Wrocław University
of Science and Technology

Faculty of Pure and Applied Mathematics
Field of study: Applied Mathematics
Specialty: Data Engineering

Master's Thesis

DEEP LEARNING METHODS FOR SOLVING ORDINARY DIFFERENTIAL EQUATIONS

Joanna Wojciechowicz

keywords:
Physics-informed Neural Network, Deep Euler Method, Deep Neural Network, ODE

short summary:
This work focuses on two primary deep learning methods for solving ordinary differential equations: physics-informed neural networks and the Deep Euler method. We theoretically examine both methods and evaluate their performance using examples from real physical systems. Additionally, we propose a correction to the physics-informed neural network for a specific ODE system. We analyze various neural network architectures and method parameters to assess the impact of different components on their performance.

Supervisor	dr hab. Janusz Szwabiński	....	....
	Title/degree/name and surname	grade	signature

Wrocław, 2024

Contents

Introduction

Many engineering problems can be modeled as

$$G\left(x, y, y', y'', \ldots\right) = 0,\ x \in [a, b] \subset \mathbb{R}, \tag{1}$$

where G is a given function specific to the problem, x is the independent variable and $y = y\left(x\right)$ is the dependent variable [26]. In most cases, it is not easy to obtain the analytic solution for y, so we have to rely on numerical methods to approximate it. These numerical methods, including the well-known Euler method [9] or the Runge-Kutta method [9], often have to balance the step size and time for computation.

In recent years, deep learning methods [5] have garnered significant attention for addressing scientific problems, including solving ordinary differential equation systems of the form (1) [12]. One strategy involves training a model to replicate the behavior of the system, enabling it to learn the correct trajectories [8]. Alternatively, deep learning can be used to enhance traditional numerical methods, addressing their well-documented limitations [26].

In this work, we will explore two approaches to integrating deep learning into solving ordinary differential equation (ODE) systems. First, we will examine physics-informed neural networks, which learn ODE patterns in an unsupervised manner and approximate the trajectory directly. Afterwards, we will focus on the Deep Euler method, which enhances the standard Euler method by predicting the local truncation error of the numerical method. In both cases, we will evaluate the performance of these methods using examples from physical systems.

Chapter 1

Preliminaries

1.1 ODEs

ODEs are extensively used in physics to model phenomena such as motion, heat conduction, wave propagation, and electrical circuits [15]. By solving these equations, we can predict the future states of physical systems and understand the underlying principles governing their behavior.

In this chapter, examples of ODEs from real-world physics problems will be introduced. Those examples will be used later in this work to test the applicability of deep learning techniques to solve the dynamical systems.

1.1.1 Newton's Law of Cooling

Newton's Law of Cooling [23] describes the rate at which an exposed body changes temperature through radiation. The law states that the rate of change of the temperature of an object is proportional to the difference between its own temperature and the ambient temperature. The formula for Newton's Law of Cooling is given by

$$\begin{cases} \frac{dT}{dt} = -k\left(T - A\left(t\right)\right), \\ T\left(0\right) = T_0, \end{cases} \tag{1.1}$$

where $T = T(t)$ is the temperature of the body, T_0 is its initial temperature, $A = A(t)$ is a given ambient temperature and $k \geq 0$ is a proportionality constant dependent on the specific heat of the body and its density. The analytic solution is of the form

$$T\left(t\right) = A + \left(T_0 - A\right)e^{-kt}. \tag{1.2}$$

1.1.2 Heavily damped oscillator

A heavily damped oscillator is a system in which the damping force is so strong that it prevents oscillations [22]. Instead, the system returns to equilibrium without overshooting. This behavior is characterized by the second-order linear differential equation

$$m\frac{d^2x}{dt^2} = -kx - b\frac{dx}{dt}, \tag{1.3}$$

where $x(t)$ is the position of the oscillator, m is its mass, k is a spring constant and b is a constant corresponding to damping. In order to obtain the heavy damping we need to meet the condition

$$b^2 \geq 4mk. \tag{1.4}$$

The analytic solution of the heavily damped oscillator problem is of the form

$$x(t) = x_0 e^{-\alpha t}, \tag{1.5}$$

where $\alpha = \frac{k}{b}$.

1.1.3 SIS model

The SIS model [14] is a simple epidemiological model used to describe the spread of infectious diseases within a population. It divides the group into two parts: susceptible and infected. Individuals in the SIS model do not acquire immunity after infection; instead, they return to the susceptible state after recovery. The model is represented by the following system of ODEs:

$$\begin{cases} \frac{dS}{dt} = -\beta SI + \gamma I, \\ \frac{dI}{dt} = \beta SI - \gamma I, \end{cases} \tag{1.6}$$

where S is the number of susceptible individuals, I is the number of infectious individuals, $S(0) = S_0$, $I(0) = I_0$ are initial conditions, $S(t) + I(t) = N$ is the total population, β is the infection rate, and γ is the recovery rate.

1.1.4 SIR model

The SIR model [14] is a classical epidemiological model used to describe the spread of infectious diseases in a population. It divides the population into three compartments: susceptible, infected, and recovered. The SIR model is widely used to understand the dynamics of infectious diseases in which individuals gain immunity after recovery, including measles, mumps and rubella. The behavior of the model is described by the system

$$\begin{cases} \frac{dS}{dt} = -\beta IS, \\ \frac{dI}{dt} = \beta IS - \gamma I, \\ \frac{dR}{dt} = \gamma I, \end{cases} \tag{1.7}$$

where S is the number of susceptible individuals, I is the number of infected individuals, R is the number of recovered individuals, $S(0) = S_0$, $I(0) = I_0$, $R(0) = R_0$ are initial conditions, $S(t) + I(t) + R(t) = N$ is the total population, β is the infection rate, and γ is the recovery rate.

1.2 Neural Networks

One of the architectures most commonly employed in the field of deep learning is the feed-forward neural network (FNN) [3]. This simple yet powerful model serves as the foundation for understanding more complex networks.

The basic unit of a neural network is the perceptron [16], a machine learning algorithm that attempts to find a line, plane, or hyperplane to separate classes in a multidimensional space. When multiple perceptrons are connected, the model is referred to as a multilayer perceptron (MLP) or an artificial neural network (ANN). As shown in Fig. 1.1, presenting the typical architecture of ANNs, they consist of an input layer, an output layer, and one or more hidden layers between them. Each of these layers is built from so-called neurons (nodes). Deep neural networks (DNNs) can contain tens or even hundreds of hidden layers.

In a feedforward neural network, information moves in one direction, from the input layer, through the hidden layers, to the output layer. Each node in a layer connects to every node in the next layer. The feedforward process involves the following steps:

1. The input layer receives the input features x.

2. Each hidden layer node receives inputs a from all nodes of the previous layer, applies a weighted sum, adds a bias, and passes the result through an activation function [25]. The transformation for the j^{th} node in the l^{th} layer can be expressed as

$$z_j^{(l)} = \sum_i w_{ij}^{(l-1)} a_i^{(l-1)} + b_j^{(l)} \tag{1.8}$$

   and

$$a_j^{(l)} = \sigma\left(z_j^{(l)}\right), \tag{1.9}$$

   where $w_{ij}^{(l-1)}$ are the weights, $b_j^{(l)}$ are the biases, $a_i^{(l-1)}$ are the activations from the previous layer and σ is the activation function.

3. The output layer processes the final transformation and produces the network's prediction.

In neural networks, the activation function introduces non-linearity into the model, enabling it to capture complex patterns and relationships in the data. By applying the activation function to the output of each neuron, the network becomes capable of learning and representing nonlinear mappings between input and output variables.

Activation functions are selected on the basis of the task's requirements. One of the most commonly used are sigmoid, ReLU (rectified linear unit) and leaky ReLU (leaky rectified linear unit) activation functions. Sigmoid is of the form

$$\sigma(x) = \frac{1}{1+e^{-x}}. \tag{1.10}$$

The ReLU is given by

$$f(x) = max(0, x). \tag{1.11}$$

The leaky ReLU is defined as

$$f(x) = \begin{cases} x, \ x \geq 0 \\ \alpha x, \ otherwise, \end{cases} \tag{1.12}$$

where α is a small positive slope, usually around 0.01 value.

Learning in feedforward neural networks [6] involves adjusting the weights and biases to minimize the error between the predicted output and the actual target. To reduce this margin, we minimize the loss function (also called the cost function),

$$J(\theta) = \sum_i \left(f_i - \hat{f}_i\right), \tag{1.13}$$

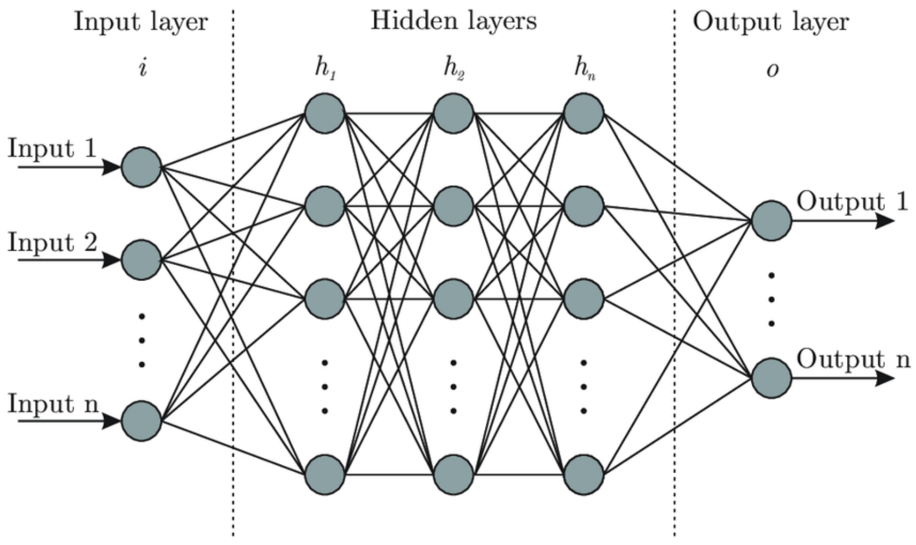

Figure 1.1: Architecture of feedworward neural network. Source: [1].

where θ are all of the neural network parameters that we are adjusting during training, f_i is the real target value, and $\hat{f}_i$ is the network's predicted value. Minimization is typically achieved through backpropagation process [2] using gradient descent [7]. The error is propagated backward through the network. Partial derivatives of the error with respect to each weight and bias are computed using the chain rule. Weights and biases are updated in the opposite direction of the gradient of the error function. This adjustment aims to reduce the error iteratively. The weight update rule for a weight w_{ij} can be expressed as

$$w_{ij} \leftarrow w_{ij} - \eta \frac{\partial J}{\partial w_{ij}}, \tag{1.14}$$

where η is the learning rate and J is the loss function.

By progressively modifying the inputs at each layer, the network learns to produce outputs that are more predictive of the desired outcome. This ability to learn complex patterns makes feedforward neural networks a powerful tool for various applications in classification, regression, and beyond.

Chapter 2

Physics-informed Neural Networks

2.1 Introduction

Since deep learning [5] has received great attention throughout the last decades, many scientists try to apply it to model various physical phenomena described by ODE systems. There are well-established numerical methods for solving ODE systems, such as the Runge-Kutta method. However, these methods have their own strengths and weaknesses regarding accuracy, stability, convergence, and computation time [9]. This presents a significant opportunity for artificial neural networks to enhance the performance of solvers.

In this part of the work we will present a physics-informed neural network (further called PINN) that solves the ODE system based on the unsupervised learning procedure [8]. We implement the algorithm in Python and evaluate its performance on several ODE examples, including Newton's Law of Cooling (Sec. 1.1.1), a heavily damped oscillator (Sec. 1.1.2), and the SIR model (Sec. 1.1.4). These examples allow us to assess the algorithm's effectiveness on both single ODEs and systems of ODEs. We investigate the impact of different neural architectures and activation functions on model performance. Comparisons are made using analytical solutions where available, or numerical solutions obtained with the Runge-Kutta method. Additionally, we analyze the effect of grid size on the performance of PINNs relative to the Runge-Kutta method.

Afterwards, we propose an enhancement to the previous PINN model [11, 13, 27] by incorporating a supervised knowledge source into the loss function. Finally, we examine the limitations of both the original PINN method and the improved PINN method.

2.2 Details of PINN

Let us consider a general initial value problem in a form

$$\begin{cases} \frac{d\mathbf{y}}{dt} = \mathbf{f}\left(t, \mathbf{y}\right), \\ \mathbf{y}\left(a\right) = \mathbf{y}_0, \end{cases} \tag{2.1}$$

defined on $a \leq t \leq b$, where $\mathbf{y}$ is the vector of unknowns trajectories $y_1,\ y_2, \ldots, y_n$ and $\mathbf{f}(t, \mathbf{y})$ is a given vector valued function. For this problem, we have existing unique solutions [4].

We use a dense neural network (Fig. 2.1) with L layers consisting of one neuron in the input layer corresponding to the independent variable t and n neurons in the output layer corresponding to unknown variables $y_1,\ y_2, \ldots, y_n$ (see Sec. 1.2 for more details regarding the basics of neural networks). We train the neural network on m sample points t from the domain $[a, b]$ and form a matrix $X = [t^{(1)}, \dots, t^{(m)}] \in \mathbb{R}^{1\times m}$. Here, $t^{(i)} \in [a, b] \subset \mathbb{R}$ is the i^{th} training example. We denote the output matrix as $\mathbf{N} \in \mathbb{R}^{n\times m}$. The neural network that we will build will predict for each $t \in [a, b]$ the trial solution of the form [18, 21]

$$\hat{y}_j (t, P_j) = a_j + (t - a) N_j (t, P_j), \ j = 1, \dots, n, \tag{2.2}$$

where a_j is the initial value of y_j trajectory, $N_k \left(t^{(i)}, P_k\right)$ is the output of the neural network of the k^{th} unknown corresponding to the i^{th} sample point and P_k is the corresponding parameters of the network, i.e. the weights and the bias. The trial solution (2.2) satisfies the initial conditions.

We train the neural network in such a way that we minimize the total cost function

$$J = \frac{1}{m} \sum_{i=1}^{m} \sum_{j=1}^{n} \left(\frac{d\hat{y}_j}{dt} - f_j \right)^2 , \tag{2.3}$$

where $f_j = f_j \left(t^{(i)}, \hat{y}_j \left(t^{(i)}, P_j\right)\right)$. We are using the Adam algorithm [17] with a learning rate equal to 0.001 from the PyTorch library [24], so that the cost function (2.3) converges to 0.

Looking at the cost function (2.3) we can see that the learning process is unsupervised since the knowledge of the trajectory is passed only through the right side of the ODE system (2.1). The function consists of the derivative of the neural network output. PyTorch automatic differentiation technique of neural networks can be used to compute the derivatives of the network output with respect to the input. The cost function consists of all the cost functions related to each of the predicted trajectories in the system.

The neural network that we use is a standard multilayer, fully connected network. The L^{th} hidden layer has the form

$$Z^L = W^L A^{L-1} + \mathbf{b}^L, \tag{2.4}$$

where

$$A^L = \sigma^L \left(Z^L\right). \tag{2.5}$$

Here W^L is the matrix that contains all the weights w^L, $\mathbf{b}^L$ is the bias in the layer L and σ^L is the activation function chosen in the layer L.

For the basic version of the algorithm, we choose the network with one hidden layer, with 60 neurons and the sigmoid activation function (1.10), 11 time steps in the grid, 3000 number of epochs.

2.3 Correction to PINN

The PINN described in Sec. 2.2 introduces physical constraints into the training explicitly by incorporating the right side of the (2.1) equation into the cost function (2.3). We will also consider a correction to the PINN that incorporates a new factor into the loss function, namely data loss [11, 13]. This factor is a difference between the predicted value and the analytic solution at the specific point.

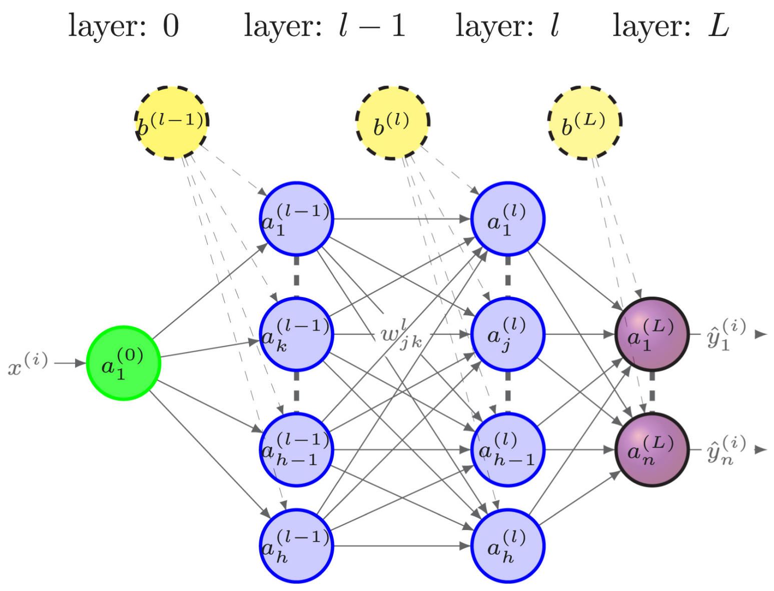

Figure 2.1: Schematic diagram of deep ANN. Source: [8].

Here, the loss function consists of two terms, residual loss and data loss. Compared to equation (2.3), the cost function now has the form

$$J = \frac{1}{m}\sum_{i=1}^{m}\sum_{j=1}^{n}[\left(\frac{d\hat{y}_j}{dt} - f_j\right)^2 + (\hat{y}_j - y_j)^2], \tag{2.6}$$

where y_j is the actual analytic solution.

It is important to note that in this version of the algorithm we train the neural network in a supervised manner. We minimize the loss function (2.6) that contains additional knowledge about the analytic solution. In terms of the SIR model (Sec. 1.1.4), we will use Runge-Kutta solution as the exact one. In this version of the algorithm the loss function also consists of loss functions of each predicted trajectory in the ODEs system.

For this proposed correction, we also use the Adam algorithm from the PyTorch library with 0.0001 learning rate. We train the network over 200000 epochs. To avoid overfitting and improve the generalization ability of the model, we implement the early stopping with patience equal to 100.

Chapter 3

Deep Euler method

3.1 Introduction

The Euler forward method is one of the most widely employed numerical techniques. Its appeal lies in its simplicity of implementation, yet it falls short in accuracy [9, 26]. This limitation results from its first-order approximation, necessitating extremely small discretization step sizes for meaningful results. Consequently, in practical applications, the Euler method is often replaced by more precise Runge-Kutta methods [9]. The errors introduced by the Euler approximation may be mitigated, at least to some extent, by deep learning methods.

In this chapter, we will focus on the so-called deep Euler method [26] (further called DEM) that combines the traditional Euler method and deep neural networks. Our approach will use a fully connected neural network trained through supervised learning. Subsequently, we compare the solutions derived from the analytical approach or obtained via the Runge-Kutta method with those generated by DEM and the standard Euler method alone, extending our evaluation beyond the scope of the training data. The experiments will be performed on different examples, Newton's Law of Cooling (Sec. 1.1.1), the SIS model (Sec. 1.1.3), and the SIR model (Sec. 1.1.4), to explore efficiency of the method on both single ODEs and their systems. We will study the results for different values of step h for each of these ODE examples by looking at the solution trajectories and their errors.

3.2 Details of DEM

Let us consider the following initial value problem,

$$\begin{cases} \frac{dy}{dx} = f\left(x, y\right), \ x \in I = [a, b], \\ y\left(a\right) = c, \end{cases} \tag{3.1}$$

where the solution $y\left(x\right) : \ I \to \Omega \subset \mathbb{R}^n$. We introduce the discretization for sampling points in x:

$$a = x_0 \leq x_1 \leq \ldots \leq x_M = b. \tag{3.2}$$

Let $h_m = x_{m+1} - x_m$ be the size of the mesh and y_m be the numerical approximation of $y\left(x_m\right)$. The forward Euler method for initial value problem (3.1) [26] is

$$\begin{cases} y_m = y_{m-1} + h_{m-1} f\left(x_{m-1}, y_{m-1}\right), \ m = 1, \ldots, M, \\ y_0 = c. \end{cases} \tag{3.3}$$

In most cases, for the forward Euler method we choose the uniform mesh with constant step size $h = h_m$, so that $x_m = a + mh$, $m = 0, 1, \dots, M-1$. The truncation error is the difference between the actual value of the function and the truncated value of the given function. It is an error caused by the approximation. Therefore, looking at the equation (3.3), we define the local truncation error as

$$R_m = y\left(x_{m+1}\right) - y\left(x_m\right) - hf\left(x_m, y\left(x_m\right)\right) = \int_{x_m}^{x_{m+1}} f\left(x, y\left(s\right)\right) ds - hf\left(x_m, y\left(x_m\right)\right) \tag{3.4}$$

and the global error as

$$e = |y\left(x_m\right) - y_m|. \tag{3.5}$$

It is known that $R_m = \mathcal{O}\left(h^2\right)$ and $e = \mathcal{O}\left(h\right)$ [19].

The idea is to improve the Euler method step y_{m+1} by reducing the local truncation error and consequently also the global one. To this end, we introduce a fully connected, multilayer, feedforward neural network that infers the local truncation error of the Euler method (see Sec. 1.2 for more details regarding the basics of neural networks). Therefore, its prediction with the results of the Euler method can give better results than the Euler approximation alone. The fully connected multilayer feedforward neural network can approximate any function [20]. Looking at the form of the local truncation error (3.4) we can consider R_m as a continuous function of variables x_m, x_{m+1}, and y_m. Therefore, we will train the neural network so that it approximates $\frac{1}{h_m^2} R_m$.

We can define DEM for the given initial value problem (3.1) as

$$\begin{cases} y_{m+1} = y_m + h_m f\left(x_m, y_m\right) + h_m^2 \mathcal{N}\left(x_m, x_{m+1}, y_m; \theta\right), \ m = 0, \dots, M-1, \\ y_0 = c, \end{cases} \tag{3.6}$$

where $\mathcal{N}\left(x_i, x_j, y_i; \theta\right) : \ \mathbb{R}^{n+1} \to \mathbb{R}^n$ is the nonlinear operator defined by neural network. The parameter θ includes all the weights and biases in the network.

Looking at the formula (3.6), we can see that it consists of two parts: the Euler approximation and the neural network one. The first part expresses the linearity of ODE. However, the use of deep learning allows us to correct the results of the Euler approximation to obtain higher accuracy and express nonlinear behavior. We can think of $\mathcal{N}$ as a parametric function that, through training, learns how to represent the local truncation error. Compared with a neural network that approximates the solution of ODE directly, DEM focuses on the non-linear part of the numerical scheme. This provides a very powerful and flexible method for solving ODEs. We can reduce the error and the constraint of the discretization step size h in the Euler numerical scheme. Because of that, DEM can either improve the accuracy of the Euler method or reduce the computation time.

3.3 Implementation

The neural network used in DEM is a standard multilayer, fully connected neural network, without any additional designs in its architecture [3, 26]. The k^{th} hidden layer has the form

$$L_k\left(z\right) = \mathbf{W}_k z + \mathbf{b}_k, \ 1 \leq k \leq K, \tag{3.7}$$

where $\mathbf{W}_k \in \mathbb{R}^{p_k \times p_{k-1}}$ is the weight matrix, $\mathbf{b}_k \in \mathbb{R}^{p_k}$ is the bias and p_k is the number of neurons in the k^{th} layer. Having the input of the form $x = \left(x_i, x_j, y_i\right) \in \mathbb{R}^{n+2}$ and hidden layers of the form (3.7), the neural network in DEM can be formulated as

$$\mathcal{N}\left(\mathbf{x}; \theta\right) = L_K \circ \sigma \circ L_{K-1} \circ \ldots \circ \sigma \circ L_1\left(\mathbf{x}\right). \tag{3.8}$$

For all of the layers in our architecture we use ReLU activation function (1.11).

For any pair $\{(x_i, z_i), (x_j, z_j)\}$ (where $x_i \leq x_j$), we introduce the local truncation error

$$R(x_i, x_j, z_i, z_j) = \frac{1}{(\Delta x)^2}[z_j - z_i - \Delta x f(x_i, z_i)], \tag{3.9}$$

where $\Delta x = x_j - x_i$. Using formula (3.9) for local truncation error we build the supervised loss for the network,

$$J(\theta) = \frac{2}{N(N-1)} \sum_{1 \leq i,j \leq N} ||\mathcal{N}(x_i, x_j, z_i; \theta) - R(x_i, x_j, z_i, z_j)||_{L^1}, \tag{3.10}$$

where N is the number of training samples. The neural network for DEM through the training learns how to approximate the local truncation error of the Euler method. The loss function (3.10) contains all the loss functions of each trajectory of the ODE system.

For the training, we generate mesh-free data, so that the neural network learns from the data not necessarily located at mesh points. To do so, we use a sample from a uniform distribution and create all possible pairs of (x_i, x_j) from it, where $x_i \leq x_j$.

For all the examples that will be studied in this work for DEM we build neural network with 8 layers, 80 neurons and ReLU activation function (1.11) in each of them. The network is trained for 50 epochs, optimized with the Adam algorithm from the PyTorch library. The learning rate is set to 5×10^{-3}. For all the examples, we average our results over 10 identical separately trained neural networks and study the standard deviation for each point on the mesh to address the consistency of the method over independent runs. Additionally, we look at the absolute error that DEM produces compared to the standard Euler method for different values of step h.

Chapter 4

Results

4.1 Physics-informed Neural Network and its correction

4.1.1 Exploratory analysis of PINN

This section details the experiments performed on the basic and corrected versions of the PINN method, as described in Chapter 3. We first evaluate the performance on Newton's Law of Cooling, followed by the heavily damped oscillator, which is considered a challenging example for traditional numerical methods. Additionally, we examine the method's performance on the ODE system represented by the SIR model.

For Newton's Law of Cooling, we investigate various neural network architectures, focusing on the evolution of the cost function during training and the training time. In both single ODE examples, we analyze the influence of grid size on the performance of the PINN method compared to the standard Runge-Kutta method. Furthermore, we assess the impact of different activation functions.

Next, we will focus on the SIR model as an example of a system of ODEs. We test the PINN method on this model and compare the performance of the basic PINN with the corrected version. Finally, we summarize the experiments by discussing the limitations of the proposed PINN methods.

4.1.2 Newton's Law of Cooling

First, let us consider Newton's Law of Cooling, introduced in Sec. 1.1.1. We generate 11 uniform grid points in the range $[0, 1]$ and train the network using these points by minimizing the cost function (2.3). Predictions are made on the same data points used for training. For the parameters of equation (1.1), we set $T_0 = 0$, $k = 2$ and $A = 10$.

We conducted an experiment to investigate the impact of the number of neurons in a hidden layer on the evolution of the cost function and training time. We used $n_h = 4,\ 17,\ 60,\ 150,\ 200,\ 250,\ 300$ neurons in our simulations. In Fig. 4.1, we observe that for a very small number of neurons in the hidden layer ($n_h = 4,\ 17,\ 60$), we do not obtain the required accuracy in the 1000 epochs. Increasing the number of neurons is beneficial for the performance of the model. The cost function converges to 0 much faster. However, increasing the number of neurons leads to a longer computation time.

Fig. 4.2 shows the convergence of the cost function during training for various network architectures. Here we have a comparison for the network with one and two hidden

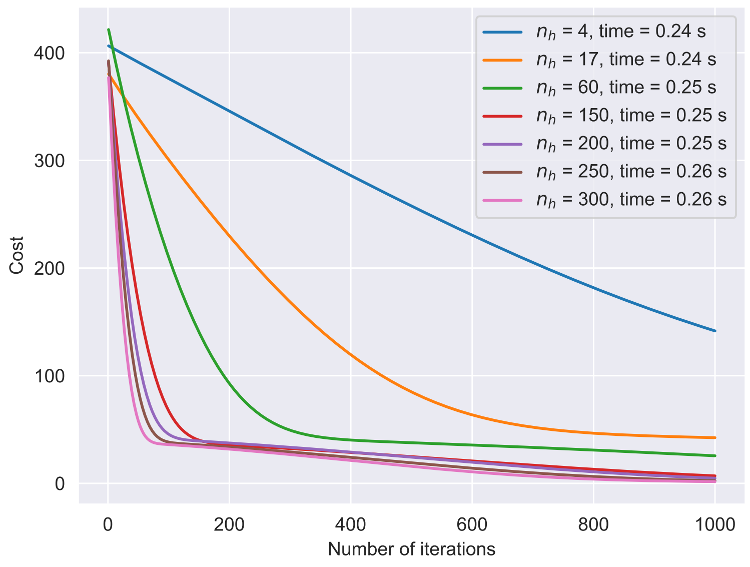

Figure 4.1: Convergence of cost functions during training for neural networks with different number of neurons n_h, in the hidden layer (the Newton's Law of Cooling example).

layers. We can see that networks with 60 neurons in one hidden layer, 4 and 60 neurons in two hidden layers and 60 neurons in two hidden layers significantly outperform the rest of the architectures in terms of convergence of the cost function. After 1000 epochs, these networks have a significantly lower value of the loss function than the others. Again, more complex architectures require longer computation times.

In Fig. 4.3, the PINN results for different numbers of grid points are compared with the analytic solution and the Runge-Kutta method. We looked at 11, 16, 21 and 26 points. The absolute error is also shown in the plots. The trend in the error function is preserved for all numbers of points, both for the PINN and Runge-Kutta method. We can see that in all cases, PINN has a greater absolute error (see Table 4.1 for more details). Interestingly, the highest PINN error value (0.090) was obtained for $m = 16$.

We also study the influence of the choice of the activation function in the hidden layer on the performance of our method. We perform experiments with sigmoid (1.10), ReLU (1.11) and leaky ReLU (1.12) functions. Fig. 4.4 shows the comparison of solutions obtained by the analytic formula and different PINNs. In Fig. 4.5, we can see the absolute errors of these solutions and in the Table 4.2, the mean absolute errors corresponding to different activation functions. All of the mean absolute errors are the same. Looking at the previously mentioned results, we can conclude that in this case changing the activation function did not significantly influence the performance.

4.1.3 Heavily Damped Oscillator

In this part of the work we will explore the heavily damped oscillator example described in Sec. 1.1.2. This equation is an example of a stiff differential equation [10]. These equations have both rapidly changing and slowly changing components, leading to widely different time scales. This disparity can cause significant stability and accuracy issues for numerical solvers.

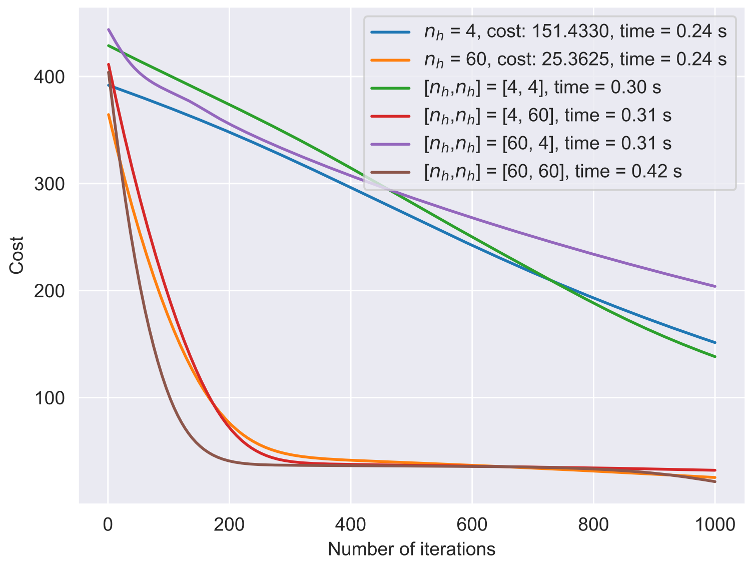

Figure 4.2: Convergence of cost functions during training for neural networks with different architectures (the Newton's Law of Cooling example).

Table 4.1: Mean absolute errors of PINN and Runge-Kutta method for different number of grid points (the Newton's Law of Cooling example).

grid size m	RK4	PINN
11	0	0.083
16	0	0.090
21	0	0.087
26	0	0.086

We generate 11 points on the $[0, 10]$ uniform grid and train the network using them as independent variable values. The whole algorithm that we use was explained in Sec. 2.2. We make predictions on the same data points that we trained the model on. As parameters we use $x_0 = 5$, $m = 1$, $k = 0.5$, $b = 10$, so that the condition (1.4) is met.

In Fig. 4.6, the solutions obtained with different methods for several numbers of points in the grid are compared. The corresponding plots with absolute errors are shown as well. The experiment was carried out for the $m = 11,\ 16,\ 21,\ 26$ grid points. Each simulation of the network shows that the PINN has a lower absolute error for all prediction points. Table 4.3 presents the mean absolute error made by each PINN. For all numbers of points on the grid, we can see that PINN performed better than the standard Runge-Kutta method and has a significantly smaller mean absolute error. The PINN has the highest MAE for $m = 26$ (equal to 0.005).

We also studied the influence of the activation function on the performance of PINN. Fig. 4.7 shows the comparison of solutions for the heavily damped oscillator obtained with sigmoid, ReLU and leaky ReLU activation functions. Fig. 4.8 shows the absolute errors of the corresponding networks. From the Table 4.4, we can conclude that since the absolute error is always the same, the choice of the activation function in one hidden layer is not significant.

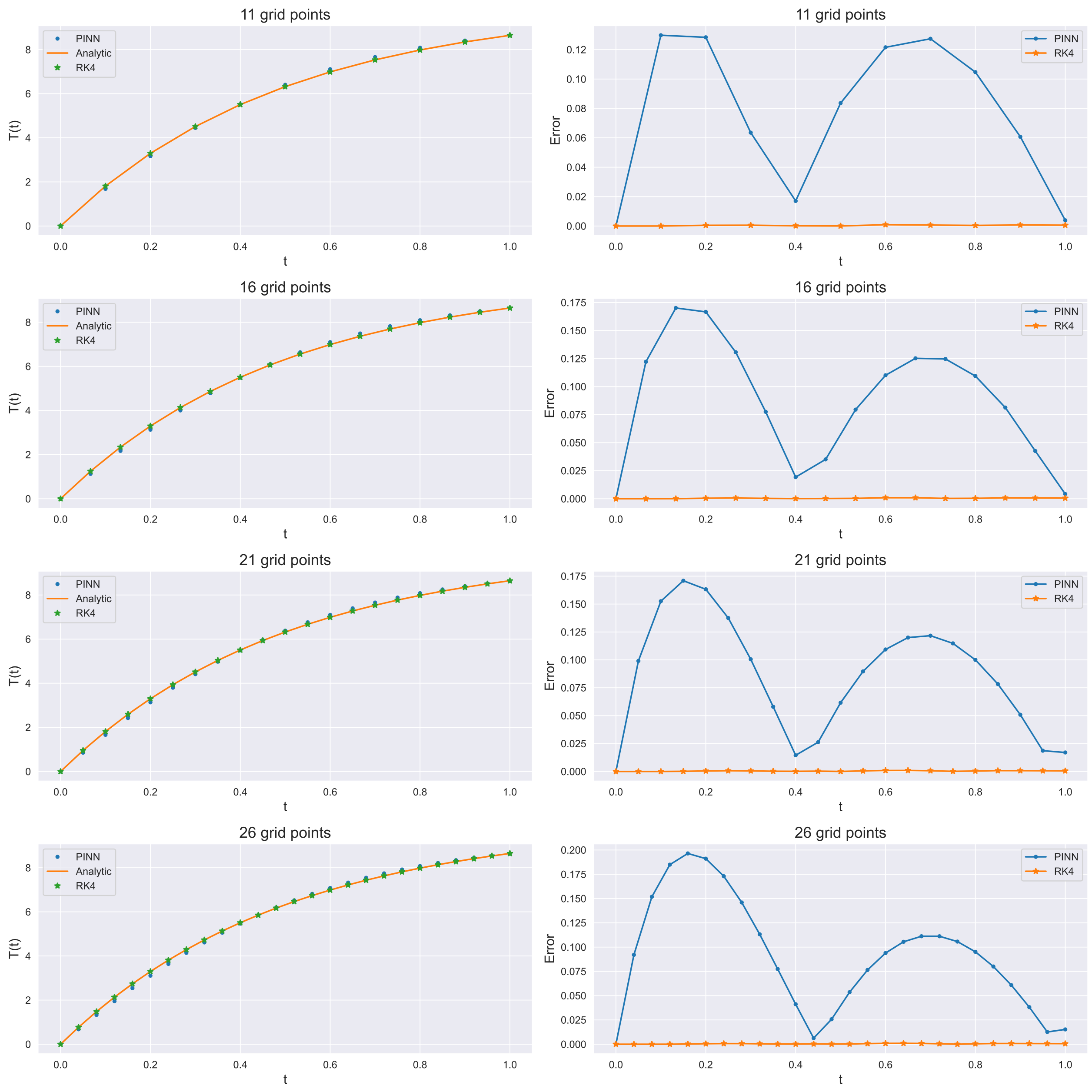

Figure 4.3: Solutions (left column) and their errors (right column) of the Newton's Law of Cooling obtained for different number of grid points. Results generated with PINN are compared with both the analytic solution and the Runge-Kutta method of the fourth order.

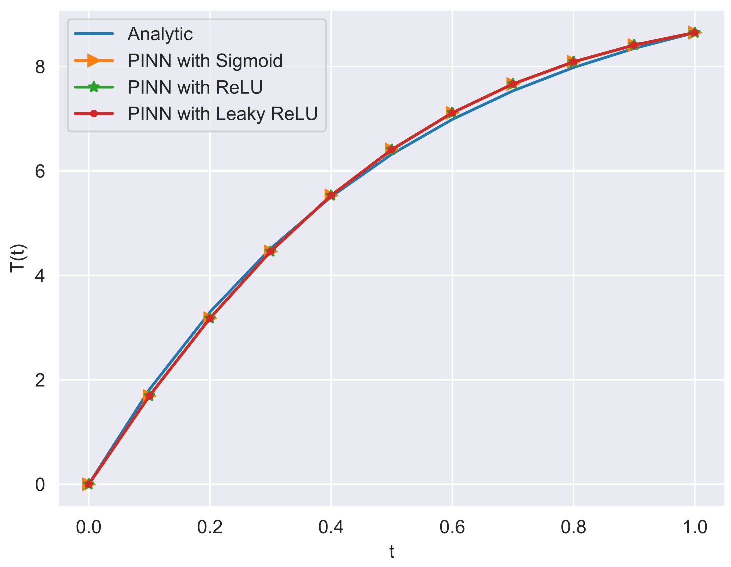

Figure 4.4: PINNs with different activation functions and their solutions for Newton's Law of Cooling.

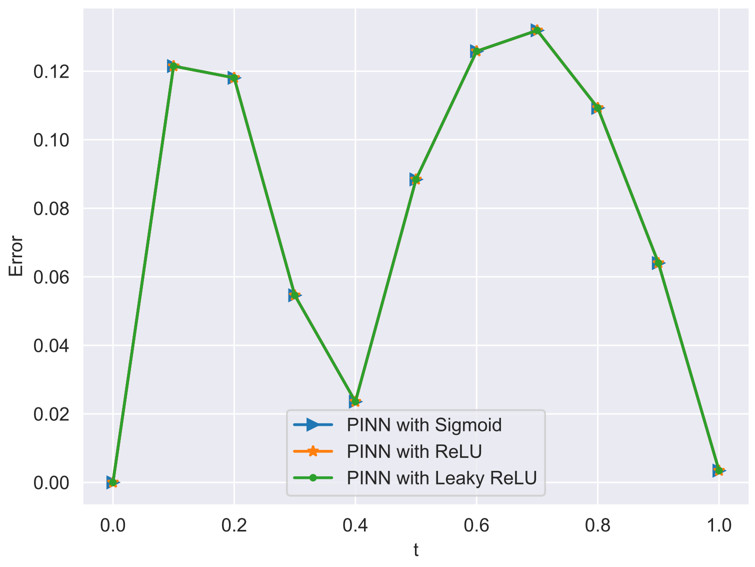

Figure 4.5: Absolute errors made by PINNs with different activation functions (the Newton's Law of Cooling example).

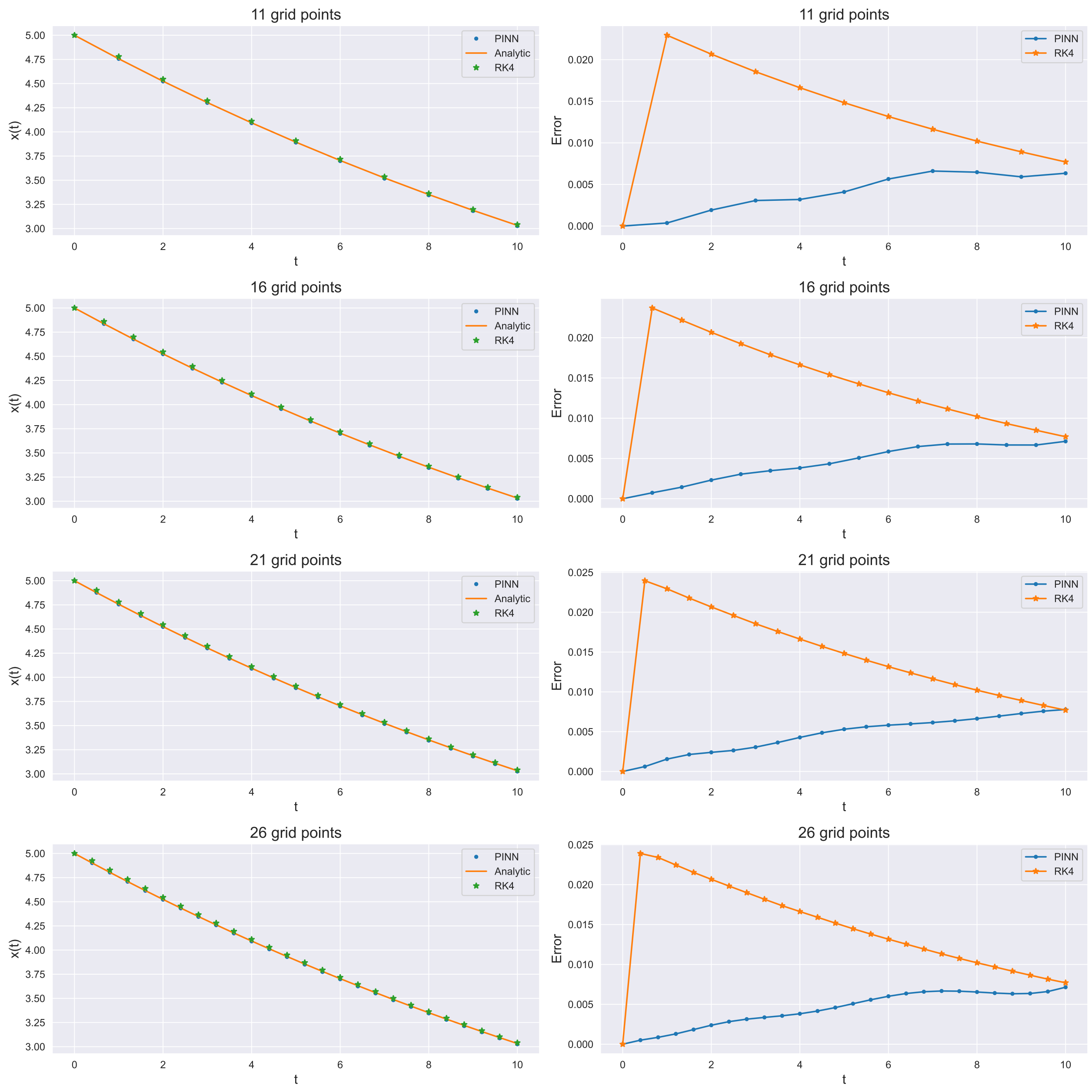

Figure 4.6: Solutions (left column) and their errors (right column) of the heavily damped oscillator obtained for different number of grid points. Results generated with PINN are compared with both the analytic solution and the Runge-Kutta method of the fourth order.

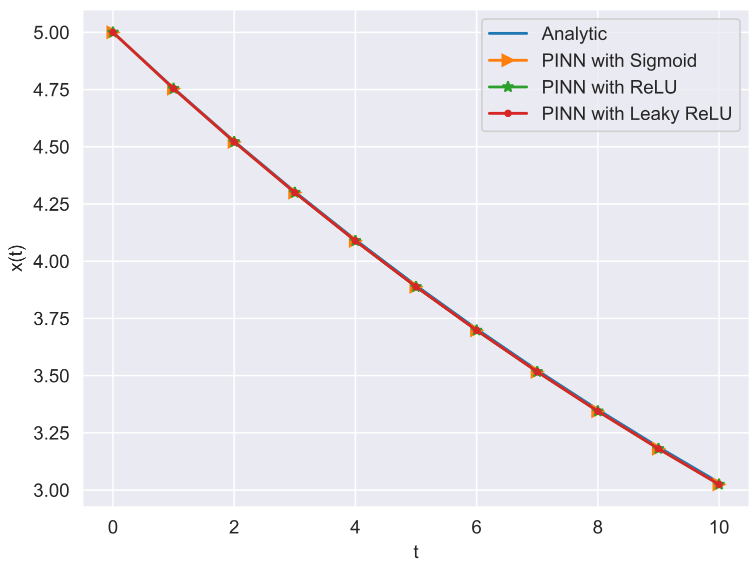

Figure 4.7: PINNs with different activation functions and their solutions of the heavily damped oscillator.

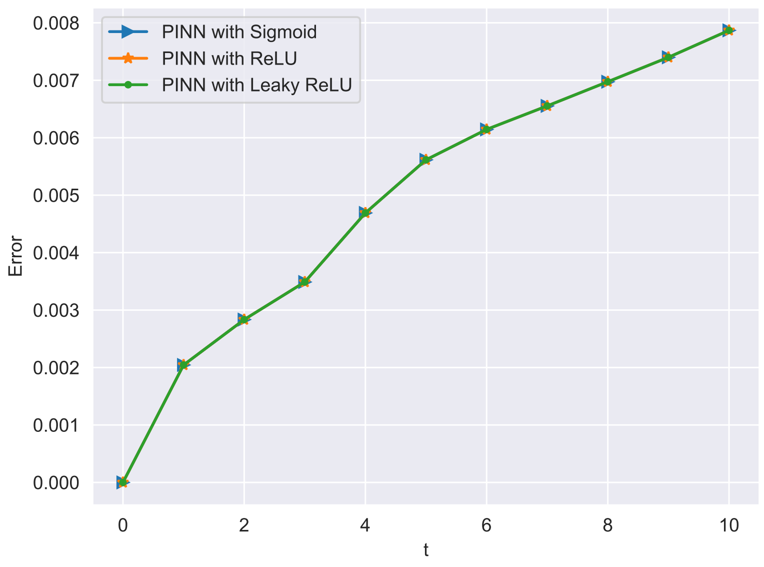

Figure 4.8: Absolute errors made by PINNs with different activation functions (the heavily damped oscillator example).

Table 4.2: Mean absolute errors of PINNs with different activation functions (the Newton's Law of Cooling example).

Sigmoid	ReLU	Leaky ReLU
0.082	0.082	0.082

Table 4.3: Mean absolute errors of PINN and Runge-Kutta method for different number of grid points (the heavily damped oscillator example).

grid size m	RK4	PINN
11	0.013	0.004
16	0.014	0.004
21	0.014	0.004
26	0.014	0.005

4.1.4 SIR model

As the last example for the standard version of PINN we will consider the SIR epidemiological model described in Sec. 1.1.4. We generate 1000 points on the $[0, 60]$ uniform grid and train the network by minimizing the loss function (2.3) that contains the loss for each equation. We make predictions on the same data points that we were training the PINN on. As parameters for the SIR model we choose $S_0 = 990$, $I_0 = 20$, $R_0 = 0$, $\beta = 0.001$ and $\gamma = 0.1$.

In Fig. 4.9, we can see the comparison between PINN and Runge-Kutta methods. PINN performed very poorly in terms of capturing the behavior of the trajectories. Not only did the method not follow the details of the trajectories, but it has a great problem with capturing even the trend. We extended our experiments by using more complex network architectures for this problem, but none of the attempts gave satisfying results. It is worth mentioning that this example is the first one in which we consider not a single ODE but a system of three ODEs. This system is significantly more complex; therefore, the network struggles to capture the physical constraints.

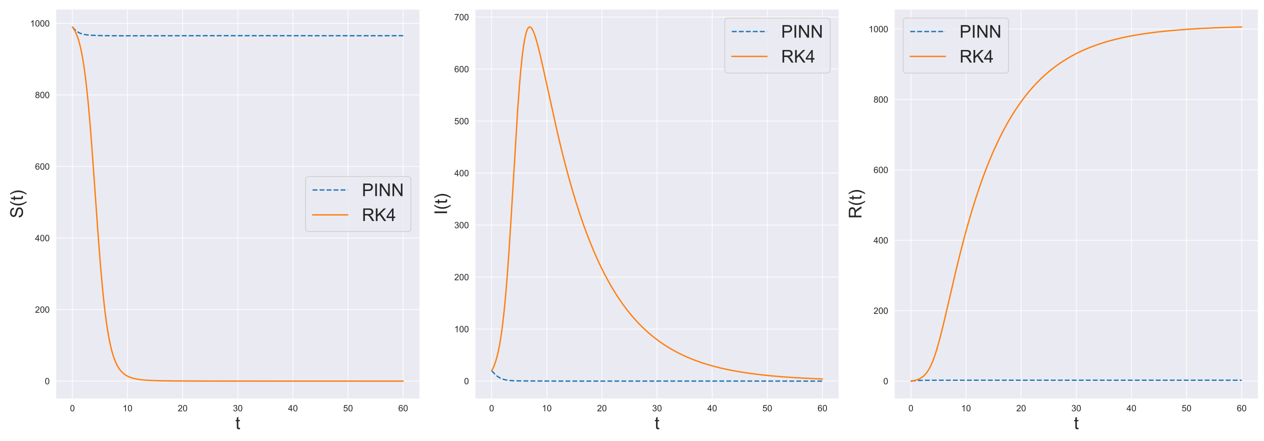

Figure 4.9: Solutions of SIR model obtained with PINN and Runge-Kutta methods. From the left to the right, the numbers of susceptible, infected and recovered individuals.

Table 4.4: Mean absolute errors of PINNs with different activation functions (the heavily damped oscillator example).

Sigmoid	ReLU	Leaky ReLU
0.004	0.004	0.004

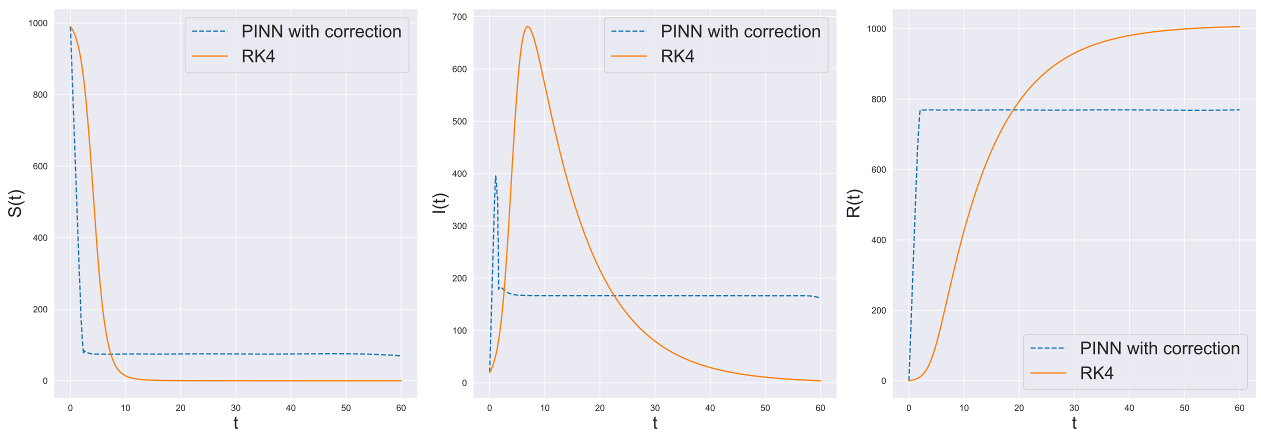

Figure 4.10: Solutions of SIR model obtained with corrected PINN and Runge-Kutta methods. See Sec. 2.3 for details.

4.1.5 SIR model with a „corrected” PINN

Since the standard version of PINN did not perform sufficiently well, in this part, we will apply a correction to PINN. This method was proposed and explained in Sec. 2.3. The number of points in the uniform grid and all the parameters of the SIR model system (1.7) are the same as in the example in Sec. 4.1.4.

In Fig. 4.10 we can see that the trajectories obtained by PINN with correction still do not follow the Runge-Kutta solution sufficiently well. However, compared to the standard PINN method, we can see that the one with correction significantly better captures the trends of trajectories.

4.1.6 Limitations of the PINN method

PINN's main idea is to capture the behavior of the ODE system in an unsupervised manner. As we presented in the examples for chosen single ODEs, it performs very well. For the one that is challenging for standard numerical methods, it outperformed the Runge-Kutta method. However, poor performance in the SIR model example shows that the method cannot be assumed to be universal and applicable to all ODE systems.

We also presented a correction to the standard PINN method. We incorporated previous knowledge on the exact solution to the loss function (2.6) of the network. This approach is supervised, because we train the neural network based on knowledge of the expected output. The proposed correction improved performance, but the trajectories were still not sufficiently close to the exact ones.

The performance of both PINN and PINN with correction for the SIR model highlights the difficulties that deep learning methods might face in solving ODE systems. It also indicates the need to adjust the method for each example separately.

4.1.7 Conclusions

The physics-informed neural network is a method that is supposed to directly capture the behavior of the ODE system and learn in an unsupervised way through knowledge only from the right-hand side of the ODE system (2.1).

First, we tested the method on single ODEs, including Newton's Law of Cooling, and a heavily damped oscillator. In both examples, the standard PINN captured the behavior of the physical system sufficiently well. The mean absolute error for Newton's Law of Cooling example shows that in this case the Runge-Kutta method performed better than PINN. However, the error of PINN was acceptable.

The heavily damped oscillator is an example of a physical system that is difficult to solve using standard numerical methods. In this case, we can see the significant advantage of the PINN method that for all experiments significantly outperformed the Runge-Kutta numerical method.

In addition, for both single ODE examples, we studied the influence of the choice of the activation function on the performance of the method. In both cases, the mean absolute error was the same for all tested activation functions. Hence, that the choice does not significantly affect the performance of the network.

We also studied the performance of the method in the case of a system of ODEs. We chose the popular SIR model as an example. The unsupervised network did not adequately capture the behavior of the physical system. Therefore, we implemented a correction to the standard PINN method by adding a new source of knowledge of the system: the exact solution in the grid points. The correction improved the performance of the network and the trajectory followed the exact solution more accurately. However, this method still did not give us satisfactory results.

The study of the performance of the PINN method in examples with different complexity highlights the need for more research towards a universal physical-informed neural network.

4.2 Deep Euler method

4.2.1 Exploratory analysis of DEM

In this part of the work we present an analysis of DEM (see Chapter 3). We test the method on the already known examples. Additionally, we will look at the SIS model of epidemics. All examples have been introduced in Chapter 1. The performance of DEM will be compared with that of the standard Euler method and the analytic solution (if exists) or a solution from the Runge-Kutta method (otherwise). We will calculate the mean absolute error of each trajectory for different time steps h to study the impact of the step size on DEM. Furthermore, since we averaged the results of the network over 10 independent simulations, we study the standard deviations of the network output to address the stability of the method. All experiments were performed in the architecture described in the Sec. 3.3.

4.2.2 Newton's Law of Cooling

Let us consider Newton's Law of Cooling in the form (1.1). The analytic solution is given by Eq. (1.2). We generate 100 random data points $\{(t_i, y(t_i))\}_{i=1}^{100}$, sampling from a uniform distribution $\mathcal{F}_t = U(0,2)$ and use them to build the training set. Then we train the deep

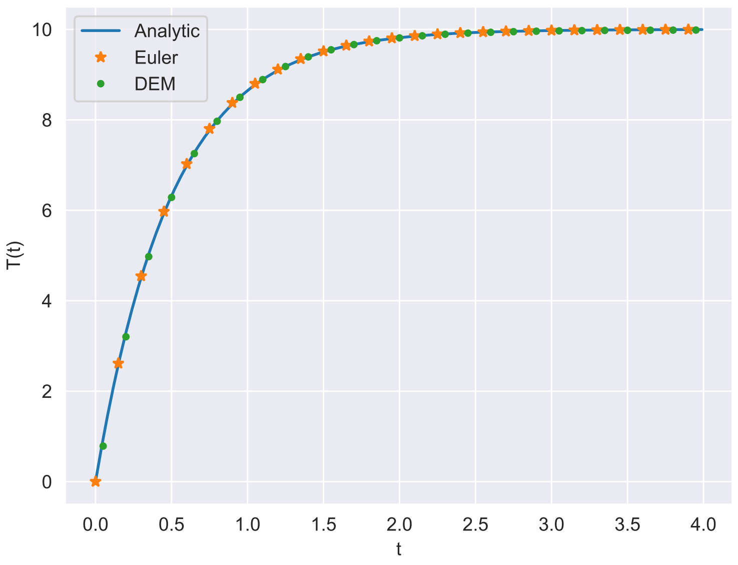

Figure 4.11: The DEM results for Newton's Law of Cooling compared with the exact solution and the standard Euler method. Step size $h = 0.01$.

neural network $\mathcal{N}$ by minimizing the loss function (3.10). We make the predictions on a test set wider than the training set and sampled from a uniform mesh with constant step size. We use $A = 10,\ k = 2,\ T_0 = 0$ as parameters, and for prediction we use the time step $h = 0.01$.

In Fig. 4.11, DEM results are compared with the analytic solution and the standard Euler approximation. Predictions are made in the range $(0, 4)$. From Fig. 4.12 it follows that at the very beginning of the trajectory DEM yields a significantly bigger error, but then it is similar to the error of Euler approximation. In Table 4.5 we can see that the mean absolute error made by DEM equals here 0.017 and the one made by Euler method is 0.013, so in this case the standard Euler method performs better.

The performance of DEM for different time steps is summarized in the Table 4.5. For small time steps ($h = 0.001,\ 0.01,\ 0.1$) Euler performs slightly better than DEM. However, for bigger ones ($h = 1,\ 2,\ 3$) we can see that DEM significantly outperforms the standard Euler method, since the latter needs very small discretization step size to obtain meaningful results.

As mentioned before, we built, trained, and predicted the trajectory 10 times. In the boxplot 4.13 we can see standard deviations for each time step for which we made the prediction. Despite some variation, all the values in the boxplot, ranging from 0 to 0.023, are close to 0. This narrow range indicates that the data points are consistently low and tightly placed around a value near 0.

4.2.3 SIS model

Now we will study an example of a system of two ODEs—the SIS epidemiological model described in Sec. 1.1.3. We generate 1000 random data points $\{(t_i, y(t_i))\}_{i=1}^{100}$, by sampling from a uniform distribution $\mathcal{F}_t = U(0, 20)$ and use them to build the training set. We use $S_0 = 990,\ I_0 = 10,\ \beta = 0.001,\ \gamma = 0.1$ as parameters, and for prediction we use the time

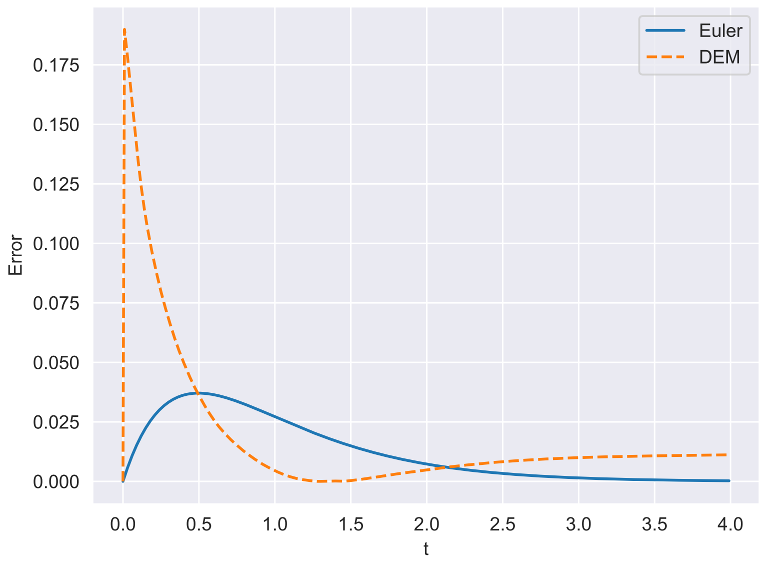

Figure 4.12: The absolute error of the deep and the standard Euler methods in the Newton's Law of Cooling example. Step size $h = 0.01$.

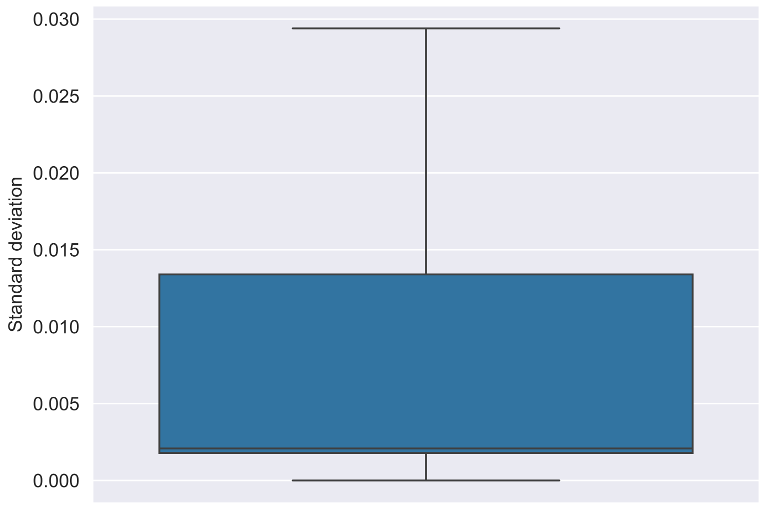

Figure 4.13: Standard deviation of DEM approximation for each point on the mesh for step size $h = 0.01$ (the Newton's Law of Cooling example).

Table 4.5: Mean absolute errors of DEM and the standard Euler method for different time steps h (the Newton's Law of Cooling example).

step size h	DEM	Euler
0.001	0.002	0.001
0.01	0.017	0.013
0.1	0.167	0.129
1	5.302	7.799
2	9.225	15.092
3	18.268	25.012

step $h = 0.01$.

In Fig. 4.14, we see the trajectories produced DEM, the Euler approximation, and the Runge-Kutta method. Predictions are made in the range $(0, 25)$. From Fig. 4.15 we can conclude that both approximations produce a very similar error. The error for the trajectory S and I increases significantly at the beginning of the range and decreases to a value close to 0 around $t = 10$. In case of both methods, we compare their performance with the solution obtained from the Runge-Kutta numerical method. From Tables 4.7 and 4.6 we can see that the mean absolute error of DEM for S is equal to 0.562 and for I is equal to 0.558. Error made with the standard Euler method totals 0.545 for S and 0.545 for I. The inaccuracy of DEM is somewhat higher for both trajectories.

The mean absolute error for both trajectories as a function of time step h is shown in Tables 4.6 and 4.7. We can see that the error increases with h for both of the methods. However, in this example, the Euler approximation has errors smaller than DEM for each value of h in both trajectories.

Since here we also built, trained the neural network and predicted the trajectory independently 10 times, we can look at boxplots of standard deviations for each time step for which the predictions were made. From Fig. 4.16 it follows that in the case of both trajectories all values in boxplots are close to 0. Therefore, the method is stable and consistent in its results through independent runs.

Table 4.6: Mean absolute errors of DEM and the Euler method for different time steps h for S trajectory in the SIS model.

step size h	DEM	Euler
0.001	0.138	0.136
0.01	0.562	0.545
0.1	4.785	4.627
1	44.566	43.206
2	84.898	81.725
3	262.916	259.205

4.2.4 SIR model

Finally, we will test DEM on the SIR epidemiological model described in Sec. 1.1.4. In this part, we also generate 1000 random data points $\{(t_i, y(t_i))\}_{i=1}^{100}$, by sampling from a uniform distribution $\mathcal{F}_t = U(0, 50)$ to build the training set, as mentioned before.

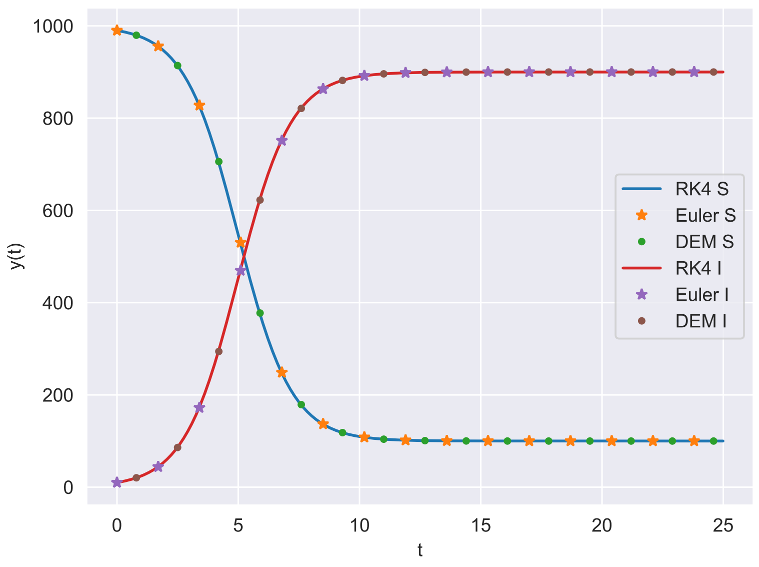

Figure 4.14: The DEM results for the SIS model compared with the Runge-Kutta solution and the standard Euler approximation. Step size $h = 0.01$.

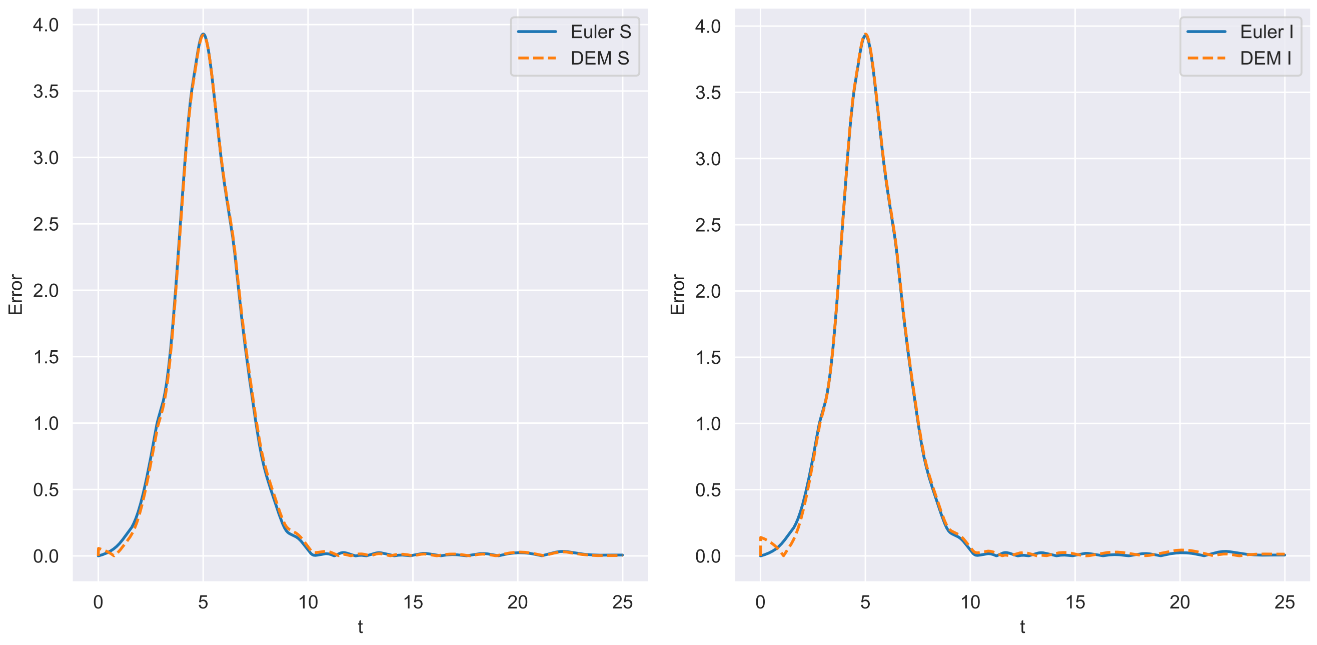

Figure 4.15: The absolute error of both the deep and the standard Euler methods for the number of susceptible individuals (left) and infected ones (right). Step size $h = 0.01$.

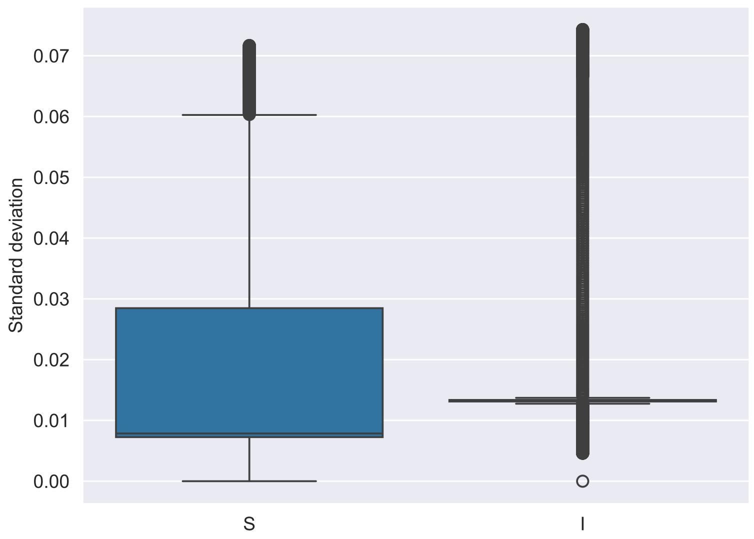

Figure 4.16: Standard deviation of DEM approximation for each point on the mesh for step size $h = 0.01$ (the SIS model example).

Table 4.7: Mean absolute errors of DEM and the Euler method for different time steps h for I trajectory in the SIS model.

step size h	DEM	Euler
0.001	0.137	0.136
0.01	0.558	0.545
0.1	4.803	4.627
1	44.622	43.206
2	82.826	81.725
3	260.941	259.205

We use $S_0 = 990$, $I_0 = 20$, $R_0 = 0$, $\beta = 0.001$, $\gamma = 0.1$ as parameters of the model. For predictions, we choose the time step $h = 0.01$.

In Fig. 4.17, we see the trajectories of the SIR model produced with DEM, the Euler approximation, and the Runge-Kutta method. Predictions are made in the range $(0, 60)$. The corresponding errors are shown in Fig. 4.18. For all trajectories the error of DEM and Euler behaves likewise. It increases significantly at the beginning of the range and decreases with time to a value around 0. From Table 4.8, 4.9 and 4.10 we can see that here the mean absolute error made by DEM for S equals 0.186, for I equals 0.247, and for R equals 0.190. The mean absolute error made by the Euler method for S equals 0.185, for I equals 0.245, and for R equals 0.190. Thus, the mean absolute errors are almost the same for DEM and Euler method.

From Tables 4.8, 4.9 and 4.10 we can also deduce how increasing the time step h influences the mean absolute error. For the S, I and R trajectory, the MAE values from DEM and Euler are very close. The value of errors increases for both DEM and Euler with the time step h. For the S trajectory, the mean absolute error of DEM for $h = 0.001,\ 0.01$ is slightly greater than that of the Euler method. For the I trajectory the MAE

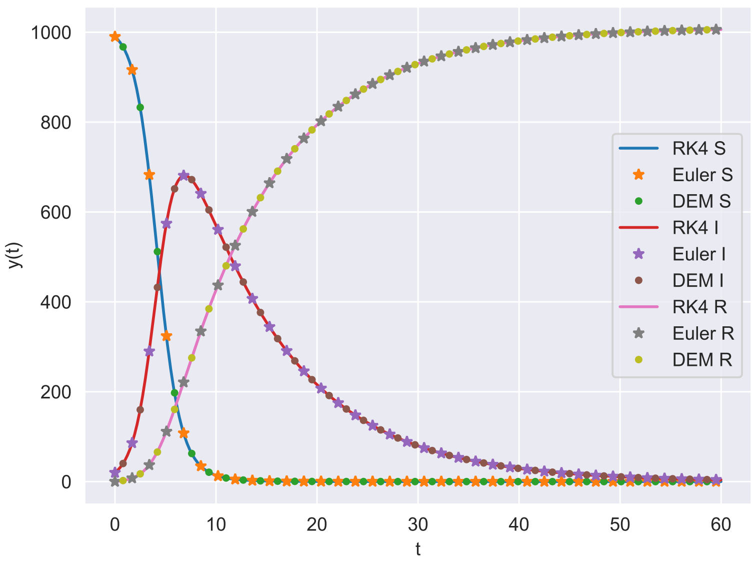

Figure 4.17: The DEM results for the SIR model compared with the Runge-Kutta solution and the standard Euler approximation. Step size $h = 0.01$.

is marginally greater for DEM for $h = 0.01,\ 0.1,\ 1,\ 2$ and for the Euler approximation for $h = 3$. The trajectory R has a narrowly greater error in the case of the DEM for $h = 0.1,\ 3$ and in case of the Euler method for $h = 1,\ 2$. In all of the examples, the differences in the errors of two methods are very small.

We also studied the stability of the method by looking at the standard deviation for each time step for which the predictions were made. From Fig. 4.19 it follows that for all three trajectories the values are close to 0, i.e. DEM makes consistent predictions.

Table 4.8: Mean absolute errors of DEM and the Euler method for different time steps h for S trajectory in the SIR model.

step size h	DEM	Euler
0.001	0.056	0.055
0.01	0.186	0.185
0.1	1.567	1.567
1	15.059	15.059
2	33.219	33.219
3	95.602	95.602

4.2.5 Conclusions

DEM is a deep learning method that is based on the standard Euler one. It focuses on approximating the local truncation error and by combining its output with standard Euler making a better approximation of the ODE trajectory.

From our experiments, we can conclude that DEM approximates the ODEs sufficiently well. All predictions made were averaged over 10 independent runs. We also studied

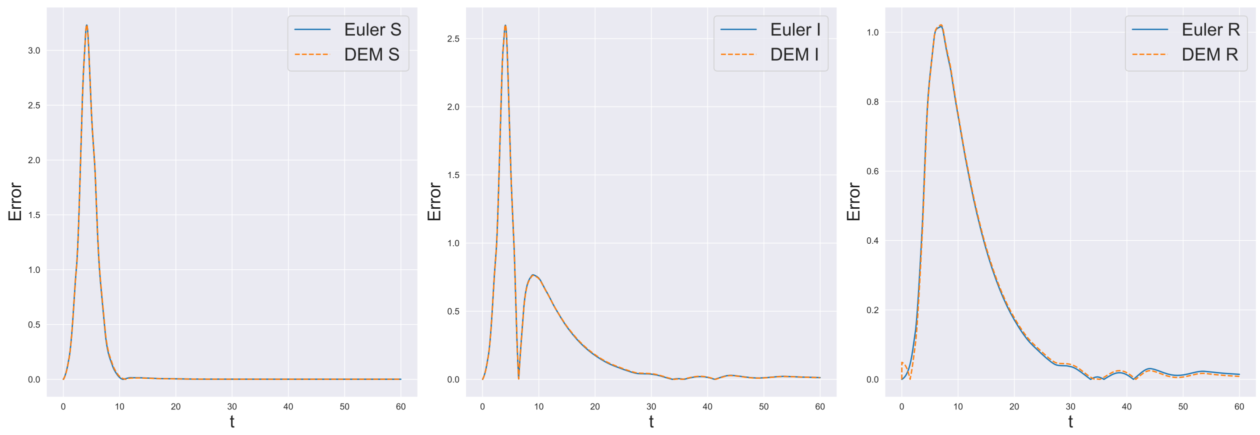

Figure 4.18: The absolute error of both the deep and the standard Euler methods for the number of susceptible individuals (left), infected individuals (center) and recovered ones (right). Step size $h = 0.01$.

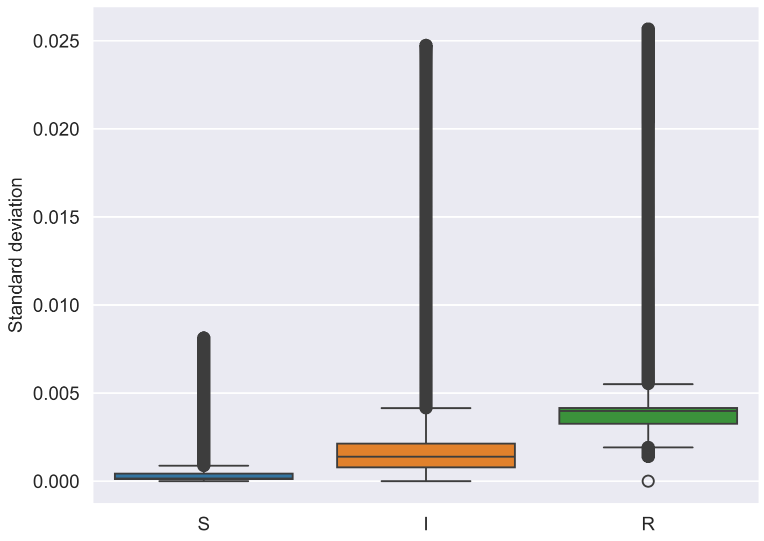

Figure 4.19: Standard deviation of DEM approximation for each point on the mesh for step size $h = 0.01$ (the SIR model example)..

Table 4.9: Mean absolute errors of DEM and the Euler method for different time steps h for I trajectory in the SIR model.

step size h	DEM	Euler
0.001	0.070	0.070
0.01	0.247	0.245
0.1	2.142	2.124
1	21.242	21.114
2	41.756	41.537
3	95.287	95.303

Table 4.10: Mean absolute errors of DEM and the Euler method for different time steps h for R trajectory in the SIR model.

step size h	DEM	Euler
0.001	0.059	0.059
0.01	0.190	0.190
0.1	1.541	1.534
1	14.668	14.685
2	29.194	29.357
3	73.611	74.398

the standard deviations of each predicted point and concluded that the method is stable and consistent in its predictions. Looking at the mean absolute error made by the DEM and the Euler method for different time steps h we found, as expected, that for longer time steps both methods have a bigger error. For a single ODE, for smaller step sizes, all errors were slightly larger for DEM. However, starting from $h = 1$, we had a larger error for the Euler method. For these time steps, the difference in errors between methods was significantly greater, indicating superiority of DEM over the standard Euler method. However, in the case of SIS model in both trajectories for larger time steps, Euler had a slightly smaller mean absolute error than DEM. In the last example, SIR model, the tendency differs among trajectories. For the S trajectory, the MAE is slightly higher for DEM with $h = 0.001,\ 0.01$. For the I trajectory starting from $h = 0.01$, DEM has a subtly larger MAE, except for the largest $h = 3$, where it slightly outperforms Euler. For the R trajectory, the DEM has a larger error for $h = 0.1$. However, for higher step sizes, DEM outperforms the standard Euler method.

Summary

In this work, we investigated the use of deep learning methods for solving systems of ordinary differential equations. We focused on two main approaches: physics-informed neural networks, which directly approximate trajectories through unsupervised learning, and the Deep Euler method, which enhances the standard Euler method by predicting its local truncation error, thereby increasing its accuracy via supervised learning. We evaluated the performance of both methods using examples from physical systems. Additionally, for the PINN approach, we proposed a correction specific to a given example.

Our study reveals that both methods have their respective advantages and disadvantages. The work on PINNs emphasizes the need for further research, particularly concerning the model's performance on more complex ODE systems, and highlights the necessity of tailoring the method to each specific system. The Deep Euler method, on the other hand, performed well across all examples. Future research could extend the current architecture so that it would significantly outperform the standard Euler method or apply the concept of predicting local truncation error to other well-established numerical methods.

Bibliography

[1] Bre, F., Gimenez, J. M., Fachinotti, V. D. Prediction of wind pressure coefficients on building surfaces using artificial neural networks. *Energy and Buildings 158* (2018), 1429–1441.

[2] Buscema, M. Back propagation neural networks. *Substance use & misuse 33*, 2 (1998), 233–270.

[3] Choi, R. Y., Coyner, A. S., Kalpathy-Cramer, J., Chiang, M. F., Campbell, J. P. Introduction to machine learning, neural networks, and deep learning. *Translational vision science & technology 9*, 2 (2020), 14–14.

[4] Coddington, E. A., Levinson, N., Teichmann, T. Theory of ordinary differential equations, 1956.

[5] Dong, S., Wang, P., Abbas, K. A survey on deep learning and its applications. *Computer Science Review 40* (2021), 100379.

[6] Dongare, A., Kharde, R., Kachare, A. D., et al. Introduction to artificial neural network. *International Journal of Engineering and Innovative Technology (IJEIT) 2*, 1 (2012), 189–194.

[7] Du, S., Lee, J., Li, H., Wang, L., Zhai, X. Gradient descent finds global minima of deep neural networks. In *Proceedings of the 36th International Conference on Machine Learning* (09–15 Jun 2019), K. Chaudhuri and R. Salakhutdinov, Eds., vol. 97 of *Proceedings of Machine Learning Research*, PMLR, pp. 1675–1685.

[8] Dufera, T. T. Deep neural network for system of ordinary differential equations: Vectorized algorithm and simulation. *Machine Learning with Applications 5* (2021), 100058.

[9] Epperson, J. F. *An introduction to numerical methods and analysis.* John Wiley & Sons, 2013.

[10] Garfinkel, D., Marbach, C. B., Shapiro, N. Z. Stiff differential equations. *Annual review of biophysics and bioengineering 6*, 1 (1977), 525–542.

[11] Grimm, V., Heinlein, A., Klawonn, A., Lanser, M., Weber, J. Estimating the time-dependent contact rate of sir and seir models in mathematical epidemiology using physics-informed neural networks. *Electron. Trans. Numer. Anal 56* (2022), 1–27.

[12] Gupta, J., Jayaprakash, B., Eagon, M., Selvam, H. P., Molnar, C., Northrop, W., Shekhar, S., et al. A survey on solving and discovering differential equations using deep neural networks. *arXiv preprint arXiv:2304.13807* (2023).

[13] Han, S., Stelz, L., Stoecker, H., Wang, L., Zhou, K. Approaching epidemiological dynamics of covid-19 with physics-informed neural networks. *Journal of the Franklin Institute* (2024), 106671.

[14] Hethcote, H. W. Three basic epidemiological models. In *Applied mathematical ecology*. Springer, 1989, pp. 119–144.

[15] Irving, J., Mullineux, N. *Mathematics in physics and engineering*, vol. 6. Academic Press, 2013.

[16] Joshi, A. V. Perceptron and neural networks. In *Machine learning and artificial intelligence*. Springer, 2022, pp. 57–72.

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

[18] Lagaris, I., Likas, A., Fotiadis, D. Artificial neural networks for solving ordinary and partial differential equations. *IEEE Transactions on Neural Networks 9*, 5 (1998), 987–1000.

[19] Larsson, S., Thomée, V. *Partial differential equations with numerical methods*, vol. 45. Springer, 2003.

[20] Leshno, M., Lin, V. Y., Pinkus, A., Schocken, S. Multilayer feedforward networks with a nonpolynomial activation function can approximate any function. *Neural networks 6*, 6 (1993), 861–867.

[21] Malek, A., Beidokhti, R. S. Numerical solution for high order differential equations using a hybrid neural network—optimization method. *Applied Mathematics and Computation 183*, 1 (2006), 260–271.

[22] Morin, D. *Introduction to classical mechanics: with problems and solutions*. Cambridge University Press, 2008.

[23] O'Sullivan, C. T. Newton's law of cooling—a critical assessment. *American Journal of Physics 58*, 10 (1990), 956–960.

[24] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., Chintala, S. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 2019, pp. 8024–8035.

[25] Rasamoelina, A. D., Adjailia, F., Sinčák, P. A review of activation function for artificial neural network. In *2020 IEEE 18th World Symposium on Applied Machine Intelligence and Informatics (SAMI)* (2020), IEEE, pp. 281–286.

[26] Shen, X., Cheng, X., Liang, K. Deep euler method: solving odes by approximating the local truncation error of the euler method. *arXiv:2003.09573* (2020).

[27] Zou, Z., Meng, X., Karniadakis, G. E. Correcting model misspecification in physics-informed neural networks (pinns). *Journal of Computational Physics 505* (2024), 112918.