# Solving the Two and Three Body Problems with Deep Learning Models

Your Name

**Abstract**

This study addresses the two-body and three-body gravitational problems using deep learning models, with a focus on Physics-Informed Neural Networks (PINNs). We cover data generation for various orbital scenarios, implement and compare different neural network architectures (MLP, LSTM, and PINN), and evaluate their performance in predicting orbital trajectories. Our approach demonstrates the potential of machine learning techniques in solving complex gravitational problems, particularly highlighting the advantages of physics-informed methods. The complete codebase for this project is available on GitHub at [your GitHub repository URL].

## 1 Introduction

The two-body and three-body problems are fundamental challenges in celestial mechanics. This study explores the application of deep learning models to solve these problems, addressing the following key aspects of the assignment:

- Simulation of two-body and three-body problems with varying initial conditions and increased acceleration scenarios.
- Implementation and comparison of different neural network architectures: Multilayer Perceptron (MLP), Long Short-Term Memory (LSTM), and Physics-Informed Neural Networks (PINNs).
- Evaluation of model performance in predicting orbital trajectories for different time horizons (10, 100, and 500 steps).
- Analysis of the advantages of incorporating physical constraints in neural networks through PINNs.

- Assessment of model robustness under different levels of measurement uncertainty.

# 2 Literature Review

Neural networks have shown promise in orbital mechanics, with Liao et al. [?] successfully applying them to find and classify periodic orbits in the three-body problem. The introduction of Physics-Informed Neural Networks (PINNs) by Raissi et al. [?] has further advanced the field, offering a framework to incorporate physical laws directly into neural network architectures. In the context of gravitational problems, Martin and Schaub [?] demonstrated PINNs' efficacy in modeling gravity fields of celestial bodies.

Deep learning approaches have also been applied to related challenges in astrodynamics. Cheng et al. [?] utilized neural networks for real-time optimal control in asteroid landings, while Gao and Liao [?] proposed an efficient gravity field modeling method based on Gaussian process regression. These advancements highlight the potential of machine learning in addressing complex gravitational dynamics.

However, challenges persist in ensuring physical constraints are properly enforced, avoiding overfitting, and generalizing to new scenarios. The physics-informed approach partially addresses these issues, but further research is needed to optimize network architectures, incorporate more comprehensive physical knowledge, and extend these methods to more complex N-body scenarios [?]. This literature review sets the stage for our study, which aims to contribute to this evolving field by exploring the application of neural networks and PINNs to two-body and three-body gravitational problems.

# 3 Methodology

## 3.1 Data Generation

Our data generation process was designed to create diverse, realistic orbital scenarios for two-body and three-body gravitational problems, while incorporating measurement uncertainties to reflect real-world challenges. We implemented three main simulation scenarios: the standard two-body problem, a two-body problem with increased acceleration, and a three-body problem. For the two-body problem, we simulated the motion of a light rotating object (spaceship) around a very heavy stationary object. The initial radius was set to 10,000 m, with the initial velocity calculated to achieve a stable circular

orbit. In the increased acceleration scenario, we introduced an additional sinusoidal force on the spaceship, simulating a propulsion system malfunction. The three-body problem involved simulating the motion of a light object (spaceship) under the gravitational influence of two heavy bodies. We generated varied initial conditions by randomizing the initial position and velocity of the spaceship within reasonable bounds. All simulations were 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 $r$. We used the Euler method for numerical integration to update the position and velocity of the objects:

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

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

To reflect real-world measurement challenges, we introduced a 1% relative uncertainty to the angle, radius, and velocity measurements of the orbiting object. These uncertainties were implemented using Gaussian noise, clipped to ensure the relative error remains within $\pm 1\%$. For each scenario, we simulated a total of 10 trajectories (spaceships) with different initial conditions, each for 100,000 time steps. We implemented rigorous checks to ensure the generated orbits are valid and realistic. For the two-body problem, we verified that the trajectory forms at least one complete orbit, spans a significant range in both x and y directions, and doesn't simply fly away. For the three-body problem, we additionally checked that the trajectory is bounded, shows sufficient complexity, and remains within a reasonable distance from the center of the system. The data was split into training, validation, and test sets. Nine spaceships were used for training and validation, with their trajectories divided into sequences and split using 5-fold cross-validation. One spaceship was reserved exclusively for the test set, providing a truly unseen trajectory for final model evaluation. 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 comprehensive data generation approach provides a robust 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 challenge of predicting orbital dynamics, we implemented and compared three neural network architectures: Multilayer Perceptron (MLP), Long Short-Term Memory (LSTM), and Physics-Informed Neural Network (PINN). Each architecture was designed to capture different aspects of the two-body and three-body problems, including scenarios with increased acceleration.

Table 1: Overview of Model Architectures

| **Feature** | **MLP** | **LSTM** | **PINN** |
|---|---|---|---|
| Input Size | 12 | 12 | 12 |
| Hidden Layers | 3 | 2 | 4 |
| Hidden Size | 256 | 128 | 256 |
| Output Size | 4 | 4 | 4 |
| Activation | LeakyReLU | ReLU | Tanh |
| Dropout Rate | 0.2 | 0.2 | 0.1 |
| Normalization | Batch Norm | Layer Norm | Batch Norm |
| Residual Connections | Yes | Yes | Yes |

The MLP serves as our baseline model, capable of learning non-linear relationships between input features and output predictions. Its architecture includes three hidden layers with 256 neurons each, using LeakyReLU activation and batch normalization.

The LSTM architecture, designed to capture temporal dependencies, utilizes a bidirectional LSTM layer with 128 hidden units and 2 layers. This is followed by two fully connected layers, each with 128 neurons and ReLU activation. Layer normalization is applied after the LSTM layer to stabilize training.

The PINN architecture incorporates physical constraints into the learning process. It consists of four hidden layers with 256 neurons each, using Tanh activation and batch normalization. Crucially, it employs a custom loss function that combines mean squared error with physics-based constraints:

$$L_{total} = L_{MSE} + \lambda(L_{energy} + L_{momentum}) \tag{1}$$

Where $L_{MSE}$ is the standard mean squared error, $L_{energy}$ ensures conservation of total energy (kinetic + potential), and $L_{momentum}$ enforces conservation of linear momentum. The hyperparameter $\lambda$ balances the influence of physical constraints.

All models use residual connections and dropout to facilitate training of deeper networks and prevent overfitting. They were trained using the Adam

optimizer with a learning rate of 0.001 and early stopping (patience of 5 epochs). We employed 5-fold cross-validation to ensure robust evaluation of each model's performance.

The selection of these architectures was motivated by their unique strengths:

1. MLP provides a simple yet effective baseline for comparison.
2. LSTM captures long-term dependencies in orbital trajectories.
3. PINN incorporates domain-specific knowledge, potentially leading to more physically plausible predictions.

By comparing these diverse approaches, we aim to determine the most effective method for predicting orbital dynamics across various scenarios and time horizons, balancing between purely data-driven learning and physics-informed modeling.

#### 3.2.1 Training Configuration

We implemented a consistent training approach across all model architectures to ensure a fair comparison:

- Optimizer: Adam with a learning rate of 0.001
- Loss function:
  - For MLP and LSTM: Mean Squared Error (MSE)
  - For PINN: Custom loss function (detailed below)
- Batch size: 32
- Maximum epochs: 20
- Early stopping: Patience of 5 epochs
- Cross-validation: 5-fold

Training was conducted for each model type (MLP, LSTM, PINN) on three dataset types: two-body problem, two-body problem with increased acceleration, and three-body problem. We employed 5-fold cross-validation to ensure robust evaluation, with one fold reserved for validation in each training iteration.

The `generic_train` function in our codebase handles the training loop, implementing early stopping and model checkpointing. Training progress and performance metrics were logged using Weights & Biases (wandb) for comprehensive experiment tracking.

### 3.2.2 Physics-Informed Neural Networks

The PINN architecture incorporates physical laws into the learning process through a custom loss function:

$$L_{\text{total}} = L_{\text{MSE}} + 0.1 \cdot (L_{\text{energy}} + L_{\text{momentum}}) \tag{2}$$

Where:

- $L_{\text{MSE}}$ is the standard mean squared error between predicted and true values.
- $L_{\text{energy}}$ enforces energy conservation:

$$L_{\text{energy}} = \frac{1}{N} \sum_{i=1}^{N} (E_{\text{pred},i} - E_{\text{true},i})^2 \tag{3}$$

  with $E = \frac{1}{2}v^2 - \frac{GM_1}{r}$, where $v$ is velocity, $G$ is the gravitational constant, $M_1$ is the mass of the central body, and $r$ is the distance from the central body.

- $L_{\text{momentum}}$ ensures momentum conservation:

$$L_{\text{momentum}} = \frac{1}{N} \sum_{i=1}^{N} (p_{\text{pred},i} - p_{\text{true},i})^2 \tag{4}$$

  where $p = mv$ is the momentum.

This loss function is implemented in the `pinn_loss` function, utilizing PyTorch's automatic differentiation capabilities. The physics-based terms are weighted by a factor of 0.1 to balance their influence with the data-driven MSE loss.

By incorporating these physics-based constraints, 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 directly addresses the assignment's focus on PINNs and their potential advantages in solving gravitational problems, particularly for complex scenarios like the three-body problem and cases with increased acceleration.

Our implementation allows for a comprehensive comparison between purely data-driven approaches (MLP and LSTM) and the physics-informed method (PINN) across different orbital dynamics scenarios, providing insights into their relative strengths and limitations.

### 3.3 Inference and Evaluation

Our evaluation process encompasses a comprehensive assessment of model performance across all folds and datasets. We employ 5-fold cross-validation, with models trained on each fold evaluated on their respective validation sets and the unseen test set. Performance metrics are calculated using mean squared error (MSE) for both position and velocity predictions, with results aggregated across all folds to provide robust estimates. We evaluate models on three distinct sets: training spaceships (to assess fitting), validation spaceships (for generalization within known scenarios), and test spaceships (for performance on entirely unseen trajectories). For each set, we compute the mean and standard deviation of MSE for both position and velocity predictions, considering prediction horizons of 10, 100, and 500 steps. This approach allows us to assess model performance across different time scales and problem complexities. Importantly, all predictions are made using normalized data and subsequently denormalized for evaluation, ensuring consistent scaling across different orbital scenarios.

## 4 Results and Discussion

### 4.1 Data Generation and Simulation

#### 4.1.1 Prediction Accuracy

#### 4.1.2 Long-term Prediction

### 4.2 Autoregressive Approach Performance

### 4.3 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.

| 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 2: 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 3: PINN performance and physical consistency metrics averaged over 5 folds

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.

- Future work: experiment with predicting veloctiy and trajectory differnet models

# 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.