UNIVERSITÀ
DEGLI STUDI
DI PADOVA

DIPARTIMENTO
DI INGEGNERIA
DELL'INFORMAZIONE

DEPARTMENT OF INFORMATION ENGINEERING

MASTER'S DEGREE IN COMPUTER ENGINEERING: HIGH PERFORMANCE AND BIG DATA

Optimization of Physics-Informed Hamiltonian Neural Networks

Supervisor
Prof. Nanni Loris

Graduating
Sabbah Samir

Co-supervisor
Prof. Fantozzi Carlo

ACADEMIC YEAR 2024-2025

Date of graduation 18/07/2025

To my Family

Abstract

We study some optimization approaches in a specific type of artificial neural network called Physics-Informed Neural Networks, PINNs for short. These types of network make use of some of the groundwork already done in the scientific field, such as Newtonian physical laws, and enforce this knowledge in the training of the network to accurately predict the simulation of the data of some systems, with as few data points as possible. The target system in this paper is the famous Hamiltonian system, which describes almost any classical/non-classical system, usually laid out in the form of partial differential equations. The main job here is to find accurate approximations for the hidden solutions of these PDEs. In this work, examples expand to problems involving Hamiltonian systems such as the mass spring, pendulum, and two-body or three-body systems. The work builds on the previous work of the Greydanus paper on Hamiltonian neural networks; however, we integrate an additional theorem involved in Hamiltonian mechanics called the Liouville theorem, which tells that the propagation of the dynamics of a conserved system is in equilibrium with respect to its combined momentum and position. The results show that enforcing this equilibrium equation on the output of the network and automatically differentiate it leads to better system integration, and therefore to better energy outputs of the entire Hamiltonian system under study.

Contents

Chapter 1

Brief Introduction to Artificial Intelligence

1.1 What is an Artificial Intelligence

In the past few years, Artificial Intelligence has gained an exponential amount of fame thanks to the rapid changes in the field and its effects in multidisciplinary fields such as STEM, biology, medicine and much more.However, the field itself has a rich history dating back to the 1950s. AI focuses on creating and implementing computer systems capable of addressing problems that typically require human intelligence. These problems often involve high-complexity or natural tasks such as vision or understanding natural language, which classical algorithmic methods cannot solve.

The core functionality that differentiates AI from traditional computing is that it utilizes various types of knowledge specific to an application domain. Consequently, issues related to knowledge acquisition, representation, and utilization are central to AI research and development. One of the most active areas in AI is the implementation of knowledge-based systems [7].

What we mean by knowledge-based systems is that they depend on chunk-based information in order to understand or solve the problem at hand. Some of the tasks required are organizing, topologically sorting, and labeling these pieces of data, which will ultimately result in a more comprehensive view of the parameters of the problem to be solved[19].

1.2 PINNs

One of the bottlenecks of AI is the abundance of data. This turns out to be a problem in the scientific field (simulating the dynamics of a Schrodinger equation, interaction between molecules, etc.). integrating traditional AI approaches towards such problems requires a huge amount of data to train on. However, due to the mathematical complexity of given problems, it is sometimes impractical to get enough data points.

In 2017, a paper with the title ”Physics-Informed Deep Learning (Part I): Data-driven Solutions of Nonlinear Partial Differential Equations” was introduced to become a blueprint for future endeavors to tackle more accurate and precise simulations for scientific works. In the original paper, it explains *physics-informed neural networks* as supervised learning tools, making use of the rich background in the scientific field so far, such as theorems and equations that have already been established that describe a certain phenomenon in physical or biological fields[12].

1.3 HNN

The project under study in this paper is called *Hamiltonian Neural Networks*, first introduced in 2019 by Sam Greydanus, then a Google Brain employee. In the original paper, Sam and his colleagues introduced the potential of integrating well-known physical formulas into neural networks.

The problem with traditional neural networks, according to the paper, is that they only approximate the behavior of a physical system, when, in fact, these networks fail to learn exact physical laws and, as a result, fail to conserve energy levels and predict the same energy patterns of the system as it is being modeled. So, the paper proposed a new methodology to try and integrate well-known physical formulas and parameterize them within the network learning pattern rather than just learning directly from the data.

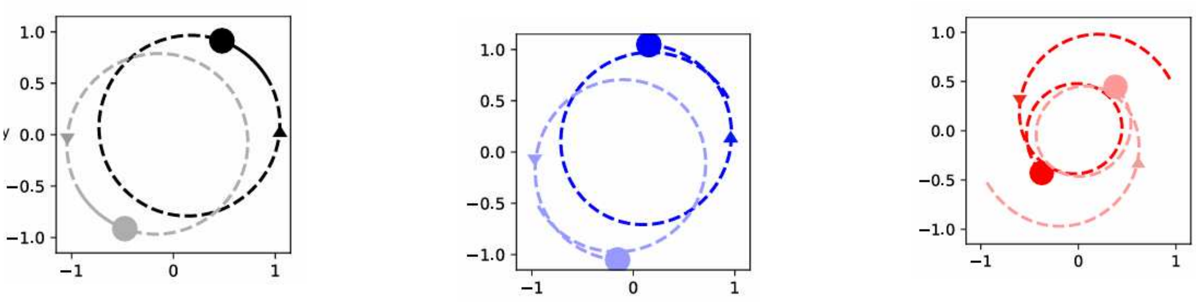
Sam et al. approached the Hamiltonian problem from two different perspectives, to consider the system in a continuous form and to predict the next state of a data point through some sort of differential operator that has a well-defined integral, since prediction of next states in a discrete time-stepping way fails to capture the dynamics in the real world. The second problem they tackled was how to simulate conservation of a system (through its energy) and not let the prediction drift away. [5]

In Hamiltonian mechanics, a system can be described through the two scalar coordinates **q**, representing the position, and **p**, for momentum. Both of these scalars can be converted into a N-dimensional space vector accordingly, as well as measuring their progress in a time-step fashion, resulting in a time vector of the following shape $(q = q1....qnandp = p1.....pn)$. Hamilton in his original work related these scalars into a system of equations that define the progress of the position (q) and momentum (p) of any system through time and thus relate at any given time the energy of the system at that exact moment.

the *Hamiltonian Neural Network* employed the following system of equations in their network to learn the exact Hamilton dynamics:

$$\frac{dq}{dt} = \frac{\partial H}{\partial p}, \frac{dp}{dt} = -\frac{\partial H}{\partial q} \tag{1.1}$$

Learning the gradients of the above system means learning the exact time evolution of systems with total energy conservation.



(a) true 2-body dynamics

(b) HNN solution

(c) Regular Baseline (normal MLP NN)

Figure 1.1: dynamics of the 2-body problem orbiting around each other

1.4 Liouville Theorem

Our optimization of the Hamiltonian network involved the addition of a mathematical theorem called the Liouville theorem. Integrating general Hamiltonian mechanics into the network and training on different experiments (mass spring, pendulum, orbiting k-bodies, etc.).) seemed too abstract, as different experiments have different complexities in real life. We propose using an additional theorem to be used, which is Liouville's theorem.

In short, the theorem states that, in a conserved system, while the dynamics of the system changes as time evolves, changing the position and momentum (p and q) of the Hamiltonian system, the space that limits the vector field where the system operates does not change the volume where it operates in[10], and therefore creates an equilibrium between the dynamics of the elements of the system, which in that case are the position, q, and the momentum p.

Given the following general representation of the Liouville equilibrium of a Hamiltonian system, it was clear that we had to go with a double forward pass. in order to approximate some integration of the following equation into the network.

given v the velocity of the Hamiltonian system:

$$v = (\dot{q},\dot{p}) = (\frac{\partial H}{\partial p}, -\frac{\partial H}{\partial q}) \tag{1.2}$$

The divergence of the velocity above, in a conserved state, must be equal to zero:

$$\begin{aligned} \nabla . v &= \frac{\partial \dot{q}}{\partial q} + \frac{\partial \dot{p}}{\partial p} \\ &= \frac{\partial}{\partial q}\frac{\partial H}{\partial p} - \frac{\partial}{\partial p}\frac{\partial H}{\partial q} \\ &= 0 \end{aligned}$$

The inspiration for doing the double forward pass was from the second line of the equation above, due to the chained partial equations.

We will start by delving deeper into the general idea of physics-informed neural networks (PINNs) in Chapter 2, then for Chapter 3 we explain in ore detail what exactly is the Hamiltonian neural network, and the different experiments used in the original paper, while focusing on the two and three body problems. In Chapter 4 we deconstruct our approach with Liouville's theorem and the tuning of other parameters in order to obtain better results. Then we conclude by comparing our results with the original HNN, as well as look at future different approaches.

Chapter 2

Introduction to Physics-Informed Neural Networks

2.1 What is a PINN

As Artificial Intelligence gains more and more publicity over its uses in technologies such as LLM, it is also now more than ever being employed in other scientific fields tailored to solve specific needs in these areas. Pharmaceuticals are one of the fields where AI is heavily employed to discover new medicines, molecular structures, or even to treat cancer[15].

However, as the complexity of the physical/biological structure increases, so does their mathematical representation through Artificial Neural networks. In other words, mapping these systems into any machine learning model becomes non-linear.Different approaches have been utilized to better approximate such systems, especially through different Gaussian processes such as Gaussian process regression[13][14][11].

Gaussian processes, despite their mathematical appeal, encounter two key limitations when handling non-linear problems. First, dealing with non-linear terms often requires simplifications like local linearization, which restricts their use to specific time domains and reduces prediction accuracy in strongly non-linear scenarios. Second, their reliance on Bayesian frameworks requires prior assumptions, which can constrain the model's flexibility and expose vulnerabilities, especially in complex nonlinear situations[12].

In light of such news, a new form of Neural Networks started taking form called *Physics-Informed Neural Networks*.in its abstract form, these networks try to discover either accurate solutions or new formulas for already established physical systems or behaviors. To be more specific, they employ certain physical facts about a system under study in order to discover correct solutions of partial differential equations (PDEs) describing the said system.

2.2 Why Use Differential Equations

PDEs can represent various behaviors of the system in space-time coordinates. whenever there is an event or a transformation in some medium, there will be some differential equation (ordinary or partial) that tries to describe it.

One field in which differential equations are used is mathematical biology. A key question they address is how the size of a population, such as dividing cells or bacteria, changes between two points in time. If we let x_n represent the population at time n and x_{n+1} the population at time $n + 1$, the growth during this period is described by the logistic map.

$$x_{n+1} = rx_n(1 - x_n) \tag{2.1}$$

Here, x_0 stands for the starting population at time zero, r is a positive constant that reflects the growth rate, and the final term accounts for increased competition as the population gets larger (for example, due to limited resources).

Models using such equations treat time as progressing in individual steps, similar to cellular automata or agent-based models, where biological rules are incorporated into the mapping process. These equations can include multiple variables or even values from various time steps without difficulty. Although the basic idea is straightforward, the resulting patterns can be unpredictable, often producing intriguing emergent phenomena.

To expand on difference equations, one can reduce the time intervals to infinitely small steps.

In this case, the equation predicts the rate at which a variable changes rather than its next value. For instance, continuous logistic growth is described by the following ordinary differential equation:

$$\frac{\mathrm{d}x(t)}{\mathrm{d}t} = rx(t)(1 - x(t)) \tag{2.2}$$

Here, $x(t)$ represents the population at time t, $\frac{\mathrm{d}x(t)}{\mathrm{d}t}$ is the rate of population change, and t can be any positive number. Discrete-time modeling works well for ecology scenarios where new organisms are born in synchronized intervals. However, this approach is less accurate for cases such as bacterial division, where division times differ between populations.[3]

As the behavior of the system increases in its complexity, especially with the codependency of variables within, more and more ordinary and partial differential equations arise with it. So, it is clear that the need to solve such equations can have an impact on several fields of science.

2.3 Architecture of PINNs

Going back to PINNs, we define them as universal approximators of PDEs[12]. The idea is that instead of just using traditional neural networks or machine learning techniques such as random forest search algorithms as blind predictors, known as *"black-box"* algorithms, PINNs inject some prior knowledge of the system into the Loss function or the activation functions of the network/model used. We formalize the above information in the following way.

We consider solving partial differential equations using data-driven methods. These equations take the general form:

$$u_t + \boldsymbol{\mathcal{N}}[u] = 0, x \in \Omega, t \in [0, T]. \tag{2.3}$$

Here, $u(t, x)$ represents the hidden (latent) solution to be approximated by the network, and $\mathcal{N}[\cdot]$ is a nonlinear differential operator.

For continuous models, we define $f(t, x)$ using the left-hand side of the equation above:

$$f := u_t + \mathcal{N}[u]. \tag{2.4}$$

To solve this, $u(t, x)$ is approximated by a deep neural network. It is the output of a regular feedforward function of any neural network. Now, the function *f(t,x)* is where we differentiate using Automatic Differentiation over the result of *u(x,t)* in order to better approximate the learnable parameters of the hidden non-linear operator described above. This combination of f and u results in a physics-informed neural network $f(t, x)$. It shares the same parameters as the neural network for $u(t, x)$ but applies different activation functions due to the differential operator N. The shared parameters between the networks are optimized by minimizing the following updated mean squared error loss[12].

$$MSE = MSE_u + MSE_f \tag{2.5}$$

With

$$MSE_u = \frac{1}{N_u} \sum_{i=1}^{N_u} |u(t_u^i, x_u^i) - u^i|^2 \tag{2.6}$$

and

$$MSE_f = \frac{1}{N_f} \sum_{i=1}^{N_f} |f(t_f^i, x_f^i)|^2 \tag{2.7}$$

The set t_u^i, x_u^i, u^i represents the initial and boundary training data for $u(t, x)$, while t_f^i, x_f^i specifies the collocation points for $f(t, x)$. The loss function MSE_u corresponds to fitting the initial and boundary conditions for $u(t, x)$, while MSE_f ensures that the structure defined by the equation above is satisfied at a finite number of collocation points.

Collecting data points for the above aforementioned equations strictly depends on the nature of the equation or system to be modeled. For example, in the HNN paper, some of the experiments were easy to simulate, such as the mass-spring experiment or the pendulum, because of the abundant research and equations available to properly model the dynamics of such systems.

Since the nature of a PINN allows tackling datasets with as low as only hundreds of data points, it did not require the researchers to generate target datasets for their experiments by deploying standard differentiation equation solvers in order to acquire such collocation points.

2.4 Examples of the use of PINNs

The one-dimensional nonlinear Schrodinger equation (NLSE) is a key equation in quantum mechanics, used to model various systems like nonlinear wave propagation in optical fibers and waveguides, Bose-Einstein condensates, and plasma waves. The ”nonlinear” aspect comes from different physical phenomena: in optics, it is due to a material's intensity-dependent refractive index, while in Bose-Einstein condensates, it arises from the mean-field interactions within a many-body system.The problem we are looking at involves the NLSE with periodic boundary conditions, defined by:

$$
\begin{aligned}
& ih_t + 0.5h_{xx} + |h|^2 h = 0, \qquad x \in [-5,5], \qquad t \in [0,\frac{\pi}{2}], \\
& h(0,x) = 2\,\mathrm{sech}(x), \\
& h(t,-5) = h(t,5), \\
& h_x(t,-5) = h_x(t,5),
\end{aligned}
$$

Here, $h(t,x)$ is a complex-valued solution.

To solve this, we define a residual function:

$$f := ih_t + 0.5h_{xx} + |h|^2 h, \tag{2.8}$$

We then use a complex-valued neural network to approximate h(t,x). This means if we represent $h(t,x)$ as $[u(t,x) \quad iv(t,x)]$ (where u is the real part and v is the imaginary part), we are essentially using a multi-output neural network to predict both u and v. This approach creates a complex-valued, multi-output Physics-Informed Neural Network (PINN).

The neural network parameters are trained by minimizing a mean squared error (MSE) loss function, which has three components:

- MSE_0 (Initial Data Loss): This term ensures that the network predictions match the given initial condition $h(0{,}x)$.
- MSE_b (Boundary Condition Loss): This term enforces the periodic boundary conditions for both h and its spatial derivative h_x.
- MSE_f (Physics Loss): This crucial term penalizes the residual function $f(t{,}x)$ if it deviates from zero, ensuring that the predicted solution $h(t{,}x)$ satisfies the NLSE.

These loss components are calculated by adding the squared errors at specific location points: x_i^0 for the initial data, t_i^b for the boundary conditions, and $(t_i^f{,}x_i^f)$ for the physics equation.

After employing different techniques in order to produce a perfect simulation including fourth-order Runge-Kutta integration steps, the data are used as target data to compare the PINN's produced dynamics of the Schrodinger equations above. Below is a figure showing the near-perfect results between the PINN produced solution and the ground-truth solution of the equations:

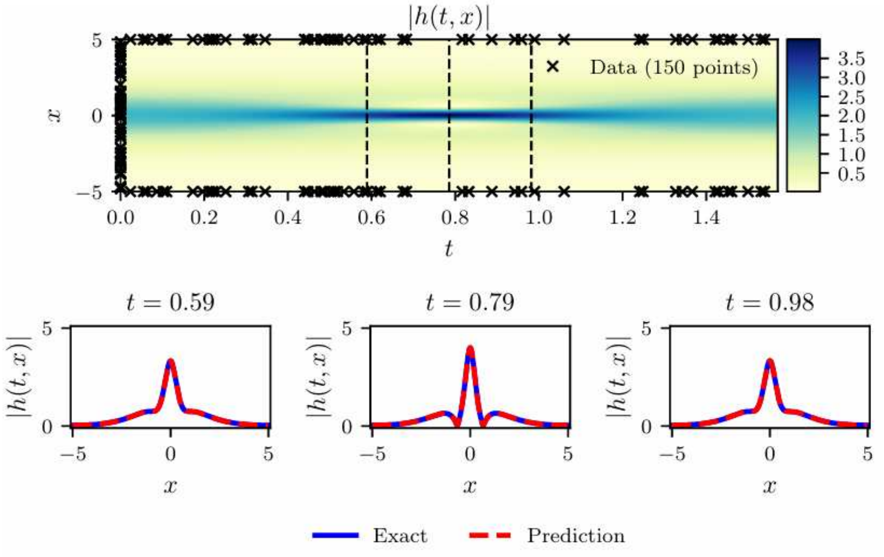


Figure 2.1: Schrodinger equation: Top:Predicted solution $|h(t,x)|$ along with the initial and boundary training data.In addition we are using 20,000 collocation points generated using a Latin Hypercube Sampling strategy. Bottom: Comparison of the predicted and exact solutions corresponding to the three temporal snapshots depicted by the dashed vertical lines in the top panel.The relative $\mathcal{L}_2$ error for this case is $1.97 \cdot 10^{-}3$
[12]

Chapter 3

Original Hamiltonian Neural Network

3.1 Mathematical Overview

Hamiltonian systems are a fundamental framework in classical mechanics that describes the evolution of a physical system using Hamilton's equations. These systems are characterized by a scalar function known as the Hamiltonian, denoted as *H(p,q)*,which represents the total energy of the system, typically the sum of the kinetic and potential energy[6]. The Hamiltonian formalism provides deep insights into the dynamics of a system, even when explicit solutions are difficult to obtain.

As we show in the Introduction, a Hamiltonian system is defined by a set of generalized coordinates **q** (position) and generalized momentum **p**, then, the evolution of the same, by the time parameter t, is maintained through the following equations:

$$\frac{dq}{dt} = \frac{\partial H}{\partial p}, \frac{dp}{dt} = -\frac{\partial H}{\partial q} \tag{3.1}$$

These equations describe how the system evolves over time in phase space. Unlike Newtonian mechanics, which relies on forces and accelerations, Hamiltonian mechanics reformulates the problem in terms of energy conservation and Symplectic geometry,expressed in the form of an integral over the Hamiltonian function, both of which are described by the phase space generated from the dynamics of the system. The reason why we stress about the phase space is because of its purpose to ensure total conservation of the system's energy as time evolves.It is a multidimensional space where each point represents a unique configuration of the system, defined by generalized coordinates p and q mentioned above, and the evolution of a system, described in such phase space, follows Hamilton's equations, ensuring that the trajectory remains deterministic and governed by the system's energy function.

It becomes particularly useful, as we will also see in the paper experiments used in HNN, as it allows for a geometric interpretation of dynamics. Instead of analyzing individual trajectories of a single data point at a time, one can study the overall structure of phase space, including conserved quantities, equilibrium points, and chaotic behavior. This ensures that the total energy remains constant throughout the evolution of the system. This property is crucial in conservative systems, such as planetary motion and harmonic oscillators, where energy is preserved.

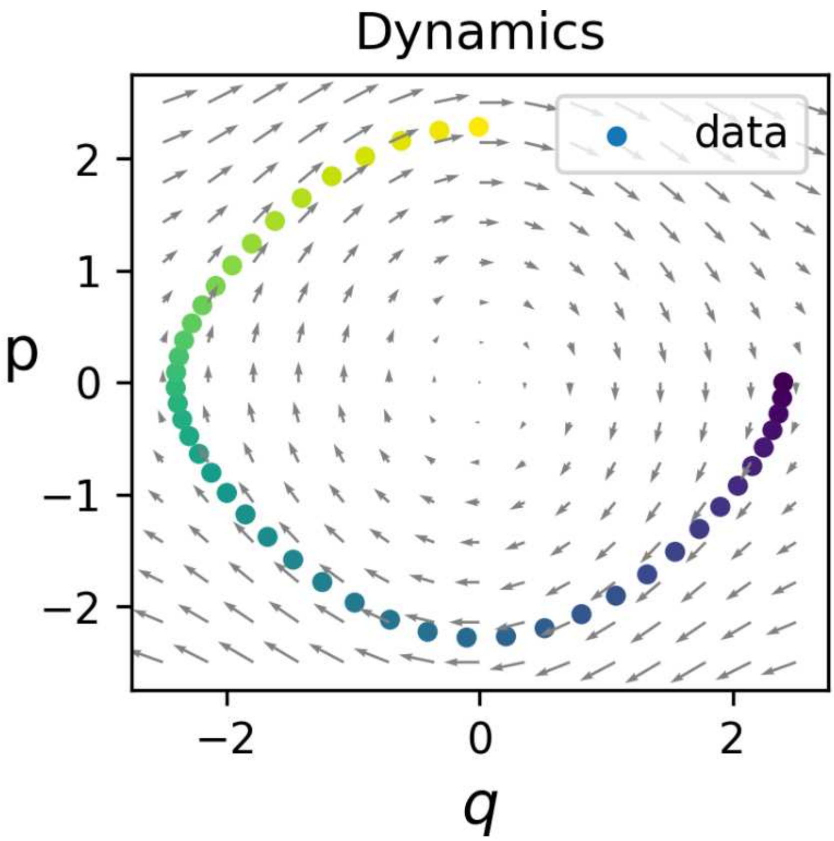


Figure 3.1: phase field of the trajectory of an ideal pendulum

3.2 Design of HNN

We begin by explaining the architecture of the HNN, specifically the modified loss function in use. Building on top of Hamiltonian mechanics and their core mathematical equations mentioned above, Greydanus et al. approached learning from data using Neural Networks through the direct modification of the loss function.

In most of the experiments mentioned in the paper (mass spring,ideal pendulum, etc.), the

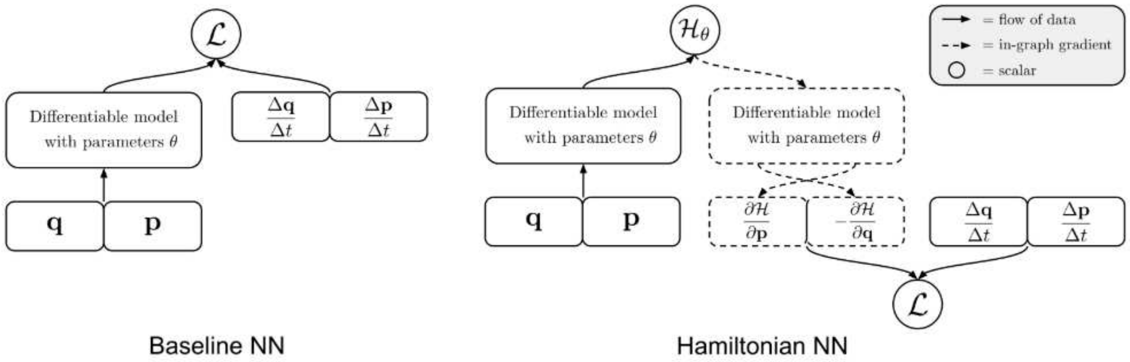

$\mathcal{L}$
Differentiable model with parameters θ
$\frac{\Delta q}{\Delta t}$
$\frac{\Delta p}{\Delta t}$
q
p
Baseline NN
$\mathcal{H}_\theta$
Differentiable model with parameters θ
Differentiable model with parameters θ
q
p
$\frac{\partial \mathcal{H}}{\partial p}$
$-\frac{\partial \mathcal{H}}{\partial q}$
$\frac{\Delta q}{\Delta t}$
$\frac{\Delta p}{\Delta t}$
$\mathcal{L}$
Hamiltonian NN
= flow of data
= in-graph gradient
= scalar

Figure 3.2: Schema of the Baseline and HNN models. The Baseline NN in the figure above represents the supervised learning approach to modeling the time derivatives of (q, p). In both cases, the inputs are the coordinates of the system and the targets are their time derivatives.

flow of the program is as follows:

- **input (x):** a PyTorch tensor data structure representing (p,q) quantities of the real experiment. In other words, these are the real coordinates (p and q) of a system that evolves over time.
- **dxdt :** this is the target tensor that has the same shape as the input, but it represents the total derivative of the input coordinate (q, p), i.e. **dxdt** represents $(\frac{dq}{dt}, \frac{dp}{dt})$
- **test x :** the test partition from the original dataset, which represents 20 percent of the dataset.
- **test dxdt :** the test partition of **dxdt** from the original dataset, which represents 20 percent of the dataset.

The shape and acquisition of the data differ in some other experiments; however, the main concept is always the same, an input tensor with some configuration representing (q,p) and the same with its target *dxdt*.

The original testing depends on two modes of execution, the baseline model and the Hamiltonian one, as shown in 3.2. However, the base model for training is a normal Multi-Layer Perceptron (MLP for short) neural network. The specifications and configuration of this model are as follows:

- input nodes with a number specific to the shape of coordinates in the **x** dataset (2 or more)
- 2 hidden layers with default number of nodes equal to 200
- 2 output nodes representing $(\frac{\partial H}{\partial p}, -\frac{\partial H}{\partial q})$
- **tanh** activation functions

Now, depending on the mode the program is running in, baseline or Hamiltonian, the training happens in the following two ways:

Algorithm 1 Baseline Training

```
Input: x (input data tensor),
Output: dxdt-hat (predicted output tensor representing some shape of (dqdt,dpdt))
for $step = 0$ to $len(args.steps)$ do
    $dxdtHat \leftarrow baselineModel(MLP).timeDerivative()$
    $loss \leftarrow L2loss(dxdtHat, dxdt)$ backpropagation algorithm over the loss optimize the gradients zero out the gradients repeat above steps for the test section of the dataset mentioned above
end forRun the newly updated model one last time over all of the dataset x and test-x
```

This is the main flow of a training epoch using the baseline model. It is noteworthy to mention that the loss function is a regular L2 loss function without any regularization introduced, meaning that it serves as a standard ”mean squared error” cost function in the following form:

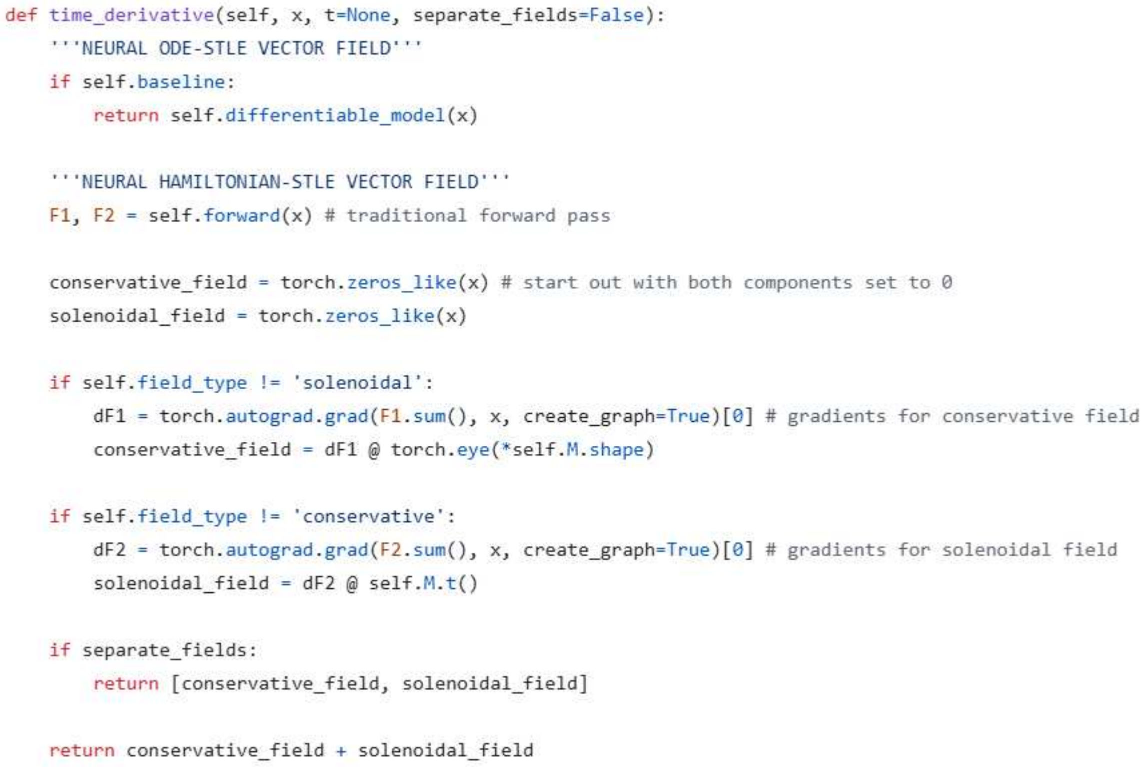
$$L2loss = ||y - f_\theta(x)||^2 \tag{3.2}$$

where y represents the target dxdt in the data set and $f_\theta(x)$ the predicted value of the model.

3.2.1 The Hamiltonian Model Flow

We subvert our attention to the specifics of the HNN and how it differs in the training of the model. The flow is the same as the algorithm 1, however, the only distinction is with what model did the program launch with in the preparation phase mentioned at the very beginning of this section. Going back to the main idea of PINNs, constructing one only requires a few modifications in the hyperparameters of the model or, in our case, tweaking the loss function so that it operates with some regularization on its cost functions.

In case of HNN,this ”regularization” is enforced before calculating the loss function and after doing the forward pass of the MLP model described above. In 3.3, the logic starts with a regular call to the forward method of the MLP model, as with the baseline approach. After attaining the two output vectors representing the change in position q and momentum p through the evolution of time, dqdt and dpdt, the integration of one of Hamilton’s properties, called the solenoidal field, begins.



```
def time_derivative(self, x, t=None, separate_fields=False):
    '''NEURAL ODE-STLE VECTOR FIELD'''
    if self.baseline:
        return self.differentiable_model(x)

    '''NEURAL HAMILTONIAN-STLE VECTOR FIELD'''
    F1, F2 = self.forward(x) # traditional forward pass

    conservative_field = torch.zeros_like(x) # start out with both components set to 0
    solenoidal_field = torch.zeros_like(x)

    if self.field_type != 'solenoidal':
        dF1 = torch.autograd.grad(F1.sum(), x, create_graph=True)[0] # gradients for conservative field
        conservative_field = dF1 @ torch.eye(*self.M.shape)

    if self.field_type != 'conservative':
        dF2 = torch.autograd.grad(F2.sum(), x, create_graph=True)[0] # gradients for solenoidal field
        solenoidal_field = dF2 @ self.M.t()

    if separate_fields:
        return [conservative_field, solenoidal_field]

    return conservative_field + solenoidal_field
```

Figure 3.3: Backpropagation over the outputs of the NN

3.2.2 Solenoidal Field in Hamiltonian Mechanics

If we go back to figure 3.1, we can see the small gray arrows that form a circular motion around the energy level of the pendulum. These represent the vector field known as the solenoidal field in Hamiltonian mechanics, and it is one of two properties that a system can possess.

In Hamiltonian mechanics, a solenoidal Hamiltonian system is a special type of system where the phase-space flow is divergence-free, which is what the authors of the paper originally intended to enforce in HNN. This means that the volume in the phase space, described by canonical coordinates (q,p), remains unchanged as the system evolves.

To understand the difference between solenoidal Hamiltonian systems and conserved Hamiltonian systems, let us examine their key properties:

- **Conserved Hamiltonian Systems**:

These systems have a well-defined Hamiltonian function *H(q,p)* that is explicitly conserved over time:

$$\frac{dH}{dt} = \frac{\partial H}{\partial q}\dot{q} + \frac{\partial H}{\partial p}\dot{p} = 0 \tag{3.3}$$

- **Solenoidal Hamiltonian Systems**:

Instead of conservation of H through it's total energy, the main characteristic is that the phase-space volume element is preserved under time evolution. Mathematically, this is expressed using the Liouville theorem, which we will dive deeper into in the next chapter and introduce how we used some version of Liouville's theorem in order to approximate a better solution than the original paper of HNN:

$$\frac{\partial \dot{q}}{\partial q} + \frac{\partial \dot{p}}{\partial p} = 0 \tag{3.4}$$

This condition implies that the Hamiltonian flow is incompressible, which means that trajectories generated by the dynamics of the system under study never shrink or expand in phase space.

Although energy may still be conserved, the defining feature is that the divergence of the vector field $(\dot{q},\dot{p})$ vanishes, keeping the motion of the system in equilibrium.

Key Difference:

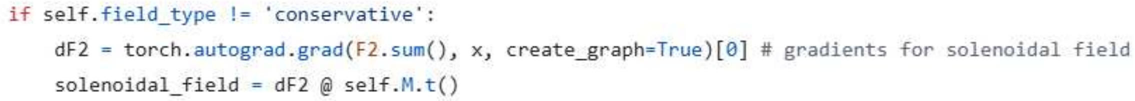
- **Conserved Hamiltonian systems** explicitly maintain energy conservation through *H(q,p)*, a well-defined, usually analytically solvable equation that can be integrated since we are dealing with calculating the total derivative of the equation itself.
- **Solenoidal Hamiltonian systems** ensure the incompressibility of phase-space flow, meaning that the trajectories preserve their volume under evolution without the need for a complete definition of the system.

Both properties can coexist, but solenoidal systems emphasize the structure of phase space rather than explicit conservation of the Hamiltonian itself.

3.2.3 Levi-Civita Permutation Tensor

Now that we have established the physical phenomena employed in the HNN learning paradigm, which is the adoption of the solenoidal field and the guarantee of free divergence in the progression of the system, we continue with the flow of the logic of the HNN.

Since we are interested in containing the dynamics of the momentum of the system, represented by $\frac{\partial H}{\partial p}$, the algorithm takes the value of the output vector representing $\frac{\partial H}{\partial p}$, represented in vector *F2* in the code section, and then calculates the gradient of the sum of the scalars in this vector with respect to the original input to the network *x*, which again represents some formation of the scalar values *(q,p)*.



```
if self.field_type != 'conservative':
    dF2 = torch.autograd.grad(F2.sum(), x, create_graph=True)[0] # gradients for solenoidal field
    solenoidal_field = dF2 @ self.M.t()
```

Figure 3.4: gradient section of solenoidal field

at first sight, the above line of code seems very straightforward. Calculate the gradients using the backpropagation algorithm and then after that in the main piece of logic, optimize the network parameters such as the hidden weights accordingly. However, this would be the case if there were no return value at the end of the current function. Instead, after calculating the gradients, they are returned and stored in a variable called *dF2*, as shown in 3.4. This is what is known as ”*gradient optimization*”, where the gradient result of the auto differentiation method becomes one of the inputs to the defined loss function.

Going back to the loss equation defined above and after integrating the above logic as part of the cost function, the new loss function becomes something like the following.

$$L2loss = ||y - \frac{\partial f_\theta(x)}{\partial x}||^2 \tag{3.5}$$

and in case of HNN, the loss function becomes something like this

$$L_{HNN} = ||\frac{\partial H_\theta}{\partial p} - \frac{\partial q}{\partial t}||_2 + ||\frac{\partial H_\theta}{\partial q} + \frac{\partial p}{\partial t}||_2 \tag{3.6}$$

which states that now, the theoretical function, or the hidden solution in this case is now being approximated by differentiating one of its outputs, in our case **F2**, according to all input parameters (p) and (q).

To be exact, research on optimizing the gradient behavior of neural networks remains relatively scarce. Schmidt and Lipson[16] employed a loss function with a similar structure, although their focus was not on the optimization of neural networks. Meanwhile, Wang et al.[18] explored gradient optimization within neural networks, but their work did not aim at Hamiltonian learning. Despite this gap in the literature, the findings from the HNN paper demonstrate that this method is not only feasible, but also delivers promising results.

The second line of 3.4 is where the last gradient optimization takes place before returning the final result to the loss function in the main training loop. It is the multiplication of "*dF2*", the variable containing the gradients of *F2* with respect to the input *x*, with the permutation tensor "**M**", which represents a Levi-Civita permutation tensor. This step mitigates the process of flipping the final results of *dF2* into this final tuple.

$$(\frac{\partial H}{\partial p}, -\frac{\partial H}{\partial q}) \tag{3.7}$$

Why Use a Permutation Tensor

Going back to the section of the solenoidal field, we mentioned that in order for a vector field to be solenoidal, the velocity of the system must be incompressible or divergent-free.

The Levi-Civita tensor is a fundamental object in tensor calculus, used extensively in differential geometry, classical mechanics, and quantum field theory. It plays a crucial role in defining antisymmetric operations, such as the cross product and determinants, and is essential for expressing conservation laws in physics.

The Levi-Civita tensor, denoted as ϵ_{ijk}, is a totally antisymmetric tensor that encodes the orientation of a coordinate system. In three dimensions, it is defined as:

$$\epsilon_{ijk} = \begin{cases} +1, & \text{if (i,j,k) is an even permutation of (1,2,3)} \\ -1, & \text{if (i,j,k) is an odd permutation of (1,2,3)} \\ 0, & \text{if any indices are repeated} \end{cases} \tag{3.8}$$

This definition ensures that swapping two indices reverses the sign of the tensor, making it a pseudo-tensor under coordinate transformations, which is perfect in the case of a Hamiltonian system as it ensures that the signs are always flipped between $(\frac{\partial H}{\partial p}, -\frac{\partial H}{\partial q})$.One of the most common uses of the Levi-Civita tensor is in expressing the cross product in three-dimensional space:

$$(A \times B)_i = \sum_{j,k} \epsilon_{ijk} A_j B_k \tag{3.9}$$

or, which is more adaptable in the HNN case is particularly useful in electromagnetism, where the curl of a vector field is given by:

$$(\nabla \times F)_i = \sum_{j,k} \epsilon_{ijk} \frac{\partial F_k}{\partial x_j} \tag{3.10}$$

Since the divergence of a curl is always zero, the Levi-Civita tensor naturally enforces solenoidal conditions in fluid dynamics and Hamiltonian mechanics.[4][20] And so if we redefine the curl of any vector field in the form of:

$$B = \nabla \times A \tag{3.11}$$

we get a divergence-free vector field since applying the divergence on a curl (any transformative curl including the Levi-Civita tensor) is nullified:

$$\nabla \cdot (\nabla \times A) = 0 \tag{3.12}$$

So, the approach of using the permutation tensor M in the code indirectly enforces incompressibility (solenoidal property) by leveraging the properties of the permutation tensor. Transformation using M^T ensures that the final computed field has no divergence, even though the raw gradient *dF2* may have contained divergent components.

3.3 Datasets and experiment results of HNN

Before we dive into our approach, we present the data used in the original HNN paper. We will go through how the datasets are created for the different experiments used and what type of integration libraries were implemented in order to integrate the training model into the aforementioned vector field above and the shape of the input data that is used in every experiment.

The HNN is tested over five main experiments in which classical physics phenomena are simulated. In each experiment, the data sets are generated through the construction of some validated well-known equations that can approximate the dynamics of the system under study. Initially, the data creation happens with either applying some integrable equation that can simulate some dynamics of the system like it's trajectory in a vector field. Then, to solve the *initial value problem* in order to obtain the trajectories of those systems or, in other words, their dynamics, the code uses a *scipy* python library function called **scipy.integrate.solve_ivp**. This will be able to integrate over the initial values (supposedly p and q or some combination of them) and then find a solution based on the following integration formula:

$$(q_1,p_1) = (q_0,p_0) + \int_{t_0}^{t_1} S(p,q)\, dt \tag{3.13}$$

where ***S*** denote the time derivatives of the coordinates of the system. The numerical algorithm used in this library by default is the fourth-order Runge-Kutta algorithm.

Ideal Mass-Spring

Starting with one of the simplest experiments, the first is to simulate the behavior of a frictionless mass-spring system. The Hamiltonian of the system, expressed in the following equation:

$$H = \frac{1}{2}kq^2 + \frac{p^2}{2m} \tag{3.14}$$

which, in addition to the main components p and q, also depends on the spring constant and the mass constant. For simplicity, both were set to the value of one. Then they generate initial coordinates to pass them to the "*initial value problem solver*" that we mentioned above with total energies uniformly distributed within the range [0, 21]. We remind the reader to remember that we are passing the "**ivp**" solver the energies computed by a certain dynamics equation, which will then integrate it using the fourth-order Runge-Kutta numerical approximator. For the final data set, they curated training and test sets, each containing 25 trajectories, and introduced Gaussian noise with a standard deviation of $\sigma^2 = 0.1$ to all data points. Each trajectory consisted of 30 observations, each of which represented a concatenation of (q,p).

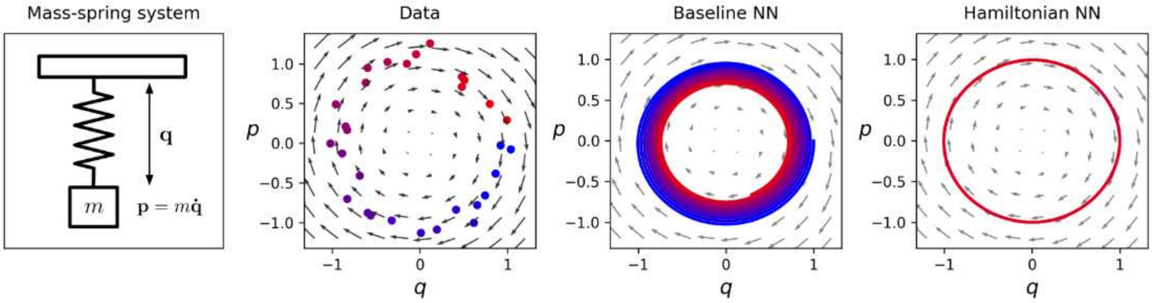


Figure 3.5: ideal mass spring energy representation: we can see how HNN integrates the predicted trajectory into the vector field in comparison with the baseline model which diverges along the field.

Real and Ideal Pendulum

The next two experiments are about the famous pendulum problem. However, the ideal pendulum experiment is the theoretical version with total conservation of energy (frictionless pendulum). while the second one represents the simulation of the pendulum by getting the data from a real experiment that was done. For the ideal pendulum, the following equation was adopted in order to simulate the dynamics of the system:

$$H = 2mgl(1 - \cos q) + \frac{l^2 p^2}{2m} \tag{3.15}$$

with the gravitational constant as g and the length of the pendulum as l.

As before, for simplicity, they set m = l = 1. This time, g was assigned a value of 3, and they selected initial coordinates with total energies ranging from [13, 23]. These values were chosen strategically to position the data set within the system's transition from linear to nonlinear dynamics. Following the approach in ideal mass spring, the authors created training and test sets, each consisting of 25 trajectories, and applied the same level of Gaussian noise.

For the real pendulum experiment, the authors used data from a Science paper by Schmidt & Lipson[16] , which also addressed the challenge of learning conservation laws from data. Compared to synthetic datasets, this one contained more noise and did not strictly adhere to any conservation laws due to the presence of slight friction in the real pendulum. Their objective in this experiment was to assess how the HNNs performed when faced with noisy and biased real-world data.

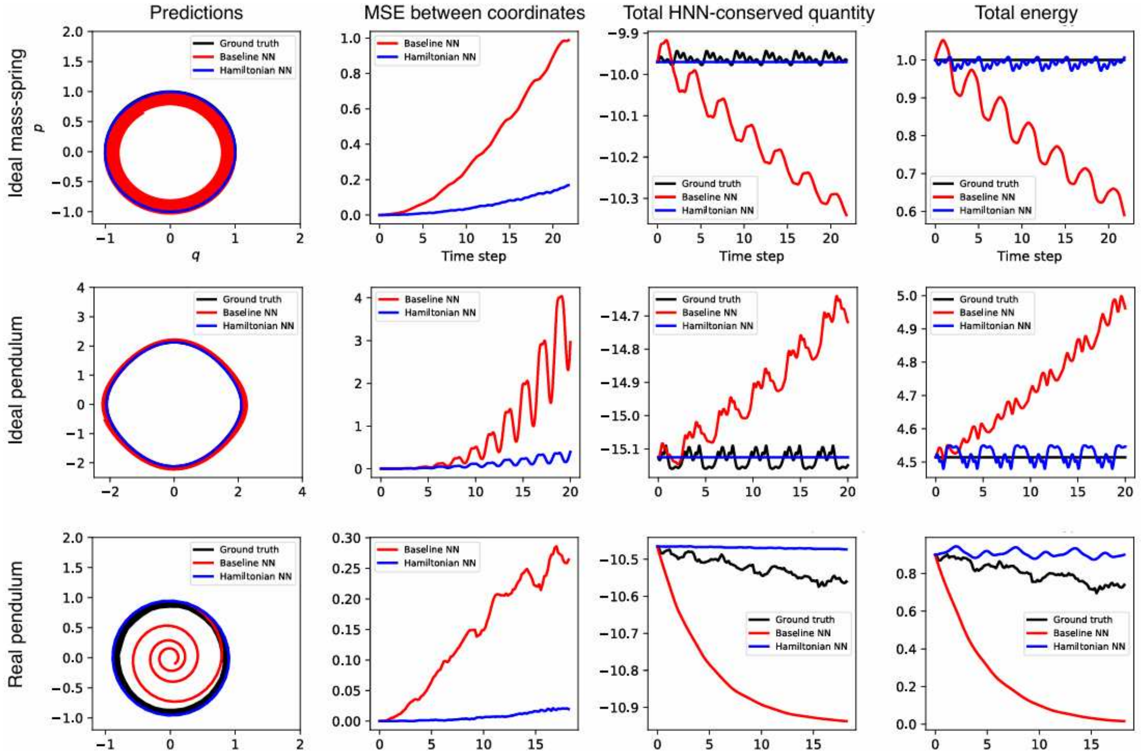


Figure 3.6: The authors analyzed models trained on three fundamental physics tasks. In the first column, they observed that the baseline model's dynamics gradually deviated from the ground truth. In contrast, the HNN maintained a high level of accuracy, even obscuring the black baseline in the first two plots. The second column showed that the baseline's coordinate MSE error diverged rapidly, whereas the HNN remained stable. In the third column, they plotted the quantity conserved by the HNN, which closely resembled the system's total energy. The fourth column confirmed this by depicting the total energy itself. As a result, the HNN approximately conserved total energy, unlike the baseline model.

3.3.1 Two and Three Body Problems

The two-body problem in physics refers to the motion of two objects interacting under mutual forces, such as gravitational or electrostatic attraction. This problem is analytically solvable because it can be reduced to a one-body problem by considering the center of mass of the system as fixed. The equations governing the motion of two bodies are derived from Newton's laws and can be expressed using Kepler's laws for celestial mechanics[2]. The solution involves determining the trajectories of objects, which typically follow elliptical, parabolic, or hyperbolic paths depending on their initial conditions and energy levels.

In contrast, the three-body problem is significantly more complex. It involves predicting the motion of three interacting bodies under mutual forces, such as gravitational attraction. Unlike the two-body problem, the three-body problem does not have a general analytical solution due to its chaotic nature. The dynamics of the system can exhibit highly unpredictable behavior, making numerical simulations essential to study its evolution[2]. Historically, the three-body problem was first investigated in the context of celestial mechanics, particularly in relation to the motion of the Earth, Moon, and Sun. The complexity arises because each body exerts a force on the others, leading to nonlinear interactions that cannot be simplified into a single equation.

One approach to solving the three-body problem is through perturbation theory, which approximates the motion of one body by considering small corrections to a known two-body solution. Another method involves numerical simulations, where computational techniques are used to integrate the equations of motion over time[1]. These simulations have revealed fascinating phenomena, such as quasi-periodic solutions, in which the system exhibits repeating patterns over long timescales[2]. Additionally, researchers have explored special cases where the three-body problem becomes integrable, such as when bodies move in a strong magnetic field, simplifying the equations governing their motion[1].

The three-body problem has practical applications in astrophysics, space exploration, and quantum mechanics. For instance, understanding the gravitational interactions between multiple celestial bodies helps astronomers predict planetary orbits and the stability of star systems. In quantum mechanics, the three-body Coulomb problem examines the interactions between charged particles, such as electrons and atomic nuclei, using integral equations derived from scattering properties[2]. These studies contribute to advances in fields like semiconductor physics, where trions, systems of two electrons and one hole, are analyzed using three-body problem techniques.

Despite its challenges, the three-body problem continues to be a subject of extensive research, with new mathematical and computational methods being developed to improve predictions and uncover hidden patterns in chaotic systems. The insights gained from studying this problem have broad implications, ranging from celestial mechanics to quantum physics, demonstrating the fundamental importance of multi-body interactions in nature.

Specification of the two and three-body problem

Starting with the more stable problem of the two-body system, the authors chose to go with the standard mathematical representation of the system using canonical q and p.

$$H = \frac{|P_{CM}|^2}{m_1 + m_2} + \frac{|p_1|^2 + |p_2^2|}{2\mu} + g\frac{m_1 m_2}{|q_1 - q_2|^2} \tag{3.16}$$

In the two-body problem, point particles interact through an attractive force, such as gravity. The authors once again define *g* as the gravitational constant and *m* as mass. The equation above presents the Hamiltonian of the system, where the reduced mass and P_{CM} denote the momentum of the center of mass. To simplify the calculations, they set $m_1 = m_2 = g = 1$. Furthermore, their experiments were limited to systems in which the center of mass had zero momentum. Despite this constraint, the system remained an intriguing challenge due to its eight degrees of freedom, determined by the coordinates $xandy$ and the momentum components of both bodies.

According to the data set collection and training procedure, the following is applied.

- The initial data set consists of 1000 near-circular two-body trajectories.
- the above trajectories were properly initialized with center of mass zero, total momentum zero, and radius $||r = q2 - q1||$ in the range $[0.5$,$1.5]$
- in order to get dynamics, some sort of velocity initialization was needed and so, the authors iterated through some values until they found the perfect match.
- As with the aforementioned experiments so far, the train/test split was an 80/20 split, with 200 trajectories used to be integrated using the fourth-order Runge-Kutta integration.
- The input of the training model now is a tensor with a shape of $[200$,$8]$ as well as the output nodes.

- Instead of 2000 training gradient steps, in two- and three-body systems they used 10000 steps.

The reason why we are going with the details of the hyper-parameters of these experiments and their input shapes and other details, which might be minute specifications in other Neural Network Architectures, is because these are the extra parameters that create the concept of a PINN which, after experimenting for a while as well found out that these variables heavily affect the final outcome of the system.

Chapter 4

Liouville's Theorem and Its Integration into HNN

After going through some of the experiments presented in the HNN paper, we introduce an additional integration into the network that improved the final results of the energy levels in all of the experiments, as well as having the ability to predict much better dynamics for the trajectories that almost fits the ground truth trajectory for any experiment.

Despite the achievements of the HNN model, the dynamics was still diverging from the absolute true path as the number of observations increased. This mishap is heavily noticed in the two-body and three-body problems. After visualizing this behavior, it is clear that there needs to be some other mechanism that can enforce the conservation of energy or the flow of the system itself in a contained environment. This is where Liouville's theorem in Hamiltonian mechanics may be of some help.

4.1 What is Liouville's Theorem

Liouville's theorem is a fundamental result in Hamiltonian mechanics that describes the conservation of the volume of phase space in a dynamical system. The distribution function of phase space remains constant along the trajectories of the system, which means that the density of states in phase space is preserved over time[9]. This theorem plays a crucial role in statistical mechanics and classical mechanics, forming the basis for many physical interpretations of dynamical systems.

Mathematical Formulation in Hamiltonian Context

We consider the same Hamiltonian system with generalized coordinates q_i and conjugate momenta p_i. The evolution of the system governed by Hamilton's equations over time is as follows:

$$\frac{dq_i}{dt} = \frac{\partial H}{\partial p_i}, \frac{dp_i}{dt} = -\frac{\partial H}{\partial q_i} \tag{4.1}$$

where $H(q,p)$ is the Hamiltonian function that represents the total energy of the system[17].

The theorem states that the phase space density $\rho(q,p,t)$ satisfies the following continuity equation:

$$\frac{d\rho}{dt} = \frac{\partial \rho}{\partial t} + \sum_i \left(\frac{\partial \rho}{\partial q_i}\frac{dq_i}{dt} + \frac{\partial \rho}{\partial p_i}\frac{dp_i}{dt}\right) = 0 \tag{4.2}$$

This equation implies that the volume of the phase space occupied by a set of trajectories remains constant over time[9]. The theorem is closely related to the incompressibility of phase-space flow, meaning that the density of states does not change as the system evolves.The above approach is crucial in statistical mechanics, where it ensures that the probability distribution of micro states remains invariant[9]. This conservation principle underlies the ergodic hypothesis and the foundations of equilibrium statistical mechanics.

In Hamiltonian mechanics, Liouville's theorem implies that the flow of trajectories in phase space behaves like an incompressible fluid[8]. This means that no region of the phase space becomes denser or more sparse with time, preserving the structure of the evolution of the system.

Applications in Physics

- **Statistical Mechanics**: The Liouville theorem is used to justify the use of ensemble averages in statistical mechanics[8].

- **Quantum Mechanics**: The theorem has an analog in quantum mechanics, where the Von Neumann equation describes the evolution of the density matrix[17].
- **Chaos Theory**: In chaotic systems, Liouville's theorem ensures that the phase-space distribution remains invariant despite the sensitive dependence on initial conditions[8].

4.2 Integration of Liouville Theorem into HNN

In the section three we introduced the idea of divergence-free physical systems and their conserved phase spaces in a space-time continuum. We also discussed the fact that the authors of the original HNN paper enforced divergence through the use of a permutation tensor with a mathematical *curl* that is divergent-free. That tensor was the Levi-Civita permutation tensor. However, after Sam et al. introduced their methods and compared them with other academic approaches, it was mentioned in their article that their approach was totally reversible over time, meaning that any mapping of the following can be *bijective/invertible*[5]

$$(q_0,p_0) \rightarrow (q_1,p_1) \tag{4.3}$$

which is a direct result of the Liouville theorem and is *the density of particles in phase space is constant* proposal.

We begin by going through the changes and the upgrades on the original code, the approach taken to integrate an approximation of Liouville's abstract theorem and the parameters and algorithms that were kept. We will also shed an extra layer of attention to the changes done in the two-body and three-body problems that are more complicated, at least from a mathematical and physical standpoint.

The architecture of the neural network model remained the same as with the original code having:

- two input neurons representing (q,p) or 8 and 12 in the case of the two-body and three-body problems.

- 2 hidden layers with 200 neurons each.
- An output layer having the same number of input neurons, or always 2 in case the network is not being trained in the Hamiltonian context (baseline).
- tanh activation function.

It is inside the Hamiltonian derivation section that we implement the changes. As the model starts its training epoch and while it is in Hamiltonian mode, the following algorithm gets executed:

Algorithm 2 Modified HNN Training

```
Input: $x^2$ (input data tensor squared)
$F1$,$F2$ ← output of first forward run over $x^2$
updated x_p$[BATCH\_SIZE,0]$ ← $x[:,0] * -1 * F2$
updated x_q$[BATCH\_SIZE,1]$ ← $x[:,1] * F1$
updated x ← concatenate updated x_p and updated x_q
updated $F1$,updated $F2$ ← forward($x * updated\_x$)
H ← $(F1 + updatedF1 * 0.1) + (F2 + updatedF2 * 0.1)$
Calculate the gradient of the output H above with respect to the original input x
```

In Algorithm 2, the input, again representing a tuple of the form (q,p), is squared in an attempt to force the neural network to enforce in its learning paradigm the correlation between the canonical coordinates and *q*,*p* and the measure of the Hamiltonian energy formula used:

$$H = q^2 + p^2 \tag{4.4}$$

The above equation is used in all the analysis notebooks written by the authors of HNN. This change can be seen as a Physics-Informed learning paradigm. This is due to the fact that when we reach the step of optimizing the gradient of the output,or in our case the optimization of

$$H = (F1 + updatedF1 * 0.1) + (F2 + updatedF2 * 0.1) \tag{4.5}$$

we are optimizing the gradient according to whatever input was passed into the first forward pass, and so by passing x^2 as the original input, the gradient is also learning to adjust accordingly.

As with the original work on HNN, we extract two output nodes from the forward pass.However, The intention with these variables will change from one step of the algorithm to the other.Before, they were supposed to represent:

$$\frac{\partial H}{\partial q}, \frac{\partial H}{\partial p} \tag{4.6}$$

and then, after optimizing the gradient, the permutation tensor flips the outputs to be in the final form of:

$$\frac{\partial H}{\partial p}, -\frac{\partial H}{\partial q} \tag{4.7}$$

Since the authors were trying to satisfy one form of divergent-free Hamiltonian systems, they followed the above implementation to satisfy their intention. Our intention is to satisfy, or to be more accurate approximate, the behavior of Liouville's theorem in the Hamiltonian space-time continuum. Formally, the Liouville theorem, in its abstract form, presents:

$$\frac{d\rho}{dt} = \frac{\partial \rho}{\partial t} + \sum_i \left(\frac{\partial \rho}{\partial q_i}\frac{dq_i}{dt} + \frac{\partial \rho}{\partial p_i}\frac{dp_i}{dt}\right) = 0 \tag{4.8}$$

As we explained earlier, the above equation introduces the concept of the density of the flow of a Hamiltonian system in the phase space. In that case, the value of ρ will be some probability distribution, probably in the Gaussian probability space. Alas, we were not able to get any results in integrating a generic form of some probability function like the Normal or Uniform distribution functions as this step seems like it requires more statistical analysis while targeting the Hamiltonian system under study (mass spring, pendulum, etc.). Fortunately, we approached the theorem from a different perspective thanks to the following academic paper that explains the three-body problem[9].

The theorem starts with the idea that the phase flow of a physical system preserves the volume containing it. In the proof of the previous statement, the following set of equations is realized.

for any region *D* we have the volume of $g^t D =$ the volume of D. The following definitions of the system are given:

- Volume V: a region of dimensions $2n$ of the phase space.
- Area A: a region of $2n - 1$ dimensionality.
- Surface S: an area that limits a volume.

We can express the change in the volume of the region D as

$$\frac{dV}{dt} = \int_S n \cdot v \, dA \tag{4.9}$$

Here, n represents a vector normal to the surface S, v is the velocity of the flow in phase space, and dA is an infinitesimal area on the surface. Next, the proof continues by applying the divergence theorem to the above integral, which integrates on the 3-dimensional level, in this case the volume V.

$$\frac{dV}{dt} = \int_V \nabla \cdot v \, dV \tag{4.10}$$

Take note of n, which was the vector normal to the surface under study, and in our case the vector representing some configuration of (q,p), has been transformed to become the gradient of this vector itself, which is exactly what we are after in order to optimize this structure ∇.

Examining the velocity v more closely, we note that

$$v = (\dot{q}, \dot{p}) = (\frac{\partial H}{\partial p}, -\frac{\partial H}{\partial q}) \tag{4.11}$$

Taking the divergence of v, or in better words, so that what we did makes the divergence of $(\dot{q}, \dot{p})$ more sense,

$$\begin{aligned} \nabla \cdot v &= \frac{\partial \dot{q}}{\partial q} + \frac{\partial \dot{p}}{\partial p} \\ &= \frac{\partial}{\partial q}\frac{\partial H}{\partial p} - \frac{\partial}{\partial p}\frac{\partial H}{\partial q} \\ &= 0 \end{aligned}$$

The order of the partial derivatives does not matter. Then the following equation holds:

$$\frac{dV}{dt} = \int_V \nabla \cdot v \, dV = 0 \tag{4.12}$$

[9]

It is now apparent that the core of algorithm 2 depends on the above equations, so the logic follows to make the model alternate in assuming what $F1$ and $F2$ dislike in each step of the divergence. Since we need to represent

$$\frac{\partial \dot{q}}{\partial q}, \frac{\partial \dot{p}}{\partial p} \tag{4.13}$$

and

$$\frac{\partial}{\partial q}\frac{\partial H}{\partial p} - \frac{\partial}{\partial p}\frac{\partial H}{\partial q} \tag{4.14}$$

we re-multiply the input x with the newly acquired $F1$,$F2$ variables, representing the total velocities in this case as with the above proof. Next we pass the new updated input into the forward pass again to get the final output of the system. The reason why we used double forward pass was to simulate the behavior of the second-order derivation of

$$\frac{\partial}{\partial q}\frac{\partial H}{\partial p} - \frac{\partial}{\partial p}\frac{\partial H}{\partial q} \tag{4.15}$$

which translates to

$$\frac{\partial^2 H}{\partial q \partial p} - \frac{\partial^2 H}{\partial p \partial q} \tag{4.16}$$

and then finally, before calculating the gradients of the outputs we have attained so far, we re–iterate the fact that the outputs serve in the total energy of the system represented by:

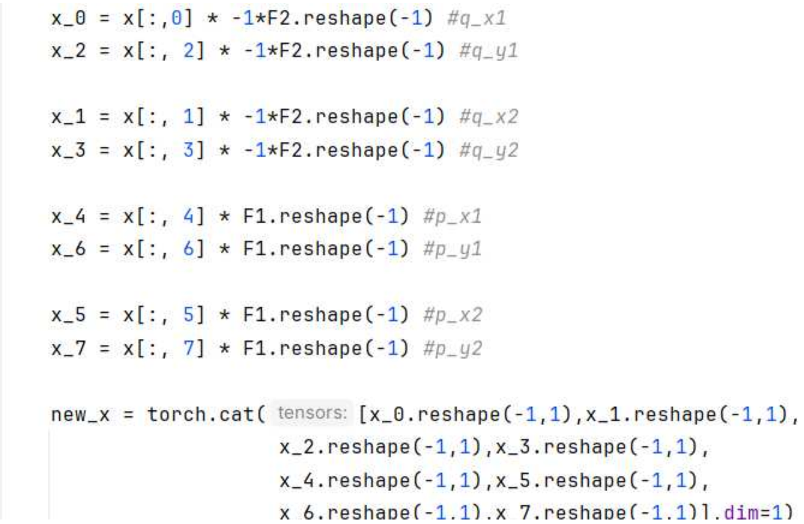
$$H = q^2 + p^2 \tag{4.17}$$

To be more exact, the final output was weighted down to include a combination of the original output variables $F1$ and $F2$ and the updated final output variables after applying the Liouville theorem. In the code, we found that reducing the weight of the updated variables $F1$ and $F2$ by 0.1 and adding them to the original results (regular values of $F1$,$F2$ from the first forward pass) was the optimal result in all experiments.

More specifications for the two-body and three-body problems

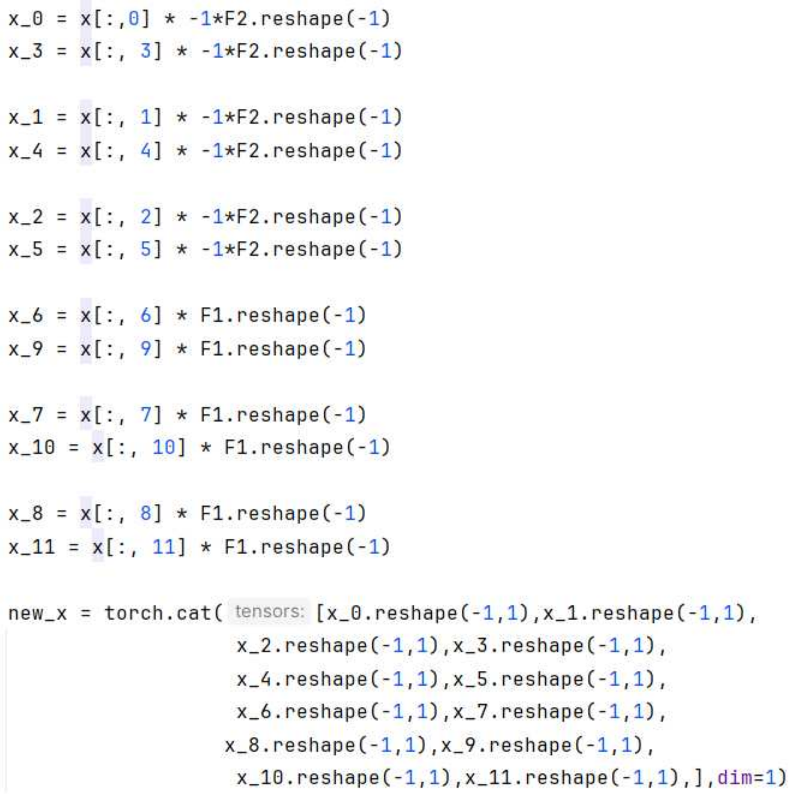
The interesting part about the two-body and three-body problems is that the Hamiltonian system involves two or more bodies in motion, each having their own set of q and p coordinates and they operate on two axis of coordinates (x,y). This, as we have established before, results in a different representation of the input tensor (q,p). and so we had to adjust the inner block of updating the outputs $F1$ and $F2$ from the first forward pass accordingly.

The changes were applied in the specific selection of the input tensor x and the multiplication of the output variables accordingly (whether it is a q_i or a p_i). The same logic was also applied for the 3-body problem.



```
x_0 = x[:,0] * -1*F2.reshape(-1) #q_x1
x_2 = x[:, 2] * -1*F2.reshape(-1) #q_y1

x_1 = x[:, 1] * -1*F2.reshape(-1) #q_x2
x_3 = x[:, 3] * -1*F2.reshape(-1) #q_y2

x_4 = x[:, 4] * F1.reshape(-1) #p_x1
x_6 = x[:, 6] * F1.reshape(-1) #p_y1

x_5 = x[:, 5] * F1.reshape(-1) #p_x2
x_7 = x[:, 7] * F1.reshape(-1) #p_y2

new_x = torch.cat( tensors: [x_0.reshape(-1,1),x_1.reshape(-1,1),
                   x_2.reshape(-1,1),x_3.reshape(-1,1),
                   x_4.reshape(-1,1),x_5.reshape(-1,1),
                   x_6.reshape(-1,1),x_7.reshape(-1,1)],dim=1)
```

Figure 4.1: 2-body HNN code section with Liouville theorem adjusted to it.



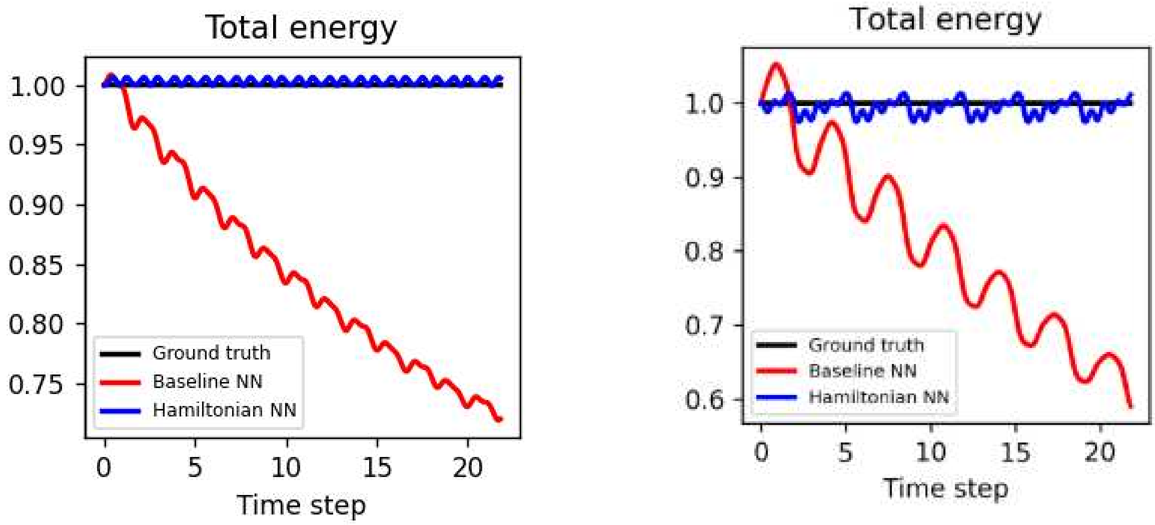
```
x_0 = x[:,0] * -1*F2.reshape(-1)
x_3 = x[:, 3] * -1*F2.reshape(-1)

x_1 = x[:, 1] * -1*F2.reshape(-1)
x_4 = x[:, 4] * -1*F2.reshape(-1)

x_2 = x[:, 2] * -1*F2.reshape(-1)
x_5 = x[:, 5] * -1*F2.reshape(-1)

x_6 = x[:, 6] * F1.reshape(-1)
x_9 = x[:, 9] * F1.reshape(-1)

x_7 = x[:, 7] * F1.reshape(-1)
x_10 = x[:, 10] * F1.reshape(-1)

x_8 = x[:, 8] * F1.reshape(-1)
x_11 = x[:, 11] * F1.reshape(-1)

new_x = torch.cat( tensors: [x_0.reshape(-1,1),x_1.reshape(-1,1),
                   x_2.reshape(-1,1),x_3.reshape(-1,1),
                   x_4.reshape(-1,1),x_5.reshape(-1,1),
                   x_6.reshape(-1,1),x_7.reshape(-1,1),
                  x_8.reshape(-1,1),x_9.reshape(-1,1),
                   x_10.reshape(-1,1),x_11.reshape(-1,1),],dim=1)
```

Figure 4.2: 3-body HNN code section with Liouville theorem adjusted to it.

4.3 Visual Results of some experiments

Now that we have formally elaborated on our work, we present the results of some experiments, while going through what their impact was and by how much the results were better than the original code of HNN.

4.3.1 Mass-spring



(a) total energy predicted in mass-spring with Liouville Addition

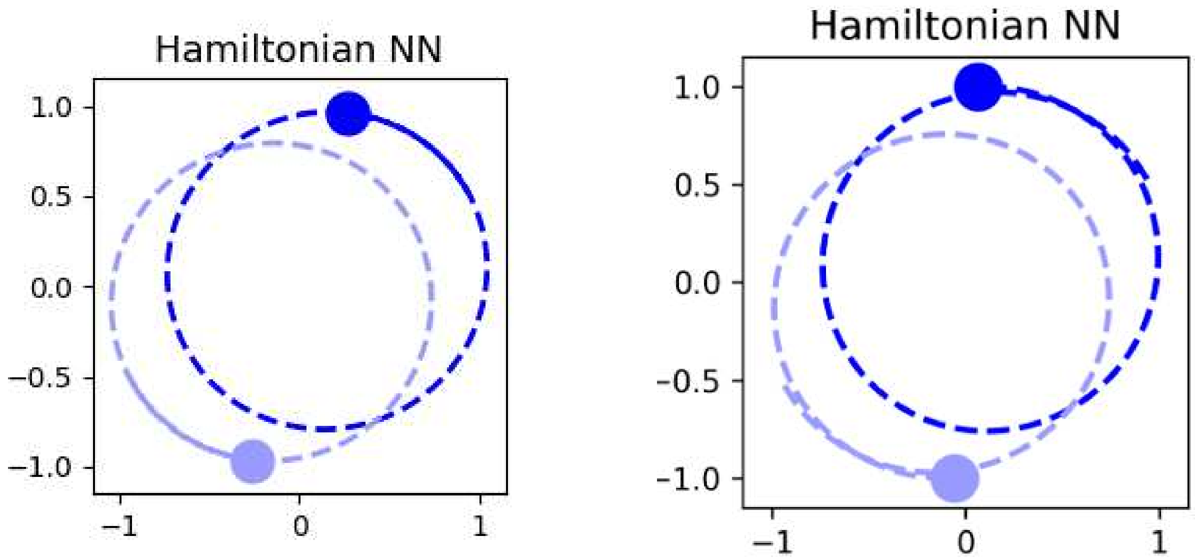
(b) total energy predicted in mass-spring with Liouville Addition

Figure 4.3: Difference between total energies predicted by both the original model and ours

Our approach showed a more promising result in the stability of the total energy predicted by the model on the problem of the mass-spring. as we can see in figure 4.3, the fluctuation, or the gap between the total energy predicted by the Hamiltonian network in 4.3a is much more stable and closer to the true ground energy. Also,as for the visual representation of the energy levels in the shape of an orbit in phase space, there was not much clear visual difference. However, we will definitely notice this change in the two-body and three-body problems. We also mention that almost the same result occurred in both the ideal and the real pendulum experiments, where the total energy graphs were almost identical to the mass spring as well.

4.3.2 The Two-body Problem

This is the part where the differences in results were most noticeable, especially in the two-body problem. Our optimization of HNN gave the best results in the two-body problem, covering all aspects like total, potential, and kinetic energies, being extremely stable and accurate.

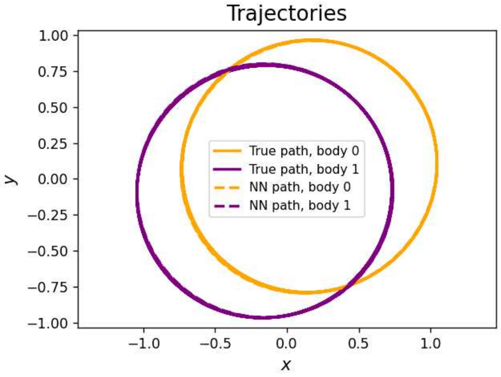


(a) visual representation of the total dynamics of the orbiting of two bodies in near-circular form by Liouville HNN

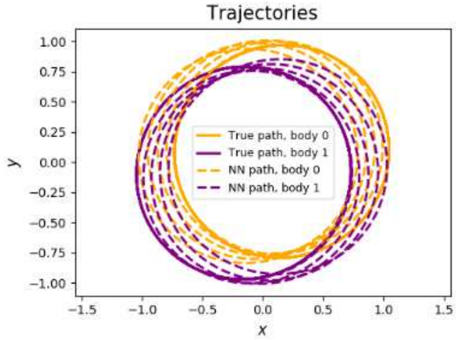
(b) visual representation of the total dynamics of the orbiting of two bodies in near-circular form by original HNN

Figure 4.4: Difference between predicted orbits of the two-body problem

In figure 4.4a, we can see how near perfect the orbit is, while in figure 4.4b the bodies start to deviate from the inner circle at the end. This shows that with the original code, the model was still diverging in its prediction of the dynamics of the system. The following is a more expressive representation of the paths that the predicted trajectories took during execution.



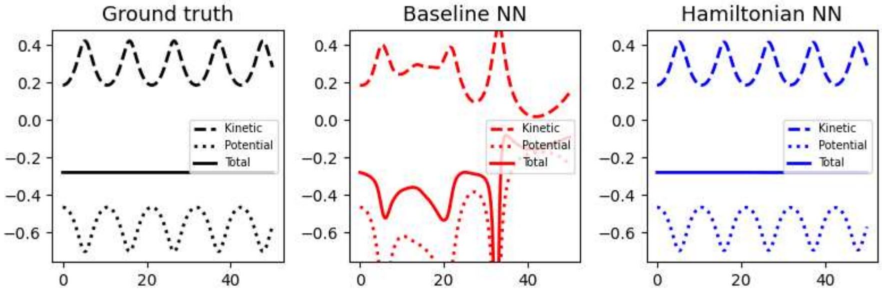
(a) visual representation of the trajectories taken by the predicted model



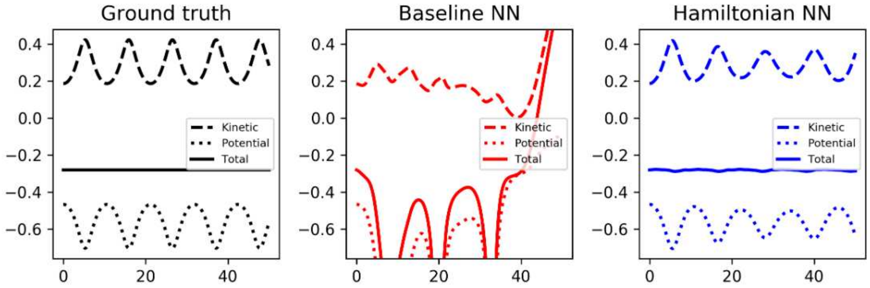
(b) visual representation of the total dynamics of the orbiting of two bodies in near-circular form by original HNN

Figure 4.5: Difference between predicted orbits of the two-body problem

It is now clear that our approach has greatly improved the prediction of the dynamics of a system. We now turn to a more important part of this entire research, that is, how well were the predicted energy levels compared to the ground truth. After all, we must not forget that our PINNs are not here to only simulate the trajectory or the motion of a certain system, as this can be done through one of the more traditional approaches of CNN (convolution neural networks) and a pictorial set of the dynamics of a real system in nature. The main benefit in this manner is the ability to perfectly predict the sequence of the energies that form the dynamics of the system under study. This is quite a tricky part as calculating the dynamics in real life usually needs some complex algebraic differentiation function, that is, if it's integral exists and is simple.



(a) Liouville energy levels



(b) original energy levels

Figure 4.6: the graph of Potential, Kinetic and total energy levels throughout the time of the experiment

What is important in the above figure are the graphs on the far right. In the original HNN, you can see in 4.6b how the potential and kinetic energies fluctuate. In fact, one might also observe that they are deteriorating and diminishing as time increases. This behavior leads to the presence of divergence in the system and weak conservation of energy, which, as a result, leads to weaker total energy in general. On the other hand, after optimizing the code with our version of Liouville, we can safely say that all three energy levels are satisfied. levels (potential, kinetic, and total) are stable and almost identical in both shape and continuity compared to the ground-truth energy levels.

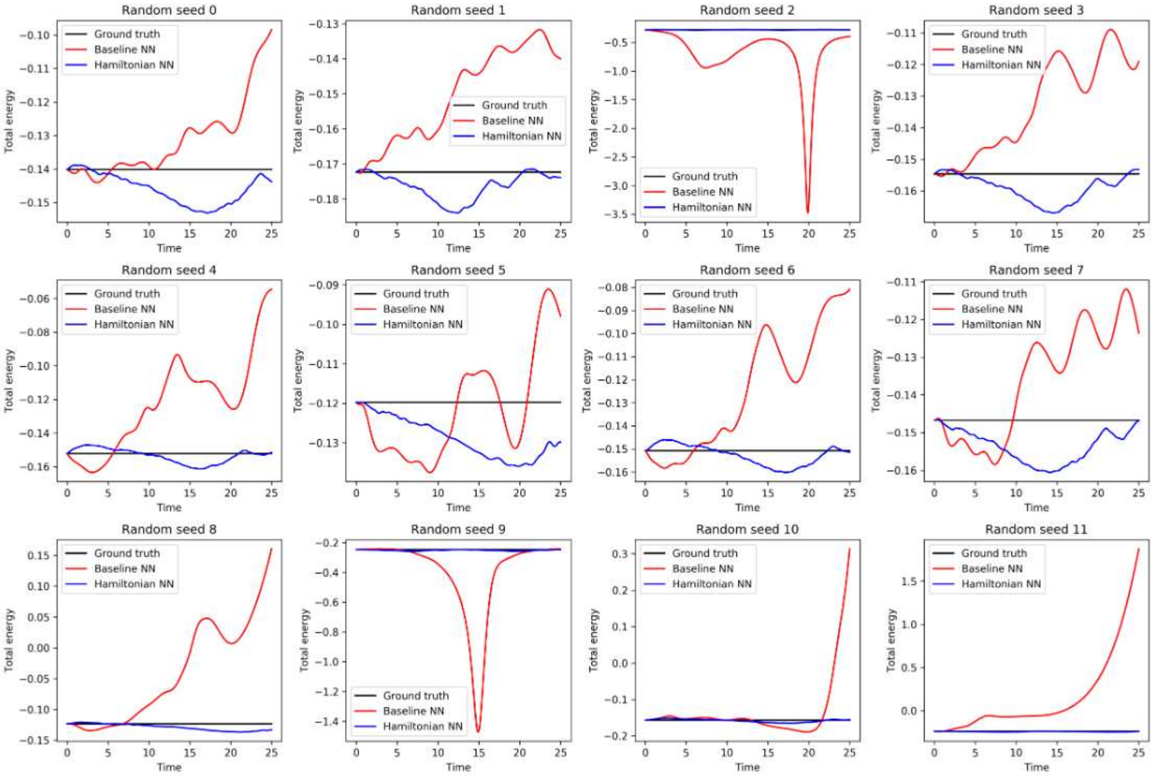


Figure 4.7: original HNN energy levels with different seeds

Figure 4.8: graphs of total energies with different seed of original HNN

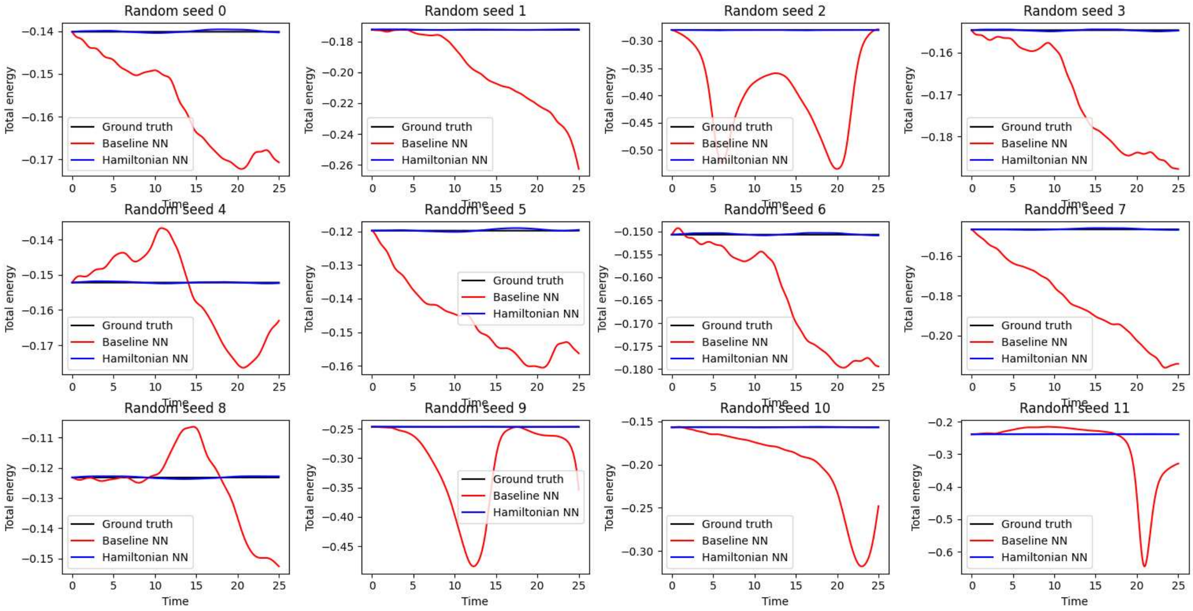
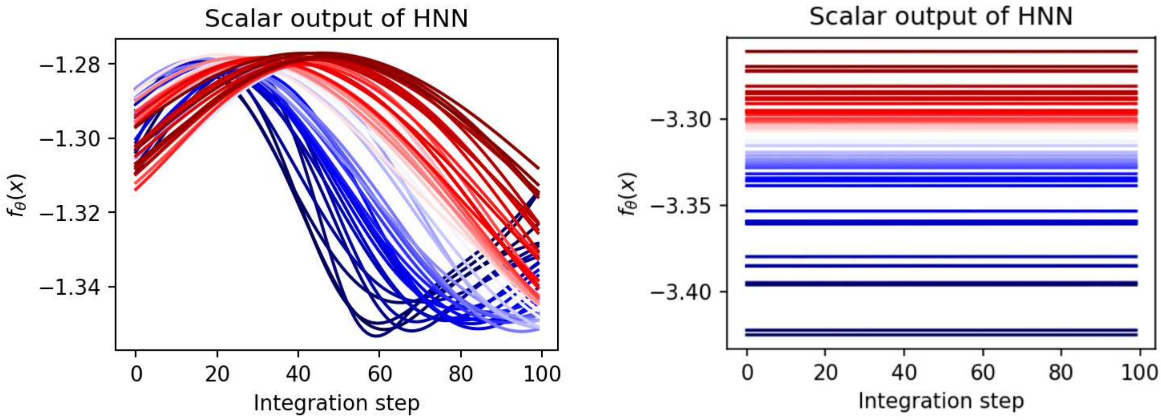


Figure 4.9: Liouville energy levels with different seeds

Figure 4.10: graphs of total energies with different seed of Liouville HNN

In fact, if we look at the total energies with multiple runs with different initialized random torch seeds, we can see that almost all of the HNN energy levels in Figure 4.9 have a much more stable and almost identical energy level to the ground truth than the runs with the original HNN code.



(a) integration steps of Liouville HNN

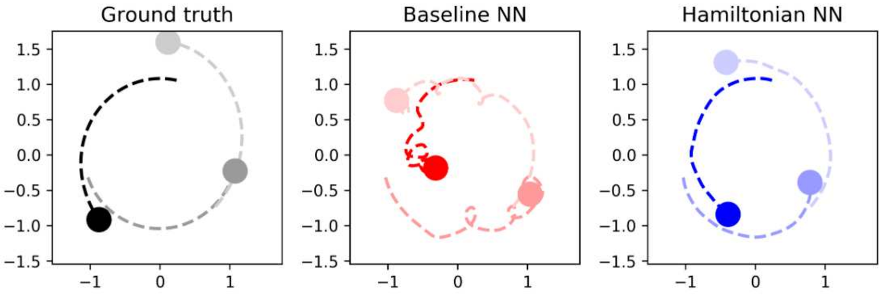
(b) integration steps of original HNN

Figure 4.11: integration steps of the trained model into the phase space

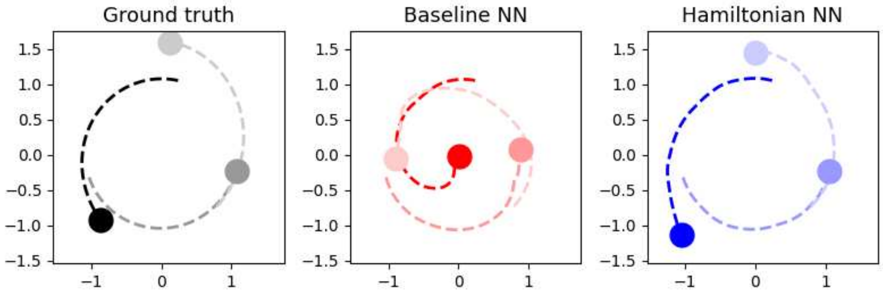
The last note is the difference in the integration steps. Figure 4.11a on the left shows a more complicated integration scheme, which means more run time to generate the dynamics of the system in the given phase space. However, the run-time did not exceed more than a few more minutes than the original code, which still makes the optimization made to the original code valid.

4.3.3 three-body problem

We now publish the results of probably the hardest problem to simulate out of all the other experiments due to its limited analytical solutions in the scientific community. Although the dynamics were not as near-perfect as the two-body problem or the rest of the experiments, the Liouville approach implemented into the HNN code still shows better energy levels than the original model. Not only were the energy levels closer to the ground truth, but a more realistic evolution of the potential and kinetic energies was predicted, leading to a more realistic transition of the energy. balanced three-body motion that correlates with each other.



(a) HNN trajectory of 3-body problem of original model

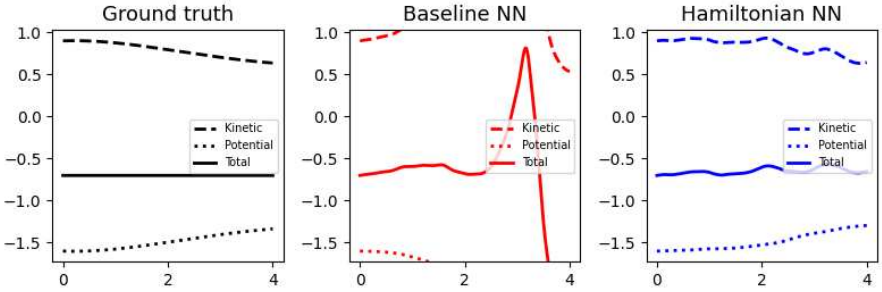


(b) HNN trajectory of 3-body problem with Liouville optimization

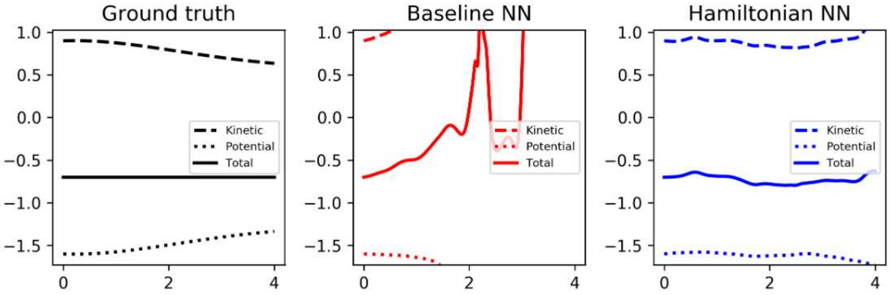
Figure 4.12: Comparison of the predicted trajectories of the 3-body problem

The orbits in the three-body problem still diverge even after our optimization. However, maintaining a quasi-stable motion between the 3 bodies on the same trajectories as the ground-truth trajectories. In fact, you can notice how the motion of the original model of HNN starts to fail upon itself and folds in rotation, indicating a total failure in the potential and kinetic energies while with our approach, the objects seem to follow a path more similar to the true paths of the system. This behavior can also be shown in figure 4.13 where one can notice how the potential and kinetic energies follow a similar dynamic (increasing and decreasing, respectively) while maintaining a balanced total energy flow in the graph of the model with the Liouville optimization, while it fails and diverges at the end in the original model of HNN leading to a collapsed system at the very end.

The above result is important for the fact that we are now able to better predict the dynamics of the system as a whole, especially for a complex system like the three-body problem, where not many analytical solutions can give optimal results. In other words, we are one step closer to making a universal numerical approximator for such problems. Although the energy levels



(a) Liouville energy levels



(b) original energy levels

Figure 4.13: the graph of Potential, Kinetic and total energy levels throughout the time of the experiment

still fluctuate and are not as stable as the two-body problem, the above and the following graphs show that optimization in the field of a PINN can always be improved. The graphs show a more stable model in terms of the predicted total energy levels with the use of different seeds for the system at different run-times.

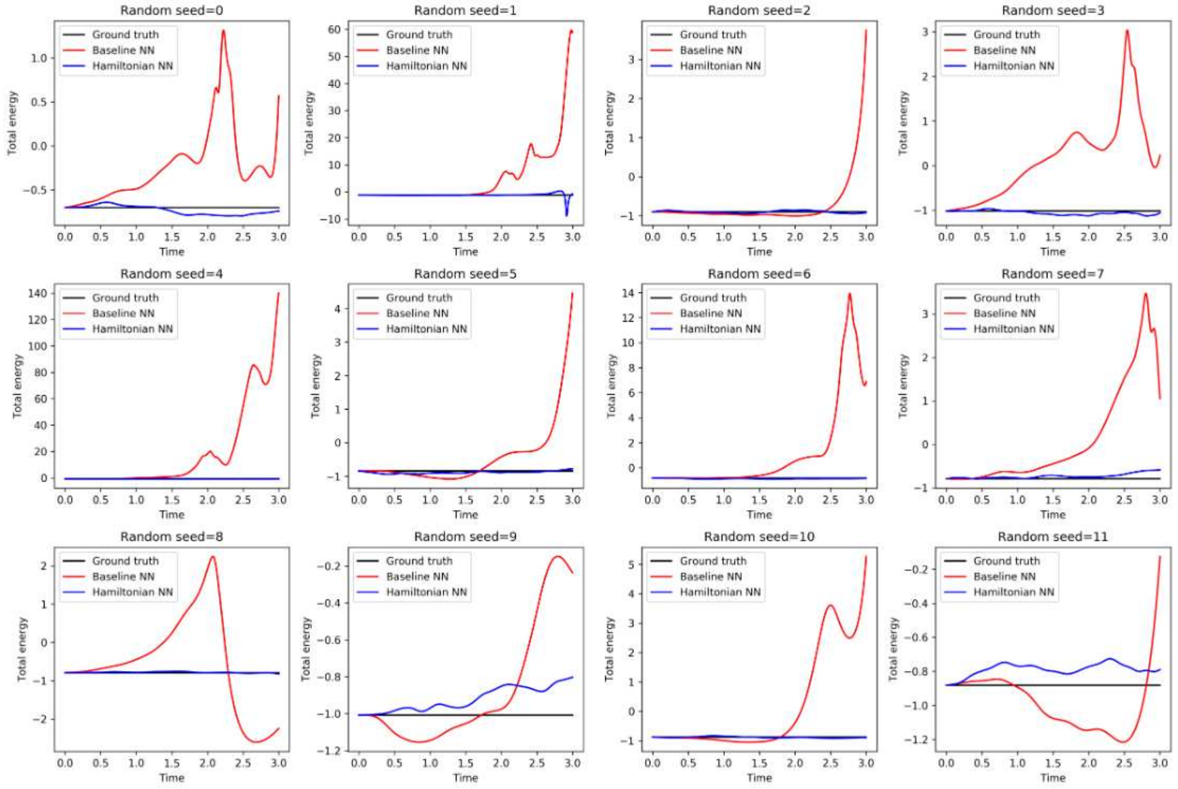


Figure 4.14: original HNN energy levels with different seeds

Figure 4.15: graphs of total energies with different seed of original HNN

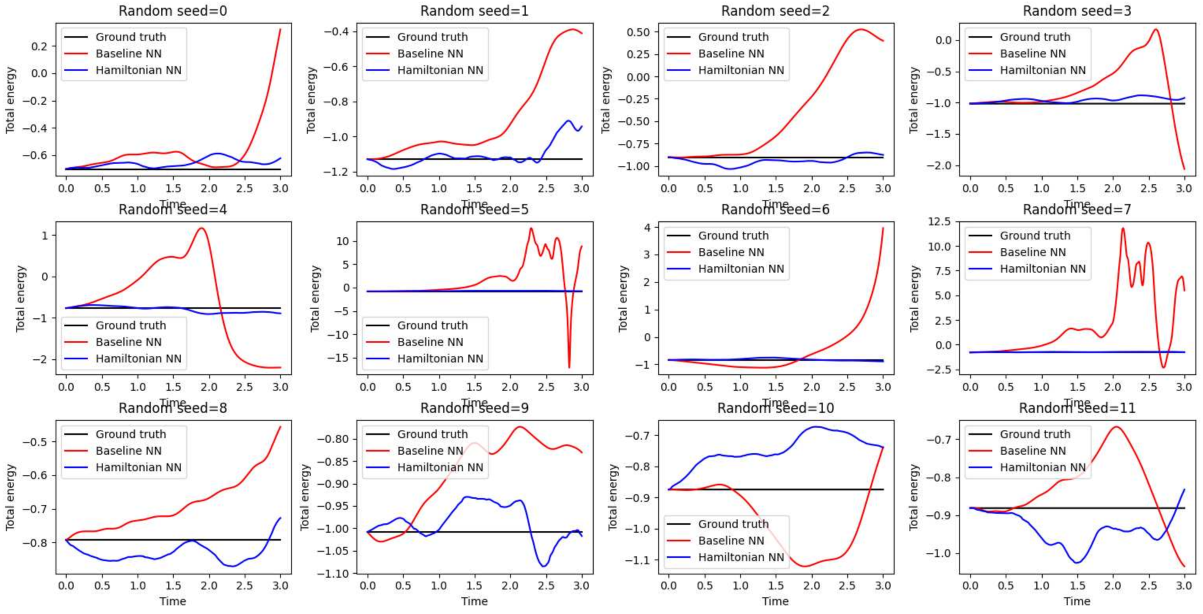
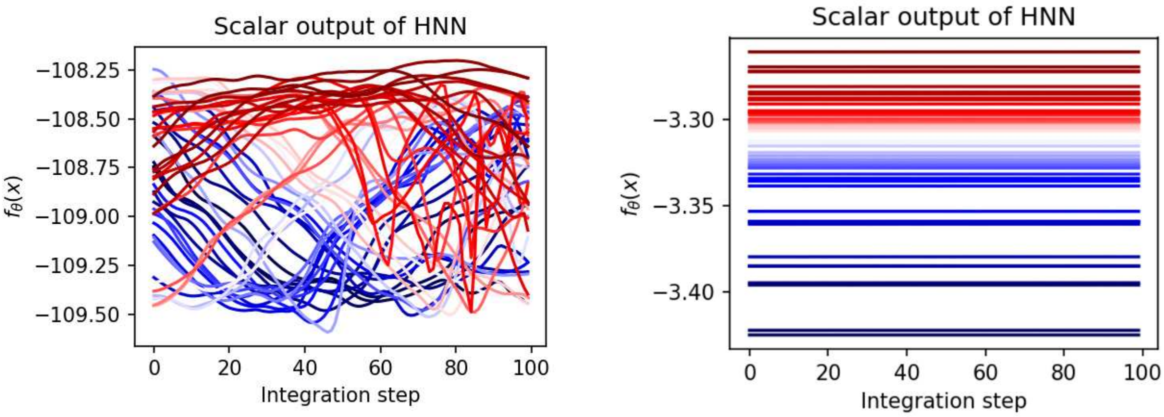


Figure 4.16: Liouville energy levels with different seeds

Figure 4.17: graphs of total energies with different seed of Liouville HNN

Before we close this section and consequently this chapter, we would like to mention a few notes. First, we remove the optimization of squaring the input x during training and before the first forward method, as it negatively affects the total behavior of the system. We could not figure out why this addition would only affect the three-body problem while improving other problems such as the two-body problem. Another thing would be the runtime overhead from testing the model as before, by integrating the dynamics of the system through into the phase space, only this time using the model itself to differentiate or in our case, predict the data. Generating the data with the new optimized model relatively suffered on the time parameter, and while this is one aspect that is looked upon in great regard in the computing community, the time overhead increase was a constant order of magnitude which, again as with the two-body problem, overhead is still a tolerable sacrifice.



(a) integration steps of Liouville HNN

(b) integration steps of original HNN

Figure 4.18: integration steps of the trained model into the phase space

We present the following tables of results that contain data from both original and optimized HNN:

	Train Loss		Test Loss		Energy	
Task	**Baseline**	**HNN**	**Baseline**	**HNN**	**Baseline**	**HNN**
1: Ideal mass-spring	38 +- 0.20	37 +- 2.0	38 +- 0.2	36 +- 1.8	113 +- 45	0.05 +-0.04
2: Ideal pendulum	33 +- 1.7	31 +- 1.7	35 +- 1.7	36 +- 1.7	251 +- 36	31 +- 5
3: Real pendulum	1.8 +- 0.1	9.3 +- 0.5	1.4 +- 2.0	5.9 +- 0.6	369 +- 74	10.8 +- 4.3
4: Two Body(10^6)	32 +- 0.4	0.15 +- 0.004	29 +- 0.4	0.13 +- 0.004	356 +- 150	**0.02 +- 0.003**
5: Three Body(10^2)	11.8 +- 9.5	107 +- 58	14 +- 13	101 +- 69	309 +- 165	**0.53 +- 0.10**

Table 4.1: table of results for the optimized HNN model with Liouville addition.

	Train Loss		Test Loss		Energy	
Task	**Baseline**	**HNN**	**Baseline**	**HNN**	**Baseline**	**HNN**
1: Ideal mass-spring	38 +- 0.20	37 +- 2.0	38 +- 0.2	36 +- 1.8	124 +- 26	0.2 +-0.006
2: Ideal pendulum	34 +- 1.6	32 +- 1.6	37 +- 1.8	34 +- 1.7	27 +- 21	30 +- 2.4
3: Real pendulum	1.8 +- 0.1	9.1 +- 0.5	1.4 +- 0.4	5.8 +- 0.6	361 +- 7	11 +- 4.3
4: Two Body(10^6)	32 +- 4.7	2.5 +- 0.006	29 +- 4.6	2.3 +- 0.003	356 +- 149	19.1 +- 3.7
5: Three Body(10^2)	11.8 +- 9.5	7.7 +- 1.8	14.4 +- 13.5	27.5 +- 24	309 +- 165	1.65 +- 0.18

Table 4.2: table of results for the original HNN model

Again, the tables above list the MSE measurements in the training and testing datasets for each experiment. We used the exact same data sets to train both the original model and the Liouville model so that we were not biased with new datasets every time we ran the experiment. All values are multiplied by 10^3 unless noted otherwise, as in the two-body and three-body system. In comparison, the results may seem a bit confusing at first, as some of the values in Table 4.1 seem to have higher error values (MSE errors) than their equivalent in Table 4.2. The most important column in these tables is the energy column. This column lists the *mean squared error MSE* of the difference between the true total energies of a system and the energies calculated from the predicted **q,p** coordinates by HNN. The best values were those of the two-body problem with an increase in magnitude of order $\mathcal{O}(10^2)$. As for the rest of the experiments, even though the results of the MSEs in their training or testing columns are higher than those of the original model, the energy levels either remained indifferent to the original results or improved with the new optimizations introduced to the model.

Pixel HNN

One experiment from the original paper was deliberately removed, that is, the Pixel HNN autoencoder approach. The authors of HNN decided to test the integration of their work on Hamiltonian mechanics and combine it with a classical image autoencoder over a simulated set of dynamics for the pendulum experiment. The idea is to test the HNN on the latent space (in this case the latent vectors) of an autoencoder.

With that in mind, they developed a data set made up of pixel observations that capture the motion of a pendulum and integrated an autoencoder with a Hamiltonian neural network (HNN) to model its dynamics. According to them, this marks the first case in which a Hamiltonian is learned directly from pixel data.

Initially, they generated 200 trajectories, each composed of 100 frames, while ensuring that the absolute angular displacement of the pendulum arm did not exceed 6 radians. They began with $400{\times}400{\times}3$ **RGB** images, which were then cropped, de-saturated, and downscaled to produce $28{\times}28{\times}1$ grayscale frames. Moreover, they merged each frame with the subsequent one to form an input tensor of shape $batch{\times}28{\times}28{\times}2$. This dual frame approach allowed one to infer the velocity from the input, an essential aspect since without observable velocity, a nonrecurrent autoencoder would struggle to capture the entire state space of the system.

For the auto-encoder component, simplicity and ease of training were the primary objectives. They opted for fully connected layers instead of convolutional layers because of their straightforward nature. Both the encoder and decoder consisted of four fully-connected layers featuring ReLU activations and residual connections. Each layer was configured with 200 hidden units, except for the latent vector $\mathbf{z}$, which was limited to two units[5].

Regarding the HNN part of the model, they employed the same architecture and parameter settings as described in Chapter 3. Unless stated otherwise, the training process mirrored the procedure described in previous experiments mentioned in their original paper, and they found that applying a small weight decay of 10^{-5} proved beneficial[5].

The main piece of information that differentiates this experiment from the rest was the loss function they employed for it, which is in the form of:

$$\mathcal{L}_{CC} = ||z_p^t - (z_q^t - z_q^{t+1})||_2 \tag{4.18}$$

The loss function introduces an auxiliary loss term that forces the second part of the latent vector, labeled z_p, to mimic the derivatives of the first part, denoted as z_q. This mechanism ensures that the combined latent representation (z_q, z_p) acquires characteristics similar to canonical coordinates (q, p), as verified by the Poisson bracket relations essential for formulating a Hamiltonian. They also noted that the inclusion of this auxiliary loss did not harm the autoencoder performance, and its design is versatile enough to work with any autoencoder with an even-sized latent space[5].

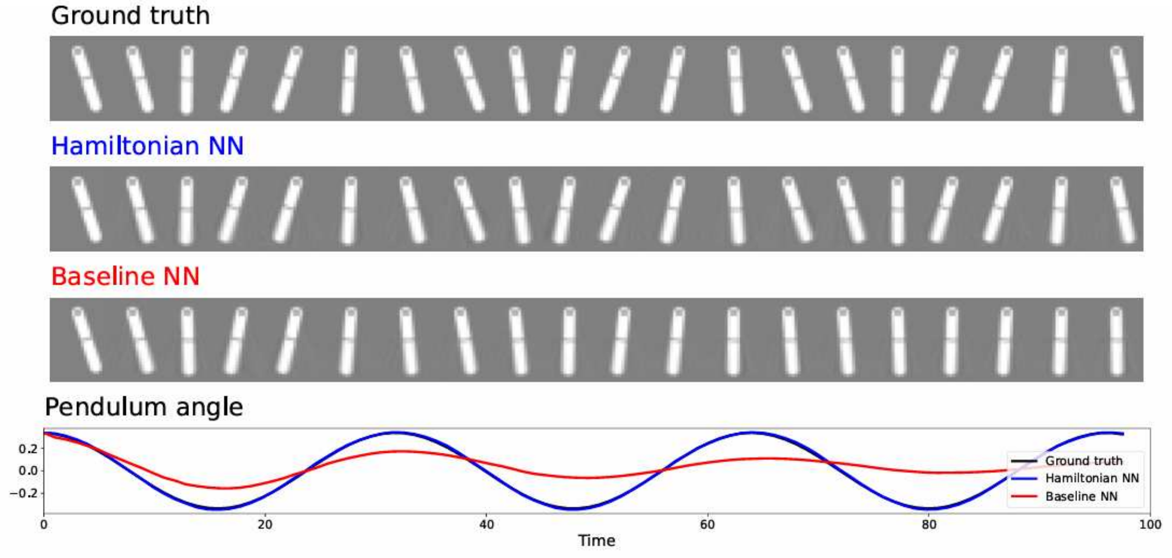


Figure 4.19: Graphics of the pixilated HNN experiment by making the model predict the dynamics of the pendulum in the latent space of the autoencoder described above. the HNN still outperforms the baseline model.

We did not attempt to test our optimization on this particular experiment because of some outdated and malfunctioning libraries used in the simulation of the dynamics of the autoencoder and its predictions. This was due to some outdated libraries or missing logic in the GitHub code published by the authors on GitHub. Also, we thought that this experiment's success was mainly due to the convolution network and auto-encoder approach employed by many networks in the scientific field. However, we decided to abandon testing this experiment due to the aforementioned facts about the scarcity of data, canonical or visual, in the scientific community, and depending on pixilated datasets from some real experiment may not always be available.

Chapter 5

Conclusion

Physics-Informed Neural Networks are becoming well known in the scientific community, especially in the pharmaceutical and biological sector, where a lot of molecular biology and chemistry are being innovated on a daily basis. The Hamiltonian Neural Network research paper was in fact an introduction to a set of future works built on top of it, like the Lagrangian Neural Network (*LNN*) written by the same authors as an improvement over the original HNN paper.The Liouville theorem approach implemented in this paper significantly improved the results of the experiments performed by the original HNN paper.

In attaining these improvements, there is definitely room for more optimization on the current approach. One can simply start isolating each experiment's behavior on its own and check how the Liouville theorem affects such an experiment rather than being an abstract approach. Another would be to properly investigate the distribution term in **4.2**,$\frac{\partial \rho}{\partial t}$, as this term was not integrated into the flow of the network of this work. An idea would be to make this distribution function of the density of the flow of particles in the phase space a learnable parameter λ by the network, and then optimize it accordingly.

It is worth noting that during our research endeavor on optimizing HNN, we stumbled upon some insightful knowledge that would have been useful for this work but were too complicated to implement at the moment. One very useful piece of information is called the *Invariant Surface Condition* or *ISC*, which turns out to be a whole branch of *Lie* group theory that addresses the problem of reducing partial differential equations from higher to lower orders. This idea seemed perfect as an approach to the Hamiltonian Network through the use of Autoencoders where the latent space would be a pool of well-known differential equations and their reductions,and the generator would try and generate new reduction forms for a given *PDE* as input.

Although the work on Hamiltonian systems through some classical physical phenomena like the k-body system or mass spring seems redundant, the benefit of such work is to showcase how useful *PINNs* are at improving scientific simulations. This work, as with many others, is improving on the current state-of-the-art simulations out there like in the energy, car and pharmaceutical industries, innovating new solutions that were once impossible to come by humans.

Bibliography

[1] F. Leyvraz A. Botero. "The two-dimensional three-body problem in a strong magnetic field is integrable". In: *Academia* (2014).

[2] R. Combescot. "Three-Body Coulomb Problem". In: *Phys. Rev. X* 7 (4 Nov. 2017), p. 041035. DOI: 10.1103/PhysRevX.7.041035. URL: https://link.aps.org/doi/10.1103/PhysRevX.7.041035.

[3] Silvia Daun et al. "Equation-based models of dynamic biological systems". In: *Journal of Critical Care* 23.4 (2008), pp. 585–594. ISSN: 0883-9441. DOI: https://doi.org/10.1016/j.jcrc.2008.02.003. URL: https://www.sciencedirect.com/science/article/pii/S0883944108000506.

[4] Judith R Goodstein and Levi-Civita Ricci. "s Tensor Analysis Paper". In: *Historia Mathematica* 4 (1977).

[5] Sam Greydanus, Misko Dzamba, and Jason Yosinski. "Hamiltonian Neural Networks". In: *CoRR* abs/1906.01563 (2019). arXiv: 1906.01563. URL: http://arxiv.org/abs/1906.01563.

[6] Hamiltonian system. *Hamiltonian system.* 2025. URL: https://en.wikipedia.org/wiki/Hamiltonian_system.

[7] Jean-Paul Haton. "A brief introduction to artificial intelligence". In: *IFAC Proceedings Volumes* 39.4 (2006). 9th IFAC Symposium on Automated Systems Based on Human Skill and Knowledge, pp. 8–16. ISSN: 1474-6670. DOI: https://doi.org/10.3182/

20060522-3-FR-2904.00003. URL: https://www.sciencedirect.com/science/article/pii/S1474667015330226.

[8] Andreas Henriksson. “Liouville’s theorem and the foundation of classical mechanics”. working paper or preprint. Apr. 2022. URL: https://hal.science/hal-03265484.

[9] ZI CHONG KAO. *CLASSICAL MECHANICS:THE THREE-BODY PROBLEM*. 2011. URL: https://math.uchicago.edu/~may/VIGRE/VIGRE2011/REUPapers/KaoZ.pdf.

[10] Peter Mann. “237Liouville’s Theorem &amp; Classical Statistical Mechanics”. In: *Lagrangian and Hamiltonian Dynamics*. Oxford University Press, June 2018. ISBN: 9780198822370. DOI: 10.1093/oso/9780198822370.003.0020. eprint: https://academic.oup.com/book/0/chapter/368894872/chapter-pdf/45359668/oso-9780198822370-chapter-20.pdf. URL: https://doi.org/10.1093/oso/9780198822370.003.0020.

[11] Houman Owhadi. “Bayesian Numerical Homogenization”. In: *Multiscale Modeling & Simulation* 13.3 (2015), pp. 812–828. DOI: 10.1137/140974596. eprint: https://doi.org/10.1137/140974596. URL: https://doi.org/10.1137/140974596.

[12] 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”. In: *Journal of Computational Physics* 378 (2019), pp. 686–707. ISSN: 0021-9991. DOI: https://doi.org/10.1016/j.jcp.2018.10.045. URL: https://www.sciencedirect.com/science/article/pii/S0021999118307125.

[13] Maziar Raissi, Paris Perdikaris, and George Em Karniadakis. “Inferring solutions of differential equations using noisy multi-fidelity data”. In: *Journal of Computational Physics* 335 (2017), pp. 736–746. ISSN: 0021-9991. DOI: https://doi.org/10.1016/j.jcp.2017.01.060. URL: https://www.sciencedirect.com/science/article/pii/S0021999117300761.

[14] Maziar Raissi, Paris Perdikaris, and George Em Karniadakis. "Machine learning of linear differential equations using Gaussian processes". In: *Journal of Computational Physics* 348 (2017), pp. 683–693. ISSN: 0021-9991. DOI: `https://doi.org/10.1016/j.jcp.2017.07.050`. URL: `https://www.sciencedirect.com/science/article/pii/S0021999117305582`.

[15] José Alberto Rodrigues. "Using Physics-Informed Neural Networks (PINNs) for Tumor Cell Growth Modeling". In: *Mathematics* 12.8 (2024). ISSN: 2227-7390. DOI: `10.3390/math12081195`. URL: `https://www.mdpi.com/2227-7390/12/8/1195`.

[16] Michael Schmidt and Hod Lipson. "Distilling Free-Form Natural Laws from Experimental Data". In: *Science* 324.5923 (2009), pp. 81–85. DOI: `10.1126/science.1165893`. eprint: `https://www.science.org/doi/pdf/10.1126/science.1165893`. URL: `https://www.science.org/doi/abs/10.1126/science.1165893`.

[17] Yusuf Seday. *Hamiltonian Mechanics*. URL: `https://www.academia.edu/43780414/Hamiltonian_Mechanics`.

[18] Jiang Wang et al. "Machine Learning of Coarse-Grained Molecular Dynamics Force Fields". In: *ACS Central Science* 5.5 (2019). PMID: 31139712, pp. 755–767. DOI: `10.1021/acscentsci.8b00913`. eprint: `https://doi.org/10.1021/acscentsci.8b00913`. URL: `https://doi.org/10.1021/acscentsci.8b00913`.

[19] C. Williams. "A Brief Introduction To Artificial Intelligence". In: *Proceedings OCEANS '83*. 1983, pp. 94–99. DOI: `10.1109/OCEANS.1983.1152096`.

[20] Intikhab Ulfat Zaheer Uddin. *Proofs of Vector Identities Using Tensors*. URL: `https://arxiv.org/pdf/1406.3060`.