N-Body Dynamics Prediction with Physics Informed Neural Networks

Naia Lum (A20461257)

November 17, 2025

Course: MMAE 500

Professor: Scott Dawson

Contents

Abstract

The Three-Body Problem is a special case of the general n-body problem, where three point masses interact through mutual gravitational forces. This paper discusses the implementation of Physics-Informed Neural Networks (PINNs) for learning and predicting the trajectories of gravitational n-body systems. A baseline PINN for the 2-body system is used as a reference for scaling up to higher number of body systems. To support the training of this model, a custom simulation environment was developed to generate 1,000 unique trajectories for systems with two to four point masses and 10,000 additional trajectories to closely study two to three point masses. Results are further looked into Section 3 and show that PINNs can successfully approximate the time-dependent trajectories of chaotic dynamical systems with interpretable physical consistency. The code is designed to be scalable and extendable to other high-dimensional dynamical systems.

1 Introduction

The Three-Body Problem is highly chaotic and leaves no closed-form solution, requiring numerical integration to approximate trajectories. The n-body problem serves as a representative model for a wide range of nonlinear systems, including celestial mechanics, star cluster formation, and plasma dynamics. A Physics-Informed Neural Network (PINN) is developed to learn and predict the continuous dynamics of n-body trajectories directly from simulation data. This approach aims to improve long-term prediction accuracy and generalization, providing a data-driven alternative to conventional solvers for high-dimensional chaotic systems.

1.1 The N-Body Problem

This dynamical problem describes the motion of n-point mass particles under mutual gravitational interactions [1]. Under a system with three or more bodies, most initial conditions cause the system to be highly nonlinear and uncontrollable. It is hard to predict how these orbits will form, and there is limited observational data.

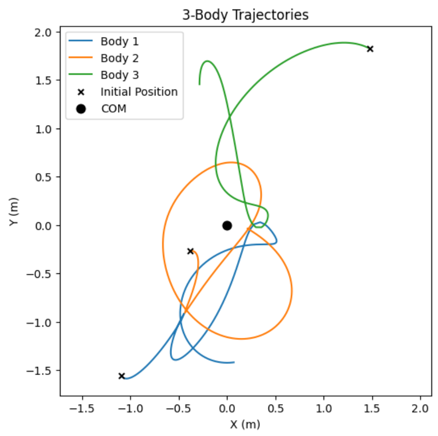


Figure 1: 3-Body Trajectory Simulation example

The n-body problem assumes that the system is isolated and that the only force acting is the gravitational force between the multiple bodies. Each body in this problem is treated as a point mass. Taking an example of a 3-body system under Newtonian mechanics, it becomes apparent that the system exhibits chaotic and sensitive dependence on initial conditions.

$$\ddot{\rho}_1 = -m_2 \frac{\rho_1 - \rho_2}{\|\rho_1 - \rho_2\|^3} - m_3 \frac{\rho_1 - \rho_3}{\|\rho_1 - \rho_3\|^3}, \tag{1}$$

$$\ddot{\rho}_2 = -m_1 \frac{\rho_2 - \rho_1}{\|\rho_2 - \rho_1\|^3} - m_3 \frac{\rho_2 - \rho_3}{\|\rho_2 - \rho_3\|^3}, \tag{2}$$

$$\ddot{\rho}_3 = -m_1 \frac{\rho_3 - \rho_1}{\|\rho_3 - \rho_1\|^3} - m_2 \frac{\rho_3 - \rho_2}{\|\rho_3 - \rho_2\|^3}. \tag{3}$$

Expressing the system in the Lagrangian framework, we define

$$L = T - U,$$

where T is the total kinetic energy and U is the total gravitational potential energy. The total kinetic energy is considered, and the motion of each body is determined by the Euler-Lagrange equations to then reproduce the three equations of motion above.

$$\mathcal{L} = \frac{1}{2}\left[m_1\dot{\rho}_1 + m_2\dot{\rho}_2 + m_3\dot{\rho}_3\right] + \left[\frac{m_2 m_3}{\|\rho_3 - \rho_2\|} + \frac{m_1 m_3}{\|\rho_3 - \rho_1\|} + \frac{m_1 m_2}{\|\rho_2 - \rho_1\|}\right].$$

There are recent numerical discoveries of periodic solutions to this problem. Trajectories such as the figure-8 orbit, butterfly orbit, moth orbit, yarn orbit, and yin-yang orbit [2] have been proven numerically but cannot be compared to any observational data as no known astronomical systems exhibit these behaviors in nature

Because mainly numerical solutions exist for simulating these n-body systems, there is not enough observable data to validate short-time horizon discrete simulations. A Physics-Informed Neural Network (PINN) is attempted to predict the continuous dynamic functions.

1.2 Physics-Informed Neural Networks (PINN)

PINNs are a class of machine learning and deep learning methods embedded with prior physics knowledge of a system in order to solve physical problems. These neural networks ensure that model predictions are not only accurate with the data, but also follow the physical governing equations. This is accomplished by incorporating these equations into the loss function as you can see in Figure 2. Coupling the data-driven model with the physical constraints allows for decreased preparation of labeled data.

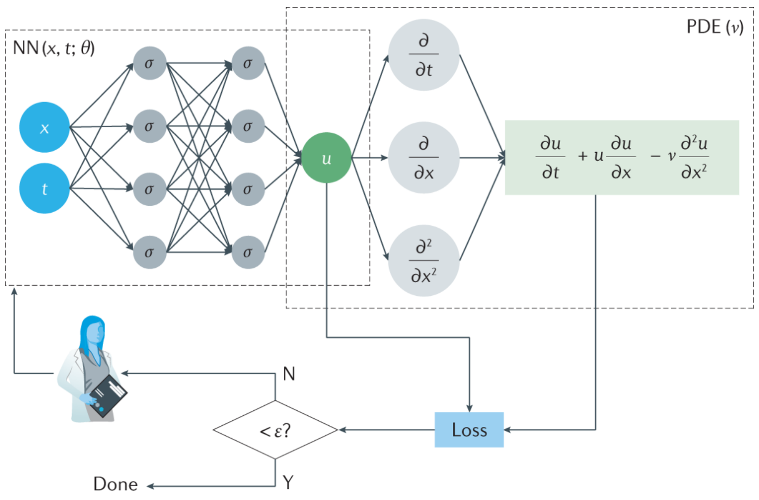


Figure 2: PINN structure with inputs, network layers, policy updates, and embedded physics [3]

2 Methods

In this project, all bodies have the same mass and are randomly placed within an L×L box. Initial velocities are assigned to each body from a uniform distribution, scaled to match the virial speed for approximate equilibrium. To ensure the simulation is in a center-of-mass frame, the total momentum is removed and the system is translated so that the center of mass is at the origin. Each simulation is governed by Newton's equations of motion, where the acceleration of each body is determined by summing the gravitational attraction from all other bodies.

These equations are integrated using numerical solvers to generate trajectory data. Because analytical solutions are insufficient for this problem where the number of bodies is greater than two, and observational data is limited, this simulated dataset provides a useful benchmark for training and evaluating data-driven models.

In this work, the PINN takes time t as the independent input of the n-body system. During training, Pytorch's automatic differentiation is used to compute the time derivatives of the network output position and velocities, which are then compared to the expected gravitational accelerations derived from Newton's law.

An Adam optimizer with a learning rate scheduler is used to minimize the combined loss. After training, the performance of the network is evaluated by computing the root mean square error between the predicted trajectories and the true simulated trajectories. Visualizations of the predicted and true orbits are generated to qualitatively assess the the learned dynamics.

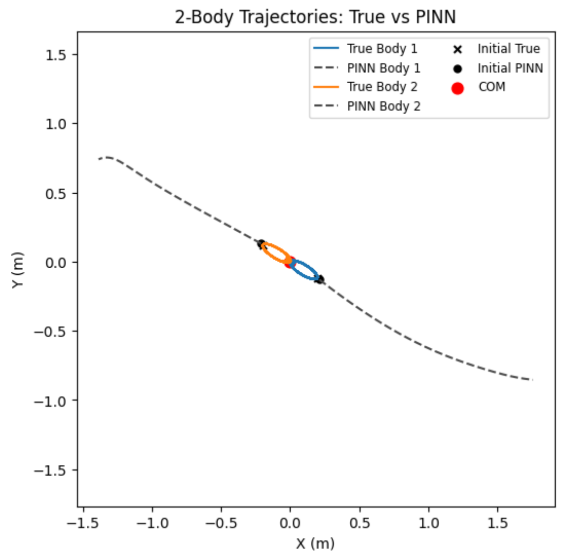
3 Results

Two main developments are required for this project: a PINN must be defined and n-body dynamics data must be acquired.

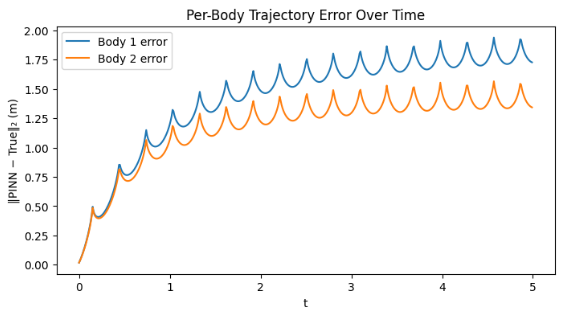
3.1 Definning PINN for N-Body Dynamics

To define the PINN architecture, a stable 2-body system is first modeled and used as a baseline for training and evaluation. In this configuration, the gravitational interaction between the two bodies follows a closed, elliptical orbit, which is analytically solvable. This 2-body system serves as a benchmark to validate the PINN network ability to learn and correct general n-body behavior.

Starting with one set of data used to model the PINN as well as a generic loss function adhered to the results shown in Figure 3



(a) 2-Body PINN Trajectory

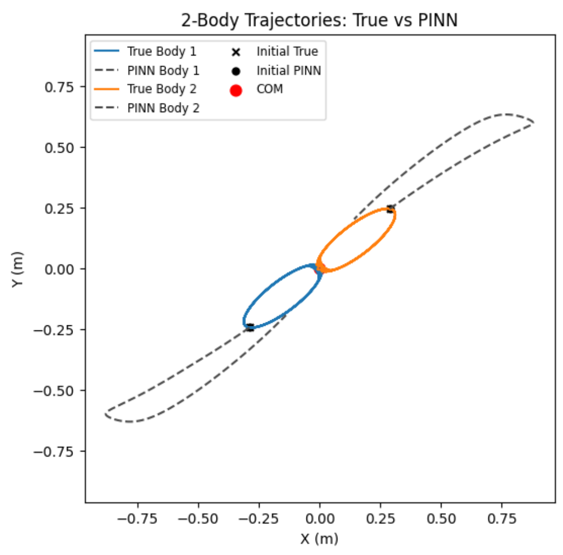


(b) 2-Body PINN Error

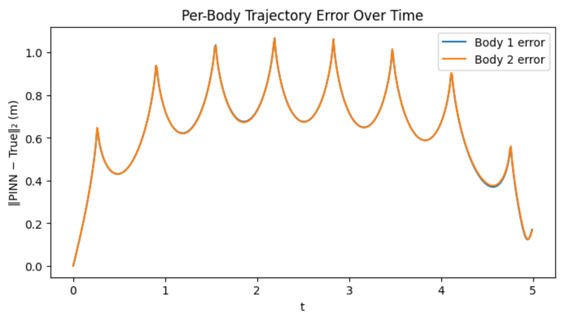
Figure 3: Comparison of 2-Body PINN Trajectory and Error

While the 2-body elliptical orbit was correctly modeled in the True bodies, the PINN was unsuccessful in converging with the cyclical dynamics. The PINN error, which was modeled as root-mean-square estimate of the true position states and PINN-predicted position states, never converges.

In attempts to improve the performance and maintain closer dynamics, it was hypothesized that the PINN was chasing large physics losses and struggling to settle in the ”smaller” oscillations of the ellipse. To combat this, the loss function was redefined to increase penalty on the initial conditions and data errors. However, when adjusting one parameter to alleviate initial large changes in dynamics, the PINN struggled to maintain the decreased error. The loss functions were calculated in a sequence based on epochs: 1) increased penalty on initial conditions, 2) increasing the physics loss, 3) balanced penalties for all three loss functions. The learning rate was also decreased to allow for slower adjustments to the changing dynamics. The neural network initially had 2 layers with 64 nodes; this was adjusted to 128 nodes. Physics collocation points were also increased.



(a) 2-Body PINN Trajectory



(b) 2-Body PINN Error

Figure 4: Comparison of 2-Body PINN Trajectory and Error with updated PINN

These changes improved the PINN error, allowing error convergence and matching the trajectories. Once the PINN was shown to follow the general dynamics of the 2-body problem, it was applied to other n-body dynamics of up to 5 bodies. This increased the complexity and instability of the system. Using this PINN as a base I applied it to the 3-Body Problem.

3.2 Synthetic Data-Trained PINN

Once a general structure for the PINN was formed, it was important to use large datasets to train this model. However, because of the rare nature of this phenomenon and the difficulty in observing it, publically available observable data is difficult to find. The synthetic simulation code was altered to create and save 1000 simulations with random initial conditions. Then the previously defined PINN was trained with an increased set of data to see the results. N-body simulation data was generated and saved for N = 2, 3, 4, and 5 bodies. With each increase of body so did the unstable dynamics and increased processing time.

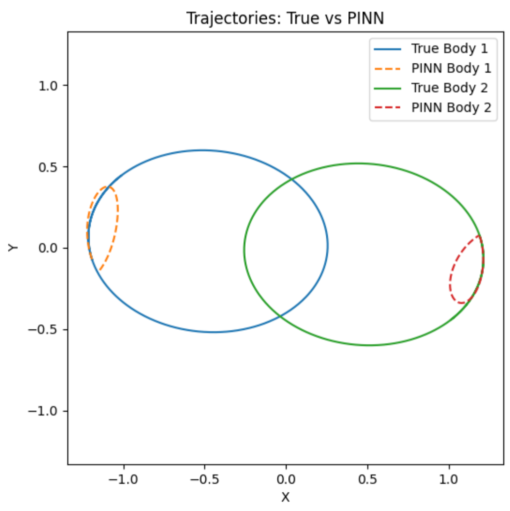
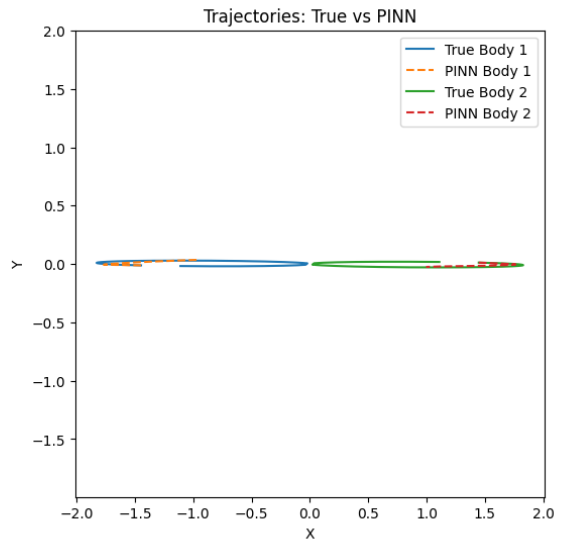
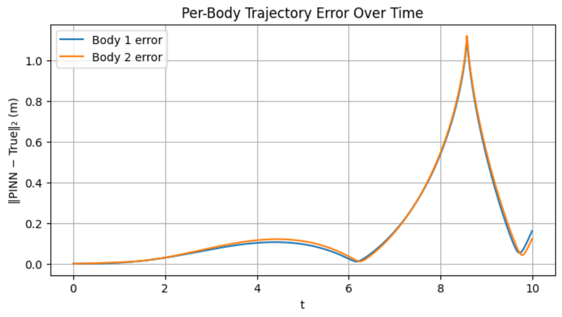


Figure 5: 2-Body PINN trained on 1000 simulations

At first glance the trajectory follows the simulated data's path better than when using one set of data. To further test if increasing the training data improves performance, the PINN is trained with 10,000 simulations for the 2-body problem.



(a) 2-Body PINN trained on 10,000 simulations



(b) Error on the 2-body, 10,000 dataset-trained PINN

Figure 6: Comparison of 2-Body PINN Trajectory and Error with increased number of unique training simulation datasets

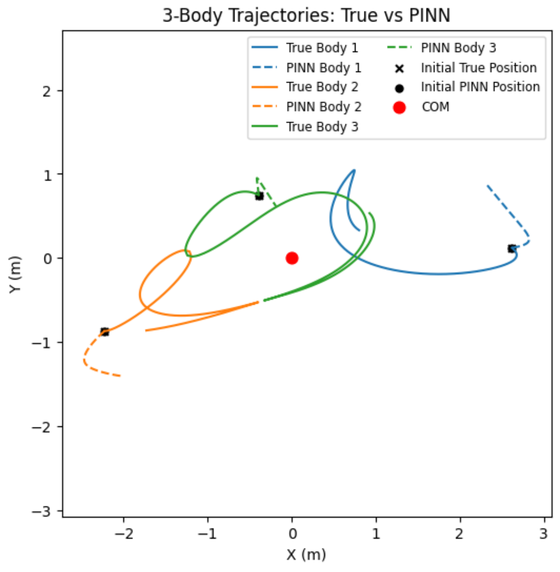
The larger training set led to a reduction in trajectory prediction error. The PINN better tracked the true trajectory across a longer time horizon with significantly reduced drift. Figure 6 shows a low trajectory error over time, with the exception of a jump in error at t=8, highlighting the network's improved performance.

These results show that increasing the variety and number of synthetic data improves PINN prediction accuracy. This effect is especially important for chaotic or long-duration systems where small errors in prediction can quickly grow due to system sensitivity.

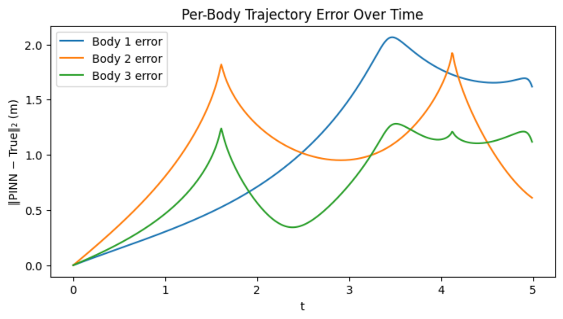
3.3 N-Body Predictions

Having validated the framework on the 2-body case, the PINN was next tested on 3-body and higher-order problems using the same architecture and training pipeline. Each simulation again started with random initial positions and velocities within a bounded LxL box. With increasing n-bodies, trajectory prediction becomes significantly more difficult due to the nonlinear and unstable nature of multi-body gravitational interactions.

The first test was a 3-body and 4-body system over one set of simulation data.

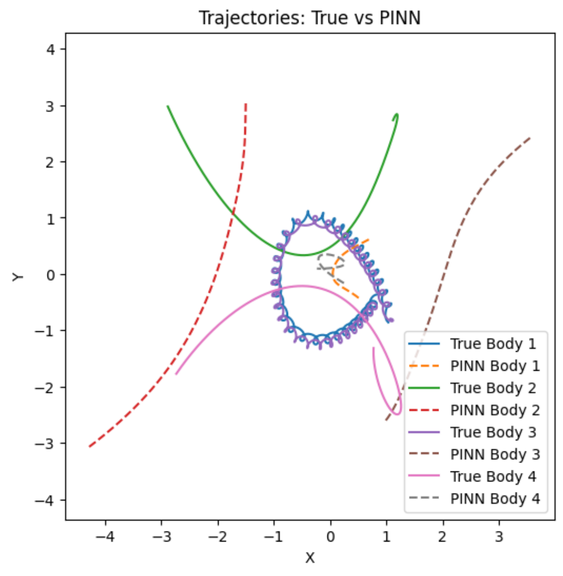


(a) 3-Body PINN Trajectory

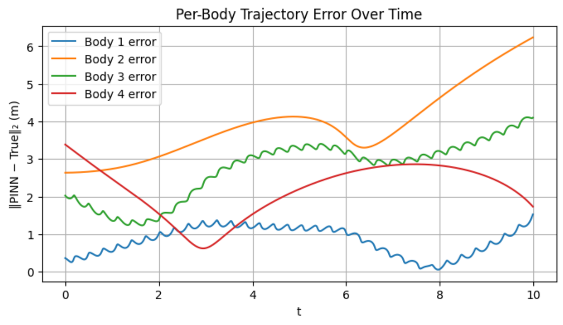


(b) 3-Body PINN Error

Figure 7: Comparison of 3-Body PINN Trajectory and Error



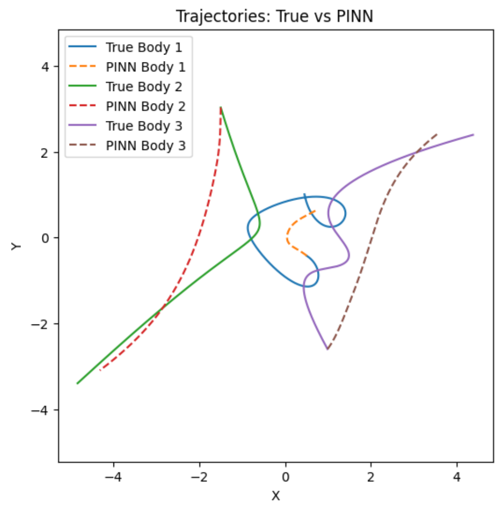
(a) 4-Body PINN Trajectory



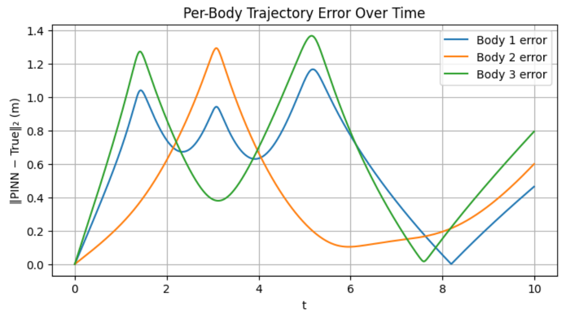
(b) 4-Body PINN Error

Figure 8: Comparison of 4-Body PINN Trajectory and Error

Following the prior work, the 3-body PINN was trained on 10,000 simulations for prediction improvement. The results are shown in Figure 9.



(a) 3-Body PINN trained on 10,000 simulations



(b) 3-Body PINN Error trained on 10,000 simulations

Figure 9: Comparison of 3-Body Trajectory and Error for PINN trained on 10,000 simulations

The network trained on 10,000 3-body simulations shows improved trajectory stability when looking at the reduced root-mean-squared error for each body's trajectory. The errors also seem to decrease within a 1.0 threshold.

4 Discussion

The results demonstrate that Physics-Informed Neural Networks (PINNs) are a promising approach for modeling nonlinear dynamical systems like the n-body problem. While perfect tracking remains difficult in chaotic regimes, the current PINN model proved capable of capturing the general orbital behavior for low n-bodies, with structured loss phasing (high penalty on data at first, with physics penalties ramped over time) allowing for smoother convergence.

Future work will continue scaling this approach by training on 4-body and 5-body datasets using the same volume (10,000 simulations), to analyze how PINN prediction accuracy decreases or remains the same amid increasing system complexity. Additionally, in nature it is unlikely that an n-body system will have equal initial conditions such as mass, gravity, and velocities. It would be important to test these variations in order for this work to be applicable to predicting natural celestial dynamics.

Acknowledgement

This project is completed with strong inspiration from this recent paper [4].

The large language model ChatGPT from Open AI [5] is used for general research purposes and fixing syntax while programming.

References

1. CONLEY, C. in *International Symposium on Nonlinear Differential Equations and Nonlinear Mechanics* (eds LaSalle, J. P. & Lefschetz, S.) 86–90 (Academic Press, 1963). ISBN: 978-0-12-395651-4. https://www.sciencedirect.com/science/article/pii/B9780123956514500143.
2. Islam, A. *Building a Physics-Informed Neural Network (PINN) Using PyTorch From Scratch* Medium article; 5 min read. Tech Spectrum. https://medium.com/tech-spectrum/building-a-physics-informed-neural-network-pinn-using-pytorch-from-scratch-cfdb161c2a14.
3. Karniadakis, G. E. *et al.* Physics-informed machine learning. *Nature Reviews Physics* **3,** 422–440. https://doi.org/10.1038/s42254-021-00314-5 (2021).
4. Pereira, M. S., Tripa, L., Lima, N., Caldas, F. & Soares, C. *Advancing Solutions for the Three-Body Problem Through Physics-Informed Neural Networks* 2025. arXiv: 2503.04585 [cs.LG]. https://arxiv.org/abs/2503.04585 (2025).
5. OpenAI. *ChatGPT* https://openai.com/chatgpt.

A Project code

```python

###### Code used to generate n-body simulation data ######
import numpy as np
from scipy.integrate import solve_ivp
import os

# simulation givens
G = 1.0 #gravity
L = 3.0 #LxL box
dt, Nt = 0.01, 1000
t_sim = np.linspace(0, dt*(Nt-1), Nt)

def n_body_rhs(t, state, n, masses):
    pos = state[:2*n].reshape(n,2)
    vel = state[2*n:4*n].reshape(n,2)
    acc = np.zeros_like(pos)
    for i in range(n):
        for j in range(n):
            if i==j: continue
            rij = pos[i]-pos[j]
            acc[i] += -G*masses[j]*rij/np.linalg.norm(rij)**3
    return np.hstack((vel.flatten(), acc.flatten()))

def random_initial_conditions(L, v_scale, n, masses):
    pos0 = np.random.uniform(-L,L,(n,2))
    vel0 = np.random.uniform(-v_scale,v_scale,(n,2))
    Mtot = masses.sum()
    COM_r = (masses[:,None]*pos0).sum(0)/Mtot
    COM_v = (masses[:,None]*vel0).sum(0)/Mtot
    pos0 -= COM_r; vel0 -= COM_v
    return np.hstack((pos0.flatten(), vel0.flatten()))

# Loop over number of bodies
for n in range(3, 4):
    masses = np.ones(n)  # masses=1
    Mtot = masses.sum()  # equals n when masses are 1
    v_scale = np.sqrt(G * Mtot / L)

    num_simulations = 10000  # change this!!!!
    save_dir = f'simulations/nbody_{n:02d}_10000'
    os.makedirs(save_dir, exist_ok=True)

    print(f"\n--- Running {num_simulations} simulations for {n}-body
   system ---")

    for sim_idx in range(num_simulations):
        x0 = random_initial_conditions(L, v_scale, n, masses)

        sol = solve_ivp(
            n_body_rhs, (0, t_sim[-1]), x0, t_eval=t_sim,
            args=(n, masses), rtol=1e-9, atol=1e-9
        )
        data_sim = sol.y.T  # (Nt,4n)

        # Save trajectory
        np.savez(f'{save_dir}/sim_{sim_idx:03d}.npz',
                 t_sim=t_sim, data_sim=data_sim,
                 initial_conditions=x0, masses=masses)

        if (sim_idx+1) % 50 == 0 or sim_idx == num_simulations-1:
            print(f"Completed {sim_idx+1}/{num_simulations} simulations for {n}-body")

print("\n All simulations completed and saved.")

###### PINN ######

import numpy as np
import torch
import torch.nn as nn
import torch.optim as optim
import torch.autograd as ag
import matplotlib.pyplot as plt
import glob, os


# Select n-body simulation to train on
n = 2
sim_dir = f'simulations/nbody_{n:02d}'
sim_file = glob.glob(os.path.join(sim_dir, '*.npz'))[0]  # pick first sim for example

data = np.load(sim_file)
t_sim = data['t_sim']
data_sim = data['data_sim']

t_data = torch.tensor(t_sim, dtype=torch.float32).unsqueeze(1)
x_data = torch.tensor(data_sim, dtype=torch.float32)

G = 1.0
masses = np.ones(n)

# PINN Architecture (MLP)
class PINN(nn.Module):
    def __init__(self, n, hidden=[128,128]):
        super().__init__()
        dims = [1] + hidden + [4*n]
        layers = []
        for i in range(len(dims)-2):
            layers += [nn.Linear(dims[i],dims[i+1]), nn.Tanh()]
        layers += [nn.Linear(dims[-2], dims[-1])]
        self.net = nn.Sequential(*layers)
    def forward(self,t):
        return self.net(t)

```

```python
model = PINN(n).float()
optimizer = optim.Adam(model.parameters(), lr=1e-3)
scheduler = optim.lr_scheduler.ReduceLROnPlateau(optimizer, patience=500,
    factor=0.5, min_lr=1e-6)
mse = nn.MSELoss()

Ncol = 1000 # collocation points
t_col = torch.tensor(np.random.uniform(0, t_sim[-1], (Ncol,1)),
                     requires_grad=True, dtype=torch.float32)

# Loss Functions: Data, initial conditions, physics. Also includes the
    loss stages here...
def data_loss():
    return mse(model(t_data), x_data)

def ic_loss():
    x0_pred = model(torch.zeros(1,1))
    return mse(x0_pred, x_data[0:1])

def physics_loss():
    x_pred = model(t_col)
    x_t = torch.cat([
        ag.grad(x_pred[:,j], t_col, grad_outputs=torch.ones_like(x_pred[:,
    j]), create_graph=True)[0].unsqueeze(1)
        for j in range(4*n)], dim=1)
    x_tt = torch.cat([
        ag.grad(x_t[:,j], t_col, grad_outputs=torch.ones_like(x_t[:,j]),
    create_graph=True)[0].unsqueeze(1)
        for j in range(4*n)], dim=1)

    pos_pred = x_pred[:, :2*n].view(-1,n,2)
    acc_pred = x_tt[:, :2*n].view(-1,n,2)

    loss_p = 0.0
    for i in range(n):
        force_sum = torch.zeros_like(acc_pred[:,i])
        for j in range(n):
            if i==j: continue
            rij = pos_pred[:,i]-pos_pred[:,j]
            force_sum += -G*masses[j]*rij/(rij.norm(dim=1,keepdim=True)
    **3)
        loss_p += mse(acc_pred[:,i], force_sum)
    return loss_p

# Multi-phase loss Training
epochs = 5000
for epoch in range(epochs):
    optimizer.zero_grad()

    if epoch < 1000:    # Phase 1: Data & IC only
        loss = data_loss() + 1000*ic_loss()
    elif epoch < 4000:  # Phase 2: Ramp up physics
        frac = (epoch-1000)/3000
        loss = data_loss() + (frac)*physics_loss() + 1000*ic_loss()
    else:                 # Phase 3: All losses
        loss = data_loss() + physics_loss() + 1000*ic_loss()

    loss.backward()
    optimizer.step()
    scheduler.step(loss)

    if epoch % 500 == 0:
        print(f"Epoch {epoch}, Loss {loss.item():.3e}")

# evaluation
model.eval()
with torch.no_grad():
    x_pinn = model(t_data).cpu().numpy()

# RMSE calculation
rmse = np.sqrt(np.mean((x_pinn - data_sim)**2))
print(f'Overall RMSE: {rmse:.3e}')

# plotting for trajectories
plt.figure(figsize=(6,6))
for i in range(n):
    plt.plot(data_sim[:,2*i], data_sim[:,2*i+1], '-', label=f'True Body {i+1}')
    plt.plot(x_pinn[:,2*i], x_pinn[:,2*i+1], '--', label=f'PINN Body {i+1}')
plt.legend()
plt.xlabel('X'); plt.ylabel('Y')
plt.title('Trajectories: True vs PINN')
plt.axis('equal')
plt.show()

# Compute error for each body
Nt = data_sim.shape[0]
n = data_sim.shape[1] // 4  # each body has x, y, vx, vy
errors = np.zeros((Nt, n))

for i in range(n):
    true_xy = data_sim[:, 2*i:2*i+2]   # true x, y
    pred_xy = x_pinn[:,  2*i:2*i+2]    # predicted x, y
    errors[:, i] = np.linalg.norm(pred_xy - true_xy, axis=1)

# Plot error over time
import matplotlib.pyplot as plt
plt.figure(figsize=(8,4))
for i in range(n):
    plt.plot(t_sim, errors[:, i], label=f'Body {i+1} error')
plt.xlabel('t')
plt.ylabel('  PINN        T r u e    (m)')
plt.title('Per-Body Trajectory Error Over Time')
plt.legend()
plt.grid(True)
plt.show()
```