UNIVERSITY
OF OSLO

Master's thesis

Deep Energy Method for Heart Mechanics

Mansur Dudajev

Digital Signal Processing

60 ECTS study points

Department of Physics

Faculty of Mathematics and Natural Sciences

Autumn 2024

Mansur Dudajev

Deep Energy Method for Heart Mechanics

Supervisors:

Henrik Nicolay Finsberg

Joakim Sundnes

Johannes Haubner

Xing Cai

Acknowledgements

First and foremost, I would like to thank my supervisors, Henrik Finsberg and Joakim Sundnes, for their invaluable feedback and guidance during this project. Also, for finding the time to meet with me every week and help me with every problem I had created for myself.

Furthermore, I would like to thank Xing Cai and Johannes Haubner for reading my thesis with a fresh set of eyes and providing me with feedback on my structure and design and much needed constructive critique of my mathematical incompetency.

I would also like to thank the Simula Research Laboratory for providing me with the opportunity to do this project and a wonderful work environment with free cake every once in a while.

And last but not least, I would like to thank my parents, Hasan and Hava, for supporting me during this project and listening to my long-winded explanations of every problem I have been stuck at.

Mansur Dudajev
Oslo, Sep 2024

Abstract

Cardiac tissue is usually modeled as an incompressible, hyperelastic material. For such materials, the equations of motion are based on constitutive laws and are solved using the finite element method. The application of the finite element method is based on a variational formulation of the problem, which can also be formulated as a minimization problem, where the goal is to minimize the potential energy stored in the system. Recently, a method known as the deep energy method has been proposed as an alternative to solve problems that can be cast in an energy minimization framework. The idea behind this method is to use the energy in the system as the loss function of a neural network and apply deep learning techniques to minimize this loss function, and thereby solve the underlying partial differential equation. In this thesis, we aim to apply the deep energy method to benchmark problems for the verification of cardiac mechanics software. This benchmark contains three problems that test different features that are important in cardiac modeling. To verify our finite element implementation, we start by solving a one-dimensional problem for which we have an analytical solution. After this, we solve two three-dimensional problems, where we test the performance of different neural network models and decide which models to use on the benchmark problems. Finally, we use models with the best performance on the problems in the benchmark.

Our results show that the deep energy method performs comparably to the finite element method on simple geometries with rough meshes, but fails at approximating the solution for fine meshes or the ellipsoid geometry used to model the left ventricle. Moreover, the finite element method converges faster on the solution than the deep energy method.

CONTENTS

CHAPTER 1

INTRODUCTION

1.1 Motivation

1.1.1 Why Model the Heart?

Cardiovascular diseases account for a third of all deaths globally and are the leading cause of death worldwide. Responsible for over 20 million deaths in 2021, cardiovascular diseases still affect more than half a billion people around the world [10]. Although advancements in medicine have reduced mortality, our knowledge of the mechanisms causing these diseases is still poor [11]. These mechanisms are modeled with mathematical models of the cardiovascular system. The increase in computational power and the development of better numerical algorithms are among the reasons why applications of mathematical models of the cardiovascular system have become common in the bioengineering and medical research community in the last decades [12].

Recent advances in applications of neural networks have sparked the scientific community's interest, and their use has also increased in this community. One such use of neural networks, developed by Nguyen-Thanh et al [41], is called the deep energy method. This method uses neural networks to solve partial differential equations in nonlinear solid mechanics problems.

1.1.2 Benchmarking

Standard tools for comparing the performance of different systems are called benchmarks [57]. In this thesis, we measure performance by how well the deep energy method approximates the mathematical model we are solving for. This measure of performance is known as verification. Verification has generally had a limited role in cardiac modeling and the analytic solutions used to test solid mechanics software do not test several important aspects of cardiac mechanics [30]. To address this problem, Land et al. [30] have written three benchmark problems focusing on features specific to cardiac mechanics problems. Section 3.2 introduces these problems. This thesis aims to use the deep energy method to solve benchmarking problems for cardiac applications and test its performance against the finite element method.

1.2 The Cardiovascular System

This section will introduce the cardiovascular system, the heart's anatomy, and the cardiac cycle. By introducing the properties of cardiac tissue, we will motivate the material models used in this thesis. The theory in this section is based on the thesis by Henrik Finsberg [11] and the books by Hoskins et al [23] and Arnold M. Katz [26]. The reader is referred to the books for more details.

1.2.1 Anatomy of the Heart

The heart is an organ whose main function is circulating blood throughout the body via the cardiovascular system. The heart makes up the cardiovascular system, together with blood and blood vessels. Figure 1.1 illustrates the blood flow in a simplified cardiovascular system. Functions of the cardiovascular system include thermoregulating, transporting molecules, and maintaining fluid between different tissues. In addition, this system is responsible for providing oxygen to organs and tissues and removing carbon dioxide, a waste product of metabolism. The cardiovascular system consists of two sub-systems: the pulmonary circulation and the systemic circulation. Pulmonary circulation is the process where the right side of the heart pumps deoxygenated blood into the lungs, wherein carbon dioxide is removed and oxygen is supplied. Systemic circulation is the process where the left side of the heart pumps oxygenated blood to the rest of the body and deoxygenated

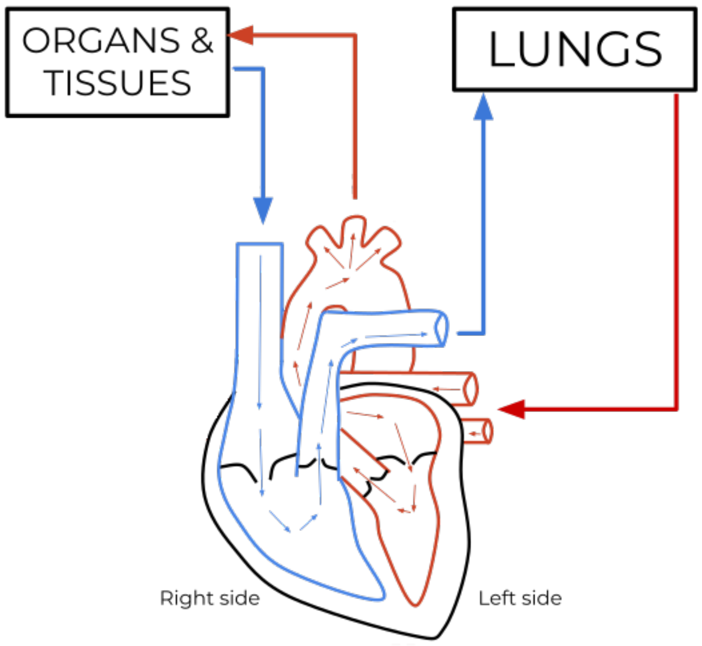


Figure 1.1: Blood flow in the cardiovascular system. **Blue:** deoxygenated blood; **Red:** oxygenated blood. Adapted from Wikipedia according to CC BY-SA 4.0 License; diagram authored by Sasha River Santilla; source: https://commons.wikimedia.org/wiki/File:Blood_oxygenation_to_the_pulmonary_and_systemic_circulation.svg.

blood returns to the right side of the heart via the veins.

Put another way, the human heart works as two pumps, one pumping blood to the lungs and one pumping blood into the body. Figure 1.2 shows a sectioned view of the heart with the veins and arteries. The heart is situated close to the center of the chest with its inferior end pointing down to the left. Surrounding the heart is the pericardium, a sac of fibrous tissue. The heart wall is called the *myocardium* and consists of muscle cells, called myocytes, which are connected to neighboring cells in a complicated fibrous network. The outer layer of the myocardium is called the epicardium and its inner layer is called the endocardium.

Although experiments have shown that the myocardium under dynamic loading produces viscoelastic response [9, 52], it is still common to model the myocardium as a nonlinearly elastic [22], which we also chose to do in this thesis. Moreover, the myocardium is modeled as incompressible [9, 22], meaning its volume does not change. The muscle fiber orientation in the myocardium changes smoothly from -60^o at the epicardium to $+60^o$ at the endocardium. Figure 1.3 shows the fiber orientations on the epicardium and the endocardium and Figure 1.4 shows how the fiber directions change throughout the myocardium.

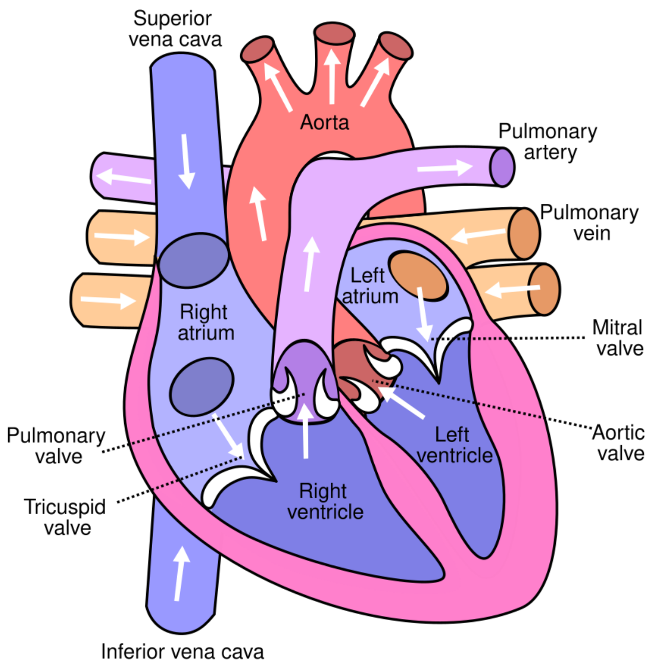


Figure 1.2: The heart, the veins, and the arteries. The normal flow of blood is illustrated with the white arrows. Reprinted from Wikipedia according to CC BY-SA 4.0 License; diagram authored by Wapcaplet and Yaddah; source: https://commons.wikimedia.org/wiki/File:Diagram_of_the_human_heart.svg.

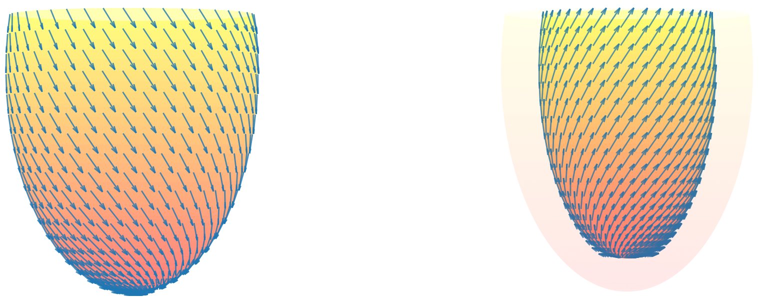


Figure 1.3: Fiber directions on the left ventricle. The fiber direction is -60^o at the epicardium (left) and $+60^o$ at the endocardium (right).

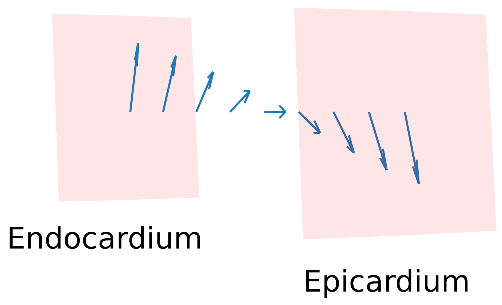


Figure 1.4: Change in fiber direction within the myocardium. The fiber direction is -60^o at the epicardium and $+60^o$ at the endocardium.

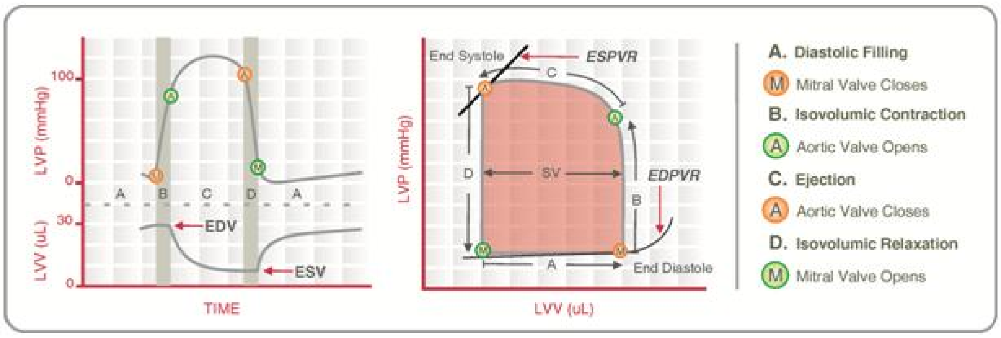


Figure 1.5: Left ventricular pressure and volume during the cardiac cycle. **Left:** pressure (top) and volume (bottom) as functions of time; **Middle:** PV-loop with volume in the LV along the x-axis and pressure in the LV along the y-axis. Adapted from Wikipedia according to CC BY-SA 3.0 License; diagram authored by Andyhenton83; source: https://commons.wikimedia.org/wiki/File:Cardiac_Pressure_Volume_Loop.jpg.

1.2.2 The Cardiac Cycle

The heartbeat, or the cardiac cycle, can be divided into two main parts: *systole* and *diastole*. The contraction of the ventricles characterizes systole and occurs during the *isovolumeric contraction* and *ejection*. Diastole occurs when the ventricles are relaxed during *isovolumeric relaxation* and *filling*.

The cardiac cycle starts with the spontaneous depolarization of cells in the sinus node in the right atrium (RA), which serves as a pacemaker. The depolarization starts an electrical signal that stimulates the different chambers in the heart causing them to contract. For the blood to flow from the atria to the ventricles, the contractions must occur only when a chamber is filled with blood, so the contraction increases the pressure in the chamber. The electrical signal propagates through the RA and left atrium (LA) and further into a region called the atrioventricular node, which serves as the only path for the electrical signal between the atria and the ventricles. The atrioventricular node also slows the electrical signal, so the ventricles contract only after the atria have relaxed.

A common way to study the different phases of the cardiac cycle is to measure the pressure and the volume inside the chambers. The two most common ways of visualizing the change in volume and pressure are to plot the change in each property over time or to plot the volume during the cycle on the x-axis and the pressure on the y-axis. The latter results in a pressure-volume (PV) loop. Figure 1.5 shows both visualizations of the cardiac cycle.

Starting at end-diastole (ED), the tricuspid and mitral valves close, and the right ventricle (RV) and left ventricle (LV) contract, causing a rise in pressure. This is known as isovolumic contraction, as no blood leaves the ventricles in this phase. When the pressure in the ventricles exceeds the pressure in the arteries (pulmonary artery for the RV and the aorta for the LV), the pulmonary and aortic valves open and the blood is ejected from the ventricles. This is the ejection phase and the end of ejection is called end-systole (ES). As the ventricular pressure drops below the pressure in the arteries, the pulmonary and the aortic valves close again, and the ventricles relax during the isovolumeric relaxation phase. After the ventricles are fully relaxed, the tricuspid and the mitral valves open again, and the cycle restarts.

1.3 Data-Driven Models vs Physics-Based Models

In engineering and natural sciences, we express physical models through constitutive relationships governing material behavior, spatial and temporal dependencies, and dynamical systems [39]. One of the biggest challenges in science and engineering is extracting governing equations from data. An example of this is the discoveries made by Kepler and Newton. Kepler used the most accurate data on planetary motion in his day to develop his theory of elliptic orbits. On the other hand, Newton understood the relationship between momentum and energy and used this to describe the underlying mechanisms that cause the elliptic orbits described by Kepler. Newton's model may be generalized to systems we have no data on and used to predict their behavior. It has also proven invaluable in engineering, allowing humanity to reach the moon, which Kepler's model alone could not [4].

1.3.1 Data-Driven Models

The last decades have seen a rapid increase in the computational power of computers due to technological advances in computer hardware. Combined with lower prices for sensors and faster and larger data storage capabilities [39], this has led to a new era in data acquisition. Terms such as *data science* and *data scientist* [7] are becoming commonplace in many fields, while the field of *data analytics* has developed into *advanced analytics*, with new theories, tools, and technologies designed to handle *big data* [7], data too large for processing by traditional methods and technologies [47]. Data-intensive science has even been called the fourth paradigm of science [20].

The complexity and volume of the available data make it challenging for humans to collect any useful information from it [36]. This challenge has motivated the development and use of data-driven methods [39], including clustering, decision trees, ensemble methods, and neural networks [37].

Neural networks (NNs), also called artificial neural networks, are at the front of data-driven methods [39]. NNs are a subfield of *machine learning*, a field that deals with algorithms that can learn [15]. In this context, learning means that the algorithms can iteratively modify their parameters so they better produce desired outcomes [40]. Modifying the parameters of a NN is called *training* the NN.

The two main methods for training NNs are unsupervised learning and supervised learning. In unsupervised learning, the goal of the NN can be thought of as finding patterns within the data [14, 37]. For example, clustering is the task of dividing a dataset into clusters that share some property [15]. In supervised learning, any input data to the NN has a corresponding target and the NN aims to approximate a function that relates the input to the target. First, the input data propagates through the network, yielding an output. After this, an error between the output and the target is calculated to assess the performance of the NN. Therefore, every input has to be paired with its target. This is called *labeling* the data. The lack of sufficient labeled data is one of the largest obstacles when training NNs [37] in a supervised fashion.

In recent years, several applications of NNs have become an integral part of everyday life. NNs are used in technologies such as voice and face recognition, image generation, self-driving cars, and virtual assistants. In addition to achieving near-human performance in recognizing handwritten digits, NNs beat world champions in Go [8] and chess [6] and score higher than humans in divergent thinking, considered an indication of potential for creativity [24]. Three widely used types of NNs are the feed-forward neural network, the convolutional neural network, and the recurrent neural network.

The simplest type of NN is the *feed-forward neural network* (FNN). An FNN is a collection of units, called nodes, arranged in several layers, all connected to nodes in the previous and following layers. In an FNN, data propagates through the nodes from the input to the output. The nodes are inspired by models of neurons in the brain, using the neuron model introduced by Warren McCulloch and Walter Pitts in 1943 [36]. When data propagates from one node to a node in the next layer, the output from the former node is multiplied with a weight and added to the weighted sum of all other nodes in the same layer. This weighed sum is then added to a bias term, and sent to the receiving node. These *weights* and *biases* are the parameters we seek to modify during the training process. The training process will be expanded upon in section 2.3. Since all nodes in any layer are connected to all nodes in the previous and following layers, we end up with thousands of parameters to optimize for even shallow FNNs.

One solution to the vast number of parameters is to use a fixed number of connections between any two layers. This is the idea behind *convolutional neural networks*. Here, a filter is used between the layers, and data in a layer is convolved with the filter corresponding to that layer as it propagates through the network. This way, the number of parameters between any two layers is independent of the number of nodes in the layers. Convolutional neural networks are common in image analysis and computer vision [33].

A second solution is to add a backward connection to the neurons between the input and the output. Adding this connection results in a *recurrent neural network*. Recurrent neural networks typically only have a few layers between the input and the output. The backward connection serves as a "memory" within the network. In contrast to FNNs and convolutional neural networks where data flows from input to output, the data flow in a recurrent neural network is bi-directional. Recurrent neural networks perform well on speech and text recognition and translation [55].

In addition to these, there are many models with more complex architectures, including *generative adversarial networks* [16], *variational autoencoders* [27], and transformers [56]. Many of the more advanced models are *generative* models, meaning they can generate data such as text, images, or music. We will not go into further detail about these models, as they are beyond the scope of this thesis.

1.3.2 Physics-Based Models

In physics-based models, the laws governing physical phenomena are represented by equations. Some examples are Newton's laws of motion, Maxwell's equations, and conservation laws. These models predict a system's behavior given a set of parameters. They are used in many disciplines including engineering and natural sciences, and their governing equations are proven mathematically or derived from measurements in experiments. Although the models represent physical laws, they are often simplifications. Even the simplest problems become much harder, and often impossible, to solve analytically when friction or drag is introduced into the model. In these cases, numerical methods are commonly used to solve problems.

Numerical methods are used in all mathematical computations performed on computers. In simulation software, for example, numerical methods are the main behind-the-scenes component, solving the equations in the model that is being simulated. Moreover, numerical methods are used to solve problems that can not be solved analytically.

One advantage of physics-based models over data-driven models is that they make it straightforward to implement physical phenomena computationally to run computer simulations. On the other hand, they require large computational power for complicated systems, and many numerical algorithms are very time-consuming.

1.3.3 Hybrid Models

One proposed workaround for both the lack of data for NNs and the complexity of numerical computations is to model systems using hybrid models. These are data-driven models that can solve problems by using the underlying physical laws in addition to the data. In this thesis, we will cover two hybrid models: physics-informed neural networks and the deep energy method.

The idea behind *physics-informed neural networks* (PINNs) is to constrain the NN output using the partial differential equations (PDEs) which express the physical laws governing the problem [29]. With this, the NN performance can be evaluated on the differential equations in addition to the target data. This results in two error criteria, one for the data and one for the PDE. PINNs were introduced by Raissi et al. [48] and they showed promising results for equations such as the Schrödinger equation and the Navier-Stokes equation. Since their publication, PINNs have gained much traction and have been used to model Darcy flow [59] and heat transfer [5]. Variations of PINNs have been used to solve forward and inverse PDE problems [58] and time-dependent PDEs [38]. In cardiac applications, Kissas et al. have used PINNs to model cardiovascular flow [28] and Herrero Martin et al. [19] used PINNs for modeling of cardiac electrophysiological tissue properties for the characterization of cardiac arrhythmias.

In contrast to PINNs, the *deep energy method* (DEM) does not use the underlying PDEs or any target. The DEM utilizes a principle in physics called the principle of stationary potential energy, which states that a system is at equilibrium when its potential energy is at a minimum. In the DEM, the potential energy of the system is calculated from the NN output. This energy is then differentiated, and the derivative is used to find its minimum. Using this principle, the NN never has to access the PDE or its true solution. In addition, the Neumann boundary conditions are included in the potential energy, and the Dirichlet boundary conditions are enforced on the NN output [41] using appropriate functions. The problem is therefore unconstrained in the DEM. Section 2.6 provides a detailed introduction to the DEM.

There are several advantages to both of these models. One advantage of PINNs over the DEM is that the DEM can only be used on systems that fulfill the principle of stationary potential energy, such as engineering problems under conservative loads [29]. PINNs can be used to solve any problem that can be formulated as a system of PDEs. Moreover, PINNs have a built-in regularization that allows the network to avoid overfitting [48]. The DEM lacks any regularization, which leads to non-physical external energy [29]. On the other hand, while PINNs require labeled data to calculate the error on the data, the DEM can calculate the energy using only the NN solution. The DEM also has the advantage that it only requires first-order differentiation even for second-order PDEs [13], because the problem is cast in an energy-minimization framework.

The scripts used to generate the results presented in the Results chapter in this thesis are available at https://github.com/mansoryashka/Masters_Thesis.

The rest of this thesis is organized in the following way. In Chapter 2, we present the mathematical models used in this thesis. In addition, we will introduce neural networks and the hybrid models from Chapter 1. and show how the finite element method is used to solve partial differential equations. Chapter 3 will introduce the problems solved in this thesis and explain the challenges we met. Results from the experiments will be presented in Chapter 4, followed by a discussion of the results in Chapter 5, before a conclusion is presented in Chapter 6.

During the writing of this thesis, the author used the ChatGPT3.5 large language model to enhance the language and gather inspiration. Following this, the author reviewed and edited the text to ensure it was accurate and reflected the author's knowledge.

CHAPTER 2

METHODS AND MODELS

2.1 Mathematical Models

This section will cover the physical models of continuum mechanics used in this thesis. To begin with, we will introduce the nonlinear solid mechanics of a continuum body in three dimensions before we extend the theory to cover the concepts of hyperelasticity and compressibility, followed by the penalty method for incompressibility. The following theory is based on the thesis by Henrik Finsberg [11] and the book by Gerhard Holzapfel [21].

2.1.1 Nonlinear Solid Mechanics

A body in which both mass and volume are continuous functions of material points is called a *continuum body* [21]. The original configuration of a body is called the *reference configuration*, while the configuration after some deformation is called its *current configuration*. Figure 2.1 illustrates the deformation of a sphere. In the reference configuration, the coordinates are denoted by $\mathbf{X}$; in the current configuration, the coordinates are denoted by $\mathbf{x}$. In addition, the reference and current configuration are denoted by Ω and ω. When a body is deformed, a particle originally at $\mathbf{X}$ in Ω is represented by $\mathbf{x}$ in ω and the coordinates $\mathbf{x}$ and $\mathbf{X}$ are related through the smooth and invertible map $\boldsymbol{\varphi} : \Omega \to \omega$ by $\mathbf{x} = \boldsymbol{\varphi}(\mathbf{X})$. Another quantity that relates positions in the current configuration to positions

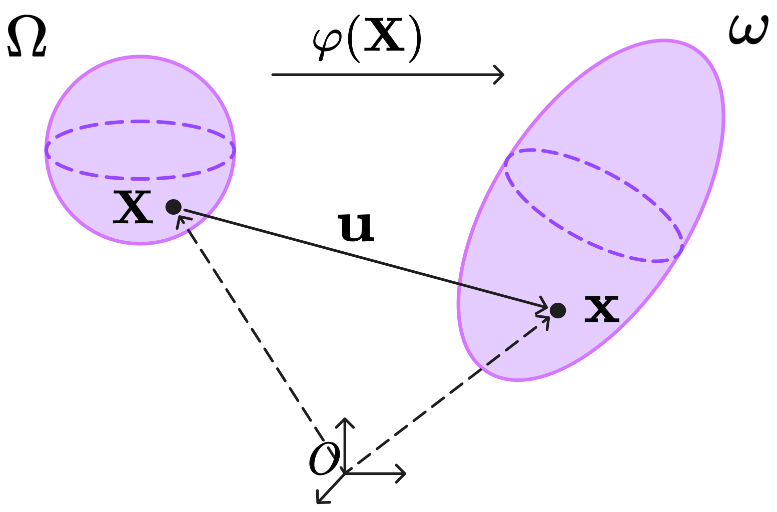


Figure 2.1: Deformation of a continuum body in $\mathbb{R}^3$. Coordinates $\mathbf{X}$, in the reference configuration, Ω, are mapped to coordinates $\mathbf{x}$, in the current configuration, ω, through a map $\boldsymbol{\varphi}$. The displacement field of the coordinates is denoted $\mathbf{u}$.

in the reference configuration is the *displacement field* ($\mathbf{u}$)

$$\mathbf{u} = \mathbf{x} - \mathbf{X}. \tag{2.1}$$

The *deformation gradient* ($\mathbf{F}$) is a rank-2 tensor that maps vectors in the reference configuration to vectors in the current configuration

$$\mathbf{F} = \nabla \mathbf{x} = \nabla \mathbf{u} + \nabla \mathbf{X} = \nabla \mathbf{u} + \mathbf{I}, \tag{2.2}$$

where ∇ is the derivative with respect to the reference coordinates. The *deformation gradient* is also used to map an infinitesimal volume element in the reference configuration, dV, to an infinitesimal volume element in the current configuration, dv,

$$dv = J\, dV, \tag{2.3}$$

where $J = \det(\mathbf{F})$ is the factor by which a volume scales for a given deformation.

The principle of conservation of linear momentum states that any change in the linear momentum of the body equals the net impulse acting on it. For a continuum body with mass density ρ, this implies that

$$\int_\omega \rho\dot{\mathbf{v}}\,dv = \mathbf{f}, \tag{2.4}$$

where

$$\mathbf{f} = \int_{\partial\omega} \mathbf{t}\,ds + \int_\omega \mathbf{b}\,dv.$$

Here, $\dot{\mathbf{v}}$ is the time derivative of the spatial velocity field, $\mathbf{t}$ is the traction acting on the boundary, $\partial\omega$, and $\mathbf{b}$ is the body force. According to *Cauchy's stress theorem*, there exists a second-order stress tensor, $\boldsymbol{\sigma}$, that fulfills the relation $\mathbf{t} = \boldsymbol{\sigma}\mathbf{n}$, where $\mathbf{n}$ is the unit normal vector in the current configuration. Using the divergence theorem we get the relation

$$\int_{\partial\omega} \mathbf{t}\,ds = \int_{\partial\omega} \boldsymbol{\sigma}\mathbf{n}\,ds = \int_\omega \nabla\cdot\boldsymbol{\sigma}\,dv. \tag{2.5}$$

By using (2.5) in (2.4) we get *Cauchy's momentum equation*

$$\nabla\cdot\boldsymbol{\sigma} + \mathbf{b} = \rho\dot{\mathbf{v}}. \tag{2.6}$$

In this thesis, we chose to neglect the effect of the inertial term $\rho\dot{\mathbf{v}}$, which gave us the force-balance equation

$$-\nabla\cdot\boldsymbol{\sigma} = \mathbf{b}. \tag{2.7}$$

Here the *Cauchy stress tensor*, $\boldsymbol{\sigma}$, defines the stresses in the current configuration. An equivalent formulation in the reference configuration becomes

$$-\nabla\cdot\mathbf{P} = \mathbf{B}, \tag{2.8}$$

where $\mathbf{P}$ is the *first Piola-Kirchhoff stress tensor*. $\mathbf{P}$ is related to $\boldsymbol{\sigma}$ through the relation

$$\mathbf{P} = J\boldsymbol{\sigma}\boldsymbol{F}^{-T}. \tag{2.9}$$

The relation (2.8) relates the stresses in a continuum body to the forces acting on it. This relation is called a partial differential equation in its strong form and our goal is to find the deformation that satisfies this equation. To achieve this, we must first express $\mathbf{P}$ as a function of the displacement using an appropriate material model.

Although the equations introduced in this section are true for any continuum and are used to characterize kinematics and stresses, they do not differentiate between different types of materials [21]. Since the focus of this thesis is hyperelastic materials we will relate the equations above to the material response of this type of solids.

2.1.2 Hyperelasticity

The hyperelastic nature of a material implies that there exists a *strain energy function* $\boldsymbol{\Psi}$, such that

$$\mathbf{P} = \frac{\partial \boldsymbol{\Psi}(\mathbf{F})}{\partial \mathbf{F}}. \tag{2.10}$$

The strain energy function is a measure of how much energy is stored in the material when subjected to strain. Given that the stress, $\mathbf{P}$, has unit Pa, the strain energy function has unit $\frac{\text{Joule}}{m^3}$.

Certain requirements must be satisfied by the strain energy function. One requirement is that the reference state is stress-free and that the energy stored increases monotonically as a function of deformation. Stated mathematically this is

$$\begin{aligned}\boldsymbol{\Psi}(\mathbf{I}) &= 0,\\ \boldsymbol{\Psi}(\mathbf{F}) &\geq 0.\end{aligned}$$

Another requirement is that it should require infinite energy to compress the body to zero or expand it to infinity, which sets the constraints on the strain energy function

$$\begin{aligned}&\boldsymbol{\Psi}(\mathbf{F}) \to \infty, &&\text{if} &&J \to 0,\\ &\boldsymbol{\Psi}(\mathbf{F}) \to \infty, &&\text{if} &&J \to \infty.\end{aligned}$$

Because J represents the scale factor of the volume change for a given deformation, $J = 0$ corresponds to compressing the body to a single point, and $J = \infty$ corresponds to expanding the body to infinity. Furthermore, incompressibility can be expressed by constraining J such that

$$J = 1. \tag{2.11}$$

To implement incompressibility, we will use the model for a compressible material and a mathematical workaround called the penalty method for incompressibility.

2.1.3 Compressibility

For a compressible material, it is common to decompose the deformation gradient, $\mathbf{F}$, into two factors

$$\mathbf{F} = \mathbf{F}_{\text{vol}}\bar{\mathbf{F}}, \tag{2.12}$$

where $\mathbf{F}_{\text{vol}}$ represents the *volumetric* (volume-changing) deformations and is defined as

$$\mathbf{F_{vol}} = J^{1/3}\mathbf{I} = \begin{bmatrix} J^{1/3} & 0 & 0 \\ 0 & J^{1/3} & 0 \\ 0 & 0 & J^{1/3} \end{bmatrix},$$

$\bar{\mathbf{F}}$ represents the *isochoric* (volume-preserving) deformations [21]. The factor $J^{1/3}$ ensures that the volumetric deformations represent any change in volume, because

$$\det(\mathbf{F}_{\text{vol}}) = J^{1/3} \cdot J^{1/3} \cdot J^{1/3} = J,$$

since $\det\left(\bar{\mathbf{F}}\right) = 1$. From this decomposition, we can derive a decoupled representation of the strain energy function

$$\mathbf{\Psi}(\mathbf{F}) = \mathbf{\Psi}_{\text{vol}}(J) + \mathbf{\Psi}_{\text{iso}}(\bar{\mathbf{F}}), \tag{2.13}$$

where $\mathbf{\Psi}_{\text{vol}}$ and $\mathbf{\Psi}_{\text{iso}}$ are scalar-valued functions representing the volumetric and isochoric elastic response of the material [21].

2.1.4 Penalty Method for Incompressibility

A common approach when modeling incompressible materials is to model them as *nearly incompressible* using the so-called *penalty method* [21]. The goal of the penalty method is to penalize any change in the volume of the body. When using the penalty method, we approximate the volumetric term $\mathbf{\Psi}_{\text{vol}}$ as a function of $J = \det(\mathbf{F})$ and a *penalty parameter*, $\kappa > 0$ [21]. A simple form of this function often used in numerical computations is

$$\mathbf{\Psi}_{\text{vol}}(J) = \frac{\kappa}{2}(J-1)^2. \tag{2.14}$$

The volumetric function in (2.14) is symmetric around $J = 1$, meaning that expansion and compression are penalized equally. Many other volumetric strain energy functions penalize compression or expansion more, and the reader is referred to [51, p. 223], since a detailed overview is beyond the scope of this thesis.

2.2 Modeling of the Myocardium

Unlike other hyperelastic materials, the myocardium can contract without being subject to external forces. For such materials, the isochoric term can be expressed as the sum of a passive and an active term. In this section, we will introduce one model proposed for the passive myocardium and one proposed for the active myocardium.

2.2.1 Modeling of the Passive Myocardium

The model for the passive myocardium used in this thesis is based on a transversely isotropic constitutive law by Giccuone et al [17]. An isotropic material has the same physical properties in all directions. In contrast, a transversely isotropic material has the same properties in two orthogonal directions, but different properties in the direction orthogonal to these. In this model, Guccione et al define a basis $\{\mathbf{f}_0, \mathbf{s}_0, \mathbf{n}_0\}$ in three orthonormal directions, fiber axis, sheet axis, and sheet-normal axis within the myocardium and apply this to the Green-Lagrange strain tensor ($\mathbf{E}$), defined as

$$\mathbf{E} = \frac{1}{2}(\mathbf{C} - \mathbf{I}), \tag{2.15}$$

where $\mathbf{C}$ is called the *right Cauchy-Green deformation tensor* and is defined as $\mathbf{C} = \mathbf{F}^T\mathbf{F}$ and $\mathbf{I}$ is the three-dimensional identity matrix. The passive strain energy function in this model is defined as

$$\boldsymbol{\Psi} = \frac{C}{2}(e^Q - 1), \tag{2.16}$$

where Q is an invariant defined as

$$\begin{aligned} Q = {} & b_f E_{ff}^2 \\ & + b_t(E_{ss}^2 + E_{nn}^2 + E_{sn}^2 + E_{ns}^2) \\ & + b_{fs}(E_{fs}^2 + E_{sf}^2 + E_{fn}^2 + E_{nf}^2), \end{aligned} \tag{2.17}$$

where E_{ij} are strain components in the $(\mathbf{f}_0, \mathbf{s}_0, \mathbf{n}_0)$ coordinate system and C, b_f, b_t, b_{fs} are material parameters.

2.2.2 Modeling of the Active Myocardium

This behavior of contracting without any external force is modeled using one of two approaches, the *active stress* and *active strain* formulation [2]. In this thesis, we will use the active stress formulation. Doing this will make it easier to compare our results to Land et al. [30] because they model the active contraction using an active stress approach, as well.

We start by rewriting the total stress as a sum of passive and active stress

$$\boldsymbol{\sigma} = \boldsymbol{\sigma}_p + \boldsymbol{\sigma}_a, \tag{2.18}$$

where the passive component ($\boldsymbol{\sigma}_p$) is derived from the material model

$$\boldsymbol{\sigma}_p = J^{-1}\mathbf{P}\mathbf{F}^T = J^{-1}\frac{\partial \boldsymbol{\Psi}}{\partial \mathbf{F}}\mathbf{F}^T. \tag{2.19}$$

The active component ($\boldsymbol{\sigma}_a$) is modeled as

$$\boldsymbol{\sigma}_a = \boldsymbol{\sigma}_{ff}\mathbf{f} \otimes \mathbf{f} + \boldsymbol{\sigma}_{ss}\mathbf{s} \otimes \mathbf{s} + \boldsymbol{\sigma}_{nn}\mathbf{n} \otimes \mathbf{n}, \tag{2.20}$$

where $\boldsymbol{\sigma}_{ff}$, $\boldsymbol{\sigma}_{ss}$ and $\boldsymbol{\sigma}_{nn}$ denote the active stress in the fiber, sheet and sheet-normal directions and $\mathbf{f} = \mathbf{F}\mathbf{f}_0$, $\mathbf{s} = \mathbf{F}\mathbf{s}_0$ and $\mathbf{n} = \mathbf{F}\mathbf{n}_0$. Applying $\mathbf{F}$ to the orthogonal directions defined above will push them from the reference configuration to the current configuration. If we assume a uniform transverse activation, the total active stress can be written as

$$\boldsymbol{\sigma}_a = T_a[\mathbf{f} \otimes \mathbf{f} + \eta(\mathbf{s} \otimes \mathbf{s} + \mathbf{n} \otimes \mathbf{n})], \tag{2.21}$$

where η is the amount of transverse activation, and T_a is a scalar representing the magnitude of the active tension. In the case that the active tension is only acting along the fibers, we can set $\eta = 0$, and the active stress becomes

$$\boldsymbol{\sigma}_a = T_a\mathbf{f} \otimes \mathbf{f}. \tag{2.22}$$

If we define a transversely isotropic invariant ($I_{4\mathbf{f}_0}$) as

$$I_{4\mathbf{f}_0} = \mathbf{f}_0 \cdot \mathbf{C}\mathbf{f}_0,$$

and use that

$$\begin{aligned} \frac{\partial I_{4\mathbf{f}_0}}{\partial \mathbf{F}} &= \frac{\partial(\mathbf{f}_0 \cdot \mathbf{C}\mathbf{f}_0)}{\partial \mathbf{F}} = 2\mathbf{f} \otimes \mathbf{f}_0, \\ \Rightarrow \mathbf{f} \otimes \mathbf{f} &= \frac{1}{2}\frac{\partial I_{4\mathbf{f}_0}}{\partial \mathbf{F}}\mathbf{F}^T, \end{aligned}$$

we get the equation for the active strain energy function

$$\boldsymbol{\Psi}_a = \frac{T_a}{2J}[I_{4\mathbf{f}_0} - 1], \tag{2.23}$$

which also gives us an expression of the active stress

$$\boldsymbol{\sigma}_a = J^{-1}\frac{\partial \boldsymbol{\Psi}_a}{\partial \mathbf{F}}\mathbf{F}^T. \tag{2.24}$$

Before moving on to other mathematical models, it is important to take a step back and summarize what we have introduced, and why. As stated above, our goal is to find the deformation of a body that satisfies the relationship between the stresses in the body and the forces acting on it. For this purpose, we express the stress tensor, $\mathbf{P}$, as a function of the displacement field, $\mathbf{u}$, via the deformation gradient, $\mathbf{F}$. Furthermore, because the myocardium is modeled as a hyperelastic material, we express $\mathbf{P}$ using a strain energy function, $\boldsymbol{\Psi}$. In addition, to simplify our calculations, we implement the incompressibility property of the myocardium using the penalty method. For this method, we model $\boldsymbol{\Psi}$ as a decoupled function of a volume-changing and volume-preserving term, $\boldsymbol{\Psi} = \boldsymbol{\Psi}_{\text{vol}} + \boldsymbol{\Psi}_{\text{iso}}$. Moreover, to account for the active and passive response of the myocardium, we express the volume-preserving term as a sum of a passive and an active strain energy function, yielding a final expression for the strain energy function

$$\boldsymbol{\Psi} = \boldsymbol{\Psi}_{\text{vol}} + \boldsymbol{\Psi}_p + \boldsymbol{\Psi}_a. \tag{2.25}$$

2.3 Neural Networks

In this section, we will introduce neural networks (NNs), explain how an NN is trained, and how its performance is evaluated. Starting, we will define the terms used in this section, and summarize the training process for an NN, before introducing the most common activation functions. Following this is a walkthrough of the forward propagation process for activating an NN, and the backpropagation algorithm for calculating the gradients of a loss function. After this, we will show how the gradients can be used to tune the NN with gradient descent techniques to achieve better performance. The idea of data splitting and the problem of data scarcity will be explained before a short, theoretical introduction to physics-informed neural networks is presented. The mathematical argumentation in section 2.3.3 is taken from the article by Nguyen-Thanh et al. [41]. For a more detailed description of backpropagation, the reader is referred to the online book by Nielsen [43], which serves as the basis for section 2.3.4. Although much of the theory in this section applies to other types of NNs, this section focuses on feed-forward neural networks, as this is the only type of NN used in this thesis.

2.3.1 Neural Network Architecture

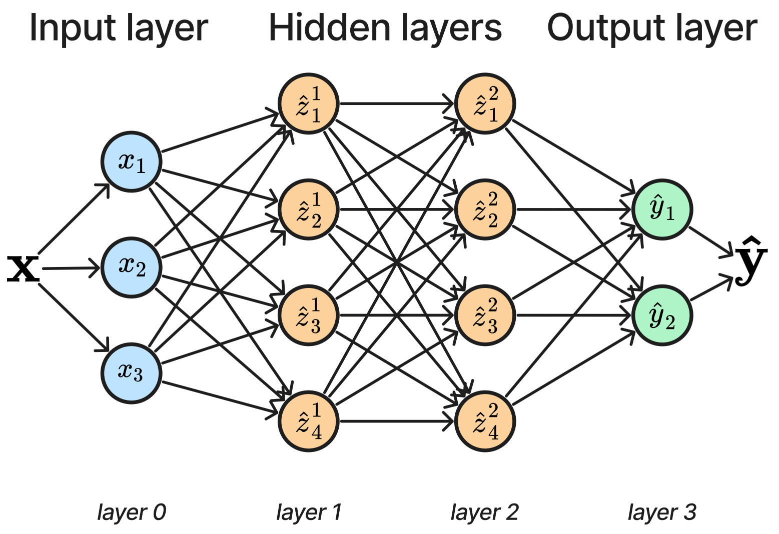


Figure 2.2: A graphic illustration of a neural network with 3 nodes in the input layer, 2 hidden layers with 4 nodes in both, and 2 nodes in the output layer. Input $\mathbf{x}$ propagates through the network, yielding an output $\hat{\mathbf{y}}$.

A NN is a collection of computational units called *nodes*, or *neurons*, connected to neighboring nodes. In Figure 2.2 the nodes are represented by circles and the connections between them by arrows. The nodes that receive the input, are said to be in the *input layer*, and the nodes that give the output in the *output layer*. All nodes between the input and output layers are in *hidden layers*. Each node in the hidden layers is connected to the nodes in the previous and following layers. The nodes in the hidden layers and the output layer contain scalar functions, called *activation functions*, while the nodes in the input layer contain identity functions. Furthermore, the data propagating between any two nodes is multiplied with a *weight* specific to the two nodes. In addition, the weighted sum of the data from all neurons in any given layer is added to a *bias*, which is specific to every receiving neuron. The total number of layers in an NN determines the *depth* of the network, while the largest number of nodes in any layer determines its *width* [35]. The NN in Figure 2.2 has a depth of 4 and a width of 4.

The goal of an NN is to approximate a function $\mathbf{f}$, which fulfills $\mathbf{y} = \mathbf{f}(\mathbf{x})$ for an input $\mathbf{x}$ and a target $\mathbf{y}$, by optimizing its weights and biases. Like a function, an NN takes in an input, denoted as $\mathbf{x}$, and gives an output $\hat{\mathbf{y}}$, where $\hat{\mathbf{y}}$ is an NN approximation to $\mathbf{y}$. The NN approximation to $\mathbf{f}$ is denoted $\hat{\mathbf{f}}$ and satisfies $\hat{\mathbf{y}} = \hat{\mathbf{f}}(\mathbf{x})$.

The standard algorithm for optimizing the parameters of an NN consists of three steps. In the first step, we activate the NN by sending an input $\mathbf{x}$ into the NN and receive an output $\hat{\mathbf{y}}$. The error between this output and the target $\mathbf{y}$ gives us a measure of the NN's performance, and we use this error by calculating its derivative with respect to every parameter in the network in the second step. In the third step, we use these derivatives to change the parameters towards a lower error.

2.3.2 Activation Functions

An *activation function* is contained within the nodes of the hidden layers. In Figure 2.3 we have plotted the most common activation functions: the *Sigmoid*

$$f(x) = \frac{1}{1+e^{-x}},$$

the *rectified linear unit (ReLU)*

$$f(x) = \max(0, x) = \frac{x+|x|}{2},$$

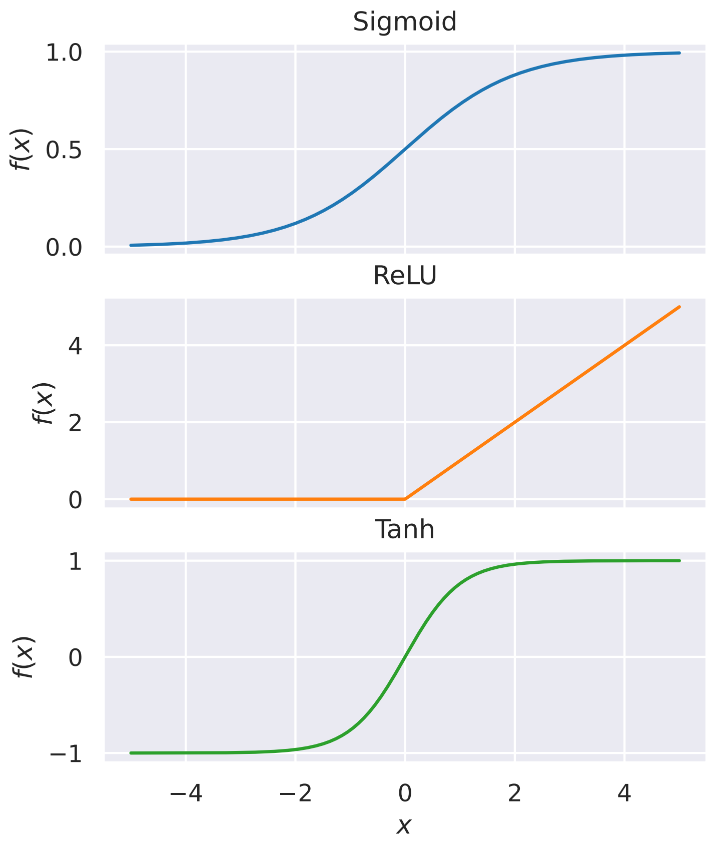


Figure 2.3: Plot of the three most common activation functions used in NNs. **Top:** Sigmoid. **Middle:** ReLU. **Bottom:** Tanh.

and the *hyperbolic tangent (Tanh)*

$$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}.$$

The activation functions serve the purpose of adding non-linearity to the NN. They are necessary because the connections between the nodes perform purely linear operations and the NN can only approximate linear functions without them. For most feed-forward NNs, ReLU is recommended because it is piecewise linear with two pieces and is therefore close to being linear. Because of its almost-linearity, the ReLU retains the properties that simplify optimization using gradient-based methods for linear models [15]. In Figure 2.3, we see that both the Sigmoid and the Tanh functions are nearly constant for high and low values of x. This means that their gradients for inputs with large magnitudes are zero, making gradient-based learning, learning through minimization methods that use gradients, challenging.

2.3.3 Forward Propagation

Before training the NN, we activate it in a step called *forward propagation*. Forward propagation consists of activating the NN by feeding data into the input layer and propagating the data throughout the network, until we get an output at the output layer.

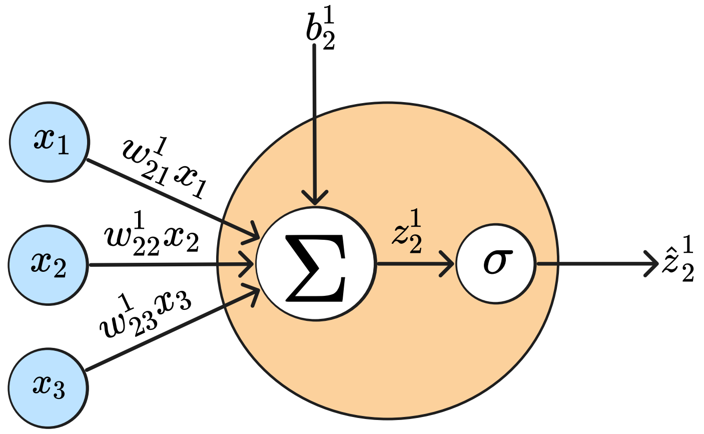


Figure 2.4: Illustration of the activation of neuron 2 in layer 1 of an NN with three nodes in the input layer. The weighted input is summed together with the bias term b_2^1 before the sum is sent through the activation function, yielding an activation. Neurons in the input layer are colored blue and neuron 2 in hidden layer 1 is colored orange.

Given an input vector $\mathbf{x} = (x_1, ..., x_{n_0})$, with $n_0 = 3$ in Figure 2.4, and an activation function σ, we can define an *activation* $(\hat{z}_k^l)$ as the output from node k in layer l. For the first hidden layer, we get the activations

$$\hat{z}_k^1 = \sigma\left(\sum_{j=1}^{n_0} w_{kj}^1 x_j + b_k^1\right), \qquad k = 1, \ldots, n_1, \tag{2.26}$$

where w_{kj}^1 is the weight for data propagating from neuron j in the input layer to neuron k in hidden layer 1, b_k^1 is the bias term added to the weighted sum propagating to neuron k in hidden layer 1 and n_0 and n_1 denote the number of nodes in the input layer and the first hidden layer. In general, n_l is the number of nodes in layer l. In (2.26), each input element, x_j, is weighted with its corresponding weight w^1_{kj} and summed. Added to this weighted sum is the bias term, b^1_k, to avoid the input to the first hidden layer becoming zero, if the input is zero. The bias term is not multiplied with any input, and will therefore shift the weighted sum away from zero. The total sum is then sent to the nodes in the first hidden layer, yielding the activation. The activation of neuron 2 in layer 1 is illustrated in Figure 2.4. For the output from the activation functions, the symbol $\hat{z}$ is used, and for the input to the activation functions, z is used. This notation gives us the relation for neuron k in layer 1

$$\hat{z}^1_k = \sigma(z^1_k), \tag{2.27}$$

where

$$z^1_k = \sum_{j=1}^{n_0} w^1_{kj} x_j + b^1_k. \tag{2.28}$$

This process is repeated for the activations from the first hidden layer with the new weights w^2_{kj} and new bias terms b^2_k, which give the activation for node k in the second hidden layer

$$\hat{z}^2_k = \sigma\left(\sum_{j=1}^{n_1} w^2_{kj}\hat{z}^1_j + b^2_k\right), \qquad k = 1, \dots, n_2. \tag{2.29}$$

For hidden layer l in the NN, the activation becomes

$$\hat{z}^l_k = \sigma\left(\sum_{j=1}^{n_{l-1}} w^l_{kj}\hat{z}^{l-1}_j + b^l_k\right), \qquad k = 1, \dots, n_l, \tag{2.30}$$

and for the final layer, L, the output is

$$\hat{y}_k = \hat{z}^L_k = \sigma^L\left(\sum_{j=1}^{n_{L-1}} w^L_{kj}\hat{z}^{L-1}_j + b^L_k\right), \qquad k = 1, \dots, n_L, \tag{2.31}$$

where σ^L is the activation function used in the output layer. This is a function that converts the data from the hidden layers into a desired output format and differs from the activation functions used in the hidden layers. Written in vector form, the algorithm becomes

$$\hat{\mathbf{z}}^l = \boldsymbol{\sigma}\left(\mathbf{W}^l \cdot \hat{\mathbf{z}}^{l-1} + \mathbf{b}^l\right), \qquad 1 \leq l \leq L, \tag{2.32}$$

where the activation function ($\boldsymbol{\sigma}$), both of the activations ($\hat{\mathbf{z}}^l$, $\hat{\mathbf{z}}^{l-1}$), and the bias term ($\mathbf{b}^l$) are written in boldface font because they are vector-valued. In addition, the weight factor ($\mathbf{W}^l$) is written in capital because it is a matrix. Using a vector-valued (2.27) for layer l, we can rewrite (2.32) as

$$\hat{\mathbf{z}}^l = \boldsymbol{\sigma}(\mathbf{z}^l), \qquad 1 \leq l \leq L, \tag{2.33}$$

where

$$\mathbf{z}^l = \mathbf{W}^l \cdot \hat{\mathbf{z}}^{l-1} + \mathbf{b}^l. \tag{2.34}$$

Since our goal is to find the weights and biases that give the best approximation $\hat{\mathbf{f}}$ to a function $\mathbf{f}$, we need a way to update the weights and biases in a way that ensures that the approximation converges towards the target. To achieve this, we define a *loss function.*

2.3.4 Backpropagation

To update the weights and biases in a way that ensures that the approximation converges toward the data we want to approximate, we must first define a loss function. This function serves as a measure of how well the NN has approximated the desired output. There are many ways to calculate the error, and different functions are used for different purposes. One common definition of a loss function ($\mathcal{L}$) is the *mean squared error*

$$\mathcal{L} = \frac{1}{N}\sum_{i=0}^{N-1}(y_i - \hat{y}_i)^2, \tag{2.35}$$

which calculates the error as a squared average over all values. Because our goal is to minimize the loss function, we will use gradient-based methods. We calculate the gradient of the loss function with respect to every weight and bias to find the direction of the minima of the loss function. With respect to the weights, this gradient is

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}}, \tag{2.36}$$

and with respect to the biases, it is

$$\frac{\partial \mathcal{L}}{\partial \mathbf{b}}. \tag{2.37}$$

The most common algorithm used to calculate the gradient of the error throughout the network is called *backpropagation.* The backpropagation algorithm uses the chain rule to calculate the gradient with respect to the network parameters. A detailed derivation of the backpropagation algorithm can be found in Appendix A.

2.3.5 Gradient Descent

The gradient of the loss function gives us the direction in which the loss function increases the steepest with the current parameters. Therefore, the negative of the gradient gives us the direction of the steepest decrease of the error with respect to the current parameters. The idea of gradient descent is to use the gradient to take a step towards a lower loss until you find the minimum of the loss function.

There are many algorithms for using the gradient to find the minima of the loss function including first-order algorithms such as *stochastic gradient descent* (SGD) and *adaptive gradient* (AdaGrad) [50] and second-order algorithms like *limited-memory Broyden–Fletcher–Goldfarb–Shanno* algorithm (L-BFGS) [34]. We will not go into the theory behind them or how they are implemented in detail since this is beyond the scope of this thesis. The main difference is that first-order algorithms use the gradient (first-order derivative) of the loss function to find the direction of the steepest descent and take a step of size η, in this direction. The scalar parameter η is called the *learning rate.* Second-order algorithms use the Hessian matrix (second-order derivative) of the loss function in addition to the gradient. The main part of gradient descent algorithms can be expressed mathematically as

$$\begin{aligned} \mathbf{W} &\leftarrow \mathbf{W} - \eta \frac{\partial \mathcal{L}}{\partial \mathbf{W}}, \\ \mathbf{b} &\leftarrow \mathbf{b} - \eta \frac{\partial \mathcal{L}}{\partial \mathbf{b}}. \end{aligned} \tag{2.38}$$

Repeating the forward propagation, backpropagation, and gradient descent steps until the error is sufficiently small, or until a set number of iterations, called *epochs* are completed is referred to as *training* the NN. This process is illustrated in Figure 2.5. After the NN has been trained, meaning its parameters have been optimized, the NN is *evaluated.*

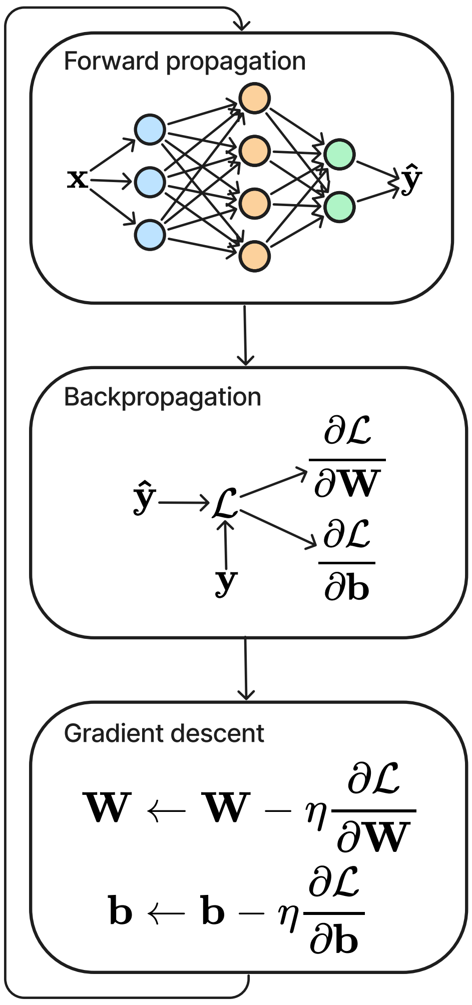

Forward propagation
$\mathbf{x}$
$\hat{\mathbf{y}}$
Backpropagation
$\hat{\mathbf{y}} \longrightarrow \mathcal{L}$
$\mathbf{y}$
$\frac{\partial \mathcal{L}}{\partial \mathbf{W}}$
$\frac{\partial \mathcal{L}}{\partial \mathbf{b}}$
Gradient descent
$\mathbf{W} \leftarrow \mathbf{W} - \eta \frac{\partial \mathcal{L}}{\partial \mathbf{W}}$
$\mathbf{b} \leftarrow \mathbf{b} - \eta \frac{\partial \mathcal{L}}{\partial \mathbf{b}}$

Figure 2.5: Illustration of the training process for an NN. The training consists of three steps: forward propagation, backpropagation, and gradient descent. During forward propagation, the NN is evaluated on input $\mathbf{x}$, yielding an output $\hat{\mathbf{y}}$. A loss is calculated from the NN output $\hat{\mathbf{y}}$ and the target $\mathbf{y}$ and the gradient of the loss with respect to all parameters is calculated with the backpropagation algorithm. These gradients are used during gradient descent to optimize the parameters by taking a step in the negative direction of the gradient.

2.3.6 Data Splitting

When training and evaluating, or *testing*, an NN, the data available is split into the categories *training set* and *test set*. The training set is used to train the NN. This set contains the data the NN tries to fit by optimizing its parameters. After the training, the test set is fed to the trained NN for a forward-propagation pass and its loss on the test set is used to evaluate the performance of the NN. The training and test sets have to be from the same data set, and, to ensure that the NN is evaluated on data it has never seen, the two sets must also be disjointed. A NN that performs well on the training set, but poorly on the test set is said to be *overfitting* the training set.

In addition to the weights and biases, NNs have settings that affect their performance but are not optimized during training. These settings are called *hyperparameters* and include among others the learning rate, the activation function, and the number of hidden layers. Some settings are chosen to be hyperparameters because they are difficult to optimize, such as the learning rate. Other settings are hyperparameters since they are not appropriate to learn on a training set because they affect model complexity, such as the number of hidden layers. In essence, an NN that learns the latter type of hyperparameter will always choose the maximum possible complexity, leading to overfitting [15].

To combat this problem, it is common to have a third category called *validation set*, which is used to evaluate the model before feeding it the test set. The validation set must also be from the same data set and be disjoint from both the training and test set. The NN is evaluated on the validation set after the model is trained or during the training. If the validation set is used after the training phase, the error on this set is used to decide which architecture to choose for testing. On the other hand, if the validation set is used during training, the model is evaluated on this set at every training step. In this case, the model is trained until the error on the validation set increases from one training step to the next, implying the model is overfitting the training data. Using the validation set to stop the NN from overfitting is known as *early stopping* [29]. It is common to use the ratio 50:25:25 or 60:20:20 for the training, validation, and test set [36] and Figure 2.6 shows data splitting with the former ratio. Furthermore, the three sets have to satisfy what is called the *i.i.d. assumpsions*, which state that the data points are independent of each other and the sets are identically distributed [15].

2.3.7 Data Scarcity

Large NN models can have tens or hundreds of millions of parameters that have to be optimized. This leads to a need for large amounts of data. The biggest challenge is that the data has to be *labeled*, meaning that for each data point x,

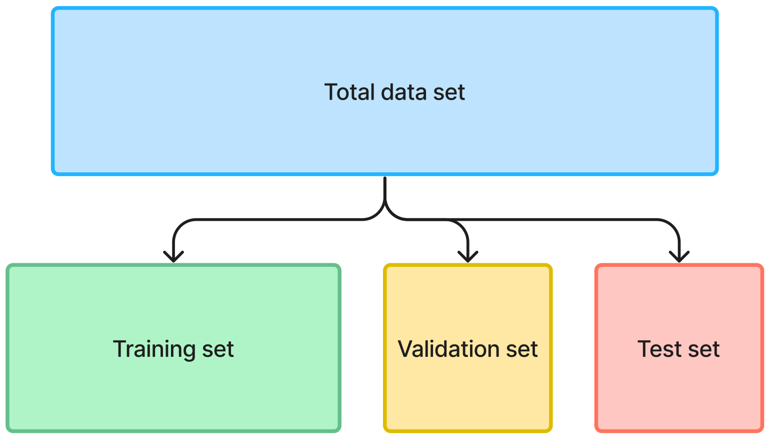


Figure 2.6: Illustration of data splitting. The total data is split into three disjoint sets: training, validation, and test.

that you want to train, validate, or evaluate your model on, you need a target, or label, y, against which you can compare the NN calculated output. If you are training an NN to recognize images of cats, then x can be an image and y can be the labels 1 for "cat" or 0 for "not cat". Labeling data takes a long time and often comes at a high cost because you need human beings to look or read through all the data and label each data point correctly. While this is easy to do for images of cats, it is much more difficult when labeling comments for a product as "positive", "negative" or "neither" or labeling sentences in one language with their translation in a target language. Many fields also require experts to label their data. For example, in cardiac applications, medical doctors are responsible for labeling data. Access to labeled data is therefore seen as a significant barrier when training new NN models [3].

Different solutions have been suggested to deal with the scarcity of labeled data. Several augmentation techniques are commonly used in image classification or segmentation [18] and different techniques for data-efficient AI have been developed [49]. In addition, a solution called *physics-informed neural network* has been suggested by Raissi et al. [48], in which they constrain the NN approximation by utilizing the physical laws governing the problem.

2.3.8 Physics-Informed Neural Networks

Many physical problems can be stated as partial differential equations (PDEs) [32]. In their paper, Raissi et al. [48] consider general PDEs of the form

$$\mathcal{D}(\mathbf{u}) = \mathbf{h}(\mathbf{x}), \tag{2.39}$$

where $\mathbf{u}$ is the solution they want to find, $\mathcal{D}(\cdot)$ is a non-linear differential operator and $\mathbf{h}$ is an arbitrary function. They move all terms in (2.39) to the left-hand side and define $\mathbf{f}$ to be the residual of the PDE

$$\mathbf{f} := \mathcal{D}(\mathbf{u}) - \mathbf{h}(\mathbf{x}), \tag{2.40}$$

which is zero for the exact solution of the PDE. Physics-informed neural networks (PINNs) use this fact by including $\mathbf{f}$ in the loss function to constrain the NN approximation. The new loss function $\mathcal{L}$ in PINNs therefore becomes

$$\mathcal{L} = \mathcal{L}^{\text{data}} + \mathcal{L}^{\text{PDE}}, \tag{2.41}$$

where $\mathcal{L}^{\text{data}}$ is a loss function on the data introduced in section 2.3.4 and $\mathcal{L}^{\text{PDE}}$ is the loss on the PDE, defined as

$$\mathcal{L}^{\text{PDE}} = \frac{1}{N_f} \sum_{i,j,k}^{N_f} |\mathcal{D}(\hat{\mathbf{u}}_{\text{i,j,k}}) - \mathbf{h}(\mathbf{x}_{\text{i,j,k}})|^2. \tag{2.42}$$

Here, N_f is the number of collocation points chosen uniformly, or randomly, within the domain to measure the error on the PDE and $\hat{\mathbf{u}}$ is the NN approximation to $\mathbf{u}$.

Using the residual in the loss function lifts some of the burden of lacking labeled data. Furthermore, since the collocation points at which the residual is evaluated are chosen randomly, the error on the residual can be weighted more or less than the error on the data by using more or fewer collocation points. In mathematics and the natural sciences, it is common to use a residual to measure the error of an approximation when the true solution is unknown. An example of this is the Galerkin method for solving PDEs when using the *finite element method*.

2.4 Finite Element Method

A common feature of mathematical models in natural sciences and most engineering disciplines is the appearance of PDEs. Due to the development of better and faster computer hardware and the improvement of numerical algorithms and scientific software, engineers and scientists can now use mathematical models and computer simulation to explore complex real-world applications [31]. Unfortunately, the analytical solutions to the PDEs are only available for simple cases. Generally, scientists are dependent on numerical techniques to find approximate solutions [25]. The finite element method (FEM) is the most popular method for solving PDEs in applied areas and is preferable to the finite difference method when dealing with complex geometries [54].

This section will cover the basics of the FEM. To start with, we will introduce the concept of elements and basis functions before moving on to the Galerkin method for the discretization of PDEs and rounding off with a short introduction to FEniCS, a Python module for solving PDEs. Section 2.4.1 and section 2.4.2 are based on the books by Hans Petter Langtangen [31, 46].

2.4.1 Elements and Basis Functions

The idea behind the FEM is to divide the domain into a finite set of *elements*, denoted together as Ω. In Figure 2.7 we have five one-dimensional elements $\Omega = \{\Omega^{(0)}, \Omega^{(1)}, \Omega^{(2)}, \Omega^{(3)}, \Omega^{(4)}\}$. Every element in Figure 2.7 shares its *nodes*, illustrated with red circles, with neighboring elements. These are called *shared nodes*. More complex elements can have *internal nodes*, as well. For each node, we define basis functions $\varphi(x)$, such that basis function $\varphi_i(x)$ is 1 at node i and 0 elsewhere. The basis functions over nodes 2 and 3 are shown in Figure 2.7. The polynomials used here are Lagrange polynomials of degree 1. The rest of the basis functions are created in a similar fashion. For PDEs in two dimensions, the elements are two-dimensional, and generally, we use n-dimensional elements for n-dimensional PDEs.

We use these basis functions to approximate the continuous solution to our PDE, $\mathbf{u}_e \in V$, where V is an infinite-dimensional space, with a discrete approximation, $\mathbf{u} \in V_h$, where $V_h \subset V$ is a finite-dimensional subspace of V. To use the FEM, we need to reformulate our PDEs in the *weak variational form* [46] which will be introduced in section 2.5.1.

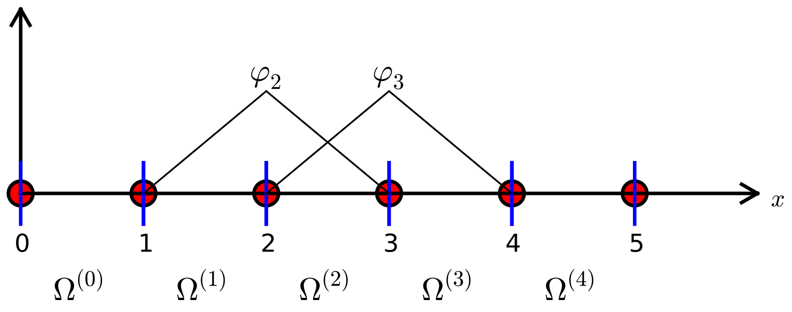


Figure 2.7: Illustration of elements and basis functions in the FEM. The domain $x \in [0,5]$ is divided into 5 elements, $\Omega^{(0)}, ..., \Omega^{(4)}$. The basis functions for nodes 2 and 3, φ_2 and φ_3 are illustrated. The basis functions in this illustration are Lagrange polynomials of degree 1. Reprinted from *Introduction to Numerical Methods for Variational Problems* by H. P. Langtangen and K. A. Mardal. Copyright (c) 2019 Hans Petter Langtagen Kent-Andre Mardal.

2.4.2 The Galerkin Method

The example used here is one-dimensional because the basis functions are easy to relate to Figure 2.7 and the mathematics is easier to grasp. The algorithm is the same for PDEs in higher dimensions but with higher-dimensional vector spaces and basis functions.

The Galerkin method, also called the *projection method* is a method for approximating the exact solution to a differential equation, u_e with a discrete approximation, u by demanding that the error between the approximation and the exact solution is orthogonal to the solution space V_h. Given a differential equation on the form

$$\mathcal{D}(u) = h(x), \qquad x \in \Omega, \tag{2.43}$$

where $\mathcal{D}(\cdot)$ is a differential operator on u and $h(x)$ is some arbitrary function. This is the one-dimensional equivalent of the PDE in (2.39). We know that this equation is fulfilled for the exact solution. Given a solution set $V_h = \text{span}\{\varphi_0(x), \varphi_1(x), \dots \varphi_N(x)\}$, we approximate the exact solution with

$$u(x) = \sum_{j=0}^{N} c_j \varphi_j(x), \tag{2.44}$$

where the coefficients $\{c_j\}_{j=0}^{N}$ are unknowns we want to find. In Galerkin literature, u is called a *trial function*. From equation 2.43, we define a *residual* (R)

$$R = \mathcal{D}(u) - h(x), \tag{2.45}$$

which is necessarily zero for the exact solution, but non-zero for our approximation. This is the same residual as defined in (2.40) for PINNs. As stated above, we can use this residual as a measure of the error of our approximation. The Galerkin method requires that the residual is orthogonal to V_h, meaning that we want to find $\{c_i\}_{i=0}^N$, such that

$$(R, v) = 0, \qquad \forall v \in V_h, \tag{2.46}$$

where (f, g) is the inner product between functions f and g, which is zero if f and g are orthogonal and is defined as

$$(f, g) = \int_\Omega f(x)g(x)\,dx,$$

and v is called a *test function* and is defined as $v = \{v_i\}_{i=0}^N$ where $v_i = \varphi_i(x)$.

In addition, the boundaries of the domain of PDEs have different conditions that must be fulfilled. These *boundary conditions* constrain the solution of the PDEs. Two common types, and ones used in this thesis, are *Dirichlet* and *Neumann* boundary conditions. Dirichlet boundary conditions are implemented by setting a constraint on the solution u along a boundary. They are on the form

$$u(x) = a, \qquad x \in \Gamma_D, \tag{2.47}$$

where Γ_D is a *Dirichlet boundary.* Neumann boundary conditions set a constraint on the derivative of the solution and are on the form

$$u'(x) = b, \qquad x \in \Gamma_N, \tag{2.48}$$

where Γ_N is a *Neumann boundary.* The total boundary can be expressed as

$$\partial\Omega = \Gamma_D \cup \Gamma_N.$$

2.4.3 FEniCS

To solve the system of equations presented above, we will use the open-source software library FEniCS. FEniCS is a library for C++ and Python designed for computational modeling, providing efficient and flexible software for solving PDEs using the finite element method [46]. When we have defined the PDE, with its domain and boundary conditions, and rewritten it in a variational formulation, we can solve it with FEniCS by writing a Python program that defines the domain, the variational formulation, and the boundary conditions using the appropriate FEniCS abstractions [46]. An advantage FEniCS has over other frameworks for solving finite element problems is that the code stays close to the mathematical formulations, even in complex problems [46]. This is because FEniCS relies on *unified form language* [1] to express variational forms of PDEs. Unified form language is a language that can be used to declare weak formulations of PDEs. Mathematical operations commonly carried out on weak formulations are implemented in unified form language allowing for compact and descriptive coding of mathematical problems [1].

2.5 Variational Formulations

When solving PDEs using the FEM, we start by rewriting the PDE in a *variational formulation* [46]. The goal here is to reduce the maximum degree of differentiation of the solution in the problem. Solving the variational formulation sets a weaker constraint on the differentiability of the solution than the original PDE. Therefore, variational formulations are often called *weak formulations.* Variational formulations of PDEs are mathematically useful when analyzing if a solution is unique, stable, or even exists [21].

This section will introduce variational formulations such as the weak formulation and the principle of stationary potential energy. We will derive the weak formulation of the PDE for a compressible, hyperelastic material and show that we arrive at the same result using the principle of stationary potential energy.

2.5.1 Weak Formulation

To obtain the weak formulation of a PDE, we need to state the *strong formulation.* The strong formulation was derived as (2.8) in section 2.1.1

$$-\nabla \cdot \mathbf{P} = \mathbf{B}.$$

To rewrite this in a weak form, we start by multiplying the equation by a test function $\mathbf{v}$, then move all terms to the left-hand side and integrate over the entire domain

$$-\int_{\Omega} (\nabla \cdot \mathbf{P}) \cdot \mathbf{v}\, dV - \int_{\Omega} \mathbf{B} \cdot \mathbf{v}\, dV = \mathbf{0}. \tag{2.49}$$

In the left-most term, we have the divergence of the stress tensor $\mathbf{P}$, which itself is a function of the gradient of the deformation. This results in a double derivative. To reduce the degree of differentiation, we can rewrite the left-hand side using the product rule for divergence

$$\begin{aligned} \nabla \cdot (\mathbf{P} \cdot \mathbf{v}) &= (\nabla \cdot \mathbf{P}) \cdot \mathbf{v} + \mathbf{P} : \nabla \mathbf{v}, \\ \Rightarrow -(\nabla \cdot \mathbf{P}) \cdot \mathbf{v} &= \mathbf{P} : \nabla \mathbf{v} - \nabla \cdot (\mathbf{P} \cdot \mathbf{v}), \end{aligned}$$

where the : operator is a double contraction, the sum of element-wise products of tensors. Using this relation, we can rewrite (2.49)

$$\begin{aligned} \int_\Omega \mathbf{P} : \nabla \mathbf{v} \, dV - \int_\Omega \nabla \cdot (\mathbf{P} \cdot \mathbf{v}) \, dV - \int_\Omega \mathbf{B} \cdot \mathbf{v} \, dV &= \mathbf{0}, \\ \Rightarrow \int_\Omega \mathbf{P} : \nabla \mathbf{v} \, dV - \int_\Omega \mathbf{B} \cdot \mathbf{v} \, dV - \int_\Omega \nabla \cdot (\mathbf{P} \cdot \mathbf{v}) \, dV &= \mathbf{0}. \end{aligned}$$

Now, we can use the divergence theorem to turn the right-most term on the left-hand side into a surface integral

$$\int_\Omega \nabla \cdot (\mathbf{P} \cdot \mathbf{v}) \, dV = \int_{\partial\Omega} (\mathbf{P} \cdot \mathbf{v}) \cdot \mathbf{N} \, dS = \int_{\partial\Omega} \mathbf{P} \cdot \mathbf{N} \cdot \mathbf{v} \, dS,$$

and get the weak formulation of the PDE

$$\int_\Omega \mathbf{P} : \nabla \mathbf{v} \, dV - \int_\Omega \mathbf{B} \cdot \mathbf{v} \, dV - \int_{\partial\Omega} (\mathbf{P} \cdot \mathbf{N}) \cdot \mathbf{v} \, dS = \mathbf{0}. \tag{2.50}$$

We can further simplify the last term on the left-hand side by inserting the first Piola-Kirchhoff traction force, defined as $\mathbf{T} = \mathbf{P} \cdot \mathbf{N}$, where $\mathbf{N}$ is the vector normal to the body surface

$$\int_\Omega \mathbf{P} : \nabla \mathbf{v} \, dV - \int_\Omega \mathbf{B} \cdot \mathbf{v} \, dV - \int_{\partial\Omega} \mathbf{T} \cdot \mathbf{v} \, dS = \mathbf{0}. \tag{2.51}$$

In addition to the method shown in this section, we can derive the weak formulation using the *principle of stationary potential energy.*

2.5.2 Principle of Stationary Potential Energy

In physics, the principle of stationary potential energy states that when the total potential energy of a system is minimized, the system is at equilibrium, and any infinitesimal changes should not add any energy to the system. This implies that minimizing the potential energy yields the deformation in which the system is at equilibrium. For a compressible, hyperelastic material, the energy in the system ($\mathbf{\Pi}$) is given by

$$\begin{aligned}\mathbf{\Pi}(\mathbf{u}) &= \mathbf{\Pi}_{\text{int}}(\mathbf{u}) - \mathbf{\Pi}_{\text{ext}}(\mathbf{u}),\\ \mathbf{\Pi}_{\text{int}}(\mathbf{u}) &= \int_\Omega \mathbf{\Psi}(\mathbf{F})\,dV,\\ \mathbf{\Pi}_{\text{ext}}(\mathbf{u}) &= \int_\Omega \mathbf{B}\cdot\mathbf{u}\,dV + \int_{\partial\Omega}\mathbf{T}\cdot\mathbf{u}\,dS.\end{aligned} \tag{2.52}$$

To find the minimum of the energy functional, we can differentiate it and set the derivative to zero. The principle of stationary potential energy implies that the directional derivative of the potential energy with respect to the displacements $\mathbf{u}$ vanish in all directions $\mathbf{v}$ [21], or, expressed mathematically:

$$D_{\mathbf{v}}\mathbf{\Pi}(\mathbf{u}) = \frac{d}{d\varepsilon}\mathbf{\Pi}(\mathbf{u}+\varepsilon\mathbf{v})|_{\varepsilon=0} = 0, \tag{2.53}$$

where $D_{\mathbf{v}}$, called the *Gâteaux operator*, is the directional derivative along a line through $\mathbf{u}$ in the direction $\mathbf{v}$, $\mathbf{\Pi}$ is the total potential energy defined in (2.52) and ε is a scalar.

To derive the variational formulation using this principle, we start by writing the definition of the potential energy

$$\mathbf{\Pi}(\mathbf{u}) = \int_\Omega \mathbf{\Psi}(\mathbf{F})\,dV - \int_\Omega \mathbf{B}\cdot\mathbf{u}\,dV - \int_{\partial\Omega}\mathbf{T}\cdot\mathbf{u}\,dS. \tag{2.54}$$

Differentiating the energy, we get

$$\begin{aligned}D_{\mathbf{v}}\mathbf{\Pi}(\mathbf{u}) = &\int_\Omega \frac{\partial\mathbf{\Psi}}{\partial\mathbf{F}}\frac{\partial\mathbf{F}(\mathbf{u}+\varepsilon\mathbf{v})}{\partial\varepsilon}|_{\varepsilon=0}\,dV\\ &-\int_\Omega \mathbf{B}\frac{\mathbf{u}+\varepsilon\mathbf{v}}{\partial\varepsilon}|_{\varepsilon=0}\,dV - \int_{\partial\Omega}\mathbf{T}\frac{\mathbf{u}+\varepsilon\mathbf{v}}{\partial\varepsilon}|_{\varepsilon=0}\,dS.\end{aligned} \tag{2.55}$$

Using the definition of $\mathbf{F}$ from (2.2), we get

$$\begin{aligned}\mathbf{F}(\mathbf{u}+\varepsilon\mathbf{v}) &= \nabla(\mathbf{u}+\varepsilon\mathbf{v}) + \mathbf{I}\\ &= \nabla\mathbf{u} + \varepsilon\nabla\mathbf{v} + \mathbf{I},\\ \Rightarrow \frac{\partial\mathbf{F}(\mathbf{u}+\varepsilon\mathbf{v})}{\partial\varepsilon} &= \nabla\mathbf{v}.\end{aligned}$$

Combining this result with (2.10), we get

$$D_{\mathbf{v}}\mathbf{\Pi}(\mathbf{u}) = \int_{\Omega} \mathbf{P} : \nabla \mathbf{v}\, dV - \int_{\Omega} \mathbf{B} \cdot \mathbf{v}\, dV - \int_{\partial\Omega} \mathbf{T} \cdot \mathbf{v}\, dS. \tag{2.56}$$

Setting this derivative to zero, we get the final formulation of the problem

$$\int_{\Omega} \mathbf{P} : \nabla \mathbf{v}\, dV - \int_{\Omega} \mathbf{B} \cdot \mathbf{v}\, dV - \int_{\partial\Omega} \mathbf{T} \cdot \mathbf{v}\, dS = \mathbf{0}, \tag{2.57}$$

which is the same result we got in (2.51).

2.6 Deep Energy Method

This section will introduce the deep energy method (DEM), show how it is related to the principle of stationary potential energy, and provide some code examples from our implementation.

2.6.1 Formulation of the Method

The idea behind the deep energy method is to minimize potential energy, following the principle of stationary potential energy, with the aid of an NN. In the DEM, the loss function is defined as the total potential energy in the system, since this is the property we seek to minimize. The loss function ($\mathcal{L}$) for the NN is therefore defined as

$$\begin{aligned} \mathcal{L}(\mathbf{w}, \mathbf{b}) &= \mathbf{\Pi}_{\text{int}} - \mathbf{\Pi}_{\text{ext}}, \\ \mathbf{\Pi}_{\text{int}} &= \int_{\Omega} \mathbf{\Psi}(\hat{\mathbf{F}})\, dV, \\ \mathbf{\Pi}_{\text{ext}} &= \int_{\Omega} \mathbf{B} \cdot \hat{\mathbf{u}}\, dV + \int_{\partial\Omega} \mathbf{T} \cdot \hat{\mathbf{u}}\, dS, \end{aligned} \tag{2.58}$$

where Ω is the domain, $\partial\Omega$ is the boundary, $\mathbf{\Psi}$ is the strain energy function, $\hat{\mathbf{F}}$ is the NN calculated deformation gradient, $\mathbf{B}$ is the body force, $\mathbf{T}$ is the traction force, $\hat{\mathbf{u}}$ is the deformation calculated by the NN and $\mathbf{X}$ are the coordinates in the reference configuration. All variables calculated by the NN are marked with hat symbols.

When comparing (2.58) to (2.52), it is easy to see how minimizing a loss function that is defined as the total potential energy is equivalent to solving a PDE by using the principle of stationary potential energy. One advantage of minimizing the energy instead of the residual, which is the goal of PINNs, is that the energy needs only be differentiated once to find the minimum. In contrast, for second-order PDEs, the NN solution has to be differentiated twice to calculate the residual when using PINNs.

The optimization problem in the DEM can be stated as *find parameters* $(\mathbf{W}, \mathbf{b})^*$ *that minimize the loss function* $\mathcal{L}(\mathbf{W}, \mathbf{b})$. Or, expressed mathematically,

$$(\mathbf{W}, \mathbf{b})^* = \underset{\mathbf{W}, \mathbf{b}}{\operatorname{argmin}} \, \mathcal{L}(\mathbf{W}, \mathbf{b}). \tag{2.59}$$

Any Neumann boundary conditions are calculated by multiplying any Neumann force with the NN output on the Neumann boundary. To ensure that the displacement fulfills Dirichlet boundary conditions, the displacement ($\hat{\mathbf{u}}$) is calculated as a function of the NN output using the formula

$$\hat{\mathbf{u}}(\mathbf{X}; \mathbf{W}, \mathbf{b}) = \mathbf{A}(\mathbf{X}) + \mathbf{B}(\mathbf{X}, \hat{\mathbf{z}}^L(\mathbf{X}; \mathbf{W}, \mathbf{b})), \tag{2.60}$$

where $\mathbf{A}(\cdot)$ and $\mathbf{B}(\cdot)$ are two functions chosen so that the Dirichlet boundary conditions are forced onto the NN solution $\hat{\mathbf{z}}^L(\mathbf{X}; \mathbf{W}, \mathbf{b})$. For example, if we are solving a problem in which a three-dimensional beam is clamped on the face corresponding to $\mathbf{X}_1 = L$, we can constrain the solution using

$$\hat{\mathbf{u}} = (\mathbf{X}_1 - L) \cdot \hat{\mathbf{z}}^L,$$

which yields $\hat{\mathbf{u}} = \mathbf{0}$ when $\mathbf{X}_1 = L$.

One consequence of forcing the boundary conditions onto the solution after the NN calculation is that the problem becomes unconstrained, meaning (2.59) is enough without further constraints.

When calculating the energy in the system as a function of a three-dimensional deformation gradient, $\hat{\mathbf{F}}$, we get a three-dimensional energy functional. To calculate the total energy, we have to integrate the energy functional over all dimensions. Nguyen-Thanh et al. [41] compared three methods of numerical integration - *the mean rule*, *the trapezoidal rule*, and *Simpson's rule* - and concluded that the Simpson's method yielded the best results. Therefore, we have only used Simpson's rule in this thesis. The FEM uses Gauss quadrature for integration [45].

2.6.2 Implementation in PyTorch

Inspired by Nguyen-Thanh et al. [41], we have implemented the DEM using object-oriented programming using PyTorch. PyTorch is an open-source library for writing machine learning code in Python that focuses on the flexibility of NN model implementation. Moreover, this library provides a core data structure called the tensor, which shares many attributes with NumPy arrays, making it simple to learn. In addition, PyTorch has many features that help perform accelerated mathematical calculations on dedicated hardware. These advantages, combined with the ease of use it offers, have made PyTorch one of the leading NN tools, especially in the research community [53].

In our implementation, a superclass `DeepEnergyMethod` stores the NN model and the energy model specific to the problem. In this class, a method called `train_model()` is used to train the NN, and a special method `__call__()` constrains the solution on $\mathbf{X}_1 = 0$, because this was the Dirichlet boundary condition for all problems, except one. For each problem, we have defined a subclass of `DeepEnergyMethod`. These subclasses are used to evaluate the NN model using the method `evaluate_model()` after training.

```python
class DeepEnergyMethod:
    def __init__(...):
        # store NN and energy models

    def train_model(...):
        # train the NN model

    def __call__(...):
        # calculate and return constrained prediction
```

```python
class DeepEnergyMethodLV(DeepEnergyMethod):
    def __call__(...):
        # calculate and return model-specific constrained prediction

    def evaluate_model(...):
        # calculate NN output on test data
```

The NN models are generated in a class called `MultiLayerNet`, which takes the number of neurons in every layer as its input and generates an NN with weights and biases chosen from a normal probability distribution. Furthermore, the energy models we use are defined as separate classes, with each term in the models - such as the passive and active strain energy functions and compressibility - implemented as separate methods and the total energy calculated in a special method.

```
class GuccioneEnergyModel:
    def __init__(...):
        # store constitutive parameters

    def _get_passive_strain_energy(...):
        # calculate passive strain energy

    def _get_compressibility(...):
        # calculate penalty?/energy? for compressibility

    def _get_invariants(...):
        # calculate and return invariants used in energy model

    def __call__(...):
        # calculate invariants
        ..., ... = self._get_invariants(...)

        # calculate and return total strain energy
        StrainEnergy = (self._get_passive_strain_energy(...)
                     + self._get_compressibility(...))
        return StrainEnergy
```

```
class GuccioneTransverseEnergyModel(GuccioneEnergyModel):
    def __init__(...):
        # store constitutive parameters in superclass
        super().__init__(...)
        # store fiber direction

    def _get_active_strain_energy(...):
        # calculate and return active strain energy

    def _get_invariants(...):
        # calculate inavariants from superclass
        ...,  _ = super()._get_invariants(...)
        # calculate and return invariants specific for transversely isotropic model
```

```
class MultiLayerNet(nn.Module):
    def __init__(self, *neurons):
        super(MultiLayerNet, self).__init__()
        # throw error if depth < 3
        if len(neurons) < 3:
            raise Exception('You have to provide at least three layes!')
        # store layers in list
        self.linears = nn.ModuleList([nn.Linear(neurons[i-1], neurons[i])
                                      for i in range(1, len(neurons))])
    def forward(self, x):
        for layer in self.linears:
            x = torch.tanh(layer(x))
        return x
```

To create an NN model to train on a left ventricle geometry with the transversely isotropic energy model, we can use the following code

```
model = MultiLayerNet(...)                          # define NN model
energy = GuccioneTransverseEnergyModel(...)         # define energy model
DemLV = DeepEnergyMethodLV(model, energy)           # instantiate DEM object
DemLV.train_model(...)                              # train model
U_pred = DemLV.evaluate_model(...)                  # evaluate NN model
```

The training domain for each problem with its boundary conditions is generated in a stand-alone function called `define_domain()`. In addition, we have defined functions for calculating the mean value of the energy, the L^2-norm of an array, and the mean squared error, the latter of which we have used to calculate the error in the Dirichlet boundary conditions.

CHAPTER 3

NUMERICAL EXPERIMENTS

This chapter will introduce the problems solved in this thesis. The first section introduces the toy problem used to verify our implementation of the FEM and the example problems used to evaluate the performance of the NNs with different hyperparameters. The second section introduces benchmark problems for cardiac software. Finally, in the third section, we explain our implementation of the problems solved in this thesis and explain the challenges we faced during this project.

3.1 Introductory Tests

3.1.1 1D Bar

Inspired by Nguyen-Thanh et al. [41], we start by studying the problem of the one-dimensional bar under traction and compare our DEM and FEM results to the analytical solution. The problem is illustrated in Figure 3.1. In this problem, a one-dimensional bar is subject to a traction force $f(X)$. The material coordinates, X, span the interval $\Omega = [-1, 1]$. The bar is clamped on its left side, resulting in a homogenous Dirichlet boundary condition. Its right side is free, and for mathematical consistency, we will model it as subject to a traction force $\bar{t} = 0$, giving us a homogeneous Neumann boundary condition. The strain energy

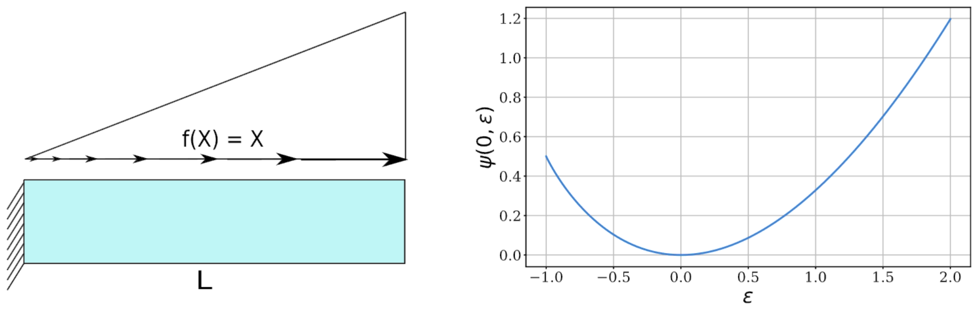


Figure 3.1: Illustration of the 1D bar problem. **Left:** the bar with the force f. **Right:** the plot of the energy, Ψ as a function of the deformation gradient ε. Adapted from *A surrogate model for computational homogenization of elastostatics at finite strain using high-dimensional model representation-based neural network* by Nguyen-Thanh et al., 2019 [42] under CC BY-NC 4.0.

function of the system is

$$\Psi(\varepsilon) = (1+\varepsilon)^{3/2} - \frac{3}{2}\varepsilon - 1, \tag{3.1}$$

where $\varepsilon = du/dX$ is the displacement gradient. A potential energy form of the problem is

$$\Pi = \int_\Omega \Psi(\varepsilon)dX - \int_\Omega f(X)u(X)\,dX - [\bar{t}u(X)]_{X=1}. \tag{3.2}$$

The strong form of the problem is

$$\frac{d}{dX}\frac{d\Psi}{d\varepsilon} + f(X) = 0, \qquad -1 \leq X \leq 1, \tag{3.3}$$

with the boundary conditions

$$u(-1) = 0, \tag{3.4}$$

$$\frac{d\Psi(\varepsilon)}{d\varepsilon}|_{X=1} = \bar{t}. \tag{3.5}$$

To satisfy the Dirichlet boundary condition, the constrained solution is chosen to be

$$\hat{u}(X) = (X+1)\cdot \hat{z}^L(X;\mathbf{W},\mathbf{b}). \tag{3.6}$$

Inserting $X=-1$ in (3.6), we get $\hat{u}(-1)=0$.

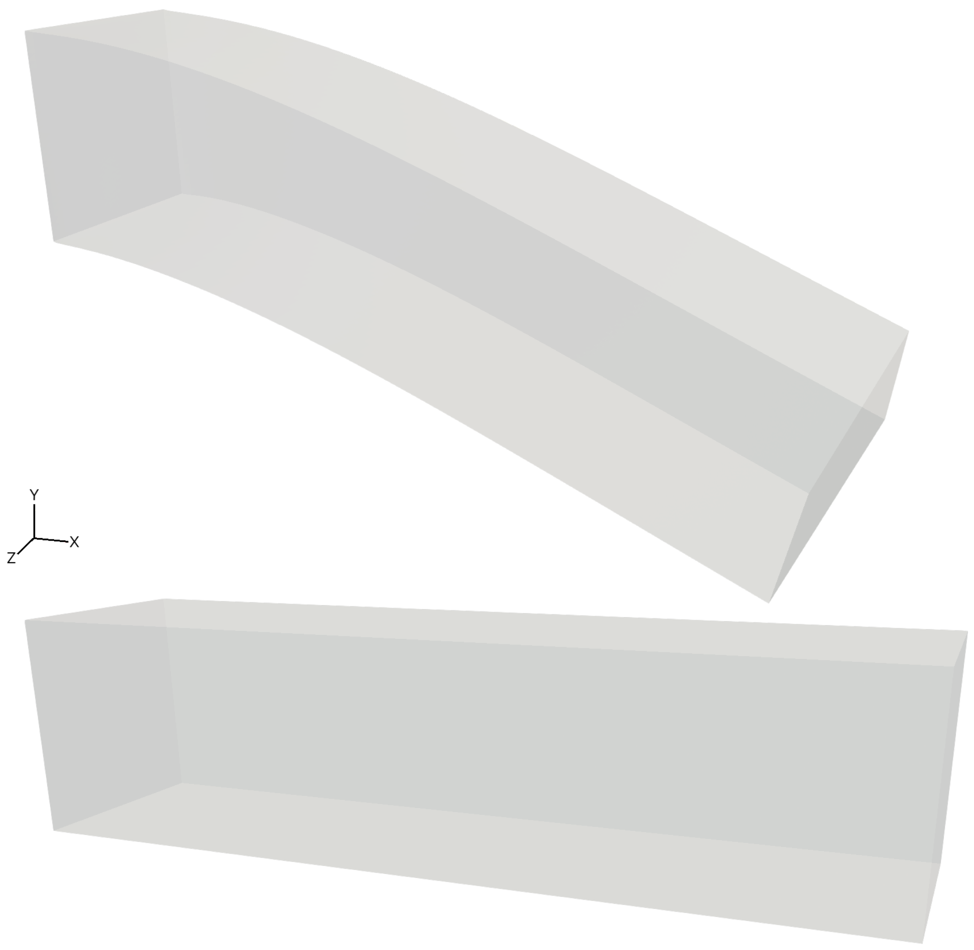


Figure 3.2: Illustration of the cantilever beam problem. **Bottom:** the undeformed beam. **Top:** an example solution.

3.1.2 Cantilever Beam

Another example solved in Nguyen-Thanh et al. [41] was a three-dimensional cantilever beam. The undeformed beam and an example solution are shown in Figure 3.2. The beam has dimensions $L = 4\text{m}$, $H = 1\text{m}$, $D = 1\text{m}$. This is a beam that has its left face clamped, resulting in a homogeneous Dirichlet boundary condition, as in the previous problem. Nguyen-Thanh et al. modeled the beam as subject to a uniform traction force on its right face, which gave them a Neumann boundary condition on this face. To test the DEM on a new problem, we replaced this boundary force with a uniform body force $\mathbf{B}$=(0, -5, 0)N. The PDE representing this problem is the PDE we used to derive the weak formulation in section 2.5.1

$$-\nabla \cdot \mathbf{P} = \mathbf{B}. \tag{3.7}$$

The energy model used in this example is the neo-Hookean model for a compressible material

$$\mathbf{\Psi}(I_1, J) = \frac{1}{2}\lambda[\ln(J)]^2 - \mu\ln(J) + \frac{1}{2}\mu(I_1 - 3), \tag{3.8}$$

with $J = \det(\mathbf{F})$ and $I_1 = \operatorname{tr}(\mathbf{F}^T\mathbf{F})$, where $\operatorname{tr}(\cdot)$ is the *trace*, defined as the sum of the diagonal elements of an input matrix. The parameters λ and μ are material constants called Lamé parameters and are defined as

$$\begin{aligned} \lambda &= \frac{E\nu}{(1+\nu)(1-2\nu)}, \\ \mu &= \frac{E}{2(1+\nu)}, \end{aligned} \tag{3.9}$$

where E is Young's modulus and ν is Poisson's ratio. In this example, we used the values $E = 1000\text{Pa}$ and $\nu = 0.3$, which gave us the parameter values $\lambda = 576.9\text{Pa}$ and $\mu = 384.6\text{Pa}$. The first two terms in (3.8) account for any change in the volume of the body, while the last term is a mathematical model for nonlinear deformation [21]. The boundary conditions, expressed mathematically, are

$$\begin{aligned} \mathbf{u} &= \mathbf{0}, & \mathbf{X}_1 &= 0, \\ \mathbf{P}\cdot\mathbf{N} &= \mathbf{0}, & \mathbf{X}_1 &\neq 0. \end{aligned}$$

From section 2.5.1, we get the weak formulation of the problem with the boundary conditions

$$\int_\Omega \mathbf{P} : \nabla\mathbf{v}\, dV - \int_\Omega \mathbf{B}\cdot\mathbf{v}\, dV - \int_{\partial\Omega} \mathbf{T}\cdot\mathbf{v}\, dS = \mathbf{0}. \tag{3.10}$$

Inserting for $\mathbf{T} = \mathbf{0}$, we get

$$\int_\Omega \mathbf{P} : \nabla\mathbf{v}\, dV - \int_\Omega \mathbf{B}\cdot\mathbf{v}\, dV = \mathbf{0}. \tag{3.11}$$

In the DEM, the constrained solution for this problem is

$$\begin{aligned} \hat{\mathbf{u}}_1(\mathbf{X}) &= \mathbf{X}_1 \cdot \hat{z}_1^L(\mathbf{X}; \mathbf{W}, \mathbf{b}), \\ \hat{\mathbf{u}}_2(\mathbf{X}) &= \mathbf{X}_1 \cdot \hat{z}_2^L(\mathbf{X}; \mathbf{W}, \mathbf{b}), \\ \hat{\mathbf{u}}_3(\mathbf{X}) &= \mathbf{X}_1 \cdot \hat{z}_3^L(\mathbf{X}; \mathbf{W}, \mathbf{b}), \end{aligned} \tag{3.12}$$

which yields $\hat{\mathbf{u}} = (0, 0, 0)$ for $\mathbf{X}_1 = 0$, which corresponds to the left face of the beam.

In this example, we ran 4 configurations of parameters when training Table 3.1 and Table 3.2 contain the parameters used in these runs.

Parameter	Run 1	Run 2
Learning rate	0.1	0.001, 0.01, 0.1, 0.5
Nr. of epochs	300	300
Nr. of points	30	30
Nr. of hidden neurons	30, 40, 50, 60	60
Nr. of hidden layers	2, 3, 4, 5	2, 3, 4, 5

Table 3.1: The parameters used for run 1 and run 2 in the cantilever beam problem.

Parameter	Run 3	Run 4
Learning rate	0.01, 0.05, 0.1, 0.5	0.01, 0.05, 0.1, 0.5
Nr. of epochs	300	300
Number of points	30	20, 30, 40, 50, 60
Number of hidden neurons	20, 30, 40, 50	50
Number of hidden layers	3	3

Table 3.2: The parameters used for run 3 and run 4 in the cantilever beam problem.

3.1.3 Contracting Cube

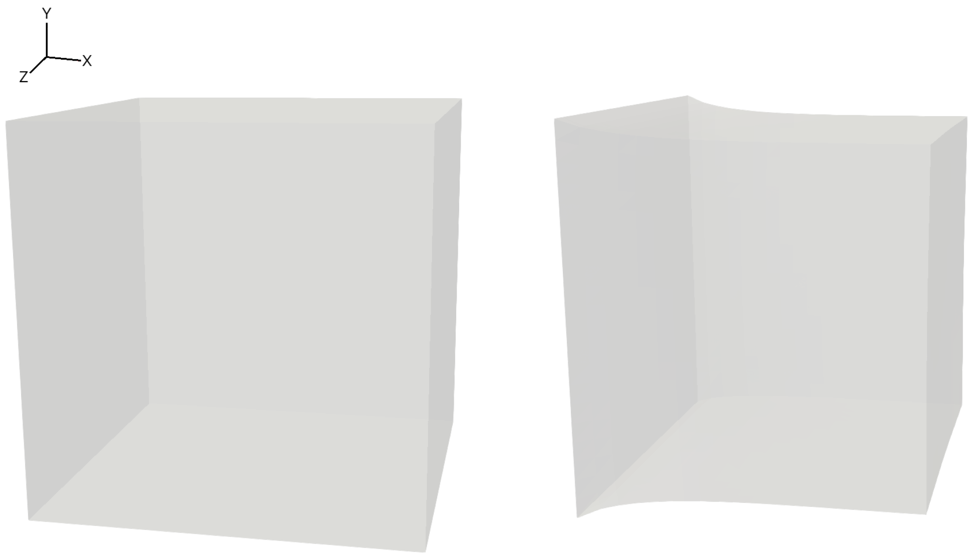


Figure 3.3: Illustration of contracting cube problem. **Left:** the undeformed cube. **Right:** an example solution.

The last example problem we solved is a three-dimensional cube subjected to active stress. The material model in this problem represents heart tissue on a simplified geometry. The undeformed cube and an example solution are shown in Figure 3.3. The cube has dimensions $L = 1\text{m}$, $H = 1\text{m}$, $D = 1\text{m}$. Similar to the previous problem, this cube has its left face clamped, resulting in a homogeneous Dirichlet boundary condition. On the right face, we have a non-homogeneous Neumann boundary condition in the form of a traction force, $\mathbf{T} = (-0.5, 0, 0)\text{N}$. There is no body force in this problem. For the passive strain energy function, we use the model for nonlinear deformation in (3.8), and for the active, we use the energy model presented in (2.23). To account for incompressibility, we use the penalty method, with the model introduced in section 2.1.4. The strain energy function for the system becomes

$$\boldsymbol{\Psi}(I_1, I_{4\mathbf{f}_0}, J) = \frac{\mu}{2}(I_1 - 3) + \frac{T_a}{2J}(I_{4\mathbf{f}_0} - 1) + \frac{\kappa}{2}(J-1)^2. \tag{3.13}$$

In this example, we use $\kappa = 1000$, and $\mu = 263.2\text{Pa}$, calculated with the second line in (3.9), is calculated using $E = 1000\text{Pa}$ and $\nu = 0.9$. The boundary conditions can be expressed mathematically as

$$\begin{aligned} \mathbf{u} &= \mathbf{0}, \qquad \mathbf{X}_1 = 0, \\ \mathbf{P}\cdot\mathbf{N} &= \mathbf{T}, \qquad \mathbf{X}_1 = 1. \end{aligned}$$

Inserting $\mathbf{B} = \mathbf{0}$ into (2.51), we get the weak formulation of the problem

$$\int_\Omega \mathbf{P} : \nabla\mathbf{v}\, dV - \int_{\partial\Omega} \mathbf{T}\cdot\mathbf{v}\, dS = \mathbf{0}. \tag{3.14}$$

In the DEM, the constrained solution for this problem is

$$\begin{aligned} \hat{\mathbf{u}}_1(\mathbf{X}) &= \mathbf{X}_1 \cdot \hat{z}_1^L(\mathbf{X}; \mathbf{W}, \mathbf{b}), \\ \hat{\mathbf{u}}_2(\mathbf{X}) &= \mathbf{X}_1 \cdot \hat{z}_2^L(\mathbf{X}; \mathbf{W}, \mathbf{b}), \\ \hat{\mathbf{u}}_3(\mathbf{X}) &= \mathbf{X}_1 \cdot \hat{z}_3^L(\mathbf{X}; \mathbf{W}, \mathbf{b}), \end{aligned} \tag{3.15}$$

which yields $\hat{\mathbf{u}} = (0, 0, 0)$ for $\mathbf{X}_1 = 0$, which corresponds to the left face of the cube.

In this example, we ran 3 configurations of parameters when training. The configurations are presented in Table 3.3.

Parameter	Run 1	Run 2	Run 3
Learning rate	0.1	0.001, 0.01, 0.1, 0.5	0.01, 0.05, 0.1, 0.5
Nr. of epochs	100	100	100
Nr. of points	40	40	20, 30, 40, 50, 60
Nr. of hidden neurons	30, 40, 50, 60	60	50
Nr. of hidden layers	2, 3, 4, 5	2, 3, 4, 5	3

Table 3.3: The parameters used for the contracting cube problem.

3.2 Cardiac Verification Benchmark

In the problems presented by Land et al [30], the strong form of the PDE, in the current configuration, can be stated as

$$\nabla \cdot \boldsymbol{\sigma} = 0, \tag{3.16}$$

where $\boldsymbol{\sigma}$ is the Cauchy stress tensor introduced in section 2.1.1. This equation is the same one we derived as (2.7) with no body force. The stress tensor relates to the strain energy function through the relation

$$J\mathbf{F}^{-1}\boldsymbol{\sigma}\mathbf{F}^{T} = \mathbf{S} = \frac{\partial \boldsymbol{\Psi}}{\partial \mathbf{E}}, \tag{3.17}$$

where $\mathbf{F}$ is the deformation gradient, $J = \det(\mathbf{F})$, $\mathbf{S}$ is the second Piola-Kirchhoff stress tensor, $\boldsymbol{\Psi}$ is the passive strain energy function, and $\mathbf{E}$ is the Green-Lagrange strain tensor. The first and second Piola-Kirchhoff stress tensors are related by the relation

$$\mathbf{P} = \mathbf{F}\mathbf{S}. \tag{3.18}$$

The passive strain energy function in these problems is the model by Guccione et al [17] introduced in section 2.2.1. In addition to the problem description above, the material is modeled as nearly incompressible with the constraint

$$J = 1. \tag{3.19}$$

In all three problems, we used the penalty method for incompressibility as introduced in section 2.1.4 to penalize solutions in which J deviated from 1.

In their paper, Land et al. [30] calculate the strain (S_i) at point $\mathbf{X}^i$ in three dimensions using the formula

$$S_i = \left(\frac{||\mathbf{x}_1^i - \mathbf{x}_2^i||}{|\mathbf{X}_1^i - \mathbf{X}_2^i||} - 1 \right) \times 100\%, \tag{3.20}$$

where $\mathbf{x}^i$ are points in the deformed configuration, and $\mathbf{X}^i$ are points in the original configuration. This formula measures the changes in distance between the points $\mathbf{X}_1^i$ and $\mathbf{X}_2^i$ in the undeformed geometry and their positions in the deformed geometry $\mathbf{x}_1^i$ and $\mathbf{x}_2^i$. In this thesis, we only calculate the strain in problem 1.

3.2.1 Problem 1

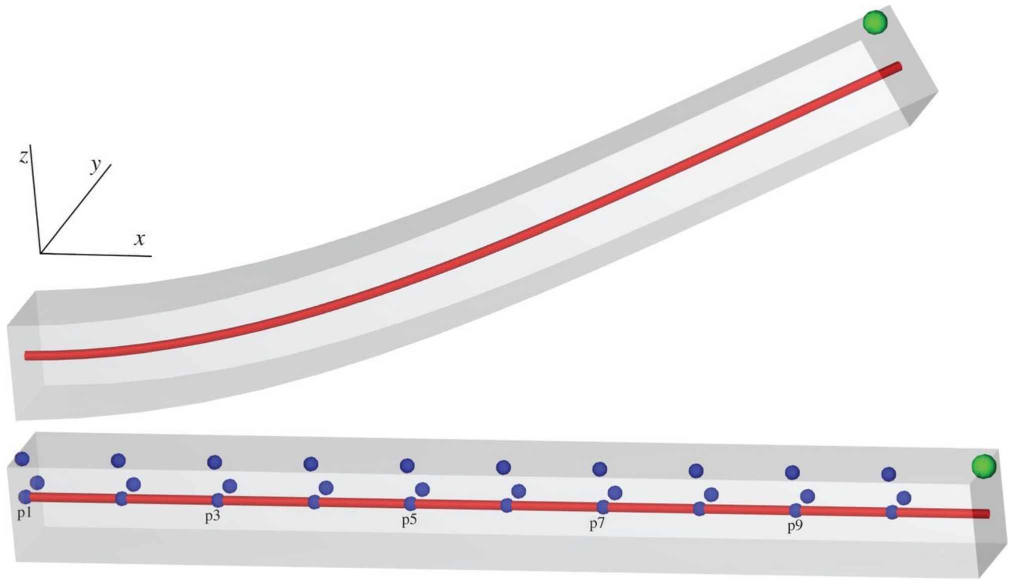


Figure 3.4: Illustration of problem 1. **Bottom:** the undeformed beam. **Top:** An example solution. The red line and the green point are used to measure the accuracy of the deformation. The blue points are used to measure the accuracy of the strain. Reprinted from *Verification of cardiac mechanics software: benchmark problems and solutions for testing active and passive material behavior* by Sander Land et al. under CC BY 4.0 License; source: https://royalsocietypublishing-org.ezproxy.uio.no/cms/asset/df6a0086-e671-4fa4-9e4c-2e431cc893c4/rspa20150641f01.jpg.

The first problem is a beam subjected to a constant force on one face. The problem and an example solution are shown in Figure 3.4. The beam has dimensions $L = 10$mm, $H = 1$mm, and $D = 1$mm. On the left face, we have a homogenous Dirichlet boundary condition; on the bottom, we have a Neumann boundary condition in the form of a pressure of $\mathbf{T} = (0, 0, 0.004)$ kPa.

Mathematically, the boundary conditions read

$$\begin{aligned} \mathbf{u} &= \mathbf{0}, & \mathbf{X}_1 &= 0, \\ \mathbf{P} \cdot \mathbf{N} &= \mathbf{T}, & \mathbf{X}_3 &= 0. \end{aligned}$$

In this problem, the constitutive parameters for the energy model are $C = 2$ kPA, $b_f = 8, b_t = 2, b_{fs} = 4$, and the fiber direction is $f_0 = (1, 0, 0)$. The constrained solution in the DEM for this problem is the same as for the introductory tests

$$\begin{aligned} \hat{\mathbf{u}}_1(\mathbf{X}) &= \mathbf{X}_1 \cdot \hat{z}_1^L(\mathbf{X}; \mathbf{W}, \mathbf{b}), \\ \hat{\mathbf{u}}_2(\mathbf{X}) &= \mathbf{X}_1 \cdot \hat{z}_2^L(\mathbf{X}; \mathbf{W}, \mathbf{b}), \\ \hat{\mathbf{u}}_3(\mathbf{X}) &= \mathbf{X}_1 \cdot \hat{z}_3^L(\mathbf{X}; \mathbf{W}, \mathbf{b}). \end{aligned} \tag{3.21}$$

3.2.2 Problem 2

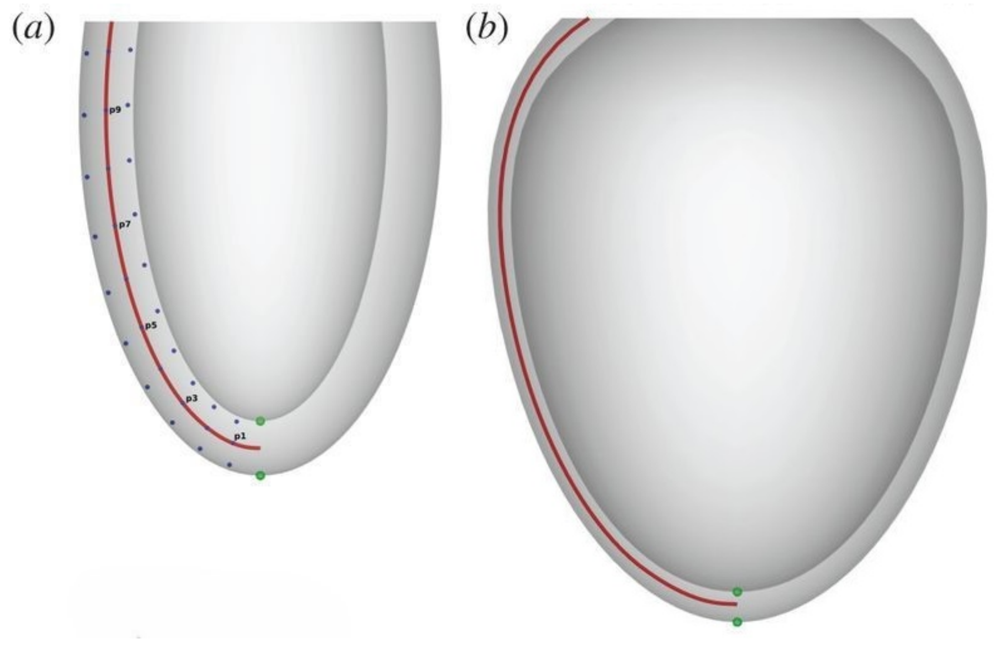


Figure 3.5: Illustration of problem 2. **Left:** the undeformed geometry viewed from the side. **Right:** an example solution. The red line and the green points are used to measure the accuracy of the deformation. The blue points are used to measure the accuracy of the strain. Adapted from *Verification of cardiac mechanics software: benchmark problems and solutions for testing active and passive material behavior* by Sander Land et al. under CC BY 4.0 License; source: https://royalsocietypublishing-org.ezproxy.uio.no/cms/asset/434abd77-0586-4032-8aaa-8b9d5e690f2c/rspa20150641f02.jpg.

The second problem deals with the inflation of an ellipsoid. The geometry is defined by the parametrization of a truncated ellipsoid

$$\mathbf{x} = \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} r_s \sin u \cos v \\ r_s \sin u \sin v \\ r_l \cos u \end{pmatrix}, \tag{3.22}$$

and is enclosed in the volume between the endocardium and epicardium. Table 3.4 gives the parameters used to define these surfaces. The ellipsoid is implicitly truncated at $z = 5$mm by the ranges for u. The undeformed geometry and an example solution are shown in Figure 3.5.

Surface	Material points	r_s	r_l	u	v
Endocardium	$\mathbf{X}_{\text{endo}}$	7	17	$[-\pi, \arccos \frac{5}{17}]$	$[-\pi, \pi]$
Epicardium	$\mathbf{X}_{\text{epi}}$	10	20	$[-\pi, \arccos \frac{5}{20}]$	$[-\pi, \pi]$

Table 3.4: The parameters used to define the endocardium and the epicardium in problems 2 and 3 in Land et al [30].

In this problem, the constitutive parameters for the energy model are $C = 10$ kPA, $b_f = 1, b_t = 1, b_{fs} = 1$, with isotropic material properties, meaning the fiber directions are not taken into account. The base plane, $z = 5$mm, is subject to a homogenous Dirichlet boundary condition, while the endocardial surface is subject to a Neumann boundary condition in the form of an outward pressure $\mathbf{T} = 10$kPa. The boundary conditions read mathematically

$$\begin{aligned} \mathbf{u} = \mathbf{0}, &\qquad \mathbf{X}_3 = 5, \\ \mathbf{P} \cdot \mathbf{N} = \mathbf{T}, &\qquad \mathbf{X} = \mathbf{X}_{\text{endo}}, \end{aligned}$$

and the constrained solution in the DEM reads

$$\begin{aligned} \hat{\mathbf{u}}_1(\mathbf{X}) &= (\mathbf{X}_3 - 5) \cdot \hat{z}_1^L(\mathbf{X}; \mathbf{W}, \mathbf{b}), \\ \hat{\mathbf{u}}_2(\mathbf{X}) &= (\mathbf{X}_3 - 5) \cdot \hat{z}_2^L(\mathbf{X}; \mathbf{W}, \mathbf{b}), \\ \hat{\mathbf{u}}_3(\mathbf{X}) &= (\mathbf{X}_3 - 5) \cdot \hat{z}_3^L(\mathbf{X}; \mathbf{W}, \mathbf{b}), \end{aligned} \tag{3.23}$$

which yields $\hat{\mathbf{u}} = (0, 0, 0)$ for $\mathbf{X}_3 = 5$, which corresponds to the base plane.

3.2.3 Problem 3

The last problem is an extension of the second and it deals with the inflation and contraction of an ellipsoid with transversely isotropic material properties. The

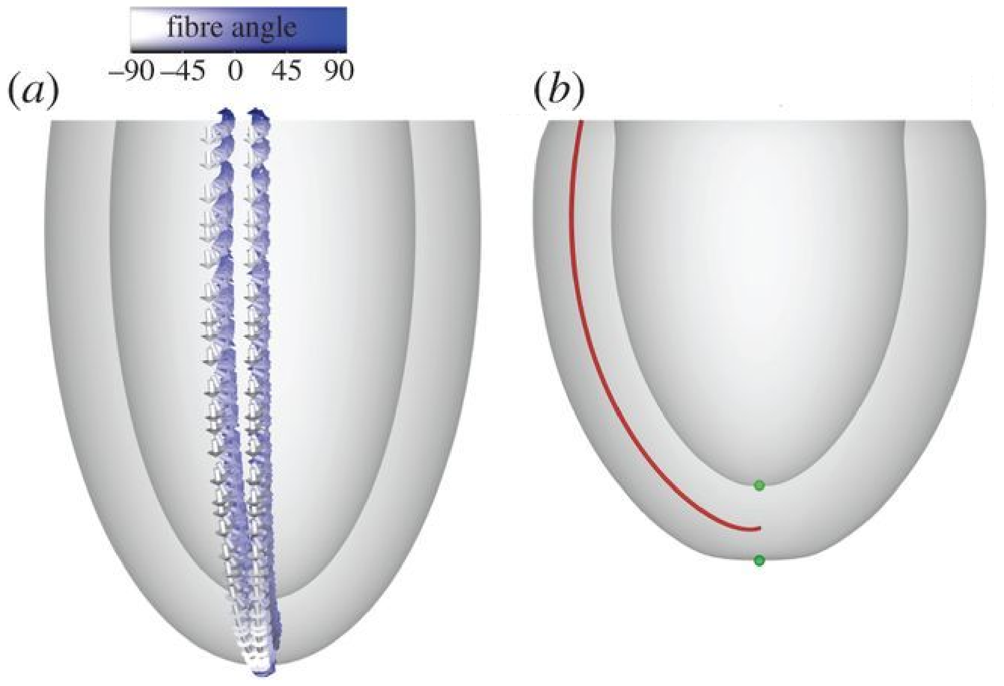


Figure 3.6: Illustration of problem 3. **Left:** the undeformed geometry viewed from the side. **Right:** an example solution. The red line and the green points are used to measure the accuracy of the deformation. Adapted from *Verification of cardiac mechanics software: benchmark problems and solutions for testing active and passive material behavior* by Sander Land et al. under CC BY 4.0 License; source: https://royalsocietypublishing-org.ezproxy.uio.no/cms/asset/434abd77-0586-4032-8aaa-8b9d5e690f2c/rspa20150641f02.jpg.

geometry and the Dirichlet boundary condition are the same as for problem 2. The constitutive parameters for this problem are $C = 2$kPa, $b_f = 8, b_t = 2, b_{fs} = 4$. The outward pressure on the endocardium is $\mathbf{T} = 15$kPa. In addition, the ellipsoid in this problem is subject to active stress $T_a = 60$kPa in the fiber direction. The fibers in this problem change from -90^o at the epicardium to 90^o at the endocardium. Instead of the observed fiber direction, introduced in section 1.2.1, these fiber directions are chosen for easier visualization of the implementation [30]. The boundary conditions read mathematically

$$\begin{aligned} \mathbf{u} &= \mathbf{0}, \qquad & \mathbf{X}_3 &= 5, \\ \mathbf{P} \cdot \mathbf{N} &= \mathbf{T}, \qquad & \mathbf{X} &= \mathbf{X}_{\text{endo}}, \end{aligned}$$

and the constrained solution again reads

$$\begin{aligned}\hat{\mathbf{u}}_1(\mathbf{X}) &= (\mathbf{X}_3 - 5) \cdot \hat{z}_1^L(\mathbf{X}; \mathbf{W}, \mathbf{b}),\\ \hat{\mathbf{u}}_2(\mathbf{X}) &= (\mathbf{X}_3 - 5) \cdot \hat{z}_2^L(\mathbf{X}; \mathbf{W}, \mathbf{b}),\\ \hat{\mathbf{u}}_3(\mathbf{X}) &= (\mathbf{X}_3 - 5) \cdot \hat{z}_3^L(\mathbf{X}; \mathbf{W}, \mathbf{b}).\end{aligned} \tag{3.24}$$

3.3 Implementation Details

This section will give a summary of the implementation details which were common throughout many of our runs. We will first explain how we chose to train the NNs, which hyperparameters we focused on during the training phase, and how we chose the best configurations to use on the problems from the cardiac verification benchmark. After this, we will show how we used the results from the test cases to solve the problems presented in the cardiac verification benchmark and how we evaluated our results. Finally, we will explain how we dealt with any obstacles.

3.3.1 Introductory Tests

Due to the lack of exact solutions for the more complex problems, we used the FEM solutions as a reference, against which we compare the DEM solutions. Therefore, we assume the FEM solutions to be good approximations to the true solutions. Because we have an exact solution for the one-dimensional bar problem, we focused in this example on verifying our FEM implementation. For this purpose, we perform an *order-of-accuracy test* to study the rate at which our solution approaches the exact solution. The order-of-accuracy test is the most rigorous criterion for verifying code implementation [44]. The purpose of this test is to examine whether or not the error in the approximation decreases at the theoretical rate as the mesh is refined. The theoretical rate can be found by analyzing the truncation error [44]. The rate of the error change of the approximation as the mesh is refined is called the convergence rate (r) and is expressed through the formula

$$r = \frac{\log(E_{i+1}/E_i)}{\log(h_{i+1}/h_i)}, \tag{3.25}$$

where E_i is the L^2-norm of the error at refinement level i, and h_i is the discretization size at this level. Furthermore, we compared the performance of the DEM to the FEM by measuring the absolute error of the DEM solutions in this problem. Because this example is one-dimensional, we plotted the error as a function of material points. By differentiating the solution, we got the deformation gradient, which we used to calculate the absolute error on the derivative of the exact solution.

When solving the three-dimensional example problems, we focused on testing different configurations of hyperparameters for NNs and evaluated the effect these had on the performance of the NN. For this purpose, we created a function, `train_and_evaluate()`, which takes all parameters we want to use during the training, instantiates an NN, and generates a training domain with the given parameters. This function then trains the NN on a training set and evaluates it on a test set. Following this, the NN approximated solution on the test set is compared to the reference solution, and an error is calculated. In addition, to minimize any stochastic bias in our results, we ran the training for every configuration 10 times and calculated the error as the average over the 10 runs.

```python
def train_and_evaluate(...):
    # arrays for storing L2-norms of error and losses
    u_norms = np.zeros((len(num_layers), len(num_neurons)))
    losses = torch.zeros((nr_losses, num_epochs, len(num_layers), len(num_neurons)))
    # energy model specific to the problem
    energy = NeoHookeanActiveEnergyModel(mu)
    # loop through the parameters
    for i, l in enumerate(num_layers):
        for j, n in enumerate(num_neurons):
            # instantiate NN model and generate domain
            model = MultiLayerNet(3, *([n]*l), 3)
            domain, dirichlet, neumann = define_domain(L, H, D, N=Ns)
            # instantiate DEM object
            DemCube = DeepEnergyMethodCube(model, energy)
            # train model
            DemCube.train_model(...)
            # evaluate model
            U_pred = DemCube.evaluate_model(x_test, y_test, z_test)
            # calculate and store L2-error
            u_norms[i, j] = L2norm3D(U_pred - u_fem20, N_test, N_test, N_test, dx, dy, dz)
            losses[:, :, i, j] = DemCube.losses
```

3.3.2 Choice of NN Architecture

Our idea is to study the performance of NNs on a couple of example problems and find out which parameters we should use to solve the problems in the cardiac verification benchmark. Therefore, we will use the three architectures that give us the best results in the cantilever beam and contracting cube problems as a metric of the performance of the DEM. Also, because this part is an evaluation of our results from the example problems, we will not be running repeated training of different architectures of NNs for these problems.

3.3.3 Computing the Error of the DEM Solution

After solving the problems using the DEM and the FEM, we needed a way to calculate the error of the DEM solution when compared to the solution using the FEM. To do this, we decided to compare the displacement in a given set of collocation points. Because we had used points in the domain as the training set, we needed to select points on which we had not trained our NN models. We chose to use the same number of points in x-, y- and z-directions as we had used for the FEM solution. We generated an array on the entire domain with two points more in each direction than the number of points used for the FEM solution. From these points, we picked out all except the first and the last points. Doing this ensured that we got the same number of points on the same domain without the training and evaluation sets sharing any elements.

```
x = np.linspace(0, L, N_test+2)[1:-1]
y = np.linspace(0, H, N_test+2)[1:-1]
z = np.linspace(0, D, N_test+2)[1:-1]
```

Furthermore, by using a function implemented in Nguyen-Thanh et al., we calculated the L^2-norm of the differences between the solutions in these points. In addition, to get the relative error, we divided this difference by the L^2-norm of the reference solution. The the L^2-norm of the error ($||e||_{L^2}$) between the DEM solution ($\mathbf{u}_{DEM}$) and the FEM solution ($\mathbf{u}_{FEM}$) can be expressed with the formula

$$\begin{aligned} ||e||_{L^2} &= \frac{||\mathbf{u}_{\text{DEM}} - \mathbf{u}_{\text{FEM}}||_{L^2}}{||\mathbf{u}_{\text{FEM}}||_{L^2}} \\ &= \frac{\sqrt{\int_\Omega (\mathbf{u}_{\text{DEM}} - \mathbf{u}_{\text{FEM}})^2 \, d\mathbf{X}}}{\sqrt{\int_\Omega \mathbf{u}_{\text{FEM}}^2 \, d\mathbf{X}}} \end{aligned} \tag{3.26}$$

```
def L2norm3D(U, Nx, Ny, Nz, dx, dy, dz):
    ### function from Nguyen-Thanh et al. ###
    Ux = np.expand_dims(U[0].flatten(), 1)
    Uy = np.expand_dims(U[1].flatten(), 1)
    Uz = np.expand_dims(U[2].flatten(), 1)
    Uxyz = np.concatenate((Ux, Uy, Uz), axis=1)
    n = Ux.shape[0]
    udotu = torch.zeros(n)
    for i in range(n):
        udotu[i] = np.dot(Uxyz[i,:], Uxyz[i,:].T)
    udotu = udotu.reshape(Nx, Ny, Nz)
    L2norm = np.sqrt(simpson(simpson(simpson(udotu, dx=dz), dx=dy), dx=dx))
    return L2norm
```

One consequence of dropping the boundary points was that the DEM solution we saved for visualization, which was the solution from the test set, did not have the boundary points in the domain. When comparing the solutions from the DEM and the FEM visually, there are therefore more discrepancies on the boundaries than there are in reality. This was not important, because our measure of the NN performance used the same points for both methods.

3.3.4 Challenges

3.3.4.1 Energy Becomes NaN in 1D Bar Problem

In the 1D example problem described above, our training was often cut short due to the energy function returning `NaN` during training. After debugging our code, we found that the problem arose from the NN solution becoming physically wrong during the training. The energy function for this problem, (3.1), only exists for $\varepsilon \geq -1$. Since ε in this case is the displacement gradient, if the NN switches the position of two points, the displacement gradient in this interval becomes negative. When this happened for a training step, the function calculating the energy could not calculate a value and returned `NaN`. As a solution, we wrote a code line in the energy function to handle this

```
def energy(u, x):
    # calculate the displacement gradient
    eps = grad(u, x, ...)
    # set values < -1 to -1
    eps[eps < -1] = -1
    # calculate and return the energy
    energy = pow(1 + eps, 3/2) - 3/2*eps - 1
    return energy
```

3.3.4.2 Compressible Model Gives Cantilever Beam Solution with Double Volume

In the example problem with the cantilever beam, we tried to implement a compressible material model by decomposing the energy function into volumetric and isochoric terms, as explained in section 2.1.3. After implementing this model in the FEM script, we saw that the solution became almost twice its original volume. After testing several model changes, we found that changing the volumetric term from

$$\boldsymbol{\Psi}_{\text{vol}}(J) = \frac{\lambda}{2}[\ln(J)]^2 - \mu \ln(J)$$

to

$$\boldsymbol{\Psi}_{\text{vol}}(J) = \frac{\kappa}{2}(J-1)^2$$

fixed this problem. One possible explanation for this is that the volumetric function used in Nguyen-Thanh et al. [41] penalizes compression much more than expansion, as illustrated in the top plot in Figure 3.7, while the substitute function we used is symmetric around $J = 1$. Moreover, the original function has a minimum at $J \approx 2$, which substantiates our explanation further. To keep the energy model for the problem as it originally was, we used $\mathbf{F}$ as it was, and did not decompose it.

3.3.4.3 Compressible Model Gives Wrong Solution Contracting Cube

As in the example problem with the cantilever beam, we tried to implement a compressible model in the example problem with a contracting cube. When writing the problem, we implemented the symmetrical volumetric function introduced in section 2.1.4, so we already had a penalty function that had worked with a decoupled strain energy function for the previous example. To model the cube as a compressible material, we then decomposed $\mathbf{F}$ into an isochoric and volumetric factor and used the isochoric factor, $J^{1/3}\mathbf{F}$, in the isochoric term in the strain energy function. With this change, the solution we got with the FEM was wrong. After debugging the code, we found that a cofactor matrix of $\mathbf{F}$, which translates the traction from the current configuration into the reference configuration, was the reason for this. This matrix had been removed from the code since it was deemed too difficult to implement in the DEM. After attempting, and failing, to implement the cofactor matrix a second time, we kept $\mathbf{F}$ as it was.

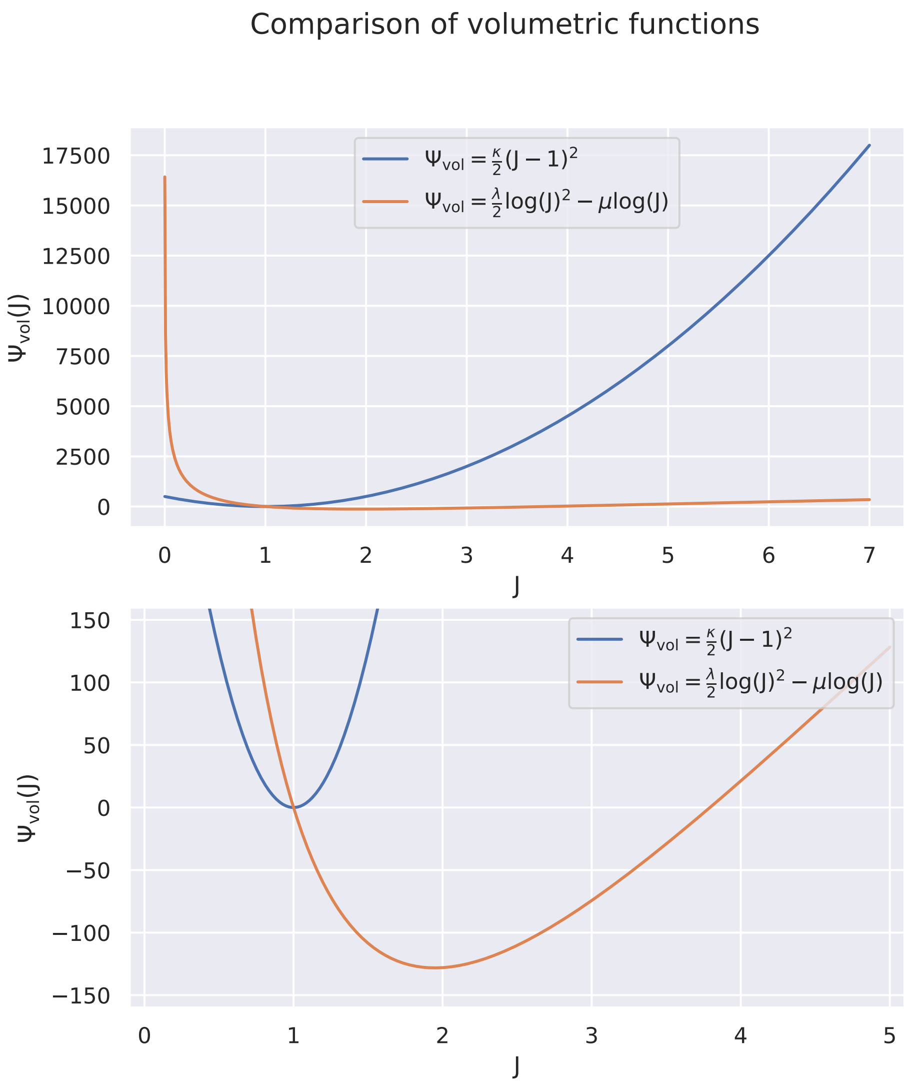


Figure 3.7: Plot of the two volumetric strain energy functions we tried using for the cantilever beam example problem. **Top:** The functions plotted against J. The function originally present in the energy model, shown here in orange, penalizes compression more than expansion, while the function we tested penalizes both equally. **Bottom:** a closer view of the bottom of both functions. The function originally in the model has a minimum at $J \approx 2$, while the one we tried using has its minimum at $J = 1$.

3.3.4.4 Early Stopping Did not Work

When training the NNs, we could not decide on any method for when to stop the training. Because the DEM focuses on minimizing energy, it seems natural to use it somehow. In most cases, the energy calculated when training the NN stabilized after 40-50 epochs. The changes in energy from epoch to epoch generally became smaller, with some fluctuations, as the training progressed. Since the energy for the

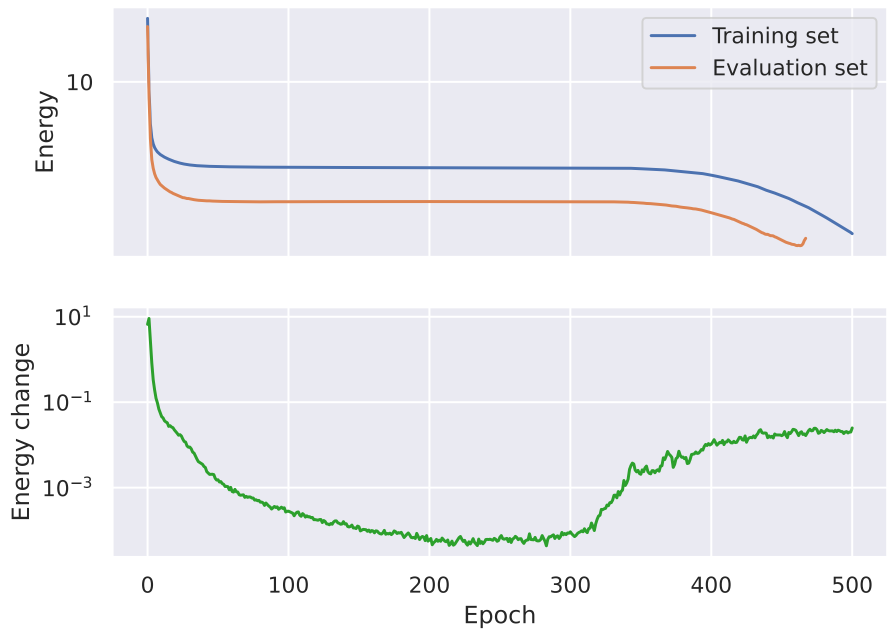


Figure 3.8: Plot of the energy and energy change during training. **Top:** the energy in the training and evaluation sets as a function of epochs. **Bottom:** The change in energy on the training set is calculated as the difference between the energy in the current epoch and the previous epoch. The energy is calculated as an average of over 20 rounds of training.

training set only decreased with time, we tried to implement early stopping using an evaluation set. The energy calculated on the evaluation set did start increasing at a certain point, but just after this happened, the energy often became `NaN`. This can be seen in the top plot in Figure 3.8. The energy on the evaluation set starts increasing at about epoch 460 but then disappears because we have the value `NaN` for some of the experiments. In addition, after around 200-300 epochs, the energy started suddenly dropping in magnitude and the changes to the energy became larger and larger. This is illustrated in the bottom plot in Figure 3.8. While studying the results, we found that the solution stabilized around the correct value for a while, but then diverged. This is due to the lack of regularization, mentioned in section 1.3.3. To counter this divergence, we stop the training at 300 epochs.

3.3.4.5 Integration in Problems 2 and 3 in Cardiac Verification Benchmark

When plotting the domain for problem 2 in the cardiac verification benchmark, we noticed that the distances between the points along the epicardium were non-uniform. Upon closer inspection, we found this to be the case for the endocardium as well as all points along any surface between the two. Figure 3.9 shows a cross-section of the myocardium model and Figure 3.10 shows the L^2-distances between the points starting from the bottom. Such non-uniform distancing of points poses a problem when integrating the energy. As a solution, we decided to use the points in the middle layer as the integration points. Doing this, the integration error is smaller than if we had used the points on the epicardium or the endocardium. To always have a layer in the middle, we will use only an odd-numbered number of layers.

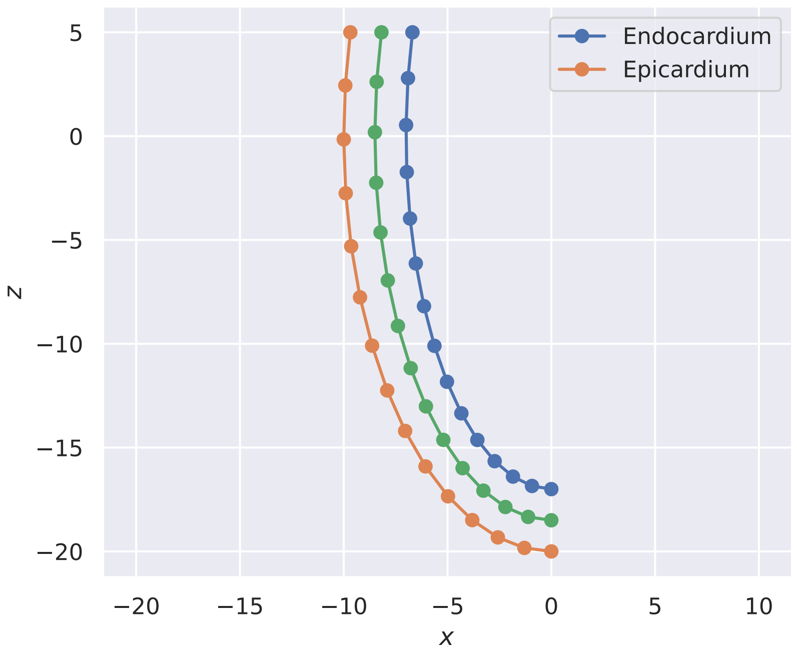


Figure 3.9: Cross-sectioned view of the myocardium model. This cross-section is generated using 15 points and 3 layers.

3.3.4.6 Non-Uniform Force Density in LV Geometry

When we solved problem 2 in the cardiac verification benchmark, we saw that the apex in our solution was pushed far down, while the ventricular wall had barely inflated. Upon inspection, we found that the density of collocation points gets higher the closer we get to the apex. This is illustrated in the left plot in Figure 3.11. This increase in density occurs because the ellipsoid gets narrower the closer we are to the apex. To counter this, we calculated the distances between pairs of points along a given height and used these distances to scale down the pressure vectors. The right plot in Figure 3.11 shows the scaled-down pressure vectors.

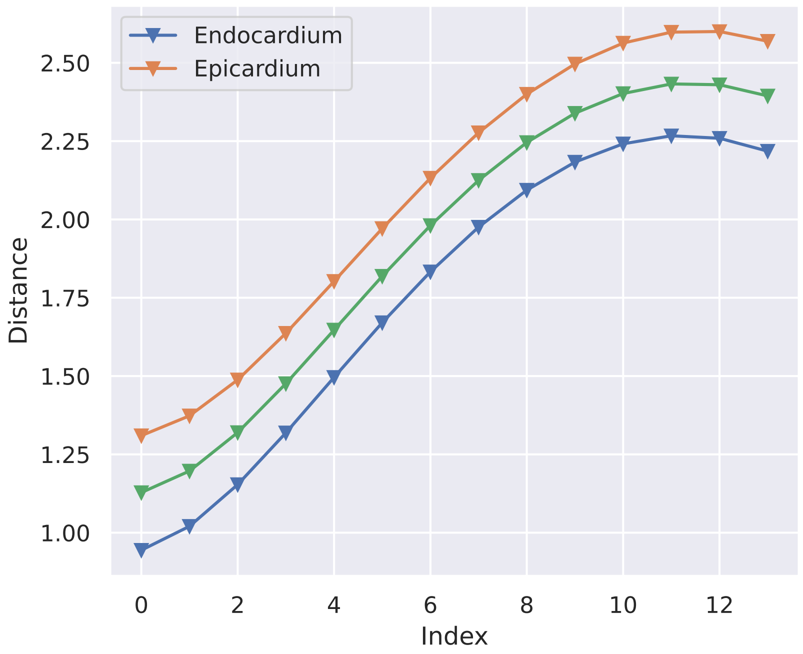


Figure 3.10: L^2-distances between points at index i and $i+1$, starting from the apex, on the endocardium and the epicardium, and a layer in the middle of the myocardium.

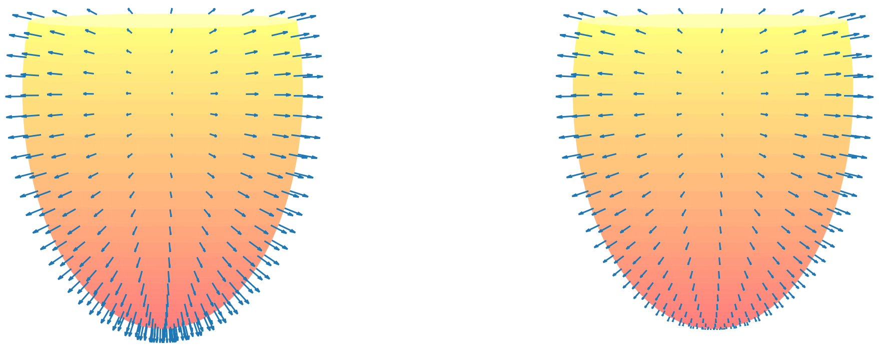


Figure 3.11: Illustration of pressure vector on the endocardium in problems 2 and 3 in cardiac verification benchmark. The density of vectors is higher around the apex than the endocardial wall. **Left:** original pressure vector view from the side. **Right:** scaled pressure vectors viewed from the side.

CHAPTER 4

RESULTS

4.1 Introductory Tests

4.1.1 1D Bar

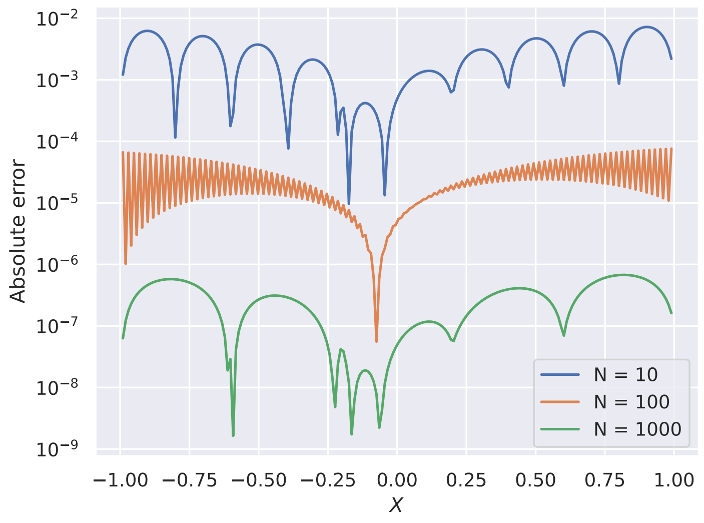


Figure 4.1: Absolute error of FEM solution as a function of material points, X.

Figure 4.1 shows the absolute error of the FEM solution as a function of material points X for different discretizations of the domain. The error is lower than 10^{-2} for even the roughest discretization, $N = 10$, and decreases by two orders of magnitude, to 10^{-4}, when the discretization increases to $N = 100$. The

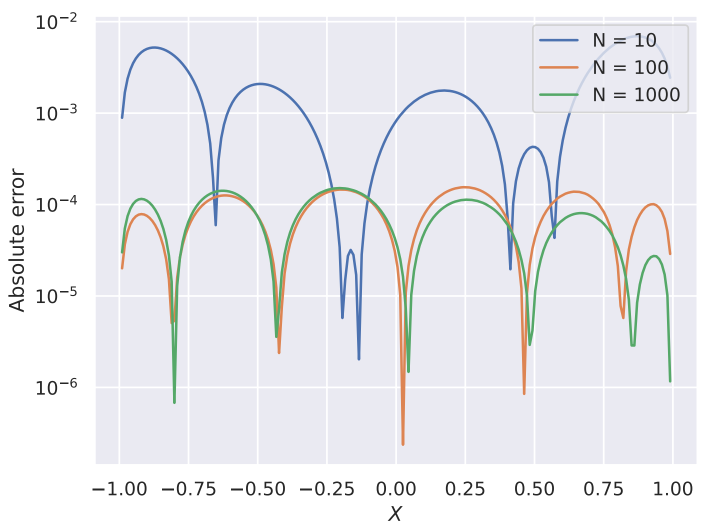


Figure 4.2: Absolute error of DEM solution as a function of material points, X.

N	r
10	-
50	2.000
100	2.021
500	1.998
1000	1.983
5000	1.997
10000	2.002

Table 4.1: Table with the convergence rate, r for the FEM solution, as a function of number of points on the domain N

same can be observed again when the discretization is increased by an order of magnitude a second time. These results hint at a convergence rate of 2 for the FEM solution and this is confirmed in Figure 4.7 and Table 4.1. When comparing Figure 4.2 to Figure 4.1, we see that a discretization of $N = 10$ and $N = 100$ yields comparable absolute errors with the DEM and the FEM. However, for $N = 1000$, the absolute error stays at 10^{-4}. Figure 4.3 shows the FEM and DEM solutions for $N = 100$ over the exact solution. Because the errors in both methods are so small throughout the domain, the solutions look identical.

In Figure 4.4 and Figure 4.5, we have plotted the absolute error on the deformation gradient as a function of material points X. Both the FEM and the DEM have an error of 10^{-1} with $N = 10$, but while the FEM error drops to 10^{-4} with $N = 100$, the DEM error only drops to 10^{-3}. In addition, the DEM does not drop further when the number of points is increased to 1000, while the

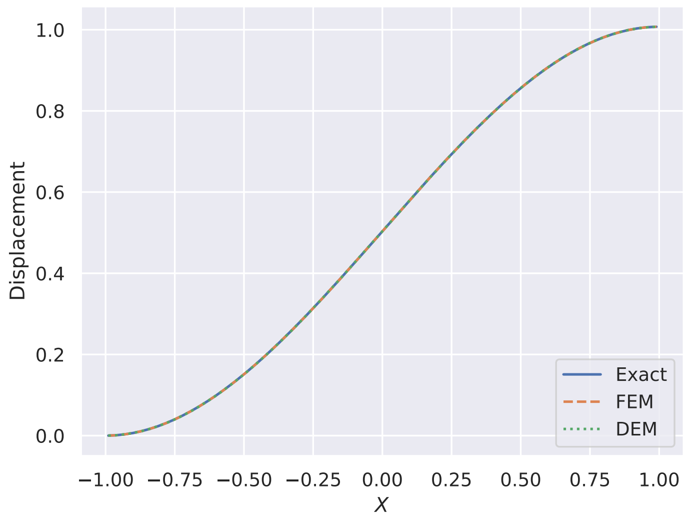


Figure 4.3: Plot of the exact solution with the FEM and DEM solutions. The FEM and the DEM solutions both used 100 points in the domain. The DEM was trained for 30 epochs with a learning rate of 0.1.

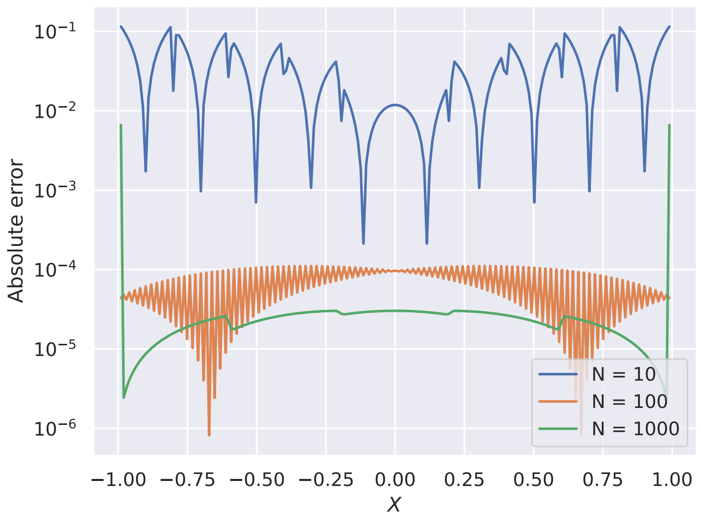


Figure 4.4: Absolute error of derivative of FEM solution as a function of material points, X.

FEM error drops by 1/3. Figure 4.6 shows the derivatives of the FEM and the DEM solutions for $N = 100$ over the exact deformation gradient. Finally, we have plotted the convergence rate for the FEM solutions presented in the figures above in Figure 4.7 and written out the values we got for each refinement step in table 4.1.

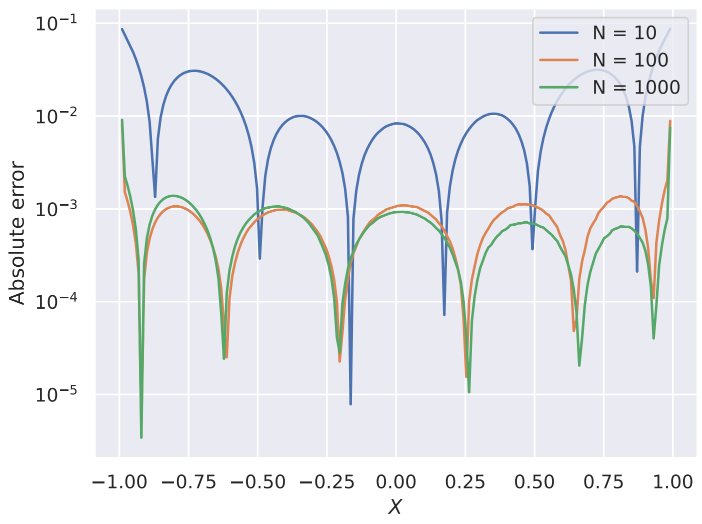


Figure 4.5: Absolute error of derivative of DEM solution as a function of material points, X.

4.1.2 Cantilever Beam

Model	Nr. of hidden layers	Nr. of hidden neurons	η
model 1	5	40	0.1
model 2	3	50	0.1
model 3	3	50	0.05

Table 4.2: Table of hyperparameter configuration that had the best performance on cantilever beam problem.

N	FEM Time [s]	DEM Training Time [s]	DEM Evaluation Time [s]
5	0.90	421.46	0.01
10	2.13	587.12	0.01
15	7.94	439.96	0.01
20	27.29	622.54	0.01
25	76.43	493.29	0.01
30	228.62	701.02	0.01
40	-	839.64	0.01
50	-	1151.45	0.01

Table 4.3: Timing measurements for the FEM solver and the training and evaluation of the DEM on the cantilever beam problem. The NN model used here is model 1 in Table 4.2. The times are calculated as an average over 4 runs.

In Figure 4.8, we have plotted the error as a function of the number of hidden layers and neurons in each hidden layer. Most of the errors measured in this

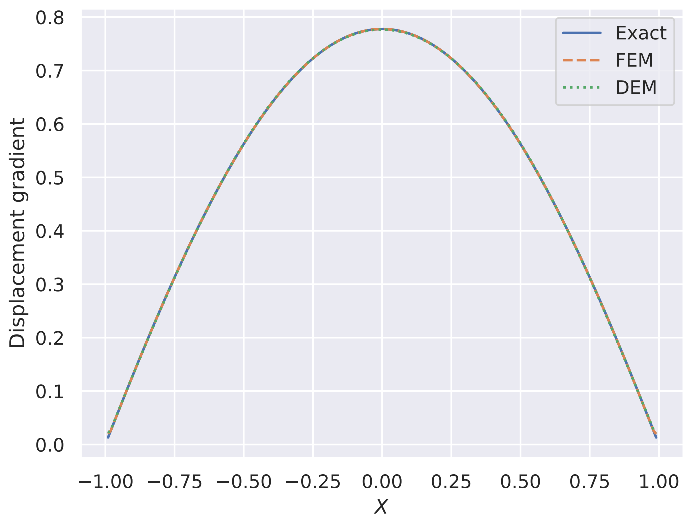


Figure 4.6: Plot of the exact deformation gradient with the derivatives of the FEM and DEM solutions. The FEM and the DEM solutions both used 100 points in the domain. The DEM was trained for 30 epochs with a learning rate of 0.1.

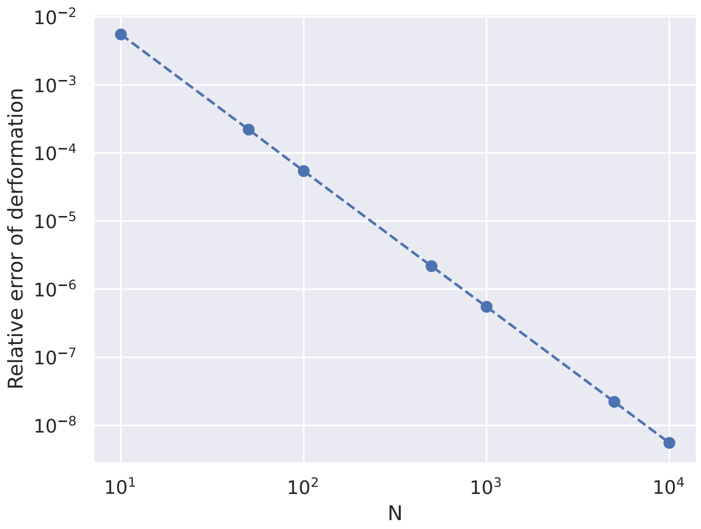


Figure 4.7: Plot of the convergence rate for FEM solution of 1D bar problem.

experiment are of a similar magnitude. The lowest error we got is $2 \cdot 10^{-3}$ with the combination of 3 hidden layers with 40 hidden neurons and $2.2 \cdot 10^{-3}$ with 4 hidden layers with 40 hidden neurons.

In the second experiment, we used a NN with 40 neurons in the hidden layers, while changing the learning rate, η, and the number of hidden layers. The L^2-norm of the errors is illustrated in Figure 4.9. The best performance in this case gave an L^2-norm of error $2 \cdot 10^{-3}$ with the configuration of a learning rate of 0.5 and 2

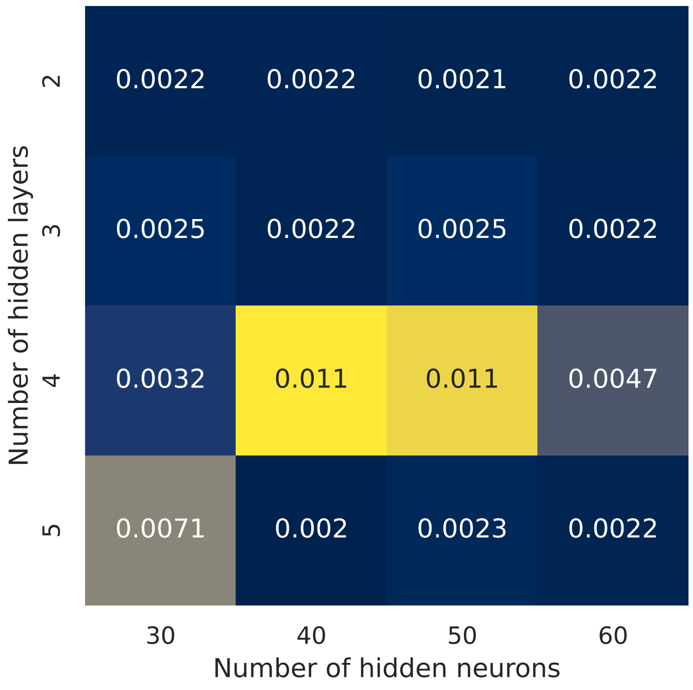


Figure 4.8: L^2-norm of error for cantilever beam example problem with N=30 and $\eta = 0.1$. The error is calculated as an average value over 10 runs.

hidden layers. This was also the only configuration that gave a good performance with this learning rate. The NN achieved its second-best performance of $2.1 \cdot 10^{-3}$ for five other configurations. Two of these configurations had a learning rate of 0.1 with 3 and 5 hidden layers, while the other three configurations had a learning rate of 0.05 with 3, 4, and 5 hidden layers.

Because a learning rate of 0.5 was so sensitive to the number of layers, we decided to test how the number of neurons affects the performance of an NN with a given learning rate. Figure 4.10 shows the results. A learning rate of 0.5 results in a much higher error than 0.1 and lower. In addition, for learning rates of 0.01 and 0.05, the performance is stable, whereas for a learning rate of 0.1, the performance is better for wider networks. The configuration of $\eta = 0.1$ and 50 hidden neurons performed best in this experiment, with an L^2-norm of error of $1.9 \cdot 10^{-3}$. The second best performance was with the configuration $\eta = 0.05$ and 50 hidden neurons, resulting in an L^2-norm of $2 \cdot 10^{-3}$.

Using these results, we ran a final experiment to study how the number of points we use to divide the domain during training affects the performance when using different learning rates. Figure 4.11 shows the results we got from this

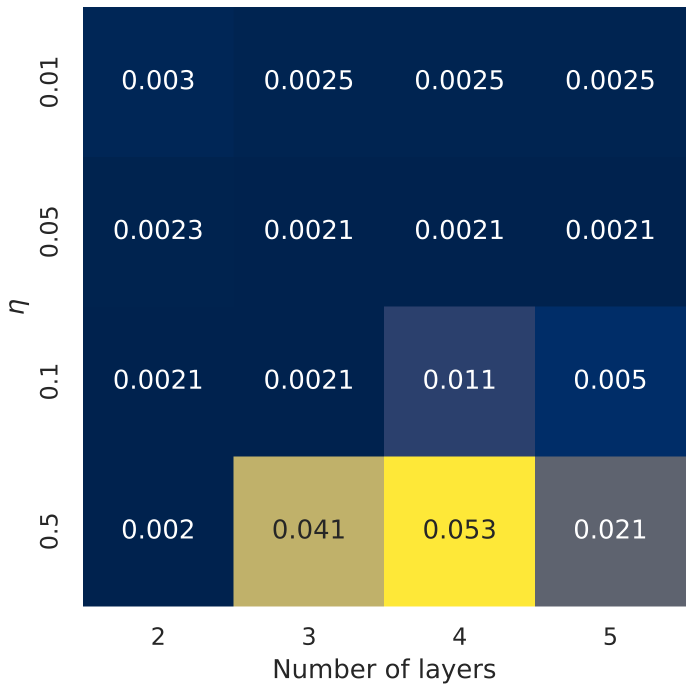


Figure 4.9: L^2-norm of error for cantilever beam example problem with N=30 and 40 neurons in each hidden layer. The error is calculated as an average value over 10 runs.

experiment. The error decreases from $N = 30$ to $N = 40$, and again to $N = 50$, but stays about the same for $N = 60$ as for $N = 50$. The highest errors are for $N = 30$ with learning rates $\eta = 0.5$ and $\eta = 0.1$.

In figure 4.12, we have plotted the deflection of the line in the middle of the beam, $\mathbf{X} = (x, 0.5, 0.5)$ using the FEM and the three NN models that performed best with the original number of points. The parameters used for these models are in Table 4.2. In the bottom plots, we can see that model 1 yields the best result out of the three. Model 1 starts slightly higher than the FEM solution on the left side and ends slightly lower than the FEM solution on the right side. In the middle of the beam, model 1 matches the FEM solution. The deformations from models 2 and 3 are lower than the FEM solution along the entire beam. In addition, these two models give an identical solution.

Finally, in Table 4.3, we have written out the time measurements of the FEM solver and the training and evaluation times for model 1 in Table 4.2 for different discretizations of the domain. In addition, we have plotted the measured times in Figure 4.13 and fitted functions to these measurements. In the left plot, we see that the FEM times are matched by an exponential function, whereas the DEM

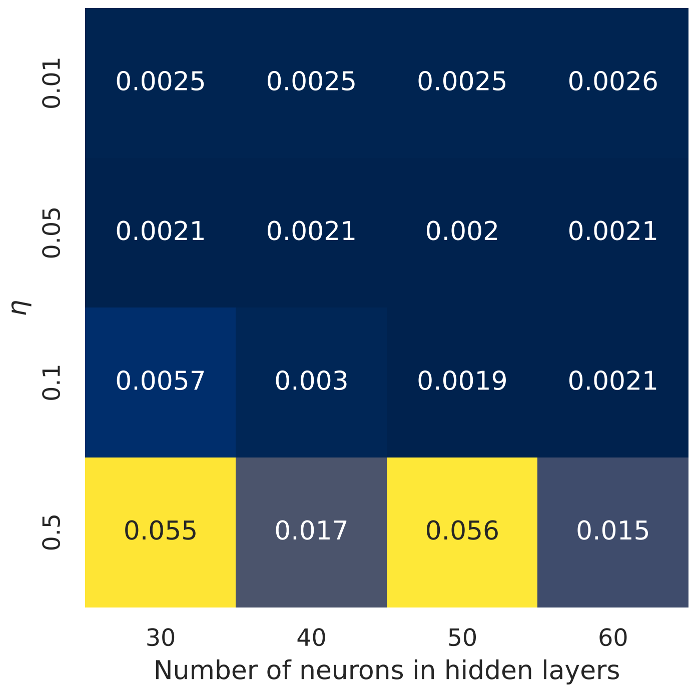


Figure 4.10: L^2-norm of error for cantilever beam example problem with N=30 and 3 hidden layers. The error is calculated as an average value over 10 runs.

times are matched by a polynomial function. The evaluation times for the DEM are constant.

4.1.3 Contracting Cube

Model	Nr. of hidden layers	Nr. of hidden neurons	η
model 1	4	20	0.1
model 2	3	30	0.1
model 3	3	40	0.5

Table 4.4: Table of hyperparameter configuration that had the best performance on contracting cube problem.

In Figure 4.14, we can see that the error is between $2.6 \cdot 10^{-2}$ and $1.6 \cdot 10^{-2}$. In this experiment, the performance is much more stable than it was for the previous problem, but the L^2-norms of the error are one order of magnitude higher. In addition, the highest error occurred for the configuration of 20 neurons in 5 hidden layers. After this, the highest error, $2.1 \cdot 10^{-2}$, was with the configuration of 30 neurons in 4 hidden layers. Otherwise, no error was higher than $1.9 \cdot 10^{-2}$. The

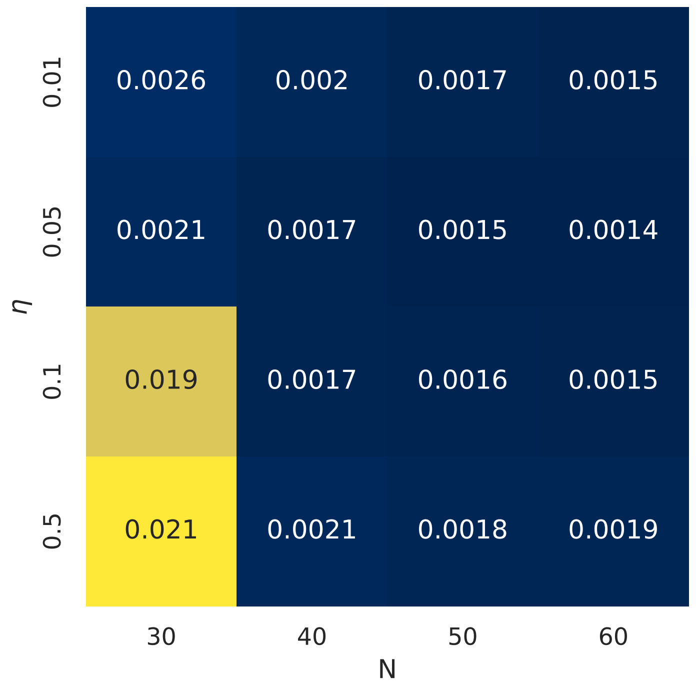


Figure 4.11: L^2-norm of error for cantilever beam example problem with 50 neurons in 3 hidden layers. The error is calculated as an average value over 10 runs.

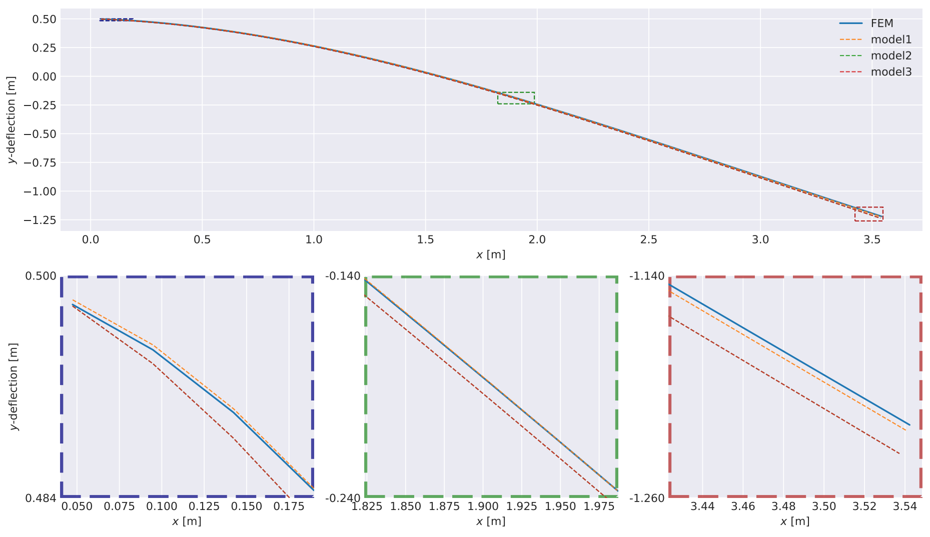


Figure 4.12: Deflection of a line through $\mathbf{X} = (x, 0.5, 0.5)$ using the FEM and three NN models. Parameters for NN models are written in Table 4.2.

best performance in this experiment was achieved for five configurations. The first and second configurations had 20 and 40 neurons in 4 hidden layers, the third

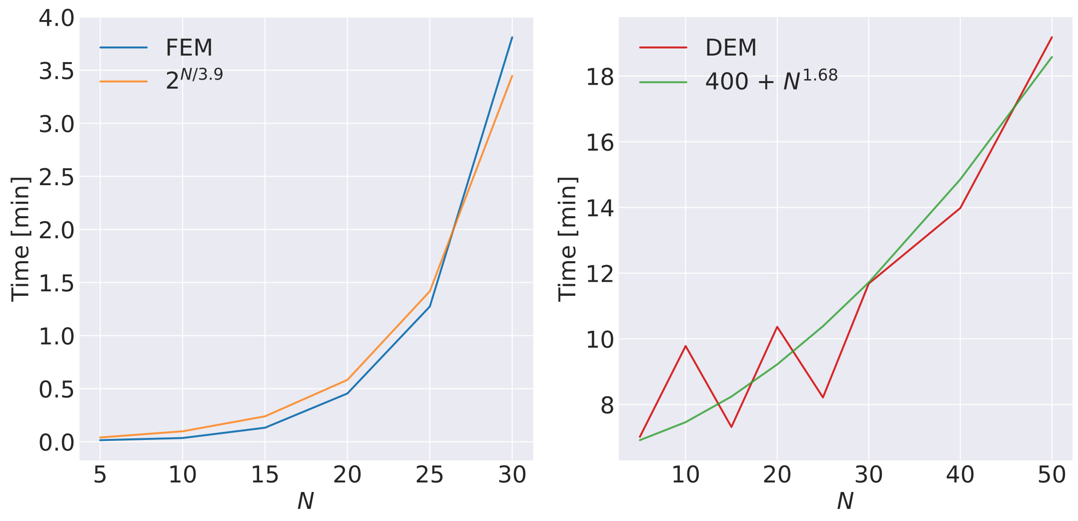
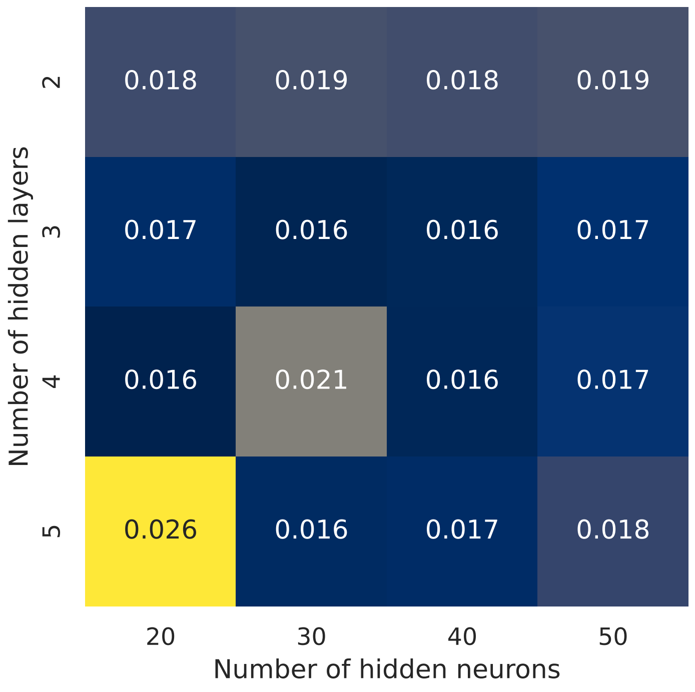


Figure 4.13: Plot of time measurements for the DEM and the FEM approximated with functions.



Number of hidden layers \ Number of hidden neurons	20	30	40	50
2	0.018	0.019	0.018	0.019
3	0.017	0.016	0.016	0.017
4	0.016	0.021	0.016	0.017
5	0.026	0.016	0.017	0.018

Figure 4.14: L^2-norm of error for example problem of cube subjected to active an active contraction with 40 points and $\eta = 0.1$. The error is calculated as an average value over 10 runs.

and fourth configurations had 30 and 40 neurons in 3 hidden layers, and the fifth configuration had 30 neurons in 5 hidden layers. The lowest error was $1.6 \cdot 10^{-2}$.

In the second experiment, we set the number of hidden layers to 3, since this

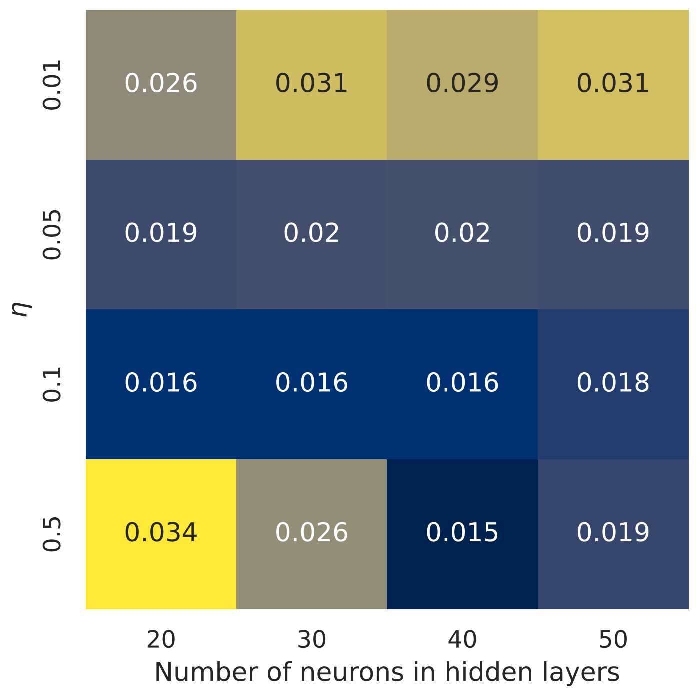


Figure 4.15: L^2-norm of error for error for example problem of cube subjected to active an active contraction with 40 points and 3 hidden layers. The error is calculated as an average value over 10 runs.

resulted in the best overall performance in the first experiment, and tested how the number of hidden neurons and the learning rate affect the performance. The parameters we used in this experiment are the same as the ones used in the second experiment in the cantilever beam problem. Figure 4.15 shows the L^2-norms of the errors for this experiment. Several configurations here get a higher error than in the last experiment, with the highest error being $3.4 \cdot 10^{-2}$ for the configuration of 20 neurons and a learning rate of 0.5. This learning rate got worse results than most configurations in this experiment. The configuration of $\eta = 0.5$ and 40 hidden achieved the best performance with an L^2-norm of $1.5 \cdot 10^{-2}$.

In the last experiment for this problem, we tested how the number of points used to divide the domain and the learning rate affect the performance. The results are illustrated in Figure 4.16. For $N = 20$, the performance is sensitive to the learning rate, with the L^2-norm for $\eta = 0.5$ over ten times the norm for $\eta = 0.01$. In addition, the performance does almost not change for $\eta = 0.01$. As the number of points in the domain increases, the errors become more similar. The highest error in this experiment is $3.7 \cdot 10^{-1}$. This error is more than ten times

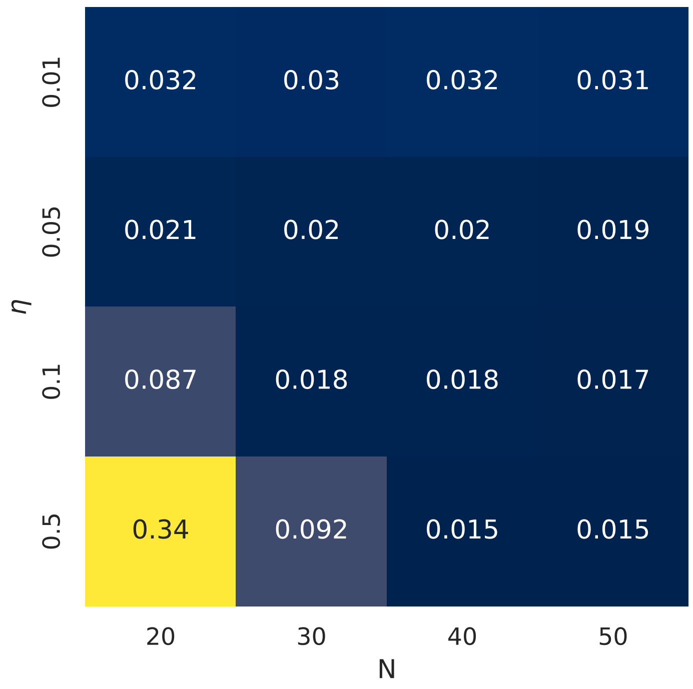


Figure 4.16: L^2-norm of error for error for example problem of cube subjected to active an active contraction with 40 neurons in 3 hidden layers. The error is calculated as an average value over 10 runs.

higher than the highest error in the second experiment. The lowest error was again $1.5 \cdot 10^{-2}$ and was achieved using a learning rate of 0.5 and $(50 \times 50 \times 50)$ and $6 \times 60 \times 60)$ points in the domain.

The result from the FEM is compared with the three NN models that achieved the best performance for this problem in Figure 4.17. The hyperparameters used for these models are in Table 4.4. In this figure, we have plotted the deflection of a line in the middle of the cube with coordinates $\mathbf{X} = (0.5, 0.5, z)$. This figure shows that model 1 performs better than models 2 and 3 at the top and the bottom of the line. Model 3 is the best in the middle, with model 2 just behind. Model 3 performs the worst along the entire line.

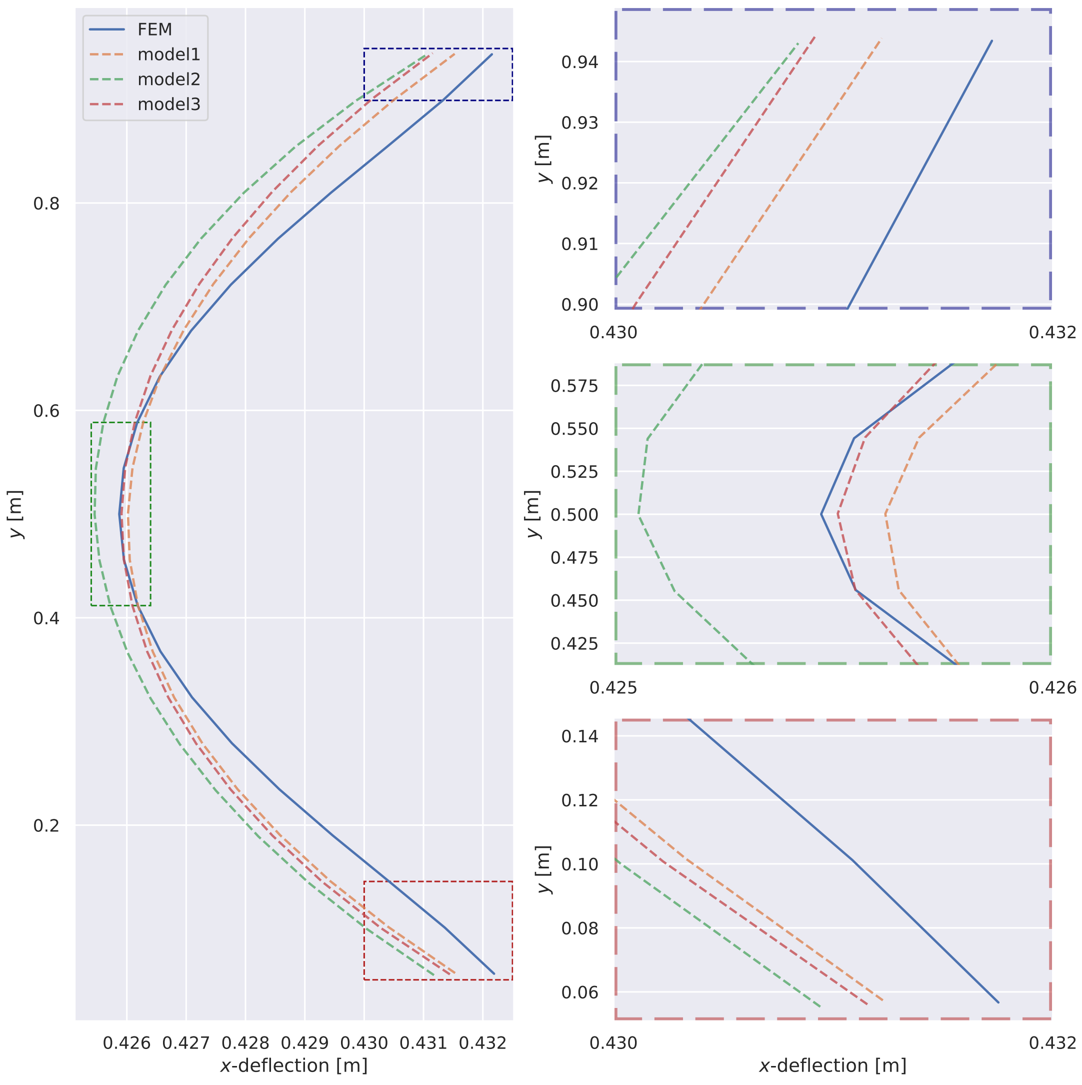


Figure 4.17: Deflection of line through $\mathbf{X} = (0.5, 0.5, z)$ using the FEM and three NN models. Parameters for NN models are written in Table 4.4.

4.2 Cardiac Verification Benchmark

4.2.1 Problem 1

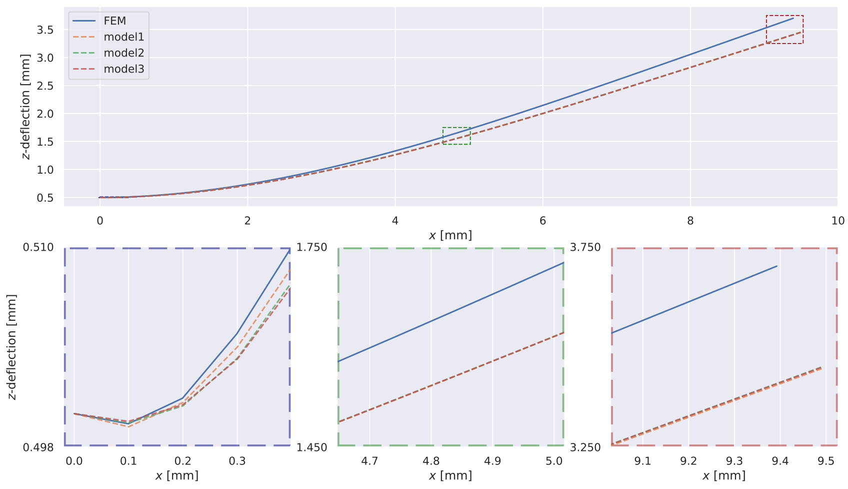


Figure 4.18: Z-deflection of a line through $\mathbf{X} = (x, 0.5, 0.5)$ for problem 1 using the FEM and three NN models. Parameters for NN models are written in Table 4.2.

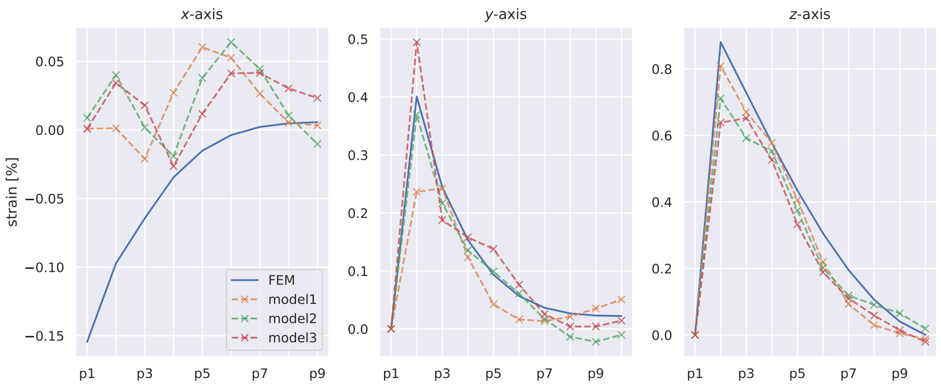


Figure 4.19: Strain plot for beam in problem 1 using the FEM and three NN models. Parameters for NN models are written in Table 4.2. The points $(p_1, ..., p_{10})$ correspond to the points in Figure 3.4.

Figure 4.18 shows the z-deflection of the line through $y = 0.5$ and $z = 0.5$. In this figure, we can see that the deflection is higher for the FEM than the DEM. Moreover, the NN solutions diverge more from the FEM solution the further to the right on the beam we measure. Model 1 is closer to the FEM solution on the left side, but the solution for all models is almost identical otherwise. In Figure

4.19, we have plotted the strain along the x-, y-, and z-axis. The strain in the y-axis and z-axis are similar to the stress calculated on the FEM solution, with model 1 performing the worst along the y-axis, and model 2 performing the best. All models perform equally along the z-axis. The strain in the x-axis is wrong for all NN models.

4.2.2 Problem 2

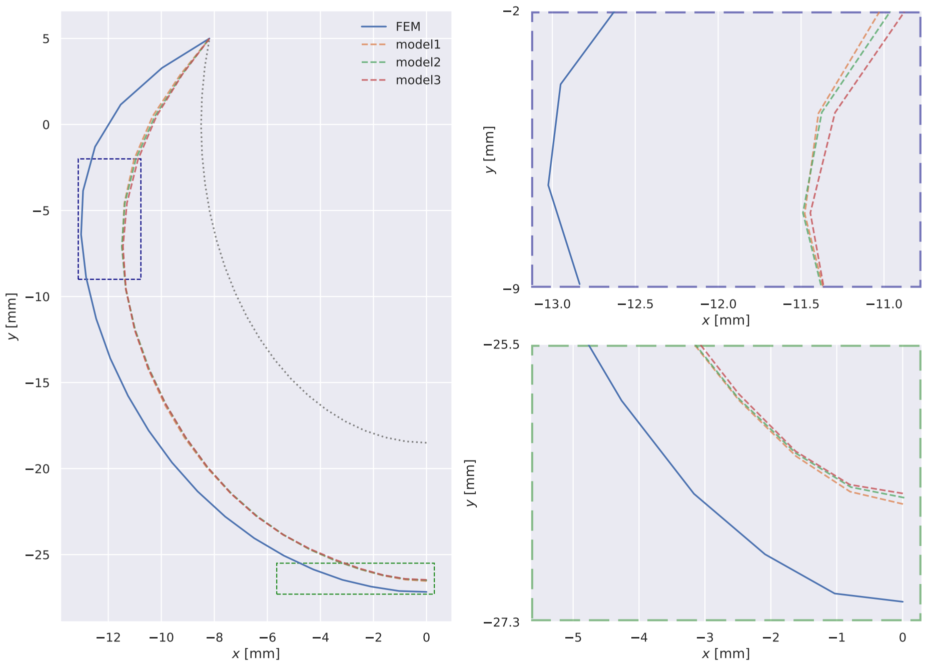


Figure 4.20: Deformation of a line in the middle of the ventricle. The line is through $y = 0$ using the FEM and three NN models. Parameters for NN models are written in Table 4.2. The reference configuration is shown as a dotted gray line.

In Figure 4.20, we have plotted the deformation of a line in the middle of the ventricle. The inflation is smaller for the NN solutions when compared to the FEM solution. The NN solutions are at the same point as the FEM solution on the base, but as we go down the ventricular wall, they diverge from the FEM solution, until the ventricular wall has its largest inflation. From this point, the NN solutions start converging toward the FEM solution, until we reach the apex. In the top right plot, we see that the three NN solutions are approximately 2.5mm from the FEM solution where the ventricle has inflated the most. At the apex, in the bottom right plot, the FEM solution ends at $z = -27$mm, while the NN solutions end at $z \approx -26$mm.

4.2.3 Problem 3

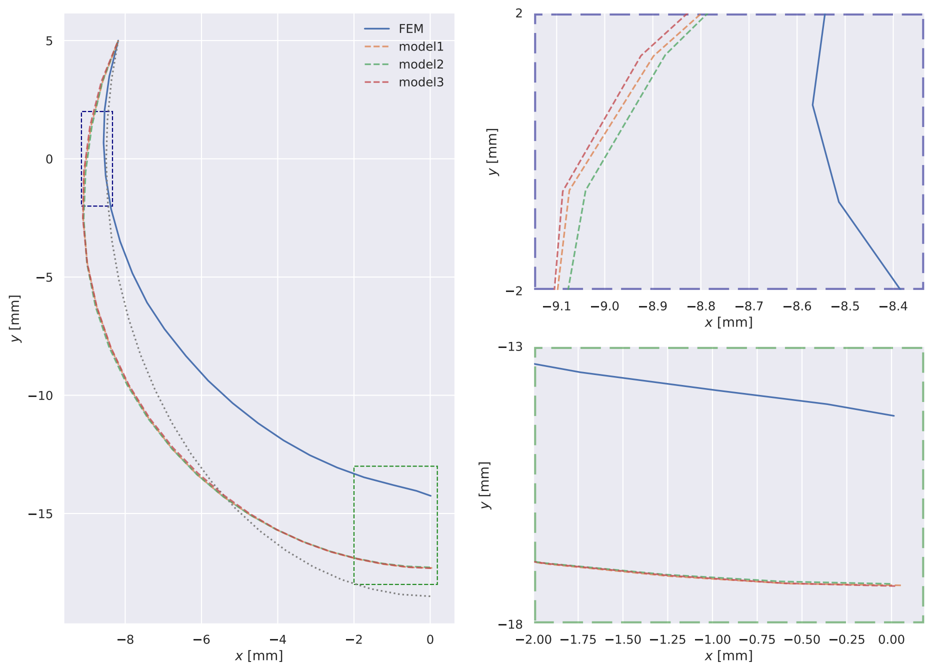


Figure 4.21: Deformation of a line in the middle of the ventricle. The line is through $y = 0$ using the FEM and three NN models. Parameters for NN models are written in Table 4.2. The reference configuration is shown as a dotted gray line.

In the final problem, we plotted the deformation of a line, like in problem 2. The results are shown in Figure 4.21. In this figure, we see that the contraction is smaller for the NN solution than for the FEM solution. Moreover, the NN solutions only diverge from the FEM solution as we move down the ventricular wall. The NN solutions and the FEM solution end the farthest apart at the apex. In the bottom right plot, we see that the NN solutions end at $z \approx -17$mm, and the FEM solution end at $z \approx -14.5$mm.

CHAPTER 5

DISCUSSION

5.1 Introductory Tests

The first example problem, introduced in section 3.1, was a one-dimensional toy problem that we used to validate our FEM implementation. We did this by measuring the error of our implementation and the analytical solution and found that for a coarse discretization, the error was lower than 10^{-2} along the entire domain. For a fine discretization, the error was no higher than 10^{-6}. In addition, we found that our FEM implementation had a convergence rate of 2. For the DEM the error was equal to the FEM error with a coarse discretization, but only 10^{-4} for a fine.

In the two following example problems, our focus was to study the effect NN hyperparameters had on the error. With this focus, we could find out which NN hyperparameters gave the best solutions, and use these results on problems from the cardiac verification benchmark. Therefore, we have represented the error of the NN solution as heatmaps in Figures 4.8-4.11 and Figures 4.14-4.16. These figures show the L^2-norm of the error as a function of sets of two different hyperparameters including the learning rate, number of hidden layers, and the number of hidden neurons. For the cantilever beam problem, we chose to stop the training at 300 epochs because the error for 50 and 60 hidden neurons with 4 hidden layers started increasing from this point and the error increased for more configurations when we trained for more epochs. When solving the contracting cube problem, the error increased earlier than for the cantilever beam problem. Therefore, we set the maximum number of epochs to 100. In addition, when using 20 points in each dimension in our domain, which is what we do in our reference solution in FEM, the L^2-norms of the error were much higher for the contracting cube problem than the L^2-norms we got for the cantilever beam problem. Therefore, the first two experiments are run with $40 \times 40 \times 40$ points throughout the domain.

Using the hyperparameter configurations that yielded the lowest L^2-norm of error (Table 4.2 and Table 4.4), we trained new NN models and plotted the NN calculated deformations along a single line for both the cantilever beam (Figure 4.12) and contracting cube (Figure 4.17) problem. For the cantilever beam, we saw that models 2 and 3 yielded almost identical solutions along the beam. Model 1 performed best along the entire line. Moreover, the solution from model 1 was comparable to the FEM solution. In contrast, none of the NN models trained on the contracting cube problem resulted in solutions that were comparable to the FEM solution. Though all three models had a similar shape to that of the FEM solution, only model 3 was close to it, and only in the middle 5 points.

For both problems, we saw that increasing the number of training points in the domain generally decreased the L^2-norm of the error. The change in this norm was smaller the more points that were used. This can be due to the energy measurements being sensitive to step size with a coarse discretization of the domain and the training being too short for the largest number of points. In other words, approximation might not have converged sufficiently for the latter cases.

One reason for the drop in performance in the contracting cube problem might be the complexity of the model. The first problem uses an isotropic material model with the neo-Hookean energy model for the passive strain energy. This model is much simpler than the transversely isotropic material model with active stress used to model the contacting cube, where we have to consider the fiber directions when calculating the energy. On the other hand, if the difference in model complexity had been the main reason for the poor performance, this would have been reflected in the error as a function of the complexity of the NN model. Put another way, if we increased the number of hidden neurons, the NN could have approximated a more complex material model, thus resulting in a lower error. But this was not the case. Although the error for the contracting cube decreased when we increased the number of hidden neurons above 20, it generally stayed the same when we added more neurons after this. Early on, we tested wider NN models, but when they did not yield much better results, and the training time increased, we chose not to use them in this thesis. Since then, we have changed part of our implementation, including using a more sophisticated integration scheme, Simpson's method instead of the mean rule. Our results might have been different if we had time to test wider models.

Another possibility is that the performance did, in fact, not drop, but that our implementation of the complex model was wrong. This would explain why the L^2-norm of the error started increasing after only 100 epochs, even though this problem was more complex than the first. Our NN models were trying to solve a different problem than the FEM, and their solution was therefore converging toward a different displacement, thus increasing the error between the two methods as the training continued.

When solving the cantilever beam problem, we timed both the FEM and the DEM (Table 4.3). Together with the measured times, we plotted functions to approximate their time complexity (Figure 4.13). For the FEM the function $y = 2^{N/3.9}$ closely matches the measured times, indicating an exponential time complexity. In contrast, the times measured for the DEM match the function $y = 400 + N^{1.68}$, a polynomial time complexity. In addition, the evaluation time for the DEM is constant, meaning its total time complexity is based on the training. Although the FEM has a higher order of complexity, it is still faster for $N = 30$ than the DEM is for $N = 5$. Moreover, due to memory limitations, it was not possible to run a discretization of more than $N = 35$ for the FEM on our hardware. These results show us that the FEM outperforms the DEM in time performance, as well.

5.2 Cardiac Verification Benchmark

Due to the poor performance of the models trained on the contracting cube problem, we used only the hyperparameters that yielded the best results for the cantilever beam problem when training NN models to solve the problems from the cardiac verification benchmark.

When comparing the deflection of the middle line for problem 1 (Figure 4.18) to the deflection of the line for the cantilever beam (Figure 4.12), we see that the NN models perform worse on the former than the latter. For problem 1, the NN models give an almost identical solution for half the beam. At the left side of the beam, model 1 is closer to the FEM solution than models 2 and 3. In the lower-left plot, we see that the NN solutions deviate from the FEM solution from the third point, at $x \approx 0.2$mm. This was not the case for the solution from NN model 1 for the cantilever beam. This solution stayed close to the FEM solution throughout the length of the beam. The poor performance in problem 1, compared to the cantilever beam problem, could be due to a more complex material model, which is transversely isotropic, as opposed to the isotropic model used for the cantilever beam. However, it could also be due to differences in the problems. Problem 1 uses a Neumann boundary condition along the bottom face of the beam, whereas the cantilever beam has a body force acting on its whole domain.

When measuring the strain for the solutions in problem 1, we found that the only result comparable to the FEM was for the strain along the y-axis using model 2. Model 1 gave the worst result along this axis. For the strain along the z-axis, all models showed lower strain than the FEM solution with no apparent difference between the NN models. For the strain in the x-axis, the NN models yielded wrong results for all points.

For problem 2, we plotted the deformation of a line in the middle of the ventricle (Figure 4.20). In this figure, we see that the inflation is smaller for the NN solutions than for the FEM solution. The two methods differ the most at the ventricular wall, where they are approximately 2.5mm apart. However, they are only 1mm apart at the apex. Moreover, we see that the NN solutions are much closer to each other than they were for problem 1, but model 3 still performs worse, both at the ventricular wall and at the apex. Model 1 performs the best in this problem.

The results are the opposite for problem 3 (Figure 4.21). Here, the NN solutions only diverge from the FEM solution as we move away from the base. The NN solutions are approximately 2.5mm from the FEM solution at the apex. Moreover, we can see in this figure that model 3 still performs the worst, but model 2 performs the best.

Though problems 2 and 3 use the same geometry, they differ in model complexity. Problem 2 uses an isotropic material model, meaning the material has the same physical properties in all directions. On the other hand, a transversely isotropic material, used in problem 3, has the same physical properties in two orthogonal directions, but different properties in the direction orthogonal to these. This affected the calculations of the active stress, much like it did for the contracting cube and the beam in problem 1. However, because of the complex fiber structure in problem 3, the energy calculations in this problem are more complicated, which is, in turn, prone to implementation errors. For problem 2, we did not have to implement any fiber structure, so the results for this problem might reflect the complexity of the geometry when compared to the cantilever beam.

The ventricle geometry also gave rise to the integration problem explained in section 3.3.4.5. The integration error has affected the energy measured by the DEM, which causes the solution to converge towards a different displacement than the FEM. The results in Figure 4.20 and Figure 4.21 hint at this, as the NN solutions are very close to each other. The results converged to the same solutions when we trained more complex models, to test whether the model complexity was the reason for the bad results in these problems. Also, we implemented a decoupled strain energy function for these problems, but this showed no apparent improvement in the solution.

Unfortunately, it is not possible to compare our results to those of Nguyen-Thanh et al. [41] in a meaningful way, because they used a different error measure to measure the performance of the NNs they trained. In addition, the only three-dimensional problem we solved that they have is the cantilever beam, but even here, we changed their Neumann boundary condition to a body force. It would be interesting, in future work, to test the models that gave the best results in their paper on the problems or material models used in this thesis.

Furthermore, the problems from the cardiac verification benchmark could be implemented using a simpler strain energy model, like the neo-Hookean used for the cantilever beam, to study how this affected performance. Moreover, implementing a better integration scheme for the ventricular geometry would provide a clearer indication of how the DEM performs.

CHAPTER 6

CONCLUSION

In this thesis, we have used the deep energy method (DEM) to solve three problems in the cardiac verification benchmark. We implemented a one-dimensional toy problem to verify our solution using the finite element method (FEM). We found that the FEM had a convergence rate of 2 and the absolute error along the domain was smaller than 10^{-6} for a discretization of $N = 1000$.

Following this, we implemented two three-dimensional problems for evaluating which configuration of hyperparameters performed best. The first problem dealt with a cantilever beam that is clamped on its left side and is subject to a body force. For this problem, we found that using the same number of points in the domain when training the NN and discretizing the domain for the FEM resulted in the L^2-norm of the error of $1.9 \cdot 10^{-3}$ for the best NN model. Moreover, we found that increasing the number of points in the domain resulted in better performance. The second problem we implemented was a cube subject to an active contraction and with a traction force on its right side. The L^2-norm of the error was generally one order of magnitude higher for this problem than the previous problem, even when we used double the number of points in every dimension than the number of points used to discretize the domain for the FEM. The increase in the error may be caused by the increased complexity when using a transversely isotropic material model and modeling active stress. The lowest L^2-norm of the error for this problem was $1.5 \cdot 10^{-2}$.

Using the results from the two problems, we chose the three NN architectures that yielded the lowest L^2-norm of the error and trained NNs with these architectures on the problems in the cardiac verification benchmark. When comparing the DEM solutions to the FEM, we saw that the DEM performance decreased as the model complexity increased. For the cantilever beam problem, we saw that though the FEM had an exponential time complexity, it still solved the problem faster than the DEM.

Future studies of the DEM could focus on implementing incompressible material models, since all our DEM models were compressible, while the cardiac verification benchmark uses incompressible models. Furthermore, a sophisticated integration scheme is necessary for ventricle geometry problems. In addition, the material models implemented in this thesis should be validated, as they may contain errors in the implementation. And lastly, the DEM method should be used with some form of regularization to avoid diverging solutions.

With the results we got in this thesis, we doubt that the DEM will be used for cardiac application in the near future.

APPENDIX A

BACKPROPAGATION ALGORITHM

The goal of backpropagation is to calculate the gradient of the loss function with respect to all parameters throughout the network. To do this, we start at the output layer and calculate the gradient of the error with respect to the weights in the output layer

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}^L} = \frac{\partial \mathcal{L}}{\partial \hat{\mathbf{z}}^L} \frac{\partial \hat{\mathbf{z}}^L}{\partial \mathbf{z}^L} \frac{\partial \mathbf{z}^L}{\partial \mathbf{W}^L}. \tag{A.1}$$

This is the result of the chain rule applied to the weights in the last layer. The first factor in this product is the gradient of the error with respect to the activations from the last layer. Using (2.33) with $l = L$, we can see that the second factor in (A.1) is the derivative of the activation function in the last layer, which gives us

$$\frac{\partial \hat{\mathbf{z}}^L}{\partial \mathbf{z}^L} = (\boldsymbol{\sigma}^L)'(\mathbf{z}^L). \tag{A.2}$$

In (2.34), we can set $l = L$ and rewrite the last factor in (A.1) as

$$\frac{\partial \mathbf{z}^L}{\partial \mathbf{W}^L} = \hat{\mathbf{z}}^{L-1}. \tag{A.3}$$

Using these relations, we can rewrite (A.1)

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}^L} = \frac{\partial \mathcal{L}}{\partial \mathbf{z}^L} (\boldsymbol{\sigma}^L)'(\mathbf{z}^L) \cdot \hat{\mathbf{z}}^{L-1}. \tag{A.4}$$

For simplicity, it is common to define a variable δ_k^l that represents the error in node k in layer l. For a vector-valued error in layer l, we simply write δ^l. In the last layer, this variable is defined as

$$\delta^L = \frac{\partial \mathcal{L}}{\partial \mathbf{z}^L} \cdot (\boldsymbol{\sigma}^L)'(\mathbf{z}^L), \tag{A.5}$$

which gives us

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}^L} = \delta^L \hat{\mathbf{z}}^{L-1}. \tag{A.6}$$

For the gradient of the error with respect to the biases, we only need to change the last factor in (A.1)

$$\frac{\partial \mathcal{L}}{\partial \mathbf{b}^L} = \frac{\partial \mathcal{L}}{\partial \hat{\mathbf{z}}^L} \frac{\partial \hat{\mathbf{z}}^L}{\partial \mathbf{z}^L} \frac{\partial \mathbf{z}^L}{\partial \mathbf{b}^L}. \tag{A.7}$$

We can use (2.34) once more to find an expression for the last factor in (A.7). Using $l = L$, we get

$$\frac{\partial \mathbf{z}^L}{\partial \mathbf{b}^L} = 1. \tag{A.8}$$

Using this, we can rewrite (A.7)

$$\frac{\partial \mathcal{L}}{\partial \mathbf{b}^L} = \delta^L. \tag{A.9}$$

For the penultimate layer, we get the equations

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}^{L-1}} = \frac{\partial \mathcal{L}}{\partial \hat{\mathbf{z}}^{L-1}} \frac{\partial \hat{\mathbf{z}}^{L-1}}{\partial \mathbf{z}^{L-1}} \frac{\partial \mathbf{z}^{L-1}}{\partial \mathbf{W}^{L-1}}, \tag{A.10}$$

and

$$\frac{\partial \mathcal{L}}{\partial \mathbf{b}^{L-1}} = \frac{\partial \mathcal{L}}{\partial \hat{\mathbf{z}}^{L-1}} \frac{\partial \hat{\mathbf{z}}^{L-1}}{\partial \mathbf{z}^{L-1}} \frac{\partial \mathbf{z}^{L-1}}{\partial \mathbf{b}^{L-1}}. \tag{A.11}$$

We can see that the last two factors in both equations are in the same form as for the last layer. This gives us the relations

$$\begin{aligned} \frac{\partial \hat{\mathbf{z}}^{L-1}}{\partial \mathbf{z}^{L-1}} &= \boldsymbol{\sigma}'(\mathbf{z}^{L-1}), \\ \frac{\partial \mathbf{z}^{L-1}}{\partial \mathbf{W}^{L-1}} &= \hat{\mathbf{z}}^{L-2}, \\ \frac{\partial \mathbf{z}^{L-1}}{\partial \mathbf{b}^{L-1}} &= 1. \end{aligned} \tag{A.12}$$

The only factor we need to rewrite now is the first one. We can rewrite it using the chain rule

$$\frac{\partial \mathcal{L}}{\partial \hat{\mathbf{z}}^{L-1}} = \frac{\partial \mathcal{L}}{\partial \hat{\mathbf{z}}^L} \frac{\partial \hat{\mathbf{z}}^L}{\partial \mathbf{z}^L} \frac{\partial \mathbf{z}^L}{\partial \hat{\mathbf{z}}^{L-1}}. \tag{A.13}$$

We recognize the product of the first two factors as δ^L. Using (2.34) we can rewrite the last factor

$$\frac{\partial \mathbf{z}^L}{\partial \hat{\mathbf{z}}^{L-1}} = \mathbf{W}^L. \tag{A.14}$$

We can now rewrite (A.10) and (A.11)

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}^{L-1}} = \delta^L \cdot \mathbf{W}^L \cdot \boldsymbol{\sigma}'(\mathbf{z}^{L-1}) \cdot \mathbf{z}^{L-2} \tag{A.15}$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{b}^{L-1}} = \delta^L \cdot \mathbf{W}^L \cdot \boldsymbol{\sigma}'(\mathbf{z}^{L-1}) \tag{A.16}$$

From (A.15) and (A.16) we can derive a general formula for the gradient with respect to the parameters in layer l

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}^{l}} = \delta^{l+1} \cdot \mathbf{W}^{l+1} \cdot \boldsymbol{\sigma}'(\mathbf{z}^{l}) \cdot \mathbf{z}^{l-1}, \tag{A.17}$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{b}^{l}} = \delta^{l+1} \cdot \mathbf{W}^{l+1} \cdot \boldsymbol{\sigma}'(\mathbf{z}^{l}). \tag{A.18}$$

These can both be simplified by defining the error variable for layers $l \neq L$

$$\delta^l = \delta^{l+1} \cdot \mathbf{W}^{l+1} \cdot \boldsymbol{\sigma}'(\mathbf{z}^{l}). \tag{A.19}$$

yielding

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}^{l}} = \delta^{l} \cdot \mathbf{z}^{l-1}, \tag{A.20}$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{b}^{l}} = \delta^{l}. \tag{A.21}$$

BIBLIOGRAPHY

[1] Martin S. Alnaes et al. *Unified Form Language: A domain-specific language for weak formulations of partial differential equations.* 2013. arXiv: 1211.4047 [cs.MS].

[2] D Ambrosi and S28990101312 Pezzuto. 'Active stress vs. active strain in mechanobiology: constitutive issues'. In: *Journal of Elasticity* 107 (2012), pp. 199–212.

[3] Stephen H. Bach et al. 'Learning the Structure of Generative Models without Labeled Data'. In: *Proceedings of the 34th International Conference on Machine Learning.* Ed. by Doina Precup and Yee Whye Teh. Vol. 70. Proceedings of Machine Learning Research. PMLR, 2017, pp. 273–282. URL: https://proceedings.mlr.press/v70/bach17a.html.

[4] Steven L Brunton, Joshua L Proctor and J Nathan Kutz. 'Discovering governing equations from data by sparse identification of nonlinear dynamical systems'. In: *Proceedings of the national academy of sciences* 113.15 (2016), pp. 3932–3937.

[5] Shengze Cai et al. 'Physics-Informed Neural Networks for Heat Transfer Problems'. In: *Journal of Heat Transfer* 143.6 (Apr. 2021), p. 060801. ISSN: 0022-1481. DOI: 10.1115/1.4050542. eprint: https://asmedigitalcollection.asme.org/heattransfer/article-pdf/143/6/060801/6688635/ht_143_06_060801.pdf. URL: https://doi.org/10.1115/1.4050542.

[6] Murray Campbell, A.Joseph Hoane and Feng-hsiung Hsu. ‘Deep Blue’. In: *Artificial Intelligence* 134.1 (2002), pp. 57–83. ISSN: 0004-3702. DOI: https://doi.org/10.1016/S0004-3702(01)00129-1. URL: https://www.sciencedirect.com/science/article/pii/S0004370201001291.

[7] Longbing Cao. ‘Data science: a comprehensive overview’. In: *ACM Computing Surveys (CSUR)* 50.3 (2017), pp. 1–42.

[8] Tanguy Chouard. ‘The Go Files: AI computer clinches victory against Go champion’. In: *Nature* (2016).

[9] Linda L Demer and FC Yin. ‘Passive biaxial mechanical properties of isolated canine myocardium.’ In: *The Journal of physiology* 339.1 (1983), pp. 615–630.

[10] Mariachiara Di Cesare et al. ‘World heart report 2023: Confronting the world’s number one killer’. In: *World Heart Federation: Geneva, Switzerland* (2023).

[11] Henrik Nicolay Finsberg. ‘Patient-specific computational modeling of cardiac mechanics’. In: *Series of dissertations submitted to the Faculty of Mathematics and Natural Sciences, University of Oslo.* (2018).

[12] L. Formaggia, A. Quarteroni and A. Veneziani. *Cardiovascular Mathematics: Modeling and simulation of the circulatory system.* MS&A. Springer Milan, 2010. ISBN: 9788847011526. URL: https://books.google.no/books?id=BQpm2VDN3kcC.

[13] Jan N Fuhg and Nikolaos Bouklas. ‘The mixed deep energy method for resolving concentration features in finite strain hyperelasticity’. In: *Journal of Computational Physics* 451 (2022), p. 110839.

[14] Zoubin Ghahramani. ‘Unsupervised Learning’. In: *Advanced Lectures on Machine Learning: ML Summer Schools 2003, Canberra, Australia, February 2 - 14, 2003, Tübingen, Germany, August 4 - 16, 2003, Revised Lectures.* Ed. by Olivier Bousquet, Ulrike von Luxburg and Gunnar Rätsch. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 72–112. ISBN: 978-3-540-28650-9. DOI: 10.1007/978-3-540-28650-9_5. URL: https://doi.org/10.1007/978-3-540-28650-9_5.

[15] Ian Goodfellow, Yoshua Bengio and Aaron Courville. *Deep Learning.* http://www.deeplearningbook.org. MIT Press, 2016.

[16] Ian J. Goodfellow et al. *Generative Adversarial Networks*. 2014. arXiv: 1406.2661 [stat.ML]. URL: https://arxiv.org/abs/1406.2661.

[17] Julius M Guccione, Kevin D Costa and Andrew D McCulloch. 'Finite element stress analysis of left ventricular mechanics in the beating dog heart'. In: *Journal of biomechanics* 28.10 (1995), pp. 1167–1177.

[18] Duc Haba. *Data augmentation with Python : enhance accuracy in deep learning with practical data augmentation for image, text, audio and tabular data*. eng. Birmingham, England, 2023.

[19] Clara Herrero Martin et al. 'EP-PINNs: Cardiac Electrophysiology Characterisation Using Physics-Informed Neural Networks'. In: *Frontiers in Cardiovascular Medicine* 8 (2022). ISSN: 2297-055X. DOI: 10.3389/fcvm.2021.768419. URL: https://www.frontiersin.org/articles/10.3389/fcvm.2021.768419.

[20] Tony Hey, Stewart Tansley, Kristin Michele Tolle et al. *The fourth paradigm: data-intensive scientific discovery*. Vol. 1. Microsoft research Redmond, WA, 2009.

[21] G.A. Holzapfel. *Nonlinear Solid Mechanics: A Continuum Approach for Engineering*. Wiley, 2000. ISBN: 9780471823049. URL: https://books.google.no/books?id=_ZkeAQAAIAAJ.

[22] Gerhard A Holzapfel and Ray W Ogden. 'Constitutive modelling of passive myocardium: a structurally based framework for material characterization'. In: *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences* 367.1902 (2009), pp. 3445–3475.

[23] P.R. Hoskins, P.V. Lawford and B.J. Doyle. *Cardiovascular Biomechanics*. Springer International Publishing, 2017. ISBN: 9783319464077. URL: https://books.google.no/books?id=MMkmDgAAQBAJ.

[24] Kent F Hubert, Kim N Awa and Darya L Zabelina. 'The current state of artificial intelligence generative language models is more creative than humans on divergent thinking tasks'. In: *Scientific Reports* 14.1 (2024), p. 3440.

[25] C. Johnson. *Numerical Solution of Partial Differential Equations by the Finite Element Method*. Dover Books on Mathematics Series. Dover Publications, Incorporated, 2012. ISBN: 9780486131597. URL: https://books.google.no/books?id=PYXjyoqy5qMC.

[26] A.M. Katz. *Physiology of the Heart*. Wolters Kluwer Health/Lippincott Williams & Wilkins Health, 2010. ISBN: 9781608311712. URL: https://books.google.no/books?id=24CcilHdzC4C.

[27] Diederik P Kingma and Max Welling. *Auto-Encoding Variational Bayes*. 2022. arXiv: 1312.6114 [stat.ML]. URL: https://arxiv.org/abs/1312.6114.

[28] Georgios Kissas et al. 'Machine learning in cardiovascular flows modeling: Predicting arterial blood pressure from non-invasive 4D flow MRI data using physics-informed neural networks'. In: *Computer Methods in Applied Mechanics and Engineering* 358 (Jan. 2020), p. 112623. ISSN: 0045-7825. DOI: 10.1016/j.cma.2019.112623. URL: http://dx.doi.org/10.1016/j.cma.2019.112623.

[29] S. Kollmannsberger et al. *Deep Learning in Computational Mechanics: An Introductory Course*. Studies in Computational Intelligence. Springer International Publishing, 2021. ISBN: 9783030765873. URL: https://books.google.no/books?id=5wY8EAAAQBAJ.

[30] Sander Land et al. 'Verification of cardiac mechanics software: benchmark problems and solutions for testing active and passive material behaviour'. In: *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences* 471.2184 (2015), p. 20150641.

[31] Hans Petter Langtangen. *Computational partial differential equations: numerical methods and diffpack programming*. Vol. 2. Springer Berlin, 2003.

[32] Hans Petter Langtangen and Kent-Andre Mardal. *Introduction to numerical methods for variational problems*. Vol. 21. Springer Nature, 2019.

[33] Yann LeCun, Koray Kavukcuoglu and Clément Farabet. 'Convolutional networks and applications in vision'. In: *Proceedings of 2010 IEEE international symposium on circuits and systems*. IEEE. 2010, pp. 253–256.

[34] Dong C Liu and Jorge Nocedal. 'On the limited memory BFGS method for large scale optimization'. In: *Mathematical programming* 45.1 (1989), pp. 503–528.

[35] Zhou Lu et al. 'The expressive power of neural networks: A view from the width'. In: *Advances in neural information processing systems* 30 (2017).

[36] S. Marsland. *Machine Learning: An Algorithmic Perspective, Second Edition*. Chapman & Hall/CRC Machine Learning & Pattern Recognition. CRC Press, 2014. ISBN: 9781466583337. URL: https://books.google.no/books?id=6GvSBQAAQBAJ.

[37] Pankaj Mehta et al. 'A high-bias, low-variance introduction to Machine Learning for physicists'. In: *Physics Reports* 810 (May 2019), pp. 1–124. ISSN: 0370-1573. DOI: 10.1016/j.physrep.2019.03.001. URL: http://dx.doi.org/10.1016/j.physrep.2019.03.001.

[38] Xuhui Meng et al. 'PPINN: Parareal physics-informed neural network for time-dependent PDEs'. In: *Computer Methods in Applied Mechanics and Engineering* 370 (2020), p. 113250. ISSN: 0045-7825. DOI: https://doi.org/10.1016/j.cma.2020.113250. URL: https://www.sciencedirect.com/science/article/pii/S0045782520304357.

[39] Francisco J. Montáns et al. 'Data-driven modeling and learning in science and engineering'. In: *Comptes Rendus Mécanique* 347.11 (2019). Data-Based Engineering Science and Technology, pp. 845–855. ISSN: 1631-0721. DOI: https://doi.org/10.1016/j.crme.2019.11.009. URL: https://www.sciencedirect.com/science/article/pii/S1631072119301809.

[40] I.E. Naqa, R. Li and M.J. Murphy. *Machine Learning in Radiation Oncology: Theory and Applications*. Springer International Publishing, 2015. ISBN: 9783319183053. URL: https://books.google.no/books?id=1N7yCQAAQBAJ.

[41] Vien Minh Nguyen-Thanh, Xiaoying Zhuang and Timon Rabczuk. 'A deep energy method for finite deformation hyperelasticity'. In: *European Journal of Mechanics-A/Solids* 80 (2020), p. 103874.

[42] Vien Minh Nguyen-Thanh et al. 'A surrogate model for computational homogenization of elastostatics at finite strain using high-dimensional model representation-based neural network'. In: *International Journal for Numerical Methods in Engineering* (June 2020). DOI: 10.1002/nme.6493.

[43] Michael A. Nielsen. *Neural Networks and Deep Learning*. misc. 2018. URL: http://neuralnetworksanddeeplearning.com/.

[44] William L. Oberkampf and Christopher J. Roy. *Verification and Validation in Scientific Computing*. Cambridge University Press, 2010.

[45] Kristian B. Ølgaard and Garth N. Wells. ‘Optimizations for quadrature representations of finite element tensors through automated code generation’. In: *ACM Transactions on Mathematical Software* 37.1 (Jan. 2010), pp. 1–23. ISSN: 1557-7295. DOI: 10.1145/1644001.1644009. URL: http://dx.doi.org/10.1145/1644001.1644009.

[46] H Petter Langtangen and A Logg. *Solving PDEs in Python—The FEniCS tutorial volume I.* 2017.

[47] F. Provost and T. Fawcett. *Data Science for Business: What You Need to Know about Data Mining and Data-Analytic Thinking.* O’Reilly Media, 2013. ISBN: 9781449374297. URL: https://books.google.no/books?id=EZAtAAAAQBAJ.

[48] Maziar Raissi, Paris Perdikaris and George E Karniadakis. ‘Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations’. In: *Journal of Computational physics* 378 (2019), pp. 686–707.

[49] Rama Ramakrishnan. ‘How to Build Good AI Solutions When Data Is Scarce’. eng. In: *MIT Sloan management review* 64.2 (2023), pp. 48–53. ISSN: 1532-9194.

[50] Sebastian Ruder. *An overview of gradient descent optimization algorithms.* 2017. arXiv: 1609.04747 [cs.LG].

[51] C.A. de Saracibar. *Nonlinear Continuum Mechanics: An Engineering Approach.* Springer International Publishing, 2023. ISBN: 9783031152078. URL: https://books.google.no/books?id=_HfSEAAAQBAJ.

[52] Gerhard Sommer et al. ‘Biomechanical properties and microstructure of human ventricular myocardium’. In: *Acta biomaterialia* 24 (2015), pp. 172–192.

[53] E. Stevens, L. Antiga and T. Viehmann. *Deep Learning with PyTorch.* Manning Publications, 2020. ISBN: 9781617295263. URL: https://books.google.no/books?id=fff1DwAAQBAJ.

[54] Aslak Tveito and Ragnar Winther. *Introduction to partial differential equations: a computational approach.* Vol. 29. Springer Science & Business Media, 2004.

[55] Amit Kumar Tyagi and Ajith Abraham. ‘Recurrent neural networks: Concepts and applications’. In: (2022).

[56] Ashish Vaswani et al. *Attention Is All You Need*. 2023. arXiv: 1706.03762 [cs.CL]. URL: https://arxiv.org/abs/1706.03762.

[57] K. Wolter et al. *Resilience Assessment and Evaluation of Computing Systems*. SpringerLink : Bücher. Springer Berlin Heidelberg, 2012. ISBN: 9783642290329. URL: https://books.google.no/books?id=BOai1mAi44AC.

[58] Liu Yang, Xuhui Meng and George Em Karniadakis. 'B-PINNs: Bayesian physics-informed neural networks for forward and inverse PDE problems with noisy data'. In: *Journal of Computational Physics* 425 (2021), p. 109913.

[59] Yinhao Zhu et al. 'Physics-constrained deep learning for high-dimensional surrogate modeling and uncertainty quantification without labeled data'. In: *Journal of Computational Physics* 394 (Oct. 2019), pp. 56–81. ISSN: 0021-9991. DOI: 10.1016/j.jcp.2019.05.024. URL: http://dx.doi.org/10.1016/j.jcp.2019.05.024.