algorithms

MDPI

Article

Optimizing Physics-Informed Neural Network in Dynamic System Simulation and Learning of Parameters

Ebenezer O. Oluwasakin *,† **and Abdul Q. M. Khaliq** *,†

Department of Mathematical Sciences, Middle Tennessee State University, Murfreesboro, TN 37132, USA
* Correspondence: eo3j@mtmail.mtsu.edu (E.O.O.); abdul.khaliq@mtsu.edu (A.Q.M.K.)
† These authors contributed equally to this work.

Abstract: Artificial neural networks have changed many fields by giving scientists a strong way to model complex phenomena. They are also becoming increasingly useful for solving various difficult scientific problems. Still, people keep trying to find faster and more accurate ways to simulate dynamic systems. This research explores the transformative capabilities of physics-informed neural networks, a specialized subset of artificial neural networks, in modeling complex dynamical systems with enhanced speed and accuracy. These networks incorporate known physical laws into the learning process, ensuring predictions remain consistent with fundamental principles, which is crucial when dealing with scientific phenomena. This study focuses on optimizing the application of this specialized network for simultaneous system dynamics simulations and learning time-varying parameters, particularly when the number of unknowns in the system matches the number of undetermined parameters. Additionally, we explore scenarios with a mismatch between parameters and equations, optimizing network architecture to enhance convergence speed, computational efficiency, and accuracy in learning the time-varying parameter. Our approach enhances the algorithm's performance and accuracy, ensuring optimal use of computational resources and yielding more precise results. Extensive experiments are conducted on four different dynamical systems: first-order irreversible chain reactions, biomass transfer, the Brusselsator model, and the Lotka-Volterra model, using synthetically generated data to validate our approach. Additionally, we apply our method to the susceptible-infected-recovered model, utilizing real-world COVID-19 data to learn the time-varying parameters of the pandemic's spread. A comprehensive comparison between the performance of our approach and fully connected deep neural networks is presented, evaluating both accuracy and computational efficiency in parameter identification and system dynamics capture. The results demonstrate that the physics-informed neural networks outperform fully connected deep neural networks in performance, especially with increased network depth, making them ideal for real-time complex system modeling. This underscores the physics-informed neural network's effectiveness in scientific modeling in scenarios with balanced unknowns and parameters. Furthermore, it provides a fast, accurate, and efficient alternative for analyzing dynamic systems.

Keywords: artificial neural networks; physics-informed neural networks; time-varying parameter; dynamical system modeling; COVID-19 Data; deep neural networks; error metrics

Citation: Oluwasakin, E.O.; Khaliq, A.Q.M. Optimizing Physics-Informed Neural Network in Dynamic System Simulation and Learning of Parameters. *Algorithms* **2023**, *16*, 547. https://doi.org/10.3390/a16120547

Academic Editors: Luca Mariot and Luca Manzoni

Received: 1 November 2023
Revised: 24 November 2023
Accepted: 26 November 2023
Published: 28 November 2023

1. Introduction

In many scientific and engineering fields, understanding and predicting a wide range of complex events is all about understanding and predicting dynamical systems. These systems often employ ordinary differential equations (ODEs) to describe and model the intricate relationships inherent in various natural and artificial processes. One such common usage is seen in compartmental models, extensively used across fields such as ecology [1], epidemiology [2], chemical engineering [3], mathematical biology [4], and economics. These models require careful estimation of numerous unknown parameters for their validity, often derived from data collected from experiments or real-world observations. This

process, termed calibration of the dynamical system, constitutes a challenging optimization problem due to its iterative nature and potential convergence issues. It is also important to keep the gap between observed data and model forecasts as small as possible. This makes the accurate estimation of parameters a key step in developing models. Moreover, the need for dynamical system models with accurate parameter estimates in chemical process engineering has surged due to process optimization and control technology advancements [5]. However, the difficulties associated with parameter estimation become compounded in models, such as ODEs or differential-algebraic equations (DAEs), that exhibit non-linearity in parameters. While many system identification techniques such as the Gauss-Newton method [6], multiple shooting, recursive estimation [7], maximum likelihood estimation, Markov chain Monte Carlo-based Bayesian inference [8,9], finite element methods [10], and collocation methods [11] are available for learning parameters and solving dynamical systems, each has limitations, including high computational costs, reliance on initial guesses, and limitations with time-varying parameters. Despite the challenges, the role of parameter estimation in model accuracy and predictability remains paramount, calling for continued innovation and research in this fundamental mathematical field.

Artificial intelligence (AI), particularly the implementation of deep neural networks (DNNs), has experienced a significant increase in applications for analyzing and interpreting complex systems [12,13]. This is especially true for dynamical systems, which present complex interactions requiring sophisticated analytic tools. Although these AI models have demonstrated remarkable efficacy in data fitting and short-term prediction tasks, their implementation may need to be improved due to certain obstacles. One major barrier is that it takes work for AI models to understand the underlying structures of a dynamic system's activities. The resulting AI model predictions and real system behaviors may be at odds with one another. Additionally, the performance of these models is heavily reliant on the quantity and quality of input data, potentially resulting in inadequate insights when the data fails to portray the system's dynamics. Artificial neural networks (ANNs) have gained popularity as an AI model because of their potential to overcome common problems in the field, such as the necessity of massive training datasets and high processing costs. It has been proposed that ANNs can be used to pioneering effect in estimating parameters in systems of differential equations. The advent of ANNs traces back to the 1940s, primarily attributed to the research article by McCulloch and Pitts [14]. However, it is only in recent years that ANNs and DNNs have seen widespread acceptance and use, owing to significant advancements in computational capabilities and data storage. DNNs, essentially ANNs with multiple hidden layers, have seen a surge in application across different sectors, including computer vision, image processing [15], pattern recognition [16], and cybersecurity [17]. The architecture of DNNs allows them to capture more variance, contributing to their success.

The effectiveness of compartmental models, which are often described by systems of ordinary differential equations, and the effectiveness of AI models in understanding and predicting the behaviors of dynamical systems have been demonstrated. However, they are not without their weaknesses. Researchers are increasingly gravitating toward combining compartmental and AI models to improve their abilities to analyze and predict the behavior of dynamical systems. The recent emergence of physics-informed neural networks (PINNs) has demonstrated a promising stride in integrating differential equations into neural networks [18]. This allows the networks to fit the data accurately, learn the parameters, and adhere to the equations. This approach employs neural networks to simulate nonlinear systems while minimizing the necessary data and limiting the model's search space with pre-existing knowledge, such as a system of differential equations. Since this development, physics-informed neural networks (PINNs) have been routinely employed as a method for nonlinear function approximation. These networks have exhibited impressive capabilities in tackling diverse scientific computing tasks across multiple fields. Moreover, various research initiatives have utilized the PINNs framework for modeling and scrutinizing dynamic systems. Oluwasakin et al. introduced the logistic-informed neural network

(LINN), inspired by the physics-informed neural network (PINN), to specifically predict the number of individuals infected by the COVID-19 Omicron variant. The major benefit of using LINN in their study lies in its ability to incorporate logistic growth behavior directly into the neural network, resulting in predictions that align well with the natural growth pattern of an epidemic. This method provides more accurate and reliable forecasts for the Omicron variant spread, crucial for timely and effective public health responses and resource allocation. By integrating logistic growth principles, LINN ensures that the model's predictions are not only data-driven but also grounded in epidemiological knowledge [19]. The PINN algorithm was used to learn the epidemiological parameters of a model for COVID-19 vaccine efficacy [20]. Long et al. applied PINN to identify and predict the time-varying parameter of the susceptible (S), infected (I), recovered (R), and death (D) (SIRD model) [21], Olumoyin et al. used the epidemiological-informed neural network (EINN), which was developed through PINN, to learn the time-varying transmission rate for the COVID-19 pandemic in various mitigation scenarios [22]. Thus, this paper seeks to make the following significant contributions:

1. We employ physics-informed neural networks (PINNs) to learn the unknown time-varying parameters within systems of differential equations.
2. Although the application of PINNs in modeling dynamic systems is well-established, our work introduces novel approaches that substantially augment the existing methodology. Firstly, we present a unique assumption where the right-hand side function of a dynamical system is posited to have an equal number of parameters and equations. This hypothesis is pivotal because it implies the existence of an exact solution for the system, a scenario rarely considered in the existing literature. By demonstrating this principle, we provide a foundational understanding that can be leveraged for more accurate modeling of certain types of dynamical systems. Secondly, we address the more complex situation where the number of parameters does not align with the number of equations in the dynamical system. Here, we focus on developing and optimizing network architectures that significantly enhance the convergence speed. This innovation results in improved computational efficiency and elevates the accuracy in estimating time-varying parameters. Our approach is particularly effective in complex systems where traditional models struggle with computational load and accuracy.
3. We thoroughly compare parameters obtained using physics-informed neural networks (PINNs) and those derived from fully connected deep neural networks (DNNs). To assess the effectiveness of both methodologies, we focus on two key aspects: computational efficiency and accuracy. Computational efficiency is evaluated based on the time and resources required to achieve convergence, while accuracy is measured in terms of the fidelity of the parameters to known or established values. We employ a suite of established error metrics, including mean absolute error (MAE), root mean square error (RMSE), and mean squared error (MSE), to evaluate the performance of the proposed method rigorously. These metrics provide a comprehensive view of the method's accuracy and reliability in different scenarios. The evaluation process is designed to validate the efficacy of PINNs in modeling dynamical systems and offer insights into how they compare with traditional DNN approaches. By analyzing the results through these error metrics, we can draw informed conclusions about the practicality of employing PINNs for complex dynamical systems, particularly regarding their ability to provide precise and computationally efficient solutions.
4. We show that employing the PINN algorithm to this specific problem becomes computationally more efficient as the number of layers increases, outperforming DNNs in terms of computational time. This finding suggests that PINNs can provide faster processing rates for complex computational tasks, providing a substantial advantage for time-sensitive applications.

The paper is organized as follows: Section 2 discusses the systems of ordinary differential equations used in the study. Section 3 explores the neural network used for analysis. Section 4 examines error metrics for model performance assessment. Section 5 analyzes computational simulations of dynamical systems. Finally, Section 6 concludes the paper, summarizing the study's main findings and key points.

2. Systems of Ordinary Differential Equation

Consider a dynamical system characterized by n ordinary differential equations encompassing m undetermined parameters (see also, [12]).

$$\frac{d\theta(t)}{dt} = \Theta(t, \theta(t), \delta(t)), \quad t \in (t_0, t_{end}), \tag{1}$$

$$\theta(t_0) = \theta_0$$

These equations make up the mathematical model that describes the system's behavior being studied, with m parameters that define the system's characteristics. In this context, $\theta(t) = [\theta_1(t), \theta_2(t), \theta_3(t), \ldots, \theta_n(t)]^T$ is a vector field composed of n components. Further, let $\delta(t) = [\delta_1(t), \delta_2(t), \ldots, \delta_m(t)] \in \mathbb{R}^m$ be the vector encompassing the unknown parameters, and $\theta_0 \in \mathbb{R}^n$ be the initial condition. On the right-hand side of (1), we have the vector-valued function

$$\Theta(t, \theta(t), \delta(t)) = \begin{bmatrix} \Theta_1(t, \theta_1(t), \theta_2(t), \ldots, \theta_n(t), \delta_1(t), \delta_2(t), \ldots, \delta_m(t)) \\ \Theta_2(t, \theta_1(t), \theta_2(t), \ldots, \theta_n(t), \delta_1(t), \delta_2(t), \ldots, \delta_m(t)) \\ \vdots \\ \Theta_\alpha(t, \theta_1(t), \theta_2(t), \ldots, \theta_n(t), \delta_1(t), \delta_2(t), \ldots, \delta_m(t)) \end{bmatrix}$$

not necessarily linear, consisting of n components designated as

$$\Theta_k(t, \theta_1(t), \theta_2(t), \ldots, \theta_n(t), \delta_1(t), \delta_2(t), \ldots, \delta_m(t))$$

with $k = 1, \ldots, n$. The data for the model equation will be represented by $\theta^*(t)$.

Using the data measured at P distinct time points for the given model, $(t_i, \theta^*(t_i))$ for $i = 1, 2, \ldots, P$, our goal is to develop physics-informed neural network (PINN) algorithm that estimates both the unidentified parameters $\delta(t)$ and the solution $\theta(t)$ across all time points t. Understanding the value and change of these parameters over time can give useful information about the system's dynamic behavior, which can be used to improve predictive models and solutions.

3. Artificial Neural Network

Artificial neural networks, commonly referred to as neural networks (NNs), are computational models inspired by the biological neural networks present within the human brain [14]. These systems can learn to carry out tasks by processing a range of examples, typically without explicit, task-specific programming. While there are numerous types of artificial neural networks, this discussion primarily focuses on deep neural networks (DNNs) and physics-informed neural networks (PINNs).

3.1. Deep Neural Networks

Deep neural networks are a type of artificial neural network with multiple layers between the input and output layers. These layers, known as hidden layers, allow DNNs to learn high-level features from the input data. DNNs are a cornerstone of deep learning and are widely used for tasks such as image recognition, speech recognition, natural language processing, and many other applications. Consider a deep neural network with similar architecture in [12].

$$\theta_n^l = \sigma^l \left(\sum_{k=1}^{l} a_k^l W_k^l + b_k^l \right)$$

where θ_n^l denotes the predicted solution, a_k^l are the input, σ^l is the activation function, W_k^l and b_k^l are the weight, and the bias, respectively. Using the DNN approach to solve and identify the parameters of the dynamical system (2.1), we find the best network parameters ω representing the biases and weights network that minimizes the loss function. In a DNN, the loss function is typically a measure of the discrepancy between the predictions of the network and the true output values from the training data.

$$L(\omega; \mu) = \frac{1}{n} \left(\sum_{i=1}^{\mu} ||\theta(t_n; \omega) - \theta(t_n)||^2 \right)$$

and

$$\theta(t_n; \omega) = \frac{d\theta(t_n; \omega)}{dt} - \Theta \left(t_n, \theta(t_n; \omega), \delta(t_n; \omega) \right)$$

where $\theta(t_n)$ are the output values, $\theta(t_n; \omega)$ are the predicted output values, and n is the number of instances.

3.2. Physics-Informed Neural Network

Physics-informed neural networks represent an advanced machine-learning technique that integrates knowledge from physical principles to provide a robust approximation of function and parameter identification. PINNs have emerged as a powerful tool among data-driven deep neural networks due to their versatility in solving inverse and forward problems [18]. A PINN is a differential system solver discovering the system parameters directly from the available data. Alternatively, it can simulate the differential system when the parameters are already known. Interestingly, PINNs are not restricted to a specific neural network architecture and can utilize architectures like feed-forward neural networks (FNNs) as their foundational framework. PINNs utilize conventional deep learning methodologies, employing standard activation functions and optimization techniques. However, the unique feature of PINNs lies in their sophisticated loss function, composed of boundary conditions, initial values, and physical restrictions. This formulation ensures that the output from the neural network adheres strictly to the underlying system of differential equations by integrating residuals from these equations into the loss function. This algorithm holds immense promise, primarily due to its ability to tackle complex, non-linear systems and process vast quantities of noisy or incomplete data. Furthermore, the duality of PINNs in managing both forward and inverse problems allows them to fit observational data effectively and generate reliable predictions based on identified parameters. Physics-informed neural networks represent a critical leap forward in machine learning. PINNs offer a dynamic and comprehensive approach to data approximation and system parameter identification by amalgamating conventional deep-learning techniques with physical principles embodied in differential equations.

3.3. Parameter Identification of Dynamical Systems Model Using PINN

Let us consider a dynamical system (1). The goal is to use the PINN approach to solve and identify the parameters of the dynamical system model. This can be conducted by finding the best network parameters ω, representing the biases and weights network that minimizes the loss function. We offer a physics-informed neural network shown in Figure 1, with two networks to do this. The first network outputs are $\theta(t_n; \omega)$, which admits t as the input data. The second network learns the parameters $\delta(t_n; \omega)$ from the data. The following loss function was minimized.

$$\zeta(\omega; \kappa) = \zeta_t(\omega; \kappa_t) + \zeta_r(\omega; \kappa_r) \tag{2}$$

$\zeta_t(\omega;\kappa_t)$ is called the training loss and $\zeta_r(\omega;\kappa_r)$ is called the residual loss. $\kappa = \kappa_r \cup \kappa_t$ in the total training datasets. We define the following:

$$\zeta_r(\omega;\kappa_r) = \frac{1}{\kappa_r}\sum_{i=1}^{p}\sum_{n=1}^{\kappa_r}|r_i(t_n)|^2 \tag{3}$$

where

$$r_i(t_n) = \frac{d\theta(t_n;\omega)}{dt} - \Theta\Big(t_n, \theta(t_n;\omega), \delta(t_n;\omega)\Big) \tag{4}$$

and

$$\zeta_t(\omega;\kappa_t) = \frac{1}{\kappa_t}\left(\sum_{n=1}^{\kappa_t}||\theta(t_n;\omega) - \theta(t_n)||^2\right) \tag{5}$$

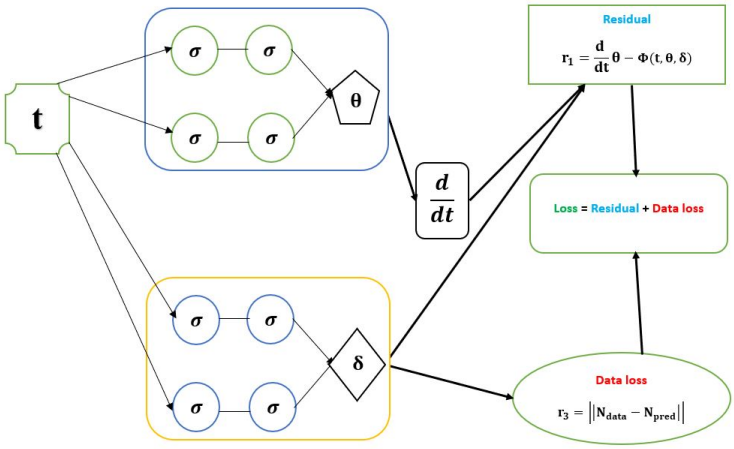


Figure 1. Schematic diagram of the PINN with the parameters of a dynamical system of ODE model.

The network is trained to minimize (3); an optimizer such as Adams is the most optimal weight and bias that would minimize (3). The neural network also obtains the optimal values for the model parameters. The $\theta(t_n)$ represents the loss function's training data. We identify the parameter δ by solving Equation (5). When trying to address the problem (5), we can categorize it into two possible scenarios:

Scenario 1: In this particular instance, we assume the right-hand side function of a dynamical system (1) has an equal number of parameters and equations. This scenario shows that a solution exists. In other words, there is an exact solution [12].

Scenario 2: In this context, the number of parameters does not match the number of dynamical system equations [12].

Remark 1. *It is important to underscore that the number of networks can exceed two, contingent upon the intricacies of the dynamical system and the parameters involved.*

The implementation of physics-informed neural networks (PINNs) is carried out using Python. Its simplicity and extensive library ecosystem make it an ideal choice for developing sophisticated models like PINNs. Key to our implementation is TensorFlow, which offers robust tools for building and training neural network models. TensorFlow's compatibility with Python and its efficient handling of large-scale numerical computations enable the effective construction of PINN architectures. Additionally, we employ NumPy, a Python library that supports large, multi-dimensional arrays and matrices, along with a collection of high-level mathematical functions to operate on these arrays. This is crucial

for handling the numerical aspects inherent in PINN development. We also utilize Pandas, a library for data manipulation and analysis. Pandas provides structures and functions that are ideal for modifying numerical tables and time series, making it useful for preprocessing and organizing the data that neural networks need to learn. The synergy of these libraries in Python creates a powerful toolkit for developing and implementing PINNs.

The source code in Figure 2 provides a comprehensive example of physics-informed neural network implementation, illustrating the practical application of parameter identification in dynamical systems. This Python-based code features a custom PINN class, which encompasses the complete workflow of constructing, training, and deploying the neural network for predictive analysis. At the core of this class is the initialization method, which is responsible for configuring the network's layers, weights, and biases. This setup is facilitated by the initialize_NN function, leveraging TensorFlow's robust computational power for efficient network architecture development. The functions neural_net and neural_net1 each articulate distinct network structures tailored for specific segments of the model. The train and loss functions represent the heart of the learning process, where TensorFlow's optimization algorithms are utilized to refine the network by minimizing the loss function. This method effectively enables the network to learn the intricacies of the dynamical system under study. Finally, the predict function demonstrates the practical use of the trained network in generating predictions for new datasets. This implementation not only showcases the coding aspects of PINNs but also exemplifies the fusion of theoretical principles with their practical application in advanced computational modeling.

```
import numpy as np
import pandas as pd
import timeit
import time
import matplotlib.pyplot as plt
import scipy.io
import tensorflow as tf


class PINN:
    def __init__(self, ...):  # Initialization with parameters and network setup
        # Network architecture setup
        self.weights, self.biases = self.initialize_NN(layers)
        ...
        self.sess = tf.Session(...)  # TensorFlow session

    def initialize_NN(self, layers):  # Initialize weights and biases
        ...

    def neural_net(self, ...): # Building the network for the input and the output solution
        num_layers = len(layers1)
        ...

    def neural_net1(self, ...): # Building the network for learning the time-varing parameter
        ...

    def loss(self, ...): # Loss function
        ...

    def train(self, nIter):  # Training process
        for it in range(nIter):
            ...

    def predict(self, t_star):  # Prediction method
        ...

# Usage
model = PINN(...)
model.train(nIter)  # Training the model
z1_pred, z2_pred, ... = model.predict(t_data)  # Making predictions
```

Figure 2. PINN source code.

4. Error Metrics

This section discusses the error metrics used for data-driven simulations, examining when and why specific metrics should be employed. These error metrics provide valuable insights into the accuracy and performance of data-driven models, such as physics-informed neural networks (PINNs). By evaluating the models based on these metrics, researchers can determine the quality of their predictions and compare the effectiveness of different approaches. In the context of learning the time-varying parameters for dynamical systems, it is important to consider the specific characteristics of the problem and the goals of the analysis when selecting the appropriate error metrics. For instance, if θ symbolizes the actual data and $\hat{\theta}$ represents the data predicted by the model, these error metrics are applied in our data-driven simulations:

1. Mean absolute error (MAE): a significant advantage of using the mean absolute error (MAE) is its relevance to quantile regression. MAE represents an average of absolute differences between predicted and actual values. The formula for mean absolute error is given by:

$$MAE = \frac{1}{N}\sum_{j=1}^{N}\left|\theta_j - \hat{\theta}_j\right|$$

2. Root mean squared error (RMSE): RMSE is in the same units as the predicted variable, providing a straightforward interpretation of the model's prediction error magnitude. The formula for root mean squared error is given by:

$$RMSE = \sqrt{\frac{1}{N}\sum_{j=1}^{N}(\theta_j - \hat{\theta})^2}$$

3. Mean squared error (MSE): mean squared error (MSE) measures the amount of error in the models. It assesses the average squared difference between the observed and predicted values.

$$MSE = \frac{1}{N}\sum_{j=1}^{N}(\theta_j - \hat{\theta})^2$$

5. Computational Simulations of Dynamical Systems

This section explores physics-informed neural networks (PINNs) for parameter identification and solution in various dynamical systems. Parameter identification is crucial, especially when working with ordinary differential equations (ODEs), as each model is essentially unique. ODEs showcase their practicality when the parameters are correctly determined, emphasizing this methodology's significance. The identification of parameters becomes evident as we develop a model employing ordinary differential equations (ODEs) to elucidate the fundamental occurrences. These parameters include different characteristics of the modeled system, such as rate constants, initial conditions, and coefficients, defining the form, rate of change, and stability of the solution. The parameters that make up an ODE can also determine the computational techniques used to solve it and have a big impact on the quality of the solution. Consequently, parameter identification is crucial for making insightful predictions and adequately assessing the simulation of the modeled dynamical system. We consider five benchmark problems derived from the existing literature in our discussion. For each example, we will specify the choice of neurons, the number of hidden layers, activation functions, and other network parameters. Two of these five problems have an exact analytical solution, which will be leveraged to validate the model's accuracy. In addition, the relative error metric showing the fitting accuracy of each dynamical system using the PINN algorithm will be provided.

5.1. First-Order Irreversible Chain Reactions

First-order irreversible chain reactions are a type of chemical reaction where a reactant undergoes a series of transformations to form a product, with the process continuing in a chain-like manner. A common example of such a reaction is the decomposition of hydrogen peroxide (H_2O_2) in the presence of an iodide ion as a catalyst. One of the most fundamental studies on chain reactions was conducted by Polanyi and Wigner [23], who published a series of papers in the 1920s and 1930s on the theory of chain reactions. Consider the subsequent first-order irreversible chain reactions [24,25].

$$\mathcal{A} \xrightarrow{k_1} \mathcal{B} \xrightarrow{k_2} \mathcal{C}$$

The first step is a first-order reaction where species $\mathcal{A}$ transforms into species $\mathcal{B}$ at a rate determined by the rate constant k_1. The second is another first-order reaction where species $\mathcal{B}$ transforms into species $\mathcal{C}$ at a rate determined by the rate constant k_2. In both steps, the reaction rate depends on the concentration of a single reactant (either A or B), which is characteristic of first-order reactions and characterized by the following model:

$$\begin{aligned} \frac{dz_1(t)}{dt} &= -k_1(t)z_1(t) \\ \frac{dz_2(t)}{dt} &= k_1(t)z_1(t) - k_2(t)z_2(t). \end{aligned} \tag{6}$$

where z_1 and z_2 symbolize the concentrations of chemical species $\mathcal{A}$ and $\mathcal{B}$, respectively and k_1, k_2 denote the rate constants which are the parameters required to be learned. Given that $k_1 = 5$, $k_2 = 1$ and $z(t = 0) = [1, 0]$, then the observed data z_1 and z_2 are given in Table 1. The data was taken from reference [12].

Table 1. Observed values of z_1 and z_2 at different time points.

t	z_1	z_2
0.0	1.000000	0.000000
0.1	0.606531	0.372883
0.2	0.367879	0.563564
0.3	0.223130	0.647110
0.4	0.135335	0.668731
0.5	0.082085	0.655557
0.6	0.049787	0.623781
0.7	0.030197	0.582985
0.8	0.018316	0.538767
0.9	0.011109	0.494326
1.0	0.006738	0.451427

The analytical solution of (6) is given as follows:

$$\begin{aligned} z_1(t) &= e^{-5t} \\ z_2(t) &= \frac{5}{4}e^{-5t}(-1 + e^{4t}), \end{aligned} \tag{7}$$

and the analytical derivatives of (7) are given by:

$$\begin{aligned} \frac{dz_1(t)}{dt} &= -5e^{-5t} \\ \frac{dz_2(t)}{dt} &= 5e^{-t} - \frac{25}{4}e^{-5t}(-1 + e^{4t}). \end{aligned} \tag{8}$$

The PINNs approach to solve and identify the parameters of the first-order irreversible chain reactions model is performed by finding the best network parameters, ω, which represents the biases and weights network that minimizes the loss function. We offer a physics-informed neural network (PINN) shown in Figure 3 with three networks to do this.

The first network outputs are $z_{1NN}(t_n;\omega)$ and $z_{2NN}(t_n;\omega)$, which admits t as the input data. The second and third networks learn the parameters $k_1(t_n;\Phi)$ and $k_2(t_n;\Phi)$ from the data. The neural network also obtains the optimal values for the model parameters. The $z_1(t_n)$ and $z_2(t_n)$ represent the loss function's training data. Finally, the first-order irreversible chain reaction model parameters' mean value was obtained using PINN with one layer of 40 neurons, 90,000 epochs, sigmoid activation function used, and 10^{-3} learning rate. The PINN Algorithm 1 for learning the optimal parameters of the first-order irreversible chain reactions model (6) is shown above.

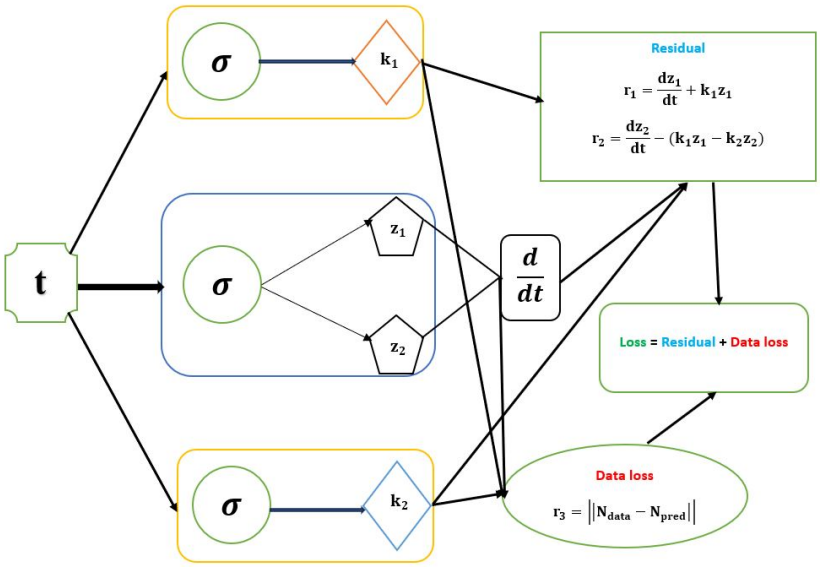


Figure 3. Schematic diagram of the PINN with the parameters of first-order irreversible chain reactions model.

We define the residual loss $\zeta_r(\omega;\kappa_r)$ and training loss $\zeta_t(\omega;\kappa_t)$ as follows:

$$\zeta_r(\omega;\kappa_r) = \frac{1}{\kappa_r}\sum_{i=1}^{2}\sum_{n=1}^{\kappa_r}|r_i(t_n)|^2 \tag{9}$$

where

$$\begin{aligned} r_1(t_n) &= \frac{dz_{1NN}(t_n;\omega)}{dt} + k_1(t_n;\Phi)z_{1NN}(t_n;\omega) \\ r_2(t_n) &= \frac{dz_{2NN}(t_n;\omega)}{dt} - \Big(k_1(t_n;\Phi)z_{1NN}(t_n;\omega) - k_2(t_n;\Phi)z_{2NN}(t_n;\omega)\Big) \end{aligned} \tag{10}$$

and

$$\zeta_t(\omega;\kappa_t) = \frac{1}{\kappa_t}\left(\sum_{n=1}^{\kappa_t}|z_{1NN}(t_n;\omega) - z_1(t_n)|^2 + \sum_{n=1}^{\kappa_t}|z_{2NN}(t_n;\omega) - z_2(t_n)|^2\right) \tag{11}$$

Two distinct scenarios were examined in the study of the first-order irreversible chain reactions model, with results detailed in Figures 4–7. For Scenario 1, the system was solved using Equation (10) at $t^* = 0.28$ and 0.19, while for Scenario 2, it was solved at $t^* = 0.22$ and 0.27. Figure 4 shows the obtained solution of the actual output and the predicted output of the first-order irreversible chain reactions model from the data. The phase space plot of the actual output of species $z_1(t)$ against species $z_2(t)$ and the predicted output of species $z_1(t)$ against species $z_2(t)$ is shown in Figure 5. This visualization provides insight into the relationships and behaviors of the species involved. Figure 6 shows the learned time-varying parameter k_1 and k_2 values of the first-order irreversible chain reaction models, shedding light on how these parameters change over time in the first-order irreversible chain reaction models. The $k_1(t)$ curve appears to have an oscillatory behavior,

and the oscillation frequency seems consistent throughout the curve. Upon examining the amplitude of oscillations over time, there is a noticeable variation in amplitude. While there are some fluctuations, there appears to be an amplitude increase toward the end of the interval. Considering the observations, a potential functional form for the curve could be a simple sinusoidal oscillation with a constant offset. A common form for such a function is:

$$k_1(t) = Asin(\lambda t + \alpha) + C.$$

where A is the amplitude, λ determines the frequency, α is the phase shift, and C is the vertical shift (which should be close to 5, given the curve's behavior). The curve $k_2(t)$ appears to have an initial rise, followed by a decrease, then stabilizes after a certain time. It appears to resemble an exponential decay combined with an exponential rise. A potential functional form for such a curve could be:

$$k_2(t) = ae^{-bt} + ce^{dt} + e.$$

where a and c are the amplitudes of the respective exponential terms, b and d are the rate constants for the decaying and rising terms, and e is a constant offset.

Algorithm 1 PINN algorithm for learning the parameters of the first-order irreversible chain reactions model

1: Construct PINN
   Specify the input: $t_n, n = 1, \ldots, X$
   Initialize PINN parameter: ω
   Output layer: $z_{1NN}(t_n; \omega)$ and $z_{2NN}(t_n; \omega), n = 1, \ldots, X$
2: Construct neural network: k_1
   Specify the input: $t_n, n = 1, \ldots, X$
   Initialize PINN parameter: Φ
   Output layer: $k_1(t_n; \Phi), n = 1, \ldots, X$
3: Construct neural network: k_2
   Specify the input: $t_n, n = 1, \ldots, X$
   Initialize PINN parameter: Φ
   Output layer: $k_2(t_n; \Phi), n = 1, \ldots, X$
4: Specify the training set
   $\kappa = \kappa_r \cup \kappa_b$ of the NN
5: Train the neural network
   Specify a loss function

$$\zeta(\omega; \kappa) = \zeta_t(\omega; \kappa_t) + \zeta_r(\beta; \alpha_r)$$
$$r_1(t_n) = \frac{dz_{1NN}(t_n; \omega)}{dt} + k_1(t_n; \Phi) z_{1NN}(t_n; \omega)$$
$$r_2(t_n) = \frac{dz_{2NN}(t_n; \omega)}{dt} - \left(k_1(t_n; \Phi) z_{1NN}(t_n; \omega) - k_2(t_n; \Phi) z_{2NN}(t_n; \omega) \right)$$

   Minimize $min_\kappa \zeta(\omega; \kappa)$ using Adam Optimizer.
6: Return PINN solution
   $z_{1NN}(t_n; \omega)$ and $z_{2NN}(t_n; \omega), n = 1, \ldots, X$
7: Return Parameter k_1:
   $k_1(t_n; \Phi), n = 1, \ldots, X$
8: Return Parameter k_2:
   $k_2(t_n; \Phi), n = 1, \ldots, X$

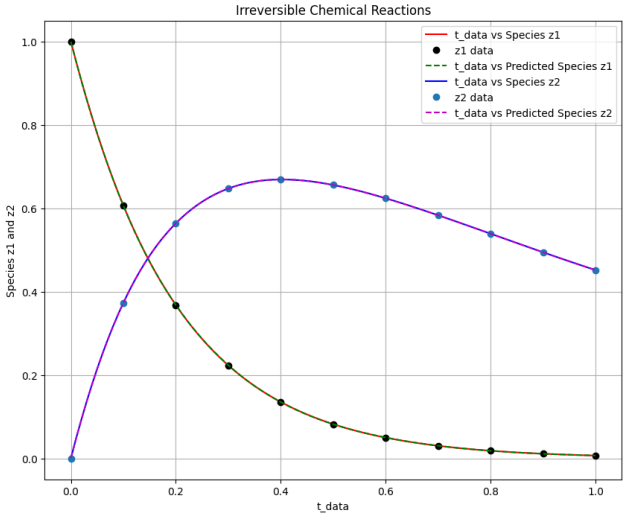


Figure 4. The first-order irreversible chain reactions solution of the actual output of species z_1, z_2 against t_{data} and the predicted output of species z_1, z_2 against t_{data}.

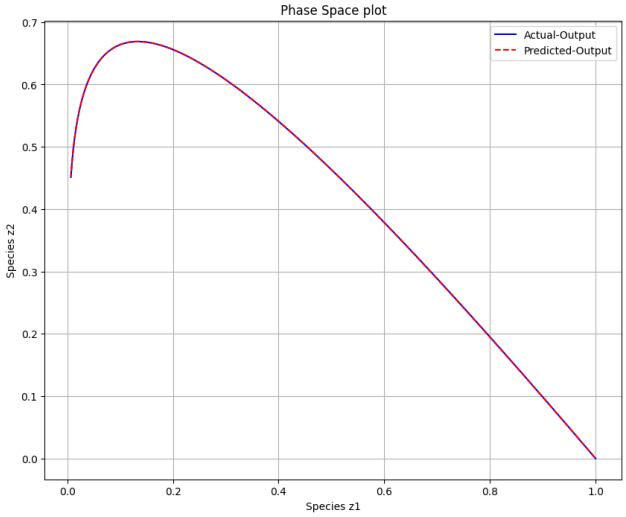
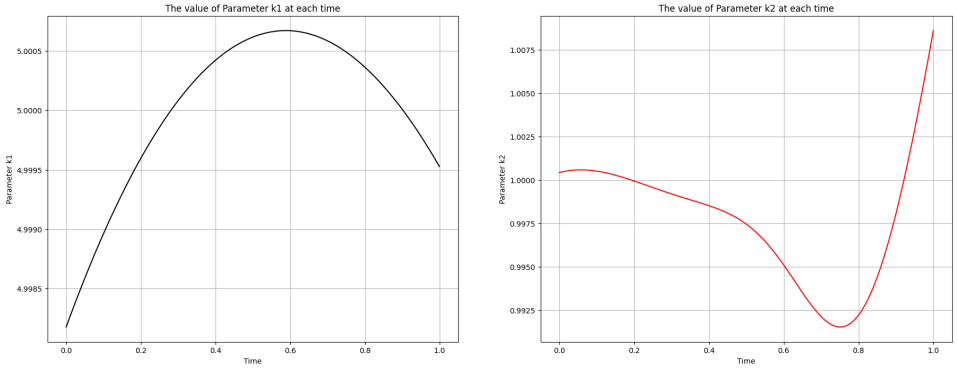
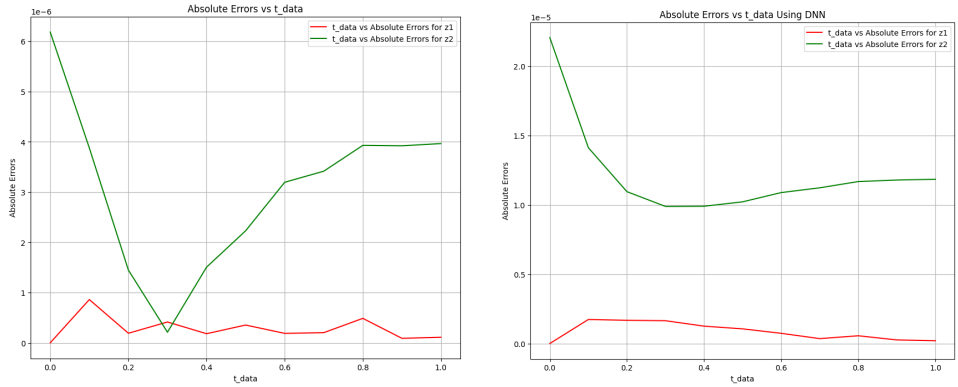


Figure 5. The phase space plot of the actual output of species z_1 against species z_2 and the predicted output of species z_1 against species z_2.



(**a**) (**b**)

Figure 6. The learned time-varying parameter values of the first-order irreversible chain reactions model. (**a**) Time-varying parameter k_1. (**b**) Time-varying parameter k_2.



(**a**) (**b**)

Figure 7. Absolute error plot between the data, PINN solution, and DNN solution. (**a**) Absolute error plot using PINN. (**b**) Absolute error plot using DNN.

Table 2 shows the obtained time-varying parameters and error metrics through the approaches described in Scenario 1 and Scenario 2. We observe that Scenario 1 proved more accurate than Scenario 2. The accuracy of the PINN technique is evaluated by comparing the predicted concentrations of z_1 and z_2 with the actual values obtained from the data. These metrics provide insights into the fitting accuracy of the PINN technique. The results of the simulations show that the PINN algorithm can accurately predict the concentrations and learn the time-varying parameters of the first-order irreversible chain reactions model. The obtained time-varying parameter values are close to the true values, and the error metrics indicate high accuracy and good fitting of the model to the data.

Table 2. Comparison of the two approaches using PINN.

	$k_1(t)$	$k_2(t)$
True Values	5.0	1.0
Approach 1	5.0000086	0.99999505
Error of Approach 1	0.0000086	0.00000495
Approach 2	5.0005193	1.0001917
Error of Approach 1	0.0005193	0.0001917
	$z_1(t)$	$z_2(t)$
MAE of Approach 1	2.7979×10^{-7}	3.0801×10^{-6}
MAE of Approach 2	1.7826×10^{-5}	3.7310×10^{-5}
MSE of Approach 1	1.3095×10^{-13}	1.1916×10^{-11}
MSE of Approach 2	4.8358×10^{-10}	2.0368×10^{-9}
RMSE of Approach 1	3.6187×10^{-7}	3.4519×10^{-6}
RMSE of Approach 2	2.1991×10^{-5}	4.5131×10^{-5}

In addition, we compared the results we achieved using physics-informed neural networks to those obtained using deep neural networks, as shown in Tables 3 and 4. This comparison study used a setup of 90,000 epochs and one hidden layer containing 40 neurons, with processing time measured in seconds.

Figure 7 explores the model's accuracy by showing the absolute error between the target and the predictions. Specifically, Figure 7a indicates that the absolute error is bounded by 1×10^{-6} when using PINN, while Figure 7b reveals that the error is bounded by 1×10^{-5} using DNN. This comparative analysis highlights the difference in accuracy between the two techniques, PINN and DNN, used in the modeling process. Finally, the results show that the PINN technique achieves comparable or better parameter estimation accuracy than the DNN technique. Additionally, the PINN technique exhibits faster computation times, making it an efficient choice for solving and identifying parameters in dynamical systems.

Table 3. Comparison of obtained results using PINN vs. DNN through the approaches described in Scenario 1. The table demonstrates a significant 67.7% improvement in computational efficiency when employing PINNs compared to DNNs, highlighting the enhanced performance and time-saving capabilities of PINNs in dynamic system modeling.

	True Values	PINN	DNN	Epochs	CPU Using PINN	CPU Using DNN
k_1	5.0	5.0000086	5.0000643	90,000	40s	124s
k_2	1.0	0.99999505	1.0000190			

Table 4. Comparison of obtained error results using PINN vs. DNN. This table presents a detailed analysis of the error metrics for both PINN and DNN models across two variables, u_1 and u_2. The results highlight the superior accuracy of PINNs, as evidenced by significantly lower error values across all metrics compared to DNNs, underscoring the effectiveness of PINNs in dynamic system modeling.

Error Metrics	$u_1(PINN)$	$u_1(DNN)$	$u_2(PINN)$	$u_2(DNN)$
MAE	2.7979×10^{-7}	2.1615×10^{-6}	3.0801×10^{-6}	2.3330×10^{-5}
MSE	1.3095×10^{-13}	7.0444×10^{-12}	1.1916×10^{-11}	5.6775×10^{-10}
$RMSE$	3.6187×10^{-7}	2.6541×10^{-6}	3.4519×10^{-6}	2.3827×10^{-5}

Furthermore, we also conducted a detailed investigation on the impact of augmenting the number of layers in the network called shallow vs. deep layers. Specifically, we focused on network architectures consisting of one, two, and three hidden layers, implemented using physics-informed neural networks. Subsequently, we compared the results obtained with deep neural networks. The respective outcomes of these analyses are succinctly presented in Table 5. The results indicate that the PINN with one hidden layer achieves good accuracy, and increasing the number of layers does not significantly improve performance. Interestingly, the three hidden layer PINN model showed a lower computation time (9 s) than the two layer model (13 s). There are various reasons for this unexpected outcome. Firstly, the increased depth (three layers) might enable more efficient learning and quicker convergence to a solution, thereby reducing the required epochs for training. Although the table shows more epochs for the two-layer configuration than the three-layer one, the actual computation time (CPU time) is less for the latter. This suggests that each epoch in the three-layer configuration is computationally less expensive.

Table 5. Comparison of PINN vs. DNN based on shallow vs. deep layers through the approaches described in Scenario 1. This table details the accuracy of parameter estimation (k_1 and k_2) and computational performance across various network depths. Notably, it showcases the increasing computational efficiency of PINNs over DNNs, with improvements of 66.67%, 60.61%, and 62.5% for different layer configurations. These findings emphasize the enhanced efficiency and adaptability of PINNs in handling complex dynamical systems, especially in deeper network architectures.

Layers	Neurons	k_1 (PINN)	k_1 (DNN)	k_2 (PINN)	k_2(DNN)	Epochs	CPU PINN	CPU DNN
1	40	4.99815	4.9925046	1.0002131	0.9969400	21378	10s	30s
2	40, 40	5.0000534	4.9999063	1.00021	1.0000036	24358	13s	33s
3	40, 40, 40	4.999991	5.0050346	0.9999847	1.0018185	16021	9s	24s

Moreover, the three-layer network might better capture the complexities of the modeled dynamical system, leading to more efficient computation per epoch. This is indirectly supported by parameter estimation accuracy (k_1 and k_2), which are very close to their ideal values in the three-layer setup, indicating a high-quality learning process. Therefore, the reduction in computation time for the three-layer networks compared to the two-layer ones can be attributed to more efficient learning per epoch, potentially due to better repre-

sentation and processing of the complex relationships in the data by the deeper network architecture.

5.2. Brusselator Model

The Brusselator model is a simple mathematical model for the behavior of chemical oscillations. Ilya Prigogine and Lefever first proposed it in 1968 [26]. The model comprises two chemical species, U and V, that interact through non-linear differential equations. The equations describe how the concentrations of the two species change over time, taking into account chemical reactions, diffusion, and other factors. The Brusselator is known for its ability to make complex, self-sustaining oscillations in the concentrations of the two chemical species. As a result, it has been used to study a wide range of phenomena in chemistry, biology, and physics, including patterns of gene expression and the behavior of simple electronic circuits [27,28]. The Brusselator model is useful in various fields because it is a simple yet rich model that can exhibit a wide range of complex behaviors.

The Brusselator reaction is a simple model of autocatalytic chemical reactions, which is a chemical reaction catalyzed by one of the products of the reaction. The reaction has been used as a model to study the emergence of self-organizing behavior in chemical systems [29]. It has been applied to various fields, including chemistry, physics, biology, and engineering. The Brusselator reaction consists of the following:

$$A \longrightarrow U \rightarrow \frac{du}{dt} = k_a a$$

$$2U + V \longrightarrow 3U \rightarrow \left(\frac{du}{dt} = k_b u^2 v, \frac{dv}{dt} = -k_b v^2 y\right)$$

$$B + U \longrightarrow V + C \rightarrow \left(\frac{du}{dt} = -k_c bu, \frac{dv}{dt} = -k_c bu\right)$$

$$U \longrightarrow D \rightarrow \frac{du}{dt} = -k_a u.$$

The overall reaction is $A + B \longrightarrow C + D$ with intermediary species U and V. A and B are reactants while C and D are products. During the chemical reaction, suppose the concentration reactant A and B are kept constant; therefore, the ordinary differential equation version of the Brusselator is given below:

$$\begin{aligned} \frac{du(t)}{dt} &= a(t) + u(t)^2 v(t) - (b(t) + 1)u(t) \\ \frac{dv(t)}{dt} &= bu(t) - u(t)^2 v(t). \end{aligned} \tag{12}$$

where $u(t)$ and $v(t)$ are the concentrations of the two chemical species involved in the reaction, and $a(t)$ and $b(t)$ are the time-varying parameters that control the system's behavior. The Brusselator model exhibits various dynamic behaviors, depending on the values of the a and b parameters [26]. They demonstrated that the system could exhibit stable, steady states, oscillations, and chaotic behavior. Finally, the parameters of the Brusselator system are also important in understanding the behavior of real-world chemical systems, which can be modeled using similar equations. Understanding how the parameters of such a system affect its behavior can provide insights into the underlying chemical reactions and help predict and control the system's behavior.

Our primary aim is to learn the time-varying parameters of the Brusselator model. Here, we used synthetic data that was generated by randomizing the numerical solution of the Brusselator model derived from an ODE-solver on the Brusselator model, where we assumed the initial conditions for the first species and the second species variables to be $u(0) = 1$ and $v(0) = 1$, respectively. We choose $a = 1.0$ and $b = 2.5$. This problem was solved on the time interval $[0.T]$, where T = 30. We used a grid with $P_x = 1000$ points on

the time interval, and every second-time step, the numerical solution vector is taken to obtain the observation or synthetic data. We use the PINN architecture shown in Figure 1 with three networks to learn the time-varying parameters of the Brusselator model. The first network outputs are $u_{NN}(t_n;\omega)$ and $v_{NN}(t_n;\omega)$, which admits t as the input data. The second and third networks learn the parameters $a(t_n;\Phi)$ and $b(t_n;\Phi)$ from the data. We define the residual loss $\zeta_r(\omega;\kappa_r)$ and training loss $\zeta_t(\omega;\kappa_t)$ which was minimized as follows:

$$\zeta_r(\omega;\kappa_r) = \frac{1}{\kappa_r}\sum_{i=1}^{2}\sum_{n=1}^{\kappa_r}|r_i(t_n)|^2. \tag{13}$$

where

$$\begin{aligned} r_1(t) &= \frac{du_{NN}(t_n;\omega)}{dt} - \Big(a(t_n;\Phi) - (b(t_n;\Phi)+1)u_{NN}(t_n;\omega) + u_{NN}(t_n;\omega)^2 v_{NN}(t_n;\omega)\Big) \\ r_2(t) &= \frac{dv_{NN}(t)}{dt} - \Big(b(t_n;\Phi)u_{NN}(t_n;\omega) - u_{NN}(t_n;\omega)^2 v_{NN}(t_n;\omega)\Big) \end{aligned} \tag{14}$$

and

$$\zeta_t(\omega;\kappa_t) = \frac{1}{\kappa_t}\Big(\sum_{n=1}^{\kappa_t}||u_{NN}(t_n;\omega) - u(t_n)||_2^2 + \sum_{n=1}^{k_t}||v_{NN}(t_n;\omega) - v(t_n)||_2^2\Big). \tag{15}$$

The time-varying parameters of the Brusselator model were obtained after using PINN and the approach of Scenario 2 with three hidden layers, 64 neurons per layer, 50,000 epochs, the tanh activation function was used, and the learning rate was 10^{-3}. The PINN Algorithm 2 is an excellent choice for learning the optimal parameters of the Brusselator model. The maximum parameter value a was solved at $t^* = 6.37$ and parameter b at $t^* = 13.76$. Figure 8 shows the obtained solution of the actual and predicted output of the Brusselator model after the parameters were learned with the same initial condition used to generate the data. The phase space plot of the actual output of species $u(t)$ against species $v(t)$ and the predicted output of species $u_{NN}(t)$ against species $v_{NN}(t)$ is shown in Figure 9.

Figure 10 shows the learned time-varying parameter values of $a(t)$ and $b(t)$ of the Brusselator model. The $a(t)$ curve appears to oscillate with an overall decreasing trend. A potential functional form for this curve is:

$$a(t) = e^{-\alpha\beta(t)} \tag{16}$$

$$b(t) = e^{\alpha\beta(t).} \tag{17}$$

where α is the constant coefficient, its value will determine the growth or decay rate. Since α is negative in (16), the function represents decay, and if positive, the function represents growth. The magnitude of α will determine how fast this growth or decay occurs. The $b(t)$ curve also appears to oscillate with an overall increasing trend. Since α is positive in (17), the function represents growth, and if negative, the function represents decay. $\beta(t)$ is a time-dependent function that appears as the exponent of the exponential term. It plays a crucial role in determining the behavior of the curve over time. Table 6 shows the parameters and error metrics obtained through the approaches described in Scenario 2. The results showed that the PINN algorithm successfully learned the parameters of the Brusselator model. The obtained parameter values were close to the true values used to generate the synthetic data. The accuracy of the predictions was evaluated using various error metrics, such as mean absolute error (MAE), mean squared error (MSE), and root mean squared error (RMSE). The error metrics indicated a high level of accuracy in the predictions, with very small errors.

Algorithm 2 PINN algorithm for learning the parameters of the Brusselator model

1: Construct PINN
   Specify the input: $t_n, n = 1, \ldots, X$
   Initialize PINN parameter: ω
   Output layer: $u_{NN}(t_n)$ and $v_{NN}(t_n), n = 1, \ldots, X$
2: Construct neural network: a
   Specify the input: $t_n, n = 1, \ldots, X$
   Initialize PINN parameter: Φ
   Output layer: $a(t_n), n = 1, \ldots, X$
3: Construct neural network: b
   Specify the input: $t_n, n = 1, \ldots, X$
   Initialize PINN parameter: Φ
   Output layer: $b(t_n), n = 1, \ldots, X$
4: Specify the training set
   $k = k_r \cup k_b$ of the NN
5: Train the neural network
   Specify a loss function

$$\zeta(\omega;\kappa) = \zeta_t(\omega;\kappa_t) + \zeta_r(\omega;\kappa_r)$$

$$r_1(t) = \frac{du_{NN}(t_n;\omega)}{dt} - \Big(a(t_n;\Phi) - (b(t_n;\Phi)+1)u_{NN}(t_n;\omega) + u_{NN}(t_n;\omega)^2 v_{NN}(t_n;\omega)\Big)$$

$$r_2(t) = \frac{dv_{NN}(t_n;\omega)}{dt} - \Big(b(t_n;\Phi)u_{NN}(t_n;\omega) - u_{NN}(t_n;\omega)^2 v_{NN}(t_n;\omega)\Big)$$

   Minimize $min_\kappa \zeta(\omega;\kappa)$ using Adam Optimizer.
6: Return PINN solution
   $u_{NN}(t_n;\omega)$ and $v_{NN}(t_n;\omega), n = 1, \ldots, X$
7: Return Parameter a:
   $a(t_n;\Phi), n = 1, \ldots, X$
8: Return Parameter b:
   $b(t_n;\Phi), n = 1, \ldots, X$

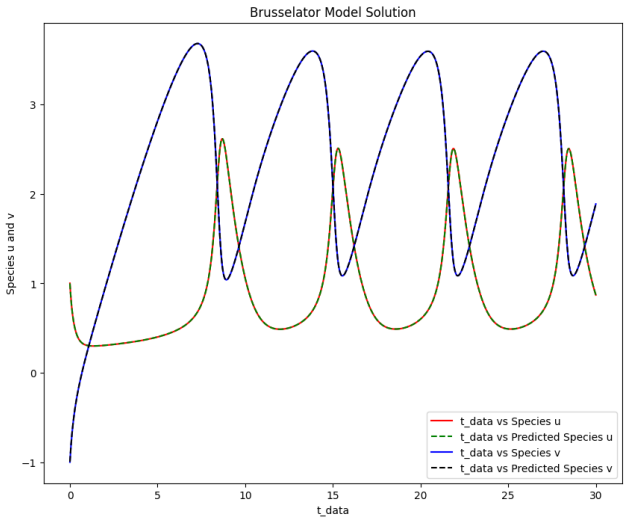


Figure 8. The Brusselator model solution of the actual output of species u,v against t_{data} and the predicted output of species u,v against t_{data}.

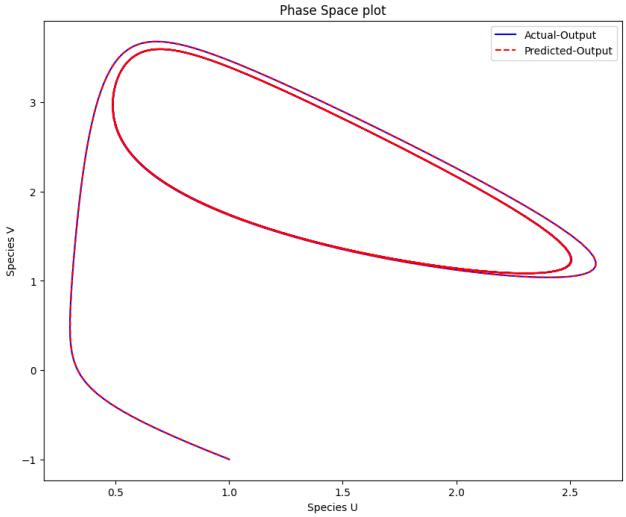
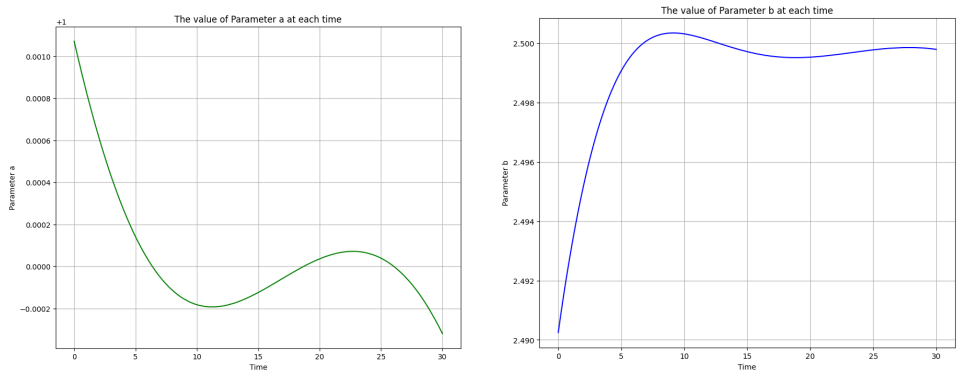


Figure 9. The phase space plot of the actual output of species u against species v and the predicted output of species u against species v.



(**a**) (**b**)

Figure 10. The learned time-varying parameter values of the Brusselator model. (**a**) Time-varying parameter a. (**b**) Time-varying parameter b.

Table 6. The optimal parameter estimation and the error metrics using PINN.

	a	b
True Values	1.0	2.5
Approach 2	1.0000001	2.5
Error of Approach 2	0.0000001	0.0000
	u	v
MAE of Approach 2	1.5247×10^{-6}	1.7520×10^{-6}
MSE of Approach 2	6.5662×10^{-12}	9.5430×10^{-12}
RMSE of Approach 2	2.5625×10^{-6}	3.0892×10^{-6}

Furthermore, we conducted a comparison between the results obtained using physics-informed neural networks (PINN) and those obtained through deep neural networks (DNN), as detailed in Tables 7 and 8. This comparative analysis was carried out under the specific setup of 50,000 epochs, three hidden layers containing 64 neurons, and employing the tanh activation function. Processing time, measured in seconds, was also considered in this comparison. The study revealed that PINN not only achieved faster processing times than DNN but also showed that the optimal parameters learned through PINN were closer

to the actual values than those determined by DNN. This emphasizes the efficiency and accuracy advantages of PINN in this particular application.

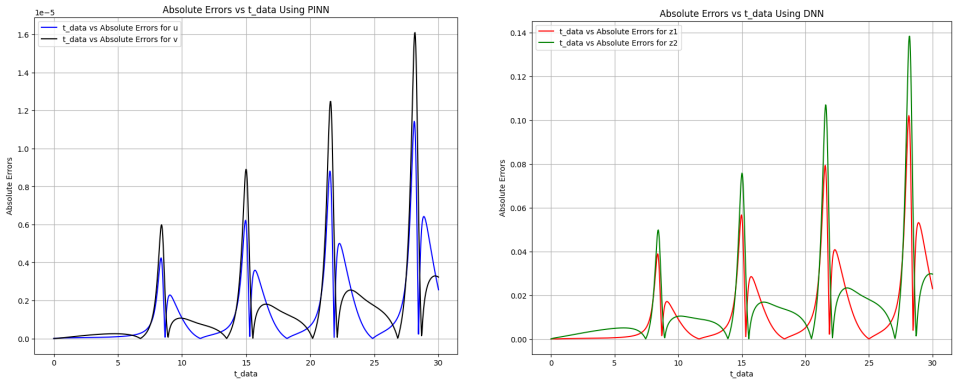
Table 7. Comparison of obtained results using PINN vs. DNN through the approaches described in Scenario 1. This table illustrates the accuracy in parameter estimation and the significant computational efficiency improvement of 88.7% with PINNs over DNNs, highlighting the robustness and speed of PINNs in complex system modeling.

	True Values	PINN	DNN	Epochs	CPU Using PINN	CPU Using DNN
a	1.0	1.0000001	1.0038480	50,000	111s	982s
b	2.5	2.5	2.5088659			

Table 8. Comparison of obtained error results using PINN vs. DNN.

Error Metrics	$u(PINN)$	$u(DNN)$	$v(PINN)$	$v(DNN)$
MAE	1.5247×10^{-6}	3.6030×10^{-2}	1.7520×10^{-6}	4.5508×10^{-2}
MSE	6.5662×10^{-12}	3.8405×10^{-3}	9.5430×10^{-12}	5.4941×10^{-3}
$RMSE$	2.5624×10^{-6}	6.1972×10^{-2}	3.0891×10^{-6}	7.4122×10^{-2}

In addition, the model's accuracy is further illustrated in Figure 11, where the absolute error between the target and predictions made by physics-informed neural networks (PINN) and deep neural networks (DNN) is presented. The comparative analysis demonstrates that the error was significantly reduced when PINN was utilized instead of DNN. This discrepancy highlights the difference in accuracy between the two techniques employed in the modeling process. Furthermore, the results demonstrate that PINN matches but often exceeds DNN regarding parameter estimation accuracy. Finally, faster computation times make PINN an especially efficient choice for solving and pinpointing parameters in dynamical systems.



(**a**) (**b**)

Figure 11. Absolute error plot between the data, PINN solution, and DNN solution for the Brusselator model. (**a**) Absolute error plot using PINN. (**b**) Absolute error plot using DNN.

Finally, an analysis of data-driven simulations for training with varying numbers of epochs, such as 30,000, 40,000, and 50,000, using three layers with 64 neurons on the Brusselator model to learn time-varying parameters is presented in Table 9. The overall loss decreases as the number of epochs increases from 30,000 to 50,000. This indicates that the model benefits from a longer training duration, refining its weights and biases more effectively to minimize the difference between its predictions and the actual data. Therefore, increasing the number of epochs generally leads to a decrease in loss and better performance in predicting the parameters of u and v.

Table 9. Analysis of the Brusselator model using different epochs.

Epochs	Loss	Error u	Error v
50,000	2.7514×10^{-2}	7.9001×10^{-4}	5.5722×10^{-4}
40,000	1.9048×10^{-1}	1.5938×10^{-3}	3.3815×10^{-3}
30,000	1.5272×10^{-1}	2.8571×10^{-3}	5.4818×10^{-3}

5.3. Biomass Transfer

Biomass transfer involves the movement of organic material from one organism to another within an ecosystem. This concept is key to understanding the flow of energy within an ecosystem, as energy is transferred from producers (like plants) to primary consumers (like herbivores) and then on to secondary and tertiary consumers (like carnivores and omnivores). Imagine a forest in Europe populated by one or two species of trees. We pick out some of the eldest among them, those on the brink of their life cycle and anticipated to wither in the coming years. We then observe their journey from living, thriving trees to becoming lifeless entities. Over time, these deceased trees decompose and topple due to natural seasonal changes and biological influences. Ultimately, these fallen trees transform into nutrient-rich humus, completing their life cycle [30]. Differential equations are often used to model biomass transfer as they can describe the rate of change of a quantity (in this case, biomass) over time.

Let us consider the dynamical system model of a biomass transfer

$$\begin{aligned} \frac{du_1(t)}{dt} &= -a(t)u_1(t) + b(t)u_2(t) \\ \frac{du_2(t)}{dt} &= -b(t)u_2(t) + c(t)u_3(t) \\ \frac{du_3(t)}{dt} &= -c(t)u_3(t). \end{aligned} \tag{18}$$

where variable $u_1(t)$, $u_2(t)$ and $u_3(t)$ is defined by

$$\begin{aligned} u_1(t) &: \text{biomass decayed into humus per time} \\ u_2(t) &: \text{biomass of dead trees per time} \\ u_2(t) &: \text{biomass of living trees} \\ t &: \text{time in decades} \end{aligned}$$

and the variable a, b and c are the parameters required to be estimated. Given that $a = 1$, $b = 3$, $c = 5$ and the initial conditions of $u(t = 0) = [0, 0, 1]$, then the observed data [12] are given in Table 10.

Table 10. Observed values of u_1, u_2 and u_3 at different time points.

t	$u1$	$u2$	$u3$
0.0	0.000000	0.000000	1.000000
0.1	0.055747	0.335719	0.606531
0.2	0.166850	0.452330	0.367879
0.3	0.282767	0.458599	0.223130
0.4	0.381125	0.414647	0.135335
0.5	0.454416	0.352613	0.082085
0.6	0.502502	0.288780	0.049787
0.7	0.528506	0.230648	0.030197
0.8	0.536641	0.181006	0.018316
0.9	0.531127	0.140241	0.011109
1.0	0.515706	0.107623	0.006738

The analytical solution of (18) is given as follows:

$$\begin{aligned} u_1(t) &= \frac{15}{8}\left(e^{-5t} - 2e^{-3t} + e^{-t}\right) \\ u_2(t) &= \frac{5}{2}\left(-e^{-5t} + e^{-3t}\right) \\ u_3(t) &= e^{-5t}. \end{aligned} \tag{19}$$

The PINN approach to solve and identify the parameters of the biomass transfer model is made by finding the best network parameters, ω, representing the biases and weights network that minimizes the loss function. Following a similar procedure in the previous application, we offer PINN with four networks to do this. The first network outputs are $u_{1NN}(t_n;\omega)$, $u_{2NN}(t_n;\omega)$ and $u_{3NN}(t_n;\omega)$, which admits t as the input data. The second, third and fourth networks learn the parameters $a(t_n;\Phi)$, $b(t_n;\Phi)$ and $c(t_n;\Phi)$ from the data.

We define the residual loss $\zeta_r(\omega;\kappa_r)$ and training loss $\zeta_t(\omega;\kappa_t)$ as follows:

$$\zeta_r(\omega;\kappa_r) = \frac{1}{\kappa_r}\sum_{i=1}^{3}\sum_{n=1}^{\kappa_r}|r_i(t_n)|^2. \tag{20}$$

where

$$\begin{aligned} r_1(t_n) &= \frac{du_{1NN}(t_n;\omega)}{dt} - \left(-a(t_n;\Phi)u_{1NN}(t_n;\omega) + b(t_n;\Phi)u_{2NN}(t_n;\omega)\right) \\ r_2(t_n) &= \frac{du_{2NN}(t_n;\omega)}{dt} - \left(-b(t_n;\Phi)u_{2NN}(t_n;\omega) + c(t_n;\Phi)u_{3NN}(t_n;\omega)\right) \\ r_3(t_n) &= \frac{du_{3NN}(t_n;\omega)}{dt} - \left(-c(t_n;\Phi)u_{3NN}(t_n;\omega)\right) \end{aligned} \tag{21}$$

and

$$\begin{aligned} \zeta_t(\omega;\kappa_t) = \frac{1}{\kappa_t}\Bigg(&\sum_{n=1}^{\kappa_t}|u_{1NN}(t_n;\omega) - u_1(t_n)|^2 + \sum_{n=1}^{\kappa_t}|u_{2NN}(t_n;\omega) - u_2(t_n)|^2 \\ &+ \sum_{n=1}^{\kappa_t}|u_{3NN}(t_n;\omega) - u_3(t_n)|^2\Bigg). \end{aligned} \tag{22}$$

The neural network obtains the optimal values for the model parameters. The $u_1(t_n)$, $u_2(t_n)$ and $u_3(t_n)$ represent the loss function's training data. The training process minimizes the residual loss, which measures the discrepancy between the model predictions and the actual differential equations, and the training loss, which measures the discrepancy between the model predictions and the observed data. Finally, the biomass transfer model's time-varying parameters were obtained using PINN with one layer of 10 neurons, 71,800 epochs, the tanh activation function, and a learning rate of 10^{-3}. The PINN Algorithm 3 for learning the optimal parameters of the biomass transfer model (18) is shown below.

Algorithm 3 PINN algorithm for learning the parameters of the biomass transfer model

1: Construct PINN
   Specify the input: $t_n, n = 1, \ldots, X$
   Initialize PINN parameter: ω
   Output layer: $u_{1NN}(t_n;\omega)$, $u_{2NN}(t_n;\omega)$ and $u_{3NN}(t_n;\omega), n = 1, \ldots, X$
2: Construct neural network: a
   Specify the input: $t_n, n = 1, \ldots, X$
   Initialize PINN parameter: Φ
   Output layer: $a(t_n;\Phi), n = 1, \ldots, X$
3: Construct neural network: b
   Specify the input: $t_n, n = 1, \ldots, X$
   Initialize PINN parameter: Φ
   Output layer: $b(t_n;\Phi), n = 1, \ldots, X$
4: Construct neural network: c
   Specify the input: $t_n, n = 1, \ldots, X$
   Initialize PINN parameter: Φ
   Output layer: $c(t_n;\Phi), n = 1, \ldots, X$
5: Specify the training set
   $\kappa = \kappa_r \cup \kappa_b$ of the NN
6: Train the neural network
   Specify a loss function

$$\zeta(\omega;\kappa) = \zeta_t(\omega;\kappa_t) + \zeta_r(\omega;\kappa_r)$$
$$r_1(t_n) = \frac{du_{1NN}(t_n;\omega)}{dt} - \Big(- a(t_n;\Phi)u_{1NN}(t_n;\omega) + b(t_n;\Phi)u_{2NN}(t_n;\omega)\Big)$$
$$r_2(t_n) = \frac{du_{2NN}(t_n;\omega)}{dt} - \Big(- b(t_n;\Phi)u_{2NN}(t_n;\omega) + c(t_n;\Phi)u_{3NN}(t_n;\omega)\Big)$$
$$r_3(t_n) = \frac{du_{3NN}(t_n;\omega)}{dt} - \Big(- c(t_n;\Phi)u_{3NN}(t_n;\omega)\Big)$$

   Minimize $min_\kappa \zeta(\omega;\kappa)$ using Adam Optimizer.
7: Return PINN solution
   $u_{1NN}(t_n;\omega)$, $u_{2NN}(t_n;\omega)$ and $u_{3NN}(t_n;\omega), n = 1, \ldots, X$
8: Return Parameter a:
   $a(t_n;\Phi), n = 1, \ldots, X$
9: Return Parameter b:
   $b(t_n;\Phi), n = 1, \ldots, X$
10: Return Parameter c:
   $c(t_n;\Phi), n = 1, \ldots, X$

The time-varying parameters $a(t)$, $b(t)$, and $c(t)$ are obtained using the above two scenarios. The corresponding Equation (19) system was solved at $t^* = 0.26$, 0.65, and 0.59 for parameters a, b, and c, and Figure 12 shows the obtained exact solution and the predicted output of the biomass transfer model. The 3D surface plot of the actual output of species u_1, u_2 and u_3 and the predicted output of species u_1, u_2 and u_3 is shown in Figure 13. Figure 14 shows the learned time-varying parameter values of the biomass transfer model for different inputs of $a(t), b(t)$, and $c(t)$. The a, b, and c curves look like quadratic functions. The plot of the absolute error between the target and the predictions is shown in Figure 15. Table 11 shows the parameters and error metrics obtained through the approaches described in Scenarios 1 and 2. We observe that Scenario 1 proved more accurate than Scenario 2. The results demonstrate that the PINN approach yields accurate parameter estimates and predictions for the biomass transfer model. The obtained parameters are close to their true values, and the error metrics indicate high accuracy and performance.

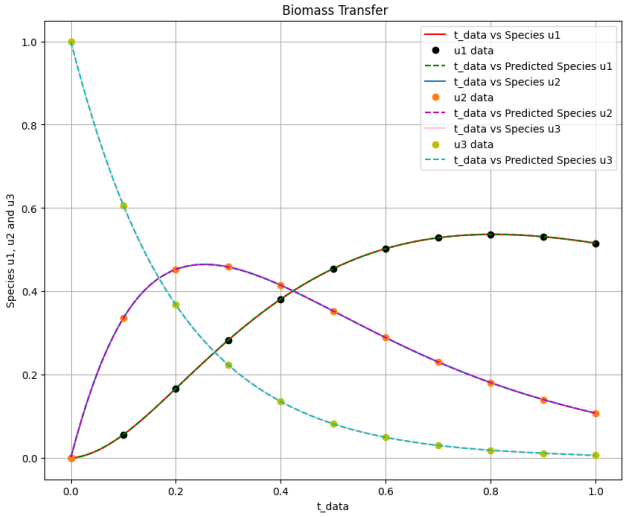
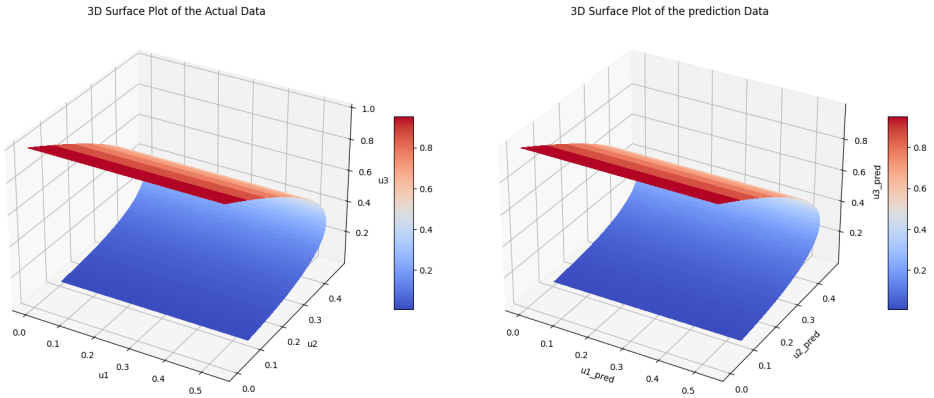


Figure 12. The biomass transfer exact solution and the predicted output of species u_1, u_2, u_3 against t_{data}.



(**a**) (**b**)

Figure 13. The true and the predicted values of species u_1, u_2 and u_3. (**a**) The true values of species U. (**b**) The predicted values of species U.

In addition, the results obtained from the physics-informed neural networks approach are compared with the values and results reported in the literature using deep neural networks [12]. This comparative study was conducted with a configuration of 71,800 epochs and one hidden layer containing 10 neurons. The results are shown in Table 12. The PINN approach outperforms the DNN approach in terms of accuracy and computational efficiency, as evidenced by comparing results and CPU times in Table 12. Furthermore, Table 13 presents a comparative analysis between physics-informed neural networks and deep neural networks, illustrating the differences in their performance across shallow and deep layers. The CPU time of PINN for one, two, and three layers with 40 neurons are 9, 9, and 6 s, while for DNN is 20, 19, and 16 s [12]. The data presented in Table 13 demonstrates that the PINN approach outperforms the DNN approach in terms of accuracy and computational efficiency, as evidenced by comparing results and CPU times, especially in both shallow and deep-layer processing capabilities. Specifically, for a network with one layer of 40 neurons, the PINN model was 55% more efficient than the DNN model. Similarly, for two layers of 40 neurons each, the efficiency improvement was 52.63%, and for three layers, it increased to 62.5%. These results highlight the potential of PINNs in providing faster computational solutions, particularly as the network complexity increases, making them highly suitable for real-time complex system modeling.

Finally, applying physics-informed neural networks in modeling biomass transfer shows promising results in accurately learning time-varying parameters and predicting the ecosystem's behavior. The PINN approach offers advantages over traditional and other neural network approaches, making it a valuable tool for studying complex ecological systems.

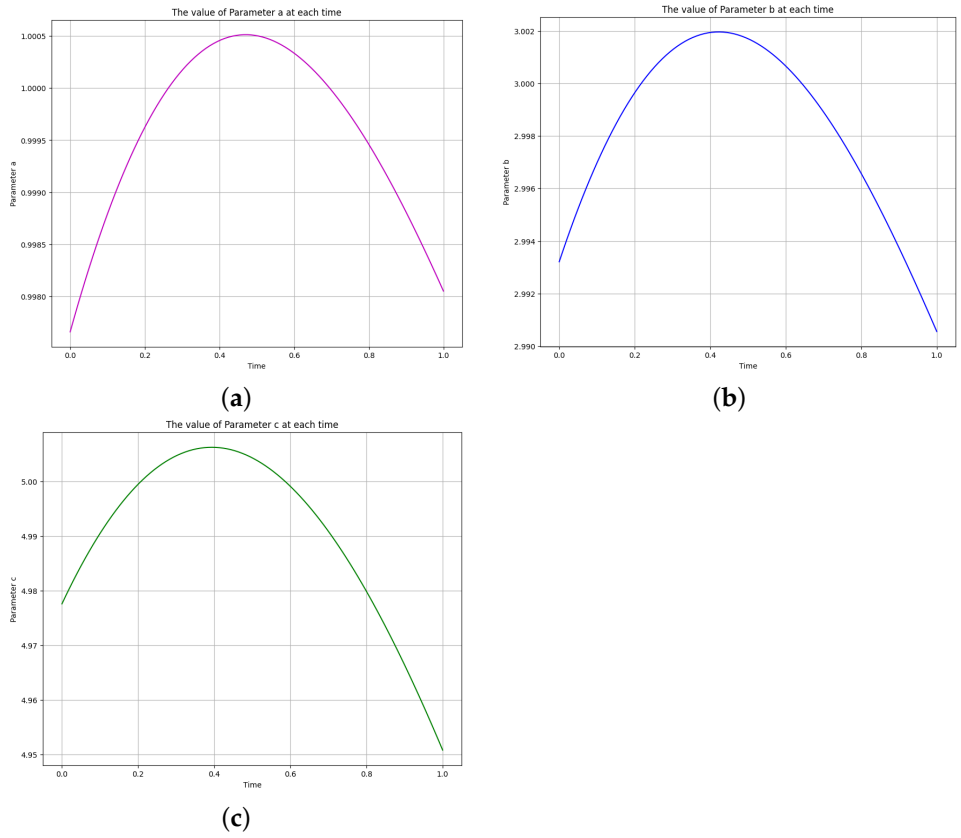


Figure 14. The learned time-varying parameter values of the biomass transfer model. (**a**) Time-varying parameter a. (**b**) Time-varying parameter b. (**c**) Time-varying parameter c.

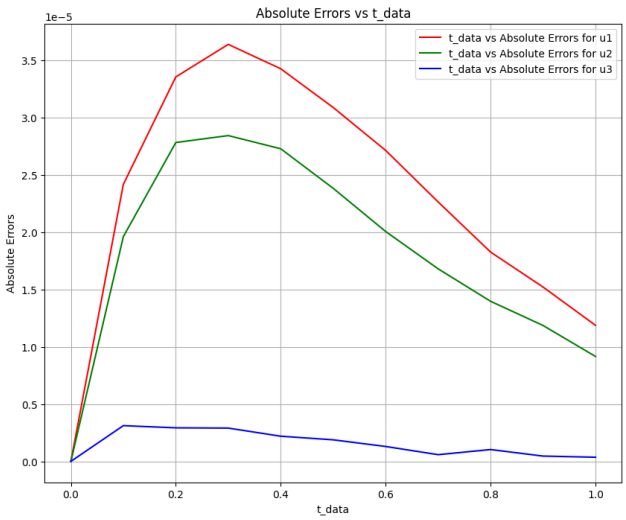


Figure 15. Absolute error plot between the data and PINN solution.

Table 11. Comparison of the two approaches using PINN.

	a	b	c
True Values	1.0	3.0	5.0
Approach 1	0.99999785	2.999931	5.000046
Error of Approach 1	0.00000215	0.000069	0.000046
Approach 2	1.0000874	3.0001755	5.0010123
Error of Approach 1	0.0000874	0.0001755	0.0010123
	u_1	u_2	u_3
MAE of Approach 1	2.3118×10^{-5}	1.8068×10^{-5}	1.5237×10^{-6}
MAE of Approach 2	2.1915×10^{-5}	2.6433×10^{-5}	3.4797×10^{-5}
MSE of Approach 1	6.4652×10^{-10}	3.9821×10^{-10}	3.5008×10^{-12}
MSE of Approach 2	6.9882×10^{-10}	8.9255×10^{-10}	1.8445×10^{-9}
RMSE of Approach 1	2.5427×10^{-5}	1.9955×10^{-5}	1.8709×10^{-6}
RMSE of Approach 2	2.6435×10^{-5}	2.9876×10^{-5}	4.2948×10^{-5}

Table 12. Comparison of obtained results using PINN vs. reported in the literature using DNN.

	True Values	**PINN**	**DNN** [12]	**Epochs**
a	1.0	0.99999785	1.0024	
				71,800
b	3.0	2.999931	3.0026	
				71,800
c	5.0	5.000046	5.0150	

Table 13. Comparison of PINN vs. reported in the literature using DNN [12] based on shallow vs. deep layers.

Layers	**Neurons**	a **(PINN)**	a **(DNN)**	b **(PINN)**	b **(DNN)**	c **(PINN)**	c **(DNN)**	**Epochs**
1	40	1.0000379	1.0044	3.0000386	2.9936	5.0002103	4.9451	20,000
2	40, 40	1.0000603	1.0051	2.9999495	3.0125	4.9998903	5.0515	15,107
3	40, 40, 40	0.99996376	1.0044	3.000064	2.9936	4.999971	4.9451	9800

5.4. Lotka-Volterra Model

The Lotka-Volterra model is a model of the evolution of a prey-predator system. A prey-predator system is a completion where one species, called the predator, has more impact (winning), and the other species, less impact (losing), called the prey. James Lotka and Vito Volterra proposed the Lotka-Volterra model in 1925 and 1926 [31,32]. If $P(t)$ is the prey population and $Q(t)$ the predator population at time t, then the ordinary differential equation of the Lotka-Volterra model is

$$\frac{dP(t)}{dt} = P(t)(a(t) - b(t)Q(t)) \tag{23}$$

$$\frac{dQ(t)}{dt} = Q(t)(c(t)P(t) - d(t)) \tag{24}$$

where variable $P(t), Q(t), a(t), b(t), c(t)$ and $d(t)$ represents

$P(t)$: the size of the prey population per time
$Q(t)$: the size of the predator population
$a(t)$: the prey per capital rate of increase per time
$b(t)$: the capture efficiency per time
$c(t)$: the conversion efficiency per time
$d(t)$: the mortality rate per time

The parameters $a(t), b(t), c(t), d(t)$ are non-negative.

The Lotka-Volterra model has an adequate system model up to some parameters that need to be determined. To solve the Lotka-Volterra model, you can use numerical methods or analytical techniques. A physics-informed neural network approach is proposed in this case. The PINN combines neural networks with physical laws or equations to learn the model's time-varying parameters from data. Following the same PINN architecture shown in Figure 3, we offer a PINN algorithm with five networks to do this. The first network outputs are $P_{NN}(t_n;\omega)$ and $Q_{NN}(t_n;\omega)$, which admits t as the input data. The second, third, fourth, and fifth networks learn the parameters $a(t_n;\Phi)$, $b(t_n;\Phi)$, $c(t_n;\Phi)$ and $d(t_n;\Phi)$ from the data. We define the residual loss $\zeta_r(\omega;\kappa_r)$ and training loss $\zeta_t(\omega;\kappa_t)$ as follows to quantify the discrepancy between the model predictions and the actual data. The PINN algorithm minimizes the combined loss function to learn the optimal parameters and obtain the solution of the Lotka-Volterra model.

$$\zeta_r(\omega;\kappa_r) = \frac{1}{\kappa_r}\sum_{i=1}^{2}\sum_{n=1}^{\kappa_r}|r_i(t_n)|^2. \tag{25}$$

where

$$\begin{aligned} r_1(t) &= \frac{dP_{NN}(t_n;\omega)}{dt} - \Big(P_{NN}(t_n;\omega)(a(t_n;\Phi) - b(t_n;\Phi)Q_{NN}(t_n;\omega))\Big) \\ r_2(t) &= \frac{dQ_{NN}(t_n;\omega)}{dt} - \Big(Q_{NN}(t_n;\omega)(c(t_n;\Phi)P_{NN}(t_n;\omega) - d(t_n;\Phi))\Big) \end{aligned} \tag{26}$$

and

$$\zeta_t(\omega;\kappa_t) = \frac{1}{\kappa_t}\left(\sum_{n=1}^{\kappa_t}|P_{NN}(t_n;\omega) - P(t_n)|^2 + \sum_{n=1}^{\kappa_t}|Q_{NN}(t_n;\omega) - Q(t_n)|^2\right) \tag{27}$$

To generate our measurement data, we numerically solve (23) and (24), utilizing an initial condition of $(P_0, Q_0) = (0.2, 0.3)$. We selected parameter values as follows: $a = 1, b = 2, c = 1$, and $d = 0.3$. The problem was resolved over a period delineated by the interval $[0, T]$, where T is set to 13, a duration that approximately encapsulates one cycle. We employ a $N_x = 100$ points grid within this temporal scope. Our measurement data is procured by extracting the numerical solution vector at every alternate time step. As a result of this method, we obtain a dataset where $N = 50$ [33]. Finally, the Lotka-Volterra model parameters were obtained using PINN and the approaches from Scenario 2 with three hidden layers, 64 neurons per layer, 50,000 epochs, the sigmoid activation function, and a learning rate of 10^{-3}. The PINN Algorithm 4 for learning the optimal parameters of the Lotka-Volterra model is shown below.

After the parameters were learned using the same initial condition used to produce the data, the results that are obtained are then compared with the true values of the parameters and the actual data. Figures are provided to visualize the predicted prey and predator populations, the phase space plot, and the absolute error between the target and predicted values. The tables present the initial and obtained parameters and error metrics such as MAE, MSE, and RMSE.

Algorithm 4 PINN algorithm for learning the parameters of Lotka-Volterra model

1: Construct PINN
   Specify the input: $t_n, n = 1, \ldots, X$
   Initialize PINN parameter: ω
   Output layer: $P_{NN}(t_n; \omega)$ and $Q_{NN}(t_n; \omega)$, $n = 1, \ldots, X$
2: Construct neural network: a
   Specify the input: $t_n, n = 1, \ldots, X$
   Initialize PINN parameter: Φ
   Output layer: $a(t_n; \Phi), n = 1, \ldots, X$
3: Construct neural network: b
   Specify the input: $t_n, n = 1, \ldots, X$
   Initialize PINN parameter: Φ
   Output layer: $b(t_n; \Phi), n = 1, \ldots, X$
4: Construct neural network: c
   Specify the input: $t_n, n = 1, \ldots, X$
   Initialize PINN parameter: Φ
   Output layer: $c(t_n; \Phi), n = 1, \ldots, X$
5: Construct neural network: d
   Specify the input: $t_n, n = 1, \ldots, X$
   Initialize PINN parameter: Φ
   Output layer: $d(t_n; \Phi), n = 1, \ldots, X$
6: Specify the training set
   $\kappa = \kappa_r \cup \kappa_b$ of the NN
7: Train the neural network
   Specify a loss function

$$\zeta(\omega; \kappa) = \zeta_t(\omega; \kappa_t) + \zeta_r(\omega; \kappa_r)$$
$$r_1(t) = \frac{dP_{NN}(t_n; \omega)}{dt} - \Big(P_{NN}(t_n; \omega)(a(t_n; \Phi) - b(t_n; \Phi)Q_{NN}(t_n; \omega))\Big)$$
$$r_2(t) = \frac{dQ_{NN}(t_n; \omega)}{dt} - \Big(Q_{NN}(t_n; \omega)(c(t_n; \Phi)P_{NN}(t_n; \omega) - d(t_n; \Phi))\Big)$$

   Minimize $min_\kappa \zeta(\omega; \kappa)$ using Adam Optimizer.
8: Return PINN solution
   $P_{NN}(t_n; \omega)$ and $Q_{NN}(t_n; \omega)$, $n = 1, \ldots, X$
9: Return Parameter a:
   $a(t_n; \Phi), n = 1, \ldots, X$
10: Return Parameter b:
   $b(t_n; \Phi), n = 1, \ldots, X$
11: Return Parameter c:
   $c(t_n; \Phi), n = 1, \ldots, X$
12: Return Parameter d:
   $d(t_n; \Phi), n = 1, \ldots, X$

Figure 16 displays the resulting solution of the actual and predicted output of the Lotka-Volterra model. Figure 17 shows the phase space plot of the anticipated output of species $P(t)$ versus species $Q(t)$ and the actual output of species $P(t)$ against species $Q(t)$. Figure 18 displays the Lotka-Volterra model's learned time-varying parameter values for various inputs of P_{NN} and Q_{NN}.

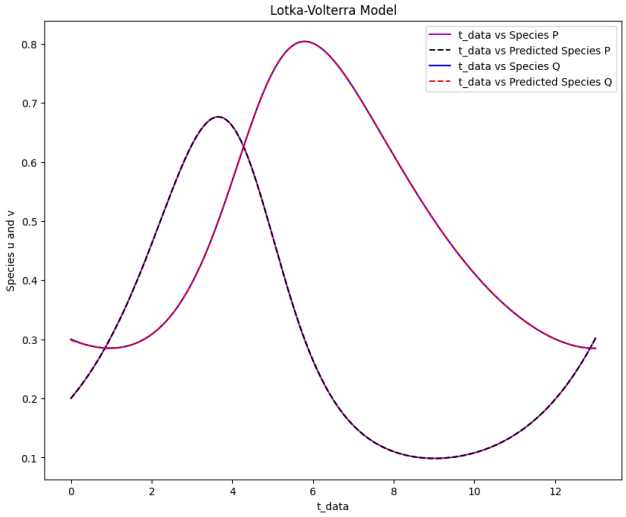


Figure 16. The Lotka-Volterra model solution for the real output of species P and Q against time data and the predicted output of species P and Q against time data.

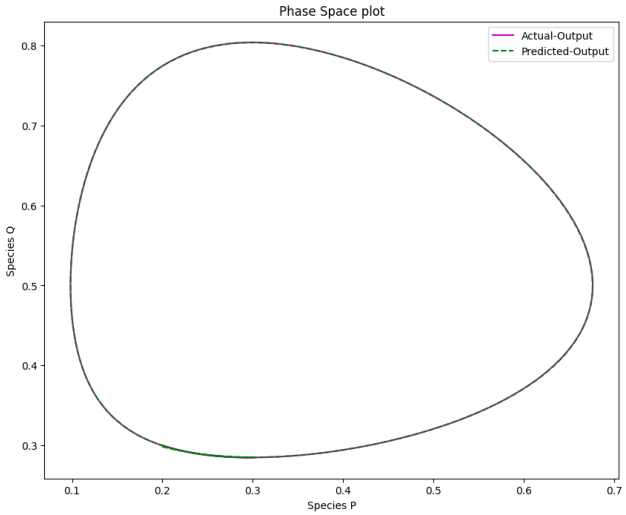


Figure 17. The phase space plot of the actual output of species P against species Q and the predicted output of species P against species Q of the Lotka-Volterra model.

The learned time-varying parameters are $a(t)$, $b(t)$, $c(t)$, and $d(t)$. It was observed that the functional form of the curve of $a(t)$ is linear, while the functional forms of $b(t)$, $c(t)$, and $d(t)$ are

$$b(t) = B\ sin(\lambda t + \alpha) + k$$
$$c(t) = C\ sin(\lambda t + \alpha) + k$$
$$d(t) = D\ sin(\lambda t + \alpha) + k$$

where B, C and D are the amplitude, λ determines the frequency, α is the phase shift, and K is the vertical shift (which should be close to the initial values, given the curve's behavior).

Figure 19 plots the absolute error between the target and the Lotka-Volterra model's predictions. Tables 14 and 15 show the initial and the obtained parameters for the Lotka-Volterra model and the error metrics through the approaches described in Scenario 2.

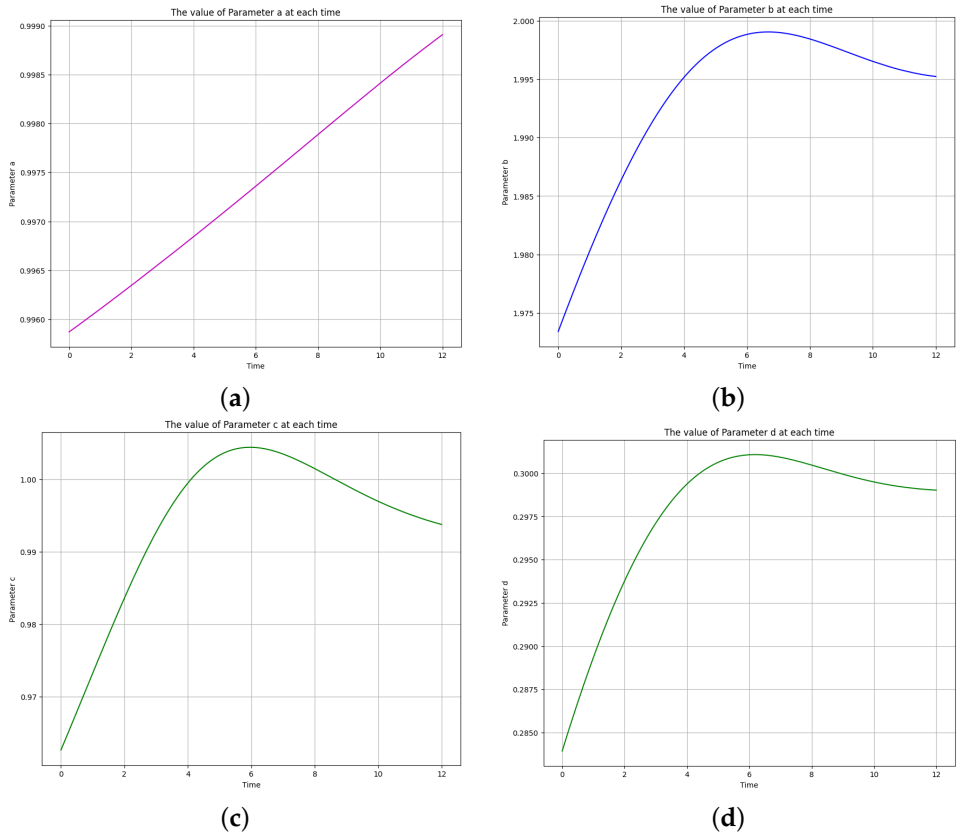


Figure 18. The learned time-varying parameter values of the Lotka-Volterra model. (**a**) Time-varying parameter a. (**b**) Time-varying parameter b. (**c**) Time-varying parameter c. (**d**) Time-varying parameter d.

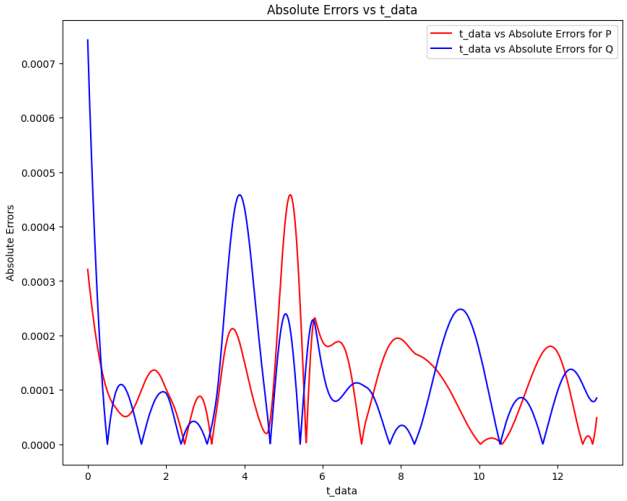


Figure 19. Absolute error plot between the data and PINN solution of the Lotka-Volterra model.

Table 14. The parameter estimation of the Lotka-Volterra model.

	a	***b***	***c***	***d***
True Values	1.0	2.0	1.0	0.3
Approach 2	0.999136	1.9990387	1.000002	0.3
Error of Approach 2	0.000864	0.0009619	0.000002	0.00

Table 15. The error metrics of the Lotka-Volterra model.

	P	Q
MAE	2.3452×10^{-4}	2.4140×10^{-4}
MSE	8.5244×10^{-8}	1.0660×10^{-7}
RMSE	2.9197×10^{-4}	3.2649×10^{-4}

Furthermore, our findings, obtained through the utilization of physics-informed neural networks (PINN), were compared with those reported in scholarly literature employing deep neural networks (DNN) [33]. As illustrated in Table 16, this comparison involved using a single hidden layer encompassing 20 neurons and a sigmoid activation function over a training span of 50,000 epochs. The comparison shows that PINNs perform better than DNNs regarding parameter estimation for the Lotka-Volterra model.

Table 16. Comparison of the obtained results using PINN vs. those reported in the literature using DNN for the Lotka-Volterra model.

	True Values	**PINN**	**DNN** [33]	**Epochs**
a	1.0	0.999876	0.9931	50,000
b	2.0	1.999771	1.9860	50,000
c	1.0	0.999869	0.9946	50,000
d	0.3	0.300004	0.2984	

5.5. SIR Model

The SIR model is a widely used mathematical model in epidemiology to understand the spread of infectious diseases within a population. It divides the population into three compartments: susceptible (S), infected (I), and recovered (R). In this model, individuals can transition between these compartments by interacting with infected others [2,34]. The SIR model assumes that the population size is constant, meaning no births, deaths, or migrations occur during the disease outbreak. Additionally, it assumes that individuals in the population mix randomly, and there is a homogeneous mixing pattern. A set of ordinary differential equations can describe the dynamics of the SIR model. Let us denote the number of susceptible individuals as $S(t)$, the number of infected individuals as $I(t)$, and the number of recovered individuals as $R(t)$. The following equations give the rates of change of these compartments over time:

$$\begin{aligned} \frac{dS}{dt} &= \frac{-\beta(t)S(t)I(t)}{N} \\ \frac{dI}{dt} &= \frac{\beta(t)S(t)I(t)}{N} - \gamma(t)I(t) \\ \frac{dR}{dt} &= \gamma(t)I(t). \end{aligned} \tag{28}$$

where variable $S(t)$, $I(t)$, $R(t)$, N, $\beta(t)$ and $\gamma(t)$ represents

$S(t)$: the numbers of susceptible individuals at time t
$I(t)$: the numbers of infected individuals at time t
$R(t)$: the numbers of individuals recovered at time t
N : the total population size.
$\beta(t)$: the transmission rate per time
$\gamma(t)$: the recovery rate per time.

The continuity equation is given by

$$N = S(t) + I(t) + R(t),\ t \geq t_0$$

where the initial conditions are denoted by $S(t_0) = S_0, I(t_0) = I_0$ and $R(t_0) = R_0$, where $t \geq t_0$ represents time in days and t_0 is the start date of the pandemic in the model. The SIR model provides insights into the dynamics of an infectious disease outbreak, such as the peak number of infected individuals, the duration of the epidemic, and the overall fraction of the infected population. These quantities depend on the model parameters β and γ values. It is important to note that the SIR model makes several simplifying assumptions. For instance, it assumes that the population is well-mixed, which may be false. We aim to learn the time-varying parameter $\beta(t)$ and $\gamma(t)$ of the SIR model from real-life data (COVID-19) using PINN. Following the same procedure as in the previous application, we present PINN Algorithm 5 with three networks to do this. The first network outputs are $S_{NN}(t_n;\omega)$, $I_{NN}(t_n;\omega)$ and $R_{NN}(t_n;\omega)$, which admits t as the input data. The second and third networks learn the parameters $\beta(t_n;\Phi)$ and $\gamma(t_n;\Phi)$ from the data, where ω and Φ represent the biases and weights of the network that minimize the loss function.

Algorithm 5 PINN algorithm for learning the parameters of the SIR model

1: Construct PINN
   Specify the input: $t_n, n = 1, \ldots, X$
   Initialize PINN parameter: ω
   Output layer: $S_{NN}(t_n;\omega)$, $I_{NN}(t_n;\omega)$ and $R_{NN}(t_n;\omega)$, $n = 1, \ldots, X$
2: Construct neural network: β
   Specify the input: $t_n, n = 1, \ldots, X$
   Initialize PINN parameter: Φ
   Output layer: $\beta(t_n;\Phi), n = 1, \ldots, X$
3: Construct neural network: γ
   Specify the input: $t_n, n = 1, \ldots, X$
   Initialize PINN parameter: Φ
   Output layer: $\gamma(t_n;\Phi), n = 1, \ldots, X$
4: Specify the training set
   Training data: using cubic spline to generate $I(t_i), R(t_i); i = 1, \ldots, X$ given from the dataset.
5: Train the neural network
   Specify a loss function

$$r_1(t) = \frac{dS_{NN}(t)}{dt} - \left(\frac{-\beta(t_n;\Phi)S_{NN}(t_n;\omega)I_{NN}(t_n;\omega)}{N}\right)$$
$$r_2(t) = \frac{dI_{NN}(t_n;\omega)}{dt} - \left(\frac{\beta(t_n;\Phi)S_{NN}(t_n;\omega)I_{NN}(t_n;\omega)}{N} - \gamma(t_n;\Phi)I_{NN}(t_n;\omega)\right)$$
$$r_3(t) = \frac{dR_{NN}(t_n;\omega)}{dt} - \left(\gamma(t_n;\Phi)I_{NN}(t_n;\omega)\right)$$

   Minimize $min_\kappa \zeta(\omega;\kappa)$ using Adam Optimizer.
6: Return PINN solution
   $S_{NN}(t_n;\omega)$, $I_{NN}(t_n;\omega)$ and $R_{NN}(t_n;\omega)$, $n = 1, \ldots, X$
7: Return Parameter β:
   $\beta(t_n;\Phi), n = 1, \ldots, X$
8: Return Parameter γ:
   $\gamma(t_n;\Phi), n = 1, \ldots, X$

We define the residual loss $\zeta_r(\omega;\kappa_r)$ and training loss $\zeta_t(\omega;\kappa_t)$ as follows:

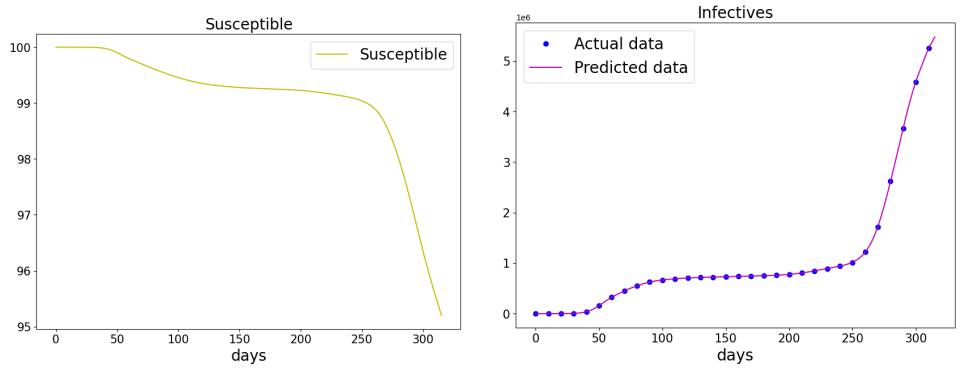
$$\zeta_r(\omega;\kappa_r) = \frac{1}{\kappa_r}\sum_{i=1}^{3}\sum_{n=1}^{\kappa_r} ||r_i(t_n)||_2^2. \tag{29}$$

where

$$\begin{aligned} r_1(t) &= \frac{dS_{NN}(t)}{dt} - \left(\frac{-\beta(t_n;\Phi)S_{NN}(t_n;\omega)I_{NN}(t_n;\omega)}{N}\right) \\ r_2(t) &= \frac{dI_{NN}(t)}{dt} - \left(\frac{\beta(t_n;\Phi)S_{NN}(t_n;\omega)I_{NN}(t_n;\omega)}{N} - \gamma(t_n;\Phi)I_{NN}(t_n;\omega)\right) \\ r_3(t) &= \frac{dR_{NN}(t)}{dt} - \left(\gamma(t_n;\Phi)I_{NN}(t_n;\omega)\right) \end{aligned} \tag{30}$$

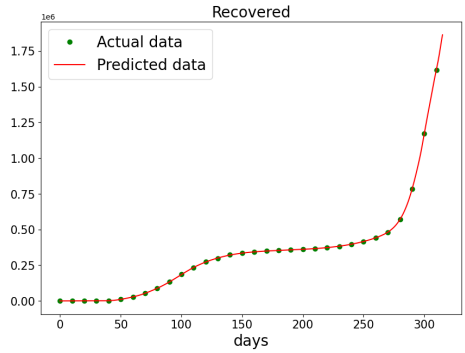
$$\begin{aligned} \zeta_t(\omega;\kappa_t) = \frac{1}{\kappa_t}\Bigg(&\sum_{n=1}^{\kappa_t} ||S_{NN}(t_n;\omega) - S(t_n)||_2^2 + \sum_{n=1}^{\kappa_t} ||I_{NN}(t_n;\omega) - I(t_n)||_2^2 \\ &+ \sum_{n=1}^{\kappa_t} ||R_{NN}(t_n;\omega) - R(t_n)||_2^2\Bigg). \end{aligned} \tag{31}$$

Here, we use data from Italy [35] starting from the date of the first reported cases to the day before vaccination data were reported, which is from 31st of January to 11 December 2020. We take the total population N to be 59.44×10^6 in Italy. Cubic spline interpolation generates 2000 training points from the cumulative infection and recovered data. The cumulative infections and recovered data are matched against the cumulative and recovered learned solutions. Algorithm 5 was implemented using publicly available COVID-19 data [35]. The parameters of the data using the SIR model and the learned cumulative infection and recovered data were obtained after using PINN and the approach of Scenario 2 with five hidden layers, 64 neurons per layer, 100,000 epochs, the tanh activation function was used, and a learning rate of 10^{-3}. Figure 20 shows the learned solution of the SIR model, comparing the actual and predicted outputs of the infected and recovered populations. The cumulative data aligns closely with an exponential function. Figure 21 displays the learned values of $\beta(t)$ and $\gamma(t)$ using PINN. The phase space plot in Figure 22 illustrates the relationship between the actual and predicted infected and recovered populations.



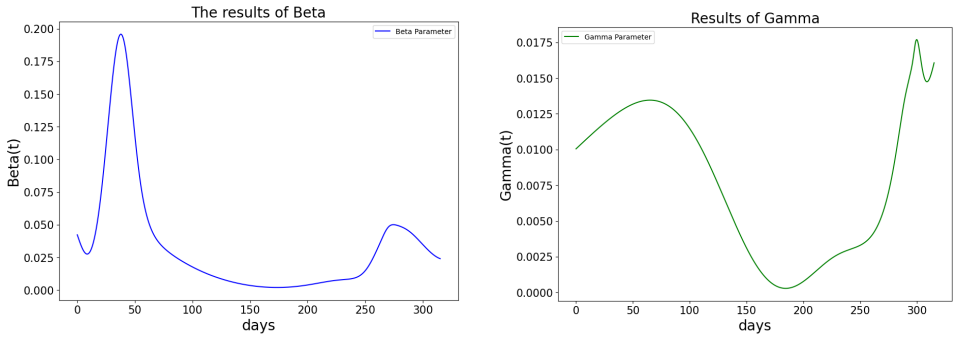
(**a**) (**b**)

Figure 20. *Cont.*



(**c**)

Figure 20. The data and the learned SIR model using PINN Algorithm 5 on COVID-19 data. (**a**) The susceptible graph. (**b**) The data and the learned infectives. (**c**) The data and the learned recovered population.



(**a**) (**b**)

Figure 21. The learned parameters of SIR model using PINN Algorithm 5 on COVID-19 data. (**a**) The learned β. (**b**) The learned γ.

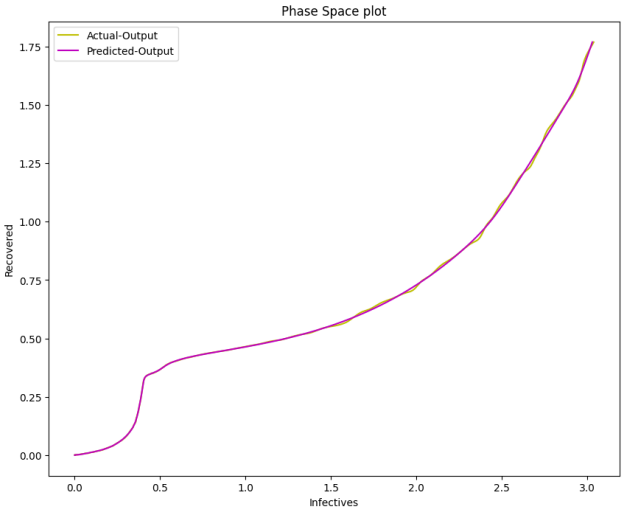


Figure 22. The phase space plot of the actual output of I against R and the predicted output of I against R of the SIR model from the COVID-19 data.

Figure 23 depicts the absolute error between the target and predicted values of the SIR model using PINN. Table 17 summarizes the error metrics for the SIR model, indicating the accuracy of the PINN solution. The graph displays the absolute errors for two datasets, "I" and "R" over time. After day 250, there is a noticeable spike in errors for both datasets. This surge might be attributed to data inconsistencies, external factors impacting the mea-

surements, or potential limitations in the predictive model if one was used. Additionally, increased variability in the data or unforeseen shifts in the underlying system around day 250 could also be factors. A thorough examination of the data collection process and any external events during that period is essential for a definitive conclusion. The metrics include mean absolute error (MAE), mean squared error (MSE), and root mean squared error (RMSE). These metrics show that the PINN solution closely approximates the COVID-19 data, demonstrating the approach's effectiveness. Finally, learning time-varying parameters from real-life data using PINN enhances our understanding of disease dynamics and can aid in making informed decisions regarding public health interventions. This approach can be applied to various infectious diseases, allowing for more accurate modeling and prediction of their spread. Combining the SIR model and PINN presents a valuable tool for studying and managing epidemics.

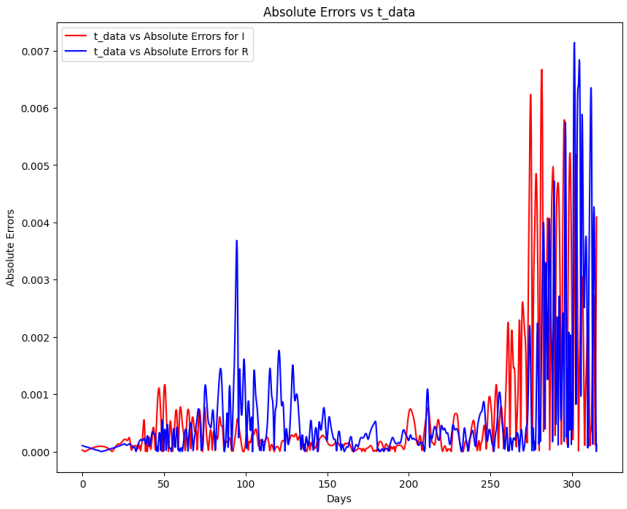


Figure 23. Absolute error plot between the COVID-19 data and the PINN solution of the SIR model.

Table 17. The error metrics of the SIR model.

	I	R
MAE	1.3053×10^{-3}	1.2820×10^{-3}
MSE	6.9586×10^{-6}	6.0906×10^{-6}
RMSE	2.6379×10^{-3}	2.4679×10^{-4}

6. Conclusions

This study has systematically explored the capabilities of physics-informed neural networks in modeling dynamic systems, marking a significant contribution to the field. Our research has highlighted the enhanced computational efficiency and accuracy of physics-informed neural networks, especially when compared to traditional deep neural networks. Physics-informed neural networks are very good at staying consistent with basic scientific laws because they use physical principles directly in the learning process. This is very important when working with complex dynamic systems. The novel contributions of this research lie in optimizing physics-informed neural networks for scenarios where the number of unknowns corresponds to the number of undetermined parameters. This approach has not only streamlined the process of dynamic system modeling but has also opened new avenues for research in this field. The application of our methodology to real-world scenarios, such as the COVID-19 pandemic model, further validates its practicality and relevance.

Quantitative comparisons have been a cornerstone of this study, providing clear evidence of the superiority of physics-informed neural networks over deep neural networks. We have demonstrated this through various error metrics. Additionally, computational

efficiency has been quantitatively assessed, revealing that physics-informed neural networks significantly improve processing speed, an aspect critical for real-time applications. Finally, our results show that physics-informed neural networks have a lot of potential as strong, fast, and accurate scientific modeling tools that can deal with the complexities of dynamic systems. This research opens up new possibilities for future studies. It shows how important it is to keep looking into and improving complex neural network designs for use in science. A key area for future work is to determine how to learn the time-dependent parameters of stiff dynamical systems.

The code for the work can be found on **github** (accessed on 23 November 2023).

Author Contributions: All the authors have contributed equally to the conceptualization, software, validation, writing the original draft preparation, and visualization. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: Data are contained within the article.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Qureshi, S.; Yusuf, A. Mathematical modeling for the impacts of deforestation on wildlife species using Caputo differential operator. *Chaos Solitons Fractals* **2019**, *126*, 32–40. [CrossRef]
2. Kermack, W.O.; McKendrick, A.G. A contribution to the mathematical theory of epidemics. *Proc. R. Soc. Lond. Ser. Contain. Pap. Math. Phys. Character* **1927**, *115*, 700–721.
3. Dua, V. An artificial neural network approximation based decomposition approach for parameter estimation of system of ordinary differential equations. *Comput. Chem. Eng.* **2011**, *35*, 545–553. [CrossRef]
4. Ning, X.; Jia, L.; Wei, Y.; Li, X.; Chen, F. Epi-DNNs: Epidemiological priors informed deep neural networks for modeling COVID-19 dynamics. *Comput. Biol. Med.* **2023**, *158*, 106693. [CrossRef] [PubMed]
5. Varziri, M.S.; Poyton, A.A.; McAuley, K.B.; McLellan, P.J.; Ramsay, J.O. Selecting optimal weighting factors in iPDA for parameter estimation in continuous-time dynamic models. *Comput. Chem. Eng.* **2008**, *32*, 3011–3022. [CrossRef]
6. Kalogerakis, N.; Luus, R. Improvement of Gauss-Newton method for parameter estimation through the use of information index. *Ind. Eng. Chem. Fundam.* **1983**, *22*, 436–445. [CrossRef]
7. Voss, H.; Timmer, J.; Kurths, J. Nonlinear dynamical system identification from uncertain and indirect measurements. *Int. J. Bifurc. Chaos Appl. Sci. Eng.* **2004**, *14*, 1905–1933. [CrossRef]
8. Ge, Y.; Zhang, W.; Wu, X.; Ruktanonchai, C.W.; Liu, H.; Wang, J. Untangling the changing impact of non-pharmaceutical pharmaceutical interventions and vaccination on European Covid-19 trajectories. *Nat. Commun.* **2022**, *13*, 3106. [CrossRef]
9. Xue, L.; Jing, S.; Miller, J.C.; Sun, W.; Li, H.; Estrada-Franco, J.G.; Hyman, J.M.; Zhu, H. A data-driven network model for the emerging COVID-19 epidemics in Wuhan, Toronto, and Italy. *Math. Biosci.* **2020**, *326*, 108391. [CrossRef]
10. Viguerie, A.; Lorenzo, G.; Auricchio, F.; Baroli, D.; Hughes, T.J.; Patton, A.; Reali, A.; Yankeelov, T.E.; Veneziani, A. Simulating the spread of COVID-19 via a spatially-resolved susceptible-exposed-infected-recovered-deceased (SEIRD) model with heterogeneous diffusion. *Appl. Math. Lett.* **2021**, *111*, 106617. [CrossRef]
11. Baden, N.; Villadsen, J. A family of collocation-based methods for parameter estimation in differential equations. *Chem. Eng. J.* **1982**, *23*, 1–13. [CrossRef]
12. Temesgen, D.T.; Chimdessa, S.Y.; Carlos, F.P. Parameter Estimation for Dynamical Systems Using a Deep Neural Network. *Appl. Comput. Intell. Soft Comput.* **2022**, *2022*, 2014510.
13. Temesgen, D.T. Deep neural network for system of ordinary differential equations: Vectorized algorithm and simulation. *Mach. Learn. Appl.* **2021**, *5*, 100058.
14. McCulloch, W.S.; Pitts, W. A logical calculus of the ideas immanent in nervous activity. *Bull. Math. Biophys.* **1943**, *5*, 115–133. [CrossRef]
15. Minaee, S.; Boykov, Y.Y.; Porikli, F.; Plaza, A.J.; Kehtarnavaz, N.; Terzopoulos, D. Image segmentation using deep learning: A survey. *IEEE Trans. Pattern Anal. Mach. Intell.* **2022**, *44*, 3523–3542. [CrossRef] [PubMed]
16. Dong, S.; Wang, P.; Abbas, K. A survey on deep learning and its applications. *Comput. Sci. Rev.* **2021**, *40*, 100379. [CrossRef]
17. Dixit, P.; Silakari, S. Deep learning algorithms for cybersecurity applications: A technological and status review. *Comput. Sci. Rev.* **2021**, *39*, 100317. [CrossRef]
18. Raissi, M.; Perdikaris, P.; Karniadakis, G.E. Physics informed deep learning: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *J. Comput. Phys.* **2019**, *378*, 686–707. [CrossRef]
19. Oluwasakin, E.O.; Khaliq, A.Q.M. Driven Deep Learning Neural Networks for Predicting the Number of Individuals Infected by COVID-19 Omicron Variant. *Epidemiologia* **2023**, *4*, 420–453. [CrossRef]

20. Torku, T.K.; Khaliq, A.Q.M.; Furati, K.M. Deep-Data-Driven Neural Networks for COVID-19 Vaccine Efficacy. *Epidemiologia* **2021**, *2*, 564–586. [CrossRef]
21. Long, J.; Khaliq, A.; Furati, K. Identification and prediction of time-varying parameters of COVID-19 model: A data-driven deep learning approach. *Int. J. Comput. Math.* **2021**, *98*, 1617–1632. [CrossRef]
22. Olumoyin, K.; Khaliq, A.; Furati, K. Data-Driven Deep-Learning Algorithm for Asymptomatic COVID-19 Model with Varying Mitigation Measures and Transmission Rate. *Epidemiologia* **2021**, *2*, 471–489. [CrossRef] [PubMed]
23. Eyring, H.; Polanyi, M.Z. Simple gas reactions. *J. Phys. Chem. B* **1931**, *12*, 279–311.
24. Esposito, W.R.; Floudas, C.A. Global optimization for the parameter estimation of differential-algebraic systems. *Ind. Eng. Chem. Res.* **2002**, *39*, 1291–1310. [CrossRef]
25. Katare, S.; Bhan, A.; Caruthers, J.M.; Delgass, W.N.; Venkatasubramanian, V. A hybrid genetic algorithm for efficient parameter estimation of large kinetic models. *Comput. Chem. Eng.* **2004**, *28*, 2569–2581. [CrossRef]
26. Prigogine, I.; Lefever, R. Symmetry breaking instabilities in dissipative systems II. *J. Chem. Phys.* **1968**, *48*, 1665–1700. [CrossRef]
27. Lv, Y.; Liu, Z. Turing-Hopf bifurcation analysis and normal form of a diffusive Brusselator model with gene expression time delay. *Chaos Solitons Fractals* **2021**, *152*, 111478. [CrossRef]
28. Domguia, U.S.; Tchakui, M.V.; Herve, S.; Woafo, P. Theoretical and Experimental Study of an Electromechanical System Actuated by a Brusselator Electronic Circuit Simulator. *Vib. Acoust.* **2017**, *139*, 061017. [CrossRef]
29. Field, R.J.; Noyes, R.M. Oscillations in chemical systems. IV. Limit cycle behavior in a model of a real chemical reaction. *Chem. Phys.* **1974**, *60*, 1877–1884. [CrossRef]
30. Gustafson, G.B. *Differential Equations and Linear Algebra, Undergraduate Mathematics Science and Engineerin*; Amazon Kindle Direct Publishing: Seattle, WA, USA, 2022; pp. 1–1729; ISBN 9798705491124/9798711123651.
31. Lotka, A.J. *Elements of Physical Biology*; Williams and Wilkins Company: Philadelphia, PA, USA, 1925.
32. Volterra, V. Variazionie fluttuazioni del numero d'individui in specie animali conviventi. *Mem. R. Accad. Naz. Lincei* **1926**, *2*, 31–113.
33. Borzì, A. *Modelling with Ordinary Differential Equations: A Comprehensive Approach*; Taylor and Francis Group, LLC: Abingdon, UK, 2020; Volume 1, pp. 1–387.
34. Fred, B.; Carlos, C. *Mathematical Models in Population Biology and Epidemiology*; Springer: New York, NY, USA, 2012; Volume 40, pp. 1–4.
35. Dong, E.; Du, H.; Gardner, L. An interactive web-based dashboard to track COVID-19 in real time. *Lancet Infect. Dis.* **2020**, *20*, 533–534. [CrossRef] [PubMed]