Using physics-informed neural networks to reduce data dependence

Hidde van den Bos

s3219496

university of groningen

faculty of science and engineering

University of Groningen
Master's Thesis
Artificial Intelligence

Using physics-informed neural networks to reduce data dependence

under the supervision of
dr. S.M. van Netten (Artificial Intelligence, University of Groningen)
dr. B.J. Wolf (Delft Center for Systems and Control, TU Delft)

Hidde van den Bos (s3219496)

August 2022

Contents

Abstract **5**

1 Introduction **6**

2 Theoretical background **8**
2.1 Differential equations 8
2.2 Physics-informed neural networks 8
2.2.1 Inference 9
2.2.2 Identification 11
2.2.3 Automatic differentiation 12

3 Methods **14**
3.1 Multilayer perceptron 14
3.2 Physics-informed multilayer perceptron 15
3.2.1 Training data 15
3.2.2 Collocation points 15
3.2.3 Combined error 15
3.3 Extreme learning machine 16
3.4 Linear physics-informed extreme learning machine 17
3.5 Non-linear physics-informed extreme learning machine 18
3.5.1 Constrained expressions 18
3.5.2 Collocation points 21
3.5.3 Non-linear harmonic oscillator 21
3.6 Numerical methods 22
3.6.1 Runge-Kutta fourth-order method 22
3.6.2 MATLAB 23
3.6.3 SciPy 23
3.7 Error 24
3.8 Linear harmonic oscillator 24
3.8.1 Analytical solution 25
3.8.2 Hyperparameters 25
3.9 Runtime 27
3.10 Anchor conditions 27
3.11 Noise 27
3.11.1 Hyperparameters 28
3.12 Logistic equation 28
3.12.1 Analytical solution 29
3.12.2 Hyperparameters 30
3.13 Partial differential equation 30
3.13.1 Analytical solution 31

4 Results **33**
4.1 Linear harmonic oscillator 33
4.1.1 Amount of training data 33
4.1.2 Numerical methods 34
4.1.3 Runtime 35

4.2 Anchor conditions 36
4.2.1 Two anchor conditions 36
4.2.2 One boundary condition and one anchor condition 37
4.2.3 One boundary condition and 200 anchor conditions 38
4.3 Noise 39
4.3.1 Two noisy training data 39
4.3.2 50 noisy training data 40
4.4 Logistic equation 41
4.4.1 Amount of training data 41
4.4.2 Numerical methods 42
4.4.3 Runtime 43
4.5 Partial differential equation 44
4.5.1 PIMLP 44
4.5.2 NL-PIELM 45

5 Discussion 46
5.1 Linear harmonic oscillator 46
5.1.1 Amount of training data 46
5.1.2 Numerical methods 46
5.1.3 Runtime 47
5.2 Anchor conditions 48
5.2.1 Two anchor conditions 48
5.2.2 One boundary condition and one anchor condition 48
5.2.3 One boundary condition and 200 anchor conditions 48
5.3 Noise 49
5.4 Logistic equation 50
5.4.1 Amount of training data 50
5.4.2 Numerical methods 52
5.4.3 Runtime 53
5.5 Partial differential equation 53
5.5.1 PIMLP 53
5.5.2 NL-PIELM 53
5.6 General remarks 54
5.6.1 Practical use 54
5.6.2 Flexibility 54
5.6.3 Ease of use 54
5.6.4 Identification 55
5.7 Possible caveats 55
5.8 Further research 56

6 Conclusion 57

References 58

Abstract

Nowadays, with computational power no longer being the bottleneck in many deep learning problems, one of the most prohibitive factors in machine learning has become data acquisition. Physics-Informed Neural Networks (PINNs) incorporate physical information, in the form of a differential equation, into neural networks to reduce the amount of data required to achieve accurate results. In this thesis the performances of an Extreme Learning Machine (ELM), a Linear Physics-Informed ELM (L-PIELM), a Non-Linear Physics-Informed ELM (NL-PIELM), a Multilayer Perceptron (MLP), and a Physics-Informed MLP (PIMLP) are compared on a linear ordinary differential equation and a non-linear ordinary differential equation given varying amounts of data. Their performance is also compared to three algorithms for numerical integration. Furthermore, the performance of the NL-PIELM and PIMLP is analyzed on a non-linear partial differential equation. The PINNs are demonstrated to outperform the ELM and MLP when data is limited, with the L-PIELM and NL-PIELM achieving higher accuracies than the PIMLP. Compared to the algorithms for numerical integration, PINNs are a small improvement accuracy wise, and they also provide a more flexible alternative to the strict data requirements of the algorithms for numerical integration. PINNs, with the exception of the NL-PIELM, are also shown to have the additional capability of reducing the effects of noise, giving significantly more accurate results as the ELM and MLP when provided with noisy training data.

1 Introduction

Over the past years, large strides have been made within deep learning. One of the main forces driving these improvements is the increase in computational power. Consequently, in many problems the bottleneck to achieving accurate results no longer is the amount of computational power available. Instead, in many cases, particularly in supervised learning, the bottleneck has shifted from computational power to data acquisition. As collecting, and if relevant labeling, training data often proves a challenging or laborious task. In many cases, the problem is solved by increasing the amount of training data, using data augmentation. For instance in image recognition, data augmentation can be applied using data warping, transforming the training data to generate new training (Mikołajczyk & Grochowski, 2018), or by using synthetic over-sampling, creating additional samples in feature-space (Wong, Gatt, Stamatescu, & McDonnell, 2016). Also in speech recognition, features of the training data can be warped to increase the amount of training data (Park et al., 2019). One of the most effective methods of data augmentation at the moment are Generative Adversarial Networks (GANs), in which a neural network is applied to generate new data with similar features as the training data (Tanaka & Aranha, 2019). Yet even though current GANs have nearly photorealistic images, they cannot completely compensate a lack of training data (Ravuri & Vinyals, 2019).

However, for many problems data augmentation is not a feasible solution to a shortage of training data, as data augmentation needs training data of a certain size to be applicable. For instance when the training data consists of measurements, data augmentation is not an option, as it is not possible to slightly modify the data, the common approach to augment the data.

As augmenting the data is not an option, the alternative is to decrease the amount of training data required to achieve an accurate result. In some fields, like biology and physics, there are large amounts of prior knowledge available, which is currently not utilised in neural networks. Providing the neural network with this additional information can reduce the amount of training data required to get an accurate prediction. Within physics, a lot of knowledge is available in the form of differential equations. Hence encoding these differential equations into a neural network can reduce the amount of training data required.

Multiple approaches were tried to decrease the amount of training data required by giving the network extra physical information in the form of differential equations. For example using Gaussian process priors (Raissi, Perdikaris, & Karniadakis, 2017a) (Raissi, Perdikaris, & Karniadakis, 2017b), and treating it as a Bayesian inference problem (Owhadi, 2015). These approaches however could only solve linear problems, and so improvements were made and methods based on Gaussian process were devised that could also solve non-linear problems (Raissi & Karniadakis, 2018) (Raissi, Perdikaris, & Karniadakis, 2018). A breakthrough was achieved when the networks utilizing Gaussian processes were replaced with a method more akin to a deep learning implementation of the collocation method (Russell & Shampine, 1972), called Physics-Informed Neural Networks (PINNs) (Raissi, Perdikaris, & Karniadakis, 2019), which outperformed the previous methods. Various studies have confirmed that PINNs indeed are able to solve linear and non-linear differential equations accurately with low amounts of training data (Shokouhi et al., 2021) (Cai, Wang, Wang, Perdikaris, & Karniadakis, 2021).

In this thesis the ability of three PINNs to accurately predict the particular solution of various differential equations is compared to the performance of two neural networks and three numerical methods. For the neural networks, a Multilayer Perceptron (MLP) and an Extreme Learning Machine (ELM) are tested. As PINNs, a Linear Physics-Informed ELM (L-PIELM), a Non-Linear Physics-Informed ELM (NL-PIELM), and a Physics-Informed MLP (PIMLP) are used. The Runge-Kutta fourth-order method, the MATLAB `bvp5c` algorithm, and the SciPy `solve_bvp` algorithm are the numerical methods the performance of the PINNs is compared to.

The performance of the different PINNs, neural networks and numerical methods are compared on three differential equations. On the linear harmonic oscillator, which is a linear Ordinary Differential Equation (ODE), on the logistic equation, which is a non-linear ODE, and on a nameless Partial Differential Equation (PDE). On each problem, the methods will be given varying amounts of training data to determine how they perform with different amounts of data. Furthermore, their ability to deal with noisy data is also analyzed.

In this thesis the research question is: Are physics-informed neural networks an improvement in solving differential equations compared to neural networks? It is expected that in scenarios with small amounts of training data, the PINNs are an improvement, as that is what the PINNs are designed for. The focus of this research will therefore be on the less researched aspects of PINNs; how they behave when not dealing with a small amount of noiseless data. It is probable that PINNs are less of an improvement, if at all, when given larger amounts of training data and when provided with noisy data, as those are not the scenarios PINNs were designed for.

2 Theoretical background

2.1 Differential equations

A differential equation is an equation with one or multiple functions and their derivatives. In most differential equations, the functions describe physical entities, and the derivatives describe their rates of change. The differential equation thus describes the relationship between the functions and their rates of change.

There are two ways to solve differential equations. They can either be solved analytically or approximated numerically. Analytically solving the differential equation entails finding an explicit formula that describes the behaviour of the differential equation, returning the correct result when given its current state, its parameters and the initial conditions. The initial conditions are the values at one boundary of the domain of the differential equation, typically at t = 0. The analytical solution is the preferred solution to a differential equation, as it gives the exact solution. Finding the analytical solution to a differential equation is however complex and time consuming. At the moment there is only a limited amount of differential equations with known analytical solutions. Most differential equations are instead approximated numerically. Numerically approximating a differential equation entails using computational power to find a particular solution given specific initial or boundary conditions. The boundary conditions are the values on the boundaries of the domains of the differential equation. The advantage of numerically approximating a differential equation is that it will always return a solution, the downside is that the solution is an approximation instead of exact, and that the solution is only applicable to one combination of parameters and boundary conditions, and the differential equation has to be solved again for any different combination of parameters and boundary conditions.

Differential equations are generally divided in two categories: Ordinary Differential Equations (ODEs) and Partial Differential Equations (PDEs). ODEs are differential equations with one variable. PDEs on the other hand are differential equations with two or more variables. As PDEs contain multiple variables, they are more difficult to solve. There are also more complex types of differential equations, like stochastic differential equations (Øksendal, 2003), which contain a random component, and functional differential equations (Hale & Lunel, 2013), which take into account the previous states of the model. The scope of this thesis is limited to ODEs and PDEs

The order of a differential equation is determined by the highest derivative in the differential equation. So a differential equation with a function and its third derivative, is a third-order differential equation. Generally speaking, higher order differential equations are more difficult to solve than lower order differential equations (Arnold, 1992).

2.2 Physics-informed neural networks

Physics-Informed Neural Networks (PINNs) are neural networks designed to numerically approximate particular solutions of differential equations. PINNs are a deep learning implementation similar to the numerical collocation method (Russell & Shampine, 1972). The collocation method utilises a basis function, often piecewise polynomial functions, to decrease the space of possible results fitting the boundary conditions. Points within this target space, called collocation points, are used to calculate the solution that fits the differential equation.

The idea of a PINN is that the physical information, i.e. the relevant differential equation, is provided to the neural network to improve the fit of the neural network. This allows the PINN to achieve an accurate result with small amounts of data, unlike standard neural networks, which are designed on the premise of finding solutions given large amounts of data.

In general any neural network can be used in a PINN, examples of existing implementations are the physics-informed echo state network (Doan, Polifke, & Magri, 2020), the physics-informed long short-term memory network (Singh et al., 2019) and the physics-informed convolutional neural network (Gao, Sun, & Wang, 2021). The role the physical information plays can differ in the different networks. The most commonly used neural network in PINNs at the moment is the multilayer perceptron. This is not necessarily because it is the most promising network to use for a PINN, but is primarily because multilayer perceptrons are flexible, easy to implement neural networks. It also helps that the most influential paper regarding PINNs utilizes a MLP (Raissi et al., 2019).

The training data of a PINN consists of values of the dependent variable of the particular solution, with the corresponding values of the independent variables. Apart from initial conditions and boundary conditions, the training data can also contain other values of the dependent variable in between the boundaries of the independent variable. These points will in this thesis be referred to as anchor conditions.

The applicability of PINNs can be divided in two categories: inference and identification, which will be shortly introduced in the following sections.

2.2.1 Inference

Inference, or finding data-driven solutions of partial differential equations, is the application of a PINN to make an accurate prediction when provided with a small amount of training data and the relevant differential equation. This is the more common application of PINNs, and the focus of this thesis. To solve any differential equation using inference, two sets of data points are required: training data and collocation points.

Training data The training data of PINNs in this thesis consists of initial conditions, boundary conditions, and anchor conditions. Initial conditions and boundary conditions are the standard type of training data when solving differential equations, both in PINNs as well as in more traditional numerical methods. Anchor conditions are introduced in this thesis to compare the performance of the PINNs to the standard neural networks with larger amounts of data, as the amount of initial conditions and boundary conditions of a dataset is limited.

The minimum amount of training data required to approximate an accurate result depends on the order of the differential equation. The number of boundary conditions or anchor conditions required for each of the independent variables of the differential equation is determined by the highest order derivative of that variable in the differential equation (Kreiss & Lorenz, 2004). So for a second-order ODE at least two initial conditions, boundary conditions or anchor conditions should be provided to have a unique particular solution. How the training data is enforced depends on the implementation of the PINN.

Collocation points The collocation points constitute the physics-informed part of the PINN. Collocation points are sampled from the domain of the independent variable of the differential equation. The collocation points solely contain the independent variable, and thus do not rely on training data. As collocation points are independent of the training data, they are the key to achieve an accurate result with a PINN when given small amounts of training data.

The collocation points are used to determine the fit of the solution with regards to the differential equation. At each of the collocation points the result of the differential equation is calculated, given the current prediction of the PINN. The prediction and the relevant derivatives of the prediction, usually calculated using automatic differentiation, are inserted in the differential equation, and the result of the differential equation can be calculated. Provided the differential equation is rewritten to equal zero, the error at a collocation points is the result of the differential equation. The error over all the collocation points is the Mean Squared Error (MSE) of the result of the differential equation, given the current prediction, over all the collocation points (Raissi et al., 2019).

In this thesis two methods are used to choose the collocations points. Latin Hypercube Sampling (LHS) (McKay, Beckman, & Conover, 1979) and Chebyshev points of the second kind (Mason & Handscomb, 2002).

LHS is a form of controlled random sampling. With LHS the domain is divided in *n* even spaces, where *n* is the amount of collocation points to be chosen. Within each of these regions the exact value of the collocation point is chosen (pseudo) randomly. So there is some randomness in the values of the collocation points, yet they will be evenly spread along the domain. This method assures a level of randomness within the collocation points, while eliminating the possibility of all collocation points getting clustered together.

When dealing with a PDE, so the collocation points contain multiple variables, the process changes slightly. If the PDE has two variables, the domain of each variable is split in *n* even spaces, resulting in a grid of *n* rows and *n* columns. The collocation points are then divided (pseudo) randomly over the spaces, with the constraint that each row and each column may only contain one collocation point. The precise value of the collocation point is chosen (pseudo) randomly within the specific row and column. The strength of this method is that the amount of collocation points required is independent of the amount of variables in the differential equation. So a PDE with two variables requires the same amount of collocation points as an ODE. With higher order PDEs the process remains the same, the amount of dimensions the LHS has to take into account merely increases.

Chebyshev points of the second kind, also known as Chebyshev extreme points, or Chebyshev–Lobatto points, are the roots of the Chebyshev polynomials of the second kind. The points are divided within the domain of (-1, 1) according to equation 1, where *n* is the number of points used, and *k* labels the specific Chebyshev point.

$$x_k = \cos(\frac{\pi k}{n-1})) \tag{1}$$

Equation 2, where *lb* is the lower bound of the domain of the independent variable, and *ub* the upper bound of the domain of the independent variable, is used to calculate the Chebyshev points of the second kind within any domain. As the collocation points are chosen using an equation containing a cosine, the collocation points will be more numerous along the edges of the domain, while the center

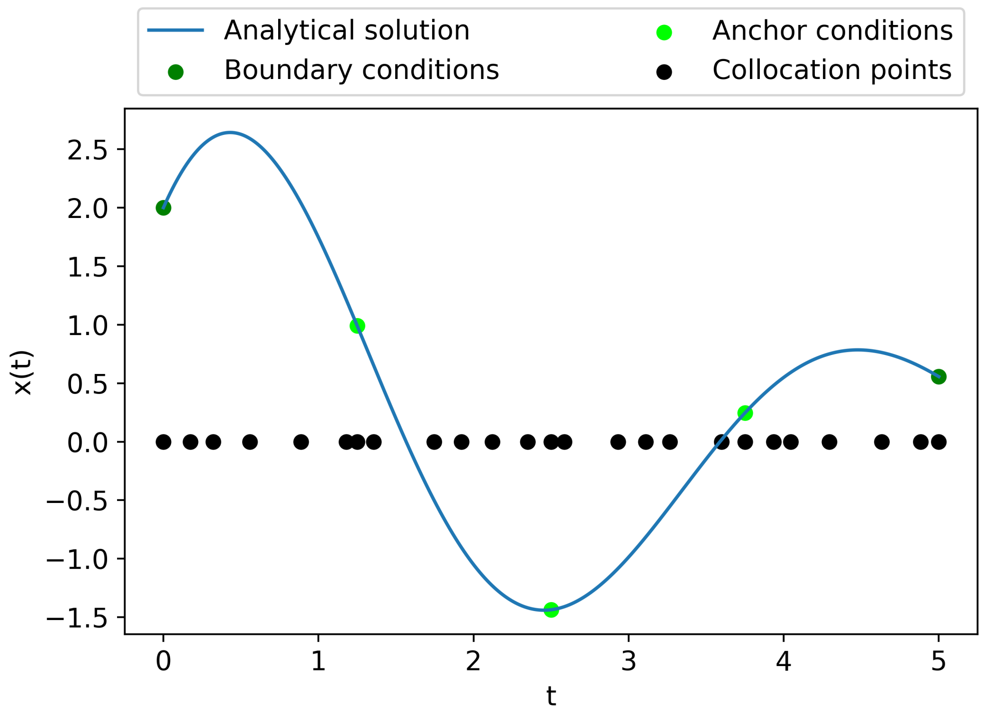

Figure 1: Boundary conditions, anchor conditions, and collocation points of a linear, damped, undriven, harmonic oscillator. The collocation points are chosen using Latin hypercube sampling

of the domain will contain fewer collocation points.

$$x_k = \cos(\frac{\pi k}{n})\frac{ub - lb}{2} + \frac{ub + lb}{2} \tag{2}$$

2.2.2 Identification

Identification, or data-driven discovery of partial differential equations, is close to the opposite of inference. With identification, the goal is to determine one or multiple of the parameters of the differential equation given a dataset and its governing differential equation. For instance in case of a linear, damped, undriven, harmonic oscillator (equation 3) the ratio of its parameters mass m, stiffness k, and friction coefficient r in a specific dataset can be determined via identification. With this specific differential equation it is only possible to find the ratio of the parameters, as the parameters are linearly dependent on one another, so for instance doubling the value of all the parameters does not change the result. It is possible to find the exact values of the parameters if one of the parameters is known, or if one of the derivatives does not have a factor with which it is multiplied.

$$m\frac{d^2x}{dt^2} + r\frac{dx}{dt} + kx = 0 \tag{3}$$

The implementation of identification is very similar to inference, using both collocations points and training data, and applying the same errors. The main difference in the implementation is that in identification the collocation points and training data are used separately. The error over the training data is used to fit the neural network. Opposite to inference, a large amount of training data is required, as the default neural network used in the PINN has to have enough data to provide an accurate prediction. Using this prediction, the error over the collocation points, so the physics-informed part of the neural network, is used to update the parameters of the differential equation to fit the solution found by the PINN based on the training data.

2.2.3 Automatic differentiation

When in this thesis a derivative is computed numerically, automatic differentiation is used. Automatic differentiation is a combination of numerical differentiation, which is relatively simple but can be inaccurate, and symbolic differentiation, which is accurate but can be very complex. The main idea of automatic differentiation is that by combining numerical and symbolic differentiation, the positive aspects of both can be utilized while their negative effects can be minimized (Baydin, Pearlmutter, Radul, & Siskind, 2018).

Automatic differentiation has been around since the sixties (Wengert, 1964), and is for instance used in most back propagation algorithm implementations. Over time automatic differentiation has improved significantly, making it possible to determine complicated differentials within a short time frame. This improvement in automatic differentiation is perhaps the most important factor behind the current capabilities of PINNs, as calculating the error over the PINNs' collocation points requires quickly calculated, accurate derivatives.

There are two types of automatic differentiation: forward mode and reverse mode. In both cases, the equation is first divided into individual components. The numerical value of the components is then calculated and the different components are combined to determine the solution of the equation. What happens next depends on the mode used. In forward mode, the derivatives are symbolically determined, and the corresponding tangent line, which is the line that touches the curve at only one specific point, is incorporated using the chain rule.

As a simple example, take the equation $f(x_1,x_2) = \cos(x_1) + x_1x_2$ with $x_1 = 4$ and $x_2 = 7$, with the goal to determine the first derivative of the equation with regards to x_1. First, the equation is split into its individual components, in this case $c_1 = \cos(x_1)$ and $c_2 = x_1x_2$. The individual components are combined in $c_3 = c_1 + c_2$. Second, the numerical value of these components and the solution of the equation is calculated given the provided values $x_1 = 4$ and $x_2 = 7$. So $c_1 = \cos(4) = -0.65$, and $c_2 = 4 \cdot 7 = 28$. Combining these gives $c_3 = -0.65 + 28 = 27.35$.

Next, the symbolic derivatives of the components are computed with respect to all the different variables. So the symbolic derivative is determined with regards to both x_1 and x_2, even though the derivative with regards to x_2 is not relevant in this problem. Computing the derivative with respect to both variables assures that the method can also be used when the derivative with regards to multiple variables is required. So $\frac{dc_1}{dx_1} = \frac{d\cos(x_1)}{dx_1} = \dot{x}_1 \cdot -\sin(x_1)$, $\frac{dc_1}{dx_1x_2} = \frac{dx_1x_2}{dx_1} + \frac{dx_1x_2}{dx_2} = \dot{x}_1x_2 + \dot{x}_2x_1$, $\frac{dc_3}{dx_1x_2} = c_1 + c_2$.

In this example, $\dot{x}_1$ has a tangent of 1. As the goal is to determine the derivative of the equation with regards to x_1, $\dot{x}_2 = 0$. Inserting $\dot{x}_1$ and $\dot{x}_2$ in the derivatives calculated above results in $\frac{dc_1}{dx_1} = 1 \cdot -\sin(4) = 0.76$ and $\frac{dc_2}{dx_1} = 1 \cdot 7 + 0 \cdot 4 = 7$. Combining the components $\frac{dc_3}{dx_1} = 0.76 + 7 = 7.76$ gives the solution of the first derivative of the equation with regards to x_1 with a tangent of 1.

Reverse mode is more complex, but more efficient if the derivative has to be determined with regards to multiple variables. While forward mode calculates one derivative at a time, in reverse mode the derivative is calculated for all variables at ones. This is done by starting at the end of the first step and setting the value of the final result to zero and traversing through the original calculation to the individual variables. So reverse mode is very similar to forward mode but the traversal through the equation is done, as the name implies, in reverse.

3 Methods

3.1 Multilayer perceptron

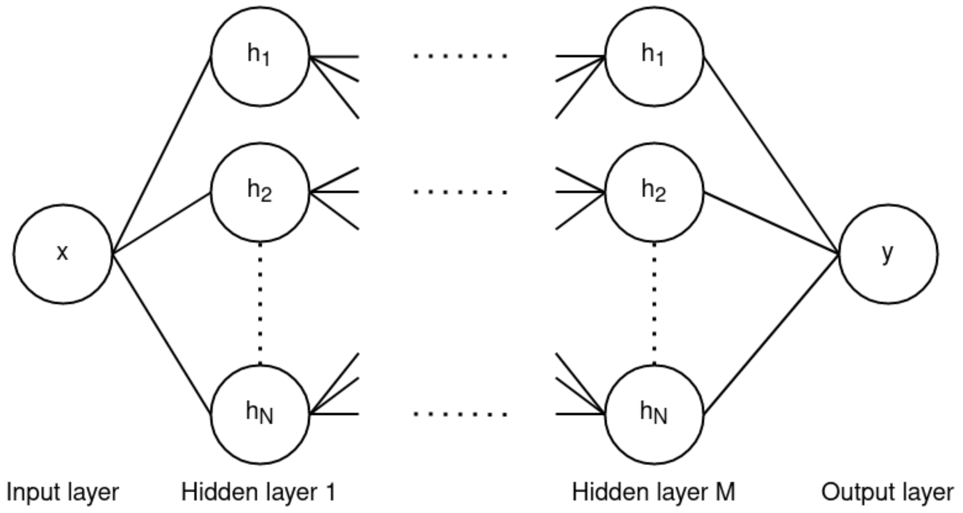

Figure 2: General overview of the multilayer perceptron. It consists of M hidden layers, with N hidden nodes per layer. The displayed multilayer perceptron receives one input, or vector of inputs, x. It returns one output, or vector of outputs, y.

As indicated by the name, the Multilayer Perceptron (MLP) is a feedforward neural network consisting of multiple layers of perceptrons. The perceptron (Rosenblatt, 1958) is a simple binary node which given any input returns a binary value. However, in this thesis, the MLP should be regarded within its more loose interpretation. The network has the same structure as the original MLP, but does not use perceptrons. Instead of binary nodes it uses nodes that apply a linear transformation, as shown in equation 4. In the equation, y is the output to be determined, x_k the input, φ the non-linear activation function, which in this thesis is the hyperbolic tangent, N the amount of inputs, l the layer, w the weight, and b the bias. Every layer is fully connected with the next layer, so each hidden node uses all the hidden nodes of the previous layer as input.

$$y(x_k^l) = \varphi(\sum_{i=1}^{N}(w_i^l x_k^l + b_i^l)) \tag{4}$$

An MLP is trained using iterative learning. Each epoch, the MLP makes a prediction of the training data using its current weights. The error is computed as the MSE or the Mean Absolute Error (MAE) of the difference between the prediction of the MLP and the correct solution given in the training data. Each epoch the gradient of the error, also called the loss, is calculated using backpropagation (Werbos, 1990), which utilizes automatic differentiation. Based on the gradient of the loss, the weights are updated. How the weights are updated, depends on which optimizer is used. In this thesis, the MLP uses the stochastic gradient-based optimization algorithm Adam as its optimizer (Kingma & Ba, 2014).

The MLP continues training until either a predetermined amount of epochs has been completed, or until the MLP does not improve over a certain amount of epochs. In this thesis the MLP is implemented using PyTorch (Paszke et al., 2019).

3.2 Physics-informed multilayer perceptron

The Physics-Informed Multilayer Perceptron (PIMLP) is, as the name implies, based on the MLP, with a general structure very similar to the MLP. The calculation of the error of the PIMLP is split in two parts: the error over the training data and the error over the collocation points.

3.2.1 Training data

The PIMLP calculates the error, or loss, over the training data using the default MLP. Like the MLP, the PIMLP calculates the prediction of the training data using its current weights. The difference between the prediction and the correct solution given in the training data is the error. This error is squared, and then the mean of the errors over the training data is taken. So during training the PIMLP uses the MSE to calculate the loss. This calculation is shown in equation 5 (Raissi et al., 2019), where N_{bc} is the amount of training data provided, $u(t)$ is the value predicted by the MLP, given the variable t, and y is the value given in the training data.

$$MSE_{bc} = \frac{1}{N_{bc}} \sum_{i=1}^{N_{bc}} (u(t_{bc}^{i}) - y^{i})^2 \tag{5}$$

3.2.2 Collocation points

Calculating the error over the collocation points is more complex. First the current prediction is determined by doing a forward pass through the PIMLP given the current weights. Automatic differentiation is then applied to calculate the relevant derivatives. So in case of the linear, damped, undriven, harmonic oscillator (equation 3), the first derivative $\frac{dx}{dt}$ and the second derivative $\frac{d^2x}{dt^2}$ are computed. The prediction and the derivatives are inserted into the differential equation, and the result of the differential equation is calculated. Provided the differential equation equals zero, if necessary the differential equation is rewritten beforehand, the error equals the result of the differential equation calculated using the current prediction. As the MSE is used, the error is squared, and the mean over the different collocation points is used to update the PIMLP. Equation 5 (Raissi et al., 2019) shows how the error of the collocation points is calculated, where $f(t^i)$ is the result of the differential equation in the i^{th} collocation point. The collocation points of the PIMLP are chosen using Latin hypercube sampling.

$$MSE_c = \frac{1}{N_c} \sum_{i=1}^{N_c} (f(t_c^i))^2 \tag{6}$$

3.2.3 Combined error

The combined error of the MLP is calculated by summing the MSE over the boundary conditions and the MSE over the collocation points as shown in equation 7 (Raissi et al., 2019). The combined error is used to update the weights of the PIMLP. Like in the MLP, backpropagation is used to calculate the gradient of the error. The weights of the PIMLP are then updated based on the gradient of the error using the Adam optimizer.

$$MSE = MSE_c + MSE_{bc} \tag{7}$$

Each epoch, the error over the boundary conditions and the error over the collocation points is calculated. Like the MLP, the PIMLP ends when either there is no more improvement in the accuracy of the PIMLP over a set amount of epochs, or when the number of epochs reaches a predetermined limit. Because the PIMLP uses iterative training to reach a solution, the PIMLP is able to solve both linear and non-linear problems.

3.3 Extreme learning machine

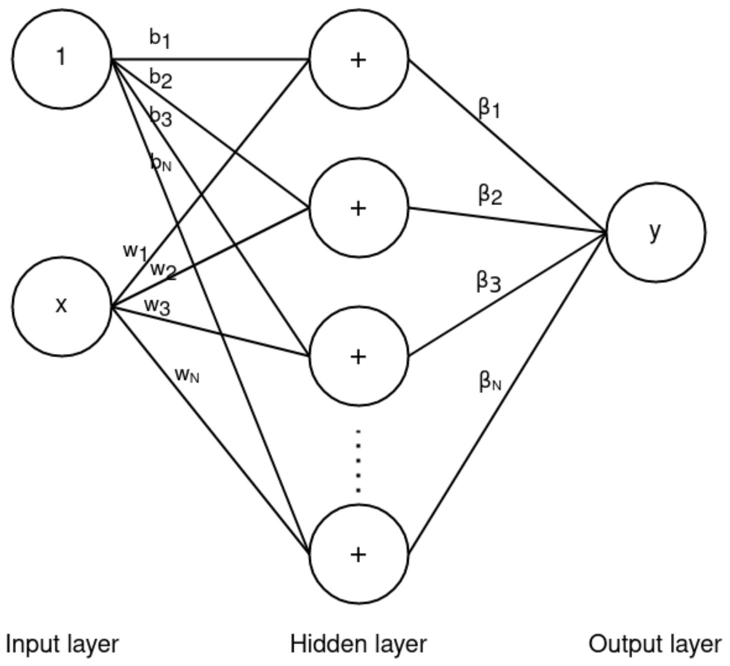

Figure 3: General overview of an extreme learning machine. Consisting of an input x, biases b, hidden weights w, output weights β, and output y

The Extreme Learning Machine (ELM) is a Single-hidden Layer Feedforward neural Network (SLFN), consisting of hidden weights, hidden layer biases, and output weights (Huang, Zhu, & Siew, 2006). A general overview of an ELM with input x and output y is shown in figure 3. What makes the ELM unique compared to other SLFN's, is that while most SLFNs train all the weights and biases, the ELM uses randomly assigned hidden weights and hidden layer biases. Randomly assigning the hidden weights and hidden layer biases decreases the complexity of the problem significantly, as only the output weights have to be trained.

The training phase of the ELM consists of just a few steps. First, the activation of each hidden node is calculated, as shown in equation 8 (Dwivedi & Srinivasan, 2020), where x_k is the input, w a hidden weight, b the corresponding bias, and φ the activation function, which in this thesis is the hyperbolic tangent. The activation is calculated for all N hidden weights and corresponding biases in the ELM.

$$\mathbf{h}(x_k) = [\varphi(w_1x_k + b_1), \varphi(w_2x_k + b_2), ..., \varphi(w_Nx_k + b_N)] \tag{8}$$

The activations of all the K hidden nodes (equation 8) of the ELM are combined in the matrix $\mathbf{H}$, as shown in equation 9 (Dwivedi & Srinivasan, 2020).

$$\mathbf{H} = [\mathbf{h}(x_1), \mathbf{h}(x_2), ..., \mathbf{h}(x_K)]^T \tag{9}$$

The goal of the ELM is to get a system where the activations of the hidden nodes **H**, multiplied by the output weights β, results in the correct result **t**, provided in the training data, as shown in equation 10 (Huang et al., 2006).

$$\mathbf{H}\beta = \mathbf{t} \tag{10}$$

Because the target **t** and the activations of the hidden nodes **H** are known, the output weights β can be calculated by multiplying the Moore-Penrose generalized inverse of the activation of the hidden nodes $\mathbf{H}^{\dagger}$ with the target output **t** (Huang et al., 2006).

$$\beta = \mathbf{H}^{\dagger}\mathbf{t} \tag{11}$$

3.4 Linear physics-informed extreme learning machine

To solve linear differential equations with the ELM, the Linear Physics-Informed Extreme Learning Machine (L-PIELM) is implemented. It is a neural network with the ELM as its base, but extended with a physics-informed component. The physics-informed component is used to process the collocation points given to the ELM during training to assure it fits the differential equation. The training data is treated the same as in an ELM. The physics-informed component is only used during training, so during testing the L-PIELM behaves identical to an ELM.

For every point in the training data, the prediction made by the L-PIELM should be match the solution provided in the training data. So the equation for predicting a point x given target y in the training data, shown in equation 12, is the same as the equation for making a prediction in an ELM. To train the output weights on the training data, like in the ELM, the Moore-Penrose generalized inverse of the activation of the hidden nodes is multiplied by the target output (equation 11).

$$\mathbf{h}(x)\beta = y \tag{12}$$

For the collocation points, the activation of the hidden nodes is computed using the physics-informed component. As no prediction can be made, because there are no output weights during training, the activation of the hidden nodes is used instead of the prediction. In the differential equation, the activation of the hidden nodes (equation 8) is used as the function. For any derivatives in the differential equation, the relevant symbolic derivative of the activation of the hidden nodes is inserted in the differential equation. Equation 13 shows as an example the equation to calculate the activation of the hidden nodes for a first derivative with regards to x_c. This equation is the symbolically derived first derivative of the equation to calculate the activation of the hidden nodes (equation 8). Provided the differential equation is rewritten, so the right side equals zero, the target for each of the collocation points is set to zero. The collocation points of the L-PIELM are chosen using Latin hypercube sampling.

$$\frac{d\mathbf{h}(x_c)}{dx_c} = [(\varphi'(w_1x_c + b_1))w_1, ..., (\varphi'(w_Nx_c + b_N))w_N] \tag{13}$$

As an example of the functioning of the L-PIELM, take the ODE shown in equation 14, with as training data the boundary condition $u(x_{bc_1}) = y_{bc_1}$.

$$\frac{du}{dx} + u^2 = 0 \tag{14}$$

For each collocation point, the activation of the hidden nodes is calculated using the physics-informed component. This is achieved by inserting the activation for the hidden nodes into the differential equation, as shown in equation 15. The function u is replaced by $\mathbf{h}(x_c)$, the activation of the hidden node of the ELM (equation 8). $\frac{d\mathbf{h}(x_c)}{dx_c}$, the symbolic first derivative of the activation of the hidden nodes, shown in equation 13, is inserted in the place of $\frac{du}{dx}$.

$$\mathbf{c}(x_c) = \frac{d\mathbf{h}(x_c)}{dx_c} + \mathbf{h}(x_c)^2 \tag{15}$$

The hidden layer output matrix $\mathbf{H}$ is formed by combining all collocation points and boundary conditions in one array, as shown in equation 16. Where x_{c_k} is the k^{th} collocation point, and x_{bc_k} is the k^{th} boundary condition.

$$\mathbf{H} = [\mathbf{c}(x_{c_1}), \mathbf{c}(x_{c_2}), ..., \mathbf{c}(x_{c_N}), \mathbf{h}(x_{bc_1})]^T \tag{16}$$

The corresponding target $\mathbf{t}$ is shown in equation 17. The target is set to zero for all collocation points, as the ODE in its current form should result in zero. For the boundary condition x_{bc_1}, the target is set to y_{bc_1}, the value provided in the training data.

$$\mathbf{t} = [0, 0, ..., 0, y_{bc_1}] \tag{17}$$

When the matrix containing the activation of the hidden nodes $\mathbf{H}$, and the target $\mathbf{t}$ are known, the output weights can be calculated in the same way as in an ELM. The Moore-Penrose generalized inverse of the hidden layer output matrix $\mathbf{H}$ is multiplied by the target $\mathbf{t}$, resulting in the output weights β, as shown in equation 18.

$$\beta = \mathbf{H}^{\dagger}\mathbf{t} \tag{18}$$

3.5 Non-linear physics-informed extreme learning machine

In case of non-linear problems, the L-PIELM is not applicable, as it calculates the output weights in a single step, without using iterative learning, which means it is unable to solve non-linear problems. Therefore, an iterative implementation of the ELM is used, based on an implementation using constrained expressions (Schiassi et al., 2020). In this thesis, this method will be referred to as the Non-Linear Physics-Informed ELM (NL-PIELM). The NL-PIELM is trained by decreasing the loss over multiple epochs, and uses non-linear least squares regression to update the output weights. Opposite to the other PINNs in this thesis, which all have a loss over the training data and a loss over the collocation points, the NL-PIELM only deals with a single loss. The NL-PIELM fits the training data analytically using constrained expressions, and therefore does not have a loss for the training data. The NL-PIELM's sole objective during training is to give an accurate prediction for the collocation points. The NL-PIELM is trained by minimizing the error over the collocation points, which is calculated using the physics-informed component.

3.5.1 Constrained expressions

Constrained expressions (Leake & Mortari, 2020), (Leake, Johnston, & Mortari, 2020), (Mortari, 2017) are functions that analytically satisfy the training data, ensuring the prediction of the NL-PIELM fits the training data. Hereby reducing the problem from a constrained one into an unconstrained problem. By analytically satisfying the training data, instead of using the neural network to fit the training data, like in the PIMLP, there is no trade-off between the accuracy on the training data and the accuracy on the collocation points, which theoretically leads to a higher accuracy. Also, the NL-PIELM requires less computations, leading to a faster runtime. The downside to analytically satisfying the training data is the same as its main advantage, namely that there is no trade-off between the accuracy in predicting the training data and the accuracy of the prediction over the collocation points. Because when the training data contains noise, the NL-PIELM will still exactly fit the training data, resulting in overfitting and a less accurate prediction. Hence analytically satisfying the training data improves the performance if the NL-PIELM is provided with noiseless data, and hurts the performance if the training data contains noise.

The easiest example of how constrained expressions work, is a first-order ODE with a single constraint, for instance the boundary condition $u(x_{bc_1}) = y_{bc_1}$. This can be transformed to the constrained expression $f(x) = g(x) + y_{bc_1} - g(x_{bc_1})$, where x is the input and $g(x)$ is a free function. In case of the NL-PIELM, the free function $g(x)$ is an ELM. At input x_{bc_1}, the constrained expression is $f(x_{bc_1}) = g(x_{bc_1}) + y_{bc_1} - g(x_{bc_1})$. As $g(x_{bc_1}) - g(x_{bc_1})$ is zero, the result when given the input x_{bc_1} is y_{bc_1}, independent of the prediction of the ELM $g(x)$. Hence any solution given by the NL-PIELM using the constrained expression will always fit the boundary condition.

When dealing with larger amounts of training data, the method to analytically satisfy the training data remains the same, for each combination of variable x and value y in the training data, the expression $y - g(x)$ is added to the constrained expression. There are however two requirements when dealing with larger amounts of training data. Each expression, in the form of $y - g(x)$, has to be linearly independent of all the other expressions within the constrained expression, and the constraints have to be imposed on the constrained expression. How the different expressions are made linearly independent, and the constraints are imposed, depends on the specific amount and type of training data available.

Equation 19 shows the structure to turn any unconstrained expression passing through n points into a constrained expression (Mortari, 2017). With $g(x)$ as the free function, in case of the NL-PIELM an ELM, x as the vector of collocation points, y_k as the dependent value of the k^{th} point in the training data, and x_k as the independent value of the k^{th} point in the training data.

$$f(x) = g(x) + \sum_{k=1}^{n} (y_k - g(x_k)) \prod_{i \neq k} \frac{x - x_i}{x_k - x_i} \tag{19}$$

If instead of n points, the function and the first n derivatives of that function at one point are known, for instance when dealing with initial conditions, the constrained expression shown in equation 20 can be created, where x_0 is the point at which the first n derivatives of the function are known. (Mortari, 2017).

$$f(x) = g(x) + \sum_{k=0}^{n} \frac{(x - x_0)^k}{k!} \left(\frac{d^k y}{dx^k} - \frac{d^k g(x_0)}{dx^k}\right) \tag{20}$$

Furthermore these two types of constrained expressions can be combined into constrained expressions containing multiple derivatives at multiple points. The general structure of the constrained expression is shown in equation 21, where η is a function computed by imposing the constraints (Mortari, 2017).

$$f(x) = g(x) + \sum_{k=1}^{n} \eta_k x^k \tag{21}$$

If the training data contains two boundary conditions, the k^{th} derivative at x_1 and the j^{th} derivative at x_2, the values of the parameters η_1 and η_2 can be calculated as shown in equation 22. If a larger amount of training data is given, the matrix is expanded to encompass all the training data (Mortari, 2017).

$$\begin{Bmatrix} \eta_1 \\ \eta_2 \end{Bmatrix} = \begin{bmatrix} \frac{d^k}{dx^k}x_1 & \frac{d^k}{dx^k}x_1^2 \\ \frac{d^j}{dx^j}x_2 & \frac{d^j}{dx^j}x_2^2 \end{bmatrix}^{-1} \begin{Bmatrix} y_1^{(k)} - g(x_1)^{(k)} \\ y_2^{(j)} - g(x_2)^{(j)} \end{Bmatrix} \tag{22}$$

As an example, take as training data the boundary conditions $\dot{g}(x_1) = \dot{y}_1$ and $g(x_2) = y_2$. The structure of the constrained expression, based on equation 21, is:

$$f(x) = g(x) + \eta_1 x + \eta_2 x^2 \tag{23}$$

The values of the parameters η_1 and η_2 can than be determined via the method shown in 22, with k as 1, as $\dot{g}(x_1) = \dot{y}_1$ is a first derivative, and j as 0, because $g(x_2) = y_2$ is a function.

$$\begin{bmatrix} \eta_1 \\ \eta_2 \end{bmatrix} = \begin{bmatrix} 1 & 2x_1 \\ x_2 & x_2^2 \end{bmatrix}^{-1} \begin{Bmatrix} \dot{y}_1 - \dot{g}(x_1) \\ y_2 - g(x_2) \end{Bmatrix} \tag{24}$$

Calculating the inverse of the matrix results in:

$$\begin{bmatrix} \eta_1 \\ \eta_2 \end{bmatrix} = \begin{bmatrix} \frac{x_2}{x_2 - 2x_1} & \frac{-2x_1}{x_2^2 - 2x_1x_2} \\ \frac{-1}{x_2 - 2x_1} & \frac{1}{x_2^2 - 2x_1x_2} \end{bmatrix} \begin{Bmatrix} \dot{y}_1 - \dot{g}(x_1) \\ y_2 - g(x_2) \end{Bmatrix} \tag{25}$$

Now η_1 and η_2 are known, and the constrained expression can be defined:

$$f(x) = g(x) + \frac{x(x_2 - x)}{x_2 - 2x_1}(\dot{y}_1 - \dot{g}(x_1)) + \frac{x(x - 2x_1)}{x_2^2 - 2x_1x_2}(y_2 - g(x_2)) \tag{26}$$

So for every different combination of training data, a different type of constrained expression is required.

For a PDE the constrained expression can be formed by creating a constrained expressions for one of the variables, and then incorporating that constrained expression in the constrained expression of the next variable. Continuing until all the constrained expressions of the various variables are incorporated. The final constrained expression can then be rewritten to combine all the separate constrained expressions into one constrained expression for all the variables. As an example of the implementation of a constrained expression on a PDE, based on an example in (Leake et al., 2020), take the boundary conditions shown in equation 27.

$$g(0,y) = \cos(y) \quad g(2,y) = 4 \quad g(x,0) = x^2 \quad g(x,1) = \sin(x) + 3 \tag{27}$$

First, the constrained expression for one of the variables, in this case x, is created. A constrained expression analytically satisfying the boundary conditions on x is:

$$g^1(x,y,g(x,y)) = g(x,y) + \cos(y) - g(0,y) + x(4 - g(2,y)) \tag{28}$$

Next, A constrained expression analytically satisfying the boundary conditions on y is created. However, instead of the ELM $g(x,y)$, the constrained expression $g^1(x,y)$ (equation 28) is used as the free function. This results in a constrained expression analytically satisfying the boundary conditions on both x and y, shown in equation 29.

$$f(x,y,g^1(x,y)) = g^1(x,y) + (1-y)(x^2 - g^1(x,0)) + y(\sin(x) + 3 - g^1(x,1)) \tag{29}$$

To combine these two constrained expressions into one, the constrained expression in equation 29 is rewritten, replacing $g^1(x,y)$ for its matching constrained expression. Resulting in a single constrained expression analytically satisfying the boundary conditions on both x and y, shown in equation 30.

$$\begin{aligned} f(x,y,g(x,y)) = g(x,y) + \cos(y) - g(0,y) + x(4 - g(2,y)) + (1-y) \\ (x^2 - g(x,0) + g(0,0) + xg(2,0)) + y(\sin(x) + 3 - g(x,1) + g(0,1) + xg(2,1)) \end{aligned} \tag{30}$$

3.5.2 Collocation points

To calculate the error over the collocation points, the NL-PIELM uses iterative learning, with updates based on the error of the prediction calculated with the current output weights. The loss function, to determine the error, is the differential equation, provided the differential equation has been rewritten to equal zero. The constrained expression is used to calculate the current prediction of the NL-PIELM, and used as the function of the differential equation. For all the derivatives in the differential equation, the elementwise gradient of the current prediction is calculated. The result of the differential equation, which is the loss, can then be calculated, as all components have a numerical value. The weights are updated using non-linear least squares regression. The collocation points of the NL-PIELM in this thesis are selected using Chebyshev points of the second kind.

3.5.3 Non-linear harmonic oscillator

As an example of the practical implementation of the constrained expression in the NL-PIELM, consider the non-linear, damped, undriven, harmonic oscillator in equation 31, with m as the mass, r as the friction coefficient, and k as the stiffness.

$$m\frac{d^2x}{dt^2} + r\frac{dx}{dt} + k\sin(x) = 0 \tag{31}$$

After creating a constrained expression $f(t)$ with any of the methods shown above, which one depending on the provided type and amount of training data, the constrained expression and its derivatives are inserted into the differential equation, as shown in equation 32. The derivatives of the constrained expression $f(t)$ are calculated using automatic differentiation. Specifically, the `egrad` implementation by the TFC library (Leake & Johnston, 2021) is used. This mimics `elementwise gradient` from the Autograd library (Maclaurin, Duvenaud, Johnson, & Townsend, 2015).

$$m\frac{d^2f(t)}{dt^2} + r\frac{df(t)}{dt} + k\sin(f(t)) = 0 \tag{32}$$

The result of this differential equation containing the constrained expression and its derivatives is minimized in multiple epochs using non-linear least squares regression to update the output weights.

3.6 Numerical methods

Three numerical methods are applied to solve the ODEs in this thesis. The Runge-Kutta fourth-order method, the MATLAB `bvp5c` algorithm (MATLAB, 2022), and the SciPy `solve_bvp` algorithm (Virtanen et al., 2020). These numerical methods are included to compare the performance of the PINNs to more traditional methods of solving differential equations, and thereby get an indication of the performance of the PINNs compared to other numerical methods.

3.6.1 Runge-Kutta fourth-order method

The Runge-Kutta fourth-order method is an iterative method developed around 1900 by Carl Runge and Wilhelm Kutta (Runge, 1895) (Kutta, 1901), yet is still a commonly used method to solve differential equations today. The Runge-Kutta fourth-order method solves an ODE beginning at the initial values and calculates the value at every next step iteratively. Hence to solve a ODE using the Runge-Kutta fourth-order method the initial conditions are required. For example, in case of a second-order ODE, the initial value of the function $y(x_0)$ and the first derivative of the initial condition $y'(x_0)$ are required to calculate the particular solution. Opposite all other methods used in this thesis, it is not possible to solve a differential equation with the Runge-Kutta fourth-order method given arbitrary boundary conditions.

The algorithm of the Runge-Kutta fourth-order method for solving a first-order ODE is shown below (Süli & Mayers, 2003). To use the algorithm, a step-size h and an initial value $y(x_0) = y_0$ are required. In the equations below, $f(x, y)$ is the first order ODE, x is the independent variable of the ODE, y is the dependent variable of the ODE, and n the current step.

$$\frac{dy}{dx} = f(x,\, y)$$

$$\begin{aligned}
k_1 &= h \cdot f(x_n,\, y_n) \\
k_2 &= h \cdot f(x_n + \frac{1}{2}h,\, y_n + \frac{1}{2}k_1) \\
k_3 &= h \cdot f(x_n + \frac{1}{2}h,\, y_n + \frac{1}{2}k_2) \\
k_4 &= h \cdot f(x_n + h,\, y_n + k_3)
\end{aligned}$$

$$\begin{aligned}
y_{n+1} &= y_n + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4) \\
x_{n+1} &= x_n + h
\end{aligned}$$

This is an easy yet effective way of solving a first-order ODE. It is even possible to solve it by hand in a short amount of time, if the amount of steps used is limited.

The method becomes more complicated when solving higher order ODEs, although it remains relatively simple. The algorithm for solving a second-order ODE using the Runge-Kutta fourth-order method is shown below (Süli & Mayers, 2003). Again, a step-size h, and an initial value $y(x_0) = y_0$ are required. As the goal is to solve a second-order ODE, also the first derivative of the initial value

$y'(x_0) = u_0$ is required. In the algorithm, $f(x,y)$ is the second-order ODE, x is the independent variable of the ODE, y is the dependent variable of the ODE, and n the current step.

$$\frac{d^2y}{dx^2} = f(x,\ u,\ y)$$

$$\begin{aligned}
m_1 &= h \cdot u_n \\
k_1 &= h \cdot f(x_n,\ u_n,\ y_n) \\
m_2 &= h \cdot (u_n + \frac{1}{2}k_1) \\
k_2 &= h \cdot f(x_n + \frac{1}{2}h,\ u_n + \frac{1}{2}k_1,\ y_n + \frac{1}{2}m_1) \\
m_3 &= h \cdot (u_n + \frac{1}{2}k_2) \\
k_3 &= h \cdot f(x_n + \frac{1}{2}h,\ u_n + \frac{1}{2}k_2,\ y_n + \frac{1}{2}m_2) \\
m_4 &= h \cdot (u_n + k_3) \\
k_4 &= h \cdot f(x_n + h,\ u_n + k_3,\ y_n + m_3)
\end{aligned}$$

$$\begin{aligned}
y_{n+1} &= y_n + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4) \\
u_{n+1} &= u_n + \frac{1}{6}(m_1 + 2m_2 + 2m_3 + m_4) \\
x_{n+1} &= x_n + h
\end{aligned}$$

It is possible to also extend the algorithm to solve higher order ODEs, yet as the highest order ODE used in this project is a second-order ODE, the algorithms for solving higher order ODEs are not relevant for this thesis. The Runge-Kutta fourth-order method does not have a separate method for solving PDEs. It is possible to transform a PDE into a set of ODEs, and solve the separate ODEs via the Runge-kutta fourth-order method.

3.6.2 MATLAB

The `bvp5c` algorithm is a MATLAB algorithm used to numerically solve ODEs. The `bvp5c` algorithm is a fifth-order method to solve boundary value problems. Due to the design of the `bvp5c` algorithm, it is not possible to provide more boundary conditions than required. E.g. in case of a first-order ODE, only one boundary condition can be provided to the algorithm. Unlike the other methods used in this thesis, the `bvp5c` algorithm requires an initial guess of the solution of the differential equation. This initial guess does not influence the prediction made by the `bvp5c` algorithm, but does influence the time `bvp5c` takes to make a prediction, with a more accurate initial guess leading to a shorter runtime, and vice versa.

3.6.3 SciPy

The `solve_bvp` method is a SciPy algorithm designed to solve a first-order system of ODEs, requiring one boundary condition per first-order ODE. It uses a collocation algorithm to achieve this. So similar to PINNs, it utilises collocation points to determine the solution to the differential equation. The main difference is that the collocation method uses polynomials to identify a finite amount of solutions, and then uses the collocation points to determine the correct solution, instead of the deep learning approach used by PINNs (Russell & Shampine, 1972). Like the `bvp5c` algorithm it is not possible to provide more boundary conditions than required.

3.7 Error

The errors of all the predictions made within this thesis are displayed as the relative Mean Absolute Error (rMAE), shown in equation 33. Where $u(x_i)$ is the prediction given input x_i, y_i is the correct solution given x_i, y_{max} is the maximum value of the dependent variable of the analytical solution, and y_{min} is the minimum value of the dependent variable of the analytical solution.

$$\frac{1}{y_{max} - y_{min}} \frac{\sum_{i=1}^{n} |u(x_i) - y_i|}{n} \tag{33}$$

The rMAE is used instead of the MAE to compare the results achieved on the different problems within this thesis more accurately. Due to the large difference in the range, defined as $y_{max} - y_{min}$, of the different problems, using the standard MAE would result in significantly larger errors in problems with a large range, and smaller errors when the problem has a small range.

3.8 Linear harmonic oscillator

To determine whether and to which degree a PINN is an improvement over normal neural networks for solving linear ODEs, the performance of the MLP, the PIMLP, the ELM, and the L-PIELM are compared on a linear problem. The NL-PIELM can also solve linear ODEs, yet is not included because compared to the L-PIELM, it is slower and gives a result with a similar accuracy. The performance of the PINNs is also compared to the performance of the numerical methods mentioned previously: the Runge-Kutta fourth-order method, the MATLAB `bvp5c` algorithm, and the SciPy `solve_bvp` method.

The different methods are compared on the linear, damped, undriven, harmonic oscillator. The differential equation of this harmonic oscillator is shown in equation 34. Where m is the mass in kg, r is the friction coefficient in $\frac{N \cdot s}{m}$, and k is the stiffness in $\frac{N}{m}$.

$$m\frac{d^2x}{dt^2} + r\frac{dx}{dt} + kx = 0 \tag{34}$$

The linear, damped, undriven, harmonic oscillator is used because it is a differential equation with a known, relatively simple analytical solution. The goal on the linear ODE is to determine the difference in performance between the (physics-informed) ELM and (physics-informed) MLP. Hence it is not relevant to determine the current limits of a PINN, or solve a previously unsolved differential equation. Given that the performance of the PINNs and neural networks should not significantly change with more complicated problems, provided the size of the network and amount of epochs used is adapted accordingly, it is not relevant to use a more complicated ODE, or an ODE with a more complicated analytical solution. With numerical methods there is no possibility to increase the size of the network to compensate for a more complicated problem, so they may perform relatively better or worse based on the difficulty of the problem.

3.8.1 Analytical solution

To determine the error of the predictions, the exact particular solution of the differential equation is computed using the analytical solution. Apart from the parameters m, r, and k, the initial values of the position x and the velocity v, the first derivative of x, are required to determine the analytical solution.

The analytical solution can be calculated in a few steps. These steps can also be rewritten into one large equation, yet to ensure the readability of the equations, they are shown here in a few steps.

First the damping coefficient λ, shown in equation 35 and angular frequency ω, shown in equation 36, are calculated using the parameters present in the differential equation.

$$\lambda = \frac{r}{2m} \tag{35}$$

$$\omega = \sqrt{\frac{k}{m}} \tag{36}$$

The damping coefficient and the angular frequency are used to calculate ω_r. ω_r determines the oscillatory behaviour of the oscillator. If ω_r is larger than 0, the oscillator is underdamped, meaning it moves towards 0 over multiple oscillations. If ω_r equals 0, the oscillator is critically damped, and it returns to 0 quickly without any oscillations. If ω_r is larger than 0, the oscillator is overdamped, meaning it also returns to 0 without oscillating, yet takes significantly longer to reach zero than in case of critical damping. With the parameters used in this thesis, ω_r is approximately 1.55, so the oscillator is underdamped.

$$\omega_r = \sqrt{\omega^2 - \lambda^2} \tag{37}$$

The phase φ of the solution depends on the initial position x_0, the initial velocity v_0, the damping coefficient λ, and the damping of the system ω_r. The exact calculation to determine the phase is shown in equation 38.

$$\varphi = \tan^{-1}\left(\frac{\frac{v_0}{x_0} + \lambda}{-\omega_r}\right) \tag{38}$$

$$A = \frac{x_0}{\cos(\varphi)} \tag{39}$$

Combining these different elements, as shown in equation 40, gives the analytical solution which returns the exact position of the oscillator at any moment in time.

$$x(t) = Ae^{-\lambda t}\cos(\omega_r t + \varphi) \tag{40}$$

Figure 4 shows the analytical solution of the linear harmonic oscillator given an initial displacement of 2 m, an initial velocity of 3 $\frac{m}{s}$, a mass of 2 kg, a stiffness of 5 $\frac{N}{m}$, and a friction coefficient of 1.2 $\frac{N\cdot s}{m}$.

3.8.2 Hyperparameters

MLP The MLP used to solve the linear harmonic oscillator consists of three layers of 40 hidden nodes per layer, with in between each layer a non-linear hyperbolic tangent activation function. The

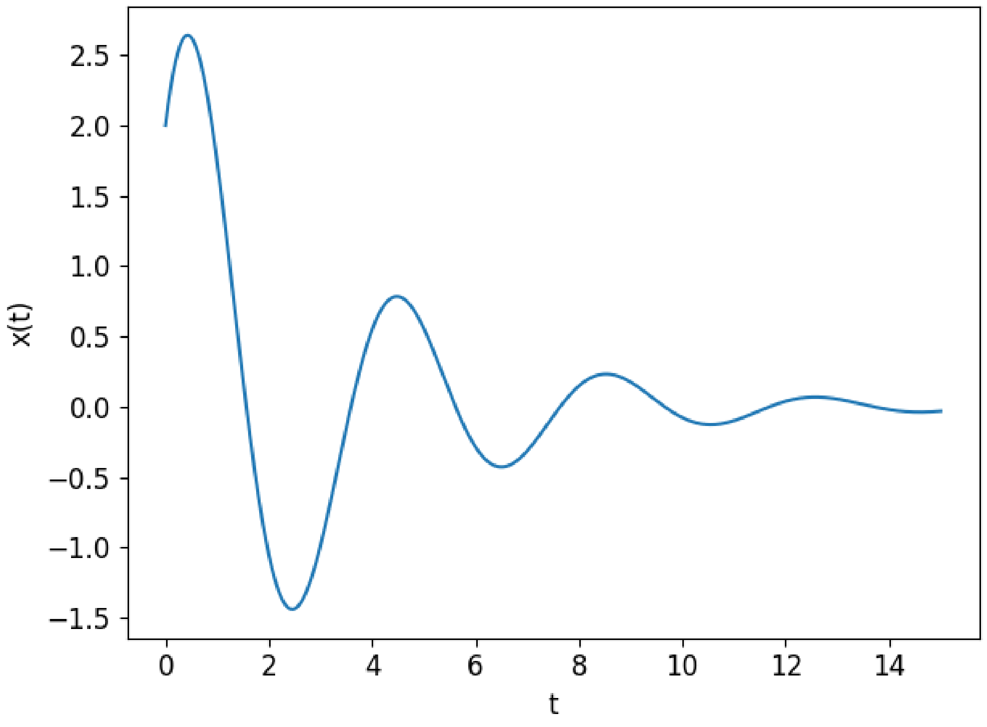

Figure 4: The analytical solution of the linear, damped, undriven, harmonic oscillator with an initial displacement of 2 m, an initial velocity of 3 $\frac{m}{s}$, a mass of 2 kg, a stiffness of 5 $\frac{N}{m}$, and a friction coefficient of 1.2 $\frac{N \cdot s}{m}$

MLP uses the Adam optimizer with a dynamic learning rate starting at 0.1. If there is no improvement for 1000 epochs, the learning rate is reduced ten-fold. The threshold for improvement is set to 10^{-12}, so every improvement smaller than 10^{-12} is not regarded as an improvement. The MLP has a maximum amount of 250000 epochs. However, if there is no improvement over a period of 5000 epochs, the MLP will also stop.

PIMLP To keep the comparison as fair as possible, the PIMLP has the same hyperparameters as the MLP. The only difference is the inclusion of the physics-informed component, for which the PIMLP has 2000 collocation points, which are selected using Latin hypercube sampling.

ELM The ELM used to solve the linear harmonic oscillator has 2000 hidden weights. This is a larger amount than required to solve the problem using an ELM, but is used to give the ELM the same hyperparameters as the L-PIELM. Because the training data is an accurate, noiseless representation of the analytical solution, there is no risk of overfitting, and the performance of the ELM does not suffer because of the excessive amount of weights. The ELM used to determine the runtime is given a smaller amount of weights, the specific amount determined via hyperparameter tuning, as using an excessive amount of weights to determine the runtime would lead to a result not representative of the true runtime of the ELM. The ELM uses the hyperbolic tangent as activation function.

L-PIELM The L-PIELM also has 2000 hidden weights, and it uses the hyperbolic tangent activation function. Like the PIMLP, the L-PIELM has 2000 collocation points, which are selected using Latin hypercube sampling.

3.9 Runtime

In this thesis, the runtime is the time it takes to, if necessary, train the model and make a prediction. The runtime is recorded with the least amount of training data for which a network achieves an accurate result. Based on the errors achieved on the linear harmonic oscillator, shown lateron in the results section in figure 7, an accurate result is regarded as an error below 10^{-4}. This means that for the L-PIELM, the runtime is recorded given two boundary conditions. The PIMLP has two boundary conditions and one anchor condition. The runtime of the MLP is measured when given two boundary conditions and 173 anchor conditions. The ELM is given two boundary conditions and 148 anchor conditions. The numerical methods all have two boundary conditions as training data.

3.10 Anchor conditions

To determine the ability of the neural networks to make a prediction outside the domain of the training data, the performance of the ELM, MLP, PIMLP, and PIELM is compared when given anchor conditions instead of (one of the) boundary conditions. The PINNs and neural networks are compared in a situation with two anchor conditions in the middle of the domain, a situation with one boundary condition and one anchor condition in the middle of the domain, and a situation with one boundary condition and 200 anchor conditions in between the boundary condition and the middle of the domain. The performance is measured on the linear harmonic oscillator with the parameters of the previous section. All the different networks are given the same hyperparameters as in the previous section.

It is not possible to compare the performance of the (physics-informed) neural networks to the performance of numerical methods with regards to anchor conditions. As all numerical methods used in this thesis are not capable of predicting the particular solution of a differential equation on the basis of a set of anchor conditions. All the numerical methods require either the initial conditions or boundary conditions to make a prediction. No other numerical methods were found that are able to solve a problem provided anchor conditions. This does not mean that there are no numerical methods capable of solving these problems. But it does indicate common methods are not able to solve a problem when given anchor conditions.

3.11 Noise

The PINNs and neural networks are also tested on the linear harmonic oscillator when provided with noisy training data. The linear harmonic oscillator used as the differential equation has the same parameters as before. The noise added to the training data is a pseudorandom sample drawn from a normal distribution. The normal distribution from where the noise is drawn has a mean of 0, and a standard deviation of 1. The particular solution of the linear harmonic oscillator to which the noise is added has a range of [-1.44, 2.64]. So the amount of noise added is relatively large. All the different networks use the same seed for the pseudorandomization of the noise, so all the networks receive identical noisy training data. The networks are tested on two different scenarios. In the first scenario, they are provided with two noisy training data, and in the second scenario they are provided with 50 noisy training data. The numerical methods are not tested on noisy training data, as they can only deal with a limited amount of training data, and are not able to make a prediction when given 50 training data.

3.11.1 Hyperparameters

MLP When provided with two noisy training data, the MLP is provided with the same hyperparameters as the MLP used on noiseless training data. So the MLP consists of three layers of 40 hidden nodes per layer, with in between each layer a non-linear hyperbolic tangent activation function. The MLP uses the Adam optimizer with a dynamic learning rate starting at 0.1.

When given 50 noisy training data, the amount of weights used is reduced drastically. The MLP used has two hidden layers consisting of two hidden weights per layer. This small amount of weights is selected because any increase in the amount of weights leads to an increase in the overfitting of the MLP and a decrease in the accuracy of the prediction. Other often effective measures against overfitting, like adding dropout layers and decreasing the amount of epochs, have been tested, yet do not improve the accuracy of the prediction on this occasion.

PIMLP When provided with two noisy training data, the PIMLP has the same hyperparameters as the MLP. Like with noiseless data, the PIMLP has 2000 collocation points, which are selected using Latin hypercube sampling.
The PIMLP has the same hyperparameters when provided with 50 noisy training data.

ELM When two noisy training data are provided, the ELM has the same hyperparameters as with noiseless data. So it has 2000 hidden weights and the hyperbolic tangent as activation function.
When given 50 noisy training data, the ELM is in a similar bind as the MLP. Like the MLP, anything but a small amount of weights leads to overfitting. Therefore the ELM has just 7 hidden weights. The specific amount of weights was determined using hyperparameter tuning.

L-PIELM The L-PIELM has the same hyperparameters as on noiseless training data in both scenarios. So the L-PIELM has 2000 hidden weights, uses the hyperbolic tangent activation function, and has 2000 collocation points which are selected using Latin hypercube sampling.

3.12 Logistic equation

The logistic equation is a non-linear first-order ODE. It is a non-linear function, with part of the solution being exponential, and is bound by a maximum capacity. The logistic equation is used in many different fields, for instance in biology, where it can be used to model the size of a population over time (Pearl & Slobodkin, 1976), or in medicine, where it can be used to model the spread of a viral illness within a population over time (Pelinovsky, Kurkin, Kurkina, Kokoulina, & Epifanova, 2020). The logistic equation has as parameters its growth rate r and its capacity k. The logistic equation is displayed in equation 41.

$$\frac{dP}{dt} = rP\left(1 - \frac{P}{k}\right) \tag{41}$$

A function very similar to the logistic equation is the logistic map. The only difference between the logistic map, shown in equation 42, and the logistic equation, shown in equation 41, is that the logistic map describes the next step instead of the derivative. This small difference has large implications. The most notable consequence is that the logistic map gives chaotic solutions when the value of r is roughly between 3.5 and 4. The logistic equation on the other hand does not become chaotic for any value of r.

$$P_{n+1} = rP_n\left(1 - \frac{P_n}{k}\right) \tag{42}$$

The performance of the MLP, PIMLP, ELM, and NL-PIELM are compared on the logistic equation. The NL-PIELM is used instead of the L-PIELM as it is a non-linear problem, which the L-PIELM cannot solve. The predictions of the neural networks are also compared against the predictions of the Runge-Kutta fourth-order method, the MATLAB `bvp5c` method, and the SciPy `solve_bvp` function.

3.12.1 Analytical solution

Three parameters are required to calculate the analytical solution of the logistic equation. The initial value p_0, the growth rate r, and the carrying capacity k. The analytical solution to the logistic equation is shown in equation 43. p_0, the initial value of P at $t = 0$, is set to 5. The growth rate r is set to 0.028. The value of the carrying capacity k is 943.

$$P(t) = \frac{k}{1 + \frac{k-p_0}{p_0}e^{-rt}} \tag{43}$$

Figure 5 shows the analytical solution of the logistic equation with the given parameters.

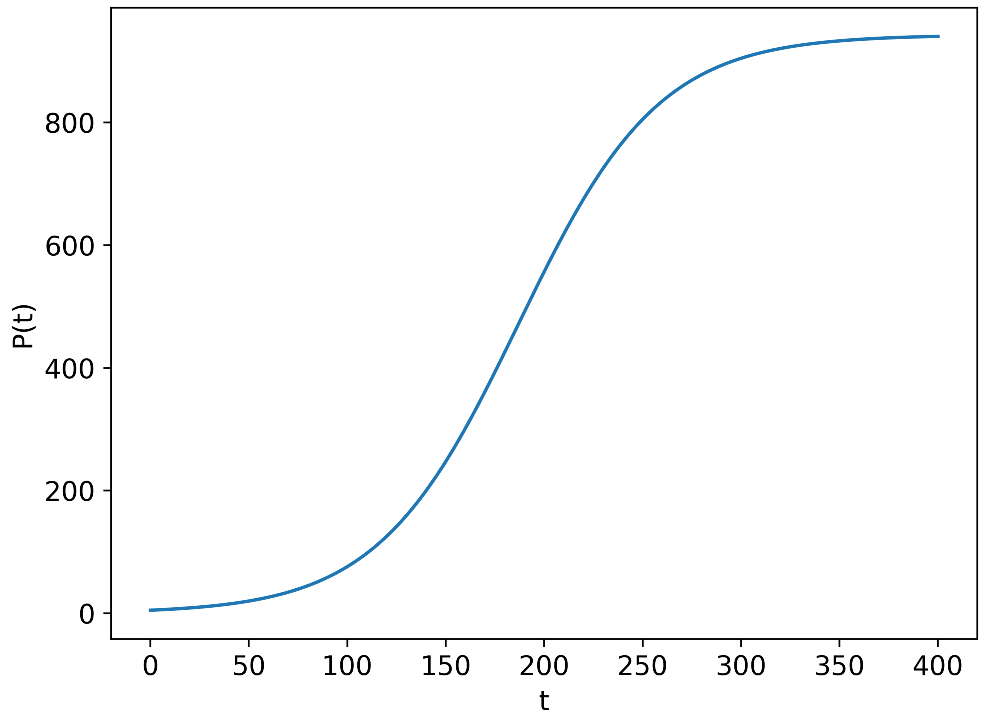

Figure 5: The plot of the analytical solution of the logistic equation, with a growth rate r of 0.028, carrying capacity k of 943, and an initial value of 5 at p_0

3.12.2 Hyperparameters

MLP The MLP used for the logistic equation has four layers with 20 nodes per layer and the hyperbolic tangent as activation function. The MLP uses the Adam optimizer, with a dynamic learning rate starting at 0.1. The MLP has a maximum of 250000 epochs, but stops early if there is no improvement after a period of 5000 epochs.

PIMLP The PIMLP used for the logistic equation has identical hyperparameters as the MLP. The PIMLP has 2000 collocation points, which are selected using Latin hypercube sampling.

ELM The ELM used to solve the logistic equation has 30000 hidden weights. This ELM has such a large amount of weights because hyperparameter tuning shows that giving the ELM this large amount of weights gives the best result, while a more standard amount of weights results in significantly worse predictions. The ELM uses the hyperbolic tangent activation function.

NL-PIELM The NL-PIELM used to solve the logistic equation consists of 1400 hidden weights, significantly less than the 30000 of the ELM. The NL-PIELM has been given 1400 instead of 30000 weights, as running it with 30000 weights would make it unreasonably slow, while simultaneously leading to a worse result. The NL-PIELM has 2000 collocation points, selected according to the distribution of Chebyshev points of the second kind.

As the boundary condition and anchor conditions used are n points taken from the analytical solution, the equation to create a constrained expression given n points, shown in equation 19, and for readability repeated here in equation 44, is used to create the constrained expression. Where n is the amount of training data given.

$$f(x) = g(x) + \sum_{k=1}^{n} (y_k - g(x_k)) \prod_{i \neq k} \frac{x - x_i}{x_k - x_i} \tag{44}$$

3.13 Partial differential equation

The performance of the PIELM and the PIMLP is also tested on a PDE. PDEs are generally more difficult to solve than ODEs, as PDEs contain multiple variables as opposed to the single variable of ODEs. PINNs should theoretically achieve similar results whether predicting PDEs or ODEs, provided that the hyperparameters of the PINNs are adapted accordingly. The PDE used in this thesis is not solved numerically, as no numerical method could be found that is capable of solving a PDE. For some specific types of PDEs it is possible to use a numerical approach, like the Crank–Nicolson method for heat conduction problems (Crank & Nicolson, 1947), yet there are no approaches applicable to the PDE used in this thesis. Other methods to solve a PDE numerically use a 2-step approach, where the PDE is first simplified to a system of ODEs, via for instance the method of lines (Schiesser, 2012), and can subsequently be solved as a set of ODEs.

The PDE used in this thesis is taken from a paper in which multiple ODEs and PDEs are defined and solved (Lagaris, Likas, & Fotiadis, 1998). The PDE was devised purely to test the performance of a method for solving differential equations, and does not necessarily describe any natural phenomenon. This PDE is used, instead of a PDE describing a natural phenomenon, because most PDEs describing natural phenomena do not have a known analytical solution, while the analytical solutions that are known are often very extensive. Given that the sole goal is to determine the performance of different methods on solving PDEs, it should not pose a problem that the PDE does not describe a natural phenomenon. The second-order PDE used in this thesis is shown in equation 45.

$$\frac{\partial^2\psi}{\partial x^2}+\frac{\partial^2\psi}{\partial y^2}+\frac{\partial\psi}{\partial y}\psi=\sin(\pi x)(2-\pi^2y^2+2y^3\sin(\pi x)) \tag{45}$$

As there is a second-order derivative of both variables in the PDE, two boundary conditions, or anchor conditions, are required per variable. In this case, the following boundary conditions are used: $\psi(0,y)=0$, $\psi(1,y)=0$, $\psi(x,0)=0$, and $\frac{\partial\psi}{\partial y}(x,1)=2\sin(\pi x)$. These boundary conditions are chosen because these are the four suggested in the paper which devised this PDE.

3.13.1 Analytical solution

The analytical solution of the PDE is shown in equation 46 (Lagaris et al., 1998) , and plotted in figure 6.

$$\psi(x,y)=y^2\sin(\pi x) \tag{46}$$

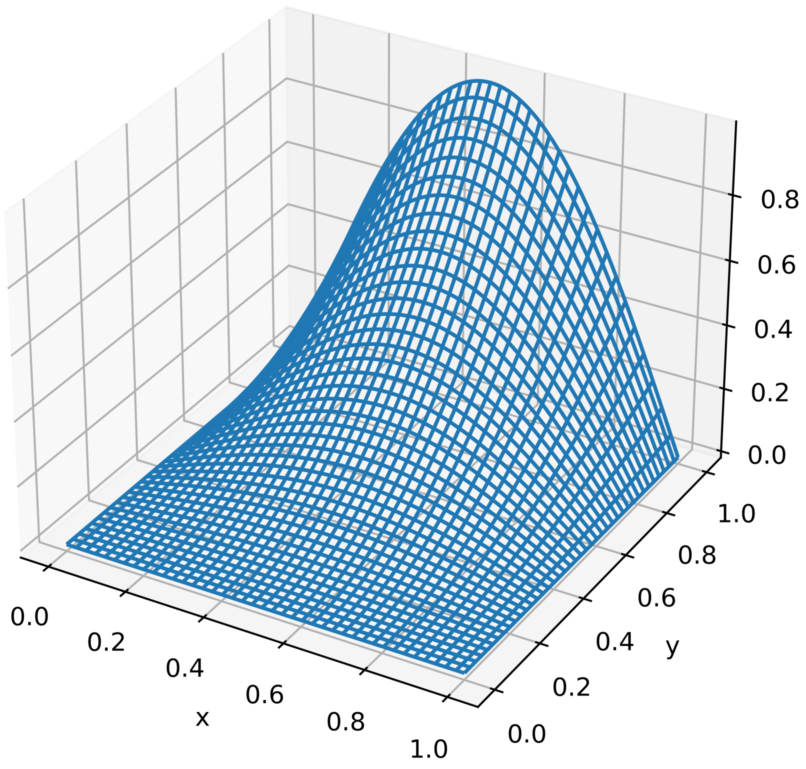

Figure 6: The plot of the analytical solution, $\psi(x,y)=y^2\sin(\pi x)$, of the partial differential equation $\frac{\partial^2\psi}{\partial x^2}+\frac{\partial^2\psi}{\partial y^2}+\frac{\partial\psi}{\partial y}\psi=\sin(\pi x)(2-\pi^2y^2+2y^3\sin(\pi x))$

PIMLP The structure of the PIMLP used to solve the PDE is based on the structure Raissi used to solve the Burgers equation. (Raissi et al., 2019). So it has nine layers consisting of 20 neurons per layer. The PIMLP uses 2000 collocation points, selected via Latin hypercube sampling, and is provided 30 boundary data per edge, so 120 boundary data in total. The PIMLP uses the Adam optimizer, and has a dynamic learning rate starting at 0.01.

NL-PIELM The NL-PIELM has 15 hidden weights. This small amount of weights is used because hyperparameter tuning showed this to be the ideal number of weights. The NL-PIELM uses the hyperbolic tangent activation function. The collocation points for each variable are selected following the distribution of Chebyshev points of the second kind.

The constrained expressions used to solve the NL-PIELM for the boundary conditions noted above, are shown in equation 47 and equation 48. The first constrained expression (equation 47) enforces the boundary conditions on y, and is inserted in the second constrained expression (equation 48), which enforces the boundary conditions on x.
The two constrained expressions can be rewritten into one constrained expression enforcing both boundary conditions, as shown in equation 49.

$$g^1(x,y,g(x,y)) = g(x,y) - (2y-y^2)g(x,0) + \frac{y^2}{2}(2\sin(\pi x) - \frac{\partial}{\partial y}g(x,1)) \tag{47}$$

$$f(x,y,g^1(x,y)) = g^1(x,y) - (1-x)g^1(0,y) - xg^1(1,y) \tag{48}$$

$$\begin{aligned} f(x,y,g(x,y)) = g(x,y) - (2y-y^2)g(x,0) + \frac{y^2}{2}(2\sin(\pi x) - \frac{\partial}{\partial y}g(x,1)) - (1-x) \\ (g(0,y) - (2y-y^2)g(0,0) - \frac{y^2}{2}\frac{\partial}{\partial y}g(0,1)) + x(g(1,y) - (2y-y^2)g(1,0) - \frac{y^2}{2}\frac{\partial}{\partial y}g(1,1)) \end{aligned} \tag{49}$$

Because of the structure of the constrained expression, the amount of boundary conditions cannot be set manually, but is dependent on the amount of the collocation points. As the collocation points are chosen using Chebyshev points of the second kind, the amount of boundary conditions is the (rounded up) square root of the amount of collocation points.
This means that as 25 boundary conditions per edge are selected, 625 collocation points are used. This amount of collocation points is significantly less than used in the PIMLP, yet similar to the number of collocation points used in comparable problems by the authors of the paper on the constrained expressions (Leake et al., 2020). Hyperparameter tuning shows that the combination of 25 boundary conditions and 625 collocation points indeed gives the optimal result. The significant difference in the amount of collocation points used in the PIMLP and the NL-PIELM is likely to influence the result. However, because of differences in the methods of the two networks, the PIMLP requires a large amount of collocation points, and the NL-PIELM requires a relatively small number. Although giving the two networks different amounts of collocation points is not ideal, in this case it is required to get the fairest comparison.

4 Results

4.1 Linear harmonic oscillator

4.1.1 Amount of training data

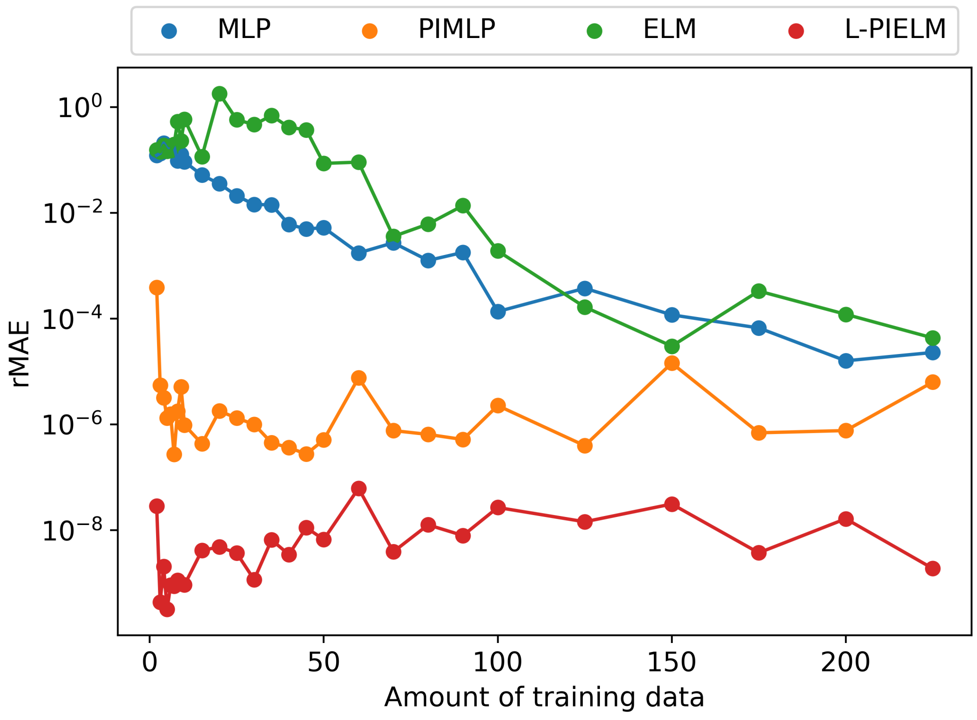

Figure 7: The rMAE of the MLP, the PIMLP, the ELM, and the L-PIELM in predicting the linear harmonic oscillator, plotted against the amount of training data provided to the network. The training data consists of two boundary conditions, extended with linearly spaced anchor conditions.

Figure 7 shows the performance of the ELM, L-PIELM, MLP, and PIMLP in predicting the particular solution of the linear harmonic oscillator.

The ELM and the MLP show a similar result. Both networks demonstrate a clear relationship between the amount of training data that is available and the accuracy of their predictions. The ELM and the MLP have a relatively high error when a small amount of training data is provided, and the error decreases as the amount of training data increases. At some point they have enough training data to make an accurate prediction, and their accuracy no longer improves despite the amount of training data increasing. In case of the linear harmonic oscillator, the performance of the MLP and ELM stabilizes around 150 training data. Increasing the amount of training data to more than 150, does not improve the result. Overall, the MLP achieves a lower error and achieves higher accuracies than the ELM on this problem, especially when a low amount of training data is available.

With low amounts of training data, the PIMLP achieves a significantly higher accuracy than the ELM and MLP. The performance of the PIMLP is largely independent of the amount of training data provided, as unlike the ELM and the MLP, the PIMLP's performance does not improve when the amount of training data increases. The exception to this rule is when the PIMLP has a very small amount of training data, as the performance of the PIMLP does improve significantly when the amount of training data is increased from two to three points.

The L-PIELM achieves the best result of the networks tested on this problem, with accuracies between 10^{-7} and 10^{-10}. The general trend of the errors of the L-PIELM is similar to the shape of the errors of the PIMLP. Again there is no significant change in the performance when the amount of training data is increased, with the exception when the amount of training data is increased from two to three points.

4.1.2 Numerical methods

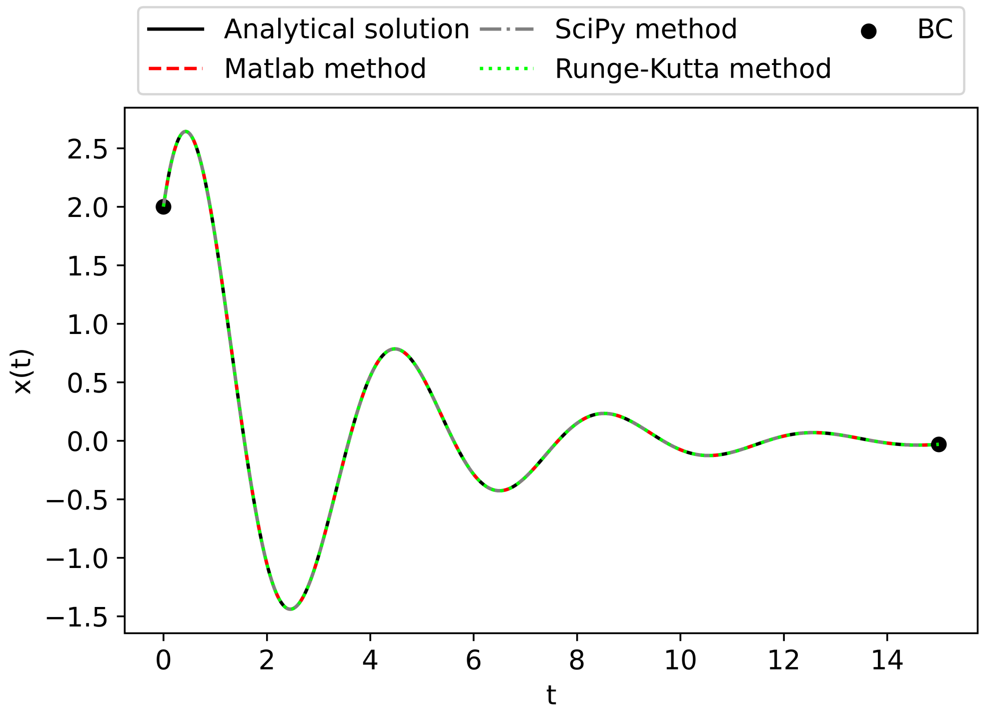

Figure 8: The prediction of the SciPy `solve_bvp` algorithm, the MATLAB `bvp5c` function, and the Runge-Kutta fourth-order method of the linear harmonic oscillator with a mass of 2 kg, a stiffness of 5 $\frac{N}{m}$, and a friction coefficient of 1.2 $\frac{N \cdot s}{m}$, given two boundary conditions.

All numerical methods are able to give an accurate prediction of the particular solution of the linear harmonic oscillator. As can be seen in figure 8, in which there are no visible discrepancies between the predictions of the numerical methods and the analytical solution. The Runge-Kutta fourth-order method has the relatively worst result with an rMAE of $8.09 \cdot 10^{-5}$. The MATLAB `bvp5c` algorithm's prediction achieves an rMAE of $6.52 \cdot 10^{-9}$. The SciPy `solve_bvp` algorithm makes the most accurate prediction, with an rMAE of $2.26 \cdot 10^{-13}$.

4.1.3 Runtime

Table 1: The runtimes of the different methods used to approximate the particular solution of the linear harmonic oscillator, given the lowest amount of training data at which they achieve an error smaller than 10^{-4}.

Method	Runtime (seconds)
MLP	36.12
PIMLP	65.98
ELM	0.42
L-PIELM	1.12
Runge-Kutta	0.01
MATLAB	2.69
SciPy	0.04

The results in table 1 show that the PIMLP is by far the slowest method, followed by the MLP. All other methods are significantly faster. The slowest method after the MLP, the MATLAB `bvp5c` algorithm, is more than 10 times as fast as the MLP. The L-PIELM takes just over a second to train and run, while the ELM requires about a third of that time. By far the fastest methods are the SciPy `solve_bvp` method and the Runge-Kutta method, taking only a fraction of a second to make a prediction.

4.2 Anchor conditions

4.2.1 Two anchor conditions

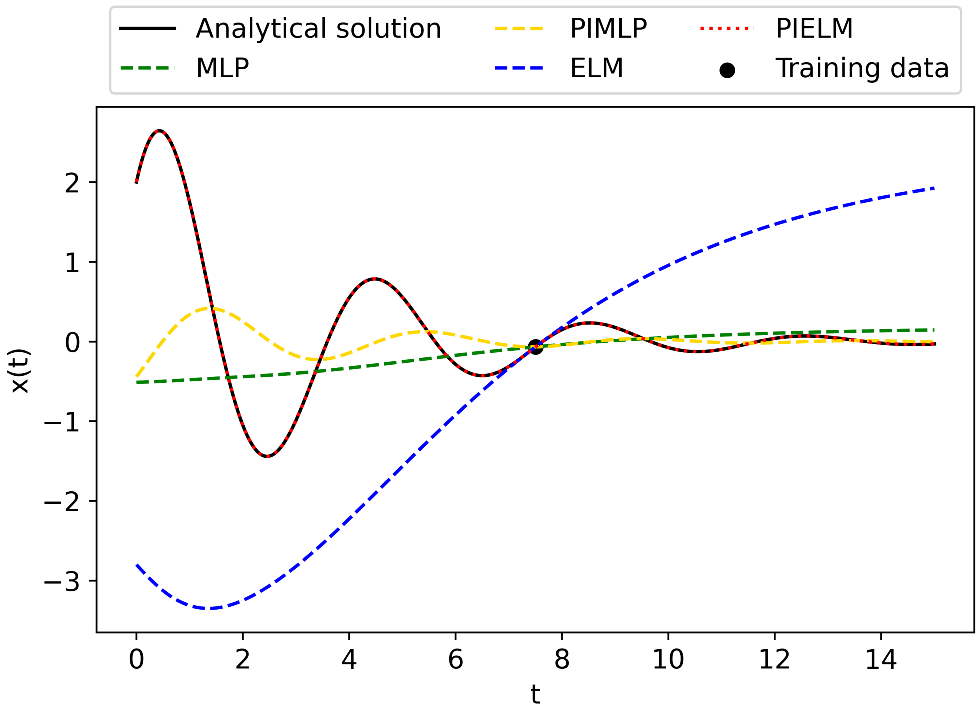

Figure 9: The prediction of the MLP, PIMLP, ELM, and L-PIELM of the particular solution of the linear harmonic oscillator with a mass of 2 kg, a stiffness of 5 $\frac{N}{m}$, and a friction coefficient of 1.2 $\frac{N \cdot s}{m}$, given two anchor conditions in the center of the solution.

Figure 9 shows the prediction of the MLP, PIMLP, ELM, and L-PIELM when they are given two anchor conditions in the center of the domain of the analytical solution as training data. The L-PIELM is still capable of making a prediction with a high accuracy and low error. It does not appear to suffer from replacing the boundary conditions for central anchor conditions. The L-PIELM achieves an error of $2.28 \cdot 10^{-8}$, slightly lower, yet comparable to the L-PIELM's performance with two boundary conditions, which was shown in figure 7.

The PIMLP manages to get a result which resembles the shape of a linear harmonic oscillator, as it contains oscillations with a decreasing amplitude. The prediction given is however inaccurate and has a large error.

The ELM and MLP both give a prediction that does not resemble the shape of a linear harmonic oscillator, nor is it similar to the particular solution.

4.2.2 One boundary condition and one anchor condition

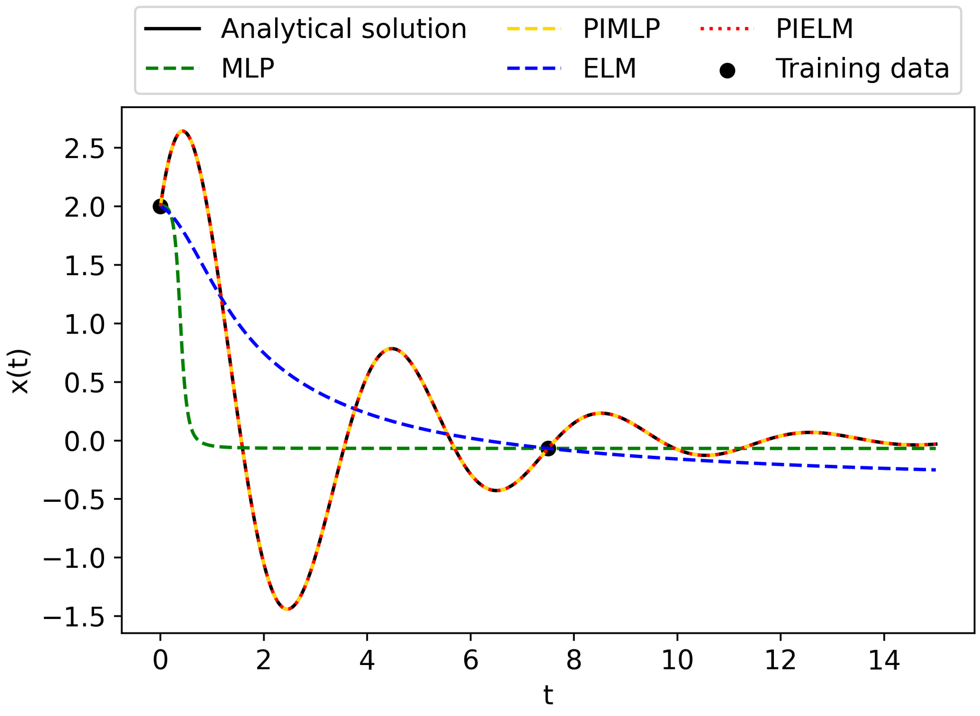

Figure 10: The prediction of the MLP, PIMLP, ELM, and L-PIELM of the particular solution of the linear harmonic oscillator with a mass of 2 kg, a stiffness of 5 $\frac{N}{m}$, and a friction coefficient of 1.2 $\frac{N \cdot s}{m}$. Given one boundary condition at the left edge of the domain, and one anchor condition in the center of the domain.

The predictions of the MLP, PIMLP, ELM, and L-PIELM when provided with one boundary condition and one anchor condition, respectively on the left edge and in the center of the domain, are shown in figure 10.

In line with the result when given two anchor conditions (figure 9), the L-PIELM manages to achieve an accurate prediction. The L-PIELM's prediction has an rMAE of $4.56 \cdot 10^{-9}$. So the L-PIELM's error is similar, but slightly smaller than the error achieved when given two anchor conditions.

The PIMLP also manages to make an accurate prediction when given one boundary condition and one anchor condition. The PIMLP achieves an rMAE of $6.71 \cdot 10^{-7}$. Surprisingly, this result is slightly better than its performance when it is given two boundary conditions, which was shown in figure 7.

The ELM and MLP again both make a prediction that fits the boundary condition and the anchor condition, but with a shape that does not resemble a linear harmonic oscillator. Both the ELM and MLP give inaccurate predictions with a high rMAE of respectively $1.67 \cdot 10^{-1}$ and $1.55 \cdot 10^{-1}$.

4.2.3 One boundary condition and 200 anchor conditions

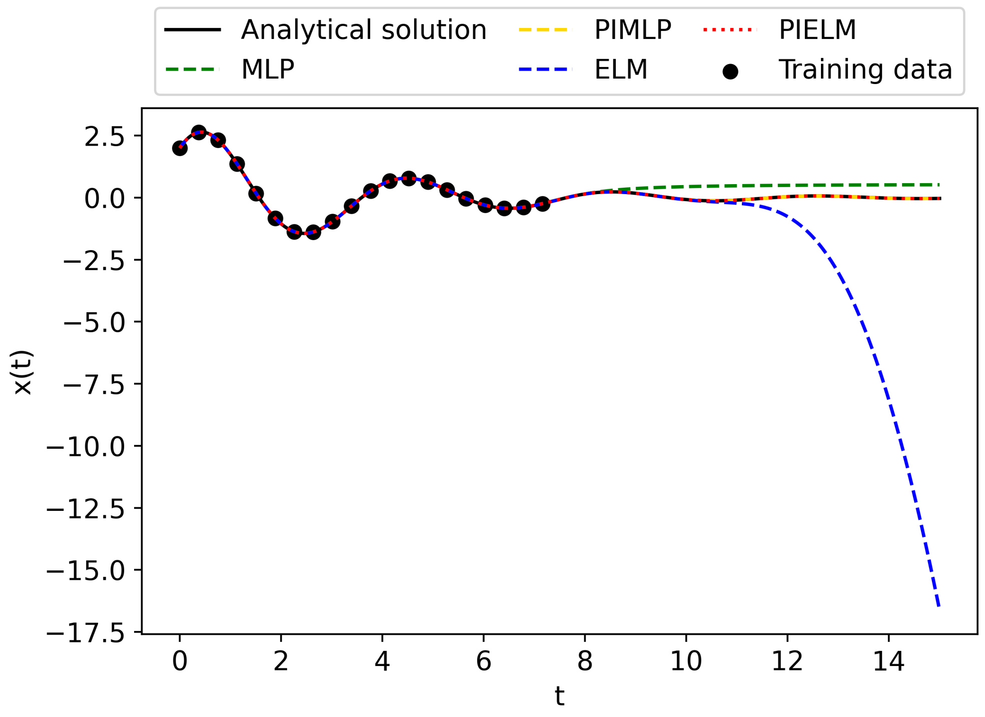

Figure 11: The prediction of the MLP, PIMLP, ELM, and L-PIELM of the particular solution of the linear harmonic oscillator with a mass of 2 kg, a stiffness of 5 $\frac{N}{m}$, and a friction coefficient of 1.2 $\frac{N \cdot s}{m}$. Given one boundary condition at the left edge of the domain, and 200 anchor conditions in between the start and the center of the domain. To ensure readability of the figure, one in every 10 anchor conditions is displayed.

The predictions shown in figure 11 show that the L-PIELM and PIMLP again give an accurate prediction, close to the analytical solution. The L-PIELM's prediction has an rMAE of $2.98 \cdot 10^{-8}$. In accordance with the results shown in figure 7, which show an increase in training data should not lead to a large increase of decrease in accuracy, the accuracy of the L-PIELM is similar to its result with one boundary condition and one anchor condition.

The PIMLP has an rMAE of $4.04 \cdot 10^{-7}$. An error similar to the result it achieved with one boundary condition and one anchor condition. Again, this is in line with the results from figure 7, as an increase in anchor conditions should not affect the accuracy of the PIMLP significantly.

The MLP and ELM both manage to give an accurate prediction for the part of the domain where there are anchor conditions. Yet both do not manage to predict the rest of the solution based on this data. The predictions made by the MLP and ELM are very different outside the domain containing training data, but both make an inaccurate prediction. The MLP does have a significantly lower rMAE than the ELM, as the prediction of the MLP remains closer to the analytical solution than the ELM's prediction.

4.3 Noise

4.3.1 Two noisy training data

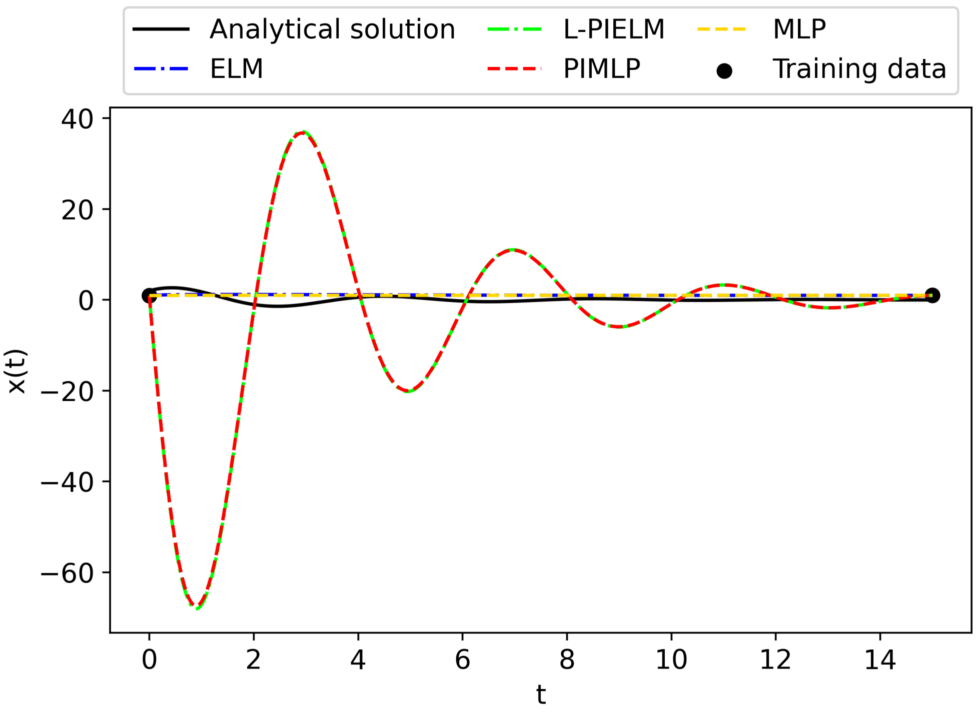

Figure 12: The prediction of the MLP, PIMLP, ELM, and L-PIELM of the particular solution of the linear harmonic oscillator with a mass of 2 kg, a stiffness of 5 $\frac{N}{m}$, and a frictional force of 1.2 $\frac{N \cdot s}{m}$. Given two noisy training data.

Figure 12 shows the prediction of the MLP, PIMLP, ELM, and L-PIELM of the particular solution of the linear harmonic oscillator with a mass of 2 kg, a stiffness of 5 $\frac{N}{m}$, and a frictional force of 1.2 $\frac{N \cdot s}{m}$, when the networks are given two noisy boundary conditions.

The ELM and MLP give almost identical predictions, both returning an almost straight line between the two given boundary conditions.
The L-PIELM gives a solution that probably satisfies the differential equation of the linear harmonic oscillator. However, because of the noisy boundary conditions, its prediction has a large error compared to the analytical solution. The PIMLP is in a similar situation, predicting a very similar solution as the L-PIELM, which likely fits the differential equation, but again has a large error compared to the analytical solution because of the noisy boundary conditions.

It is not possible to determine the analytical solution given these noisy boundary conditions, because the first derivative of the initial condition is required to calculate the analytical solution of the linear harmonic oscillator, and the first derivative of the initial condition is not known. Hence it is not possible to compare the performance of the L-PIELM and PIMLP against the analytical solution given the noisy data. Though given that they are able to accurately solve a linear harmonic oscillator, and the fact that both the L-PIELM and PIMLP give the same prediction, it is very likely that their prediction is accurate for these noisy boundary conditions.

4.3.2 50 noisy training data

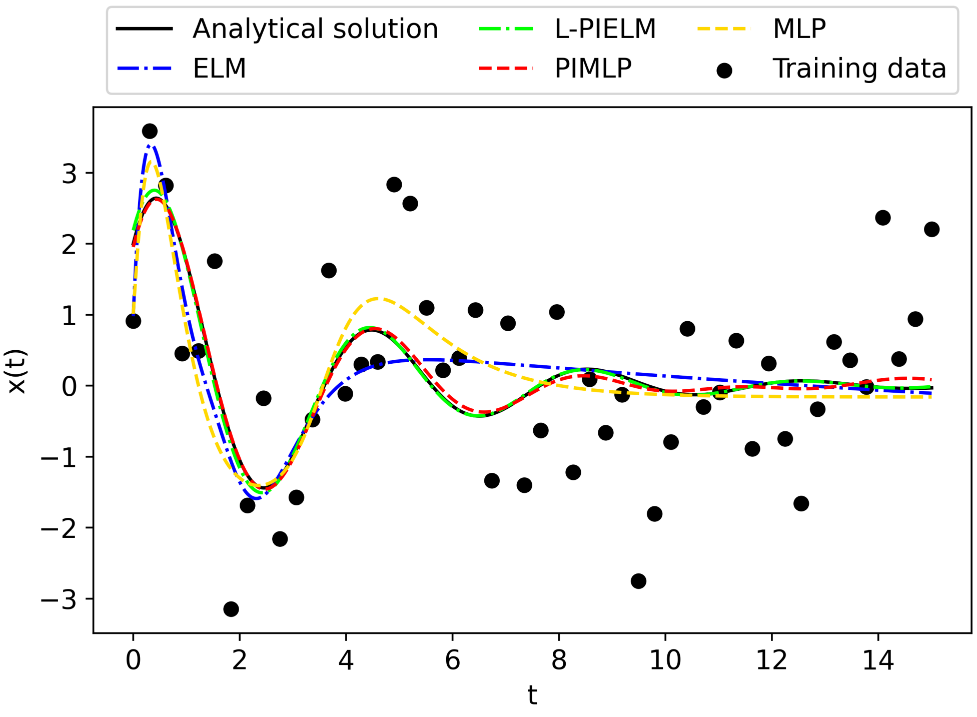

Figure 13: The prediction of the MLP, PIMLP, ELM, and L-PIELM of the particular solution of the linear harmonic oscillator with a mass of 2 kg, a stiffness of 5 $\frac{N}{m}$, and a frictional force of 1.2 $\frac{N \cdot s}{m}$. Given 50 noisy training data.

Figure 13 shows the prediction of the MLP, PIMLP, ELM, and L-PIELM of the particular solution of the linear harmonic oscillator with a mass of 2 kg, a stiffness of 5 $\frac{N}{m}$, and a frictional force of 1.2 $\frac{N \cdot s}{m}$, when each network is given 50 noisy training data.

Both the MLP and ELM manage to make a prediction that resembles the general shape of the analytical solution, yet are there are some clear inaccuracies visible. The MLP and ELM achieve similar accuracies, with the MLP's prediction having an rMAE of $1.15 \cdot 10^{-1}$, and the ELM achieving an rMAE of $9.40 \cdot 10^{-2}$.

The L-PIELM and the PIMLP both give predictions significantly closer to the analytical solution. Their performance is worse compared to a situation without noise (figure 7), but both their predictions appear relatively accurate with regards to the analytical solution. The L-PIELM makes a prediction with an rMAE of $1.01 \cdot 10^{-2}$, while the PIMLP's prediction has an rMAE of $2.10 \cdot 10^{-2}$. So significantly worse than their respective accuracies achieved on a noiseless dataset, yet clearly better than the prediction of the ELM and MLP.

4.4 Logistic equation

4.4.1 Amount of training data

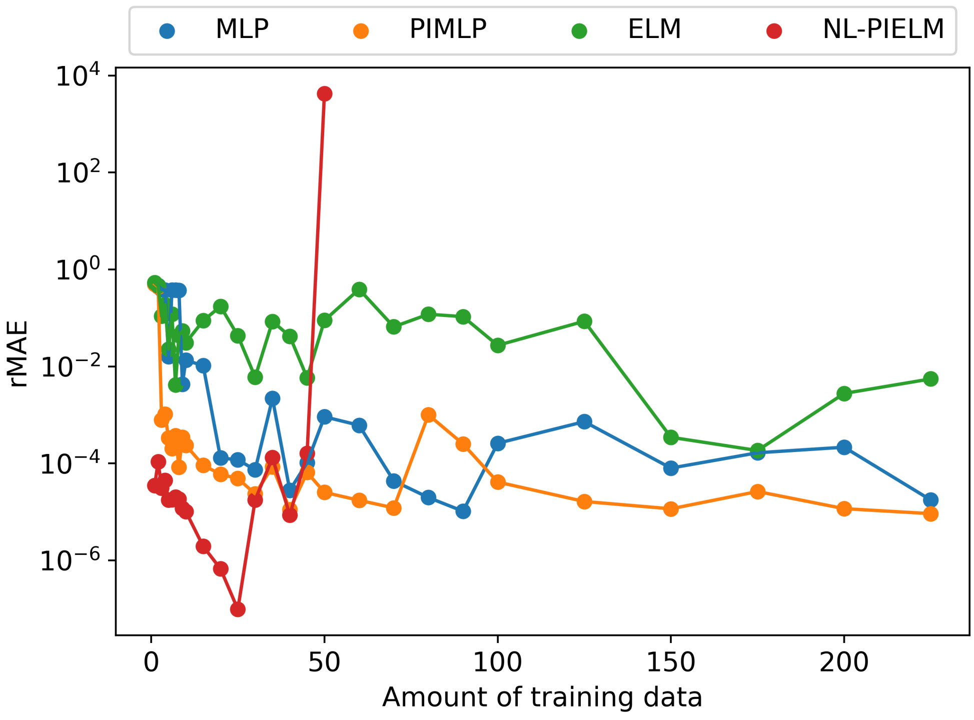

Figure 14: The rMAE of the MLP, the PIMLP, the ELM, and the NL-PIELM in predicting the particular solution of the logistic equation with a growth rate of 0.028 and a capacity of 943, plotted against the amount of training data provided to the network.

Figure 14 shows the rMAE of the MLP, the PIMLP, the ELM, and the NL-PIELM on the logistic equation. The NL-PIELM outperforms the other neural networks by a large margin when a small amount of training data is available. When only one boundary condition is provided, the NL-PIELM achieves an error of $3.49 \cdot 10^{-5}$, and its accuracy keeps improving until it reaches its most accurate prediction with 30 training data. However, with the increase in training data, the size of the constrained expression is expanding rapidly. At first this leads to a decrease in the accuracy achieved by the NL-PIELM, but when the amount of training data exceeds 50, the constrained expression has become so extensive that it requires more memory to solve than available, and the program runs out of memory. Hence the results achieved for the NL-PIELM are not available after the amount of training data exceeds 50.

The PIMLP is not able to provide an accurate prediction when only one boundary condition is provided. When the amount of training data is increased to two boundary conditions, the performance improves significantly. Further increasing the amount of training data does not cause any large differences in the accuracy of the PIMLP. With the exception when the PIMLP has 80 training data, in which case the PIMLP achieves a significantly worse result than with 70 or 100 training data. Overall, when the amount of training data equals or exceeds two, the PIMLP achieves a stable result with an rMAE around 10^{-4}.

The MLP does not manage to make an accurate prediction if provided a limited amount of data. When the amount of training data exceeds 25, the MLP's predictions are of a similar accuracy as the PIMLP. Although in most cases, the PIMLP remains slightly more accurate than the MLP.

The ELM performs similarly to the MLP when a low amount of training data is given, yet unlike the MLP, the ELM does not immediately improve its performance when the amount of training data is increased. When the amount of training data is increased to over a 100, the ELM's predictions become more accurate, and the performance becomes more similar to the performance of the MLP. The ELM however continues to achieve worse accuracies than the MLP.

4.4.2 Numerical methods

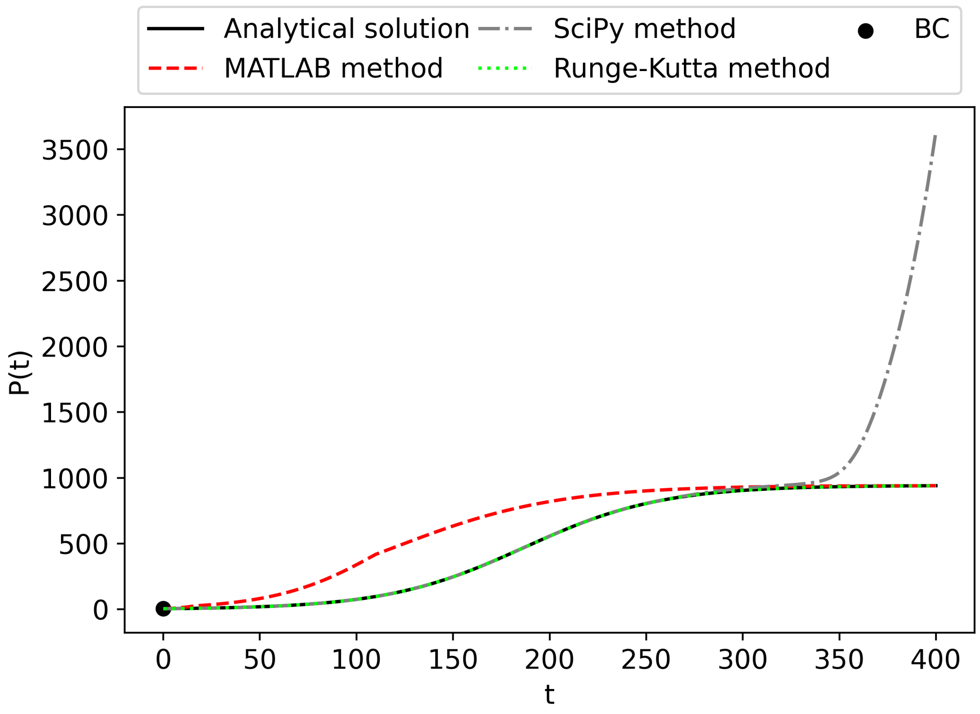

Figure 15: The prediction of the SciPy `solve_bvp` algorithm, the MATLAB `bvp5c` algorithm, and the Runga-Kutta fourth-order method of the logistic equation, given one boundary condition on the left edge of the solution

Figure 15 shows the prediction of the Runga-Kutta fourth-order method, the SciPy `solve_bvp` algorithm, and the MATLAB `bvp5c` algorithm of the logistic equation. They are given the initial value of P(0) as sole boundary condition. The Runga-Kutta fourth-order method finds an accurate solution with an rMAE of $1.54 \cdot 10^{-4}$. A similar accuracy as the Runga-Kutta fourth-order method achieved on the linear harmonic oscillator. The SciPy `solve_bvp` algorithm gives a less accurate prediction and achieves an rMAE of $1.44 \cdot 10^{-1}$. The MATLAB `bvp5c` algorithm's prediction looks more similar to the shape of the logistic equation, yet has an rMAE of $1.46 \cdot 10^{-1}$, slightly higher than the SciPy `solve_bvp` algorithm. This is somewhat counter intuitive based on figure 15, but is a result of using the rMAE as the error. Using the relative Mean Squared Error, (rMSE) instead of the rMAE, results in an error of 40.27 for MATLAB's `bvp5c` algorithm, and an error of 220.97 for SciPy's `solve_bvp` algorithm.

4.4.3 Runtime

Table 2: The runtimes of the different methods used to approximate the particular solution of the logistic equation given the lowest amount of training data at which they achieve an error smaller than 10^{-3}. Except for the SciPy `solve_bvp` and the MATLAB `bvp5c` algorithm, which each get one boundary condition.

Method	Runtime (seconds)
MLP	60.37
PIMLP	99.17
ELM	5.30
NL-PIELM	12.60
Runge-Kutta	0.01
MATLAB	3.07
SciPy	0.02

The runtimes shown in table 2 are the runtimes with the least amount of training data with which the networks achieve an rMAE of 10^{-3}. Except for the SciPy `solve_bvp` and the MATLAB `bvp5c` algorithm, which do not manage to achieve an rMAE lower than 10^{-3}. They are given one boundary condition each, as they cannot make a prediction based on multiple training data in case of a first-order ODE.

The runtimes are mostly in line with the runtimes recorded on the linear harmonic oscillator. Again the PIMLP and MLP are the slowest methods by a wide margin. while the Runge-Kutta fourth-order method and the SciPy `solve_bvp` algorithm are again the fastest algorithms. With the SciPy method, opposed to the other methods, even being slightly faster than it was on the linear harmonic oscillator.

The NL-PIELM is significantly slower on the logistic equation than the L-PIELM was on the linear harmonic oscillator. This is in line with the differences in their respective methods, as the L-PIELM is trained in a single step, whereas the NL-PIELM utilises iterative training.

The main outlier with regards to the runtimes in the linear harmonic oscillator is the ELM. Compared to the linear harmonic oscillator, the ELM requires a lot more time to train the model and make a prediction. This large increase in runtime is caused by the amount of hidden weights used. The ELM used to solve the logistic equation has multiple times as many hidden weights as the ELM used to solve the linear harmonic oscillator. This leads to an increase in the amount of calculations required to determine the activations of the hidden nodes and to calculate the Moore-Penrose inverse. Thus the ELM requires more time to train the model.

4.5 Partial differential equation

4.5.1 PIMLP

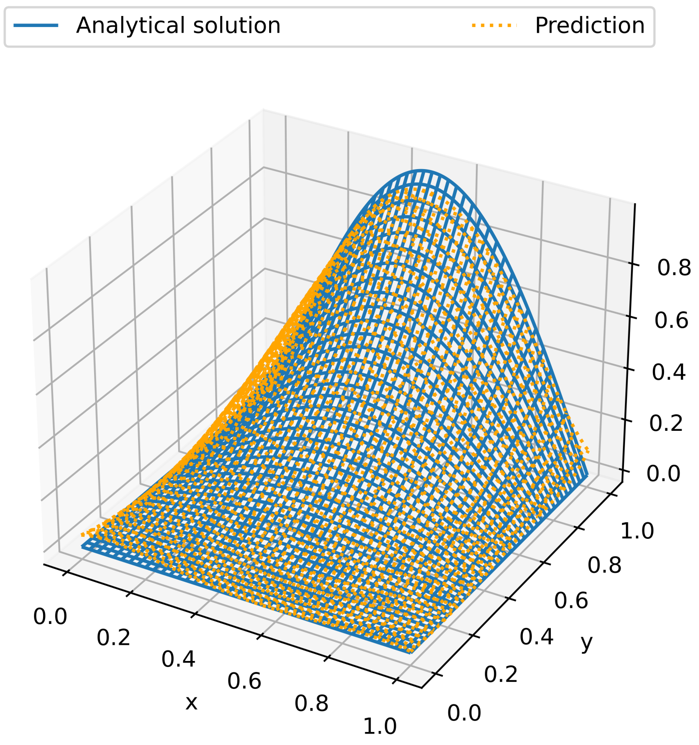

Figure 16: The analytical solution of the PDE shown in equation 46 and the prediction by the PIMLP of the PDE shown in equation 45

The prediction of the PIMLP is shown in figure 16. The general shape of the prediction appears to be quite similar to the analytical solution. There is however a clear difference discernible between the boundaries of the analytical solution and the boundaries of the prediction. During training however, the PIMLP has a similar MSE over the training data as over the collocation points, indicating the prediction over the collocation points is also not as accurate as it may seem. Overall, the prediction of the PIMLP has an rMAE of $6 \cdot 10^{-2}$.

4.5.2 NL-PIELM

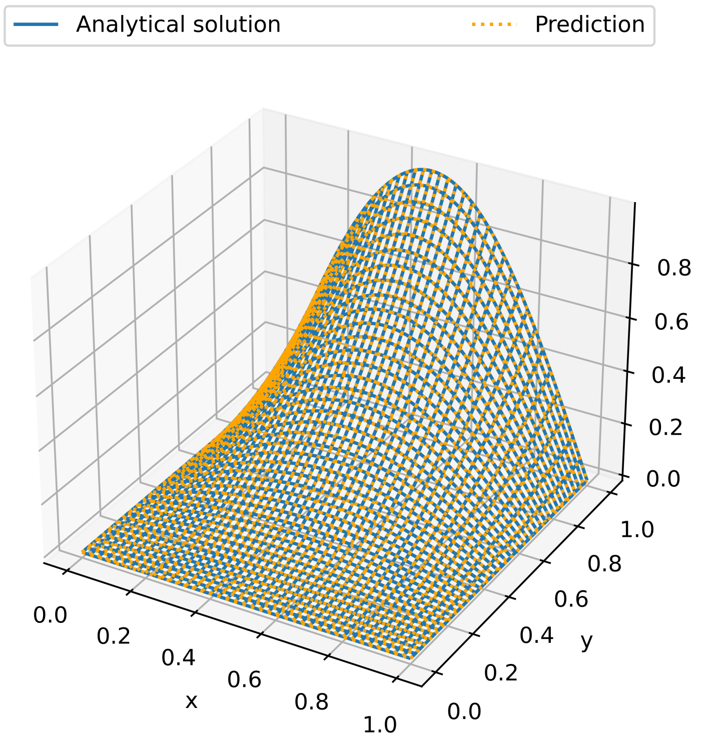

Figure 17: The analytical solution of the PDE shown in equation 46 and the prediction by the NL-PIELM of the PDE shown in equation 45

Figure 17 shows the error of the prediction of the NL-PIELM of the PDE shown in equation 45 compared to the analytical solution of the PDE shown in equation 46. There are no discernible differences between the analytical solution and the NL-PIELM's prediction. This aligns with the rMAE of $2.30 \cdot 10^{-16}$ that the NL-PIELM achieves. It takes the NL-PIELM 2.75 seconds to train the model and make a prediction.

5 Discussion

5.1 Linear harmonic oscillator

5.1.1 Amount of training data

The errors achieved on the linear harmonic oscillator with various amounts of training data, which were shown in figure 7, are mostly in line with the standard behaviour of the different neural networks. The ELM's and MLP's performance improves when more data is provided. At some point the neural network has enough data to make an accurate prediction, and the performance no longer improves when the amount of training data increases. In many practical applications, an excessive amount of training data can lead to overfitting. However, in this noiseless scenario, the training data is an accurate representation of the test data, and overfitting is therefore not a possibility.

The performance of the PIMLP is in line with the principles of a PINN. With low amounts of training data, the PIMLP has a large advantage compared to the standard MLP, because of the physics-informed regularization. This advantage decreases as the amount of training data increases, as the physics-informed component does not improve when more training data is provided, as it does not utilize any training data. The fit of the particular solution improves slightly if larger amounts of training data are provided, as the PIMLP benefits from the larger amount of points at which the error between the training data and the prediction can be calculated. The consistently lower error of the PIMLP compared to the MLP does show that even with a large amount of training data the physical information improves the fit slightly, keeping the performance of the PIMLP ahead of the performance of the ELM and the MLP.

The performance of the L-PIELM is also generally in line with the expectations. Because of the low error with a low amount of data, it is also not really possible for the extra training data to improve the results. As it uses a single step of learning instead of iterative training, its error is normally the result of either noise, of which there is none, or a lack of data, which is compensated by the physics-informed part. The error is therefore a combination of small inaccuracies, mainly in the calculations, which normally do not play a significant role in the error. With an error of 10^{-9} however, these small inaccuracies have a limiting effect on the final accuracy.

Overall the PIELM and PIMLP outperform the ELM and MLP when there is a small amount of data available. However, if enough data is available, the ELM and MLP achieve results similar to the PIELM and the PIMLP, although the PIELM and PIMLP are at all points more accurate. So the physics-informed component provides less of an improvement with a larger amount of data available. Furthermore, the advantage that it provides is less relevant in cases with larger amounts of training data, as the MLP and ELM also achieve accurate results. In most practical applications, the PIMLP achieving an error of 10^{-6} instead of 10^{-4} like the MLP, will have a limited effect on the effectiveness of the application. These results align with the framework behind PINNs, as they were designed to require less data to solve problems, which these results show is indeed their main strength.

5.1.2 Numerical methods

All the numerical methods achieve accurate results with low errors. Given that the Runga-Kutta fourth-order method has been shown to accurately solve similar problems (Hairer, Lubich, & Roche, 2006), it is no surprise that it is also able to solve this problem accurately. The MATLAB `bvp5c` algorithm and the SciPy `solve_bvp` algorithm are perhaps a bit less robust methods, but given that MATLAB is a highly regarded program, and SciPy is a well respected package, it is not surprising that these methods also provide accurate results.

5.1.3 Runtime

The runtimes of the different methods recorded in table 1 are influenced by many factors, and therefore likely not an entirely accurate representation. The runtime of the MLP and PIMLP is for instance dependent on how many epochs are made and what learning rate is applied. These hyperparameters have generally been tuned, but in a broad way. It is very likely that fine-tuning the learning rate would lead to a slightly faster runtime. All these differences however would not lead to any significant changes in the runtimes. Given the large differences between the runtimes, it is therefore safe to say that the PIMLP and MLP are the slowest, despite possible improvements to their runtimes. The PIMLP and MLP are the slowest methods because they both use iterative learning. A robust, accurate, but slow strategy. As the PIMLP also takes into account the physical information, by calculating the error over the collocation points, on top of the training data, the MLP is the faster one of the two.

The L-PIELM and ELM are both significantly faster than the MLP and PIMLP, primarily because they both train their output weights in a single step. The L-PIELM requires longer than the ELM to make a prediction because, like the PIMLP, the L-PIELM has to make extra calculations for the collocation points in the physics-informed component.

The three numerical methods have the advantage, in terms of runtime, that they do not train a model to make a prediction. This allows them to save a lot of time, as in all the neural networks the training of the model is what constitutes the majority of the runtime. That the MATLAB solver is nonetheless slower than the L-PIELM and ELM is therefore a bit of a surprise. As the MATLAB algorithm is run in a different environment than Python, in which all other methods are run, it is difficult to pinpoint exactly what causes this relatively slow runtime. Most likely it is the result of `bvp5c′s` algorithm. The runtime of the `bvp5c` method is also in general difficult to objectively ascertain, as it depends on the initial guess made. Running the `bvp5c` algorithm with a very accurate initial guess will lead to a lower runtime than reported, and vice versa. In this thesis the initial guesses provided were inaccurate, yet of the right order. This gives an initialization similar to the other methods, which do not require an initial guess.

It should be noted that to ensure a fair comparison of the runtimes, the runtimes where recorded with all methods running on a CPU. While this is the most fair comparison, it does not reflect that the MLP, PIMLP, and NL-PIELM can easily be run on a GPU, which would improve their runtime. The ELM and L-PIELM on the other hand are not easily transferable to the GPU, as they are build using the NumPy library, which cannot be run on a GPU. It is also doubtful for the ELM and L-PIELM whether the increase in the speed of calculations gained by the GPU would outweigh the time it takes to transfer the data from the CPU to the GPU. So in a practical setting the MLP, PIMLP, and NL-PIELM will have a faster runtime, while the runtime of the L-PIELM, the ELM, and the numerical methods, which also cannot be run on a GPU, would not improve. The decrease in the runtime of the MLP, PIMLP, and NL-PIELM will not be large enough to change the order of the different methods, yet will decrease the gap between different methods somewhat.

5.2 Anchor conditions

5.2.1 Two anchor conditions

The results in figure 9 show large differences in the predictions of the different networks. That the MLP and ELM both are unable to give accurate solutions is no surprise, as they both lack the information required to make an accurate prediction. Having only two data points next to each other in the center of the domain as training data, they draw a line between those points and extrapolate that information to make an inaccurate prediction in the rest of the domain.

The L-PIELM does not suffer from the fact that the boundary conditions have been replaced by anchor conditions. Because the L-PIELM is trained in a single step, it should not get stuck in a local minimum, the largest risk for other networks because of the limited information the anchor conditions provide. The L-PIELM confirms this and is indeed able to find the correct solution, despite the information provided by the training data being limited.

The most surprising result is the inaccuracy of the PIMLP. As it has the same information as the L-PIELM, so should theoretically be able to reach a similar result. The PIMLP is not able to reach the same accuracy as the L-PIELM, because the PIMLP gets stuck in a local minimum because of its iterative learning, a problem the L-PIELM does not suffer due to to its single step learning. Because the PIMLP's knowledge gained from the training data is significantly worse with two anchor conditions in the center of the particular solution, as compared to the situation with two boundary conditions as training data, the PIMLP is more likely to get stuck in a local minimum far from the analytical solution, which is what happens in this case.

5.2.2 One boundary condition and one anchor condition

When the (physics-informed) neural networks are given one boundary condition and one anchor condition, they all behave according to their respective capabilities. The MLP and ELM again only have 2 datapoints as training data, so cannot do more than predict a line that passes through both points.

For the L-PIELM the situation does not change significantly with respect to the situation with two anchor conditions. As it was already able to find an accurate solution, it is expected that it again manages to find an accurate solution, as the L-PIELM is provided with enough information to be able to find the particular solution.

The PIMLP is the only that network that significantly improves when one of the anchor conditions is replaced with a boundary condition. This improvement occurs because the boundary condition provides the PIMLP with more information than the anchor condition directly next to the second anchor condition did. This extra information enables the PIMLP to find an accurate solution without getting stuck in a local minimum.

5.2.3 One boundary condition and 200 anchor conditions

When given 200 anchor conditions, the PIMLP and L-PIELM behave according to the results shown in figure 7. The results of the PIMLP and the L-PIELM confirm that increasing the amount of training data, without altering the domain in which they lie, does not significantly affect the accuracy of the PINNs.

The MLP and ELM both show the behaviour expected from standard neural networks. Within the domain of the training data, so in between the boundary condition and the anchor conditions, they manage to give an accurate prediction. However, outside the domain of the training data, beyond the last anchor condition, their accuracy drops. As the MLP and ELM both do not have any sort of memory incorporated into their networks, and unlike the L-PIELM and PIMLP do not have any additional information in the form of collocation points, their networks are not trained to deal with data far away from any training data. Therefore they are not able to make an accurate prediction.

Overall, the PINN is a suitable solution for a situation in which only part of the domain contains training data. If the training data is also limited, for instance if only two anchor conditions are given, PINNs are, at least in the realm of neural networks, exceptional in finding an accurate solution. As mentioned before in the methods, it cannot be ruled out that there are numerical methods that can also produce an accurate prediction.

If a lot of training data is available, like in the scenario with one boundary condition and 200 anchor conditions, a PINN is also a suitable network. It is however not the only neural network that can achieve an accurate result. Also more standard neural networks can be used, provided they have some sort of memory incorporated into the network. Using a recurrent neural network, for instance an echo state network (Jaeger, 2001) or a long short-term memory network (Hochreiter & Schmidhuber, 1997), should result in a satisfactory prediction.

5.3 Noise

The results from figure 12 show that the PINNs lose their primary use when noise is present, as they are no longer able to make accurate predictions with small amounts of data. This is not a surprise, as any method using a small amount of training data will suffer badly from noise. There is after all no way to average out the noise if dealing with a small amount of training data. Also, as the whole prediction relies on a few data points, a small change in the training data will often lead to a large change in the obtained prediction.

It should be noted that the PINNs still give a prediction that is very likely to be the correct solution if the noisy boundary conditions were the actual boundary conditions. As the PINN still longer works if provided with noisy data, it gives a solution fitting the differential equation, yet for the wrong boundary conditions. In a practical application, the PINN will be ineffective if there is some noise in the measurements and the amount of data is limited, as an inaccurate prediction is often useless, even if the prediction does fit the provided differential equation.

The results shown in figure 13 show that the PINN can decrease the influence of noise on the neural network if a larger amount of training data is provided. This is a useful attribute, as noise is often a big challenge in practical applications. There are however many experiments done and papers written about ways to deal with noisy data, so the ability is not as unique as its core ability to decrease the required amount of training data. Given the strict requirements to be able to use a PINN, namely that a differential equation is required, there are more general ways to reduce the noisy data, whose broad applicability perhaps outweigh the PINN's capabilities.

The NL-PIELM is not included in this comparison, as the structure of the NL-PIELM already reveals what its prediction would look like. With two noisy boundary conditions as training data, it would give a prediction very similar to the L-PIELM and the PIMLP, assuming that they indeed predicted the correct particular solution for those noisy boundary conditions. With 50 noisy training data, the NL-PIELM would be strongly overfitted. As the the constrained expression analytically enforces the training data, the NL-PIELM would give a prediction passing through all training data, leading to an inaccurate result. Due to the structure of the NL-PIELM, it is not possible to decrease the overfitting using hyperparameter tuning or other measures, as the constrained expressions is not effected by hyperparameters, and will at all times analytically enforce the training data.

Compared to other numerical ways of solving differential equations, PINNs are the better choice when noise is present. The exception being PINNs like the NL-PIELM, that analytically satisfy the training data. Both PINNs and numerical methods will give an inaccurate prediction if only noisy boundary conditions are provided, but a PINN has the advantage that it can also use anchor conditions to determine the particular solution, whereas numerical solutions rely on small amounts of initial conditions or boundary conditions. In any situation where more information is available than just the boundary conditions, a PINN will therefore outperform numerical methods.

So overall, noise is a bit of a double edged sword for PINNs. Compared to normal neural networks, PINNs are as, if not more, vulnerable to noise when dealing with small amounts of data. Meaning that in situations were noise is present, which are almost all practical settings, the PINNs' main strength, the ability to solve problems with a small amount of training data, is diminished greatly. However, opposite to a noiseless situation, where a PINN's advantage compared to a normal neural network is reduced significantly if a sufficient amount of training data is provided, a PINN is very useful in a noisy situation when a larger amount of training data is given, as the physical information can be used to obtain a prediction that is less vulnerable to noise. Further research is required to determine whether the PINNs abilities are great enough to outperform other more widely applicable methods of reducing the effect of noise.

5.4 Logistic equation

5.4.1 Amount of training data

The most notable of the results achieved by the different neural networks on the logistic equation is the result of the NL-PIELM. Because of the constrained expressions, it achieves a low error with small amounts of data, but runs out of memory when larger amounts of training data are given. Why the NL-PIELM runs out of memory with larger amounts of training data, can be explained by looking at equation 19, which shows how the constrained expression is formed when given n points, and is the constrained expression used by the NL-PIELM in the logistic equation. Because it contains a sum over a product over the whole dataset for each boundary or anchor condition, the size of the constrained expression grows significantly with each anchor or boundary condition added to the training data. When the constrained expression becomes too large, this first leads to problems with the calculations, which causes the accuracy of the NL-PIELM to decrease. The expansion of the constrained expression also leads to an increase in the amount of memory required to calculate it. Eventually, the constrained expressions becomes so large that it requires more memory than available, and the constrained expression can no longer be calculated. On a modern CPU or GPU though, the amount of memory is large enough that the bottleneck will be the decrease in accuracy, not the lack of memory. In practice, this problem can often be avoided by ignoring part of the training data if the amount of training data exceeds the capabilities of the NL-PIELM. As the result shows that the NL-PIELM is able to achieve accurate results with limited amounts of data.

The PIMLP shows a result similar to the result it achieved on the linear harmonic oscillator. This makes sense, as for the PIMLP nothing really changes when the problem is non-linear instead of linear. It already uses iterative learning, and is capable of solving non-linear problems. The only situation in which the PIMLP does not manage to make an accurate prediction, is when a single boundary condition is provided. Although the PIMLP should theoretically be able to solve the problem given one boundary condition, it gets stuck in a local minimum. This is mainly because the single boundary condition causes the PIMLP to diverge in the wrong direction initially, and not be able to correct this problem later on because of the local minimum it gets stuck in. This problem could be avoided by providing an initial guess to the PIMLP, to assure its local minimum is the global minimum. This would however increase the knowledge required to use the system, as it would require the user to know the general shape of the solution beforehand. Other methods, like running the PIMLP multiple times with different random initializations, could also solve the problem, at the cost of a large increase in runtime.

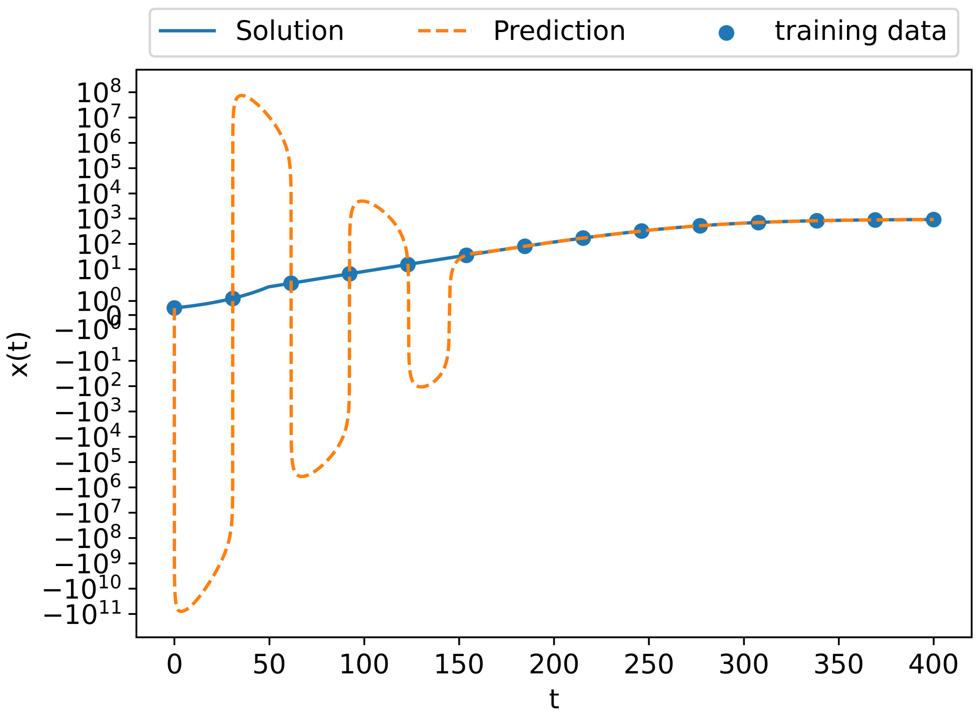

Figure 18: The prediction of the ELM on the logistic equation with 100 hidden weights, given 14 training data

The ELM's result is generally in line with expectations, if somewhat less accurate than expected. The most interesting thing about the results of the ELM are therefore not the accuracies shown in figure 14, but that the ELM achieves its most accurate results when given 30000 weights, despite only having at most a few hundred training data as input, while giving a more standard amount of weights leads to large errors. The errors with low amounts of hidden weights, an example of which is shown in figure 18, occur at the left side of the domain of the prediction. Based on the training data it is not an incorrect prediction, as the ELM gives an accurate prediction on all the training data. The prediction is however highly unlikely to be correct, based on the provided training data. Why this error occurs, is not clear. The most common reason for these sorts of problems is overfitting. Looking at the result this is a possibility, although the data given to the ELM is accurate, which means overfitting should not pose a problem. Also, one of the solutions to overfitting is decreasing the amount of weights, whereas in this case the error increases if the amount of weights is decreased. Another common reason for unexpected mistakes in ELMs is that the randomly initialized input weights have an unfavorable distribution. As these are random, and are not improved during training, the distribution has a large impact on the result. In this case, testing showed the initialization had a large impact, but the impact of the initialization seemed to be whether the ELM achieved a large error, or a very large error. Extensive testing showed no initialization lead to a reasonable error. The most likely possibility is that the error is a result of the non-linearity of the problem. As the ELM is likely to find a linear result because of its single step of learning. The large amount of weights would in this case serve as a way to decrease the accuracy of the ELM slightly, as some of the data is lost during the inverse and forward calculations with 30000 weights, thereby decreasing the error.

The MLP is the only network that shows the expected result on the logistic equation. Its performance improves when the amount of training data is increased, as is the norm in neural networks.

5.4.2 Numerical methods

As expected, the Runge-Kutta fourth-order method gives an accurate prediction. Because of its iterative training, it does not suffer from the fact that the differential equation is non-linear. Also the fact that it is a first-order problem, and therefore it is only provided one initial condition, only makes it simpler to make a prediction. As the Runge-Kutta fourth-order method, unlike all the other methods, has a separate algorithm for dealing with first-order problems, as was shown in the method section.

Surprisingly, the MATLAB `bvp5c` algorithm gives a quite inaccurate prediction. Even though boundary value problems are often regarded as problems with multiple boundary conditions, it is explicitly mentioned that the MATLAB `bvp5c` algorithm can also solve first-order differential equations if given one boundary condition. It is hard to say why the MATLAB function is not able to give a more accurate result, although the result suggests it assumes a larger first derivative at the initial condition than there is. It is also possible that the inaccuracy is because of the non-linearity of the problem, although there is nothing that suggests that the algorithm should not be able to deal with non-linear problems.

The reason for SciPy's `solve_bvp` algorithm's inaccurate prediction is also difficult to determine. The most likely option is that SciPy's `solve_bvp` algorithm is optimized to be used with multiple boundary conditions. Interestingly, when a boundary condition on the right side of the domain is used, the `solve_bvp` algorithm produces a prediction similar to the prediction made by MATLAB's `bvp5c` algorithm. This suggests that both methods have a similar reason for their inaccuracy, likely related to the fact that a single boundary condition is used, and possibly related to the fact that it is a non-linear problem.

5.4.3 Runtime

As the runtimes recorder on the logistic equation are very similar to those achieved on the linear harmonic oscillator, they don't require an extensive discussion. The only outliers recorder on the logistic equation are the NL-PIELM and the ELM. The NL-PIELM was not used on the linear harmonic oscillator, and it is no surprise that the NL-PIELM takes a longer time to run compared to the L-PIELM, because it utilises iterative learning instead of learning in a single step like the L-PIELM. The longer runtime of the ELM is, as mentioned before, because of the large amount of hidden weights used, and therefore is not a good indication of the ELM's runtime on similar problems.

5.5 Partial differential equation

5.5.1 PIMLP

The PIMLP's prediction of the PDE is significantly worse than the prediction of the NL-PIELM, and also quite a lot worse than expected based on its performance in other papers (Raissi et al., 2019). Why the PIMLP does not manage to make an accurate prediction is hard to say. Given that the error the PIMLP has over the boundary conditions as well as over the collocation points is in line with the achieved accuracy on the test set, it is not a structural problem within the PIMLP itself. A likely explanation for the inaccurate result is that the PIMLP is stuck in a local minimum. Most indicative of this possibility is the error over the boundary conditions. As minimizing the error over the boundary conditions should be a very simple task to solve for the PIMLP, it should be able to achieve a significantly lower error than it manages to on this problem. If only trained on the boundary conditions, effectively turning the PIMLP into an MLP, it is indeed able to predict the value of the boundary conditions with significantly higher accuracy. This suggests that in the PIMLP the error over the boundary conditions is not minimized because of the error over the collocation points, which acts as a counterweight. The error over the collocation points increases when the error over the collocation points decreases, and vice versa. Hence the PIMLP is likely in a local minimum. Another possibility is that the inaccurate result is caused by small inaccuracies with regards to the second derivative, as the prediction of the PIMLP is primarily reflected in the PDE in the second derivatives. Given that second derivatives are usually more fickle than first derivatives and the actual prediction, it is possible that small inaccuracies are enlarged within the second derivatives, leading to an unsolvable situation for the PIMLP. This is however a less likely explanation.

5.5.2 NL-PIELM

The error of the prediction of the NL-PIELM on the PDE is very small, and a lot smaller than the error of the best prediction of the NL-PIELM on the logistic equation. This result is counterintuitive, as PDEs are generally more difficult to solve than ODEs. The result however aligns with accuracies reported by the authors of the paper about constrained expressions, whom also achieved accuracies around 10^{-16} when solving PDEs (Schiassi et al., 2020). The NL-PIELM is able to achieve such a low error because of the constrained expressions. As the constrained expressions ensure the prediction fits the boundary conditions. The solution space of the possible result is also bounded, and the prediction over the PDE becomes relatively simple, because it is strongly limited by the constrained expression. This also explains why the NL-PIELM requires such a low amount of hidden weights to solve the problem, as only a small amount of possible solutions fit the constrained expression.

5.6 General remarks

5.6.1 Practical use

One of the major downsides of the PINN is that it is a solution to a very specific problem. The general idea of using physical information to improve the fit of neural networks is very promising. However, given that the physical information is computed using collocation points, this currently entails being able to solve differential equations, or problems that can be described via differential equations. PINNs would be applicable in a wider range of problems if they could deal with various types of physical information, and were not restricted to differential equations.

The limited practical use is also noticeable in the current publications on PINNs, which are almost exclusively examples of how various differential equations can be solved using different type of PINNs. There are some papers applying PINNs in a practical setting (Kissas et al., 2020), (Arzani, Wang, & D'Souza, 2021), yet their number is limited for now.

5.6.2 Flexibility

Perhaps the main advantage of using a PINN compared to more standard, numerical methods is the flexibility of PINNs. A major downside of most numerical methods of solving differential equations is that they can only solve a specific type of problem given a specific type and amount of information. The Runge-Kutta fourth-order method is significantly faster, and easier to implement than any PINN. It is less accurate than the most accurate PINNs, yet accurate enough for most practical problems. However, the Runge-Kutta fourth-order method is only applicable if the initial conditions are known. Meaning that in most practical applications its use will be severely limited, especially with higher order problems, where it is only able to find the particular solution of a differential equation if the relevant derivatives are provided. While in most cases it is a lot more complicated to measure e.g. the speed of an object than its current position.

Boundary value problem solvers like the MATLAB `bvp5c` method and the SciPy `solve_bvp` algorithm used in this thesis are able to solve problems if boundary conditions are given. This is already a lot more versatile than the Runge-Kutta method, but still the domain of what can be predicted is restricted to the boundary conditions. Similar to the Runge-Kutta fourth-order method, this means that these methods require derivatives if the problem is third-order or higher.

A PINN on the other hand can solve a differential equation when given initial conditions, but also when given boundary conditions, or anchor conditions, or a combination of different conditions. In practice this means it will be able to solve a differential equation provided it is given enough training data to find a unique solution, so in the case of a second-order ordinary differential equation it will require two data points of any kind. Perhaps most consequentially this means that, unlike the numerical methods used, a PINN never requires any derivatives to solve a problem, as anchor points can be included if the order of the problem increases.

5.6.3 Ease of use

One of the most positive aspects of the PINNs in regards to numerical methods is that they are easy to use, provided you have a working PINN. Especially in case of PDEs, there is no need to rewrite them to first-order ODEs, or perform any other transformations.

The limiting factor with regards to the usability of PINNs at the moment is that there are no easy ways to implement a PINN. Unlike many other neural networks, there are currently no packages that make it easy to implement a PINN, due to which the PINN has to be implemented manually. Especially in case of the L-PIELM and NL-PIELM, about which there has only been a small amount of research, this is a limiting factor for broad applicability. This can be overcome, but it does mean that currently PINN's will be used sparsely because other, more easily implementable methods are available. For instance the numerical methods are at this moment a more easily implementable alternative. An easily implementable PINN could however make some numerical methods of solving differential equations practically obsolete.

5.6.4 Identification

In this thesis the focus was exclusively on inference. But identification is also a promising method. The problem at the moment is that identification is, even compared to inference, limited to extremely specific situations. The current application of identification requires a dataset governed by a known differential equation. If these conditions are met, identification can be used to determine the parameters of the differential equation. There aren't many problems that fit this exact concept, hence the limited use of it at the moment.
However, the principle idea behind it is very promising. If an extension could be devised that can find a differential equation based on the dataset instead of only the parameters of a differential equation, it would provide the ability to use physical neural networks for a much larger range of problems.

5.7 Possible caveats

Despite trying to minimize the influence of any outside forces, there are external factors that may have played a role in the performance, and the differences in the performances, of the various methods.

Although there is no reason to believe this played a significant role, it is possible that the different packages used for the various neural networks have had an influence on the results. The ELM, the L-PIELM and the NL-PIELM all are implemented using NumPy, while the MLP and PIMLP are implemented via PyTorch. These are both widely used packages, so quality wise there should not be a big difference between them, yet some processes may be handled differently.

Another possible factor in the performance of the different methods is how the derivatives are calculated. In the PIMLP the derivative is calculated numerically using automatic differentiation on the current prediction. In the L-PIELM on the other hand, the derivative of the functions for the hidden nodes are calculated symbolically, and used as the derivatives of the differential equation. In the NL-PIELM the derivatives of the constrained expression are calculated via the elementwise gradient, and the relevant derivatives of the constrained expression are inserted in the loss function's differential equation. As all methods are confirmed to be accurate, this should not lead to any major differences. There may however be certain scenarios where the different methods lead to discrepancies in the calculated derivatives. The ELM and MLP both do not have to calculate any derivatives, as they do not use the physical information of the differential equations, so this has no influence on their results.

A factor that played a role in the results achieved by the PIMLP and MLP was the learning rate. The PIMLP and MLP had a dynamic learning rate with a minimum of 10^{-8}. It is possible that if the minimum learning rate was even smaller, and the amount of epochs increased, the MLP may reach even more accurate results. The decision to keep the minimum at 10^{-8} was twofold. First, there was the practical reason that the learning rate schedulers have a minimum learning rate of 10^{-8}, which meant that to decrease the learning rate further, a new learning rate scheduler would have to be written. Although this is not a particularly daunting task, it would have likely slowed the networks down, as the current learning rate scheduler are well optimized. The second reason is that the lower learning rate would have only played a role if the PIMLP and MLP had been given larger amounts of epochs, which would increase their runtimes, and make the methods less viable practically.

5.8 Further research

There are multiple ways in which this research, and research on PINNs in general, can be expanded upon. The most straightforward direction is more research focused on how PINNs can be improved. Most of the research currently taking place is focused on this topic. Personally however I do not believe this is the most relevant direction at the moment. As currently the application of PINNs is not limited by the accuracy achieved or the speed of the PINN, but by practical limitations. Research into how to decrease these practical limitations would therefore be interesting. This could for instance be research on whether PINNs can, with some modifications, be used to solve other problems than differential equations. The main constraint for expanding the practical limitations is the structure of the PINN. As its physical information is utilized via collocation points, any expansion to solve different problems would have to be implemented in such a way that it solves the problem using collocation points.

A theoretically very promising but complex expansion of the PINNs possibilities is extending the capabilities of identification and combining it with inference. Combined these expanded methods would be able to identify the differential equation governing a dataset, and use this differential equation to infer information about the dataset with significantly less training data. This could theoretically mean that any dataset which can be described as a differential equation could be solved with reduced amounts of training data, using PINNs. A large factor in the viability of this method is to what extent datasets can be described by a differential equation. A topic that, as far as I know, is limited research about at the moment. It is therefore possible that many problems cannot be expressed as differential equations. The practical applicability of this method therefore depends on the amount of datasets which can be described by a differential equation.

6 Conclusion

Physics-informed neural networks fulfill their main promise. They are indeed able to solve differential equations with significantly less training data than standard neural networks. The L-PIELM, the NL-PIELM, and the PIMLP all outperform the standard ELM and MLP by a considerable margin when a limited amount of data is available. If a large amount of data is available, the PINNs present less of an improvement over the standard neural networks, with the exception of noisy datasets. When noise is present, the PINNs can be utilized as noise regularizers and improve the performance. Using a PINN as a noise regularizer is therefore also a viable use of the PINN, as noisy data is a common problem in many practical settings.

Overall, PINNs prove to be an accurate method to solve differential equations, and can rival any other method designed to solve differential equations numerically. In a more practical sense, their current applicability is severely limited. As they only provide a significant improvement when a differential equation has to be solved with a limited amount of data. Therefore PINNs are currently among the most accurate ways of solving differential equations, yet their practical applicability is limited at the moment.

References

Arnold, V. I. (1992). *Ordinary differential equations*. Springer Science & Business Media.

Arzani, A., Wang, J.-X., & D'Souza, R. M. (2021). Uncovering near-wall blood flow from sparse data with physics-informed neural networks. *Physics of Fluids*, *33*(7), 071905.

Baydin, A. G., Pearlmutter, B. A., Radul, A. A., & Siskind, J. M. (2018). Automatic differentiation in machine learning: a survey. *Journal of Machine Learning Research*, *18*, 1–43.

Cai, S., Wang, Z., Wang, S., Perdikaris, P., & Karniadakis, G. E. (2021). Physics-informed neural networks for heat transfer problems. *Journal of Heat Transfer*, *143*(6).

Crank, J., & Nicolson, P. (1947). A practical method for numerical evaluation of solutions of partial differential equations of the heat-conduction type. In *Mathematical proceedings of the cambridge philosophical society* (Vol. 43, pp. 50–67).

Doan, N. A. K., Polifke, W., & Magri, L. (2020). Physics-informed echo state networks. *Journal of Computational Science*, *47*, 101237.

Dwivedi, V., & Srinivasan, B. (2020). Physics informed extreme learning machine (pielm)–a rapid method for the numerical solution of partial differential equations. *Neurocomputing*, *391*, 96–118.

Gao, H., Sun, L., & Wang, J.-X. (2021). Phygeonet: Physics-informed geometry-adaptive convolutional neural networks for solving parameterized steady-state pdes on irregular domain. *Journal of Computational Physics*, *428*, 110079.

Hairer, E., Lubich, C., & Roche, M. (2006). *The numerical solution of differential-algebraic systems by runge-kutta methods* (Vol. 1409). Springer.

Hale, J. K., & Lunel, S. M. V. (2013). *Introduction to functional differential equations* (Vol. 99). Springer Science & Business Media.

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, *9*(8), 1735–1780.

Huang, G.-B., Zhu, Q.-Y., & Siew, C.-K. (2006). Extreme learning machine: theory and applications. *Neurocomputing*, *70*(1-3), 489–501.

Jaeger, H. (2001). The "echo state" approach to analysing and training recurrent neural networks-with an erratum note. *Bonn, Germany: German National Research Center for Information Technology GMD Technical Report*, *148*(34), 13.

Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Kissas, G., Yang, Y., Hwuang, E., Witschey, W. R., Detre, J. A., & Perdikaris, P. (2020). Machine learning in cardiovascular flows modeling: Predicting arterial blood pressure from non-invasive 4d flow mri data using physics-informed neural networks. *Computer Methods in Applied Mechanics and Engineering*, *358*, 112623.

Kreiss, H.-O., & Lorenz, J. (2004). *Initial-boundary value problems and the navier-stokes equations*. SIAM.

Kutta, W. (1901). Beitrag zur naherungsweisen integration totaler differentialgleichungen. *Z. Math. Phys.*, *46*, 435–453.

Lagaris, I. E., Likas, A., & Fotiadis, D. I. (1998). Artificial neural networks for solving ordinary and partial differential equations. *IEEE transactions on neural networks*, *9*(5), 987–1000.

Leake, C., & Johnston, H. (2021). *TFC: A Functional Interpolation Framework.* Retrieved from `https://github.com/leakec/tfc`

Leake, C., Johnston, H., & Mortari, D. (2020). The multivariate theory of functional connections: Theory, proofs, and application in partial differential equations. *Mathematics*, *8*(8), 1303.

Leake, C., & Mortari, D. (2020). Deep theory of functional connections: A new method for estimating the solutions of partial differential equations. *Machine learning and knowledge extraction*, *2*(1), 37–55.

Maclaurin, D., Duvenaud, D., Johnson, M., & Townsend, J. (2015). *Autograd.* Retrieved from `https://github.com/HIPS/autograd`

Mason, J. C., & Handscomb, D. C. (2002). *Chebyshev polynomials*. Chapman and Hall/CRC.

MATLAB. (2022). *Version r2022a*. Natick, Massachusetts: The MathWorks Inc.

McKay, M. D., Beckman, R. J., & Conover, W. J. (1979). A comparison of three methods for selecting values of input variables in the analysis of output from a computer code. *Technometrics*, *21*(2), 239–245.

Mikołajczyk, A., & Grochowski, M. (2018). Data augmentation for improving deep learning in image classification problem. In *2018 international interdisciplinary phd workshop (iiphdw)* (pp. 117–122).

Mortari, D. (2017). The theory of connections: connecting points. *Mathematics*, *5*(4), 57.

Øksendal, B. (2003). Stochastic differential equations. In *Stochastic differential equations* (pp. 65–84). Springer.

Owhadi, H. (2015). Bayesian numerical homogenization. *Multiscale Modeling & Simulation*, *13*(3), 812–828.

Park, D. S., Chan, W., Zhang, Y., Chiu, C.-C., Zoph, B., Cubuk, E. D., & Le, Q. V. (2019). Specaugment: A simple data augmentation method for automatic speech recognition. *arXiv preprint arXiv:1904.08779*.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., … Chintala, S. (2019). Pytorch: An imperative style, high-performance deep learning library. In *Advances in neural information processing systems 32* (pp. 8024–8035). Curran Associates, Inc.

Pearl, R., & Slobodkin, L. (1976). The growth of populations. *The Quarterly Review of Biology*, *51*, 6–24.

Pelinovsky, E., Kurkin, A., Kurkina, O., Kokoulina, M., & Epifanova, A. (2020). Logistic equation and covid-19. *Chaos, Solitons & Fractals*, *140*, 110241.

Raissi, M., & Karniadakis, G. E. (2018). Hidden physics models: Machine learning of nonlinear partial differential equations. *Journal of Computational Physics*, *357*, 125–141.

Raissi, M., Perdikaris, P., & Karniadakis, G. E. (2017a). Inferring solutions of differential equations using noisy multi-fidelity data. *Journal of Computational Physics*, *335*, 736–746.

Raissi, M., Perdikaris, P., & Karniadakis, G. E. (2017b). Machine learning of linear differential equations using gaussian processes. *Journal of Computational Physics*, *348*, 683–693.

Raissi, M., Perdikaris, P., & Karniadakis, G. E. (2018). Numerical gaussian processes for time-dependent and nonlinear partial differential equations. *SIAM Journal on Scientific Computing*, *40*(1), A172–A198.

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

Ravuri, S., & Vinyals, O. (2019). Seeing is not necessarily believing: Limitations of biggans for data augmentation. *ICLR*.

Rosenblatt, F. (1958). The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, *65*(6), 386.

Runge, C. (1895). Über die numerische auflösung von differentialgleichungen. *Mathematische Annalen*, *46*(2), 167–178.

Russell, R., & Shampine, L. F. (1972). A collocation method for boundary value problems. *Numerische Mathematik*, *19*(1), 1–28.

Schiassi, E., Leake, C., De Florio, M., Johnston, H., Furfaro, R., & Mortari, D. (2020). Extreme theory of functional connections: A physics-informed neural network method for solving parametric differential equations. *arXiv preprint arXiv:2005.10632*.

Schiesser, W. E. (2012). *The numerical method of lines: integration of partial differential equations*. Elsevier.

Shokouhi, P., Kumar, V., Prathipati, S., Hosseini, S. A., Giles, C. L., & Kifer, D. (2021). Physics-informed deep learning for prediction of co2 storage site response. *Journal of Contaminant Hydrology*, *241*, 103835.

Singh, S. K., Yang, R., Behjat, A., Rai, R., Chowdhury, S., & Matei, I. (2019). Pi-lstm: Physics-infused long short-term memory network. In *2019 18th ieee international conference on machine learning and applications (icmla)* (pp. 34–41).

Süli, E., & Mayers, D. F. (2003). *An introduction to numerical analysis*. Cambridge university press.

Tanaka, F. H. K. d. S., & Aranha, C. (2019). Data augmentation using gans. *arXiv preprint arXiv:1904.09135*.

Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., … SciPy 1.0 Contributors (2020). SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, *17*, 261–272.

Wengert, R. E. (1964, aug). A simple automatic derivative evaluation program. *Commun. ACM*, *7*(8), 463–464.

Werbos, P. J. (1990). Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, *78*(10), 1550–1560.

Wong, S. C., Gatt, A., Stamatescu, V., & McDonnell, M. D. (2016). Understanding data augmentation for classification: when to warp? In *2016 international conference on digital image computing: techniques and applications (dicta)* (pp. 1–6).