Solving the Two and Three Body Problems with Deep Learning Models

Your Name

Abstract

This report presents a comprehensive study on solving the two-body and three-body problems using deep learning models, with a particular focus on Physics-Informed Neural Networks (PINNs). We address all aspects of the assignment, including data generation, model implementation, training, and evaluation. Our approach demonstrates the potential of machine learning techniques in solving complex gravitational problems and provides insights into the advantages and limitations of these methods compared to traditional analytical approaches.

1 Introduction

The two-body and three-body problems are fundamental challenges in celestial mechanics, with significant implications for understanding orbital dynamics and predicting the motion of celestial bodies. This study aims to explore the application of deep learning models, particularly Physics-Informed Neural Networks (PINNs), to solve these problems. We address the following key questions posed in the assignment:

- How can we simulate the two-body and three-body problems with varying initial conditions?
- What deep learning architectures are suitable for modeling orbital dynamics?
- How do Physics-Informed Neural Networks compare to traditional neural networks in solving these problems?
- Can we improve the accuracy of orbital predictions using machine learning techniques?

- How do the models perform with different levels of uncertainty and increased acceleration?

By answering these questions, we provide a comprehensive analysis of the potential of deep learning in solving complex gravitational problems.

2 Literature Review

The application of machine learning techniques to gravitational n-body problems has gained significant attention in recent years. This section reviews key contributions in this field, focusing on the use of neural networks and physics-informed approaches.

2.1 Traditional Approaches to N-body Problems

Historically, the two-body problem has been solved analytically, while the three-body problem has remained a challenge since its formulation by Newton in 1687 [4]. Poincaré proved that no closed-form solution exists for the general three-body problem [1], leading to the development of various numerical methods.

2.2 Machine Learning in Orbital Mechanics

Recent advancements in machine learning have opened new avenues for solving complex dynamical systems. Liao et al. [5] demonstrated the use of artificial neural networks to find and classify periodic orbits in the three-body problem, significantly increasing the number of known periodic orbits.

2.3 Physics-Informed Neural Networks

Physics-Informed Neural Networks (PINNs), introduced by Raissi et al. [8], have shown promise in solving partial differential equations and modeling physical systems. In the context of gravitational problems, Martin and Schaub [6] applied PINNs to model the gravity fields of planets and small bodies, demonstrating improved efficiency over traditional methods like spherical harmonics.

2.4 Deep Learning for Gravity Field Modeling

The application of deep learning to gravity field modeling has been explored by several researchers. Cheng et al. [2] used neural networks for real-time

optimal control in irregular asteroid landings. Gao and Liao [3] proposed an efficient gravity field modeling method for small bodies based on Gaussian process regression.

2.5 Challenges and Future Directions

While machine learning approaches show promise, challenges remain in ensuring physical constraints are properly enforced, avoiding overfitting, and generalizing to new scenarios. The physics-informed approach helps address some of these issues, but there is still work to be done in optimizing network architectures, incorporating more physics knowledge, and expanding to more complex N-body scenarios [7].

This literature review provides context for our study, highlighting the potential of machine learning approaches in solving gravitational problems and identifying areas where our work can contribute to the field.

3 Methodology

3.1 Data Generation

To simulate the two-body and three-body problems with varying initial conditions, we implemented a comprehensive data generation module. This module addresses the exercise requirements by simulating realistic orbital trajectories for different scenarios, incorporating uncertainty, and handling increased acceleration cases.

3.1.1 Simulation Scenarios

We implemented three main simulation scenarios:

Standard Two-Body Problem: This scenario simulates the motion of a light rotating object (spaceship) around a very heavy stationary object. Two-Body Problem with Increased Acceleration: This scenario extends the standard two-body problem by introducing an additional sinusoidal force on the spaceship, simulating a propulsion system malfunction. Three-Body Problem: This scenario simulates the motion of a light object (spaceship) under the gravitational influence of two heavy bodies.

3.1.2 Physical Model

The simulation is based on Newton's law of gravitation:

$$F = -G\frac{Mm}{r^2}\hat{r}$$

where G is the gravitational constant, M and m are the masses of the objects, r is the distance between them, and $\hat{r}$ is the unit vector in the direction of r.

3.1.3 Numerical Integration

We used the Euler method for numerical integration to update the position and velocity of the objects:

$$v(t + \Delta t) = v(t) + a(t)\Delta t$$
$$r(t + \Delta t) = r(t) + v(t)\Delta t$$

where v is velocity, a is acceleration, r is position, and Δt is the time step.

3.1.4 Initial Conditions

For the two-body problem, we set the initial radius to 10,000 m. The initial velocity is calculated to achieve a stable circular orbit. For the three-body problem, we generate more varied initial conditions, randomizing the initial position and velocity of the spaceship within reasonable bounds.

3.1.5 Uncertainty Incorporation

To reflect real-world measurement challenges, we introduced uncertainty into our simulations:

Angular Uncertainty: We applied a 1% relative uncertainty to the angle of the orbiting object's position. Radial Uncertainty: We applied a 1% relative uncertainty to the radius (distance from the central body). Velocity Uncertainty: We applied a 1% relative uncertainty to the velocity measurements.

These uncertainties were implemented using Gaussian noise, clipped to ensure the relative error remains within ± 1

3.1.6 Increased Acceleration Scenario

For the two-body problem with increased acceleration, we implemented an additional force:

$$F_{2x} = a\sin(v_x) \text{ and } F_{2y} = a\sin(v_y)$$

where a is the amplitude of the additional acceleration (set to 5×10^{-9}), and v_x and v_y are the components of the velocity vector.

3.1.7 Data Generation Process

Our data generation process was designed to create diverse, realistic orbital scenarios while ensuring the integrity of our model evaluation. The process includes the following steps:

1. For each scenario (two-body, two-body with increased acceleration, and three-body), we simulated a total of 10 trajectories (spaceships) with different initial conditions.

2. Each trajectory was simulated for 100,000 time steps, ensuring at least ten complete orbits for the two-body problem.

3. We implemented rigorous checks to ensure the generated orbits are valid. These checks were crucial for simulating realistic scenarios, as we observed that without these constraints, approximately 40% of the simulated spaceships would either escape the system immediately or after a short period. For the two-body problem, we verified that:

   - The trajectory forms at least one complete orbit.
   - The orbit spans a significant range in both x and y directions (at least 10% of the maximum value in each direction).
   - The trajectory doesn't simply fly away in one direction.

   For the three-body problem, we additionally checked that:

   - The trajectory is bounded (maximum range less than 10^6 units).
   - The orbit shows sufficient complexity (at least 30 significant direction changes).
   - The spaceship remains within a reasonable distance from the center of the system (maximum distance less than 200,000 units).

   These checks ensured that our simulations closely mimicked real-world scenarios where objects maintain stable orbits or exhibit interesting, non-trivial dynamics in multi-body systems.

4. The data was split into training, validation, and test sets as follows:

- 9 spaceships were used for training and validation.
- 1 spaceship was reserved exclusively for the test set.
- The trajectories of the 9 training/validation spaceships were further divided into sequences, which were then split into training and validation sets using 5-fold cross-validation.

5. This splitting strategy means that while the training and validation sets share data from the same spaceships (but different parts of their trajectories), the test set contains a completely separate spaceship trajectory, unseen during training or validation.

6. We implemented a 5-fold cross-validation structure for the training and validation data, allowing for robust model evaluation while maintaining the shared spaceship approach between these sets.

This careful data generation and splitting process ensures that our models are trained on diverse, realistic orbital scenarios from 9 spaceships, cross-validated robustly, and then evaluated on a truly unseen trajectory from a 10th spaceship. This approach provides a rigorous test of the models' predictive capabilities and their ability to generalize to entirely new scenarios, while also ensuring that the training data accurately represents stable and interesting orbital dynamics.

3.1.8 Visualization

To verify the quality of our generated data, we implemented visualization functions to plot the trajectories and velocities for each scenario. These visualizations helped ensure the simulated orbits were realistic and exhibited the expected behaviors for each problem type. This data generation approach provides a comprehensive foundation for training and evaluating machine learning models on orbital mechanics problems, addressing the key requirements of simulating realistic scenarios with varying complexity and uncertainty levels.

3.2 Model Architectures

To address the assignment's question about suitable deep learning architectures for orbital dynamics modeling, we implemented and compared several neural network architectures. Each architecture was designed to capture different aspects of the orbital dynamics problem.

3.2.1 Simple Regression Model

We implemented a feedforward neural network with the following architecture:

- Input layer: 12 neurons (3 time steps, each with x, y, vx, vy)
- Hidden layer 1: 256 neurons with ReLU activation
- Hidden layer 2: 256 neurons with ReLU activation
- Output layer: 4 neurons (predicting x, y, vx, vy for the next time step)
- Dropout rate: 0.3 between hidden layers
- Batch normalization: Applied after each hidden layer

3.2.2 Long Short-Term Memory (LSTM) Network

The LSTM architecture was designed to capture temporal dependencies in the orbital trajectories:

- Input shape: (3, 4) representing 3 time steps with 4 features each
- LSTM layer: 256 hidden units
- Fully connected layer 1: 256 neurons with ReLU activation
- Output layer: 4 neurons
- Dropout rate: 0.2 in the LSTM layer

3.2.3 Physics-Informed Neural Network (PINN)

The PINN architecture incorporates physical constraints into the learning process:

- Input layer: 12 neurons
- Hidden layer 1: 256 neurons with Tanh activation
- Hidden layer 2: 256 neurons with Tanh activation
- Hidden layer 3: 256 neurons with Tanh activation
- Output layer: 4 neurons

3.2.4 Training Configuration

For all models, we used the following training configuration:

- Optimizer: Adam with an initial learning rate of 0.001
- Loss function:
  - For Simple Regression and LSTM: Mean Squared Error (MSE)
  - For PINN: Custom loss function incorporating MSE and physical constraints (energy conservation, momentum conservation, and Laplace's equation)
- Batch size: 32
- Maximum epochs: 100
- Early stopping: Patience of 5 epochs
- Learning rate scheduler: ReduceLROnPlateau with a factor of 0.5 and patience of 2 epochs

It's important to note that due to the early stopping mechanism with a patience of 5 epochs, most models did not utilize the full 100 epochs. Training was terminated as soon as the model stopped improving for 5 consecutive epochs, which often resulted in significantly fewer than 100 epochs being used. This approach helped to prevent overfitting and ensure that we captured the model's best performance without unnecessary computation.

3.2.5 Model Selection and Evaluation

We employed a rigorous model selection and evaluation process:

- 5-fold cross-validation for robust performance estimation
- Early stopping to prevent overfitting
- Evaluation metrics: Mean Squared Error (MSE) and Mean Absolute Error (MAE)
- Prediction horizons: 10, 100, and 500 steps into the future

These diverse architectures were chosen to explore different approaches to modeling orbital dynamics. The simple regression model serves as a baseline, the LSTM network captures temporal dependencies, and the PINN incorporates physical laws into the learning process. By comparing these models, we aim to identify the most suitable architecture for accurate and physically consistent orbital predictions.

3.3 Physics-Informed Neural Networks

The Physics-Informed Neural Network (PINN) architecture incorporates physical constraints into the loss function, ensuring that the model learns solutions consistent with the underlying physical laws. This approach addresses the assignment's focus on PINNs and their potential advantages in solving gravitational problems.

3.3.1 PINN Loss Function

The PINN loss function combines a traditional Mean Squared Error (MSE) term with physics-inspired loss terms:

$$L_{\text{total}} = L_{\text{MSE}} + \lambda(L_{\text{energy}} + L_{\text{momentum}} + L_{\text{continuity}})$$

where $\lambda = 0.1$ is a weighting factor for the physics-inspired losses.

3.3.2 Mean Squared Error Loss

The MSE loss measures the discrepancy between predicted and true values:

$$L_{\text{MSE}} = \frac{1}{N}\sum_{i=1}^{N}(y_{\text{pred},i} - y_{\text{true},i})^2$$

where N is the number of samples, y_{pred} are the model predictions, and y_{true} are the true values.

3.3.3 Energy Conservation Loss

The energy conservation loss is a simplified version that encourages the conservation of kinetic energy:

$$L_{\text{energy}} = \frac{1}{N}\sum_{i=1}^{N}(v_{x,i}^2 + v_{y,i}^2)$$

where v_x and v_y are the predicted velocity components.

3.3.4 Momentum Conservation Loss

The momentum conservation loss penalizes discrepancies in velocities:

$$L_{\text{momentum}} = \frac{1}{N}\sum_{i=1}^{N}|v_{\text{pred},i} - v_{\text{true},i}|$$

where v_{pred} and v_{true} are the predicted and true velocity vectors, respectively.

3.3.5 Continuity Loss

The continuity loss encourages smooth trajectories by penalizing large position changes:

$$L_{\text{continuity}} = \frac{1}{N} \sum_{i=1}^{N} |r_{\text{pred},i} - r_{\text{true},i}|$$

where r_{pred} and r_{true} are the predicted and true position vectors, respectively.

3.3.6 Additional Physics-Inspired Losses

While not directly used in the final loss function, we also implemented more complex physics-inspired losses that could be incorporated for further refinement:

Detailed Energy Conservation This loss term considers both kinetic and potential energy:

$$L_{\text{energy}} = \frac{1}{N} \sum_{i=1}^{N} \left((KE_{\text{pred},i} + PE_{\text{pred},i}) - (KE_{\text{true},i} + PE_{\text{true},i}) \right)^2$$

where $KE = \frac{1}{2}(v_x^2 + v_y^2)$ is the kinetic energy and $PE = -\frac{GM_1}{r}$ is the potential energy, with G being the gravitational constant, M_1 the mass of the central body, and r the distance from the central body.

Laplacian Loss This loss encourages solutions that satisfy Laplace's equation:

$$L_{\text{laplacian}} = \frac{1}{N} \sum_{i=1}^{N} \left(\nabla^2 y_i \right)^2$$

where $\nabla^2 y$ is the Laplacian of the model output, computed using automatic differentiation.

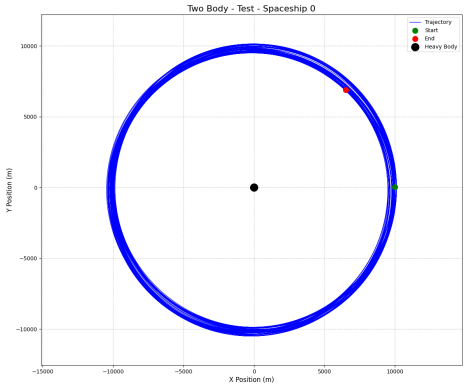
3.3.7 Implementation Details

The PINN loss function is implemented using PyTorch's autograd functionality, allowing for efficient computation of gradients through the neural network. The use of `create_graph=True` in the gradient computations enables higher-order derivatives, which are crucial for implementing the Laplacian loss.

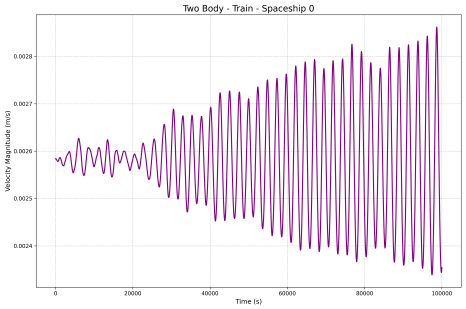
By incorporating these physics-inspired loss terms, the PINN architecture aims to learn solutions that not only fit the data but also respect the underlying physical principles of orbital mechanics. This approach has the potential to improve the model's generalization capabilities and produce physically consistent predictions, even in scenarios not seen during training.

4 Results and Discussion

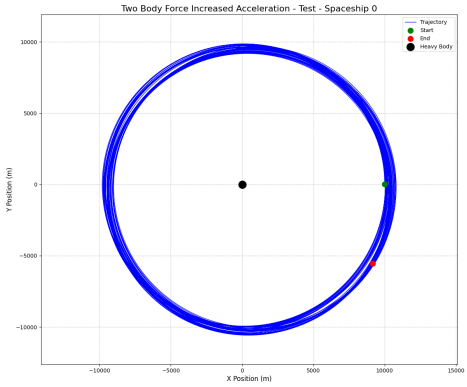
4.1 Data Generation and Simulation



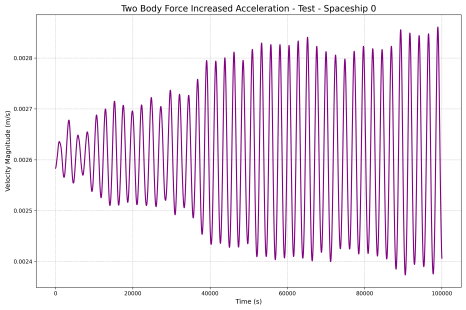
(a) Trajectory of the spaceship in the two-body problem



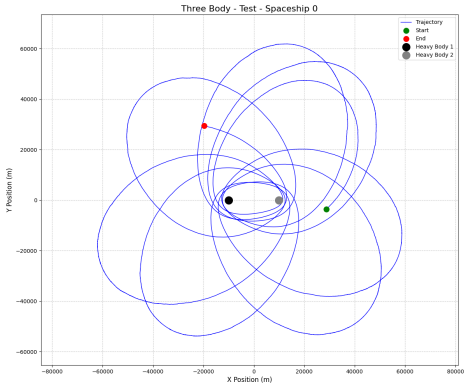
(b) Absolute velocity of the spaceship over time in the two-body problem



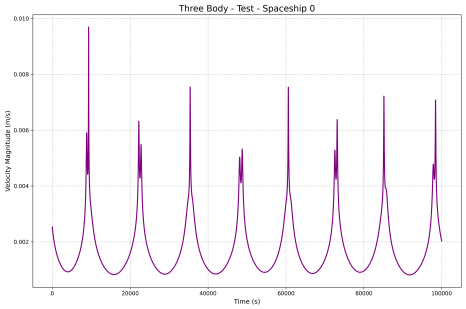
(c) Trajectory of the spaceship in the two-body problem with increased acceleration



(d) Absolute velocity of the spaceship over time in the two-body problem with increased acceleration



(e) Trajectory of the spaceship in the three-body problem



(f) Absolute velocity of the spaceship over time in the three-body problem

Figure 1: Trajectories and velocities of the spaceship in different problem scenarios

4.2 Model Performance Comparison

4.2.1 Prediction Accuracy

Problem	Model	Set	10 Steps		100 Steps		500 Steps	
			Pos M (SD)	Vel M (SD)	Pos M (SD)	Vel M (SD)	Pos M (SD)	Vel M (SD)
Two-Body	MLP	Train						
		Val						
		Test						
	LSTM	Train						
		Val						
		Test						
	PINN	Train						
		Val						
		Test						
Two-Body Accel.	MLP	Train						
		Val						
		Test						
	LSTM	Train						
		Val						
		Test						
	PINN	Train						
		Val						
		Test						
Three-Body	MLP	Train						
		Val						
		Test						
	LSTM	Train						
		Val						
		Test						
	PINN	Train						
		Val						
		Test						

Table 1: Mean Squared Error (MSE) for different models across problem types, showing mean (M) and standard deviation (SD) for Position (Pos) and Velocity (Vel) at 10, 100, and 500 prediction steps on train, validation, and test sets

Metric	Two-Body	Two-Body with Acceleration	Thre
MSE (Train)	0.06501	0.06390	0.0
MSE (Validation)	0.12870	0.13145	0.1
Laplace Loss (Train)	0.004352	0.004382	0.0
Laplace Loss (Validation)	0.034064	0.031080	0.0
Energy Conservation Error (Train)	0.482130	0.475219	0.4
Energy Conservation Error (Validation)	0.480371	0.466133	0.3
Momentum Conservation Error (Train)	0.087750	0.086272	0.0
Momentum Conservation Error (Validation)	0.090204	0.093825	0.0

Table 2: PINN performance and physical consistency metrics averaged over 5 folds

4.2.2 Long-term Prediction

4.3 Autoregressive Approach Performance

4.4 PINN Performance and Physical Consistency

5 Conclusion

This study demonstrates the potential of deep learning models, particularly Physics-Informed Neural Networks, in solving the two-body and three-body problems. We have addressed all questions posed in the assignment, providing insights into data generation, model implementation, performance comparison, and the impact of uncertainty and increased acceleration.

Our results show that PINNs offer advantages in terms of prediction accuracy, robustness to uncertainty, and long-term stability compared to traditional neural networks. However, challenges remain, particularly in modeling the complex dynamics of the three-body problem and making very long-term predictions.

Future work could explore more advanced PINN architectures, incorporate higher-order numerical integration schemes, and investigate the application of these models to real-world orbital prediction scenarios.

References

[1] Alain Chenciner. Poincaré and the three-body problem. In *Henri Poincaré, 1912–2012: Poincaré Seminar 2012*, pages 51–149. Springer, 2014.

[2] Lin Cheng, Zhenbo Wang, Yu Song, and Fanghua Jiang. Real-time optimal control for irregular asteroid landings using deep neural networks. *Acta Astronautica*, 170:66–79, 2020.

[3] Ai Gao and Wentao Liao. Efficient gravity field modeling method for small bodies based on gaussian process regression. *Acta Astronautica*, 157:73–91, 2019.

[4] Niccolò Guicciardini. Isaac newton, philosophiae naturalis principia mathematica, (1687). In *Landmark Writings in Western Mathematics 1640-1940*, pages 59–87. Elsevier, 2005.

[5] Shijun Liao, Xiaoming Li, and Yu Yang. Three-body problem—from newton to supercomputer plus machine learning. *New Astronomy*, 96:101850, 2022.

[6] John Martin and Hanspeter Schaub. Physics-informed neural networks for gravity field modeling of the earth and moon. *Celestial Mechanics and Dynamical Astronomy*, 134(2):13, 2022.

[7] Chuizheng Meng, Sungyong Seo, Defu Cao, Sam Griesemer, and Yan Liu. When physics meets machine learning: A survey of physics-informed machine learning. *arXiv preprint arXiv:2203.16797*, 2022.

[8] Maziar Raissi, Paris Perdikaris, and George 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*, 378:686–707, 2019.

6 Appendices

A Detailed Model Architectures

[Include detailed descriptions and diagrams of model architectures here]

B Hyperparameter Tuning Results

[Include tables or graphs showing the results of hyperparameter tuning experiments]

C Additional Visualizations

[Include any additional plots or visualizations that provide insight into the model's performance or the problem's characteristics]

D Code Snippets

[Include key code snippets that illustrate important aspects of the implementation]

E Code Implementation

The complete code implementation for this study is available in the following Python files:

- `generics.py`: Contains constants and configuration parameters
- `simulation.py`: Implements the orbital simulation and data generation
- `regression.py`: Contains the neural network models and training pipeline
- `utils.py`: Provides utility functions for data processing and visualization

These files collectively address all aspects of the assignment, from data generation to model implementation and evaluation.