 briefly reviews the standard isoparametric finite element procedure. Chapter 4.2 presents the method for generating 8-node quadrilateral finite elements by deep learning, including data generation, network configuration and training, and the construction of the stiffness matrix. Chapter 4.3 presents the method for generating 4-node quadrilateral finite elements. Chapter 4.4 reports the basic test results of the deep learned finite elements, and Chapter 4.5 and 2.6 discuss the performance of the obtained finite elements through various numerical examples. Finally, the concluding remarks are presented in Chapter 4.7.

4.1 Isoparametric finite element procedure

The procedure to generate deep learned finite elements is based on the formulation of isoparametric finite elements. In this chapter, we briefly review the isoparametric finite element procedure for a q-node 2D quadrilateral solid element [1].

The geometry of the q-node element is interpolated by

$$\mathbf{x} = \sum_{i=1}^{q} h_i(r,s)\mathbf{x}_i \quad \text{with} \quad \mathbf{x} = \begin{bmatrix} x & y \end{bmatrix}^{\mathrm{T}} \quad \text{and} \quad \mathbf{x}_i = \begin{bmatrix} x_i & y_i \end{bmatrix}^{\mathrm{T}}, \tag{4-1}$$

where $\mathbf{x}_i$ is the position vector of node i in the global Cartesian coordinate system as shown in Fig. 4.1(a) and $h_i(r,s)$ are the shape functions defined in the natural coordinate system as shown in Fig. 4.1(b).

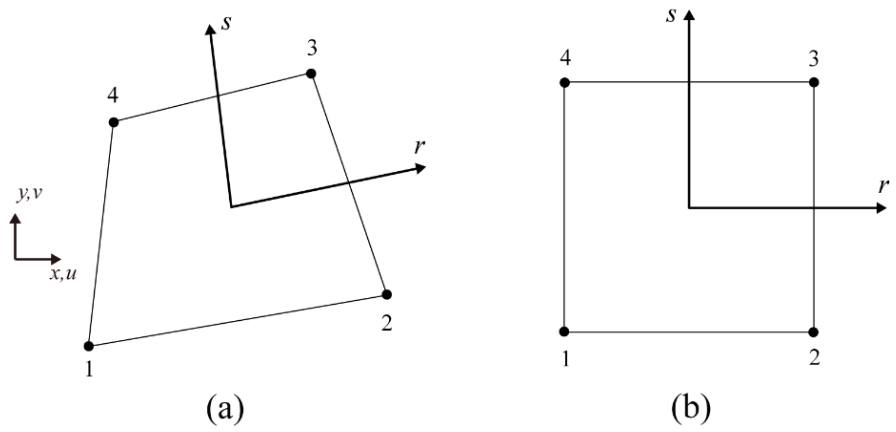

Figure 4.1. 4-node quadrilateral element in the (a) global Cartesian coordinate system and (b) natural coordinate system.

The corresponding displacement interpolation is given by

$$\mathbf{u} = \sum_{i=1}^{q} h_i(r,s)\mathbf{u}_i \quad \text{with} \quad \mathbf{u} = \begin{bmatrix} u & v \end{bmatrix}^{\mathrm{T}} \quad \text{and} \quad \mathbf{u}_i = \begin{bmatrix} u_i & v_i \end{bmatrix}^{\mathrm{T}}, \tag{4-2}$$

in which $\mathbf{u}_i$ is the displacement vector of node i.

The derivatives of the displacement with respect to the global coordinates are calculated using the Jacobian matrix $\mathbf{J}$:

$$\begin{bmatrix} \frac{\partial u}{\partial x} \\ \frac{\partial u}{\partial y} \end{bmatrix} = \mathbf{J}^{-1} \begin{bmatrix} \frac{\partial u}{\partial r} \\ \frac{\partial u}{\partial s} \end{bmatrix}, \quad \begin{bmatrix} \frac{\partial v}{\partial x} \\ \frac{\partial v}{\partial y} \end{bmatrix} = \mathbf{J}^{-1} \begin{bmatrix} \frac{\partial v}{\partial r} \\ \frac{\partial v}{\partial s} \end{bmatrix} \quad \text{with} \quad \mathbf{J} = \begin{bmatrix} \frac{\partial x}{\partial r} & \frac{\partial y}{\partial r} \\ \frac{\partial x}{\partial s} & \frac{\partial y}{\partial s} \end{bmatrix}. \tag{4-3}$$

The derivatives in Eq. (4-3) are used to obtain the strain-displacement matrix $\mathbf{B}$:

$$\boldsymbol{\varepsilon} = \mathbf{B}(r,s)\mathbf{u}, \tag{4-4}$$

where $\boldsymbol{\varepsilon}$ is the strain vector and $\mathbf{u}$ is the nodal displacement vector

$$\boldsymbol{\varepsilon} = \begin{bmatrix} \varepsilon_{xx} & \varepsilon_{yy} & \gamma_{xy} \end{bmatrix}^{\mathrm{T}} \quad \text{with} \quad \gamma_{xy} = 2\varepsilon_{xy},$$

$$\mathbf{u} = \begin{bmatrix} u_1 & u_2 & \cdots & u_q & v_1 & v_2 & \cdots & v_q \end{bmatrix}^{\mathrm{T}}. \tag{4-5}$$

Note that the matrix $\mathbf{B}$ has the dimensions of $3\times 2q$ because the strain and nodal displacement vectors contain 3 and $2q$ components, respectively.

The stiffness matrix ($\mathbf{K}$) of the 2D solid element with the thickness t is given by

$$\mathbf{K} = t\int_{-1}^{1}\int_{-1}^{1} \mathbf{B}^{\mathrm{T}}(r,s)\mathbf{C}\mathbf{B}(r,s)\det\mathbf{J}(r,s)drds, \tag{4-6}$$

in which $\mathbf{C}$ is the material law matrix.

The stiffness matrix in Eq. (4-6) is numerically calculated using the $p\times p$ Gaussian quadrature

$$\mathbf{K} = t\sum_{i=1}^{p}\sum_{j=1}^{p} {}^{(i,j)}w\,{}^{(i,j)}\mathbf{B}^{\mathrm{T}}\,{}^{(i,j)}\mathbf{C}\,{}^{(i,j)}\mathbf{B}\,{}^{(i,j)}J, \tag{4-7}$$

where ${}^{(i,j)}w$ denotes the weight factor at the Gauss point (r_i, s_j), ${}^{(i,j)}J = \det\mathbf{J}(r_i, s_j)$, and ${}^{(i,j)}\mathbf{B} = \mathbf{B}(r_i, s_j)$.

4.2 Procedures to generate 8-node deep learned finite elements

In this chapter, we introduce the procedure to generate the strain-displacement matrix of an 8-node quadrilateral finite element *via* deep learning and to obtain its stiffness matrix. The methodology for constructing the neural

network model is presented in detail, including data generation, network configuration and training. Finally, we present how to obtain the stiffness matrix using the trained neural network.

4.2.1 Data generation

In order to construct a neural network that generates the strain-displacement matrix of an arbitrary finite element, a large amount of strain data corresponding to randomly given geometries, displacements, and material properties are required. Since processing all these data is extremely difficult, a key point in this study was to appropriately reduce the amount of data for efficient network training.

In this study, the geometry of 8-node finite elements is limited to a quadrilateral whose mid-side node (nodes 5–8 in Fig. 4.2) are placed at the center of the adjacent corner nodes (nodes 1–4 in Fig. 4.2). In other words, the shape of the element is determined by the locations of its four corner nodes. We also use normalized geometry as a representative of all the similar shapes. Here, the normalized geometry refers to a quadrilateral where the two nodes at either end of the maximum length side are located at (0, 0) and (1, 0) in 2D Cartesian coordinates.

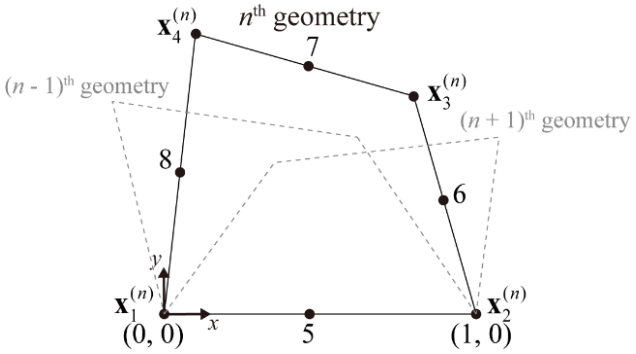

Figure 4.2. Random generation of the n^{th} normalized element geometry.

The n^{th} normalized random geometry is generated as follows. Let the position vector of node i corresponding to the n^{th} geometry be denoted as $\mathbf{x}_i^{(n)}$. Node 1 ($\mathbf{x}_1^{(n)}$) and node 2 ($\mathbf{x}_2^{(n)}$) of the n^{th} geometry are fixed at the coordinates of (0, 0) and (1, 0), respectively, as shown in Fig. 4.2. Then, the coordinates of the other two corner nodes ($\mathbf{x}_3^{(n)}$ and $\mathbf{x}_4^{(n)}$) are randomly placed where the distance between node 1 and node 2 should be the maximum length edge (see Appendix D).

In this way, normalized element geometries can be generated. Here, we excluded severely distorted geometries such as quadrilaterals with an interior angle of less than 10° or greater than 170° and ratios between the maximum and minimum side lengths of greater than 10. Young's modulus ($E = 2.0\times10^{11}$) was adopted, and Poisson's ratio ($\nu^{(n)}$) was randomly applied with a uniform distribution in the range of 0–0.499999999.

To derive the relation between the nodal displacements ($\mathbf{u}_i^{(n)}$) and the corresponding reference strains in an element with the n^{th} geometry and $\nu^{(n)}$ (hereafter called element n), nodal displacements are generated randomly with a uniform distribution in the range of -0.25 to 0.25, as shown in Fig. 4.3(a). The reference strain data are obtained from a reference data model having the same geometry as element n with a uniform $N \times N$ mesh, see Fig. 4.3(b). In this study, the reference data model was discretized using standard 4-node quadrilateral elements (Q4) [31].

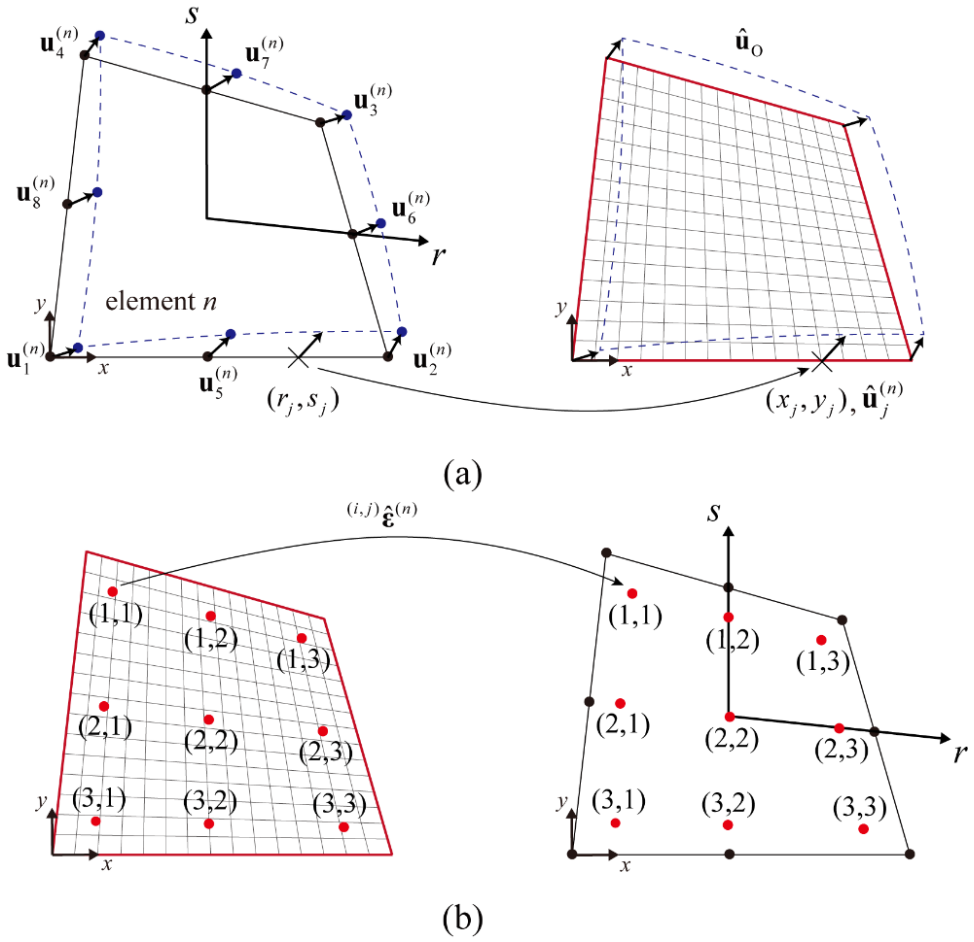

Figure 4.3. Schematics of the reference strain data generation procedure. (a) Random displacements generation and mapping of the outer displacement to the reference data model. (b) Strain extraction from the reference data model. The red dots represent the location of Gauss points where strain values are extracted.

The reference data model has $(N+1)^2$ nodes and $4 \times N$ outer nodes. We then map the outer displacements of the element n into those of the reference data model. To do so, the displacement $\hat{\mathbf{u}}_j^{(n)}$ at each outer node of the model is obtained from the displacement interpolation of the element n. The displacement vector at outer node j is given by

$$\hat{\mathbf{u}}_j^{(n)} = \sum_{i=1}^{8} h_i(r_j, s_j)\mathbf{u}_i^{(n)}, \tag{4-8}$$

in which (r_j, s_j) represents the natural coordinates of the element n corresponding to outer node j of the reference data model, and h_i is i^{th} shape function of standard 8-node quadrilateral element [31]. Note that $N = 50$ was used in this study. (see Appendix B).

After the displacements of all outer nodes are mapped, the outer nodal displacements are applied to the reference data model as prescribed displacements. The nodal displacement vector of the reference data model is divided into the inner displacements ($\hat{\mathbf{u}}_{\text{I}}^{(n)}$) and outer displacements ($\hat{\mathbf{u}}_{\text{O}}^{(n)}$), and the equilibrium equation is as follows:

$$\hat{\mathbf{K}}^{(n)}\begin{bmatrix}\hat{\mathbf{u}}_{\text{I}}\\ \hat{\mathbf{u}}_{\text{O}}\end{bmatrix} = \begin{bmatrix}\mathbf{0}\\ \hat{\mathbf{R}}_{\text{O}}\end{bmatrix} \quad \text{with} \quad \hat{\mathbf{K}}^{(n)} = \begin{bmatrix}\hat{\mathbf{K}}_{\text{II}} & \hat{\mathbf{K}}_{\text{IO}}\\ \hat{\mathbf{K}}_{\text{OI}} & \hat{\mathbf{K}}_{\text{OO}}\end{bmatrix}, \tag{4-9}$$

where $\hat{\mathbf{K}}^{(n)}$ is the stiffness matrix of the reference data model and $\hat{\mathbf{R}}_{\text{O}}$ is the reaction force vector.

Then, the inner nodal displacements are calculated using

$$\hat{\mathbf{u}}_{\text{I}}^{(n)} = -\left(\hat{\mathbf{K}}_{\text{II}}\right)^{-1}\hat{\mathbf{K}}_{\text{IO}}\hat{\mathbf{u}}_{\text{O}}^{(n)}. \tag{4-10}$$

Note that the inner nodal displacements are obtained regardless of Young's modulus, but depend on Poisson's ratio. For this reason, Young's modulus was not randomly applied (see Appendix A).

Using the displacements calculated in Eq. (4-10), the strain field of the reference data model is obtained. Then, strain values are extracted from the reference data model at the points corresponding to the 3×3 Gauss points in element n. The strain vector corresponding to Gauss point (i, j) is defined by

$${}^{(i,j)}\hat{\boldsymbol{\varepsilon}}^{(n)} = \begin{bmatrix}{}^{(i,j)}\hat{\varepsilon}_{xx}^{(n)} & {}^{(i,j)}\hat{\varepsilon}_{yy}^{(n)} & {}^{(i,j)}\hat{\gamma}_{xy}^{(n)}\end{bmatrix}^{\text{T}}. \tag{4-11}$$

Poisson's ratio, the nodal coordinates of the normalized geometry, the nodal displacements, and the strain values extracted from the reference data model are made into one training data. The n^{th} training data ($\mathbf{D}^{(n)}$) is configured as

$$\mathbf{D}^{(n)} = \begin{bmatrix}\nu^{(n)} & \mathbf{D}_x^{(n)} & \mathbf{D}_u^{(n)} & \mathbf{D}_\varepsilon^{(n)}\end{bmatrix} \tag{4-12}$$

with

$$\mathbf{D}_x^{(n)} = \begin{bmatrix}\mathbf{x}_3^{(n)\text{T}} & \mathbf{x}_4^{(n)\text{T}}\end{bmatrix},\ \mathbf{D}_u^{(n)} = \begin{bmatrix}\mathbf{u}_1^{(n)\text{T}} & \cdots & \mathbf{u}_8^{(n)\text{T}}\end{bmatrix} \text{ and } \mathbf{D}_\varepsilon^{(n)} = \begin{bmatrix}{}^{(1,1)}\hat{\boldsymbol{\varepsilon}}^{(n)\text{T}} & \cdots & {}^{(3,3)}\hat{\boldsymbol{\varepsilon}}^{(n)\text{T}}\end{bmatrix},$$

where $\nu^{(n)}$, $\mathbf{D}_x^{(n)}$, $\mathbf{D}_u^{(n)}$, and $\mathbf{D}_\varepsilon^{(n)}$ denote Poisson's ratio (1 value), nodal coordinates (2×2 values), nodal displacements (8×2 values), and strain values (3×3×3 values), respectively. In total, the n^{th} training data contain 48 values.

4.2.2 Network configuration and training

The output of the network is ${}^{(i,j)}_{\text{output}}\mathbf{B}^{(n)}$, the n^{th} normalized strain-displacement matrix (3×16) at Gauss point (i, j). The row and column of the matrix correspond to 3 strains and 16 nodal displacements (eight for u and eight for v), respectively.

Poisson's ratio ($\nu^{(n)}$) and the nodal coordinates ($\mathbf{D}_x^{(n)}$) of the training data are the inputs of the neural network, while the nodal displacements ($\mathbf{D}_u^{(n)}$) and strain values ($\mathbf{D}_\varepsilon^{(n)}$) of the training data are used for the following cost function $C(\boldsymbol{\theta})$:

$$C(\boldsymbol{\theta}) = \frac{1}{27M}\sum_{n=1}^{M}\sum_{i=1}^{3}\sum_{j=1}^{3}\sum_{k=1}^{3} {}^{(i,j)}w \left| \frac{\sum_{l=1}^{16}\left({}^{(i,j)}_{\text{output}}\mathrm{b}_{kl}^{(n)}(\boldsymbol{\theta})u_l^{(n)}\right) - {}^{(i,j)}\hat{\varepsilon}_k^{(n)}}{{}^{(i,j)}\hat{\varepsilon}_k^{(n)}} \right|, \tag{4-13}$$

where $\boldsymbol{\theta}$ denote the network weights, M is the number of training data, ${}^{(i,j)}_{\text{output}}\mathrm{b}_{kl}^{(n)}(\boldsymbol{\theta})$ is the component at the k^{th} row and l^{th} column of ${}^{(i,j)}_{\text{output}}\mathbf{B}^{(n)}(\boldsymbol{\theta})$, and ${}^{(i,j)}w$ denotes the weight factor corresponding to Gauss point (i, j). In Eq. (4-13), ${}^{(i,j)}\hat{\varepsilon}_1^{(n)} = {}^{(i,j)}\hat{\varepsilon}_{xx}^{(n)}$, ${}^{(i,j)}\hat{\varepsilon}_2^{(n)} = {}^{(i,j)}\hat{\varepsilon}_{yy}^{(n)}$, and ${}^{(i,j)}\hat{\varepsilon}_3^{(n)} = {}^{(i,j)}\hat{\gamma}_{xy}^{(n)}$.

It is physically essential that finite elements should produce zero-strain energy under rigid body translations and rotations. Accordingly, the strain-displacement matrix generated from the network should satisfy the following conditions:

$${}^{(i,j)}_{\text{output}}\mathbf{B}^{(n)}(\boldsymbol{\theta})\Delta\mathbf{u} = \mathbf{0}, \tag{4-14}$$

in which $\Delta\mathbf{u}$ is the displacement vector corresponding to rigid body translations.

The three strain components (ε_{xx}, ε_{yy} and γ_{xy}) should be zero for the x- and y-directional rigid body translations at all 3×3 Gauss points, which yields the following equations:

$$\sum_{l=1}^{8} {}^{(i,j)}_{\text{output}}\mathrm{b}_{kl}^{(n)}(\boldsymbol{\theta}) = 0 \quad \text{and} \quad \sum_{l=9}^{16} {}^{(i,j)}_{\text{output}}\mathrm{b}_{kl}^{(n)}(\boldsymbol{\theta}) = 0 \quad \text{for } i, j, k=1, 2, 3. \tag{4-15}$$

Note that the 1st to 8th columns correspond to the x-directional displacements (u) and the 9th to 16th columns correspond to the y-directional displacements (v) as shown in Eq. (4-5).

To enforce the matrix ${}^{(i,j)}_{\text{output}}\mathbf{B}^{(n)}(\boldsymbol{\theta})$ complying the constraints in Eq. (4-15), we generate an intermediate strain-displacement matrix ${}^{(i,j)}\breve{\mathbf{B}}^{(n)}(\boldsymbol{\theta})$ in the network. The intermediate matrix contains only the first 7 columns for each x- and y-directional displacements. That is, the 8th and 16th columns of ${}^{(i,j)}_{\text{output}}\mathbf{B}^{(n)}(\boldsymbol{\theta})$ are excluded and thus the intermediate matrix has the dimensions of 3×14. Then, the excluded columns are calculated according to Eq.

(4-15) to produce the 3×16 matrix ${}^{(i,j)}_{\text{output}}\mathbf{B}^{(n)}(\boldsymbol{\theta})$ in the network:

$$ {}^{(i,j)}_{\text{output}}\mathrm{b}^{(n)}_{kl}(\boldsymbol{\theta}) = \begin{cases} {}^{(i,j)}\breve{\mathrm{b}}^{(n)}_{kl}(\boldsymbol{\theta}), & l = 1,2,\cdots,7 \\ -\sum_{l=1}^{7} {}^{(i,j)}\breve{\mathrm{b}}^{(n)}_{kl}(\boldsymbol{\theta}), & l = 8 \\ {}^{(i,j)}\breve{\mathrm{b}}^{(n)}_{k(l-1)}(\boldsymbol{\theta}), & l = 9,10,\cdots,15 \\ -\sum_{l=8}^{14} {}^{(i,j)}\breve{\mathrm{b}}^{(n)}_{kl}(\boldsymbol{\theta}), & l = 16 \end{cases} \quad \text{for } i,\ j,\ k=1,\ 2,\ 3\ , \tag{4-16} $$

where ${}^{(i,j)}\breve{\mathrm{b}}^{(n)}_{kl}(\boldsymbol{\theta})$ are the components of the intermediate matrix ${}^{(i,j)}\breve{\mathbf{B}}^{(n)}(\boldsymbol{\theta})$.

A network was constructed as shown in Fig. 4.4. A fully connected neural network of 6 layers was employed because its structure can be used as a universal approximator [32, 33], and batch normalization was applied to each layer. An exponential linear unit was used as an activation function at each layer before the output. The network width was 378 (= 3×3×3×14), which was the same as the number of all components of ${}^{(i,j)}\breve{\mathbf{B}}^{(n)}(\boldsymbol{\theta})$ (3×14) generated at 3×3 Gauss points. After the 378 outputs were reshaped, ${}^{(i,j)}\breve{\mathbf{B}}^{(n)}(\boldsymbol{\theta})$ was generated at each Gauss point. Finally, ${}^{(i,j)}_{\text{output}}\mathbf{B}^{(n)}(\boldsymbol{\theta})$ was obtained using Eq. (4-16).

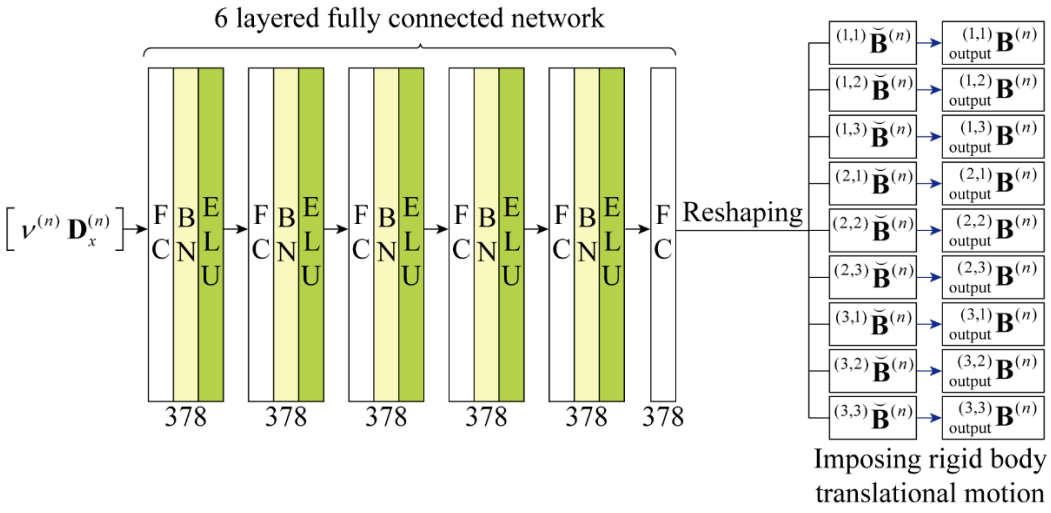

Figure 4.4. Network configuration for deep learned finite elements. (FC: fully connected layer, BN: batch normalization, ELU: exponential linear unit).

The network was trained with a total of 300,000 data (M = 300,000). To test the network, 50,000 data were generated of which 30,000 data were randomly selected for testing. When the strain value ${}^{(i,j)}\hat{\varepsilon}^{(n)}_{k}$ is close to 0, the cost function $C(\boldsymbol{\theta})$ in Eq. (4-13) increases sensitively even though the network generates a strain close enough to the input strains ($\mathbf{D}^{(n)}_{\varepsilon}$). Therefore, training data containing an absolute value of less than 0.005 in the components of $\mathbf{D}^{(n)}_{\varepsilon}$ were excluded.

When such training data were excluded in this way, the generated element largely failed in the shearing patch test because data corresponding to pure shearing were not included in training data. Therefore, additional training data for pure shearing were generated and used for network training. We generated 3000 x-directional shearing data by applying $\mathbf{u}_i^{(n)} = \begin{bmatrix} y_i^{(n)} & 0 \end{bmatrix}^{\mathrm{T}}$, and 3000 y-directional shearing data by applying $\mathbf{u}_i^{(n)} = \begin{bmatrix} 0 & x_i^{(n)} \end{bmatrix}^{\mathrm{T}}$. In addition, 1200 test data (600 shearing data for each direction) were generated. The zero-strain components generated from the additional data were replaced with 0.5% of the maximum strain component in each data because the cost function did not converge well when the zero values were trained.

The network was implemented using TensorFlow [34], Adam optimization [35] was adopted as the optimizer, and Xavier initializer [36] was applied to initialize the weights of the network. We performed training for 30,000 epochs, and a batch size of 50,000 was used. The learning rate converged linearly from 0.01 to 0 as the epoch progressed. In the early stage of training, the learning rate of the network weights was set to a large value to broadly search $C(\boldsymbol{\theta})$. Then, the learning rate was gradually decreased to zero, and thereby $C(\boldsymbol{\theta})$ precisely reached the minimum value. As a result of training, the average error for the training data was 1.24% and the average error for the test data was 1.67%.

4.2.3 Construction of the stiffness matrix

Normalized geometries were employed for the efficient training of the network. In order to apply the trained network to elements with arbitrary geometries, pre-processing for the input of the trained network is necessary. In addition, post-processing of the network output is required to generate the stiffness matrix.

4.2.3.1 Pre-processing of the network input

Geometry normalization is performed for an element with an arbitrary geometry. The element connectivity is assigned so that the side length between node 1 and node 2 is the longest. Then, as shown in Fig. 4.5, the nodal coordinates of the element ($\mathbf{x}_i$) are translated, rotated, and resized to obtain the input normalized nodal coordinates (${}_{\text{input}}\mathbf{x}_i$) where node 1 and node 2 are positioned at (0, 0) and (1, 0), respectively.

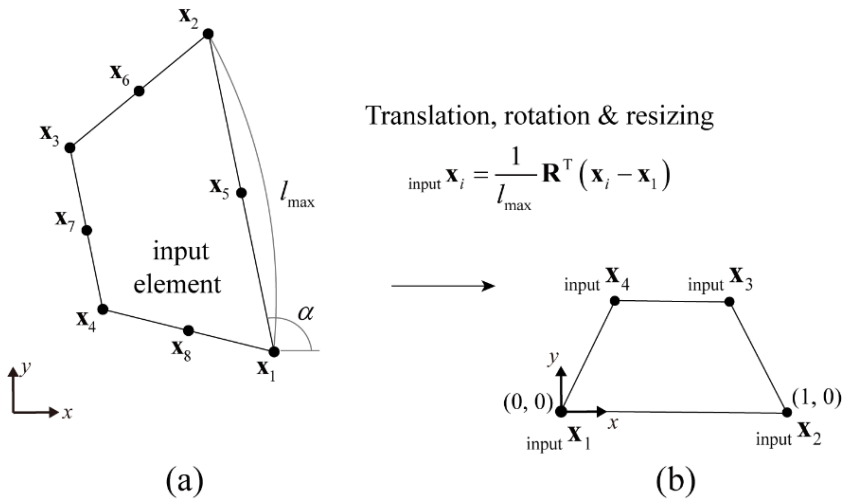

Figure 4.5. Pre-processing procedure to obtain the network input. (a) Original element geometry. (b) Normalized element geometry.

The normalized nodal coordinates are obtained by

$$ {}_{\text{input}}\mathbf{x}_i = \frac{1}{l_{\max}} \mathbf{R}^{\mathrm{T}} \left(\mathbf{x}_i - \mathbf{x}_1 \right) \qquad \text{for} \quad i = 1, 2, 3, 4 \tag{4-17} $$

with $\mathbf{R} = \begin{bmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{bmatrix}$,

in which α is the angle between the longest side and the *x*-axis (see Fig. 4.5(a)), and $l_{\max}$ is the longest side length. The coordinates of ${}_{\text{input}}\mathbf{x}_3$ and ${}_{\text{input}}\mathbf{x}_4$ obtained from this process and Poisson's ratio ν are used as input data for the trained network.

4.2.3.2 Post-processing of the network output

The trained network outputs the strain-displacement matrices for the element (${}^{(i,j)}_{\text{output}}\mathbf{B}$) of normalized geometry at every Gauss point. The post-processing for the network output is performed to calculate the strain-displacement matrix of the element (${}^{(i,j)}_{\text{DL8}}\mathbf{B}$) of the original geometry, as shown in Fig. 4.6.

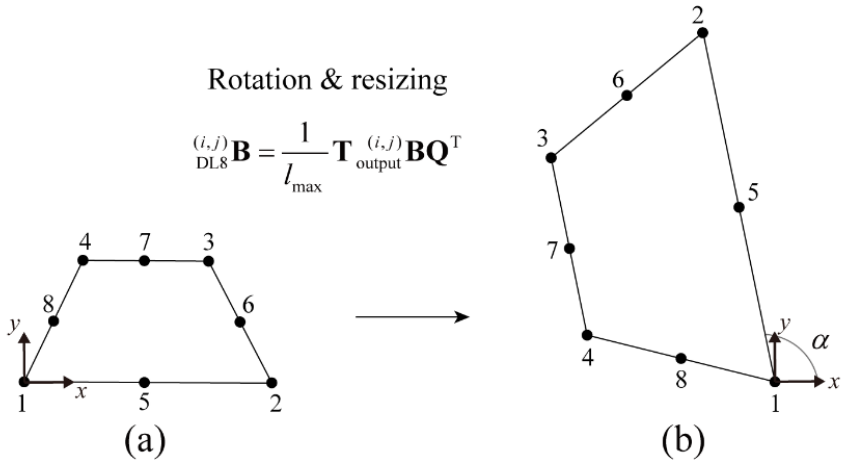

Figure 4.6. Post-processing procedure for the network output to obtain strain-displacement matrices for the original element geometry. (a) Normalized element geometry. (b) Original element geometry.

The strain-displacement matrix $^{(i,j)}_{\mathrm{DL8}}\mathbf{B}$ is obtained using the following equation

$$^{(i,j)}_{\mathrm{DL8}}\mathbf{B} = \frac{1}{l_{\max}}\mathbf{T}\,{}^{(i,j)}_{\mathrm{output}}\mathbf{B}\mathbf{Q}^{\mathrm{T}}, \tag{4-18}$$

with

$$\mathbf{T} = \begin{bmatrix} \cos^2\alpha & \sin^2\alpha & -\sin\alpha\cos\alpha \\ \sin^2\alpha & \cos^2\alpha & \sin\alpha\cos\alpha \\ 2\sin\alpha\cos\alpha & -2\sin\alpha\cos\alpha & \cos^2\alpha-\sin^2\alpha \end{bmatrix},$$

$\mathbf{Q} = \left(q_{kl}\right) \in \mathbb{R}^{16\times 16}$ with $q_{kl} = \delta_{kl}\cos\alpha - \delta_{k(l-8)}\sin\alpha + \delta_{(k-8)l}\sin\alpha$,

in which $\mathbf{T}$ is the strain transformation matrix [37], $\mathbf{Q}$ is the displacement rotation matrix, and δ_{kl} represents the Kronecker delta. Note that $\mathbf{Q}$ in Eq. (4-18) is given according to the order of the components of $\mathbf{u}$ in Eq. (4-5).

Since $^{(i,j)}_{\mathrm{DL8}}\mathbf{B}$ is the strain-displacement matrix at Gauss point (i, j) approximated by the network, it is very hard to make the element pass the patch tests exactly. Therefore, our goal is for the element to pass the patch tests as close as possible. To do so, we correct the matrix using the well-known B-bar method [38]. The corrected strain-displacement matrix $^{(i,j)}_{\mathrm{DL8}}\overline{\mathbf{B}}$ is obtained as

$$^{(i,j)}_{\mathrm{DL8}}\overline{\mathbf{B}} = {}^{(i,j)}_{\mathrm{DL8}}\mathbf{B} + {}_{\mathrm{DL8}}\mathbf{B}' \quad \text{with} \quad {}_{\mathrm{DL8}}\mathbf{B}' = \frac{t}{V}\sum_{i=1}^{3}\sum_{j=1}^{3} {}^{(i,j)}w\left({}^{(i,j)}_{\mathrm{Q8}}\mathbf{B} - {}^{(i,j)}_{\mathrm{DL8}}\mathbf{B}\right){}^{(i,j)}J, \tag{4-19}$$

where V is the element volume and $^{(i,j)}_{\mathrm{Q8}}\mathbf{B}$ is the strain-displacement matrix of the standard 8-node quadrilateral element at Gauss point (i, j) [31].

Using the corrected strain-displacement matrix $^{(i,j)}_{\mathrm{DL8}}\overline{\mathbf{B}}$, the stiffness matrix of the element is finally calculated by

$$ {}_{\text{DL8}}\mathbf{K} = t\sum_{i=1}^{3}\sum_{j=1}^{3} {}^{(i,j)}w\, {}^{(i,j)}_{\text{DL8}}\overline{\mathbf{B}}^{\text{T}}\, \mathbf{C}\, {}^{(i,j)}_{\text{DL8}}\overline{\mathbf{B}}\, {}^{(i,j)}J \,. \tag{4-20} $$

In the deep learned finite elements developed in this study, a strain-displacement matrix is only generated at Gauss points. Therefore, strain and stress values are calculated at Gauss points. The strain and stress fields in the element are obtained by extrapolating the strain and stress values at the Gauss points.

4.3 Procedures to generate 4-node deep learned finite elements

This chapter describes the procedure to generate the strain-displacement matrix of a 4-node quadrilateral element and to obtain its stiffness matrix. The 4-node element is degenerated from the 8-node deep learned quadrilateral element obtained in Chapter 4.2

The corner and mid-side nodes of the 8-node element are considered as compatible and incompatible nodes, respectively, as shown in Fig. 4.7.

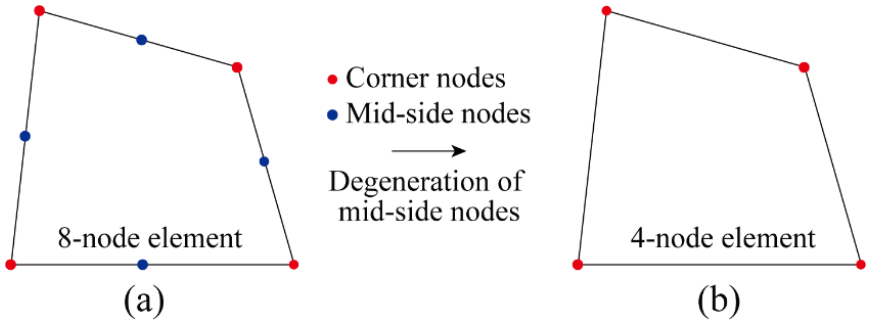

Figure 4.7. Degeneration process for the 4-node deep learned quadrilateral element. (a) 8-node deep learned finite element generated from the trained network. (b) 4-node deep learned element obtained through the degeneration of mid-side nodes.

The strain-displacement matrix for the 8-node deep learned quadrilateral element, ${}^{(i,j)}_{\text{DL8}}\mathbf{B}$ in Eq. (4-18), is divided into two parts corresponding to the displacements at compatible and incompatible nodes as follows:

$$ {}^{(i,j)}_{\text{DL8}}\mathbf{B}\mathbf{u} = \begin{bmatrix} {}^{(i,j)}\mathbf{B}_{\text{C}} & {}^{(i,j)}\mathbf{B}_{\text{I}} \end{bmatrix} \begin{bmatrix} \mathbf{u}_{\text{C}} \\ \mathbf{u}_{\text{I}} \end{bmatrix}, \tag{4-21} $$

where ${}^{(i,j)}\mathbf{B}_{\text{C}}$ and ${}^{(i,j)}\mathbf{B}_{\text{I}}$ are the strain-displacement matrices corresponding to displacements at compatible and incompatible nodes, respectively, and $\mathbf{u}_{\text{C}}$ and $\mathbf{u}_{\text{I}}$ are displacement vectors corresponding to compatible and incompatible nodes, respectively.

To make the element pass the patch test more closely, the strain-displacement matrices are corrected as follows

$$ {}^{(i,j)}\overline{\mathbf{B}}_{\mathrm{C}} = {}^{(i,j)}\mathbf{B}_{\mathrm{C}} + \mathbf{B}'_{\mathrm{C}} \quad \text{and} \quad {}^{(i,j)}\overline{\mathbf{B}}_{\mathrm{I}} = {}^{(i,j)}\mathbf{B}_{\mathrm{I}} + \mathbf{B}'_{\mathrm{I}}, \tag{4-22} $$

with

$$ \mathbf{B}'_{\mathrm{C}} = \frac{t}{V}\sum_{i=1}^{3}\sum_{j=1}^{3} {}^{(i,j)}w\left({}^{(i,j)}_{\mathrm{Q4}}\mathbf{B} - {}^{(i,j)}\mathbf{B}_{\mathrm{C}} \right) {}^{(i,j)}J \,, $$

$$ \mathbf{B}'_{\mathrm{I}} = -\frac{t}{V}\sum_{i=1}^{3}\sum_{j=1}^{3} {}^{(i,j)}w^{(i,j)}\mathbf{B}_{\mathrm{I}} {}^{(i,j)}J \,, $$

in which ${}^{(i,j)}\overline{\mathbf{B}}_{\mathrm{C}}$ and ${}^{(i,j)}\overline{\mathbf{B}}_{\mathrm{I}}$ are the corrected strain-displacement matrices and ${}^{(i,j)}_{\mathrm{Q4}}\mathbf{B}$ is the strain-displacement matrix of the standard 4-node element.

Using the corrected ${}^{(i,j)}\overline{\mathbf{B}}_{\mathrm{C}}$ and ${}^{(i,j)}\overline{\mathbf{B}}_{\mathrm{I}}$ in Eq. (4-22), the stiffness matrix is calculated as

$$ \mathbf{K} = \begin{bmatrix} \mathbf{K}_{\mathrm{CC}} & \mathbf{K}_{\mathrm{CI}} \\ \mathbf{K}_{\mathrm{IC}} & \mathbf{K}_{\mathrm{II}} \end{bmatrix} = t\sum_{i=1}^{3}\sum_{j=1}^{3} {}^{(i,j)}w\, {}^{(i,j)}_{\mathrm{DL4}}\mathbf{B}^{\mathrm{T}}\, \mathbf{C}\, {}^{(i,j)}_{\mathrm{DL4}}\mathbf{B}\, {}^{(i,j)}J \tag{4-23} $$

with

$$ {}^{(i,j)}_{\mathrm{DL4}}\mathbf{B} = \begin{bmatrix} {}^{(i,j)}\overline{\mathbf{B}}_{\mathrm{C}} & {}^{(i,j)}\overline{\mathbf{B}}_{\mathrm{I}} \end{bmatrix}. $$

The submatrices related to incompatible displacements in Eq. (4-23) are eliminated using the static condensation procedure. Finally, the stiffness matrix of the deep learned 4-node finite element is obtained

$$ {}_{\mathrm{DL4}}\mathbf{K} = \mathbf{K}_{\mathrm{CC}} - \mathbf{K}_{\mathrm{CI}}\left(\mathbf{K}_{\mathrm{II}}\right)^{-1}\mathbf{K}_{\mathrm{IC}} \,. \tag{4-24} $$

4.4 Basic numerical tests

In this chapter, zero energy mode and patch tests are performed for the deep learned 8-node (DL8) and 4-node (DL4) finite elements.

4.4.1 Zero energy mode tests

In the zero energy mode test, the zero eigenvalues of the stiffness matrix of a single unsupported element are counted. Undistorted (in Fig. 4.8(a)) and distorted (in Fig. 4.8(b), (c) and (d)) element geometries are considered with unit thickness. Young's modulus $E = 1.5\times10^{3}$ and Poisson's ratio $\nu = 0.3$ are given.

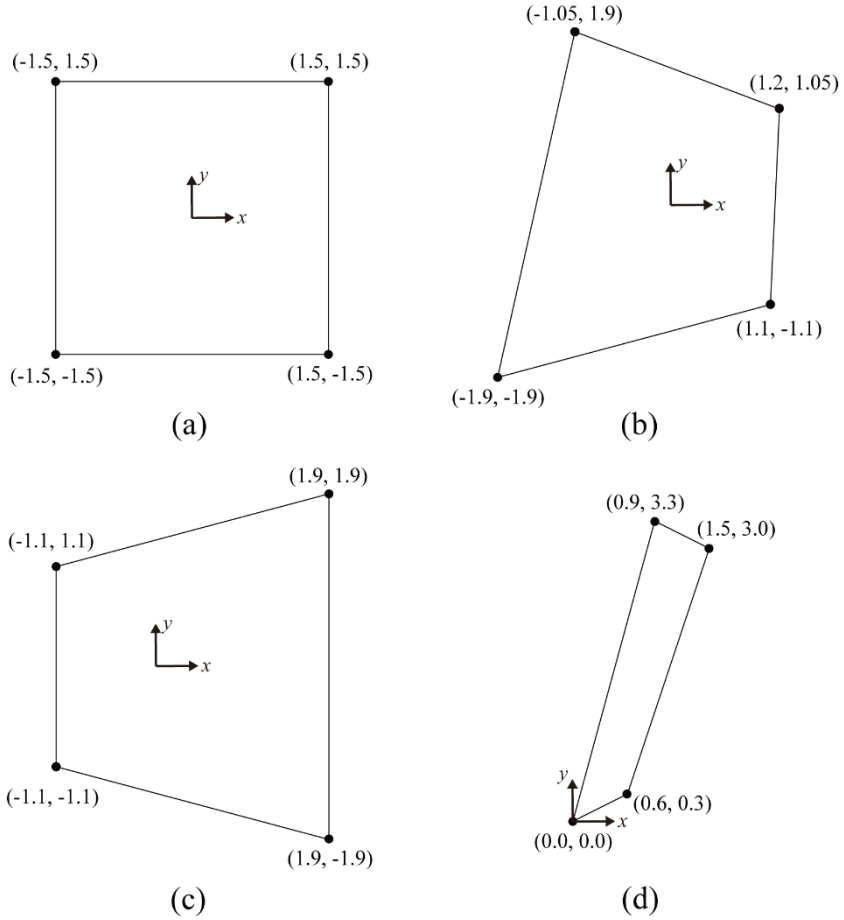

Figure 4.8. Element geometries used for the zero energy mode test: (a) Geometry 1, (b) Geometry 2, (c) Geometry 3, and (d) Geometry 4.

Table 4.1 presents the eigenvalues calculated up to the sixth strain energy modes. The first three eigenvalues correspond to the three rigid body modes (two translations and one rotation modes) for all geometry cases. The eigenvalue of mode 3 (rotation mode) shows larger than those of mode 1 and 2 (translation modes). However, the three eigenvalues are sufficiently smaller than those of the deformation modes (modes 4, 5 and 6) and thus the practical use of the DL8 and DL4 elements is available.

Table 4.1. Eigenvalues corresponding to the 1st~6th modes for the various geometries (in Fig. 4.8) of DL8 and DL4 elements. (The 1st, 2nd and 3rd modes correspond to rigid body motions.)

Mode	Geometry 1		Geometry 2		Geometry 3		Geometry 4	
	DL8	DL4	DL8	DL4	DL8	DL4	DL8	DL4
1	5.88E-14	4.82E-13	3.32E-13	1.83E-13	1.31E-13	8.31E-13	2.45E-13	7.93E-13
2	3.83E-13	9.43E-13	4.84E-13	7.84E-12	6.68E-13	1.25E-12	7.50E-13	2.86E-12
3	3.36E-03	1.31E-03	2.24E-04	4.87E-03	1.04E-04	5.01E-03	1.44E-03	2.33E-03
4	4.16E+02	1.15E+03	4.33E+02	4.36E+02	1.96E+02	4.69E+02	1.80E+01	7.41E+01

5	4.19E+02	1.15E+03	4.71E+02	5.25E+02	3.87E+02	5.02E+02	1.04E+02	2.73E+02
6	5.07E+02	4.97E+03	4.98E+02	1.14E+03	4.92E+02	1.18E+03	1.23E+02	2.12E+03

4.4.2 Patch tests

Three patch tests are performed with the mesh geometry in Fig. 4.9(a) for *x*- and *y*-directional stretching and shearing [1]. The loading and displacement boundary conditions are shown in Fig. 4.9(b)–(d). The patch of elements is subjected to the minimum number of constraints to prevent rigid body motions and nodal point forces on the boundary corresponding to constant stress states are applied. If the constant stress fields are calculated, the patch tests are passed [39, 40].

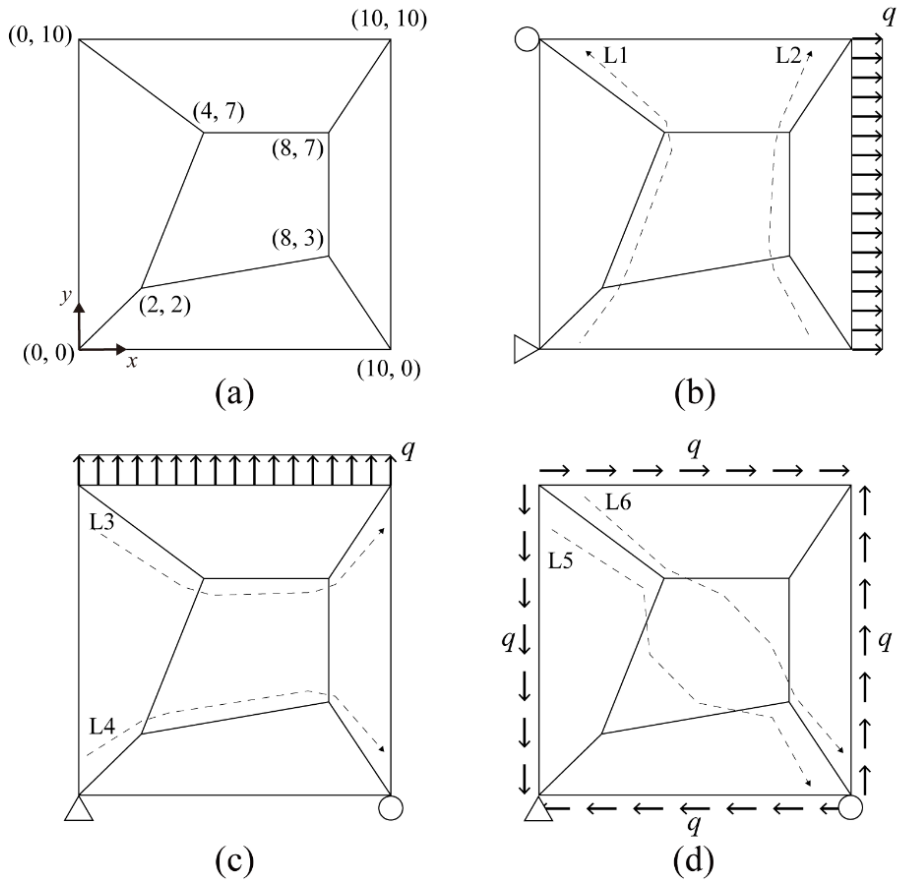

Figure 4.9. Mesh geometry used for the patch tests is shown in (a) ($q = 1.0$, thickness $= 1.0$, $E = 3.0 \times 10^7$, $\nu = 0.3$). Loading and boundary conditions and the lines through element Gauss points for stress evaluation are shown in (b), (c) and (d).

The deep learned finite elements (DL8 and DL4) practically pass the patch tests. The results of the patch test are shown in Fig. 4.10. Table 4.2 presents the minimum and maximum stress values across Gauss points. In other words, the deep learned finite elements can represent constant strain fields with practically sufficient accuracy

Table 4.2. Minimum and maximum stress values across Gauss points in the patch tests (minimum value – maximum value).

		σ_{xx}	σ_{yy}	σ_{xy}
x-directional stretch in Fig. 4.9(b)	DL8 element	0.9950 – 1.0055	-0.0043 – 0.0037	-0.0055 – 0.0040
	DL4 element	0.9810 – 1.0169	-0.0074 – 0.0051	-0.0140 – 0.0134
	Ref. values	1.0	0.0	0.0
y-directional stretch in Fig. 4.9(c)	DL8 element	-0.0055 – 0.0050	0.9932 – 1.0065	-0.0051 – 0.0057
	DL4 element	-0.0057 – 0.0069	0.9895 – 1.0144	-0.0188 – 0.0264
	Ref. values	0.0	1.0	0.0
Shearing in Fig. 4.9(d)	DL8 element	-0.0059 – 0.0062	-0.0044 – 0.0059	0.9918 – 1.0052
	DL4 element	-0.0097 – 0.0073	-0.0050 – 0.0033	0.9884 – 1.0139
	Ref. values	0.0	0.0	1.0

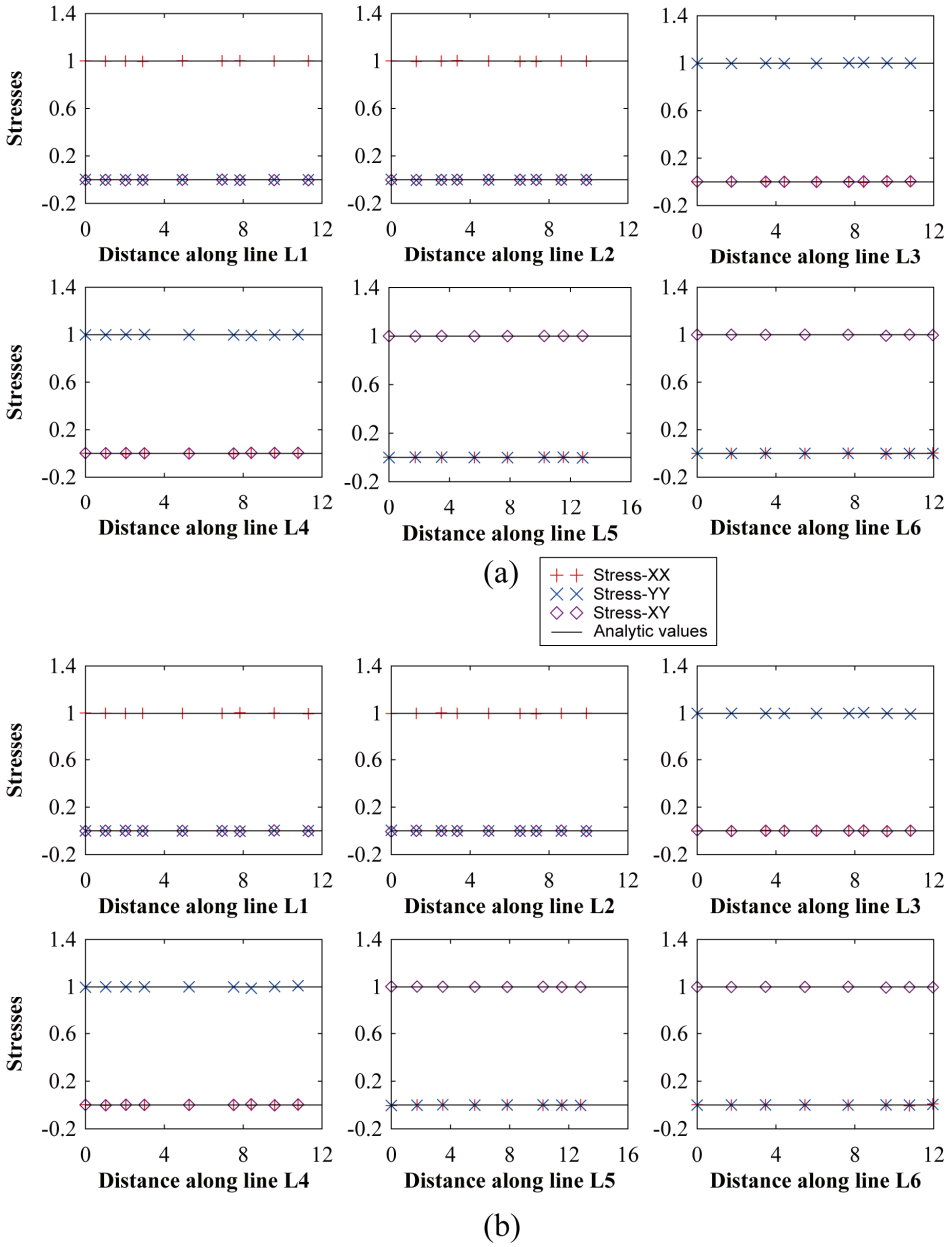

Figure 4.10. Stresses along lines L1–L6 for the patch tests: (a) DL8 element. (b) DL4 element.

4.5 Numerical examples

In this chapter, the performance of the proposed elements (DL8 and DL4) is investigated through various numerical problems: Cook's skew beam problem, taper beam problem, block problem, cantilever beam problem, and wrench problem.

The obtained results are compared with those of various existing elements as follows:

- Q4: Standard 4-node quadrilateral element [31]

- QM6: 4-node quadrilateral element with incompatible modes [41]
- Q8: Standard 8-node quadrilateral element [31]
- Q9: Standard 9-node quadrilateral element [31]
- P182: 4-node quadrilateral element in ANSYS [42]
- P183: 8-node quadrilateral element in ANSYS [42]

Therefore, 4 quadratic elements (Q8, Q9, P183 and DL8) and 4 linear elements (Q4, QM6, P182 and DL4) are considered. In general, 9-node elements have higher accuracy than 8-node elements, but 8-node elements have less degrees of freedom.

To investigate the predictive capability of the elements in detail, convergence studies are performed. The solution convergence is measured using the relative strain energy error given as

$$E_e = \left| \frac{E_{\text{ref}} - E_h}{E_{\text{ref}}} \right|, \tag{4-25}$$

in which E_{ref} and E_h denote the strain energy stored in the entire structure obtained from the reference and finite element solutions, respectively [43-45].

The optimal convergence of the relative strain energy error is estimated as $E_e \cong ch^{2k}$ where c is a constant, k = 1 and 2 for linear and quadratic elements, respectively, and h is the element size [1]. We use $h = 1/N$ for the linear elements and $1/2N$ for the quadratic elements.

4.5.1 Cook's skew beam problem

The skew beam problem proposed by Cook [46] is considered. The problem description on the geometry and boundary conditions is illustrated in Fig. 4.11. The skew beam is subjected to a distributed shearing force of magnitude q = 1/16 (force per length) at the right end, and the left end is clamped. Plane stress condition is employed with Young's modulus $E = 1.0$ and Poisson's ratio $\nu = 1/3$. The skew beam is discretized by using $N \times N$ element meshes (N = 2, 4, 8, 16, and 32).

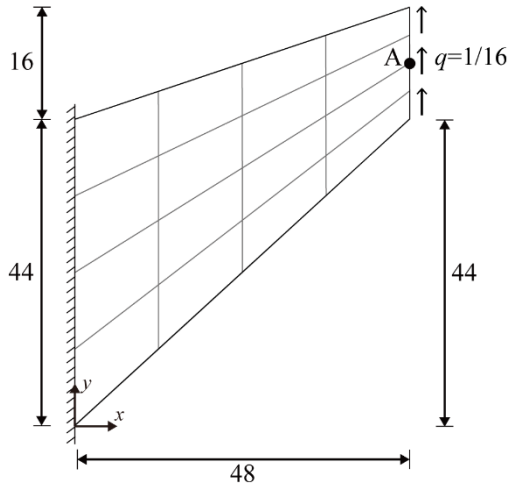

Figure 4.11. Cook's skew beam problem description ($E = 1.0$, $\nu = 1/3$, thickness $= 1.0$).

Table 4.3 shows the tip deflections (v_A) at point A (shown in Fig. 4.11) normalized by the reference solution. The reference solution is obtained using a 100×100 mesh of Q9 elements.

Table 4.3. Normalized deflections at point A in the Cook's skew beam problem (reference solution: 23.9662).

Mesh	Quadratic elements				Linear elements			
	Q8	Q9	P183	DL8	Q4	QM6	P182	DL4
2×2	0.9479	0.9717	0.9668	0.9868	0.4942	0.8783	0.8783	0.8756
4×4	0.9892	0.9947	0.9900	0.9992	0.7635	0.9604	0.9604	0.9606
8×8	0.9966	0.9983	0.9967	0.9995	0.9213	0.9884	0.9884	0.9883
16×16	0.9987	0.9993	0.9989	0.9996	0.9776	0.9965	0.9965	0.9964
32×32	0.9995	0.9998	0.9996	0.9997	0.9938	0.9989	0.9989	0.9988

Fig. 4.12 presents the convergence curves of the linear (Q4, QM6, P182 and DL4) and quadratic (Q8, Q9, P183 and DL8) elements. The DL8 element outperforms the other quadratic elements considered while the three linear elements (QM6, P182 and DL4) show almost the same convergence behavior.

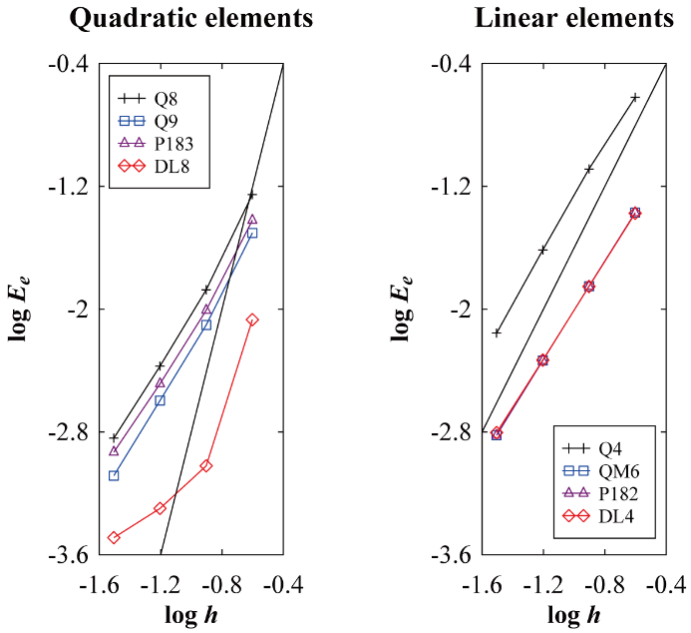

Figure 4.12. Convergence curves in the Cook's skew beam problem: The bold lines represent the optimal convergence rates.

Fig. 4.13 displays the distributions of the shear stress (τ_{xy}) obtained using the DL8 element when N = 2, 4, and 8. The reference stress distribution shown in Fig. 4.13(d) is given by a 64×64 mesh of Q9 elements. As N increases, the stress solution of the DL8 element well converge to the reference solution.

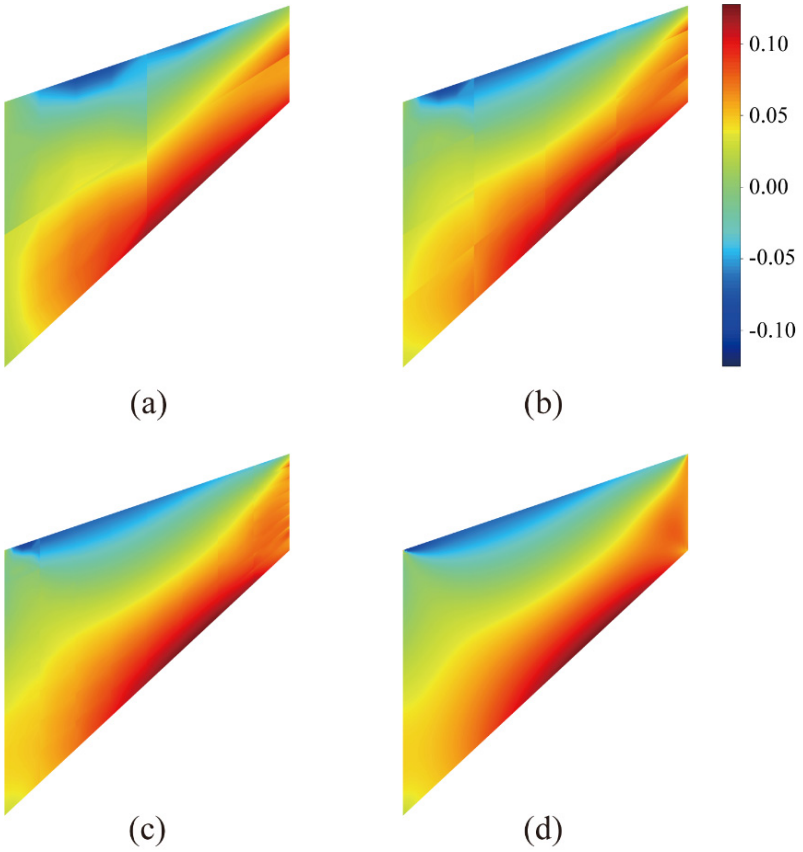

Figure 4.13. Stress distributions (τ_{xy}) calculated in Cook's skew beam problem using the DL8 element: (a) N = 2, (b) N = 4, and (c) N = 8. (d) Reference solution obtained using the Q9 element with N = 64.

4.5.2 Tapered beam problem

The tapered beam problem described in Fig. 4.14 is solved. The beam is subjected to the uniformly distributed load of q = 1 along the top side and the left side is fully clamped. Plane stress condition is considered with Young's modulus $E = 3.0\times10^2$ and Poisson's ratio $\nu = 0.3$. The taper beam is discretized by using $N \times 3N$ element meshes (N = 2, 4, 8, and 16).

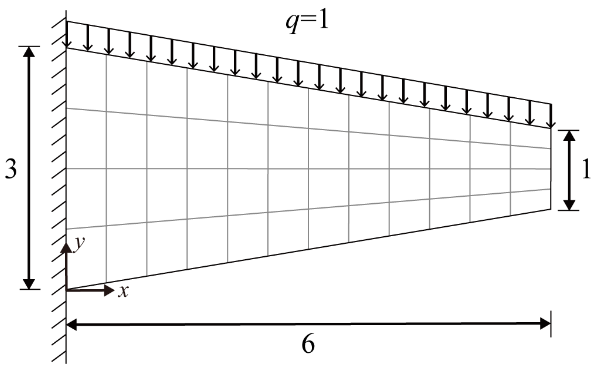

Figure 4.14. Tapered beam problem ($E = 3.0\times10^2$, $\nu = 0.3$, thickness = 0.1).

Fig. 4.15 displays the convergence curves for the linear and quadratic elements considered. The reference solution is obtained using a 64×64 mesh of Q9 elements. The DL8 element shows an improved convergence behavior compared to the other quadratic elements.

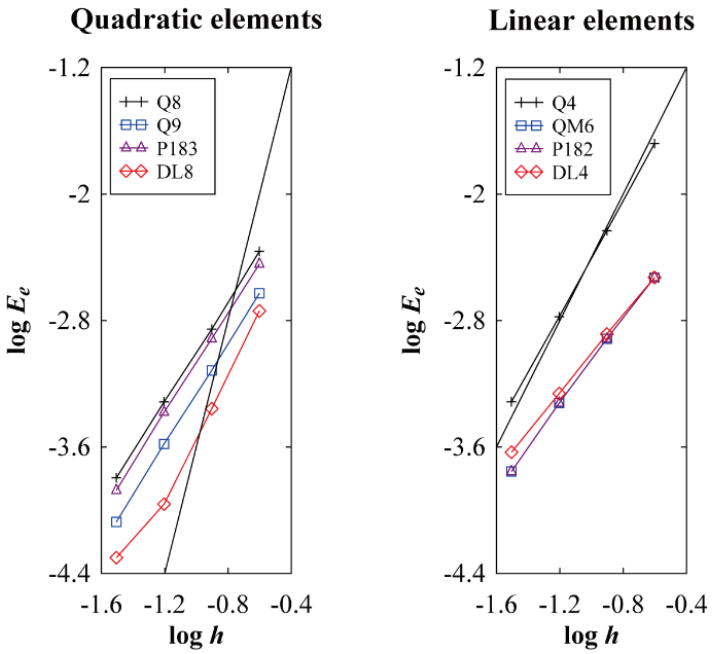

Figure 4.15. Convergence curves in the tapered beam problem: The bold lines represent the optimal convergence rates.

4.5.3 Block problem

The block problem described in Fig. 4.16 is investigated. The geometry and boundary conditions are shown in Fig. 4.16. A uniformly distributed load of $q = 1$ is applied with 45° tilted direction on the right half of the top side. The bottom side of the structure is fully clamped. Plane stress condition with Young's modulus $E = 3.0\times10^7$ and Poisson's ratio $\nu = 0.3$ is considered.

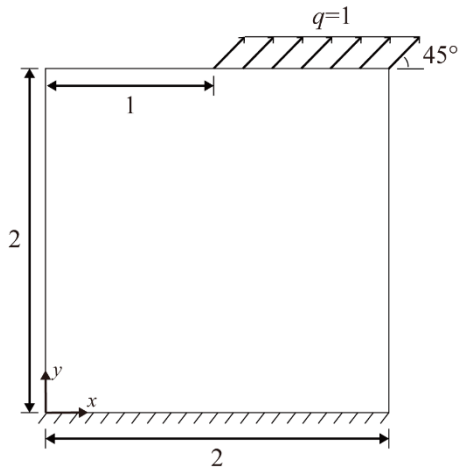

Figure 4.16. Block problem ($E = 3.0\times10^7$, $\nu = 0.3$, thickness $= 1.0$).

The regular and distorted meshes are employed as shown in Fig. 4.17(a) and (b). The distorted meshes are obtained by randomly repositioning the interior nodes of the corresponding regular meshes. The nodal coordinates of the distorted meshes (x' and y') are determined by

$$x' = x + \beta_x h \quad \text{and} \quad y' = y + \beta_y h, \tag{4-26}$$

in which x and y are respectively the *x*- and *y*-nodal coordinates of the regular meshes, and β_x and β_y are uniformly generated random numbers ranging from -0.35 to 0.35.

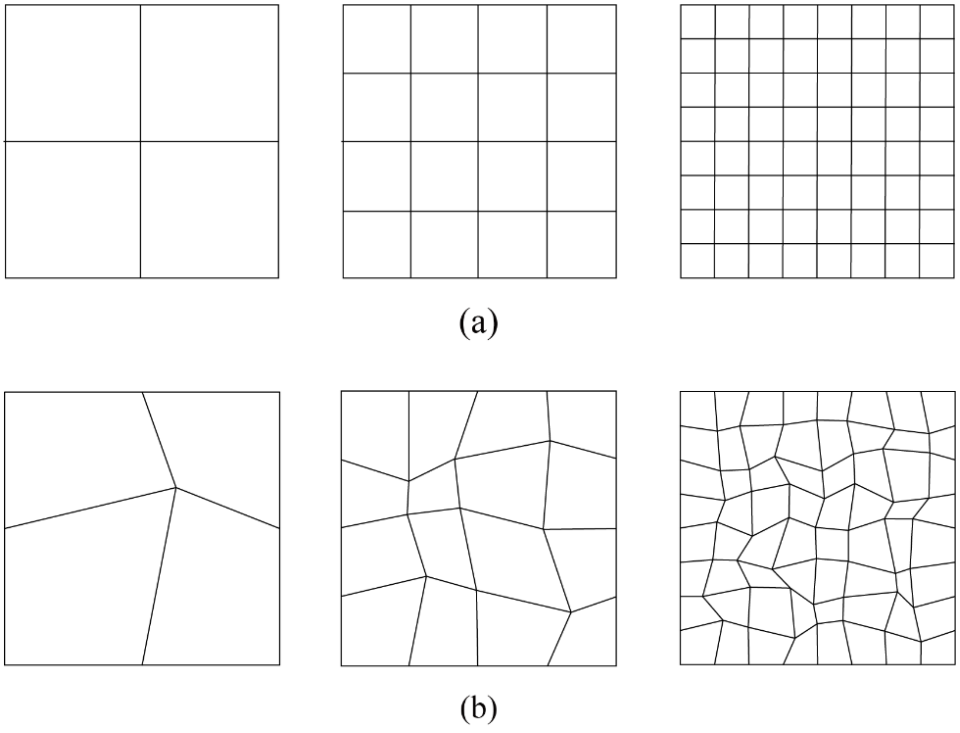

Figure 4.17. Meshes used for the block problem: (a) Regular meshes used with *N* = 2, 4 and 8. (b) Distorted

meshes used with $N = 2$, 4 and 8.

Fig. 4.18 presents the convergence curves for the linear and quadratic elements. The reference solution is obtained using a 64 × 64 regular mesh of Q9 elements. The DL8 element shows the superior convergence behavior even in distorted meshes.

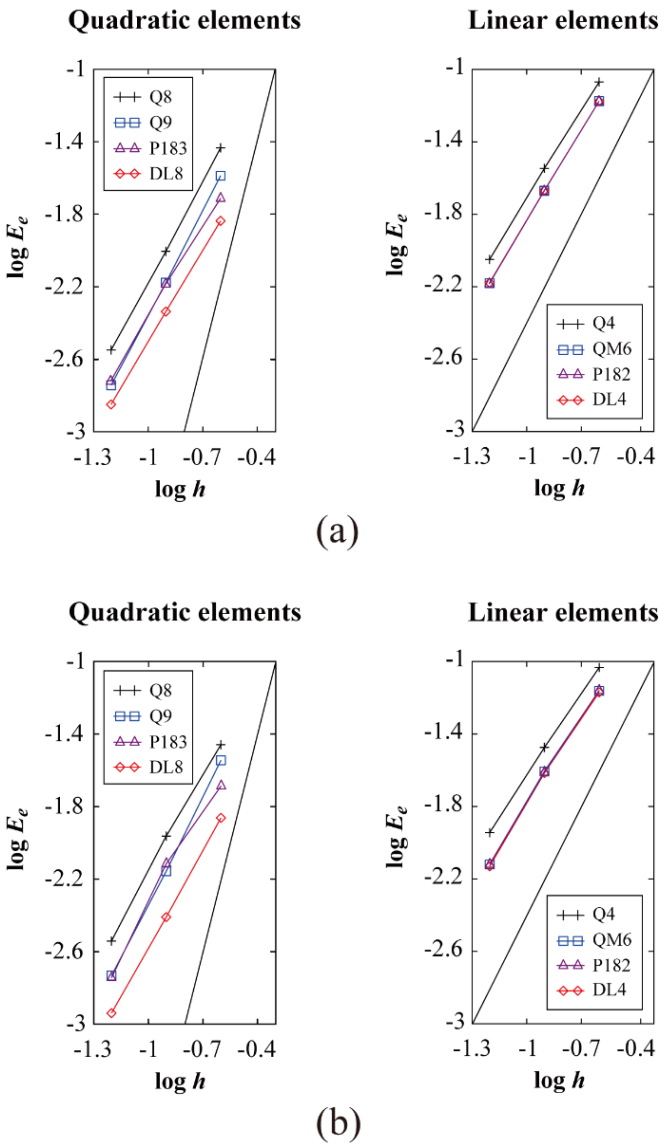

Figure 4.18. Convergence curves in the block problem in (a) regular meshes and (b) distorted meshes: The bold lines represent the optimal convergence rates.

4.5.4 Cantilever beam problem

Finally, we consider the cantilever beam problem as shown in Fig. 4.19. The beam is clamped at the left end and subjected to a uniformly distributed load of $q = 1$ at the free tip. Plane stress conditions is considered with Young's modulus $E = 1.0 \times 10^7$ and Poisson's ratio $\nu = 0.3$. Three different mesh patterns of meshes shown in Fig.

4.19(a)–(c) are adopted.

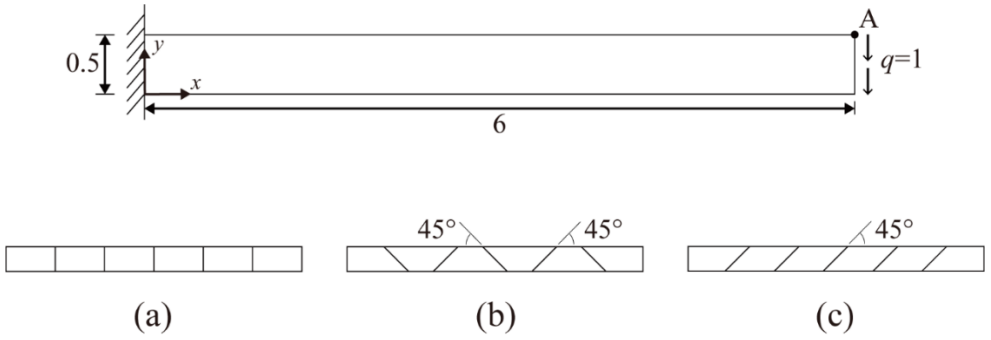

Figure 4.19. Cantilever beam problem ($E = 1.0\times10^{7}$, $\nu = 0.3$, thickness $= 0.1$). (a) Regular mesh. (b) Trapezoidal mesh. (c) Parallelogram mesh.

Table 4.4 shows the vertical displacements at point A normalized by the reference solution. The reference solution is obtained using a 10 × 60 regular mesh of Q9 elements. The Q8 element shows the performance deterioration in the trapezoidal mesh pattern. The performance of the DL8 element is comparable to that of the Q9 element despite having fewer DOFs.

Table 4.4. Normalized vertical displacements at point A in the cantilever beam problem (reference solution: -3.4694×10^{-3}).

Mesh	Quadratic elements				Linear elements			
	Q8	Q9	P183	DL8	Q4	QM6	P182	DL4
Rectangular	0.9861	0.9935	0.9877	0.9862	0.3796	0.9937	0.9937	1.0115
Trapezoidal	0.8984	0.9877	0.9704	0.9940	0.1351	0.2064	0.2064	0.2080
Parallelogram	0.9888	0.9898	0.9997	0.9866	0.1492	0.7932	0.7932	0.7888

4.5.5 Wrench problem

We consider the wrench problem described in Fig. 4.20. The geometry and boundary conditions are shown in Fig. 4.20. A uniformly distributed load of q = 10^6 is applied along the line AB. The plane stress condition is considered using Young's modulus $E = 2.0\times10^{11}$ and Poisson's ratio $\nu = 0.3$. Three different meshes are considered, as shown in Fig. 4.20(a)–(c). The reference solution is obtained using Q9 elements and the mesh in Fig. 4.20(d).

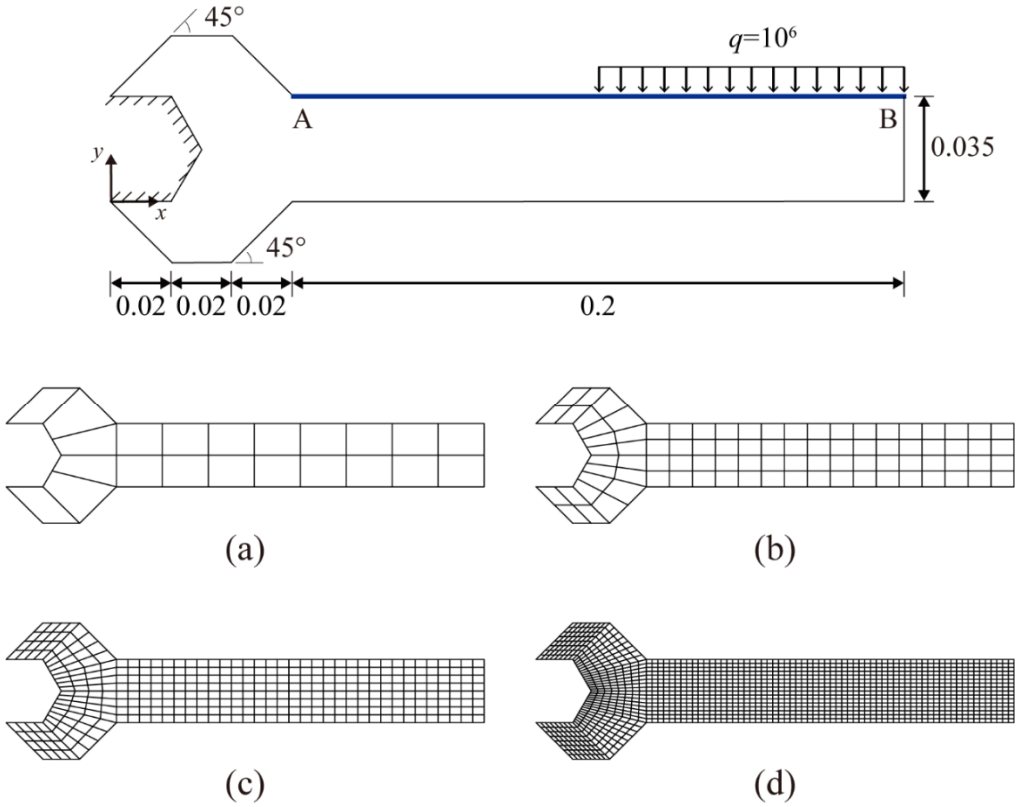

Figure 4.20. Wrench problem ($E = 2.0\times10^{11}$, $\nu = 0.3$, thickness $= 0.01$): (a) Coarse mesh ($N = 2$). (b) Medium mesh ($N = 4$). (c) Fine mesh ($N = 8$). (d) Mesh used for the reference solution.

Fig. 4.21 displays the convergence curves for the quadratic and linear elements. Fig. 4.22 shows errors in the vertical displacement ($E_d = \left|(v - v_{ref}) / v_{ref}\right|$) along the line AB shown in Fig. 4.21. The results show that the DL8 and DL4 elements perform very well.

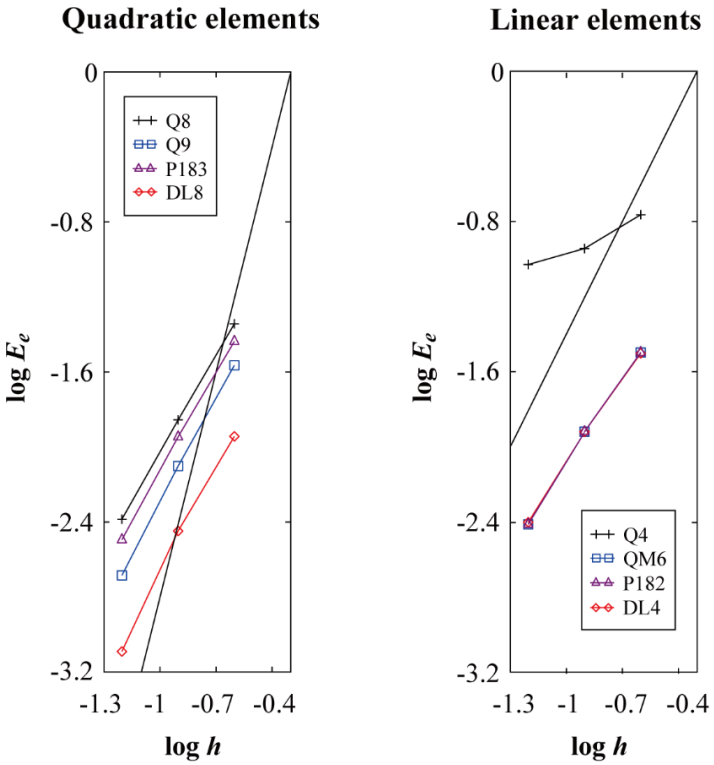

Figure 4.21. Convergence curves in the wrench problem: The bold lines represent the optimal convergence rates.

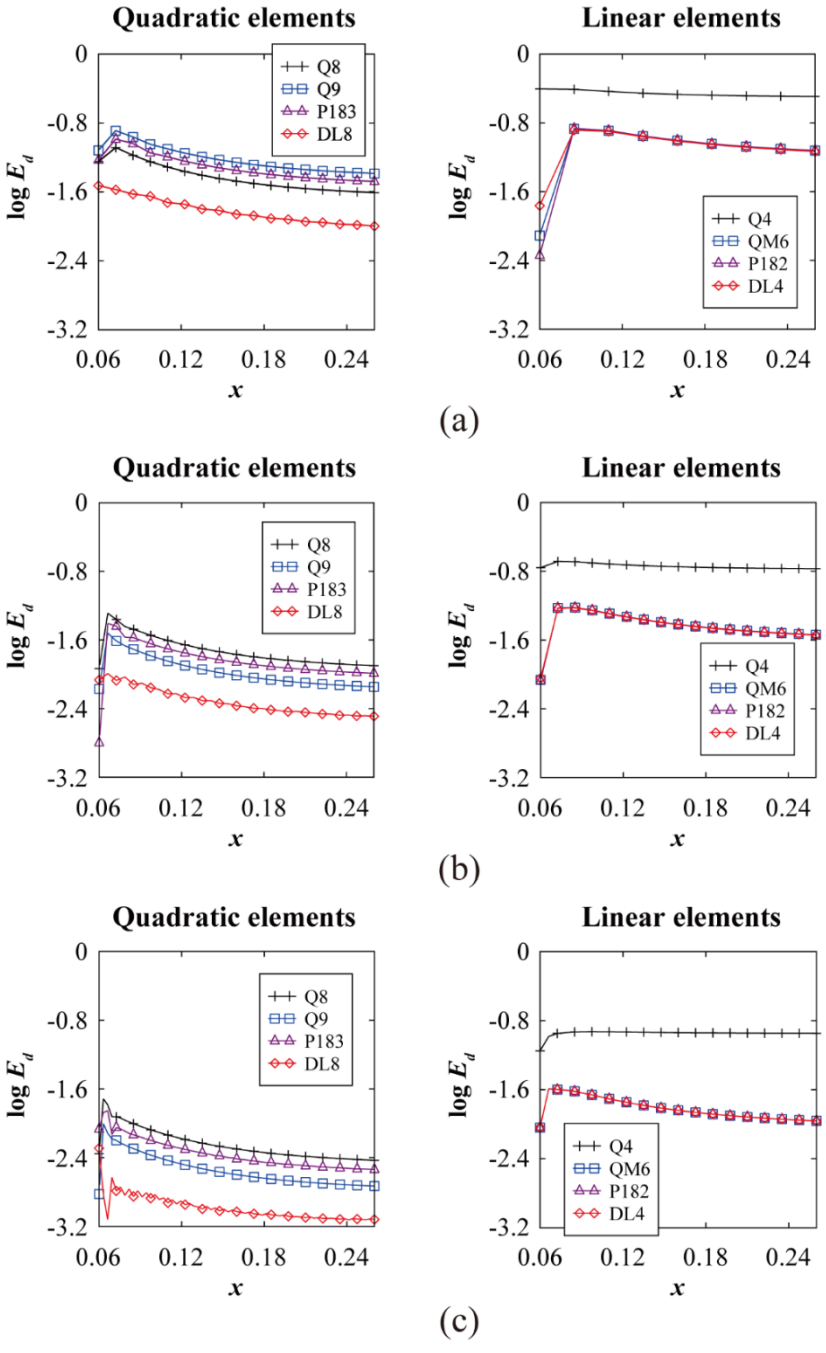

Figure 4.22. Errors in the vertical displacement along the line AB in the wrench problem: (a) Coarse mesh. (b) Medium mesh. (c) Fine mesh.

4.6 Computational efficiency

The computation cost of the proposed deep learned finite elements (DL8 and DL4) is assessed through the Cook's skew beam problem described in Chapter 4.5.1. The computation times taken from obtaining the stiffness matrices to solving the linear equations are measured. All calculations were performed using a quad-core workstation (Intel(R) Core (TM) i7-2600 CPU @ 3.40 GHz, 12 GB memory, Microsoft Windows 10 64bit) under Python environment. The linear equations were solved using a direct solver in NumPy library [47].

Fig. 4.23 displays the relations between the computation time versus solution accuracy relative errors in the energy norm in Eq. (4-25). Regular meshes with N = 16, 32, and 64 are used for the assessment. In Cook's skew beam problem, the DL8 element outperforms in the aspect of computational efficiency among the tested elements. At similar accuracy levels, the DL8 element gives less computation times compared with the other quadratic elements. That is, the DL8 element outperforms in the aspect of computational efficiency among the elements. However, the computational efficiency of the DL4 element is not as good as that of the QM6 element.

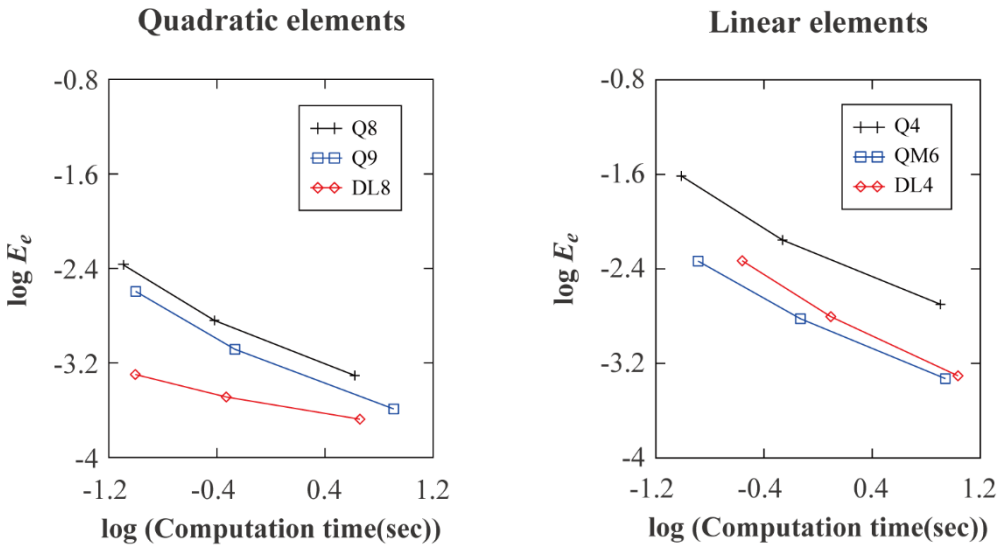

Figure 4.23. Computational efficiency curves in the Cook's skew beam problem. The computation times are measured in seconds.

4.7 Concluding remark

The deep learned 8- and 4-node quadrilateral elements were developed. Various new concepts and processes are presented: normalized element geometry, reference data model for the training data, pre-processing for the input, and post-processing for the output. We also proposed a way to make finite elements better represent rigid body

motions and constant strain fields. The performance of the DL8 and DL4 elements was evaluated through various numerical examples. In particular, the DL8 element showed promising ability in both accuracy and computational efficiency.

Chapter 5. Self-Updated Finite Elements

4-node quadrilateral finite element was not improved the performance of the QM6 even with the method of Chapter 4. In this chapter, a new 4-node element is presented based on a mode-based formulation using analytical strains and an update method using deep learning to improve the performance of the DL4. 4-node quadrilateral finite elements have been studied for a long time to reduce the error due to the deformation of the mesh while passing the patch test and zero energy mode test.

The thin beam problem proposed by MacNeal and Harder [56] is an example that check the error due to the distortion of the 4-node element. Wilson et al [76] proposed Q6 element which has two internal DOFs on the Q4, showing improved results in MacNeal's thin beam. However, Q6 didn't pass the patch test. Taylor et al [41] proposed modified Q6, named as QM6. QM6 passed the patch test and showed improved results compared to the Q4 in the MacNeal's thin beam. However, the results of QM6 were inaccurate in the case of trapezoidal mesh. To improve the shortcomings, PS, PEAS7, QACM4, SPS, SYHP, QE2, $\bar{B}$-QE4, and CQAC6 elements were developed, but the solution was still inaccurate when the mesh is distorted to a trapezoid shape [57, 59-61, 63-65]. After that, F-M QUAD4-R and US-ATFQ4 elements were developed, and the results were close to the exact solution in the benchmark problem even if the mesh was distorted [67, 74]. However, the formulations of F-M QUAD4-R are much closer to the mesh-free method, and US-ATFQ4 is an unsymmetric element, which make it difficult to handle in a general FEM solver. In addition, QAC4θ, QAC4θM, HSF-Q4θ-7β, and US-Q4θ elements with drilling DOF were proposed [68, 75, 77]. These elements showed improved results in several benchmark problems.

In this chapter, self-updated finite element is presented. Chapter 5.1 shows the mode based formulation of 2D solid finite elements. Chapter 5.2 presents the method for generating SUFE using deep learning, including data generation, network configuration and training, and the construction of the stiffness matrix applying iteration. Chapter 5.3 reports the basic test results of the deep learned finite elements, and Chapter 5.4 discuss the performance of the obtained finite elements through various numerical examples. Finally, the concluding remarks are presented in Chapter 5.5.

5.1 Mode based formulation of 2D solid finite elements

A 4-node plane element in 2D space has 2 nodal degrees of freedom (DOF) at each node and total 8 nodal degrees of freedom. The number of kinematic modes which the element can represent is also exactly the same to the number of its total degrees of freedom:

- 3 rigid body or zero strain modes (2 transitional and 1 rotational rigid body modes)
- 3 constant strain modes (1 shearing and 2 stretching modes)

- 2 linear strain modes (2 bending modes).

To order to decompose the kinematic modes of the plane element, it is necessary to find the 8 kinematic modes of an arbitrarily shaped element. Let us consider the i^{th} kinematic mode of the 4-node plane element in x-y plane,

$$\vec{\phi}_i = \{\phi_{1,i} \quad \phi_{2,i} \quad \phi_{3,i} \quad \phi_{4,i} \quad \phi_{5,i} \quad \phi_{6,i} \quad \phi_{7,i} \quad \phi_{8,i}\}^{\mathrm{T}}, \tag{5-1}$$

in which the mode vector has 8 components corresponding to the 8 nodal displacements, u_1, u_2, u_3, u_4, v_1, v_2, v_3 and v_4, sequentially. We here present how to obtain the 8 kinematic mode vectors as shown in Fig. 5.1:

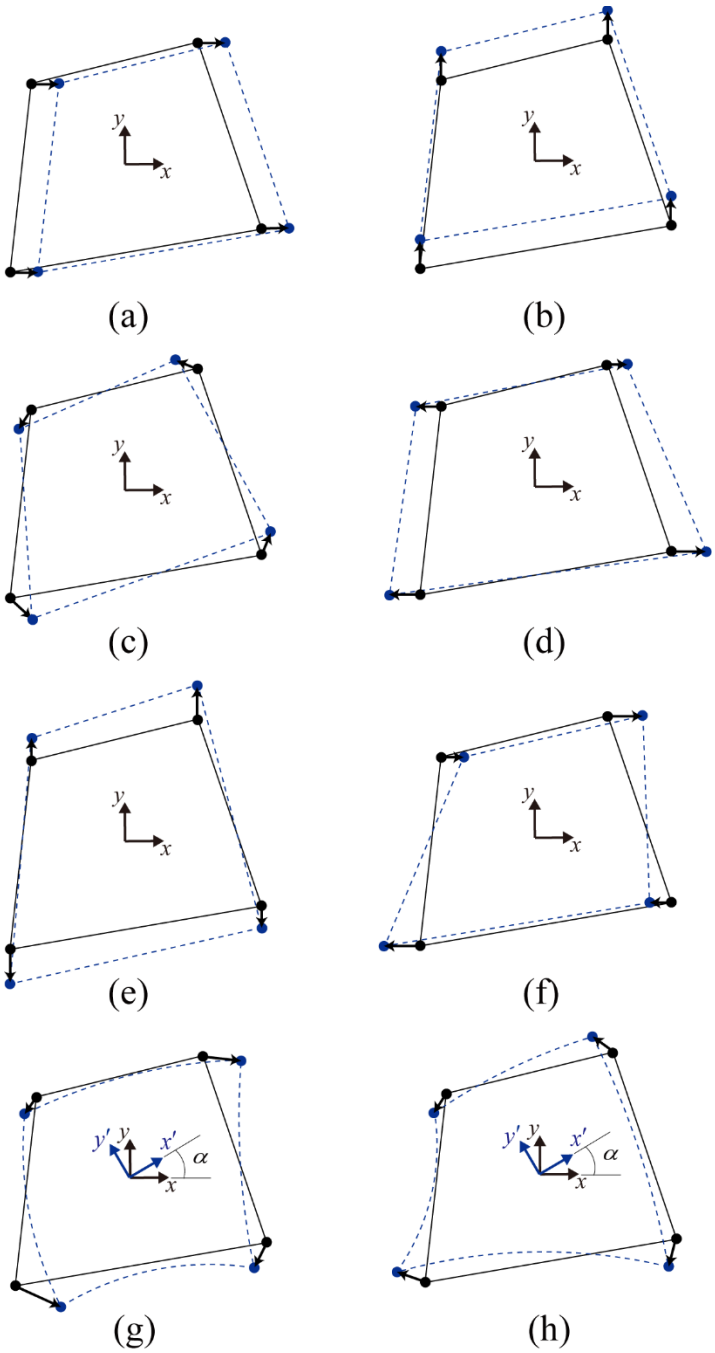

Figure 5.1. Kinematic modes of the 4-node plane element in x-y plane (a) Transitional rigid body mode corresponding to the x-direction (b) Transitional rigid body mode corresponding to the y-direction (c) Rotational rigid body mode (d) Stretching modes corresponding to the x-direction (e) Stretching modes corresponding to the y-direction (f) Shearing mode (g) Bending mode1 (h) Bending mode2.

- $\vec{\phi}_1$ (Transitional rigid body mode corresponding to the *x*-direction in the global Cartesian coordinate system). This transitional rigid body modes with a unit transitional movement are corresponding to the displacement field, $u = 1$ and $v = 0$, and

$$\vec{\phi}_1 = \{1 \quad 1 \quad 1 \quad 1 \quad 0 \quad 0 \quad 0 \quad 0\}^{\mathrm{T}}. \tag{5-2}$$

- $\vec{\phi}_2$ (Transitional rigid body mode corresponding to the *y*-direction). Similarly, this mode corresponds $\mathrm{u} = 0$ and $\mathrm{v} = 1$, and

$$\vec{\phi}_2 = \{0 \quad 0 \quad 0 \quad 0 \quad 1 \quad 1 \quad 1 \quad 1\}^{\mathrm{T}}. \tag{5-3}$$

- $\vec{\phi}_3$ (Rotational rigid body mode corresponding to the rotation about the *z*-axis). The displacement field in this rotation is $\mathrm{u} = -\mathrm{y}$ and $\mathrm{v} = \mathrm{x}$, and the mode vector is

$$\vec{\phi}_3 = \{-y_1 \quad -y_2 \quad -y_3 \quad -y_4 \quad x_1 \quad x_2 \quad x_3 \quad x_4\}^{\mathrm{T}}, \tag{5-4}$$

where x_i and y_i are the coordinates of the i^{th} nodal point.

- $\vec{\phi}_4$ (Constant stretching modes corresponding to the *x*-direction). The corresponding displacement is $\mathrm{u} = \mathrm{x}$ and $v = 0$, and

$$\vec{\phi}_4 = \{x_1 \quad x_2 \quad x_3 \quad x_4 \quad 0 \quad 0 \quad 0 \quad 0\}^{\mathrm{T}}, \tag{5-5}$$

where υ is Poisson's ratio.

- $\vec{\phi}_5$ (Constant stretching modes corresponding to the *y*-direction). The corresponding displacement is $u = 0$ and $\mathrm{v} = \mathrm{y}$, and

$$\vec{\phi}_5 = \{0 \quad 0 \quad 0 \quad 0 \quad y_1 \quad y_2 \quad y_3 \quad y_4\}^{\mathrm{T}}. \tag{5-6}$$

- $\vec{\phi}_6$ (Shearing mode in the *x*-*y* plane). The displacement field is $\mathrm{u} = \mathrm{y}$ and $v = 0$, and the mode vector is

$$\vec{\phi}_6 = \{y_1 \quad y_2 \quad y_3 \quad y_4 \quad 0 \quad 0 \quad 0 \quad 0\}^{\mathrm{T}}. \tag{5-7}$$

- $\vec{\phi}_7$ (Bending mode1 in the *x*-*y* plane). The displacement field is $\mathrm{u}' = \mathrm{x}'\mathrm{y}'$ and $v' = -\frac{1}{2}\left(x'^2 + \nu y'^2\right)$, and the

mode vector is

$$\vec{\phi}_7 = \{ x_1' y_1' \quad ... \quad x_4' y_4' \quad -\frac{1}{2}\left(x_1'^2 + \nu y_1'^2\right) \quad ... \quad -\frac{1}{2}\left(x_4'^2 + \nu y_4'^2\right) \}^{\mathrm{T}}, \tag{5-8}$$

where x' and y' are new local Cartesian coordinate for bending modes, and u' and v' are the displacements in the new local Cartesian coordinate system.

- $\vec{\phi}_8$ (Bending mode2 in the *x*-*y* plane). The displacement field is $u' = -\frac{1}{2}\left(\nu x'^2 + y'^2\right)$ and $v' = x'y'$, and the mode vector is

$$\vec{\phi}_7 = \{ -\frac{1}{2}\left(\nu x_1'^2 + y_1'^2\right) \quad ... \quad -\frac{1}{2}\left(\nu x_4'^2 + y_4'^2\right) \quad x_1' y_1' \quad ... \quad x_4' y_4' \}^{\mathrm{T}}. \tag{5-9}$$

The two bending modes can be differently chosen depending on the new local Cartesian coordinates x' and y'.

The nodal displacement of the 4-node plane element can be expressed by the 8 kinematic modes,

$$\begin{Bmatrix} u_1 \\ u_2 \\ \vdots \\ v_4 \end{Bmatrix} = \begin{Bmatrix} \phi_{1,1} \\ \phi_{2,1} \\ \vdots \\ \phi_{8,1} \end{Bmatrix} q_1 + \begin{Bmatrix} \phi_{1,2} \\ \phi_{2,2} \\ \vdots \\ \phi_{8,2} \end{Bmatrix} q_2 + \cdots\cdots + \begin{Bmatrix} \phi_{1,8} \\ \phi_{2,8} \\ \vdots \\ \phi_{8,8} \end{Bmatrix} q_8, \tag{3-10}$$

where q_i are variables to express how much the mode *i* is contained in the nodal displacement, and, in matrix form,

$$\begin{Bmatrix} u_1 \\ u_2 \\ \vdots \\ v_4 \end{Bmatrix} = \begin{bmatrix} \phi_{1,1} & \phi_{1,2} & \cdots\cdots & \phi_{1,8} \\ \phi_{2,1} & \phi_{2,2} & \cdots\cdots & \phi_{2,8} \\ \vdots & \vdots & \cdots\cdots & \vdots \\ \phi_{8,1} & \phi_{8,2} & \cdots\cdots & \phi_{8,8} \end{bmatrix} \begin{Bmatrix} q_1 \\ q_2 \\ \vdots \\ q_8 \end{Bmatrix} \quad \text{or} \quad \mathbf{U} = \boldsymbol{\psi}\,\mathbf{Q}, \tag{5-11}$$

where $\mathbf{U}$ is the nodal displacement vector, $\boldsymbol{\psi}$ is the kinematic modal matrix, and $\mathbf{Q}$ is the vector of q_i.

Using the basis transformation in Eq. (5-11), the local strain can be expressed in terms of $\mathbf{Q}$,

$$\boldsymbol{\varepsilon} = \mathbf{B}\mathbf{U} = \mathbf{B}\boldsymbol{\psi}\mathbf{Q} = \hat{\mathbf{B}}\mathbf{Q}, \tag{5-12}$$

where $\boldsymbol{\varepsilon}$ is the local strain vector, $\mathbf{B}$ is the strain displacement matrix, $\hat{\mathbf{B}}$ is the local strain matrix by kinematic modes.

The assumed strain method for each kinematic mode can be applied. $\hat{\mathbf{B}}$ in Eq. (5-12) is assumed as

$$\hat{\mathbf{B}} = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 & e_1^7 & e_1^8 \\ 0 & 0 & 0 & 0 & 1 & 0 & e_2^7 & e_2^8 \\ 0 & 0 & 0 & 0 & 0 & 1 & e_3^7 & e_3^8 \end{bmatrix} \text{ where } \mathbf{e}^7 = \mathbf{T}\begin{bmatrix} y' \\ -\nu y' \\ 0 \end{bmatrix}, \ \mathbf{e}^8 = \mathbf{T}\begin{bmatrix} -\nu x' \\ x' \\ 0 \end{bmatrix}, \tag{5-13}$$

with

$$\mathbf{T}=\begin{bmatrix} \cos^2\alpha & \sin^2\alpha & -\sin\alpha\cos\alpha \\ \sin^2\alpha & \cos^2\alpha & \sin\alpha\cos\alpha \\ 2\sin\alpha\cos\alpha & -2\sin\alpha\cos\alpha & \cos^2\alpha-\sin^2\alpha \end{bmatrix}$$

in which $\mathbf{e}^i$ is the local strain when the kinematic mode i is activated, $\mathbf{T}$ is the strain transformation matrix [37], and α is the angle between the x-axis and the x'-axis as shown in Fig. 5.1(g). The assumed strains of $\hat{\mathbf{B}}$ are based on the analytical solutions.

Finally, we have the local strain vector and, using Eq. (5-12), the local strain is directly represented in terms of the nodal displacement vector as

$$\boldsymbol{\varepsilon}=\hat{\mathbf{B}}\,\mathbf{A}=\hat{\mathbf{B}}\,\boldsymbol{\psi}^{-1}\,\mathbf{U}=\mathbf{B}_m\mathbf{U}\,, \tag{5-14}$$

where $\mathbf{B}_m$ is the modified strain displacement matrix from assumed strains.

Depending on the choice of the local coordinates x' and y' for bending modes, many different elements can be constructed.

5.2 Procedures to generate self-updated finite element

Conventional finite elements use a prescribed formulation regardless of displacements. These elements generate accurate solution for certain shapes and deformations, but show deteriorated solution for distorted shapes or the other deformations [59, 64-68, 70]. The accuracy of the element formulated as Chapter 5.1 is also degraded when its displacements are not suitable for the bending modes obtained from the selected local coordinates.

In this work, we propose an element that improves accuracy by updating the local coordinates according to the elemental displacements. When displacements are given to the finite element, the strain energy varies according to the local coordinates. It is physically ideal that strain energy is minimized for given displacements. Therefore, the problem of determining the local coordinates can be approached as a strain energy optimization problem using the shape of finite elements and displacements. However, it is time consuming to solve this optimization problem for determining the local coordinates at each element. The result of the optimization problem can be approximated through deep learning and its computing time is much less than solving the optimization problem [55]. Therefore, we use deep learning to determine the local coordinates of the SUFE.

In the sub-chapter, we introduce the concept of SUFE and the procedure to generate 4-node quadrilateral SUFE using deep learning. The methodology for constructing the neural network model is presented in detail, including data generation, network configuration and training. Finally, we present how to obtain the stiffness matrix of the

SUFE iteratively.

5.2.1 Concept

SUFE is an element that improves the accuracy of the solution by determining the local coordinates according to the given element shape and displacements based on the formulation in Chapter 5.1. Deep learning is used to determine the local coordinates. The procedure to obtain the stiffness matrix of SUFE is as follows. First, the analysis domain is solved using preceding elements, and the geometry and displacements of each element are input into the trained network to approximate the angle of the local coordinates. Updated strain displacement matrix ($\mathbf{B}_m$) is generated using the angle and Eq. (5-13) and (5-14). The stiffness matrix of SUFE is updated using $\mathbf{B}_m$. The analysis domain is solved using the updated stiffness matrix, and this procedure is repeated until the strain energy of the domain decreases.

5.2.2 Data generation

To construct a neural network that inference the local coordinates of an arbitrary 4-noded finite element, a large amount of training data corresponding to random geometries, displacements, and material properties are required. As processing all space of these data is extremely difficult, we aim to reduce the number of data for achieving efficient network training using the method used in Chapter 4.

We use normalized geometry as a representative of all the similar shapes. Here, the normalized geometry refers to a quadrilateral where the two nodes ($\mathbf{x}_1^{(n)}$ and $\mathbf{x}_2^{(n)}$) at either end of the maximum length side are located at (0, 0) and (1, 0) in 2D Cartesian coordinates. Note that superscript (n) means n^{th} data. n^{th} geometries are generated by placing the other two nodes ($\mathbf{x}_3^{(n)}$ and $\mathbf{x}_4^{(n)}$) randomly as shown in Fig. 5.2, and severely distorted geometries such as quadrilaterals with an interior angle of less than 1° or greater than 179° and ratios between the maximum and minimum side lengths of greater than 100 are excluded.

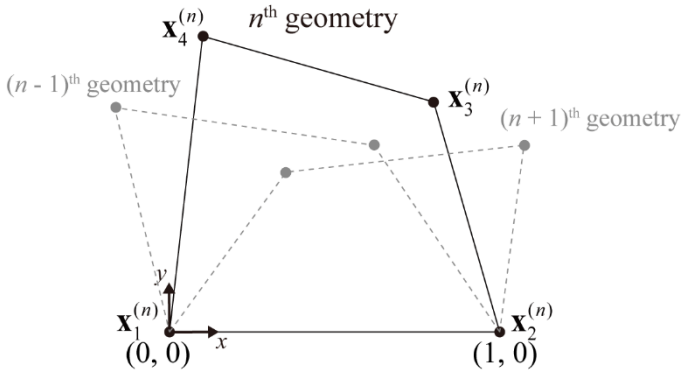

Figure 5.2. Random generation of the n^{th} normalized element geometry.

We also use the normalized displacements ($\overline{u}_i^{(n)}$ and $\overline{v}_i^{(n)}$). n^{th} nodal displacements are generated randomly with a uniform distribution in the range of -0.25 to 0.25. Using rigid body modes, the DOFs of displacements are reduced from 8 to 5 and the displacements are normalized by the maximum value as,

$$ {}^0\overline{u}_i^{(n)} = u_i^{(n)} - u_1^{(n)}, \tag{5-15} $$

$$ {}^0\overline{v}_i^{(n)} = v_i^{(n)} - v_1^{(n)}, \tag{5-16} $$

$$ {}^1\overline{u}_i^{(n)} = {}^0\overline{u}_i^{(n)} - y_i^{(n)}\,{}^0\overline{v}_2^{(n)}, \tag{5-17} $$

$$ {}^1\overline{v}_i^{(n)} = {}^0\overline{v}_i^{(n)} + x_i^{(n)}\,{}^0\overline{v}_2^{(n)}, \tag{5-18} $$

$$ \overline{u}_i^{(n)} = \frac{{}^1\overline{u}_i^{(n)}}{d_{\max}}, \tag{5-19} $$

$$ \overline{v}_i^{(n)} = \frac{{}^1\overline{v}_i^{(n)}}{d_{\max}} \quad \text{for} \quad i = 1,2,3,4 \tag{5-20} $$

with $d_{\max} = \max(\left|{}^1\overline{u}_1^{(n)}\right|, \left|{}^1\overline{u}_2^{(n)}\right|, \left|{}^1\overline{u}_3^{(n)}\right|, \left|{}^1\overline{u}_4^{(n)}\right|, \left|{}^1\overline{v}_1^{(n)}\right|, \left|{}^1\overline{v}_2^{(n)}\right|, \left|{}^1\overline{v}_3^{(n)}\right|, \left|{}^1\overline{v}_4^{(n)}\right|)$. $\overline{u}_1^{(n)}$, $\overline{v}_1^{(n)}$ and $\overline{v}_2^{(n)}$ become 0 through Eq. (5-15~3-20) and are excluded from the data. Note that rigid body modes and the size of $\mathbf{U}$ do not affect the determination of the local coordinates.

Young's modulus (E) was adopted 2.0×10^{11}, and Poisson's ratio ($\nu^{(n)}$) was randomly applied with a uniform distribution in the range of 0–0.499999999. Note that Young's modulus does not affect the internal strains (see Appendix A). It means that Young's modulus has no effect on the determination of the local coordinates.

$\alpha^{(n)}$ is required as label data for the generated n^{th} geometry, displacements, and material properties. Determining $\alpha^{(n)}$ is a strain energy optimization problem as described in Chapter 5.2. $\alpha^{(n)}$ is obtained through following conditions:

$$ \alpha^{(n)} = \underset{\alpha}{\text{minimize}}\, E^{(n)} \quad \text{where} \quad 0° \le \alpha^{(n)} \le 90°, \tag{5-21} $$

where $E^{(n)}$ is the strain energy stored in the n^{th} element. Nelder-Mead method [73] is used to find an optimum. Thirty initial seeds are generated evenly distributed in the range of 0° to 90°.

Poisson's ratio, the nodal coordinates of the normalized geometry, the normalized nodal displacements, and the angle of the local coordinates are made into one training data. The n^{th} training data ($\mathbf{D}^{(n)}$) is configured as

$$ \mathbf{D}^{(n)} = \begin{bmatrix} \nu^{(n)} & \mathbf{D}_x^{(n)} & \mathbf{D}_u^{(n)} & \alpha^{(n)} \end{bmatrix} \tag{5-22} $$

with

$$ \mathbf{D}_x^{(n)} = \begin{bmatrix} \mathbf{x}_3^{(n)\mathrm{T}} & \mathbf{x}_4^{(n)\mathrm{T}} \end{bmatrix} \text{ and } \mathbf{D}_u^{(n)} = \begin{bmatrix} \overline{u}_2^{(n)} & \overline{u}_3^{(n)} & \overline{u}_4^{(n)} & \overline{v}_3^{(n)} & \overline{v}_4^{(n)} \end{bmatrix}, $$

where $\nu^{(n)}$, $\mathbf{D}_x^{(n)}$, $\mathbf{D}_u^{(n)}$, and $\alpha^{(n)}$ denote Poisson's ratio (1 value), nodal coordinates (2×2 values), nodal displacements (5 values), and angle (1 values), respectively. In total, the n^{th} training data contain 11 values.

5.2.3 Network configuration and training

The output of the network is $\alpha_o^{(n)}(\boldsymbol{\theta})$. Poisson's ratio ($\nu^{(n)}$), the nodal coordinates ($\mathbf{D}_x^{(n)}$), and the nodal displacements ($\mathbf{D}_u^{(n)}$) of the training data are the inputs of the neural network, while angle ($\alpha^{(n)}$) of the training data are used for the following cost function $C(\boldsymbol{\theta})$:

$$C(\boldsymbol{\theta}) = \frac{1}{M}\sum_{n=1}^{M}\left|\alpha_o^{(n)}(\boldsymbol{\theta}) - \alpha^{(n)}\right|^{\frac{1}{2}}, \tag{5-23}$$

where $\boldsymbol{\theta}$ denote the network weights, M is the number of training data.

A network was constructed as shown in Fig. 5.3. A fully connected neural network of 10 layers was employed, and batch normalization was applied to each layer. An exponential linear unit was used as an activation function at each layer before the output. The network width was 320. After the training of the network, the approximated $\alpha_o^{(n)}$ is generated.

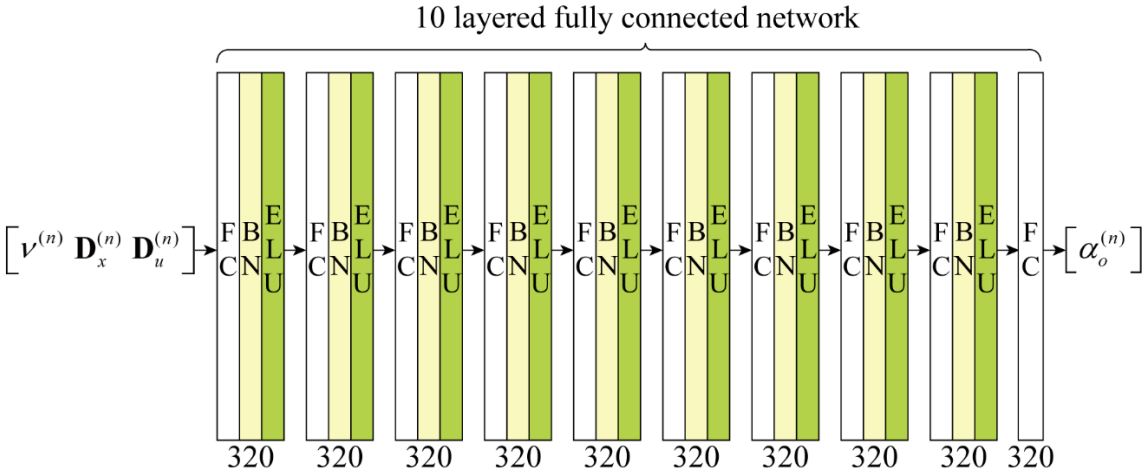

Figure 5.3. Network configuration for deep learned finite elements. (FC: fully connected layer, BN: batch normalization, ELU: exponential linear unit).

The network was trained with a total of 3,000,000 data (M = 3,000,000). To test the network, 50,000 data were generated additionally.

The network was implemented using TensorFlow [34], Adam optimization [35] was adopted as the optimizer, and Xavier initializer [36] was applied to initialize the weights of the network. We performed training for 30,000 epochs, and a batch size of 50,000 was used. The learning rate converged linearly from 0.01 to 0 as the epoch progressed to approach better results. As a result of training, the cost function value of the training data was 0.40 and the cost function value of the test data was 0.45.

5.2.4 Construction of the stiffness matrix

Normalized geometries and displacements were employed for the efficient training of the network. In order to apply the trained network to elements with arbitrary geometries and displacements, pre-processing for the input of the trained network is necessary. In addition, it is necessary to adjust the angle of the network output to generate the stiffness matrix, and iterative analysis is required with the generated stiffness matrix to elaborate the solution.

5.2.4.1 Pre-processing of the network input

Geometry normalization is performed for an element with an arbitrary geometry. The element connectivity is assigned so that the side length between node 1 and node 2 is the longest. Then, as shown in Fig. 5.4, the nodal coordinates of the element ($\mathbf{x}_i$) are translated, rotated, and resized to obtain the input normalized nodal coordinates (${}_{\text{input}}\mathbf{x}_i$) where node 1 and node 2 are positioned at (0, 0) and (1, 0), respectively.

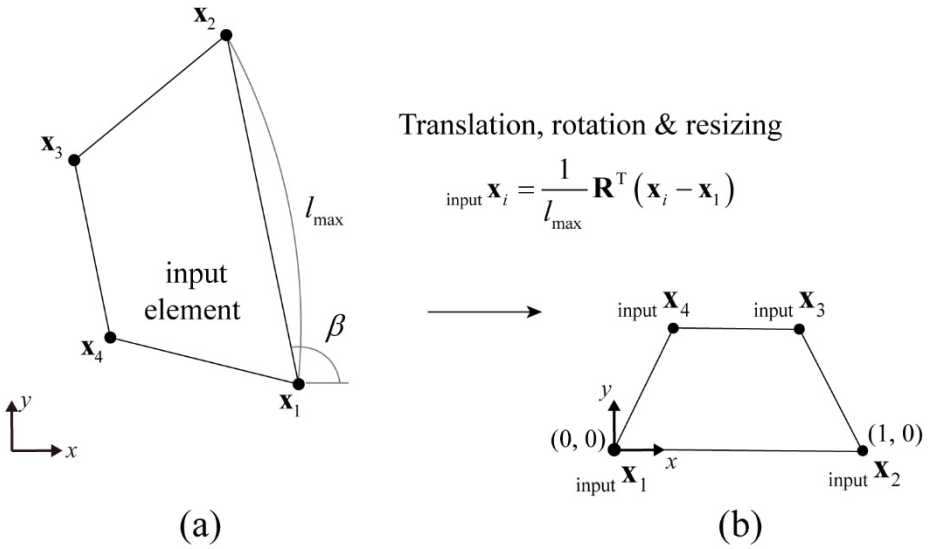

Figure 5.4. Pre-processing procedure to obtain the network input. (a) Original element geometry. (b) Normalized element geometry.

The normalized nodal coordinates are obtained by

$$ {}_{\text{input}}\mathbf{x}_i = \frac{1}{l_{\max}}\mathbf{R}^{\mathrm{T}}\left(\mathbf{x}_i - \mathbf{x}_1\right) \quad \text{for} \quad i = 1,\ 2,\ 3,\ 4 \tag{5-24} $$

with $\mathbf{R} = \begin{bmatrix} \cos\beta & -\sin\beta \\ \sin\beta & \cos\beta \end{bmatrix}$,

in which β is the angle between the longest side and the x-axis, and $l_{\max}$ is the longest side length as shown in Fig. 5.4. The coordinates of ${}_{\text{input}}\mathbf{x}_3$ and ${}_{\text{input}}\mathbf{x}_4$ obtained from this process. and Poisson's ratio υ are used as input data for the trained network.

Displacements are required as a network input. The displacement is obtained as in Chapter 5.2.4.3, and normalization of displacements is performed through Eq. (5-15~3-20).

Poisson's ratio (ν), the normalized nodal coordinates, and the normalized displacements are made into one data for network input as,

$$_{\text{input}}\mathbf{D} = \begin{bmatrix} \nu & \mathbf{D}_x & \mathbf{D}_u \end{bmatrix}, \tag{5-25}$$

with

$$\mathbf{D}_x = \begin{bmatrix} _{\text{input}}\mathbf{x}_3^{\mathrm{T}} & _{\text{input}}\mathbf{x}_4^{\mathrm{T}} \end{bmatrix} \text{ and } \mathbf{D}_u = \begin{bmatrix} \bar{u}_2 & \bar{u}_3 & \bar{u}_4 & \bar{v}_3 & \bar{v}_4 \end{bmatrix},$$

where $\bar{u}_i$ and $\bar{v}_i$ are the i^{th} normalized displacements.

5.2.4.2 Post-processing of the network output

The trained network outputs α_o. α_o is the angle of the local coordinates based on the normalized geometry. The local coordinate of the original element is obtained from the angle as,

$$\alpha = \alpha_o + \beta . \tag{5-26}$$

$\mathbf{B}_m$ is generated using Eq. (5-13, 5-14) and the angle from Eq. (5-26). The stiffness matrix of SUFE is calculated by

$$\mathbf{K}_m = \int_V \mathbf{B}_m^{\mathrm{T}} \mathbf{C} \mathbf{B}_m dV , \tag{5-27}$$

where V is the element volume, and $\mathbf{C}$ is the material law matrix.

5.2.4.3 Iteration procedure for updating the stiffness matrix

The initial displacements are obtained by solving the analysis domain using the elements whose strain displacement matrix ($\bar{\mathbf{B}}$) is generated using B-bar method [38] and $\mathbf{B}_m$ with a fixed angle of 0.001° as follows

$$\bar{\mathbf{B}} = \mathbf{B}_m + \mathbf{B}' \quad \text{with} \quad \mathbf{B}' = -\frac{1}{V}\int_V (\mathbf{B}_m - \mathbf{B}_{\mathrm{Q4}}) dV , \tag{5-28}$$

where $\mathbf{B}_{\mathrm{Q4}}$ is the strain-displacement matrix of the standard 4-node quadrilateral element [31].

First, strain energies are calculated by the stiffness matrices with each of $\bar{\mathbf{B}}$ and $\mathbf{B}_m$ ($\bar{\mathbf{K}}$ is stiffness matrix generated from $\bar{\mathbf{B}}$, and $\mathbf{K}_m$ is generated from $\mathbf{B}_m$) using initial displacements. Next, the results are compared. If the strain energies ($\bar{E}$ is calculated by $\bar{\mathbf{K}}$, and E is calculated by $\mathbf{K}_m$) are equal, it means that the displacements are determined only by $\vec{\phi}_1$ - $\vec{\phi}_6$, so no update is performed. If the results are different, the displacements are obtained by solving the analysis domain with $\mathbf{K}_m$, and the local coordinate is updated through

the trained network using the obtained displacements as shown in Fig. 5.5.

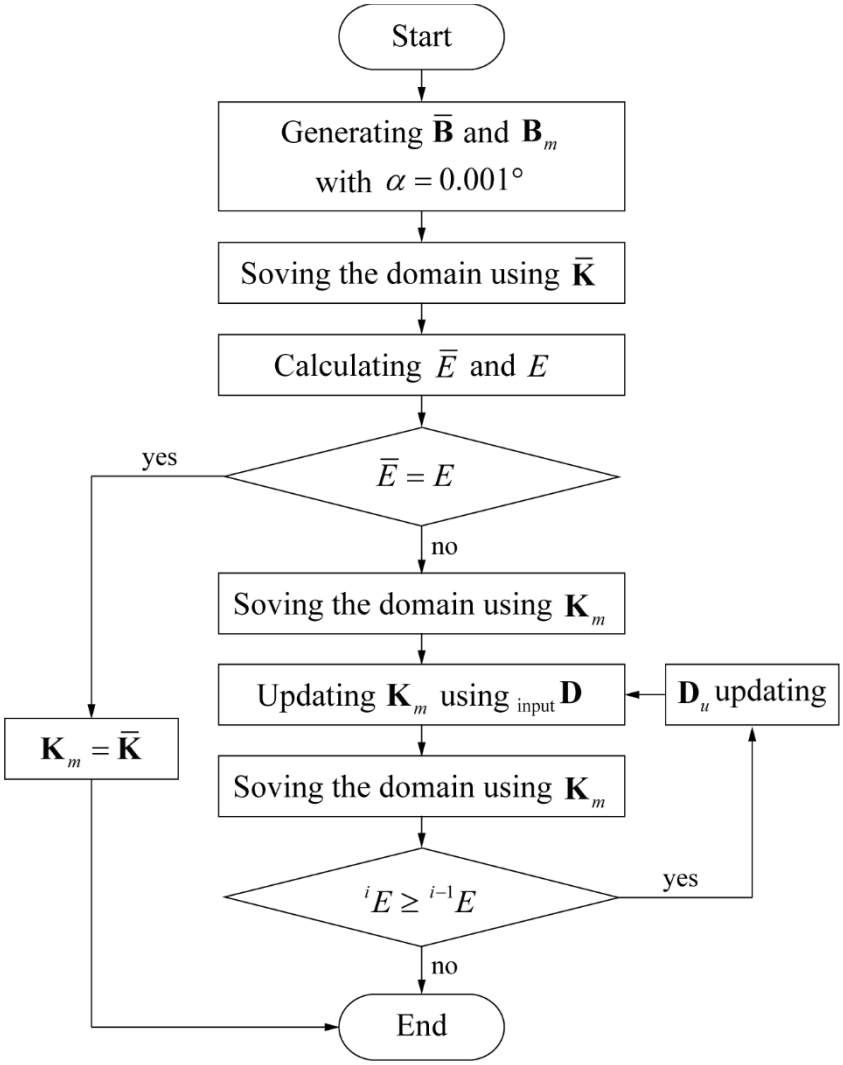

Figure 5.5. SUFE Stiffness matrix iteration procedures.

The analysis domain is solved with $\mathbf{K}_m$ from Eq. (5-27), and the strain energy stored in the entire structure is obtained. Suppose the obtained strain energy is the i^{th} strain energy (^{i}E). The strain energy obtained in the preceding analysis (^{i-1}E) is compared to ^{i}E. If the strain energy increases, $\mathbf{D}_u$ is updated using the i^{th} displacements, and the angle and the stiffness matrix are recalculated using updated $_{\text{input}}\mathbf{D}$. This procedure is repeated until the strain energy decreases as shown in Fig. 5.5.

5.3 Basic numerical tests

In this chapter, zero energy mode and patch tests are performed for SUFE. The computer program for SUFE was Python and executed under Microsoft Windows 10 64bit OS and Anaconda platform. The linear equations were

solved using a direct solver in NumPy library [47].

5.3.1 Zero energy mode tests

In the zero energy mode test, the zero eigenvalues of the stiffness matrix of a single SUFE are counted. Undistorted (in Fig. 5.6(a)) and distorted (in Fig. 5.6(b), (c) and (d)) element geometries are considered with unit thickness. Young's modulus $E = 1.5 \times 10^3$ and Poisson's ratio $\nu = 0.3$ are given.

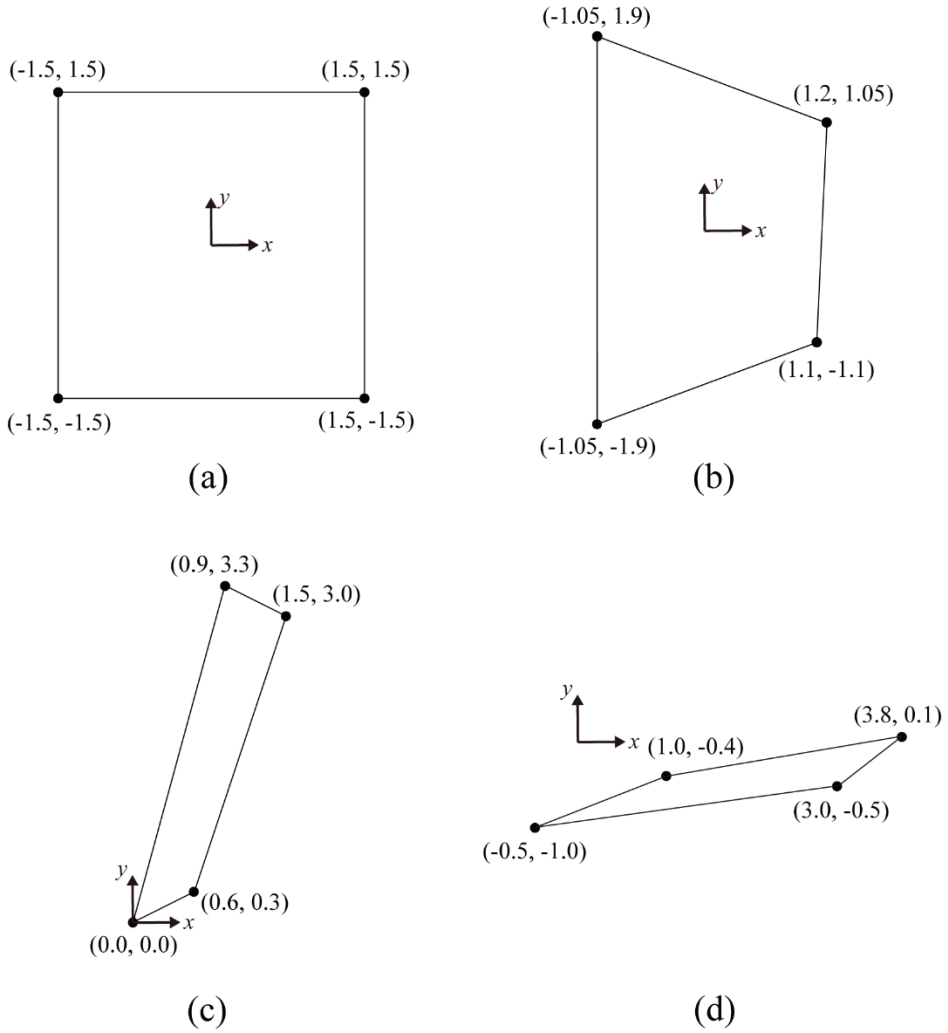

Figure 5.6. Element geometries used for the zero energy mode test: (a) Geometry 1, (b) Geometry 2, (c) Geometry 3, and (d) Geometry 4.

Table 5.1 presents the eigenvalues calculated up to the sixth strain energy modes. The first three eigenvalues correspond to the three rigid body modes (two translations and one rotation modes) for all geometry cases. The eigenvalue of mode 1-3 shows sufficiently smaller than those of the deformation modes (mode 4-6). Therefore, SUFE passes the zero energy mode test.

Table 5.1. Eigenvalues corresponding to the 1st~6th modes for the various geometries (in Fig. 5.5) of SUFE. (The 1st, 2nd and 3rd modes correspond to rigid body motions.)

Mode	Geometry 1	Geometry 2	Geometry 3	Geometry 4
1	-2.76E-14	5.58E-14	2.37E-14	1.06E-12
2	-6.12E-14	1.48E-13	-4.25E-14	-3.66E-12
3	-8.60E-14	-1.00E-13	7.37E-13	9.96E-12
4	5.00E+02	4.04E+02	2.61E+02	1.94E+01
5	5.00E+02	8.43E+02	3.18E+02	1.66E+02
6	1.15E+03	1.06E+03	2.62E+03	5.06E+03

5.3.2 Patch tests

Three patch tests are performed with the mesh geometry in Fig. 4.9(a) for *x*- and *y*-directional stretching and shearing [1]. The loading and displacement boundary conditions are shown in Fig. 4.9(b)–(d). If the constant stress fields are calculated, the patch tests are passed [39, 40]. SUFE pass the patch tests. In other words, SUFE can represent constant strain fields.

5.4 Numerical examples

In this chapter, the performance of the proposed elements is investigated through various numerical problems: MacNeal's thin cantilever beam, cantilever beam divided by two elements for mesh distortion test, cantilever beam divided by five elements, cantilever beam divided by four elements, thick curving beam, thin curving beam, cantilever beam for rotation dependency test, and Cook's skew beam problem.

The obtained results are compared to the results of the other elements in Table 5.2. All the elements to be compared are passed the patch test.

Table 5.2. List of elements for comparison.

Symbol	Description	Ref.
Q4	Standard 4-node quadrilateral element	[31]
QM6	4-node quadrilateral element with incompatible modes	[41]
P-S	4-node hybrid stress element	[57]
SPS	4-node hybrid stress elements with adjustable parameters	[61]
SYHP	4-node hybrid stress elements with adjustable parameters	[61]
CPS4I	4-node incompatible element with assumed strains	[62]
QE2	4-node assumed strain element	[64]

$\bar{\mathbf{B}}$-QE4	4-node assumed strain element (B-bar)	[63]
QACM4	4-node incompatible element using QACM-I	[60]
QAC4θ	4-node element with drilling DOFs	[75]
QAC4θM	4-node element with drilling DOFs	[75]
CQAC6	4-node incompatible element using QACM-I and III	[65]
F-M QUAD4-P	'FE-meshfree' 4-node element with polynomial basis functions	[66]
F-M QUAD4-R	'FE-meshfree' 4-node element with radial basis functions	[67]
HSF-Q4θ-7β	4-node hybrid stress-function element with drilling DOFs	[68]
NQ6	4-node incompatible hybrid stress element	[69]
NQ10	4-node incompatible hybrid stress element	[70]
HH4-3β	4-node hybrid stress element based on Hamilton principle	[58]
GC-Q6	4-node generalized conforming element	[71]
QC6	4-node quasi-conforming element	[72]
PEAS7	4-node assumed strain element	[59]
US-ATFQ4	4-node unsymmetric element	[74]
US-Q4θ	4-node unsymmetric element with drilling DOFs	[77]

5.4.1 MacNeal's thin cantilever beam

This example, proposed by MacNeal [56], is a benchmark for testing the sensitivity to mesh distortion of quadrilateral elements. The beam is clamped at the left end and two loading cases are considered: (1) pure bending M and (2) shear loading P at the free tip as shown in Fig. 5.7. Plane stress conditions is considered with Young's modulus E=1.0×10^7 and Poisson's ratio $\nu = 0.3$. Three different mesh patterns of meshes shown in Fig. 5.7(a)–(c) are adopted.

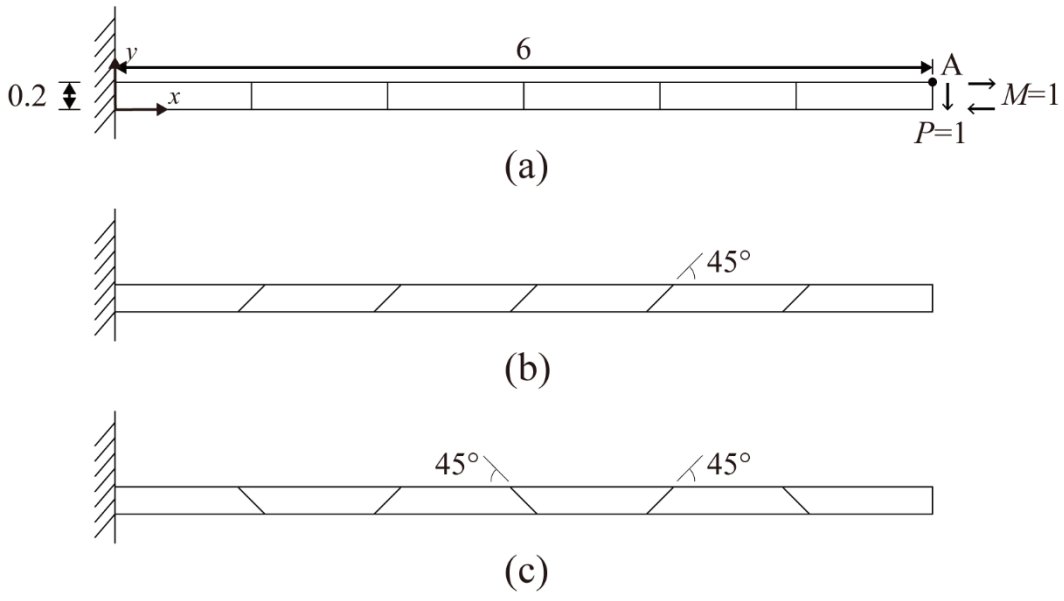

Figure 5.7. MacNeal's thin cantilever beam description ($E = 1.0 \times 10^7$, $\nu = 0.3$, thickness $= 0.1$). (a) Regular mesh. (b) Parallelogram mesh. (c) Trapezoidal mesh.

Table 5.3 shows the vertical displacements at point A normalized by the reference solution. The SU4 element is insensitive to mesh distortion, although it is a symmetric element, and it needs only one iteration in this problem, regardless of the mesh type.

Table 5.3. Normalized vertical displacements at point A in the MacNeal's thin cantilever beam using different meshes (Fig. 5.7), data in bold are the results obtained by the elements proposed in this paper, and the number in bracket is the number of iterations.

Elements	Load *P*			Load *M*		
	Mesh (a)	Mesh (b)	Mesh (c)	Mesh (a)	Mesh (b)	Mesh (c)
Q4	0.093	0.034	0.027	0.093	0.031	0.022
QM6 [41]	0.993	0.623	0.044	1.000	0.722	0.037
CPS4I [62]	0.993	0.632	0.050	1.000	0.725	0.047
P-S [57]	0.993	0.798	0.221	1.000	0.852	0.167
PEAS7 [59]	0.982	0.795	0.217	-	-	-
QACM4 [60]	0.995	0.635	0.052	1.000	0.722	0.046
F-M QUAD4-P [66]	0.984	0.963	0.932	1.000	1.000	1.000
HSF-Q4θ-7β [68]	0.993	0.988	0.991	1.000	1.000	1.000
QAC4θ [75]	0.904	0.867	0.906	0.910	0.880	0.930
QAC4θM [75]	0.993	0.984	0.988	1.000	0.992	0.998
US-ATFQ4 [74]	0.993	0.992	0.992	1.000	1.000	1.000
US-Q4θ [77]	0.993	0.993	0.989	1.000	1.000	1.000
SU4	**0.993(1)**	**0.994(1)**	**0.994(1)**	**1.000(1)**	**1.000(1)**	**1.000(1)**
Reference [56]	1.000 (the value: -0.1081)			1.000 (the value: -0.0054)		

5.4.2 Cantilever beam divided by two elements for mesh distortion test

The cantilever beam is considered for mesh distortion test as shown in Fig. 5.8. The shape of the two elements varies with a distortion parameter *e*. When *e* = 0, both elements are rectangular. With the increase of *e* value, the mesh is distorted more and more severely. The beam is subjected to a pure bending moment at the right end, and the left end is fixed as shown in Fig. 5.8. Plane stress condition is employed with Young's modulus *E*=1500 and Poisson's ratio $\nu = 0.25$.

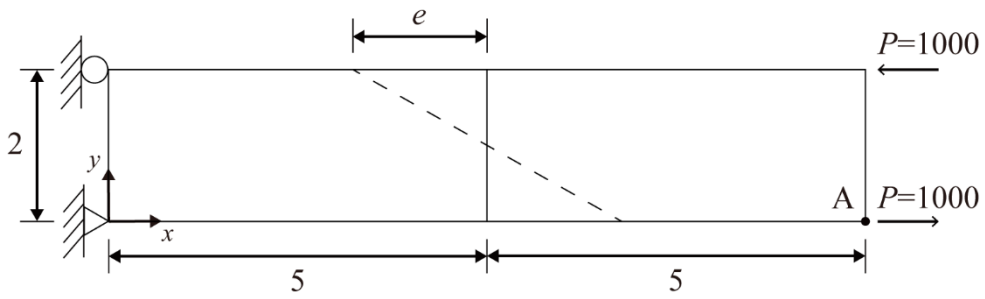

Figure 5.8. Cantilever beam represented by two elements with distortion parameter e ($E = 1500$, $\nu = 0.25$, thickness = 1).

Table 5.4 shows the tip deflections (v_A) at point A (shown in Fig. 5.8) normalized by the reference solution. The zero iteration in the Table 5.4 means there is no iteration because $\bar{E}$ and E are equal. When e varies from 0 to 5, the SU4 element shows results close to the exact solution through only one iteration.

Table 5.4. Normalized vertical displacements at point A in the cantilever beam for mesh distortion test with a distortion parameter e (Fig. 5.8), data in bold are the results obtained by the elements proposed in this paper, and the number in bracket is the number of iterations.

Elements	e						
	0	0.5	1	2	3	4	4.9
Q4	0.280	0.210	0.141	0.097	0.083	0.072	0.062
QM6 [41]	1.000	0.809	0.627	0.544	0.536	0.512	0.468
P-S [57]	1.000	0.810	0.629	0.550	0.547	0.531	0.498
SPS [61]	-	-	1.100	1.205	1.327	1.471	1.626
SYHP [61]	-	-	1.100	1.205	1.328	1.475	1.633
CPS4I [62]	1.000	0.735	0.562	0.503	0.504	0.494	0.466
QE2 [64]	1.000	0.812	0.634	0.565	0.575	0.579	0.569
$\bar{\mathbf{B}}$-QE4 [63]	1.000	0.812	0.634	0.565	0.575	0.579	0.569
QACM4 [60]	1.000	0.838	0.665	0.601	0.614	0.603	0.560
CQAC6 [65]	0.099	0.838	0.665	0.601	0.614	0.603	0.560
F-M QUAD4-P [66]	0.993	0.099	0.102	0.111	0.120	0.126	0.129
F-M QUAD4-R [67]	1.000	0.993	0.993	0.993	0.993	0.993	0.993
HSF-Q4θ-7β [68]	1.000	0.999	0.995	0.960	0.871	0.719	0.525
QAC4θ [75]	1.000	0.999	0.989	0.998	1.020	1.022	1.003
QAC4θM [75]	1.000	1.000	1.000	1.000	1.000	1.000	1.000
US-ATFQ4 [74]	1.000	1.000	1.000	1.000	1.000	1.000	1.000
SU4	**1.000(0)**	**1.000(1)**	**1.000(1)**	**1.000(1)**	**1.000(1)**	**1.000(1)**	**1.000(1)**

Exact [74]	1.000 (the value: 100)

5.4.3 Cantilever beam divided by five elements

The cantilever beam problem described in Fig. 5.9 is solved. The beam is divided by five irregular quadrilateral elements, and two loading cases are considered: (1) pure bending M, and (2) shear loading P at the free tip. The left end is fixed as shown in Fig. 5.9. Plane stress condition is employed with Young's modulus E=1500 and Poisson's ratio $\nu = 0.25$.

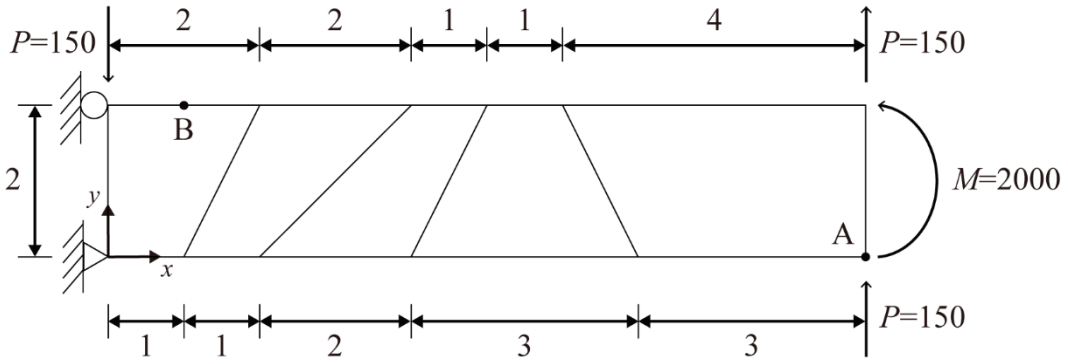

Figure 5.9. Cantilever beam divided by five distorted elements ($E = 1500$, $\nu = 0.25$, thickness $= 1$).

The results of the vertical deflection at point A (v_A) and the stress at point B (σ_{xB}) are given in Table 5.5. The SU4 element provides exact solution for pure bending cases, and high precision results for shear loading cases.

Table 5.5. the vertical deflection at point A (v_A) and the stress at point B (σ_{xB}) in the cantilever beam divided by five distorted elements (Fig. 5.9), data in bold are the results obtained by the elements proposed in this paper, and the number in bracket is the number of iterations.

Elements	Load M		Load P	
	v_A	σ_{xB}	v_A	σ_{xB}
Q4	45.7	-1761	50.7	-2448
QM6 [41]	96.1	-2497	98.0	-3235
CPS4I [62]	92.3	-2996	97.0	-3932
P-S [57]	96.2	-3014	98.2	-4137
NQ6 [69]	96.1	-2439	98.0	-3294
NQ10 [70]	96.0	-2986	97.9	-4021
SPS [61]	101.8	-3003	-	-
SYHP [61]	101.8	-3002	-	-
GC-Q6 [71]	95.0	-3036	96.1	-4182

QC6 [72]	96.1	-2439	98.1	-3339
QE2 [64]	96.5	-3004	98.3	-3906
$\overline{\mathbf{B}}$-QE4 [63]	96.5	-3004	98.3	-3906
QACM4 [60]	96.0	-3015	98.0	-4135
CQAC6 [65]	96.0	-3015	98.0	-4135
QAC4θ [75]	100.0	-3000	98.6	-3931
QAC4θM [75]	100.0	-3000	101.0	-3937
US-ATFQ4 [74]	100.0	-3000	101.5	-3938
SU4	**100.0(1)**	**-3000(1)**	**102.5(3)**	**-4173(3)**
Exact [74]	100.0	-3000	102.6	-4050

5.4.4 Cantilever beam divided by four elements

The cantilever beam problem described in Fig. 5.10 is investigated. The beam is divided by four distorted quadrilateral elements, and subjected to a quadratic distributed shear load at the right end. The left end is fully fixed as shown in Fig. 5.10. Plane stress condition is employed with Young's modulus E=30,000 and Poisson's ratio $\nu = 0.25$.

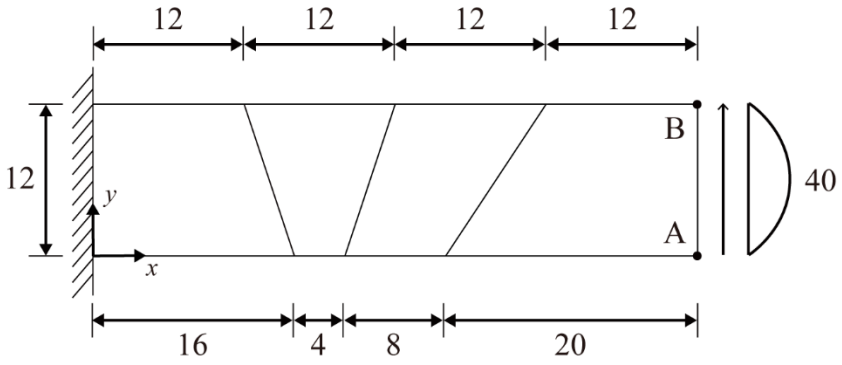

Figure 5.10. Cantilever beam divided by four distorted elements ($E = 30000$, $\nu = 0.25$, thickness = 1).

The results of the normalized vertical deflection at point A and B (v_A , v_B) are shown in Table 5.6. The SU4 element presents similar accuracy to the US-ATFQ4, which is unsymmetric element, and the result is close to the exact solution more than 99%.

Table 5.6. Normalized vertical deflection at point A and B (v_A , v_B) in the cantilever beam divided by four distorted elements (Fig. 5.10), data in bold are the results obtained by the elements proposed in this paper, and the number in bracket is the number of iterations.

Elements	Tip deflection		
	v_A	v_B	Average

Q4	0.598	0.599	0.598
QM6 [41]	0.917	0.924	0.920
CPS4I [62]	0.925	0.932	0.928
QACM4 [60]	0.922	0.929	0.926
CQAC6 [65]	0.922	0.929	0.926
HSF-Q4θ-7β [68]	0.985	0.975	0.980
QAC4θ [75]	0.990	0.988	0.989
QAC4θM [75]	0.990	0.988	0.989
US-ATFQ4 [74]	0.996	0.996	0.996
SU4	**0.993(2)**	**0.991(2)**	**0.992(2)**
Exact [74]	1.000 (the value: 0.3558)		

5.4.5 Thick curving beam

Thick curving beam is considered as shown in Fig. 5.11. The beam meshed into four elements is subjected to a shear force at the free end, and the bottom side is fully clamped as shown in Fig. 5.11. Plane stress condition is employed with Young's modulus E=1000 and Poisson's ratio $\nu = 0$.

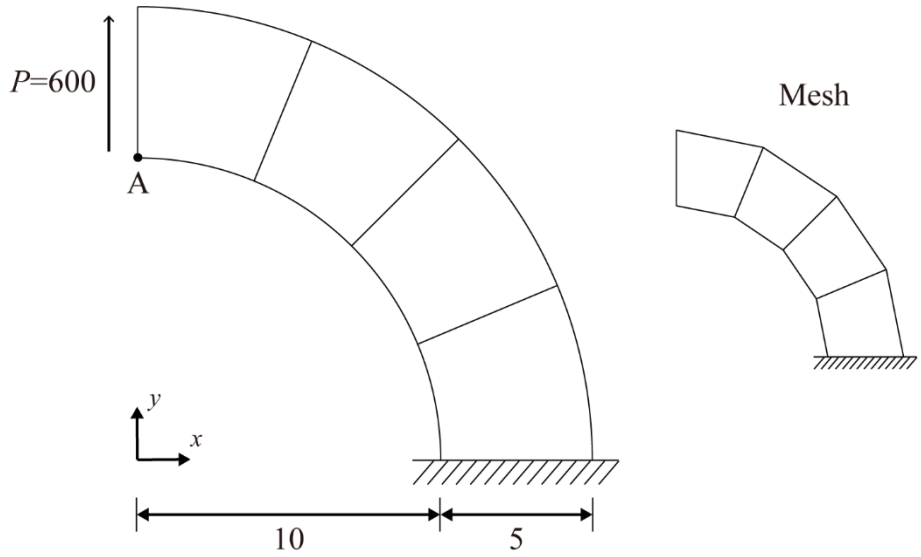

Figure 5.11. Thick curving beam description ($E = 1000$, $\nu = 0$, thickness $= 1$).

Table 5.7 shows the vertical deflection at point A (v_A). The SU4 element shows the best results among the elements that do not add drilling DOFs.

Table 5.7. Normalized vertical deflection at point A in the thick curving beam (Fig. 5.11), data in bold are the results obtained by the elements proposed in this paper, and the number in bracket is the number of iterations.

Elements	Tip deflection
Q4	0.643
QM6 [41]	0.928
P-S [57]	0.939
CPS4I [62]	0.939
PEAS7 [59]	0.939
QACM4 [60]	0.939
US-ATFQ4 [74]	0.958
US-Qθ [77]	0.995
SU4	**0.963(8)**
Exact [74]	1.000 (the value: 90.1)

5.4.6 Thin curving beam

Two type of thin curving beam described in Fig. 5.12 is solved. Two ratios of thickness-radius, (1) h/R=0.03 (E=365,010) and (2) h/R=0.006 (E=44,027,109), are considered. It is meshed into five elements. The beam is subjected to a shear force at the free end, and the bottom side is fully clamped as shown in Fig. 5.12. Plane stress condition is employed with Poisson's ratio $\nu = 0$.

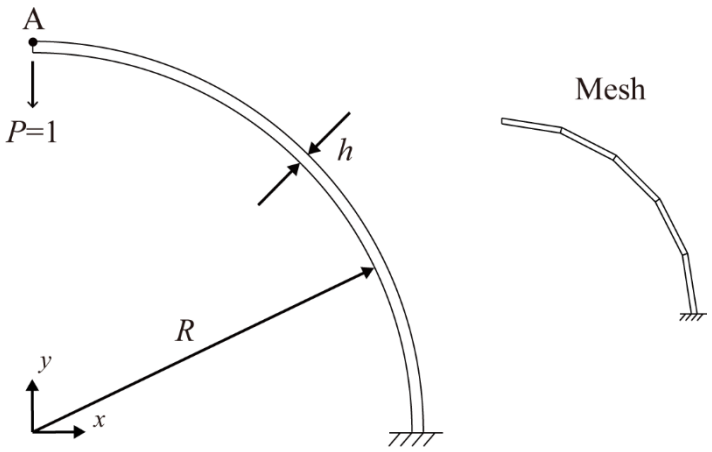

Figure 5.12. Thin curving beam description ((1) h/R=0.03 ($E = 365,010$, $\nu = 0$, thickness $= 1$) and (2) h/R=0.006 ($E = 44,027,109$, $\nu = 0$, thickness $= 1$)).

Table 5.8 shows the vertical deflection at point A (v_A). Regardless of h/R, the SU4 elements presents excellent results comparable to that of the element using the drilling DOFs.

Table 5.8. The vertical deflection at point A in the thin curving beam (Fig. 5.12), data in bold are the results obtained by the elements proposed in this paper, and the number in bracket is the number of iterations.

Elements	Tip deflection	
	h/*R*=0.03	*h*/*R*=0.006
Q4	0.016	0.001
QM6 [41]	0.339	0.022
CPS4I [62]	0.650	0.173
QACM4 [60]	0.639	0.026
QAC4θ [75]	0.712	0.645
QAC4θM [75]	1.000	1.000
US-ATFQ4 [74]	0.987	0.987
US-Q4θ [77]	1.000	1.008
SU4	**1.005(7)**	**1.003(4)**
Exact [74]	1.000	

5.4.7 Cantilever beam for rotation dependency test

The cantilever beam described in Fig. 5.13 is considered for rotation dependency test. This benchmark problem is proposed by Cen e*t al.* [74]. The beam is divided by two irregular quadrilateral elements. It is subjected to a bending moment at the right end, and the left side is fully clamped as shown in Fig. 5.13. The beam is rotated counterclockwise from 0° to 90° in steps of 10°. Plane stress condition is employed with Young's modulus E=100 and Poisson's ratio $\nu = 0.3$.

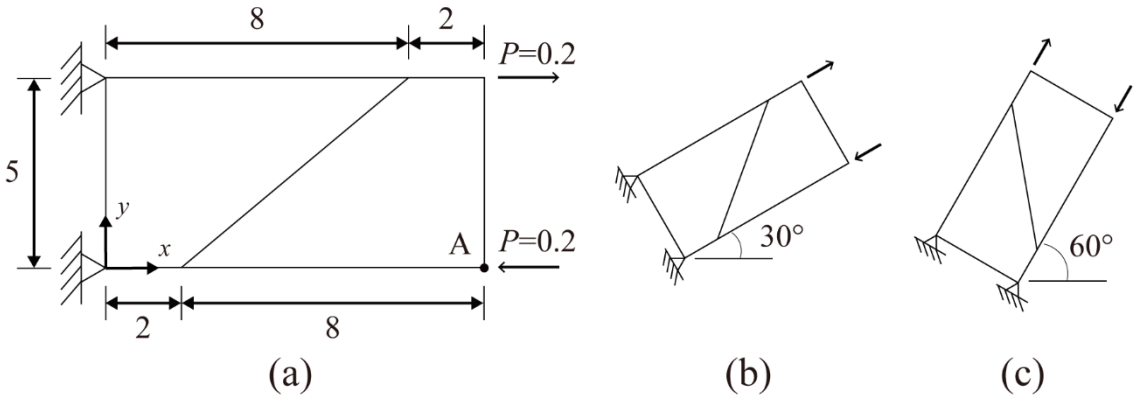

Figure 5.13. Cantilever beam represented ($E = 100$, $\nu = 0.3$, thickness = 1) by two elements with rotation angle of (a) 0°, (b) 30°, and (c) 60°.

The displacements at point A (u_A , v_A) are monitored and shown in Table 5.9 for each rotated angle. The iteration number of SU4 element increases as the coordinates rotate, but the result is almost invariant.

Table 5.9. The displacements at point A according to rotation angle in the cantilever beam for rotation dependency

test (Fig. 5.13) using the SUFE, and the number in bracket at the last column is the number of iterations.

Rotation angle	Tip deflection			
	u_A	v_A	$\sqrt{u_A{}^2+v_A{}^2}$	Normalized
0°	2.4000E-02	4.8000E-02	0.05367	0.9938(1)
10°	1.5306E-02	5.1397E-02	0.05363	0.9931(4)
20°	6.1526E-03	5.3264E-02	0.05362	0.9929(5)
30°	3.1876E-03	5.3519E-02	0.05361	0.9928(6)
40°	1.2456E-02	5.2180E-02	0.05365	0.9934(6)
50°	2.1325E-02	4.9222E-02	0.05364	0.9934(8)
60°	2.9540E-02	4.4766E-02	0.05363	0.9932(6)
70°	3.6846E-02	3.8950E-02	0.05362	0.9929(5)
80°	4.3173E-02	-3.1982E-02	0.05373	0.9950(3)
90°	4.8000E-02	2.4000E-02	0.05367	0.9938(1)
Ref. solution [74]	-	-	0.054	1.000

5.4.8 Cook's skew beam problem

The Cook's skew beam problem described in Chapter 4.5.1 is solved. Table 5.10 shows the tip deflections (v_A) at point A. The SU4 element outperforms the other elements.

Table 5.10. The tip deflections (v_A) at point A according to mesh densities in Cook's skew beam problem (Fig. 4.11), data in bold are the results obtained by the elements proposed in this paper, and the number in bracket is the number of iterations.

Elements	v_A				Normalized v_A			
	2×2	4×4	8×8	16×16	2×2	4×4	8×8	16×16
Q4	11.80	18.29	22.08	23.43	0.492	0.763	0.921	0.978
QM6 [41]	21.05	23.03	-	23.88	0.878	0.961	-	0.996
CPS4I [62]	21.05	23.02	23.69	23.88	0.878	0.961	0.989	0.996
P-S [57]	21.13	23.02	23.69	23.88	0.882	0.961	0.989	0.996
HH4-3β [58]	22.08	23.44	23.78	23.91	0.921	0.978	0.992	0.998
QE2 [64]	21.35	23.04	-	23.88	0.891	0.961	-	0.996
$\bar{\mathbf{B}}$-QE4 [63]	21.35	23.04	-	23.88	0.891	0.961	-	0.996
QACM4 [60]	20.74	22.99	23.69	-	0.865	0.959	0.989	-
F-M QUAD4-P [66]	21.57	23.57	23.86	23.92	0.900	0.984	0.996	0.998
F-M QUAD4-P [67]	20.40	23.19	23.76	23.89	0.851	0.968	0.991	0.997
HSF-Qθ-7β [68]	22.55	23.44	23.79	23.90	0.941	0.978	0.993	0.997

QAC4θ [75]	21.00	23.05	23.66	-	0.876	0.962	0.987	-
QAC4θM [75]	22.25	23.42	23.78	-	0.928	0.977	0.992	-
US-ATFQ4 [74]	22.76	23.43	23.79	23.91	0.950	0.978	0.993	0.998
US-Q4θ [77]	22.55	23.44	23.79	23.90	0.941	0.978	0.993	0.997
SU4	**23.80 (118)**	**23.93 (75)**	**23.96 (6)**	**23.95 (4)**	**0.993 (118)**	**0.998 (75)**	**1.000 (6)**	**0.999 (4)**
Ref. solution [74]	23.9652				1.000			

To investigate the predictive capability of the elements in detail, convergence studies are performed. Fig. 5.14 displays the convergence curves. The reference solution is obtained using a 100×100 mesh of standard 9-node quadrilateral elements. The SU4 element outperforms the other elements even if it does only one iteration.

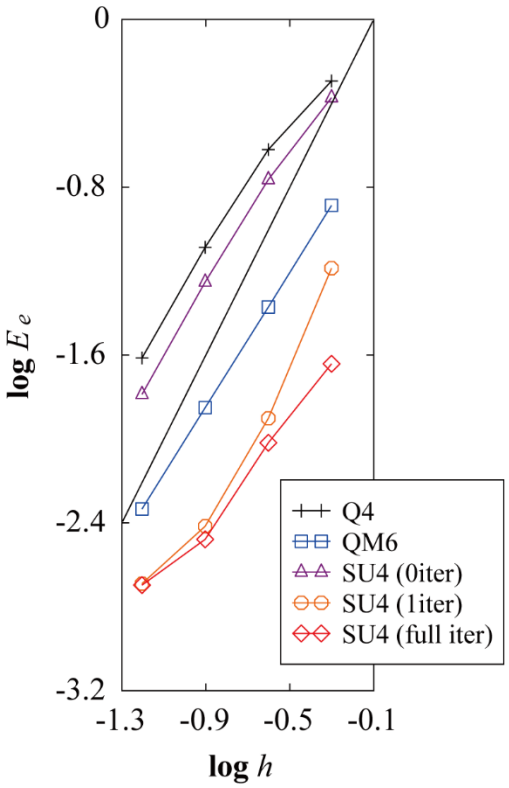

Figure 5.14. SU4 element convergence curves in the Cook's skew beam problem: The bold lines represent the optimal convergence rates.

Fig. 5.15 shows the strain energies according to the number of iterations. The change of strain energy is rapid at the beginning of iteration, and the change converges as the iteration continues.

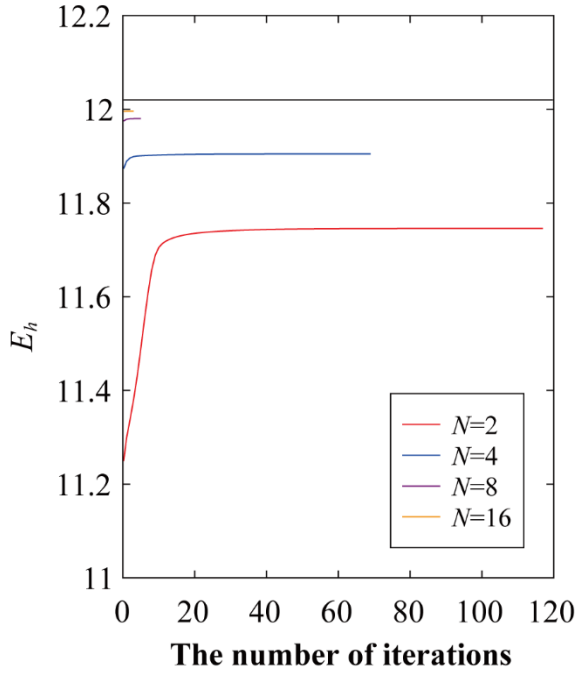

Figure 5.15. Strain energies of SU4 according to the number of iterations in the Cook's skew beam problem: The black bold lines represent the reference value.

5.5 Computational efficiency

The computation cost of the proposed self-updated finite elements is measured through the Cook's skew beam problem described in Section 5.4.8. The Q4, QM6 and SU4 are compared, and SU4 is limited to one iteration. The computation times taken from reading the input file to solving the linear equations are measured. All calculations were performed using a quad-core desktop (Intel(R) Core (TM) i7-2600 CPU @ 3.40 GHz, 12 GB memory, Microsoft Windows 10 64bit) under Python environment. The linear equations were solved using a direct solver in NumPy library [47].

Fig. 5.16 displays the relations between the computation time versus solution accuracy. Fig. 5.16(a) shows the result using the relative errors in the energy norm in Eq. (4-25) as the solution accuracy, and Fig. 5.16(b) shows the result using the relative errors in the vertical displacement ($E_v = \left|(v - v_{ref}) / v_{ref}\right|$) at the Point A in Fig. 15 as the solution accuracy. Meshes with $N = 2$, 4, 8 and 16 are used for the assessment. In Cook's skew beam problem, the efficiency of SU4 element by one iteration is slightly better than that of QM6.

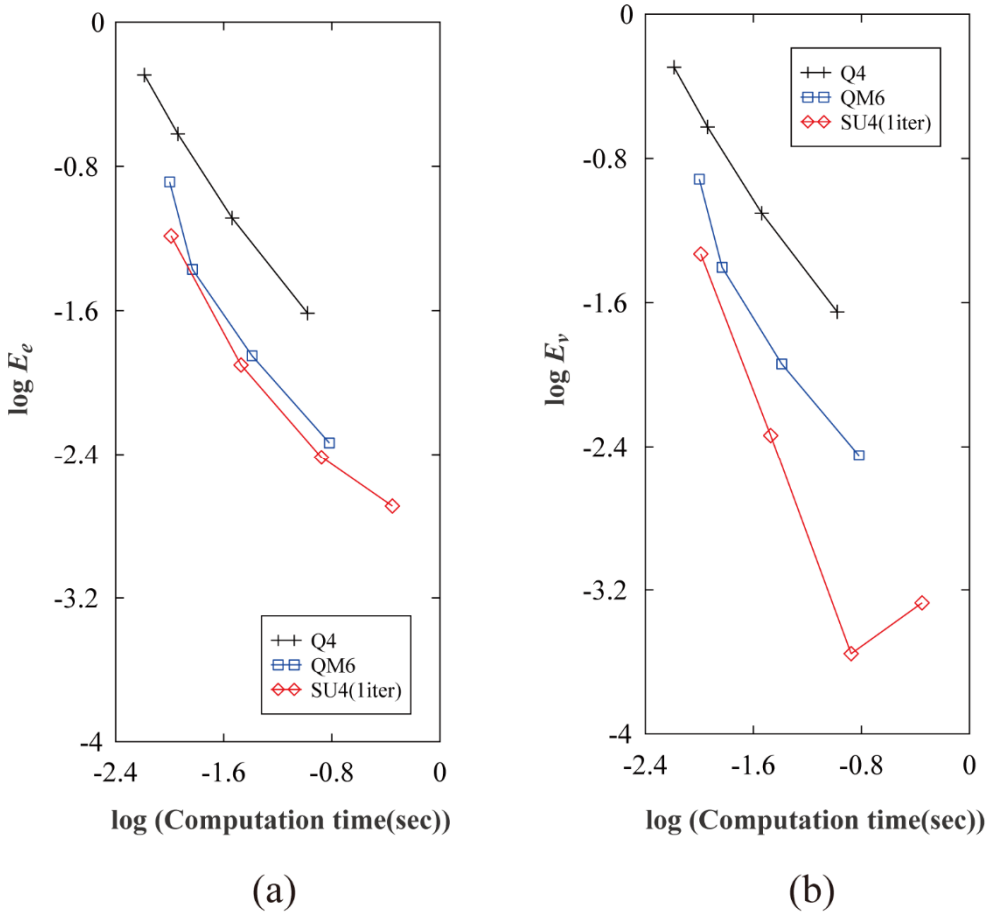

Figure 5.16. SU4 computational efficiency curves in the Cook's skew beam problem. The *y*-axes are relative errors (a) in strain energy and (b) in the vertical displacement.

5.6 Concluding remarks

Self-Updated 4-node solid element is presented. The generation of elements uses an analytical solution for each deformation mode. The local coordinates for bending modes are selected iteratively according to the displacements of the element using the deep learning network as a brain of the element. The performance of SUFE was evaluated through various numerical examples. While maintaining the excellent results of the developed elements, the results of the Cook's skew beam problem showed better results without any mesh refinement.

Chapter 6. Conclusions

In this paper, we proposed a methodology to generate stiffness matrices of finite elements using deep learning.

In Chapter 4, The DL8 and DL4 elements were developed. In particular, the DL8 element showed promising ability in both accuracy and computational efficiency. This method is not limited to the 2D solid elements, and can be extended to various finite elements, including 3D solid, beam, and shell finite elements [48-54].

In Chapter 5, we propose Self-Updated 4-node solid element. Since the stiffness matrix of the proposed element is generated symmetrically, there is no problem in solving the engineering domain using the general FEM solver. The SUFE showed better performance without any mesh refinement at the Cook's skew beam problem while maintaining the excellent results of the developed elements.

These methods have great implications by showing that artificial intelligence can be used for finite element development. Of course, applying the method to nonlinear analysis is also very valuable.

Appendix A. Effect of material properties on the internal displacements

We here investigate the effect of two material properties (i.e., Young's modulus and Poisson's ratio) on the internal displacements $\hat{\mathbf{u}}_{\mathrm{I}}^{(n)}$ calculated by the prescribed outer displacements $\hat{\mathbf{u}}_{\mathrm{O}}^{(n)}$. The internal displacement vector $\hat{\mathbf{u}}_{\mathrm{I}}^{(n)}$ is calculated using the matrices $\hat{\mathbf{K}}_{\mathrm{II}}$ and $\hat{\mathbf{K}}_{\mathrm{IO}}$ in Eq. (4-10).

The material law matrix $\mathbf{C}^{(n)}$ is a function of Young's modulus (E) and Poisson's ratio (ν) represented by

$$\mathbf{C}^{(n)}(E,\nu) = E\overline{\mathbf{C}}^{(n)}(\nu), \tag{A1}$$

in which $\overline{\mathbf{C}}^{(n)}$ is a function of Poisson's ratio.

Then, the following equation can be derived

$$\hat{\mathbf{K}}_{\mathrm{II}} = E\overline{\mathbf{K}}_{\mathrm{II}} \quad \text{and} \quad \hat{\mathbf{K}}_{\mathrm{IO}} = E\overline{\mathbf{K}}_{\mathrm{IO}}, \tag{A2}$$

in which $\overline{\mathbf{K}}_{\mathrm{II}}$ and $\overline{\mathbf{K}}_{\mathrm{IO}}$ are matrices independent of Young's modulus.

Substituting Eq. (4-A2) into Eq. (4-10), the internal displacement vector $\hat{\mathbf{u}}_{I}^{(n)}$ is obtained regardless of Young's modulus (E):

$$\hat{\mathbf{u}}_{\mathrm{I}}^{(n)} = -\left(\overline{\mathbf{K}}_{\mathrm{II}}\right)^{-1}\overline{\mathbf{K}}_{\mathrm{IO}}\hat{\mathbf{u}}_{\mathrm{O}}^{(n)}. \tag{A3}$$

Appendix B. Mesh density of the reference data model

The deep learned finite elements are based on the reference data model; thus, their performance depends on the mesh density of the reference data model. Here, we study the dependency considering three reference data models with various mesh densities: N = 10, 30, and 50. The three neural networks corresponding to the mesh densities were obtained via the procedure described in Chapter 4.2.1 and 2.2.2

Table A1 represents the training and test data errors of the trained neural networks. The use of the fine mesh reference data model (i.e., N = 50) leads to less error compared to that of the coarse mesh reference data model (N = 10 and 30). Fig. A1 shows the convergence curves of the DL8 and DL4 elements generated from the three reference data models in the wrench problem illustrated in Fig. 4.20.

Table A1. Averaged errors of the trained neural networks according to the mesh density of the reference data model (N).

N	Training data error (%)	Test data error (%)
10	1.57	2.68
30	1.55	1.98
50	1.24	1.67

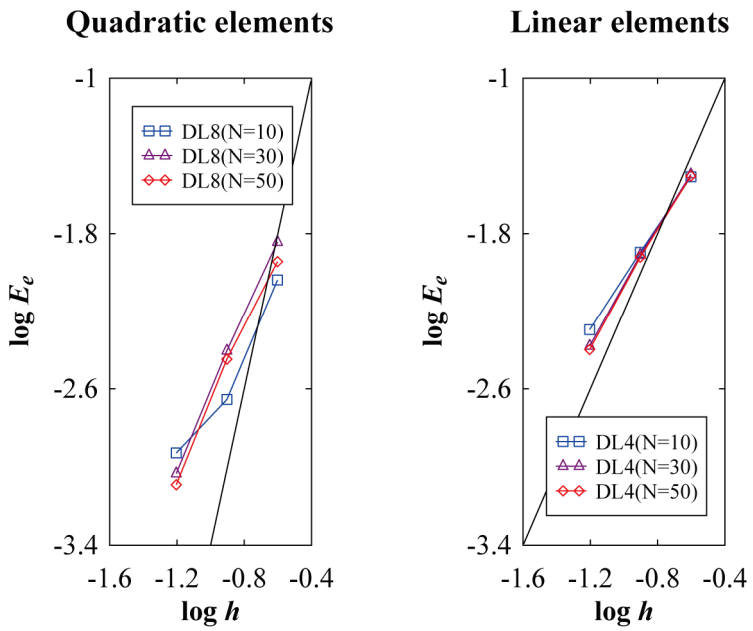

Figure A1. Convergence curves according to the mesh density of the reference data model (N) in the wrench problem: The bold lines represent the optimal convergence rates.

Appendix C. Convergence behavior of the DL8 element in a curved geometry model

As mentioned in Chapter 4.2.1, the geometry of the DL8 element is limited to a quadrilateral whose mid-side node are placed at the center of the adjacent corner nodes and thus curved geometries were not trained for the element. Nevertheless, it is highly interesting to investigate the convergence behavior of the DL8 element when modeling a curved geometry.

Herein, we consider the tool zig problem described in Fig. A2. The geometry and boundary conditions are shown in Fig. A2. A uniformly distributed load of q = 1 is applied along the top side. The plane stress condition is considered using Young's modulus $E = 2.0\times10^{11}$ and Poisson's ratio $\nu = 0.3$. Three different meshes are considered, as shown in Fig. A2(a)–(c). The reference solution is obtained using the Q9 elements and the mesh in Fig. A2(d).

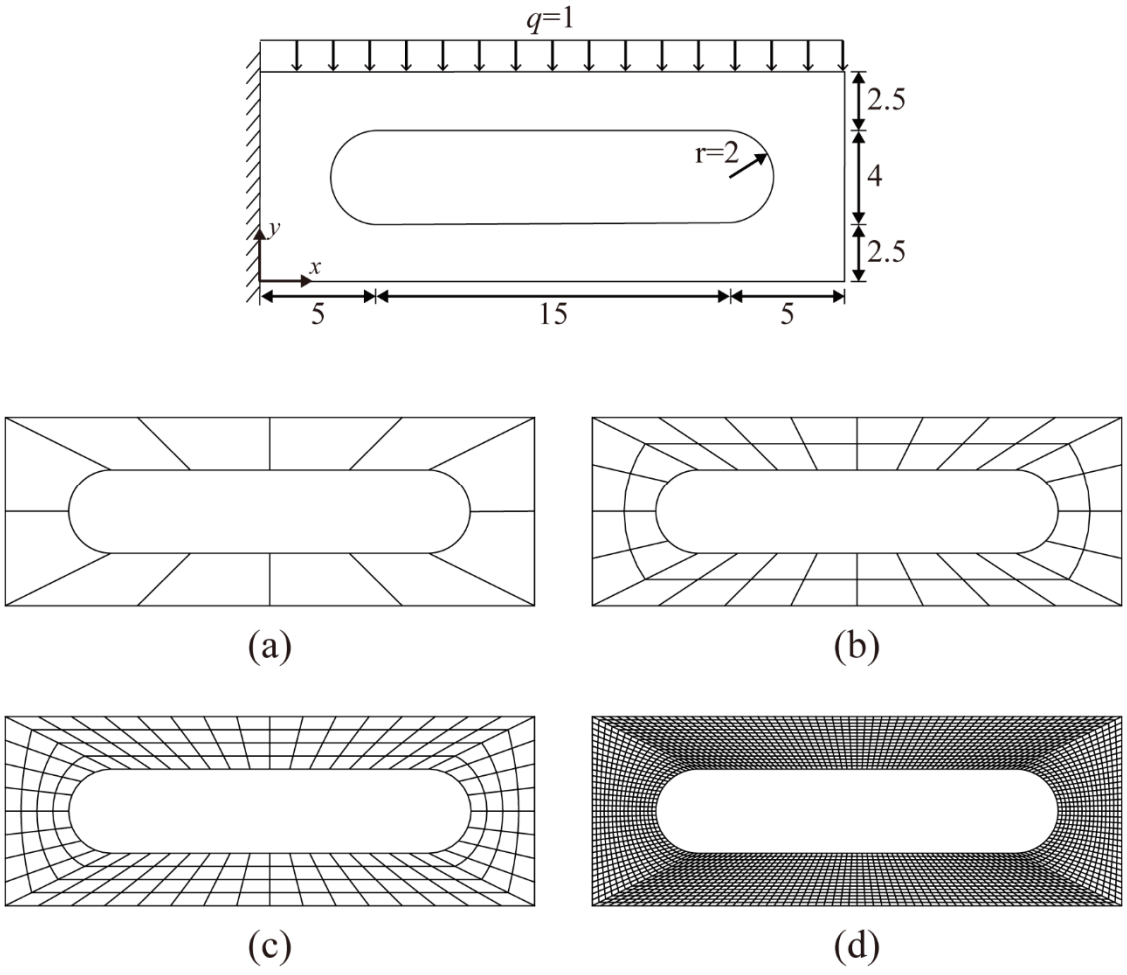

Figure A2. Tool zig problem ($E = 2.0\times10^{11}$, $\nu = 0.3$, thickness $= 2.0$): (a) Coarse mesh (N = 2). (b) Medium mesh (N = 4). (c) Fine mesh (N = 8). (d) Mesh used for the reference solution.

Fig. A3 shows the convergence curves for the quadratic elements. As expected, the DL8 element does not exhibit a good convergence behavior, compared to other quadratic elements.

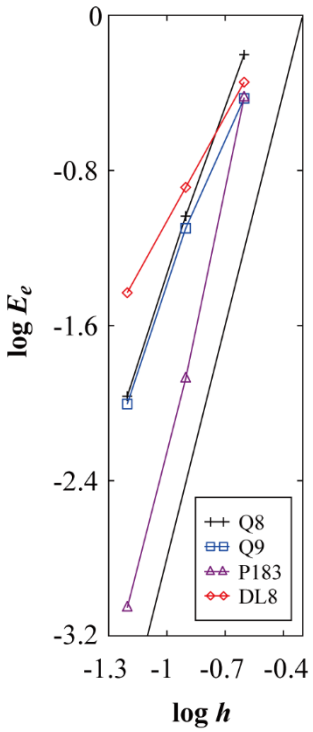

Figure A3. Convergence curves in the tool zig problem: The bold line represents the optimal convergence rates.

Appendix D. Effect of data sampling method on network training in the deep learned finite element

We here investigate the effect of the data sampling method. The normalized geometry of the element is determined by $\mathbf{x}_3$ and $\mathbf{x}_4$, and the determination method is various. In this appendix, the following three methods were used to investigate the effect of the method.

(Method 1) The position x_i and y_i are evenly spaced on Cartesian coordinates as,

$$x_3 = -1 + i \cdot dx, \tag{A4}$$

$$y_3 = j \cdot dy, \tag{A5}$$

$$x_4 = -1 + k \cdot dx, \tag{A6}$$

$$y_4 = l \cdot dy \quad \text{for} \quad i, j, k, l = 0, 1, 2, ..., p, \tag{A7}$$

with $dx = 3/p$ and $dy = 1/p$,

where p is a number to determine the training data size, and the (i, j, k, l) set is unique.

(Method 2) The position x_i and y_i are determined using polar coordinate system as,

$$x_3 = 1 + i \cdot dr \cos(j \cdot d\theta), \tag{A8}$$

$$y_3 = i \cdot dr \sin(j \cdot d\theta), \tag{A9}$$

$$x_4 = k \cdot dr \cos(l \cdot d\theta), \tag{A10}$$

$$y_4 = k \cdot dr \sin(l \cdot d\theta) \quad \text{for} \quad i, j, k, l = 0, 1, 2, ..., p, \tag{A11}$$

with $dr = 1/p$ and $d\theta = 1/p$,

where p is a number to determine the training data size, and the (i, j, k, l) set is unique.

(Method 3) The position x_i and y_i are randomly determined as,

$$x_3 = 1 + \beta_3 \cos(\theta_3), \tag{A12}$$

$$y_3 = \beta_3 \sin(\theta_3), \tag{A13}$$

$$x_4 = \beta_4 \cos(\theta_4), \tag{A14}$$

$$y_4 = \beta_4 \sin(\theta_4), \tag{A15}$$

where β_3 and β_4 are randomly generated radius with a uniform distribution in the range of -1 to 1, and θ_3 and θ_4 are randomly generated radian angles with a uniform distribution in the range of 0 to π.

Method 1 and Method 2 generate data regularly, and Method 3 used in this study generates data randomly. In the generated data, we excluded severely distorted geometries such as quadrilaterals with an interior angle of less than 10° or greater than 170° and ratios between the maximum and minimum side lengths of greater than 10. Young's modulus ($E = 2.0\times10^{11}$) was adopted, and Poisson's ratio ($\nu^{(n)}$) was randomly applied with a uniform distribution in the range of 0–0.499999999

Various data sets are generated for each method. The network for each set is trained according to Chapter 4.2.2. Fig. A4 shows a comparison of the errors according to data size for each method. Here, as the test data, 30,000 data were generated. The results show that training with the dataset of Method 3 is the best of the suggested methods

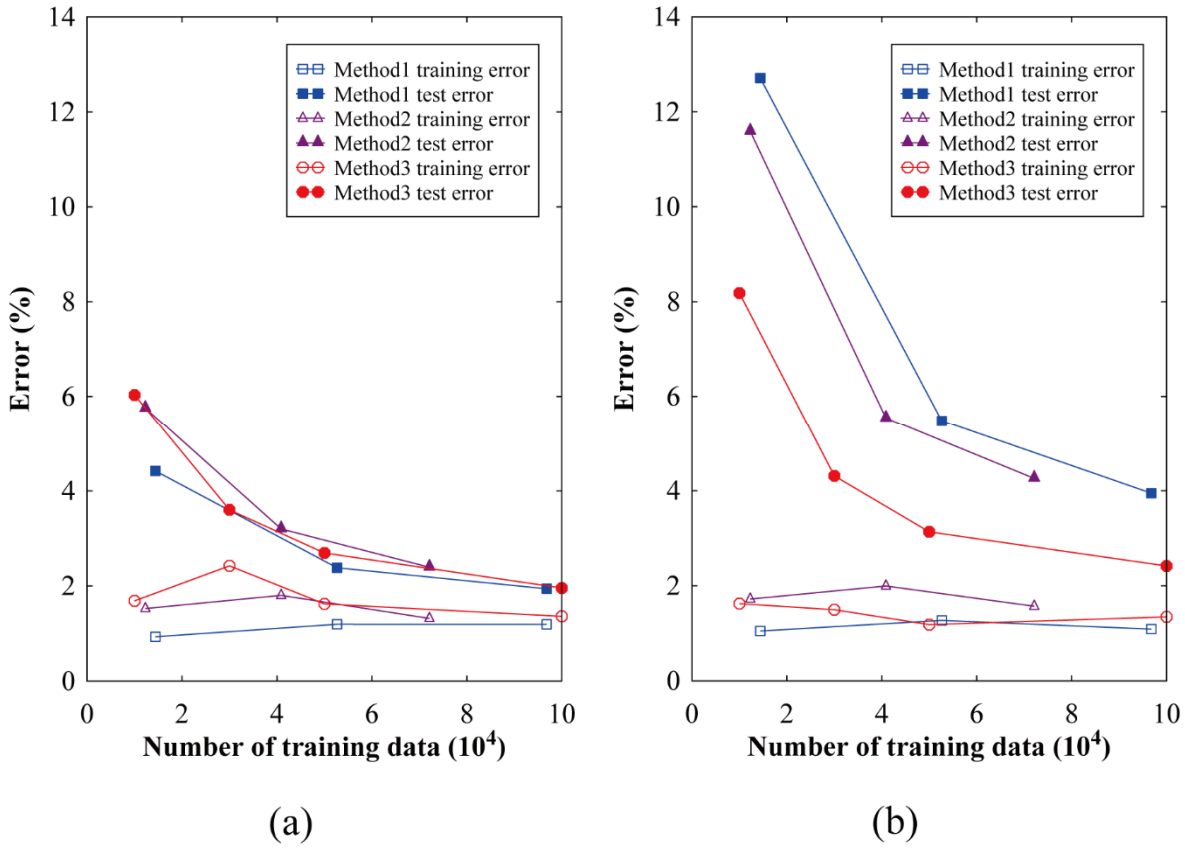

Figure A4. Error curves according to the data generation method using the test data generated in (a) Method 1 and (b) Method 3.

.

Appendix E. Effect of input degree of freedom on the training in the deep learned finite element

The 3D finite elements increase the input degree of freedom and the training difficulty when applying the DLFE method. In this appendix, we investigate the effect of input degree of freedom on training to evaluate the applicability of the proposed method to 3D finite elements.

In order to increase the input degree of freedom, we randomly rotate the normalized geometry generated according to Chapter 4.2.1, and $\mathbf{x}_2^{(n)}$ is added to $\mathbf{D}_x^{(n)}$ as,

$$\mathbf{D}_x^{(n)} = \left[\mathbf{x}_2^{(n)\mathrm{T}} \quad \mathbf{x}_3^{(n)\mathrm{T}} \quad \mathbf{x}_4^{(n)\mathrm{T}} \right]. \tag{A16}$$

The network is trained according to Chapter 4.2.2 with the rotated data. Table A2 shows the training results according to the depth and width of the network in Fig. 4.4 for each size of training data. The training error is increased compared to the results in Chapter 4.2.2. The results shown in Table A1 are improved as increasing the network width or data size. In order to apply the proposed method to 3D finite elements, more training data and more effective and appropriate networks is needed.

Table A2. Averaged errors of training according to the depth and width of the network in Fig. 4.4 for each size of the increased input degree of freedom data (training data error (%)/test data error (%)).

Number of weights per layer	Number of Training data	Number of layers				
		4	5	6	7	8
378	300,000	4.46/5.96	4.31/6.02	4.20/6.08	4.22/6.21	4.06/6.16
378	600,000	4.20/4.97	4.04/4.94	4.02/5.01	4.03/5.13	3.88/5.08
756	300,000	3.04/4.86	3.07/5.04	2.72/4.93	2.90/5.54	2.81/6.00
756	600,000	2.64/3.64	2.72/3.88	2.85/4.15	2.73/4.06	2.75/4.15
1512	300,000	2.53/5.97	2.31/4.82	2.21/7.51	2.26/4.85	2.35/5.06
1512	600,000	2.25/3.51	2.32/3.74	2.02/3.59	2.15/4.12	2.16/3.66

Appendix F. Effect of network structure and data size in the deep learned finite element

In this appendix, we investigate the effect of the network structure and data size in the DLFE. Training is performed by changing the width, depth of the network and data size according to Chapter 4.2.2 except batch size. The batch size is used as half of the training data size.

Table A3 shows the training results. As the depth and width of the network, and data size increases, the training and test error decrease. However, errors do not show a significant reduction for more than six layers.

Table A3. Averaged errors of training according to the depth and width of the network in Fig. 4.4 for each size of training data (training data error (%)/test data error (%)).

Number of weights per layer (network width)	Number of Training data	Number of layers (network depth)				
		2	4	6	8	10
189	10,000	11.70/22.19	4.49/11.23	3.17/10.32	3.15/10.22	3.16/11.14
189	50,000	11.55/14.02	3.65/5.69	2.98/5.11	2.82/5.11	2.71/4.96
189	100,000	11.74/13.20	3.38/4.48	2.80/4.16	2.66/3.97	2.67/4.10
378	10,000	11.08/20.77	3.77/10.55	2.67/9.73	2.44/9.37	2.50/11.30
378	50,000	10.28/12.86	2.57/4.63	1.89/4.04	1.97/4.22	1.97/4.39
378	100,000	9.99/11.51	2.52/3.81	2.05/3.32	1.78/3.13	1.70/3.08
756	10,000	9.54/19.58	2.60/9.06	2.14/8.83	3.04/9.61	2.17/8.46
756	50,000	8.96/11.65	1.92/3.93	1.83/3.95	1.46/3.64	1.67/3.82
756	100,000	9.04/10.47	1.86/3.08	1.57/2.83	1.40/2.63	1.50/2.86

Appendix G. Effect of network hyper parameter in the deep learned finite element

In this appendix, we investigate the effect of the learning rate, batch size, activation function type in the DLFE network. Training is performed by changing the learning rate, batch size, activation function type according to Chapter 4.2.2. The network is trained with a total of 50,000 data (M = 50,000). To test the network, 10,000 data is generated.

Table A4 shows the training results. The results show that training is the best performed when the learning rate decreases from 0.01 to 0 as the epoch progressed. And when the batch size is equal to the training data size, the training result is not good. In the activation function, ELU and Leaky ReLU shows better results than other activation function, but when the learning rate is reduced from 0.01 to 0, there is no significant difference depending on the activation function.

Table A4. Averaged errors of training according to the learning rate, batch size, activation function type using the network in Fig. 4.4 (training data error (%)/test data error (%)).

Type of activation function	Batch size	The learning rate				
		0.001	0.005	0.01	0.05	0.01→0
ELU	10,000	4.27/6.39	4.23/6.29	4.32/6.19	7.29/10.52	2.43/4.52
ELU	25,000	5.60/7.51	4.63/6.57	4.42/6.21	7.40/10.37	2.20/4.30
ELU	50,000	9.68/11.72	7.50/9.41	8.38/10.67	10.76/12.79	2.55/4.81
Sigmoid	10,000	7.86/9.79	10.90/13.40	14.51/17.82	30.42/37.42	2.07/3.96
Sigmoid	25,000	9.47/11.43	8.87/11.25	13.06/15.27	22.17/26.39	2.41/4.35
Sigmoid	50,000	16.94/18.80	12.03/13.59	16.50/20.07	24.41/30.25	3.33/5.19
tanh	10,000	6.66/8.97	7.37/10.00	8.55/11.82	12.38/19.19	2.20/4.53
tanh	25,000	8.00/10.63	7.41/8.89	8.13/10.21	13.13/16.06	1.97/4.21
tanh	50,000	13.26/15.70	9.84/12.45	11.02/13.28	20.43/24.50	2.60/4.90
Leaky ReLU	10,000	4.40/6.83	6.16/8.24	4.91/7.25	6.91/9.55	2.50/4.95
Leaky ReLU	25,000	6.35/9.47	4.08/6.97	4.01/6.45	7.27/10.24	2.18/5.19
Leaky ReLU	50,000	9.57/13.34	6.73/9.10	6.76/8.79	9.23/12.33	2.37/5.77

Appendix H. Effect of network structure and data size in the self-updated finite element

In this appendix, we investigate the effect of the network structure and data size in the SUFE. Training is performed by changing the width, depth of the network and data size according to Chapter 5.2.3 except batch size. The batch size is used as half of the training data size.

Table A5 shows the training results. As the depth and width of the network, and data size increases, the training and test error decrease.

Table A5. Cost function value of training and test data according to the depth and width of the network in Fig. 5.3 for each size of training data (cost function value of the training data/cost function value of the test data).

Number of weights per layer (network width)	Number of Training data	Number of layers (network depth)				
		2	4	6	8	10
160	10,000	3.20/3.56	2.23/3.43	1.21/2.76	1.03/2.46	1.05/2.28
160	50,000	3.14/3.23	1.80/2.51	0.84/1.79	0.70/1.50	0.63/1.30
160	100,000	3.09/3.13	1.84/2.28	0.80/1.49	0.62/1.20	0.56/1.02
160	200,000	3.05/3.05	1.88/2.10	0.94/1.35	0.64/1.01	0.57/0.84
320	10,000	3.22/3.61	1.91/3.37	0.92/2.59	0.90/2.38	0.93/2.36
320	50,000	3.06/3.18	1.59/2.48	0.72/1.72	0.59/1.40	0.49/1.29
320	100,000	3.03/3.08	1.52/2.16	0.62/1.47	0.47/1.13	0.43/1.01
320	200,000	3.03/3.06	1.62/2.00	0.64/1.22	0.46/0.91	0.44/0.80
640	10,000	3.08/3.67	1.33/3.16	0.79/2.54	0.79/2.26	0.64/2.21
640	50,000	2.99/3.16	1.15/2.43	0.42/1.62	0.51/1.38	0.39/1.19
640	100,000	2.97/3.05	1.34/2.15	0.47/1.38	0.36/1.09	0.36/0.96
640	200,000	2.98/3.01	1.49/1.92	0.51/1.20	0.38/0.88	0.37/0.81

Appendix I. Effect of network hyper parameter in the self-updated finite element

In this appendix, we investigate the effect of the learning rate, batch size, activation function type in the SUFE network. Training is performed by changing the learning rate, batch size, activation function type according to Chapter 5.2.3. The network is trained with a total of 100,000 data (M = 100,000). To test the network, 10,000 data is generated.

Table A6 shows the training results. The results show that training is the best performed when the learning rate decreases from 0.01 to 0 as the epoch progressed. And the smaller the batch size, the better the training results. There is no significant difference depending on the activation function.

Table A6. Cost function value of training and test data according to the learning rate, batch size, activation function type using the network in Fig. 5.3 (cost function value of the training data/cost function value of the test data).

Type of activation function	Batch size	The learning rate				
		0.001	0.005	0.01	0.05	0.01→0
ELU	25,000	2.26/2.46	0.84/1.45	0.85/1.46	0.84/1.47	0.65/1.44
ELU	50,000	3.74/3.87	0.99/1.54	0.83/1.47	0.96/1.51	0.64/1.48
ELU	100,000	4.67/4.76	1.71/1.98	1.16/1.63	1.05/1.61	1.25/1.79
Sigmoid	25,000	2.00/2.20	1.09/1.48	1.29/1.54	1.61/1.72	0.56/1.22
Sigmoid	50,000	3.74/3.84	1.11/1.56	1.08/1.51	1.52/1.66	0.72/1.50
Sigmoid	100,000	4.70/4.77	1.68/1.95	1.19/1.67	1.26/1.57	1.14/1.79
tanh	25,000	2.17/2.56	0.72/1.48	0.79/1.43	1.24/1.52	0.60/1.45
tanh	50,000	3.74/4.02	0.79/1.59	0.88/1.55	1.03/1.36	0.46/1.55
tanh	100,000	4.65/4.85	1.74/2.16	0.89/1.68	0.96/1.61	1.12/2.01
Leaky ReLU	25,000	2.19/2.48	0.99/1.54	0.88/1.46	1.08/1.53	0.61/1.48
Leaky ReLU	50,000	3.76/3.95	1.04/1.63	0.97/1.52	1.00/1.57	0.64/1.50
Leaky ReLU	100,000	4.61/4.78	1.62/1.95	1.13/1.64	1.07/1.57	1.10/1.81

Bibliography

[1] K.-J. Bathe, *Finite element procedures*, Prentice Hall, 2006.

[2] J.-M. Jin, *The finite element method in electromagnetics*, John Wiley & Sons, 2015.

[3] P.M. Gresho, R.L. Sani, *Incompressible flow and the finite element method. Volume 1: advection-diffusion and isothermal laminar flow*, John Wiley & Sons, 1998.

[4] P. Lesaint and P.-A. Raviart, "On a finite element method for solving the neutron transport equation," *Publications mathématiques et informatique de Rennes*, no. S4, pp. 1-40, 1974.

[5] J. Donea, S. Giuliani, J.-P. Halleux, "An arbitrary Lagrangian-Eulerian finite element method for transient dynamic fluid-structure interactions", *Computer methods in applied mechanics and engineering*, vol. 33, no. 1-3, pp. 689-723, 1982.

[6] J.L. Volakis, A. Chatterjee, L.C. Kempel, *Finite element method electromagnetics: antennas, microwave circuits, and scattering applications*, John Wiley & Sons, 1998.

[7] J.N. Reddy, D.K. Gartling, *The finite element method in heat transfer and fluid dynamics*, CRC press, 2010.

[8] V. Girault and P.-A. Raviart, "Finite element approximation of the Navier-Stokes equations," *Lecture Notes in Mathematics*, Berlin Springer Verlag, vol. 749, 1979.

[9] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *The bulletin of mathematical biophysics*, vol. 5, no. 4, pp. 115-133, 1943.

[10] F. Rosenblatt, "The perceptron: a probabilistic model for information storage and organization in the brain," *Psychological review*, vol. 65, no. 6, p. 386, 1958.

[11] G. E. Hinton, S. Osindero, and Y.-W. Teh, "A fast learning algorithm for deep belief nets," *Neural Computation*, vol. 18, no. 7, pp. 1527-1554, 2006.

[12] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, no. 6088, pp. 533-536, 1986.

[13] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *in Advances in neural information processing systems*, pp. 1097-1105, 2012.

[14] D. Silver, A. Huang, C.J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, D. Hassabis, "Mastering the game of Go with deep neural networks and tree search," N*ature*, vol. 529, no. 7587, pp. 484-489, 2016.

[15] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. van den Driessche, T. Graepel, D. Hassabis, "Mastering the game of go without human knowledge," *Nature*, vol. 550, no. 7676, pp. 354-359, 2017.

[16] 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.

[17] M. Raissi, "Deep hidden physics models: Deep learning of nonlinear partial differential equations," The *Journal of Machine Learning Research*, vol. 19, no. 1, pp. 932-955, 2018.

[18] M. Raissi, P. Perdikaris, and G. E. 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.

[19] J. Tompson, K. Schlachter, P. Sprechmann, and K. Perlin, "Accelerating eulerian fluid simulation with convolutional networks," *in International Conference on Machine Learning*, pp. 3424-3433, 2017.

[20] X. Guo, W. Li, and F. Iorio, "Convolutional neural networks for steady flow approximation," *in Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pp. 481-490, 2016.

[21] O. Hennigh, “Lat-net: compressing lattice Boltzmann flow simulations using deep neural networks,” *arXiv Preprint*, arXiv:1705.09036, 2017.

[22] J. Ling, A. Kurzawski, and J. Templeton, "Reynolds averaged turbulence modelling using deep neural networks with embedded invariance," *Journal of Fluid Mechanics*, vol. 807, pp. 155-166, 2016.

[23] Z. J. Zhang and K. Duraisamy, "Machine learning methods for data-driven turbulence modeling," *in 22nd AIAA Computational Fluid Dynamics Conference*, p. 2460, 2015.

[24] A.D. Beck, D.G. Flad, C.-D. Munz, “Deep neural networks for data-driven turbulence models,” *arXiv Preprint*, arXiv:1806.04482, 2018.

[25] J. Takeuchi and Y. Kosugi, "Neural network representation of finite element method," *Neural Networks*, vol. 7, no. 2, pp. 389-395, 1994.

[26] L. Liang, M. Liu, C. Martin, and W. Sun, "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, p. 20170844, 2018.

[27] A. Chamekh, H. B. H. Salah, and R. Hambli, "Inverse technique identification of material parameters using finite element and neural network computation," *The International Journal of Advanced Manufacturing Technology*, vol. 44, no. 1-2, p. 173, 2009.

[28] Y. M. Hashash, S. Jung, and J. Ghaboussi, "Numerical implementation of a neural network based material model in finite element analysis," *International Journal for numerical methods in engineering*, vol. 59, no. 7, pp. 989-1005, 2004.

[29] D. Chen, D. I. Levin, S. Sueda, and W. Matusik, "Data-driven finite elements for geometry and material design," *ACM Transactions on Graphics (TOG)*, vol. 34, no. 4, p. 74, 2015.

[30] A. Oishi and G. Yagawa, "Computational mechanics enhanced by deep learning," *Computer Methods in Applied Mechanics and Engineering*, vol. 327, pp. 327-351, 2017.

[31] O.C. Zienkiewicz, R.L. Taylor, *The finite element method: the basis*, Butterworth-Heinemann, 2000.

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

[33] N. Le Roux and Y. Bengio, "Deep belief networks are compact universal approximators," *Neural computation*, vol. 22, no. 8, pp. 2192-2207, 2010.

[34] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, , M. Kudlur, J. Levenberg, R. Monga, S. Moore, D.G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, X. Zheng, "Tensorflow: A system for large-scale machine learning," *in 12th USENIX symposium on operating systems design and implementation*, pp. 265-283, 2016.

[35] D.P. Kingma, J. Ba, “Adam: a method for stochastic optimization,” *arXiv Preprint*, arXiv:1412.6980, 2014.

[36] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," *in Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pp. 249-256, 2010.

[37] D. Roylance, “Transformation of stresses and strains,” *Lecture Notes for Mechanics of Materials*, 2001.

[38] O.C. Zienkiewicz, R.L. Taylor, *The finite element method: basic formulation and linear problems*, McGraw-Hill, 1989.

[39] Y. Ko, P.-S. Lee, and K.-J. Bathe, "The MITC4+ shell element and its performance," *Computers & Structures*, vol. 169, pp. 57-68, 2016.

[40] C. Lee and P.-S. Lee, "The strain-smoothed MITC3+ shell finite element," *Computers & Structures*, vol. 223, p. 106096, 2019.

[41] R. L. Taylor, P. J. Beresford, and E. L. Wilson, "A non-conforming element for stress analysis," *International Journal for numerical methods in Engineering*, vol. 10, no. 6, pp. 1211-1219, 1976.

[42] P.C. Kohnke, *ANSYS Theory Reference: Release 5.5*, ANSYS, Inc., 1998.

[43] K.-J. Bathe and P.-S. Lee, "Measuring the convergence behavior of shell analysis schemes," *Computers & structures*, vol. 89, no. 3-4, pp. 285-301, 2011.

[44] P.-S. Lee and K.-J. Bathe, "The quadratic MITC plate and MITC shell elements in plate bending," *Advances in Engineering Software*, vol. 41, no. 5, pp. 712-728, 2010.

[45] Y. Ko, Y. Lee, P.-S. Lee, and K.-J. Bathe, "Performance of the MITC3+ and MITC4+ shell elements in widely-used benchmark problems," *Computers & Structures*, vol. 193, pp. 187-206, 2017.

[46] R.D. Cook, D.S. Malkus, M.E. Plesha, R.J. Witt, *Concepts and applications of finite element analysis*, John Wiley & Sons, 2007.

[47] S.V. Walt, S.C. Colbert, G. Varoquaux, "The NumPy array: a structure for efficient numerical computation," *Computing in science & engineering*, vol. 13, no. 2, pp. 22-30, 2011.

[48] K. Yoon and P.-S. Lee, "Nonlinear performance of continuum mechanics based beam elements focusing on large twisting behaviors," *Computer Methods in Applied Mechanics and Engineering,* vol. 281, pp. 106-130, 2014.

[49] Y. Lee, P.-S. Lee, and K.-J. Bathe, "The MITC3+ shell element and its performance," *Computers & Structures*, vol. 138, pp. 12-23, 2014.

[50] Y. Ko, P.-S. Lee, and K.-J. Bathe, "A new 4-node MITC element for analysis of two-dimensional solids and its formulation in a shell element," *Computers & Structures*, vol. 192, pp. 34-49, 2017.

[51] S. Kim and P.-S. Lee, "New enriched 3D solid finite elements: 8-node hexahedral, 6-node prismatic, and 5-node pyramidal elements," *Computers & Structures*, vol. 216, pp. 40-63, 2019.

[52] H.-M. Jeon, Y. Lee, P.-S. Lee, and K.-J. Bathe, "The MITC3+ shell element in geometric nonlinear analysis," *Computers & Structures*, vol. 146, pp. 91-104, 2015.

[53] D.-N. Kim and K.-J. Bathe, "A triangular six-node shell element," *Computers & Structures*, vol. 87, no. 23-24, pp. 1451-1460, 2009.

[54] M. L. Bucalem and K. J. Bathe, "Higher-order MITC general shell elements," *International Journal for Numerical Methods in Engineering*, vol. 36, no. 21, pp. 3729-3754, 1993.

[55] Y. Yu, T. Hur, J. Jung, I. G. J. S. Jang, and M. Optimization, "Deep learning for determining a near-optimal topological design without any iteration," *Structural and Multidisciplinary Optimization*, vol. 59, no. 3, pp. 787-799, 2019.

[56] R. H. Macneal and R. L. Harder, "A proposed standard set of problems to test finite element accuracy," *Finite elements in analysis and design*, vol. 1, no. 1, pp. 3-20, 1985.

[57] T. H. Pian and K. Sumihara, "Rational approach for assumed stress finite elements," *International Journal for Numerical Methods in Engineering*, vol. 20, no. 9, pp. 1685-1695, 1984

[58] S. Cen, T. Zhang, C.-F. Li, X.-R. Fu, and Y.-Q. Long, "A hybrid-stress element based on Hamilton principle," *Acta Mechanica Solida Sinica*, vol. 26, no. 4, pp. 625-634, 2010

[59] U. Andelfinger and E. Ramm, "EAS-elements for two-dimensional, three-dimensional, plate and shell structures and their equivalence to HR-elements," *International journal for numerical methods in engineering*, vol. 36, no. 8, pp. 1311-1337, 1993.

[60] S. Cen, X.-M. Chen, and X.-R. Fu, "Quadrilateral membrane element family formulated by the quadrilateral area coordinate method," *Computer Methods in Applied Mechanics and Engineering*, vol. 196, no. 41-44, pp. 4337-4353, 2007.

[61] K. Sze, "On immunizing five-beta hybrid-stress element models from 'trapezoidal locking'in practical analyses," *International Journal for Numerical Methods in Engineering*, vol. 47, no. 4, pp. 907-920, 2000.

[62] D. Systemes, *Abaqus 6.9 Documentation,* Dassault Systemes Simulia Corp, Providence, RI, USA, 2009.

[63] R. Piltner and R. Taylor, "A systematic construction of B-bar functions for linear and non-linear mixed-enhanced finite elements for plane elasticity problems," *International Journal for Numerical Methods in Engineering*, vol. 44, no. 5, pp. 615-639, 1999.

[64] R. Piltner and R. Taylor, "A quadrilateral mixed finite element with two enhanced strain modes," *International Journal for Numerical Methods in Engineering*, vol. 38, no. 11, pp. 1783-1808, 1995.

[65] Z.-F. Long, S. Cen, L. Wang, X.-R. Fu, and Y.-Q. Long, "The third form of the quadrilateral area coordinate method (QACM-III): theory, application, and scheme of composite coordinate interpolation," *Finite Elements in Analysis and Design*, vol. 46, no. 10, pp. 805-818, 2010.

[66] S. Rajendran and B. Zhang, "A "FE-meshfree" QUAD4 element based on partition of unity," *Computer Methods in Applied Mechanics and Engineering*, vol. 197, no. 1-4, pp. 128-147, 2007.

[67] J. Xu and S. Rajendran, "A partition-of-unity based 'FE-Meshfree'QUAD4 element with radial-polynomial basis functions for static analyses," *Computer methods in applied mechanics and engineering*, vol. 200, no. 47-48, pp. 3309-3323, 2011.

[68] S. Cen, M.-J. Zhou, and X.-R. Fu, "A 4-node hybrid stress-function (HS-F) plane element with drilling degrees of freedom less sensitive to severe mesh distortions," *Computers & Structures*, vol. 89, no. 5-6, pp. 517-528, 2011.

[69] T. H. Pian and C.-C. Wu, *Hybrid and incompatible finite element methods*. CRC press, 2005.

[70] C.-C. Wu, M.-G. Huang, and T. H. PIAN, "Consistency condition and convergence criteria of incompatible elements: general formulation of incompatible functions and its application," *Computers & structures*, vol. 27, no. 5, pp. 639-644, 1987.

[71] Y.-Q. Long, S. Cen, and Z.-F. Long, Advanced finite element method in structural engineering. Springer, 2009.

[72] C. Wanji and T. Limin, "Tsoparametric Quasi-Conforming Element [J]," *Journal of Dalian University of Technology*, vol. 1, 1981.

[73] F. Gao and L. Han, "Implementing the Nelder-Mead simplex algorithm with adaptive parameters," *Computational Optimization and Applications*, vol. 51, no. 1, pp. 259-277, 2012.

[74] S. Cen, P. L. Zhou, C. F. Li, and C. J. Wu, "An unsymmetric 4-node, 8-DOF plane membrane element perfectly breaking through MacNeal's theorem," *International Journal for Numerical Methods in Engineering*, vol. 103, no. 7, pp. 469-500, 2015.

[75] X.-M. Chen, S. Cen, J.-Y. Sun, and Y.-G. Li, "Four-node generalized conforming membrane elements with drilling DOFs using quadrilateral area coordinate methods," *Mathematical Problems in Engineering*, vol. 2015, 2015.

[76] E. Wilson, R. Taylor, W. Doherty, and J. Ghaboussi, "Incompatible displacement models," *Numerical and computer methods in structural mechanics*: Elsevier, 1973, pp. 43-57.

[77] Y. Shang and W. Ouyang, "4-node unsymmetric quadrilateral membrane element with drilling DOFs insensitive to severe mesh-distortion," *International Journal for Numerical Methods in Engineering*, vol. 113, no. 10, pp. 1589-1606, 2018.

[78] P. J. Werbos, "Backpropagation through time: what it does and how to do it," *Proceedings of the IEEE*, vol. 78, no. 10, pp. 1550-1560, 1990.

[79] P. Werbos, "Beyond regression:" new tools for prediction and analysis in the behavioral sciences," *Ph. D. dissertation, Harvard University*, 1974.

[80] I. Babuska and B. Szabo, "On the rates of convergence of the finite element method," *International Journal for Numerical Methods in Engineering*, vol. 18, no. 3, pp. 323-341, 1982.

[81] K. Fukushima, "Neural network model for a mechanism of pattern recognition unaffected by shift in position-Neocognitron," *IEICE Technical Report, A*, vol. 62, no. 10, pp. 658-665, 1979.

[82] K. Fukushima, "A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position," *Biological Cybernetics.*, vol. 36, pp. 193-202, 1980.

[83] Y. LeCun et al., "Backpropagation applied to handwritten zip code recognition," *Neural computation*, vol. 1, no. 4, pp. 541-551, 1989.

[84] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278-2324, 1998.

[85] M. Lukoševičius and H. Jaeger, "Reservoir computing approaches to recurrent neural network training," *Computer Science Review*, vol. 3, no. 3, pp. 127-149, 2009.

[86] T. Mikolov, S. Kombrink, L. Burget, J. Černocký, and S. Khudanpur, "Extensions of recurrent neural network language model," *in 2011 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, 2011: IEEE, pp. 5528-5531.

[87] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," *arXiv preprint arXiv:1312.6114*, 2013.

[88] I. Goodfellow et al., "Generative adversarial nets," *in Advances in neural information processing systems*, 2014, pp. 2672-2680.

[89] M. Mirza and S. Osindero, "Conditional generative adversarial nets," *arXiv preprint arXiv:1411.1784*, 2014.

[90] L. Yu, W. Zhang, J. Wang, and Y. Yu, "Seqgan: Sequence generative adversarial nets with policy gradient," *in Thirty-first AAAI conference on artificial intelligence*, 2017.

[91] X. Chen, Y. Duan, R. Houthooft, J. Schulman, I. Sutskever, and P. Abbeel, "Infogan: Interpretable representation learning by information maximizing generative adversarial nets," *in Advances in neural information processing systems*, 2016, pp. 2172-2180.

[92] L. Chongxuan, T. Xu, J. Zhu, and B. Zhang, "Triple generative adversarial nets," *in Advances in neural information processing systems*, 2017, pp. 4088-4098..

[93] T. Zhang, A. Wiliem, S. Yang, and B. Lovell, "Tv-gan: Generative adversarial network based thermal to visible face recognition," *in 2018 international conference on biometrics (ICB), 2018: IEEE*, pp. 174-181.

[94] H. Wang et al., "Graphgan: Graph representation learning with generative adversarial nets," *arXiv preprint arXiv:1711.08267*, 2017.