Solving the Double Pendulum Problem with Physics-Informed Neural Networks (PINNs)

Zanko Ventsislavov Mezdrichki

November 4, 2025

1 Introduction

1.1 The Double Pendulum

The double pendulum is a classic system in physics and dynamics. It consists of a pendulum attached to the end of another pendulum. The behavior of the physical system, obtainable by solving the associated Euler-Lagrange equations, generates two second-order ordinary differential equations that are complex and lengthy to solve. Moreover, it is a classic example of a chaotic system, which, by slightly changing the initial conditions, evolves in a completely different way over time.

1.2 Project Objective

The primary objective of this project is to implement and train a **Physics-Informed Neural Network (PINN)** to solve the equations of motion for a double pendulum.

To achieve this goal, the following steps are performed:

1. **Obtain the real solution:** Use standard Lagrangian mechanics and numerical integration (via SciPy) to create an accurate, conventional solution.
2. **Train the PINN:** Build a neural network that learns the system's dynamics. This network will be trained with a small set of data from the scipy simulation and by enforcing the physical laws as part of its training loss.

The project aims to demonstrate that a PINN can accurately reproduce the complex, chaotic motion of the double pendulum, offering a modern machine-learning-based alternative to traditional differential equation solvers.

2 Methodology

The project is structured into three main components: the physics simulation (`physics.py`), the neural network model (`AI_model.py`), and the main training script (`main.py`). The parameters to use are provided directly as input through the terminal (`config.py`).

2.1 Scipy Simulation (`physics.py`)

This script serves as the benchmark for the AI model. It calculates the "correct" motion of the pendulum using established physics principles.

1. **Lagrangian Mechanics:** The dynamics of the system are derived by obtaining the Lagrangian of the two pendulums:

$$T = T1 + T2 \quad (Kinetic \quad energy)$$
$$V = V1 + V2 \quad (Potential \quad energy)$$
$$L = T - V \quad (Lagrangian)$$

2. **Symbolic Derivation:** The Python library **SymPy** is used to define all variables ($m_1, m_2, L_1, L_2, \theta_1(t), \theta_2(t)$) as mathematical symbols. It symbolically calculates the kinetic and potential energies and, from them, the Lagrangian.

3. **Equations of Motion:** The **Euler-Lagrange equations** are applied for each angle θ_i:

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{\theta}_i}\right) - \frac{\partial L}{\partial \theta_i} = 0$$

   SymPy computes these derivatives and simplifies the expressions, resulting in two complex, coupled second-order ordinary differential equations.

4. **Solution:** The `smp.solve` function is used to algebraically isolate the second-derivative terms ($\ddot{\theta}_1$ and $\ddot{\theta}_2$). The system is then converted into a set of four first-order ODEs using `smp.lambdify`. Through the `odeint` function, by providing the initial conditions, it is possible to obtain the dynamics of the system. The solution is discrete; in fact, as a argument, a list containing different times is also necessary, which are the points in time where the solution is then found.

2.2 PINN Architecture (`AI_model.py`)

The AI model is a feed-forward neural network defined using PyTorch.

- **Input:** A single scalar value, time (t).
- **Output:** A 4-element vector representing the state of the system at time t: $[\theta_1(t), \dot{\theta}_1(t), \theta_2(t), \dot{\theta}_2(t)]$.
- **Structure:** The network consists of:
  1. An input layer (1 neuron).
  2. Four hidden layers of 256 neurons each, using the **Tanh** activation function.
  3. An output layer (4 neurons).
- **Initialization: Xavier normal initialization** is used for the weights. This helps maintain a stable variance of activations and gradients throughout the network, preventing issues like vanishing or exploding gradients and leading to more effective training.

2.3 Model Training and Loss Function (`main.py`)

This script is the core of the project, where the PINN is trained.

The total loss is a weighted sum of three distinct components:

$$L_{total} = \lambda_{ic} L_{ic} + \lambda_{data} L_{data} + \lambda_{phys} L_{phys}$$

1. **Initial Condition Loss (L_{ic}):**
   - **Purpose:** To force the network to obey the specified initial conditions at $t = 0$.

- **How:** The model's output at $t = 0$ is compared to the known initial state vector `y0_tensor`(provided by `config.py`). The Mean Squared Error (MSE) between the prediction and the true value is calculated.

2. **Data Loss (L_{data}):**
   - **Purpose:** To "anchor" the network's solution to known-correct values at various points in time.
   - **How:**A small set of data is selected from the solution using odeint(1/20 of the total), which is then compared with the data predicted by the PINN using MSE.
3. **Physics Loss (L_{phys}):**
   - **Purpose:** To ensure the network's output obeys the laws of physics at all times.
   - **How:** This loss is calculated over a large set of "collocation points". It has two parts:
     - **a) Consistency Loss:** Enforces internal consistency between the network's predictions. Specifically, it ensures that the predicted angular velocity $\dot{\theta}_i$ matches the time derivative of the predicted angle θ_i, computed via `torch.autograd.grad`.
     - **b) Equation Residual Loss:** Encourages the network to respect the underlying physical laws. The network's outputs (and their time derivatives computed via `torch.autograd.grad`) are substituted into the Euler–Lagrange equations, and the Mean Squared Error of the resulting residual is minimized.

3 Parameter Justification

The choice of parameters, particularly the loss weights, is critical to the model's success.

- **Physical & Simulation Parameters (`config.py`):** This script defines all the physical and training parameters required by the simulation. The parameters are obtained through the `obtain_config()` function, which interactively asks the user to input the values. If the user provides invalid or empty inputs, default values are automatically assigned.
  - **Physical Parameters:** Obtained from user input or set to default values if not provided.
    - `L1, L2`: Lengths of the first and second pendulums (default: 1.0 m).
    - `m1, m2`: Masses of the two pendulums (default: 1.0 kg).
    - `g`: Gravitational acceleration (default: 9.81 m/s^2).
  - **Initial Conditions:** The user is asked to specify the initial angles and angular velocities of both pendulums:

    $$\theta_{1,0},\ \theta_{2,0},\ \dot{\theta}_{1,0},\ \dot{\theta}_{2,0}$$

    If no input is given, default values are used: $\theta_{1,0} = \theta_{2,0} = \pi/2$ rad and $\dot{\theta}_{1,0} = \dot{\theta}_{2,0} = 0$ rad/s.
  - **PINN Training Settings:** Additional simulation and network parameters are also set through user input:
    - `t_max`: Total simulation time (default: 10.0 s).
    - `n_points`: Number of time points used for training (default: 1001).

* `epochs`: Number of training epochs (default: 10,000). For even more accurate analysis, increase the number; however, the training process will be longer.
* `lr`: Learning rate for the optimizer (default: 10^{-3}).

These values are stored in a Python dictionary and passed to the main scripts for both the SciPy simulation and the PINN training.

- **Training & Network Parameters (`main.py`):**
  - **Optimizer (`Adam`):** Adam is an adaptive optimizer that automatically adjusts the learning rate of each model parameter during training.
  - **Scheduler (`ReduceLROnPlateau`):** If the total loss stops improving for a set number of epochs (`patience=75`), the learning rate is automatically reduced(`factor=0.4`).
  - **Loss Weights (λ values):** These are chosen empirically to balance the three loss components.
    * `lambda_ic = 800.0` **(High):** Respecting the initial condition is essential. In a chaotic system, an infinitesimal change at $t = 0$ leads to a completely different dynamic. This high weight forces the model to be correct at the start.
    * `lambda_data = 70.0` **(Medium-High):** Data points obey the physical laws, so they must be given considerable attention by the PINN.
    * `lambda_phys = 1.0` **(Low):** A lower value prevents the physics term from dominating too early, allowing the network to first fit the observed data and maintain stable training. By keeping it lower, problems of instability and divergences are avoided.

4 Results and Conclusion

The `main.py` script automatically evaluates the trained model and produces key outputs:

1. **Console Output (MAE):** The Mean Absolute Error for both θ_1 and θ_2 is printed, providing a measure of accuracy.
2. **Loss History (`training_history.png`):** This plot shows the total loss and its three components over the training epochs.
3. **Comparison Plot (`pinn_comparison.png`):** This plots the PINN's predicted $\theta(t)$ (dashed line) directly on top of the angles calculated by SciPy.
4. **Animation (`pendulum_simulations.mp4`):** A side-by-side animation visually compares the SciPy-simulated pendulum with the PINN-simulated pendulum.

Conclusion: The project aims to show how, by using correct parameters for training the PINN, it is possible to have a fairly realistic simulation of the double pendulum, although some deviations appear at longer time scales (due to chaos and numerical instability).

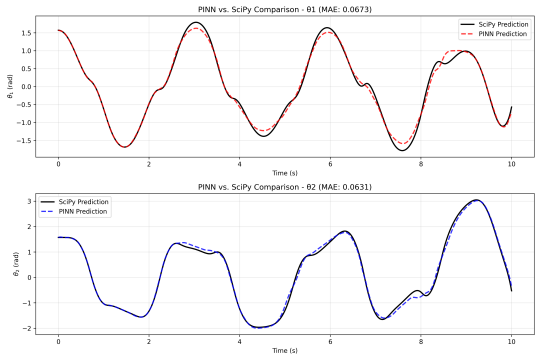
The parameters provided in the previous paragraphs are not optimal, but they are the best combination of values I was able to find for the default simulation(See Figure 1).

However, by changing the duration of the simulation, it is still necessary to modify other parameters as well, such as increasing the epochs, the number of time and data points , but above

all increasing the `lambda_phys`, because the PINN tries to cheat by minimizing the loss only of the second angle, which is less chaotic(see Figure 2).

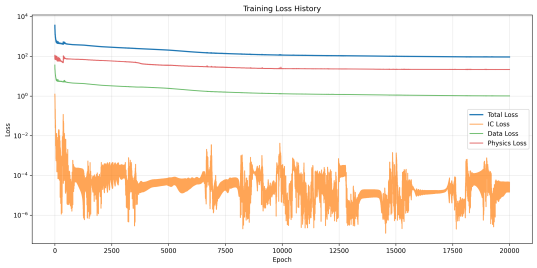
The introduction of methods related to machine learning could be useful in cases where there is limited experimental data and the underlying physics is known. Another advantage is that the solution from the PINN is continuous, so it is possible to obtain the configuration of the two pendulums at any time, unlike `odeint`, which provides the solution only at certain selected points (thus, in this case, the solution is discrete).

(a) Training history of of the PINN with the default data. A general decrease in all the losses as the epochs progress can be noticed. By increasing the epochs, even more accurate predictions are obtained, since the training process is not yet completed.

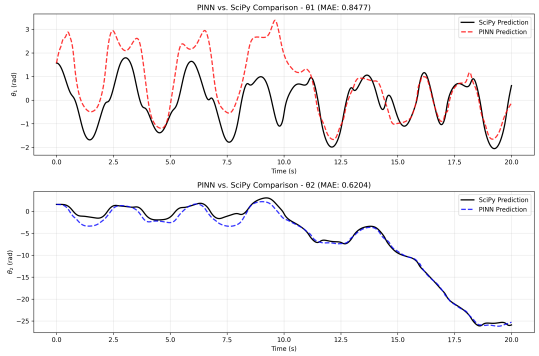


(b) Angles calculated with SciPy and angles predicted by the PINN with the default data. For the first few seconds of the animation, the values are almost equal, but as time passes, the deviations become more evident.

Figure 1: Results obtained with the default values.



(a) Training history of of the PINN with the default physical data and modified values for the PINN. It can be observed that training towards the end of the epochs does not further improve the knowledge of the PINN.



(b) It is evident that the PINN did not fully understand the physics of the system, trying to minimize the loss of θ_2 but not enforcing the physics for the first one. It is essential to raise `lambda_phys` in order to prevent the PINN from cheating.

Figure 2: Results obtained with the modified values(t_{max}=20s, n_{points}=2001, epochs=20000).

5 Future improvements

This is my first approach to the world of ML and PINNs.

In the future, I would like to improve the project structure by adding a PyQt5 interface for configuration.

To improve the effectiveness of the PINN, I consider it appropriate to implement adaptive loss weighting for better training balance (At this moment they have been found empirically). I intend to learn the use and functioning of `GradNorm`.

Alternatively, I would also like to test other architectures such as SIREN or Fourier-feature networks, however, I do not yet have the necessary knowledge to do this, so it will be a future implementation.