Physics-Informed Neural Networks for Charge Dynamics in Air

Master's thesis in Complex Adaptive Systems

Árni Konráðsson

DEPARTMENT OF Electrical Egineering

CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2022
www.chalmers.se

Master's thesis 2022

Physics-Informed Neural Networks for Charge Dynamics in Air

Árni Konráðsson

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Electrical Engineering
Division of High Voltage Engineering
Chalmers University of Technology
Gothenburg, Sweden 2022

Physics-Informed Neural Networks for Charge Dynamics in Air

Árni Konráðsson

Supervisors: Olof Hjortstam, Hitachi Energy
Examiner: Yuri Serdyuk, Department of Electrical Engineering

Master's Thesis 2022
Department of Electrical Engineering
Division of High Voltage Engineering
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2022

Physics-Informed Neural Networks for Charge Dynamics in Air
Árni Konráðsson
Department of Electrical Engineering
Chalmers University of Technology

Abstract

Physics-Informed Neural Networks (PINNs) are neural networks trained to consider the laws of physics outlined in Partial Differential Equations (PDEs). PINNs are a new tool aimed at solving difficult problems, such as systems arising from PDEs, complementing or replacing traditional methods that already exist. This thesis focuses on applying PINNs to the drift-diffusion equation and the Poisson equation, which are fundamental PDEs that describe the physics of electrical discharges in air and other gases under high-voltage stress. The aim of the work is to predict the electric field in an air gap between two electrodes and to study how ionic charges drifting in the gap modify the electrostatic field distribution. In the first case, the equations are considered by PINN independently. In the second case, they are coupled together to implement the effect of the space charge on the electric field. The results obtained in both cases are compared with respective analytical solutions and/or solutions obtained from the Finite Element Method (FEM) using COMSOL Multiphysics software. Details of the implementation of PINNs as well as encountered limitations are discussed. In particular, best practices when setting up problems and choosing hyperparameters are highlighted. The obtained results indicate that PINNs can, in some aspects, outperform FEM in accuracy although using significantly longer processing time. Further research is needed to confirm the feasibility of PINNs for electrical gas discharges, taking into account additional effects of more complex geometries and physics. Additionally, the usability of PINNs for solving the inverse problem (i.e., extracting physical parameters defining the PDE from its solution) needs to be analyzed further.

Keywords: Physics-Informed Neural Network, Artificial Neural Network, Partial Differential Equation, Discharge physics, Air discharges, DeepXDE

Acknowledgements

This thesis is written under the supervision of Hitachi ABB Power Grids, a world-leading supplier of grid infrastructure. In particualar it was carried out in collaboration with the research center located in Västerås, Sweden.
My gratitude goes out to my family for having the patience to support me through the project and providing me with immense moral support. I would also like to give special thanks to my supervisors and examiner, Olof Hjortstam, Christian Häger and Yuriy Serdyuk, for providing helpful insights and discussions that have shaped my work.

Árni Konráðsson, Gothenburg, June 2022

Contents

List of Figures

List of Tables

1
Introduction

1.1 Background

Due to the worldwide need for a strong reduction in greenhouse gases to suppress ongoing climate changes, the electrical energy system is under great changes. An important part of the development of high voltage equipment needed for this challenge is to predict the insulation performance of air around such equipment. Physical processes in air discharges are normally modeled utilizing hydrodynamic approaches [1]. One such model is the drift-diffusion equation for charge carriers coupled with Poisson's equation.

$$\begin{aligned} \frac{\partial n_e}{\partial t} + \nabla \cdot (n_e \mathbf{W}_e - D_e \nabla n_e) &= R_e \\ \frac{\partial n_p}{\partial t} + \nabla \cdot (n_p \mathbf{W}_p - D_p \nabla n_p) &= R_p \\ \frac{\partial n_n}{\partial t} + \nabla \cdot (n_n \mathbf{W}_n - D_n \nabla n_n) &= R_n \end{aligned} \tag{1.1}$$

Here, the subscripts e, n and p indicate electrons, positive ions, and negative ions. D stands for diffusion coefficient. The first term represents the rate in change of the number density, n, over time. The second and third terms are divergence of the flux due to advection and diffusion of particles respectively. The source terms of Equation 1.1 incorporate rates of processes to be considered in the model:

$$\begin{aligned} R_e &= \alpha n_e |W_e| - \eta n_e |W_e| - \beta_{ep} n_e n_p + R_0 + R_{ph} \\ R_p &= \alpha n_e |W_e| - \beta_{ep} n_e n_p - \beta_{pn} n_p n_n + R_0 + R_{ph} \\ R_e &= \eta n_e |W_e| - \beta_{pn} n_p n_n \\ R_p + R_e + R_n &= 0 \end{aligned} \tag{1.2}$$

where β_{ep} and β_{np} are electron-ion recombination coefficients and ion-ion recombination coefficients, respectively, R_0 and R_{ph} are the rate of background and photo ionization, respectively, α is the first ionization coefficient and η is the attachment coefficient.

Every one of the source terms have strong non-linear field dependence as most of the parameters in these drift-diffusion equation are dependant on the electric field $\mathbf{E}$, in which case we couple them with the Poisson's equation

$$\begin{aligned} \nabla \cdot (-\epsilon_0 \epsilon_r \nabla V) &= e(n_p - n_e - n_n) \\ \mathbf{E} &= -\nabla V. \end{aligned} \tag{1.3}$$

Here, ϵ_0 is the permittivity of vacuum and ϵ_r is the relative permittivity, V is the electric potential, and e is the electron charge. This is a complex transport and charge generation model with strong non-linear behaviors. Current solution methods such as the Finite Element method (FME) have their shortcomings, the drift-diffusion equation is in-stable, high speed charges need high resolution on the computational grid and small time steps which results in time and memory consuming calculations. The time scales of the problem are also vastly different: parts of the discharge physics take place at short time scales, while other parts, such as drift of ions, take a much longer time. Georgi Karman simplifies Equations 1.1-1.3 to one dimension in his thesis work [2], we will refer to his work for the choices of parameters in Chapter 3.

In recent years, researchers in machine learning have started to develop a method known as physics-informed neural networks (PINNs). This approach was first proposed by Raissi et al.[4][5]
PINNs are neural networks trained to solve supervised learning tasks while respecting physical laws described by general nonlinear partial differential equations (PDE). There are two types of PINNs, the data-driven solution of PDEs and the data-driven discovery of PDEs. In the data-driven solution, also called the forward problem, neural networks are used to solve a given PDE, and in the data-driven discovery of PDEs, also called inverse problems, neural networks are used to find suitable parameters for PDEs. Although the use of experimental data and PINN to find PDEs is a promising area of study, we will focus solely on the forward problem and leave the inverse problem for future studies.

1.2 Purpose

This thesis investigates charge transport in an air gap between two electrodes using PINNs. The PDEs used in the study are Poisson's equation and the drift-diffusion equation, and the long-term goal is to be able to predict electric breakdown in air insulation. The Poisson equation is solved analytically and used for comparison, whereas the drift-diffusion equation is compared to a solution found using FEM. The research questions that this work aims to answer are as follows.

1. How useful are PINNs for problems arising in discharge physics?
2. What are the best practices when using PINNs to solve PDEs?
3. What are the limitations and advantages of using PINNs compared to traditional numerical solvers?

1.3 Limitations

Using PINN to solve Equations 1.1-1.3 would be ideal, but its complexity is too high for the current thesis. We therefore have to accept some limitations. First, we only look at simple geometry, one-dimensional interval in both coaxial and Cartesian coordinates. Second, we have a simple physical system that does not account for ion generation, recombination, and more. Third, we make use of parameters to PDEs

that are found using approximations, simplifications, and results from experiments. This will result in PDE that will never be completely accurate. Typical parameters for air discharge are taken from Karman [2].

1.4 Related Work

The work done in this study is largely based on the work by Raissi et al. [4][5]. Part one of their paper focuses on showing the promise of using PINNs to solve PDEs for both continuous and discrete time. The second paper [5] focuses on finding the best parameters of a PDE describing the observed data. The equations studied are Burger's equation, Schrödinger's equation, Navier-Stokes equation, Kortewedge Vries equation, and Allen-Cahn equation. Since the publication of the article by Raissi et al., there have been a number of studies exploring the application of PINNs. In particular Poisson's equation and diffusion equation in 1D in mechanics [15], the wave equation, KdV and KdV-Burger equations in 1D [13], and the swing equation for both forward and inverse problem [14]. Other applications include the heat transfer equation in 1D [16], coupled PDE [18], and many more.
Using PINNs to replace traditional solvers is already being studied. Markids [12], studies the Poisson equation and how choosing different hyperparameters affects training time and convergence. Many insightful thoughts and lessons can be taken from his paper.

2

Background

2.1 Deep Neural Networks

A deep neural network is an artificial neural network with multiple layers between the input and output layers. Many different neural networks have been developed in the past decades including convolutional neural networks, recurrent neural networks (RNN), residual neural networks (ResNet) and feed forward neural networks (FFNN). We consider FFNN in this study. Using other networks could be of use in further research.
Let us now give a short description of a deep neural network. We have $\mathcal{N}^L(\mathbf{x})$ a $L-$layered neural network from $\mathbb{R}^d$ to $\mathbb{R}^D$ where d is the input dimension and D the output dimension. Layer i has N_i neurons, so $N_0 = d$ and $N_L = D$. Now we denote the weight matrix and bias vector in layer ℓ by $\mathbf{W}^\ell \in \mathbb{R}^{N_\ell \times N_{\ell-1}}$ and $\mathbf{b}^\ell \in \mathbb{R}^{N_\ell}$ respectively. Given a nonlinear activation function σ applied elementwise we can define a FFNN recursively with

$$\begin{aligned} \text{input layer}: &\quad \mathcal{N}^0(\mathbf{x}) = \mathbf{x} \in \mathbb{R}^d \\ \text{hidden layers}: &\quad \mathcal{N}^\ell = \sigma(\mathbf{W}^\ell \mathcal{N}^{\ell-1} + \mathbf{b}^\ell) \in \mathbb{R}^{N_\ell} \text{ for } 1 \leq \ell \leq L-1 \\ \text{output layer}: &\quad \mathcal{N}^L = \mathbf{W}^L \mathcal{N}^{L-1} + \mathbf{b}^L \in \mathbb{R}^D \end{aligned}$$

A schematic graph of a typical FFNN can be seen in Figure 2.1, x is the spatial input, t is the time input, and $\hat{u}$ is the output from the net.

PINNs require computation of derivatives of the network outputs with respect to the inputs. This can be done in four different ways [11], hand-coding analytical derivative, numerical approximation (like finite element method), symbolic differentiation (used in Mathematica, Maple and such) and the fourth is automatic differentiation (AD). Deep learning utilizes a specialized technique of AD called backpropagation for evaluating the derivatives.
Deep neural network represent a compositional function and therefore AD can apply the chain rule repeatedly to compute the derivative. AD can be boiled down to two steps, first a forward pass to calculate the value of all variables and then a backward pass to compute the derivatives. A simple example of how AD works can be found in [10].

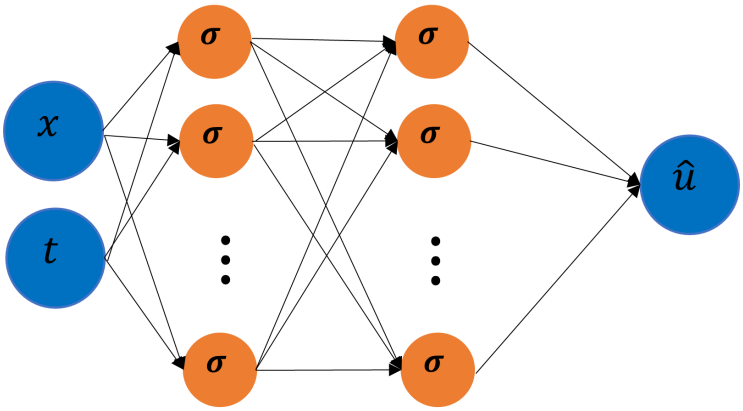

Figure 2.1: Schematic graph of a simple feed forward neural network

2.2 Physics-Informed Neural Network

PINNs are introduced in this section starting with an overview and background. The second part of the section focuses on DeepXDE, a framework for implementing PINNs.

2.2.1 Overview

Deep neural networks need a large amount of training data to obtain accurate results. However when analyzing complex physical systems, the cost of acquiring data is prohibitive which forces us to draw conclusions and make decisions with only partial information. PINNs do not have this need for large amount of data, instead they make use of prior knowledge, such as physical laws, empirically validated results or other domain expertise. This prior knowledge acts as a regularization agent that constrains the space of available solutions to a more manageable size, resulting in less data needed for training.
PINNs can be be used for two types of problems, the forward and the inverse problem. The forward problem is a so-called data-driven solution of PDEs, focusing on finding solutions to a predetermined PDE. The inverse problem is a so-called data-driven discovery of PDEs, focusing on finding parameters of PDEs that fit given data. The work done in [4][5] is split into two types of models, continuous-time and discrete-time models. This study will only consider continuous-time forward problems and leave inverse problems for future research.

The general idea for both data-driven solutions and data-driven discovery of PDEs is as follows. Let us consider a parameterized and nonlinear PDE of the general form

$$u_t + \mathcal{N}[u; \lambda] = 0, \quad x \in \Omega, \quad t \in [0, T] \tag{2.1}$$

where $u(t, x)$ denotes the latent (hidden) solution, $\mathcal{N}[\cdot; \lambda]$ is a nonlinear operator

parameterized by a vector λ and Ω is a subset of $\mathbb{R}^d$. We use the notation u_t, u_x and u_{xx} for, u differentiated by time, $x-$coordinate and twice by $x-$coordinate, respectively.
When dealing with a forward problem, we assume that all λ are known and constant. We define $f(t,x)$ as given by the left-hand side of Equation 2.1,

$$f := u_t + \mathcal{N}[u]. \tag{2.2}$$

An approximate of $u(t,x)$ is found with the use of a deep neural network. This approximation along with 2.2 result in a PINN $f(t,x)$.
The shared parameters between the neural networks $u(t,x)$ and $f(t,x)$ can be learned by minimizing the loss function. In particular, the loss given by summing together the mean squared loss from network u and network f

$$\text{MSE} = \text{MSE}_u + \text{MSE}_f \tag{2.3}$$

where

$$\text{MSE}_u = \frac{1}{N_u}\sum_{i=1}^{N_u}(u(t_u^i, x_u^i) - u^i)^2 \quad \text{and} \quad \text{MSE}_f = \frac{1}{N_f}\sum_{i=1}^{N_f}(f(t_f^i, x_f^i))^2. \tag{2.4}$$

Here $\{t_u^i, x_u^i, u^i\}_{i=1}^{N_u}$ denote the initial and boundary training data on $u(t,x)$ and $\{t_f^i, x_f^i\}_{i=1}^{N_f}$ are the collocation points for $f(t,x)$. The loss MSE_u corresponds to the boundary and initial conditions, while MSE_f enforces the PDE structure at a finite set of so-called collocation points chosen in the domain of interest. Figure 2.2 shows a general PINN schematic. The neural network inputs are points in space (x) and time (t) at the initial and boundary conditions. The net outputs an approximation ($\hat{u}$) of the PDE solution u which is then used to calculate the gradients appearing in the PDE. The results are provided by the function f.
The inverse problem uses an idea similar to the forward problem keeping λ from Equation 2.1 as a variable parameter. λ then becomes a parameter of the PINN $f(t,x)$.

2.2.2 DeepXDE Framework

There are a number of different frameworks for creating machine learning models, among the most popular is the Tensorflow framework created by Google. Tensorflow has many built-in modules and is useful for deep learning. Tensorflow is not specifically built for PINNs, so every function, class, or module would need to be created from scratch. That is where DeepXDE comes in.
DeepXDE was designed by Lu et al. [10], and is built on Tensorflow. It serves both as an educational tool and as a research tool to solve problems in computational science and engineering. It can solve both forward problems given initial and boundary conditions, as well as the inverse problem given specific data. It was designed so that the code written is short and comprehensive, resembling the mathematical formulations.

Solving PDEs with DeepXDE is done in nine steps:

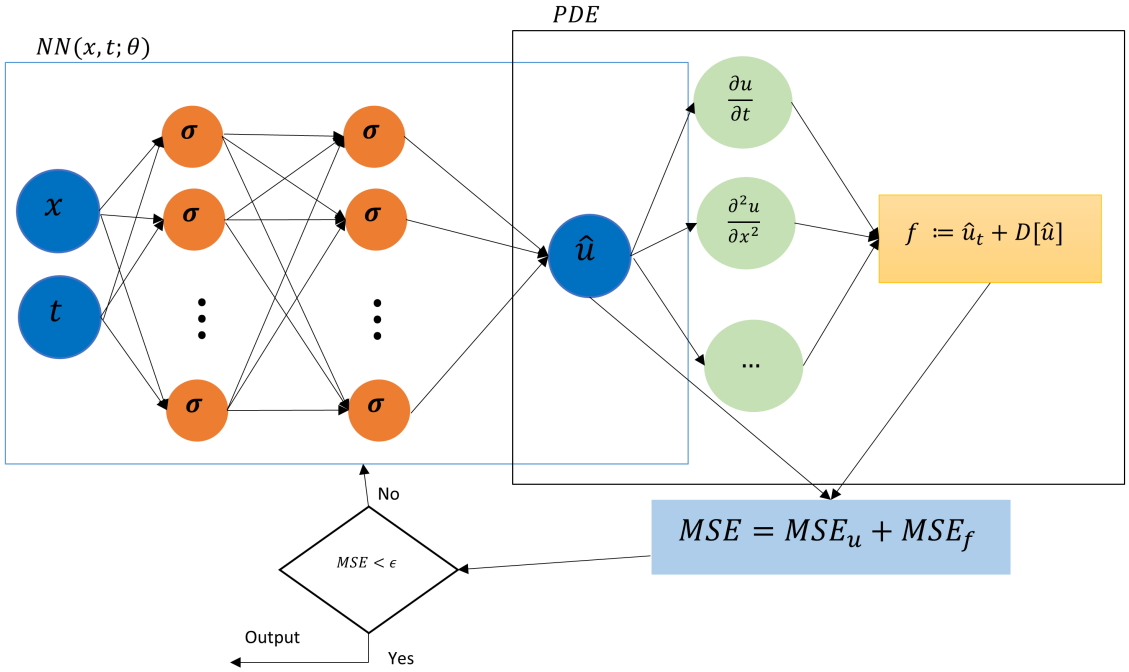

Figure 2.2: Schematic representation of a PINN

1. Specify the computational domain
2. Specify the PDE
3. Specify the boundary/initial condition
4. Use a built in function to create training and testing points for the system defined by 1,2 and 3.
5. Construct neural network
6. Define a model
7. Set hyperparameters and compile the model
8. Train the network
9. Predict the PDE solution

When looking at each step more closely, we first look at the geometry. DeepXDE has many built-in geometries, but we only use an interval and combine it with a time interval when we have time-dependent PDEs. When specifying the PDE, we use grammar from Tensorflow. DeepXDE was developed in Tensorflow 1 and even though it has the option to use other backends, like Tensorflow 2, we found that it worked best using Tensorflow 1. DeepXDE has four common boundary conditions, Dirichlet, Neumann, Robin, and periodic. In this study we only have Dirichlet boundary conditions. The fourth step is to combine the geometry, PDE, and boundary/initial conditions together, creating either time-dependent or time-independent training data. We do this with a one line command where we specify how many training points we want in the domain and the boundaries. We also specify which distribution should be used to sample training points from the domain. If we have an analytical solution to the PDE, can we also input that function, then we can add a metric to our training that evaluates the PINN. Defining a model is

a simple one-line command combining the data from step 4 and the network from step 5. Then we choose hyperparameters, like optimizer, learning rate and more, see Section 3.2. Then it is time to compile and train the model. After training we have a model capable of predicting the desired PDE solution.

2.3 Finite Element Method

Numerical methods are an established way of solving PDEs. The performance of such methods is crucial when analytical solutions are either hard or impossible to find.
The finite element method is a popular numerical method, especially when dealing with complex geometry. To solve a problem, FEM divides a large system into smaller, simpler parts called finite elements. This is done by discretization in the space dimensions which is implemented by construction of a mesh of the object, which has a finite number of points. The method approximates the unknown function over the domain. The simple equations that model these finite elements are then gathered together to make a larger system that models the entire problem. FEM then approximates a solution by minimizing an associated error function.[9] FEM will be used for validation of the equations solved by PINN in the cases where no analytical solution exists.

3
Methods

This chapter introduces the methods used. First, in Section 3.1 we provide a detailed description of the problems we will examine. Then in Section 3.2 we go over what hyperparameters to consider.

3.1 Problem Description

The target of the present work is to investigate some specific features of 1D formulations of the equation presented in 1.1-1.3: Namely, the coupling of Poisson equation and the drift-diffusion for a case with only one charge species and no source terms R. First, we look at each of them separately. Then combine them and show how predictions of the electric field can be made.

3.1.1 Laplace and Poisson's Equations

The general Poisson's equation is given by

$$\nabla^2 U = -\frac{\rho}{\epsilon} \tag{3.1}$$

where U is the electric potential, ρ is the space-charge distribution and ϵ is the permittivity. If we have no space charge present in the system, ρ will be zero, and Equation 3.1 will become Laplace's equation.
The solutions to both the Laplace and Poisson equations in 1D Cartesian coordinates are quite simple, given that the charge distribution is not too complex. A more complex but still simple geometry that is relevant to study of high voltage applications is the coaxial geometry that can be described by coaxial coordinates. Therefore, when looking at Laplace's equation in one-dimensional coaxial coordinates, we model the electric potential as a function of the radius according to

$$\frac{d^2U}{dr^2} + \frac{1}{r}\frac{dU}{dr} = 0 \tag{3.2}$$

Looking at Figure 3.1, we have a coaxial electrode setup with an inner electrode with some radius R_0 and a fixed potential. Then we have an outer electrode, a grounded cage, with radius R_1. Then we say that R_0 is the inner radius and R_1 the outer radius. For simplicity we want to model Equation 3.2 with boundary conditions

$$U(r = R_0) = 1V \quad U(r = R_1) = 0V \tag{3.3}$$

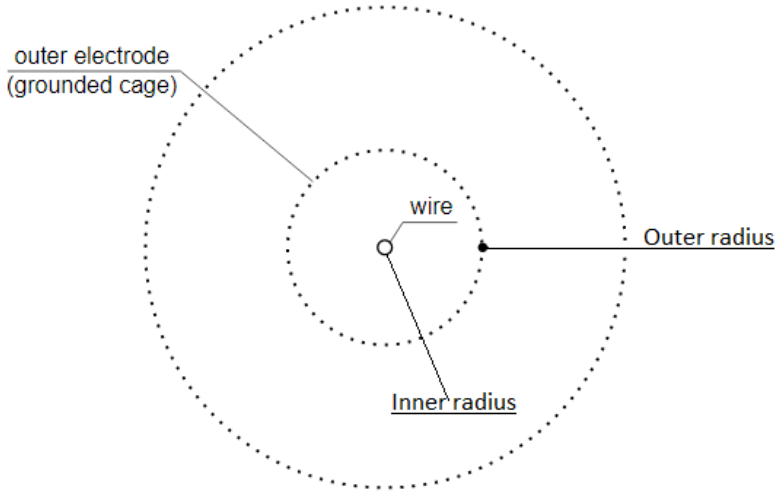

Figure 3.1: Coaxial electrode setup

To study geometries relevant for air discharge experiments [2], the dimensions of $R_0 = 1mm$ and $R_1 = 0.5m$ are relevant to study. We have an analytical solution to Equation 3.2 with these boundary conditions.

$$U(r) = \frac{\ln r - \ln R_1}{\ln R_0 - \ln R_1} \tag{3.4}$$

Thus we can compare with the PINN model. From a physical point of view, the electrical field is a important parameter that can be calculated if the potential distribution is known. We know the relation between the field and the potential expressed in coaxial coordinates is,

$$E = -\frac{dU}{dr} \tag{3.5}$$

and the analytical solution for the field is

$$E = -\frac{1}{r}\frac{1}{\ln R_0 - \ln R_1} \tag{3.6}$$

Figure 3.2 shows the schematic representation of the PINN used to solve the one-dimensional coaxial Laplace equation.

3.1.2 Drift-Diffusion Equation

The next problem we look at is mono polar charge transport in an external electric field in one dimension. A suitable PDE for this is the drift-diffusion equation found in Karman [2], we only look at one charge species and leave out the source term,

$$\frac{\partial n}{\partial t} = -w\frac{\partial n}{\partial x} + D_{iff}\frac{\partial^2 n}{\partial x^2} \tag{3.7}$$

Table 3.1 shows the description and values used for the parameters of Equation 3.7, all parameters are taken from Karman [2].

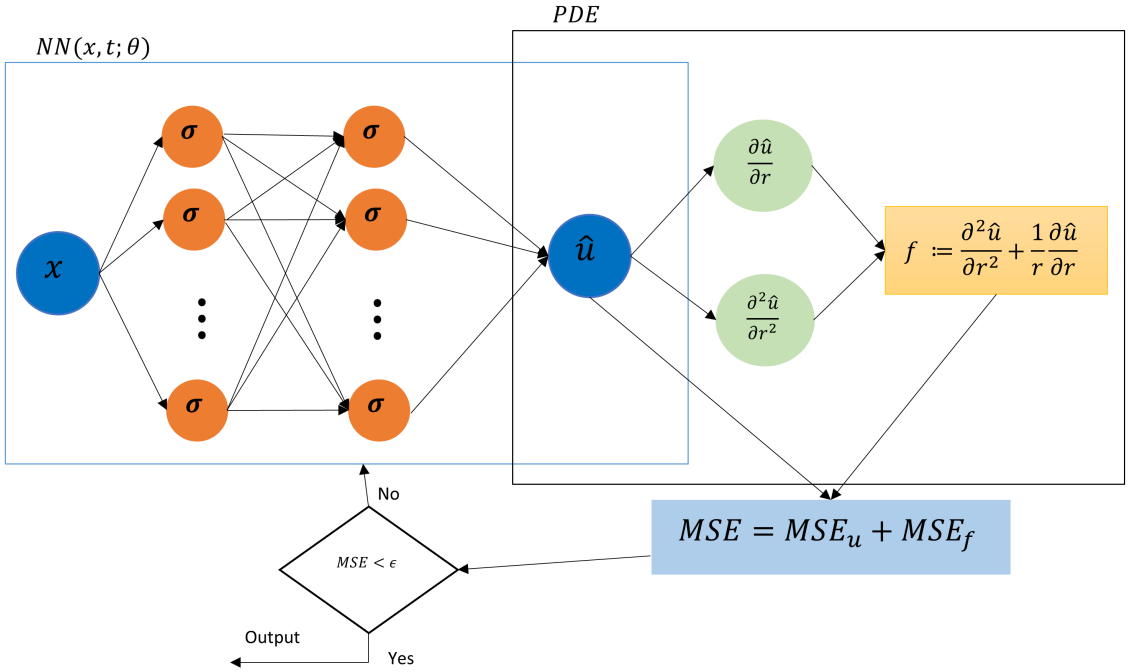

Figure 3.2: Schematic representation of PINN for 1D Laplace equation

Now we have two electrodes separated by one meter air gap. We have a constant external electric field over the gap. We want to model what happens when we inject n_{inj} ions into the left boundary, given a constant density n_0 before injection. This can be modeled using the boundary conditions

$$n(t, x = 0) = n_{inj}, \quad n(t = 0, x) = n_0 \tag{3.8}$$

Table 3.1: Parameters and relevant descriptions of the drift-diffusion equation for ionic drift in air [2].

n	ion density	$(ions/m^3)$	
w	speed of charges	(m/s)	$w = \mu_n E_{ext} = 200$
D_{iff}	Diffusion coefficient	(m^2/s)	$D_{iff} = \frac{\mu_n k_b T}{q} = 5.05e^{-6}$
μ_n	mobility of positive ions	(m^2/Vs)	$\mu_n = 2e^{-4}$
E_{ext}	External electric field	(V/m)	$E_{ext} = 1e^6$
k_b	Boltzmann's constant	(J/K)	$k_b = 1.38e^{-23}$
T	Temperature	(K)	$T = 293$
q	elementary electric charge	(C)	$q = 1.602e^{-19}$

Figure 3.3 shows the schematic representation of the PINN used to solve the one-dimensional time-dependent drift-diffusion equation.

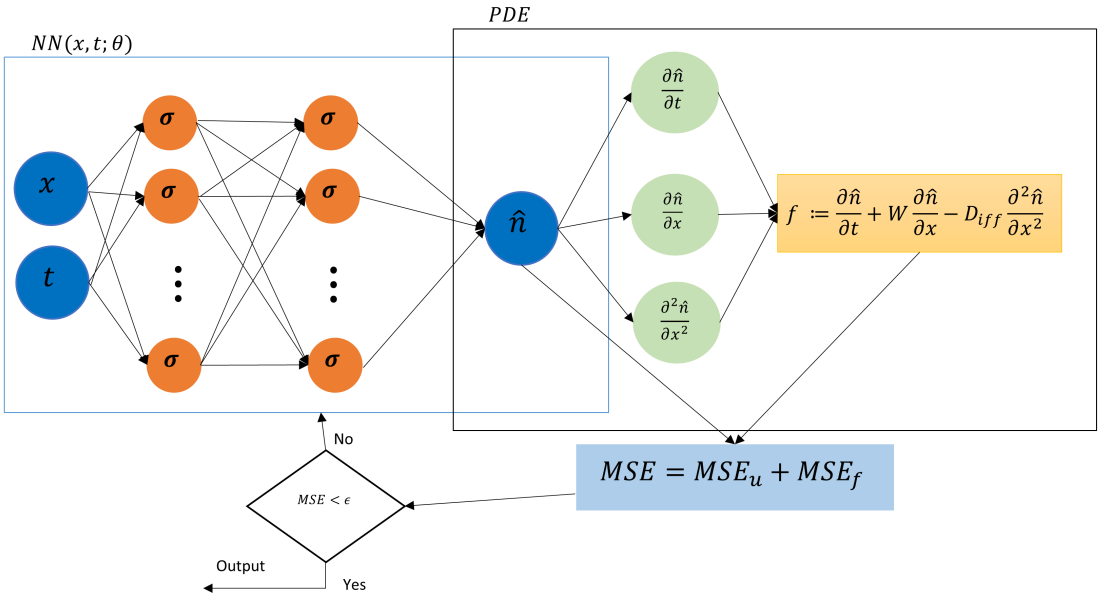

Figure 3.3: Schematic of the PINN network for the drift-diffusion equation

3.1.3 Mono Polar Charge Transport Coupled with Poisson's equation

Now we want to see how the electric field distribution changes over time in the air gap if ionic charges are drifting in the electric field. For this we couple Equation 3.7 with Poisson's equation in 1 dimensional Cartesian coordinates.

$$\begin{aligned} \frac{\partial n}{\partial t} &= -w\frac{\partial n}{\partial x} + D_{iff}\frac{\partial^2 n}{\partial x^2} \\ \frac{\partial^2 U}{\partial x^2} &= -\frac{q \cdot n}{\epsilon_0} \end{aligned} \tag{3.9}$$

Here the charge distribution n feeds into Poisson's equation and the electrical field can be calculated by, $E = -\frac{\partial U}{\partial x}$. At one electrode we have $1MV$ electric potential and at the other zero. We start with a low injection of charges, with a charge density of $1ion/m^3$ at the injecting boundary, under so low a charge density, the equations are essentially uncoupled. We then increase the charge density, aiming to have a solution for a model with injection of $5e^{13}ions/m^3$.

Figure 3.4 shows a schematic of the PINN for Equation 3.9.

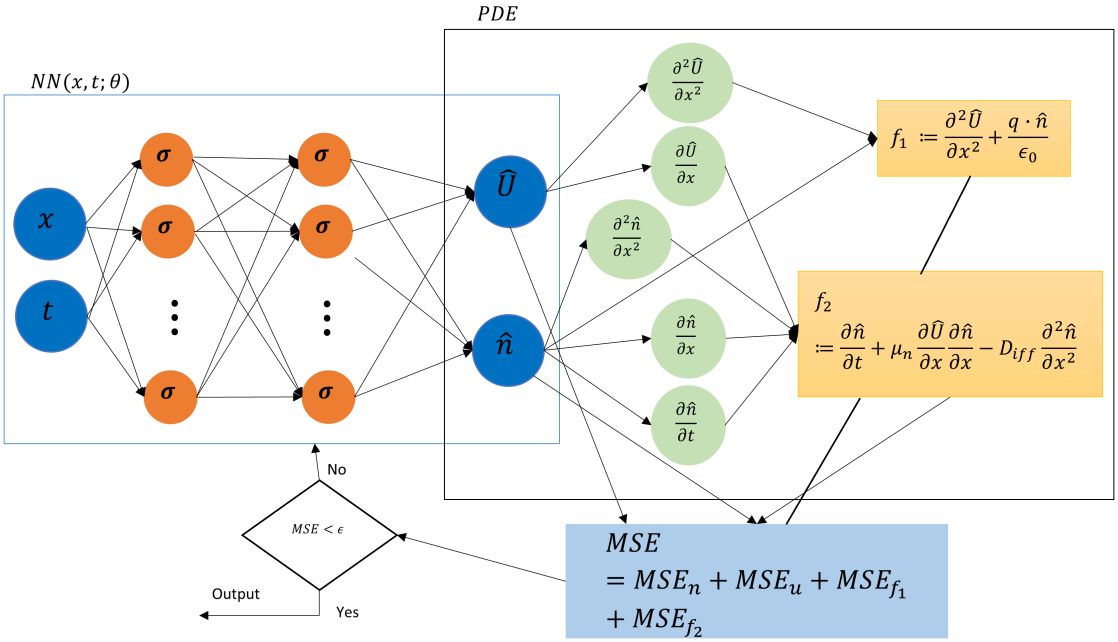

Figure 3.4: Schematic of PINN network for mono polar charge transport coupled with Poisson equation

3.2 Hyperparameters

When creating PINNs there are numerous so-called hyperparameters that need to be considered. Table 3.2 lists them and gives a brief description.

3.2.1 Activation Function tanh and Scaling.

Choosing the activation function is an important step for its performance, different activation functions can provide considerably different accuracy and convergence. In fact, it has been shown theoretically [6] and confirmed experimentally [12] that non-smooth activation functions such as ReLU (Rectified Linear Unit) are not consistently convergent, so it is recommended to use smooth functions such as the hyperbolic tangent, tanh, or the sigmoid function. In this paper, we choose tanh as the activation function. Tanh is defined by

$$\tanh(x) = \frac{2}{1 + e^{-2x}} - 1 \tag{3.10}$$

and squeezes values into a range between -1 and 1. Because of this, it is important to rescale the input to the network and to scale the parameters of the PDE so that the solution is also in this range. If we do not rescale, then training can become difficult, as the net output is of order 1 but the output of the PDE could be of order 10 or even higher.
When selecting which activation function to use, we tried training with every built-in function in DeepXDE. It soon became clear that non-smooth functions did not

Table 3.2: Description of hyperparameters of PINNs

Activation function	Adds non-linearity to neural networks, tanh and sigmoid are most used in PINNs
Scaling	Rescaling of PDE parameters and inputs to the network
Network architecture	Number of hidden layers and number of nodes in each layer
Collocation Points	Points where the PDE residual function is evaluated at.
Optimizer	Algorithm used to optimize weights of neural network. Most commonly L-BFGS or Adam
Learning rate	Determines how large a step to take at each iteration while moving towards the minimum of the loss function.
Loss weights	How much each loss function should contribute to the overall loss
Hard boundary conditions	Function applied to the output of the neural network that ensures that one or more boundary conditions are met
Loss function	The loss function in a neural network qualifies the difference between the expected outcome and the outcome from the model

converge as fast as the others and when comparing the smooth functions, tanh was always either fastest to converge or the most accurate. We then decided tanh was a good choice.

3.2.2 Collocation Points

Collocation points are the input points in the integration domain making up the training data set. In practice, a large number of collocation points result in larger values of the loss function as well as longer training time. The distribution of the collocation points can also impact the loss. The three most used distributions are uniform, pseudo-random and Sobol. In a uniform distribution, the data set is uniformly spaced on the simulation domain. In a pseudo-random distribution, the points are sampled randomly from the simulation domain. In the Sobol distribution, the points are from the Sobol low-discrepancy sequence [20]. Sobol is the default distribution when using DeepXDE. Selecting a good distribution is more relevant when we have higher dimension and larger domain. When we have a small domain and are in 1 dimension, the number of collocation points is more important. After trying out different distributions we decided to use Sobol for the Laplace equation and pseudo-random for the time dependent equations.

3.2.3 Optimizers and Learning Rate

Most widely used optimizer for training neural networks is the Adam optimizer [21]. Adam is a variant of the stochastic gradient descent method that considers the estimates of the first and second order moments of the gradients i.e., the mean and

the uncentered variance. The moments are inserted into the update formula for the network parameters. The main advantages of Adam are that it works well with noisy or sparse data and has fast convergence.
Another optimizer used in present thesis is Limited memory Broyden-Fletcher-Goldfarb-Shanno (L-BFGS) [22] optimizer, it is a quasi-Newton method that uses limited memory and approximates the BFGS algorithm. The L-BFGS algorithm uses an estimate of the inverse of the Hessian matrix to minimize the second-order Taylor expansion of the loss function. Since the Hessian is found with the second order partial derivatives while the Adam optimizer uses the first order derivative, L-BFGS optimizer converges to a more accurate results.

3.2.4 Loss Weights and Hard Boundary Conditions

The loss we have is a combination of two or more losses

$$\text{MSE} = \text{MSE}_u + \text{MSE}_f \tag{3.11}$$

where MSE_u is the loss from boundary and initial conditions and MSE_f is the loss from the PDEs. Say we have two boundary conditions, boundary A and B, and two PDEs, call them C and D. Then the loss becomes

$$\text{MSE} = \text{MSE}_A + \text{MSE}_B + \text{MSE}_C + \text{MSE}_D \tag{3.12}$$

Now, if one of the losses is much larger than the others, we will have a problem training. S.Wang et al. [7] detail this problem and suggest an algorithm to deal with this problem, but for simplicity this study does not implement it. Instead we assign weights to each loss and manually assign values so the losses have similar magnitude.

Another approach that mitigates this problem and while also simplifying the training is to use hard boundary constraints introduced by Lu et al. [8]. Contrary to the normal way to enforce boundary conditions via loss functions, hard boundaries strictly impose the boundary conditions by modifying the network architecture. Only Dirichlet and period BC were considered by Lu and this study will only consider Dirichlet BC.
Dirichlet boundary condition for the solution u is given by

$$u(\mathbf{x}) = g(\mathbf{x}), \quad \mathbf{x} \in \Gamma_D \tag{3.13}$$

where $\Gamma_D \subset \partial\Omega$ is a subset of the boundary. To make the approximate solution $\hat{u}(\mathbf{x};\theta_u)$ satisfy this BC, they construct the solution.

$$\hat{u}(\mathbf{x};\theta_u) = g(\mathbf{x}) + \ell(\mathbf{x})\mathcal{N}(\mathbf{x};\theta_u) \tag{3.14}$$

Here, $\mathcal{N}(\mathbf{x};\theta_u)$ is the network output and ℓ is a function satisfying

$$\begin{cases} \ell(\mathbf{x}) = 0, & \mathbf{x} \in \Gamma_D \\ \ell(\mathbf{x}) > 0, & \mathbf{x} \in \Omega - \Gamma_D \end{cases} \tag{3.15}$$

If Γ_D is a simple geometry we can choose $\ell(\mathbf{x})$ analytically. For example, if $\Gamma_D = a, b$ i.e., the boundary of an interval from a to b, we can choose $\ell(x)$ as $(x-a)(b-x)$

or $(1 - e^{a-x})(1 - e^{x-b})$. For more complex domains it becomes difficult to find $\ell(\mathbf{x})$ analytically so it is possible to use a spline function to approximate $\ell(\mathbf{x})$ but this is outside of the scope of this study.
Now if we impose hard constraints on one of the boundaries the corresponding loss would be zero thus reducing the complexity of the overall loss minimization improving training time and accuracy. Schematic of the network architecture modification can bee seen in Figure 3.5.

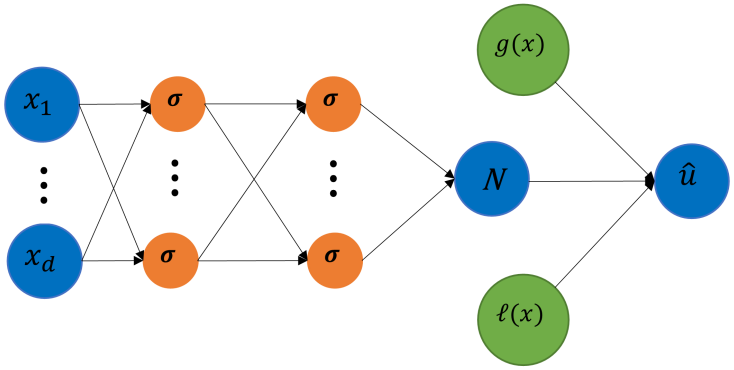

Figure 3.5: Schematic of network modification for applying hard constraints

3.2.5 Loss Function

The loss function in a neural network quantifies the difference between the expected outcome and the outcome produced by the model. From the loss function, we can derive the gradients that are used to update the weights. The average over all losses constitutes the cost. In this paper, we used only the mean squared error (MSE). Other possible loss functions include mean absolute error, logarithmic MSE, mean l2 relative error, softmax cross entropy error, and more.

4
Results

This chapter presents the results of various created models.

4.1 Coaxial Laplace Equation

The first model we consider is the one dimensional coaxial Laplace equation. Like we said in Section 3.1.1, a reasonable relevant target for inner and outer radii is $R_0 = 0.001m$ and $R_1 = 0.5m$. The Laplace equation with these boundaries is highly non-linear, we can increase the linearity by having a larger inner radius.
The theory on using neural network as function approximator states that any function can be approximated with a neural network given sufficiently many nodes. However, larger neural networks take longer to train, and the same can be said for the training set. Starting with a small network and few collocation points is therefore beneficial to keeping training time short.
It was found that it was much easier to solve Equation 3.2 for larger values of R_0 compared to smaller values. To handle this, the following approach was used to be able to solve the equation for as small R_0 as possible.

1. Set an initial R_0
2. Add hard constraints
3. Solve for as low R_0 as possible for the current setup.
4. Change Hyperparameter: collocation points, network architecture, Optimizer and learning rate, activation function.
5. Solve for current R_0, if successful go to Step 3 else go to Step 4

It is worth mentioning that all parameters and the inner and outer radii are of order one so no scaling is required here.
Starting with an inner radius of $0.1m$ we set up a small net with two hidden layers, each with 10 nodes, tanh activation function, sample 30 points on the interval and one on each boundary. Table 4.1 shows how many iterations and how long it takes using either hard boundary conditions or soft boundary conditions. It is clear that using hard boundary conditions speeds up the training time considerably especially when using the Adam optimizer.
After many iterations of changing hyperparameters we where able to find a reliable solution for $0.005m$ inner radius. One setup that gives good results every time is a net with 3 hidden layers with 40 nodes and tanh activation function. We sampled 128 points on the interval and two on the boundaries with hard boundary conditions. The model is trained for 5000 iterations using Adam with 0.001 learning rate and then the L-BFGS optimizer. One realization of this setup can be seen in Figure 4.1

Table 4.1: Time and iterations different boundary conditions and optimizer take until results have a L2 error of less than $1e^{-2}$ for $R_0 = 0.1m$.

Boundary Condition	Optimizer	Nr iterations	time	L2 error
Hard	Adam - lr 0.001	1500	6s	$8e^{-3}$
Soft	Adam - lr 0.001	6000-12000	9-30s	$5 - 8e^{-3}$
Hard	L-BFGS	150	7s	$2.3e^{-5}$
Soft	L-BFGS	314	10s	$1.8e^{-4}$

and the corresponding electric field prediction can be seen in Figure 4.2.
When the inner radius is lower than $0.005m$ we encounter problems, mainly because the training loss does not converge to a good solution consistently. Chapter 5 details possible reasons for this and possible solutions.

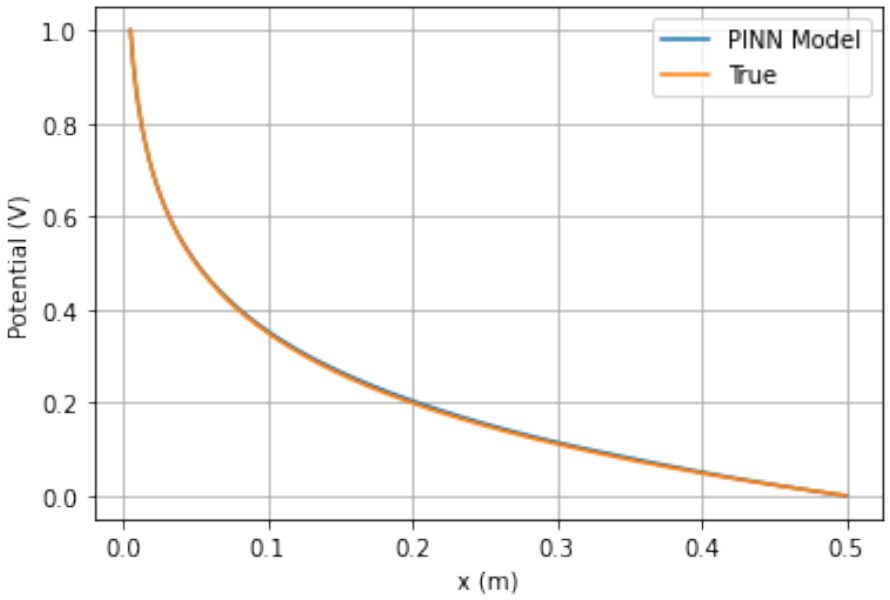

Figure 4.1: Comparison of model prediction of the potential and the true values for the Laplace coaxial equation

4.2 Drift-Diffusion Equation

We mentioned in Section 3.1.2 that a realistic values for the drift speed w is 200 m/s and the diffusion coefficient D_{iff} is $5.05e^{-6}$. Like with the Laplace equation, it is beneficial to look at a simplified case, so we choose to ignore diffusion, i.e. set D_{iff} to zero, and set the drift speed to be 1 m/s. The omission of diffusion is not an unreasonable simplification because the diffusion is small relative to the system. We find a good setup, using a net with 8 layers of 20 neurons, with 500 points in the domain, 200 on the boundaries and 150 at the initial time. To see how well the PINN can model charge transport we used FEM with the help Comsol [19] using implementations found in Karman [2]. We compare the solution we get from PINN

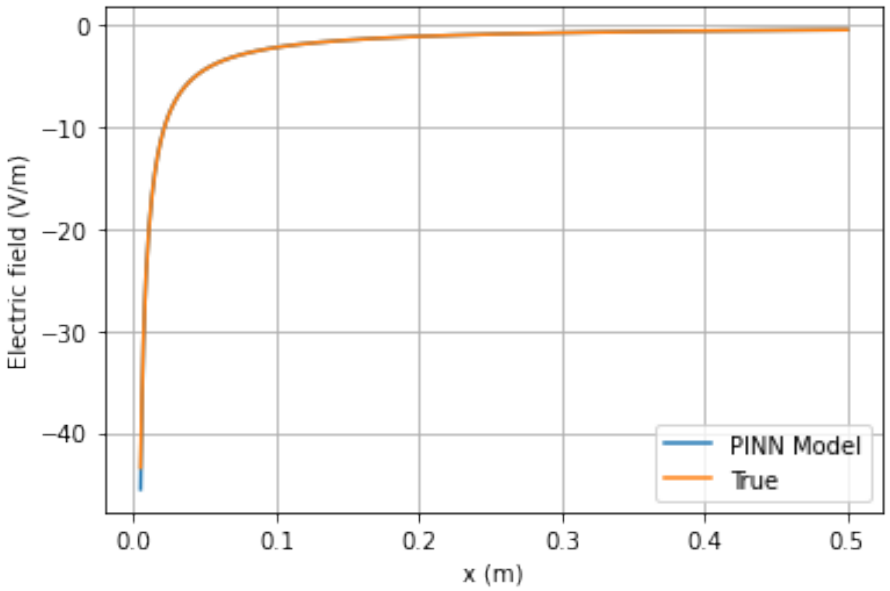

Figure 4.2: Comparison of model prediction of the electrical field and the true values for Laplace coaxial equation

to the solution found using Comsol with 200 mesh points both with and without numerical stabilization.

Figures 4.3 and 4.4 show a comparison of the PINN model with Comsol simulations for 11 different times; each drop from 1 to zero is a prediction for a different time.

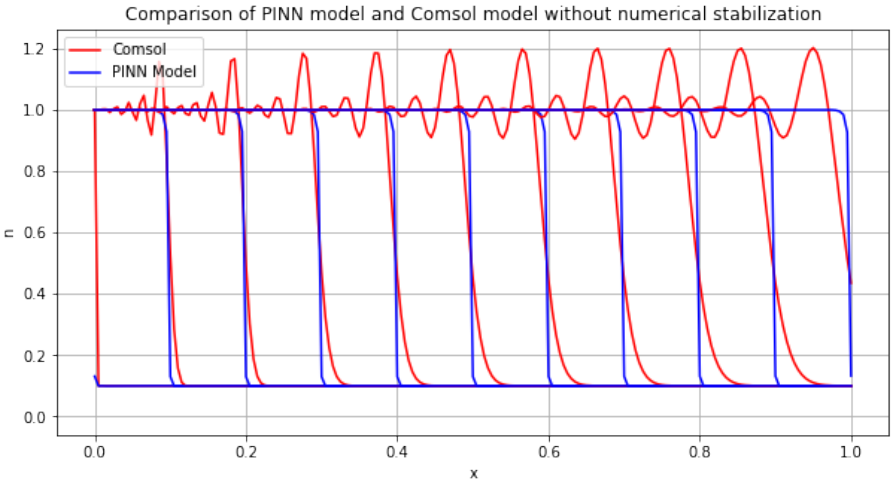

Figure 4.3: Comparison of the drift model with the solution found using Comsol without adding numerical stabilization.

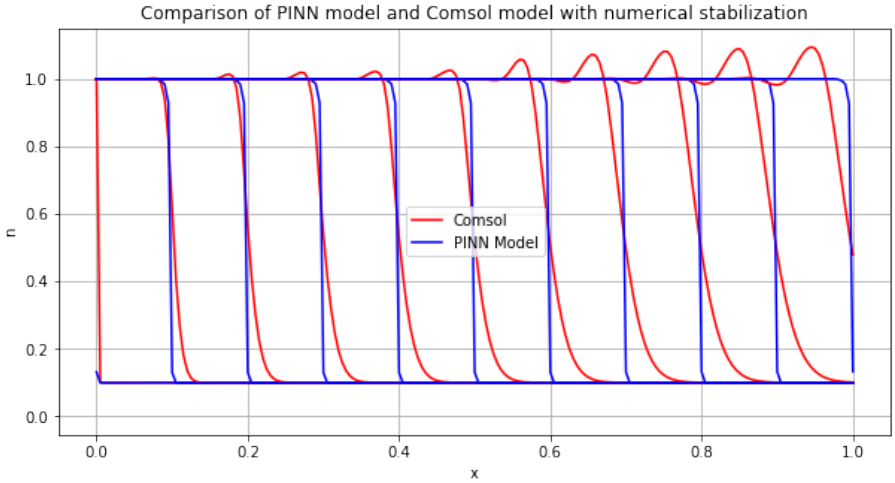

Figure 4.4: Comparison of drift model with solution found using Comsol with numerical stabilisation

Table 4.2: Test loss, training time, and optimizer specifications for PINNs shown in Figures 4.3-4.7.

Figure	Adam	lr	L-BFGS	Time	Loss
4.3 and 4.4	-	-	356	37s	$4.2e^{-8}$
4.5	-	-	1210	269s	$2.9e^{-6}$
4.6	-	-	911	11s	$3.2e^{-3}$
4.7	2000	0.01	521	30s	$7.5e^{-7}$

Now we move on to the more realistic case with $w = 200m/s$ and $D_{iff} = 5.05e^{-6}$. First thing to consider is scaling, since w is not of order one. After dividing Equation 3.7 by w, we have all parameters of order one:

$$\frac{1}{w}\frac{\partial n}{\partial t} + \frac{\partial n}{\partial x} + \frac{D_{iff}}{w}\frac{\partial^2 n}{\partial x^2} = 0 \tag{4.1}$$

It is also worth noting that applying hard constraints to this case does not improve training and convergence as consistently as for the Laplace equation. The reason for this could be the discontinuity on the left boundary.
When using the same model setup as in the previous example, we can get good results (Figure 4.5, takes 3.5-5min), however we can also get results that are not as good (Figure 4.6, takes 0-3min). This randomness comes from how the values in the weight matrix are initialized. We can achieve faster training time by first training with Adam (as recommended by Markids [12]). The results obtained from this setup are still subject to the randomness of the initialization. It is difficult finding a configuration that consistently outputs good results. Figure 4.7 shows best results found for Equation 4.1, it took 30s with the same setup as before except it is trained for 2000 iterations with Adam with learning rate 0.01 before training with L-BFGS.

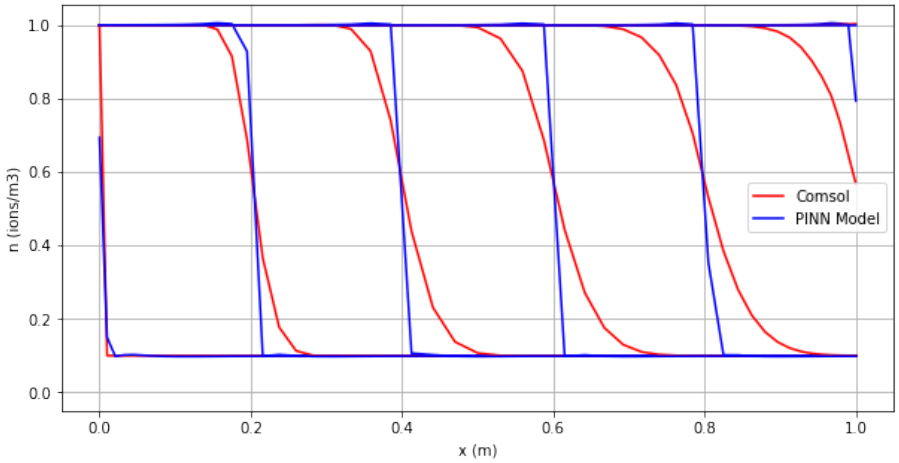

Figure 4.5: Good result form a drift-diffusion PINN model compared with solution found using Comsol with numerical stabilization on 50 mesh point grid

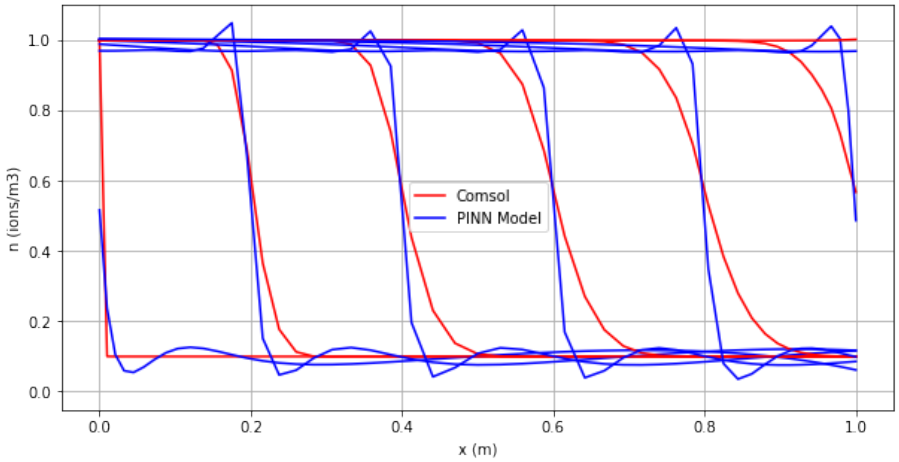

Figure 4.6: Not so good result form a drift-diffusion PINN model compared with solution found using Comsol with numerical stabilisation on 50 mesh point grid

4.3 Mono Polar Charge Transport Coupled with Poisson's Equation

Next, we couple the two previous equations, Poisson and the drift-diffusion equation, together. We use the knowledge that we have gained from the previous equations. It works well to use hard boundary conditions for Poisson's equation but not for the drift diffusion equation and we should use scaling when parameters are not of order one. It is also beneficial to use Adam for a few thousand iterations before using L-BFGS. As mentioned in Section 3.1.3, we first look at the case where we have a low injection of charges. Using the setup that gave the best result for the drift-diffusion equation we get good results, see Figure 4.8, Table 4.3 shows how long training takes and how low training loss we can achieve.

When increasing the concentration of charges injected we run into problems, as

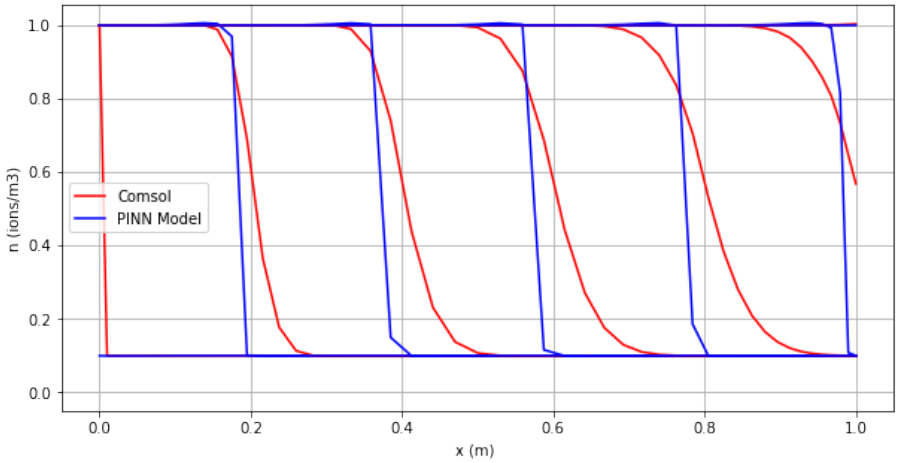

Figure 4.7: Best result form a drift-diffusion PINN model compared with solution found using Comsol with numerical stabilisation on 50 mesh point grid

Table 4.3: Test loss, training time, and optimizer specifications for PINNs shown in Figures 4.8-4.9.

Figure	Adam	lr	L-BFGS	Time	Loss
4.8	2000	0.01	850	96s	$4.4e^{-6}$
4.9	4000	$1e^{-2}, 1e^{-3}, 5e^{-4}, 1e^{-4}$	10482	312s	$4.6e^{-4}$

stated before it is better to have parameters and boundary conditions of order one, and when scaling the injection for the drift-diffusion equation we have to re-scale it up for when it enters in the Poisson equation so the effects are measurable in the electric field. When computed using the same network setup as before we get decent results in 2 minutes, we can improve our results by increasing the number of layers and having more iterations with Adam, the improvement in the electric field prediction will however not be substantial and the increase in computation time is high. Figure 4.9 shows one realization when injection is $1e^{9} ions/m^3$ (highest we where able to solve for). Training time, optimizer specification, and loss achieved can be seen in Table 4.3.
Figure 4.9 shows that we can detect changes in the electric field, although they are small. When increasing the injection further the loss becomes large and L-BFGS gets stuck in local minima and never getting close to the global minima.

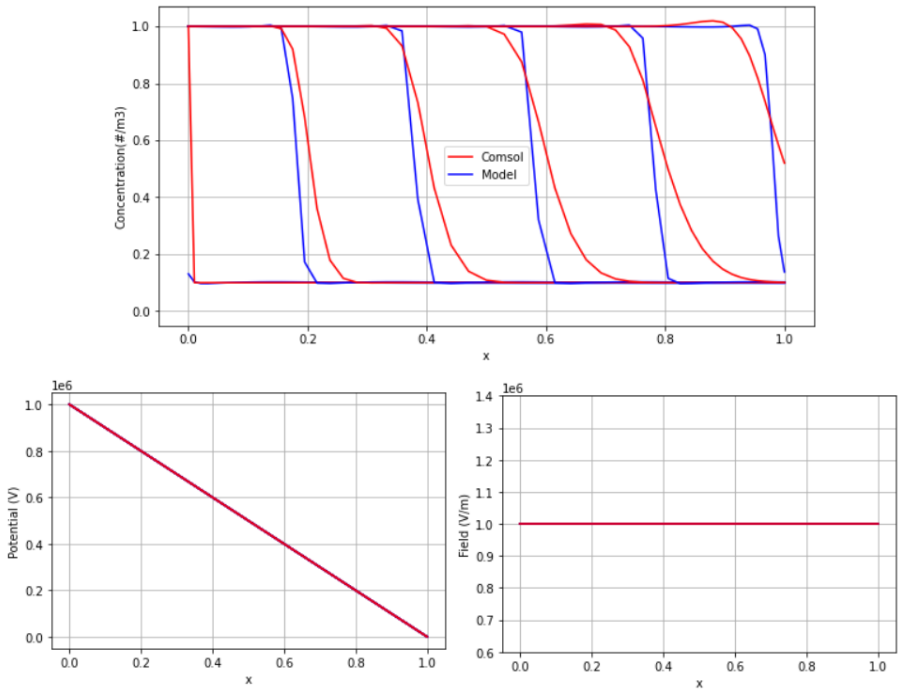

Figure 4.8: Coupled PDE PINN model with low injection compared with solution found using Comsol with numerical stabilization on 50 mesh point grid

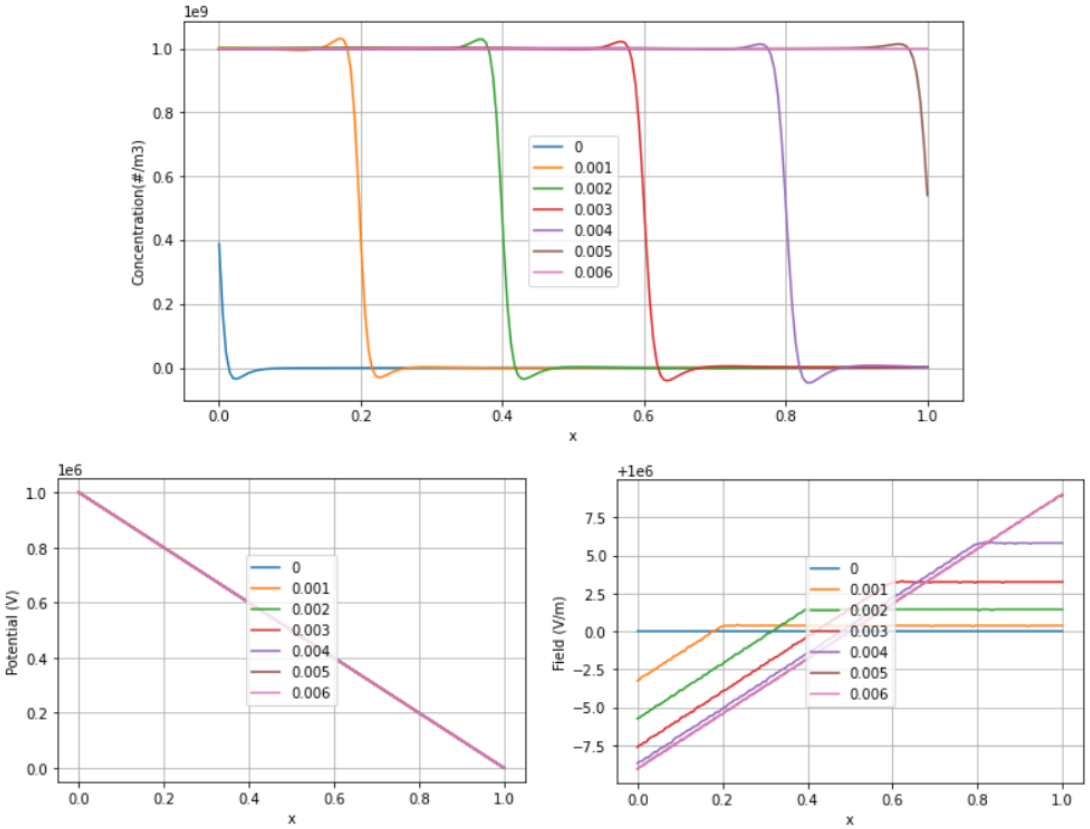

Figure 4.9: Coupled PDE PINN model with the highest injection solved, $1e^9$ ions/m^3.

4.4 Summary of Best Practices

For all equations:

- Start with a simple problem.
- Have a smooth activation function. In our case tanh was used.
- Scale parameters such that they are of order one.
- Use hard boundaries or loss weights to ensure losses have similar magnitudes.
- Start training with adam and then use L-BFGS optimizer.

In the Laplace set-up we get better results increasing the number of neurons in each layer than increasing the number of layers. However in the time dependent cases, drift-diffusion and coupled equation, we get better results using deeper nets with fewer nodes in each layer. The network used for each problem can be seen in Table 4.4

Table 4.4: Number of layers and neurons used for the best results for each problem.

Problem	Layers	Neurons
Laplace best	3	40
Drift-diffusion	8	20
Coupled equations	8	20

5
Discussion

Now we have gone through one approach to solve coupled PDEs with PINN model and it is time to reflect on what lessons we can learn. The coaxial Laplace equation shows that even for a simple example that has an analytical solution, the PINN can run into problems if the system is highly nonlinear. When the inner radius is smaller, the non-linearity of the system increases. This could be the reason for the difficulties we have when set R_0 lower than 0.005m. Possible solutions that have not been explored include changing the network architecture using another type of network, such as ResNet. ResNet uses the input at every layer and we have the input in the PDE so that could be beneficial. Another possible solution could be to write Equation 3.2 differently. When r is small, $1/r$ is large and having large values in the PDE can prove troublesome in training as we saw in the coupled equation. Therefore, it could be beneficial to multiply Equation 3.2 by r.
The drift-diffusion equation was more of a success, it shows that it is possible to get more accurate results form a PINN model than from FEM. There are still improvements to be made if PINNs are to be used. The PINN is currently much slower than FEM, simulations that take a few seconds in Comsol take up to half a minute with PINN. Then there is the problem of consistency. When the solution space gets more complex like when we introduce diffusion and increase the drift-speed the initial weight matrix starts to have a bigger effect on weather we converge to a good solution or not. To help mitigate this problem it might be useful use transfer learning, i.e. train a net on a similar problem that is simpler and then use that as the initial net for the more complex problem.

When we moved on to the coupled equation it was beneficial to have already looked at the equations separately. That allowed us to find out what worked well for each equation and implement it for the coupled equation. When all the parameters of the coupled PDE were of order 1, the PINN was able to find good solutions. However when some parameter gets large it makes the training slow and inefficient. Possible solutions are to re-scale the whole system or possibly applying a logarithmic transform to the PDEs making everything a lot smaller.

5.1 Future Work

We have only just began answering the question, How useful are PINNs for problems arising in discharge physics. Alot more research is needed to determine that. In this section we will suggest how we think it is possible to build upon our work.

5.1.1 Improve Current Work

We think that the next step would be to find a working solution to the mono polar charge transport equation coupled with Poisson's equation for up to $5e^{13}$ charges injected. As mentioned earlier in the report several possible strategies for improved convergence are suggested such as using: ResNets, transfer learning, RNN, more advanced weights for the loss function, transformer network or another loss function [1]. Then adding more complexity to the equations by introducing ion-generation, ion-ion recombination, electrons and possibly more. Having done that, we think it would be a good idea to take the Poisson equation and solve it in 2D and 3D with increasingly complex geometry. To better understand the advantages of PINN over FEM with respect to propagation of steep charge density gradients a systematic comparison between FEM and PINN is needed to be done. In such as comparison PINN collocation points and FEM mesh and time steps needs to be well chosen to make a fair comparison.

5.1.2 Use PINNs for Inverse Problem

When finding the parameters of Equations 1.1 and 1.2 many assumptions and simplifications are made, resulting in only approximates of the real parameters, there is a need for methods that extract these physical parameters from experimental data. One such method is using PINNs data-driven discovery of PDEs. The problem of data-driven discovery of partial differential equations poses the following question, given a small set of scattered and potentially noisy observations of the hidden state of a system, what are the parameters that best describe the observed data? This method has been shown to give good results.

[1] Suggested by thesis opponents Eric Jergéus and Leo Karlson Oinonen.

6
Conclusion

The results show that although PINNs are promising, there is still a need to develop how to use them effectively. They can be useful as shown by good results in the drift-diffusion equation.
From the three case studies we learn that:

- Laplace equation in coaxial coordinates can be solved if the non-uniformity of the solution is not too large.
- The drift-diffusion equation can be solved with a higher accuracy than for FEM. However, the solution time is much longer.
- Coupled Poisson a and drift-diffusion equation can be successfully solved by PINN if the equations are weakly coupled. Further development is needed to solve these equations if they are strongly coupled.

It is a good practice to start with a simple problem and then gradually increase the complexity. One needs to know about every hyperparameter and how changing them affects training time and convergence.
Current limitations of PINNs that we encountered are: Highly non-linear equations rusult in slow convergence or non-convergent models. Also when systems are complex, model accuracy, taining time, and convergence tend to be inconsistent, varying a lot for every initialization of the weight matrix. We found no clear cut advantages using PINNs rather than traditional numerical solvers. Further research into the uses of PINNs for discharge physics is needed.

Bibliography

[1] S.Singh, "Computational framework for studying charge transport in high-voltage gas-insulated systems", Doctor Thesis, 2017

[2] G. Karman, "Simulation and Analysis of Corona Currents in Large Scale Coaxial Geometry due to Triangular Voltages", Masters Thesis, 2013

[3] Hyuk Lee and In Seok Kang, "Neural algorithm for solving differential equations", In: *Journal of Computational Physics* 91.1(1990) Pages 110-131

[4] Maziar Raissi and Paris Perdikaris and George E. Karniadakis, "Physics Informed Deep Learning (Part I): Data-driven Discovery of Nonlinear Partial Differential Equations"(2017), http://arxiv.org/abs/1711.10561

[5] Maziar Raissi and Paris Perdikaris and George E. Karniadakis, "Physics Informed Deep Learning (Part II): Data-driven Discovery of Nonlinear Partial Differential Equations"(2017), http://arxiv.org/abs/1711.10566

[6] S. Mishra and R. Molinaro, "Estimates on the generalization error of Physics Informed Neural Networks (PINNs) for approximating PDEs"(2020), https://arxiv.org/abs/2006.16144

[7] S. Wang and X. Yu and P. Perdikaris, "When and why PINNs fail to train: A neural tangent kernel perspective"(2020), https://arxiv.org/abs/2007.14527

[8] L. Lu and R. Pestourie and W. Yao and Z. Wang and F. Verdugo and S.G. Johnson, "Physics-informed neural networks with hard constraints for inverse design"(2021), https://arxiv.org/abs/2102.04626

[9] Daryl L. Logan (2011). A first course in the finite element method. Cengage Learning. ISBN 978-0495668251.

[10] L. Lu and X. Meng and Z. Mao and G.E. Karniadakis, "DeepXDE: A Deep Learning Library for Solving Differential Equations"(2021), https://doi.org/10.1137/19M1274067

[11] A.G. Baydin and B.A. Pearlmutter and A.A Radul, "Automatic differentiation in machine learning: a survey"(2015), http://arxiv.org/abs/1502.05767

[12] S. Markidis, "The Old and the New: Can Physics-Informed Deep-Learning Replace Traditional Linear Solvers?" (2021), https://arxiv.org/abs/2103.09655

[13] Y. Guo and X. Cao and B. Liu and M. Gao, "Solving Partial Differential Equations Using Deep Learning and Physical Constraints", (2020), https://www.mdpi.com/2076-3417/10/17/5917

[14] G.S Misyris and A. Venzke and S. Chatzivasileiadis, "Physics-Informed Neural Networks for Power Systems", (2019), https://arxiv.org/abs/1911.03737

[15] Q. Zhang and Y. Chen and Z. Yang, "Data driven solutions and discoveries in mechanics using physics-informed neural network", (2020) http://cs229.stanford.edu/proj2020spr/report/Zhang$_C hen_Y ang.pdf$

[16] N. Zobeiry and K.D. Humfeld, "A physics-informed machine learning approach for solving heat transfer equation in advanced manufacturing and engineering applications", (2021) https://www.sciencedirect.com/science/article/pii/S0952197621000798
[17] F. Bragone, "Physics-informed machine learning in power transformer dynamic thermal modelling", Master thesis, 2021
[18] M. Barreau and J. Liu and K.H. Johansson, "Learning-based State Reconstruction for a Scalar Hyperbolic PDE under noisy Lagrangian Sensing", (2021), https://proceedings.mlr.press/v144/barreau21a.html
[19] Comsol Multiphysics software, https://comsol.com
[20] https://en.wikipedia.org/wiki/Sobol$_s$*equence*
[21] D.P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization", 2014, https://arxiv.org/abs/1412.6980
[22] D.C. Liu and J. Nocedal, "On the limited memory BFGS method for large scale optimization.", Mathematical Programming 45, 503–528 (1989), https://doi.org/10.1007/BF01589116

DEPARTMENT OF ELECTRICAL ENGINEERING
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY