Dynamic System Discovery with Recursive Physics-Informed Neural Networks

Jarrod Mau
Utah State University

Follow this and additional works at: https://digitalcommons.usu.edu/etd

Part of the Mathematics Commons, and the Statistics and Probability Commons

Recommended Citation
Mau, Jarrod, "Dynamic System Discovery with Recursive Physics-Informed Neural Networks" (2022). All Graduate Theses and Dissertations. 8524.
https://digitalcommons.usu.edu/etd/8524

This Thesis is brought to you for free and open access by the Graduate Studies at DigitalCommons@USU. It has been accepted for inclusion in All Graduate Theses and Dissertations by an authorized administrator of DigitalCommons@USU. For more information, please contact digitalcommons@usu.edu.
DYNAMIC SYSTEM DISCOVERY WITH RECURSIVE PHYSICS-INFORMED NEURAL NETWORKS

by

Jarrod Mau

A thesis submitted in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

in

Mathematics

Approved:

Jia Zhao, Ph.D.
Major Professor

Zhaohu Nie, Ph.D.
Committee Member

Joe Koebbe, Ph.D.
Committee Member

D. Richard Cutler, Ph.D.
Vice Provost of Graduate Studies

UTAH STATE UNIVERSITY
Logan, Utah
2022
ABSTRACT

Dynamic System Discovery with Recursive Physics-Informed Neural Networks

by

Jarrod Mau, Master of Science
Utah State University, 2022

Major Professor: Jia Zhao, Ph.D.
Department: Mathematics and Statistics

How to identify mathematical models from existing data has been an active research topic for decades. This thesis focuses on identifying dynamical systems from data. In particular, it presents a data driven recursive physics informed neural network to learn the right hand side of dynamical system. The novelty is to add recursive steps in the baseline physics informed neural networks to decrease the local truncation error. Adding the recursive step improves accuracy of the learned right hand side. To illustrate the idea, we show the theoretical superiority of the recursive approach. Then, we also present several numerical examples to demonstrate the proved properties.

(41 pages)
PUBLIC ABSTRACT

Dynamic System Discovery with Recursive Physics-Informed Neural Networks

Jarrod Mau

This thesis presents a novel method, recursive Physics informed neural network, to learn the right hand side of differential equations. The neural network takes in data, then trains, and then acts as a proxy for the differential equation which can be used for modeling. We show the theoretical superiority of the recursive approach. We also use computer simulations to demonstrate the proved properties.
This thesis is dedicated to my wife, Alexis.
ACKNOWLEDGMENTS

Jia Zhao has been the perfect mentor for me. The process of going from undergrad to graduate student has been challenging to say the least. I’m grateful to have Jia Zhao as my advisor and look forward to continue working with him as I pursue my PhD. He has provided me with the tools, literally and figuratively, to accomplish this thesis. I’d like to thank Dr. Joe Kobbe and Dr. Zhaohu Nie for guiding me and giving me invaluable advice. I’m grateful for all the time they’ve spend helping me shape my thesis into what it is now. Finally I’d like to thank Mr. Gary Tanner for everything he does for me and for the department. I wouldn’t be here without him.

Jarrod Mau
CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Introduction</td>
<td>1</td>
</tr>
<tr>
<td>1.1</td>
<td>A brief introduction of deep neural network</td>
<td>1</td>
</tr>
<tr>
<td>1.2</td>
<td>Some existing works using neural networks</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>Recursive Physics-informed Neural Networks for Model Discovery</td>
<td>5</td>
</tr>
<tr>
<td>2.1</td>
<td>Research question</td>
<td>5</td>
</tr>
<tr>
<td>2.2</td>
<td>Using physics-informed neural networks for model discovery</td>
<td>5</td>
</tr>
<tr>
<td>2.3</td>
<td>Using Recursive Physics-Informed Neural Networks for Model Discovery</td>
<td>6</td>
</tr>
<tr>
<td>2.3.1</td>
<td>Recursive forward Euler method</td>
<td>8</td>
</tr>
<tr>
<td>2.3.2</td>
<td>Recursive Runge-Kutta method</td>
<td>8</td>
</tr>
<tr>
<td>2.3.3</td>
<td>Recursive Taylor series method</td>
<td>9</td>
</tr>
<tr>
<td>2.3.4</td>
<td>Recursive backward Euler method</td>
<td>9</td>
</tr>
<tr>
<td>2.3.5</td>
<td>Implicit method</td>
<td>9</td>
</tr>
<tr>
<td>2.3.6</td>
<td>Remarks</td>
<td>10</td>
</tr>
<tr>
<td>3</td>
<td>Dynamical System Discovery using Synthetic Data</td>
<td>12</td>
</tr>
<tr>
<td>3.1</td>
<td>Two-dimensional damped harmonic oscillator</td>
<td>13</td>
</tr>
<tr>
<td>3.2</td>
<td>Glycolytic oscillator</td>
<td>16</td>
</tr>
<tr>
<td>3.3</td>
<td>Hopf bifurcation</td>
<td>20</td>
</tr>
<tr>
<td>3.4</td>
<td>Multiscale System</td>
<td>23</td>
</tr>
<tr>
<td>3.5</td>
<td>Lorenz system</td>
<td>26</td>
</tr>
<tr>
<td>4</td>
<td>Conclusion</td>
<td>30</td>
</tr>
</tbody>
</table>
LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>relative $l^2$ errors for two-dimensional damped harmonic oscillator in (3.1) calculated with the learned model with 10000 epochs.</td>
<td>13</td>
</tr>
<tr>
<td>3.2</td>
<td>$l^2$ error of the solutions predicted by the learned model for the glycolytic oscillator systems in (3.2) for 20000 epochs</td>
<td>17</td>
</tr>
<tr>
<td>3.3</td>
<td>$l^2$ error of the predicted solutions using the learned model with different settings for the parametrized model in (3.3) with 20000 epochs.</td>
<td>21</td>
</tr>
<tr>
<td>3.4</td>
<td>$l^2$ error of the predicted solutions using the learned model with different settings for the model in (3.4) with 10000 epochs.</td>
<td>24</td>
</tr>
<tr>
<td>3.5</td>
<td>$l^2$ error of the predicted solutions using the learned model with different settings for the model in (3.5) with 10000 epochs.</td>
<td>27</td>
</tr>
</tbody>
</table>
## LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>Here we illustrate an example neural network. This neural network has a two dimensional input and two dimensional output. We see two hidden layers, the first with 5 nodes and the second with 3 nodes. $b_k^l$ is the bias that is added to the $k^{th}$ neuron in the $l^{th}$ layer. $w_{jk}^{l-1}$ is the weight from the $k^{th}$ neuron in the $(l-1)^{th}$ layer to the $j^{th}$ neuron in the $l^{th}$ layer. $a_k^l$ is the activation (i.e. after the bias, weights, and activation function has been applied) of the $k^{th}$ neuron in the $l^{th}$ layer.</td>
</tr>
<tr>
<td>3.1</td>
<td>Model discovery of the two-dimensional damped harmonic oscillator with $\Delta t = 0.1$ and various $M$. Left: $M=1$, Middle: $M=5$, Right: $M=20$.</td>
</tr>
<tr>
<td>3.2</td>
<td>Model discovery of the two-dimensional damped harmonic oscillator with $\Delta t = 0.2$ and various $M$. Left: $M=1$, Middle: $M=5$, Right: $M=20$.</td>
</tr>
<tr>
<td>3.3</td>
<td>Model discovery of the two-dimensional damped harmonic oscillator with $\Delta t = 0.4$ and various $M$. Left: $M=1$, Middle: $M=5$, Right: $M=20$.</td>
</tr>
<tr>
<td>3.4</td>
<td>Model discovery for the glycolytic oscillator with $\Delta t = 0.05$ and various $M$. Left: $M=1$, Middle: $M=18$, Right: $M=25$. Note the real dynamics is the same for all three cases but shown on different scales to encapsulate the learned dynamics.</td>
</tr>
<tr>
<td>3.5</td>
<td>Model discovery for the glycolytic oscillator with $\Delta t = 0.1$ and various $M$. Left: $M=1$, Middle: $M=18$, Right: $M=25$. Note the real dynamics is the same for all three cases but shown on different scales to encapsulate the learned dynamics.</td>
</tr>
<tr>
<td>3.6</td>
<td>Model discovery for the Hopf bifurcation with $\Delta t = 1$ and various $M$.</td>
</tr>
<tr>
<td>3.7</td>
<td>Model discovery for the Hopf bifurcation with $\Delta t = 2$ and various $M$.</td>
</tr>
<tr>
<td>3.8</td>
<td>Model discovery of a multiscale dynamical system with $\Delta t = 0.1$. Left: $M=1$, Middle $M=10$, Right: $M=20$.</td>
</tr>
<tr>
<td>3.9</td>
<td>Model discovery of a multiscale dynamical system with $\Delta t = 0.25$. Left: $M=1$, Middle $M=10$ Right: $M=20$.</td>
</tr>
<tr>
<td>3.10</td>
<td>Model discovery of the Lorenz system with $\Delta t = 0.05$ and various $M$. Left: $M=1$, Right: $M=20$.</td>
</tr>
<tr>
<td>3.11</td>
<td>Model discovery of the Lorenz system with $\Delta t = 0.1$ and various $M$. Left: $M=1$, Right: $M=20$.</td>
</tr>
</tbody>
</table>
1.1 A brief introduction of deep neural network

Computer were traditionally used as algorithm processors [14, 15, 17]. Computers can perform these algorithms much faster than humans, yet only using computers as algorithm machines only scratches the surface of a computers’ potential. The advancements in machine learning has progressed research in many fields and has a wide scope of uses [3, 4, 15, 18, 21]. Machine learning has been used in self driving cars, facial recognition, search engine result refining, chemical compound discovery, and even playing complex games like chess and go [2, 6, 9, 13]. Machine learning has been used in many fields but its application is far from being exhausted.

Mathematically, a feed-forward neural network can be described as a composition of non-linear functions. So, for a vector input $x$, we can denote a general forward feed neural network as

$$a^{[1]} = x \in \mathbb{R}^{n_1},$$
$$a^{[k]} = \sigma(W^{[k]}a^{[k-1]} + b^{[k]}) \in \mathbb{R}^{n_k}, \quad \text{for} \ k = 2, 3, \cdots, L, \quad (1.1)$$

where $W^{[k]} \in \mathbb{R}^{n_k \times n_{k-1}}$ and $b^{[k]} \in \mathbb{R}^{n_k}$ are the weights and biases at layer $k$, and $\sigma$ is the activation function [11]. We used tanh as our activation function in this thesis. An alternative was the rectifier function which would have produced a quicker computation but we used tanh because tanh’s higher order derivatives aren’t trivial which what we’d see in complex systems. At layer $k$ there are $n_k \ast n_{k-1} + n_k$ free parameters from $W^k$ and $b^k$. A schematic feed-forward neural network is shown in Figure 1.1.
We now explain how a neural network is trained. A neural network typically initializes with random weights and biases. So whatever objective function your neural network is trying to emulate probably won’t do well at first. To address this we need a cost function. This thesis uses a Quadratic cost function. A Quadratic cost function takes the squared difference of the output with the desired output \[11\]. The network then attempts to minimize the cost function by tuning the weights and biases through backpropagation. To understand backpropagation we need to introduce some notation.

In other words:

\[
a^l_k = \sigma(\Sigma_k(w_{jk}^la_{l-1}^k + b^l_k)),
\]

\[
Cost C = C(a^L_k),
\]

where \(L\) is the last layer of the neural network. Now we tune the weights and biases of the last layer of the network based on their derivatives relative to the cost function. These derivatives are given by

\[
\frac{\delta C}{\delta b^L_k} = \frac{\delta C}{\delta a^L_k} \sigma'(\Sigma_k(w_{jk}^L a_{l-1}^L + b^L_k)),
\]

\[
\frac{\delta C}{\delta w^L_{jk}} = \frac{\delta C}{\delta a^L_k} \sigma'(\Sigma_k(w_{jk}^L a_{l-1}^L + b^L_k))(a_{l-1}^L). \tag{1.3}
\]
This process is repeated for the \((l - 1)\text{th}\) layer treating the derivative of the cost function with respect to the activation in the \(l\text{th}\) layer as the cost function. We then update the weights and biases by subtracting their derivatives multiplied by a constant. This constant is called the learning rate [11]. The training data is then re-entered into the neural network and then into the cost function. The weights and biases are updated again. Every iteration is called an epoch [11]. Our goal is to minimize our cost function, thereby minimizing the difference between the output of the neural network and the desired output. A learning rate that is too large will overshoot the minimum and a learning rate that is too small will take too long to reach the minimum. Often a variable learning rate is used [11]. A variable learning rate will adjust over each epoch and will eventually decay to zero [11].

1.2 Some existing works using neural networks

Systems in physics and biology are governed by differential equations. We’ve seen a few approaches in the literature. We initially saw [15] showed that deep neural networks can learn the right hand side of a governing differential equation given some data. They named their method a physics informed neural network, or PINN for short. Their formulation was straightforward but the results were promising. [20] then improved on the technique utilizing more advanced and higher order numerical methods which improved the accuracy. Then [7, 18] used these techniques on high dimensional PDEs and [9] did the same but on noisy data. Meanwhile, [12] and [1] used a PINN to learn the right hand side of a differential equation with partially observed systems. [10] combined the ideas of [12] and [9] to learn chaotic dynamics from noisy and partial data sets. Another approach used by [15] was to determine the coefficients of a PDE model by a PINN. Furthermore, using Sparse Identification of Nonlinear Dynamics, SINDy approach can identify the exact right hand side of some differential equations [4, 8]. You can also use neural networks to learn the parameters of unknown dynamical systems [19] using synthetic data and using real data [21] in biology. Flow mapping, where the neural network can indicate the next state \((\Delta t \text{ later})\) based on the current state based on data is an alternative approach [14].

Existing approaches to model discovery have strong requirements. We introduce in this
thesis a novel recursive PINN that has a smaller local truncation error for approximating
the integral thereby allowing for longer time lags between data points. Our structure also
allows for non-uniform time lags between data points.
CHAPTER 2
Recursive Physics-informed Neural Networks for Model Discovery

2.1 Research question

Let us first formulate our data-driven model discovery problem. We consider the unknown autonomous parameterized dynamical system

\[ y_t = F(y, \mu), \quad (2.1) \]

with \( F \) as the right hand side functional and \( \mu \) as the parameters. Assume we have some data pairs

\[ \{(y^1_j, y^2_j, \Delta t_j, \mu_j)\}_{j=1}^{N} \quad (2.2) \]

available to us. So \( y^1_j \) and \( y^2_j \) are solutions to (2.1) with time lag \( \Delta t_j \) and the parameter \( \mu_j \).

Our parameterized problem can be restated as a non-parameterized problem by defining a new variable \( \Psi = \begin{bmatrix} y \\ \mu \end{bmatrix} \) and function \( F = \begin{bmatrix} F(\Psi) \\ 0 \end{bmatrix} \). So \( y_t = F(y, \mu) \) with data points \( \{(y^1_j, y^2_j, \Delta t_j, \mu_j)\}_{j=1}^{N} \) becomes \( \Psi_t = F(\Psi) \) with data points \( \{(\Psi^1_j, \Psi^2_j, \Delta t_j)\}_{j=1}^{N} \). Our goal is to develop a neural network that will act as a black box function with inputs \( \Psi \) and outputs \( F(\Psi) \).

2.2 Using physics-informed neural networks for model discovery

We will use a feed forward neural network

\[ N_{\theta} : (\Psi; \theta) \rightarrow N_{\theta}(\Psi; \theta) \quad (2.3) \]
to approximate the right hand side of the ODE, $F$, where $\theta$ represents the weights and biases for which $N_F$ can be tuned. For the sake of simplicity, we’ll introduce

$$
N_F : \Psi \rightarrow N_F(\Psi) \quad (2.4)
$$

that has the same weights and biases as $N_g$. Since $N_F$ is an approximation of $F$ we have $\Psi_t = N_F(\Psi)$ and from the fundamental theorem of calculus we know that

$$
\Psi^2_j = \Psi^1_j + \int_0^{\Delta t_j} N_F(\Psi) dt.
$$

(2.5)

Since the integral of $N_F$ is unknown, numerical approximations will be used. We can approximate the integral using left-hand rectangles using the formula $\Psi^2_j = \Psi^1_j + \Delta t_j N_F(\Psi^1_j) + O(\Delta t^2_j)$. From here, we can define our residual function $N_r$ as

$$
N_r : (\Psi^1_j, \Delta t_j) \rightarrow \Psi^2_j - \Psi^1_j - \Delta t_j N_F(\Psi^1_j). \quad (2.6)
$$

Now that we have $N_F$ and $N_r$ defined, to approximate (2.1) we formalize our problem as

$$
\text{Find } N_F(\Psi; \theta) \text{ such that } \min_{\theta \in N_F} \sum_{j=1}^{N} \|N_r(\Psi^1_j, \Delta t_j)\|^2. \quad (2.7)
$$

If the residual function is small, mathematically, this forces $N_F$ to be a good approximation of $F$ in (2.1). Unfortunately this formulation hinges on $\Delta t_j N_F(\Psi^1_j)$ being a good approximation of the integral which means $\Delta t_j$ must be small. This constraint on our data can be problematic. An experiment may have limited sampling capabilities or resource limitations making $\Delta t_j$ large. To address this, we introduce a recursive physics informed neural network with a modified $N_r$.

2.3 Using Recursive Physics-Informed Neural Networks for Model Discovery

Here we introduce the recursive physics-informed neural network (recursive PINN). We acknowledge the formula (2.6) where $\Delta t_j N_F(\Psi^1_j)$ approximates the integral. From a differ-
ent perspective though $\Psi_j^1 + \Delta t_j N_F(\Psi_j^1)$ approximates $\Psi_j^2$. If $N$ is a good approximation of the right hand side of the differential equation and $\Delta t_j$ is small, then $\Psi_j^1 + \Delta t_j N_F(\Psi_j^1)$ approximates $\Psi_j^2$ well. However, it presents a problem if $\Delta t_j$ is large. To address this problem we discretize the time interval into $M$ segments. Let

$$h = \frac{\Delta t}{M}. \quad (2.8)$$

For each of the methods we outlined in the next subsection $N_r$ will be defined as

$$N_r : (\Psi_j^1, \Psi_j^2, \Delta t_j) \rightarrow \Psi_j^2 - \Psi_{j,M} \quad (2.9)$$

where $\Psi_{j,M}$ is our approximation of $\Psi_j^2$ by means of existing numerical approaches. We use initial information $\Psi_j^1$ and our neural network $N_F$ acts as the right hand side of the differential equation. The idea of this thesis is to use existing numerical approaches of solving differential equations numerically to get a better approximation of $\Psi_j^2$. Our problem is still

$$\text{Find } N_F(\Psi) \text{ such that } \arg \min_{\theta} \sum_{j=1}^{N} \|N_r(\Psi_j^1, \Psi_j^2, \Delta t_j)\|^2. \quad (2.10)$$

Again, our goal is to learn the right hand side of the ODE, $F$. Here we outline the theoretical advantage of using the recursive approach. Assume we have two neural network structures $N^1$ and $N^M$ both with structure from (2.4) attempting to solve the same ODE system given some data (2.2). Assume both $N^1$ and $N^M$ approximate $F$ perfectly. $N^1$ uses $N^1_r$ from (2.9) and $\Psi_{j,M}$ from (2.12) with $M = 1$. Since $N^1$ approximates $F$ perfectly, the local truncation error for $\Psi_j^2$ would be $O(\Delta t^4)$. This can present a problem since the neural network will then adjust $\theta$ to satisfy (2.10) making $N^1$ a worse approximation of $F$. It follows that for our neural network to have a chance to approximate $F$ well, we need to create a neural network with a local truncation error for $\Psi_j^2$ as small as possible. In theory, we could make $\Delta t$ smaller when collecting the data, however, this may not be practical due to the nature of the dynamics or due to experimental costs. Now let’s look at $N^M$. We use the same formulation as $N^1$ but with $M = M$. The local truncation error for $\Psi_{j,1}$ is
\( O(\frac{\Delta t^4}{M^4}) \). The local truncation error for \( \Psi_j^2 \) is \( MO(\frac{\Delta t^4}{M^4}) = O(\frac{\Delta t^4}{M^4}) \). This can be a major improvement. This thesis will demonstrate this in practice in Chapter 3.

2.3.1 Recursive forward Euler method

Utilizing the forward Euler method, \( \Psi_{j,M} \) is found by the following recursive formula,

\[
\begin{align*}
\Psi_{j,0} &= \Psi_j^1, \\
\Psi_{j,s+1} &= \Psi_{j,s} + hN_f(\Psi_{j,s}) \quad s = 0, 1, \cdots, M - 1.
\end{align*}
\]

(2.11)

Notice that this formulation of a recursive PINN is equivalent to the traditional PINN when \( M = 1 \).

2.3.2 Recursive Runge-Kutta method

The four stage Runge Kutta method out competes the forward Euler method by its reduced local truncation error. We can therefore introduce a higher order approximation for the integral inspired by the Runge-Kutta method.

Here \( \Psi_{j,M} \) is found by the recursive formula

\[
\begin{align*}
\Psi_{j,0} &= \Psi_j^1, \\
\Psi_{j,s+1} &= \Psi_{j,s} + \frac{h}{6}(K_{s1} + K_{s2} + K_{s3} + K_{s4}), \\
K_{s1} &= N_F(\Psi_{j,s}), \\
K_{s2} &= N_F(\Psi_{j,s} + \frac{h}{2}K_{s1}), \\
K_{s3} &= N_F(\Psi_{j,s} + \frac{h}{2}K_{s2}), \\
K_{s4} &= N_F(\Psi_{j,s} + hK_{s3}), \quad s = 0, 1, \cdots, M - 1.
\end{align*}
\]

(2.12)

This is the method we use in the code for this thesis. As we’ll see, the lower truncation error for estimating the integral provides better results (especially for larger \( \Delta t \) tested) in learning the dynamics. Note when \( M=1 \), this method is the method used by [20].
2.3.3 Recursive Taylor series method

The Taylor series method can also be used. We note that the Taylor series method illustrated below can have a small local truncation error but requires the function we are considering to have high order differentiability. Here $\Psi_{j,M}$ is found by the recursive formula

\[
\begin{align*}
\Psi_{j,0} &= \Psi_{j}^1, \\
\Psi_{j,s+1} &= \Psi_{j,s} + h \Psi'_j(t_s) + \frac{h^2}{2!} \Psi''_j(t_s) + \ldots \\
\Psi'_j(t_s) &= N_f(\Psi_{j,s}), \\
\Psi''_j(t_s) &= N'_f(\Psi_{j,s}) \Psi'_j(t_s) = N'_f(\Psi_{j,s}) N_f(\Psi_{j,s}).
\end{align*}
\tag{2.13}
\]

The formulation above could be expanded further by using more terms of the Taylor series to have an even smaller local truncation error. But doing so would require stronger assumptions about the differentiability of the system (i.e., the system would need to have higher order differentiability).

2.3.4 Recursive backward Euler method

Backward Euler can also be used. This implicit method has the benefit of enhanced stability over Forward Euler. Here $\Psi_{j,M}$ is found by the recursive formula

\[
\begin{align*}
\Psi_{j,0} &= \Psi_{j}^1, \\
\Psi_{j,s+1} &= \Psi_{j,s} + h N_f(\Psi_{j,s+1}), \quad s = 0, 1, \ldots, M - 1.
\end{align*}
\tag{2.14}
\]

The formulation above poses a problem. There is no obvious method to solve directly for $\Psi_{j,s+1}$. To approximate $\Psi_{j,s+1}$ we let $g(\Psi) = \Psi - \Psi_{j,s} - h N_f(\Psi)$. So $\frac{dg}{d\Psi} = 1 - h \frac{dN_f}{d\Psi}(\Psi)$. With initial guess of $\Psi_{j,s} = \Psi_{j,s+1}$ we can use Newton’s method to approximate the root of $g(\Psi)$. Notice the root of $g$ solves the equation $\Psi_{j,s+1} = \Psi_{j,s} + h N_f(\Psi_{j,s+1})$ where $\Psi_{j,s+1}$ is the unknown. We remark the method above is still an explicit formulation.

2.3.5 Implicit method

A truly implicit method could improve the neural networks effectiveness. Constructing
the neural network based on the backward Euler method without using Newton’s method to approximate $\Psi_{j,s+1}$ could prove useful. Constructing such a neural network is outside the scope of the thesis but interested readers are encouraged to look into this. Since numerical methods form a broad subject, we note there are many other recursive neural network designs other than the four illustrated in the previous subsections.

### 2.3.6 Remarks

We note a few possibilities for adding a recursive step in a PINN model but we note there are many more than mentioned here. Each one is something that is worth studying for learning the dynamics. For the purposes of testing, we chose the Runge-Kutta method as it had one of the lowest local truncation errors. The lower local truncation error, we theorize, will have better results. For large $t$ the error associated with approximating the integral with the traditional PINN increases thereby limiting the application of the traditional PINN model to only situation where large amounts of data is available with small $\Delta t$ between data points. Below we show pseudo code of the methods outlined above.

```
1 def net_f(self, u0):
2     return self.neural_net(u0, self.weights, self.biases)
3
4 def net_f_u(self, u0):
5     f = self.net_f(u0)
6     f_u = tf.gradients(f, u0)[0]
7     return f_u
8
9 def net_taylor(self, u0, dt):
10    return u0 + dt * self.net_f(u0) + dt ** 2 * self.net_f_u(u0) * self.net_f(u0)/2
11
12 def net_forward_Euler(self, u0, dt):
13    return u0 + dt * self.net_f(u0)
14
15 def net_g(self, u0, u1, dt):
16    return u1 - u0 - dt * self.net_f(u1)
17
18 def net_g_prime(self, u, u_t1, dt):
19    return 1 - dt * self.net_f_u(u_t1)
20
21 def net_backward_Euler(self, u0, dt):
22    u1 = u0
23    error = 1
24    epsilon = 10**-8
25    while error > epsilon:
26          
```
Listing 2.1: Definition of Recursive Physics-informed Residual Neural Networks
CHAPTER 3
Dynamical System Discovery using Synthetic Data

In this section, we show the effectiveness of our approach. We will use well known ODE systems to assess the recursive PINN efficacy in uncovering dynamical systems from data. In this thesis, we will consider 5 ODE autonomous systems with uniform time lag between data $\Delta t$. The numerical tests are conducted in the following steps:

- Using the solution of a classical dynamical system we generate synthetic data for which we can compare. We randomly sampled for the initial conditions $\{\Psi^1_j\}_{j=1}^N \subset \Omega$ in a fixed sampling domain $\Omega$. We then use the solution to the dynamical system to solve for $\Psi^2_j$. After this process the sampling data are in the format of $\{(\Psi^1_j, \Psi^2_j, \Delta t)\}_{i=1}^N$.

- We then construct the physics informed neural network $\mathcal{N}_F$ and the physics-informed recursive neural network $\mathcal{N}_r$. We then attempt to optimize (2.7).

- We then train the neural network on a training set. After several thousand iterations of fine tuning the parameters in the neural network, $\mathcal{N}_F$ will be a good approximation to $\mathcal{F}$. The neural network will act then as a black box function $\Psi_t = \mathcal{N}_F(\Psi)$ and be compared with the real solution using the traditional $l^2$ norm.

We note that fine tuning the hyper-parameters of our network is outside our current interests. So we fix the hyper-parameters thereby fixing the structure of our neural network. We use a feed-forward neural network with one hidden layer of 128 neurons. We use tanh as the activation function. For the recursive step, we used the Runge-Kutta inspired structure defined in (2.12). We use the Adam method and L-BFGS method of optimization.
3.1 Two-dimensional damped harmonic oscillator

We start with the relatively simple two-dimensional damped harmonic oscillator with cubic reactive dynamics. The dynamics are governed by the following system of equations

\[ \dot{X} = \begin{bmatrix} -0.1 & 2.0 \\ -2.0 & -0.1 \end{bmatrix} X^3, \quad \text{where} \ X = \begin{bmatrix} x \\ y \end{bmatrix}. \] (3.1)

We then pick the domain \( \Omega = [-2.5, 2.5] \times [-2.5, 2.5] \), and sample \( N = 2000 \) points from \( \Omega \) as \( \{\Psi_j\}_{j=1}^N \) using the Latin hypercube sampling method. We then calculate \( \{\Psi_j\}_{j=1}^N \) using the real solution and the corresponding time lag \( \Delta t \). After training we randomly choose an initial condition \( X = [2, 0]^T \). We then use the discovered model to calculate the dynamics for \( t \in [0, 25] \).

The relative \( l^2 \) errors for the model for the corresponding \( M \) and \( \Delta t \) are given in Table 3.1. We see the highest error corresponds with a larger \( \Delta t \) using the classical method (i.e. \( M=1 \)). Having a greater error associated with a higher \( \Delta t \) is no surprise. However, adding more recursive steps, thereby filling in the information between points, reduces the error for every case of \( \Delta t \). We note the trade off of more recursive steps is computation time. The trade off is good when considering the improved accuracy.

<table>
<thead>
<tr>
<th>( \Delta t )</th>
<th>( M=1 )</th>
<th>( M=5 )</th>
<th>( M=20 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \Delta t = 0.1 )</td>
<td>0.1578</td>
<td>0.0099</td>
<td>0.0036</td>
</tr>
<tr>
<td>( \Delta t = 0.2 )</td>
<td>1.1132</td>
<td>0.0064</td>
<td>0.0024</td>
</tr>
<tr>
<td>( \Delta t = 0.4 )</td>
<td>0.6817</td>
<td>0.1505</td>
<td>0.0081</td>
</tr>
</tbody>
</table>

We see in Table 3.1 with 20 recursive steps the recursive PINN model has the smallest \( l^2 \) error. To better understand the local truncation errors, we provide the graphs associated with each case for both dimensions.
We see in Figure 3.1 the learned dynamics matches fairly well with the true dynamics in all three cases. We must keep in mind though that this is for the smallest $\Delta t$ tested and therefore the local truncation error is also minimal in all three cases. That being said, it is also clear that the recursive approach is marginally superior in learning the dynamics (i.e. $M=5$ and $M=20$).
For Figure 3.2 we have a larger $\Delta t$ than the previous example. We see the traditional method (i.e. $M=1$) retains some information related to the dynamics such as the oscillating nature and the range. However, we see the learned dynamics diverges from the real dynamics after $t = 1$.

For Figure 3.3 we have a larger $\Delta t$ than the previous example. We see the traditional method (i.e. $M=1$) retains some information related to the dynamics such as the oscillating nature and the range. However, we see the learned dynamics diverges from the real dynamics after $t = 1$. 

Fig. 3.2: Model discovery of the two-dimensional damped harmonic oscillator with $\Delta t = 0.2$ and various $M$. Left: $M=1$, Middle: $M=5$, Right: $M=20$.

Fig. 3.3: Model discovery of the two-dimensional damped harmonic oscillator with $\Delta t = 0.4$ and various $M$. Left: $M=1$, Middle: $M=5$, Right: $M=20$. 
For Figure 3.3 we test $\Delta t = 0.4$ which is relatively large for this problem. We see the traditional PINN does not approximate the dynamics well. We see the recursive approach does approximate the dynamics with the recursive PINN with the largest number of recursive steps fitting the real dynamics the best. This makes intuitive sense as for larger $\Delta t$s the difference between the local truncation errors for the traditional PINN and the recursive PINN is larger.

We observe reduced error in both dimensions with the recursive approach. This is particularly apparent for larger $\Delta t$ values. We hypothesise this is due to the lower local truncation error involved with approximating $\Psi_j^2$ when using the recursive approach. The most interesting point is for larger $\Delta t$’s, the only highly successful approach was the recursive approach with a larger number of recursive steps. This suggests that even with a large $\Delta t$, we can select a sufficiently large number of recursive steps in our recursive PINN model and still successfully learn the dynamics. We consider the recursive PINN model a success for the first ODE system investigated.

### 3.2 Glycolytic oscillator

Next, we consider a glycolytic oscillator which can be defined with 7 coupled ODEs [5], for the concentration of 7 biochemical species, given by

$$
\begin{align*}
\frac{dS_1}{dt} &= J_0 - \frac{k_1 S_1 S_6}{1 + (S_6/K_1)^q}, \\
\frac{dS_2}{dt} &= 2 \frac{k_1 S_1 S_6}{1 + (S_6/K_1)^q} - k_2 S_2(N - S_5) - k_6 S_2 S_5, \\
\frac{dS_3}{dt} &= k_2 S_2(N - S_5) - k_3 S - 3(A - S_6), \\
\frac{dS_4}{dt} &= k_3 S_3(A - S_6) - k_4 S_4 S_5 - \kappa(S_4 - S_7), \\
\frac{dS_5}{dt} &= k_2 S_2(N - S_5) - k_4 S_4 S_5 - k_6 S_2 S_5, \\
\frac{dS_6}{dt} &= -2 \frac{k_1 S_1 S_6}{1 + (S_6/K_1)^q} + 2 k_3 S_3(A - S_6) - k_5 S_6, \\
\frac{dS_7}{dt} &= \psi_k(S_4 - S_7) - k_i S_7.
\end{align*}
$$

We use coefficients $J_0 = 2.5$, $k_1 = 100$, $k_2 = 6$, $k_3 = 16$, $k_4 = 100$, $k_5 = 1.28$, $k_6 = 12$, $k = 1.8$, $\kappa = 13$, $q = 4$, $K_1 = 0.52$, $\psi = 0.1$, $N = 1$, and $A = 4$ [5]. Note (3.2) is an oscillatory
system. This system, arguably more complex than the first ODE system investigated, will provide more information on how well the recursive PINN Model performs.

Similarly to the first two-dimensional damped harmonic oscillator, we then choose the sampling domain \( \Omega = [0, 2] \times [0, 3] \times [0, 0.5] \times [0, 0.5] \times [0, 0.5] \times [0, 0.05, 0.15] \), and using Latin hypercube sampling to generate 2000 points as \( \{\Psi_1^1\}_{j=1}^N \), and calculate \( \{\Psi_2^2\}_{j=1}^N \) from (3.2). We then train the neural network for 10000 iterations. The resulting trained neural network will act as the learned model \( \Psi_t = N_f(\Psi) \). We then pick a random initial condition \( [1.1, 1.0, 0.075, 0.175, 0.25, 0.9, 0.095] \) to calculate solutions in the time interval \([0, 5]\).

Table 3.2: \( l^2 \) error of the solutions predicted by the learned model for the glycolytic oscillator systems in (3.2) for 20000 epochs

<table>
<thead>
<tr>
<th></th>
<th>M=1</th>
<th>M=18</th>
<th>M = 25</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \Delta t = 0.05 )</td>
<td>3.6932</td>
<td>0.3531</td>
<td>0.0981</td>
</tr>
<tr>
<td>( \Delta t = 0.1 )</td>
<td>13.9863</td>
<td>0.3523</td>
<td>0.0506</td>
</tr>
</tbody>
</table>

Table 3.2 shows general improvement in accuracy for the recursive PINN method over the traditional PINN. The error appears to be inversely correlated with the number of recursive steps. Furthermore, the \( l^2 \) error is close to zero when M=25 in both \( \Delta t \) cases.
Fig. 3.4: Model discovery for the glycolytic oscillator with $\Delta t = 0.05$ and various $M$. Left: $M=1$, Middle: $M=18$, Right: $M=25$. Note the real dynamics is the same for all three cases but shown on different scales to encapsulate the learned dynamics.
Fig. 3.5: Model discovery for the glycolytic oscillator with $\Delta t = 0.1$ and various $M$. Left: $M=1$, Middle: $M=18$, Right: $M=25$. Note the real dynamics is the same for all three cases but shown on different scales to encapsulate the learned dynamics.
In Figure 3.4 we see gradual improvements for the PINN to learn the dynamics when M increases. For the traditional PINN, the learned dynamics doesn’t resemble the real dynamics at all. With 18 recursive steps, the learned dynamics roughly corresponds with the real dynamics in the first 6 dimensions. With 25 recursive steps, we see the learned dynamics correspond with the real dynamics in all 7 dimensions. We again see steady improvements in both choices of $\Delta t$ as we take greater Ms in Figure 3.5. This is very promising results since as researchers we don’t always get to choose a sufficiently small $\Delta t$ to use in modeling our problem. We saw the recursive PINN method perform well for the two-dimensional damped harmonic oscillator and now we see it perform well for this higher dimensional ODE system too. This is very promising when assessing the recursive PINN model.

3.3 Hopf bifurcation

In the previous 2 examples, we’ve shown the recursive neural network has a smaller error than the traditional method. This shows the recursive neural network can accurately approximate the kinetics of non-linear autonomous dynamical systems. We have not seen a parameterized dynamical system in the examples so far. We consider the following parameterized nonlinear dynamical system, which was used in [16]:

$$\begin{align*}
\dot{x} &= \mu x + y - x(x^2 + y^2), \\
\dot{y} &= -x + \mu y - y(x^2 + y^2),
\end{align*}$$

where $\mu$ is a parameter.

We fix domain $\Omega = [-1, 1] \times [-2, 2] \times [-1, 1]$, and use Latin hypercube sampling to randomly sample 2000 points from $\Omega$ as $\{\Psi_j^1\}_{i=1}^N$, and calculate the corresponding $\{\Psi_j^2\}_{i=1}^N$ from (3.3). Once the model is trained, we utilize the neural network to calculate the solution for $t \in [0, 75]$.

The $l^2$ errors for the trained model with various $\Delta t$ and M recursive steps are illustrated in Table 3.3. We see improvement from the single step method in both the $\Delta t = 1$ and the $\Delta t = 2$ cases. The traditional PINN method (single step) particularly struggles to
capture the dynamics in the $\Delta t = 2$ case. This is no surprise as $\Delta t = 2$ is a rather large $\Delta t$ for the purposes of modeling. However, the recursive approach does capture the dynamics accurately.

Table 3.3: $l^2$ error of the predicted solutions using the learned model with different settings for the parametrized model in (3.3) with 20000 epochs.

<table>
<thead>
<tr>
<th></th>
<th>M=1</th>
<th>M = 5</th>
<th>M = 10</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\Delta t = 1.0$</td>
<td>0.1929</td>
<td>0.0086</td>
<td>0.0161</td>
</tr>
<tr>
<td>$\Delta t = 2.0$</td>
<td>0.8930</td>
<td>0.0192</td>
<td>0.0101</td>
</tr>
</tbody>
</table>

Table 3.3 shows the largest $l^2$ errors for the traditional PINN network. The traditional PINN network has the largest error for $\Delta t = 2$ which is what we postulated in chapter two of this thesis. But we see the $l^2$ error nearly vanishes with the recursive PINN.
Fig. 3.6: Model discovery for the Hopf bifurcation with $\Delta t = 1$ and various $M$. 

(a) Real Dynamics

(b) Learned Dynamics with $M=1$

(c) Learned Dynamics with $M=5$

(d) Learned Dynamics with $M=10$
Fig. 3.7: Model discovery for the Hopf bifurcation with $\Delta t = 2$ and various $M$.

Visually, in Figure 3.6, we see no difference in the learned dynamics and real dynamics for the traditional PINN and the recursive PINN. However, in Figure 3.7b, we see the learned dynamics for the traditional PINN is nothing like the real dynamics. This difference vanishes for $M=5$ and $M=10$. This result agrees with the other dynamics we’ve looked at so far and is promising for the recursive PINN technique.

### 3.4 Multiscale System

We now investigate a major branch of dynamical systems, a multiscale dynamical system. The ODE system is given by
\[\begin{align*}
\dot{x}_1 &= x_2 - x_3, \\
\dot{x}_2 &= x_1 + \frac{1}{5} x_2, \\
\dot{x}_3 &= \frac{1}{5} + x_4 - 5x_3, \\
\dot{x}_4 &= \frac{x_4}{\mu} + \frac{x_1 x_3}{\mu},
\end{align*}\]  

(3.4)

where \(\mu\) is a parameter. We set \(\mu = 0.1\).

We fix domain \(\Omega = [-15, 15] \times [-15, 10] \times [-5, 25] \times [-30, 140]\), and use Latin hypercube sampling to randomly sample 1000 points from \(\Omega\) as \(\{\Psi_j^1\}_{j=1}^N\), and calculate the corresponding \(\{\Psi_j^2\}_{j=1}^N\) using (3.4). Once the model is trained, we utilize the neural network to calculate the solution for \(t \in [0, 20]\).

Table 3.4: \(l^2\) error of the predicted solutions using the learned model with different settings for the model in (3.4) with 10000 epochs.

<table>
<thead>
<tr>
<th>(\Delta t)</th>
<th>M=1</th>
<th>M = 10</th>
<th>M = 20</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>1.6765</td>
<td>0.6234</td>
<td>0.6080</td>
</tr>
<tr>
<td>0.25</td>
<td>3.6766</td>
<td>10.9516</td>
<td>0.5370</td>
</tr>
</tbody>
</table>

In Table 3.4 we see the the recursive model having a lower error in every case of \(\Delta t\) than the traditional PINN. Furthermore, in the \(\Delta t = 0.01\) and the \(\Delta t = 0.05\) the traditional PINN fails to approximate the differential equation.

Figure 3.8 shows the traditional approach does not approximate the dynamics. However, the recursive approach approximates the dynamics well. This is very encouraging as multiscale dynamics is a particularly difficult problem to solve.
Fig. 3.8: Model discovery of a multiscale dynamical system with $\Delta t = 0.1$. Left: $M=1$, Middle $M=10$, Right $M=20$.

Figure 3.9 shows the traditional approach does even worse for our larger $\Delta t$. When $M=10$, the learned dynamics follows the real dynamics for $t \in [0,5]$ but then diverges from the real solution after $t = 5$. For $M=20$, we see the recursive PINN approximates the dynamics well for the entire time interval investigated.
Fig. 3.9: Model discovery of a multiscale dynamical system with $\Delta t = 0.25$. Left: $M=1$, Middle $M=10$ Right $M=20$.

3.5 Lorenz system

For our final example, we look at a notable chaotic system. The Lorenz system is highly sensitive to initial conditions thereby making an interesting case for us to investigate. The Lorenz system, which was used by [16], is given by:
\begin{equation}
\dot{y}_1 = 10(y_2 - y_1),
\end{equation}
\begin{equation}
\dot{y}_2 = y_1(28 - y_3) - y_2,
\end{equation}
\begin{equation}
\dot{y}_3 = y_1 y_2 - \frac{8}{3} y_3.
\end{equation}

We fix domain \( \Omega = [-25, 25] \times [-25, 25] \times [0, 50] \), and use Latin hypercube sampling to randomly sample 1000 points from \( \Omega \) as \( \{\Psi^1_j\}_{i=1}^N \), and calculate the corresponding \( \{\Psi^2_j\}_{i=1}^N \) with the real solution. Once the model is trained, we utilize the neural network to calculate the solution for \( t \in [0, 10] \).

Table 3.5: \( l^2 \) error of the predicted solutions using the learned model with different settings for the model in (3.5) with 10000 epochs.

\begin{center}
\begin{tabular}{|c|c|c|}
\hline
& M=1 & M = 20 \\
\hline
\( \Delta t = 0.05 \) & 0.4263 & 0.4239 \\
\hline
\( \Delta t = 0.1 \) & 0.6683 & 0.3329 \\
\hline
\end{tabular}
\end{center}

Table 3.5 shows once again the advantages of the recursive PINN. We see noticeable improved accuracy over the traditional PINN. We again see the greatest improvement in \( l^2 \) error for the larger \( \Delta t \).

For Figure 3.10 we see the traditional PINN follows the dynamics for \( t \in [0, 4] \) before it diverges from the true solution. For 20 recursive steps there is a small but measurable improvement. Since Lorenz is a chaotic system, and small errors compound quickly, having even a small improvement in accuracy is very promising.
Fig. 3.10: Model discovery of the Lorenz system with $\Delta t = 0.05$ and various $M$. Left: $M=1$, Right: $M=20$.

In Figure 3.11 the traditional PINN network follows the dynamics for $t < 1$. The recursive approach however does very well, matching the real dynamics for $t \in [0, 9]$, a clear improvement.
Fig. 3.11: Model discovery of the Lorenz system with $\Delta t = 0.1$ and various $M$. Left: $M=1$, Right: $M=20$. 
CHAPTER 4

Conclusion

In conclusion, we introduced a novel recursive PINN to learn the dynamics from data and act as a black box function. The black box function acts as the right hand side of a differential equation. The recursive PINN method reduced the error in the numerical example compared to a traditional PINN method without recursion. We note the recursive PINN method could be improved by tuning the hyper-parameters. The recursive PINN could also be applied to higher dimensional PDEs. Both these prospects are outside the scope of this paper but could be looked into in future research.

We observed a greater degree of accuracy when learning the dynamics of a dynamical system when utilizing the recursive PINN approach for all dynamical systems investigated and for all $\Delta t$s tested. The greatest difference we observed was when $\Delta t$ was large. This suggests the recursive PINNs method may be more useful than the traditional method as it can learn the dynamics for a wider range of $\Delta t'$s.
REFERENCES


