5-14-2023

# Numerical Methods for Integral Equations

Yuzhen Liu
*Syracuse University*

# Abstract

We first propose a multiscale Galerkin method for solving the Volterra integral equations of the second kind with a weakly singular kernel. Due to the special structure of Volterra integral equations and the "shrinking support" property of multiscale basis functions, a large number of entries of the coefficient matrix appearing in the resulting discrete linear system are zeros. This result, combined with a truncation scheme of the coefficient matrix, leads to a fast numerical solution of the integral equation. A quadrature method is designed especially for the weakly singular kernel involved inside the integral operator to compute the nonzero entries of the compressed matrix so that the quadrature errors will not ruin the overall convergence order of the approximate solution of the integral equation. We estimate the computational cost of this numerical method and its approximate accuracy. Numerical experiments are presented to demonstrate the performance of the proposed method.

We also exploit two methods based on neural network models and the collocation method in solving the linear Fredholm integral equations of the second kind. For the first neural network (NN) model, we cast the problem of solving an integral equation as a data fitting problem on a finite set, which gives rise to an optimization problem. In the second method, which is referred to as the NN-Collocation model, we first choose the polynomial space as the projection space of the Collocation method, then approximate the solution of the integral equation by a linear combination of polynomials in that space. The coefficients of the linear combination are served as the weights between the hidden layer and the output layer of the neural network. We train both neural network models using gradient descent with Adam optimizer. Finally, we compare the performances of the two methods and find that the NN-Collocation model offers a more stable, accurate, and efficient solution.

# Numerical Methods for Integral Equations

By

**Yuzhen Liu**

B.S., Henan University of Technology, 2010

M.S., Sun Yat-sen University, 2012

Ph.D., Sun Yat-sen University, 2016

M.S., Syracuse University, 2021

Dissertation

Submitted in partial fulfillment of the requirements for the degree of

Doctor of Philosophy in Mathematics

Syracuse University

May 2023

# Acknowledgments

First and foremost, I would like to express my deepest gratitude to my advisor, Lixin Shen. During my three years of study at Syracuse University. Professor Shen gave me a great deal of support and encouragement for my research, coursework, and job searching. Without his help, I could not have successfully passed every academic milestone of my Ph.D. program. I also very much appreciate that Prof. Shen sacrificed a lot of his personal time to revise my Ph.D. thesis and gave me so much useful feedback.

I am grateful to the faculty members and staff at Syracuse University, especially Professors Uday Banerjee, Pinyuen Chen, Leonid Kovalev, Graham Leuschke, Moira McDermott, Minghao Rostami, Qinru Qiu, and William Wiley for their inspiring courses and various kinds of help. I also sincerely appreciate my committee members Professors Uday Banerjee, Pinyuen Chen, and Qinru Qiu.

I am thankful to all my colleagues, classmates, and friends for making my life in Syracuse enjoyable. Especially thanks to Jianchen Wei, Jianqing Jia, Bhargavi Parthasarathy, Jesse Hulse, and Chishu Yin.

Finally, my frankest and greatest appreciation belongs to my mom. She is my hero. She sacrificed her own happiness to raise me and my brother after my father passed away when I was only one year old. I would be nowhere without her love and support. No matter what kind of difficulties encountered, her love keeps encouraging me to be fearless and lightening me in the long dark night. Glory to my mom.

# Contents

# Chapter 1

# Introduction

## 1.1 Problem Statement

This dissertation studies numerical methods for solving integral equations. An integral equation is an equation containing an unknown function under the integral sign. In general, an integral equation has the form of

$$cu(x) = f(x) + \mu \int_a^b k(x,s)u(s)\,\mathrm{d}s, \tag{1.1}$$

where the function $k(\cdot,\cdot)$ is called the kernel of the integral equation, $\mu$ is a non-zero constant, $c \in \{0,1\}$. Both $k(\cdot,\cdot)$ and $f$ are given functions while $u$ is unknown and needs to be determined. According to the value of $c$ in the above equation, the associated integral equations have two main types.

- $c = 0$. Equation (1.1) is called the integral equation of the first kind;
- $c = 1$. Equation (1.1) is called the integral equation of the second kind.

Fredholm integral equations and Volterra integral equations are two well known integral equations. Specifically, if both the integral limits $a$ and $b$ in (1.1) are constants, the integral equation is called a Fredholm integral equation. When one of the integral limits is a con-

stant, the other is the variable $x$, the resulting integral equation is called a Volterra integral equation.

Integral equations have wide applications in engineering and mechanics. Fredholm integral equations appear as mathematical models for some problems in signal and image processing while Volterra integral equations usually arise in various fields of physics and engineering, e.g. potential theory and Dirichlet problems, and diffusion problems. These integral equations usually can't be solved analytically, therefore, seeking efficient numerical methods to solve them is practically important. In this thesis, we will propose a multiscale Galerkin method for solving Volterra integral equations of the second kind and neural network-based methods for solving Fredholm integral equations.

## 1.2 Motivations

Numerical methods for approximating the solutions of integral equations have been studied extensively in the literature. Traditional numerical methods including quadrature methods, collocation methods, and Galerkin methods are well known and developed prior to the mid-1980s. Unfortunately, all of these approaches result in linear systems with dense discretization matrices, and when the order of the coefficient matrix is large, the computational cost for generating the matrix and then solving the corresponding linear system is huge. This is identified as a bottleneck in traditional numerical methods.

The application of multiscale methods in computational mathematics in recent decades has enlightened a new direction for solving integral equations. Compared to traditional numerical methods, multiscale methods make use of the multiscale feature and the vanishing moment property of the multiscale piecewise polynomial basis for a singular integral equation resulting in a numerically sparse coefficient matrix, which, in combination with a truncation strategy, leads to a fast numerical solution of the Fredholm integral equation. Despite its success in solving Fredholm integral equations [1] [2] [3], the application of multiscale

methods to Volterra integral equations has yet to be explored. Given the specific structure of Volterra integral equations, we propose that the multiscale Galerkin method would be a suitable fit to solve Volterra integral equations with singular kernels. To be more specific, this method introduces a new integral, with the integral limits $x$ of the Volterra integral being the variable of the integration of this new integral. This structure, along with the "shrinking support" property of the multiscale basis functions, results in a large number of entries in the resulting matrix being zeros, reducing a significant computational cost.

Additionally, the increasing popularity of deep learning neural networks has sparked interest in applying them to solve mathematical problems [4] [5] [6] [7]. Motivated by the successful application of this technique in other domains and the theoretical foundation established by universal approximation theory [8], we believe that deep learning neural network models have the potential to be a viable approach for solving integral equations. Our initial Neural network (NN) approach is entirely based on learning by casting the original problem into a data-fitting problem. However, this approach falls short in terms of performance compared to the current existing mathematical methods for solving Fredholm integral equations. This deficiency may be due to an inadequate representation space. To address this issue, we refine our approach by restricting the representation space to a polynomial space, as polynomial spaces are dense in the solution space of integral equations.

.

## 1.3 Contributions of This Dissertation

In this thesis, we employ various approaches based on Galerkin, Collocation methods, and the neural network model to solve Volterra and Fredholm integral equations. Main contributions of this dissertation are as follows.

- We applied the multiscale Galerkin method to solve the Volterra integral equation of the second kind with a weakly singular kernel. The multiscale feature and the

vanishing moment property of the multiscale piecewise polynomial basis lead to a linear system with a numerically sparse coefficient matrix. We designed a truncation scheme for that coefficient matrix and developed a fast numerical algorithm based on the compression scheme. We proposed a numerical quadrature rule for estimating the nonzero entries. The error control strategy of the quadrature rule is designed so that the quadrature error will not ruin the overall convergence order of the multiscale Garlekin method. In summary, we reduced the computational cost from $s(n)^2$ to $s(n)\log s(n)$ without ruining the convergence order of the approximation solution, where $s(n)$ is the dimension of the underlying multiscale basis.

- We developed two approaches to solve Fredholm integral equation with neural network models. In the first approach, we observed that the neural network models do have the potential to solve Fredholm integral equation as the universal approximation theorem implies, but the accuracy cannot compete with the existing traditional mathematical methods, due to either an inadequate representational space or limitations of the learning algorithm utilized. To improve the performance of our neural network model, we incorporated the traditional collocation method with the neural network model in our second approach. The numerical results showed that the improved model outperformed other existing methods.

## 1.4 Organization of This Dissertation

The outline of the rest of this dissertation is as follows.

Chapter 2 presents our main contribution on multiscale methods for solving Volterra integral equations. We begin by outlining the general setup of the multiscale Galerkin method for solving the Volterra integral equation of the second kind with a singular kernel. We then review the construction of multiscale basis functions and provide a truncation strategy for the coefficient matrix that results from the proposed multiscale Galerkin method. Finally, we

propose a numerical quadrature strategy for computing the nonzero entries of the compressed coefficient matrix.

Chapter 3 gives a preliminary review of machine learning. In particular, we review basic machine learning concepts that will be used in the following chapters.

Chapter 4 is about solving Fredholm integral equations of the second kind based on a neural network model.

Chapter 5 is to further improve the approach in Chapter 4. We propose a method that is based on the Collocation method by restricting the representational space of the neural network model into a polynomial space.

Chapter 6 is devoted to the special function neural network model [7], which is an additional research work that I did during my internship at Argonne National lab.

Chapter 7 is the conclusion of this thesis.

# Chapter 2

# A Multiscale Galerkin Method for Solving Volterra Integral Equations of the Second Kind with Weakly Singular Kernels

Volterra integral equations are named after Italian mathematician Vito Volterra (1860-1940) who first introduced them in his paper "Sull'integrazione di alcune equazioni funzionali" in 1903. In Volterra's paper, he studied equations of the form:

$$\mathcal{K}u = f \tag{2.1}$$

with the integral operator $\mathcal{K}$ defined by

$$(\mathcal{K}u)(x) = \int_0^x k(x,s)u(s)\mathrm{d}s, \quad x \in [0,T] \ \text{ with } \ T < \infty.$$

However, this equation was first named as Volterra integral equation of the first kind by Lalesco in 1908. Volterra's work on integral equations laid the foundations for many of

the modern methods and techniques used in the field today. In the decades that followed, Volterra integral equations became an important area of research in mathematics and physics. A detailed description of Volterra's contributions in integral equations can be found in [9].

Mathematically, a Volterra integral equation is an equation which involves an unknown function and an integral of that function. Furthermore, if the unknown function appears only under the integral sign of the Volterra equation, the integral equation is called a first kind Volterra integral equation. The general form of a first kind Volterra integral equation can be written as:

$$\int_0^x k(x,s)u(x)\mathrm{d}s = f(x), \quad x \in [0,T].$$

In contrast, if the unknown function $u$ appears both inside and outside the integral sign of the Volterra equation, the corresponding equation is called a second kind Volterra integral equation:

$$u(x) + \int_0^x k(x,s)u(x)\mathrm{d}s = f(x), x \in [0,T],$$

where function $k$ is called the kernel function, $k$ and $f$ are given functions, while $u$ is the unknown function and needs to be determined.

A Volterra integral equation of the first kind is equivalent to a Volterra integral equation of the second kind under some conditions on their kernels. To see it, taking the first-order derivative with respect to $x$ on both sides of the first kind Volterra integral equation leads to

$$k(x,x)u(x) + \int_0^x \frac{\partial k(x,s)}{\partial x} u(s)\ ds = f'(x).$$

Assume that $k(x,x) \neq 0$. We obtain from the above equation that

$$u(x) + \int_0^x \frac{1}{k(x,x)} \frac{\partial k(x,s)}{\partial x} u(s)\ ds = \frac{f'(x)}{k(x,x)},$$

which is a second kind Volterra integral equation. Volterra integral equations of the first kind are inherently ill-posed problems, meaning that the solution is generally unstable, and

small changes to the problem can cause very large changes to the answers obtained. To overcome the ill-posedness, different regularization methods, like Tikhonov regularization and Lavrentiv regularization, can be applied to convert a first kind Volterra integral equation to a second kind one. For these reasons, the discussion in this thesis will be mainly given for the Volterra integral equations of the second kind.

Volterra integral equations arise from many applications such as demography, population dynamics, elasticity, plasticity, semi-conductors, scattering theory, seismology, fluid flow dynamics, chemical reactions, and oscillation theory [10]. Volterra integral equations can also be derived from initial value problems [11]. These equations are usually difficult to solve analytically, numerical methods are often needed. There are a variety of methods such as the successive approximations method, Laplace transform method, spline collocation method, Runge-Kutta method, and more recently developed methods including the Adomian decomposition method, the modified decomposition method, and the variational iteration method to handle Volterra integral equations.

A class of Volterra integral equations with weakly singular kernels plays an important role in many applications including microscopy, seismology, radio astronomy, electron emission, atomic scattering, and radar ranging, see [12] and the references therein. The kernel in a Volterra integral operator is singular if the kernel becomes infinite at one or more points within the range of integration. This kernel is weakly singular if it is singular and the integral of the absolute value of the kernel function is finite. Two such typical kernels have forms

$$k(x,s) = g(x,s)|x-s|^{\theta-1}$$

and

$$k(x,s) = g(x,s)\log|x-s|,$$

where $g$ is a smooth function, $0 \le s \le x \le 1$, and $0 < \theta < 1$. In the above two kernels, the singularity occurs when $s$ approaches $x$. The existence of the solution to the Volterra

integral equation with a weakly singular kernel was discussed in [13].

Galerkin methods are typically exploited for the discretization of Volterra integral operators, which belong to the type of projection methods. A common characteristic of projection methods is their ability to address equations of the form $\mathcal{A}u = f$, where $\mathcal{A} : \mathcal{X} \to \mathcal{X}$ is a bounded linear operator, $\mathcal{X}$ is a Hilbert space. In these method, a sequence $\{\mathcal{X}_n : n \in N\}$ of subspaces of $\mathcal{X}$ is specified and an element $u_n$ in $\mathcal{X}_n$ is selected in such a way that the residual error

$$r_n := \mathcal{A}u_n - f \tag{2.2}$$

is deemed "small" in some sense. Different strategies for making $r_n$ "small" produce different projection methods. In particular, for Galerkin methods, the residual is chosen as

$$r_n = \mathcal{A}u_n - f \in \mathcal{X}_n^{\perp},$$

this is equivalent to

$$\mathcal{P}_n\mathcal{A}u_n = \mathcal{P}_n f \tag{2.3}$$

where $\mathcal{P}_n : \mathcal{X} \to \mathcal{X}_n$ is the orthogonal projection operator. Without carefully choosing bases of $\mathcal{X}_n$, (2.3) will result in a linear system of equations with a dense coefficient matrix generally. A dense matrix potentially leads to an expensive computational algorithm for solving the resulting linear system. Therefore, it will be desirable to have a sparse matrix by carefully choosing an orthogonal basis.

In this chapter, we will develop a multiscale Garlekin method for solving Volterra integral equations with weakly singular kernels. To this end, we will choose the basis of $\mathcal{X}_n$ from a set of multiscale piecewise polynomials.

# Symbols

| | |
|---|---|
| $N$ | Set of positive integers |
| $R$ | All the real number |
| $Z_n$ | Set of integers $\{0, 1, 2, \ldots, n-1\}$ for $n \in N$ |
| $Z_\mu^j$ | $Z_\mu \times \cdots \times Z_\mu$ ($j$ times) |
| $I$ | $[0, 1]$ |
| $\phi_\epsilon(I)$ | $\{\phi_\epsilon(x) : x \in I\}$ |
| $I_{\epsilon,\mu}$ | $[\frac{\epsilon}{\mu}, \frac{\epsilon+1}{\mu}]$ |
| $L^2(I)$ | Linear space of all real-valued square-integrable functions |
| $L^\infty(I)$ | Linear space of all real-valued essentially bounded measurable functions |
| $(u, v)$ | $\int_0^1 u(x)v(x)\mathrm{d}x$ |
| $\|u\|_p$ | Norm of $L^p(I)$ |
| $P_\sigma$ | The space of all polynomials of degree less than or equal to $\sigma - 1$ on $I$ |
| $W^{\sigma,p}$ | Sobolev Space |
| $\mathcal{H}^\sigma$ | $W^{\sigma,2}$ |
| $\mathcal{B}(\mathcal{X})$ | Banach Space on $\mathcal{X}$ |
| $\|\mathcal{K}\|$ | Norm of operator $\mathcal{K}$ in $\mathcal{B}(L^2(I))$ |
| $\mathcal{X}_i$ | Multiscale polynomial space with multiscale level $i$ |

| | |
|---|---|
| $\mathcal{W}_i$ | Orthogonal complement of $\mathcal{X}_i$ in $\mathcal{X}_{i+1}$ |
| $s(i)$ | $\dim\mathcal{X}_i$ |
| $w(i)$ | $\dim\mathcal{W}_i$ |
| $m$ | $\dim\mathcal{W}_1$ |
| $card(A)$ | Cardinality of $A$ |
| $U_n$ | $\{(i,j) : i \in Z_n, j \in Z_{w(i)}\}$ |
| $\nu(e_i)$ | $\mu^{i-2}\epsilon_0 + \cdots + \mu\epsilon_{i-3} + \epsilon_{i-2}$, for some $e_i = (\epsilon_0, \epsilon_1, \cdots, \epsilon_{i-2}) \in Z_\mu^{i-1}$, where $i \in N$ and $i > 1$. |
| $u_{ij}$ | $j$th basis of $\mathcal{W}_i$ |
| $S_{ij}$ | Support set of basis function $u_{ij}$ |
| $d_i$ | $\text{meas}(S_{ij})$ |
| $\cup^\perp$ | Union of orthogonal sets |
| $\oplus^\perp$ | Orthogonal direct sum of spaces |
| $\chi_\Omega$ | Characteristic function of the set $\Omega$ |
| $dist(A,B)$ | $\min\{\|x-s\| : x \in A, s \in B\}$ |
| $I(f)$ | $\int_0^1 f(x)\mathrm{d}x$ |

## 2.1 Volterra Integral Equations with Weakly Singular Kernels

We begin with the definition of a Volterra integral operator with a weakly singular kernel.

**Definition 1** (Volterra integral operator with weakly singular kernels)**.** *Let* $S := \{(x,s) : 0 \le s \le x \le 1\} \subset I \times I$. *For a given continuous function* $g$ *on* $I \times I$ *and a parameter* $0 < \theta < 1$, *we define a function* $k : S \to R$ *as follows:*

$$k(x,s) = g(x,s)|x-s|^{\theta-1}. \tag{2.4}$$

*Then, the operator $\mathcal{K}$ given by*

$$(\mathcal{K}u)(x) = \int_0^x k(x,s)u(s)\,\mathrm{d}s, \quad x \in I \tag{2.5}$$

*is called the Volterra integral operator with the weakly singular kernel $k$.*

As shown in the next result, the Volterra integral operator $\mathcal{K}$, given in Definition (1), is well-defined on $L^2(I)$ and a bounded linear operator.

**Proposition 1.** *The integral operator $\mathcal{K}$ defined by (2.5) with a weakly singular kernel $k$ in (2.4) is a bounded linear operator from $L^2(I)$ to $L^2(I)$ with the operator norm $\|\mathcal{K}\| \leq M_g(I)/\theta$, where $M_g(I) = \sup\{|g(x,s)| : x, s \in I\}$.*

*Proof.* First, for any $u \in L^2(I)$ we know by the Fubini theorem that

$$\int_0^1 \int_0^x u^2(s)|x-s|^{\theta-1}\,\mathrm{d}s\,\mathrm{d}x = \int_0^1 u^2(s)\left(\int_s^1 |x-s|^{\theta-1}\,\mathrm{d}x\right)\mathrm{d}s \leq \frac{1}{\theta}\|u\|_2^2. \tag{2.6}$$

Therefore, the function $v$ defined for all $x \in I$ as

$$v(x) := \int_0^x u^2(s)|x-s|^{\theta-1}\,\mathrm{d}s$$

exists at almost every $x \in I$ and is integrable.

Next, for every pair $(x,s) \in S := \{(x,s) : 0 \leq s \leq x \leq 1\}$ we observe that

$$\begin{aligned} |k(x,s)u(s)| &\leq M_g(I)|u(s)| \cdot |x-s|^{\theta-1} \\ &\leq \frac{M_g(I)}{2}|x-s|^{\theta-1} + \frac{M_g(I)}{2}u^2(s)|x-s|^{\theta-1}. \end{aligned}$$

Since both terms on the right-hand side of the above inequality are integrable with respect to $0 \leq s \leq x$, we conclude that $|k(x,s)u(s)|$ is finite for almost every $s \in [0,x]$.

Finally, by the Cauchy-Schwarz inequality, we have that

$$\begin{aligned}[(\mathcal{K}u)(x)]^2 &= \left(\int_0^x k(x,s)u(s)\,\mathrm{d}s\right)^2 \\ &\leq \left(\int_0^x g^2(x,s)|x-s|^{\theta-1}\,\mathrm{d}s\right)\left(\int_0^x u^2(s)|x-s|^{\theta-1}\,\mathrm{d}s\right) \\ &\leq \frac{M_g^2(I)}{\theta}v(x).\end{aligned}$$

Hence, $\mathcal{K}$ is defined on $L^2(I)$ and maps $L^2(I)$ to $L^2(I)$ with

$$\|\mathcal{K}u\|_2^2 \leq \frac{M_g^2(I)}{\theta^2}\|u\|_2^2,$$

for $u \in L^2(I)$. This completes the proof. □

The Volterra integral equation of the second kind with a weakly singular kernel is

$$f(x) = u(x) + \mathcal{K}u(x), \quad x \in I, \tag{2.7}$$

where $f \in L^2(I)$ is known and $\mathcal{K}$ the Volterra integral operator is given in Definition 1. The existence of a solution to (2.7) and the properties of the corresponding solution were discussed in [13–16] and the references therein. Particularly, for $f$ being a continuous function on $I$, we recall an existence theorem from [13].

**Theorem 2.1.1.** *Consider the integral equation*

$$f(x) = u(x) + \int_0^x (x-s)^{\theta-1}M(x,u(s))\,\mathrm{d}s, \quad x \in [0,1],\ \theta \in (0,1), \tag{2.8}$$

*where $M : S \times R \to R$ with $S = \{(x,s) : 0 \leq s \leq x \leq 1\}$. Assume that $f$ is continuous in $[0,1]$ and real analytic in $(0,1)$, and let the kernel $M(x,y)$ be real analytic in $S \times R$. Then the solution $u$ of (2.8) is real analytic in the open interval $(0,1)$.*

This theorem clearly indicates that the solution of the Volterra integral equation of the

second kind (2.7) is analytic if $f$ is continuous and $M(x, u(s)) = g(x, s)u(s)$ with $g$ being analytic in $S$.

In the following discussion, we will concentrate on numerical solutions to the Volterra integral equations of the second kind. Projection methods and quadrature methods are two popular approaches for the numerical treatment of the Volterra integral equation of the second kind. Projection methods, such as the Collocation and Galerkin methods, aim to find an approximating operator of the Volterra integral operator; quadrature methods including the trapezoidal rule, Simpson's rule, and Gaussian quadrature approximate the integral in the integral equation by replacing the integral with sum. There are many other approaches for solving Volterra integral equations of the second kind with a weakly singular kernel. For example, a method based on the double exponential transformation was proposed in [17] and an interpolation method based on the barycentric Lagrange interpolation was discussed in [18]. However, these approaches all result in linear systems with numerically dense coefficient matrices.

In this chapter, we will present a multiscale Galerkin method for solving (2.7) using multiscale piecewise polynomials, incorporating a compression strategy, and a carefully designed numerical integration scheme. Multiscale methods have played a crucial role in approximating the solution of integral equations. Due to its vanishing moment and "shrinking support" properties of the multiscale piecewise polynomial basis, the multiscale method results in a linear system with a numerically sparse coefficient matrix which allows us to develop fast numerical algorithms. Besides that, considering the special structure of Volterra integral equations and the Galerkin method, a large number of entries of the resulting coefficient matrix would be zeros. Thus, the multiscale Galerkin method would be an efficient candidate for solving Volterra integral equation. Since the continuity of function $g$ in (2.4) doesn't affect the way we design our numerical integration strategy, for simplicity, we particularly

consider the Volterra integral equation with the weakly singular kernels:

$$k(x,s) = a(x-s)^{\theta-1}, \quad 0 < \theta < 1, \tag{2.9}$$

where $a$ is a nonzero constant.

We will give a brief review of the multiscale Galerkin method in the next section.

## 2.2 Multiscale Galerkin Methods

The main purpose of this section is to present multiscale Galerkin methods for solving the Volterra integral equation (2.7). Let's first give a brief review of the classical Galerkin method.

Let $\mathcal{Y} := L^2(I)$ and let $N$ denote the set of all natural numbers. Suppose that $\mathcal{Y}_i$, $i \in N$, is a sequence of finite-dimensional subspaces of $\mathcal{Y}$ such that

$$\mathcal{Y} = \overline{\cup_{i\in N}\mathcal{Y}_i}.$$

Let $\mathcal{P}_i$ be the orthogonal projection from $\mathcal{Y}$ onto $\mathcal{Y}_i$, i.e., $\mathcal{P}_i v = v$ for all $v \in \mathcal{Y}_i$. Then, the Galerkin method for solving (2.7) is to find $u_i \in \mathcal{Y}_i$ such that

$$(\mathcal{I} + \mathcal{P}_i\mathcal{K})u_i = \mathcal{P}_i f. \tag{2.10}$$

When a basis of the subspace $\mathcal{Y}_i$ is chosen, the operator equation (2.10) is equivalent to a system of linear equations. More precisely, let the set $\{v_j : j = 1, 2 \ldots, D_i\}$ be the orthogonal basis of $\mathcal{Y}_i$, where $D_i := \dim\mathcal{Y}_i$, then solving the operator equation (2.10) is equivalent to seek an approximate solution

$$u_i = \sum_{j=1}^{D_i} b_j v_j \in \mathcal{X}_n,$$

such that

$$(u_i + \mathcal{P}_i\mathcal{K}u_i, v_j) = (f, v_j), \quad \text{for} \quad j = 1, 2, \ldots, D_i. \tag{2.11}$$

hold. (2.11) can be further rewritten as the following system of equations:

$$\begin{pmatrix} (v_1 + \mathcal{P}_i\mathcal{K}v_1, v_1) & (v_2 + \mathcal{P}_i\mathcal{K}v_2, v_1) & \cdots & (v_{D_i} + \mathcal{P}_i\mathcal{K}v_{D_i}, v_1) \\ (v_1 + \mathcal{P}_i\mathcal{K}v_1, v_2) & (v_2 + \mathcal{P}_i\mathcal{K}v_1, v_{D_i}) & \cdots & (v_{D_i} + \mathcal{P}_i\mathcal{K}_{D_i}, v_2) \\ \cdots & \cdots & \cdots & \cdots \\ (v_1 + \mathcal{P}_i\mathcal{K}v_1, v_{D_i}) & (v_2 + \mathcal{P}_i\mathcal{K}v_2, v_{D_i}) & \cdots & (v_{D_i} + \mathcal{P}_i\mathcal{K}v_2, v_{D_i}) \end{pmatrix} \begin{pmatrix} b_1 \\ b_2 \\ \cdots \\ b_{D_i} \end{pmatrix} = \begin{pmatrix} (f, v_1) \\ (f, v_2) \\ \cdots \\ (f, v_{D_i}) \end{pmatrix}.$$

Normally, the above system has a dense coefficient matrix. To have a numerically sparse coefficient matrix, we should pay special attention to the choice of the basis for each $\mathcal{Y}_i$. In this thesis, a multiscale piecewise polynomial basis will be adopted and the corresponding method for solving the operator equation (2.10) is called the multiscale Galerkin method.

### 2.2.1 Multiscale Basis Functions

We propose to use the Multiscale-Galerkin method to solve the integral equation (2.7) with the weakly singular kernel (2.9). To this end, we first review the way to construct the multiscale basis functions [1] [19] that generate a sequence of finite-dimensional subspaces of $L^2$ space.

There are two main ingredients in the construction of the multiscale basis. The first one is a set of contractive mappings on $I$ while the other one is a subspace $\mathcal{X}_0$ of $L^2(I)$. For a fixed positive integer $\mu > 1$, we define a set $\{\phi_\epsilon : \epsilon \in Z_\mu\}$ of contractive mappings on $I$ by

$$\phi_\epsilon(t) := \frac{\epsilon + t}{\mu}, \quad t \in I, \quad \epsilon \in Z_\mu.$$

Let $I_{\epsilon,\mu} := [\frac{\epsilon}{\mu}, \frac{\epsilon+1}{\mu}]$, then clearly, $\phi_\epsilon(I) = I_{\epsilon,\mu}$, and $\{I_{\epsilon,\mu} : \epsilon \in Z_\mu\}$ forms a partition of $I$. i.e.,

$$I = \cup_{\epsilon \in Z_\mu} I_{\epsilon,\mu} \quad \text{and} \quad \text{meas}(I_{\epsilon,\mu} \cap I_{\epsilon',\mu}) = 0 \quad \textbf{if} \quad \epsilon \neq \epsilon'.$$

Associated with these mappings, we introduce a set of orthogonal isometries $\{\mathcal{T}_\epsilon : \epsilon \in Z_\mu\}$, where each $\mathcal{T}_\epsilon : L^2(I) \cap L^\infty(I) \to L^2(I)$ is defined by

$$(\mathcal{T}_\epsilon f)(t) := \sqrt{\mu}(f \circ \phi_\epsilon^{-1})(t)\chi_{I_{\epsilon,\mu}}(t) = \begin{cases} \sqrt{\mu} f(\mu t - \epsilon), & t \in I_{\epsilon,\mu}, \\ 0, & t \notin I_{\epsilon,\mu}, \end{cases} \tag{2.12}$$

for $f \in L^2(I) \cap L^\infty(I)$. The purpose of the isometry $T_\epsilon$ is to transform the given function in the horizontal direction. To pin down this problem, let's take a point of view of the simplest case where $\mu = 2$.

**Example 2.2.1** Let $\mu = 2$ and suppose

$$f(x) := \begin{cases} -(x - \frac{3}{10})^2 + \frac{7}{10}, & \text{if } x \in [0, 1], \\ 0, & \text{otherwise}, \end{cases} \tag{2.13}$$

then if we apply operators $\mathcal{T}_\epsilon$, for $\epsilon \in Z_2$ on $f$, we will have

$$\mathcal{T}_0 f(x) := \begin{cases} -(2x - \frac{3}{10})^2 + \frac{7}{10}, & \text{if } x \in [0, \frac{1}{2}], \\ 0, & \text{otherwise}. \end{cases} \tag{2.14}$$

and

$$\mathcal{T}_1 f(x) := \begin{cases} 0, & \text{otherwise}, \\ -(2x + \frac{13}{10})^2 + \frac{7}{10}, & \text{if } x \in [\frac{1}{2}, 1]. \end{cases} \tag{2.15}$$

Fig. 2.2.1 and Fig. 2.2.2 illustrate the results of applications of operators $\mathcal{T}_\epsilon$.

Next, we recall one result from [1], which shows that the functions resulting from applications of the operators $\mathcal{T}_\epsilon$ with different $\epsilon$ are orthogonal.

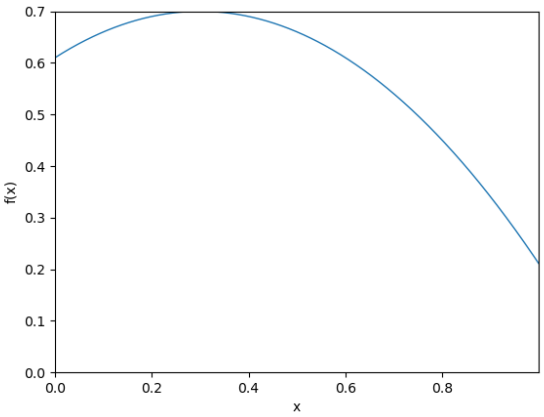
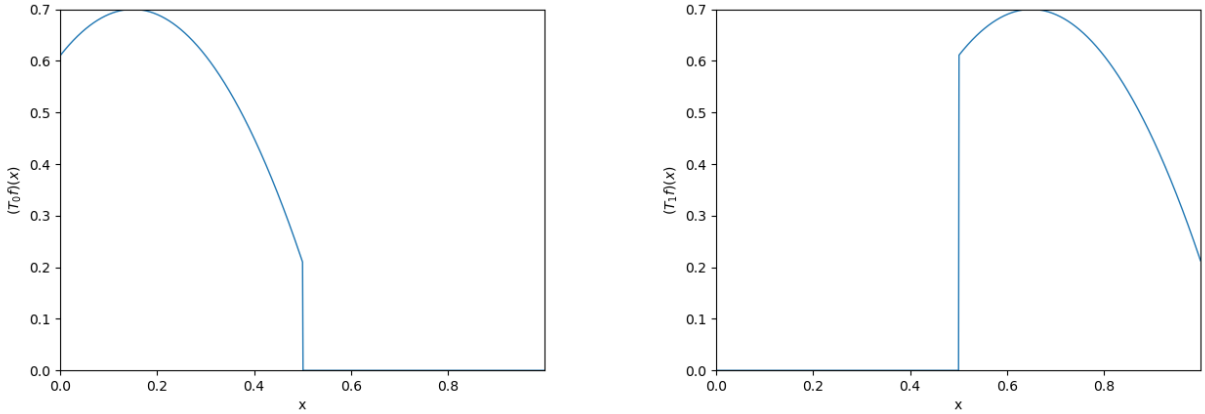



Figure 2.2.1: Graph of $f$




(a) (b)

Figure 2.2.2: (a) Graph of $\mathcal{T}_0 f$; (b) Graph of $\mathcal{T}_1 f$.

**Proposition 2.** *For $f, g \in L^2(I)$ and $\epsilon, \epsilon' \in Z_\mu$, we have*

$$(\mathcal{T}_\epsilon f, \mathcal{T}_{\epsilon'} g) = \Delta_{\epsilon,\epsilon'}(f, g). \tag{2.16}$$

*where $(\cdot, \cdot)$ is the inner product of $L^2(I)$ and*

$$\Delta_{\epsilon,\epsilon'} := \begin{cases} 1, & \text{if } \epsilon = \epsilon', \\ 0, & \text{otherwise.} \end{cases} \tag{2.17}$$

Let $\mathcal{X}_0 := P_\sigma$ be the space of all polynomials of degree less than or equal to $\sigma - 1$ on $I$. With the given subspace $\mathcal{X}_0$ and linear operator $\mathcal{T}_\epsilon$, we generate recursively a sequence of multiscale piecewise polynomial spaces $\{\mathcal{X}_i\}_{i\in N}$ by

$$\mathcal{X}_{i+1} = \oplus_{\epsilon\in Z_\mu}\mathcal{T}_\epsilon\mathcal{X}_i, \quad i \geq 0. \tag{2.18}$$

Obviously, $\mathcal{X}_i$ is the space of piecewise polynomials of degree less than or equal to $\sigma - 1$ with dimension $s(i) := \sigma\mu^i$ and $\mathcal{X}_{i-1} \subseteq \mathcal{X}_i$. Let $X_0$ denote an orthonormal basis for $\mathcal{X}_0$, then Proposition 2 ensures that, for each $i \geq 0$, elements in $\mathcal{T}_\epsilon X_i$ are orthonormal and $\mathcal{T}_\epsilon X_i \perp \mathcal{T}_{\epsilon'} X_i$ for any $\epsilon, \epsilon' \in Z_\mu$ with $\epsilon \neq \epsilon'$. This result, combines with (2.18), implies that

$$X_{i+1} := \cup^{\perp}_{\epsilon\in Z_\mu}\mathcal{T}_\epsilon X_i, \quad \text{for} \quad i \geq 0 \tag{2.19}$$

forms an orthonormal basis for $\mathcal{X}_{i+1}$.

In order to construct the multiscale basis of $\mathcal{X}_n$, we further decompose $\mathcal{X}_n$ as the orthogonal direct sum of its subspaces. To be more precise, let $\mathcal{W}_n$ denote the orthogonal complement of $\mathcal{X}_{n-1}$ in $\mathcal{X}_n$, and with the convention that $\mathcal{W}_0 = \mathcal{X}_0$, we have

$$\mathcal{X}_n = \mathcal{W}_0 \oplus^{\perp} \mathcal{W}_1 \oplus^{\perp} \cdots \oplus^{\perp} \mathcal{W}_n. \tag{2.20}$$

For convenience, we denote $m := \dim\mathcal{W}_1$, then

$$m := \dim\mathcal{X}_1 - \dim\mathcal{X}_0 = (\mu - 1)\sigma,$$

and

$$w(i) = \dim\mathcal{X}_i - \dim\mathcal{X}_{i-1} = \mu^{i-1}m \quad \text{for} \quad i > 1.$$

To establish a multiscale orthonormal basis for $\mathcal{X}_n$, it suffices to construct an orthonormal basis $W_i$ for the subspace $\mathcal{W}_i$. The technique we used is the Gram-Schmidt Orthonormal-

ization. Let $X_0 := \{u_{0l} : l \in Z_\sigma\}$ be an orthogonal basis of $\mathcal{X}_0$ on $I$. Our goal is to construct an orthonormal basis $\{u_{1j} : j \in Z_{w(1)}\}$ of space $\mathcal{W}_1$. To this end, we first construct a basis $X_1 := [\hat{v}_{1j} : j \in Z_{s(1)}]$ of $\mathcal{X}_1$ via (2.19). From the functions in $X_1$, we form a linearly independent set $\{\overline{u}_{1j}\}_{j=0}^{m-1}$ that are orthogonal to all elements of $X_0$. Next, we orthonormalize these $m$ functions. Mathematically, we begin with

$$v_{10} = \overline{u}_{10} \quad \text{and} \quad u_{10} = \frac{v_{10}}{\sqrt{\int_0^1 v_{10}^2(x)\,\mathrm{d}x}}.$$

The next function is formulated as

$$u_{11} = \frac{v_{11}}{\sqrt{\int_0^1 v_{11}^2(x)\,\mathrm{d}x}}$$

with

$$v_{11} = \overline{u}_{11} + a_{10}u_{10}, \quad \text{where} \quad a_{10} = -\int_0^1 \overline{u}_{11}(x)u_{10}(x)\,\mathrm{d}x.$$

Following this procedure, we have

$$u_{1i} = \frac{v_{1i}}{\sqrt{\int_0^1 v_{1i}^2(x)\,\mathrm{d}x}}$$

with

$$v_{1i} = \overline{u}_{1i} + a_{i0}u_{10} + a_{i1}u_{11} + \cdots + a_{i,i-1}u_{1,i-1},$$

where

$$a_{ij} = -\int_0^1 \overline{u}_{1i}(x)u_{1j}(x)\,\mathrm{d}x, \quad i \in Z_{w(1)}, j = 0, 1, \ldots, i-1.$$

With this construction, the set $\{u_{1i}\}_{j=0}^{m-1}$ serves as an orthonormal basis for $\mathcal{W}_1$.

In the following example, we choose $\mu = 2, \sigma = 3$ to demonstrate the process of generating the orthonormal basis for the space $\mathcal{W}_1$, based on the given orthonormal basis of $\mathcal{W}_0$.

**Example 2.2.2** Let $\mu = 2$, $\sigma = 3$, and an orthonormal basis of $\mathcal{W}_0$ are given by

$$u_{00}(t) = 1, \quad u_{01}(t) = \sqrt{3}(2t-1), \quad t \in [0,1].$$

Apply operator $\mathcal{T}_0$ on $u_{00}$ and $u_{01}$, respectively, we obtain

$$\hat{v}_{10}(t) := (\mathcal{T}_0 u_{00})(t) = \begin{cases} 1, & t \in [0, \frac{1}{2}] \\ 0, & t \in (\frac{1}{2}, 1], \end{cases} \tag{2.21}$$

and

$$\hat{v}_{11}(t) := (\mathcal{T}_0 u_{01})(t) = \begin{cases} \sqrt{3}(4t-1), & t \in [0, \frac{1}{2}] \\ 0, & t \in (\frac{1}{2}, 1]. \end{cases} \tag{2.22}$$

Likewise, apply operator $\mathcal{T}_1$ on $u_{00}$ and $u_{01}$ respectively, we obtain

$$\hat{v}_{12}(t) := (\mathcal{T}_1 u_{00})(t) = \begin{cases} 0, & t \in [0, \frac{1}{2}] \\ 1, & t \in (\frac{1}{2}, 1], \end{cases} \tag{2.23}$$

and

$$\hat{v}_{13}(t) := (\mathcal{T}_1 u_{01})(t) = \begin{cases} 0, & t \in [0, \frac{1}{2}] \\ \sqrt{3}(4t-3), & t \in (\frac{1}{2}, 1]. \end{cases} \tag{2.24}$$

Then for any $\hat{v} \in \mathcal{X}_1$, there exist some constants $c_1, c_2, c_3, c_4$ such that

$$\hat{v} = c_1\hat{v}_{10} + c_2\hat{v}_{11} + c_3\hat{v}_{12} + c_4\hat{v}_{13}.$$

To obtain a basis for $\mathcal{W}_1$, we first search for a pair of independent functions in $\mathcal{X}_1$ that are orthogonal to $\mathcal{X}_0$. This is equivalent to solving the system of equations:

$$\begin{cases} \int_0^1 (c_1\hat{v}_{10} + c_2\hat{v}_{11} + c_3\hat{v}_{12} + c_4\hat{v}_{13})(t) u_{00}(t) \mathrm{d}t = 0, \\ \int_0^1 (c_1\hat{v}_{10} + c_2\hat{v}_{11} + c_3\hat{v}_{12} + c_4\hat{v}_{13})(t) u_{01}(t) \mathrm{d}t = 0. \end{cases} \tag{2.25}$$

To simplify the process, we set $c_2 = c_4 = 1$ and $c_2 = 1, c_4 = 0$, respectively, and solve the resulting system of equations. In this way, we find a basis of $\mathcal{W}_1$ as follows:

$$\begin{cases} \overline{u}_{10}(t) := \frac{\sqrt{3}}{3}\hat{v}_{10} + \hat{v}_{11} - \frac{\sqrt{3}}{3}\hat{v}_{12} + \hat{v}_{13}, \\ \overline{u}_{11}(t) := \frac{\sqrt{3}}{6}\hat{v}_{10} + \hat{v}_{11} - \frac{\sqrt{3}}{6}\hat{v}_{12}. \end{cases} \tag{2.26}$$

Finally, we just need to use the Gram-Schmidt process to construct an orthonormal basis $\{u_{10}, u_{11}\}$ of $\mathcal{W}_1$, based on $\overline{u}_{10}, \overline{u}_{11}$.

For the construction of the basis of $\mathcal{W}_i$, $i = 2, 3, \cdots, n$, we refer to the following proposition.

**Proposition 3.** *For $i \in N$,*

$$W_{i+1} = \cup_{\epsilon \in Z_\mu} \mathcal{T}_\epsilon W_i \tag{2.27}$$

*forms an orthonormal basis of $\mathcal{W}_{i+1}$.*

*Proof.* We propose to prove this proposition by induction. Assume that $W_i$ is an orthonormal basis of $\mathcal{W}_i$ for some $i \geq 1$, then we only need to show that $W_{i+1}$ is an orthonormal basis of $\mathcal{W}_{i+1}$. For that end, let $W = \cup_{\epsilon \in Z_\mu} \mathcal{T}_\epsilon W_i$, then by Proposition 2, we know that elements in $W$ are orthonormal. Since $W_i \subseteq \mathcal{W}_i \subset \mathcal{X}_i$, we conclude that

$$W \subset \cup_{i \in Z_\mu} \mathcal{T}_\epsilon \mathcal{X}_i = \mathcal{X}_{i+1}. \tag{2.28}$$

On the other hand, the induction hypothesis that $W_i \perp X_{i-1}$, combines Proposition 2 ensuring that $\mathcal{T}_\epsilon W_i \perp \mathcal{T}_{\epsilon'} X_{i-1}$ for any $\epsilon, \epsilon' \in Z_\mu$, or in other word, $W \perp X_i$. This result and (2.28) imply that $W \subset \mathcal{W}_{i+1}$. The desired result is ensured by the fact that elements in $W$ are orthonormal and Card $W = \dim \mathcal{W}_{i+1}$. □

According to Proposition 3, each basis function $u_{ij}$ may be expressed in terms of consecutive application of operators $\mathcal{T}_\epsilon$ to a basis function of $W_1$. To see that, note that each

$j \in Z_{\mu^{i-1}}$ can be uniquely written as

$$j = \mu^{i-2}\epsilon_0 + \cdots + \mu\epsilon_{i-3} + \epsilon_{i-2}, \quad i = 2, 3, \ldots, n$$

for some $e_i = (\epsilon_0, \epsilon_1, \ldots, \epsilon_{i-1}) \in Z_\mu^{i-1}$. We define

$$\mathcal{T}_{e_i} := \mathcal{T}_{\epsilon_0} \circ \mathcal{T}_{\epsilon_1} \circ \cdots \circ \mathcal{T}_{\epsilon_{i-2}},$$

then, for $j \in Z_{w(i)}$, $j = \nu(e_i)m + \ell$ with $i > 1$ and $\ell \in Z_m$, we have

$$u_{ij} = \mathcal{T}_{e_i} u_{1\ell},$$

with support

$$S_{ij} := I_{\nu(e_i),\mu^{i-1}} = \left[\frac{\nu(e_i)}{\mu^{i-1}}, \frac{\nu(e_i)+1}{\mu^{i-1}}\right], \quad i = 2, 3, \ldots, n.$$

In particular, $S_{ij} = I$ for $i = 0, 1$. Thus, we conclude that $\{u_{ij} : (i,j) \in U_n\}$ forms a multiscale orthonormal basis of $\mathcal{X}_n$. For convenience, in the rest of the paper, we use $[q_{ij}, q'_{ij}]$ to represent $[\frac{\nu(e_i)}{\mu^{i-1}}, \frac{\nu(e_i)+1}{\mu^{i-1}}]$ assuming that $j = \nu(e_i)r + \ell$ for some $e_i = (\epsilon_0, \epsilon_1, \ldots, \epsilon_{i-2}) \in Z_\mu^{i-1}$.

We end this subsection by presenting a simple example that demonstrates the relationship between the second subscript $j$ of the multiscale basis function $u_{ij}$ and the location of the support set $S_{ij}$.

**Example 2.2.3** Let $\mu = 2, \sigma = 3$, then $m = \mu(\sigma - 1) = 2 \times 3 - 2 = 4$.

In particular, for $i = 2$ and $j \in Z_{w(2)} = Z_8$, we can represent $j$ as

$$j = 4(\epsilon_0) + \ell, \quad \text{for} \quad \epsilon_0 \in Z_2, \ell \in Z_4.$$

To be more specific, the wavelet basis at level $i = 2$ was grouped into two sets of four

functions each having the same "support interval", i.e.

$$S_{20} = S_{21} = S_{22} = S_{23} = [0, \frac{1}{2}],$$

and

$$S_{24} = S_{25} = S_{26} = S_{27} = [\frac{1}{2}, 1].$$

Similarly, at level $i = 3$, for $j \in Z_{w(3)} = Z_{16}$, $j$ can be expressed as:

$$j = 4(2\epsilon_1 + \epsilon_0) + \ell, \quad \text{for} \quad \epsilon_0, \epsilon_1 \in Z_2, \ell \in Z_4,$$

which implies 16 wavelet basis was grouped into 4 sets of 4 functions each sharing the same "support interval" with length $\frac{1}{4}$.

### 2.2.2 Formulation of Multiscale Garlekin Methods

With the sequence of multiscale piecewise polynomial spaces $\mathcal{X}_n$ and under the fact that $\cup_{n\in N}\mathcal{X}_n$ is dense in $L^2(I)$, we can define the orthogonal projector

$$\mathcal{P}_n : L^2(I) \to \mathcal{X}_n$$

which for $u \in L^2(I)$ satisfies

$$(u - \mathcal{P}_n u, u_{ij}) = 0, \quad \forall \quad u_{ij} \in X_n. \tag{2.29}$$

In what follows, the notation $a \sim b$ means that there are two positive constants $c$ and $c'$ such that $ca \leq b \leq c'a$. We use $c$ to denote a universal constant that can be distinct at different occurrences.

**Proposition 4.** *The following properties hold [1]:*

1. *"Vanishing moment property": For any $p \in P_\sigma$ the set of polynomials of degree less than or equal to $\sigma - 1$ on $I$, $(u_{ij}, p) = 0$, for $(i,j) \in U_n$.*

2. *For each $j \in Z_{w(i)}$, there exist a $e_i \in Z_\mu^{i-1}$ such that for $j = \nu(e_i)m + \ell$ with $i > 1$ and $\ell \in Z_m$, and $u_{ij}(s) = 0, s \notin S_{ij}$.*

3. *There exists a constant $c$ such that for $(i,j) \in U_n$,*

$$\|u_{ij}\|_2 = 1 \quad \text{and} \quad \|u_{ij}\|_\infty \le c\mu^{i/2}.$$

4. *"Shrinking support property": If denote $d_i := \text{meas}(S_{ij})$, then $s(i) \sim \mu^i, w(i) \sim \mu^i$ and $d_i \sim \mu^{-i}$.*

5. *The operator $\mathcal{P}_i$ are well defined and converge pointwise to the identity operator $\mathcal{I}$ in $L^2(I)$ as $i \to \infty$, that is, for each $g \in L^2(I)$, $\lim_{i\to\infty} \|\mathcal{P}_i g - g\|_2 = 0$ holds.*

6. *There exists a positive constant $c$ such that, for all $u \in H^\sigma(I)$, $\|u - \mathcal{P}_n u\|_2 \le cs(n)^{-\sigma n}\|u\|_{H^\sigma}$.*

The proof of these properties can be found in [1]. The multiscale Galerkin method of the integral equation (2.7) is to find $u_n \in \mathcal{X}_n$ that satisfies the operator equation

$$(\mathcal{I} + \mathcal{K}_n)u_n = \mathcal{P}_n f, \tag{2.30}$$

where $\mathcal{K}_n = \mathcal{P}_n\mathcal{K}$. Using the multiscale bases $\{u_{ij} : (i,j) \in U_n\}$ as basis for spaces $\mathcal{X}_n$ , the above Galerkin method is to seek a vector $\mathbf{b}_n := [b_{ij} : (i,j) \in U_n]$ such that the function

$$u_n := \sum_{(i,j)\in U_n} b_{ij}u_{ij} \in \mathcal{X}_n$$

satisfied

$$(u_n, u_{i'j'}) + (\mathcal{K}u_n, u_{i'j'}) = (f, u_{i'j'}), \quad \forall \quad u_{i'j'} \in X_n, \quad (i',j') \quad \in U_n, \tag{2.31}$$

or equivalently, $\mathbf{b}_n$ is the solution of the linear system of equations

$$(\mathbf{E}_n + \mathbf{K}_n)\mathbf{b}_n = \mathbf{f}_n, \tag{2.32}$$

where

$$\mathbf{E}_n := [(u_{ij}, u_{i'j'}) : (i,j), (i',j') \in U_n],$$

$$\mathbf{K}_n := [(\mathcal{K}u_{ij}, u_{i'j'}), (i,j), (i',j') \in U_n],$$

and

$$\mathbf{f}_n := [(f, u_{i'j'}), (i',j') \in U_n].$$

Due to the orthogonality property of the multiscale bases, we conclude that $\mathbf{E}_n$ is a block diagonal matrix. Noting that the coefficient matrix $\mathbf{K}_n$ is a dense matrix, when its size $s(n)$ is large, it is expensive to generate it. By properties 1 and 4 of Proposition 4, the use of the multiscale basis allows us to compress the matrix $\mathbf{K}_n$ to a sparse matrix due to the absolute values of a significant amount of its entries being relatively small. As a consequence, a fast numerical algorithm can be developed.

## 2.3 Matrix Compression

In this section, we propose a matrix compression strategy for the coefficient matrix of the linear system (2.32). The results in this subsection are new and form a part of the original work in this thesis.

We partition $\mathbf{K}_n$ as a block matrix

$$\mathbf{K}_n := [\mathbf{K}_{ii'} : i, i' \in Z_{n+1}],$$

where

$$\mathbf{K}_{ii'} := [K_{ij,i'j'} : j \in w(i), j \in w(i')],$$

and $K_{ij,i'j'}$ takes the form:

$$\begin{aligned} K_{ij,i'j'} &= \int_{S_{i'j'}} \mathcal{K}u_{ij}(x)u_{i'j'}(x)\,\mathrm{d}x \\ &= a\int_{S_{i'j'}}\int_0^x (x-s)^{\theta-1}u_{ij}(s)\,\mathrm{d}s\,u_{i'j'}(x)\,\mathrm{d}x \\ &= a\int_{S_{i'j'}}\int_{[0,x]\cap S_{ij}} (x-s)^{\theta-1}u_{ij}(s)u_{i'j'}(x)\,\mathrm{d}s\,\mathrm{d}x, \quad (i,j),(i',j')\in U_n. \end{aligned} \tag{2.33}$$

According to $S_{ij} := [q_{ij}, q'_{ij}]$ and $S_{i'j'} = [q_{i'j'}, q'_{i'j'}]$ the supports of $u_{ij}$ and $u_{i'j'}$ in the above equation, there are four different cases that are described in Figures 2.3.3-2.3.6.

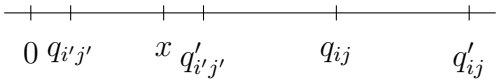



Figure 2.3.3: case 1

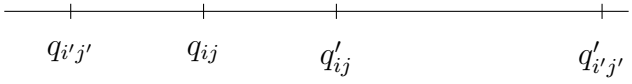



Figure 2.3.4: case 2

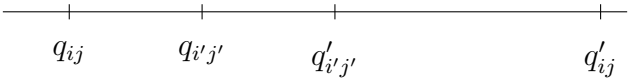



Figure 2.3.5: case 3

- Case 1. The support of $u_{i'j'}$ lies to the left of the one of $u_{ij}$, i.e. $q'_{i'j'} \le q_{ij}$, since $\mathrm{meas}([0,x]\bigcap S_{ij}) = 0$, we have $K_{ij,i'j'} = 0$.

- Case 2, 3. Either $[q_{i'j'}, q'_{i'j'}] \subseteq [q_{ij}, q'_{ij}]$ or $[q_{ij}, q'_{ij}] \subseteq [q_{i'j'}, q'_{i'j'}]$, $\mathrm{dist}(S_{ij}, S_{i'j'}) = 0$. The entries corresponding to those cases usually would be "large", so we just leave these unchanged.

- Case 4. The support of $u_{ij}$ lies to the left of the support of $u_{i'j'}$, i.e. $q'_{ij} \le q_{i'j'}$, when the distance between those two support sets is larger than some predefined threshold,


Figure 2.3.6: case 4

the absolute value of $K_{ij,i'j'}$ will be sufficiently small in magnitude to some degree of precision, they can be neglected without affecting the overall accuracy of the approximation.

Thus, we propose a matrix truncation strategy for the entries of $K_n$ that satisfy case 4. For that end, let's give an estimate of $K_{ij,i'j'}$ first. For the case that $q'_{ij} \leq q_{i'j'}$, we have the following estimate for $K_{ij,i'j'}$.

**Lemma 1.** *For $i, i' \in Z_{n+1}$, $\sigma$ as a positive integer, if $q'_{ij} \leq q_{i'j'}$ and there is a constant $r > 1$ such that*

$$\operatorname{dist}(S_{ij}, S_{i'j'}) \geq r(d_i + d_{i'}), \tag{2.34}$$

*then there exists a positive constant $c$ such that*

$$|K_{ij,i'j'}| \leq c(d_i d_{i'})^{\sigma-\frac{1}{2}} \int_{S_{i'j'}} \int_{S_{ij}} \frac{1}{(x-s)^{2\sigma+(1-\theta)}} \,\mathrm{d}s \,\mathrm{d}x, \tag{2.35}$$

*where $d_i := \operatorname{meas}(S_{ij})$.*

*Proof.* Let $x_0, s_0$ be centers of the sets $S_{ij}$ and $S_{i'j'}$, respectively. Obviously, for $x \neq s$, $k(x,s)$ has continuous partial derivatives, we apply Taylor's theorem to $k(x,s)$ at the point $x_0, s_0$, yielding

$$k(x,s) = p(x,s) + q(x,s) + \frac{(s-s_0)^\sigma (x-x_0)^\sigma}{(\sigma!)^2} R_\sigma(x,s),$$

where $p(x,\cdot), q(\cdot,s)$ are polynomials of total degree $\leq \sigma - 1$ in $x$ and $s$, respectively, and

$$R_\sigma(x,s) := \int_0^1 \int_0^1 \theta_1^{\sigma-1} \theta_2^{\sigma-1} D_x^\sigma D_s^\sigma k(x + \theta_2(x_0 - x)), s + \theta_1(s_0 - s)) \,\mathrm{d}\theta_2 \,\mathrm{d}\theta_1.$$

On the other hand, since $q'_{ij} \leq q_{i'j'}$ , we have $S_{ij} \cap [0,x] = S_{ij}$. By the vanishing moment

property of $u_{ij}$ and $u_{i'j'}$ (Proposition 4) we have that

$$\int_{S_{ij}} (\int_{S_{i'j'}} p(x,s)u_{i'j'}(x)\,\mathrm{d}x)u_{ij}(s)\,\mathrm{d}x = 0$$

and

$$\int_{S_{i'j'}} (\int_{S_{ij}} q(x,s)u_{ij}(s)\,\mathrm{d}s)u_{i'j'}(x)\,\mathrm{d}x = 0,$$

which leads to

$$K_{ij,i'j'} = a\int_{S_{i'j'}}\int_{S_{ij}} \frac{(s-s_0)^\sigma (x-x_0)^\sigma}{(\sigma!)^2} R_\sigma(x,s)u_{ij}(s)u_{i'j'}(x)\,\mathrm{d}s\,\mathrm{d}x.$$

With property 3 and property 4 of proposition 4, we continue to have the estimate that

$$|K_{ij,i'j'}| \le \frac{c}{(\sigma!)^2} d_i^{\sigma-\frac{1}{2}} d_{i'}^{\sigma-\frac{1}{2}} \int_{S_{i'j'}}\int_{S_{ij}} |R_\sigma(x,s)|\,\mathrm{d}s\,\mathrm{d}x. \tag{2.36}$$

By the mean value theorem, the following inequality

$$|R_\sigma(x,s)| \le (2\sigma-1)!(x'-s')^{-(2\sigma+(1-\theta))} \tag{2.37}$$

hold for some $x' \in S_{i'j'}$ and $s' \in S_{ij}$. Furthermore, for any $x \in S_{i'j'}$, $s \in S_{ij}$ the assumption (2.34) yields

$$(x'-s') \ge (1-r^{-1})(x-s), \tag{2.38}$$

Thus, combining (2.37) and (2.38) and substituting them into (2.36) proves the desired result. □

Next, for each $i, i' \in Z_{n+1}$, we define a truncation parameter as

$$\delta_{ii'} := \max\{\mu^{-n+\alpha(n-i)+\alpha'(n-i')}, r(d_i+d_{i'})\}. \tag{2.39}$$

For the matrix $\mathbf{K}_n$, we compress it into a new matrix $\widetilde{\mathbf{K}}_n := [\widetilde{\mathbf{K}}_{ii'} : i, i' \in Z_{n+1}]$, where each

block $\widetilde{\mathbf{K}}_{ii'}$ as a compressed matrix of $\mathbf{K}_{ii'}$ is given by

$$\widetilde{K}_{ij,i'j'} := \begin{cases} 0, & \text{if } \ q_{i'j'} - q'_{ij} \geq \delta_{ii'} \\ 0, & \text{if } \ q'_{i'j'} \leq q_{ij} \\ K_{ij,i'j'}, & \text{otherwise.} \end{cases} \tag{2.40}$$

We then replace the matrix $\mathbf{K}_n$ in the linear system (2.32) by $\widetilde{\mathbf{K}}_n$ and find a vector $\widetilde{\mathbf{b}}_n = [\widetilde{b}_{ij} : (i,j) \in U_n]$ that solves the compressed linear system

$$(\mathbf{E}_n + \widetilde{\mathbf{K}}_n)\widetilde{\mathbf{b}}_n = \mathbf{f}_n. \tag{2.41}$$

Next, we introduce an abstract operator $\widetilde{\mathcal{K}_n}$ which corresponds to the compressed matrix $\widetilde{\mathbf{K}}$.

**Definition 2.** *Define a linear operator $\widetilde{\mathcal{K}}_n$ on $\mathcal{X}_n$ as*

$$\widetilde{\mathcal{K}}_n u_{ij} = \sum_{(l,k)\in U_n} b_{lk,ij} u_{lk},$$

*and*

$$\widetilde{\mathcal{K}}_n(\alpha u_{ij} + \beta u_{i'j'}) = \alpha \widetilde{\mathcal{K}}_n u_{ij} + \beta \widetilde{\mathcal{K}}_n u_{ij},$$

*where*

$$b_{lk,ij} = (\widetilde{\mathbf{K}}_n \mathbf{E}_n^{-1})_{lk,ij}.$$

Then we have the following proposition.

**Proposition 5.** *Solving the linear integral equation (2.41) is equivalent to finding*

$$\widetilde{u}_n = \sum_{(i,j)\in U_n} \widetilde{b}_{ij} u_{ij} \in \mathcal{X}_n$$

*such that*

$$(\mathcal{I} + \widetilde{\mathcal{K}}_n)\widetilde{u}_n = \mathcal{P}_n f. \tag{2.42}$$

*Proof.* For $(i,j),(i',j') \in U_n$,

$$(\widetilde{\mathcal{K}}_n u_{ij}, u_{i'j'}) = \sum_{(l,k)\in U_n} b_{lk,ij}(u_{lk}, u_{i'j'}) = (\widetilde{\mathbf{K}}_n \mathbf{E}_n^{-1}\mathbf{E}_n)_{ij,i'j'} \tag{2.43}$$

Equation (2.43) implies that

$$\widetilde{\mathbf{K}}_n = \{(\widetilde{\mathcal{K}}_n u_{ij}, u_{i'j'}) : (i,j),(i',j') \in U_n\},$$

we complete the proof. □

We give the estimate of the discrepancy between $\|\mathbf{K}_{i'i} - \widetilde{\mathbf{K}}_{i'i}\|_2$ in the next lemma. For convenience, we first introduce a notation

$$D^{j'}_{\delta_{ii'}} := \{j : j \in Z_{w(i)}, q_{i'j'} - q'_{ij} > \delta_{ii'}\}.$$

Obviously, for fixed $i, i', j'$, $D^{j'}_{\delta_{ii'}}$ is the index set of $j$ corresponding to $K_{ij,i'j'}$ that we truncated off.

**Lemma 2.** *For a positive integer $\sigma$, then for any $r > 1$, there exists a constant $c$ such that if we choose truncation parameter $\delta_{ii'} := \max\{\mu^{-n+\alpha(n-i)+\alpha'(n-i')}, r(d_i + d_{i'})\}$, the following inequality holds,*

$$\|\mathbf{K}_{ii'} - \widetilde{\mathbf{K}}_{ii'}\|_2 \le c(d_i d_{i'})^{\sigma} \delta_{ii'}^{-2\sigma+\theta} \tag{2.44}$$

*Proof.* First of all, since the spectral radius of a matrix is less than or equal to any of its matrix norms, the following inequality holds,

$$\|\mathbf{K}_{ii'} - \widetilde{\mathbf{K}}_{ii'}\|_2^2 \le \|\mathbf{K}_{ii'} - \widetilde{\mathbf{K}}_{ii'}\|_1 \|\mathbf{K}_{ii'} - \widetilde{\mathbf{K}}_{ii'}\|_\infty, \tag{2.45}$$

now we just need to estimate $\|\mathbf{K}_{ii'}-\widetilde{\mathbf{K}}_{ii'}\|_\infty$ and $\|\mathbf{K}_{ii'}-\widetilde{\mathbf{K}}_{ii'}\|_1$,respectively. By the definition of $\widetilde{\mathbf{K}}_{ii'}$, we have

$$\|\mathbf{K}_{ii'}-\widetilde{\mathbf{K}}_{ii'}\|_\infty = \max_{j'\in Z_{w(i')}} \sum_{j\in D^{j'}_{\delta_{ii'}}} |K_{ij,i'j'}|,$$

then following the estimate of $K_{ij,i'j'}$ from lemma 1, we further have

$$\begin{aligned}
\|\mathbf{K}_{ii'}-\widetilde{\mathbf{K}}_{ii'}\|_\infty &\le c(d_i d_{i'})^{\sigma-\frac{1}{2}} \max_{j'\in Z_{w(i')}} \sum_{j\in D^{j'}_{\delta_{ii'}}} \int_{S_{i'j'}}\int_{S_{ij}} \frac{1}{(x-s)^{2\sigma+(1-\theta)}}\,\mathrm{d}s\,\mathrm{d}x \\
&\le c(d_i d_{i'})^{\sigma-\frac{1}{2}} d_{i'} \max_{j'\in Z_{w(i')}} \max_{x\in S_{i'j'}} \sum_{j\in D^{j'}_{\delta_{ii'}}} \int_{S_{ij}} \frac{1}{(x-s)^{2\sigma+(1-\theta)}}\,\mathrm{d}s \\
&\le c(d_i d_{i'})^{\sigma-\frac{1}{2}} d_{i'} \int_{t>\delta_{ii'}} \frac{1}{t^{2\sigma+(1-\theta)}}\,\mathrm{d}t \\
&\le c(d_i d_{i'})^{\sigma-\frac{1}{2}} d_{i'} \delta_{ii'}^{-2\sigma+\theta}.
\end{aligned} \tag{2.46}$$

Similarly, by the definition of the $L^1$ norm of a matrix, we have

$$\begin{aligned}
\|\mathbf{K}_{ii'}-\widetilde{\mathbf{K}}_{ii'}\|_1 &= \max_{j\in Z_{w(i)}} \sum_{j'\in D^{j}_{\delta_{ii'}}} |K_{ij,i'j'}| \\
&\le c(d_i d_{i'})^{\sigma-\frac{1}{2}} \max_{j\in Z_{w(i)}} \sum_{j'\in D^{j}_{\delta_{ii'}}} \int_{S_{i'j'}}\int_{S_{ij}} \frac{1}{(x-s)^{2\sigma+(1-\theta)}}\,\mathrm{d}s\,\mathrm{d}x \\
&\le c(d_i d_{i'})^{\sigma-\frac{1}{2}} d_i \max_{j'\in Z_{w(i')}} \max_{s\in S_{ij}} \sum_{j'\in D^{j}_{\delta_{ii'}}} \int_{S_{i'j'}} \frac{1}{(x-s)^{2\sigma+(1-\theta)}}\,\mathrm{d}x \\
&\le c(d_i d_{i'})^{\sigma-\frac{1}{2}} d_i \int_{t>\delta_{ii'}} \frac{1}{t^{2\sigma+(1-\theta)}}\,\mathrm{d}t \\
&\le c(d_i d_{i'})^{\sigma-\frac{1}{2}} d_i \delta_{ii'}^{-2\sigma+\theta}.
\end{aligned} \tag{2.47}$$

Substituting the above two estimates into the right-hand side of (2.45) yields the desired results. □

The next theorem provides a stability estimate for operator $\mathcal{I}-\widetilde{\mathcal{K}}_n$.

**Theorem 2.3.1.** *Let $\sigma$ be a positive integer and $r>1$. Suppose that we choose truncation*

*parameter* $\delta_{ii'} := \max\{\mu^{-n+\alpha(n-i)+\alpha'(n-i')}, r(d_i + d_{i'})\}$ *with* $\alpha = 1, \alpha' > \frac{\sigma}{2\sigma-\theta}$, *then* $(\mathcal{I} - \widetilde{\mathcal{K}}_n)^{-1}$ *exists and is uniformly bounded.*

*Proof.* According to the proof of Lemma 5.11 in [1] and the result from Lemma 5, for any $u, v \in \mathcal{X}$, we have

$$\begin{aligned} |(\mathcal{K}_n - \widetilde{\mathcal{K}}_n)\mathcal{P}_n u, \mathcal{P}_n v)| &\leq \sum_{i,i' \in Z_{n+1}} c(d_i d_{i'})^{\sigma} (\delta_{ii'})^{-(2\sigma-\theta)} \|u\|_2 \|v\|_2 \\ &\leq \sum_{i,i' \in Z_{n+1}} \mu^{(\sigma-\alpha(2\sigma-\theta))(n-i)+(\sigma-\alpha'(2\sigma-\theta))(n-i')} \mu^{-\theta n} \|u\|_2 \|v\|_2 \end{aligned} \tag{2.48}$$

On the other hand, $(\mathcal{K}_n - \widetilde{\mathcal{K}}_n) = \mathcal{P}_n(\mathcal{K}_n - \widetilde{\mathcal{K}}_n)$ leads to

$$\|(\mathcal{K}_n - \widetilde{\mathcal{K}}_n)\mathcal{P}_n u\| = \sup_{v \in \mathcal{X}, \|v\|_2 \neq 0} \frac{|(\mathcal{P}_n(\mathcal{K}_n - \widetilde{\mathcal{K}}_n)\mathcal{P}_n u, v)|}{\|v\|_2} = \sup_{v \in \mathcal{X}, \|v\|_2 \neq 0} \frac{|(\mathcal{K}_n - \widetilde{\mathcal{K}}_n)\mathcal{P}_n u, \mathcal{P}_n v)|}{\|v\|_2} \tag{2.49}$$

Since there exists a constant $c$ such that $\mu^{(\sigma-\alpha(\sigma-\theta)(n-i)+(\sigma-\alpha'(\sigma-\theta)(n-i')} \mu^{-\theta n} \leq c$, (see the proof of Theorem 5.12 in [1]), combining equation (2.48) with (2.49) yields

$$\|(\mathcal{K}_n - \widetilde{\mathcal{K}}_n)v\| \leq c\|v\|_2. \text{ for } v \in \mathcal{X}_n.$$

This, with the stability estimate of the standard Galerkin method yields

$$\|(\mathcal{I} - \widetilde{\mathcal{K}}_n)v\| \geq \|(\mathcal{I} - \mathcal{K}_n)v\| - \|(\mathcal{K}_n - \widetilde{\mathcal{K}}_n)v\| \geq c\|v\|_2,$$

for any $v \in \mathcal{X}_n$. The above stability estimate ensures the desired result of this Theorem. □

We establish the convergence order of $\widetilde{u}_n$ in the following theorem.

**Theorem 2.3.2.** *Let* $\sigma$ *be a positive integer,* $u \in \mathcal{H}^{\sigma}(I)$ *be the exact solution of equation (2.7). Suppose the truncation parameters* $\delta_{ii'}$ *are chosen as*

$$\delta_{ii'} := \max\{\mu^{-n+\alpha(n-i)+\alpha'(n-i')}, r(d_i + d_{i'})\} \tag{2.50}$$

*with* $\alpha = 1$ *and* $\alpha' > \frac{\sigma}{2\sigma-\theta}$, *then there exists a constant* $c$ *such that*

$$\|u - \widetilde{u}_n\| \leq cs(n)^{-\sigma}\|u\|_{\mathcal{H}^\sigma(I)},$$

*Proof.* See the proof of theorem 5.13 in [1]. □

### 2.3.1 Computational Complexity

The number of non-zero entries of the compressed matrix is used to evaluate the computational complexity of the proposed multiscale Galerkin method. In the next Theorem, we give an estimate for $\mathcal{N}(\widetilde{\mathbf{K}}_n)$, the number of nonzero entries in matrix $\widetilde{\mathbf{K}}_n$.

**Theorem 2.3.3.** *For each* $i, i' \in Z_{n+1}$, *and arbitrary* $r > 1$, *we choose the truncation parameter* $\delta_{ii'}$ *satisfy*

$$\delta_{ii'} \leq \max\{\mu^{-n+\alpha(n-i)+\alpha'(n-i')}, r(d_i + d_{i'})\} \tag{2.51}$$

*then*

$$\mathcal{N}(\widetilde{\mathbf{K}}_n) := \begin{cases} \mathcal{O}(s(n)\log^2(s(n)), & \alpha = \alpha' = 1, \\ \mathcal{O}(s(n)\log(s(n)), & \text{otherwise.} \end{cases} \tag{2.52}$$

*Proof.* We first estimate the number of nonzero entries of block $\widetilde{\mathbf{K}}_{ii'}$. we denote

$$S^i_{i'j'} := \{x \in I : 0 \leq q'_{i'j'} - x \leq d_i + d_{i'} + \delta_{ii'} \text{ or } 0 \leq x - q'_{i'j'} \leq d_i\}\}.$$

The zero entries of block $\widetilde{\mathbf{K}}_{ii'}$ mainly come from two parts. Part one includes those entries due to the truncation procedure while the other part contains those entries involving the basis functions $u_{ij}$ and $u_{i'j'}$ satisfying $q'_{i'j'} \leq q_{ij}$. Thus, if $\widetilde{K}_{ij,i'j'} \neq 0$, $S_{ij}$ must be a subset of

$S^i_{i'j'}$. Since there are $k$ basis functions $u_{ij}$ having support contained in $S_{ij}$, we have

$$\begin{aligned}\mathcal{N}(\widetilde{\mathbf{K}}_{i,i'}) &\le \sigma \sum_{j'\in w(i')} \frac{\mathrm{meas}(S^i_{i'j'})}{\mu^{-(i-1)}} \le k^2(2d_i + d_{i'} + \delta_{ii'})\mu^{i+i'-1} \\ &\le \sigma^2\mu^{i+i'-1}((r+1)(d_i+d_{i'}) + d_i + \mu^{-n+\alpha(n-i)+\alpha'(n-i')}).\end{aligned} \tag{2.53}$$

Because

$$\mathcal{N}(\widetilde{\mathbf{K}}_n) = \sum_{i'\in Z_{n+1}} \sum_{i\in Z_{n+1}} \mathcal{N}(\widetilde{\mathbf{K}}_{ii'}), \tag{2.54}$$

we substitute estimate (2.53) into (2.54), and obtain

$$\begin{aligned}\mathcal{N}(\widetilde{\mathbf{K}}_n) &\le \sum_{i'\in Z_{n+1}} \sum_{i\in Z_{n+1}} \sigma^2\mu^{i+i'-1}((r+2)\mu^{-i+1} + (r+1)\mu^{-i'+1}) + \mu^{-n+\alpha(n-i)+\alpha'(n-i')}) \\ &= 2\sigma^2(r+2)(n+1)\sum_{i\in Z_{n+1}} \mu^i + \mu^{n-1}\sum_{i\in Z_{n+1}} \mu^{(\alpha-1)(n-i)} \sum_{i'\in Z_{n+1}} \mu^{(\alpha'-1)(n-i')} \\ &= \begin{cases} \mathcal{O}(\mu^n(n+1)^2), & \alpha = \alpha' = 1, \\ \mathcal{O}(\mu^n(n+1)), & \text{otherwise}, \end{cases}\end{aligned} \tag{2.55}$$

as $n \to \infty$. Finally, the desired result of this theorem follows from the fact that $\mu^n \sim s(n)$. □

In order to solve (2.41), we still need to estimate the nonzero entries of the matrix $\widetilde{K}_n$. In the next section, we will describe a numerical quadrature scheme which is specially designed according to our singular kernel so that the quadrature errors will not ruin the overall convergence order of the approximate solution of the integral equation.

## 2.4 A Numerical Quadrature Scheme

The nonzero entries of the matrix $\widetilde{\mathbf{K}}_n$ are defined in terms of double integrals whose integrands involve the products of the weakly singular kernel and a piecewise polynomial. In this section, we shall present a numerical quadrature scheme to compute these nonzero

entries.

The entries of matrix $\widetilde{\mathbf{K}}_n$ are integrals of integrands in the form

$$k_{ij,i'j'}(x) := (\int_0^x k(x,s)u_{ij}(s)\mathrm{d}s)u_{i'j'}(x),$$

for $x \in S_{i'j'}$ and $(i,j) \in U_n$. To compute $K_{ij,i'j'} := I(k_{ij,i'j'})$, we divide $S_{i'j'}$ into $p$ equal subintervals of width $h := \frac{\mu^{-i'+1}}{p}$ with endpoints $[t_{\overline{a}}^{i'}, t_{\overline{a}+1}^{i'}]$, for $\overline{a} \in Z_p$ and $t_{\overline{a}}^{i'} := q_{i'j'} + ah$. Then we use the following composite $q$ points Gaussian quadrature rule

$$I(S(k_{ij,i'j'})) := \sum_{\overline{a}\in Z_p}\sum_{\iota\in Z_q} \omega_\iota^{\overline{a}} k_{ij,i'j'}(\tau_\iota^{\overline{a}}),$$

to approximate $I(k_{ij,i'j'})$. Here $\omega_\iota^{\overline{a}}$ and $\tau_\iota^{\overline{a}}$ are the associated Gauss weights and Gauss points on $[t_{\overline{a}}^{i'}, t_{\overline{a}+1}^{i'}]$, respectively.

Next, we turn to estimate each $k_{ij,i'j'}(\tau_\iota^{\overline{a}})$. To this end, for any $\gamma \in (0,1)$ and $w \in N$, we define set $\pi^{ii'} := \{\zeta_l := x - \gamma^l : l \in Z_w\} \cup \{\zeta_w := x\}$, rearrange the elements of

$$(\pi^{ii'} \cup \{q_{ij}, q'_{ij}\}) \cap S_{ij}$$

in the increasing order, and write them as a new sequence $q_{ij} = q_0 < q_1 < \cdots < q_{m'} = \min\{q'_{ij}, x\}$ with $m' \le w+2$. We define

$$k_{ij}(x,s) := k(x,s)u_{ij}(s)$$

and the associated partition of $[0,x] \cap S_{ij}$ as

$$\Pi(k_{ij}) := \{Q_\alpha := [q_\alpha, q_{\alpha+1}) : \alpha \in Z_{m'}\}.$$

Obviously, for any $Q_\alpha$, there exists an $l \in Z_w$ such that $Q_\alpha \subset [\zeta_l, \zeta_{l+1}]$. We approximate $k_{ij,i'j'}$ on each interval $Q_\alpha \subset [\zeta_l, \zeta_{l+1}]$ using the Gauss-Legendre quadrature. That is, we first

construct a piecewise polynomial $S_\alpha(k_{ij})$ of order $k'$ which interpolates $k_{ij}$ at the Gauss points on $Q_\alpha$ and is equal to zero outside $Q_\alpha$, in case that $x \le q'_{ij}$, we set $S_{m'-1}(k_{ij}) = 0$. Next, we define $S(k_{ij}) = \sum_{\alpha \in Z_{m'}} S_\alpha(k_{ij})$ and use $I(S(k_{ij}))(x)u_{i'j'}(x)$ to approximate $k_{ij,i'j'}(x)$.

In the rest of this section, we analyze the convergence order of this integration method. We first introduce two index sets

$$Z^{j'}_{\delta_{ii'}} := \{j : j \in Z_{w(i)}, \mathrm{dist}(S_{ij}, S_{i'j'}) \le \delta_{ii'} \ \text{ and } \ q'_{i'j'} \ge q_{ij}\},$$

and

$$Z^{\ell}_{i'j',i} := \{j \in Z^{j'}_{\delta_{ii'}} : j = \upsilon(\mathbf{e}_i)\sigma + \ell\},$$

for some $\mathbf{e}_i \in Z^{i-1}_{\mu}$. Then, for $\ell \in Z_\sigma$ and $(i', j') \in U_n$, we set

$$\overline{u}_{i\ell,i'j'} = \sum_{j \in Z^{\ell}_{i'j',i}} u_{ij},$$

and

$$h_{i\ell,i'j'}(x) = (\int_0^x k(x,s)\overline{u}_{i\ell,i'j'}(s)\,\mathrm{d}s)u_{i'j'}(x),$$

Obviously, for $j_1, j_2 \in Z^{l}_{i'j',i}$, if $j_1 \ne j_2$, $\mathrm{meas}(S_{ij_1} \bigcap S_{ij_2}) = 0$.

It follows the above composite $q$ point Gaussian quadrature rule that

$$I(S(h_{i\ell,i'j'})) := \sum_{\overline{a} \in Z_p} \sum_{\iota \in Z_q} \omega^{\overline{a}}_{\iota} h_{i\ell,i'j'}(\tau^{\overline{a}}_{\iota}).$$

Next, we give the approximation of the $I(S(h_{i\ell,i'j'}))$ according to our quadrature strategy. Let

$$\pi(h_{i\ell,i'j'}) = \cup_{j \in Z^{\ell}_{i'j',i}} (\pi^{ii'} \cup \{q_{ij}, q'_{ij}\}) \cap S_{ij},$$

then we rearrange the elements of $\pi(h_{i\ell,i'j'})$ in the increasing order and write them as a new sequence $\overline{q}_{i\ell} = \overline{q}_0 < \overline{q}_1 \cdots < \overline{q}_{m''} = \min\{\overline{q}'_{i\ell}, x\}$.

Associated with the above notations, we define

$$f_{i\ell,i'j'} := k(x,s)\overline{u}_{il,i'j'}(s),$$

$$\Pi(f_{i\ell,i'j'}) := \{\overline{Q}_\alpha := [\overline{q}_\alpha, \overline{q}_{\alpha+1}) : \alpha \in Z_{m''}\}.$$

and

$$S(f_{i\ell,i'j'}) = \sum_{\alpha\in Z_{m''}} S_\alpha(f_{i\ell,i'j'}), \qquad \widetilde{h}_{i\ell,i'j'} = I(S(f_{i\ell,i'j'}))u_{i'j'},$$

Our next lemma gives the estimate of the difference between $I(h_{i\ell,i'j'})$ and $I(S(\widetilde{h}_{i\ell,i'j'}))$.

**Lemma 3.** *Let $E_{i\ell,i'j'} := I(h_{i\ell,i'j'}) - I(S(\widetilde{h}_{i\ell,i'j'}))$, for $p,q \in N$, $p>0, q>0$, there exists a positive constant $c$ such that for any $i \in Z_{n+1}$, $\ell \in Z_\sigma$, $(i',j') \in U_n$ and $\gamma \in (0,1)$, we have the estimate*

$$E_{i\ell,i'j'} \le c_0 p^{-2q} + cpq(\frac{c_1\gamma^{\theta w}}{\theta} + c_2(\frac{1-\gamma}{\gamma})^{2k'+1}\frac{\gamma^\theta}{1-\gamma^\theta}[(d_i + d_{i'} + \delta_{ii'})\mu^{i-1}]^k),$$

The proof of this lemma requires one technical lemma. Actually, by the triangle inequality, we have

$$|E_{i\ell,i'j'}| \le |I(h_{il,i'j'}) - I(S(h_{i\ell,i'j'}))| + |I(S(h_{i\ell,i'j'})) - I(S(\widetilde{h}_{i\ell,i'j'}))| \tag{2.56}$$

For the first term on the right side of (2.56), according to the error estimate of the composite Gauss formula, there exists a positive constant $c_0$ such that

$$|I(h_{i\ell,i'j'}) - I(S(h_{i\ell,i'j'}))| \le c_0 p^{-2q} \tag{2.57}$$

By choosing $p,q$ large enough, this error will be sufficiently small. Thus, we just need to estimate the second term and we have the following lemma.

**Lemma 4.** *There exists a constant $c$ such that for any $i \in Z_{n+1}, j \in Z^{j'}_{\delta_{ii'}}$, $(i',j') \in U_n$ and*

$\gamma \in (0,1)$,

$$|I(S(h_{i\ell,i'j'}))-I(S(\widetilde{h}_{i\ell,i'j'}))| \le cpq(\frac{c_1\gamma^{\theta w}}{\theta}+c_2(\frac{1-\gamma}{\gamma})^{2k'+1}\frac{\gamma^{\theta}}{1-\gamma^{\theta}}[(d_i+d_{i'}+\delta_{ii'})\mu^{i-1}]^{\sigma}). \quad (2.58)$$

*Proof.* We introduce index set $\Gamma_l := \{\alpha \in Z_{m''} : \overline{Q}_\alpha \in \Pi(f_{i\ell,i'j'}), \overline{Q}_\alpha \subset [\zeta_l, \zeta_{l+1}]\}$, for $l \in Z_w$. Associated with these index sets, we set

$$E_l(f_{i\ell,i'j'}) := \sum_{\alpha\in\Gamma_l}\int_{\overline{Q}_\alpha} |f_{i\ell,i'j'} - S(f_{i\ell,i'j'})|\,\mathrm{d}s$$

and

$$E(f_{i\ell,i'j'}(x)) := \sum_{\Gamma_l\neq\emptyset} E_l(f_{i\ell,i'j'}).$$

We first estimate $E_{w-1}(f_{i\ell,i'j'})$. If $x < \overline{q}'_{i\ell}$, we have

$$E_{w-1}(f_{i\ell,i'j'}) \le a\int_{\zeta_{w-1}}^{x} |(x-s)^{\theta-1}\overline{u}_{i\ell,i'j'}(s)|\,\mathrm{d}s \le \frac{c_1\gamma^{\theta w}}{\theta}. \quad (2.59)$$

Otherwise, for all $l \in Z_w$, by the error estimate of Gaussian quadrature, there exist $\eta_\alpha \in \overline{Q}_\alpha$ such that

$$E_l(f_{i\ell,i'j'}) := \sum_{\alpha\in\Gamma_l}\frac{|D_s^{2k'}f_{il,i'j'}(x,\eta_\alpha)|}{(2k')!}\left|\int_{\overline{Q}_\alpha}(s-\tau_0^\alpha)^2\cdots(s-\tau_{k'-1}^\alpha)^2\,\mathrm{d}s\right|,$$

where $\tau_i^\alpha, i \in Z_{k'}$ are the $k'$ zeros of the Legendre polynomial of degree $k'$ on $\overline{Q}_\alpha$. By the Leibniz rule of a product of two functions, we obtain that

$$\begin{aligned}|D_s^{2k'}f_{i\ell,i'j'}(\eta_\alpha)| &\le \sum_{\beta\in Z_\sigma} C_{2k'}^\beta |D^{2k'-\beta}k(x,\eta_\alpha)\overline{u}_{i\ell,i'j'}^\beta(\eta_\alpha)| \\ &\le \sum_{\beta\in Z_\sigma} C_{2k'}^\beta (x-\eta_\alpha)^{-(2k'+1-\theta-\beta)}\mu^{\beta(i-1)}|\sum_{j\in Z^\ell_{i'j',i}} u_{1\ell}^\beta \phi^{-1}_{\mathbf{e}(s)}|.\end{aligned} \quad (2.60)$$

Since $\gamma^{l+1} \leq x - \eta_\alpha \leq d_i + d_{i'} + \delta_{ii'}$, we have

$$\begin{aligned} E_l(f_{i\ell,i'j'}(x)) &\leq c\frac{\gamma^{-(l+1)(2k'+1-\theta)}}{(2k'-\sigma)!}(\gamma^l - \gamma^{l+1})^{2k'+1} \sum_{\beta \in Z_\sigma} [(d_i + d_{i'} + \delta_{ii'})\mu^{i-1}]^\beta \\ &\leq c_2 \left(\frac{1-\gamma}{\gamma}\right)^{2k'+1} \gamma^{(l+1)\theta}[(d_i + d_{i'} + \delta_{ii'})\mu^{i-1}]^\sigma. \end{aligned} \tag{2.61}$$

Therefore,

$$\begin{aligned} E(f_{i\ell,i'j'}) = \sum_{\Gamma_l \neq \emptyset} E_l(f_{i\ell,i'j'}) &\leq \frac{c_1\gamma^{(\theta)w}}{\theta} + \sum_{l=1}^{w-1} E_l(f_{ij}) \\ &= \frac{c_1\gamma^{(\theta)w}}{\theta} + c_2(\frac{1-\gamma}{\gamma})^{2k'+1}\gamma^\theta[(d_i + d_{i'} + \delta_{ii'})\mu^{i-1}]^\sigma \sum_{l=1}^{w-1} \gamma^{\theta l} \\ &\leq \frac{c_1\gamma^{(\theta)w}}{\theta} + c_2(\frac{1-\gamma}{\gamma})^{2k'+1}\frac{\gamma^\theta}{1-\gamma^\theta}[(d_i + d_{i'} + \delta_{ii'})\mu^{i-1}]^\sigma. \end{aligned} \tag{2.62}$$

Using estimate (2.62), we have

$$\begin{aligned} |I(S(h_{i\ell,i'j'})) - I(S(\widetilde{h}_{i\ell,i'j'}))| &= |\sum_{a\in Z_p}\sum_{\iota\in Z_q} \omega_\iota^a h_{i\ell,i'j'}(\tau_\iota^a) - \sum_{a\in Z_p}\sum_{\iota\in Z_q} \omega_\iota^a \widetilde{h}_{i\ell,i'j'}(\tau_\iota^a)| \\ &\leq \sum_{a\in Z_p}\sum_{\iota\in Z_q} \omega_\iota^a |h_{i\ell,i'j'}(\tau_\iota^a) - \widetilde{h}_{i\ell,i'j'}(\tau_\iota^a)| \\ &= \sum_{a\in Z_p}\sum_{\iota\in Z_q} \omega_\iota^a |(I(f_{i\ell,i'j'}(\tau_\iota^a)) - I(S(f_{i\ell,i'j'}(\tau_\iota^a))))u_{i'j'}(\iota^a)| \\ &\leq cpqE(f_{i\ell,i'j'}) \\ &\leq cpq(\frac{c_1\gamma^{(\theta)w}}{\theta} + c_2(\frac{1-\gamma}{\gamma})^{2k'+1}\frac{\gamma^\theta}{1-\gamma^\theta}[(d_i + d_{i'} + \delta_{ii'})\mu^{i-1}]^\sigma). \end{aligned} \tag{2.63}$$

□

Lemma 3 follows from lemma 4 and the triangle inequality (2.56). Similarly as before, we define $\hat{\mathbf{K}}_n := [\hat{\mathbf{K}}_{ii'} : i, i' \in Z_n]$ with

$$\hat{\mathbf{K}}_{ii'} := [\hat{K}_{ij,i'j'} : j' \in Z_{w(i)}, j \in Z_{w(j)}],$$

where

$$\hat{K}_{ij,i'j'} := \begin{cases} 0, & \text{if } \ q_{i'j'} - q'_{ij} \geq \delta_{ii'} \ \text{ or } \ q_{ij} > q'_{i'j'} \\ I(S((\widetilde{k}_{ij,i'j'}))), & \text{otherwise.} \end{cases} \tag{2.64}$$

for $(i,j),(i',j') \in U_n$.

Instead of solving (2.42), we solve

$$(\mathbf{E}_n + \hat{\mathbf{K}}_n)\hat{\mathbf{b}}_n = \mathbf{f}_n, \tag{2.65}$$

for $[\hat{b}_{ij} : (i,j) \in U_n]$ and use

$$\hat{u}_n := \sum_{(i,j)\in U_n} \hat{b}_{ij} u_{ij} \in \mathcal{X}_n$$

to approximate the exact solution $u$.

In the next lemma, we give an estimate for the discrepancy between the block $\widetilde{\mathbf{K}}_{ii'}$ and $\hat{\mathbf{K}}_{ii'}$.

**Lemma 5.** *There exists a positive constant $c$ such that for all $i, i' \in Z_{n+1}$ and $n \in N$,*

$$\|\widetilde{\mathbf{K}}_{ii'} - \hat{\mathbf{K}}_{ii'}\|_2 \leq c_0\sigma p^{-2q} + c_1\sigma pq(c\frac{\gamma^{\theta w}}{\theta} + c_2(\frac{1-\gamma}{\gamma})^{2k'+1}\frac{\gamma^{\theta}}{1-\gamma^{\theta}}[(d_i + d_{i'} + \delta_{ii'})\mu^{\hat{i}-1}]^{\sigma}). \tag{2.66}$$

*where $\hat{i} = \max\{i, i'\}$.*

*Proof.* We first note that

$$\|\hat{\mathbf{K}}_{ii'} - \widetilde{\mathbf{K}}_{ii'}\|_\infty = \max_{j'\in Z_{w(i')}} \sum_{l\in Z_\sigma} |E(h_{il,i'j'})|,$$

therefore, by using lemma 3, we have that

$$\|\hat{\mathbf{K}}_{ii'} - \widetilde{\mathbf{K}}_{ii'}\|_\infty \leq c_0\sigma p^{-2q} + c_1\sigma pq(\frac{c\gamma^{\theta w}}{\theta} + c_2(\frac{1-\gamma}{\gamma})^{2k'+1}\frac{\gamma^{\theta}}{1-\gamma^{\theta}}[(d_i + d_{i'} + \delta_{ii'})\mu^{i-1}]^{\sigma}).$$

Likewise, we can prove that

$$\|\hat{\mathbf{K}}_{ii'} - \widetilde{\mathbf{K}}_{ii'}\|_1 \leq c_0\sigma p^{-2q} + c_1\sigma pq(\frac{c\gamma^{\theta w}}{\theta} + c_2(\frac{1-\gamma}{\gamma})^{2k'+1}\frac{\gamma^\theta}{1-\gamma^\theta}[(d_i + d_{i'} + \delta_{ii'})\mu^{i'-1}]^\sigma).$$

Since

$$\|\hat{\mathbf{K}}_{ii'} - \widetilde{\mathbf{K}}_{ii'}\|_2^2 \leq \|\hat{\mathbf{K}}_{ii'} - \widetilde{\mathbf{K}}_{ii'}\|_\infty \|\hat{\mathbf{K}}_{ii'} - \widetilde{\mathbf{K}}_{ii'}\|_1, \tag{2.67}$$

we have

$$\|\hat{\mathbf{K}}_{ii'} - \widetilde{\mathbf{K}}_{ii'}\|_2 \leq c_0 kp^{-2q} + c_1 kpq(\frac{c\gamma^{\theta w}}{\theta} + c_2(\frac{1-\gamma}{\gamma})^{2k'+1}\frac{\gamma^\theta}{1-\gamma^\theta}[(d_i + d_{i'} + \delta_{ii'})\mu^{\hat{i}-1}]^\sigma). \tag{2.68}$$

□

Likewise, we introduce an abstract linear operator $\hat{\mathcal{K}}_n$ which corresponds to the matrix $\hat{\mathbf{K}}_n$. Similar as the proof of Proposition 5 that we presented in section 2.3, we also can show that solving the linear integral equation $(\mathcal{I} + \hat{\mathcal{K}}_n)\hat{u}_n = \mathcal{P}_n f$ is equivalent to solve the new linear system of equations (2.65).

In order to ensure that the numerical integration will not ruin the convergence order of the Galerkin method, we choose special $q, w$ for each $i, i' \in Z_{n+1}$ and use $q(i,i'), w(i,i')$ to indicate $q, w$ with respect to different $i, i'$, respectively. The existence and uniform boundness of operator $(\mathcal{I}+\hat{\mathcal{K}}_n)^{-1}$ is guaranteed by Lemma 5 and the specific way we choose $q(i,i'), w(i,i')$ in the next theorem (See proof of theorem 2.3.1).

**Theorem 2.4.1.** *If $u \in H^\sigma(I)$, for $i, i' \in Z_{n+1}$, $\delta_{ii'}$ are chosen as*

$$\delta_{ii'} := \max\{\mu^{-n+\alpha(n-i)+\alpha'(n-i')}, r(d_i + d_{i'})\}, \tag{2.69}$$

*with $\alpha = 1, 1 > \gamma > \frac{1}{2}$ and $\alpha' > \frac{\sigma}{2\sigma-\theta}$, moreover, for fixed $p \in N$, choose*

$$q(i,i') \geq \frac{\sigma(i+i')\log\mu}{2\log p}, \tag{2.70}$$

$$k' \geq \frac{\frac{-\sigma(\hat{i}+i+i')\log\mu}{\log(\frac{1-\gamma}{\gamma})} - 1}{2}, \tag{2.71}$$

*and*

$$w(i,i') \geq \frac{-\sigma(i+i')\log\mu}{\theta\log\gamma}, \tag{2.72}$$

*then there exists a positive constant $c$ and a positive integer $N$ such that for all $n > N$,*

$$\|u - \hat{u}\| \leq cs(n)^{-\sigma}\|u\|_{H^\sigma(I)}.$$

*Proof.* Since $(d_i + d_{i'} + \delta_{ii'}) \leq (r^{-1}+1)\delta_{ii'}$, we have

$$[(d_i + d_{i'} + \delta_{ii'})\mu^{\hat{i}-1}]^\sigma \leq c(\delta_{ii'}\mu^{\hat{i}-1})^\sigma.$$

In order to prove this theorem, according to the proof of lemma 5.13 in [1], it suffices to prove that

$$c_0 p^{-2q(i,i')} + c_1 pq(i,i')(\frac{c_1\gamma^{\theta w(i,i')}}{\theta} + c_3(\frac{1-\gamma}{\gamma})^{2k'+1}\frac{\gamma^\theta}{1-\gamma^\theta}[(d_i+d_{i'}+\delta_{ii'})\mu^{\hat{i}-1}]^\sigma) \leq c\delta_{ii'}^{-(2\sigma-\theta)}\mu^{-\sigma(i+i')},$$

we only need to show that by the way we choose $q(i,i'), w(i,i')$, for some positive constants $c$, the following inequalities hold:

$$p^{-2q(i,i')} \leq c\delta_{ii'}^{-(2\sigma-\theta)}\mu^{-\sigma(i+i')},$$

$$pq(i,i')\frac{c_1\gamma^{\theta w(i,i')}}{\theta} \leq c\delta_{ii'}^{-(2\sigma-\theta)}\mu^{-\sigma(i+i')},$$

and

$$(\frac{1-\gamma}{\gamma})^{2k'+1}\frac{\gamma^\theta}{1-\gamma^\theta}[(d_i + d_{i'} + \delta_{ii'})\mu^{\hat{i}-1}]^\sigma) \leq c\delta_{ii'}^{-(2\sigma-\theta)}\mu^{-\sigma(i+i')}.$$

By taking the logarithmic function on both side, one can verify this hold, thus, the conclusion

of this theorem follows. On the other hand, by triangle inequality, we have

$$\|\mathbf{K}_{ii'} - \hat{\mathbf{K}_{ii'}}\|_2 \leq \|\mathbf{K}_{ii'} - \widetilde{\mathbf{K}}_{ii'}\|_2 + \|\widetilde{\mathbf{K}}_{ii'} - \hat{\mathbf{K}}_{ii'}\|_2 \leq c(\delta_{ii'})^{-(2\sigma-\theta)}\mu^{-\sigma(i+i')} \tag{2.73}$$

By proof of theorem 5.13 in [1], we can have the desired results.

□

## 2.5 Numerical experiments

In this section, we present numerical examples to demonstrate the accuracy of the proposed multiscale Galerkin method and to verify the error estimates derived in the previous section. The computations associated with the examples are performed by visual studio 2017.

We choose $\mu = 2, r = 1.2, \gamma = 0.8, q = 32, p = 5$ in the following numerical experience. The basis functions of $\mathcal{X}_0$ are given by

$$u_{00} = 1, \quad u_{01} = \sqrt{3}(2t-1), \quad u_{02} = \sqrt{5}(6t^2 - 6t + 1)$$

and the orthonormal basis function of $\mathcal{W}_1$ are given by

$$u_{10} := \begin{cases} 1 - 6t, & t \in [0, \frac{1}{2}] \\ 5 - 6t, & t \in (\frac{1}{2}, 1]. \end{cases} \tag{2.74}$$

$$u_{11} := \begin{cases} \frac{\sqrt{93}}{31}(240t^2 - 116t + 9), & t \in [0, \frac{1}{2}] \\ \frac{\sqrt{93}}{31}(3 - 4t), & t \in (\frac{1}{2}, 1]. \end{cases} \tag{2.75}$$

$$u_{12} := \begin{cases} \frac{\sqrt{93}}{31}(4t - 1), & t \in [0, \frac{1}{2}] \\ \frac{\sqrt{93}}{31}(240t^2 - 364t + 133), & t \in (\frac{1}{2}, 1]. \end{cases} \tag{2.76}$$

Table II: $L^2$ error between approximate solution and exact solution for Example 2.5.1.

| $\sigma$ | $n$ | $s(n)$ | $\|u - u_n\|_2$ |
|---|---|---|---|
| 3 | 5 | 96 | $6.310376e-04$ |
| 3 | 6 | 192 | $1.189533e-04$ |

Table III: Absolute Error between approximate solution and exact solution for Example 2.5.1.

| $x$ | Exact Solution | Absolute Error for $n=5$ | Absolute Error for $n=6$ |
|---|---|---|---|
| 0.0 | 0.000000 | $0.000000e+00$ | $0.000000e+00$ |
| 0.1 | 0.316288 | $2.722910e-06$ | $8.077163e-06$ |
| 0.2 | 0.447214 | $1.152158e-05$ | $1.097334e-06$ |
| 0.3 | 0.547723 | $1.530953e-05$ | $1.476194e-06$ |
| 0.4 | 0.632456 | $2.440685e-05$ | $6.315013e-06$ |
| 0.5 | 0.707107 | $2.108546e-05$ | $2.477836e-06$ |
| 0.6 | 0.774597 | $2.085941e-05$ | $9.794490e-07$ |
| 0.7 | 0.836660 | $2.231643e-05$ | $1.578995e-06$ |
| 0.8 | 0.894427 | $2.201259e-05$ | $6.668973e-07$ |
| 0.9 | 0.948683 | $2.587488e-05$ | $4.117267e-06$ |
| 1.0 | 1.000000 | $2.755061e-05$ | $4.593362e-06$ |

**Example 2.5.1** Consider the weakly singular VIE of the second kind with $\theta = 1/2$, $a = \frac{1}{\Gamma(1/2)}$,

$$u(x) + \frac{1}{\Gamma(1/2)} \int_0^x \frac{u(s)}{\sqrt{x-s}}\, \mathrm{d}s = \Gamma(3/2)x + x^{1/2}.$$

The exact solution to the above integral equation is $u(x) = x^{1/2}$. The computed results are listed in Table II and III.

For this example, $u_n$ approximate $u$ in order $10^{-6}$ except on a small interval around $[1/256, 1/128]$ where the approximate order is $10^{-4}$, by taking larger $\sigma$ or $n$, the maximum error can be reduced quite well. For example, for $\sigma = 4, n = 7$, $\|u - u_n\|_2$ is reduced to order $10^{-6}$.

Table IV: $L^2$ error between the approximate solution and the exact solution for Example 2.5.2.

| $\sigma$ | $n$ | $s(n)$ | $\|u-u_n\|_2$ |
|---|---|---|---|
| 3 | 5 | 96 | $1.915886e-05$ |
| 3 | 6 | 192 | $6.104507e-06$ |

Table V: Absolute Error between the approximate solution and the exact solution for Example 2.5.2.

| $x$ | Exact Solution | Absolute Error for $n=5$ | Absolute Error for $n=6$ |
|---|---|---|---|
| 0.0 | 0.00 | $0.000000e+00$ | $0.000000e+00$ |
| 0.1 | 0.01 | $6.147957e-07$ | $1.128921e-05$ |
| 0.2 | 0.04 | $1.410004e-07$ | $1.935597e-06$ |
| 0.3 | 0.09 | $1.331423e-06$ | $3.114641e-06$ |
| 0.4 | 0.16 | $1.428655e-05$ | $6.661716e-06$ |
| 0.5 | 0.25 | $7.754397e-06$ | $3.004420e-06$ |
| 0.6 | 0.36 | $1.716972e-05$ | $2.106952e-06$ |
| 0.7 | 0.49 | $2.300065e-05$ | $3.140744e-06$ |
| 0.8 | 0.64 | $2.653462e-05$ | $9.921378e-07$ |
| 0.9 | 0.81 | $3.505066e-05$ | $3.932623e-06$ |
| 1.0 | 1.00 | $3.925652e-05$ | $2.004118e-06$ |

**Example 2.5.2** Consider the weakly singular VIE of the second kind with $\theta = 1/2$, $a = 1$,

$$u(x) + \int_0^x \frac{u(s)}{\sqrt{x-s}}\,\mathrm{d}s = x^2 + \frac{16}{15}x^{\frac{5}{2}}.$$

The exact solution is $u(x) = x^2$. Numerical results are listed in Table IV and V.

**Example 2.5.3** Consider the weakly singular VIE of the second kind with $\theta = 0.9$, $a_1 = \frac{1}{\Gamma(0.9)}$,

$$u(x) + \frac{1}{\Gamma(0.9)}\int_0^x \frac{u(s)}{(x-s)^{0.1}}\,\mathrm{d}s = \frac{\Gamma(\sqrt{3}+1)}{\Gamma(\sqrt{3}+1.9)}x^{\sqrt{3}+0.9} + x^{\sqrt{3}}.$$

The exact solution is $u(x) = x^{\sqrt{3}}$. Numerical results are listed in Table VI and VII.

To show the efficiency of our method, we plot matrix $\hat{K}_n$ according to our truncation

Table VI: $L^2$ error between the approximate solution and the exact solution for Example 2.5.3.

| $\sigma$ | $n$ | $s(n)$ | $\|u - u_n\|_2$ |
|---|---|---|---|
| 3 | 5 | 96 | $4.071175e-06$ |
| 3 | 6 | 192 | $2.721076e-06$ |

Table VII: Absolute Error between the approximate solution and the exact solution for Example 2.5.3.

| $x$ | Exact Solution | Absolute Error for $n = 5$ | Absolute Error for $n = 6$ |
|---|---|---|---|
| 0.0 | 0.000000 | $0.000000e+00$ | $0.000000e+00$ |
| 0.1 | 0.018533 | $4.900351e-06$ | $4.566543e-06$ |
| 0.2 | 0.061567 | $3.998990e-07$ | $4.326019e-07$ |
| 0.3 | 0.124262 | $2.096073e-06$ | $2.070912e-06$ |
| 0.4 | 0.204525 | $1.785370e-06$ | $1.742832e-06$ |
| 0.5 | 0.301023 | $3.550855e-06$ | $3.147336e-06$ |
| 0.6 | 0.412807 | $2.210742e-06$ | $2.279901e-06$ |
| 0.7 | 0.539140 | $5.206239e-07$ | $5.714249e-07$ |
| 0.8 | 0.679433 | $1.084797e-06$ | $1.168992e-06$ |
| 0.9 | 0.833193 | $4.057441e-09$ | $3.132106e-09$ |
| 1.0 | 1.000000 | $3.244073e-06$ | $3.282330e-06$ |

strategy. Figures 2.5.7 and 2.5.8 shows the distribution of values of $\hat{K}_n$ with $n = 4$ and $n = 5$, respectively. Black grids represent nonzero entries, gray grids represent zero entries corresponding to case 1 while white grids represent entries that were compressed to zeros. Table VIII lists the number of entries we need to calculate with our numerical quadrature rule after compression. $T_n$ means the number of entries corresponding to case 1. $O_n$ means the number of entries that can be truncated. $C_n$ represents the total number of entries that we need to calculate. When $n > 7$, the value of $\frac{s(n)\log s(n)}{C_n} \sim 10$.

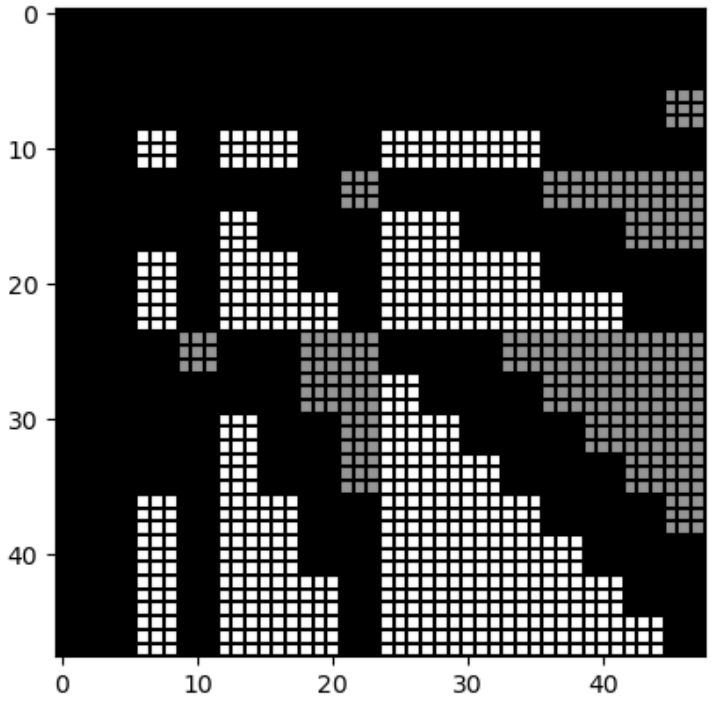



Figure 2.5.7: Value distribution of matrix $\hat{K}_4$.

## 2.6 Conclusion

The Multiscale-Galerkin method has been applied for solving the linear Volterra integral equation of the second kind with a weakly singular kernel. In this approach, we take advan-

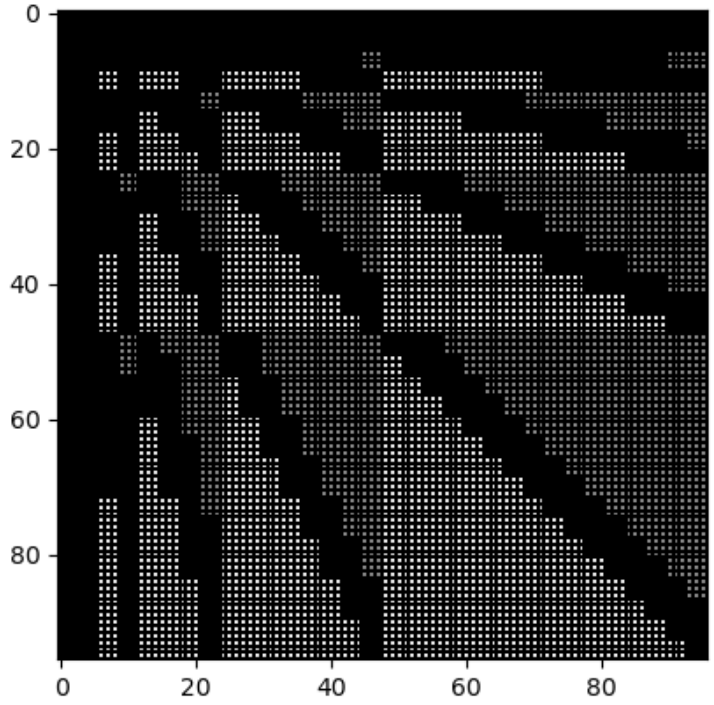



Figure 2.5.8: Value distribution of Matrix $\hat{K}_5$.

tage of the structure of the Volterra integral equation and the shrinking support property of multiscale basis functions to eliminate large many zero entries of the resulting coefficient matrix. For example, let $T_n$ represent the number of zero entries in $\mathbf{K}_n$, then $T_4/s(4)^2$ is 639/2304, $T_5/s(5)^2$ is 3303/9216, $T_6/s(6)^2 = 15255/36864$. When $n$ is getting larger, the ratio $T_n/s(n)^2$ is getting smaller. The numerical results show that the efficiency of our method and its accuracy in approximating exact solutions can compete with the existing methods

Table VIII: Numerical result for truncation strategy

| $n$ | $s(n)$ | $T_n$ | $O_n$ | $C_n$ | $\frac{s(n)\log s(n)}{C_n}$ |
|---|---|---|---|---|---|
| 4 | 48 | 639 | 270 | 186 | 7.5 |
| 5 | 96 | 3303 | 2151 | 3762 | 8.6 |
| 6 | 192 | 15255 | 12114 | 9495 | 9.4 |

for solving Volterra integral equations.

# Chapter 3

# Preliminary Review of Machine Learning

In recent years, deep learning neural networks as a type of machine learning model are getting more and more popular. A wave of interest in applying deep learning neural networks to solving all kinds of mathematical equations emerged. Inspired by the existing works, we have tried to apply the deep learning neural network model to solve integral equations and to evaluate special functions. In this chapter, we shall give a brief review of basic machine learning concepts which will be used in the following chapters of this thesis. For details of machine learning concepts, one can refer to [20] [21] [22].

Most tasks of a deep learning algorithm consist of learning a function that maps an input vector to an output vector based on an optimization algorithm. The main components of a deep learning algorithm include a dataset, a feed-forward neural network model, a cost function, and an optimization algorithm. We shall review these components in the following sections.

## 3.1 Feed-forward Deep Neural Network

We begin with the description of a feed-forward deep neural network model. The goal of a feed-forward neural network is to approximate some function $f(X)$ given input $X$ by defining a mapping $f^*(X; w, b)$ and training the values of the parameters $w, b$ that can lead to best function approximation in some sense. A feed-forward neural network model is usually associated with a directed acyclic chain structure which describes how functions are composed from one side of the chain to the other side. To be more specific, the output of the neural network model $\hat{y} = f^*(X; w, b)$ is obtained by applying affine transformation $z = wX + b$ accompanied with an activation $a(z)$ function on each layer, which results in a chain of composite functions $a_L \circ a_{L-1} \circ \cdots \circ a_1$. In this case, the function $a_i$ represents $i$th layer of the network. The overall length of the chain is called the depth of the model by [20]. The name "deep learning" arose from this terminology.

### 3.1.1 Deep Neural Network Design

Consider a feed-forward neural network with input $X_0$, output $X_L$, and $L+1$ layers. At each layer $\ell$ for $\ell = 1, \ldots, L+1$, we have

- $X_{\ell-1}$ input nodes, with $X_{\ell-1}$ a column vector, $X_{\ell-1} \in \mathbb{R}^{N_{\ell-1} \times 1}$
- $W_\ell$ weights, with $W_\ell$ a matrix, $W_\ell \in \mathbb{R}^{N_{\ell-1} \times N_\ell}$
- $b_\ell$ biases, with $b_\ell$ a column vector, $b_\ell \in \mathbb{R}^{N_\ell \times 1}$

Let us call $z_\ell = W_\ell X_{\ell-1} + b_\ell$ the input to the $\ell$th layer, and $X_\ell = a_\ell(z_\ell)$ the output of $\ell$th layer, where $a_\ell$ is an activation function. As a result, the neural network model to approximate the unknown function at layer $\ell$ is given as

$$X_\ell = a_\ell(W_\ell X_{\ell-1} + b_\ell),$$

which would recursively contribute to the network output $\hat{y} = X_L = a_L(W_L X_{L-1} + b_L)$.

Fig. 3.1.1 shows a four layers feed-forward neural network with two hidden layers, where $X_0 = [x_0, x_1, x_2, x_3], X_1 = [h_1^{(1)}, h_2^{(1)}, h_3^{(1)}, h_4^{(1)}]$ and $X_2 = [h_1^{(2)}, h_2^{(2)}, h_3^{(2)}]$.

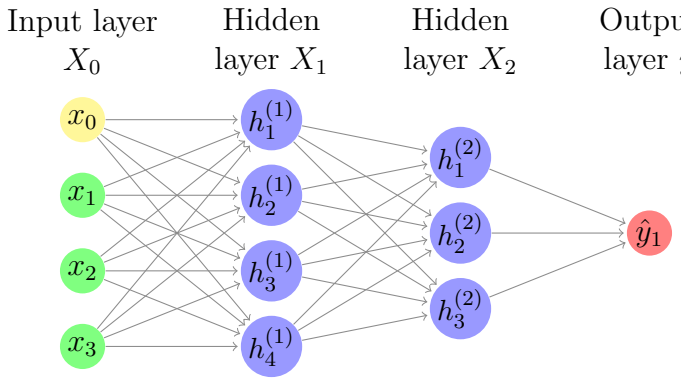



Figure 3.1.1: An example of a feed-forward neural network.

For simplicity, in the rest of the paper, we shall use $w, b$ to represent all the parameters $W_\ell$ and $b_\ell$, for $\ell = 1, 2, \ldots, L$, respectively, and use $\theta$ to represent $(w, b)$.

## 3.2 Activation Function

From the above section, we can see that activation functions are applied to the output data of the previous layer before passing them to the next layer. The purpose of the activation function is to introduce non-linearity into the output of a neuron. A network comprised of only linear functions is easy to train, but cannot learn complex functions. Nonlinear activation functions allow neural networks to model complex non-linear functions, hence the neurons have the ability to learn more complex structures in the data. Commonly used non-linear activation functions are sigmoid, tanh, and Relu (See [23], [24]), which are defined as follows, respectively,

1. Sigmoid: $a(t) = \frac{1}{1+e^{-t}}$, Fig. 3.2.2.

2. Hyperbolic tangent: $a(t) = \frac{e^t - e^{-t}}{e^t + e^{-t}}$, Fig. 3.2.3.

3. Rectified Linear: $a(t) = \max\{0, t\}$, Fig. 3.2.4.

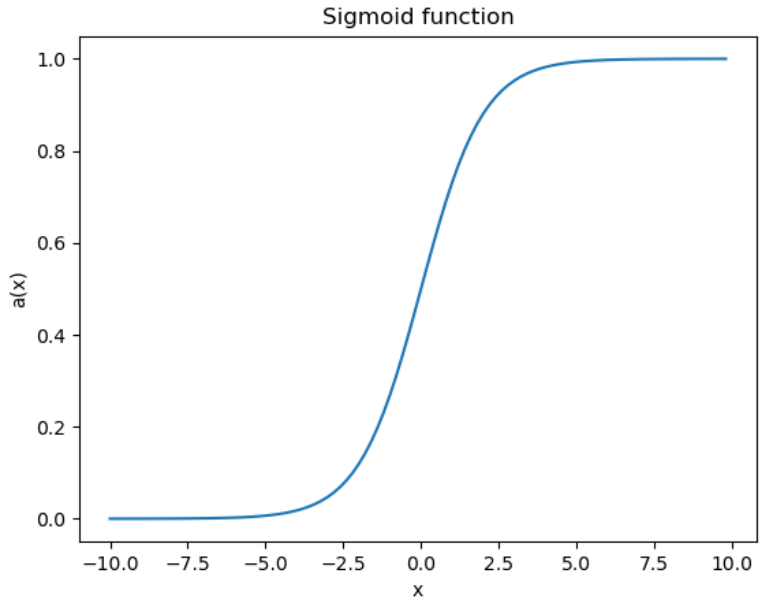



Figure 3.2.2: Graph of the sigmoid function

The activation functions discussed above each possess distinct advantages and limitations. Sigmoid and tanh, being differentiable everywhere, are crucial for the effective back-propagation required in training deep neural networks. In addition, their smooth and monotonic shape makes optimization easier and helps avoid getting stuck in local minima. However, both functions suffer from a general problem, known as "saturation", wherein the output almost remains constant as the input grows too large or too small, resulting in near-zero gradients. This, in turn, negatively affects the learning process and the model's weight adjustment. Another significant limitation of sigmoid and tanh is the "vanishing gradients" problem. "Vanishing gradients" cause the gradients to "vanish" during back-propagation, which makes it difficult to know the direction for parameter adjustment to optimize the model (See [20]). On the other hand, Relu doesn't suffer from the "vanishing gradients" problem and is simple and computationally efficient. Nonetheless, the lack of differentiability at zero can cause challenges in optimization algorithms like gradient descent. Furthermore, Relu's tendency to discard negative values can hamper the model's ability to fit the data properly.

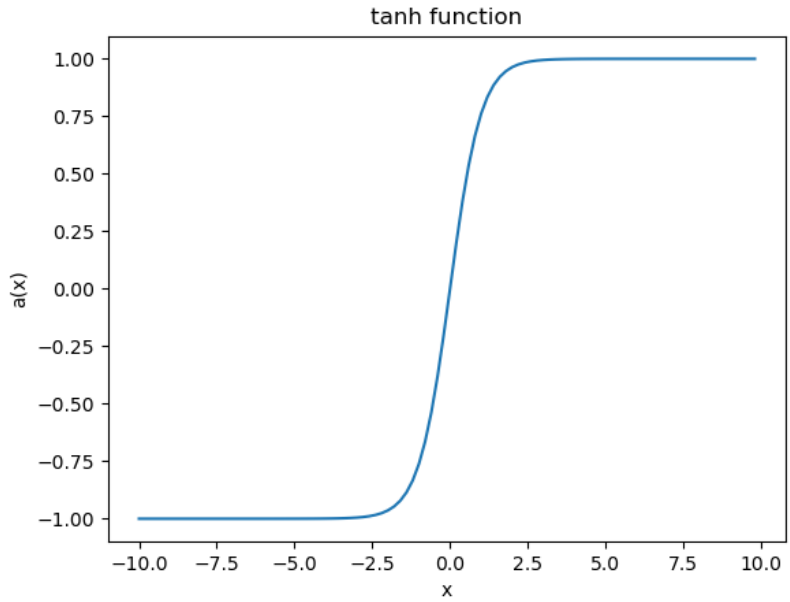



Figure 3.2.3: Graph of the Hyperbolic tangent function

## 3.3 Cost Function

In order to evaluate the performance of a designed deep neural network model, we also need to choose an appropriate quantitative measure. In the field of machine learning, people use the cost function, also known as the objective function, to measure "how wrong the model is", i.e. the difference between the predicted output of a model and the true output. Most neural networks use the negative log-likelihood as their cost function (See, for example, [20] [21] [22]), which is given by

$$\mathcal{C}(\theta) = -E_{x,y\sim p_{\text{data}}} \log p_{\text{model}}(y|x,\theta), \tag{3.1}$$

where $p_{\text{data}}$ denotes probability distribution of the data while $p_{\text{model}}$ denotes the predicted probability distribution by the learning algorithm and $p_{\text{model}}(y|x,\theta)$ denotes the corresponding conditional probability function of $y$ given $x$. The specific form of the cost function $\mathcal{C}$ depends on how the $p_{\text{model}}(y|x,\theta)$ is defined. In particular, if we assume that, for each data point $x$, the predict value $y$ is drawn from a Gaussian distribution with mean $f^*(x,\theta)$ and

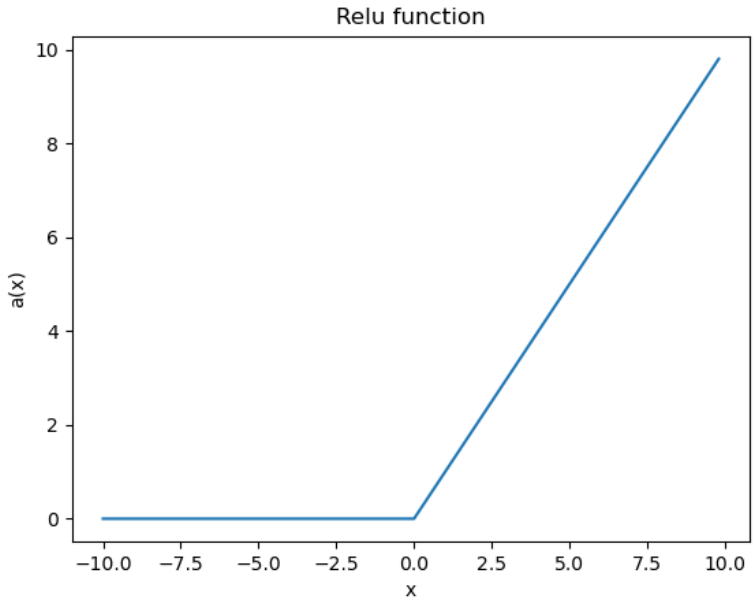



Figure 3.2.4: Graph of the Relu function

variance $\sigma^2$, i.e.

$$p_{\text{model}}(y|x,\theta) := \frac{1}{\sqrt{2\pi\sigma^2}} \exp^{-\frac{(f^*(x,\theta)-y)^2}{2\sigma^2}},$$

then by applying the principle of Maximum likelihood, we recover the mean squared error (MSE) cost

$$\mathcal{C}(\theta) = \frac{1}{2} E_{p_{x,y\sim\text{data}}} \|y - f^*(x,\theta)\|^2. \tag{3.2}$$

Details can be found in section 5.5 in [20] and the references therein. MSE is commonly used for regression models. We adopt MSE as our cost function for neural network models presented in this thesis. Specifically, for a given data set $D := \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$ and a deep learning model with $L$ layers, the MSE cost between training data and the model distribution can be formulated as:

$$\mathcal{C}(\theta) = \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}(a_L(x_i;\theta), y_i) \tag{3.3}$$

where

$$\mathcal{L}(a_L(x_i;\theta), y_i) = (y_i - a_L(x_i;\theta))^2, \quad i = 1, 2 \ldots, n \tag{3.4}$$

is called the squared loss, which represents the squared error incurred by one single sample.

## 3.4 Regularization

A good machine learning algorithm should perform well not just on the training data, but also on unseen data. In order to improve the performance of the model on test data, many strategies have been explored. These strategies are known collectively as regularization. In [20], the author defines regularization as "any modification we make to a learning algorithm that is intended to reduce its generalization error but not its training error." Simply speaking, regularization is a technique used to prevent "overfitting". One commonly used regularization strategy in the regression model is adding extra terms in the objective function, which discourages the model from assigning too large values to any parameter, such as $L_1$ (e.g. Lasso regression) regularization and $L_2$ (e.g. Ridge regression) regularization. Lasso regression (3.5) adds a $L_1$ norm penalty while Ridge regression (3.6) adds a $L_2$ norm penalty to the original objective function, and $\lambda \in (0, \infty)$ is a hyperparameter that weights the relative contribution of norm penalty term in the objective function, in other words, it determines the trade-off between model complexity and overfitting.

In general, by adding a regularization term, the model is encouraged to have small coefficients, which led to a simpler, more interpretable model with improved generalization performance. In particular, in this thesis, we choose $L_2$ regularization considering it's differentiable everywhere. Lasso regression is not differentiable at zero, but it usually results in sparse solutions, for that reason, it can be used for feature selection.

$$\text{Lasso regression}: \ \mathcal{C}_1(a_L(x_i;\theta), y_i) = \frac{1}{n}\sum_{i=1}^{n}(a_L(x_i;\theta) - y_i)^2 + \frac{\lambda}{n}\sum_{j\in L}\|W_j\|_1 \tag{3.5}$$

$$\text{Ridge regression}: \ \mathcal{C}_2(a_L(x_i;\theta), y_i) = \frac{1}{n}\sum_{i=1}^{n}(a_L(x_i;\theta) - y_i)^2 + \frac{\lambda}{n}\sum_{j\in L}\|W_j\|_2^2 \tag{3.6}$$

## 3.5 Back-propagation

When we use a feed-forward neural network, the input provides the initial information that then propagates up to the output layer through hidden layers and finally produces the predicted output. This process is called forward propagation. The back-propagation algorithm (See [25]) is an automatic differentiation algorithm that allows the information from the cost $\mathcal{C}$ to flow backward through the network by computing the gradients $\frac{\partial \mathcal{C}}{\partial w}$ and $\frac{\partial \mathcal{C}}{\partial b}$. Specifically, the error in the predicted output will be propagated to the parameters on each hidden layer in the form of gradients.

## 3.6 Gradient Decent

The objective during training is to minimize the value of the cost function by iteratively adjusting the model's parameters. To this end, iterative optimizers are utilized for training the neural networks. Gradient descent is an optimization algorithm used to minimize a function by iteratively updating its parameter in the direction of the steepest descent. This is achieved by computing the gradient of the function with respect to its parameters and updating them accordingly until the optimization algorithm converges to a minimum. Mathematically, for a given differentiable function $f(w)$, $w \in R^n$, according to Taylor expansion, with sufficient small $\|w_{k+1} - w_k\|_2$, we have

$$f(w_{k+1}) \approx f(w_k) + (w_{k+1} - w_k)^T \nabla f(w_k). \tag{3.7}$$

For the purpose of minimizing $f(w)$, we expect that for each iteration,

$$f(w_{k+1}) - f(w_k) < 0, \quad k \in N,$$

thus, we update $w$ through

$$w_{k+1} = w_k - \alpha \nabla f(w_k), \tag{3.8}$$

where $\alpha > 0$ is called the learning rate.

In particular, to achieve the goal of minimizing the cost function of the neural network $\mathcal{C}$, weights and bias can be updated as follows according to (3.8),

$$W_\ell \leftarrow W_\ell - \alpha \frac{\partial \mathcal{C}}{\partial W_\ell} \tag{3.9}$$

$$b_\ell \leftarrow b_\ell - \alpha \frac{\partial \mathcal{C}}{\partial b_\ell} \tag{3.10}$$

for $\ell = 1, \ldots, L$.

### 3.6.1 Batch Gradient Decent

Gradient descent optimization algorithms that use the entire training set are called batch gradient methods. In batch gradient descent, the gradients of all training samples are computed, averaged, and then used to update the model's parameters. The approach involves processing the entire training set during each iteration or epoch. Batch gradient descent is a reliable and efficient optimization algorithm for small datasets. However, the computational cost of processing large datasets becomes excessive. To tackle this problem, people have proposed using statistical estimation of the gradient instead of the mean gradient of the entire dataset. Stochastic Gradient Descent is one of the optimization algorithms based on this idea.

### 3.6.2 Stochastic Gradient Descent

Gradient descent optimization algorithms that use only a single example at a time are called stochastic methods. Stochastic gradient descent (SGD) (See [26]) or minibatch SGD is probably the most used optimization algorithm for machine learning in general. In minibatch

stochastic gradient descent (SGD), we consider using k samples in each minibatch to take a single step. We can obtain an unbiased estimate of the gradient by taking the average gradient on a minibatch of k samples drawn independently and identically from the data-generating distribution. The following steps are taken in one epoch for minibatch SGD:

1. Take $k$ samples randomly.
2. Feed those samples to the neural network.
3. Calculate the average of the gradients of $k$ samples.
4. Use the gradient we calculated in the last step to update the weights.
5. Repeat the above steps for all the samples in the training data set.

Mathematically, the above procedure can be formulated as follows:

$$W_\ell \leftarrow W_\ell - \alpha \frac{1}{k} \sum_{i=1}^{k} \frac{\partial \mathcal{L}}{\partial W_\ell}, \tag{3.11}$$

$$b_\ell \leftarrow b_\ell - \alpha \frac{1}{k} \sum_{i=1}^{k} \frac{\partial \mathcal{L}}{\partial b_\ell}. \tag{3.12}$$

## 3.7 Momentum

The stochastic gradient descent (SGD) algorithm is widely used in machine learning, but its efficiency can be limited due to noisy gradients, especially for objective functions with poorly conditioned Hessian matrices. To address this issue, Polyak (1987) (See [27]) introduced the concept of momentum, which accelerates the learning of the algorithm by overcoming the oscillations in gradients. In comparison with the SGD, SGD with momentum introduces a new variable $v$ (velocity), which is the exponentially weighted moving average of past gradients. The idea behind "momentum" can be analogous to particle motion in

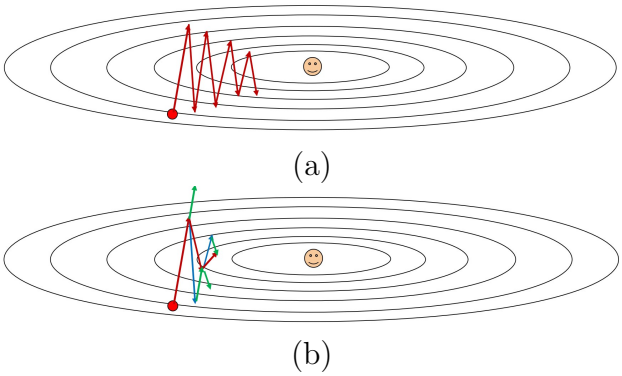
physics. Instead of just considering the current gradient to update $\theta = (w, b)$, momentum also factors in previous gradients. The process can be written as follows:

$$v_0 = 0 \tag{3.13}$$

$$v_n = \beta v_{n-1} + (1 - \beta)\nabla_\theta \mathcal{C}(X_L(x_i; \theta), y_i) \tag{3.14}$$

$$\theta \leftarrow \theta - \alpha v_n. \tag{3.15}$$

Here we use subscript $n$ to represent $n$th iteration. $\beta \in (0, 1)$ is called the momentum constant, it determines how fast the influence of the previous gradient decay. The effects of SGD and SGD with Momentum on a cost function that has elongated contour lines are illustrated in Figure 3.7.5 [28].



(a)

(b)

Figure 3.7.5: (a) SGD update: The red path indicates the learning rule followed by stochastic gradient decent algorithm; (b) SGD with Momentum: The red path depicts the path followed by the momentum learning rule, while the green line and blue indicate the current gradient decent direction and previously accumulated gradient decent direction, respectively.

## 3.8 Adaptive Learning Rate Algorithm

The previous approach uses the same fixed learning rate for all weights updates. But in practice, it is necessary to change the learning rate gradually. Since large learning rate for weights that have steep gradient may cause over-correction, while a small learning rate for

weights that has gentle gradient leads to converging too slowly. In the face of this, it's natural to propose the idea that we should change the learning rate over time. Well-known adaptive learning rate algorithms are AdaGrad, RMSProp, and Adam. For AdaGrad, this algorithm individually adapts the learning rates of all model parameters by scaling them inversely proportional to the square root of the historical square values of the gradient (See [29]), it works well for convex function. But if the cost function is not convex, AdaGrad shrinks the learning rate according to the entire history of the gradient and may have made the learning rate too small before arriving at the local minima (See [20]). RMSProp (See [30]) modifies AdaGrad by replacing the sum of previously squared gradients with an exponentially weighted moving average. Briefly speaking, RMSProp discards extreme gradients from the past that makes it converge rapidly whenever it finds a local convex bowl. Adam (See [31] and Algorithm 1 below) can be understood as a modified combination of momentum and RMSProp, it possesses the strengths of both methods and also utilizes initialization bias correction terms to account for their initialization at the origin. It's one of the states of art optimizers that consistently delivers exceptional performance. Therefore, we have selected the Adam optimization algorithm for the purpose of minimizing in this thesis.

**Algorithm 1** Adam Algorithm [31]

**Input** Stepsize: $\alpha$, Exponential decay rate for the moment estimate:$\beta_1, \beta_2 \in [0, 1)$. Initial parameter vector $\theta_0$, first moment vector $m_0 = 0$, second moment vector $v_0 = 0$, iteration time: $n = 0$.

**While** $\theta_n$ not converge do:

Sample a minibatch of $k$ examples $\{(x_1, y_1), \ldots, (x_k, y_k)\}$ from the training set.

Compute gradient at $n$th iteration: $g_n \leftarrow \nabla_\theta \sum_{i=1}^{k} \mathcal{C}_n(a_L(x_i; \theta_{n-1}), y_i))$

$n \leftarrow n + 1$

Update biased first moment estimate: $m_n = \beta_1 m_{n-1} + (1 - b_1) g_n$

Update biased second moment estimate: $v_n = \beta_2 v_{n-1} + (1 - \beta_2) g_n^2$

Compute bias-corrected first moment estimate: $\hat{m}_n \leftarrow m_n / (1 - \beta_1^n)$

Compute bias-corrected second moment estimate: $\hat{v}_n \leftarrow v_n / (1 - \beta_2^n)$

Update parameter: $\theta_n \leftarrow \theta_{n-1} - \alpha \hat{m}_n / (\sqrt{\hat{v}_n} + \epsilon)$

In the above algorithm, $\epsilon$ is a small constant term that is used to prevent division by zero. By default, $\epsilon = 10 - 8$, $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.99$.

## 3.9 Xavier Initialization

Deep learning models are a form of iterative algorithm and thus require the initialization of the parameters to commence iterations. The initial points play a crucial role in determining the convergence of the algorithm, and when the algorithm does converge, the rate of convergence of the algorithm is also contingent upon the initial points selected. The aim of a carefully designed weight initialization is to enhance the convergence rate by preventing layer activation outputs from exploding or vanishing during the course of a forward pass through a deep neural network. If either situation mentioned above occurs, loss gradients will either be too large or too small to flow backward.

As an illustration, when the sigmoid function is utilized as an activation function, if the weights are excessively small, the variance of the input signal declines as it proceeds through each layer in the network, eventually leading to the input value falling within the range where the sigmoid function behaves in a nearly linear fashion, consequently resulting in the loss of the model's nonlinearity. On the other hand, if the weights are excessively large, the variance of input data tends to surge rapidly, attaining a value within the range where the sigmoid function is almost flat. This, in turn, causes the gradient to be zero, and learning ceases prior to reaching the minima. To address this issue, Xavier Glorot and Yoshua proposed the Xavier initialization (See [32]). Xavier initialization involves selecting the initial weights from a normal distribution with a mean of zeros and a variance that is based on the number of inputs and outputs of the layer:

$$W \sim N[0, \frac{\sqrt{2}}{\sqrt{n_1 + n_2}}] \tag{3.16}$$

where $n_1$ is the number of inputs to the nodes and $n_2$ is the number of outputs from the layer. The main idea behind Xavier initialization is to initialize the weights of a neural network layer in such a way that the variance of the outputs of the layer is roughly equal to the variance of its inputs.

In the rest of this section, we use network Fig. 3.1.1 as an example to show how (3.16) could maintain the variance of activations and back-propagated gradients all the way up or down the layers of a network.

To prevent the gradients of the network's activations from vanishing or exploding, we shall stick to two rules. Firstly, the mean of the activation outputs should be zero. Secondly, the variance of the activation outputs should stay the same across every layer. For simplicity, let's assume that the input $X_0$ and weights $W_1$ are drawn from independent identically distributed Gaussian distributions, respectively. Moreover, for illustration purposes, we also assume that the network 3.1.1 only consists of a chain of matrix multiplications, with no nonlinearities. Then the input value $h_1^{(1)}$ of the first hidden layer can be formulated as

$$h^{(1)} = \sum_{i=0}^{3} w_{1i} x_i, \tag{3.17}$$

where $w_{1i}, i = 0, 1, 2, 3$ represent the weights between the input layer and the first neuron of the first hidden layer. Taking variance of both sides of (3.17), and by the fact that the input $X_0$ and weights $W_1$ are iid, we have

$$\mathrm{Var}(h_1^{(1)}) := \sum_{i=0}^{3} \mathrm{Var}(w_{1i} x_i) = \sum_{i=0}^{3} [E(x_i)^2 \mathrm{Var}(w_{1i}) + E(w_{1i})^2 \mathrm{Var}(x_i) + \mathrm{Var}(w_{1i})\mathrm{Var}(x_i)], \tag{3.18}$$

since the input $X_0$ and weights $W_1$ all have mean zero, (3.18) can further be reduced to

$$\mathrm{Var}(h_1^{(1)}) := \sum_{i=0}^{3} \mathrm{Var}(w_{1i})\mathrm{Var}(x_i) = 4\mathrm{Var}(w_{1i})\mathrm{Var}(x_i). \tag{3.19}$$

If we expect the variance of output $h_1^{(1)}$ to be the same as variance of input $X_0$, we need $\mathrm{Var}(w_{1i}) = \frac{1}{4}$. Generally, to ensure that the variance of output matches that of input, it is necessary to set $\mathrm{Var}(w_{1i}) = \frac{1}{N}$, where $N$ is the number of neurons on the corresponding output layer. In other words, the weights of each layer must be randomly sampled from a normal distribution $N(0, \frac{1}{N})$. This applies to both forward propagation and backward

propagation.

Xavier initialization is a widely used method for initializing weights in neural networks, and it has been shown to be effective in a variety of applications.

## 3.10 Summary

Deep learning is a type of machine learning. It can simulate models that involve a greater amount of compositions, so it has more potential than other machine learning models in solving mathematical problems. The next two chapters will discuss the application of deep learning in solving integral equations.

# Chapter 4

# Solving Fredholm Integral Equations of the Second Kind using a Neural Network Model

## 4.1 Introduction

The universal approximation theorem [33] [34] states that a feed-forward network with a linear output layer and at least one hidden layer with any "squashing" activation function can approximate any Borel measurable function from one finite-dimensional space to another with any desired accuracy, provided that the network is given enough hidden units [32]. This theorem provides a theoretical foundation for solving various kinds of mathematical equations with deep learning models. Since solutions of Fredholm integral equations (FIEs) lies in the space where functions are Borel measurable, provided that their solution exists, it should be feasible to use deep learning neural network model to handle Fredhlom integral equations.

In this chapter, we present our approach for solving Fredholm integral equations of the second kind via a neural network model. A brief review of the Fredholm integral equation is provided in section 2, while section 3 presents a formulation of the neural network method

for second-kind FIEs. In section 4, numerical experiments are conducted to demonstrate the performance of the proposed method.

## 4.2 Fredholm Integral Equation of the Second Kind

In this section, we give a brief review of Fredholm integral equations of the second kind. Unlike Volterra integral equations which have variable integral limits, a Fredholm equation is an integral equation in which the kernel function term has constants as integration limits. But similar to Volterra integral equations, Fredholm integral equations also can be divided into the first kind and the second kind. A Fredholm integral equation of the first kind is an integral equation of the form

$$f(t) = \int_0^1 k(t,s)u(s)\mathrm{d}s, \quad t \in [0,1], \tag{4.1}$$

while a Fredholm integral equation of the second kind can be expressed as

$$f(x) = u(x) + \mu \int_0^1 k(x,s)u(s)\mathrm{d}s, \quad x \in [0,1], \quad \mu \neq 0, \tag{4.2}$$

where the Fredholm integral operator is defined by

$$(\mathcal{K}u)(x) = \int_0^1 k(x,s)u(s)\,\mathrm{d}s.$$

We will focus on Fredholm integral equations of the second kind due to the fact that the first kind equations can be converted to the second kind through regularization. However, for the purpose of evaluating the performance of a deep learning model in solving integral equations, we have chosen to begin with simpler cases. Specifically, we have opted to work with continuous functions for the kernel $k(t,s) : [0,1] \to [0,1]$ and $f(t) : [0,1] \to R$. The existence of the solution of Fredholm integral equations of the second kind with a continuous

kernel is discussed in [1].

## 4.3 Neural Network Formulations

Recall that the Fredholm integral equation of the second kind takes the form:

$$f(x) = u(x) + \mu \mathcal{K} u(x), \quad x \in [0, 1], \tag{4.3}$$

with the integral operator defined by

$$\mathcal{K} u(x) = \int_0^1 k(x, s) u(s) \, \mathrm{d}s.$$

A neural network model for solving (4.3) is to find an approximate solution $u_{w,b}(x)$ such that

$$\mathcal{C}_{w,b} = \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}_{w,b}(x_i) + \frac{\lambda}{n} \|w\|_2^2 \tag{4.4}$$

with

$$\mathcal{L}_{w,b}(x_i) = (\mu \mathcal{K} u_{w,b}(x_i) + u_{w,b}(x_i) - f(x_i))^2,$$

can be minimized. $\{x_1, x_2, \ldots, x_n\}$ is a set of distinct points in $[0, 1]$.

Specifically, let's consider a neural network model comprises of one hidden layer with $m$ hidden units and one linear output unit, then $u_{w,b}$ can be formulated as

$$u_{w,b}(x) = W_2(a_1(W_1 x + b_1)) + b_2. \tag{4.5}$$

The above process is illustrated in Fig. 4.3.1. The weights between the input layer and the hidden layer of the network are $W_1^T := [w_{11}, w_{12}, \ldots, w_{1m}]$, the activation function on neurons of the hidden layer is $a_1$, and the weights between the hidden layer and output layer are $W_2^T := [w_{21}, w_{22}, \ldots, w_{2m}]$, additionally, $z_i = a_1(W_1 x_i + b_1)$.

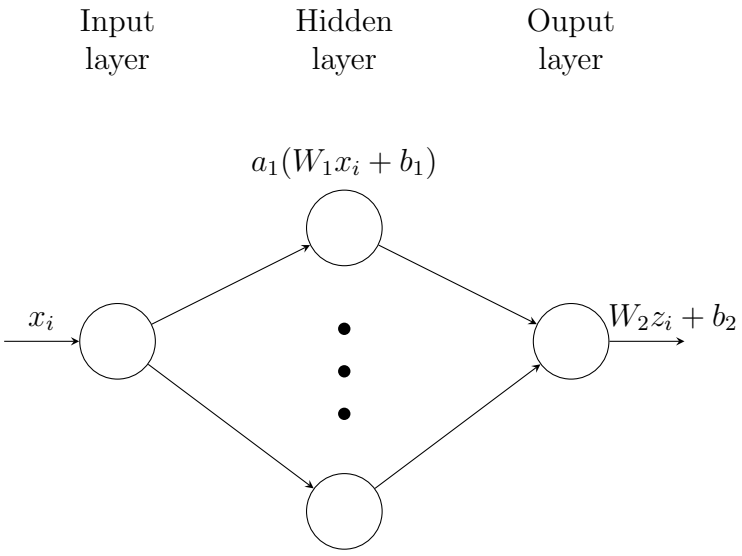



Figure 4.3.1: Diagram of NN-collocation model.

By taking partial derivative of $\mathcal{C}_{w,b}$ with respect to $w_{lr}, b_l$, respectively, we have

$$\frac{\partial \mathcal{C}_{w,b}}{\partial w_{lr}} = \frac{2}{n}\sum_{j\in Z_n}(U_j + \mu K_j - f_j)(\frac{\partial U_j}{\partial w_{lr}} + \mu\frac{\partial K_j}{\partial w_{lr}}) + \frac{2\lambda}{n}w_{lr},$$

and

$$\frac{\partial \mathcal{C}_{w,b}}{\partial b_l} = \frac{2}{n}\sum_{j\in Z_n}(U_j + \mu K_j - f_j)(\frac{\partial U_j}{\partial b_l} + \mu\frac{\partial K_j}{\partial b_l}),$$

where

$$U_j = u_{w,b}(x_j),$$

$$K_j = \int_0^1 k(x_j, s)u_{w,b}(s)\,ds, \tag{4.6}$$

$$\frac{\partial K_j}{\partial w_{lr}} = \int_0^1 k(x_j, s)\frac{\partial u_{w,b}(x_j)}{\partial w_{lr}}\,ds, \tag{4.7}$$

$$\frac{\partial K_j}{\partial b_l} = \int_0^1 k(x_j, s)\frac{\partial u_{w,b}(x_j)}{\partial b_l} \tag{4.8}$$

$$f_j = f(x_j)$$

for $j \in Z_n, l = 1, 2$ and $r = 1, 2, \ldots, m$. The most expensive computational cost are from calculating $K_j, \frac{\partial K_j}{\partial w_{lr}}$ and $\frac{\partial K_j}{\partial b_l}$, where the integrands involve parameter $w, b$. We have to

recalculate the integrals whenever we get $w, b$ updated. In order to save computational cost, instead of using the entire training data set to calculate the gradient, we choose mini-batch Stochastic gradient descent (SGD) with sample size $k$ to estimate $\frac{\partial C_{w,b}}{\partial w_{lr}}$ and $\frac{\partial C_{w,b}}{\partial b_l}$. The whole procedure is summarized in Algorithm 2.

**Algorithm 2** Our Algorithm

1: **Input** regularization parameter $\lambda$, number of training data $n$, training set $S_n := \{x_i : i \in Z_n\}$ and mini-batch size $k$. Initialize the weights $\{w_j : j \in Z_m\}$ by Xavier initialization [32].
2: **Compute** $U_j, K_j, \frac{\partial U_j}{\partial w_{lr}}, \frac{\partial U_j}{\partial b_l}, \frac{\partial K_j}{\partial w_{lr}}, \frac{\partial K_j}{\partial b_l}$ and then $\frac{\partial C}{\partial w_{lr}}, \frac{\partial C}{\partial b_l}$.
3: **Update** the weights $w_{lr}$ and of $b_l$ with the Adam optimizer [35], until meet a stop criteria
4: **Compute** $u_{w,b}$ via (4.5) with the optimal weights $W_i, i = 1, 2$.

## 4.4 Numerical Integration

In order to calculate $K_j$, $\frac{\partial K_j}{\partial w_{lr}}$ and $\frac{\partial K_j}{\partial b_l}$, we need to develop an efficient numerical method to estimate integrals involved in (4.6), (4.7) and (4.8) with high accuracy. Since kernel $k(x, s)$ is continuous, we choose Gaussian quadrature within tolerance to approximate these integrals.

## 4.5 Results

In this section, we will present three examples of various types of Fredholm integral equations to illustrate the performance of our proposed neural network (NN) model. Our algorithm was implemented in Python using the Numpy library [36] for efficient matrix manipulation. Our neural model comprises one hidden layer with ten hidden units and one linear output unit. Although adding more hidden neurons or training points can potentially improve the model's performance, this approach may not always be viable due to the vanishing gradient problem. In our numerical experiments, we used a training set of $n = 1000$ numbers sampled uniformly at random from the interval $[0, 1]$, while the test set consisted

Table I: Partial iterative results of the loss function for Example 1.

| Iteration | loss | Iteration | loss |
|---|---|---|---|
| 10 | 22.18 | 150 | 0.005 |
| 20 | 3.05 | 200 | 0.0004 |
| 50 | 0.29 | 500 | 0.00032 |
| 100 | 0.025 | 1000 | 0.00025 |

of $n = 1000$ numbers equally spaced from the same interval.

**Example 1.** Consider the following linear FIE of the second kind

$$u(x) - \int_0^1 2e^{x+y}u(y)\,dy = e^x.$$

The equation has $u(x) = \frac{e^x}{2-e^2}$ as its exact solution.

For example 1, the number of training iterations is 1000. The loss function converges to $e-4$, and the generalization MSE converges to $9.4e-5$. Figure 4.5.2 shows the graphs of the exact solution and the neural network's approximate solution, the red curve depicts the graph of the Exact solution while the green curve depicts the graph of the neural network's approximate solution. The convergence of the loss function is shown in Fig. 4.5.3(a), 4.5.3(b) shows the generalization MSE between the exact solution and the approximation solution. Partial iterative results of the loss function showed in Table II.

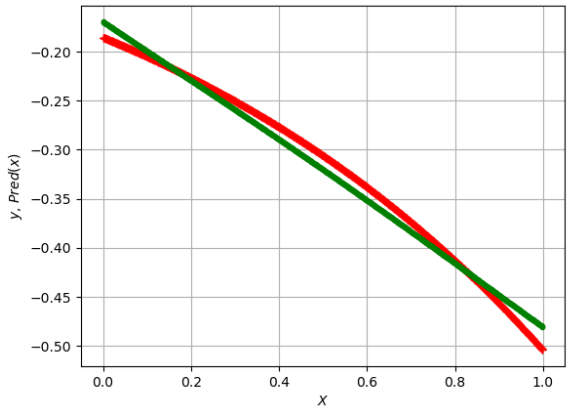
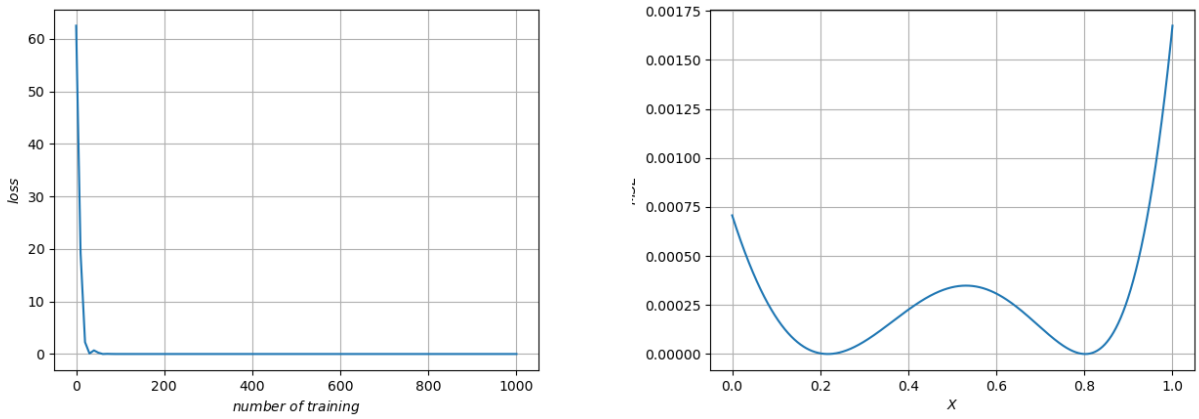



Figure 4.5.2: Example 1: Exact solution vs Neural network approximate solution.




(a) (b)

Figure 4.5.3: Example 1: (a) Convergence of the loss function; (b) MSE between the exact solution and the approximation solution.

Table II: Partial iterative results of the loss function for Example 2.

| Iteration | loss | Iteration | loss |
|---|---|---|---|
| 10 | 0.13 | 200 | 0.0004 |
| 20 | 0.09 | 250 | 1.89e-5 |
| 50 | 0.03 | 400 | 2.9e-6 |
| 100 | 0.01 | 500 | 2.78e-6 |

**Example 2.** Consider the following linear FIE of the second kind

$$u(x) + \int_0^1 x(e^{xy} - 1)u(y)\,dy = e^x - x.$$

The equation has the $u(x) = 1$ as its exact solution.

For Example 2, the number of iterations is 500. The loss function converges to $e-6$, and the generalization MSE converges to $2.7e-6$. Figure 4.5.4 shows the graphs of the exact solution and the neural network approximate solution. The convergence of the loss function is shown in Fig. 4.5.5(a), 4.5.5(b) shows the generalization MSE between the exact solution and the approximate solution. Partial iterative results of the loss function showed in Table II.

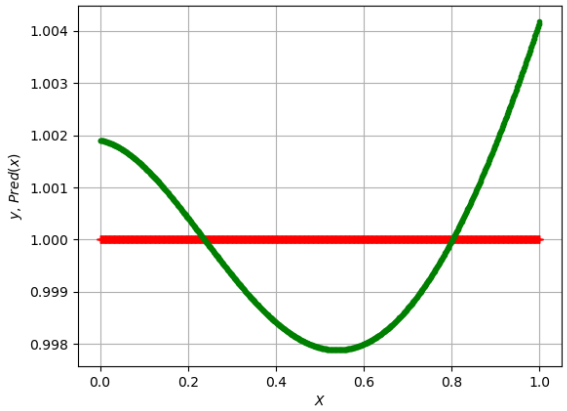



Figure 4.5.4: Example 2: Exact solution vs Neural network approximate solution.

Table III: Partial iterative results of the loss function for Example 3.

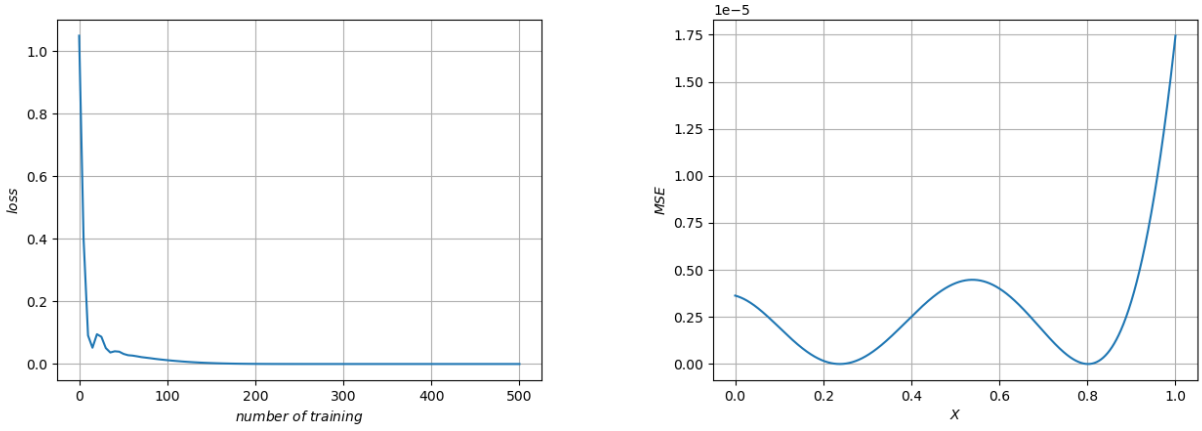
| Iteration | loss | Iteration | loss |
|---|---|---|---|
| 10 | 5.24 | 200 | 0.0056 |
| 20 | 0.44 | 500 | 0.0011 |
| 50 | 0.048 | 1000 | 0.0002 |
| 100 | 0.0083 | 1200 | 0.00013 |

**Example 3.** Consider the following linear FIE of the second kind

$$u(x) - \int_0^1 9xyu(y)\,dy = x^2.$$

The equation has he exact solution $u(x) = x(x - \frac{9}{8})$.

For example 3, the number of iteration is 1000. The loss function converges to $e - 4$, and the generalization MSE converges to $1.3e - 4$. Figure 4.5.6 show the graphs of the exact solution and the neural network approximate solution. The convergence of the loss function is shown in Figure 4.5.7(a), 4.5.7(b) shows the corresponding generalization MSE. Partial iterative results of the loss function showed in Table III.




(a) (b)

Figure 4.5.5: Example 2: (a) Convergence of the loss function; (b) MSE between the exact solution and the approximation solution.

## 4.6 Conclusion

In this chapter, we presented our approach for solving Fredholm integral equations of the second kind via a neural network model. The problem with this approach is its expensive computation cost since the solution is totally attained by learning. To be more specific, each time the weights get updated, one has to re-compute to integral which involves the product of the kernel and the derivative of the approximate solution with respect to the weights. Furthermore, numerical errors are introduced when evaluating the integral, and as the number of iterations increases, the cumulative error also grows. It is also worth noting that even though the universal approximation theorem states that a multi-layer perceptron with a large number of layers can approximate any Borel measurable function, it does not guarantee that we can find the optimal parameter values to correspond to the optimal function. The representative capacity of the learning algorithm plays a significant role in this regard. Although the neural network model exhibits promising results in solving integral equations, the accuracy of this approach is inferior to other existing mathematical methods. These findings reinforce our concerns that applying deep learning models to solve mathematical problems without a comprehensive understanding of the optimization process can

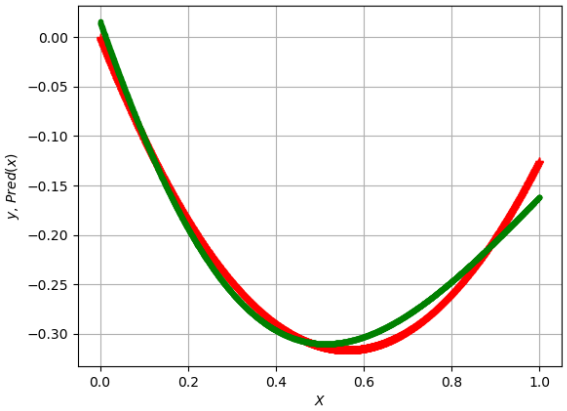
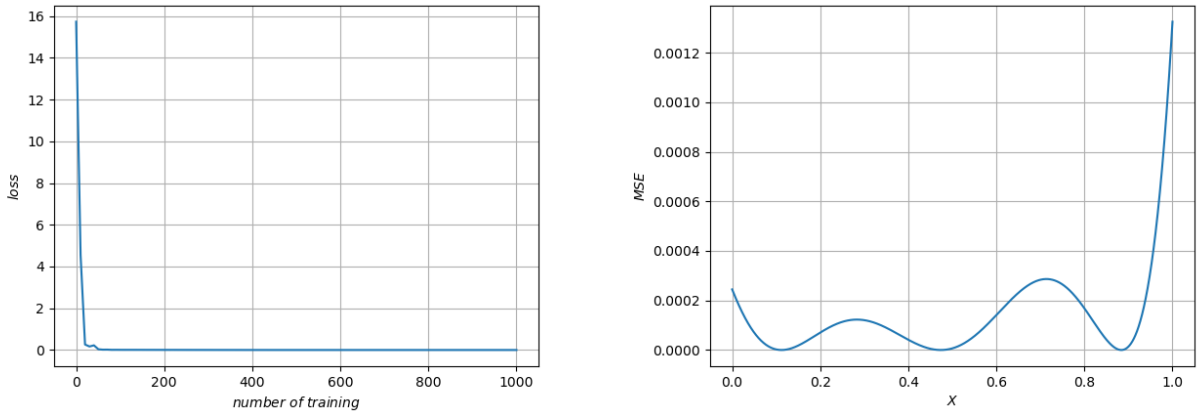



Figure 4.5.6: Example 3: Exact solution vs Neural network approximate solution.

lead to significant uncertainties.

To address the inadequate representative capacity and the significant computational burden of computing the integrals at each iteration, we propose an alternative approach which combines the traditional collocation method with Adam optimization in the next chapter.




(a) (b)

Figure 4.5.7: Example 3: (a) Convergence of the loss function; (b) MSE between the exact solution and the approximation solution.

# Chapter 5

# A Collocation method-based Neural Network model for Solving Fredholm Integral Equations

Our first attempt to employ a neural network for solving Fredholm integral equations, as discussed in chapter 4, doesn't work well due to the limited representational capacity of the model. To be more specific, the model specifies a family of functions in the form of $w_2(e^{-(w_1 x+b)}) + b_2$ that the learning algorithm can choose from. But without a sufficient number of neurons in the hidden layer, functions in this form may not be guaranteed to approximate the exact solution well. In response, we introduce an alternative approach in this chapter based on the collocation method, which improves the initial neural network model by restricting its representational capacity to a polynomial space. That is, we first choose the space of polynomials as the projection space for the collocation method, then approximate the solution of the integral equation by a linear combination of polynomials in that space. This model can also be viewed as a neural network learning model, but with different activation functions in the hidden layer. The coefficients of the linear combination of polynomials function are served as the weights between the hidden layer and the output

layer of the neural network. In the remainder of this chapter, we refer to this model as the NN-Collocation model for brief.

The collocation method is another popular projection technique for solving integral equations that has garnered significant attention in practical applications. One of the key advantages of this method is its simplicity and flexibility, particularly when utilized to incorporate neural network models. It offers an ideal structure for converting integral equations into data-fitting models, without the need for introducing additional integrals as is the case with the Galerkin method. Consequently, the computational requirements associated with collocation are considerably lower.

We organize this chapter into two sections. In section 1, we present the formulation of the neural network (NN)-Collocation method for the second kind FIEs. Numerical experiments are provided in Section 2 to demonstrate the performance of the proposed method.

## 5.1 NN-Collocation Formulations

We consider an integral equation that takes the form of

$$u(x) + \mu \int_0^1 k(x,y)u(y)\,\mathrm{d}y = f(x), \quad x, y \in [0,1], \tag{5.1}$$

where $f \in C[0,1]$, $k(x,y) \in C[0,1] \times C[0,1]$ are known, $u \in C[0,1]$ is unknown and

$$\mathcal{K}(u) = \int_0^1 k(x,y)u(y)\,\mathrm{d}y,$$

is the Fredholm integral operator. With this notation, the above integral equation can be rewritten as

$$(\mathcal{I} + \mu\mathcal{K})u = f. \tag{5.2}$$

where $\mathcal{I}$ is the identity operator.

In chapter 2, we introduced the intuition behind the Galerkin method, which seeks an

approximate solution that makes the residual small via minimizing $\|\mathcal{A}u_n - f\|$, where $\mathcal{A} := \mathcal{I} + \mu\mathcal{K}$. In comparison, the Collocation method makes the residual $r_n := \mathcal{A}u_n - f$ small by making it zero on some finite set of points. Specifically, we choose a finite dimensional space $\mathcal{X}_n$ and a finite set $S \subset [0,1]$ of points, and want to find an approximate solution $u_n \in \mathcal{X}_n$ such that

$$\mathcal{A}u_n(x) - f(x) = 0, \quad \text{for} \quad x \in S.$$

In particular, the traditional collocation method solving integral equation (5.2) seeks vectors $\mathbf{w}_n := [w_j : j \in Z_n]$ such that

$$u_n = \sum_{j \in Z_n} w_j \psi_j \tag{5.3}$$

is the solution of

$$(\mathcal{I} + \mu\mathcal{K})u_n(x_i) = f(x_i), i \in Z_m \tag{5.4}$$

where $S = \{x_i : i \in Z_m\}$ is a finite subset of $[0,1]$, $\{\psi_j\}_{j=1}^n$ are the basis functions of the projection space $\mathcal{X}_n$. The equivalent system of equations form of (5.4) is

$$(\mathbf{E}_n + \mu\mathbf{K}_n)\mathbf{w}_n = \mathbf{f}_n, \tag{5.5}$$

where

$$\mathbf{E}_n = \{E_{ij}, i \in Z_m, j \in Z_n\},$$

$$\mathbf{K}_n = \{K_{ij}, i \in Z_m, j \in Z_n\},$$

$$\mathbf{f}_n = \{f_i, i \in Z_m\},$$

with

$$E_{ij} = \psi_j(x_i), \quad K_{ij} = \int_s k(x_i, y)\psi_j(y)\,\mathrm{d}y, \ f_i = f(x_i).$$

The NN-Collocation method based on this idea tries to minimize the cost function which

is formulated by the means square error with $L^2$ regularization

$$\mathcal{C} = \frac{1}{m}\sum_{i\in Z_m} L_n(\hat{f}(x_i, \mathbf{w}_n), f_i) + \frac{\lambda}{m}\|\mathbf{w}_n\|_2^2 \tag{5.6}$$

with the loss function

$$L_n(\hat{f}(x_i, \mathbf{w}_n), f_i) = (\hat{f}(x_i, \mathbf{w}_n) - f_i)^2, \tag{5.7}$$

where $\hat{f}(x_i, \mathbf{w}_n) := (\mathcal{I} + \mu\mathcal{K})u_n(x_i)$, $\lambda$ is the regularization parameter, $\|\mathbf{w}_n\|_2$ represents the Euclidean norm of $\mathbf{w}_n$. With the notations introduced in (5.5), the loss function (5.7) of the NN-Collocation model can be rewritten as

$$L_n(\hat{f}(\mathbf{x}_i, \mathbf{w}_n), f_i) = \left(\sum_{j\in Z_n}(E_{ij} + \mu K_{ij})w_j - f_i\right)^2. \tag{5.8}$$

In other words, we use $\{(x_i, f_i)\}_{i=0}^m$ as our training data set to implement a learning process for a trail model such that the MSE between the predicted value $(\mathcal{I}+\mu\mathcal{K})u_n$ and exact value of $(\mathcal{I}+\mu\mathcal{K})u$ at the training data set is minimized while emphasizing on maximizing margin to avoid the overfitting problem.

The process described above can be interpreted as a feed-forward network with one hidden layer and a linear output layer. The weights between the input layer and the hidden layer of the network are all set to 1, while the activation function of the $j$th neuron in the hidden layer is $\phi_j(x) = (\mathcal{I} + \mu\mathcal{K})\psi_j(x)$, and the weights between the hidden layer and the output layer are $w_i$s. The network is illustrated in Fig. 5.1.1.

We intend to find $\mathbf{w}_n$ that can minimize $\mathcal{C}$ by using the gradient descent with Adam optimizer. To this end, we need to calculate $\frac{\partial L_n}{\partial w_j}$s. By taking partial derivative with respect to $w_j$ in (5.8), we have

$$\frac{\partial L_n}{\partial w_j} = 2(\sum_{k\in Z_n}(E_{ik} + \mu K_{ik})w_j - f_i)(E_{ij} + \mu K_{ij}). \tag{5.9}$$

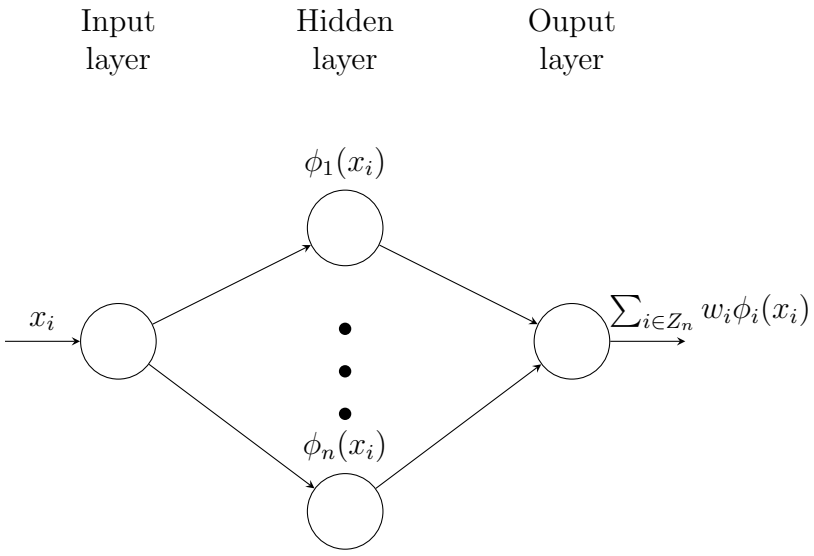



Figure 5.1.1: Diagram of NN-collocation model.

The primary computational cost is attributed to the calculations of $K_{ij}$s, which involve evaluating integrals, but only once. We store the values of $(E_{ij} + \mu K_{ij})$s in a matrix. In this paper, we use the Gauss-quadrature for the evaluation of $K_{ij}$s. The whole procedure is summarized in Algorithm 3.

**Algorithm 3** Our Algorithm

1: **Input** regularization parameter $\lambda$, number of training data $m$, training set $S := \{x_i : i \in Z_m\}$. Initialize the weights $\{w_j : j \in Z_n\}$ by Xavier initialization [32].
2: **Compute** $\phi_j(x_i) := E_{ij} + \mu K_{ij}$, and store the result in matrix $A$ with $A_{ij} = \phi_j(x_i)$, $i \in Z_m, j \in Z_n$.
3: **Compute** $\frac{\partial C}{\partial w_j} = \frac{1}{m}\sum_{i\in Z_m} \frac{\partial L_n}{\partial w_j} + \frac{2\lambda}{m} w_j$ according to chain rule.
4: **Update** the weights $w_j$ with the Adam optimizer.
5: **Compute** the approximation of (5.3) by using the optimal weights $\{w_i : i \in Z_n\}$.

## 5.2 Results

For the purpose of comparison, we use the same numerical examples listed in chapter 4 to demonstrate the NN-Collocation model's performance. The projection space is polynomial space with shifted Legendre polynomials on interval $[0, 1]$ of degree up to $n$ as bases. we utilized a training set of $m = 2000$ numbers sampled uniformly at random from the interval

$[0, 1]$. Testing sets consisting of 2000 evenly spaced values on the interval $[0, 1]$.

**Example 1.** Consider the following linear FIE of the second kind

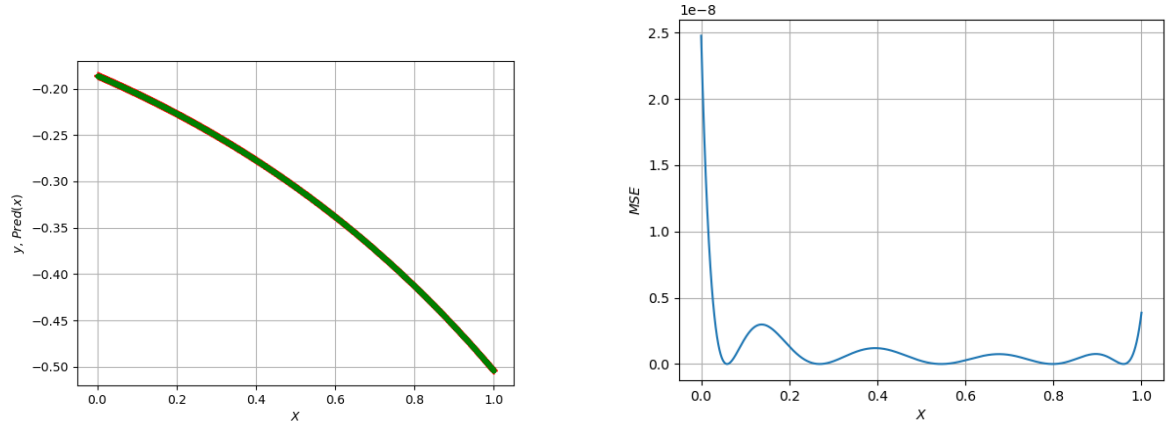
$$u(x) - \int_0^1 2e^{x+y} u(y)\, \mathrm{d}y = e^x,$$

with the exact solution $u(x) = \frac{e^x}{2-e^2}$.

For $n = 6$, the generalization MSE is $e - 09$ after 20000 iterations, and the associated optimal weights are

$$\begin{aligned} \mathbf{w}_n &= [-0.23858496, -0.15568655, -0.11762842, \\ &\quad 0.02070347, -0.0158611, 0.00271201]. \end{aligned}$$

Figure 5.2.2(a) shows the graph of the exact solution and the approximate solution, the red curve depicts the graph of the exact solution while the green curve depicts the graph of the approximate solution, and Figure 5.2.2(b) shows the generalization MSE. The numerical results can be found in Table I.




(a) (b)

Figure 5.2.2: Example 1: (a) Exact solution vs NN-Collocation approximate solution with n=6; (b) MSE between the exact solution and the approximate solution.

Table I: The MSE error for Example 1.

| n | MSE on Training set | MSE on Testing set |
|---|---|---|
| 3 | 1.09689663e-06 | 9.59122925e-07 |
| 4 | 1.27886941e-07 | 1.12720023e-07 |
| 5 | 1.8071884e-08 | 2.2415795e-08 |
| 6 | 1.0746993e-09 | 2.98199195e-09 |
| 7 | 1.72614602e-09 | 1.53534553e-09 |

Table II: The MSE error for Example 2.

| n | MSE on Training set | MSE on Testing set |
|---|---|---|
| 3 | 0.00474284 | 0.00475947 |
| 4 | 7.23014255e-05 | 7.26806487e-05 |
| 5 | 8.22319408e-07 | 6.70043825e-07 |
| 6 | 8.39516517e-08 | 1.02247121e-07 |
| 7 | 2.89689923e-11 | 2.40565254e-11 |

**Example 2.** Consider the following linear FIE of the second kind

$$u(x) + \frac{1}{3}\int_0^1 e^{2x-\frac{5y}{3}} u(y)\, dy = e^{2x+\frac{1}{3}},$$

with the exact solution $u(x) = e^{2x}$.

For $n = 7$, the generalization MSE is around $e - 11$ after 30000 iterations, and the corresponding optimal weights are

$$\begin{aligned} \mathbf{w}_n &= [1.87982320, 2.75563720, 1.95524998, 5.05200551e-01, \\ &\quad 2.75259395e-01, 5.81048575e-04, 1.72820486e-02]. \end{aligned}$$

Figure 5.2.4(a) shows the graph of the exact solution and approximate solution, and Figure 5.2.4(b) shows the generalization MSE. The numerical results can be found in Table II.

**Example 3.** Consider the following linear FIE of the second kind

$$u(x) + \int_0^1 x(e^{xy} - 1)u(y)\, \mathrm{d}y = e^x - x,$$

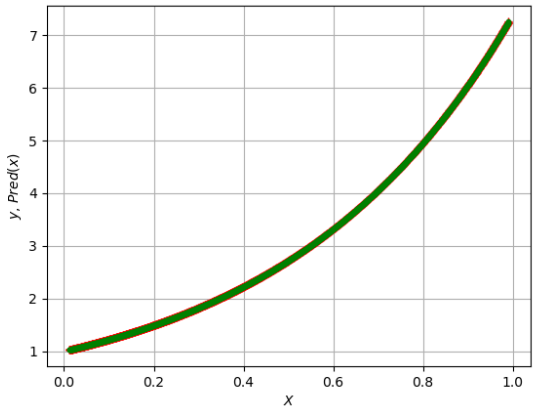



Figure 5.2.3: Example 2: Exact solution vs NN-Collocation approximate solution with n=6;

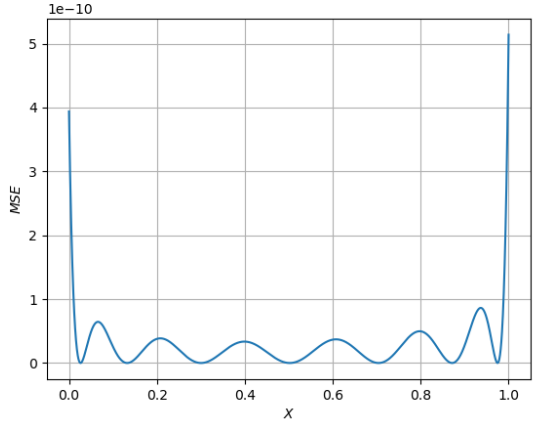



Figure 5.2.4: Example 2: MSE between the exact solution and the approximation solution.

with the exact solution $u(x) = 1$.

For $n = 4$, the generalization MSE is around $e - 15$ after 30000 iteration with optimal weights

$$\begin{aligned} \mathbf{w}_n &= [9.99998676e-01, 2.92296511e-06, \\ &\quad -2.37639142e-06, 9.37088253e-07]. \end{aligned}$$

And order of accuracy around $e-16$ for $n=3$ after 50000 iteration with optimal weights

$$\mathbf{w}_n = [9.99999927e-01, 1.43643199e-07, -8.92090066e-08]$$

Figure 5.2.6(a) shows the graph of the exact solution and approximate solution, and Figure 5.2.6(b) shows the accuracy measured in MSE. The numerical results can be found in Table III.

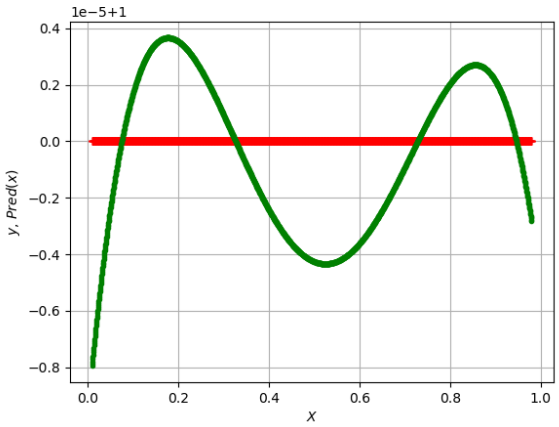



Figure 5.2.5: Example 3: Exact solution vs NN-Collocation approximate solution with n=6

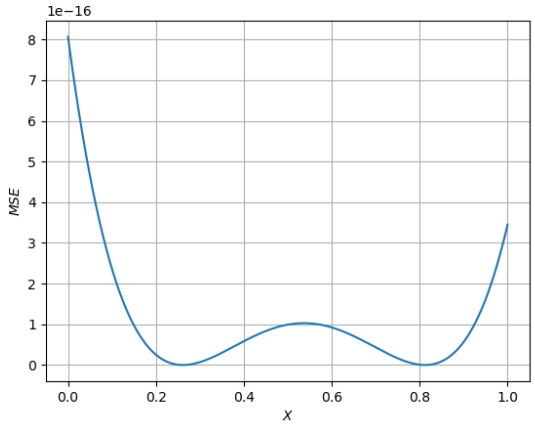



Figure 5.2.6: (Example 2: MSE between the exact solution and the approximate solution.

Table III: The MSE error for Example 3.

| n | MSE on Training set | MSE on Testing set |
|---|---|---|
| 3 | 1.09178389e-16 | 9.83884687e-17 |
| 4 | 2.72825599e-15 | 2.19793864e-15 |
| 5 | 8.72100376e-12 | 7.84314389e-12 |
| 6 | 7.75215323e-11 | 6.39991518e-11 |
| 7 | 8.23544625e-11 | 9.99376595e-11 |

Table IV: The MSE error for Example 4.

| a | n | MSE on Training set | MSE on Testing set |
|---|---|---|---|
| 0 | 3 | 2.00812521e-14 | 1.7978776e-14 |
| 0 | 4 | 2.20219156e-12 | 2.10087744e-12 |
| 5 | 3 | 1.25071358e-15 | 1.12903254e-15 |
| 5 | 4 | 3.17235264e-14 | 2.71073112e-14 |

**Example 4.** Consider the following linear FIE of the second kind

$$u(x) - \int_0^1 9xyu(y)\,dy = ax^2 - 4x^2,$$

with the exact solution $u(x) = (a-4)x(x-\frac{9}{8})$.

The approximate solution has order of accuracy around $e-14$ for $a = 0, n = 3$ after 30000 iteration, with optimal weights

$$\mathbf{w}_n = [-1.33333235, 4.49999805, -2.66666542]$$

For, $n = 3, a = 5$, the generalization MSE is around $e-15$ after 30000 iteration, and the corresponding optimal weights are

$$\mathbf{w}_n = [0.33333309, -1.12499951, 0.66666636]$$

Figure 5.2.8(a) and Figure 5.2.10(a) show the exact and approximate solutions for $a = 0, n = 3$ and $a = 5, n = 3$, respectively. Figure 5.2.8(b) and Figure 5.2.10(b) show the corresponding accuracy measured in MSE. The numerical results can be found in Table IV.

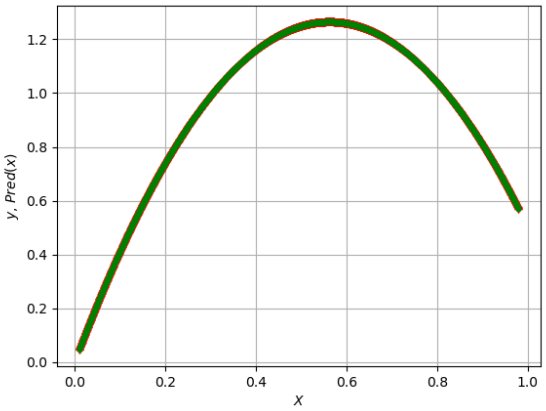



Figure 5.2.7: Example 4: Exact solution vs NN-Collocation approximate solution with $a = 0$ and $n = 3$;

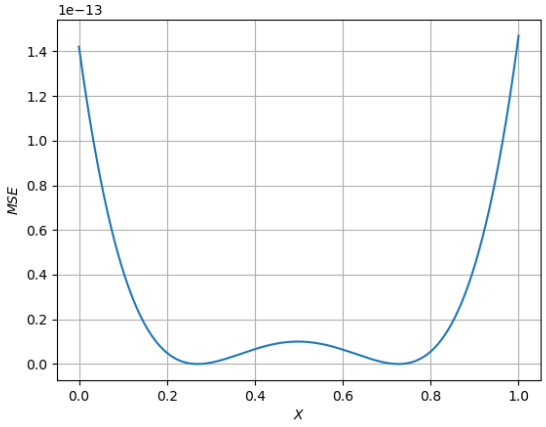



Figure 5.2.8: Example 4: MSE between the exact solution and the approximate solution with $a = 0$ and $n = 3$.

## 5.3 Conclusion

Numerical results show that the NN-Collocation model provides solutions with higher accuracy in comparison to the previous neural network approach for solving Fredholm integral equations of the second kind. The model offers a low computational cost since the required integrals only need to be evaluated once. It is worth pointing out that the accuracy of the NN-Collocation model on the testing set is as good as on the training set, indicating that the NN-Collocation model is stable and reliable. Moreover, this approach overcomes the

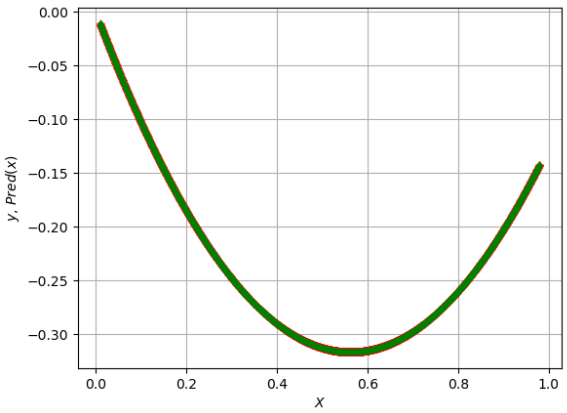



Figure 5.2.9: Example 4: Exact solution vs NN-Collocation approximate solution with a=5,n=3

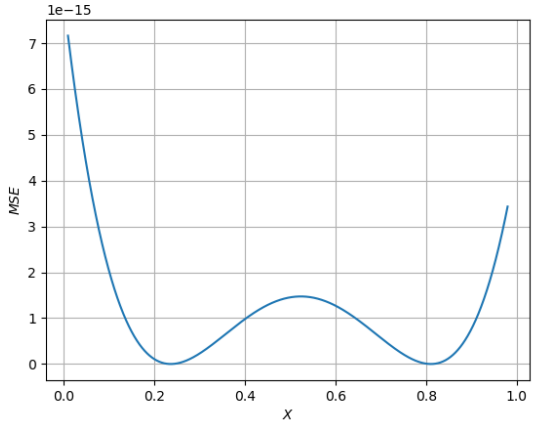



Figure 5.2.10: Example 4: MSE between the exact solution and the approximation solution with $a = 5, n = 3$.

problem of the traditional collocation method which requires the number of bases to match the number of collocation points to ensure a square matrix after discretization. However, this NN-Collocation model only works for linear integral equations.

# Chapter 6

# Special Function Neural Network (SFNN) Models

Robust implementations of special functions have been a concern in many scientific areas, from electromagnetics to statistics. For example, the kernel of the Helmholtz equation in a boundary integral formulation is based on Hankel functions, or the Matérn covariance in statistics depends on the functions Gamma and modified Bessel. Traditionally these special functions are implemented using known asymptotic expansions on certain critical intervals. The strategy we introduce here is to replace asymptotic expansions with neural network (NN) models taking advantage that NNs can be provably considered to be universal approximators. This approach facilitates a plethora of operations previously inaccessible. For instance, high-order derivatives of a neural network model preserve the accuracy of the trained model and, as such, can be more reliable than derivatives of asymptotic expansions. Implementations of series expansions may be computationally prohibitive and prone to numerical errors in regions where they do not converge sufficiently fast. In the current work, we develop neural network models to be a stand-in for special functions, focusing on the Bessel functions of the first and second kind, and corresponding derivatives. Special functions may require different series expansions for different ranges of the argument. We showcase a strategy for using the

same neural network model over any interval within the domain of definition of the function, that would otherwise require different asymptotic expansion representations.

Special functions lack a formal definition, and a commonly accepted description of what renders a function *special* is that it is non-algebraic. In this spirit, even commonly used functions such as the trigonometric functions *sine, cosine* or the exponential and logarithmic functions qualify as special functions. However, due to their ubiquitous use in applications, their implementations are highly reliable, and due to their apparent simplicity, they also possess analytically known derivatives. Subsequently, the tendency is to regard as special some of the functions less encountered in the literature, e.g., Bessel functions, hypergeometric functions, etc. Bessel functions, for example, are known as solutions of differential equations, and different parameters yield different types of Bessel functions. Particularly, the Bessel functions of 1st and 2nd kind, denoted as $J_0(x)$ and $Y_0(x)$ stem from $x^2\frac{d^2y}{dx^2}+x\frac{dy}{dx}+(x^2-\alpha^2)\,y=0$ as a solution pair corresponding to $y(x)$, where the arbitrary $\alpha$ is known as the order of the Bessel function.

It should be noted that special functions garnered an unfair reputation of being a niche topic in scientific computing. However, these functions are to be found in many areas of computational science, as kernels of the Helmholtz equation, or in renowned algorithms such as the Fast Multipole Method [37], or Ewald summation [38], and even in statistics in the Matérn kernel. The study of such functions was prevalent prior to the advent of high-performance computing and the general reliance on libraries in scientific computing. The implementation of special functions relies on asymptotic expansions [39] in regions of interest and may require different numerical treatments as they converge to either infinity or in the vicinity of a zero, e.g., $x$ where $x$ satisfies $f(x) = 0$. Several libraries [40], [41] have been developed to provide accurate evaluations across the interval of definition of such functions, equipped with a range of dedicated algorithms [42]. In many circumstances, particularly for mathematical software such as MATLAB, numpy, Mathematica, the underlying

implementation is not readily available, it may be unreliable [1] and prone to errors as these functions are subject to complex manipulations. The lack of numerically stable expressions for some special functions or their by-products prompts a user to invoke multi-precision libraries, which are notorious for slowing down computations.

In the current work, we advocate for replacing convoluted analytical expressions with neural network models, which may converge slowly to a reliable model, however, these models may be far more reliable in practice and intuitive to a novice user. This approach is justified by theoretical results, such as the universal approximation theorem [**?**], which assure that a multi-layer feed-forward neural network can mimic any known function. Additionally, such models can be deployed as a JIT (just in time) library and trained at compute time.

The work is organized as follows, in Sec. 6.0.1 we detail some of the known issues crippling special function evaluations and operations. In Sec. 6.1 we describe the approach to designing neural network models and some of the theoretical results that can be used to define adequate loss functions and accelerate the training phase. Sec. 6.2 gathers current results for some basic special functions, here the Bessel functions of 1st and 2nd kind, as well as accuracy considerations in using the neural network models in complex computations, while Sec. 6.3 and 6.4 summarize the status of the current work and identify future tracks.

### 6.0.1 Challenges of special functions

The main challenge posed by special functions is of a practical nature; handling complex analytical expressions is cumbersome and time-consuming since the expressions and properties differ from one special function to another. Beyond this difficulty, the range of available formulas, most notably gathered in compendiums [39], [43], prove to be insufficient as we seek expressions for high-order derivatives, numerically stable formulations for complex orders, or compounded special functions.

Of the many expressions available for a Bessel function let us assess the Bessel function

[1] `www.advanpix.com/2016/05/12/accuracy-of-bessel-functions-in-matlab/`

of the first kind, positive order $\alpha$ and real argument $x$, given in terms of $\Gamma$, the Gamma function:

$$J_\alpha(x) = \sum_{m=0}^{\infty} \frac{(-1)^m}{m!\Gamma(m+\alpha+1)} \left(\frac{x}{2}\right)^{2m+\alpha}.$$

We note a few of the under-reported issues encountered in applications

**Numerical errors** The truncation of series expansions used in conjunction with analytical formulas to derive more complicated expressions using special functions leads to undesired cancellations that alter the final result. It is well established that interlacing numerical expressions with analytical formulas may be unreliable. Special functions libraries may be deceiving to a user since they do not consistently uncover the specific low-level implementation.

**Erroneous derivative computations** Even if a series expansion is valid for a given function in a certain range, there is no guarantee that taking derivatives of each individual term in the series yields a novel series sufficiently converged to the correct value or even a converging series. This aspect becomes highly problematic for derivatives of special functions, and analytical derivatives used in numerical computations may introduce numerical mismatches which accumulate and propagate.

**Sensitivity to round-off errors** The computation of derivatives is one of the most sensitive operations to round-off errors. However, such sensitivities can also be observed for Bessel functions of complex order, etc. In such cases, a multi-precision library could alleviate the onset of round-off errors. Multiprecision libraries, though, lead to inefficient evaluations. Therefore if the function is known and used extensively, an analytical derivative could be better suited. However, explicit derivatives, if not evaluated themselves correctly or of a different nature than the original function, may induce numerical artifacts.

## 6.1 Special function neural network (SFNN) model

We seek to develop neural network models that approximate special functions with high accuracy, but are also sufficiently compact to be evaluated efficiently. The accuracy is determined by the choice of loss function and activation functions, while the efficiency of a neural network model depends on the size of the neural network layers. We focus primarily on the accuracy of a neural network model, however we choose layer sizes that may be implemented efficiently on Graphical processing units (GPU). We note that GPUs display significantly higher computational speeds for arithmetically intense algorithms and reliable SFNNs should be implemented at compiler level.

Consider a feed-forward neural network with input $x_0$, output $x_L$, and $L$ layers. At each layer $\ell$, for $\ell = 1, \ldots, L$ we have

- $x_{\ell-1}$ input nodes, with $x_{\ell-1}$ a column vector, $x_{\ell-1} \in \mathbb{R}^{N_{\ell-1}\times 1}$
- $W_\ell$ weights, with $W_\ell$ a matrix, $W_\ell \in \mathbb{R}^{N_{\ell-1}\times N_\ell}$
- $b_\ell$ biases, with $b_\ell$ a column vector, $b_\ell \in \mathbb{R}^{N_\ell \times 1}$

Let us call $z_\ell = W_\ell x_{\ell-1} + b_\ell$ the input to a layer, and $x_\ell = a_\ell(z_\ell)$ the output of a layer, where $a_\ell$ is an activation function. A special function neural network model (SFNN) is given at a layer $\ell$ as

$$x_\ell = a_\ell(W_\ell x_{\ell-1} + b_\ell) \ ,$$

which would recursively contribute to the network output $x_L = a_L(z_L) = a_L(W_L x_{L-1} + b_L)$.

To define a loss function there are several approaches available. We can either train the SFNN by taking a data fitting approach, a residual minimization approach [44] or construct a complex multi-objective loss function [45]. Here, we seek to model a function $y(x)$ by restricting ourselves to a cost functional

$$\mathcal{L}(x_L, y(x)) = ||x_L - y(x)||_* \ ,$$

where $||\cdot||_*$ can be the $L_1$, $L_2$ norms, or any other norm of choice.

**SFNN design** By adding an $L_2$ regularization term we consider mainly loss functions of the type

$$\mathcal{L}(x_L, y(x)) = \frac{1}{N}\sum_{i\in N}(x_L - y(x_i))^2 + \lambda \sum_{j\in L}\|W_j\|^2 \tag{6.1}$$

where $N$ is the size of the training set, $\lambda$ is the regularization parameter, and $L$ the number of layers. The training will be performed by drawing samples $x \in \mathcal{X}_t$, where $\mathcal{X}_t$ is a random distribution over a training interval $[a,\ b]$. The initialization of weights is performed using the Xavier weight initialization [32].

The obtained SFNN model will be available as a function $\hat{y}(x_0)$ given as

$$\hat{y}(x_0) = W_L a_L(\ldots W_2 a_1(W_1 x_0 + b_1) + b_2 \ldots) + b_L\ , \tag{6.2}$$

where $a_1, \ldots, a_L$ are the activation functions per layer, and $x_0$ is the evaluation point which should be in the training interval $[a,\ b]$. The restriction that a model is usually not available outside its training interval poses no inconveniences for special function models since asymptotic series expansions are also valid only within certain ranges. The optimization algorithm used for updating the weights and biases in the training phase is ADAM [35].

**Large interval evaluations** Since the initial weight and bias depend on the scale of the input the performance of the training as well as accuracy decays for larger intervals. By selecting smaller ranges, typically on the order of $[a,\ b]$ with $|a-b| < 10$ we achieve higher convergence and preserve well balanced weights and biases. We focus on small intervals $[a,\ b]$ and larger intervals, or intervals far away from the origin, are mapped to the original $[a,\ b]$. We give preference to intervals centered around the origin, which has a two-fold interpretation: a) assures a proper balance of the weights, b) conforms with the symmetry properties of the modeled functions. The mapping of an interval $[c,\ d]$ to the target interval

$[a,\ b]$ relies on the formula

$$x \in [a,\ b], \quad x^* = \frac{(x+b)(d-c)}{b-a} + a\ .$$

This amounts to a shift of the model to be trained on $[a,\ b]$, which yields the intermediary model $\hat{y}^*$. To retrieve the correct value on the appropriate interval we reverse the shift of $\hat{y}(x^*)$ to

$$\hat{y}(x) = \hat{y}^*\left(\frac{(b-a)(x-c)}{d-c} + a\right)\ ,$$

where $\hat{y}(x)$ will now be the special function model on the interval $[c,\ d]$.

The final goal of the current work is to provide reliable and efficient SFNNs which are also flexible and have the following properties

- the SFNN model can be easily imported as a function composition
- the SFNN model can be identically generated across the entire domain of definition without modifications.
- the SFNN model is easy to generate on-the-fly for novel special functions and their derivatives
- minimal modeling is required of the user to generate novel SFNN models

## 6.2 Results

The main appeal of SFNN models is that they can provide reliable derivative computations, unattainable for expressions relying on series expansions. Although many existing special functions libraries achieve computer precision accuracy in most of the domain of definition, the absolute errors decrease to values close to single precision, i.e., $10^{-7}$ in certain regions. Coincidentally, since many GPU implementations are still in the range of single-precision, such models can already be useful when double precision is not immediately

achievable. In this spirit, we find sufficient to impose an SFNN model accuracy on the order of $10^{-8}$. Our initial goal is to obtain robust SFNN models, which can subsequently improve by tightening the accuracy demands, extending the number of iterations, and computing the weights and biases in multi-precision truncated to double-precision upon convergence.

We focus on two special functions: the Bessel function of 1st kind and zero-order $J_0(x)$ and the Bessel function of 2nd kind and zero-order $Y_0(x)$, which we illustrate in Fig. 6.2.1. These Bessel functions serve as a stepping stone to constructing other more sophisticated functions, such as the Hankel function defined as $H_0^0(x) = J_0(x) + iY_0(x)$. Note that training a neural network in the complex space can be highly difficult, and it is desirable to construct a model for the Hankel function based on its real $\mathcal{R}e(H_0(x))$ and imaginary $\mathcal{I}m(H_0(x))$ components.

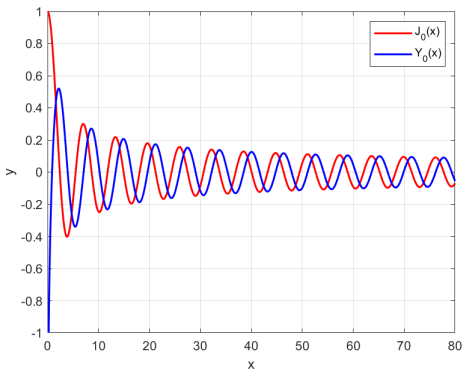



Figure 6.2.1: The behaviour of Bessel functions $J_0(x)$, $Y_0(x)$ on a large interval.

**Bessel function of 1st kind** $J_\alpha(x)$ One of the most challenging properties of Bessel functions of 1st kind is owed to their highly oscillatory nature, see Fig. 6.2.1. Each zero of the function poses problems to implementations. For example, MATLAB can achieve only one correct digit in relative error in the vicinity of zeros [2]. The absolute error of Bessel function evaluations, however, is accurate within computer precision. The relative error gains importance in contexts in which numerical differentiation is needed, or evaluations which should yield cancellations unexpectedly lead to accumulations of errors, etc.

[2] `www.advanpix.com/2016/05/12/accuracy-of-bessel-functions-in-matlab/`

We note that for any real values of order $\alpha$, the behavior of the Bessel function of the first kind is similar in nature, and the difficulties of selecting and training an appropriate SFNN model share sufficient similarities to allow us to focus on the order $\alpha = 0$. In the design of a model for $J_0(x)$ we identified that a simple network with a single hidden layer of size $N = 36$ and the sigmoid

$$\sigma(x) = \frac{1}{1 + e^{-x}} \ ,$$

as the activation function provides the fastest convergence for an accuracy of $10^{-8}$. The full model reads

$$\hat{y}(x_0) = W_2 a_1(W_1 x_0 + b_1) + b_2 \ . \tag{6.3}$$

The training was performed on the interval $[-5,\ 5]$ and approximately $10^5$ epochs were necessary to attain the imposed accuracy. In Fig. 6.2.2 we illustrate the approximation using intervals mapped to $[-5,\ 5]$ which yield a robust model with bounded and balanced weights and biases. We note that other mappings are possible as in Sec. 6.1, however we did not observe significant advantages by choosing sub-intervals of $[-5,\ 5]$, and $[-5,\ 5]$ represented an upper limit of interval size.

The model is validated by applying it to values outside the training set $\mathcal{X}_t$, i.e. $x_0 \notin \{\mathcal{X}_t\}$. The error incurred between the model $\hat{y}(x)$ and $J_0(x)$ provided by a special functions library is illustrated in Fig. 6.2.3. The error behavior is bounded around $10^{-7}$, however, it has offshoots at the end of the training interval. This was observed consistently for a range of intervals, constraints, and layer sizes and is a numerical artifact that evokes the Gibbs phenomenon encountered in high-order polynomial fitting. We observed that performing the training on intervals $[a,\ b]$ and applying the model on intervals $[a + \epsilon,\ b - \epsilon]$ with $\epsilon = 0.1$ is highly reliable and alleviates this issue. In Fig. 6.2.5 we show that training the model on a rescaled interval using an affine transformation yields a SFNN model with similar properties as for intervals in a small vicinity of the origin. The behaviour of the Bessel function in the range $[120,\ 125]$ would typically require a different series expansion than in the range

$[-5,\ 5]$, see [46]. The error incurred by remapping the interval of definition to a region close to the origin is insignificant. A higher impact could be expected on the evaluations of the model itself. However, in Fig. 6.2.5 we note that the error range is bounded by $10^{-7}$ on the inner domain with spikes at the interval ends as observed also for the model developed on $[-5,\ 5]$.

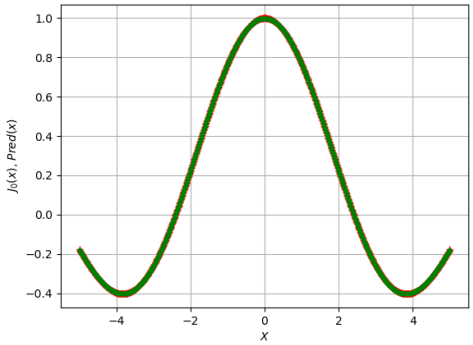



Figure 6.2.2: SFNN model of the Bessel function of 1st kind, $J_0(x)$, on the interval $[-5, 5]$.

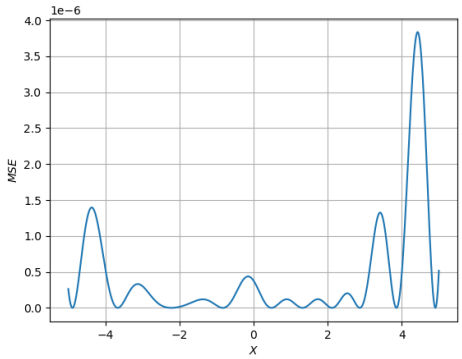
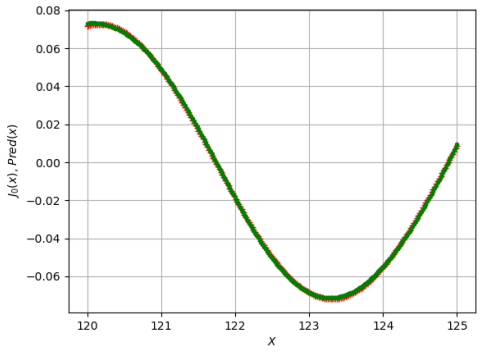


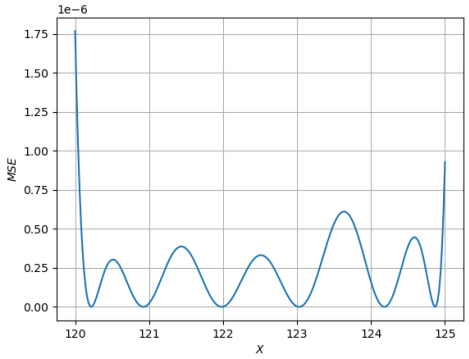

Figure 6.2.3: MSE for the SFNN model of the Bessel function of 1st kind, $J_0(x)$, on the interval $[-5, 5]$.

Bessel functions of the first kind are characterized by several symmetry properties [39] that can be leveraged in constructing multi-objective loss functions. For example for an integer order $\alpha$ we have that $J_{-\alpha}(x) = (-1)^{\alpha} J_{\alpha}(x)$, and such knowledge has been previously integrated in the loss function, see [45]. Since we intend to remove the modeling burden from the user we avoided the incorporation of such identities in the loss function provided




,

Figure 6.2.4: SFNN model of the Bessel function of the 1st kind, $J_0(x)$, on the interval $[120, 125]$, using interval remapping.




,

Figure 6.2.5: MSE for the SFNN model of the Bessel function of 1st kind, $J_0(x)$, on the interval $[120, 125]$, using interval remapping.

in Eq. 6.1. However, we noticed that training on symmetric intervals as opposed to non-symmetric ones is significantly slower in terms of number of iterations as well as numerical stability of the produced model.

**Bessel function of 2nd kind** $Y_\alpha(x)$ The Bessel function of 2nd kind, $Y_\alpha(x)$ shares the oscillatory behaviour of $J_\alpha(x)$ and poses an additional difficulty in the vicinity of $x = 0$ where it decays asymptotically as $Y_\alpha(x) \approx -\infty$. The numerical evaluation of $Y_\alpha(x)|_{x\approx 0}$ is in essence unfeasible and an interval too close to the origin is either provided as a limit value, truncated or avoided altogether. To train an SFNN model for $Y_0(x)$ we restrict ourselves to an interval away from zero and for the same loss function as in Eq. 6.1 we observe that a

larger network is necessary. Without compromising generality we focus on $Y_0(x)$, the Bessel function of the second kind and zero order. In this case we consider a network with three hidden layers given as

$$\hat{y}(x_0) = W_4(a_3(W_3 a_2(W_2 a_1(W_1 x_0 + b_1) + b_2) + b_3) + b_4 \ , \tag{6.4}$$

with activation functions $a_{1,3}(x) = max(0, x)$ (ReLU) and $a_2(x) = \sigma(x)$ (sigmoid). The training on the interval $[0.1,\ 1]$ achieved the imposed tolerance of $10^{-8}$ in approximately $5 \times 10^3$ epochs.

In Fig. 6.2.6 we illustrate a comparison between the model $\hat{y}(x)$ and the reference implementation $Y_0(x)$ on the interval $[0.1,\ 1]$. The error displayed in Fig. 6.2.7 exhibits the same spikes at the end of the training interval and we recommend to consider in practice the model $\hat{y} : [0.1 + \epsilon, 1 - \epsilon] \to \mathbb{R}$.

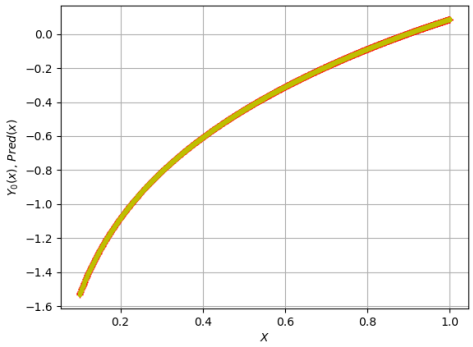



Figure 6.2.6: Bessel function of 2nd kind, $Y_0(x)$, on the interval $[0.1, 1]$.

**Derivatives of special functions** One of the most glaring computational inconveniences of special functions is that the most reliable expressions are derived analytically even if used in complex numerical implementations. This interplay between analytical and numerical evaluations may lead to precision mismatches that accumulate and propagate, leading ultimately to entirely flawed results. A common source of inaccuracies arises from derivatives of special functions. For example, the derivative of the Bessel function of 1st kind, zero

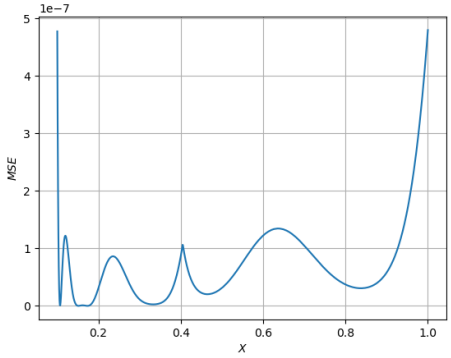



Figure 6.2.7: MSE for the SFNN model of the Bessel function of 2nd kind, $Y_0(x)$, on the interval $[0.1, 1]$.

order, can be determined analytically from known properties to be $\frac{d}{dx}J_0(x) = -J_1(x)$. It is appropriate to implement the derivative of the first order Bessel as the negative Bessel function of the first order, however, the evaluation error may not be of the same nature as for the function $J_0$.

Using a SFNN model representation for $J_0$, as in Eq. 6.3, we can take derivatives safely either using the automatic differentiation implementation of Tensorflow, or by traversing the NN model using the chain rule and the known derivatives of activation functions. In Fig. 6.2.8 we illustrate that the derivative of the constructed $J_0(x)$ SFNN model is a good approximation of the analytical derivative $-J_1(x)$, and in Fig. 6.2.9 we illustrate that the error preserves the behaviour of $J_0(x)$ and incurs an error less than one order of magnitude.

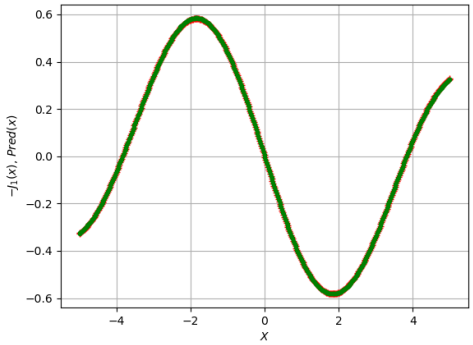



Figure 6.2.8: Derivative of the Bessel function of 1st kind and zero order on the interval $[-5,\ 5]$.

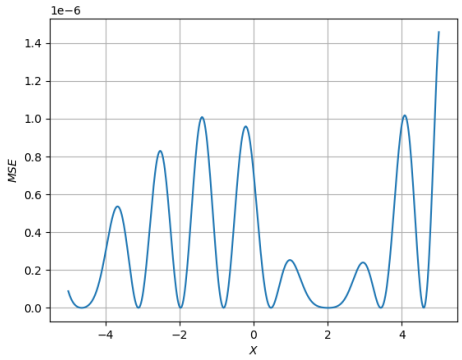



Figure 6.2.9: Error of the derivative of $J_0(x)$ on the interval $[-5,\ 5]$.

## 6.3 Future work

The final goal of the current work is to construct reliable neural network models for the approximation of special functions, which can be used by applications at compute time. Although the current reliability does not surpass single-precision, we identified interesting behaviors worthy of in-depth studies, such as off-shoots at interval ends and unconstrained neural network behaviors consistent with known special functions properties. Incipient work indicated that higher accuracy can be achieved by seeking convergence in higher-precision. We intend to conduct specialized studies on neural network behavior in conjunction with multi-objective loss functions, which embed function properties as well as minimize the residual of the differential equation defining the special function. Most importantly we seek to apply the neural network models to intractable expressions and derivatives of special functions.

## 6.4 Conclusion

We explored the advantages and limitations of using neural networks as models for special functions, referred to as SFNN (special function neural networks). The highlight of this work is that we identified that intervals which traditionally require different asymptotic expansions can be treated identically using neural network models. We chose two classical and well-

known functions, Bessel of 1st and 2nd kind, which display several numerically challenging issues. Both functions are oscillatory and have a range of solutions on their interval of definition, while Bessel of 2nd kind is undefined at the origin. The focus was on the design and training of neural networks to preserve the numerical accuracy of the original functions. The loss function was taken as a regularized MSE norm between the model output and discrete values of the function under training. Different training intervals were explored, and a strategy for large intervals was identified. Due to the oscillatory nature, smaller intervals containing at most one or two zeros of the modeled function provide the best convergence rate and accurate neural networks. To avoid spurious effects in the weights or biases, we used an interval remapping strategy. Consistently we noted offshoots at the ends of the training interval which we corrected by differentiating between the training interval $[a,\ b]$ and the model definition interval $[a+\epsilon,\ b-\epsilon]$ with $\epsilon$ selected heuristically as $\epsilon = 0.1$. The maximal accuracy we obtained did not exceed $10^{-7}$. However, the error behavior is preserved and the magnitude is bounded under derivative operations performed on the SFNN model which indicates that such models have the potential to fully replace current special function implementations.

# Acknowledgment

This research was supported in part by an appointment with the National Science Foundation (NSF) Mathematical Sciences Graduate Internship (MSGI) Program sponsored by the NSF Division of Mathematical Sciences. This program is administered by the Oak Ridge Institute for Science and Education (ORISE) through an interagency agreement between the U.S. Department of Energy (DOE) and NSF. ORISE is managed for DOE by ORAU. All opinions expressed in this paper are the author's and do not necessarily reflect the policies and views of NSF, ORAU/ORISE, or DOE. Additionally, this research was supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing

Research under Contract No. DE-AC02-06CH11357 at Argonne National Laboratory.

# Chapter 7

# Conclusion and Future Work

We employed the multiscale Galerkin method to solve Volterra integral equations of the second kind with a weakly singular kernel in chapter 2. Through leveraging the structure of the Volterra integral operator and the vanishing moment properties of multiscale basis functions, we eliminated a substantial portion of zero entries from the coefficient matrix of the corresponding discrete system of equations. We then designed a truncation strategy to compress the coefficient matrix for the remaining nonzero entries. To estimate the rest of the nonzero entries, we introduced a numerical quadrature rule, with an error control strategy designed to prevent the quadrature error from ruining the overall convergence order of the multiscale Galerkin method. The numerical results demonstrate the robustness of this approach.

In chapters 4 and 5, we propose neural network models to solve numerically the linear Fredholm integral equations of the second kind with a continuous kernel. The cost function is defined using the average residual of the Fredholm integral equation on some finite set of points and is optimized by the Adam method. The numerical results confirm the potential of applying this method in solving integral equations, but the accuracy of the neural network is not as good as that of the traditional mathematical method. Specifically, the approach discussed in chapter 4 is totally by learning, the solution of this method is restricted in

form $w_2(\frac{1}{1+e^{-(w_1 x+b_1)}})+b_2$. Without a sufficient number of neurons in the hidden layer, this form of function has limitations in approximating the exact solution. To overcome this limitation, in chapter 5, we adopt a new approach that first restricts the solution space to the polynomial space and designed a Collocation method-based neural network model. The second approach is more efficient and accurate than the first one and is comparable to traditional mathematical methods in some cases. However, the performance of the neural network model highly depends on the learning algorithm, especially the choice of the cost function. In our proposed approaches in chapter 4 and chapter 5, instead of minimizing the difference between the exact and approximate solutions directly, we minimize the average residue of the integral equation at the given input data. Hence, even if we find the minimum of the cost function value, we are not guaranteed to be close to the desired solution. This is a typical under-fitting problem that occurs in machine learning algorithms. In our future work, we plan to explore how to improve the cost function of the neural network model to enhance its performance. One possible approach is to use quadrature methods to discretize the integral equation and solve it for a range of input values. The resulting output values then can be used as the training data for the neural network. Another potential way to achieve this is to impose additional restrictions on the cost function. Additionally, it would be worthwhile to investigate combining the collocation method and the Galerkin method, with the cost function being the residue of the Fredholm integral equation resulting from both methods.

In the last chapter, we developed a neural network model to approximate special functions with high accuracy, the main appeal of our proposed SFNN model is that it can provide reliable derivative computations [47]. The error behavior is preserved and the magnitude is bounded under derivative operations performed on the SFNN model which indicates that such models have the potential to fully replace current special function implementations.

# Bibliography


[1] Z. Chen, C. A. Micchelli, and Y. Xu, *Multiscale methods for Fredholm integral equations*. Cambridge University Press, 2015, vol. 28.

[2] Y. Liu, L. Shen, Y. Xu, and H. Yang, "A collocation method solving integral equation models for image restoration," *JIE*, pp. A 28. 263–307. 10.1216, 2016.

[3] C. A. Micchelli, Y. Xu, and Y. Zhao, "Wavelet galerkin methods for second-kind integral equations," *Journal of Computational and Applied Mathematics*, vol. 86, no. 1, pp. 251–270, 1997, dedicated to William B. Gragg on the ocassion of his 60th Birthday. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S037704279700160X

[4] B. R. Effati, S., "A neural network approach for solving fredholm integral equations of the second kind," *Comput and Applic 21*, p. 843–852, 2012.

[5] I. E. Lagaris, A. Likas, and D. I. Fotiadis, "Artificial neural networks for solving ordinary and partial differential equations," *IEEE transactions on neural networks*, vol. 9, no. 5, pp. 987–1000, 1998.

[6] Y. Guan, T. Fang, and D. Zhang, "Solving fredholm integral equations using deep learning," *Int. J. Appl. Comput*, 2022.

[7] Y. Liu and O. Marin, "Special function neural network (sfnn) models," in *2021 IEEE International Conference on Cluster Computing (CLUSTER)*, 2021, pp. 680–685.

[8] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural networks*, vol. 2, no. 5, pp. 359–366, 1989.

[9] A. Weinstein., "Review: Vito Volterra: Opere mathematiche. memorie e note," *Amer. Math. Soc. 70 (3)*, pp. 335–337, 1964.

[10] C. Corduneanu and I. Sandberg, *Volterra Equations and Applications (1st ed.)*, 2000.

[11] A.-M. Wazwaz, *Linear and Nonlinear Integral Equations: Methods and Applications*, 2011.

[12] V. K. Singh, R. K. Pandey, and O. P. Singh, "New stable numerical solutions of singular integral equations of abel type by using normalized bernstein polynomials," *Applied Mathematical Sciences*, vol. 3, no. 5, pp. 241–255, 2009.

[13] C. Lubich, "Runge-kutta theory for volterra and abel integral equations of the second kind," *Mathematics of Computation*, vol. 41, no. 163, pp. 87–102, 1983.

[14] R. K. Miller and A. Feldstein, "Smoothness of solutions of volterra integral equations with weakly singular kernels," *SIAM Journal on Mathematical Analysis*, vol. 2, no. 2, pp. 242–258, 1971. [Online]. Available: https://doi.org/10.1137/0502022

[15] A. Friedman and M. Shinbrot, "Volterra integral equations in banach space," *Transactions of the American Mathematical Society*, vol. 126, no. 1, pp. 131–179, 1967. [Online]. Available: http://www.jstor.org/stable/1994417

[16] G. Gripenberg, "On a nonlinear volterra integral equation in a banach space," *Journal of Mathematical Analysis and Applications*, vol. 66, no. 1, pp. 207–219, 1978. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0022247X78902780

[17] K. Maleknejad and A. Ostadi, "Numerical solution of system of volterra integral equations with weakly singular kernels and its convergence analysis," *Applied Numerical Mathematics*, vol. 115, 01 2017.

[18] E. Shoukralla, B. Ahmed, M. Sayed, and A. Saeed, "Interpolation method for solving volterra integral equations with weakly singular kernel using an advanced barycentric lagrange formula," *Ain Shams Engineering Journal*, vol. 13, no. 5, p. 101743, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2090447922000545

[19] S. Mallat, "A theory for multiresolution signal decomposition: the wavelet representation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 11, no. 7, pp. 674–693, 1989.

[20] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[21] C. Bishop, "Mixture density networks," Aston University, WorkingPaper, 1994.

[22] K. P. Murphy, *Machine learning : a probabilistic perspective*. Cambridge, Mass. [u.a.]: MIT Press, 2013. [Online]. Available: https://www.amazon.com/Machine-Learning-Probabilistic-Perspective-Computation/dp/0262018020/ref=sr_1_2?ie=UTF8&qid=1336857747&sr=8-2

[23] K. Jarrett, K. Kavukcuoglu, M. Ranzato, and Y. LeCun, "What is the best multi-stage architecture for object recognition?" in *2009 IEEE 12th International Conference on Computer Vision, ICCV 2009*, ser. Proceedings of the IEEE International Conference on Computer Vision, 2009, pp. 2146–2153, copyright: Copyright 2010 Elsevier B.V., All rights reserved.; 12th International Conference on Computer Vision, ICCV 2009 ; Conference date: 29-09-2009 Through 02-10-2009.

[24] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *International Conference on Machine Learning*, 2010.

[25] H. G. Rumelhart, D. and R. Williams, "Learning representations by back-propagating errors," *Nature*, pp. 323, 533–536. 13, 17, 22, 200, 221, 367,472,477, 1986a.

[26] L. Bottou, "Online algorithms and stochastic approximations," in *Online Learning and Neural Networks*, D. Saad, Ed. Cambridge, UK: Cambridge University Press, 1998, revised, oct 2012. [Online]. Available: http://leon.bottou.org/papers/bottou-98x

[27] B. Polyak, "Some methods of speeding up the convergence of iteration methods," *USSR Computational Mathematics and Mathematical Physics*, vol. 4, no. 5, pp. 1–17, 1964. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0041555364901375

[28] Q. Qiu, "Introduction of machine learning," 2021.

[29] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *Journal of Machine Learning Research*, vol. 12, no. 61, pp. 2121–2159, 2011. [Online]. Available: http://jmlr.org/papers/v12/duchi11a.html

[30] Y. N. Dauphin, H. de Vries, J. Chung, and Y. Bengio, "Rmsprop and equilibrated adaptive learning rates for non-convex optimization." *CoRR*, vol. abs/1502.04390, 2015. [Online]. Available: http://dblp.uni-trier.de/db/journals/corr/corr1502.html#DauphinVCB15

[31] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2015. [Online]. Available: http://arxiv.org/abs/1412.6980

[32] 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*. JMLR Workshop and Conference Proceedings, 2010, pp. 249–256.

[33] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural Networks*, vol. 2, no. 5, pp. 359–366, 1989. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0893608089900208

[34] G. V. Cybenko, "Approximation by superpositions of a sigmoidal function," *Mathematics of Control, Signals and Systems*, vol. 2, pp. 303–314, 1989.

[35] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2017.

[36] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant, "Array programming with NumPy," *Nature*, vol. 585, no. 7825, pp. 357–362, Sep. 2020. [Online]. Available: https://doi.org/10.1038/s41586-020-2649-2

[37] N. Engheta, W. Murphy, V. Rokhlin, and M. Vassiliou, "The fast multipole method (fmm) for electromagnetic scattering problems," *IEEE Transactions on Antennas and Propagation*, vol. 40, no. 6, pp. 634–641, 1992.

[38] J. Kolafa and J. W. Perram, "Cutoff errors in the ewald summation formulae for point charge systems," *Molecular Simulation*, vol. 9, no. 5, pp. 351–368, 1992. [Online]. Available: https://doi.org/10.1080/08927029208049126

[39] M. Abramowitz and I. A. Stegun, "Handbook of mathematical functions with formulas, graphs, and mathematical table," in *US Department of Commerce*. National Bureau of Standards Applied Mathematics series 55, 1965.

[40] W. Cody, "The construction of numerical subroutine libraries," *SIAM Review*, vol. 16, no. 1, pp. 36–46, 1974.

[41] L. W. Fullerton, "Portable special function routines," in *Portability of Numerical Software*. Springer, 1977, pp. 452–483.

[42] D. Amos, "Algorithm 644: A portable package for bessel functions of a complex argument and nonnegative order," *ACM Transactions on Mathematical Software (TOMS)*, vol. 12, no. 3, pp. 265–273, 1986.

[43] "*NIST Digital Library of Mathematical Functions*," http://dlmf.nist.gov/, Release 1.1.2 of 2021-06-15, f. W. J. Olver, A. B. Olde Daalhuis, D. W. Lozier, B. I. Schneider, R. F. Boisvert, C. W. Clark, B. R. Miller, B. V. Saunders, H. S. Cohl, and M. A. McClain, eds. [Online]. Available: http://dlmf.nist.gov/

[44] M. Raissi, P. Perdikaris, and G. E. Karniadakis, "Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations," *Journal of Computational Physics*, vol. 378, pp. 686–707, 2019.

[45] M. Ngom and O. Marin, "Fourier neural networks as function approximators and differential equation solvers," *Statistical Analysis and Data Mining: The ASA Data Science Journal*, 2020.

[46] G. Nemes, "Error bounds for the large-argument asymptotic expansions of the hankel and bessel functions," *Acta Applicandae Mathematicae*, vol. 150, no. 1, pp. 141–177, 2017.

[47] K. Hornik, M. Stinchcombe, and H. White, "Universal approximation of an unknown mapping and its derivatives using multilayer feedforward networks," *Neural Networks*, vol. 3, no. 5, pp. 551–560, 1990. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0893608090900056

# Biographical Data

NAME OF AUTHOR: Yuzhen Liu

PLACE OF BIRTH: Jining, Shandong, China

DATE OF BIRTH: September 4, 1988

GRADUATE AND UNDERGRADUATE SCHOOLS ATTENDED:

Henan University of Technology, Zhengzhou, Henan, China

Syracuse University, Syracuse, New York, USA

Sun Yat-sen University, Guangzhou, Guangdong, China

DEGREES AWARDED:

B.S. Information and Computing Science, Henan University of Technology, 2010

M.S. Information and Computing Science, Sun Yat-sen University, 2012

PhD. Information and Computing Science, Sun Yat-sen University, 2016

M.S. Mathematics, Syracuse University, 2021