NTNU
Norwegian University of Science and Technology
Faculty of Engineering
Department of Civil and Environmental Engineering

Kvalbein, Asbjørn Voll

Application of physics-informed neural network to forward and inverse consolidation and bearing capacity problems

Master's thesis in Civil- and enviromental engineering
Supervisor: Ivan Depina
August 2023

Kvalbein, Asbjørn Voll

Application of physics-informed neural network to forward and inverse consolidation and bearing capacity problems

Master’s thesis in Civil- and enviromental engineering
Supervisor: Ivan Depina
August 2023

Norwegian University of Science and Technology
Faculty of Engineering
Department of Civil and Environmental Engineering

Abstract

This thesis explores the applications of physics-infomed neural networks (PINN) in geotechnical engineering problems of consolidation and undrained bearing capacity for forward and inverse problems. A physics-informed neural network is a type of deep learning which is constrained by any physical laws that governs the implemented problem, often in the way of a PDE. The PINNs are implemented into a scientific library designed for usage of PINNs called DeepXDE. The performance of these models is first checked on consolidation followed by bearing capacity. A forward analysis seeks the hidden solution based on the physical laws of the problem given its boundary data. The inverse analysis seeks to approximate a unknown state for the physical problem and learn its material parameters.

The physical laws governing the consolidation is here represented by Terzaghi's one dimensional uncoupled consolidation equation, represented as a PDE in the model. This equation, which predicts consolidation through excess pore pressure, is implemented into a PINN for forward and inverse analysis of the single drained cases, with one open permeable boundary, and the double drained cases with two permeable boundaries. After these cases, noise is added to the datasets for the PINNs in order to check their performance with corruption. The results from the PINNs where evaluated up against the analytical solution for 1D consolidation. The PINNs implemented converges well and showed good accuracy while keeping a low time to train.

The governing PDEs for equilibrium in 2D describes the physical laws for the bearing capacity problems. In order to ensure yielding of the soil, the Tresca criterion for undrained Mohr-Coulomb soil is derived and implemented along with the PDEs into a PINN. The PINN was first implemented with Cartesian coordinates before switching to polar. The results where evaluated up against results for the same problem in Plaxis. Both the forward and inverse analysis of undrained bearing capacity showed very good accuracy and a modest time to train.

In general, it is clear that problems of consolidation and bearing capacity can be implemented into PINNs and show good performance, which in turn showes great potential for the further use of PINNs in geotechincal engineering.

Sammendrag

Denne masteroppgaven undersøker applikasjonen for fysikkinformerte neurale nettverk (FINN) i geoteknisk analyse av konsolidering og udrenert bæringskapasitet for fram- og bakoverorienterte problemer. Et fysikkinformert neuralt nettverk er en type dyplærings som blir begrenset av de fysiske lover som beskriver det implementerte problemet. FINN blir implementert in i et vitenskapelig bibliotek kalt DeepXDE. Ytelsen til disse modellene blir først sjekket på konsoliderings problem før bæringskapasiet. Fram analysen søker en skjult løsning basert på de fysiske lover som styrer problemet, gitt dets grenseverdier. Bakover analysen prøver å approximere en ukjent tilstand for det fysiske problemet og gjennom det lære material parameterne.

De fysiske lover om beskriver konsolideringen er her beskrevet av Terzaghis endimensjonal ukoblet konsoliderings ligning, representert som en PDL. Denne ligningen beskriver konsolidering gjennom å undersøke overtrykket i poretrykket, og blir implementert inn i en FINN for fram og bakover analyse for enside og toside drenerte problemer. Etter disse casene blir støy introdusert til datasettet for FINN og deretter sjekket for deres ytelse med korrupterte data. Resultatene fra FINN blir evaluerte opp imot den analytiske løsningen for 1D konsolidering. De implementerte FINN viser good convergens og god nøyaktighet samtidig som de tar kort tid å trene.

De beskrivende PDLer for likevekt i 2D beskriver de fysiske lover for bæringskapasitets problemene. For å forsikre seg om at jorden gir etter deriveres Tresca criteriet for udrenert Mohr-Coulomb jord og implementeres sammen med PDLene inn i FINN. FINNen ble først implementert med kartesiske koordinater før det ble byttet til polare. Resultatene ble evaluert opp imot resultatene for det samme problemet i Plaxis. Både de fram- og bakover orienterte analysene for udrenert bæringskapasitet viste god konvergens med veldig god nøyaktighet med en moderat treningstid.

Generelt er det klart at konsoliderings og bæringskapasitets problem kan bli implementerte inn i FINN og vise good ytelse, noe som igjen viser stort potential for framtiden for FINN i geoteknisk ingeniørarbeid.

Preface

This thesis concludes my Master of science degree in Geotechnical Engineering at the department of Civil and Environmental engineering at the Norwegian University of Science and Technology.

I would like to thank my supervisor Ivan Depina for his help and guidance through this thesis. Thank you! I would also like to thank Sissel for the support during my years as a student. An additional thank you to Olaus for letting me borrow his computer these months for running the models

Asbjørn Voll Kvalbein

Contents

List of Figures

List of Tables

Abbreviations

List of all abbreviations in alphabetic order:

- **1-D** One dimensional
- **AI** Artificial intelligence
- **ANN** Artificial neural network
- **BC** Boundary conditions
- c_v Coefficient of consolidation
- **DD** Double drained
- **FEM** Finite-element method
- **FNN** Fully-connected neural network
- **k** Coefficient of permeability
- **LR** Learning rate
- **LHS** Latin hypercube sampling
- **MAE** Mean absolute error
- **ML** Machine learning
- **MOO** Multi-Objective optimization
- m_v Confined compressability
- **MSE** Mean square error
- **NN** Neural network
- **NTNU** Norwegian University of Science and Technology
- **PDEs** Partial differential equations
- **PFNN** Parallel fully-connected neural network
- **PINN** Physics-informed neural networks
- **ReLU** Rectified linear unit function
- **SD** Single drained
- s_u Undrained shear strength
- **Tanh** Hyperbolic tangent function

1 Introduction

This thesis is written for the geotechnical department at NTNU. Geotechnics is the study of soils and rock pertaining to construction purposes. This includes design of foundations, retaining walls, settlements and earthworks. The aim of this thesis is the further examine the application of physics-informed neural networks in forward and inverse problems in geotechnical processes.

1.1 Background

With the rise of computational power and availability of data, machine learning has become a popular tool to solve complex problems in science and engineering. Machine learning can use different algorithms to approximate a solution. The algorithm is then optimized using a artificial neural network which mimics the process of learning done by a human brain. Deep learning is a term when this artificial neural network (ANN) consists of more than one layer. Deep learning has been successfully applied to a number of problems related to science and engineering such as weather forecasting (Ren et al., 2021), image based medical diagnosis (Chen et al., 2022) and autonomous driving (Grigorescu et al., 2020). Deep learning has also had great success at solving problems within applied mathematics, specifically partial differential equations (PDEs). Both machine learning and deep learning is reliant on datasets from which they learn, however, adequate dataset are not always easily obtained. Datasets could be insufficient in size and/or have missing values and/or spurious extreme values which makes convergence and good accuracy less likely to be achieved. Because of this there is a need for more robust ANNs that can handle such complications and still reach convergence with good accuracy.

In 2018 (Raissi, Perdikaris, and Karniadakis, 2019) realized the potential of deep learning to solve PDEs and introduced physics - informed neural network (PINN(s)), which is a deep neural network that can be trained through self-supervised learning while being constrained by a PDE that governs the problem. By minimising the loss function, a PINN approximates the solution of one or more PDEs. The loss function checks not only if the PDE is satisfied in the domain, but also it's boundaries and initial conditions. PINN are in all essence a mesh-free technique, which converts the problem of solving a PDE directly by it's governing equations into a problem of optimization with respect to the loss function. PINNs has been applied to several problems within fields of geotechnical engineering by the method of data-driven solutions and data-driven discovery of partial differential equations such as unsaturated flow (Depina et al., 2021) and one dimensional consolidation (Bekele, 2020). Note that data-driven solutions and discovery is also called forward and inverse analysis respectively.

1.2 Project description

The purpose of this thesis is to examine the following questions:

Main objective

Investigate the application of PINNs on problems of consolidation and bearing capacity in geotechnical engineering

Sub-objectives This will be completed by fulfilling a set of sub-objectives:

- Perform a literature study on PINNs and the DeepXDE library
- Implement one-dimensional consolidation equation into a PINN and check performance for forward and inverse analyses
- Introduce noise to the datasets for one-dimensional consolidation PINN and check performance
- Implement undrained bearing capacity problem into a PINN for both forward and inverse analysis and check performance

By examining this question, we hope to to further the possible application of PINN in geotechnical engineering. This will be done by applying PINNs to problems of consolidation and bearing capacity using a python package called DeepXDE (L. Lu et al., 2021) which is a library for scientific physics-informed learning. The thesis starts with the uncoupled problem of Terzaghi's one dimensional consolidation equation in the elastic range followed by the coupled problem of bearing capacity for undrained analysis in Mohr-Coulomb soil.

1.3 Structure of the thesis

After the first introductory chapter, the second chapter of the thesis is dedicated to the relevant theory for the thesis. First presented is a general introduction to machine learning and deep learning before continuing to physics-informed neural networks. After that, the theory on one-dimensional consolidation followed by the theory of bearing capacity. Chapter three is dedicated to the software that is used in the thesis. Chapter four shows the method of implementation of PINNs into python code and the programming package DeepXDE. In chapter five shows the different case studies and a discussion around the results from each separate case studie. The thesis concludes with chapter six and the discussion of the results and the possible paths for further examination of the subject before presenting the conclusion in chapter 7. All code and relevant data will be added into the appendix or uploaded to github.

2 Theory

The sections on theory starts off with a small subsection explaining what source the material has been inspired by before moving on to presenting the actual theory.

2.1 Machine learning, Deep learning and Physics-informed neural networks

The content of this thesis concerning the topics of machine learning, deep learning and physics - informed neural networks are heavily inspired the chapters two, three and five from the book (Kollmannsberger et al., 2021).

Machine learning, a subset of the general term artificial intelligence (AI), uses a large amount of data that is subject to a set of previously defined rules. The more formal way of defining machine learning is "a computer program is said to learn from experience E with respect to some class T, as measured by P, improves with experience E" (Mitchell, 1997). The majority of algorithms used in machine learning can be summed up as a sum of the components: a dataset, a cost function, an optimization procedure and a parameterized model. It is worth pointing out that the cost function can also be referred to as the loss function which is the term used in this thesis.

There are three different ways for the machine learning algorithm to learn, as shown in 2.1: Supervised-, Deep- and Reinforcement learning.

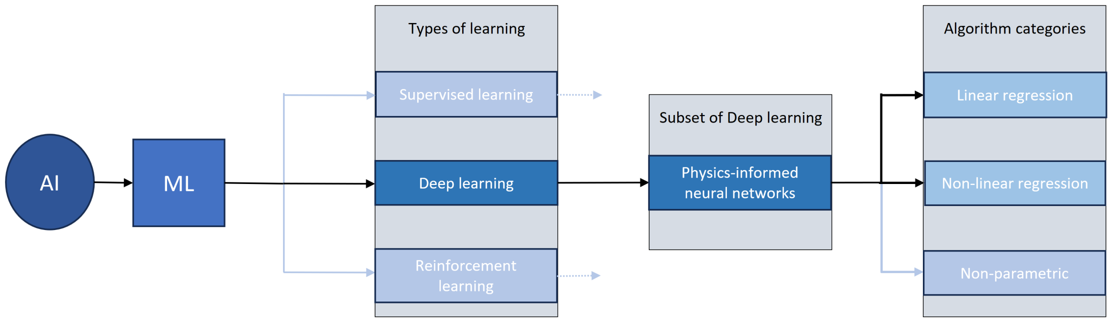


Figure 2.1: Different types of learning

Supervised learning is a method that by using a labeled dataset for training, the model learns to make predictions about unseen data differing from the original dataset. With regression in conjunction with supervised learning, the goal is usually to predict a numerical value. In short, this means the output of the algorithm is a function which maps the input to an output, often in the form of a real number. The input vector to such a regression model could be $x \in \mathbf{R}^n$ which is used to predict the vector $\hat{y} \in \mathbf{R}$. The vector $\hat{y}$ represents the labeled dataset comprising of the targets y_i. However, the mathematical model that is required for regression analysis is not always that easy to define. A possible solution to this is artificial neural networks (ANN).

2.1.1 Fully-connected neural networks

A artificial neural network that utilizes supervised learning is in many ways and regards a parameterized function that maps $y = f_{NN}(x)$. The difference from a normal optimization problem is in the fact that a neural network usually consists of several different functions within the network. As an example: $f_{NN}(x) = f_3(f_2(f_1(x)))$ where the functions f_i represents a layer in the neural network. The input which is represented by x goes through a user-specified number of "hidden layers" in the neural network, where each hidden layer consists of another user-specified number of neurons in each layer, before the last layer which is called the output layer. This is why it is called a feed forward neural network since it runs through from left to right. The number of layers determines the depth of the neural net, and the number of neurons the width. It is also worth noting that if the neural network consists of more than one layer, it is usually referred to as a "deep neural network" or just "Deep learning ". A simple example which consist of three hidden layers with three nodes in each layer is shown in figure **??**.

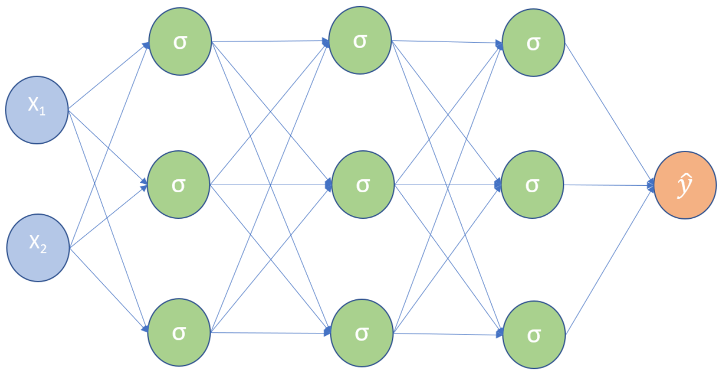


Figure 2.2: Simple ANN. Adapted from (Kollmannsberger et al., 2021)

Each circle within the figure represents a neuron. x_1 and x_2 are the two inputs in the input layer for this ANN, represented here by the blue color. Further, σ is the activation function applied to each neuron within each hidden layer represented by the orange color and finally, $\hat{y}$ is the prediction in the output layer shown with the orange color. The arrows pointed in between the each neuron can be thought of as the weights that governs the inputs. If each neuron in the former layer is connected to all other neurons in the next layer it is referred to as a fully-connected neural network (FNN). When the network has gone through all layers from left to right, it has done one iteration of the network. In the process of learning, the network runs through these layers a user specified number of times. To further explain what is happening within the network, each neuron receives an input in form of a vector of features denoted $\mathbf{x}$ which is further mapped and outputs a scalar $a(\mathbf{x}$ to the next neuron. In short, the input vector is transformed as shown in equation 2.1.

$$z = \mathbf{w}^T\mathbf{x} + b \tag{2.1}$$

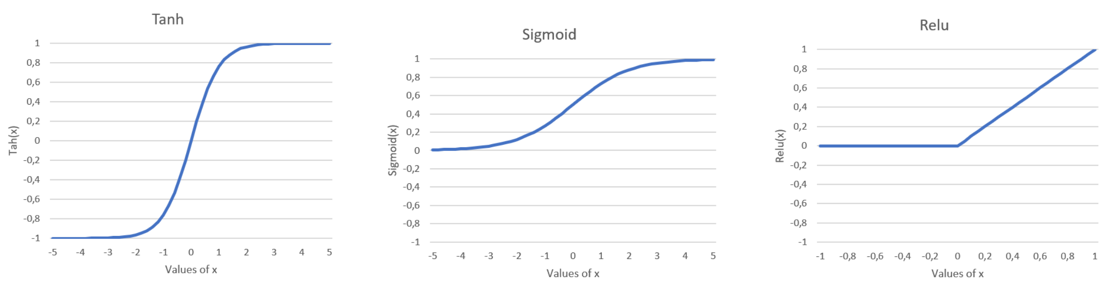
where it is given the weight $\mathbf{w}$ and bias b. The output for the neuron is calculated by applying a linear or non - linear activation function σ to the scalar z shown by equation 2.2

$$a(\mathbf{x}) = \sigma(z) \tag{2.2}$$

σ is called the activation function and is usually the same for all layers within the neural network, with the exception of the output layer which can have just a linear function that does not alter or change the scalar. The activation functions used in this thesis are the Sigmoid function (eq 2.3), the rectified linear unit (ReLU) (eq 2.4) and the hyperbolic tangent function (Tanh) (eq 2.5). Following are the equations for the Sigmoid, ReLU and Tanh functions respectively.

$$\sigma(z) = \frac{1}{1+e^{-x}} \quad (2.3) \qquad \sigma(z) = max(0,x) \quad (2.4) \qquad \sigma(z) = tanh(x) \quad (2.5)$$

The activation functions all have their positives and negatives. In machine learning, it's normal to normalize the datasets form it's original values to values between values ranging from $[-1, 1]$. This is due to the decrease in computational cost and time because of small values being easier to compute and the activation functions being defined within that interval. But there are differences for the activation functions. For instance, the non-linear function Tanh is more applicable if you have a dataset consisting of values ranging from $[-1, 1]$ as shown in figure 2.3a. Figure 2.3b shows the Sigmoid function, also called the logistic function provides another non-linear function but for values between $[0, 1]$. Lastly, shown in figure 2.3c, the ReLU function just returns the value of z if $z \geq 0$. Since both the Tanh and Sigmoid functions are non-linear functions, these take considerably longer to compute than the ReLU function. This is one of the reasons why the ReLU function has become a popular choice for the hidden layers of the neural network in recent years. Another reason is that with the non-linear activation functions, there is a problem of "vanishing gradients" which is caused by increasing weights that saturate the function which leads to small gradients that hinder the network from learning further (Kollmannsberger et al., 2021).



(a) Tanh **(b)** Sigmoid **(c)** ReLU

Figure 2.3: Activation functions. Adapted from Kollmannsberger et al., 2021

2.1.2 Loss function and fitting the data

Regression in conjunction with the activation functions measure the performance, but there is also a need to measure the performance of the model. One common way to do this is by the use of square error loss. If the prediction is $\hat{y}$ and the labeled data is $\mathbf{y}$, the squared error loss is defined as $(y - \hat{y})^2$. But as can be seen from the equation, this only gives the square error loss for one point (x_i, y_i). In order to find a measurement for the whole dataset there are two commonly used methods called mean square error (MSE) and mean absolute error (MAE) shown, respectively, in the equations below:

$$MSE = \frac{1}{m}\sum_{i=1}^{m}(y - \hat{y})^2 \tag{2.6a}$$

$$MAE = \frac{1}{m}\sum_{i=1}^{m}|y - \hat{y}|^2 \tag{2.6b}$$

Shown by equations (2.1a) and (2.1b), the closer $\hat{y}_j$ is to y_j, the lower the error will be. Mean square error and mean absolute error are both popular choices for the cost function, also referred to as the loss function, because they are continuously derivative and penalize large differences. For the algorithm, it's goal is to optimize, meaning minimize, the loss function. This is what is essence of learning, but what is the differing factor from an optimization problem to machine learning optimization is that machine learning divides the dataset into different parts for training, validation and testing. MSE is then applied to these separately and individually. The training set is used to train the model, while the validation set is used to evaluate the models performance during training. The test set is hitherto unseen data for the model and is used after training to evaluate the models performance. The loss functions for training and validation is a good predictor for if the model is over- or underfitting. Overfitting means that the model has learned the detail and noise in the set so well that it impacts it's performance negatively when it's presented with new data. The reason for this is that the model chooses such a complex function that it fits the data perfectly due to high variance. Underfitting is the opposite, meaning that the model has not learned from the dataset and the function it unfit to represent the data because of a high bias. An illustrative example is provided in figure **??**

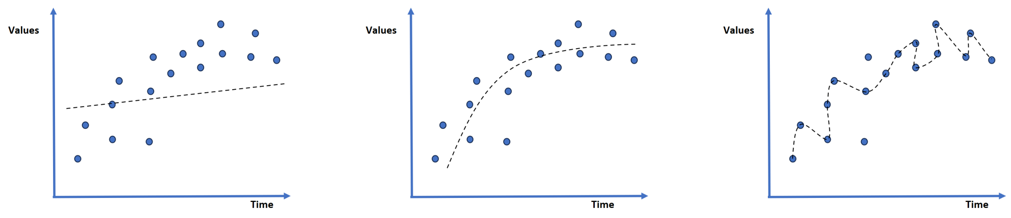


Figure 2.4: Underfitting, robust fit and overfitting

2.1.3 Optimization functions, parameters and hyper-parameters

There are two optimization function that has been applied to the problems dealt with in this thesis - the Adam optimizer (P.Kingma and Ba, 2014) which is the most commonly used optimizer for machine learning problems and the L-BGFS optimizer (Taylor et al., 2022).

The Adam optimizer has a number of benefits such as suitability for large datsets and problems, requires little memory and is computes efficiently. It is utilizes a method of stochastic decent by combining two methods "AdaGrad" and "RMSProp" in order to efficiently reach the global minimum.

The BFGS optimizer computes the Hessian matrix of the loss function using gradient evaluations. The Hessian matrix does not need matrix inversion which results in a computational complexity of $O(n^2)$. The fact that it does not need matrix inversion makes it computationally efficient but the use of the Hessian matrix gives rapid growth in memory usage. The L-BFGS optimization algorithm or Limited-Memory Broyden-Fletcher-Goldfarb-Shanno method is an algorithm that works well with large datasets because of it requiring less memory than the original BFGS algorithm due to it only saving only a few vectors of the Hessian matrix as a representation.

Both the Adam and the L-BFGS optimizer are gradient based optimizers, meaning they seek the smallest minima during training. But the L-BFGS optimizer can get stuck on a local minima rather than a global minima. The Adam optimizer does not have this same issue. Due to this, it is common to implement the Adam optimizer first and train the network on this optimizer before switching to the L-BFGS optimizer at the end of the training cycle.

There are two collective terms for a lot of what goes into an ANN, parameters and hyper-parameters. Hyper parameters are what controls the learning process sets the parameters that the algorithm ends up learning. This means that the hyper-parameters, with hyper meaning top level parameters, are external to the model and does not change during training of the model. Examples are: Learning rate (LR), activation function, number of hidden layers and neurons, iterations, etc.

Parameters are not external to the model, but internal. These are the the parameters that will be updated by the model during training. A prime example is the weights and biases of the ANN which usually starts at zero and is updated as the training progresses.

2.1.4 Physics-informed neural networks

In the previous section, supervised machine learning with ANNs was introduced. One of the weaknesses of such a model is that it requires a large amount of data about the solution in order to predict or generate an accurate surrogate model. This type of data which can be produced by simulations or experiments can be hard to come by and/or expensive to produce. In 2019 (Raissi, Perdikaris, and Karniadakis, 2019). proposed physics-informed neural networks, commonly denoted and also referred to in this thesis as PINNs, which utilizes the underlying physics that describes the problem the machine learning model is trying to generate. Normally, the physics governing the problem in the form of equations, ordinary differential equations or partial differential equations that represents the problem is known in advance. By incorporating this knowledge into the process of learning, the solution space becomes much more narrow which in turn leads to a less of a need for training data in order to reach the solution. Due to the physics part, PINNs are much better at handling datasets with noise than regular machine learning models. Before going into a general example to exemplify what PINNs are about, a small explanation of what forward and inverse analysis is. The forward problem are where the coefficients are known and the hidden solution is calculated based on the the boundary data and/or initial data and PDE constraints of the model. The inverse problem is where the coefficients are unknown and by giving the model scattered data of the the examined problem, the model utilizes this data in conjunction with the constraints as defined in the forward analysis, to approximate the unknown variables like material coefficients of the examined case. PINNs cannot compete with other numerical methods, like finite element method (FEM), when it comes to the forward analysis. But PINNs seem to outperform other numerical methods, including FEM, when it comes to inverse analysis.

$$\frac{du}{dt} + \mathcal{N}[p;\lambda] = 0, \quad x \in \Omega \quad t \in \sqcup \tag{2.7}$$

An example of a PINN is shown by the nonlinear partial differential equation 2.7. In this example, the solution to the problem p(x,t) is dependent on the factors for time and space where $t \in [0, T]$ and $x \in \Omega$. In this case Ω is representative of a space in $\mathbb{R}^D$. Further, the nonlinear part $\mathcal{N}[p;\lambda]$ represents a non-linear differential operator with the coefficients λ. This description is just illustrative but could cover different problems like the diffusion equation. If a classic FNN were implemented, the left hand side of equation 2.7 would evaluate the network as it approximates the solution p(x,t). The ANN and the physics part are both using the same parameters in terms of weights and biases. In a normal supervised learning scenario, there would only be one loss function that works, but in a PINNs, there are more than one. A general representation with MSE is:

$$L = MSE_u + MSE_f \tag{2.8}$$

The first term MSE_u calculates the error at the known points which are the initial and/or boundary conditions. The second term MSE_f uses a larges set of collocation points randomly sampled within the domain to enforce the PDE. In addition, the term MSE_f checks the error between the left hand side of equation 2.7 and the right hand side and penalizes it at every collocation point within the domain of the PDE. There are in general only two ways to enforce the boundary conditions for a forward problem, hard and weak. Weak enforcement means that there are multiple terms that enforces the solution on the boundaries. In case of a hard boundary conditions, the output of the network is transformed so that it automatically satisfies the boundary conditions, which simplifies the loss function due to the negation of terms.

An example of PINN architecture is:

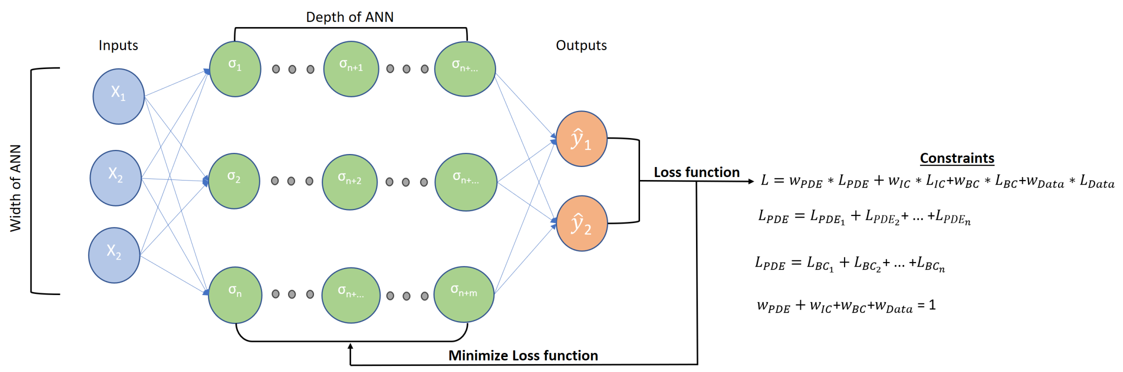


Figure 2.5: PINN architecture

As shown in the figure, the loss function consists of several terms that act as the constraints for the PINN. Loss functions like this fall into the category of Multi-Objective optimization (MOO) since they have several terms with their individual weights. For example, L_{PDE} could be given significantly higher weights than L_{BC}. By tuning the weights, one could control each part of the loss functions influence on the total loss. These parameters are very sensitive and should only be tweaked carefully.

2.1.5 Sampling methods

In the previous section, it was mentioned that the term MSE_f in equation 2.8 uses collocation points which are sampled within the domain. There are several different sampling methods for these collocation points, but in this thesis there are only two applied methods: Uniform (equispaced grid) sampling("Uniform distribution", n.d.) and Latin hypercube sampling (Shields and Zhang, 2015) (LHS).

Uniform training distribution means that the sampling function samples points at an equispaced distance within the domain including the initial and boundary data. This distribution is the default distribution in the DeepXDE library.

Latin hypercube sampling is a statistical method of taking sampling points at near-random from a distribution of several dimensions. A latin hypercube sampling means that if you have a square grid of $n\ x\ m$ dimensions, there has to be one, and only one sample in each row and column. An example is provided for this in figure 2.6.

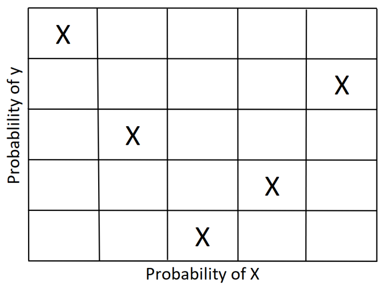


Figure 2.6: Latin hypercube sampling

2.2 Consolidation

The theory concerning one dimensional consolidation is inspired by (Craig and Knappett, 2020a). Consolidation is the study of vertical displacement of the soil surface over time. The study of consolidation is a very relevant field for the geotechnical engineer, and improvements made in terms of efficiency made by PINNs is an important investigation. First presented is the theory of one-dimensional consolidation which will be used to predict consolidation of the soil due to dissapation of excess pore pressure over time represented by the coefficient of consolidation c_v, followed by the initial- and boundary conditions for the relevant consolidation problems before ending with the analytical solutions to the consolidation equation.

2.2.1 Theory of one-dimensional consolidation

The analytical model for determining consolidation within the soil at any time that was developed by Terzaghi (1943) makes a number of assumptions: That the soil is homogeneous and fully saturated. The solid particles in the soil as well as water is incompressible. That compression and flow are only vertical in dimension. Strains will be and remain small. All hydraulic gradients are valid according to Darcy's law. The coefficients of permeability (k) and volume compressibility (m_v will remain constant. Lastly, there exists a special and unique relationship which is independent of time between the void ratio and effective stress.

The following three quantities, excess porewater pressure (p), the distance form the top of the soil layer (y) and the transitioned time (t) from the application of the load (q), relates to the theory. Considering the small element shown in figure 2.7 when a total vertical stress q is applied. Darcy's law defines the flow through the element as

$$\nu_{\mathbf{y}} = ki_y = -k\frac{\partial h}{\partial y} \tag{2.9}$$

Where k is the coefficient of permability and i_z is the total head. The change in total head at any fixed position is only due to a change in pore water pressure:

$$\nu_{\mathbf{y}} = -\frac{k}{\gamma_w}\frac{\partial p}{\partial y} \tag{2.10}$$

If there is no change to the element and the assumption that water is incompressible, the water entering the element and leaving it per unit of time has to be zero. Thus:

$$\frac{\partial \nu_y}{\partial y} = 0 \tag{2.11}$$

However, if there is a volumetric change, the equation of continuity is:

$$\frac{\partial \nu_y}{\partial y}dxdydz = \frac{dV}{dt} \tag{2.12}$$

Using equation 2.12 as a basis, equation 2.10 becomes:

$$-\frac{k}{\gamma_w}\frac{\partial^2 p}{\partial y^2}dxdydz = \frac{dV}{dt} \tag{2.13}$$

$\frac{dV}{dt}$ is the rate of volume change per unit time. It can also be expressed as

$$\frac{dV}{dt} = m_v \frac{\partial q}{\partial t}dxdydz \tag{2.14}$$

m_v is the confined compressability of the element. Over time, the total stress q is absorbed by the soil skeleton which leads to the effective stresses increasing with the decrease of excess pore pressure. This leads to:

$$\frac{dV}{dt} = -m_v \frac{\partial p}{\partial t}dxdydz \tag{2.15}$$

Now, combining equation 2.13 and 2.15 into:

$$m_v \frac{\partial p}{\partial t} = \frac{k}{\gamma_w}\frac{\partial^2 p}{\partial y^2} \tag{2.16}$$

or expressed as:

$$\frac{\partial p}{\partial t} = C_v \frac{\partial^2 p}{\partial y^2} \tag{2.17}$$

The unit of $c_v = \frac{k}{m_v \gamma_w}$ is called the coefficient of consolidation which in this thesis has the denotation $\frac{m^2}{year}$. Following the assumption that the coefficient of permeability and compressability of volumetric change is constant, c_v remains constant throughout the process. The consolidation equation is uncoupled due to the differential equations only depending on one variable.

2.2.2 Initial- and boundary conditions

It is assumed that the load (q) is constant throughout the process ($t > 0$) and defines the initial conditions for both the single and double drained as:

$$p = p_0 = q \quad \}\, \mathbf{t = 0} \tag{2.18}$$

The boundary conditions applied to the problems of consolidation are twofold. The single drained case, where the top is open permeable boundary and the bottom closed impermeable boundary. In the double drained case, both boundaries are open and permeable. There are two different boundary conditions applied to these problems: Dirichlet and Von Neumann (Tveito and Winther, 2009). The boundary conditions for the single drained case is

$$\left.\begin{array}{ll} y = 0: & p = 0 \\ y = h: & \frac{\partial p}{\partial z} = 0 \end{array}\right\} \mathbf{t > 0} \tag{2.19}$$

where the top boundary (z=0) is the Dirichlet boundary condition which is open and the bottom boundary (z=h) is closed due to the Von Neumann boundary condition. For the double drained case both boundaries are open the boundary conditions are:

$$\left.\begin{aligned} y = \frac{h}{2}: &\quad p = 0 \\ y = -\frac{h}{2}: &\quad p = 0 \end{aligned}\right\} \mathbf{t} > \mathbf{0} \tag{2.20}$$

The difference between the two cases is apparent in terms of the boundary conditions as well as the different values for h. For the double drained case the depth of the domain is 2h and the single drained case it's h. For the open layered (double drained) case, the center of the sample is located at y = 0. This is due to the symmetry around the center line for which there is no flow, which in turn provides is what provides the closed off boundary for the half - layered case (single drained). An illustration of the problems is shown in figure 2.7. In the center figure is what is called the isochrone curves which are a product of plotting the porewater pressures p against z for different values of t. The factor of $T_v = \frac{c_v t}{d^2}$ is called the Time factor and is a dimensionless number. Note that the values for T_v in this case are pure approximate guesses for illustrative purposes and does not correlate directly to the cases in question later in the thesis. The value of T_v is called the dimensionless time factor.

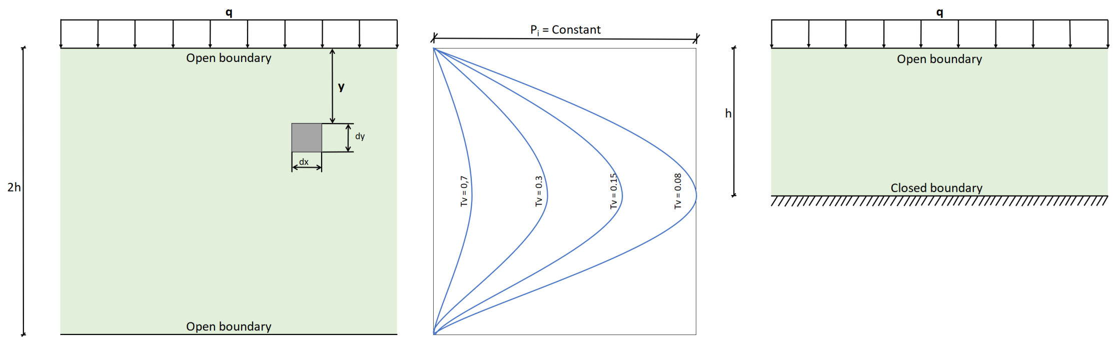


Figure 2.7: Illustration of the consolidation problems

2.2.3 Analytical solution

The solution to the consolidation equation, equation 2.17, (Verruijt, 2013) is solved using Laplace transform:

$$\overline{p} = \int_0^\infty p * e^{-st} dt \tag{2.21}$$

which transforms the equation 2.17 into

$$\frac{d^2\overline{p}}{dy^2} = \lambda^2(\overline{p} - \frac{p_0}{s}) \quad \textbf{with} \quad \lambda^2 = \frac{s}{C_v} \tag{2.22}$$

The solution to equation 2.22 is

$$\overline{p} = \frac{p_0}{s} + A\cosh{(\lambda z)} + B\sinh{(\lambda z)} \tag{2.23}$$

In this solution, the integration constants A and B are independent z. The constant B = 0 due to the lower boundary condition in equation 2.19, which leads to A being $\frac{-p_0}{\cosh \lambda h}$. This gives the equation

$$\overline{p} = \frac{p_0}{s} - \frac{p_0 \cosh{(\lambda y)}}{s \cosh{(\lambda y)}} \tag{2.24}$$

The second term of this equation has a pole at s = 0, and subsequent poles when $\cosh{(\lambda h)}$ is equal to zero. λ and s are zero when

$$\lambda_k = (2k-1)\frac{i\pi}{2h} \quad and \quad s_k = -(2k-1)^2 \frac{c\pi^2}{4h^2}, \quad k = 1, 2, 3.... \tag{2.25}$$

The first term only has one pole which is at s = 0 and the residual being p_o. Concerning the second term, when s = 0, the residual is $-p_0$ which cancels out the first term. When s = s_k, the residual is

$$Res_k = [-\frac{p_0 \cosh{(\lambda y)} e^{st}}{d[s \cosh{(\lambda h)}]/ds}]_{s=sk} \tag{2.26}$$

Due to the fact that

$$\frac{d}{ds}\cosh{(\lambda h)} = h \sinh{(\lambda h)} \frac{d\lambda}{ds} \quad and \quad \frac{d\lambda}{ds} = \frac{d\sqrt{s/c_v}}{ds} = \frac{1}{2\sqrt{sc_v}} = \frac{1}{2\lambda c_v} \tag{2.27}$$

Using equations 2.25 it gives

$$Res_k = \frac{4p_0}{\pi}\frac{(-1)^{k-1}}{2k-1} \cos{[(2k-1)\frac{\pi y}{2h}]} \exp{[(-2k-1)^2 \frac{\pi^2}{4}\frac{c_v t}{4h^2}]} \tag{2.28}$$

The complete normalized solution for the single drained case is

$$\frac{p}{p_0} = \frac{4}{\pi}\sum_{k=1}^{\infty} \frac{(-1)^{k-1}}{2k-1} \cos[(2k-1)\frac{\pi y}{2h}] \exp[-(2k-1)^2 \frac{\pi^2 c_v t}{4h^2}] \tag{2.29}$$

The solution to the double drained case is

$$\frac{p}{p_0} = \frac{4}{\pi}\sum_{k=1}^{\infty} \frac{(-1)^{k-1}}{2k-1} \cos[(2k-1)\frac{\pi y}{h}] \exp[-(2k-1)^2 \frac{\pi^2 c_v t}{h^2}] \tag{2.30}$$

The difference between the two solutions is due to the difference in the domain with one being open layered and the other being half layered as shown in figure 2.7.

2.3 Bearing capacity

The inspiration for this part of the thesis came from the article "Complete limiting stres solutions for the bearing capacity of strip footings on Mohr-Coulomb soil" (Smith, 2005). Bearing capacity within geotechnical engineering is almost present in every analysis, and is characterized by the search for the peak pressure, q_{ult}, that the soil can withstand. There are two types of bearing capacity problems, undrained and drained analysis. Undrained analysis is short-term analysis of the bearing capacity of the soil. Drained is long-term. In this thesis, only undrained will be examined.

The chapter starts with an explanation of the material model before continuing to the governing PDE for the system, thereby followed by the derivation of the yield function equations for the yield functions for both the undrained and drained case. It finishes with a short introduction to the failure zones produced by q_{ult}

2.3.1 Mohr-coulomb soil

Previously, the theory of one-dimensional consolidation was within the elastic range. In order to explain the Mohr - Coulomb model (Craig and Knappett, 2020b), the concept of elastic - perfectly plastic material has to be introduced. As shown in figures 2.8 and 2.9, which are adapted from the source material, the material model for the elastic - perfectly plastic behavior is highly idealized compared to the actual relationship between stress and strain in the soil soil. From figure 2.8, E represents the elastic part of the curve, Y is the yield, F is the failure and P the line for plastic strain, also called plastic flow. The concept of plastic flow is shown in 2.10, where if the load is applied as an increment shown in A, it will force the stresses back down to the failure surface shown in line Y - P. If the load increment is as B, the response is elastic.

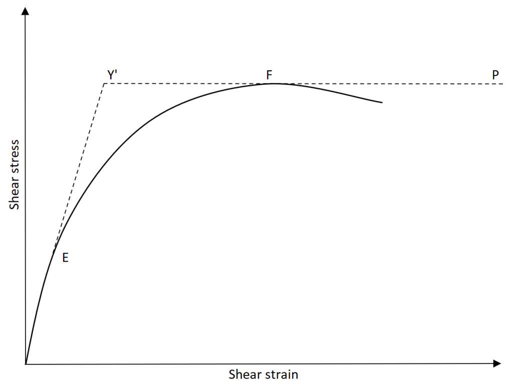


Figure 2.8: Actual relationship

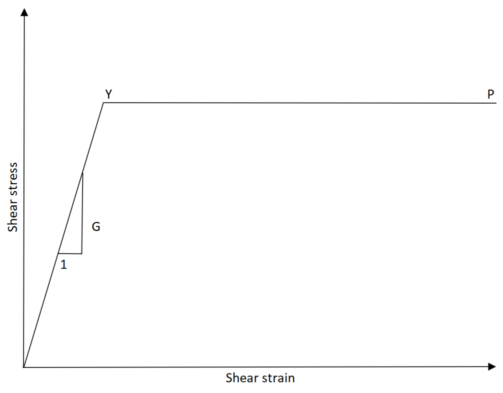


Figure 2.9: Elastic - Perfectly plastic

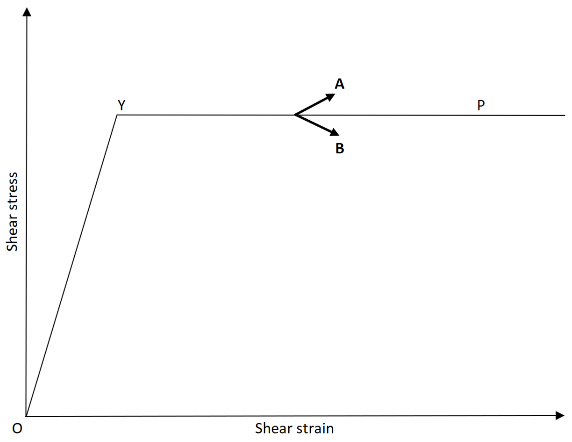


Figure 2.10: Plastic flow

If the load increments are cumulatively large enough that they reach the yield surface, the soil will obviously yield. The associated yield function, also called Tresca citerion, can be written both in terms of principal stresses and stress components.

For the undrained soil, this yield function will be derived later. Although standard Mohr - Coulomb soil notation usually uses σ_1 and σ_3 for major and minor stress component. However, within this thesis the notation of minor and major principal stress is denoted as σ_x and σ_y.

2.3.2 Equations of equilibrium in two dimensions

Previously the problems of consolidation has been in one-dimension with only vertical stress. The bearing capacity problem is in two dimensions. Within a typical element of soil, the loading would be vertical, due to the self weight, and horizontal due to the applied loading externally (e.g a foundation). This may additionally introduce an applied shear stress to the element. The governing partial differential equations for such a two dimensional system both for total and effective stresses in equilibrium is given in Craig's soil mechanics (Craig and Knappett, 2020b) and shown in equation 2.31. Further, a visual representation of the two dimensional element in equilibrium is shown in figures 2.11 and 2.12.

$$
\begin{aligned}
&\frac{\partial \sigma_x}{\partial x} + \frac{\partial \tau_{yx}}{\partial y} = 0 &\qquad& \frac{\partial \sigma_{x'}}{\partial x} + \frac{\partial \tau_{yx}}{\partial y} = 0 \\
&\frac{\partial \tau_{xy}}{\partial x} + \frac{\partial \sigma_y}{\partial y} - \gamma = 0 && \frac{\partial \tau_{xy}}{\partial x} + \frac{\partial \sigma_{y'}}{\partial y} - (\gamma - \gamma_w) = 0
\end{aligned}
\tag{2.31}
$$

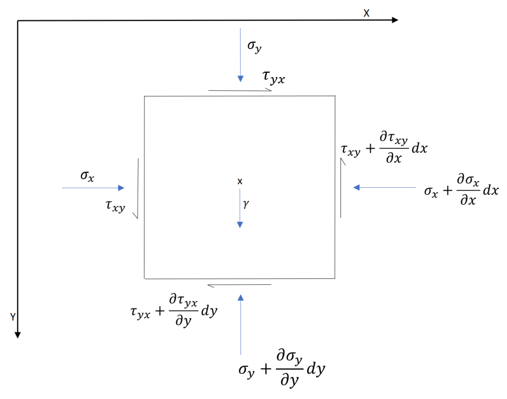

x
Y
σ_y
τ_{yx}
σ_x
τ_{xy}
x
γ
$\tau_{xy} + \frac{\partial \tau_{xy}}{\partial x} dx$
$\sigma_x + \frac{\partial \sigma_x}{\partial x} dx$
$\tau_{yx} + \frac{\partial \tau_{yx}}{\partial y} dy$
$\sigma_y + \frac{\partial \sigma_y}{\partial y} dy$

Figure 2.11: Total stress

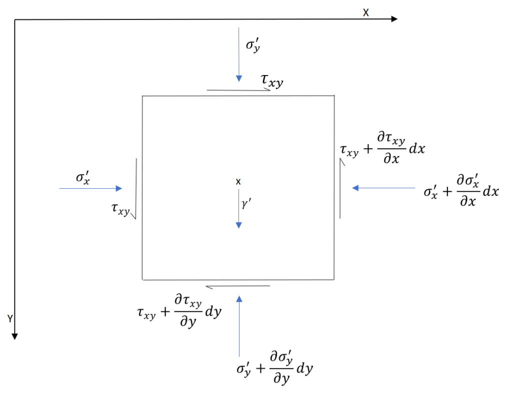

x
Y
σ_y'
τ_{xy}
σ_x'
τ_{xy}
x
γ'
$\tau_{xy} + \frac{\partial \tau_{xy}}{\partial x} dx$
$\sigma_x' + \frac{\partial \sigma_x'}{\partial x} dx$
$\tau_{xy} + \frac{\partial \tau_{xy}}{\partial y} dy$
$\sigma_y' + \frac{\partial \sigma_y'}{\partial y} dy$

Figure 2.12: Effective stress

2.3.3 Failure zones

In bearing capacity problems there are three different modes of failure (Barnes, 2016), general, local and punching shear, with general shear being the main focus in this thesis. General shear has a few characteristics:

- The continuous slip surface is well defined and reaches ground level
- On both sides of the foundation, there occurs heaving with a tilt to one side after the final collapse.
- The failure is catastrophic and sudden with the ultimate value for the load being peak value.

For the undrained failure zone, the shape of general shear to the left side of the foundation is

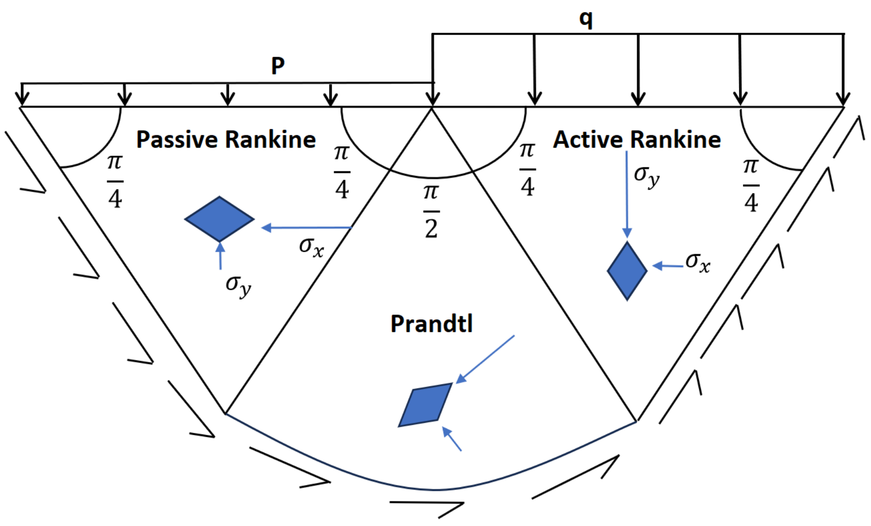


Figure 2.13: Failure zone - Undrained

There are three zones that form when general shear occurs - active and passive Rankine and Prandtl zone. The active and passive Rankine zones are characterized with the stress orientation being constant throughout the zone. For the active Rankine, the major principal stress is vertical which pushes the soil down to the force on the foundation q. In the Prandtl zone, the stress orientation is not constant and rotates through the zone which pushes the soil sideways. In the passive Rankine zone, the stress is rotated to such a degree that the major principal stress is horizontal, and for this zone, the stress is also constant throughout. This leads to the soil being pushed sideways and up.

For the undrained analysis, it is known that the maximum shear stress coincides with the critical plane which is on an orientation of $\frac{\pi}{4}$ for the active and passive Rankine zones. This leads to the Prandtl zone having the shape of a circle, which is what gives it the opening $\frac{\pi}{2}$. The active and passive Rankine zones are represented in a Mohr's circle as

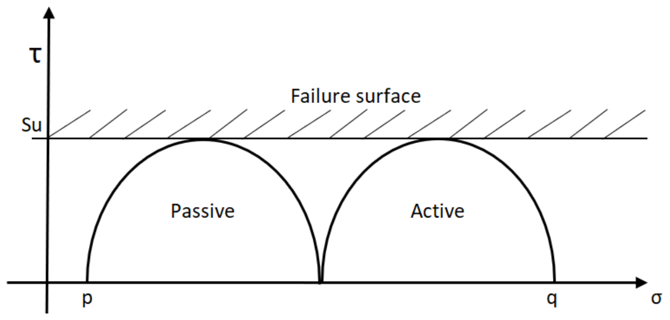


Figure 2.14: Undrained Mohr's circle

2.3.4 Yield function for undrained Mohr-Coulomb soil

The simples way to derive the yield function for undrained soil is to represent the major and minor principal stresses of undrained soil in a Mohr's circle. For the undrained analysis, the yield surface is horizontal to the undrained shear strength (s_u) exhibited by the soil. Following the figure of the failure zone for the undrained case, the s_u value is equal to the critical shear strength τ_{max} that occurs along the three zones border. With q being the major principal stress and p being the minor - it is possible to use Pythagoras in order to derivative the yield function, as is shown equations 2.32. Note that R = s_u'. Note that the yield function can also be referred to as the Tresca criterion.

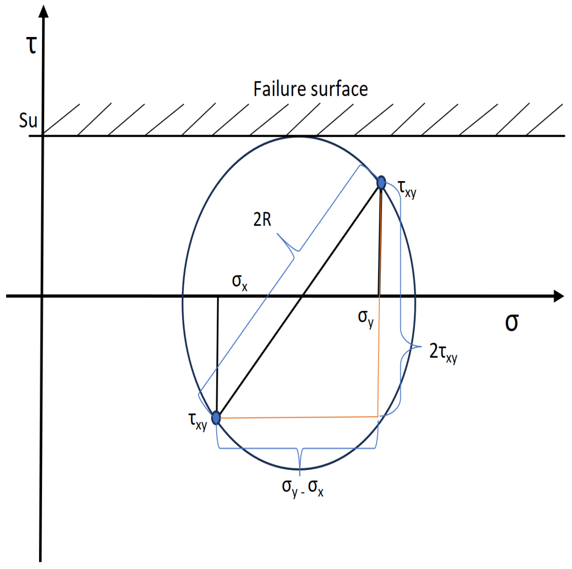


Figure 2.15: Yield function - Undrained

$$(2s_u)^2 = (\sigma_y - \sigma_x)^2 + (2\tau_{xy})^2$$

$$4s_u^2 = (\sigma_y - \sigma_x)^2 + 4\tau_{xy}^2 \tag{2.32}$$

$$\frac{1}{4}(\sigma_x - \sigma_y)^2 + \tau_{xy}^2 - s_u^2 = 0$$

3 Software

This section describe the software that has been used either to implement or validate the case studies in this thesis. It is divided into two subsections: Finite element software, which has been used for validation and programming applied for implementation of the physics - informed neural networks.

3.1 Finite element software

3.1.1 Plaxis 2D Ultimate

Plaxis 2D is a finite element software created by Bentley systems (“Plaxis 2D”, n.d. Plaxis can perform analysis of both stability and deformation with respect to geotechnical processes. Plaxis 2D uses one procedure for input where it applies a CAD -like drawing system for easy definition of loads type and element type, and one procedure for output where it gives detailed presentation of the results from the computation. Plaxis’s easy-to-use systems along with it’s fast computational powers and leading industry standard is the reason is was used in this thesis.

3.2 Programming software

3.2.1 Spyder

Spyder is a open source integrated development envirometn (IDE) for scientific programming in python (“Spyder”, n.d. This framework has been used since it provides multiple IPython consoles. Since machine learning is a lot of trial and error, the easy comparison of plots with older plots in an IPython console have been especially handy along with its debugging tool.

3.2.2 DeepXDE

DeepXDE (L. Lu et al., 2021) is a library used for scientific machine and physics-informed learning. DeepXDE has been applied in the problems of this thesis due to its easy implementation of complex problems in compact code, especially for PINNs. The implementation of code into DeepXDE also resembles closely the mathematical formulation of the problem making the code easy to understand if the reader has an understanding of the mathematical basis of the problems. DeepXDE supports backends for tensor libraries such as Tensorflow as well as others.

3.3 Packages

Within DeepXDE, the tensorflow backend has been applied. Tensorflow (“Tensorflow”, n.d.) is an open source platform for machine learning. Other packages that has been used is NumPy and Matplotlib. NUmPy (“NumPy”, n.d.) is used for production and manipulation of arrays in the code. Matplotlib (“Matplotlib”, n.d.) has been used for the plots of predictions by the network and analytical solutions.

4 Method

Following is the section with a general description of the workflow for the problems of PINN within DeepXDE in consolidation. The code applied will be general and simplified just for explanatory purposes alone. Since the purpose is to explain workflow, the examples used are taken from the problems of consolidation as these are the simplest.

4.1 Creation of datasets and sampling method

4.1.1 Forward analysis

A PINN implemented into DeepXDE will be constrained by a PDE and the constraints on the boundary and initial conditions. For the forward analysis DeepXDE uses one of several training distributions to sample training points both in the domain and on its boundaries and/or initial points. The two methods applied are as indicated in the chapter on theory; uniform and Latin hypercube sampling. This means that DeepXDE does not require an external dataset as a target for its neural network since it uses the PDE as a constrain within the domain for the PDE or PDEs in question. An illustration of a the way data is sampled for training is provided in the following figure where beige are the training points for the domain, blue is for the boundary points and orange are the initial conditions.

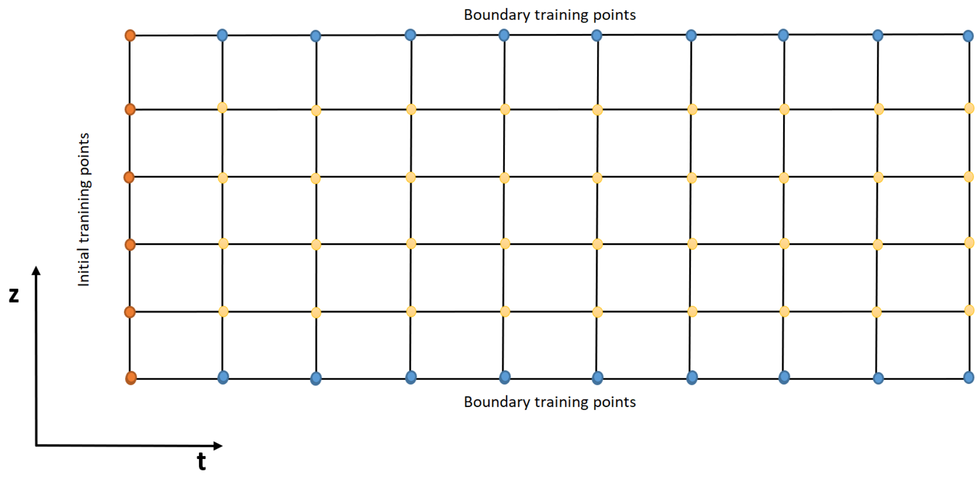


Figure 4.1: Training points for the domain

4.1.2 Inverse analysis

In order to perform an inverse analysis, the model is reliant on external data in the form of imported values. These training sampling points become points of constraint for the model which it uses to train an external trainable variable. It finds the parameters λ that describes the observed data best. Contrary to the forward analysis, these observed data are chosen at random within the network.

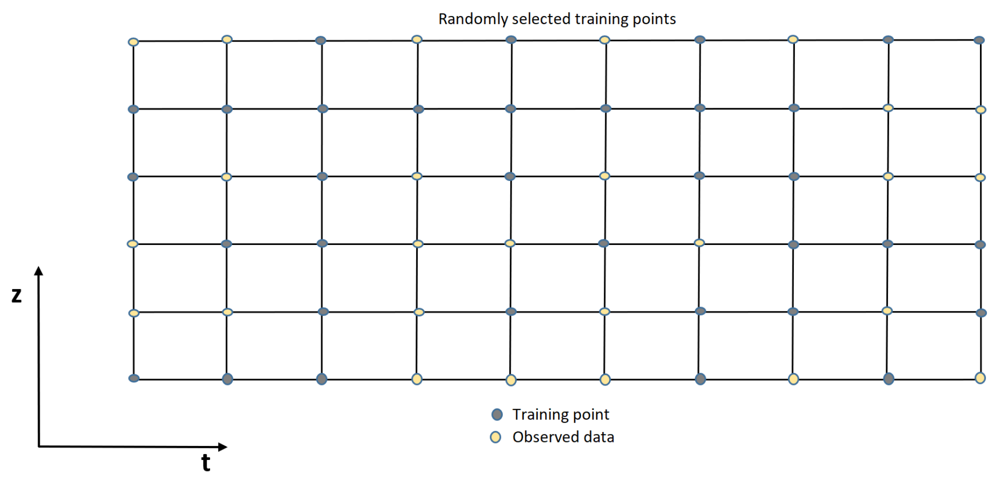


Figure 4.2: Illustration of sampling points for inverse analysis

4.2 Defining partial differential equation and initial and boundary conditions

4.2.1 PDE

```python
def PDE(x,y):
    du_t = dde.grad.jacobian(y,x, j = 1)
    du_xx = dde.grad.hessian(y,x, j = 0)
    return du_t - du_xx
```

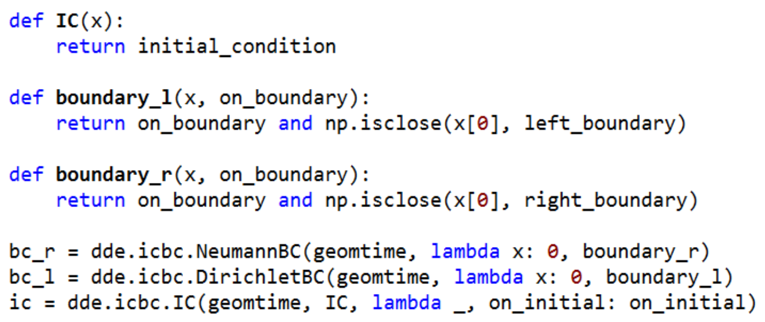
Figure 4.3: Simplified code for the PDE

Figure 4.3 shows a simple function for the diffusion equation in DeepXDE. It consists of two inputs, x and y. x is a set of columns containing the variables for the domain, be it in space or space - time. y has in these cases the same structure but contains the underived variables for the output. The two modules applied called jacobian and hessian referres to the Jacobian and hessian matrix for the first and second derivative respectively ("Jacobian and hessian matrix", n.d.). An example of the two matrices is provided in equation 4.1.

$$J_{ij} = \frac{dy_i}{dx_j} \qquad H_{ij} = \frac{d^2y}{dx_i dx_j} \tag{4.1}$$

4.2.2 Initial- and boundary conditions

Defining the initial- and boundary conditions requires an explanation of two functions that are applied to evaluate the location of said conditions: *on_boundary* and *np.isclose*(). An example is shown in the following figure:

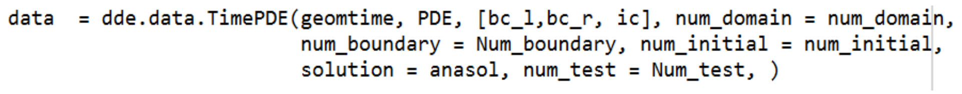


```
def IC(x):
    return initial_condition

def boundary_l(x, on_boundary):
    return on_boundary and np.isclose(x[0], left_boundary)

def boundary_r(x, on_boundary):
    return on_boundary and np.isclose(x[0], right_boundary)

bc_r = dde.icbc.NeumannBC(geomtime, lambda x: 0, boundary_r)
bc_l = dde.icbc.DirichletBC(geomtime, lambda x: 0, boundary_l)
ic = dde.icbc.IC(geomtime, IC, lambda _, on_initial: on_initial)
```

Figure 4.4: Simplified code of Initial- and boundary conditions

In DeepXDE, on_boundary is a boolean function that returns true or false depending on if the criteria is met. In order to ensure that the function does not exclude points that are very close to the boundary but not directly at - numpy.isclose is applied. Numpy.isclose is a boolean array that checks if conditions a and b are met. The default relative and absolute tolerance of this function ($rtol = 1e^{-5}, atol = 1e^{-8}$) is unchanged. Further after the functions for finding the location of both initial and boundary boundaries - they are given their boundary conditions using DeepXDE's built-in functions. The theory behind the boundary conditions is referenced in equation 2.18 and 2.20 for the Dirichlet and Von Neumann BC respectively. The functions for both Dirichlet and Von Neumann uses the geometry, or in this case geometry and time to define the domain. The lambda function whether or not the x is on the boundary or not which is evaluated by the function previously explained. For the initial conditions, the same is done using an in-house function for DeepXDE - on_initial. Just to clarify, if the boundary conditions were the same, the same function for the boundaries (on_boundary) would be applied since it evaluates the whole of the domain. But that is not the case in this thesis.

4.3 Compiling data for the PDE

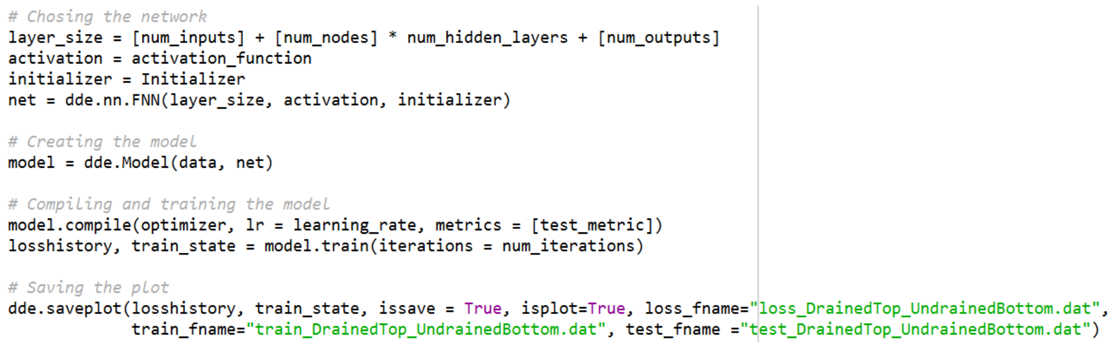


```
data  = dde.data.TimePDE(geomtime, PDE, [bc_l,bc_r, ic], num_domain = num_domain,
                         num_boundary = Num_boundary, num_initial = num_initial,
                         solution = anasol, num_test = Num_test, )
```

Figure 4.5: Simplified code for TimePDE

For DeepXDE, the data need to be compiled into one. For consolidation this is done by using the class module TimePDE. An example is provided in figure 4.5. The attentive reader will recognise the variables shown for the PDE and initial- and boundary conditions. Geomtime is the compiled domain for geometry and time. The variables for $num\_initial$ and $num\_boundary$ specifies the number of sampling points on the given boundary to be used for traning. Num_test is the number of points to be excluded from the training set and applied to the test set. Solution = anasol is a reference solution for the PDE to be evaluated up against a user specified test-metric which is later specified in the compiling of the model in the next section.

4.4 Compiling model



```
# Chosing the network
layer_size = [num_inputs] + [num_nodes] * num_hidden_layers + [num_outputs]
activation = activation_function
initializer = Initializer
net = dde.nn.FNN(layer_size, activation, initializer)

# Creating the model
model = dde.Model(data, net)

# Compiling and training the model
model.compile(optimizer, lr = learning_rate, metrics = [test_metric])
losshistory, train_state = model.train(iterations = num_iterations)

# Saving the plot
dde.saveplot(losshistory, train_state, issave = True, isplot=True, loss_fname="loss_DrainedTop_UndrainedBottom.dat",
             train_fname="train_DrainedTop_UndrainedBottom.dat", test_fname ="test_DrainedTop_UndrainedBottom.dat")
```

Figure 4.6: Simplified code for the compiling the model

As shown in figure 4.6, DeepXDE allows for neural networks to be compiled with just a few lines of code using its in-house functions. Starting from the top, the layer_size defines the architecture of the neural network from the number of inputs and output, the number of nodes and hidden layers. The number of hidden layers is created using python notation for multiplication of list which gives . As an example: $[32] * 4 = [32, 32, 32, 32]$. The activation functions used in this thesis has been previously explained in figures 2.3c, 2.3b and 2.3a as well as their corresponding equations 2.4, 2.3 and 2.5. The initializer is a function that determines the how the initial weights are set for the hidden layers. Net is where the actual network is compiled. Seen in the code it uses *FNN* which stands for Fully-connected neural network which is explained in section 2.1.1. Model is simply the definition of the neural network that trains on data.

Model compile is the configuration of the model previously specified. The optimizer, which is Adam has previously been explained in the same section as fully - connected neural networks. The learning rate defines the step size when moving through the loss function. Not shown here is that within compile is where the loss function is defined. By default it is set to *MSE*. Metric is the metric which the training points taken by num_test is tested on. Further, the command model.train defines the number of iterations the network is supposed to run though the model. As seen, model.train returns the losshistory and the train_state. Using the in-house functions of saveplot, these can, depending on their boolean values, be saved or plotted.

5 Case studies

The goal for this thesis is as stated to see if physics-informed neural networks can predict both consolidation and bearing capacity. Since this thesis deals with two quite different problems they are separated by into their own sections.

5.1 Consolidation

The methods of implementation for consolidation was inspired by (Bekele, 2020). First presented is the forward analysis for the single and double drained case respectively. Followed by the inverse analysis for both of the cases and the chapter on consolidation ends with the both cases in forward and inverse analysis with added noise to the datasets. The sections of the chapter follows the same structure. First presented are the mathematical properties of the problem followed by the implementation of the problem into the neural network. Lastly, the results are presented before moving on to a small discussion/development of the results. The models here are run on a Nvidia Geforce 250MX GPU.

5.1.1 Case study 1: Forward analysis: Drained top and undrained bottom

Domain, boundary conditions and initial conditions

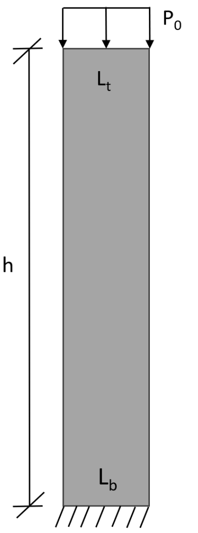


Figure 5.1: Domain for the Drained top Undrained bottom. Adapted from (Bekele, 2020)

The first case is the case with a drained boundary at the top and an undrained boundary at the bottom. As stated in the theory before, the Dirichlet boundary condition represents an open permeable boundary with drainage while the Von Neumann BC represents a closed impermeable boundary. In figure 5.1 is an illustration of the domain where L_t and L_b represents the top and bottom boundary. The boundary conditions are

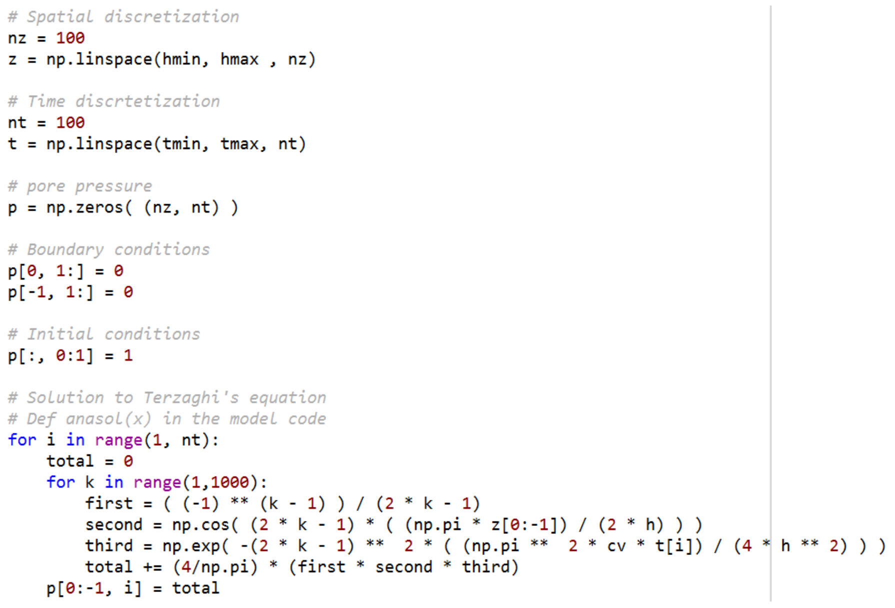
$$\left.\begin{aligned} L_t \quad & P = 0 \\ L_b \quad & \frac{\partial p}{\partial t} = 0 \end{aligned}\right\} \mathbf{t} > \mathbf{0} \tag{5.1}$$

The initial conditions are as shown in figure 5.1 as being $P = P_0 = q$ for t = 0. The value for h is called the drainage path which for the single drained case is equal to the height of the domain. In keeping with normalized values the variables for the spatial - temporal domain - the values for h and t are both set to [0,1]. The origin has in this case been chosen at the top meaning we have $0 \leq y \leq h$. Since the rate of dissapation of the excess porepressure can only happen through one boundary, the factor of c_v is chosen as $0.5\ \frac{m^2}{yr}$.

Neural network
The analytical solution, which is the test-metric, is implemented into the code for the model in such that it acts as a test-metric for the model to be evaluated against in addition to the test metric. The test-metric is evaluated using L2 relative error (L. Lu et al., 2021). The formula for L2 relative error is given as

$$L_2 \textbf{ Relative error} = \frac{y - \hat{y}}{y} \tag{5.2}$$

The In the case for forward analysis of 1D consolidation, the function for the test-metric is the analytical solution. In the code for the analytical solution used to create an array with the exact solution with the spatial and temporal discretization of $N_z = 100$ and $N_t = 100$ and the same limits for the variables z and t is shown here in the following figure. The value for the rate of consolidation has been chosen as $C_v = 0.5\frac{m^2}{yr}$ for the single drained case. The number of points for the test-metric is num_test = 10000.



```python
# Spatial discretization
nz = 100
z = np.linspace(hmin, hmax , nz)

# Time discrtetization
nt = 100
t = np.linspace(tmin, tmax, nt)

# pore pressure
p = np.zeros( (nz, nt) )

# Boundary conditions
p[0, 1:] = 0
p[-1, 1:] = 0

# Initial conditions
p[:, 0:1] = 1

# Solution to Terzaghi's equation
# Def anasol(x) in the model code
for i in range(1, nt):
    total = 0
    for k in range(1,1000):
        first = ( (-1) ** (k - 1) ) / (2 * k - 1)
        second = np.cos( (2 * k - 1) * ( (np.pi * z[0:-1]) / (2 * h) ) )
        third = np.exp( -(2 * k - 1) **  2 * ( (np.pi **  2 * cv * t[i]) / (4 * h ** 2) ) )
        total += (4/np.pi) * (first * second * third)
    p[0:-1, i] = total
```

Figure 5.2: Code for the analytical solution of the single drained case

In order to compress the text and have an easy way to show the specifics of the model, the choices for the the neural network is shown in table 5.1. I should be mentioned that this is the result of trail and error. Several activation functions were applied and with differing length and width of the neural network. A comparison between Tanh and Sigmoid activation function will be presented in the discussion. The results shown in table 5.1 the chosen variables for this presentation.

Hidden Layers	Nodes	Activation	Initializer	Optimizer	LR	Iterations
3	32	Sigmoid	Glorot uniform	Adam	0.001	20 000

Table 5.1: Neural network for the forward analysis of the single drained case

The sampling of training data for the forward problem involves the initial- and boundary values of z and t in order to predict the excess pore pressure values p(z,t) at the given sampling point. Using mean square error, the training loss is calculated as:

$$MSE_p = \frac{1}{N}\sum_{k=1}^{N} |p(z_k, t_k) - \hat{p}(z_k, t_k)|^2 \tag{5.3}$$

where the total number of training data are N and (z_k, t_k) is the point of the training data. The governing PDE, equation 2.17, that provides the physical constraint is sampled at randomly generated collocation points. For the forward cases, a uniform sampling strategy is applied. The constraint by the PDE is evaluated at the collocation points using the predicted excess pore pressure and automatic differentiation:

$$f_c = \frac{\partial \hat{p}}{\partial t_c} - C_v \frac{\partial^2 \hat{p}}{\partial z_c^2} \tag{5.4}$$

The loss from this constraint is also calculated using the mean square error:

$$MSE_c = \frac{1}{N_c}\sum_{k=1}^{N_c} |f_c(z_{ck}, t_{ck})|^2 \tag{5.5}$$

Here N_c is the total number of collocation points, and (z_{ck}, t_{ck}) represents a collocation data point. The total loss from training is thus:

$$MSE = MSE_p + MSE_c \tag{5.6}$$

This is more or less the same way that the loss function is for all the cases in this thesis, with small modifications. For the forward problem, the number of points sampled in each iteration is num_domain = 400, num_boundary = 200 and num_initial = 100. The loss function for the forward consolidation analysis is:

$$\begin{aligned}
L_{Tot} &= W_{PDE}L_{PDE} + W_{BC}L_{BC} + W_{IC}L_{IC} \\
L_{PDE} &= L_{PDE} \\
L_{BC} &= W_{BC1}L_t + W_{BC2}L_b \\
L_{IC} &= L_{IC}
\end{aligned} \tag{5.7}$$

Result

The network described in the previous section was used to train the model. The results are presented in the following tables and figures:

Training loss	Test loss	Test metric	Training time
$7.44e-3$	$8.01e-3$	$7.38e-1$	142.53 sec

Table 5.2: Losses and time for single drained

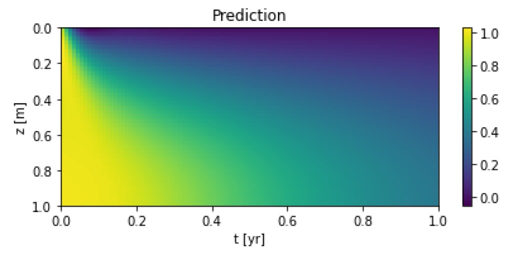


Figure 5.3: Prediction-Single drained

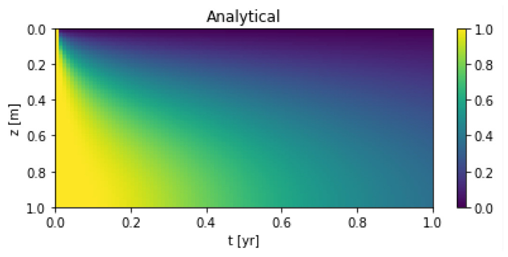


Figure 5.4: Analytical-Single drained

Discussion

Since this was the simplest and original case, it bore the brunt of frustration when it came to the optimization in terms of changing the parameters. As can be seen when comparing the two plots in figures 5.3 and 5.4, they show a large degree of similarity even after a relatively short training period. But looking at the left side of the plot, the initial conditions, it becomes apparent that the model cannot accurately predict the same initial conditions as they are in the analytical model. At first the Tanh activation function was applied, but after testing with other parameters like the ReLU activation function, which gave bad results because it cannot calculate second derivatives, and the Sigmoid function, it became apparent that the Sigmoid function gave a smoother transition between the zones as well as having a more closely related output to the analytical solution than the Tanh function. Tanh seemed to underpredict the dissipation. A comparison of the activation functions with the analytical solution is shown in figures:

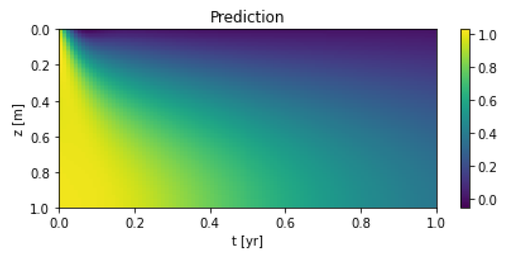


Figure 5.5: Prediction-Sigmoid

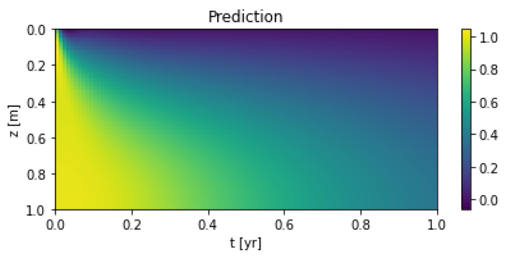


Figure 5.6: Prediction-Tanh

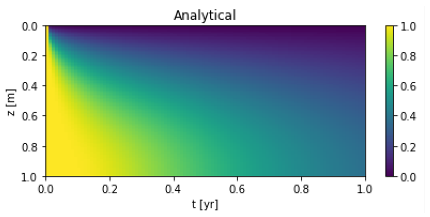


Figure 5.7: Analytical - Single drained

Sigmoid performs better when comparing the initial conditions alone. Tanh however shows a more similar decline in excess pore pressure after the initial conditions. It should be noted that the differences are small. Since DeepXDE allows the user to determine the number of points taken at boundary, domain and initial - a test was done with a different value for num_initial with both the Tanh and Sigmoid function in order to see if it improved the results. First num_initial was increased to 300. Then the test was done with an extreme example by multiplying num_initial = 100 by ten. These values where chosen by trial and error in order to find the most optimal value. The results are:

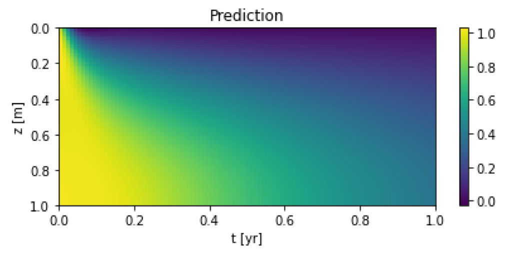


Figure 5.8: Sigmoid:num_init=300

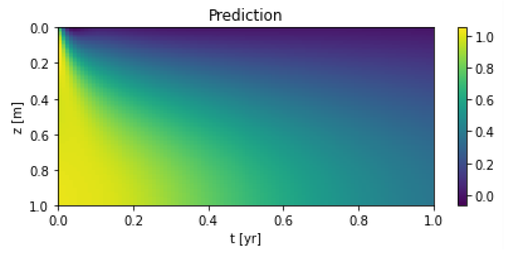


Figure 5.9: Tanh:num_init=300

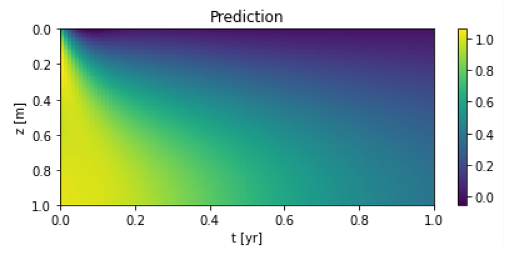


Figure 5.10: Sigmoid:num_init=1000

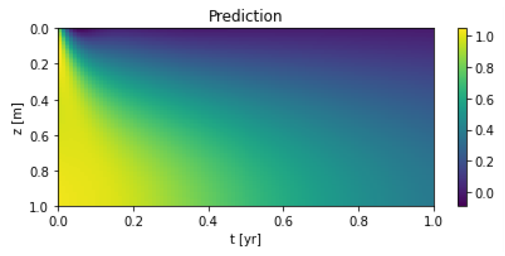


Figure 5.11: Tanh:num_init=1000

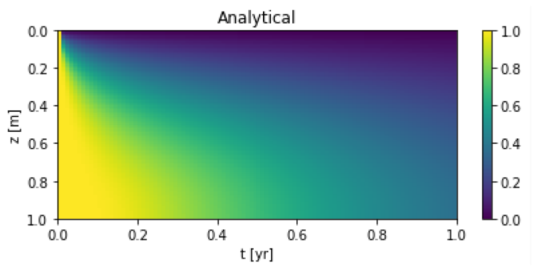


Figure 5.12: Analytical

As seen from the plots the best results in the authors opinion is the Sigmoid function with num_initial = 300. It can't reproduce the hard boundary conditions like the analytical solution can but the plots looks close to identical after that fact. It's also shown that if the variable for num_initial is raised to an extreme, it supersedes the other training points and makes the prediction for the domain less correlated to the analytical solution.

As a last observation is the colourbar besides the plots for which both exceeds the analytical range $[0, 1]$. The Sigmoid function in general produces output values closer to the original interval than the Tanh function. The exception being for num_initial = 1000 where they give similar values.

5.1.2 Case study 2: Forward analysis: Drained top and bottom

Domain, boundary conditions and test-metric

Concerning the domain for the drained top and bottom, the only thing that changes are the boundary conditions and the height of the domain. As previously explained in figure 2.7, the model starts at y = 0 and the height of the model ranges from $\frac{-h}{2} \leq y \leq \frac{h}{2}$ since there is no flow along the center line. This differs from what is shown in figure 2.7, but this domain was chosen for numerical reasons in an attempt to keep the values for the domain between [0,1]. In figure 5.13 the dotted line represents the center line. Also shown is the open boundary at the bottom which means the present boundary conditions are Dirichlet at both boundaries as shown:

$$\left.\begin{array}{ll} L_t & P = 0 \\ L_b & P = 0 \end{array}\right\} \mathbf{t > 0} \tag{5.8}$$

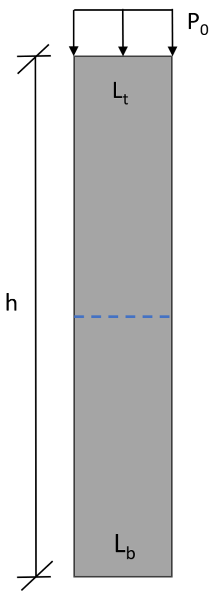
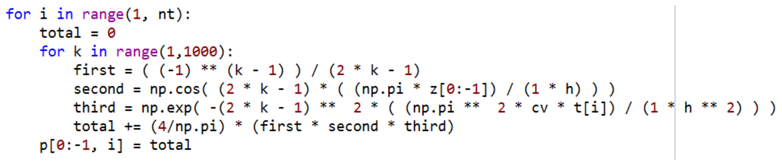


Figure 5.13: Domain:Drained top and bottom. Adapted from (Bekele, 2020)

Concerning the test-metric, the solution changes due to y being place at center. This gives the function in figure 5.14. The values for output of this function as well as the network are again normalized. Since this case has two open boundaries where the excess pore pressure can dissipate through. Because of the excess pore pressure having two borders to escape through, the rate of consolidation is faster than the single drained case. Therefore it is chosen as $C_v = 0.2\frac{m^2}{yr}$ for the double drained case.



```
for i in range(1, nt):
    total = 0
    for k in range(1,1000):
        first = ( (-1) ** (k - 1) ) / (2 * k - 1)
        second = np.cos( (2 * k - 1) * ( (np.pi * z[0:-1]) / (1 * h) ) )
        third = np.exp( -(2 * k - 1) **  2 * ( (np.pi **  2 * cv * t[i]) / (1 * h ** 2) ) )
        total += (4/np.pi) * (first * second * third)
    p[0:-1, i] = total
```

Figure 5.14: Analytical solution for double drained case

Neural network

For the specifics of the neural network, they are the same as for the previous case, shown in table 5.1. The loss function becomes slightly simplified since the double drained case has the Dirichelt BC on both boundaries. Carried over from the single drained case is the change sampling points: num_initial = 300. For the other sampling numbers, they remain the same num_domain = 400, num_boundary = 200 and num_test = 10000.

$$\begin{aligned} L_{Tot} &= W_{PDE}L_{PDE} + W_{BC}L_{BC} + W_{IC}L_{IC} \\ L_{PDE} &= L_{PDE} \\ L_{BC} &= L_{BC} \\ L_{IC} &= L_{IC} \end{aligned} \tag{5.9}$$

Results

As before, the results are as follows:

Training loss	Test loss	Test metric	Training time
$5.97e-3$	$7.68e-3$	$4.93e-1$	136.62s

Table 5.3: Losses and time for double drained

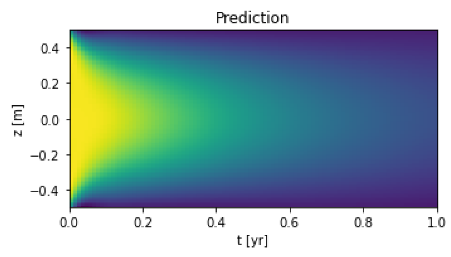


Figure 5.15: Prediction DD

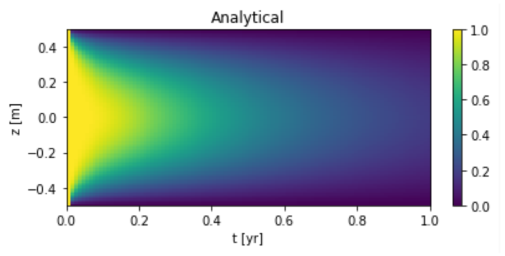


Figure 5.16: Analytical DD

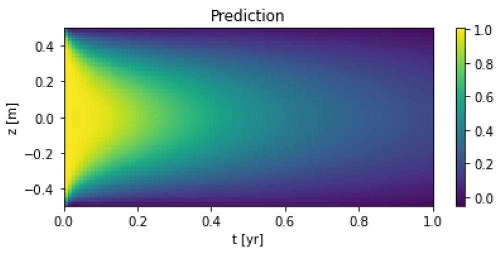
Discussion

As with the single drained case, the same problem occurs, meaning the initial conditions. Both the Sigmoid and Tanh function were tested and check what produced the best results, but again, the Sigmoid function gave the best approximation to the analytical results overall. It is however noted that in this case, it seems to be that the model predicts a lower degree of dissipation over time than the analytical solution along the boundaries and end of time. In order to test if this was a matter of number of iterations, a convergence study was done for this case. The only thing changed in this study is the number of hidden layers and nodes.

Hidden layers	Nodes	Training loss	Test loss
3	32	$5.97e-3$	$7.68e-3$
3	64	$1.63e-2$	$1.69e-2$
5	32	$6.29e-3$	$7.29e-3$
5	64	$3.78e-3$	$5.68e-3$
Iterations = 35 000			
5	32	$3.84e-3$	$7.34e-3$

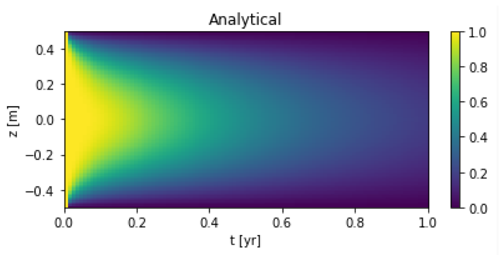
Table 5.4: Convergence study-double drained

As seen from table 5.4 the loss does not decrease with a significant amount due to the added number of hidden layers and nodes. Since the best results was from five hidden layers and 32 nodes, an additional analysis was made with 35 000 iterations shown in the last row of table 5.4. The results from the loss function are not noticeably improved compared to the others suggesting that the model has indeed reached a convergence and the curves flattened out. Looking at the comparative plots however, they are more closely comparative compared to before, except for the initial conditions of course. The results have improved with adjustments but worth noting is that for the normal analysis of three hidden layers with 32 nodes in each averaged around 140 seconds for the training of the model, while with five hidden layers and 32 nodes in each with 35 000 iterations - the time to train was 720 seconds.



[h]

Figure 5.17: Prediction DD: 35 000 iterations



[h]

Figure 5.18: Analytical DD

5.1.3 Case study 3: Inverse analysis for drained top and undrained bottom

Domain, boundary conditions and import of data

For the inverse analysis of the single drained case, it is the same as for the forward analysis in terms of domain and boundary conditions. What does change is that the inverse analysis does not use collocation points in the same way to approximate a solution. It uses imported data to train the model while approximating an external variable within the physical constraint defined to the model. More on that later. The imported data is created using the analytical solution, same as before with a 100x100 matrix consisting in a total of 10 000 points that can be sampled for each variable (z,t,p). The sampled data for inverse analysis is usually larger than for the forward analysis and in this case 2000 points are sampled from the imported data at random.

The imported data which consists of the combined vertical columns of the spatial - temporal variables z and t and the excess pore pressure p, is fed to DeepXDE through a function called PointSetBC which compares the output, that is associated with the points, with the target data, meaning the imported points, using a Dirichlet BC. These values are then evaluated up agains the values for z and t which are fed to the model as Anchors. During training of the model, these are hard constraints that must be satisfied.

Neural network

In order to ensure that the sampled points are evenly spread out, the Latin Hypercube sampling function was applied. The use of LHS also decreases the computation time due to it picking points that are the only ones in each axis-aligned hyperplane which contains it. The trainable value from the original training data will be the coefficient of consolidation like:

$$f_c = \frac{\partial \hat{p}}{\partial t} - C_{vt}\frac{\partial^2 \hat{p}}{\partial z^2} \tag{5.10}$$

where the subscript $_t$ represents trainable. The value for C_{vt} is initially set at $1.0\frac{m^2}{year}$ and the target value for C_{vt} is $0.5\frac{m^2}{year}$. Although the Sigmoid function has given the best results for the forward analysis - the inverse analysis is more timeconsuming. The Tanh function is slightly less complex than Sigmoid, and was thus chosen for the inverse analysis. The remaining specifics of this neural network is gathered in the following table and are unchanged from the previous neural nets.

Hidden Layers	Nodes	Activation	Initializer	Optimizer	LR	Iterations
3	32	Tanh	Glorot uniform	Adam	0.001	20 000

Table 5.5: Neural network for the inverse analysis of the single drained case

The loss function for the inverse single drained consolidation analysis is

$$
\begin{aligned}
L_{Tot} &= W_{PDE}L_{PDE} + W_{BC}L_{BC} + W_{IC}L_{IC} + W_{Data}L_{Data} \\
L_{PDE} &= L_{PDE} \\
L_{BC} &= W_{BC1}L_t + W_{BC2}L_b \\
L_{IC} &= L_{IC} \\
L_{Data} &= L_{Data}
\end{aligned}
\tag{5.11}
$$

Result

The results are shown below. A short note is that the final value for the predicted C_{vt} was only printed with six decimals which is why the error is given as being less than $1e-6$. Also note that the error for the variable c_v is calculated by the following equation:

$$
e = \frac{|\hat{p} - p|}{|p|} \tag{5.12}
$$

Training loss	Test loss	Training time	Final Variable	Error
$4.92e-3$	$4.92e-3$	336.71s	0.50000	$e < 1e-6$

Table 5.6: Losses and time for inverse single drained

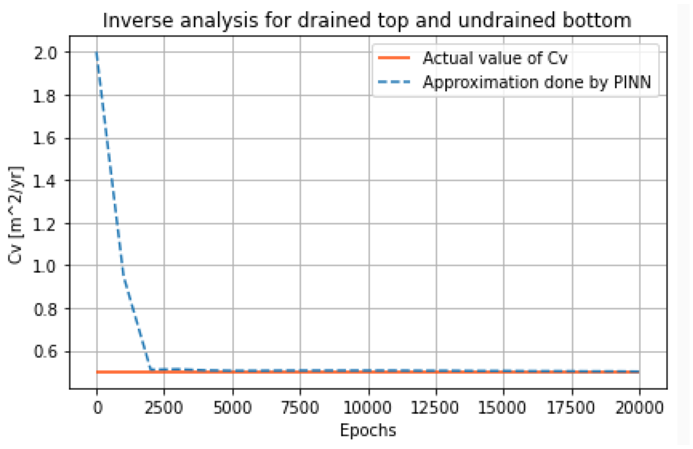


Figure 5.19: Inverse analysis for the single drained case

Discussion

In the case of the inverse analysis for the same problem, the network converged towards a different solution when fed the analytical solution as it was created. But through trial and error, the matrix was fed to the inverse model in the same way as it was outputted by the forward model, by rotating the matrix for the analytical solution 270 degrees. After doing so, the network converges towards the correct solution for both high positive values and negative values that are not to extreme.

In order to prove that the inverse analysis for this case works - some results with different initial values and consolidation factors are given in the plots 5.20, 5.21, 5.22 and 5.23. These plots show the extremes of the model. It can approximate the solution from very high positive values but does not converge for very low negative values.

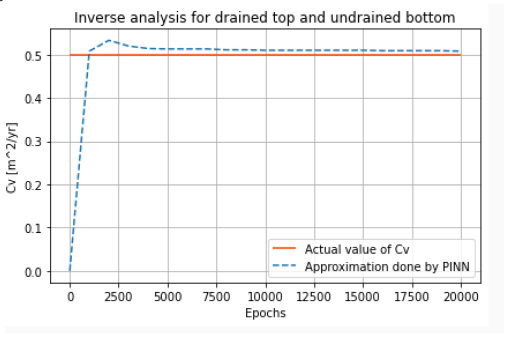


Figure 5.20: Initial value = 0

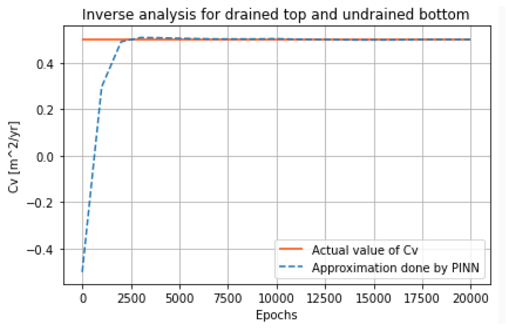


Figure 5.21: Initial value = -0.5

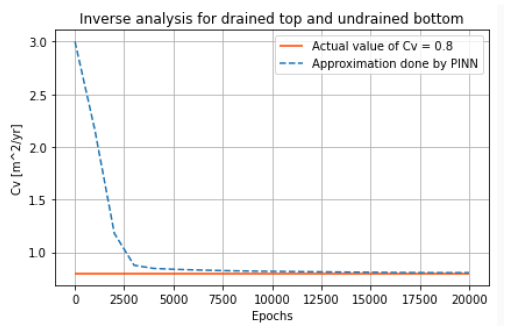


Figure 5.22: initial val = 3

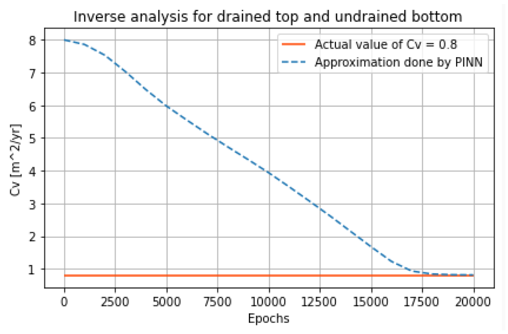


Figure 5.23: Initial value = 8

5.1.4 Case study 4: Inverse analysis: Drained top and bottom

Domain, boundary conditions and import of data

The domain and boundary conditions are the same here as in the forward case for the double drained. The importing of data is done with the same method as for the single drained case, just using the analytical solution for the double drained case.

Neural network

The neural network has the same specifics as the single drained case seen in table 5.5. For the double drained case, the consolidation factor is set to a value of $C_v = 0.2$ where the initial value for the trainable variable $C_{vt} = 0.5$. Other than that everything is the same as for the single drained case. The loss function for the double drained inverse analysis is:

$$\begin{aligned}
&L_{Tot} = W_{PDE}L_{PDE} + W_{BC}L_{BC} + W_{IC}L_{IC} + W_{Data}L_{Data} \\
&L_{PDE} = L_{PDE} \\
&L_{BC} = L_{BC} \\
&L_{IC} = L_{IC} \\
&L_{Data} = L_{Data}
\end{aligned} \tag{5.13}$$

Result

Results are as follows:

Training loss	Test loss	Training time	Final Variable	Error
$5.68e-3$	$5.68e-3$	383.95s	0.204	0.02

Table 5.7: Losses and time for double drained

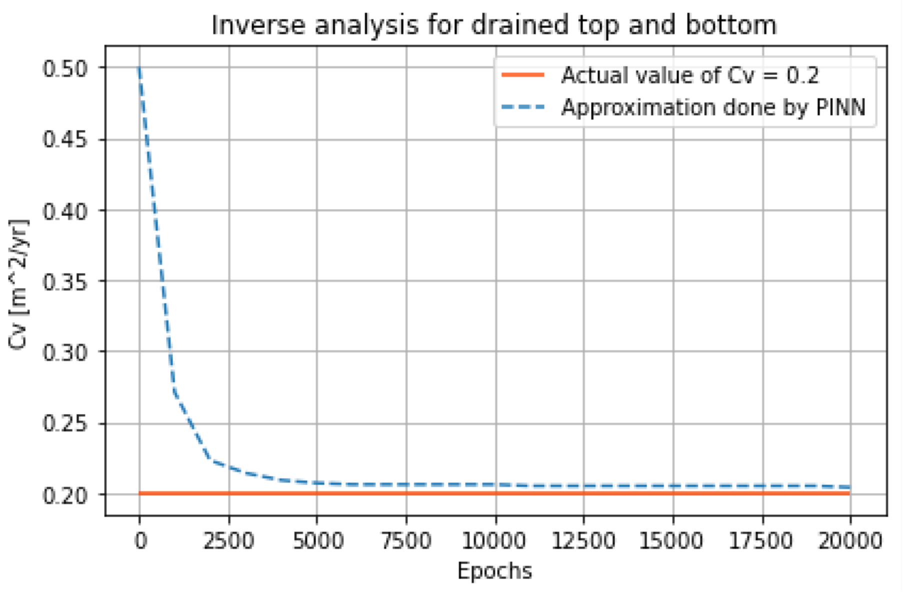


Figure 5.24: Inverse analysis for the double drained case

Discussion
Initially, the results does not seem as accurate as they where with the single drained case where the the error was less than $1e-6$ compared to $2e-2$ but it is important to note that this may also be because of a really good training cycle by the single drained model and a not so good training cycle by the double drained model. In order to see if the model was indeed as robust as the single drained case - a series of test were performed. The summary of those tests are found in table 5.8.

Initial value	Target value	Results	Error
4.0	0.2	0.205	0.025
-0.2	0.2	0.203	0.015

Table 5.8: Extremes of inverse double drained

As shown, it was not as robust as the single drained model. This may be due to the 100x100 matrix being for a full layer domain which gives it more complex structure than the single drained which leads to a weaker prediction. Optimizing hyper-parameters was attempted but did not yield a significant improvement. A further convergence study with more iterations was also attempted but the loss-function and the approximation by the model towards c_v flattened out after 20 000 iterations.

5.1.5 Case study 5: Forward analysis with noise: Both cases

In this section there is nothing new to the models except the introduction of noise to the dataset. Because of this the information about the models used will not be restated here and the reader is referred to the cases on the respective models.

In order to introduce the model to noise - Numpys normal distribution function ("Numpy normal", n.d.) is applied which add noise to the dataset by using Gaussian distributions. The probability density for Gaussian distribution is

$$p(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \tag{5.14}$$

where μ is the mean, σ is the standard deviation and σ^2 is the variance. It is assumed that the reader is familiar with statistics, but if not, a short summary

- Mean value, also called the expected value, is the average number that's found by adding all the sampling points together and dividing it by the the number of sampling points
- Standard deviation is a value for the amount of dispersion or deviation there is form the dataset. A high σ means the values are spread out over a larger range. A low σ means it's closer to the mean.
- Variance is measure of dispersion over the dataset, meaning how far the data points spread out from the mean.

Results
In the cases with added noise, the noise is added by increasing the standard deviation thus increasing the limits of the dataset. Because of this fact it was tried to see how well the model would predict with added noise to the dataset for z and t. The results are:

Single drained	Training loss	Test loss	Test metric	Training time
Without noise	$7.44e-3$	$8.01e-3$	$7.38e-1$	142.53 sec
With noise	$1.12e-2$	$1.12e-3$	$7.27e-1$	186.42 sec

Double drained	Training loss	Test loss	Test metric	Training time
Without noise	$5.97e-3$	$7.68e-3$	$4.93e-1$	136.62s
With noise	$6.43e-3$	$1.01e-2$	$4.92e-1$	129.97

Table 5.9: Losses and time for forward analysis with noise

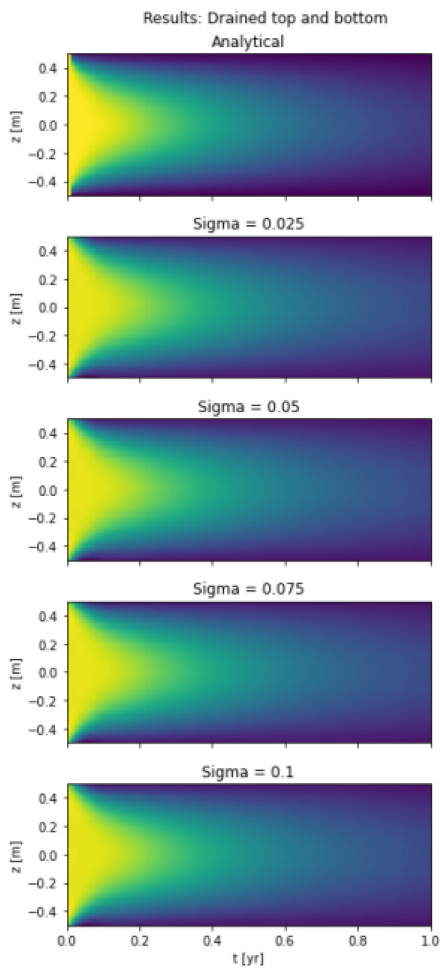


Figure 5.25: DD with noise

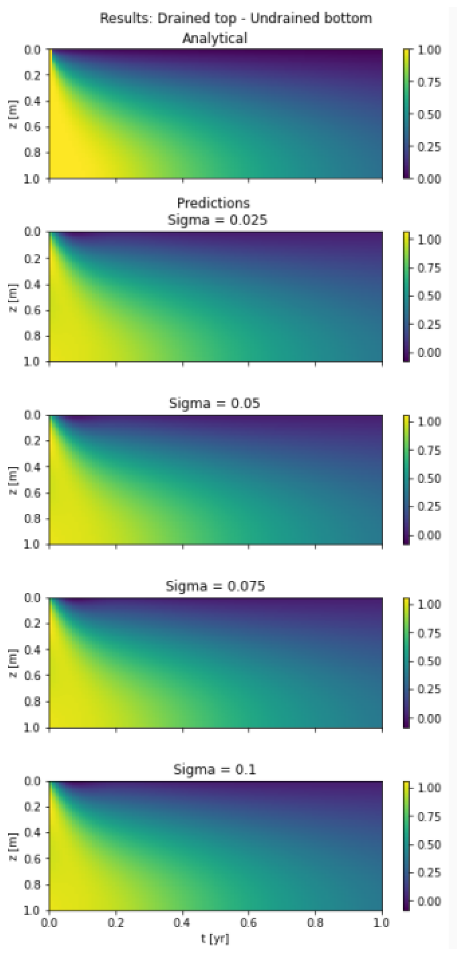


Figure 5.26: SD with noise

Discussion

As seen from the plots, the model is able to predict the correct solution in terms of form, but the dissipation is not as strong as for the analytical solution. That being said, the solutions does not show a significant decrease in performance due to the added noise, meaning that the model can predict relatively good results despite having corrupted data in its dataset. The losses and time, although added to the results does not have that much applicability here since the noise was added to the dataset used for prediction. Overall, the model shows robustness against corrupted data for its predictions.

5.1.6 Case study 6: Inverse analysis with noise: Both cases

For the inverse analysis it is doable to apply noise to the dataset used for training. Since the forward case had noise applied to the spatial - temporal variables - the inverse case had noise added to the imported values for excess pore pressure (p) with the following results:

Single drained	Training loss	Test loss	Training time	Final Variable	Error
Without noise	$4.92e-3$	$4.92e-3$	336.71s	0.50000	$e < 1e-6$
With noise	$1.84e-2$	$1.84e-2$	397.65s	0.5001	$2e-4$
Double drained	Training loss	Test loss	Training time	Final Variable	Error
Without noise	$5.68e-3$	$5.68e-3$	383.95s	0.204	0.02
With noise	$1.59e-2$	$1.59e-2$	416.72s	0.204	0.02

Table 5.10: Losses and time for inverse analysis with noise

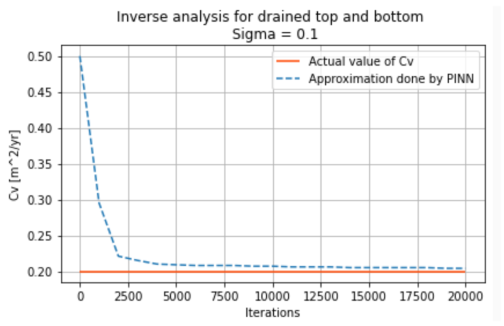


Figure 5.27: Inverse DD with noise

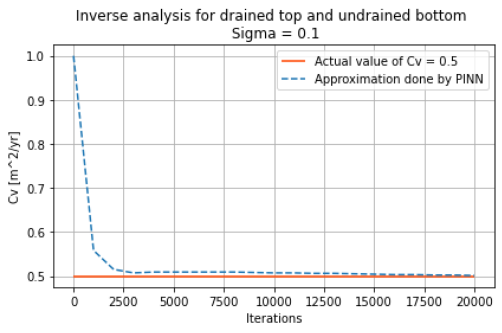


Figure 5.28: Inverse SD with noise

Discussion

Concerning the losses and training time for this case, both the single and double drained cases show the expected results of decreased losses and increased training time. This is probably due to the model actually having to evaluate the noise added to the excess pore pressure since it is fed to the model through the observed values which are evaluated against the output, see case study 3. This gives the noisy dataset higher influence which in turn gives the expected difference shown in table 5.10. But despite this it does predict the values for C_v very well with a difference of approximately $e-2$ and exactly the same for the single and double drained respectively.

5.2 Bearing capacity: Cartesian to polar coordinates

The bearing capacity problem as a problem was inspired by (Smith, 2005) and his paper on "Complete limiting stress solutions of bearing capacity of strip footings on Mohr-Coulomb soil". The following section explains shortly the failed attempt at predicting bearing capacity using Cartesian coordinates first by introducing the initial domain, its boundary conditions and defining the loads for the system before moving on to showing the transformation from Cartesian to polar coordinates. It should be noted that here as well, the model are run on a Nvidia Geforce 250MX GPU.

In order to investigate the problem, a simple model of the problem was analysed in Plaxis 2D. Using the full domain of the Plaxis model and examining the stresses created by the loading conditions - half of the model with its corresponding boundary conditions was extrapolated form that model. Examples are as follows

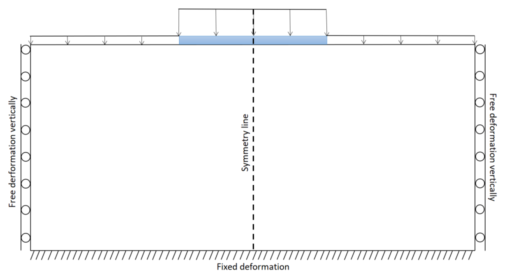


Figure 5.29: Full domain

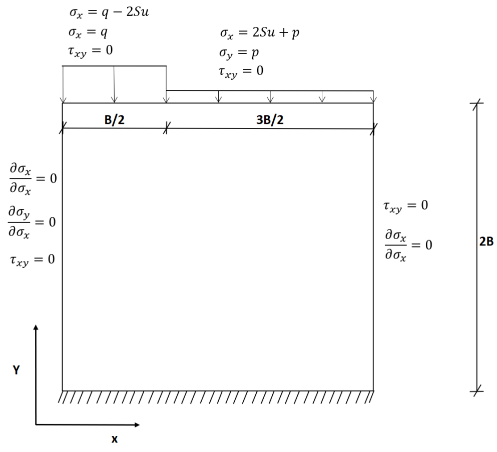

$\sigma_x = q - 2Su$
$\sigma_x = q$
$\tau_{xy} = 0$
$\sigma_x = 2Su + p$
$\sigma_y = p$
$\tau_{xy} = 0$
B/2
3B/2
$\frac{\partial\sigma_x}{\partial\sigma_x} = 0$
$\frac{\partial\sigma_y}{\partial\sigma_x} = 0$
$\tau_{xy} = 0$
$\tau_{xy} = 0$
$\frac{\partial\sigma_x}{\partial\sigma_x} = 0$
2B
Y
x

Figure 5.30: Half domain and boundary conditions

Seen from figure 5.30 is also the load conditions. At first it was attempted to reach yield with just a vertical load for both q and p. This was because only vertical loads produces punching shear in the soil and not the stress field for a yield zones as wanted. Because of this, the collapse loads (Nordal, 2020) was augmented with a lower bound solution for the undrained case with the factors of $\sigma_x = q - 2s_u$ and $\sigma_x = 2s_u + p$ for the active and passive Rankine zones respectively in order to produce the wanted yield. Note that the derivation for these collapse loads is a lower bound solution following only one zone in Mohr's cicle like figure 2.14.

$$s_u = \sqrt{\frac{1}{4}(\sigma_x - \sigma_y)^2 + \tau_{xy}^2} \tag{5.15}$$

Using the predicted shear strength, one could predict the bearing capacity for the foundation. Assuming that the variable p, the analytical solution (Eiksund et al., 2019) for bearing capacity for undrained vertically loaded soil is

$$q = p + N_c s_u, \quad N_c = \pi + 2 = 5.14 \tag{5.16}$$

N_c is a factor that varies with the roughness of the foundation. In the cases for bearing capacity the foundations are assumed to be smooth foundations $r = 0$. Skipping the description of implementation of the neural network, its specifics are the same as in table 5.1, and using the activation function ReLU, due to its mentioned inability to calculate second order derivatives, in order to enhance and show the problem when using Cartesian coordinates - the results are the following. Notice that the axis are unimportant but the colourbar shows the predicted value for s_u.

Training loss	Test loss	Training time
$4.5e-3$	$4.5e-3$	305.5s

Table 5.11: Losses and training time for unnormalized undrained bearing capacity

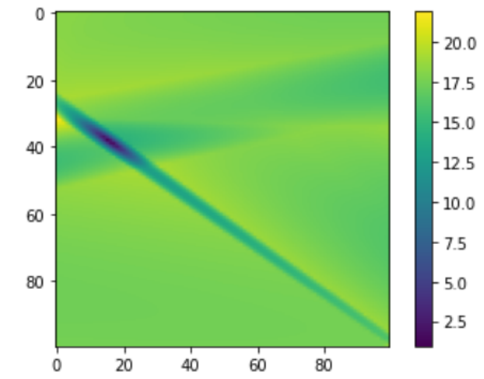


Figure 5.31: Prediction of s_u

A short explanation of the figure 5.31: Shown is the prediction of the undrained shear strength of the soil in this case. In the model - the load is applied on the left side of the plot. It was first attempted to apply the loads at the exact location that they where placed $q \in 0 \leq x \leq \frac{B}{2}$ and $p \in \frac{B}{2} \leq x \leq 2B$. This caused the singularity to happen each time at the same spot since the network didn't know how to handle the point at which there was a gap with no defined load, regardless of how small the intersection was made. Another attempt was made at leaving a larger area of 0.5 meters, like $q \in 0 \leq x \leq \frac{B}{2}$ and $p \in \frac{B}{2} + 0.5m \leq x \leq 2B$, where the load was undefined in order to see what the network would do with that area as in the case in figure 5.31. However, the singularity continued to form despite this change and it was suggested that in order to counteract this problem the domain would be transformed from cartesian to polar coordinates.

5.2.1 Case study 1: Forward undrained

Domain, PDEs and yield function

Changing from a cartesian to polar coordinate system changes the domain quite significantly. The biggest change is the exclusion of boundary conditions on both sides of the domain because it is confined to the failure zones. The loads however stay the same. This means that the upper boundary is the only one present in the geometry that's defined for the model. An outline of the change in domain is

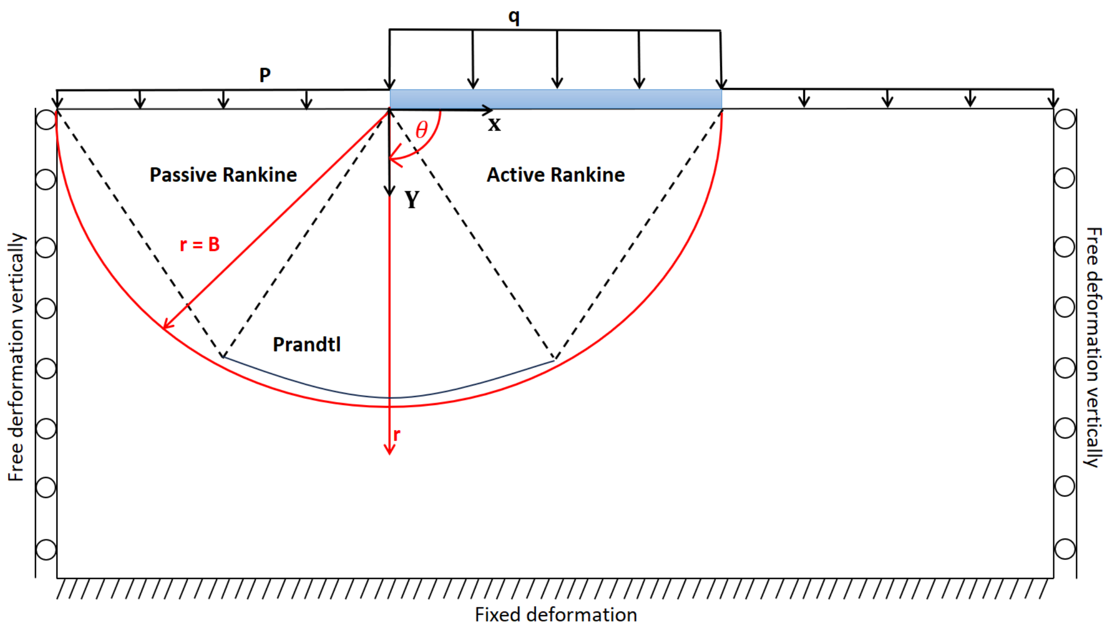


Figure 5.32: Full domain to polar domain

Due to this new domain, the PDEs and yield function must also be transformed to polar coordinates. This is done by the equations

$$\begin{aligned}\frac{\partial}{\partial x} &= \cos\theta\frac{\partial}{\partial r} - \frac{1}{r}\cos\theta\frac{\partial}{\partial \theta} \\ \frac{\partial}{\partial y} &= \sin\theta\frac{\partial}{\partial r} + \frac{1}{r}\cos\theta\frac{\partial}{\partial \theta}\end{aligned} \tag{5.17}$$

This leads to a new set of PDEs and yield function. These also has to be normalized in order to train the network efficiently. This derivation is quite long and the derivation is shown in appendix B. Note that the original PDEs and yield function are presented in equations 2.31 and 2.32. The new normalized PDEs and yield function with normalized stresses, radius and θ shown by (') which in this case means normalized and not effective stress, are:

$$r^{'}\pi\cos\theta^{'}\frac{\partial\sigma_{x'}}{\partial r^{'}} - \sin\theta^{'}\frac{\partial\sigma_{x'}}{\partial\theta^{'}} + r^{'}\pi\sin\theta^{'}\frac{\partial\tau^{'}_{xy}}{\partial r^{'}} + \cos\theta^{'}\frac{\partial\tau^{'}_{xy}}{\partial\theta^{'}} = 0$$

$$r^{'}\pi\cos\theta^{'}\frac{\partial\tau^{'}_{xy}}{\partial r^{'}} - \sin\theta^{'}\frac{\partial\tau^{'}_{xy}}{\partial\theta^{'}} + r^{'}\pi\sin\theta^{'}\frac{\partial\sigma_{y'}}{\partial r^{'}} + \cos\theta^{'}\frac{\partial\sigma_{y'}}{\partial\theta^{'}} - \frac{r^{'}\pi\gamma B}{q+\gamma B} \tag{5.18}$$

$$\frac{1}{4}(\sigma^{'}_{x} - \sigma^{'}_{y})^2 + \tau^{'2}_{xy} - (\frac{s_u}{q+\gamma B})^2 = 0$$

Neural network
Implementing the transformed domain into the neural network presented some problem. First of, the half-circle represented in figure 5.32 is not the type of geometry that DeepXDE has a function for. Since this problem had only two loads to define and no boundary conditions, the neural network had to interpret it a different way as a rectangle with boundaries $0.01 \leq r \leq B$ and $0 \leq \theta \leq \pi$ where the loads could be applied at either end of the domain like

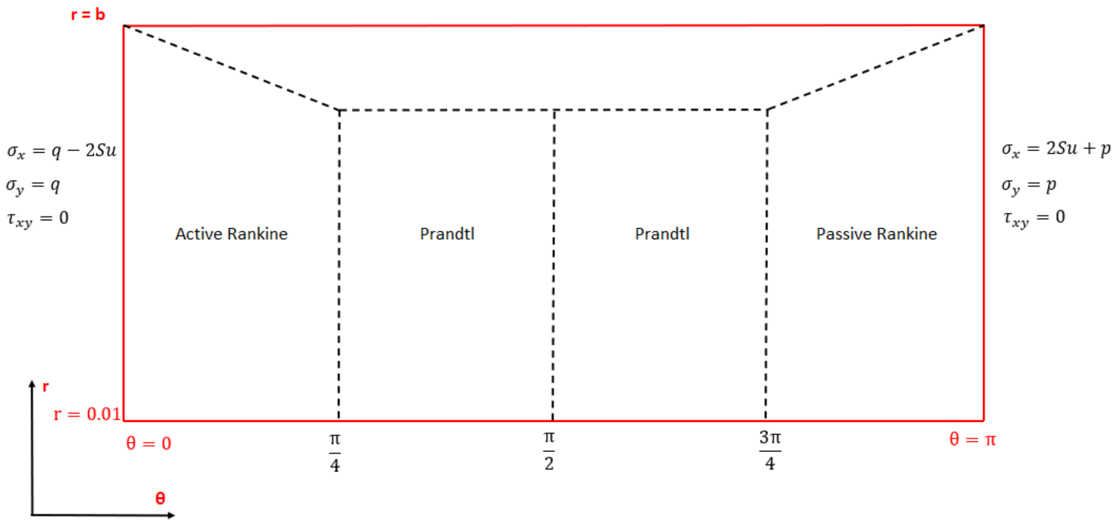


Figure 5.33: Polar domain in model

The left side now represents the boundary on top of the foundation while the right side is besides the foundation. The top unnamed area is an area that is undefined for the domain. As a consequence of the load being applied on just two sides of the rectangle - the loss function now only samples those two boundaries with the rest of the points being labeled domain. This yields the following loss function:

$$\begin{aligned} L_{Tot} &= w_{PDE}L_{PDE} + w_{BC}L_{BC} \\ L_{PDE} &= L_{PDE_1} + L_{PDE_2} + L_{Yield} \\ L_{BC} &= L_{Found_{\sigma_x}} + L_{Found_{\sigma_y}} + L_{Found_{\tau_{xy}}} + L_{Side_{\sigma_x}} + L_{Side_{\sigma_y}} + L_{Side_{\tau_{xy}}} \end{aligned} \tag{5.19}$$

This reduces the number of constraint that the model has but also decreases the time to train. Example of training points sampling for bearing capacity is

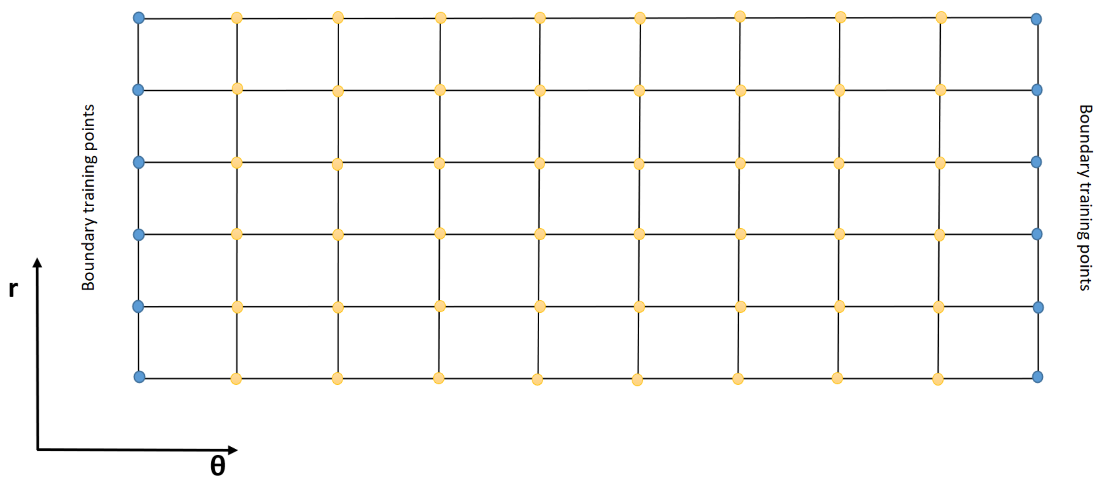


Figure 5.34: Training points sampling polar

Another constraint to the model will be the undrained yield function shown in 5.18. It acts as an additional constraint on the model in addition to the two PDEs.

```
return [diff_1, diff_2, constr]
```

In this part of the thesis, the model will be trained with two optimization algorithms. First the Adam optimizer for 20 000 iterations and then the model will be compiled again and trained again with the L-BFGS optimizer. The L-BFGS optimizer does not need a fixed learning rate since it adjusts the learning rate for each iteration to the approximation given by the Hessian matrix. This makes it ideal for late stage optimization. The remaining variables for the network is summed up as

Hidden Layers	Nodes	Activation	Initializer	Optimizer	LR	Iterations
3	50	Tanh	Glorot uniform	Adam	0.001	35 000
-||-	-||-	-||-	-||-	L-BFGS	-||-	Runs until converged

Table 5.12: Neural network - Forward undrained Bearing capacity

Result

Presented below is the table with results from the training of the model followed by the output of the model - first normalized, then shown unnormalized in the failure zones. In addition, the value for undrained shear strength s_u was defined as an external trainable variable in the network, and it is this value that the network uses to predict the bearing capacity with. The results are

Optimizer	Training loss	Test loss	Training time
Adam	$6.12e-5$	$5.75e-5$	204.69s
L-BFGS	$8.84e-7$	$8.56e-7$	120.77s

Table 5.13: Losses and time for forward undrained bearing capacity

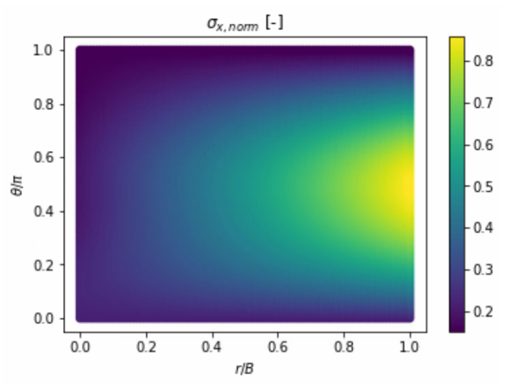


Figure 5.35: Normalized $\sigma_x^{'}$

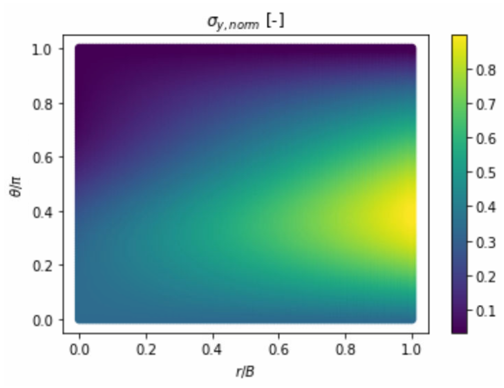


Figure 5.36: Normalized $\sigma_y^{'}$

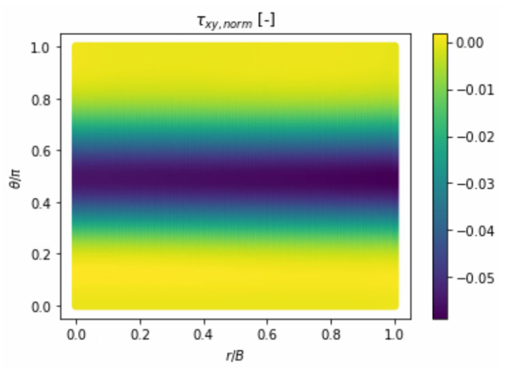


Figure 5.37: Normalized $\tau_{xy}^{'}$

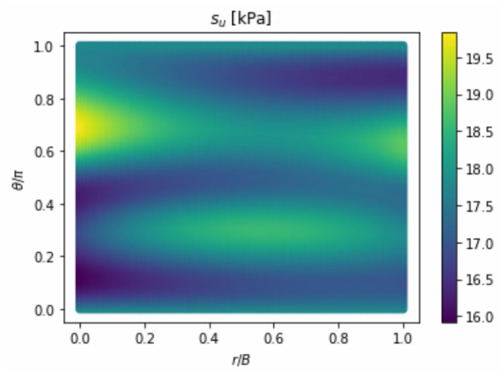


Figure 5.38: Predicted s_u

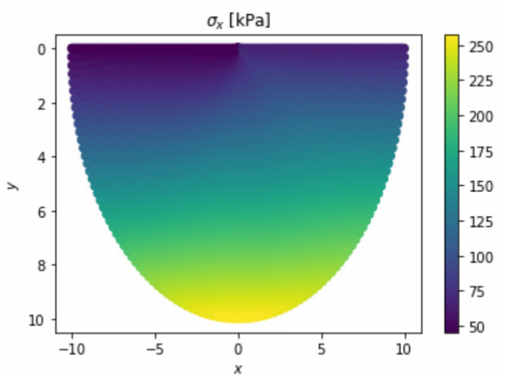


Figure 5.39: σ_x

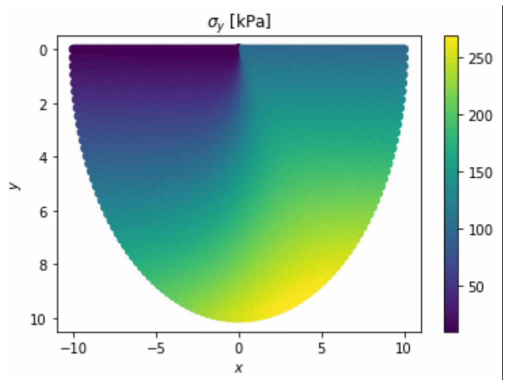


Figure 5.40: σ_y

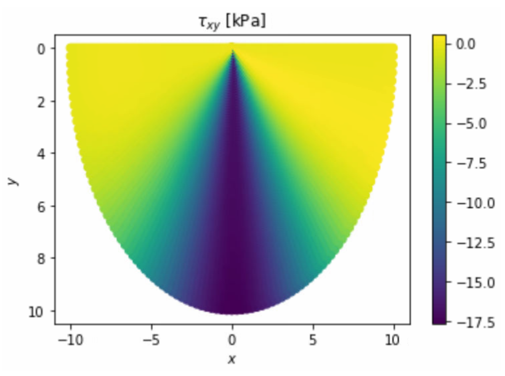


Figure 5.41: τ_{xy}

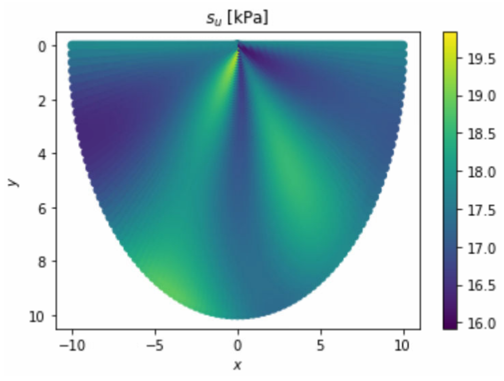


Figure 5.42: Predicted s_u

As mentioned, the results will be compared with Plaxis, but first it is compared to the analytical solution $q - p = N_c s_u$. r $= 0$ gives $N_c = 5.14$ which yields:

$$q - p = N_c * s_u - > s_u = \frac{100kPa - 10kPa}{5.14} = 17.51kPa \tag{5.20}$$

The variables applied for the Plaxis model is summed up in table below along with the maximum values for the stresses as given by the Plaxis model along with the mean predicted value for s_u by the model.

$\gamma_{unsat} = \gamma_{sat}$	$E_{u.ref}$	ν_u	$s_{u,ref}$
$20\frac{kN}{m^3}$	$20000kPa$	0.495	$17.5kPa$
σ_x	σ_y	τ_{xy}	$s_{u,ana}$
$250.54kPa$	$254.62kPa$	$17.346kPa$	$17.51kPa$
$\hat{\sigma_x}$	$\hat{\sigma_y}$	$\hat{\tau_{xy}}$	$\hat{su_{mean}}$
$249.13kPa$	$269.90kPa$	16.847	$17.580kPa$
$\lvert\frac{\hat{\sigma_x}-\sigma_x}{\sigma_x}\rvert$	$\lvert\frac{\hat{\sigma_y}-\sigma_y}{\sigma_y}\rvert$	$\lvert\frac{\hat{\tau_{xy}}-\tau_{xy}}{\tau_{xy}}\rvert$	$\lvert\frac{\hat{Su}-S_{u,ana}}{s_{u,ana}}\rvert$
$5.63e-3$	$6.0e-2$	$2.89e-2$	$3.99e-3$

Table 5.14: Forward: Plaxis variables, model prediction and absolute error

Development/Discussion

Before implementing the PDE's in the way that they are, it was attempted to implement the bearing capacity problem with a different approach to the constrained PDE's which involved transforming the yield function into a function for $\tau_{xy} = \frac{1}{2}\sqrt{\sin{(\phi)}^2(\sigma_x + \sigma_y)^2 - (\sigma_x - \sigma_y)^2}$ and taking the partial derivatives of this function in order to make the model predict only σ_x and σ_y. τ_{xy} could thereafter be calculated analytically. This was thought to be a more computationally efficient way of training the model, but when implemented, it blew up at the first iteration and did not learn anything due to exploding gradients.
A lot of the problems with the implementation of the model that predicts all three stresses and their solutions concerning domain, boundary conditions and normalization of the PDEs and yield function have already been addressed previously in the case study. Once those solutions where in place, the problem lay in optimizing the model in order to prevent the formation of singularities and spurious extreme predictions by the network. One such problem was the formation of a large singularity when predicting the undrained shear strength. A solution that was tested was changing the parameters and hyper-parameters.

In the end, the solution to this problem lay in the radius for the domain. By reducing it from 0.01 meters to 0.001 meters, the singularity did not form. Singularities usually form when there are values very close to or at zero. Using the yield function in equation 5.18 as an example would be when the model test values that yield $\sigma_x - \sigma_y = 0$. Here it seems that the reduced radius reduced the influence of r on the PDE's enough which made it converge towards the correct solutions and eliminate the singularity.

5.2.2 Case study 2: Inverse undrained

Importing external dataset
Since the domain and boundary conditions are the same here as for the forward case, the difference lies in the importation of data and its implementation into the neural network. The importation of the external dataset was done by extrapolating the data from the same Plaxis model that defined the forward problem. The values for r, θ, x and y where created synthetically and has the same values as in the forward case. The values for σ_x, σ_y and τ_{xy} where extrapolated from Plaxis within the failure zone as shown in figure 5.32. The code for extracting data from Plaxis is uploaded to Github as shown in appendix 7.

Using 400 of the imported values from the variables, this was fed into the network in the same way as done in the inverse analysis for consolidation using PointSetBC and anchors. The imported values where normalized in the same way that the stresses in the normalized PDEs were, like:

$$r^{'} = \frac{r}{B} \qquad \theta^{'} = \frac{\theta}{\pi} \tag{5.21a}$$

$$\sigma_x^{'} = \frac{\sigma_x}{q+\gamma B} \qquad \sigma_y^{'} = \frac{\sigma_y}{q+\gamma B} \qquad \tau_{xy}^{'} = \frac{\tau_{xy}}{q+\gamma B} \tag{5.21b}$$

Neural network
The neural network type for the case of inverse undrained is not the same as in forward undrained. It has been changed from a FNN to a PFNN. A PFNN is a parallel fully-connected neural network. In the normal FNN, there are one network which predicts all the outputs. In a PFNN, there are independent sub-networks that predicts each variable individually. Following the example for a FNN shown in figure 2.5, the same architecture with a PINN in this case is:

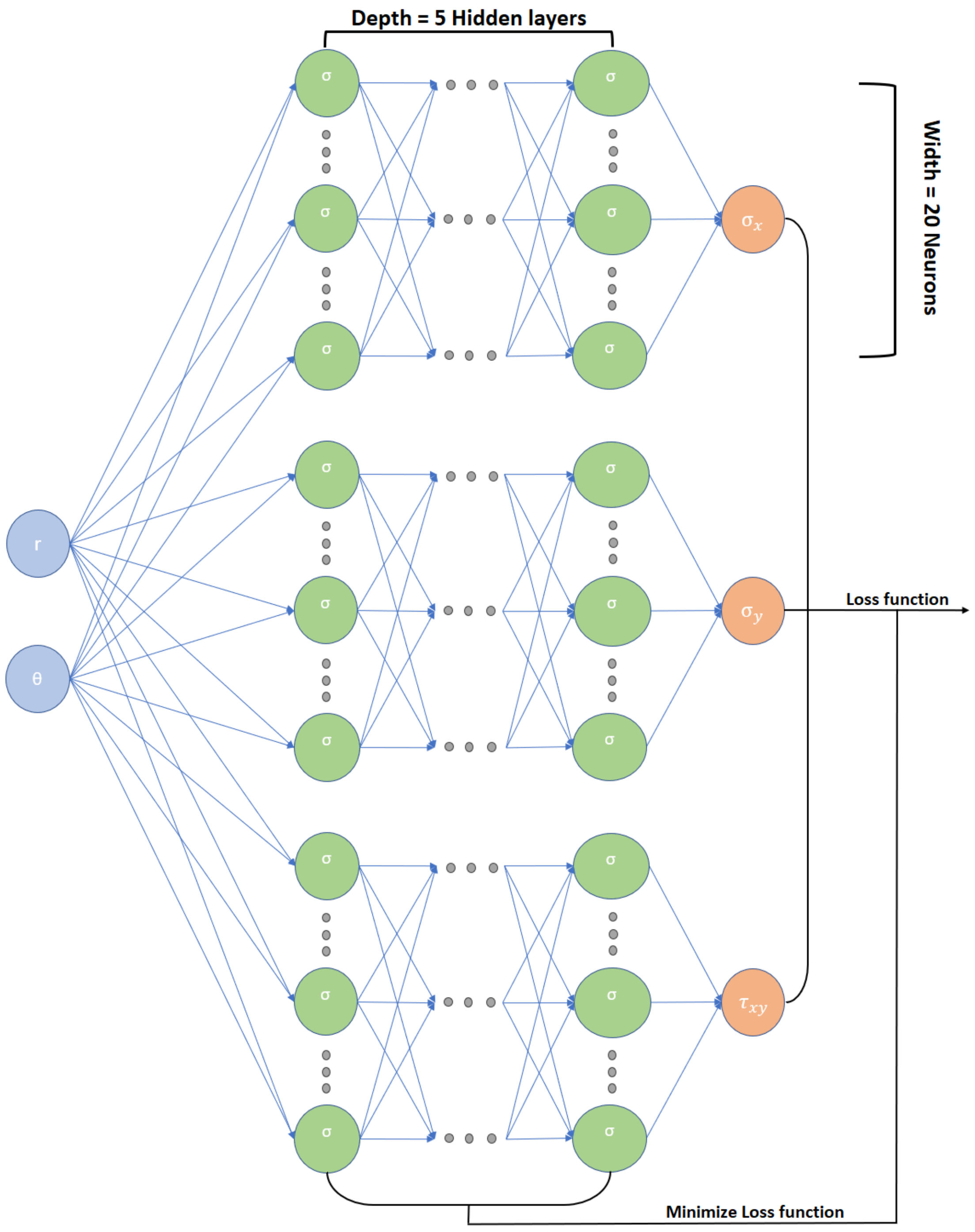


Figure 5.43: PFNN architecture

In addition to the new network architecture, there was also added a point resampler, which resamples the points within the PDE at a specified interval. In this case, the interval was chosen at every 100 iteration. The number of points sampled follows the parameter num_domain and num_boundary. Also changed from the forward case is the parameters for the weights of the imported shear forces and the global tolerance for the L-BFGS optimizer. The changed values are listed in the following table. Besides these, the parameters and hyper-parameters stay the same.

Loss_weights	gtol	num_domain	num_boundary	num_test
$[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 14]$	1e-10	500	200	1000

Table 5.15: Neural network - Forward undrained bearing capacity

A short explanation for the list for the loss_weights is that the first three from the left represents the two governing PDEs and the yield function. Thereby followed by the six boundary conditions, one for each stress component on the foundation and beside it. Lastly are the three observed values defined by PointSetBC for the three stress components. DeepXDE allows for individual weight to be changed in the loss function. This yields the following loss function for inverse bearing capacity analysis:

$$\begin{aligned}
&L_{Tot} = W_{PDE}L_{PDE} + W_{BC}L_{BC} + W_{Data}L_{Data}\\
&L_{PDE} = L_{PDE_1} + L_{PDE_2} + L_{Yield}\\
&L_{BC} = L_{Found,\sigma_x} + L_{Found,\sigma_y} + L_{Found,\tau_{xy}} + L_{Side,\sigma_x} + L_{Side,\sigma_y} + L_{Side,\tau_{xy}}\\
&L_{Data} = L_{Data,\sigma_x} + L_{Data,\sigma_y} + w_{Data,\tau_{xy}}L_{Data,\tau_{xy}}
\end{aligned} \tag{5.22}$$

Results

The results are presented in the same plots as for the forward analysis with the sampled imported training points in the same plot.

Optimizer	Training loss	Test loss	Training time
Adam	$4.25e-4$	$4.19e-5$	1649.11s
L-BFGS	$8.84e-7$	$8.56e-7$	309.38s

Table 5.16: Losses and time for inverse undrained bearing capacity

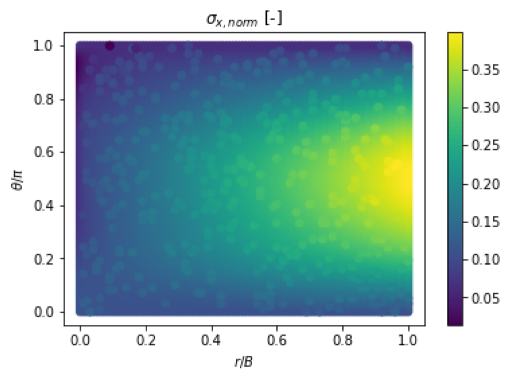


Figure 5.44: Normalized σ_x

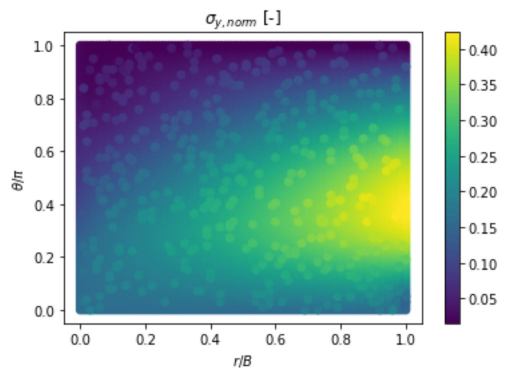


Figure 5.45: Normalized σ_y

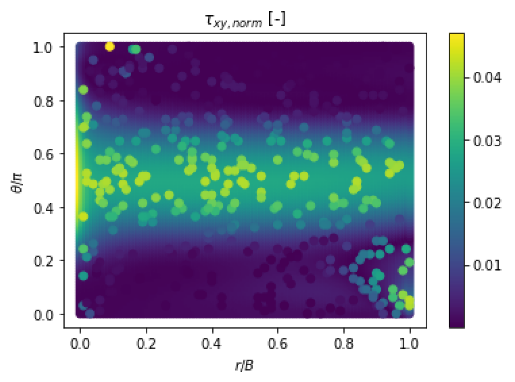


Figure 5.46: Normalized τ_{xy}

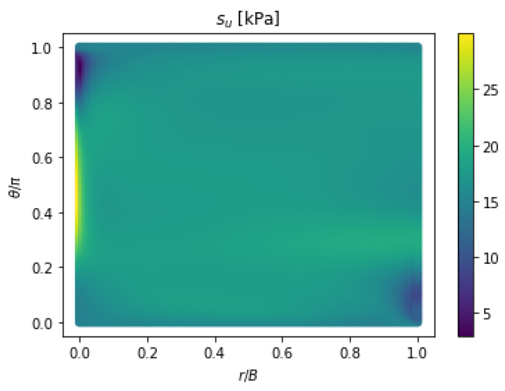


Figure 5.47: Predicted Su

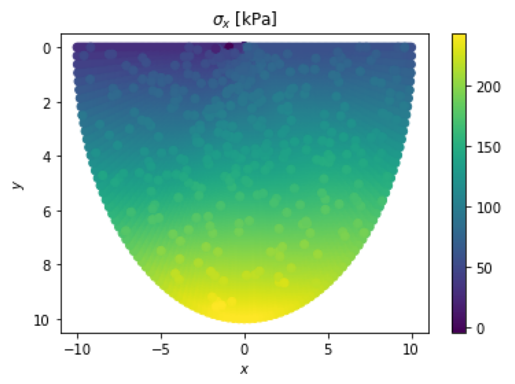


Figure 5.48: σ_x

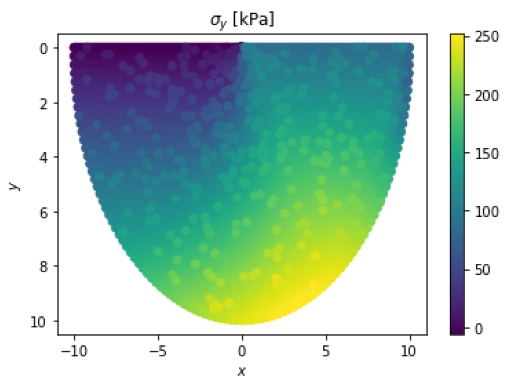


Figure 5.49: σ_y

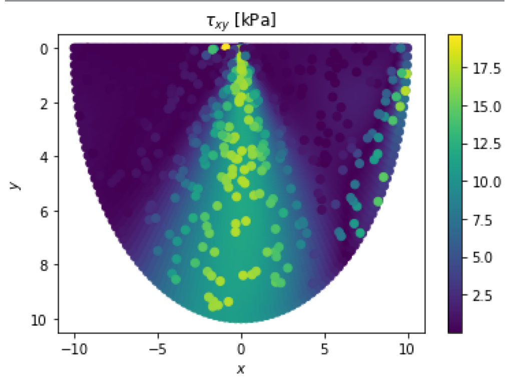


Figure 5.50: τ_{xy}

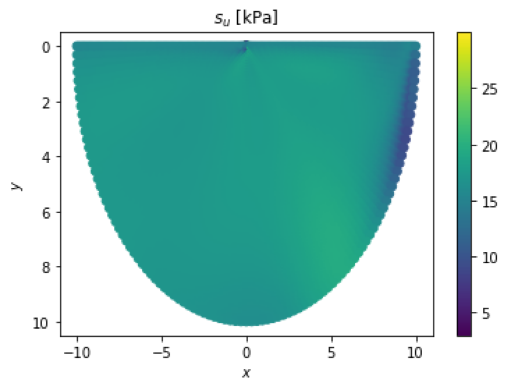


Figure 5.51: Predicted Su

σ_x	σ_y	τ_{xy}	$S_{u,ana}$
$250.54kPa$	$254.62kPa$	$22.96kPa$	$17.51kPa$
$\hat{\sigma_x}$	$\hat{\sigma_y}$	$\hat{\tau_{xy}}$	$\hat{Su_{mean}}$
$250.373kPa$	$265.86kPa$	$29.513kPa$	$17.446kPa$
$\lvert\frac{\hat{\sigma_x}-\sigma_x}{\sigma_x}\rvert$	$\lvert\frac{\hat{\sigma_y}-\sigma_y}{\sigma_y}\rvert$	$\lvert\frac{\hat{\tau_{xy}}-\tau_{xy}}{\tau_{xy}}\rvert$	$\lvert\frac{\hat{Su}-Su}{Su}\rvert$
$6.66e-4$	$4.41e-2$	$2.85e-1$	$3.655e-3$

Table 5.17: Inverse:Plaxis variables, model predictions and absolute error

Development/Discussion
This case had some problems with singularities in the predictions of s_u. Initially the network was defined in the same way as for the forward problem, but it was later discovered that when importing the stress components that since the failure zone consisted of the whole foundation due to the value for B, it also imported shear forces of equal on opposite sign on the other side of the symmetry line shown in figure 5.29. This lead to the network initially predicting shear forces for both positive and negative values, which lead to an increase in predicted s_u at certain points. It was attempted to just remove the opposite values but this also gave bad predictions. By changing the sign of the opposite shear force it gave a much more clean predictions across the domain. The increased weights for the imported shear functions was chosen as the optimal value after trail and error due to it being overestimated in comparison with the other forces.

But the model was still not give a consistent value for s_u throughout the domain ever time the model was trained. This problem could be fixed by increasing the number of iterations significantly, but this decreased the performance since it also increased the training time. By introducing the PFNN and the PDE-resampler, this allowed the model to predict a clean result in a much shorter time with more consistently.

The results for the σ_x and s_u had a very good accuracy, which is also represented by the low loss function for training and test in table 5.16, but the model overestimates both σ_y and τ_{xy}. Looking at the plots for these two variables, σ_y has a uniform plot across the domain for the prediction by the model and its imported values which shows that it does indeed overestimate this value. But in the plot for τ_{xy}, the imported values are for the majority, quite close to the predictions made by the model except for one especially spurious value shown by the color yellow in the plot. This single value is probably what gives the large value that's shown in 5.17. Excluding this spurious value, the predictions of τ_{xy} by the model is very good compared to the imported values. The time to train has been increased a significant amount 500% compared to the forward model. This was due to the inverse undrained model giving uneven predictions after training. But with the addition of two extra sub-ANNs and a resampler which resamples values in the domain ten times each iteration, the time to compute is still relatively low.

6 Discussion

This section is divided into two parts. Consolidation and bearing capacity. In both parts, the implementation of the PINNs will be discussed along with its problems and results. Additionally, the further improvements and cases that could expand on these PINNs will also be presented discussed.

6.1 Consolidation

A lot of the initial work on the implementation of 1D consolidation built upon the paper by (Bekele, 2020). Because of this, it is natural to compare the results from the consolidation cases with this papers results as far as that is possible. (Bekele, 2020) will be referred to simply as "Bekele" from now on. The models in this thesis will be refereed to by their case abbreviations. E.g single drained = SD, double drained = DD, etc. It should be noted that Bekele measures the performace of his forward models by MSE and L_2 relative error. Mean L_2 relative error was attemted to implement into DeepXDE as a loss function but gave infinity in the loss function immediately.

6.1.1 Forward cases

The first implementation of the the single drained consolidation problem gave good results when compared to the analytical solution. This was also reflected by the low loss of the loss function. Comparing the results for the MSE loss at the end of training in DeepXDE with Bekeles', the results were $7.44e-3$ vs $1e-5$. Bekele's produced a better accuracy than the DeepXDE model. When training the model in DeepXDE, it was attempted to use a larger number of iterations in order to reach better accuracy, but the loss-function curve flattened out. This despite increasing the sampling number of collociation points in the domain. The same result is apparent for the DD case and is shown in table 5.4. Thus, with the implementation that is done into DeepXDE, the accuracy of the model did not outperform Bekele's. Where it did outperform Bekele's model was in the time to train. The difference between the two was approximately 12 minutes versus 142.52 seconds for Bekele and DeepXDE respectively. This is a remarkable improvement in efficiency. By reducing the time to train by almost nine and a half minute, this shows that the DeepXDE library is very computationally efficient for running PINNs. It should be noted that the hardware the models have been trained on was a Nvidia Tesla K80 for Bekele versus Nvidia GeForce MX250. A comparison between the two GPU's is given in ("Comparison - Tesla K80 and GeForce MX250", n.d.). Bekele's GPU is much stronger than the GPU used in this thesis which further proves the point that the code used in this thesis in accordance with the DeepXDE library is very computationally efficient. This is an important factor for a geotechnical engineer as efficiency is key factor in the design process.

For the double drained case, Bekele does not state neither the training time or the MSE error. Assumptions could be made based on the time to train taking almost as long with the same accuracy but a comparison between the values in between the would be pure speculation and will therefore be left out. The implementation of this problem into DeepXDE was very effortless after the single drained was implemented. The time to train ,136.62 seconds, was shorter than for the single drained case and the accuracy about the same $5.97e-3$.

In both forward problems, predicting the initial conditions accurately was a problem. One solution to this problem could be to create a function which is the solution to the problem in terms of boundary conditions and enforce that as a hard constraint on the ICs. DeepXDE enforces the ICs as soft constraints by default as is done in the thesis. The difference between hard and soft ICs is that the hard ICs must be satisfied exactly. If such a function where created, it would be assumed that it would fix the issue with the ICs. Further improvements to the forward SD and DD cases could include the implementation of the L-BFGS optimizer at the end of the training cycle. Implementing this is expected to increase the accuracy of the model and reduce MSE of the loss-function and bring it closed to the accuracy shown by Bekele. For the DD case, since the model showed a lower degree of dissapation, the PDEPointResampler, which was applied in the bearing capacity cases could also be a relevant improvement here.

6.1.2 Inverse cases

The implementation of the cases were relatively straight forward compared to the forward cases in terms of geometry and neural network. Comparing the SD case first - Bekele's took about eight minutes to train the model. The DeepXDE model took 336.71 seconds. What is interesting here is that Bekele's code had a reduction in time to train by $\frac{1}{3}\%$ whereas the code in DeepXDE had an increase in time to train by 35.7%. The expectation would be that both models took longer to train due to the extra trainable variables into the the constraints. One reason why Bekele's model may have gotten the time to train reduced is that for the forward cases, 10 000 collocation points where applied as training points versus 2000 randomly selected points from the analytical solution in the inverse case. For the DeepXDE model, it trains the inverse model more or less the same way as for the forward model, but with an additional 2000 randomly selected points from the analytical solution. This is what allows it to predict s_u in the forward bearing capacity cases. But these are assumptions based on Bekele's paper and not on his code. Concerning the accuracy of the models, Bekele SD model achieved an accuracy of of $3e-3$ for the estimation of c_v. The SD model achieved an accuracy of $e > 1e-6$ which is a remarkably good result showing that the DeepXDE model outperformed Bekele's in terms of both time and accuracy. On the DD however, Bekele achieved an error of $6e-4$ compared to $2e-2$ for the DeepXDE model. Not accounting for time, this proves that Bekele's models for the inverse cases are more consistent in term of accuracy than the DeepXDE models. In identifying material parameters, accuracy would be very important which makes this a drawback for this model. The result is sill within 2% which is an acceptable result.

Improvement could be made when it comes to the inverse cases for the DeepXDE models. The hyper-parameters for the sampling in the domain, boundary- and initial conditions were kept the same as they were in the forward cases and not optimized for the inverse case. By optimizing these, the model could reach higher accuracy and/or reduce time to train with little effort. A convergence study was done for the DD case but the results did not improve with more iterations. One improvement that could help here as well is the the L-BFGS optimizer for the same reason as stated in the forward cases. Since both of the cases have almost reached their converged values before 10 000 iterations, the number of iterations with Adam optimizer could safely be cut in half with the L-BFGS optimizer while still reaching better convergence. This is represented in the results for the SD and DD case studies. L-BFGS would be especially beneficial for the double drained case where the accuracy was low. As before, other implementations like PDEPointResampler could also be beneficial but this increases time to train and does not come without a cost. Another implementation that would be interesting to implement into these inverse problems would be to use a PyTorch backend when importing the values. DeepXDE as a library, with this backend allows for the imported values to be sampled and fed to the model in batches at random. Compared to the way it's implemented in the DeepXDE model here, all 2000 points are fed to the model and only sampled once. 2000 points is a relatively large amount but it does not represent the whole of the data as well as it would if sampled randomly in batches of 200 for 20 000 iterations. This would probably lead to a higher accuracy and is expcted to not increase the time to train by a significant amount since the data is loaded for the whole of the imported domain once and then sampled.

6.1.3 Forward and inverse with added noise

Lastly in consolidation came the introduction of noise to the datasets. There are no comparison with Bekele's results here since he did not add noise to his datasets. However, while the results for these cases are discussed in the case studies, it would again be interesting to see how well the models would perform with the improvements suggested in the previous discussions. By doing this, the models could possibly handle even more noise and corrupted data. In the cases for this thesis, Gaussian normal distribution was applied. A further examination of the effect of noise on PINNs would be to remove values form the dataset and check performance. A limit analysis of how much noise the PINN could handle and still perform within acceptable parameters would also be a further prospect.

6.1.4 Further suggestion

In terms of benefits for a geotechnical engineer, the 1D consolidation models does not have all that much benefits in terms of forward analysis, but the inverse analysis can be very beneficial. A suggestion for the further work with consolidation PINNs would be to create a plug-in for Plaxis where the inverse models could be used to approximate a material coefficient given the data in Plaxis or external data from real life projects. Also, all these models are in 1-D. It is not suggested to upscale the consolidation PINNs to 2-D as that has already been done (Y. Lu and Mei, 2022). Instead it is suggested to upscale the problem of consolidation to 3D next. But before doing that, the addition of more than one layer into the models would also a point for focus.

6.2 Bearing capacity

There is no comparative paper on this part that the author of this thesis knows of. The problem was first implemented with Cartesian coordinates and then converted to polar coordinates. After which it was implemented for the forward and inverse undrained case.

6.2.1 Forward undrained

The implementation for the forward undrained bearing capacity problem has been thoroughly discussed in the case study along with its implementation. The results showed great accuracy and short time to train. The absolute error where all below $2.89e-2$. However, the forward problem was implemented with a FNN despite it having to predict three variables. This would mean that the PFNN would probably help this model reach even better accuracy and thus produce an even cleaner plot for s_u especially. Further investigations into this problem would be the introduction of inclined load on the foundation or the addition of more than one layer.

One discovery that was made when working on the forward undrained problem was that the boundary condition seem to be under-influencing as a constraint when contributing to the loss function. The governing equation always seem to have the largest contribution on the loss function. The way this was discovered was through trying to implement the load q as an external trainable variable. When training the model, the variable would only differ with $\pm 1e-2$ from it's original value. Having q as a trainable variable would be a very good way to predict bearing capacity and would have great benefits in terms of usage of PINNs in geotechnical engineering. The suggestion for further work to implement q as a trainable variable would be to implement q as a variable in accordance with the stresses σ_x and σ_y into the PDE constraint as that has the highest influence on the model. Note that it was tested to increase the weights on the boundary conditions that contained q in an effort to make the model focus on this variable, but this did not yield any significant postivie results.

6.2.2 Inverse undrained

The inverse undrained is the most elaborate of the models in this thesis and the last to be created, thus it is also the most optimized. Inverse undrained has been explained in great detail in the case study, both in terms of its hyper-parameters and parameters. It produced predictions with a very good accuracy compared to the Plaxis model for the same case except for τ_{xy}. Why this was, was discussed in the case study. The imported values can produce spurious extreme values, as was shown for τ_{xy}. Because of this, the inverse model would benefit greatly from an adaptive scaling method to be implemented in terms of its parameters. This method would scale the weights of the model while training. For the forward undrained case - many of the same suggestions for further work will also work here, like layers, inclined load and more. As for consolidation, inverse analysis is a PINNs strength in terms of usefulness. It is also suggested here that a plug-in could be created for use in commercial programming software.

Bearing capacity as a problem is suggested to be expanded into drained bearing capacity. The introduction of drained analysis would introduce three more material parameters to predict $\phi, a, c^{'}$. If drained PINNs was produces, further investigation into layers with differing waterline would is suggested. It should be noted that the drained problem was attempted to be solved in this thesis, but the model for the forward case produced a singularity for the stresses which made the model unusable. The singularity was not large and the model predicted correctly for the stresses σ_x and τ_{xy} when compared to Plaxis. But it overpredicted σ_y by almost 15%. An external trainable variable was defined for ϕ, which is the the angle of friction in drained analysis, and the model converged on this soution, but the singularity made it so that the accuracy was $\geq$ 6% due to the singularites. Many things i order to solve this including transforming the trigonometric functions of the PDEs form sin and cos to tan, a small value $1e-8$ was added to all the stresses for them to not be zero, and different collapse loads were derived and implemented. But in the end, the singularity remained. What was an interesting discovery from this analysis and the undrained case was that the collapse loads that correctly predicted the forces, in the undrained case, and had the best approximation, in the drained case were the lowest bound solutions i could find. In the case of undrained: $\sigma_x = q - 2s_u$ and $\sigma_x = 2s_u + p$ for the active and passive Rankine zone respectively, and in the drained case: q/N for active and N*P for passive Rankine zones. The factor of N is $\tan^2(\frac{\pi}{4} + \frac{\phi}{2})$ and is derived from the Mohr's circle for the drained case. A way for further investigation would be to derive the PDE for equilibrium in 2D by the method of characteristic into ordinary partial differential equations and implement into PINNs.

Further suggestions for the bearing capacity cases is to expand the problem to 3D or introduce layers into the 2D undrained cases. Also, introduction of other material models like elasto-plasticity would be of interest. Especially for the analysis of bearing capacity on sand.

7 Conclusion

The purpose of this thesis was to investigate the following question:

Investigate the application of PINNs on problems of consolidation and bearing capacity in geotechnical engineering

which would be completed by fulfilling a set of sub-objectives:

- Perform a literature study on PINNs and the DeepXDE library
- Implement one-dimensional consolidation equation into a PINN and check performance for forward and inverse analyses
- Introduce noise to the datasets for one-dimensional consolidation PINN and check performance
- Implement undrained bearing capacity problem into a PINN for both forward and inverse analysis and check performance

The purpose of this thesis was to check the application of PINNs on consolidation and bearing capacity. The first six case studies shows that PINNs can predict 1D consolidation through uncoupled PDEs with good accuracy and in a short amount of time, but also that these models can be improved to produce better results. Also show that the consolidation models can handle corrupted data and still performing reasonably well for both the forward and inverse analyses. The last two case studies shows that PINNs can make the transition from 1D to 2D and shows that PINNs can make produce good predictions with coupled PDEs. The performance for the models have been gone from good in the consolidation cases to very good in the bearing capacity. that are included in this thesis.

The benefit of the work in this thesis is very theoretical and for the consolidation, rather simple. But with the rising applications of machine learning, deep learning and PINNs, the possibility of expansion is very large. The concepts introduced here can be expanded upon with addition more complex coupled problems with more material parameters. By optimizing the models created here, the forward analysis would predict the numerical solutions faster on very little data because of the physical constraint. This could be very useful for use in a digital-twin where quick numerical predictions wanted. The inverse analysis would be very useful for optimization of constitutive material and model parameters. It is clear that PINNs can be implemented for both forward and inverse analysis for a variety of problems and have great performance while doing so in a reasonably short time.

References

Ren, X., Li, X., Ren, K., Song, J., Xu, Z., Deng, K., & Wang, X. (2021). Deep learning-based weather prediction: A survey. *Big Data Research*, *23*, 100178. https://doi.org/https://doi.org/10.1016/j.bdr.2020.100178

Chen, X., Wang, X., Zhang, K., Fung, K.-M., Thai, T. C., Moore, K., Mannel, R. S., Liu, H., Zheng, B., & Qiu, Y. (2022). Recent advances and clinical applications of deep learning in medical image analysis. *Medical Image Analysis*, *79*, 102444. https://doi.org/10.1016/j.media.2022.102444

Grigorescu, S., Trasnea, B., Cocias, T., & Macesanu, G. (2020). A survey of deep learning techniques for autonomous driving. *Journal of Field Robotics*, *37*(3), 362–386. https://doi.org/10.1002/rob.21918

Raissi, M., Perdikaris, P., & Karniadakis, G. (2019). Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, *378*, 686–707. https://doi.org/https://doi.org/10.1016/j.jcp.2018.10.045

Depina, I., Jain, S., Valsson, S., & Gotovac, H. (2021). Application of physics-informed neural networks to inverse problems in unsaturated groundwater flow. *Georisk: Assessment and Management of Risk for Engineered Systems and Geohazards*, *16*, 1–16. https://doi.org/10.1080/17499518.2021.1971251

Bekele, Y. W. (2020). Deep learning for one-dimensional consolidation.

Lu, L., Meng, X., Mao, Z., & Karniadakis, G. E. (2021). DeepXDE: A deep learning library for solving differential equations. *SIAM Review*, *63*(1), 208–228. https://doi.org/10.1137/19M1274067

Kollmannsberger, S., D'Angella, D., Jokeit, M., & Herrmann, L. (2021). *Deep learning in computational mechanics - an introductory course*. Springer.

Mitchell, T. M. (1997). *Machine learning*. McGraw Hill.

P.Kingma, D., & Ba, J. (2014). *Adam: A Method For Stochastic Optimization*, 1. https://doi.org/https://doi.org/10.48550/arXiv.1412.6980

Taylor, J., Wang, W., Bala, B., & Bednarz, T. (2022). Optimizing the optimizer for data driven deep neural networks and physics informed neural networks.

Uniform distribution. (n.d.). https://deepxde.readthedocs.io/en/latest/_modules/deepxde/nn/initializers.html

Shields, M., & Zhang, J. (2015). The generalization of latin hypercube sampling. *Reliability Engineering [?] System Safety*, *148*, 97. https://doi.org/10.1016/j.ress.2015.12.002

Craig, R., & Knappett, J. (2020a). *Craig's soil mechanics*. Taylor Francis Group.

Tveito, A., & Winther, R. (2009). *Introduction to partial differential equations - a computational approach*. Springer-Verlag Berlin Heidelberg.

Verruijt, A. (2013). *Theory and problems of poroelasticity*. Delft university of Technology.

Smith, C. (2005). Complete limiting stress solutions for the bearing capacity of strip footings on a mohr-coulomb soil. *Géotechnique*, *55*, 607–612. https://doi.org/10.1680/GEOT.2005.55.8.607

Craig, R., & Knappett, J. (2020b). *Craig's soil mechanics*. Taylor Francis Group.

Barnes, G. (2016). *Soil mechanics - principles and practice*. RED GLOBE PRESS.

Plaxis 2d. (n.d.). Retrieved June 12, 2023, from https://www.bentley.com/software/plaxis-2d/

Spyder. (n.d.). Retrieved July 12, 2023, from https://www.spyder-ide.org

Tensorflow. (n.d.). Retrieved June 12, 2023, from https://www.tensorflow.org/overview

Numpy. (n.d.). Retrieved June 12, 2023, from https://numpy.org

Matplotlib. (n.d.). Retrieved June 12, 2023, from https://matplotlib.org

Jacobian and hessian matrix. (n.d.). Retrieved June 13, 2023, from https://deepxde.readthedocs.io/en/latest/modules/deepxde.html?highlight=jacobian#deepxde.gradients.jacobian

Numpy normal. (n.d.). Retrieved June 22, 2023, from https://numpy.org/doc/stable/reference/random/generated/numpy.random.normal.html

Nordal, S. (2020). *Geotechnical engineering - advanced course.* NTNU.

Eiksund, G., Grimstad, G., Janbu, N., Nordal, S., Emdal, A., & Grande. (2019). *5100 theoretical soil mechanics.* NTNU.

Comparison - tesla k80 and geforce mx250. (n.d.). Retrieved July 30, 2023, from https://technical.city/en/video/Tesla-K80-vs-GeForce-MX250

Lu, Y., & Mei, G. (2022). A deep learning approach for predicting two-dimensional soil consolidation using physics-informed neural networks (pinn). *Mathematics*, *10*(16). https://doi.org/10.3390/math10162949

Appendices

A - Github repository

All code and latex-files used in this document are included in the Github repository linked below. Further explanations are given in the readme-file.

Github repository link

- https://github.com/Asbjorvk/Thesis

B - Normalization of undrained PDEs and Yield function

Original PDEs

$$\frac{\partial \sigma_x}{\partial x} + \frac{\partial \tau_{xy}}{\partial y}$$

$$\frac{\partial \tau_{xy}}{\partial x} + \frac{\partial \sigma_y}{\partial y}$$

Normalizing with respect to r and θ

$$r' = \frac{r}{B} \qquad \theta' = \frac{\theta}{\pi}$$

$$\frac{\partial \sigma_x}{\partial x} = \cos(\theta'\pi)\frac{\partial \sigma_x}{\partial r} - \frac{1}{r'B}\sin(\theta'\pi)\frac{\partial \sigma_x}{\partial \theta}$$

$$\frac{\partial \sigma_x}{\partial x} = \cos(\theta'\pi)\frac{\partial \sigma_x}{\partial r'}\frac{\partial r'}{\partial r} - \frac{1}{r'B}\sin(\theta'\pi)\frac{\partial \sigma_x}{\partial \theta'}\frac{\partial \theta'}{\partial \theta}$$

$$\frac{\partial \sigma_x}{\partial x} = \cos(\theta'\pi)\frac{\partial \sigma_x}{\partial r'}\frac{1}{B} - \frac{1}{r'B}\sin(\theta'\pi)\frac{\partial \sigma_x}{\partial \theta'}\frac{1}{\pi}$$

$$\frac{\partial \sigma_y}{\partial y} = \sin(\theta'\pi)\frac{\partial \sigma_y}{\partial r} + \frac{1}{r'B}\cos(\theta'\pi)\frac{\partial \sigma_y}{\partial \theta}$$

$$\frac{\partial \sigma_y}{\partial y} = \sin(\theta'\pi)\frac{\partial \sigma_y}{\partial r'}\frac{\partial r'}{\partial r} + \frac{1}{r'B}\cos(\theta'\pi)\frac{\partial \sigma_y}{\partial \theta'}\frac{\partial \theta'}{\partial \theta}$$

$$\frac{\partial \sigma_y}{\partial y} = \sin(\theta'\pi)\frac{\partial \sigma_y}{\partial r'}\frac{1}{B} + \frac{1}{r'B}\cos(\theta'\pi)\frac{\partial \sigma_y}{\partial \theta'}\frac{1}{\pi}$$

$$\frac{\partial \tau_{xy}}{\partial x} = \cos(\theta'\pi)\frac{\partial \tau_{xy}}{\partial r'}\frac{1}{B} - \frac{1}{r'B}\sin(\theta'\pi)\frac{\partial \tau_{xy}}{\partial \theta'}\frac{1}{\pi}$$

$$\frac{\partial \tau_{xy}}{\partial y} = \sin(\theta'\pi\frac{\partial \tau_{xy}}{\partial r'}\frac{1}{B} + \frac{1}{r'B}\cos\theta'\pi\frac{\partial \tau_{xy}}{\partial \theta'}\frac{1}{\pi}$$

Insert into PDEs

$$\frac{\partial\sigma_x}{\partial x}+\frac{\partial\tau_{xy}}{\partial y}=0=>$$

$$\cos{(\theta^{'}\pi)}\frac{\partial\sigma_x}{\partial r'}\frac{1}{B}-\frac{1}{r^{'}B}\sin{(\theta^{'}\pi)}\frac{\partial\sigma_x}{\partial\theta'}\frac{1}{\pi}+\sin{(\theta^{'}\pi)}\frac{\partial\tau_{xy}}{\partial r'}\frac{1}{B}+\frac{1}{r^{'}B}\cos{(\theta^{'}\pi)}\frac{\partial\tau_{xy}}{\partial\theta'}\frac{1}{\pi}=0$$

$$\frac{\partial\tau_{xy}}{\partial x}+\frac{\partial\sigma_y}{\partial y}=\gamma=>$$

$$\cos{(\theta^{'}\pi)}\frac{\partial\tau_{xy}}{\partial r'}\frac{1}{B}-\frac{1}{r^{'}B}\sin{(\theta^{'}\pi)}\frac{\partial\tau_{xy}}{\partial\theta'}\frac{1}{\pi}+\sin{(\theta^{'}\pi)}\frac{\sigma_y}{\partial r'}\frac{1}{B}+\frac{1}{r^{'}B}\cos{(\theta^{'}\pi)}\frac{\partial\sigma_y}{\partial\theta'}\frac{1}{\pi}=\gamma$$

Normalizing with respect to stresses

Larges value is assumed as $q+\gamma B=100\frac{kN}{m^2}+20\frac{kN}{m^3}*10m=300\frac{kN}{m^2}$. Multiplying polar PDE with $r^{'}\gamma B$ yields:

$$r^{'}\pi\cos{(\theta^{'}\pi)}\frac{\partial\sigma_x}{\partial r'}-\sin{(\theta^{'}\pi)}\frac{\partial\sigma_x}{\partial\theta'}+r^{'}\pi\sin{(\theta^{'}\pi)}\frac{\partial\tau_{xy}}{\partial r'}+\cos{(\theta^{'}\pi)}\frac{\partial\tau_{xy}}{\partial\theta'}=0$$

$$r^{'}\pi\cos{(\theta^{'}\pi)}\frac{\partial\tau_{xy}}{\partial r'}-\sin{(\theta^{'}\pi)}\frac{\partial\tau_{xy}}{\partial\theta'}+r^{'}\pi\sin{(\theta^{'}\pi)}\frac{\partial\sigma_y}{\partial r'}+\cos{(\theta^{'}\pi)}\frac{\partial\sigma_y}{\partial\theta'}=r^{'}\pi B\gamma$$

Stress normalization - PDE1

$$\sigma_x^{'} = \frac{\sigma_x}{q+\gamma B} \quad \sigma_y^{'} = \frac{\sigma_y}{q+\gamma B} \quad \tau_{xy}^{'} = \frac{\tau_{xy}}{q+\gamma B}$$

$$r^{'}\pi\cos{(\theta^{'}\pi)}\frac{\partial\sigma_x}{\partial\sigma_x^{'}}\frac{\partial\sigma_x^{'}}{\partial r^{'}} - \sin{(\theta^{'}\pi)}\frac{\partial\sigma_x}{\partial\sigma_x^{'}}\frac{\partial\sigma_x^{'}}{\partial\theta^{'}} +$$

$$r^{'}\pi\sin{(\theta^{'}\pi)}\frac{\partial\tau_{xy}}{\partial\tau_{xy}^{'}}\frac{\partial\tau_{xy}^{'}}{\partial r^{'}} + \cos{(\theta^{'}\pi)}\frac{\partial\tau_{xy}}{\partial\tau_{xy}^{'}}\frac{\partial\tau_{xy}^{'}}{\partial\theta^{'}} = 0$$

$$r^{'}\pi\cos{(\theta^{'}\pi)}(q+\gamma B)\frac{\partial\sigma_x^{'}}{\partial r^{'}} - \sin{(\theta^{'}\pi)}(q+\gamma B)\frac{\partial\sigma_x^{'}}{\partial\theta^{'}} +$$

$$r^{'}\pi\sin{(\theta^{'}\pi)}(q+\gamma B)\frac{\partial\tau_{xy}^{'}}{\partial r^{'}} + \cos{(\theta^{'}\pi)}(q+\gamma B)\frac{\partial\tau_{xy}^{'}}{\partial\theta^{'}} = 0 \quad |/(q+\gamma B)$$

$$r^{'}\pi\cos{(\theta^{'}\pi)}\frac{\partial\sigma_x^{'}}{\partial r^{'}} - \sin{(\theta^{'}\pi)}\frac{\partial\sigma_x^{'}}{\partial\theta^{'}} +$$

$$r^{'}\pi\sin{(\theta^{'}\pi)}\frac{\partial\tau_{xy}^{'}}{\partial r^{'}} + \cos{(\theta^{'}\pi)}\frac{\partial\tau_{xy}^{'}}{\partial\theta^{'}} = 0$$

Stress normalization - PDE2

$$r^{'}\pi\cos{(\theta^{'}\pi)}\frac{\partial\tau_{xy}}{\partial\tau_{xy}^{'}}\frac{\partial\tau_{xy}^{'}}{\partial r^{'}}-\sin{(\theta^{'}\pi)}\frac{\partial\tau xy}{\partial\tau_{xy}^{'}}\frac{\partial\tau_{xy}^{'}}{\partial\theta^{'}}+$$

$$r^{'}\pi\sin{(\theta^{'}\pi)}\frac{\partial\sigma_{y}}{\partial\sigma_{y}^{'}}\frac{\partial\sigma_{y}^{'}}{\partial r^{'}}+\cos{(\theta^{'}\pi)}\frac{\partial\sigma_{y}}{\partial\sigma_{y}^{'}}\frac{\partial\sigma_{y}^{'}}{\partial\theta^{'}}=r^{'}\pi B\gamma$$

$$r^{'}\pi\cos{(\theta^{'}\pi)}(q+\gamma B)\frac{\partial\tau_{xy}^{'}}{\partial r^{'}}-\sin{(\theta^{'}\pi)}(q+\gamma B)\frac{\partial\tau_{xy}^{'}}{\partial\theta^{'}}+$$

$$r^{'}\pi\sin{(\theta^{'}\pi)}(q+\gamma B)\frac{\partial\sigma_{y}^{'}}{\partial r^{'}}+\cos{(\theta^{'}\pi)}(q+\gamma B)\frac{\partial\sigma_{y}^{'}}{\partial\theta^{'}}=r^{'}\pi B\gamma\quad|/(q+\gamma B)$$

$$r^{'}\pi\cos{(\theta^{'}\pi)}\frac{\partial\tau_{xy}^{'}}{\partial r^{'}}-\sin{(\theta^{'}\pi)}\frac{\partial\tau_{xy}^{'}}{\partial\theta^{'}}+$$

$$r^{'}\pi\sin{(\theta^{'}\pi)}\frac{\partial\sigma_{y}^{'}}{\partial r^{'}}+\cos{(\theta^{'}\pi)}\frac{\partial\sigma_{y}^{'}}{\partial\theta^{'}}=\frac{r^{'}\pi B\gamma}{(q+\gamma B)}$$

Yield function

$$S_u^2=(\frac{\sigma_x-\sigma_y}{2})^2+\tau_{xy}^2$$

$$S_u^2=(\frac{\sigma_x^{'}(q+\gamma B)-\sigma_y^{'}(q+\gamma B)}{2})^2+\tau_{xy}^2(q+\gamma B)^2$$

$$(\frac{S_u}{q+\gamma B})^2=(\frac{\sigma_x^{'}-\sigma_y^{'}}{2})^2+\tau_{xy}^{'2}$$