

SOLVING QUANTUM EIGENVALUE PROBLEMS USING PHYSICS- INFORMED NEURAL NETWORKS

BY

IDRIS ODUOLA

A Thesis Presented to the
COLLEGE OF GRADUATE AND INTERDISCIPLINARY STUDIES
KING FAHD UNIVERSITY OF PETROLEUM & MINERALS
DHAHRAN, SAUDI ARABIA

In Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

In

MATHEMATICS

MAY 2024

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS
DHAHRAN 31261, SAUDI ARABIA

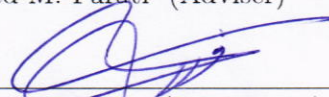
DEANSHIP OF GRADUATE STUDIES

This thesis, written by **IDRIS ODUOLA** under the direction of his thesis adviser and approved by his thesis committee, has been presented to and accepted by the Dean of Graduate Studies, in partial fulfillment of the requirements for the degree of **MASTER OF SCIENCE IN MATHEMATICS**.

Thesis Committee



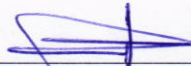
Prof. Khaled M. Furati (Adviser)



Prof. Fahhad Al Harbi (Co-Adviser)



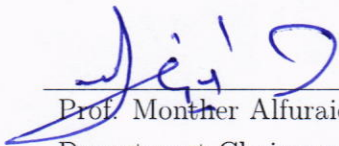
Dr. Abdullah Shah (Member)



Dr. Rachid Ait Haddou (Member)



Dr. Zaid Salwan (Member)



Prof. Monther Alfuraidean
Department Chairman



Prof. Suliman Al Homidan
Dean of Graduate Studies

Date



©IDRIS ODUOLA
2024

To my late father and beloved family.

ACKNOWLEDGMENTS

All praise to ALLAH, Lord of the worlds, the Almighty, with whose gracious help it was possible to accomplish this work. I extend my deepest appreciation to my supervisor, Prof. Khaled M. Furati, for his exceptional guidance, steadfast support, and invaluable insights throughout my research journey. His expertise and mentorship were pivotal in shaping this thesis. I am grateful to the Department of Mathematics for their continuous support and finding me worthy of the scholarship that facilitated my studies. The unwavering assistance has been indispensable in my academic pursuit. I also express sincere gratitude to my thesis committee members, most especially to Prof. Fahhad Alharbi for generously sharing his expertise and his invaluable assistance throughout this thesis. Finally, I thank everyone, both family and friends, who contributed to my learning in this great University.

TABLE OF CONTENTS

ACKNOWLEDGMENTS	vi
LIST OF TABLES	ix
LIST OF FIGURES	x
LIST OF ABBREVIATIONS	xii
ABSTRACT (ENGLISH)	xiii
ABSTRACT (ARABIC)	xiv
CHAPTER 1 INTRODUCTION	1
1.1 Motivation	1
1.2 Eigenvalue Problems on Finite Dimensional Spaces	2
1.3 1D Time-Independent Schrödinger Equation	4
1.3.1 Infinite Potential Well Problem	5
1.3.2 Single Finite Potential Well Problem	8
CHAPTER 2 BACKGROUND AND LITERATURE REVIEW	14
2.1 Neural Network Overview	14
2.1.1 Neural Network as a Universal Approximator	15
2.1.2 Neurons	16
2.1.3 Activation Function	16
2.1.4 Neural Network Optimization Process	18
2.2 Physics Informed Neural Network	20

2.3	Overview	23
CHAPTER 3 RQPINN FOR EIGENVALUE PROBLEMS		26
3.1	RqPINN Model	27
3.1.1	Computation of Eigenvalues	28
3.1.2	Loss Computation	29
3.2	The Loss Function	29
3.2.1	Physics Loss	30
3.2.2	Regularization Terms	30
3.3	Boundary Condition	34
3.4	Training the PINN	35
CHAPTER 4 RQPINN FOR QUANTUM EIGENVALUE PROBLEMS		38
4.1	Metric and Datasets	38
4.2	Performance Analysis of RqPINN	40
4.3	RqPINN for 1D Time-Independent Schrödinger Equation	44
4.3.1	Infinite Potential Well using RqPINN	48
4.3.2	Single Finite Potential Well using RqPINN	52
CHAPTER 5 REMARKS AND FUTURE WORKS		58
5.1	General Remarks	59
5.2	Future Work	60
REFERENCES		61
VITAE		68

LIST OF TABLES

4.1	Relative error for the largest eigenvalue of (4.3) corresponding to different collocation points and method of selection.	41
4.2	Mean Absolute Error for squared of the eigenfunction obtained with different number of collocation points and methods of selection.	41
4.3	Relative eigenvalue errors with respect to different neurons and layers .	42
4.4	Mean absolute error of the squared eigenfunction with different neurons and layers	44
4.5	Comparison of Energy Levels	50
4.6	Mean Absolute Error (MAE) for ψ values	51
4.7	Comparison of energy levels of the finite well problem obtained by RqPINN with the chebfun toolbox. The last column indicates the epoch number at which the stopping criteria is satisfied.	56

LIST OF FIGURES

1.1	Pictorial depiction of the infinite potential well with a particle inside the well.	6
1.2	Pictorial depiction of the single finite potential well with a particle inside. The well can be divided into three regions, two regions outside the well and one inside.	9
1.3	Plot of equations (1.22), (1.23), and (1.27). The points of intersection are the solution sets and can be used to obtain the required energy levels.	13
2.1	Illustration of a 3-layer neural network with 4 neurons in each hidden layer, and 1 neuron in the output layer.	16
3.1	PINN proposed by [20] for solving 1D time independent Schrödinger equation. At the beginning of training, an array of constants is fed into the neurons that computes the eigenvalues.	28
3.2	RqPINN for solving 1D eigenvalue problems.	30
3.3	Plot of the parametric equation (3.12), for $\Omega = [-6, 6]$	36
3.4	Illustration of the PINN training process.	37
4.1	Plot of the relative error during training of the RqPINN.	42
4.2	Plot of moving average calculated over a window of 1000 epochs. The moving average removes the oscillations observed in Figure 4.1, demonstrating a noticeable convergence of the RqPINN.	43
4.3	Plot of the eigenvalues during training of RqPINN and PINN by [20]. RqPINN converges rapidly to the exact solution while the PINN by [20] oscillates for over 4000 epochs before it starts to converge.	45

4.4	Plot of the relative errors between RqPINN and PINN by [20]. RqPINN has superior accuracy compared to the other algorithm.	46
4.5	First and second wavefunctions of the infinite potential well	49
4.6	Third and fourth wavefunctions of the infinite potential well	49
4.7	Fifth and sixth wavefunctions of the infinite potential well	50
4.8	Plot of loss function and energy levels during training	51
4.9	Illustration of the finite potential well.	52
4.10	Output of PINN after approximating the gaussian (4.5) for 5000 epochs.	54
4.11	First and second wavefunctions of the finite potential well	55
4.12	Third and fourth wavefunctions of the finite potential well	55
4.13	Fifth and sixth wavefunctions of the finite potential well	56
4.14	Plot of all six learned wavefunctions by RqPINN inside the finite well. .	57
4.15	Plot of loss function and energy levels during training of RqPINN for the finite well problem. Each spike on the loss function plot (left) indicates that the stopping criteria is met and the network is updated to search for the next solution pair. The dashed horizontal lines of the energy plot (right) indicates the required energy level, and RqPINN can be seen oscillating very close around them until the stopping criteria is met.	57

LIST OF ABBREVIATIONS

CNN	Convolutional Neural Network
GPU	Graphics Processing Units
LHS	Latin Hypercube Sampling
ML	Machine Learning
NaN	Not a Number
NN	Neural Network
PDE	Partial Differential Equations
PINN	Physics-Informed Neural Networks
RqPINN	Rayleigh quotient based PINN

THESIS ABSTRACT

NAME: IDRIS ODUOLA

TITLE OF STUDY: Solving Quantum Eigenvalue Problems using Physics-Informed Neural Networks

MAJOR FIELD: Mathematics

DATE OF DEGREE: May 2024

Eigenvalue problems arise in several areas of science and technology. In quantum mechanics, they are instrumental in characterizing the behaviour of physical systems. These problems are often very challenging to solve analytically. Thus, in this thesis we propose a machine learning technique based on Physics-Informed Neural Network (PINN). We demonstrate how this approach could be used to obtain eigenpairs of 1D time-independent Schrödinger equation. Furthermore, the performance analysis, merits and limitations of this technique are discussed.

ملخص الرسالة

الاسم: إدريس أودولا

عنوان الدراسة: حل مسائل القيمة الذاتية الكمومية باستخدام الشبكات العصبية المستنيرة بالفيزياء

التخصص: رياضيات

تاريخ الدرجة العلمية: مايو 2024

تنشأ مشاكل القيمة الذاتية في العديد من مجالات العلوم والتكنولوجيا. تظهر مشكلات القيم الذاتية في عدة مجالات من العلوم والتكنولوجيا. في ميكانيكا الكم، تُعد هذه المشكلات أساسية في توصيف سلوك الأنظمة الفيزيائية. غالبًا ما يكون حل هذه المشكلات تحليليًا تحديًا كبيرًا. وعليه، نقترح في هذه الرسالة تقنية تعلم آلي تستند إلى الشبكة العصبية الموجهة بالفيزياء (PINN) تُظهر كيف يمكن استخدام هذا النهج للحصول على أزواج القيم الذاتية لمعادلة شرودنجر غير المعتمدة على الزمن في بُعد واحد. علاوة على ذلك، يُناقش تحليل الأداء والمزايا والقيود لهذه التقنية.

CHAPTER 1

INTRODUCTION

1.1 Motivation

Machine Learning (ML) has led to a revolution in technology. It has brought things which were once seen as fiction to real life, from the development of autonomous vehicles, to recommendation systems, to anomaly detection and many others. It has also gained popularity due to its significant contribution to various fields of science and engineering. For instance, machine learning can be used to model and predict environmental changes [21], to enhance safety and efficiency in aircraft design [10], to monitor the health and safety of structures such as bridges and buildings [44], amongst many others.

Machine learning has also been utilized to solve Partial Differential Equations (PDE). Since PDEs form a basis for many problems in science and engineering, extensive research has been done to develop various methods of solving them. An approach to solve PDEs using ML is done by embedding the knowledge of physics laws into a Neural Network (NN) (a Machine Learning algorithm) learning process. Using ML

approach has been presented as early as the 1990's in works such as [24], but the name Physics-Informed Neural Networks (PINN) was coined years later in [33]. PINN utilizes neural networks which are universal function approximators, and it uses prior knowledge of the PDE as a regularization agent that reduces the space of admissible functions. One advantage of this technique over classical numerical methods is its mesh-free nature, which enables it to easily handle problems with complex geometries and irregular domains.

1.2 Eigenvalue Problems on Finite Dimensional Spaces

Let $T : V \mapsto V$ be a linear operator on an n -dimensional vector space V over a field $\mathbf{F}$. Suppose there exist an ordered basis $\mathcal{B} = \{x_1, x_2, \dots, x_n\}$ such that the matrix representation of T is a diagonal matrix D . Then $[T]_{\mathcal{B}} = D$ if and only if:

$$T(x_k) = \lambda_k x_k. \tag{1.1}$$

The diagonal matrix D is called the diagonal form or diagonal representation of T , and T is said to be diagonalizable. Diagonalization simplifies computation and sufficient information's could be obtained with just a glance of the matrix D . For example, the range is the subspace spanned by the basis vectors x_k 's where $\lambda_k \neq 0$ [17]. and operations such as exponential can be easily performed on the operator T , because it is very straightforward on a diagonal matrix.

The scalars $\lambda_k \in \mathbf{F}$ are called the eigenvalues (characteristic values) corresponding to any x_k such that (1.1) holds, and each x_k is called an eigenvector (characteristic vector).

For n -dimensional vector space V over $\mathbf{F}$, the linear operator T can be represented by an $n \times n$ matrix A over $\mathbf{F}$. Thus, the eigenvalues of A are scalars λ such that the matrix $(A - \lambda I)$ is singular or $|A - \lambda I| = 0$. The monic polynomial $P(x) = |A - xI|$ is called the characteristic polynomial of A and it is used to obtain the eigenvalues of A , which are just the roots of $P(x)$.

The problem of finding eigenvalues and their corresponding eigenvectors of a given linear operator T (or matrix A) is called an "Eigenvalue problem". This problem arises naturally in science and engineering, e.g., in the longitudinal vibration of an elastic bar [1], signal processing [29], graph theory [7], and quantum mechanics [8].

Solving the eigenvalue problems can be challenging, as there can be many types, corresponding to the properties of the matrix. Thus, efficiently solving this problem involves recognizing its mathematical type [3]. For example:

- Hermitian Eigenvalue problems: These are eigenvalue problems of the form $Ax = \lambda x$, where A is Hermitian, i.e. $A = A^*$, A^* is the conjugate transpose of A .
- Generalized Hermitian Eigenvalue problems: These are of the form $Ax = \lambda Bx$, where A and B are Hermitian, and B is positive definite.

If A is Hermitian, all the eigenvalues are real and can be written in sorted order, $\lambda_1 \leq \lambda_2 \leq \dots \leq \lambda_n$. If all $\lambda_i > 0$, then A is called positive definite. Each eigenvector

$x_i, i = 1, \dots, n$ form a basis of a subspace of V , called the eigenspace or invariant space. With this in mind, we can choose each eigenvector such that the L2-norm is 1, i.e. $\|x_i\|_2 = 1$.

1.3 1D Time-Independent Schrödinger Equation

In 1925, the French physicist Louis de Broglie postulated that particles (like electrons) exhibit wave behaviour. This phenomena is referred to as wave-particle duality. In 1926, a concrete evidence of the de Broglie's hypothesis was carried out through an experiment by Clinton Davisson and Lester Germer at the Bell Laboratories. They wanted to study the behavior of electron interaction with the surface of a crystalline target. They observed that when slow-moving electrons were scattered by the nickel crystal, they displayed a diffraction pattern, which confirms the wave nature of electrons [34].

The electron wave-particle duality could only be modeled through a fundamental modification of basic physical concepts, which was later termed "Quantum Mechanics". Furthermore, Erwin Schrödinger in 1926 proposed an equation that models the behavior of an electron in a system, based on the following assumptions:

- Since electrons exhibit wave-particle duality, the momentum and position cannot be determined precisely (Heisenberg principle). Thus, the state of a system can be described by a wave function ψ which is a function of space and time.
- $|\psi(x)|^2$ denotes the probability of finding an electron or particle at a position x and time t .

- The sum of probabilities of all possible values of the coordinates of the system must be 1, which is called the normalization condition

$$\int |\psi(x)|^2 dx = 1. \quad (1.2)$$

In steady state, the Schrödinger equation reduces to:

$$\left[\frac{-\hbar^2}{2m} \frac{d^2}{dx^2} + V(x) \right] \psi(x) = E\psi(x). \quad (1.3)$$

The terms in the square bracket can be seen as an operator acting on ψ , and is called the Hamiltonian operator $\hat{H}$. $V(x)$ is the potential energy term, $\hbar$ is the reduced Planck's constant and m is the mass of the electron.

It can be seen that the time-independent Schrödinger equation is an eigenvalue problem, with the energy E as the eigenvalue and the wavefunction ψ as the eigenfunction.

1.3.1 Infinite Potential Well Problem

The potential energy $V(x)$ is the energy due to the system that acts on the electron or particle. Suppose we consider the 1D problem of a particle in a well like Figure 1.1, such that the potential $V(x)$ is defined as:

$$V(x) = \begin{cases} 0, & 0 \leq x \leq L, \\ \infty, & \text{otherwise.} \end{cases}$$

Since the potential is infinitely high outside the well, this means that the particle cannot exist there except for where $V(x) = 0$. Applying this information to the Schrödinger equation (1.3) reduces it to

$$\frac{-\hbar^2}{2m} \frac{d^2\psi}{dx^2} = E\psi(x). \quad (1.4)$$

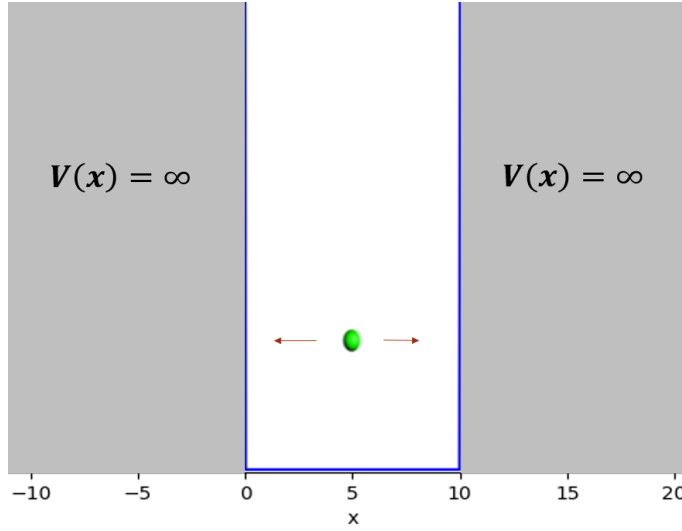


Figure 1.1: Pictorial depiction of the infinite potential well with a particle inside the well.

Equation (1.4) is a second-order differential equation which can be rewritten as:

$$\frac{d^2\psi}{dx^2} = -k^2\psi(x), \quad (1.5)$$

where $k = \sqrt{\frac{2mE}{\hbar^2}}$, and the solution is:

$$\psi(x) = A \sin(kx) + B \cos(kx). \quad (1.6)$$

To obtain the values of the parameters A and B , $\psi(0)$ and $\psi(L)$ must be zero

because the wavefunction vanishes outside the well.

$$\psi(0) = \psi(L) = 0.$$

The condition $\psi(0) = 0$ implies that $B = 0$ and (1.6) becomes:

$$\psi(x) = A \sin(kx). \quad (1.7)$$

Applying the second boundary condition,

$$\psi(L) = A \sin(kL) = 0,$$

we obtain that

$$k = \frac{n\pi}{L}.$$

Therefore,

$$\psi_n(x) = A_n \sin\left(\frac{n\pi x}{L}\right), \quad n = 1, 2, \dots \quad (1.8)$$

and

$$E_n = \frac{\hbar^2}{2m} \left(\frac{n\pi}{L}\right)^2, \quad n = 1, 2, \dots \quad (1.9)$$

is the eigenvalue solution (Energy) for the infinite potential well problem, for $n \in \mathbb{N}$.

Finally, the normalization condition (1.2) of the wave function implies that:

$$\int_0^L \left[A_n \sin\left(\frac{n\pi x}{L}\right) \right]^2 dx = 1.$$

Integrating the above, we obtain

$$A_n = \sqrt{\frac{2}{L}}.$$

Hence, the wavefunction is given by

$$\psi_n(x) = \sqrt{\frac{2}{L}} \sin\left(\frac{n\pi x}{L}\right). \quad (1.10)$$

It is important to note that n is restricted to be positive integers because $n = 0$ implies $\psi_n = 0$ (trivial state), and $n < 0$ yields $-\psi_n$, which belongs to the eigenspace generated by ψ_n .

1.3.2 Single Finite Potential Well Problem

The case where potential is infinite outside the well may be unrealistic in real life. A more appropriate situation is where the potential energy is a finite value outside the well. Here we define the potential as:

$$V(x) = \begin{cases} 0, & -L \leq x \leq L, \\ V_0, & \text{otherwise.} \end{cases} \quad (1.11)$$

This potential has an even symmetry, $V(x) = V(-x)$, which is found to influence the form of the wavefunctions. When the potential is symmetric, the solution of the time-independent Schrödinger equation (1.3) can be classified as either odd or even. Even wavefunctions correspond to those that are symmetric about the center of

the well ($\psi(x) = \psi(-x)$), while the odd wavefunctions correspond to those that are antisymmetric about the center ($\psi(x) = -\psi(-x)$). To solve (1.3), we first consider some continuity conditions:

1. $\psi(x)$ must be continuous
2. $\psi(x)$ must be smooth i.e. $\psi'(x)$ must be continuous

We can divide the well into three regions, two regions outside the well and one region inside the well. Then the Schrödinger equation becomes:

$$\begin{cases} \frac{d^2\psi(x)}{dx^2} = -\frac{2mE}{\hbar^2}\psi(x), & -L \leq x \leq L, \\ \frac{d^2\psi(x)}{dx^2} = \frac{2m(V_0-E)}{\hbar^2}\psi(x), & \text{otherwise.} \end{cases} \quad (1.12)$$

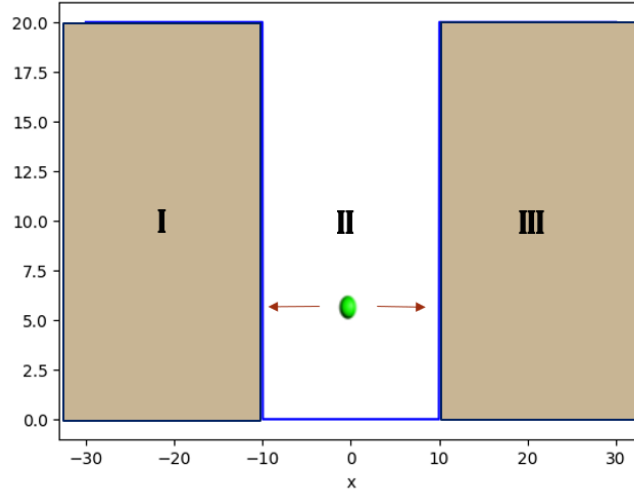


Figure 1.2: Pictorial depiction of the single finite potential well with a particle inside. The well can be divided into three regions, two regions outside the well and one inside.

Let

$$k = \sqrt{\frac{2mE}{\hbar^2}}, \quad \alpha = \sqrt{\frac{2m(V_0 - E)}{\hbar^2}}. \quad (1.13)$$

Then (1.12) can be rewritten as

$$\begin{cases} \frac{d^2\psi(x)}{dx^2} = -k^2\psi(x), & \text{for } -L \leq x \leq L, \\ \frac{d^2\psi(x)}{dx^2} = \alpha^2\psi(x), & \text{otherwise.} \end{cases} \quad (1.14)$$

Solving (1.14), we obtain

$$\psi(x) = \begin{cases} Ae^{\alpha x} + Be^{-\alpha x}, & x < -L \text{ (region I),} \\ C \sin(kx) + D \cos(kx), & -L \leq x \leq L \text{ (region II),} \\ Ee^{\alpha x} + Fe^{-\alpha x}, & x > L \text{ (region III).} \end{cases} \quad (1.15)$$

Applying the boundary conditions, we expect that $\psi(-\infty) = \psi(\infty) = 0$. Thus (1.15) becomes

$$\psi(x) = \begin{cases} Ae^{\alpha x}, & x < -L, \\ C \sin(kx) + D \cos(kx), & -L \leq x \leq L, \\ Fe^{-\alpha x}, & x > L. \end{cases} \quad (1.16)$$

Since the wavefunctions are either even or odd (as a consequence of the symmetry of the potential $V(x)$). This means that we can deal with each of them one at a time.

We consider the first case, when $\psi(x)$ is even. Therefore, $C = 0$ and so we have that

$$\psi(x) = \begin{cases} Ae^{\alpha x}, & x < -L, \\ D \cos(kx), & -L \leq x \leq L, \\ Fe^{-\alpha x}, & x > L. \end{cases} \quad (1.17)$$

From the conditions of continuity and smoothness of $\psi(x)$, it follows that $A = F$,

$$Ae^{-\alpha L} = D \cos(kL), \quad (1.18)$$

and,

$$\alpha Ae^{-\alpha L} = kD \sin(kL). \quad (1.19)$$

Therefore, it follows from (1.13) that

$$k^2 + \alpha^2 = \frac{2mV_0}{\hbar^2}. \quad (1.20)$$

Dividing (1.19) by (1.18), we obtain:

$$\alpha = k \tan(kL). \quad (1.21)$$

Multiplying (1.21) by L , (1.20) by L^2 and letting $u = kL$, $v = \alpha L$, and $R_0^2 = \frac{2mL^2V_0}{\hbar^2}$, we obtain the non-linear system

$$v = u \tan(u), \quad (1.22)$$

$$u^2 + v^2 = R_0^2. \quad (1.23)$$

Substituting (1.22) in (1.23), yields

$$u^2 \sec^2(u) = R_0^2. \quad (1.24)$$

This is a non-linear equation that can be solved using a non-linear solver or graphically by plotting (1.22) and (1.23). The solution set will be the points of intersection $(u_n, v_n), n = 1, 2, \dots$, of the plots of (1.22) and (1.23). Since all other parameters are known, then we can obtain the energy levels E_n as:

$$E_n = \frac{\hbar^2 \pi^2}{2mL^2} \left(\frac{u_n}{\pi/2} \right)^2, \quad n \in \mathbb{N}. \quad (1.25)$$

In (1.25), the term outside the bracket is the energy ground state of the infinite quantum well (1.9), i.e. $n = 1$. This implies that if $u = \frac{\pi}{2}$, then $R_0 \rightarrow \infty$ and so we resolve back to an infinite well problem. Therefore, (1.25) can be rewritten as

$$E_n = E_{\infty,1} \left(\frac{u_n}{\pi/2} \right)^2, \quad n \in \mathbb{N}. \quad (1.26)$$

Where $E_{\infty,1}$ is the energy ground state of the infinite well problem. If we consider the second case where $\psi(x)$ is odd, then $D = 0$ in (1.16) and following the same procedure,

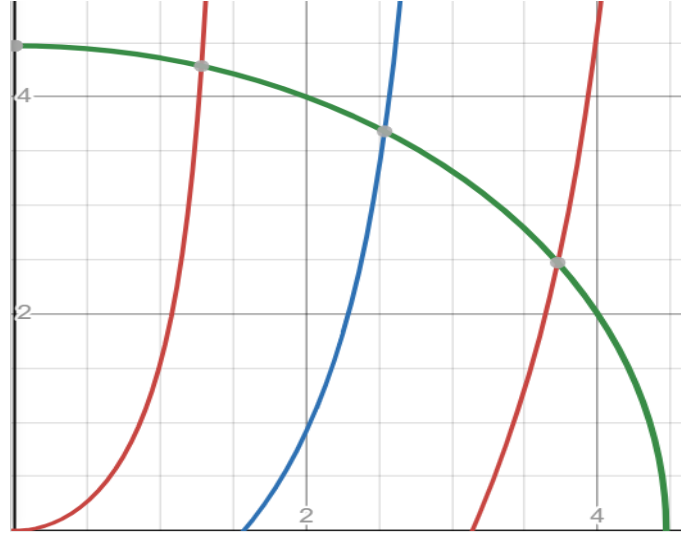


Figure 1.3: Plot of equations (1.22), (1.23), and (1.27). The points of intersection are the solution sets and can be used to obtain the required energy levels.

we will obtain

$$v = -u \cot(u), \quad (1.27)$$

$$u^2 + v^2 = R_0^2. \quad (1.28)$$

Hence, we can obtain both the even and odd energy levels of the finite well, graphically.

CHAPTER 2

BACKGROUND AND LITERATURE REVIEW

2.1 Neural Network Overview

Let $n_i, i = 0, 1, 2, \dots, k + 1$ be positive integers and

$$\Theta^i : \mathbb{R}^{n_i} \mapsto \mathbb{R}^{n_{i+1}}, \quad i = 0, 1, \dots, k, \quad (2.1)$$

be an affine function defined as

$$\Theta^i(x) = W^i(x) + b^i, \quad i = 0, 1, \dots, k, \quad (2.2)$$

where $W^i \in \mathbb{R}^{n_{i+1} \times n_i}$ and $b^i \in \mathbb{R}^{n_{i+1}}$. Let a non-linear activation function σ_i over Θ^i be of the form

$$\sigma_i : \mathbb{R}^{n_{i+1}} \mapsto \mathbb{R}^{n_{i+1}}, \quad i = 0, 1, \dots, k. \quad (2.3)$$

Then a neural network from $\mathbb{R}^{n_0}$ to $\mathbb{R}^{n_{k+1}}$ [16] is defined as

$$NN(x) = \Theta^k \circ \sigma_k \circ \Theta^{k-1} \circ \sigma_{k-1} \cdots \circ \Theta^1 \circ \sigma_1 \circ \Theta^0(x). \quad (2.4)$$

In essence, a neural network is a parametric function that is produced by sequentially composing affine functions, followed by a non-linear (activation) function. The linear transformation matrices W^i and translation vectors b^i of the affine functions serve as the neural network's parameters and are called weights and bias respectively [19]. The neural network (2.4) is called a $(k+1)$ -layer neural network, with k hidden layers. If $k = 1$, then the neural network is said to be shallow and if k is large, it is called a deep neural network.

Next, we provide brief descriptions of the entire architecture of a neural network.

2.1.1 Neural Network as a Universal Approximator

Neural networks (both shallow and deep) are a class of universal function approximators. They can approximate any Borel measurable function from one finite dimensional space to another, to any desired degree of accuracy, provided that $n_1, n_2, \dots, n_k$ are sufficiently large [18]. This is done by updating the parameters through an optimization process called back-propagation that searches for the minimum of a loss function $\mathcal{L}(\theta)$.

2.1.2 Neurons

Neural networks are always illustrated as stacks of connected neurons such as in Figure 2.1. A neuron is a unit of a neural network that represents a single composition of an affine function, and can be defined as

$$nn_j(x) = \sigma \circ \Theta_j^i(x), \quad j > 0, \quad (2.5)$$

where σ is an activation function and $\Theta_j^i(x) : \mathbb{R}^{n_i} \mapsto \mathbb{R}$ is an affine function with

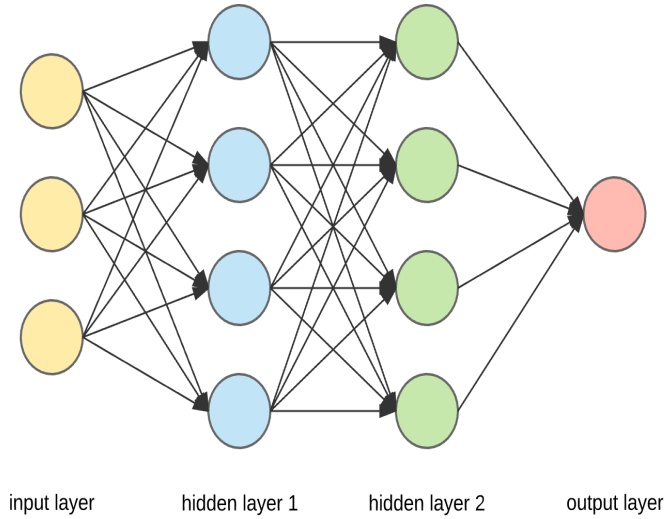


Figure 2.1: Illustration of a 3-layer neural network with 4 neurons in each hidden layer, and 1 neuron in the output layer.

a scalar output. A neural network layer is composed of neurons that are stacked together and whose combined output is received by each neuron in the next layer.

2.1.3 Activation Function

Without the non-linear activation function σ , neural networks are just compositions of linear functions, which in fact is itself linear. The activation function σ introduces

non-linearity to the neural network, which allows it to better approximate not only linear functions but also general non-linear functions. There are different types of activation functions, such as

- Sigmoid: It is also called logistic activation function, and is used for binary classification (i.e. $f = \{0, 1\}$). It is defined as:

$$\sigma(\Theta) = \frac{1}{1 + e^{-\Theta}}.$$

- ReLU: Abbreviated from Rectified Linear Unit, ReLU is a very popular activation function which is defined as:

$$\sigma(\Theta) = \max\{0, \Theta\}.$$

ReLU is mostly preferred because of its computational efficiency to compute and differentiate (making it very suitable for large neural networks), and also preventing common problems like vanishing gradient when training a neural network. Recently, [16] showed that using ReLU in a deep neural network is identical to the set of linear finite element functions used in numerical PDEs.

There are different variants of the ReLU activation function, such as leaky ReLU, GeLU, ELU, SeLU and so on.

- Sine: The sine activation function is basically, using the trigonometric sinusoidal

function to non-linearize the linear functions of a neural network

$$\sigma(\Theta) = \sin(\Theta).$$

It was proposed in 2020 by [37], and is believed to work best for functions that are periodic.

2.1.4 Neural Network Optimization Process

Loss Function

Let $n, M \in \mathbb{N}$, and $x_1, x_2, \dots, x_M \in \mathbb{R}^n$, $y_1, y_2, \dots, y_M \in \mathbb{R}$ are data points such that

$$y_k = f(x_k), \tag{2.6}$$

for every $k = 1, 2, 3, \dots, M$. Where $f \in C(\mathbb{R}^n, \mathbb{R})$ is an unknown function that maps $x_k \in \mathbb{R}^n$ to $y_k \in \mathbb{R}$. We aim to find a function $\hat{f}$, such that $\hat{f}(x_i) \approx f(x_i)$ for $i \notin \{1, 2, 3, \dots, M\}$ without using explicit knowledge of f . To do this, we try to obtain $\hat{f} \in C(\mathbb{R}^n, \mathbb{R})$ such that a function $\mathcal{L} : C(\mathbb{R}^n, \mathbb{R}) \mapsto [0, \infty)$ is minimum. The function $\mathcal{L}(\phi)$ is called the loss function, and can be defined as

$$\mathcal{L}(\phi) = \frac{1}{M} \left[\sum_{k=1}^M |\phi(x_k) - y_k|^2 \right] \in [0, \infty), \tag{2.7}$$

for every $\phi \in C(\mathbb{R}^n, \mathbb{R})$ [19].

Back Propagation

$\hat{f}$ is obtained through a process called back propagation. Back propagation is an optimization process that updates the parameters of a neural network through the use of gradients or partial derivatives computed from the loss function $\mathcal{L}(\phi)$. Back propagation uses the concept of chain rule to update the weights W and biases b of the neural network sequentially, from the closest layer to the final output of the neural network to the first hidden layer. Some common back propagation methods are gradient descent, stochastic gradient descent and ADAM optimization.

The concept of back propagation is an inverse problem, where we obtain the best parameters W and b of $\Theta = Wx + b$ that gives the best approximation $\hat{f}$ in (2.6). A very simple back propagation process is the gradient descent, which is defined as

$$W := W - \alpha \frac{\partial \mathcal{L}}{\partial W}, \quad (2.8)$$

$$b := b - \alpha \frac{\partial \mathcal{L}}{\partial b}. \quad (2.9)$$

The values of W and b are updated iteratively, with each iteration called "epoch". The loss function can be seen as a valley or combination of valleys, where the objective is to find the lowest point, which corresponds to minimizing the loss $\mathcal{L}(\phi)$. The variable α is called "learning rate", and it is a parameter (usually called a hyper-parameter) that is fixed before training. It indicates the size of steps taken while descending down the valley of the loss function. Just as in real-world hiking, where a larger step may help descend more quickly but risks overshooting the optimal path, the learning rate determines the magnitude of parameter updates in each iteration.

of the optimization process. A higher learning rate means larger steps, potentially leading to faster convergence but with a greater risk of oscillation or overshooting the minimum. Conversely, a smaller learning rate ensures more cautious progress, reducing the risk of overshooting but potentially slowing down convergence. Finding the right balance is crucial, as it ensures steady progress towards the optimal solution without compromising stability or efficiency..

2.2 Physics Informed Neural Network

In recent years, Machine (deep) learning has been extended to solve PDEs. This method involves first reformulating the given PDE as an infinite dimensional stochastic optimization problem, which is reduced to a finite stochastic optimization problem using deep neural networks as approximations for the PDE solution including its derivatives. Then, this optimization problem is solved using optimization methods like gradient descent[19]. There are various schemes where deep learning is utilized in solving PDEs, among which is Physics Informed Neural Networks (PINNs).

PINNs is a deep learning approach of solving PDEs that incorporates physical laws into a neural network architecture. This is done by embedding the knowledge of the physics within the loss function during training. PINNs exploits the universal approximation property of neural networks to approximate PDEs, both linear and non-linear, without requiring local time-stepping, linearization, or any prior assumptions to be made [33]. Since the main component of PDEs are derivatives, PINNs utilizes automatic differentiation, otherwise called "autodiff", to compute the derivatives of

the PDEs. This is a technique that generates numerical derivative evaluation by computing derivatives through the accumulation of values during code execution[4].

As an illustration, consider the PDE of the form

$$u_t + \mathcal{N}[u] = 0, x \in \Omega, t \in [0, T], \quad (2.10)$$

where u is the solution of the PDE, $\mathcal{N}$ is a non-linear differential operator, and $\Omega \subset \mathbb{R}^D$.

To obtain u through PINN, let $\hat{u}$ be the output of a neural network as defined in (2.4).

We aim to approximate u such that $|u - \hat{u}| \approx 0$.

Now, define $f(t, x)$ such that

$$f := u_t + \mathcal{N}[u]. \quad (2.11)$$

Then, the loss function for the neural network optimization process will be defined as follows:

$$\mathcal{L}(u) = MSE_i + MSE_b, \quad (2.12)$$

where

$$MSE_i = \frac{1}{M_i} \left[\sum_{k=1}^{M_i} |f(t_k, x_k)|^2 \right], \quad (2.13)$$

and

$$MSE_b = \frac{1}{M_b} \left[\sum_{k=1}^{M_b} |u(t_k, x_k) - \hat{u}(t_k, x_k)|^2 \right]. \quad (2.14)$$

Here, M_i denotes the collocation points for $f(t, x)$ and M_b denotes training data from the initial and boundary conditions on $u(t, x)$. The MSE_i and MSE_b drives the neural

network to search for solutions $\hat{u}$ in the solution space of the given PDE by enforcing the structure of the equation and boundary conditions such that the mean-squared error is minimum, through gradient-descent type optimization methods.

The use of PINN and other deep learning approaches to solve differential equations poses several benefits over traditional numerical methods. We highlight some of these benefits below.

- Deep learning approaches offer a promising solution to the curse of dimensionality, a prevalent challenge in many numerical methods. Conventional approaches often employ dimensionality reduction techniques for high-dimensional problems, which can compromise solution accuracy. The use of deep learning has been demonstrated to solve very high dimensional problems[14]. Furthermore, deep learning approaches alleviate the difficulties associated with discretizing domains into appropriate grids or meshes, providing a more flexible and adaptable approach to solving complex problems.
- Deep learning approaches enable the utilization of rapidly advancing machine learning tools, such as Graphics Processing Units (GPU), to solve scientific problems. GPUs are particularly well-suited for accelerating the training and inference processes of deep learning models due to their parallel processing capabilities. By leveraging GPUs, deep learning approaches can significantly reduce computation time and enable the efficient solution of complex scientific problems.
- The output obtained from these approaches are analytic functions and can gen-

eralize well to unseen data points. Obtaining the solution at these points is almost instantaneous after training for the first time.

- Traditional numerical methods often rely on specific assumptions or simplifications to solve PDEs, and different methods may be required for linear versus nonlinear equations, or for problems in different dimensional spaces. PINN offer a unified approach that can handle linear and nonlinear PDEs, as well as problems in both high and low dimensional spaces, using the same underlying framework.

2.3 Overview

In 1959, Arthur Samuel defined Machine learning as the field of study that give computers the ability to learn without being programmed explicitly. The concept of machine learning started in mid 20th century, when researchers wanted computers that can do things without "explicitly programmed". However, due to limited computing power and limitations on early algorithms, there became a decline in machine learning research.

With the discovery of back-propagation in the 1990s [35], increased computing power and availability of data due to the rise of internet, research in the field of machine learning once again re-emerged and this time led to groundbreaking discoveries, such as Convolutional Neural Network (CNN) [25][23], reinforcement learning [22], and deep learning (or deep neural networks) [26]. This led to computers performing tasks that surpass human capabilities.

During the resurgence of machine learning in the 1990s, Works like [24], [9], [38] and [27] emerged utilizing neural networks (NN) to solve differential equations. Since many PDE problems such as high dimensional non-linear PDEs cannot be solved analytically, and current approaches such as finite-element method, spectral galerkin method and other numerical approaches suffer from limitations such as curse of dimensionality and finding an appropriate mesh, researchers began to utilize deep-learning approach in solving PDEs.

With the availability of computational power, recent deep learning approaches such as the Deep Ritz method [43], Deep Galerkin method [36], and Physics Informed Neural Network (PINN) [33] gained attractions. These methods have since been extended with remarkable results in solving very high dimensional PDEs [14], fractional PDEs [31], heat transfer problems [5], inverse problems [6] and even in finance [2].

Another interesting area that deep learning approaches can prove very useful is in solving eigenvalue problems, where not only one solution is desired, but a whole family of solutions. While PINN and the other deep learning approaches can solve both linear and non-linear PDEs, they do not work well in all problems [12] [40]. Thus, either problem specific novel deep learning approaches have to be developed or existing methods have to be extended in solving specific problems where they fail. With respect to this and among many others, [11] introduced a deep learning approach utilizing CNN for eigenvalue problems in mechanics, [15] proposed the using a diffusion Monte Carlo inspired deep neural network to solve high-dimensional eigenvalue problems, and more recently, [42] introduced a neural network based on the power method, inverse power method to obtain the largest and smallest eigenpairs.

While developing new algorithms may seem good, coming up with different methods for every problem may not be a feasible thing to do. It may be way better to extend existing methods to further generalize such problems. This can be seen in [20] where they extended PINN to solve quantum eigenvalue problems, through carefully adding a neuron in each layer of the neural network to compute the eigenvalues, and adding regularization terms to better inform the neural network of the desired eigenfunctions.

In this thesis, we improve the work of [20] with a focus on developing a more efficient way of informing the physics specific to quantum eigenvalue problems into a neural network. This follows the same intuition of a standard PINN, however, the major difference between our work and [20] is how the eigenvalues are computed, the regularization terms used and the type of collocation points used. We observe that our approach is able to increase accuracy, reduce computational cost and provide solutions where the PINN by [20] may fail. This thesis contributes to PINN that can be adapted to solve eigenvalue problems, and in the nearest future we aim to extend this into areas where the neural network may face difficulties in obtaining the desired solutions.

CHAPTER 3

RQPINN FOR EIGENVALUE PROBLEMS

Motivated by the work of [20], we introduce Rayleigh quotient based PINN (RqPINN), for solving eigenvalue problems in 1D, particularly the Schrödinger equation (1.3). Our novel approach leverages PINN to approximate eigenfunctions, while utilizing Rayleigh quotient to obtain the corresponding eigenvalues. This approach have enhanced accuracy and convergence through redefining regularization terms in [20] and introducing different methods of choosing collocation points for training.

The RqPINN is introduced due to the following issues observed in the algorithm by [20]:

- The concept of switching symmetries is embedded into the PINN by [20], so that it can learn perfectly odd or even eigenfunctions [28]. Switching between symmetries can make the algorithm computationally expensive, resulting in longer convergence time. Besides, it means the PINN is constrained to only solve

problems with symmetric potentials $V(x)$, therefore limiting its applicability. Embedding symmetry also implies that we need to have prior solutions at hand to know when the PINN is learning a false solution and when we need to switch between symmetries.

- While the PINN in [20] is used to solve finite and multiple well problems, it has difficulty learning the simple solutions of the infinite well problem, as it always converges to a false solution after learning one or two eigenstates. The solutions of the infinite well problem 1.3.1 are all odd, so the concept of switching symmetries is not applicable here.
- Tuning the hyper parameters can be a more difficult task, since it is observed that some regularisation terms have very large value contribution than others.

To address these issues, we improve the existing PINN architecture for solving the Schrödinger equation as a specific case, thereby proposing a more efficient algorithm with increased accuracy and convergence. Figure 3.1 illustrates the PINN algorithm by [20].

3.1 RqPINN Model

We propose a PINN model for boundary value problems of the form:

$$\mathcal{G}y(x) = \lambda y(x), \quad \text{in } \Omega, \quad (3.1)$$

$$y = f_{bc}, \quad \text{on } \partial\Omega, \quad (3.2)$$

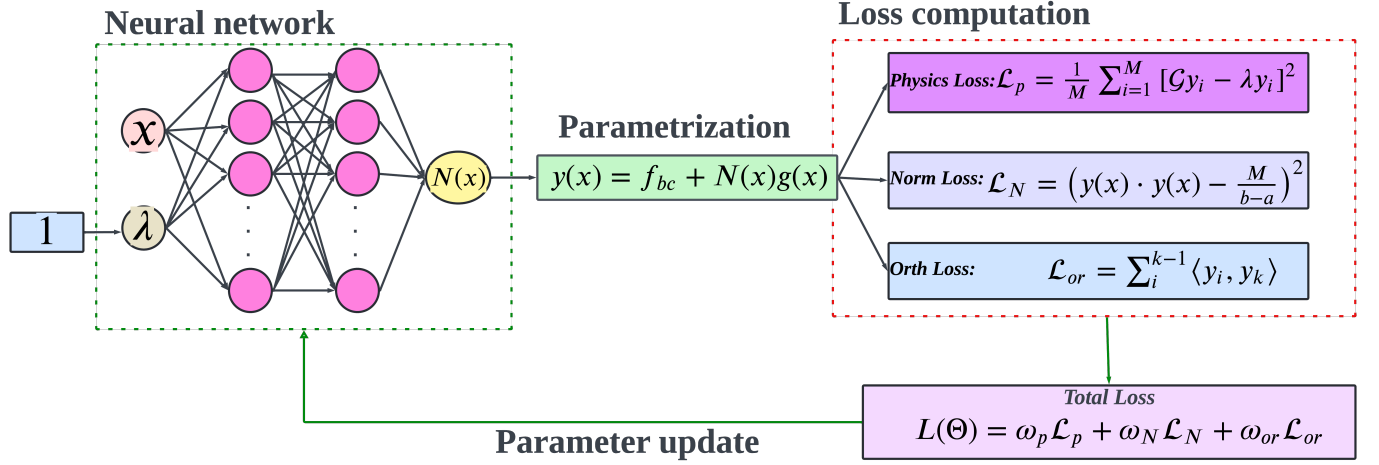


Figure 3.1: PINN proposed by [20] for solving 1D time independent Schrödinger equation. At the beginning of training, an array of constants is fed into the neurons that computes the eigenvalues.

where Ω is an open interval, $\mathcal{G}$ is an operator, λ is the eigenvalue and $y = y(x)$ is the corresponding eigenfunction. Considering the 1D time-independent Schrödinger equation, equation (1.3) is rewritten as (3.1). The next two subsections discuss the major novelty of our work, and in the following sections we describe the components of the loss function, the boundary condition and training the RqPINN. Figure 3.2 illustrates the algorithm of the RqPINN.

3.1.1 Computation of Eigenvalues

We propose a simple fully connected neural network to compute the eigenfunctions and then use Rayleigh quotient to obtain the corresponding eigenvalues, without incorporating the notion of symmetry. The Rayleigh quotient is defined as

$$\lambda = \frac{\langle y, \mathcal{G}y \rangle}{\langle y, y \rangle}, \quad (3.3)$$

where $\mathcal{G}$, y and λ are as in (3.1). The utilization of the Rayleigh quotient has eliminated the cost needed to train the eigenvalues as outputs of specific neurons. As a result, we can now calculate them directly from the associated eigenfunctions. This is observed to increase both the convergence and accuracy of the PINN.

3.1.2 Loss Computation

Regularization terms are additional components added to the loss function of a neural network during training, to further inform or guide the network to the desired solutions. In the work of [20], regularization terms are incorporated into the loss function to guide the PINN toward learning non-trivial wavefunctions that exhibit orthogonality. We adjust and normalize these regularization terms to enhance our ability to regulate the contributions of each term to the training process of the PINN, making it much more interpretable. Furthermore, we append additional regularization terms to steer the PINN towards sequentially learning the eigenvalues, and to prevent convergence to false solution.

3.2 The Loss Function

The optimization process of the RqPINN algorithm involves minimizing a loss function comprising of a physics loss and some regularization terms iteratively. This iterative process aims to derive the best function approximation for the eigenfunction and its corresponding eigenvalue. In this section, we delve into the specifics of the loss and regularization terms within our PINN architecture.

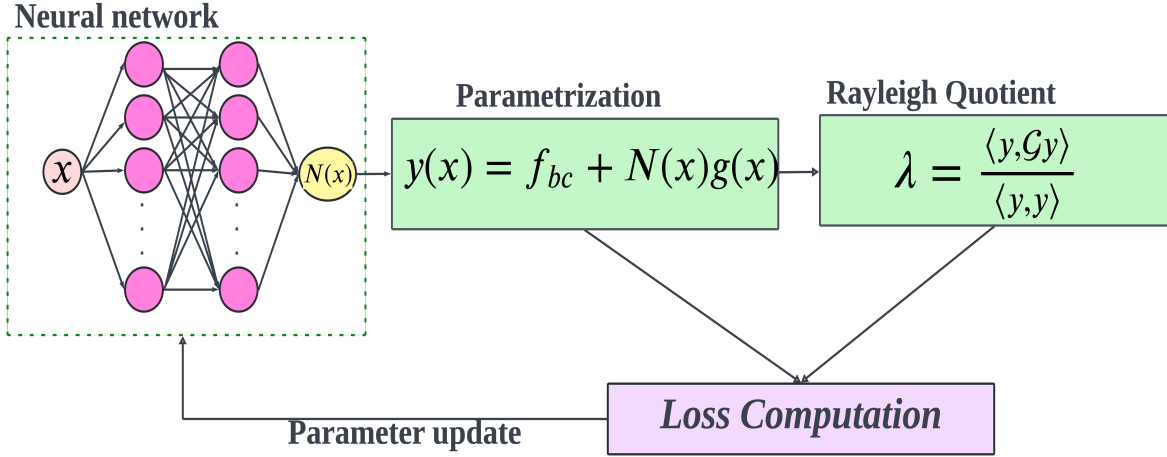


Figure 3.2: RqPINN for solving 1D eigenvalue problems.

3.2.1 Physics Loss

The Physics loss is the cornerstone of PINNs architecture, it is what distinguishes a PINN from conventional neural network. It is analogous to the loss defined in (2.13). Considering (3.1), it is defined as

$$\mathcal{L}_p = \frac{1}{M} \sum_{i=1}^M [\mathcal{G}y_i - \lambda y_i]^2, \quad (3.4)$$

where M is the number of collocation points sampled from the domain Ω , using Latin Hypercube Sampling (LHS), uniformly spaced points or orthogonal polynomial roots shifted to points in Ω .

3.2.2 Regularization Terms

We adapt certain regularization terms to provide additional guidance to the PINN in learning the desired solutions. These terms are derived from a foundational under-

standing of eigenvalue functions, with particular emphasis on the Schrödinger equation. As mentioned earlier, each regularization term is normalized to ensure equitable contribution to the optimization process. This approach facilitates easier adjustment or prioritization of specific regularization terms as required.

Next, we discuss each regularization term in details.

1. Non-triviality Loss

In the absence of any regularization term, the PINN will converge to the non-trivial solution, which also minimizes the physics loss. To avoid this, we adapt the normalization condition (1.2)

$$\int_{\Omega} [y(x)]^2 dx = 1.$$

This condition is satisfied only by the non-trivial eigenfunctions. Therefore, we define a regularization term known as the non-triviality loss. Since we consider a finite number of collocation points in the domain Ω , we incorporate (1.2) into the PINN architecture using the discrete norm. The non-triviality loss is then defined as:

$$\mathcal{L}_{nt} = \left[1 - \frac{(b-a)}{M} ||y|| \right]^2, \quad (3.5)$$

where $\Omega = [a, b]$, M is the number of collocation points in Ω and $|| \cdot ||$ is the discrete norm. We employ the square root of the PyTorch [32] dot product function to compute the discrete norm in (3.5). This implementation of the non-triviality loss guides the PINN to search for solutions that are non-trivial.

2. Orthogonality Loss

We desire that the PINN learns multiple eigenfunctions and eigenvalues of (3.1) in the space of admissible solutions. To do this, an additional information is needed to guide the network after a solution has been found to a satisfactory level. Thus, we introduce an additional regularization term based on the orthogonality property of eigenfunctions,

$$\langle y_n, y_m \rangle = 0, \text{ for } n \neq m. \quad (3.6)$$

This means that for n eigenfunctions,

$$\sum_{i=1}^{n-1} \langle y_i, y_n \rangle = 0, \text{ for } n > 1. \quad (3.7)$$

We can incorporate this as another regularization term in the PINN so that after successfully converging to an eigenfunction, the neural network is conditioned to learn another solution that is orthogonal to the previous one. This procedure is continued until we obtain n eigenfunction's that satisfies (3.7).

We define the orthogonal loss term as:

$$\mathcal{L}_{orth} = \sum_{j=1}^{k-1} \frac{\langle y_j, y_k \rangle}{||y_j|| \cdot ||y_k||}, \quad (3.8)$$

where y_k is the current eigenfunction to be learned and y_j for $j = \{1, 2, \dots, k-1\}$ are the previous eigenfunctions obtained by the PINN. This drives the PINN to search for not just multiple solutions, but solutions that are orthogonal. The norms are computed as in (3.5), while the inner product is obtained by taking the squares.

3. Eigenvalue Loss

While the regularization terms mentioned earlier can guide the RqPINN towards the desired solution, there are many instances where it converges to a false solution, with its eigenvalue lying between two previously learned eigenvalues. Hence, it becomes crucial to instruct the network on the direction to search for eigenfunctions. To address this, we introduce another regularization term called the eigenvalue loss. This term ensures that if the eigenvalues of equation (3.1) are monotonically increasing, then λ_k must be greater than λ_{k-1} , where k represents the current learned eigenvalue. The eigenvalue loss is defined as:

$$\mathcal{L}_{en} = \begin{cases} 0, & d > 0, \\ e^{-d}, & d \leq 0, \end{cases} \quad (3.9)$$

where $d = \lambda_k - \lambda_{k-1}$. The same principle applies to problems with eigenvalues that are monotonically decreasing.

4. Nodal Loss

Incorporating the eigenvalue loss term implies that if the RqPINN overlooks an eigenpair, it will not revisit it. To address this, we propose a regularization term that deters the network from deviating from any eigenpair. This term is formulated based on the number of peaks in the eigenfunction.

In many scenarios, the number of peaks in the eigenfunction increases by one as the eigenfunction level rises (e.g., the ground state eigenfunction has one peak). This

property can be leveraged to devise a regularization term that ensures the presence of the required number of peaks in the eigenfunction. This term is defined as:

$$\mathcal{L}_{nl} = |P_{est}(y_k) - P_{exact}|. \quad (3.10)$$

Here, P_{est} represents a function that estimates the number of peaks in the currently learned eigenfunction, and P_{exact} denotes the expected number of peaks.

However, it is important to note the following:

- A proper way to define the nodal loss may be necessary since the definition is a constant which when differentiated (during back-propagation) is zero. Consequently, this definition (3.10) of the nodal loss might not significantly impact the training process.
- Introducing too many regularization terms can lead to competition with the physics, potentially increasing instability in the training process. It also increases the non-convexity of the loss function, making it difficult to converge to minimum during training.
- Proper weighting of the regularization terms is still required, and these weights need to be carefully tuned to ensure accurate solutions.

3.3 Boundary Condition

In contrast to the PINN in [33], we employ the use of a parametric equation as outlined in [20]. This approach involves incorporating the boundary condition directly into

the PINN architecture, rather than introducing an additional loss function for the boundary. A similar technique has been utilized in [24].

In this thesis, we focus on Dirichlet's boundary condition and formulate its incorporation into the PINN as follows:

$$y(x) = f_{bc} + g(x)N(x), \quad (3.11)$$

where f_{bc} represents the boundary value, $N(x)$ denotes the initial output of the PINN, and $g(x)$ is a parametric function that vanishes on the boundary $\partial\Omega$. We define $g(x)$ as:

$$g(x) = (1 - e^{-(x-a)})(1 - e^{(x-b)}), \quad (3.12)$$

for $\Omega = [a, b]$. This approach to enforcing boundary conditions is called "hard boundary conditions", and it offers a novel perspective on enhancing the performance of the PINN model by eliminating the need to introduce an additional term in the loss function. A plot of the parametric equation (3.12) is shown in Figure 3.3.

3.4 Training the PINN

The overall loss function utilized to train the RqPINN comprises the physics loss and the previously defined regularization terms. It is expressed as:

$$\mathcal{L}(\Theta, \lambda_i) = \omega_p \mathcal{L}_p + \omega_{nt} \mathcal{L}_{nt} + \omega_{orth} \mathcal{L}_{orth} + \omega_{en} \mathcal{L}_{en} + \omega_{nl} \mathcal{L}_{nl}, \quad (3.13)$$

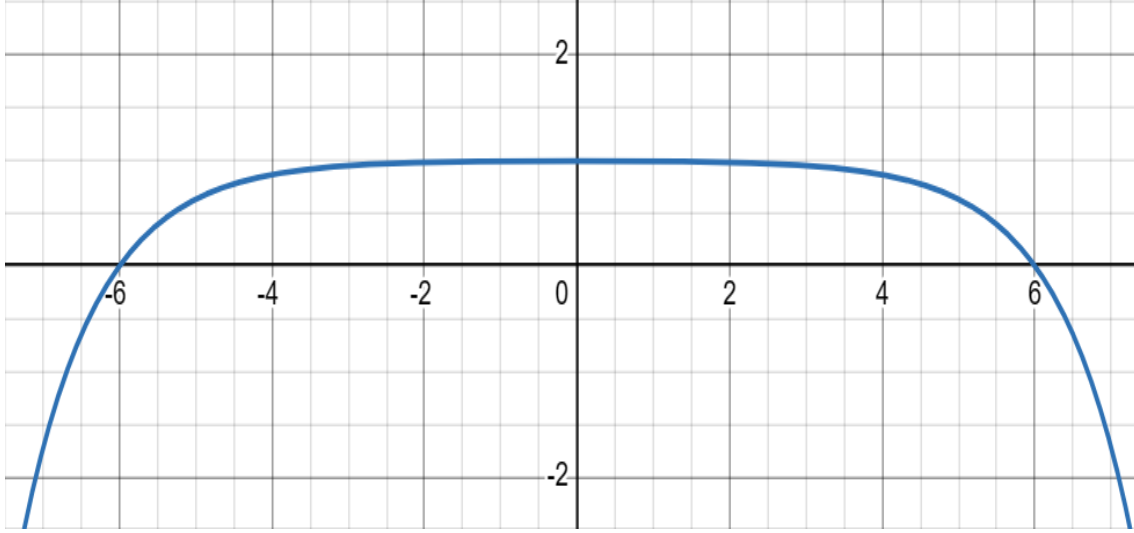


Figure 3.3: Plot of the parametric equation (3.12), for $\Omega = [-6, 6]$.

where Θ denotes the affine functions containing the parameters (weights and biases) needed to be updated during training of the PINN, λ_i represents the current eigenvalue being learned, and ω 's are fixed parameter values called the weights of the loss. They determine the relative contributions of the DE loss and each regularization term during the neural network training process. They are among a set of parameters called hyperparameters, and are meticulously chosen before training to ensure convergence to the desired solutions. It is noteworthy that when $i = 1$, ω_{orth} and ω_{en} are set to zero.

To initiate the network training process, we start by randomizing the parameters of Θ using the Glorot uniform initializer [13]. Subsequently, these parameters are updated via back-propagation, employing optimization procedures similar to (2.8).

We select M collocation points within the domain Ω , where the neural network undergoes training. At these chosen points, the neural network generates approximations of the eigenfunction for which the boundary condition is then enforced using parametric equation (3.11). Next, this eigenfunction is utilized to calculate the corresponding

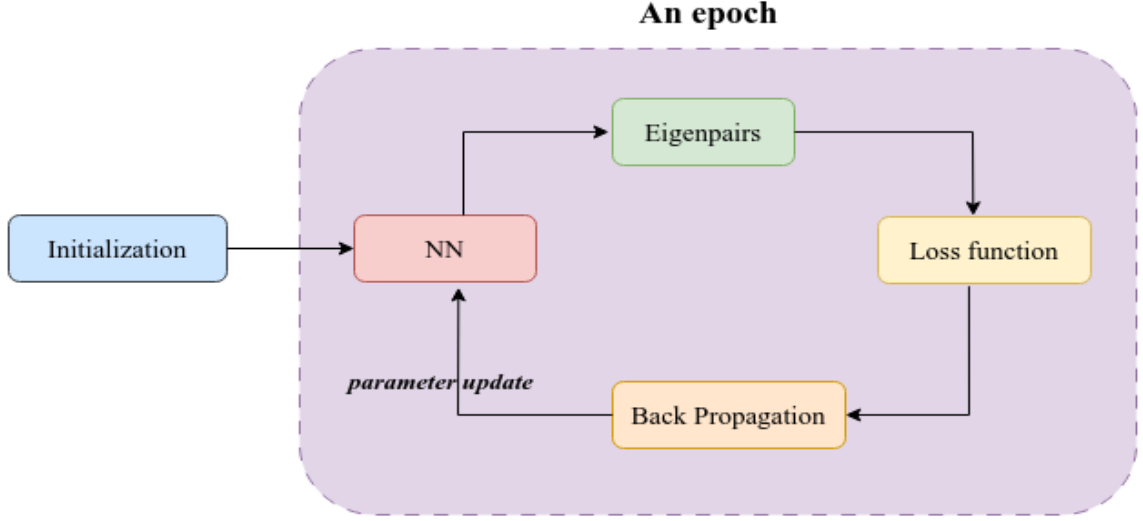


Figure 3.4: Illustration of the PINN training process.

eigenvalue via (3.3). The computed values of the eigenfunction and eigenvalue are then employed to estimate the loss function in (3.13), which is in turn used in the optimization process to update the weights and biases of the PINN. This iterative process continues until the loss function is minimized, resulting in the best approximation for both the eigenvalue and eigenfunction.

Once the loss function reaches a certain minimal value, we conclude that the PINN has converged to an eigenpair. At this stage, we update the orthogonal loss term, prompting the PINN to begin searching for the next set of eigenpairs. The overall iterative process during training can be seen in Figure 3.4.

CHAPTER 4

RQPINN FOR QUANTUM EIGENVALUE PROBLEMS

In this chapter, we implement the RqPINN in Figure 3.2 to the Schrödinger equation (1.3). We consider quantitative metrics in evaluating the results, and make a comparison with the PINN by [20].

4.1 Metric and Datasets

We consider some 1D eigenvalue problems with Dirichlet’s boundary condition, and evaluate the performance of the proposed approach through some analysis. While the concept of stability in training a PINN may lack a precise definition due to various contributing factors such as hyper-parameter tuning, initialization, learning rate, and choice of loss function, it can be characterized by the behavior of the solution (eigenvalue or eigenvector) during training. Instances of abrupt or substantial oscillations in the solution indicate instability and divergence, even if the final solution closely

approximates the exact solution. In such cases, convergence occurs by chance rather than through systematic convergence.

We measure the convergence of the RqPINN using moving average. This is done by smoothing the learning curve over a predefined window. This moving average removes the oscillations and lets us determine better if the learning curve is converging.

If a reference solution is defined as $\{x_i, \psi_i\}_{i=1,2,\dots,M}$, and the predictions of the proposed PINN at the same points are $\{x_i, \psi_\theta(x_i)\}_{i=1,2,\dots,M}$. Then we evaluate the performance (accuracy) of the PINN with respect to the mean absolute error of the eigenfunction squared,

$$\epsilon_\psi = \frac{1}{M} \sum_{i=1}^M \left| |\psi_i|^2 - |\psi_\theta(x_i)|^2 \right|, \quad (4.1)$$

and that of the eigenvalue is obtained based on the relative error

$$\epsilon_\lambda = \frac{|\lambda_{exact} - \lambda_{pinn}|}{|\lambda_{exact}|}, \quad (4.2)$$

where λ_{exact} is the referenced eigenvalue solution and λ_{pinn} is the PINN obtained solution after some iterations (epochs).

We consider M collocation points from Latin-Hypercube sampling, uniformly spaced collocation points, and some orthogonality-based collocation points (e.g., Chebyshev, Hermite). Except otherwise stated, we consider two hidden layers with $\frac{M}{2}$ neurons in each layer. Finally, we choose the sine activation function and Adam optimizer with a learning rate of 10^{-2} to train the PINN.

All simulations are run using PyTorch on Google Colab with NVIDIA T4 Tensor Core GPU, and so training times are with respect to simulations using this setup.

4.2 Performance Analysis of RqPINN

To demonstrate the performance of the proposed approach, We consider the first eigenpair obtained from the following problem:

$$\begin{cases} y'' + 100y = \lambda y, & \text{in } \Omega, \\ y = 0, & \text{on } \partial\Omega. \end{cases} \quad (4.3)$$

Our analysis begins by comparing the accuracy in terms of how the collocation points is sampled and the number of collocation points used to train the PINN. The neural network training parameters for this comparison include 20,000 epochs, with a network structure of 3 layers, each containing 25 neurons. Tables 4.1 and 4.2 summarizes our findings regarding the eigenvalue and eigenfunction, respectively. Generally, we observe an increase in accuracy as we move down the tables, with the method of selecting collocation points also influencing the results. We also deduce that the LHS technique for selecting the collocation points outperforms other methods in terms of accuracy.

We consider another comparison, this time focusing on the impact of varying the number of neurons and layers on the performance of the PINN. We select 1001 collocation points and 20,000 epochs, while using the LHS technique to choose the points. Figure 4.3 and 4.4 provide a concise overview of our observations. It becomes evident that increasing the number of layers does not significantly enhance accuracy, unlike the effect observed when increasing the number of neurons. However, it's worth

No. of col pts	Method of selection			
	Chebyshev	Hermite	Uniform	LHS
11	3.7463e-05	1.0045e-05	1.7956e-06	5.2510e-06
55	6.5882e-07	1.0507e-05	1.0259e-07	1.3299e-07
101	1.1753e-06	2.0415e-06	6.6708e-08	2.5559e-08
501	3.3717e-06	3.6099e-06	1.7940e-08	1.9958e-09
1001	5.5934e-06	2.7322e-06	1.7940e-08	8.0192e-12
10001	2.8506e-06	6.5569e-06	1.7940e-08	2.0372e-09

Table 4.1: Relative error for the largest eigenvalue of (4.3) corresponding to different collocation points and method of selection.

No. of col pts	Method of selection			
	Chebyshev	Hermite	Uniform	LHS
11	0.0014	4.6466e-4	5.0410e-4	1.9927e-4
55	2.757e-4	2.7150e-4	2.9186e-4	1.3855e-4
101	2.8472e-4	9.1909e-5	1.2981e-4	1.9583e-4
501	2.0978e-4	8.2235e-5	1.2416e-4	4.4491e-5
1001	6.6726e-5	5.1863e-5	4.4064e-5	3.8424e-5
10001	3.035e-4	1.3553e-4	9.3897e-5	9.6447e-5

Table 4.2: Mean Absolute Error for squared of the eigenfunction obtained with different number of collocation points and methods of selection.

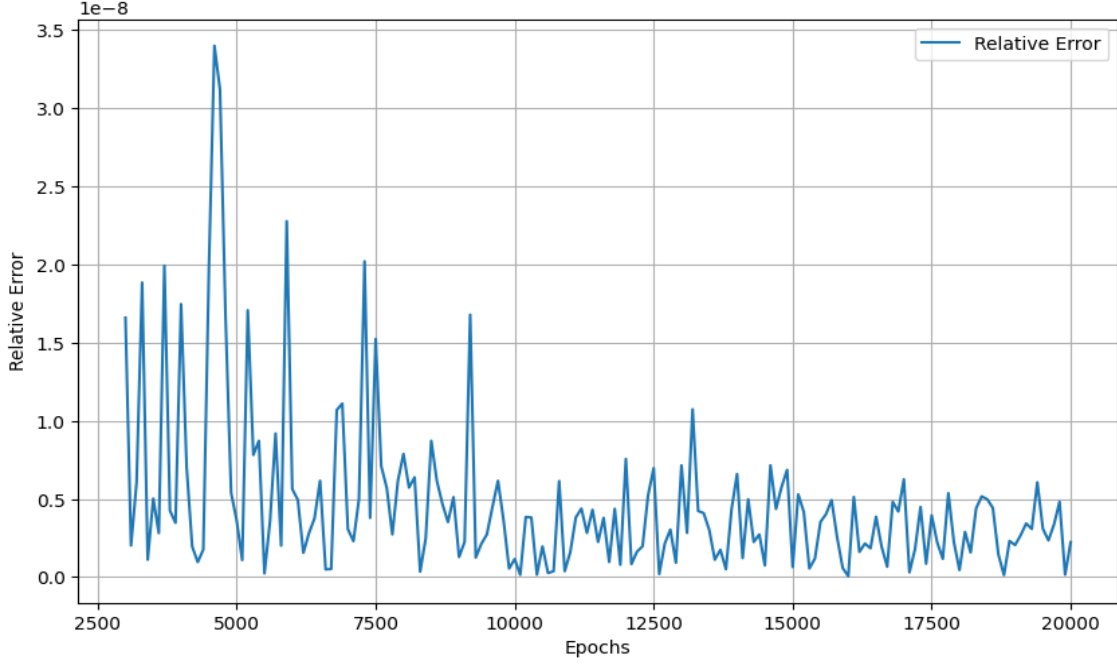


Figure 4.1: Plot of the relative error during training of the RqPINN.

noting that the convergence time varies across these configurations.

Layers	10	20	30
2	6.0852e-09	7.7527e-08	8.6423e-10
4	2.1044e-09	1.0653e-09	2.2607e-09
6	1.5776e-09	1.5440e-09	5.9467e-10
8	1.3439e-09	2.5989e-09	7.1389e-09

Table 4.3: Relative eigenvalue errors with respect to different neurons and layers

A comparison based on different activation functions is not done. This is due to the periodicity of the solutions, as it is observed that the sinusoidal activation function performs best. Furthermore, such extensive experiments regarding the impact of activation functions has been carried out in [39]. Figures 4.1 and 4.2 shows the plots of the relative error and the moving average during training. The convergence of the

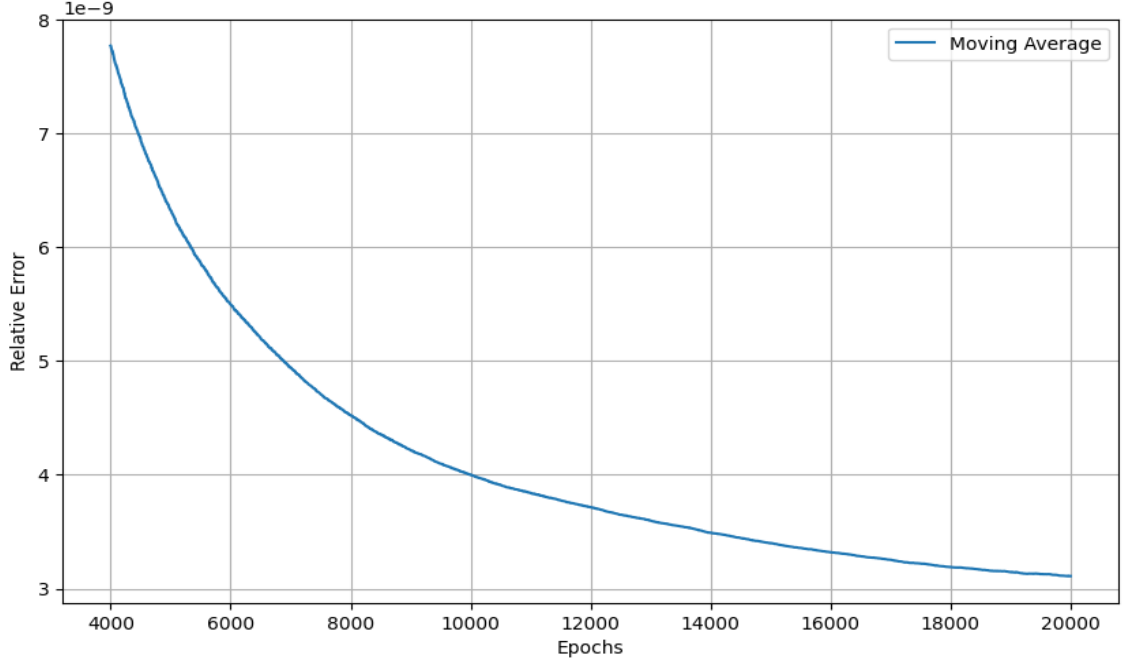


Figure 4.2: Plot of moving average calculated over a window of 1000 epochs. The moving average removes the oscillations observed in Figure 4.1, demonstrating a noticeable convergence of the RqPINN.

RqPINN is obvious from 4.2.

To further demonstrate the efficiency of the RqPINN, we compare the result of the first eigenvalue obtained by RqPINN and the PINN by [20] for the infinite potential well problem 1.3.1. We train both algorithms with 201 collocation points with 5000 epochs, and the collocation points for RqPINN is selected through LHS. Figures 4.3 and 4.4 illustrates the rapid convergence of RqPINN to the exact solution and the increased accuracy when compared with the PINN by [20].

Layers	Neurons		
	10	20	30
2	4.6249e-05	2.1656e-04	2.6212e-05
4	5.5113e-05	7.9274e-05	4.3441e-05
6	8.4463e-05	9.8429e-05	5.3523e-05
8	5.7831e-05	8.5529e-05	1.1219e-04

Table 4.4: Mean absolute error of the squared eigenfunction with different neurons and layers

4.3 RqPINN for 1D Time-Independent Schrödinger Equation

We now investigate the performance of the proposed PINN architecture on the Schrödinger equation with different potentials $V(x)$. We consider the infinite and finite potential wells.

In these problems, we are interested in obtaining multiple eigenvalues and eigenfunctions. Thus, in addition to the loss function and regularization terms, we consider the following in the neural network architecture to ensure effective training of the required solutions.

Transfer Learning

Transfer learning refers to using the knowledge (parameters) of a pre-trained network as a starting point to train a network on a related task. Transfer learning is used to improve a machine learning model or algorithm from one domain by transferring information from a related domain [41]. It helps to reduce the need and effort to

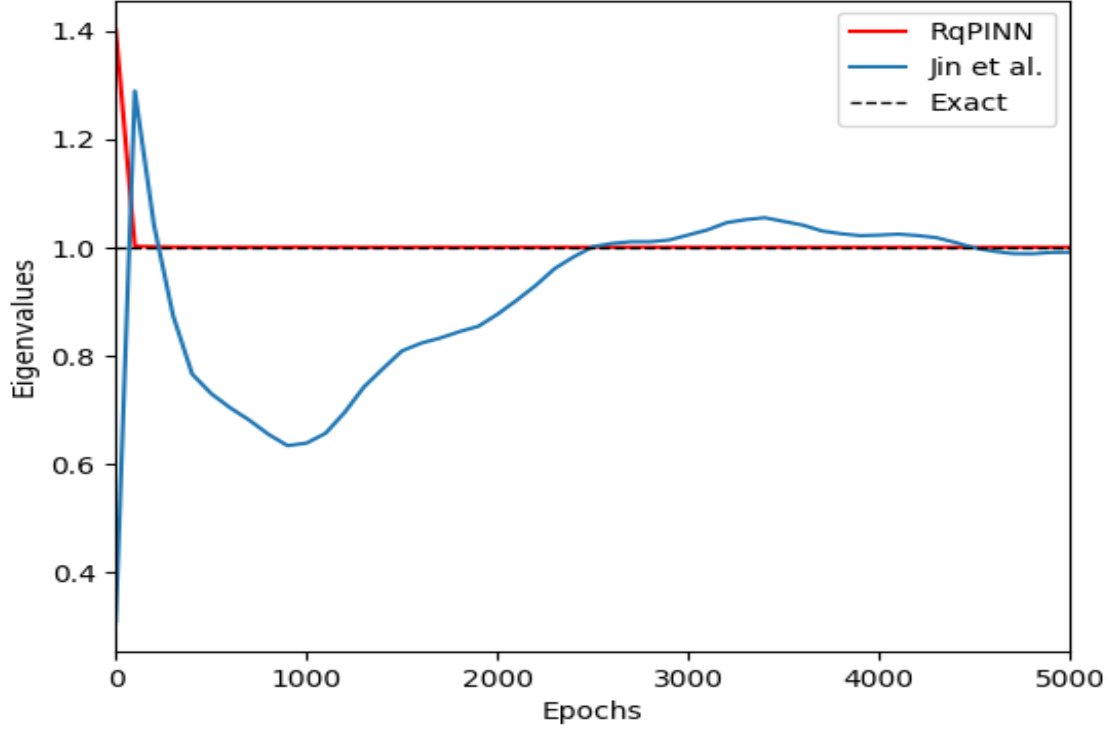


Figure 4.3: Plot of the eigenvalues during training of RqPINN and PINN by [20]. RqPINN converges rapidly to the exact solution while the PINN by [20] oscillates for over 4000 epochs before it starts to converge.

recollect training data [30], such that the new model can be fine-tuned using smaller dataset and less computational resources compared to training from scratch.

To obtain multiple eigenpairs, we employ transfer learning, leveraging parameter values from previously learned eigenpairs to train subsequent ones. This approach not only accelerates convergence to the desired solutions but also mitigates the risk of converging to false solutions that may arise from random re-initialization of the neural network.

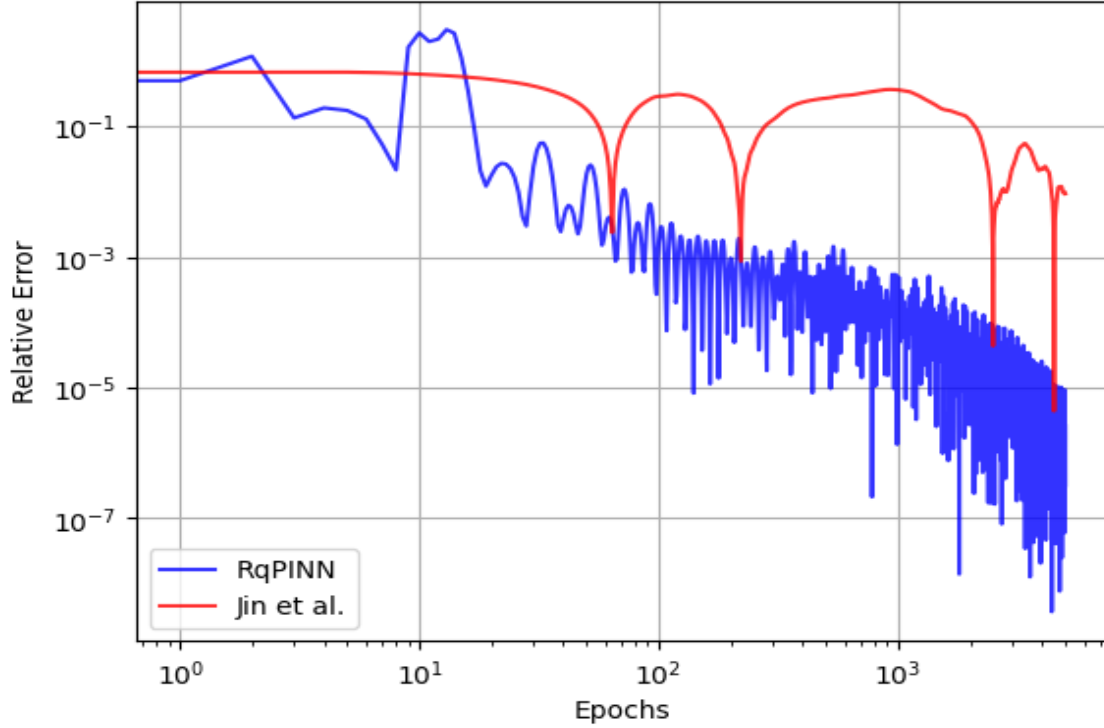


Figure 4.4: Plot of the relative errors between RqPINN and PINN by [20]. RqPINN has superior accuracy compared to the other algorithm.

Stopping Criteria

One way to train multiple eigenpairs is to fix the number of epochs needed to train each of them. However, this may not be computationally efficient as the desired accuracy of each eigenpair may not be achieved. Moreover, considering the orthogonality loss term (3.8), it becomes apparent that the accuracy of subsequent eigenfunctions depends on those previously learned. Thus, smaller number of epochs may result in false solutions, while large number of epochs may yield computational inefficiency and even over-fitting.

A robust approach is to specify a convergence value as a stopping criteria. This is done by using the rolling mean over a specified window on the eigenvalues obtained in each epoch. We conclude that an eigenpair is obtained once the rolling mean reaches

the specified convergence value, prompting the neural network to automatically initiate the search for the next eigenpair.

However, it has been observed that the network swiftly reaches this convergence value for the first eigenpair with less accuracy, subsequently impacting the convergence of subsequent eigenpairs due to the orthogonality loss's dependence on previously learned eigenfunctions. This phenomenon prolongs the training time for subsequent eigenpairs before reaching the convergence value. To address this issue, we impose a minimum training time e.g. 3000 epochs, for each eigenpair before the network evaluates the convergence value. For both the infinite and finite well problems, we set the minimum training time to 9000 epochs.

Gradient Clipping

As the training of the PINN proceeds (i.e. after learning some eigenpairs), sometimes the network becomes unstable and numerically overflows leading to Not a Number (NaN) values of the gradients. We observe this is due to gradient explosion, which is the counterpart of gradient vanishing.

Gradient explosion is a phenomenon that occurs during training of a deep neural network, where the gradients (derivatives) of the loss functions with respect to the network parameters becomes extremely large and blows up. This leads to instability, divergence and NaN values caused by updating the parameters with very large magnitudes during optimization.

To mitigate this, we employ gradient clipping. This involves restricting or "clipping" the magnitude of the gradients to a predefined threshold to prevent them from

becoming too large.

4.3.1 Infinite Potential Well using RqPINN

We now consider the infinite potential well problem. Recall in 1.3.1, the potential $V(x)$ is defined as:

$$V(x) = \begin{cases} 0, & 0 \leq x \leq L, \\ \infty, & \text{otherwise.} \end{cases}$$

The exact wavefunction is

$$\psi_n(x) = \sqrt{\frac{2}{L}} \sin\left(\frac{n\pi x}{L}\right),$$

and the corresponding energy E_n is obtained as

$$E_n = \left(\frac{n\pi}{L}\right)^2,$$

for $n \geq 0$.

We utilize the proposed PINN to solve this problem for the first 6 eigenpairs, taking $L = \pi$ and the coefficient of the derivative in the Schrödinger equation (1.3) as 1. Using 201 collocation points and a convergence (stopping) criteria of $5e - 9$, Figures 4.5, 4.6 and 4.7 depicts the performance of our PINN in obtaining the first 6 wavefunctions. It can be seen that it perfectly approximates the wavefunctions except a little difference at the peaks of the 6th wavefunction. This little difference can be removed by allowing more running time, reducing the value of the stopping criteria

or increasing the number of collocation points for training.

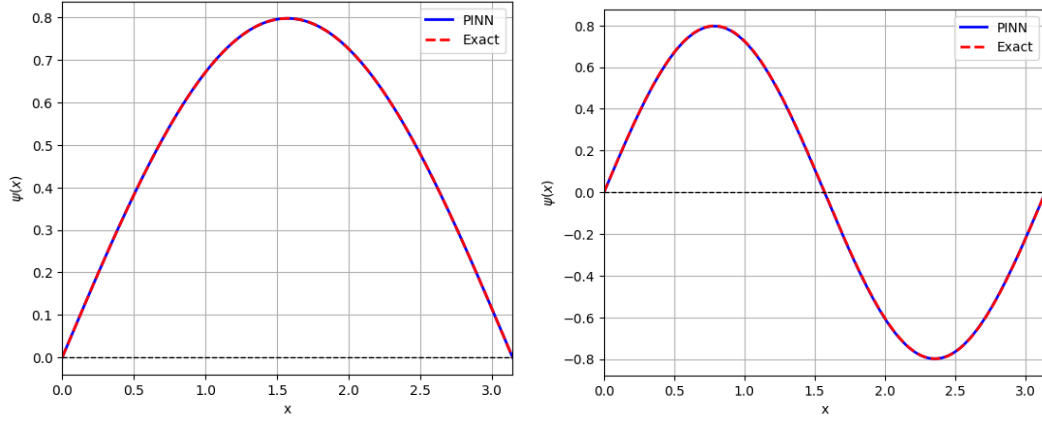


Figure 4.5: First and second wavefunctions of the infinite potential well

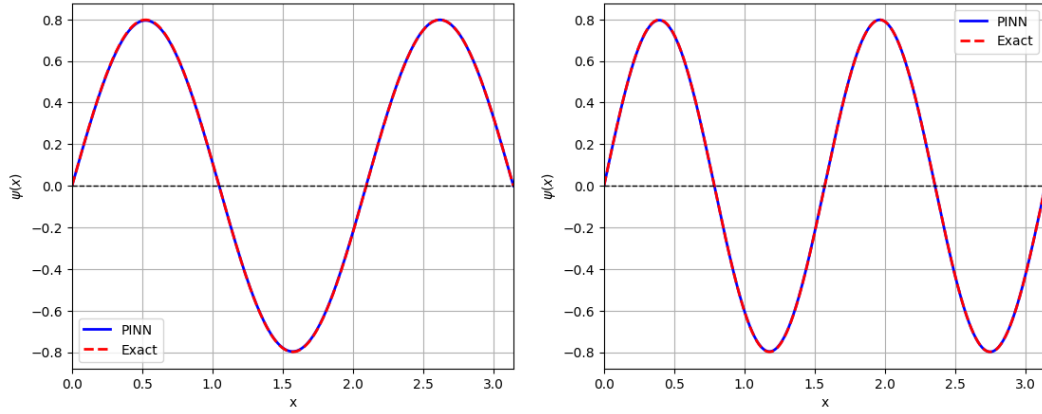


Figure 4.6: Third and fourth wavefunctions of the infinite potential well

Table 4.5 shows the values of the 6 Energy levels (eigenvalues) obtained by PINN and makes a comparison with the exact energy levels of the infinite well problem, using the relative error defined in (4.2). PINN performs very well with some energy levels accurate up to the 6th decimal place. The last column of the table indicates the epoch number at which the stopping criteria is satisfied and the network is updated to start searching for the next eigenpair.

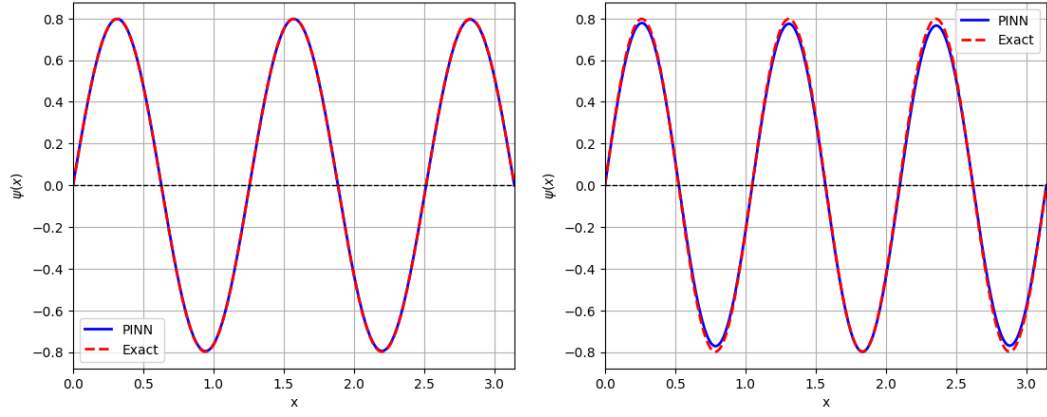


Figure 4.7: Fifth and sixth wavefunctions of the infinite potential well

Energy Level	Exact	PINN	Relative Error	Epochs
E_1	1.0000	0.9999	5.6646×10^{-7}	9,001
E_2	4.0000	4.0000	9.0726×10^{-7}	18,014
E_3	9.0000	8.9999	1.1859×10^{-6}	27,065
E_4	16.0000	15.9999	1.0175×10^{-7}	36,109
E_5	25.0000	24.9998	5.7576×10^{-6}	45,210
E_6	36.0000	35.9915	2.3454×10^{-4}	55,622

Table 4.5: Comparison of Energy Levels

The comparison by the mean absolute error (4.1) of the exact wavefunctions and those obtained by PINN is shown in table 4.6. In general, it is expected that accuracy reduces as we move right the table. This is due to the orthogonality loss term, whose current value is impacted by the accuracy of the previously learned wavefunctions.

	ψ_1	ψ_2	ψ_3	ψ_4	ψ_5	ψ_6
MAE	9.2085e-05	1.3901e-03	1.4661e-03	8.7987e-04	1.883e-03	1.818e-02

Table 4.6: Mean Absolute Error (MAE) for ψ values

A plot of the loss function and the energy levels at each epoch is shown in Figure 4.8. Each spikes of the loss function plot indicates where the stopping criteria is satisfied and the network's orthogonality loss term is updated to start searching for the next eigenpair. The energy history plot displays the energy levels at each epochs during the training process. The horizontal dashed lines represents the exact energy levels and the jumps also indicates where the stopping criteria is satisfied and network is updated. It can be seen that the PINN converges rapidly to the energy levels and oscillates around the exact value until the stopping criteria is met.

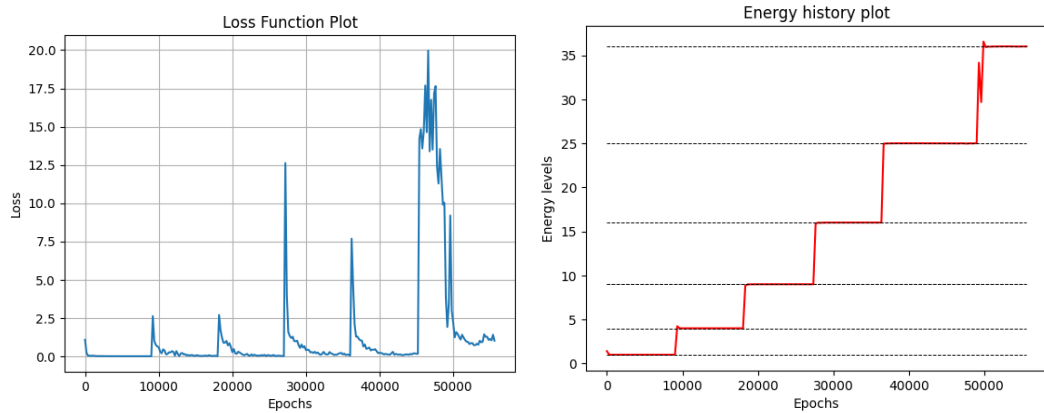


Figure 4.8: Plot of loss function and energy levels during training

4.3.2 Single Finite Potential Well using RqPINN

As a final illustration of the capability of RqPINN, we now solve the single finite potential well problem. We consider the potential as in (1.11) with $V_0 = 10$ and we restrict our domain to $\Omega = [-20, 20]$, so that $\psi(-20) = \psi(20) = 0$. The width of the well is taking to be between -5 and 5 , i.e. $L = 5$ and for simplicity, $\hbar$ and m of the Schrödinger equation (1.3) is taken as 1. Figure 4.9 gives an illustration of the well discussed.

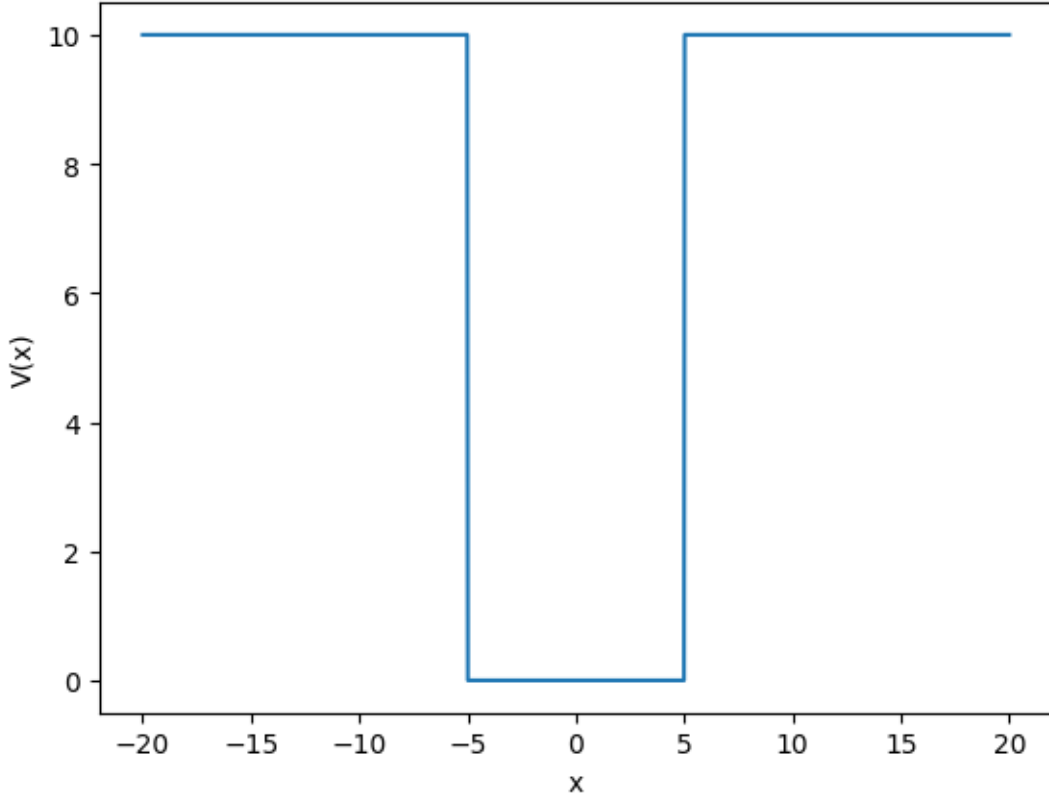


Figure 4.9: Illustration of the finite potential well.

Like the infinite potential well problem, the finite well problem have infinite eigenpairs. There are discrete number of eigenpairs inside the well and continuum of eigenpairs outside the well. The eigenpairs outside the well indicates the particle is in

a free state. Energy levels E less than V_0 indicates the particles are inside the well, while energy levels greater than V_0 represents energies of particles outside the well. We are only concerned with the behaviour of particles inside the well i.e. $E < V_0$.

Due to how the eigenvalue is computed in the RqPINN architecture (3.3) and the sinusoidal activation function used (outputs will be between -1 and 1), this means that RqPINN starts learning eigenpairs outside the well (i.e. $E > V_0$). Thus, skipping all wavefunctions and energies inside the well.

One approach to solve this is to introduce a new loss term called the "discreet loss", which is defined as

$$\mathcal{L}_{dis} = \begin{cases} e^{(E-V_0)}, & E - V_0 \geq 0, \\ 0, & \text{otherwise.} \end{cases} \quad (4.4)$$

While this can be helpful in informing the neural network to learn solutions inside the well, adding additional losses increases the networks complexities, computational cost and even makes it more ill-conditioned (unstable).

Another more efficient approach is through utilizing the physics, i.e. prior knowledge of the above problem. This is to train the PINN for a certain number of epochs (say 5000) to approximate the gaussian defined as

$$\psi_{pred} = e^{(x^2)}. \quad (4.5)$$

This is just like the predictor-corrector method in Numerical methods, where a method

is used to begin the approximation then the main method is used to correct the approximation to better accuracy.

This provides an efficient physics-based approach to enforce the PINN to start searching for solutions inside the well. It updates the network's parameters to approximate a function inside the well, then we correct this into our desired solution.

Figure 4.10 shows the output of the neural network, after pre-training on the gaussian.

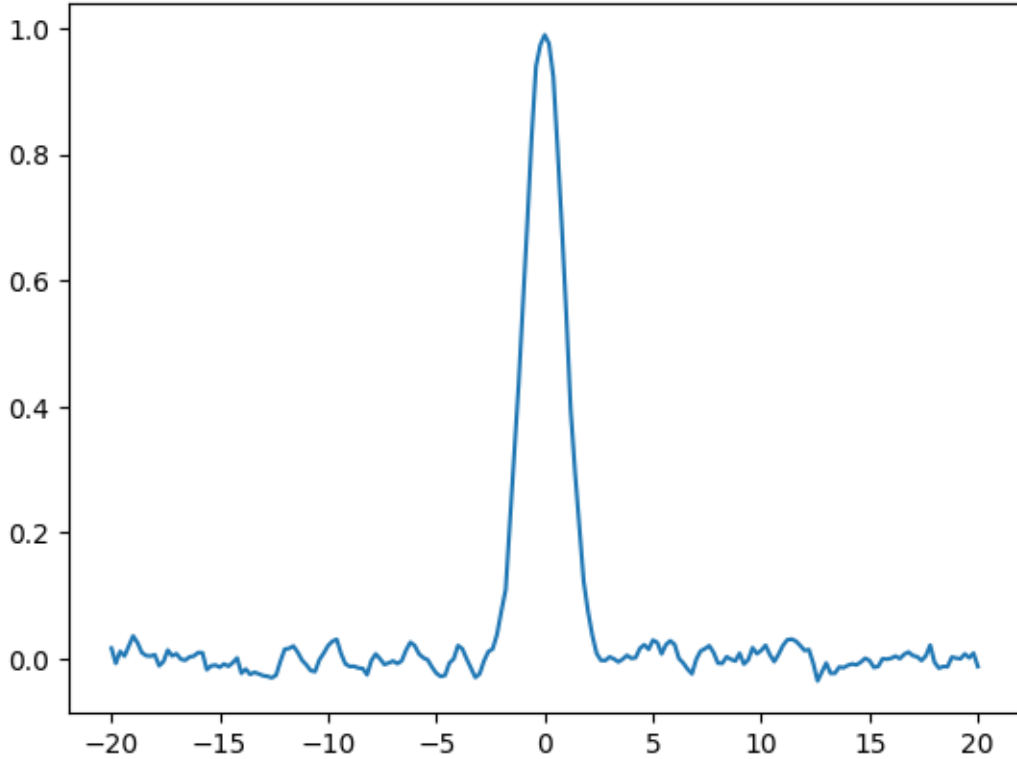


Figure 4.10: Output of PINN after approximating the gaussian (4.5) for 5000 epochs.

With the above predictor-corrector approach, we solve for the first 6 wavefunctions and their corresponding energy levels. The PINN is trained on 201 collocation points from LHS, with a convergence (stopping) criteria of $5e - 9$. The obtained solutions are then compared with that of MATLAB's Chebfun toolbox.

The plots of the first 6 wavefunctions obtained by PINN are shown in Figures 4.11, 4.12, and 4.13. Table 4.7 details the comparison of the energy levels obtained by MATLAB's `chebfun` and those obtained by PINN, then the relative errors are obtained accordingly.

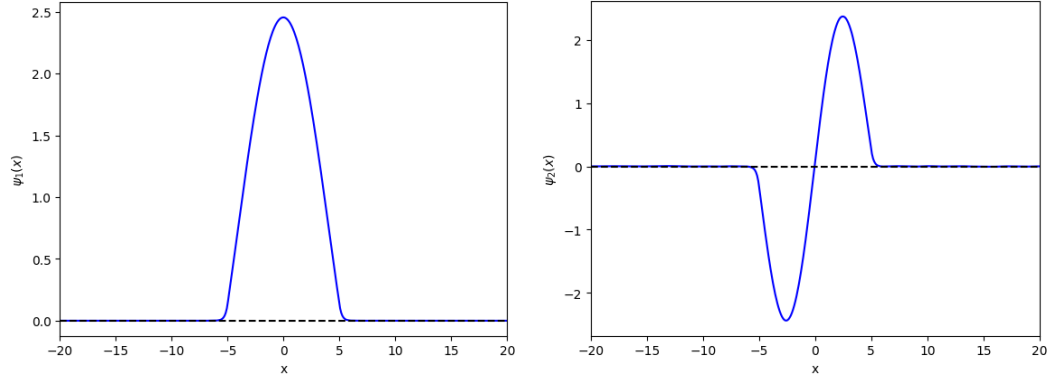


Figure 4.11: First and second wavefunctions of the finite potential well

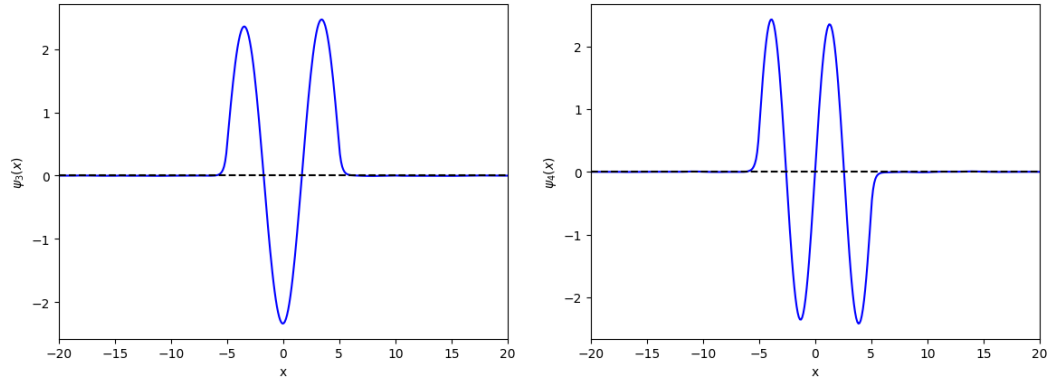


Figure 4.12: Third and fourth wavefunctions of the finite potential well

All 6 wavefunctions are plotted together inside the well and is shown in Figure 4.14. Finally, Figure 4.15 shows the loss function plot and the plot of the energy levels at each epochs. The large error at the beginning of the loss function plot stems from the correction of the gaussian function that was initially approximated.

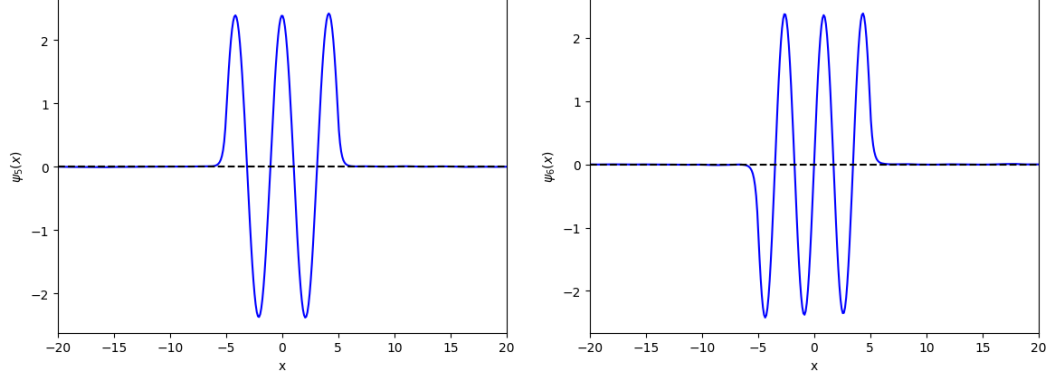


Figure 4.13: Fifth and sixth wavefunctions of the finite potential well

Energy Level	Chebfun	PINN	Relative Error	Epochs
E_1	0.0452	0.0456	8.2654×10^{-3}	16,132
E_2	0.1808	0.1816	4.4045×10^{-3}	25,244
E_3	0.4067	0.4067	1.5191×10^{-4}	34,491
E_4	0.7226	0.7246	2.6863×10^{-3}	43,550
E_5	1.1284	1.1332	4.2694×10^{-3}	53,023
E_6	1.6236	1.6263	1.6833×10^{-3}	62,164

Table 4.7: Comparison of energy levels of the finite well problem obtained by RqPINN with the chebfun toolbox. The last column indicates the epoch number at which the stopping criteria is satisfied.

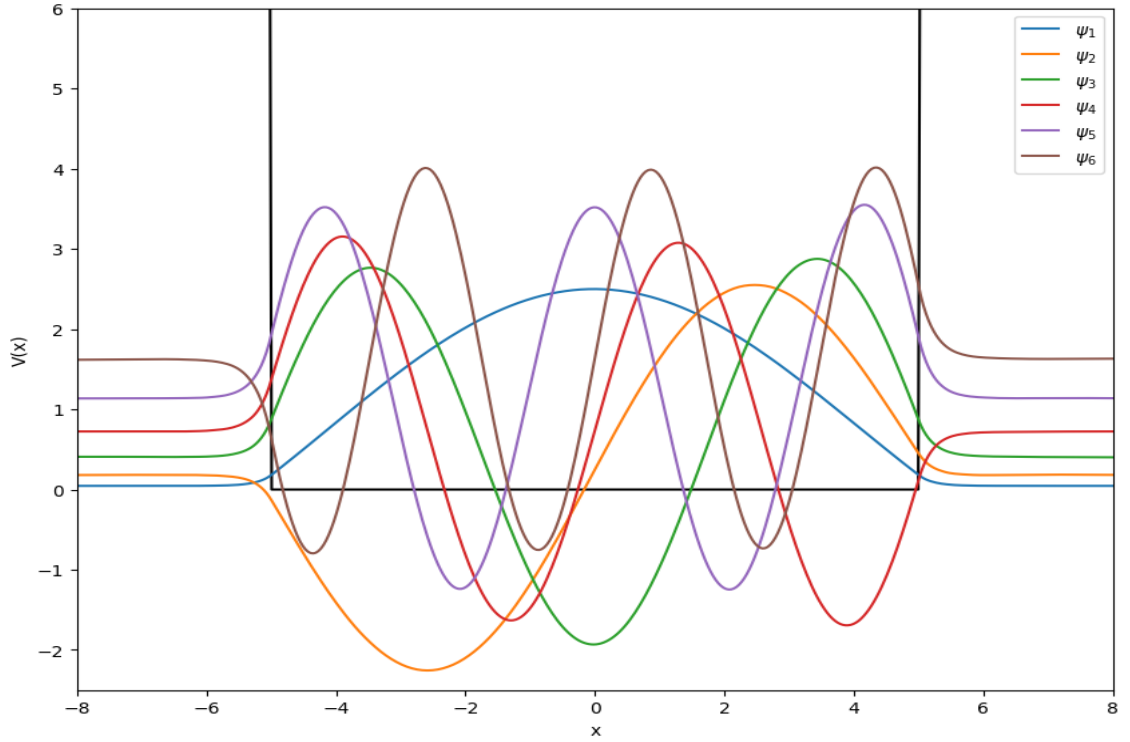


Figure 4.14: Plot of all six learned wavefunctions by RqPINN inside the finite well.

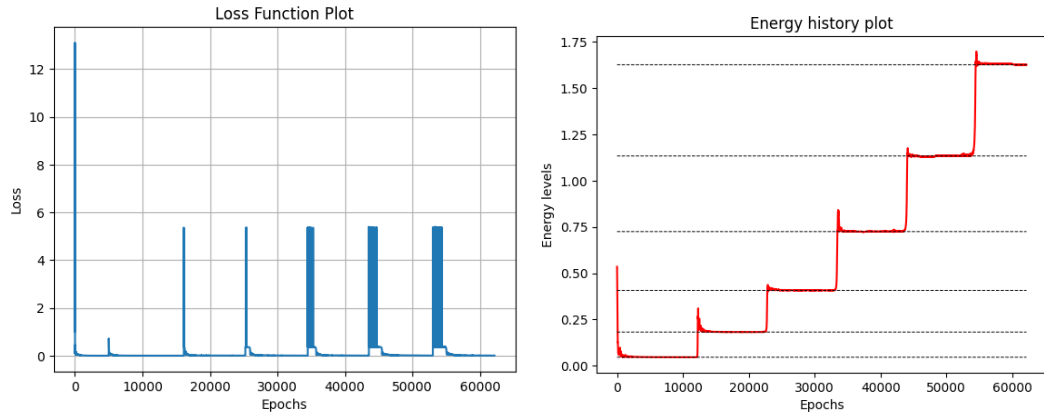


Figure 4.15: Plot of loss function and energy levels during training of RqPINN for the finite well problem. Each spike on the loss function plot (left) indicates that the stopping criteria is met and the network is updated to search for the next solution pair. The dashed horizontal lines of the energy plot (right) indicates the required energy level, and RqPINN can be seen oscillating very close around them until the stopping criteria is met.

CHAPTER 5

REMARKS AND FUTURE WORKS

In this thesis, a Physics-Informed Neural Network (PINN) for solving eigenvalue problems is introduced, which we called RqPINN. This PINN compute eigenfunctions and their corresponding eigenvalues. The parameterized output of the neural network is used to approximate the eigenfunction while the Rayleigh quotient is used to obtain the corresponding eigenvalue. The obtained eigenpairs are required to minimized the loss function and some regularization terms that are needed to guide the PINN to the desired solutions. We utilize automatic differentiation to compute the derivatives of the PDE, and the parameters of the PINN are updated through back-propagation. We perform an analysis on the PINN by solving a simple eigenvalue problem and finally, we demonstrate the performance of the PINN on the infinite and single finite well problem. We see that the PINN can easily solve both problems.

The PINN performs well in solving these problems, but it is still important to dis-

cuss some general remarks in it's training, and possible way forward for this research.

5.1 General Remarks

While PINN can be adjusted to solve a variety of problem, it is not a one-size-fits all approach as it is accompanied by certain limitations that need to be considered before training an effective PINN .

In the context of the RqPINN, the training can be computationally expensive, particularly for complex potentials and the use of large datasets. This is due to the iterative nature of PINN training and the need to perform extensive computations during each epochs. The accuracy of RqPINN is dependent on the size of datasets used in training, and this can be time consuming and so computationally expensive. Besides, RqPINN is highly sensitive to hyperparameters such as the number of epochs, learning rate, number of layers, number of neurons per layer, and regularization parameters. Choosing the optimal hyperparameters is a very challenging task and often requires extensive tuning.

Furthermore, how a neural network arrives at a prediction is still not known. Neural networks can effectively learn complex patterns and relationships from data, but there are limited insights into the specific features or decision-making processes it relies on. This makes a neural network lack interpretability, leading to what is commonly referred to as black-box nature of neural networks. This black-box nature makes it difficult to understand the underlying mechanism driving predictions of the PINN, and can be a significant drawback in solving problems such as that of eigenvalues,

where model transparency is crucial.

5.2 Future Work

The RqPINN model has been shown to solve the infinite and single finite well problems. However, more problems still need to be explored to better interpret the RqPINN model, to identify where it may fail and how it can be used to generalize different kinds of eigenvalue problems.

It is also important to note that research in the use of PINN are still in the early stage, and most of the advantages of PINN are yet to be exploited. This makes traditional computational techniques out-perform PINN in some specific tasks. A huge contribution of RqPINN to literature will be utilizing it to solve eigenvalue problems in higher dimensions.

REFERENCES

- [1] I. Babuška and J. Osborn. Eigenvalue problems. In *Finite Element Methods (Part 1)*, volume 2 of *Handbook of Numerical Analysis*, pages 641–787. Elsevier, 1991.
- [2] Y. Bai, T. Chaolu, and S. Bilige. The application of improved physics-informed neural network (IPINN) method in finance. *Nonlinear Dynamics*, 107(4):3655–3667, 2022.
- [3] Z. Bai, J. Demmel, J. Dongarra, A. Ruhe, and H. van der Vorst. *Templates for the solution of algebraic eigenvalue problems: a practical guide*. SIAM, 2000.
- [4] A. G. Baydin, B. A. Pearlmutter, A. A. Radul, and J. M. Siskind. Automatic differentiation in machine learning: a survey. *Journal of Machine Learning Research*, 18(153):1–43, 2018.
- [5] S. Cai, Z. Wang, S. Wang, P. Perdikaris, and G. E. Karniadakis. Physics-informed neural networks for heat transfer problems. *Journal of Heat Transfer*, 143(6):060801, 2021.

- [6] Y. Chen, L. Lu, G. E. Karniadakis, and L. Dal Negro. Physics-informed neural networks for inverse problems in nano-optics and metamaterials. *Optics Express*, 28(8):11618–11633, 2020.
- [7] F. R. Chung. Eigenvalues of graphs. In *Proceedings of the International Congress of Mathematicians: August 3–11, 1994 Zürich, Switzerland*, pages 1333–1342. Springer, 1995.
- [8] S. De Leo and G. Sclarici. Right eigenvalue equation in quaternionic quantum mechanics. *Journal of Physics A: Mathematical and General*, 33(15):2971, 2000.
- [9] M. Dissanayake and N. Phan-Thien. Neural-network-based approximations for solving partial differential equations. *Communications in Numerical Methods in Engineering*, 10(3):195–201, 1994.
- [10] Y. Dong, J. Tao, Y. Zhang, W. Lin, and J. Ai. Deep learning in aircraft design, dynamics, and control: Review and prospects. *IEEE Transactions on Aerospace and Electronic Systems*, 57(4):2346–2368, 2021.
- [11] D. Finol, Y. Lu, V. Mahadevan, and A. Srivastava. Deep convolutional neural networks for eigenvalue problems in mechanics. *International Journal for Numerical Methods in Engineering*, 118(5):258–275, 2019.
- [12] O. Fuks and H. A. Tchelepi. Limitations of physics informed machine learning for nonlinear two-phase transport in porous media. *Journal of Machine Learning for Modeling and Computing*, 1(1), 2020.

- [13] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pages 249–256. JMLR Workshop and Conference Proceedings, 2010.
- [14] J. Han, A. Jentzen, and W. E. Solving high-dimensional partial differential equations using deep learning. *Proceedings of the National Academy of Sciences*, 115(34):8505–8510, 2018.
- [15] J. Han, J. Lu, and M. Zhou. Solving high-dimensional eigenvalue problems using deep neural networks: A diffusion Monte Carlo like approach. *Journal of Computational Physics*, 423:109792, 2020.
- [16] J. He, L. Li, J. Xu, and C. Zheng. Relu deep neural networks and linear finite elements. *arXiv preprint arXiv:1807.03973*, 2018.
- [17] K. Hoffman and R. Kunze. *Linear Algebra (2nd Edition)*. Prentice-Hall, 1971.
- [18] K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989.
- [19] A. Jentzen, B. Kuckuck, and P. von Wurstemberger. Mathematical introduction to deep learning: Methods, implementations, and theory. *arXiv preprint arXiv:2310.20360*, 2023.
- [20] H. Jin, M. Mattheakis, and P. Protopapas. Physics-informed neural networks for quantum eigenvalue problems. In *2022 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2022.

- [21] N. Jones. How machine learning could help to improve climate forecasts. *Nature*, 548(7668), 2017.
- [22] L. P. Kaelbling, M. L. Littman, and A. W. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence research*, 4:237–285, 1996.
- [23] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet Classification with Deep Convolutional Neural Networks. *Advances in Neural Information Processing Systems*, 25, 2012.
- [24] I. E. Lagaris, A. Likas, and D. I. Fotiadis. Artificial neural networks for solving ordinary and partial differential equations. *IEEE Transactions on Neural Networks*, 9(5):987–1000, 1998.
- [25] Y. LeCun, Y. Bengio, et al. Convolutional networks for images, speech, and time series. *The Handbook of Brain Theory and Neural Networks*, 3361(10):1995, 1995.
- [26] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.
- [27] H. Lee and I. S. Kang. Neural algorithm for solving differential equations. *Journal of Computational Physics*, 91(1):110–131, 1990.
- [28] M. Mattheakis, P. Protopapas, D. Sondak, M. Di Giovanni, and E. Kaxiras. Physical Symmetries Embedded in NeuralNetworks. *arXiv preprint arXiv:1904.08991*, 2019.

- [29] R. R. Nadakuditi and J. W. Silverstein. Fundamental limit of sample generalized eigenvalue based detection of signals in noise using relatively few signal-bearing and noise-only samples. *IEEE Journal of Selected Topics in Signal Processing*, 4(3):468–480, 2010.
- [30] S. J. Pan and Q. Yang. A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, 22(10):1345–1359, 2009.
- [31] G. Pang, L. Lu, and G. E. Karniadakis. fPINNs: Fractional physics-informed neural networks. *SIAM Journal on Scientific Computing*, 41(4):A2603–A2626, 2019.
- [32] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in Neural Information Processing Systems*, 32, 2019.
- [33] 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*, 378:686–707, 2019.
- [34] M. Razeghi. *Fundamentals of solid state engineering*. Springer, 2006.
- [35] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986.

- [36] J. Sirignano and K. Spiliopoulos. Dgm: A deep learning algorithm for solving partial differential equations. *Journal of Computational Physics*, 375:1339–1364, 2018.
- [37] V. Sitzmann, J. Martel, A. Bergman, D. Lindell, and G. Wetzstein. Implicit neural representations with periodic activation functions. *Advances in Neural Information Processing Systems*, 33:7462–7473, 2020.
- [38] B. P. van Milligen, V. Tribaldos, and J. Jiménez. Neural network differential equation and plasma equilibrium solver. *Physical Review Letters*, 75(20):3594, 1995.
- [39] H. Wang, L. Lu, S. Song, and G. Huang. Learning specialized activation functions for physics-informed neural networks. *arXiv preprint arXiv:2308.04073*, 2023.
- [40] S. Wang, X. Yu, and P. Perdikaris. When and why pinns fail to train: A neural tangent kernel perspective. *Journal of Computational Physics*, 449:110768, 2022.
- [41] K. Weiss, T. M. Khoshgoftaar, and D. Wang. A survey of transfer learning. *Journal of Big data*, 3:1–40, 2016.
- [42] Q. Yang, Y. Deng, Y. Yang, Q. He, and S. Zhang. Neural networks based on power method and inverse power method for solving linear eigenvalue problems. *Computers & Mathematics with Applications*, 147:14–24, 2023.
- [43] B. Yu et al. The deep ritz method: a deep learning-based numerical algorithm for solving variational problems. *Communications in Mathematics and Statistics*, 6(1):1–12, 2018.

- [44] F.-G. Yuan, S. A. Zargar, Q. Chen, and S. Wang. Machine learning for structural health monitoring: challenges and opportunities. *Sensors and Smart Structures Technologies for Civil, Mechanical, and Aerospace Systems 2020*, 11379:1137903, 2020.

VITAE

- Name: Idris Adewale Oduola
- Nationality: Nigerian
- Email: idrisoduola47@gmail.com
- Permenant Address: No. 17 Adedapo Street, Era-Ijanikin, Lagos, 102101, Nigeria.

I obtained my BSc in Mathematics from the University of Ilorin, Nigeria, with a first class honours. Then I moved to KFUPM as an international scholarship student, for my MSc in Mathematics.

The result of this thesis was presented at the first KFUPM interdisciplinary graduate students conference, where it was awarded the 3rd presentation award, winning a cash price of 2000 SAR. At the time of writing this vitae, the thesis results has not been published in any journal, but the manuscript for a paper is being prepared.