Міністерство освіти і науки України
Національний університет «Києво-Могилянська академія»
Факультет інформатики
Кафедра математики

Кваліфікаційна робота

освітній ступінь – бакалавр

на тему: **«Combining machine learning with physical laws to solve inverse problems»**

Виконав: студент 4-го року навчання
освітньої програми «Прикладна математика»,
спеціальності 113 Прикладна математика

Надєєв Ілля Олексійович

Керівник: Крюкова Г.В.
кандидат фіз.-мат. наук, доцент

Рецензент: -

Кваліфікаційна робота захищена
з оцінкою _____

Секретар ЕК_____
(підпис)

«_____» _____ 20___р.

Київ - 2026

Міністерство освіти і науки України
Національний університет «Києво-Могилянська академія»
Факультет інформатики
Кафедра математики

ЗАТВЕРДЖУЮ
Зав.кафедри математики,
доцент, кандидат фіз.-мат. наук
_____ *Чорней Р.К.*
(підпис)
“____”_____ 2025

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ
для кваліфікаційної роботи
студенту 4-го курсу, факультету інформатики
Надєєву Іллі Олексійовичу

Тема: «Combining ML with physical laws to solve inverse problems»

Зміст кваліфікаційної роботи:

Анотація

1. Вступ
2. Диференціальні рівняння та обернені задачі.
3. Нейронні мережі та фізико-орієнтовані нейронні мережі.
4. Розв’язання оберненої задачі за допомогою фізико-орієнтованої нейронної мережі.

Висновки

Список літератури

Дата видачі “____”_____ 2025 Керівник _____
(підпис)
Завдання отримав _____
(підпис)

Графік підготовки кваліфікаційної роботи до захисту

Графік узгоджено «_____» _____ 2025р.

№ з/п	Перелік робіт	Термін виконання етапу	Підпис наукового керівника	Дата ознайомлення наукового керівника	Примітка
1.	Отримання теми кваліфікаційної роботи.	19.09.2025			
2.	Ознайомлення з темою кваліфікаційної роботи.	01.10.2025			
3.	Консультація з науковим керівником.	15.10.2025			
4.	Робота з науковою літературою.	15.01.2026			
5.	Вивчення джерел літератури.	20.02.2026			
6.	Робота над текстовим оформленням теоретичної частини та одержаних результатів.	26.02.2026			
7.	Попередній аналіз кваліфікаційної роботи. Виправлення помилок.	04.04.2026			
8.	Попередній захист кваліфікаційної роботи.	14.05.2026			
9.	Захист кваліфікаційної роботи.	04.06.2026			

Науковий керівник: Крюкова Галина Віталіївна.

Виконавець кваліфікаційної роботи: Надєєв Ілля Олексійович.

Contents

Annotation

This bachelor's thesis investigates the use of neural-network-based methods for solving forward and inverse problems in dynamical systems. The work considers neural networks (NNs) and Physics-Informed Neural Networks (PINNs).

The thesis provides a description of differential equations, inverse problems, and the theoretical foundations of Physics-Informed Neural Networks. The practical part considers Euler's planar two-fixed-centers problem as a model dynamical system for studying forward and inverse PINN formulations. Numerical experiments are conducted for both forward trajectory reconstruction and inverse parameter recovery.

Keywords: Machine learning, Neural networks, Physics-informed neural networks, Inverse problems, Differential equations, Dynamical systems, Euler's two-fixed-centers problem, Parameter identification.

Анотація

В бакалаврській роботі досліджено використання нейронних мереж для розв'язання прямих та обернених задач у динамічних системах. Розглянуто нейронні мережі (NNs) та фізико-орієнтовані нейронні мережі (PINNs).

У роботі подано опис диференціальних рівнянь, обернених задач та теоретичних основ фізико-орієнтованих нейронних мереж. У практичній частині розглянуто задачу Ейлера про два нерухомі центри як модельну динамічну систему для побудови прямої та оберненої постановки. Чисельні експерименти проведено як для продовження траєкторії у прямій задачі, так і для відновлення невідомих параметрів в оберненій задачі.

Ключові слова: машинне навчання, нейронні мережі, фізико-орієнтовані нейронні мережі, обернені задачі, диференціальні рівняння, динамічні системи, задача Ейлера про два нерухомі центри, ідентифікація параметрів.

Introduction

Relevance and motivation

Inverse problems arise in mathematics and physics when unknown causes, parameters, or properties of a system must be estimated from available data. Many physical systems are described by differential equations, including ordinary and partial differential equations. However, the numerical solution of such models can be challenging in the presence of nonlinearity, complex domains, noisy data, or other complicating factors [1].

Neural networks (NNs) and their physics-informed extension, Physics-Informed Neural Networks (PINNs) are promising machine learning methods for studying inverse problems. In particular, PINNs make it possible to incorporate the underlying physical laws directly into the training process. However, PINNs are still a relatively new approach, and their performance may depend on the problem formulation, available data, and training strategy. This work contributes to their further study by applying PINNs to Euler's planar two-fixed-centers problem.

Aim of the research

The **purpose** of the qualification work is to examine NNs and PINNs as approaches to solving forward and inverse problems for dynamical systems. The work highlights the features of each method, the advantages and limitations of PINNs, and study their application to trajectory reconstruction and parameter identification.

The **object** of the research is inverse problems in physics and physical systems described by differential equations, as well as neural-network-based methods used to solve them.

The **subject** of the research is trajectory reconstruction and identification of unknown physical parameters using PINNs. Special attention is paid to the performance of neural-network-based methods when only a limited amount of data is available. The practical part considers Euler's two-fixed-centers problem, where the inverse formulation is connected with the identification of unknown attraction parameters from observation data.

The **practical significance** of this research lies in the possibility to:

1. investigate the accuracy of physical parameter estimation;

2. study inverse problems with limited or noisy data;

3. analyze the role of physical constraints in neural network training.

1 Differential equations and inverse problems

Many physical, biological, chemical, and other natural processes can be described using ordinary differential equations (ODEs) or partial differential equations (PDEs). Although this work primarily focuses on neural-network-based methods, it is important to provide the necessary background on differential equations and inverse problems in order to understand the context in which these methods are applied. Since the practical part of this thesis is based on a dynamical system described by ODEs, the following discussion mainly uses ODE-based intuition and notation.

1.1 Differential equations in physical modeling

In general, a differential equation is an equation that involves an unknown function and its derivatives. In the case of ordinary differential equations, the unknown function depends on one independent variable, usually time. Such equations are commonly used to describe dynamical systems whose state evolves continuously. In many practical cases, it is difficult or impossible to obtain an analytical solution of such equations. Therefore, a wide range of analytical, numerical, and computational methods are used to approximate their solutions.

We consider a general ordinary differential equation in operator form:

$$\mathcal{F}\left(t, u(t), \frac{du}{dt}, \ldots, \frac{d^m u}{dt^m}; \gamma\right) = 0, \qquad t \in [t_0, T],$$

$$u(t_0) = u_0, \qquad \frac{du}{dt}(t_0) = u_1, \qquad \ldots$$

where t denotes time, $u(t)$ is the unknown state function, and γ represents physical parameters of the system. The operator $\mathcal{F}$ describes the governing differential relation and may include derivatives of $u(t)$ up to order m. The interval $[t_0, T]$ defines the time domain on which the solution is considered.

To determine a particular solution, the differential equation is supplemented with initial conditions. For higher-order ODEs, these conditions may include not only the value of the function at the initial time, but also the values of its derivatives. In the case of a dynamical system, this usually corresponds to specifying the initial state of the system.

1.2 Forward and inverse problems

For a differential equation model, the forward problem usually assumes that the governing equation, physical parameters, and initial conditions are known. The task is then to determine the solution of the system on the considered time interval. In the context of a dynamical system, this means predicting the evolution of the system state from the given model and initial data.

In contrast, inverse problems arise when some information about the system is unknown and has to be recovered from observations. These unknown quantities may include physical parameters, initial conditions, external forces, or other properties of the model. In many practical cases, the full solution is not available, and only partial or noisy measurements of the system are given.

When the structure of the differential equation is known, but some physical parameters are unknown, the inverse problem can be viewed as a parameter identification problem. In this case, the goal is not only to approximate the state function $u(t)$, but also to estimate the unknown parameters γ using the available observation data.

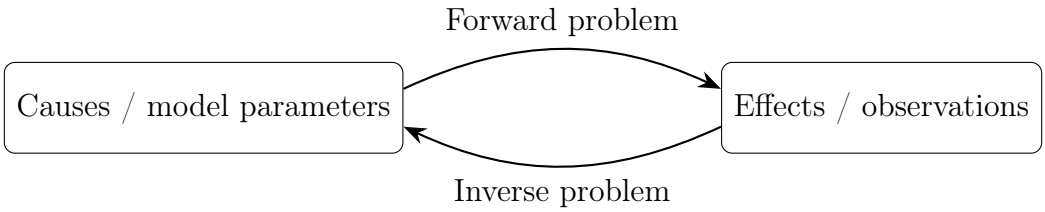

Figure 1: Simple conceptual relation between forward and inverse problems.

2 Machine learning approach

2.1 Artificial Neural Networks

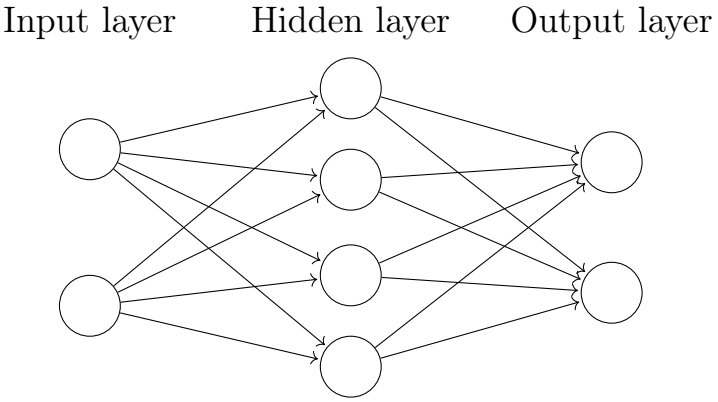

Figure 2: Schematic simplified structure of a feedforward neural network.

With the rapid development of machine learning methods, new opportunities for improving the solution of differential equations have emerged. Neural networks (NNs), inspired by biological neural structures [2], are widely used in many fields, ranging from computer vision to economics. According to the Universal Approximation Theorem, NNs are capable of approximating any continuous function to an arbitrary degree of accuracy [3].

A classical NN consists of interconnected neurons organized into layers (Fig. 2). Each neuron receives inputs, applies a linear transformation defined by weights and biases, and then passes the result through a nonlinear activation function. By combining multiple layers and optimizing a loss function during training, the NN learns from data and can represent a parametric function that maps an input vector to an output.

In this context, $u(t)$ represents an unknown function that we aim to approximate. If $\hat{u}_\theta(t)$ denotes the neural network approximation, where θ represents network parameters (weights and biases), the approximation can be written as:

$$\hat{u}_\theta(t) \approx u(t).$$

The parameters θ of the neural network are determined during training. This process is described in more detail in the next subsection.

2.2 Loss Functions and Optimization

Training a neural network can be formulated as an optimization problem. The network output depends on the trainable parameters θ, and the goal is to choose these parameters so that the loss function is minimized:

$$\theta^* = \arg\min_{\theta} \mathcal{L}(\theta).$$

The loss function $\mathcal{L}(\theta)$ defines how the quality of the approximation is measured. In supervised learning, it is usually constructed from the difference between the network predictions and the available target values. Let $\{(t_i, u_i)\}_{i=1}^{N}$ be a set of training data, where t_i is an input point and u_i is the corresponding observed value. The prediction error at t_i is

$$e_i = \hat{u}_\theta(t_i) - u_i.$$

Different loss functions measure this error in different ways and therefore affect how strongly small and large prediction errors are penalized during training.

Activation functions. Activation functions introduce nonlinearity into a neural network. Without them, a composition of several layers would still be equivalent to a linear mapping. Common activation functions include the sigmoid function,

$$\sigma(z) = \frac{1}{1 + e^{-z}},$$

the hyperbolic tangent,

$$\tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}},$$

and the rectified linear unit,

$$\mathrm{ReLU}(z) = \max(0, z).$$

In this work, the hyperbolic tangent (tanh) activation function is used in the hidden layers, which is a common choice for PINNs. This activation function is suitable because tanh is smooth and continuously differentiable, which is necessary when derivatives of the network output are used in the physics loss.

Mean squared error. One of the most common loss functions is the mean squared error (MSE), defined as

$$\mathcal{L}_{\mathrm{MSE}}(\theta) = \frac{1}{N} \sum_{i=1}^{N} \|\hat{u}_\theta(t_i) - u_i\|_2^2 .$$

The squared form makes this loss sensitive to large errors, since larger deviations contribute disproportionately more to the total loss. This property is useful when large prediction errors should be penalized strongly, but it can also make the training process more sensitive to noisy observations or outliers.

Mean absolute error. Another common choice is the mean absolute error (MAE), defined as

$$\mathcal{L}_{\mathrm{MAE}}(\theta) = \frac{1}{N} \sum_{i=1}^{N} \|\hat{u}_\theta(t_i) - u_i\|_1 .$$

In contrast to MSE, this loss increases linearly with the magnitude of the error. Therefore, MAE is less sensitive to large deviations and can be more robust when the data contains noise or outliers. However, because it does not penalize large errors as strongly as MSE, it may sometimes lead to slower or less precise convergence near the optimum.

Huber loss. The Huber loss combines the behaviour of MSE and MAE [6]. For a scalar error e, it is defined as

$$\ell_\delta(e) = \begin{cases} \frac{1}{2}e^2, & |e| \le \delta, \\ \delta\left(|e| - \frac{1}{2}\delta\right), & |e| > \delta, \end{cases}$$

where $\delta > 0$ is a threshold parameter. For small errors, the Huber loss behaves quadratically, similarly to MSE. For large errors, it grows linearly, similarly to MAE. This makes it useful in cases where the training data may contain noisy measurements or occasional large deviations.

Optimization and Adam optimizer. After the loss function is defined, training consists of minimizing it with respect to the network parameters. This is

usually done using gradient-based optimization methods, which update the parameters in the direction that reduces the loss. In general, the update step can be written as

$$\theta_{k+1} = \theta_k - \alpha \nabla_\theta \mathcal{L}(\theta_k),$$

where α is the learning rate and $\nabla_\theta \mathcal{L}(\theta_k)$ is the gradient of the loss function at iteration k.

In this work, the Adam optimizer is used. Adam is an adaptive gradient-based optimization method that uses estimates of the first and second moments of the gradients to adjust the update size for each parameter [7]. This makes it convenient for training neural networks, especially when the loss landscape is complex and different parameters may require different effective learning rates.

2.3 Physics-Informed Neural Networks

Although NNs are powerful universal function approximators, the classical training approach relies solely on observational data. In many scientific problems, however, the available data may be noisy or limited. At the same time, the underlying physical laws or relationships that describe the system are often known and can be expressed in the form of differential equations. Standard NNs do not explicitly enforce these physical constraints during training. As a result, the learned approximation may fit the data but still violate the governing equations.

The idea of using neural networks to solve differential equations is not entirely new. For example, Lagaris et al. [4] proposed a method in which a neural network approximates the solution and is trained to satisfy the differential equation.

Building on this general direction, the modern framework of Physics-Informed Neural Networks (PINNs) was introduced and popularized by Raissi et al. (2019). This modification of neural networks [5] has generated a lot of interest in the scientific community. Citations of the original work have grown rapidly, as has the number of papers related to PINNs.

PINN loss function. PINNs approximate the solution $u(t)$ using a neural network $\hat{u}_\theta(t)$ with parameters θ. Instead of relying purely on data, PINNs incorporate the differential equation ($\mathcal{L}_{phys}$) and initial/boundary conditions ($\mathcal{L}_{cond}$)

into the loss function, as well as data loss:

$$\mathcal{L}(\theta;\gamma) = \lambda_d \mathcal{L}_{data}(\theta) + \lambda_p \mathcal{L}_{phys}(\theta;\gamma) + \lambda_c \mathcal{L}_{cond}(\theta). \tag{1}$$

Here, γ denotes the physical parameters of the differential equation, and the coefficients λ_d, λ_p, and λ_c control the relative importance of the loss terms during training.

Physics residual and collocation points. The physics loss is the main difference between a standard neural network and a PINN. It is constructed by substituting the neural network approximation $\hat{u}_\theta(t)$ into the differential equation. Using the operator notation introduced earlier, the physics loss can be written as:

$$\mathcal{L}_{phys}(\theta;\gamma) = \frac{1}{N_p} \sum_{j=1}^{N_p} \left\| \mathcal{F}\left(t_j^p, \hat{u}_\theta(t_j^p), \frac{d\hat{u}_\theta}{dt}(t_j^p), \ldots, \frac{d^m \hat{u}_\theta}{dt^m}(t_j^p); \gamma \right) \right\|^2,$$

where t_j^p are collocation points at which the governing equation is evaluated, and N_p is the number of such points. These points do not necessarily correspond to observed data points. Instead, they are selected inside the time domain and are used to enforce the differential equation during training. The required derivatives of $\hat{u}_\theta(t)$ with respect to t are computed using automatic differentiation. Therefore, the network is trained not only to fit the observed data, but also to satisfy the governing equation at selected points in the time domain.

In inverse problems, some physical parameters of the differential equation may be unknown. In this case, these parameters can be treated as additional trainable variables and optimized together with the neural network parameters θ. Thus, a PINN can be used not only to approximate the solution $u(t)$, but also to identify unknown parameters of the physical system.

Limitations and related works. As concluded in the original work by Raissi et al., their discovery raised more questions than answers. Therefore, there is strong motivation to further explore this direction.

Cuomo et al. summarize PINNs as a framework with wide applicability for both forward and inverse problems [8]. However, PINNs depend on many factors and should not be considered a universal solution. For example, Krishnapriyan et

al. discuss possible failure modes of PINNs and show that difficulties can appear when the optimization problem becomes too complex [9]. One possible solution is curriculum regularization, where the complexity of training is increased gradually. This idea is related to the staged training strategy used later in this work.

Quality of observational data used in PINNs is also important. Satyadharma et al. [10] investigate PINN performance with sparse noisy velocity data for a lid-driven cavity flow problem. They show that data assimilation can reduce the computational cost of PINN simulations, but that the final accuracy depends strongly on the quality, amount, and distribution of the available data. In particular, noisy observations can reduce accuracy and make the model more dependent on the data term.

Another related direction is the use of transfer learning to improve PINN training. Mustajab et al. [11] study PINNs for high-frequency and multi-scale problems and note that such problems can lead to slow convergence and training instability. They propose to start from a simpler low-frequency problem and then gradually transfer the trained model to more complex cases. Although this approach is different from the staged loss-weight strategy used in this work, both ideas reflect the same general observation: PINN training often benefits from increasing the difficulty of the optimization problem gradually.

It is also worth mentioning that, in some cases, it may be useful to divide the whole problem domain into smaller subdomains. This idea is used in XPINNs, where the space-time domain is decomposed and different parts of the problem can be treated separately [12].

3 Problem Formulation and Methodology

3.1 Euler's Two-Fixed-Centers Problem

The practical part of this work is based on Euler's planar two-fixed-centers problem. This is a classical problem of mechanics in which a moving particle is influenced by two fixed attracting centers [13]. In this thesis, the planar version of the problem is considered.

Let the two fixed centers be located symmetrically on the x-axis:

$$C_1 = (-a, 0), \qquad C_2 = (a, 0),$$

where $a > 0$ determines half of the distance between the centers. The position of the moving particle is denoted by

$$q(t) = (x(t), y(t)).$$

The distances from the particle to the fixed centers are

$$r_1 = \sqrt{(x(t) + a)^2 + y(t)^2}, \qquad r_2 = \sqrt{(x(t) - a)^2 + y(t)^2}. \tag{2}$$

The attraction strengths of the two centers are denoted by μ_1 and μ_2. For attractive centers, these parameters are assumed to be positive. The corresponding potential is

$$V(x, y) = -\frac{\mu_1}{r_1} - \frac{\mu_2}{r_2}.$$

The force acting on the particle is determined by the negative gradient of the potential:

$$\mathbf{F}(x, y) = -\nabla V(x, y).$$

Assuming unit mass of the moving particle, Newton's second law gives

$$\ddot{q}(t) = \mathbf{F}(q(t)) = -\nabla V(q(t)).$$

Therefore, the equations of motion are

$$\ddot{x}(t) = -\mu_1 \frac{x(t) + a}{r_1^3} - \mu_2 \frac{x(t) - a}{r_2^3}, \tag{3}$$

$$\ddot{y}(t) = -\mu_1 \frac{y(t)}{r_1^3} - \mu_2 \frac{y(t)}{r_2^3}. \tag{4}$$

To determine a particular trajectory, the system is supplemented with initial conditions:

$$x(t_0) = x_0, \qquad y(t_0) = y_0,$$
$$\dot{x}(t_0) = v_{x,0}, \qquad \dot{y}(t_0) = v_{y,0}.$$

Thus, the system state is determined by the particle position, velocity, and physical parameters a, μ_1, and μ_2. In this work, this model is used as a benchmark dynamical system for studying forward and inverse PINN formulations.

Related work has also explored PINN-based approaches for the full three-body problem. Pereira et al. [14] show that physics-informed losses can improve physical consistency, although their results are mixed and the method is not universally better in all cases, especially near close encounters. This motivates the use of the Euler two-fixed-centers problem as a more controlled benchmark.

Also note that Euler two-fixed-centers problem is integrable and has exact solution in elliptic coordinates [15]. In this work, reference trajectories are generated numerically using `scipy.integrate` module. The obtained trajectory is then used as reference data for constructing observations in the forward and inverse experiments.

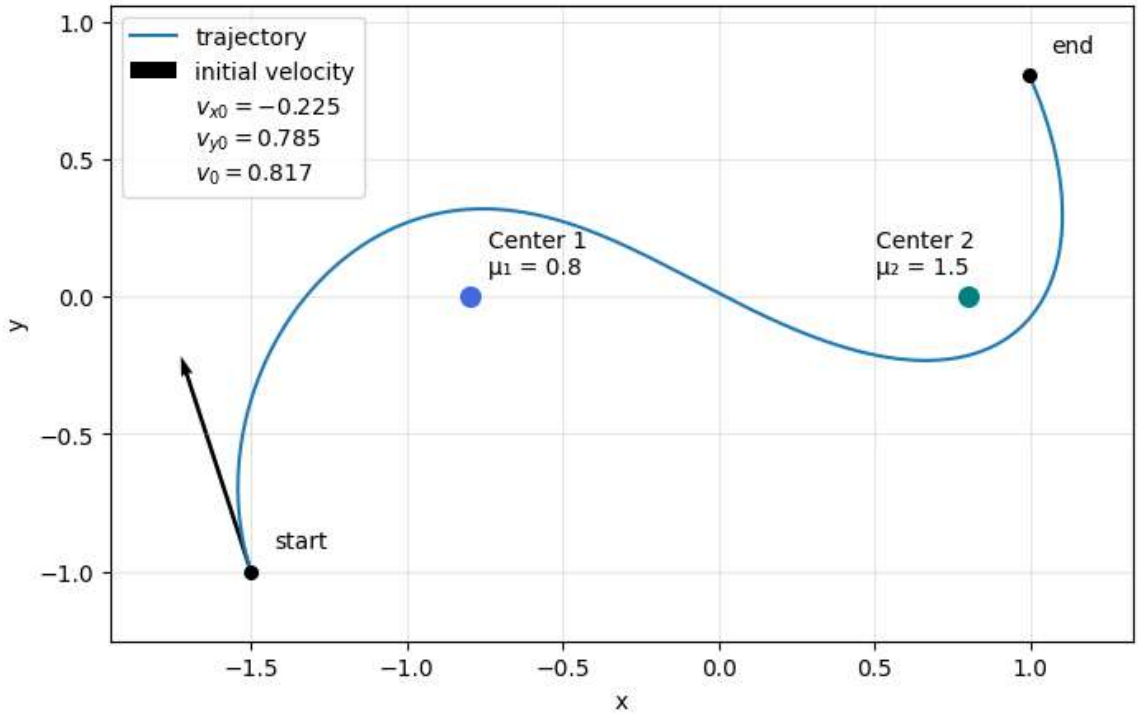

Figure 3: Example of the reference trajectory.

3.2 PINN Setup for the Euler Problem

For both forward and inverse experiments, the trajectory of the moving particle is approximated by a neural network. The input of the network is time, and the output is the predicted position of the particle:

$$t \mapsto (\hat{x}_\theta(t), \hat{y}_\theta(t)) \,,$$

where θ denotes the trainable parameters of the neural network.

The neural network does not directly output velocity or acceleration. Instead, automatic differentiation is used to compute derivatives of the predicted coordinates with respect to time. This means differentiating $\hat{x}_\theta(t)$ and $\hat{y}_\theta(t)$:

$$\dot{\hat{x}}_\theta(t), \quad \dot{\hat{y}}_\theta(t), \qquad \ddot{\hat{x}}_\theta(t), \quad \ddot{\hat{y}}_\theta(t).$$

In the physics residual, the second-order derivatives are used, while the first-order derivatives are needed when the initial velocity condition is included in the loss.

Substituting the neural network approximation into the equations of motion gives the physics residuals. Similarly to the distance definitions in Eq. (2), but using the neural network predictions, let

$$\hat{r}_1 = \sqrt{(\hat{x}_\theta(t) + a)^2 + \hat{y}_\theta(t)^2}, \qquad \hat{r}_2 = \sqrt{(\hat{x}_\theta(t) - a)^2 + \hat{y}_\theta(t)^2}.$$

Using the equations of motion (3)–(4), the residuals are defined as

$$R_x(t) = \ddot{\hat{x}}_\theta(t) + \mu_1 \frac{\hat{x}_\theta(t) + a}{\hat{r}_1^3} + \mu_2 \frac{\hat{x}_\theta(t) - a}{\hat{r}_2^3},$$

$$R_y(t) = \ddot{\hat{y}}_\theta(t) + \mu_1 \frac{\hat{y}_\theta(t)}{\hat{r}_1^3} + \mu_2 \frac{\hat{y}_\theta(t)}{\hat{r}_2^3}.$$

For the exact solution, both residuals should be equal to zero.

The physics loss is evaluated at collocation points t_j^p inside the chosen time interval:

$$\mathcal{L}_{phys} = \frac{1}{N_p} \sum_{j=1}^{N_p} \left(R_x(t_j^p)^2 + R_y(t_j^p)^2 \right).$$

Here, t_j^p are not observation data points, but additional points generated inside the chosen time interval to evaluate the physics residual. The number of collocation points N_p is specified before training. The collocation interval may

coincide with the interval where observation data are available, or it may be extended beyond it if the goal is to guide prediction outside the observed part of the trajectory.

The data loss measures the mismatch between the predicted and observed particle positions. If observations are given at time points t_i, the data loss is

$$\mathcal{L}_{data} = \frac{1}{N_d} \sum_{i=1}^{N_d} \left[(\hat{x}_\theta(t_i) - x_i)^2 + (\hat{y}_\theta(t_i) - y_i)^2 \right].$$

This expression corresponds to the mean squared error data loss. In experiments with noisy observations, alternative data losses such as MAE and Huber loss are also considered.

The initial-condition loss anchors the neural network solution at the beginning of the trajectory. It tells the model where the particle starts and, if velocity is included, how it initially moves. Depending on the experiment and available data, this term may play a more or less important role:

$$\mathcal{L}_{IC} = (\hat{x}_\theta(t_0) - x_0)^2 + (\hat{y}_\theta(t_0) - y_0)^2 + \left(\dot{\hat{x}}_\theta(t_0) - v_{x,0} \right)^2 + \left(\dot{\hat{y}}_\theta(t_0) - v_{y,0} \right)^2.$$

Following the general PINN loss in Eq. (1), the total loss for the Euler problem is written as

$$\mathcal{L} = \lambda_d \mathcal{L}_{data} + \lambda_p \mathcal{L}_{phys} + \lambda_{IC} \mathcal{L}_{IC},$$

where the coefficients λ_d, λ_p, and λ_{IC} control the relative influence of the data, physics, and initial-condition terms.

In practice, the coefficients in this loss are not fixed in the same way for the whole training process. Since the data loss and physics loss can have different scales and may compete with each other, a staged training strategy is used. At the beginning, the model is trained mainly to fit the observation data and the initial condition. After the trajectory approximation becomes more stable, the physics loss is introduced gradually by increasing its weight. If the physics term is given a large weight from the first epochs, the optimization process may lead to unrealistic trajectories and physically incorrect behaviour instead of improving the solution.

3.3 Implementation Pipeline

The numerical experiments in this work were implemented as a modular Python pipeline. The pipeline includes generation of reference trajectories, construction of observation data, preparation of tensors for PINN training, model training, and evaluation of the obtained results. This structure makes it possible to use the same physical model for both forward and inverse experiments while changing only the training setup, loss weights, and parameters treated as unknown.

Reference trajectory generation. The first step of the pipeline is to generate a reference trajectory for the Euler two-fixed-centers system. For this purpose, the second-order equations of motion are rewritten as a first-order system for the state vector

$$s(t) = (x(t), y(t), \dot{x}(t), \dot{y}(t)) .$$

The resulting initial value problem is solved numerically using the `solve_ivp` function from the SciPy library. The solver receives the physical parameters, the initial state, the time interval, and the number of evaluation points. The generated trajectory contains the time values, particle coordinates, and velocity components, and is used as the reference solution for constructing observations and evaluating the PINN results.

Observation and collocation data. Observation data are constructed by sampling the reference trajectory at selected time points. If the full numerical trajectory is given by

$$\mathcal{D}_{full} = \{(t_k, x_k, y_k) : k = 1, \dots, N\} ,$$

then sparse observations are obtained by taking every s-th point from the selected interval:

$$\mathcal{I}_{obs} = \{1, 1 + s, 1 + 2s, \dots\} \subset \{1, \dots, N\},$$

where s is the observation stride. The resulting sparse observation set is

$$\mathcal{D}_{obs} = \{(t_i, x_i, y_i) : i \in \mathcal{I}_{obs}\} .$$

Only the particle position is used as observation data, while velocity values are not directly given to the network as training targets.

To test the inverse problem under imperfect measurements, noisy observations are also considered. They are obtained by adding Gaussian noise independently to the observed coordinates:

$$\mathcal{D}_{noise} = \{(t_i, x_i + \varepsilon_{x,i}, y_i + \varepsilon_{y,i}) : i \in \mathcal{I}_{obs}\},$$

where

$$\varepsilon_{x,i}, \varepsilon_{y,i} \sim \mathcal{N}(0, \sigma^2).$$

The parameter σ denotes the standard deviation of the observation noise.

In addition to observation points, collocation points are generated inside the selected time interval. Before training, both observation data and collocation points are converted into tensors used by the neural network.

Neural network architecture. The same neural network architecture is used in all experiments. The model is a fully connected feedforward network with four hidden layers, each containing 64 neurons. The hyperbolic tangent activation function is used between hidden layers. This architecture provides a differentiable approximation of the trajectory, which is required for computing the physics residuals using automatic differentiation.

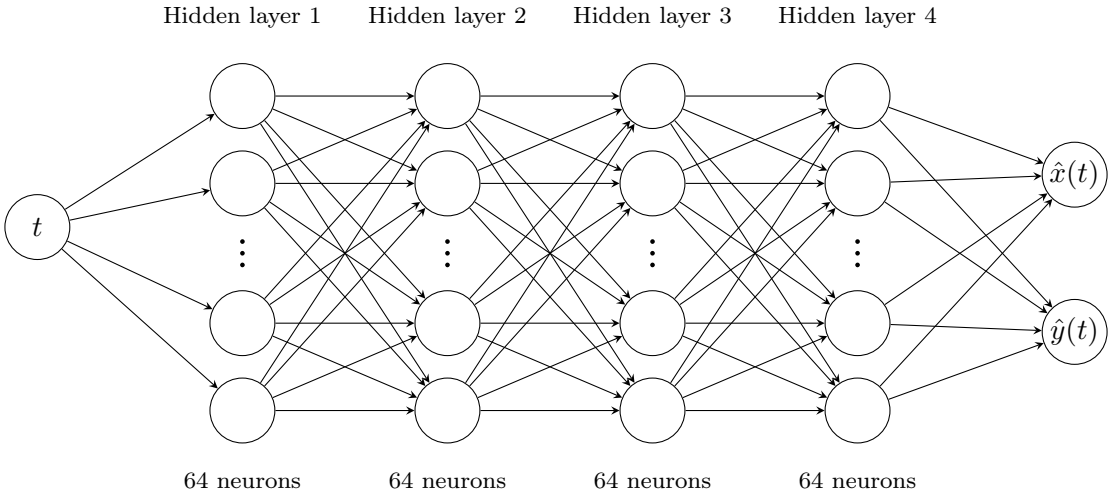

Figure 4: Schematic architecture of the neural network used in the experiments.

Forward and inverse setups. Two PINN formulations are considered in this work. In the forward setup, the physical parameters a, μ_1, and μ_2 are assumed to be known. Therefore, the neural network parameters θ are the only trainable

variables, and the goal is to reconstruct the particle trajectory while satisfying the equations of motion.

In the inverse setup, the parameter a remains fixed, while the attraction strengths μ_1 and μ_2 are unknown. They are optimized together with the neural network parameters. Since the centers are assumed to be attractive, the learned values of μ_1 and μ_2 must remain positive. To enforce this constraint, unconstrained trainable variables η_1 and η_2 are introduced, and the physical parameters are represented using the softplus transformation:

$$\mu_1 = \text{softplus}(\eta_1), \qquad \mu_2 = \text{softplus}(\eta_2),$$

where

$$\text{softplus}(z) = \log(1 + e^z).$$

Thus, the optimizer updates the unconstrained variables η_1 and η_2, while the physics residuals use the positive values μ_1 and μ_2.

Training procedure. All models are trained using the Adam optimizer. The objective is the weighted PINN loss, which combines data, physics, and initial-condition terms. Since these terms may have different scales, staged training is used: the model is first trained mainly on data and initial conditions, and the physics loss is introduced or increased in later stages.

In the forward setup, Adam updates only the neural network parameters:

$$\Theta_{\text{fwd}} = \{\theta\}.$$

In the inverse setup, the optimized parameter set also includes the unconstrained variables that define the unknown physical parameters:

$$\Theta_{\text{inv}} = \{\theta, \eta_1, \eta_2\}, \qquad \mu_i = \text{softplus}(\eta_i), \quad i = 1, 2.$$

Thus, the optimizer updates both the trajectory approximation and the unknown attraction strengths. In some inverse experiments, separate learning rates are used for θ and for η_1, η_2, which allows the neural-network weights and physical parameters to be updated at different speeds.

Evaluation metrics. The trained models are evaluated by comparing the predicted trajectory with the reference numerical trajectory. The main trajectory error is measured by the mean squared error

$$\mathrm{MSE}_{traj} = \frac{1}{N}\sum_{k=1}^{N}\left[(\hat{x}_\theta(t_k) - x_k)^2 + (\hat{y}_\theta(t_k) - y_k)^2\right].$$

For the forward problem, this error is also computed separately on the observed part of the trajectory and on the prediction interval, which makes it possible to distinguish interpolation quality from extrapolation behaviour.

For the inverse problem, the accuracy of the recovered physical parameters is evaluated using absolute and relative errors:

$$E^{abs}_{\mu_i} = |\hat{\mu}_i - \mu_i|, \qquad E^{rel}_{\mu_i} = \frac{|\hat{\mu}_i - \mu_i|}{|\mu_i|}, \qquad i = 1, 2.$$

In addition, the final values of the data, physics, and initial-condition losses are recorded to analyze how well the trained model fits the observations and satisfies the governing equations.

4 Numerical Experiments and Results

4.1 Forward Problem

The forward experiment considers the case where the physical parameters are known:

$$a = 5, \qquad \mu_1 = 2.0, \qquad \mu_2 = 1.5.$$

The initial state is

$$(x_0, y_0, v_{x,0}, v_{y,0}) = (6.0, 2.0, 0.5, -0.8).$$

Training data are taken only from the first half of the trajectory, $t \in [0, 15]$, giving 34 sparse observation points.

To evaluate the role of the physics-informed term, a plain neural network is compared with a forward PINN under the same architecture and observation setup. The plain neural network is trained using only the data loss, while the PINN also includes the physics and initial-condition losses.

The future-interval MSE during training is shown in Fig. 5. The plain neural network reduces the error only at the beginning, after which it remains large. In contrast, the PINN error decreases significantly during staged training, showing the benefit of including the equations of motion in the loss function.

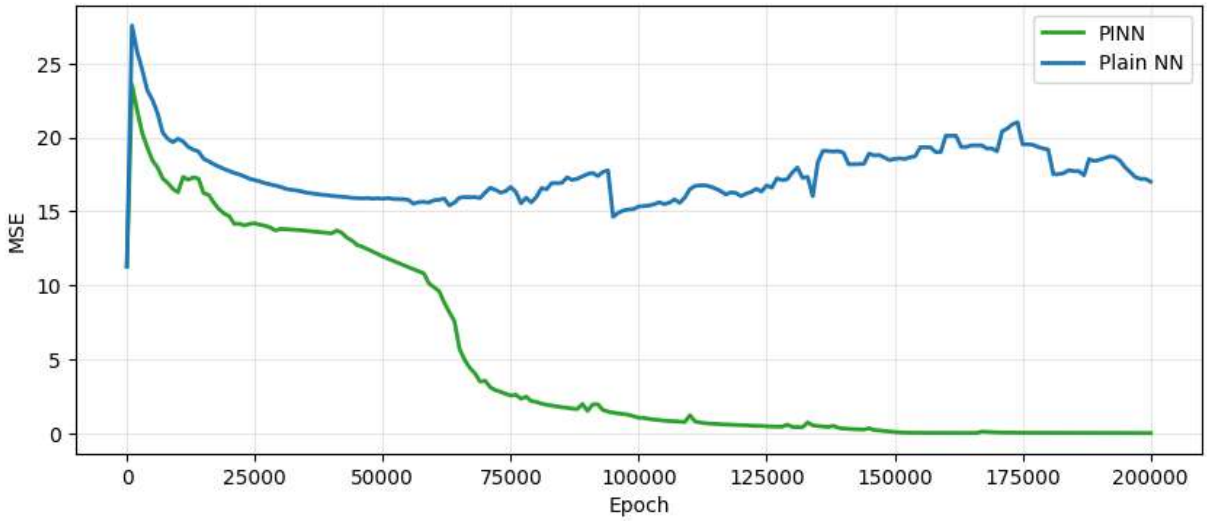

Figure 5: Future-interval MSE during training for the plain neural network and the forward PINN.

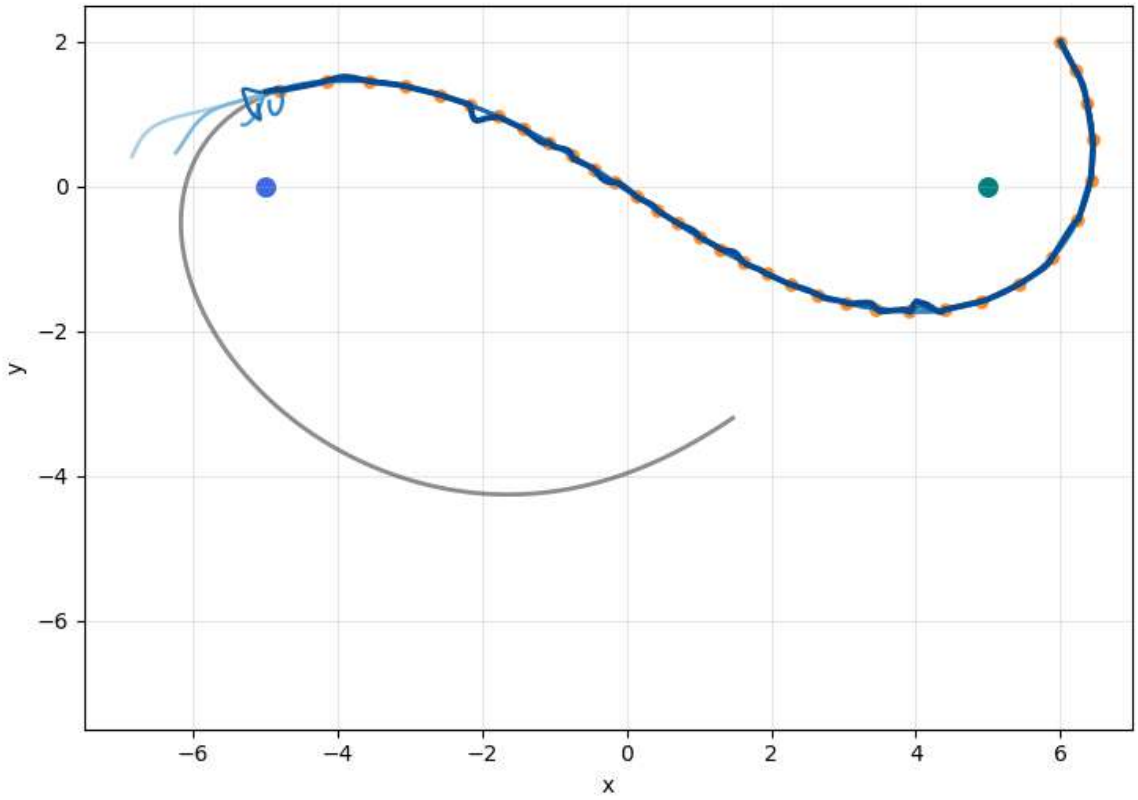

Figure 6: Forward prediction of the plain neural network during training.

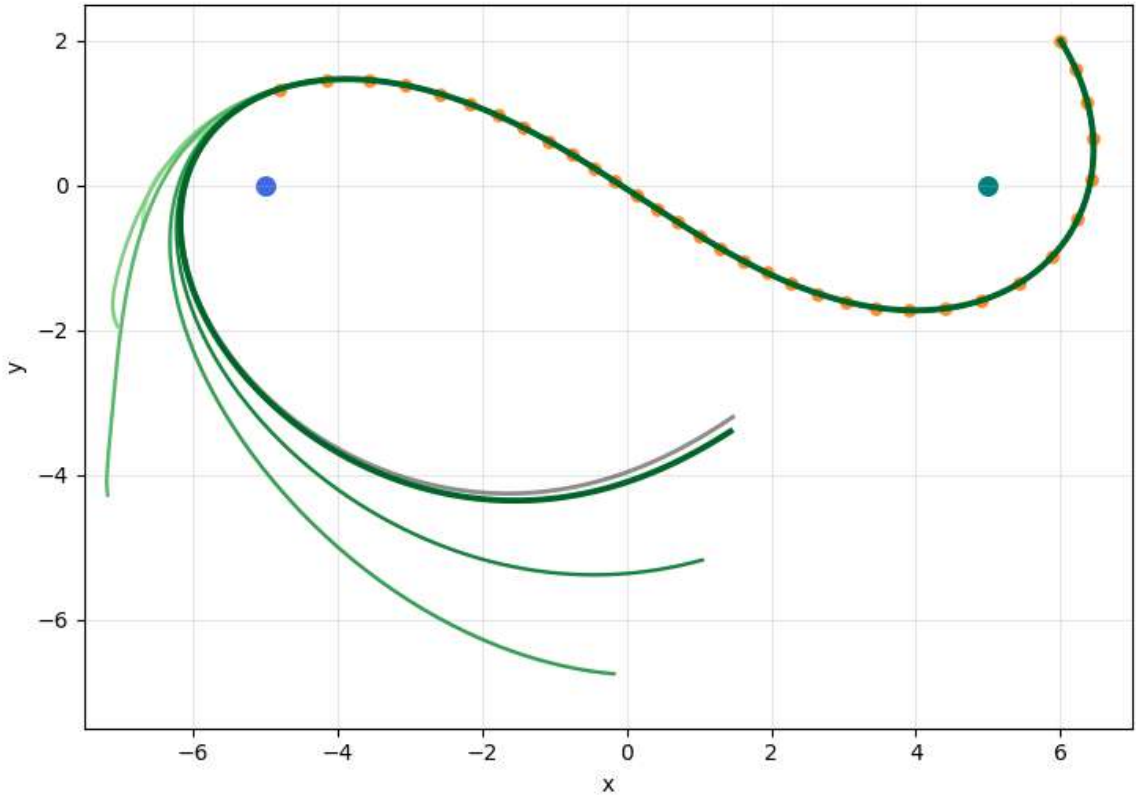

Figure 7: Forward prediction of the PINN during training.

The trajectory plots in Figures 6–7 illustrate the model predictions at selected training stages. They show how the plain neural network and the PINN converge differently toward the reference trajectory. The plain neural network mainly fits the observed part of the trajectory and does not recover the correct motion outside the training interval. In contrast, the PINN gradually approaches the reference trajectory also beyond the observed interval, where no observation points are provided.

The results in Table 1 shows that the plain neural network fits the observed interval well but performs poorly on the future interval. In contrast, the PINN greatly reduces both the full-trajectory and future-interval errors.

Model	MSE_{train}	MSE_{future}	MSE_{full}
Plain NN	2.779×10^{-3}	1.7005×10^{1}	8.5041
PINN	1.4×10^{-5}	5.581×10^{-3}	2.797×10^{-3}

Table 1: Forward problem results for the plain neural network and the PINN.

Staged learning was used for PINN training. The detailed training schedule is summarized in Table 2. For this experiment, the time variable is used in its original scale. The PINN initial-condition loss is computed using both the initial position and the initial velocity.

Model	Stage	Epochs	Learning rate	λ_d	λ_p	λ_{ic}^{pos}, λ_{ic}^{vel}
Plain NN	Data only	200000	10^{-3}	1.0	0.0	0.0, 0.0
PINN	1	10000	10^{-3}	1.0	0.0	10.0, 1.0
PINN	2	10000	10^{-3}	1.0	10^{-3}	10.0, 1.0
PINN	3	20000	10^{-3}	1.0	10^{-2}	10.0, 1.0
PINN	4	20000	10^{-3}	1.0	10^{-1}	10.0, 1.0
PINN	5	140000	10^{-3}	1.0	1.0	10.0, 1.0

Table 2: Staged training setup for the forward experiment.

4.2 Inverse Problem

In the inverse experiment, the parameter a is assumed to be known, while the attraction strengths μ_1 and μ_2 are treated as unknown and are learned together with the neural network parameters. The true values are

$$a = 1.0, \qquad \mu_1 = 2.0, \qquad \mu_2 = 0.6.$$

The initial state is

$$(x_0, y_0, v_{x,0}, v_{y,0}) = (0.1, 2.0, 0.5, 0.2),$$

and the experiment is performed on the time interval $t \in [0, 12]$. Sparse observations are used, with a total of 67 observation points.

In this and all subsequent inverse experiments, the time variable is normalized before being passed to the neural network, and the initial-condition loss is computed only for the initial position. The physical interval $t \in [t_0, T]$ is mapped to $\tau \in [-1, 1]$ by

$$\tau = \frac{2(t - t_0)}{T - t_0} - 1.$$

The inverse PINN is trained using a two-stage procedure. The first stage focuses on fitting the available data and the initial condition, while the second stage introduces the physics loss to refine both the trajectory and the unknown parameters.

Stage	Epochs	Learning rate	λ_d	λ_p	λ_{IC}
1	5000	10^{-3}	1.0	0.0	10.0
2	55000	10^{-4}	1.0	10^{-4}	10.0

Table 3: Staged training setup for the inverse experiment.

Figure 8 shows the learned trajectory together with sparse observations. The reconstructed trajectory follows the observed data and preserves the overall shape of the reference motion.

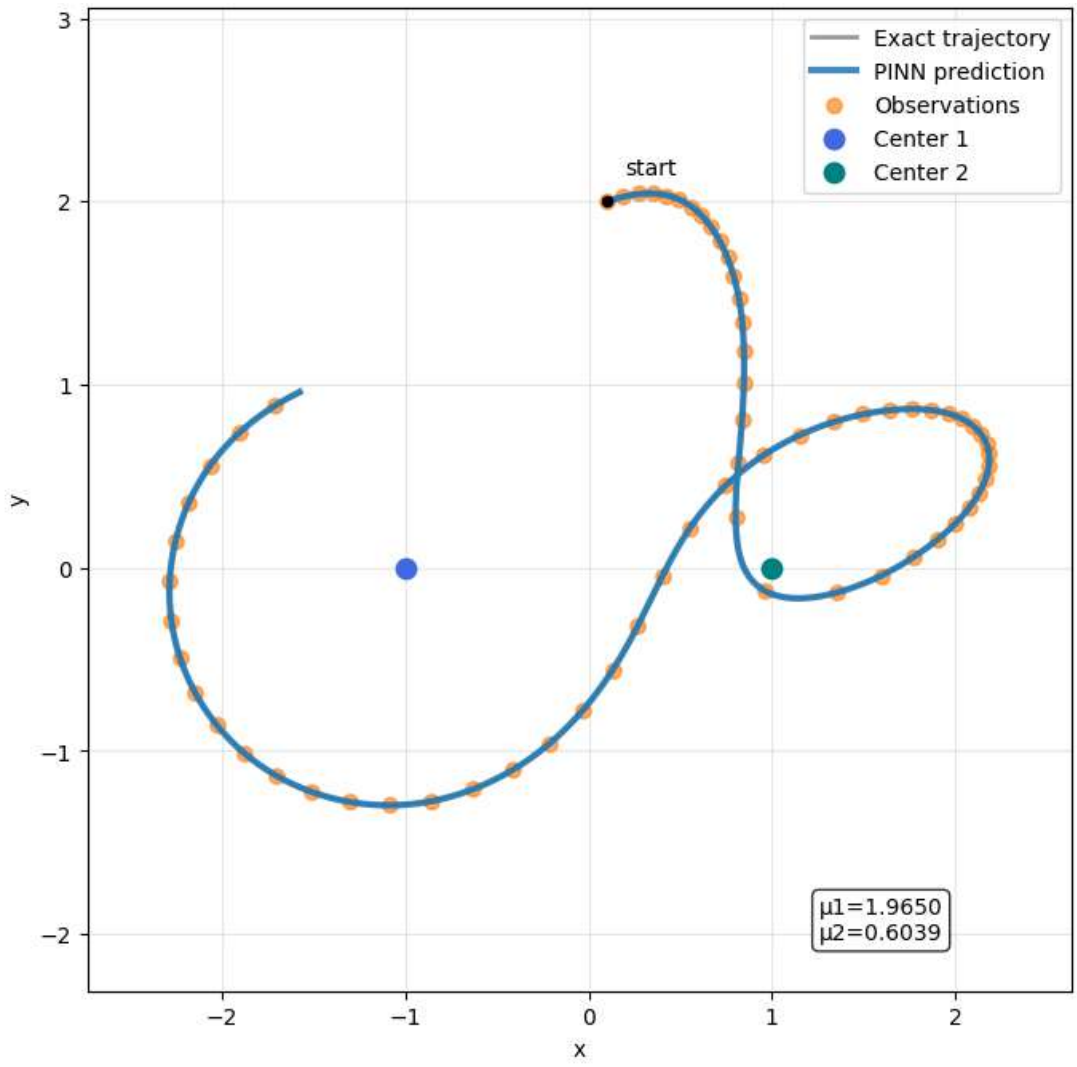

Figure 8: Learned trajectory of the inverse PINN with observations data.

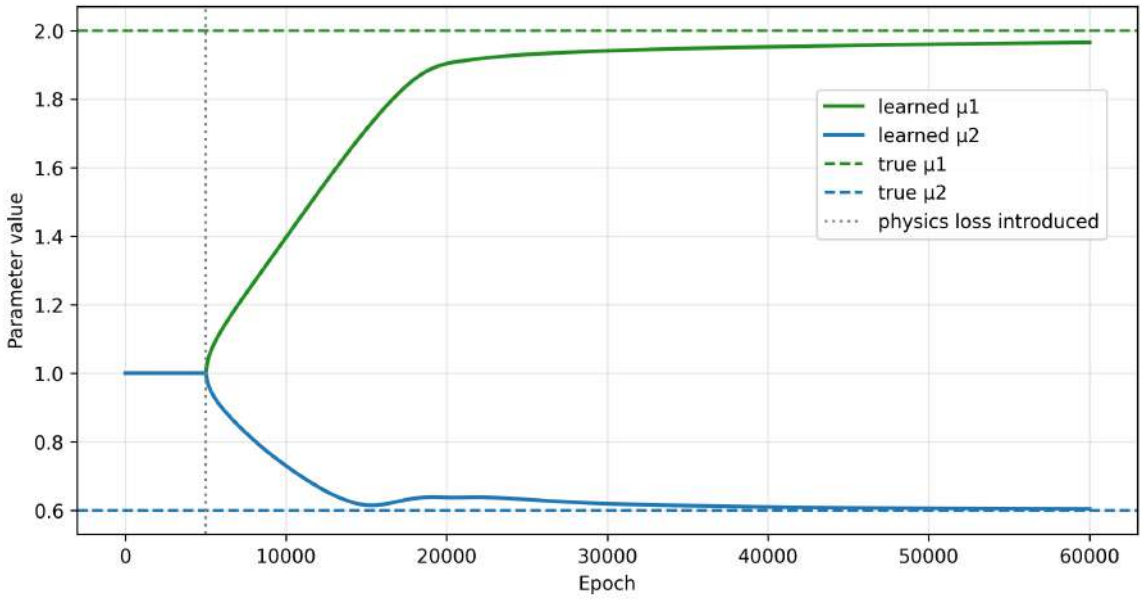

Figure 9: Recovered parameters μ_1 and μ_2 during inverse PINN training.

The evolution of the recovered parameters during training is shown in Fig. 9. After the physics loss is introduced, both parameters move toward their true values and stabilize near the correct solution.

Also, final recovered values are placed in Table 4. The learned parameters are close to the true ones, with relative errors below 2%.

Parameter	True value	Learned value	Absolute error	Relative error
μ_1	2.0	1.9650	0.0350	1.7478%
μ_2	0.6	0.6039	0.0039	0.6438%

Table 4: Recovered physical parameters in the inverse experiment.

Effect of time normalization. The same inverse setup was also tested without time normalization. The final errors remained relatively small in both cases, but the normalized-time version produced slightly better parameter recovery and a lower trajectory error.

Setup	Learned μ_1	Learned μ_2	Absolute error	Relative error
Without time norm.	1.9485	0.6025	0.0270	1.4964%
With time norm.	1.9650	0.6039	0.0194	1.1958%

Table 5: Comparison of PINN results with and without time normalization.

Generalization across configurations. To check whether the inverse PINN setup works beyond a single configuration, several additional runs were performed with different initial states and physical parameters. The same staged training strategy was used in all runs, with fewer optimization steps than in the main inverse experiment: first, a data-only stage with $\lambda_d = 1.0$, $\lambda_p = 0$, $\lambda_{IC} = 10.0$, learning rate 10^{-3}, and 5000 epochs; second, a physics-informed stage with $\lambda_d = 1.0$, $\lambda_p = 10^{-4}$, $\lambda_{IC} = 10.0$, learning rate 10^{-4}, and 35000 epochs. The input configurations are listed in Table 6, and the corresponding parameter recovery results are reported in Table 7.

The additional runs show that the learned parameters move toward the reference values in all considered configurations, while the final accuracy varies between runs.

Run	a	μ_1	μ_2	x_0	y_0	$v_{x,0}$	$v_{y,0}$	Time span	N_{obs}
1	3.0	3.5	2.6	-0.2	-2.0	0.1	1.0	$[0, 12]$	63
2	3.0	3.0	2.0	-3.0	-2.5	-1.0	1.0	$[0, 14]$	63
3	2.0	0.6	1.6	-0.5	1.0	0.5	0.5	$[0, 10]$	56
4	1.5	2.5	1.2	0.2	0.5	0.0	-1.5	$[0, 8]$	56
5	1.0	1.5	1.5	1.5	-1.0	-1.3	-0.2	$[0, 6]$	84
6	1.0	0.8	0.4	-0.5	0.3	0.3	-0.8	$[0, 4]$	84

Table 6: Input configurations for additional inverse PINN runs.

Run	True μ_1	True μ_2	Learned μ_1	Learned μ_2	Abs. error	Rel. error
1	3.5	2.6	3.5001	2.7152	0.0577	2.2178%
2	3.0	2.0	3.0098	1.9963	0.0068	0.2558%
3	0.6	1.6	0.5533	1.5984	0.0241	3.9385%
4	2.5	1.2	2.5107	1.2455	0.0281	2.1081%
5	1.5	1.5	1.5043	1.5039	0.0041	0.2726%
6	0.8	0.4	0.8008	0.4469	0.0238	5.9079%

Table 7: Additional inverse PINN runs for different parameter and trajectory configurations.

4.3 Inverse Problem with Noisy Observations

The inverse problem was also tested with sparse noisy observations. The same Gaussian noise model as defined earlier was used, and several values of the noise level σ were considered.

All noisy experiments used the same physical setup:

$$a = 0.8, \qquad \mu_1 = 0.8, \qquad \mu_2 = 1.5,$$

with initial state

$$(x_0, y_0, v_{x,0}, v_{y,0}) = (-1.5, -1.0, -0.225, 0.785),$$

on the time interval $t \in [0, 5]$. Only the noise level was changed between runs.

For this setting, a modified staged training strategy was used. First, the model was trained only on the data and initial-condition terms while the physical parameters were kept fixed. Then the physics loss was introduced, with a smaller learning rate for the trainable physical parameters. In the final stage, the parameter learning rate and physics weight were increased to improve parameter recovery.

Figure 10 shows the learned trajectory for the case $\sigma = 0.1$. Although the observations are visibly noisy, the PINN reconstructs a smooth trajectory instead of following the noise point by point. This reflects the regularizing effect of the physics-informed formulation.

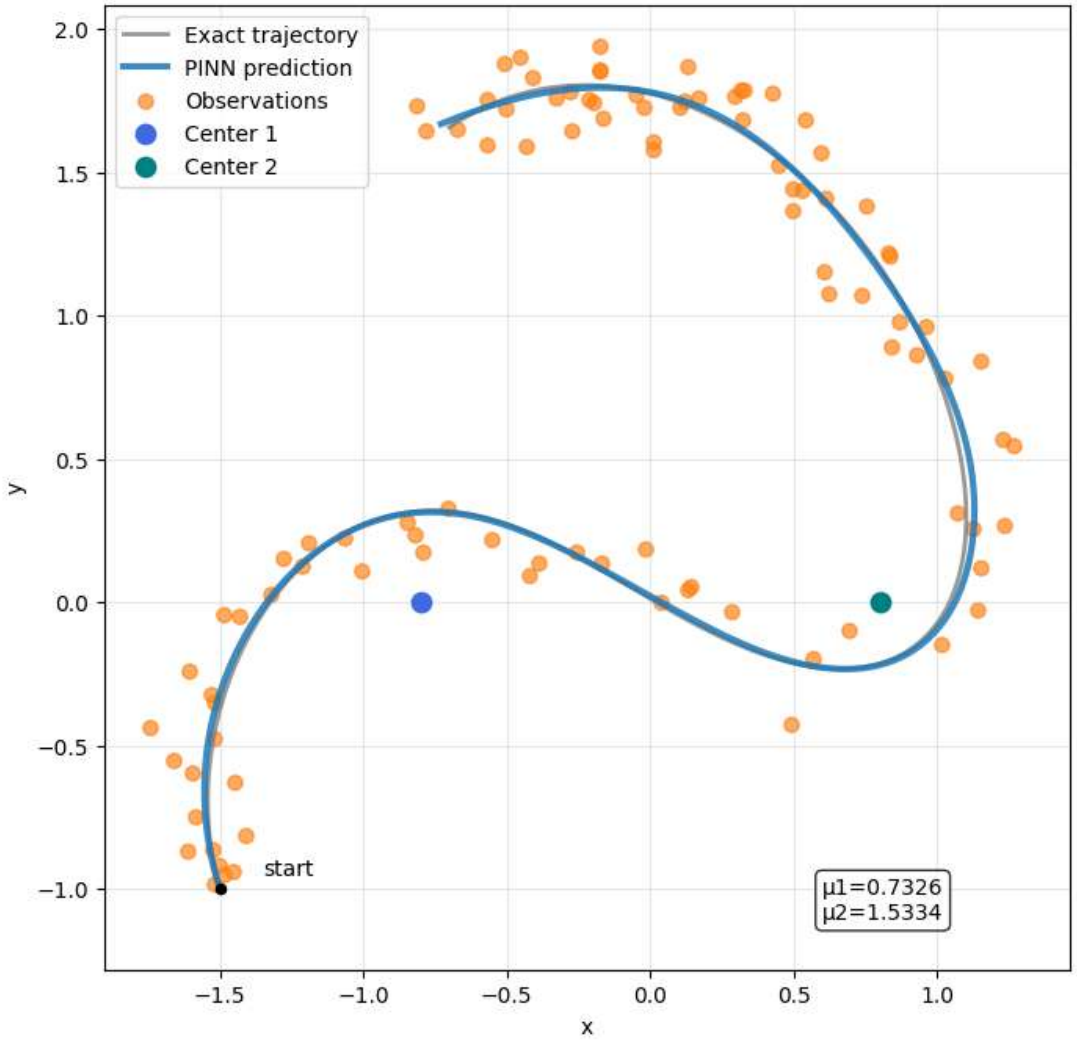

Figure 10: Inverse PINN prediction for noisy observations with $\sigma = 0.1$.

The results for different noise levels are summarized in Table 9. The same architecture, inverse formulation, and training strategy were used in all runs.

Stage	Epochs	NN lr	Param. lr	λ_d	λ_p	λ_{IC}
Data only	5000	10^{-3}	0	1.0	0	10.0
Slow parameter update	20000	10^{-4}	10^{-5}	1.0	10^{-4}	10.0
Parameter recovery	25000	10^{-4}	10^{-4}	0.5	5×10^{-4}	10.0

Table 8: Staged training strategy used for noisy inverse experiments.

Noise level σ	Learned μ_1	Learned μ_2	Mean abs. error	Mean rel. error
0.01	0.7848	1.5026	0.0089	1.0361%
0.03	0.7594	1.5152	0.0279	3.0433%
0.05	0.7326	1.5334	0.0504	5.3251%
0.1	0.6655	1.5594	0.0970	10.39%

Table 9: Inverse PINN results for sparse noisy observations.

4.4 Comparison of Data Loss Functions

The influence of the data loss term was also tested for the noisy inverse problem. Since MSE can be sensitive to large observation errors, the experiment was repeated with alternative data losses. The model architecture, physics loss, initial-condition loss, and training strategy were kept the same, while only the data mismatch term was changed.

The tested losses were mean squared error, mean absolute error, and Huber loss. The comparison was performed for the same noisy observation setting, with noise level $\sigma = 0.1$.

Data loss	Learned μ_1	Learned μ_2	Mean abs. error	Mean rel. error
MSE	0.6655	1.5594	0.0970	10.39%
MAE	0.3180	1.4861	0.2479	30.59%
Huber	0.7640	1.5171	0.0266	2.82%

Table 10: Comparison of data loss functions for noisy data with $\sigma = 0.1$.

In this experiment, the Huber loss gives the best parameter recovery. The weaker MAE result shows that robust losses are not automatically better in this setting.

4.5 Discussion of Results

Forward problem. In the forward problem, the main result is that the PINN provides a reliable continuation of the trajectory beyond the observed interval. The plain neural network approximates the observed part of the trajectory well, but it does not have information about the physical law that determines the particle motion. As a result, its prediction becomes inaccurate outside the interval where training points are available. In contrast, the PINN is constrained by the equations of motion and is therefore able to recover the general continuation of the trajectory on the unobserved interval. This shows that the physics loss improves the extrapolation ability of the model in the considered forward setting.

Inverse problem. In the inverse setting, the PINN is able to recover the unknown attraction parameters μ_1 and μ_2 with small relative errors in the main experiment. The parameter histories show that the introduction of the physics loss plays an important role in moving the learned parameters toward the reference values. As expected, the comparison with and without time normalization revealed that normalization can slightly improve parameter recovery and trajectory accuracy. Additional inverse runs show that the same training strategy works for several configurations, although the final accuracy depends on the chosen trajectory and physical parameters.

Noisy observations and data losses. For noisy observations, the results show that parameter recovery becomes more difficult as the noise level increases. Nevertheless, the PINN still reconstructs a decent trajectory and does not simply interpolate the noisy data points. This suggests that the physics-informed formulation has a regularizing effect. The comparison of data loss functions shows that the Huber loss gives the best recovery for the tested noisy case with $\sigma = 0.1$, while MAE performs worse than MSE in this experiment. Therefore, robust data losses can be useful, but their effectiveness depends on the optimization process and on their balance with the physics loss.

Conclusions

Main conclusions

In this thesis, a computational pipeline was developed for applying Physics-Informed Neural Networks for both forward and inverse formulations of Euler's planar two-fixed-centers problem. Several experimental settings were investigated, including sparse observations, noisy observations, time normalization, staged training, and different data loss functions. In the forward problem, the PINN was able to continue the trajectory beyond the observed interval more accurately than a plain neural network. The inclusion of the physics loss and initial-condition loss helped the model converge toward a physically consistent prediction. In the inverse formulation, the PINN recovered the unknown attraction parameters μ_1 and μ_2 with good accuracy in the main experiment and produced reasonable results for different configurations, including noisy data. The experiments also showed that staged training is an important part of the method, since it helps the network first approximate the trajectory and then refine the solution using the physics loss. Among the tested data losses, the Huber loss gave the best parameter recovery in the considered noisy case, which suggests that it can be useful when the observations contain larger noise deviations.

Future work

Future work may include the following directions:

- improving the training strategy, neural network architecture, and robustness of the method in the presence of noisy observations;
- comparing the obtained results with other approaches, including optimization-based methods, other machine learning models, and classical numerical methods for inverse problems;
- increasing the complexity of the benchmark problem, for example by considering three-dimensional motion, a larger number of attracting centers, or systems with moving bodies.

AI disclosure. In accordance with the methodological guidelines "Writing and Defence of the Qualification Thesis" of the Department of Mathematics, Faculty of Informatics, National University of Kyiv-Mohyla Academy, the use of AI-assisted tools in this thesis is disclosed below.

During the preparation of this thesis, AI-assisted tools were used in a limited and controlled way. OpenAI GPT-5.5 Thinking model was used for language editing, improving clarity and flow of thesis sections, checking consistency of terminology, assisting with LaTeX formatting, and refining LaTeX plots and figure-related code. The same AI model was used for code-related tasks, including debugging, implementation refinement, and improving the structure of selected code fragments.

All AI-assisted outputs were reviewed, edited, and critically evaluated by the author. The experimental design, implementation decisions, numerical experiments, interpretation of the results, and final conclusions were carried out by the author. AI tools were not used to fabricate data, modify experimental results, or replace the author's analysis.

References

[1] Quarteroni A., Valli A. *Numerical Approximation of Partial Differential Equations.* // Springer. — 1994.

[2] Rosenblatt F. *The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain* // Psychological Review. — 1958. — Vol. 65. — pp. 386–408.

[3] K. Hornik, M. Stinchcombe, and H. White, *Multilayer feedforward networks are universal approximators* // Neural Networks, vol. 2, no. 5, pp. 359–366, 1989. doi: 10.1016/0893-6080(89)90020-8.

[4] Lagaris I. E., Likas A., Fotiadis D. I. *Artificial neural networks for solving ordinary and partial differential equations* // IEEE Transactions on Neural Networks. — 1998.

[5] M. Raissi, P. Perdikaris, and G. E. Karniadakis, *Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations* // Journal of Computational Physics, vol. 378, pp. 686–707, 2019.

[6] P. J. Huber, *Robust Estimation of a Location Parameter* // The Annals of Mathematical Statistics, vol. 35, no. 1, pp. 73–101, 1964. DOI: 10.1214/aoms/1177703732.

[7] D. P. Kingma and J. Ba, *Adam: A Method for Stochastic Optimization* // International Conference on Learning Representations (ICLR), 2015. arXiv:1412.6980.

[8] S. Cuomo, V. Schiano Di Cola, F. Giampaolo, G. Rozza, M. Raissi, and F. Piccialli, *Scientific Machine Learning Through Physics-Informed Neural Networks: Where we are and What's Next* // arXiv:2201.05624, 2022.

[9] A. S. Krishnapriyan, A. Gholami, S. Zhe, R. M. Kirby, and M. W. Mahoney, *Characterizing possible failure modes in physics-informed neural networks* // arXiv:2109.01050, 2021.

[10] A. Satyadharma, M.-H. Chern, H.-C. Kan, Marinaldi, and J. Julian, *Assessing Physics Informed Neural Network Performance with Sparse Noisy Velocity Data* // Physics of Fluids, 2024. DOI: 10.1063/5.0213522.

[11] A. H. Mustajab, H. Lyu, Z. Rizvi, and F. Wuttke, *Physics-Informed Neural Networks for High-Frequency and Multi-Scale Problems Using Transfer Learning* // Applied Sciences, vol. 14, no. 8, 3204, 2024. DOI: 10.3390/app14083204.

[12] A. D. Jagtap and G. E. Karniadakis, *Extended Physics-Informed Neural Networks (XPINNs): A Generalized Space-Time Domain Decomposition Based Deep Learning Framework for Nonlinear Partial Differential Equations* // Communications in Computational Physics, vol. 28, no. 5, pp. 2002–2041, 2020.

[13] N. Martynchuk, H. R. Dullin, K. Efstathiou, and H. Waalkens, *Scattering invariants in Euler's two-center problem* // arXiv preprint arXiv:1801.09613, 2018.

[14] M. Santos Pereira, L. Tripa, N. Lima, F. Caldas, and C. Soares, *Advancing Solutions for the Three-Body Problem Through Physics-Informed Neural Networks* // 75th International Astronautical Congress (IAC), Milan, Italy, 2024. arXiv:2503.04585.

[15] F. Biscani and D. Izzo, *A complete and explicit solution to the three-dimensional problem of two fixed centres* // Monthly Notices of the Royal Astronomical Society, vol. 455, no. 4, pp. 3480–3493, 2015.