Indian Institute of Technology Bombay

Master's Thesis

Exploring Modified Gravity Theories Through Neural Networks

Author: Devesh Dhole

Supervisor: Prof. Shankaranarayanan S

A thesis submitted in fulfillment of the requirements for the degree of M.Sc. Physics
in the
Department of Physics

January 13, 2024

Declaration of Authorship

I, **Devesh Dhole**, hereby declare that the following master's thesis, titled **"Exploring Modified Gravity Theories through Neural Networks"** is my own original work, and I have written it without any unauthorized assistance or use of sources beyond those referenced in the text. All contributions of others to this work, including guidance, ideas, and resources, have been duly acknowledged in the thesis.

I further affirm that:

- This thesis is not a reproduction or adaptation of any other work, and it has not been submitted as a thesis for another degree or qualification.
- Any direct quotations, paraphrased material, or ideas borrowed from other sources have been properly cited and referenced according to the citation style specified in the thesis guidelines.
- I take full responsibility for the originality, accuracy, and integrity of the content presented in this thesis.
- I am aware of and accept the consequences of plagiarism or academic misconduct in accordance with the policies and regulations of **Indian Institute of Technology Bombay**.

Date of Submission: 6th November 2023
Program: M.Sc.(Physics)
Signed: Devesh Dhole

Acknowledgements

It is a genuine pleasure to express my profound gratitude and deep regard to my advisor **Prof. Shankaranarayanan S.** for his guidance, valuable feedback, and constant encouragement throughout the project. His valuable suggestions were of immense help. I sincerely acknowledge his constant support and guidance during the project.

Besides my advisor, I would like to thank **Mr. Semin Xavier**, who is a Ph.D. student here at the Department of Physics, for his continuous support during this project. I am also grateful to the **Indian Institute of Technology, Bombay**, for providing all the required facilities. Last but not least, I would like to thank my family: my parents and my siblings for their continuous support during this time.

Abstract

Machine learning and artificial intelligence stand as technological marvels, revolutionizing how we tackle intricate problems, drive data-centric decisions, and automate tasks. The time has come to unite this cutting-edge field with the challenges of physics. Enter a groundbreaking neural network category known as Physics-Informed Neural Networks (PINNs). These remarkable function approximators have the capacity to comprehend the fundamental physical principles underlying any process by ingesting Partial Differential Equations (PDEs). They excel at generating predictions grounded in physics when provided with data points as their learning input. Our aspiration is to harness the immense potential of PINNs, employing them to make predictions on parameters that can be observed. To achieve this, we feed PINNs with PDEs derived from an array of modified theories of gravity. This undertaking serves as a bridge between the innovative world of artificial intelligence and the intricate landscapes of gravitational physics. The fusion of these realms promises to shed light on new horizons and unlock insights into the behavior of the universe under the influence of these modified theories of gravity.

Table of Contents

List of Figures

Chapter 1

Introduction

Physics-Informed Neural Networks (PINNs) represent an exciting addition to the ever-evolving landscape of neural network methodologies. Their potential for addressing a wide range of complex problems in the realm of physics and beyond is truly remarkable. In this thesis, we embark on a journey to unlock the full potential of neural networks by imbuing them with a deep understanding of various modified theories of gravity. By doing so, we aim to empower these networks with the capacity to make highly accurate predictions regarding observable data, even when working with a limited number of data points. This is achieved by leveraging the network's innate knowledge of the fundamental physical laws governing the behavior of the observables we seek to predict.

In this section, we will delve into the core principles of PINNs, shedding light on their inner workings and capabilities. Moreover, we will elucidate the current scope and aspirations of this thesis, outlining the exciting research directions and challenges that lie ahead.

1.1 Physics-Informed Neural Networks, a Fusion of Science and AI

Physics-Informed Neural Networks, or PINNs, represent a harmonious fusion of cutting-edge artificial intelligence with the fundamental principles of physics. These networks possess a unique ability to not only learn from data but also to incorporate the governing physical laws into their architecture. This distinctive feature makes them exceptionally well-suited for tackling a wide array of problems in fields such as astrophysics and cosmology.

1.2 Harnessing the Power of Neural Networks for Modified Theories of Gravity

Our thesis revolves around harnessing the remarkable power of neural networks to comprehend and effectively work with various modified theories of gravity. These theories, which extend or modify Einstein's General Theory of Relativity, play a pivotal role in our quest to understand the universe's most enigmatic phenomena. By training neural networks to grasp the intricacies of these theories, we equip them with the capacity to predict observables accurately.

1.3 Reducing Data Demands with Inherent Physical Understanding

One of the most compelling advantages of our approach is the potential to make predictions with far fewer data points. Traditional machine learning methods often require copious amounts of data to achieve reasonable accuracy. In contrast, PINNs, by virtue of being informed by the underlying physical laws, can make accurate predictions even when working with a limited dataset. This not only enhances the efficiency of our predictions but also opens up new avenues for research and application in scenarios where data is scarce or expensive to acquire.

1.4 Setting the Stage for Our Thesis

In this introductory section, we've provided an overview of the promising realm of Physics-Informed Neural Networks and the pivotal role they play in our research. As we progress through this thesis, we will dive deeper into the intricacies of PINNs, exploring their mathematical foundations and practical implementation. Additionally, we will discuss the specific challenges and opportunities in the context of modified theories of gravity and how our approach can reshape the landscape of predictions in this domain. By the end of this journey, we hope to showcase the true potential of PINNs as a powerful tool for bridging the gap between physics and artificial intelligence.

Chapter 2

Physics Informed Neural Networks(PINNs)

Since its introduction by M. Raissi et al., 2019, Physics-Informed Neural Networks (PINNs) have witnessed significant growth in their development. They have evolved into powerful tools capable of making real-world data predictions and serving as versatile universal function approximators. This evolution has given birth to a new class of Partial Differential Equation (PDE) solvers with the potential to offer faster and more accurate solutions. Numerous endeavors have sought to integrate the concept of PINNs into code and libraries, resulting in implementations like *NeuroDiffEq* (Feiyu Chen et al., 2020) which is written in python. While *NeuroDiffEq* is a valuable resource, it remains a work in progress, with room for the addition of various functionalities.

In this chapter, we embark on a journey of exploration. We dive into the mathematical foundations of PINNs, unraveling the principles that empower these networks to bridge the gap between physics and artificial intelligence. This understanding is crucial to appreciate the elegance and effectiveness of PINNs in solving complex problems.

Furthermore, we delve into the practical implementation and usage of the *NeuroDiffEq* package. This toolkit serves as our trusted companion, allowing us to harness the potential of PINNs for our specific case. While *NeuroDiffEq* is already a valuable asset, we acknowledge that it is not yet fully developed, leaving opportunities for further enhancements and functionalities.

As we navigate this chapter, our aim is to uncover the rich possibilities that PINNs, along with the *NeuroDiffEq* package, offer for advancing knowledge and exploring complex physical phenomena. It is through this blend of theory and practical implementation that we unlock the true potential of these innovative tools, which may shape the future of scientific computation and prediction.

2.1 Mathematical Foundations of PINNs

When dealing with differential equations, we typically have three essential criteria to meet. Firstly, we must have the differential equation itself, which describes how a function behaves in a certain domain. Secondly, there are the initial conditions, which tell us the state of the solution at the very beginning. Lastly, we encounter boundary conditions, specifying the values of the solution at the boundaries of our problem domain. In essence, if a function can meet all these three criteria, it should be the solution to the given set of partial differential equations (PDEs). Therefore, our main goal is to construct a function that can be approximated by a neural network and ensure that it satisfies all three of these conditions.

We start out by writing a general form of any differential equation,

$$\mathcal{L}u - f = 0 \tag{2.1}$$

where $\mathcal{L}$ is a differential operator that acts on $u(x,t)$ which is a function of x and t and also the solution to the equation that we seek and f is the source term or the forcing term. Now, as we convert it into a function of neural network we write it as $u_N(x,t;p)$ where p is the set of parameters of the neural network which are the weights and biases of the network. Now, in terms of u_N we can write equation (2.1) as,

$$\mathcal{L}u_N - f = 0 \tag{2.2}$$

And the residual of the function can be written as,

$$\mathcal{R}\left(u_N\right) = \mathcal{L}u_N - f \tag{2.3}$$

which should be identically zero for the solution to be valid. Now we wish to use this residual as the loss function for the network. We can converts this into the following optimization problem,

$$\min_p \left(\mathcal{L}u_N - f\right)^2 \tag{2.4}$$

But note that the choice of loss function is not unique. Now we wish the solution to satisfy the initial and boundary conditions as well. For example, if we have the initial condition like $u\left(x,t_0\right) = u_0(x)$ then we can incorporate the same in the following way which again is not unique,

$$\min_p \left[\left(\mathcal{L}u_N - f\right)^2 + \lambda\left(u_N\left(x,t_0\right) - u_0(x)\right)^2\right] \tag{2.5}$$

However, in practice, the above formulation fails to satisfy the initial conditions exactly. Fortunately, there is another method that can help us satisfy the initial conditions exactly. The second method changes the neural network function u_N itself such that the initial conditions are automatically satisfied. Following is the reformulation of the function u_N to $\widetilde{u}$,

$$\widetilde{u}(x,t;p) = u_0(x) + \left(1 - e^{-(t-t_0)}\right) u_N(x,t;p) \tag{2.6}$$

Accordingly, the optimization problem turns out to be,

$$\min_p \left(\mathcal{L}\widetilde{u} - f\right)^2 \tag{2.7}$$

Now, we wish to satisfy the boundary conditions as well which is achieved by a similar process. The reformulation looks like the following,

$$\widetilde{u}(x,t;p) = A\left(x,t;x_{\text{boundary}}\,,t_0\right) u_N(x,t;p) \tag{2.8}$$

where $A\left(x,t;x_{\text{boundary}}\,,t_0\right)$ is such that $\widetilde{u}(x,t)$ satisfies the correct boundary conditions. Both of these approaches have their own merits. The first method is straightforward to implement and lends itself well to handling high-dimensional PDEs and PDEs defined on intricate domains. On the other hand, the second approach guarantees the exact satisfaction of initial and boundary conditions. Given that differential equations can be highly sensitive to these conditions, this becomes a crucial factor. Additionally, the second method offers the advantage of potentially reducing the training effort required for Artificial Neural Networks (ANNs).

2.2 *NeuroDiffEq*: A Python package for solving differential equations with neural networks

NeuroDiffEq is a versatile package designed for tackling the intricate realm of differential equations using neural networks. Differential equations are mathematical tools that describe the relationships between a function and its derivatives, cropping up in numerous scientific and engineering fields. Traditionally, these problems are addressed through numerical techniques like finite difference or finite element methods. While effective, these methods have their limitations in representing functions that are continuous or differentiable.

The intriguing potential lies in our quest to compute solutions for differential equations that exhibit these continuous and differentiable traits. This is where artificial neural networks come into play as universal function approximators. They've displayed the capability to solve both ordinary differential equations (ODEs) and partial differential equations (PDEs) under specific initial or boundary conditions. *NeuroDiffEq's* mission is to harness these existing techniques and implement them in a way that ensures flexibility, enabling the software to adapt to a wide spectrum of user-defined problems.

We will try to explore the different functionalities of the *NeuroDiff* package with examples. Later on, some additional features will be explained independently. Consider the following set of equations, called as *Lotka-Volterra* equations,

$$\frac{du(t)}{dt} = \alpha u(t) - \beta u(t)v(t), \qquad u(0) = u_0 \tag{2.9}$$

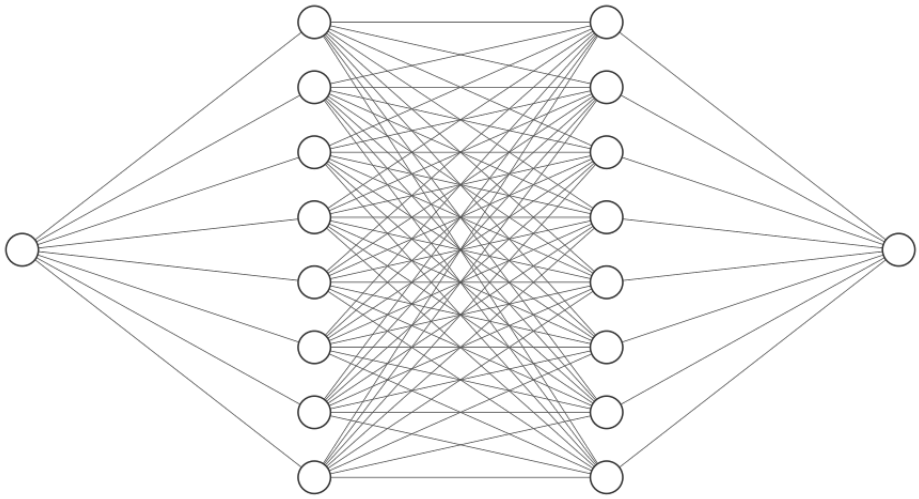
$$\frac{dv(t)}{dt} = \delta u(t)v(t) - \gamma v(t), \qquad v(0) = v_0 \tag{2.10}$$

Physically, u and v are population of prey and predator respectively. We will cast this equations and initial conditions into something that can be ran using the *NeuroDiffEq* package. Following is the code for the same,

```
1 def ode_system(u, v, t):
2     return [diff(u,t)-(u-u*v), diff(v,t)-(u*v-v)]
3
4 conditions = [IVP(t_0=0.0, u_0=1.5), IVP(t_0=0.0, u_0=1.0)]
5 nets = [FCNN(actv=SinActv), FCNN(actv=SinActv)]
6
7 solver = Solver1D(ode_system, conditions, t_min=0.1, t_max=10.0, nets=nets)
8 solver.fit(max_epochs=100000)
9 solution = solver.get_solution()
```

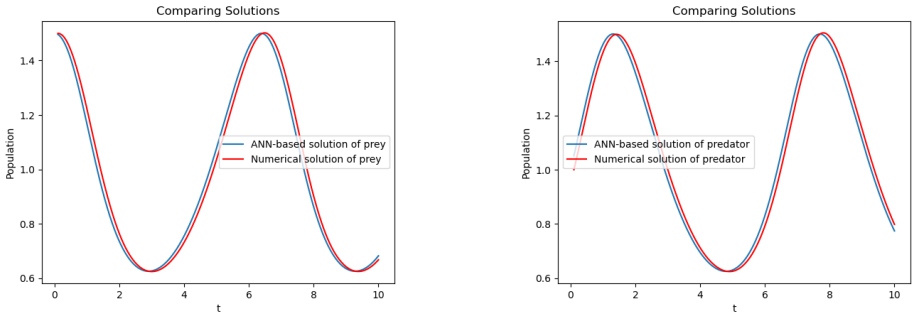
Let's look at the above code line by line. Line 1 and 2 defines the system of equation as described in equation (2.9) and (2.10). *NeuroDiffEq* provides a functionality to differentiate the function of dependent variables with respect to independent variables out of the box and thus we have used it here. For example, *diff(u,t)* denotes differentiation of u with respect to t. NeuroDiffEq provides a plethora of different types of initial and boundary conditions out of the box to make our job easier. Line 4 describes the two initial conditions given in equation (2.9) and (2.10) with values of $u_0 = 1.5$ and $v_0 = 1.0$. Here, IVP stands for Initial Value Problem. Now, we need to define the structure of the neural network which is defined in line 5 in our case. Here the default configuration has been applied which is a Fully Connected Neural Network(FCNN) with one neuron in input layer, one neuron in output layer and two hidden layers with 32 neurons each. The activation function used is *SinActv* which is basically our classic sin function. Finally we define the solver which takes input as the system of equations, initial/boundary conditions, configuration of the system and the domain over which we need to solve the system. In line 8, we give the number of

epochs over which the neural network will be trained. In general, the more the number of epochs better will be the training but once saturation is reached it is no longer beneficial to increase the number of epochs. Finally, line 9 will execute the solver to train the neural network. Following is the architecture of the given network,



Input Layer(1) Hidden Layer 1(32) Hidden Layer 2(32) Output Layer(1)

Now, we also integrate the equation using Runge Kutta 4 method to get solution numerically. Following is the comparison between the two solutions for variable u(population of prey) and v(population of predator),



(a) Prey Population (b) Predator Population

Figure 2.1: 2 Solutions to *Lotka-Voltera* system of equations

As we can see the numerical solution using RK4 and using ANN(Artificial Neural Network) are pretty close. Following plot visualizes the scale of error as compared to the actual value,

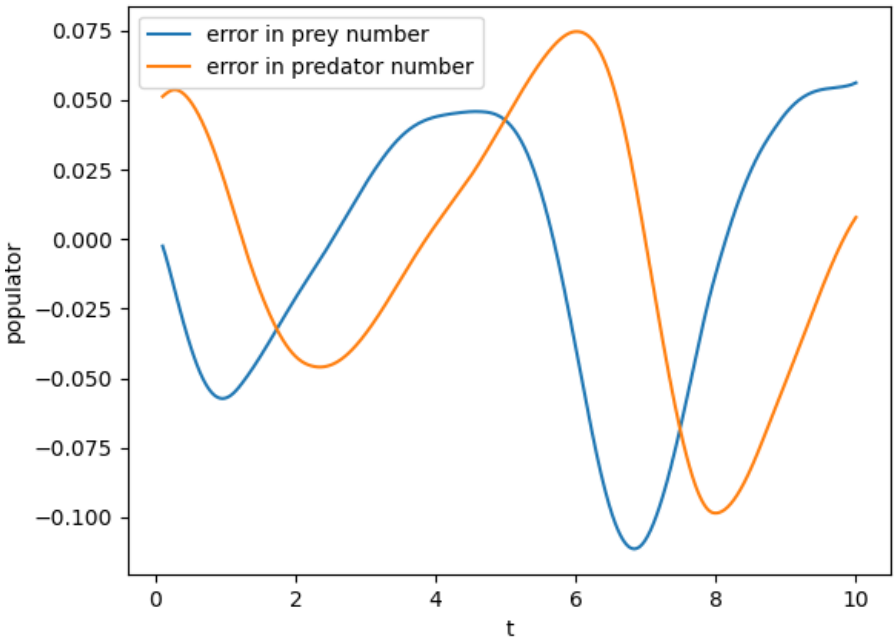


Figure 2.2: Error of ANN solution from numerical solution

In this example, we looked at implementing differential equation with initial conditions. Let us now look at setting up boundary conditions. For this, we will consider the Laplace equation. We know laplace equation can be written as,

$$F(u, x, y) = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0 \tag{2.11}$$

for $(x, y) \in [0, 1] \times [0, 1]$ s.t.

$$\begin{aligned} u(x, y)|_{x=0}^{|} &= \sin(\pi y) \\ u(x, y)|_{x=1} &= 0 \\ u(x, y)|_{y=0} &= 0 \\ u(x, y)|_{y=1} &= 0 \end{aligned}$$

The analytical solution is

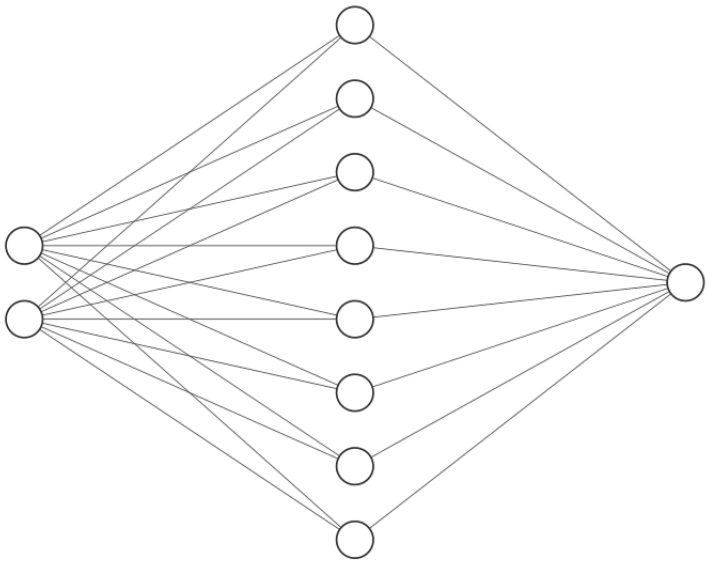
$$u(x, y) = \frac{\sin(\pi y)\sinh(\pi(1 - x))}{\sinh(\pi)} \tag{2.12}$$

We now need to feed the above equation and the boundary conditions to the Neural Network. Let's look at the code for the same,

```
def pde_system(u, x, y):
    return [diff(u, x, order=2) + diff(u, y, order=2)]

conditions = [
    DirichletBVP2D(
        x_min=0, x_min_val=lambda y: torch.sin(np.pi*y),
        x_max=1, x_max_val=lambda y: 0,
        y_min=0, y_min_val=lambda x: 0,
```

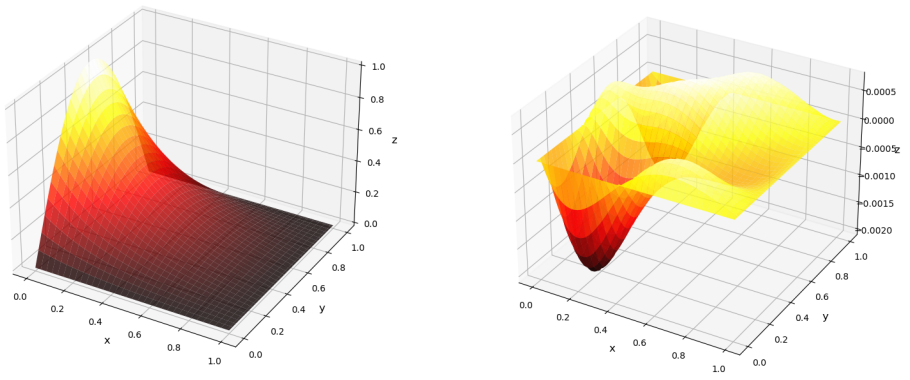
```
        y_max=1, y_max_val=lambda x: 0,
    )
]
nets = [FCNN(n_input_units=2, n_output_units=1, hidden_units=(512,))]

solver = Solver2D(pde_system, conditions, xy_min=(0, 0), xy_max=(1, 1),
↪  nets=nets)
solver.fit(max_epochs=2000)
solution = solver.get_solution()
```

Most part of the code is just an extension from 1D problem to 2D problem as compared to *Lotka-Voltera* equations. The boundary conditions which are essentially **Dirichlet** has been implemented in *NeuroDiffEq* as *DirichletBVP2D*. The lines from 4 to 11, implements all the four boundary conditions one by one. One more difference as compared to the previous set of equations is that there is only one output but two inputs. Thus we need our neural network to have two input neurons and one output neuron along with neurons in hidden layers which we have chosen to be 512. Thus our neural network looks like the following,



Input Layer(2) Hidden Layer(512) Output Layer(1)

Let's look at the 2D solution of the equation as well as the residue derived from the analytical solution given by equation (2.12),



(a) $u(x, y)$ as solved by neural network

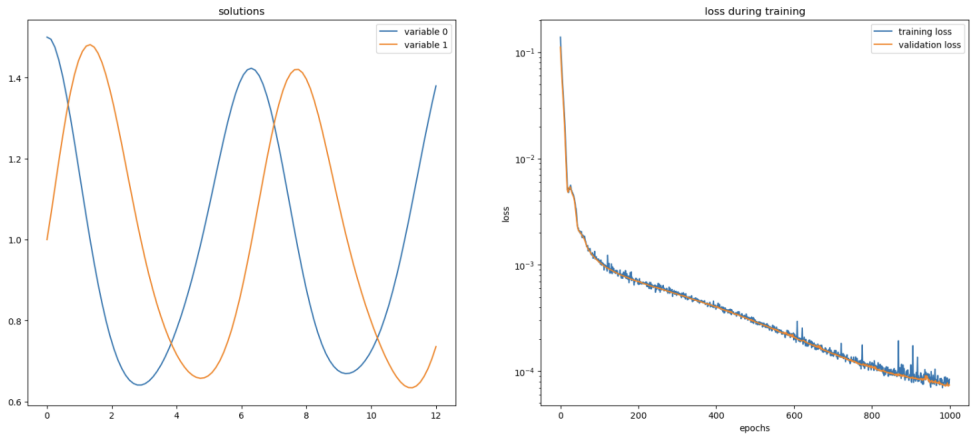
(b) Residual of the neural network solution

Figure 2.3: 2 Solution and Residue of laplace equation

Now, there are a few additional functionalities that comes out of the box with *NeuroDiffEq* that might be useful to solve complex problems. First such functionality is Monitor. Using the Monitor functionality we can essentially monitor how the solutions comes about with every n epochs. Let's have a look at the code snippet for the same,

```
...
monitor = Monitor1D(t_min=0.1, t_max=10.0, check_every=100)
solver.fit(max_epochs=1000,callbacks=[monitor.to_callback()])
```

For every 100 epochs, the below plots are updated helping us see the evolution of the solution converging and loss function values dropping with every iteration,



Next, we might want to have a custom network which specifically suits for a particular set of equations or problems. It is not guaranteed that a given configuration of network will work for all set of equations. Let's have a look at a code snippet that gives us the freedom to define our custom network,

```
...
net1 = FCNN(n_input_units=..., n_output_units=..., hidden_units=[..., ...,
↪  ...], activation=...)
nets = [net1, net2, ...]
```

In the above code, we can essential modify anything ranging from number of input/output neurons, number of hidden layer neurons, activation function, type of neural network and number of hidden layers. This gives us a huge amount of freedom in terms of the structure of the neural network.

Within *NeuroDiffEq*, the neural networks undergo training by minimizing the loss function that measures the residuals of ordinary or partial differential equations (ODE/PDE) evaluated at specific points within the problem domain. These points are sampled randomly, and with each iteration, the sampling is refreshed. To maintain control over the quantity, distribution, and boundaries of these sampled points, you have the flexibility to define your custom training and validation data generators. This feature allows you to tailor the training process to your specific needs, ensuring that the neural networks are effectively honed to handle the intricacies of your differential equations. Whether you require a particular number of sample points, a specific distribution pattern, or the confinement of sampling to predefined domains, *NeuroDiffEq* empowers you to fine-tune the training process for optimal results. Let's have a look at a code snippet which allows you define your own generators,

```
...
g1 = Generator1D(size=..., t_min=..., t_max=..., method=..., noise_std=...)
g2 = Generator1D(size=..., t_min=..., t_max=..., method=..., noise_std=...)
solver = Solver1D(..., train_generator=g1, valid_generator=g2)
```

The *size* parameter allows you to specify the number of points in the domain that you want to sample. *t_min* and *t_max* is the lower and upper bound of the domain. The *method* parameter lets you define the kind of distribution we want to have. Some of the possible distributions are uniform and log-space-noisy. The *noise_std* parameter defines the standard deviation of the noise(if applicable).

While *NeuroDiffEq* offers a multitude of functionalities, our focus has been primarily on the key features that align with the goals of our thesis. These notable functions have been essential for our research endeavors. However, our exploration is far from complete.

In the upcoming section, we will delve into a fascinating yet highly non-linear system—the double pendulum. This system is a captivating example of chaos, where small variations in initial conditions can lead to vastly different outcomes. By examining such intricate systems, we aim to expand our understanding and demonstrate the capabilities of *NeuroDiffEq* in tackling complex and chaotic phenomena.

2.3 Exploring Chaotic Dynamics using *NeuroDiffEq*: The Double Pendulum

A major question that comes into our mind when talking about the capability of PINNs is to solve non-linear system that show chaotic behaviour. In this section we will see that PINNs are so powerful that they can even tackle such systems. Let's setup the mathematics required for solving the double pendulum system. Let's say the two degrees of freedom are taken to be θ_1 and θ_2, the angle of each pendulum rod from the vertical.

The Lagrangian, $\mathcal{L}$ can be written as:

$$\mathcal{L} = \frac{1}{2}(m_1 + m_2) l_1^2 \dot{\theta}_1^2 + \frac{1}{2} m_2 l_2^2 \dot{\theta}_2^2 + m_2 l_1 l_2 \dot{\theta}_1 \dot{\theta}_2 \cos(\theta_1 - \theta_2) + (m_1 + m_2) l_1 g \cos\theta_1 + m_2 g l_2 \cos\theta_2.$$

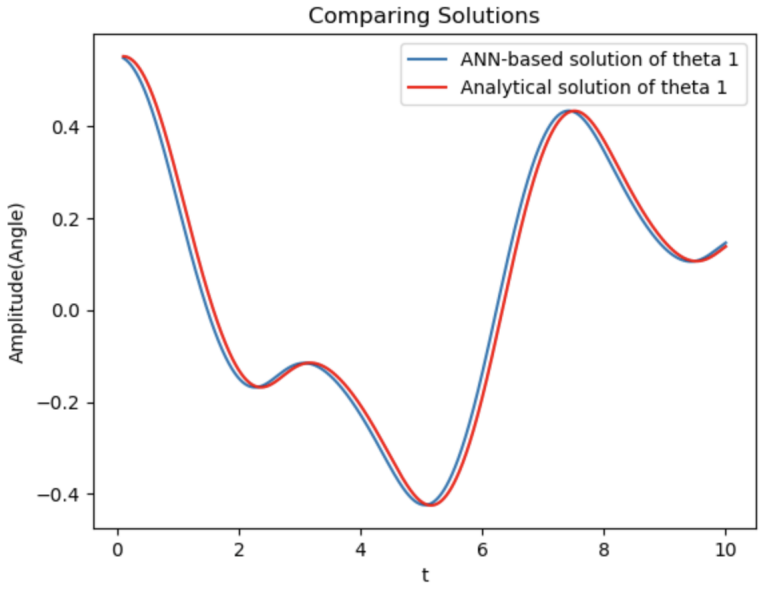
The Euler-Lagrange equations are:

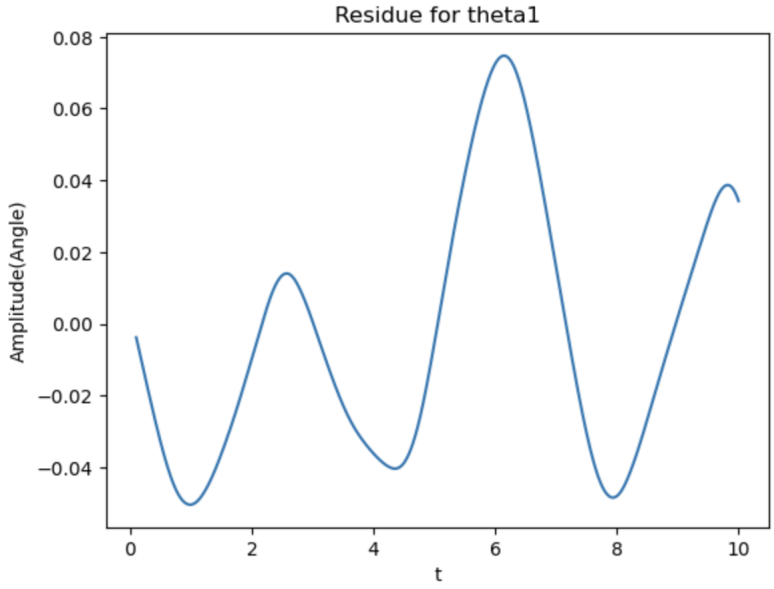
$$\frac{\mathrm{d}}{\mathrm{d}t}\left(\frac{\partial \mathcal{L}}{\partial \dot{q}_i}\right) - \frac{\partial \mathcal{L}}{\partial q_i} = 0 \quad \text{for } q_i = \theta_1, \theta_2$$

That gives us the equations of motion:

$$(m_1 + m_2)\, l_1 \ddot{\theta}_1 + m_2 l_2 \ddot{\theta}_2 \cos(\theta_1 - \theta_2) + m_2 l_2 \dot{\theta}_2^2 \sin(\theta_1 - \theta_2) + (m_1 + m_2)\, g \sin\theta_1 = 0,$$
$$m_2 l_2 \ddot{\theta}_2 + m_2 l_1 \ddot{\theta}_1 \cos(\theta_1 - \theta_2) - m_2 l_1 \dot{\theta}_1^2 \sin(\theta_1 - \theta_2) + m_2 g \sin\theta_2 = 0.$$

We fed these equations to NueroDiffEq and we have plotted below the result for θ_1 along with numerical solution obtained using RK4 method,





As we can see that *NeuroDiffEq* is completely capable of solving chaotic non-linear systems such as double pendulum.

Chapter 3

Applying PINNs to General Relativity

Having witnessed the remarkable capabilities of Physics-Informed Neural Networks (PINNs), ranging from solving relatively straightforward problems like the Laplace equation to tackling intricate and even chaotic systems like the double pendulum, it's now time to take a significant step forward. Our journey leads us closer to the application of PINNs in the realm of General Relativity (GR) and subsequently extends to Modified Gravity Theories.

In this chapter, our primary focus is on General Relativity (GR). We embark on a quest to derive fundamental metrics in this gravitational framework, notably the Schwarzschild metric and the Reissner–Nordström metric. Furthermore, we lay the groundwork for the mathematical machinery required for implementing the Kerr Metric. It's important to note that while our intention is to harness the power of PINNs in this context, the implementation is an ongoing work in progress. Our journey through this chapter represents a crucial step in our exploration of the fascinating intersection of machine learning and the complexities of gravitational physics.

3.1 Schwarzschild Metric

Let's envision a scenario where we have a matter distribution with central symmetry, giving rise to a spherically symmetric metric. In this setup, if there's any matter in motion, it should be moving radially. When we adopt spherical space coordinates, represented as r, θ, ϕ, the most general expression for the interval ds^2 in this centrally symmetric system is as follows:

$$ds^2 = h(r,t)dr^2 + k(r,t)\left(\sin^2\theta d\phi^2 + d\theta^2\right) + l(r,t)dt^2 + a(r,t)drdt \tag{3.1}$$

where a, h, k, l are certain functions of the "radius vector" r and the "time" t. Making use of the possibility that we can transform the coordinates r and t into each other, we choose the coordinate r and the time t in such a way that the coefficient $a(r,t)$ of $drdt$ in the expression for ds^2 vanishes and, secondly, the coefficient $k(r,t)$ becomes equal simply to $-r^2$. As we will see afterwards, it is convenient to write the quantities h and l in exponential form, as $-e^{\lambda}$ and e^{v} respectively, where λ and v are some functions of r and t. Thus we obtain the following expression for ds^2 :

$$ds^2 = e^v c^2 dt^2 - r^2\left(d\theta^2 + \sin^2\theta d\phi^2\right) - e^{\lambda}dr^2 \tag{3.2}$$

Denoting by x^0, x^1, x^2, x^3, respectively, the coordinates ct, r, θ, ϕ, we have for the nonzero components of the metric tensor the expressions

$$g_{00} = e^{\nu}, \quad g_{11} = -e^{\lambda}, \quad g_{22} = -r^2, \quad g_{33} = -r^2 \sin^2\theta$$

Clearly,

$$g^{00} = e^{-v}, \quad g^{11} = -e^{-\lambda}, \quad g^{22} = -r^{-2}, \quad g^{33} = -r^{-2} \sin^{-2}\theta$$

With these values it is easy to calculate the christoffel symbols Γ^k_{ij}. The calculation leads to the following expressions:

$$\begin{array}{lll} \Gamma^1_{11} = \frac{\lambda'}{2}, & \Gamma^0_{10} = \frac{v'}{2}, & \Gamma^2_{33} = -\sin\theta\cos\theta, \\ \Gamma^0_{11} = \frac{\lambda}{2}e^{\lambda-\nu}, & \Gamma^1_{22} = -re^{-\lambda}, & \Gamma^1_{00} = \frac{v'}{2}e^{v-\lambda}, \\ \Gamma^2_{12} = \Gamma^3_{13} = \frac{1}{r}, & \Gamma^3_{23} = \cot\theta, & \Gamma^0_{00} = \frac{\dot{v}}{2} \\ \Gamma^1_{10} = \frac{\dot{\lambda}}{2}, & \Gamma^1_{33} = -r\sin^2\theta e^{-\lambda}. & \end{array}$$

From Γ^k_{ij} we can derive, the Ricci Tensor R^j_i. Applying Einstein Field Equations leads to the following equations,

$$\frac{8\pi k}{c^4}T^1_1 = -e^{-\lambda}\left(\frac{v'}{r} + \frac{1}{r^2}\right) + \frac{1}{r^2}, \tag{3.3}$$

$$\frac{8\pi k}{c^4}T^2_2 = \frac{8\pi k}{c^4}T^3_3 = -\frac{1}{2}e^{-\lambda}\left(v'' + \frac{v'^2}{2} + \frac{v'-\lambda'}{r} - \frac{v'\lambda'}{2}\right) + \frac{1}{2}c^{-v}\left(\lambda' + \frac{\lambda^2}{2} - \frac{\lambda\dot{\nu}}{2}\right) \tag{3.4}$$

$$\frac{8\pi k}{c^4}T^0_0 = -e^{-\lambda}\left(\frac{1}{r^2} - \frac{\lambda'}{r}\right) + \frac{1}{r^2} \tag{3.5}$$

$$\frac{8\pi k}{c^4}T^1_0 = -e^{-\lambda}\frac{\lambda}{r} \tag{3.6}$$

Thus, for the vacuum solutions we set the stress-energy tensor T^j_i to be zero. Thus the LHS of all equations 3.3 to 3.6 goes to zero leading to the following set of equations,

$$e^{-\lambda}\left(\frac{v'}{r} + \frac{1}{r^2}\right) - \frac{1}{r^2} = 0 \tag{3.7}$$

$$e^{-\lambda}\left(\frac{\lambda'}{r} - \frac{1}{r^2}\right) + \frac{1}{r^2} = 0 \tag{3.8}$$

$$\dot{\lambda} = 0 \tag{3.9}$$

The analytical solution to the set of equations (3.7) to (3.9) gives,

$$e^{-\lambda} = e^{v} = 1 + \frac{\text{const}}{r} \tag{3.10}$$

Now we need a boundary condition to get the constant. Taking the weak field approximation, i.e. at large distances Newton's law should hold gives the constant as const $= -\left(2GM/c^2\right)$. This quantity has the dimensions of length and is famously called the Schwarzschild radius given by,

$$r_s = \frac{2GM}{c^2} \tag{3.11}$$

Thus we finally obtain the space-time metric in the form:

$$ds^2 = \left(1 - \frac{r_s}{r}\right)c^2dt^2 - r^2\left(\sin^2\theta d\phi^2 + d\theta^2\right) - \frac{dr^2}{1 - \frac{r_s}{r}} \tag{3.12}$$

Now, our task is to solve the system of equations 3.7 to 3.9 using *NeuroDiffEq*. Equation (3.9) just dictates that the solution is static thus the problem is converted into one independent variable problem with variable being r. Also, as computer do not understand units we need to convert the equation into a dimensionless equation. Let's define a new dimentionless variable R defined by,

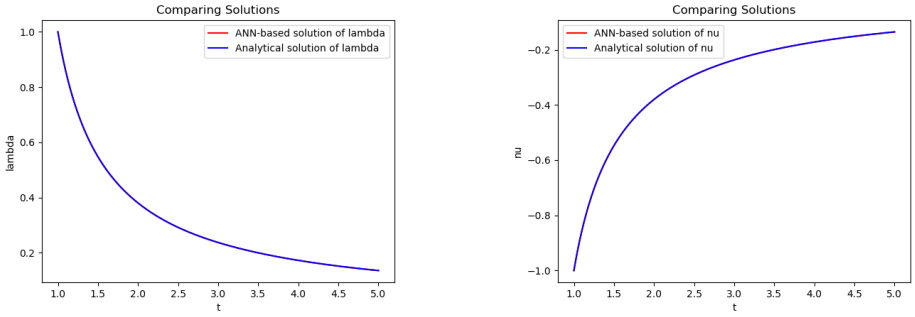
$$R = \frac{r}{r_h} \tag{3.13}$$

where, r_h is horizon radius. Substituting this R into equation (3.7) and (3.8) gives the following set of equations,

$$e^{-\lambda}\left(\frac{v'}{R} + \frac{1}{R^2}\right) - \frac{1}{R^2} = 0 \tag{3.14}$$

$$e^{-\lambda}\left(\frac{\lambda'}{R} - \frac{1}{R^2}\right) + \frac{1}{R^2} = 0 \tag{3.15}$$

The above equations are essentially look the same except that now they are dimensionless and the integration variable goes from 1 to ∞ rather than 0 to ∞ which would have been difficult to implement using **NeuroDiffEq**. Following are the results of running this system of equations on *NeuroDiffEq*,

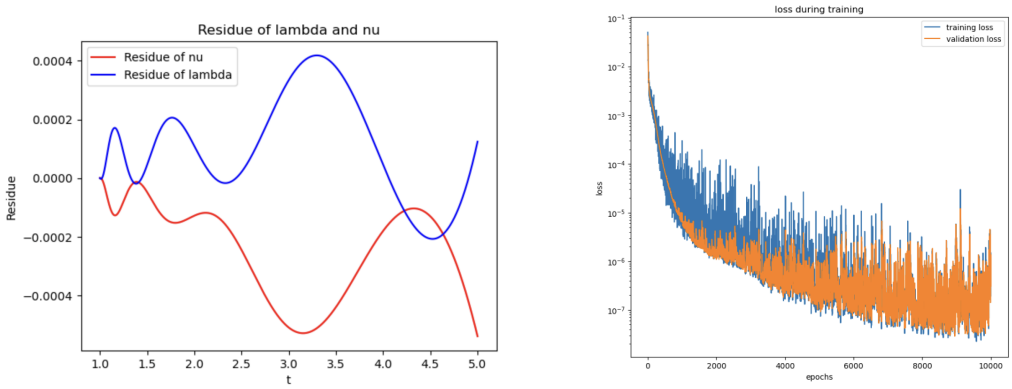


(a) Plot for λ

(b) Plot for ν

Figure 3.1: Comparing numerical and ANN based solutions for λ and ν

We have run the system with the default configuration for the neural network configuration as provided by *NeuroDiffEq*. From the above plots we can see that in both the plot the two solutions are closely hugging together and thus **NueroDiffEq** seems to have solved the system of equations successfully giving us the Schwarzschild metric. To analyze further we can look at the residuals of λ and ν as well as the loss curve over the epochs.



(a) Plot for residue of λ and ν

(b) Loss during training

Figure 3.2: Residue Plot and Loss Plot

Upon examining the residuals, we observe that they are substantially lower when compared to the actual values of the solution, which is a promising sign of a well-attained solution. Additionally, the loss curve displays a rapid descent, indicative of effective learning by the neural network. These combined observations suggest that the network has successfully grasped and represented the underlying patterns and relationships in the data.

3.2 Reissner–Nordström Metric

Having previously explored the metric describing an uncharged and non-rotating black hole, known as the Schwarzschild metric, we now turn our attention to another intriguing spacetime geometry: the metric of a non-rotating charged black hole, commonly referred to as the Reissner–Nordström Metric. Just as we delved into the mathematical underpinnings of deriving the Schwarzschild metric, we will embark on a similar journey in this case. As previously described in equation 3.2, any general static and sperically symmetric metric can be written as,

$$ds^2 = e^{\nu}c^2dr^2 - e^{\lambda}dr^2 - r^2\left(d\theta^2 + \sin^2\theta d\phi^2\right) \tag{3.16}$$

The difference between Schwarzschild and Reissner–Nordström Metric arises because there is non zero stress-energy tensor because of the electromagnetic field generated by the charge on the blackhole. The stress-energy tensor can be written as,

$$T_{\mu\nu} = \frac{1}{4\pi}\left(\frac{1}{4}g_{\mu\nu}f_{\alpha\beta}f^{\alpha\beta} - f_{\mu\alpha}f^{\alpha}_{\nu}\right). \tag{3.17}$$

A spherically symmetric vector potential A_μ, due to the electromagnetic field, will have the following vanishing components: $A_2 = A_3 = 0$. Because of the spherical symmetry, only A_0 and A_1 can differ from zero. But after taking into account the gauge freedom, the only nonvanishing component left of the electromagnetic vector potential is A_0 . Thus, the electromagnetic field tensor $f_{\mu\nu}$, is given by,

$$f_{01} = \frac{\partial A_0}{\partial r} = -f_{10} \tag{3.18}$$

Finally using equation (3.17), we then the stress-energy tensor as,

$$T_{\mu}^{\nu} = \frac{1}{8\pi} e^{-(r+\lambda)} \left(\frac{\partial A_0}{\partial r}\right)^2 \begin{pmatrix} 1 & & & \\ & 1 & & \\ & & -1 & \\ & & & -1 \end{pmatrix} \tag{3.19}$$

Now we need to solve Maxwell equations with a vanishing current, $j^{\alpha} = 0$ given by,

$$\frac{1}{\sqrt{-g}} \frac{\partial \left(\sqrt{-g} f^{\alpha\beta}\right)}{\partial x^{\beta}} = \frac{4\pi}{c} j^{\alpha} \tag{3.20}$$

Using the metric tensor from equation (3.16) we then find,

$$\sqrt{-g} = r^2 e^{(\nu+\lambda)/2} \sin\theta. \tag{3.21}$$

The Maxwell equations then reduce to the following two equations,

$$\frac{\partial \left(\sqrt{-g} f^{01}\right)}{\partial r} = -\frac{\partial \left(r^2 e^{-(r+\lambda)/2} A_0'\right)}{\partial r} \sin\theta = 0 \tag{3.22}$$

$$\frac{\partial \left(\sqrt{-g} f^{10}\right)}{\partial r} = \frac{\partial \left(r^2 e^{-(r+\lambda)/2} A_0'\right)}{\partial r} \sin\theta = 0 \tag{3.23}$$

From equations (3.22) and (3.23) we obtain,

$$r^2 e^{-(n+\lambda)/2} A_0' = \text{ constant.} \tag{3.24}$$

The constant appearing on the right-hand side of the above equation, which is independent of the coordinates r and t, may be determined by going to large distances r. In that limit the exponential functions tend to unity, and one obtains $r^2 A_0' =$ constant. Since $A_0 = e/r$ in the absence of gravitation, where e is the total charge of the gravitating body, we conclude that our constant is equal to -e. equation (3.24) may therefore be written as,

$$A_0' = -\frac{e}{r^2} e^{(\nu+\lambda)/2} \tag{3.25}$$

Now, using equation (3.19) we obtain,

$$T_{\mu}^{\nu} = \frac{e^2}{8\pi r^4} \begin{pmatrix} 1 & & & \\ & 1 & & \\ & & -1 & \\ & & & -1 \end{pmatrix}$$

for the stress-energy tensor of the electromagnetic field produced by the gravitating body. Now, using Einstein field equations, we get,

$$e^{-\lambda} \left(\frac{\lambda'}{r} - \frac{1}{r^2}\right) + \frac{1}{r^2} = \frac{Ge^2}{c^4 r^4} \tag{3.26}$$

$$e^{-\lambda} \left(\frac{v'}{r} + \frac{1}{r^2}\right) - \frac{1}{r^2} = -\frac{Ge^2}{c^4 r^4} \tag{3.27}$$

$$\dot{\lambda} = 0 \tag{3.28}$$

Their analytical solutions are given by,

$$e^{\nu} = e^{-\lambda} = 1 - \frac{r_s}{r} + \frac{r_e^2}{r^2}. \tag{3.29}$$

where, r_s and r_e are given by,

$$r_s = \frac{2GM}{c^2} \tag{3.30}$$

$$r_e^2 = \frac{Ge^2}{c^4}. \tag{3.31}$$

To summarize the above results, we have obtained for the Reissner solution the following metric tensor:

$$g_{\mu\nu} = \begin{pmatrix} 1 - \frac{r_s}{r} + \frac{r_e^2}{r^2} & & & \\ & -\left(1 - \frac{r_s}{r} + \frac{r_e^2}{r^2}\right)^{-1} & & \\ & & -r^2 & \\ & & & -r^2 \sin^2\theta \end{pmatrix} \tag{3.32}$$

As we did for the case of Schwarzschild metric, we need to make equations dimensionless so again following the same strategy we define a new variable R as,

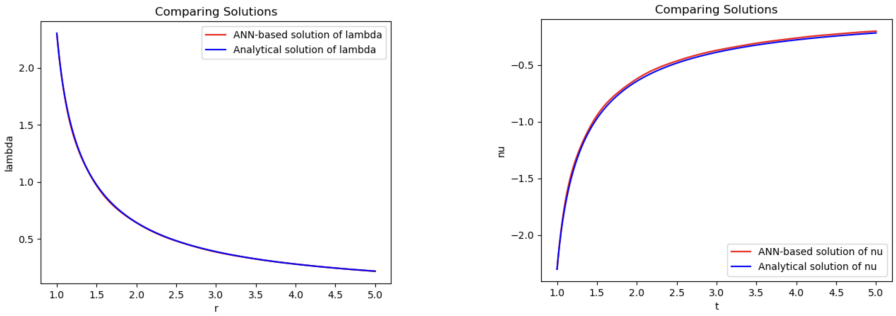
$$R = \frac{r}{r_e} \tag{3.33}$$

This converts the equations to,

$$e^{-\lambda}\left(\frac{\lambda'}{R} - \frac{1}{R^2}\right) + \frac{1}{R^2} = \frac{\alpha}{R^4} \tag{3.34}$$

$$e^{-\lambda}\left(\frac{v'}{R} + \frac{1}{R^2}\right) - \frac{1}{R^2} = -\frac{\alpha}{R^4} \tag{3.35}$$

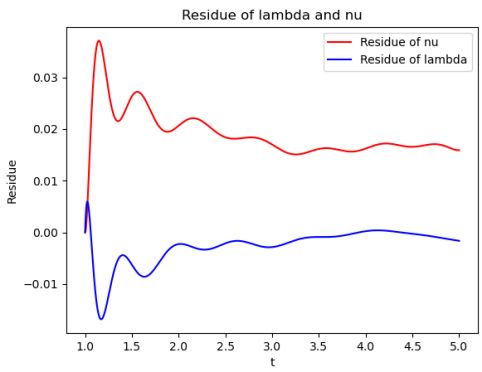
where α is some dimensionless constant. We now feed this system of equations along with boundary conditions to give us the following solutions outside the outer horizon,



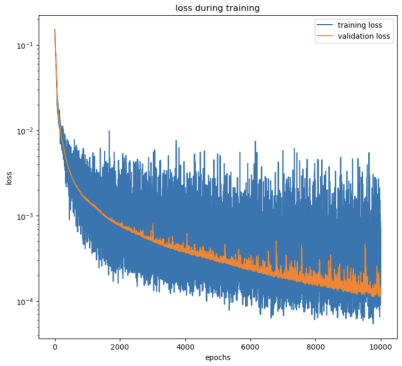
(a) Plot for λ (b) Plot for ν

Figure 3.3: Comparing numerical and ANN based solutions for λ and ν

We have run the system with the default configuration for the neural network configuration as provided by *NeuroDiffEq*. From the above plots we can see that in both the plot the two solutions are closely hugging together and thus *NueroDiffEq* seems to have solved the system of equations successfully giving us the Reissner–Nordström metric. To analyze further we can look at the residuals of λ and ν as well as the loss curve over the epochs.



(a) Plot for residue of λ and ν



(b) Loss during training

Figure 3.4: Residue Plot and Loss Plot

In the next section, we will try to do derive the metric for Kerr space-time.

3.3 Kerr metric

In this section, we will develop the mathematical machinery to arrive at the metric of a non-charged rotating blackhole. The mathematics behind the derivation for Kerr blackhole is different as compared to Schwarzschild and Reissner–Nordström metric.

Within the realm of gravitational metrics, a specific class known as "Kerr-Schild" metrics offers a valuable approach for linearization. By employing a suitable coordinate system, we can examine metrics characterized by the following components:

$$g_{\mu\nu} = \bar{g}_{\mu\nu} + l_{\mu\nu} \quad \text{with} \quad l_{\mu\nu} = f\left(x^{\rho}\right) l_{\mu} l_{\nu} \tag{3.36}$$

where $\bar{g}_{\mu\nu}$ is a metric of a known 'background' spacetime, $f\left(x^{\rho}\right)$ is an arbitrary function, and the vector field $l^{\mu} = \bar{g}^{\mu\nu} l_{\nu}$ is null and geodesic, that is, it satisfies

$$\bar{g}_{\mu\nu} l^{\mu} l^{\nu} = 0, \quad l^{\nu} \bar{D}_{\nu} l^{\mu} = 0 \tag{3.37}$$

where $\bar{D}$ is the covariant derivative associated with $\bar{g}_{\mu\nu}$. The calculation of the Ricci tensor of such a metric is a good exercise. We find

$$R^{\mu}_{\nu} = \bar{R}^{\mu}_{\nu} - l^{\mu\rho} \bar{R}_{\rho\nu} + \bar{D}_{\rho}\left(\bar{g}^{\mu\lambda} \Delta^{\rho}_{\nu\lambda}\right), \quad \text{where} \quad \Delta^{\mu}_{\nu\rho} = \frac{1}{2}\left(\bar{D}_{\nu} l^{\mu}_{\rho} + \bar{D}_{\rho} l^{\mu}_{\nu} - \bar{D}^{\mu} l_{\nu\rho}\right) \tag{3.38}$$

The curvature scalar reduces to

$$R = \bar{R} - l^{\mu\nu} \bar{R}_{\mu\nu} + \bar{D}_{\mu} V^{\mu}, \quad \text{where} \quad V^{\mu} = l^{\mu} \bar{D}_{\nu}\left(f l^{\nu}\right) \tag{3.39}$$

so that $\sqrt{-\bar{g}} \bar{D}_{\mu} V^{\mu} = \partial_{\mu}\left[l^{\mu} \partial_{\nu}\left(\sqrt{-\bar{g}} f l^{\nu}\right)\right]$.s

Therefore, given a background metric $\bar{g}_{\mu\nu}$ and a null geodesic vector l^{μ}, the equation $R = 0$ will determine the function f. The metric $g_{\mu\nu}$ is then known, but it still must be checked to see whether or not it is a solution of the vacuum equations, that is, if it is Ricciflat, $R^{\mu}_{\nu} = 0$.

To obtain the Kerr solution in Kerr-Schild coordinates, as the background metric we take the Minkowski metric in spheroidal coordinates:

$$d\bar{s}^2 = -dT^2 + \frac{\rho^2}{r^2+a^2}dr^2 + \rho^2 d\theta^2 + \left(r^2+a^2\right)\sin^2\theta d\varphi^2 \text{ with } \rho^2 \equiv r^2 + a^2\cos^2\theta \quad (3.40)$$

where a is a constant, and as the null geodesic vector we take

$$l^\mu = \left(1, -1, 0, \frac{a}{r^2+a^2}\right) \quad \Longrightarrow \quad l_\mu = \left(-1, -\frac{\rho^2}{r^2+a^2}, 0, a\sin^2\theta\right) \quad (3.41)$$

Solving the equation $R = 0$, we get,

$$f = \frac{\frac{c1\left(\frac{a^2\cos(2\theta)r^2}{2} + \frac{a^2r^2}{2} + \frac{r^4}{6}\right)}{2} + F2(\theta)r + F3(\theta)}{\rho^2} \quad (3.42)$$

We could argue that first term in the above equation will diverge at infinity if c1 is non-zero hence it must be zero. Thus we get that,

$$f = \frac{F2(\theta)r + F3(\theta)}{\rho^2} \quad (3.43)$$

Further, $F2(\theta)$ and $F3(\theta)$ could be further constrained using the boundary conditions. This work still in progress and implementation of the above set of equations into *NeuroDiffEq* is pending post which result will be added to this thesis.

Chapter 4

Conclusion and Future Scope

Our journey through the realm of Physics-Informed Neural Networks (PINNs) has showcased their remarkable prowess, spanning from tackling straightforward problems like Laplace equations to unraveling the complexities of chaotic dynamics in systems like the double pendulum. Moreover, our exploration has ventured into the profound domain of General Relativity, where we've endeavored to derive metrics for diverse properties of black holes.

We've observed that a non-spinning, uncharged black hole gives rise to the Schwarzschild metric, while a charged black hole yields the Reissner–Nordström metric. As we transition to more intricate metrics, such as the Kerr metric, characterized by two independent variables and a complex system of equations, we acknowledge that this remains a work in progress.

Nonetheless, PINNs exhibit their prowess in solving a diverse range of partial differential equation (PDE) systems even with nonlinear behavior. By imbuing the neural networks with an awareness of the underlying physical laws that govern the data, PINNs ensure that data points adhere to these laws, resulting in efficient and effective training that demands fewer data points.

As we look to the future, the scope of this thesis extends to a broader horizon. We aim to feed different modified gravity theories into PINNs, exploring whether these networks can accurately predict physical observables in these alternative theories. This journey holds the promise of advancing our understanding of gravitational physics and potentially unlocking novel insights into the universe's behavior under the influence of modified theories of gravity.

Chapter 5

Bibliography

[1] V. Mukhanov, "CMB-slow, or How to Estimate Cosmological Parameters by Hand," International Journal of Theoretical Physics, vol. 43, no. 3, pp. 623–668, Mar. 2004, doi: 10.1023/B:IJTP.0000048168.90282.db.
[2] "Improving Physics-Informed Neural Networks through Adaptive Loss Balancing — by Rafael Bischof — Towards Data Science." Accessed: Oct. 03, 2023. [Online]. Available: https://towardsdatascience.com/ improving-pinns-through-adaptive-loss-balancing-5566275
[3] F. Chen et al., "NeuroDiffEq: A Python package for solving differential equations with neural networks," JOSS, vol. 5, no. 46, p. 1931, Feb. 2020, doi: 10.21105/joss.01931.
[4] 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," Journal of Computational Physics, vol. 378, pp. 686–707, Feb. 2019, doi: 10.1016/j.jcp.2018.10.045.
[5] N. Deruelle and J.-P. Uzan, Relativity in modern physics, First edition. in Oxford graduate texts. Oxford, United Kingdom: Oxford University Press, 2018.
[6] A. Sunny, S. Xavier, and S. Shankaranarayanan, "Slowly rotating black hole solutions in $f(R)$ gravity: A need for enhancement of the no-hair conjecture." arXiv, Mar. 08, 2023. Accessed: Nov. 03, 2023. [Online]. Available: http://arxiv.org/abs/2303.04684
[7] L. D. Landau, E. M. Lifshitz, and L. D. Landau, The classical theory of fields, 4th rev. English ed. in Course of theoretical physics, no. v. 2. Oxford; New York: Pergamon Press, 1975.
[8] M. Visser, "The Kerr spacetime: A brief introduction." arXiv, Jan. 14, 2008. Accessed: Oct. 23, 2023. [Online]. Available: http://arxiv.org/abs/0706.0622
[9] A. S. Cornell, A. Ncube, and G. Harmsen, "Using physics-informed neural networks to compute quasinormal modes," Phys. Rev. D, vol. 106, no. 12, p. 124047, Dec. 2022, doi: 10.1103/PhysRevD.106.124047.